Perancangan dan Analisis Algoritma (EF234405)

Kuliah 1 — Fondasi, Kerangka Kerja, dan Persoalan Pertama

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*, Bab 1

Dosen: Irfan Subakti — `yifana@gmail.com`

Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya

Hari ini kita membangun metode yang akan kita pakai sepanjang semester. Buku: Bab 1, serta Cormen et al., Introduction to Algorithms, edisi ke-4, Bab 1.

Agenda

1. Apa itu algoritma, dan apa bedanya dengan program dan dengan struktur data.

2. Dua sasaran: **kebenaran** (dibuktikan, bukan sekadar diuji) dan **efisiensi** (diukur terhadap sebuah anggaran).

3. **Lima paradigma perancangan** dan **kerangka lima fase**: Memahami, Mengabstraksi, Merancang, Menganalisis, Mengimplementasikan.

4. Enam soal juri yang sesungguhnya, dari rumusan soal sampai C++ yang *Accepted*: *Columns*, *Journey*, *Unique Numbers*, *Minimum and Maximum Numbers*, *Interesting Sequence*, *Recursive Function 1*.

5. Para juri, buku acuan, peta jalan — dan meja kerja Anda: **VS Code dan kompilator C++17** di Windows, macOS, atau Linux.

Butir 1 sampai 3 mengisi paruh pertama pertemuan, butir 4 paruh kedua; butir 5 adalah pekerjaan rumah yang memungkinkan Bab 2. Buku: Bab 1, Subbab 1.1–1.19.

Dari Mana Kita Mulai

Mata ajar ini melatih **berpikir komputasional**, yang berarti empat hal dan dipraktikkan di setiap bab.

Memodelkan persoalan secara algoritmis — persis apa yang masuk, persis apa yang harus keluar, persis masukan mana yang mungkin.

Merancang algoritma — memilih satu dari segelintir strategi dan menata langkah-langkahnya.

Menganalisis kebenaran dan kompleksitas — *membuktikan* bahwa langkah-langkahnya benar untuk setiap masukan yang diizinkan, lalu menghitung waktu dan memori yang diperlukannya.

Mengimplementasikan dengan rapi dalam C++. Perhatikan urutannya: **kode adalah langkah terakhir, bukan yang pertama.**

Buku ini tidak mengandaikan apa pun: setiap lambang diperkenalkan dengan kata-kata sebelum dipakai, dan setiap rumus diturunkan, bukan diumumkan. Buku: Subbab 1.1.

Apa Itu Algoritma

DEFINISI. Sebuah **algoritma** adalah prosedur komputasi langkah demi langkah yang berhingga, yang mengubah sebuah *masukan* menjadi sebuah *keluaran*.

Kata ini berasal dari al-Khwarizmi, cendekiawan Persia abad kesembilan. Resep adalah analogi pertama yang terbaik: untuk *nasi goreng*, nasi, telur, dan bawang merah adalah **masukan**, langkah-langkah yang berurutan adalah prosedurnya, dan sepiring nasi goreng adalah **keluaran**.

Lima sifat menurut Knuth: **(1)** masukan yang terdefinisi jelas; **(2)** keluaran yang terdefinisi jelas; **(3) kepastian** — "tambahkan 3 gram garam", bukan "garam secukupnya".

(4) Keberhinggaan — ia berhenti pada *setiap* masukan yang sah. Aturan Collatz ("bagi dua jika genap, jika ganjil $3n+1$") tidak diketahui selalu berhenti, jadi ia bukan algoritma untuk apa pun. **(5) Keterlaksanaan** — setiap langkah sungguh dapat dikerjakan.

CONTOH. "Ingat bilangan pertama; untuk setiap bilangan sisanya, jika lebih besar, ingat bilangan itu sebagai gantinya; di akhir, umumkan yang Anda ingat." Masukan, keluaran, kepastian, keberhinggaan (n langkah), keterlaksanaan (satu perbandingan) — kelimanya terpenuhi. Ini algoritma, dan program C++ pertama kita.

Buku: Subbab 1.2.1.

Program dan Struktur Data

Sebuah **program** adalah algoritma yang diungkapkan dalam bahasa pemrograman. Algoritma adalah *gagasannya*; program adalah *wujud nyatanya*. Anda bisa keliru di dua tempat — gagasannya atau ketikannya — dan hanya pembuktian yang memberi tahu di mana. Algoritmanya pun bertahan: C++ suatu hari akan tergantikan, *binary search* tidak.

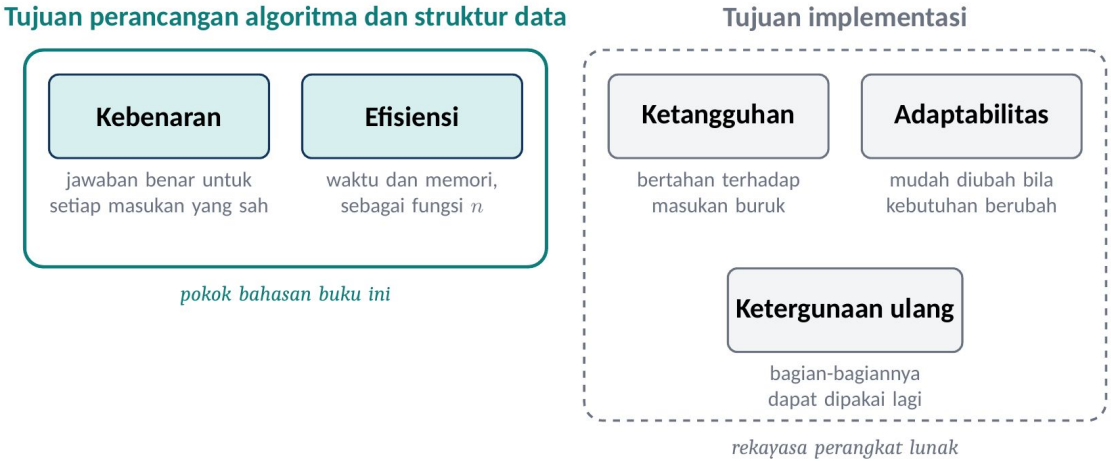
Sebuah **struktur data** menata data agar dapat diakses dan diubah secara efisien. Resep dan dapur: resep yang sama, peralatan yang lebih baik, hidangan lebih cepat jadi. Tabel 1.1 adalah peta wilayahnya.

Tabel 1.1 Struktur data yang dipakai dalam buku ini, dan satu hal yang paling dikuasai masing-masing.

Struktur	Nama di C++	Keunggulan utamanya
Larik / larik dinamis	`int a[]`, `vector`	Membaca atau menulis elemen ke-i seketika
Senarai berantai	`list`	Menyisipkan dan menghapus di tengah dengan murah
Tumpukan	`stack`	Masuk terakhir, keluar pertama — *undo*, rekursi, pencocokan kurung
Antrean	`queue`	Masuk pertama, keluar pertama — penjadwalan, penelusuran melebar
Peta hash	`unordered_map`	Mencari berdasarkan kunci, rata-rata dalam waktu konstan
Pohon pencarian seimbang	`map`, `set`	Sama, tetapi kuncinya tetap tersimpan terurut
Heap / antrean prioritas	`priority_queue`	Berulang kali mengambil elemen terkecil atau terbesar
Graf	senarai ketetanggaan	Memodelkan hubungan, jalan, tautan, kebergantungan

Jangan menghafal Tabel 1.1 sekarang; setiap barisnya datang ketika sebuah persoalan membuat Anda membutuhkannya. Buku: Subbab 1.2.2–1.2.3 dan Tabel 1.1.

Sasaran yang Hendak Kita Capai



Gambar 1.1 Sasaran perancangan struktur data dan algoritma (kebenaran, efisiensi) disandingkan dengan sasaran implementasi (ketangguhan, keluwesan, kegunaan ulang).

Gambar 1.1: kebenaran dan efisiensi (kiri) milik perancangan algoritma dan menjadi pokok mata ajar ini; ketangguhan, kemampuan beradaptasi, dan kebergunaan ulang (kanan) milik rekayasa perangkat lunak. Algoritma yang cepat, dapat dipakai ulang, tangguh, tetapi salah tidak ada gunanya. Buku: Subbab 1.2.4 dan Gambar 1.1.

Sasaran Pertama — Kebenaran

> **DEFINISI.** Sebuah algoritma disebut **benar** jika, untuk *setiap* masukan yang diizinkan rumusan persoalannya, ia berhenti dan menghasilkan keluaran yang diminta persoalan itu.

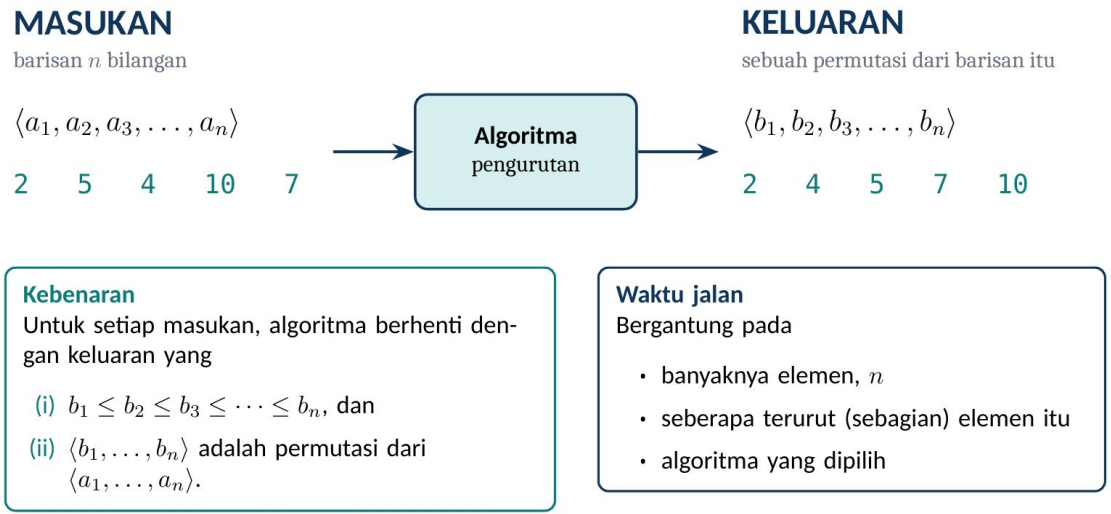
Dengan $1 \le n \le 1000$ dan $1 \le a_i \le 1000$ ada 1000^{1000} masukan untuk ukuran 1000 saja. Pengujian dapat menunjukkan adanya kutu, tidak pernah ketiadaannya. Hanya pembuktian yang bisa.

Bukti langsung — sebuah batas bawah, lalu sebuah konstruksi yang mencapainya. **Bukti dengan kontradiksi** — andaikan jawabannya salah lalu turunkan hal yang mustahil (*Journey*, Subbab 1.7).

Invarian gelung (*loop invariant*) — benar sebelum gelung, terpelihara oleh setiap putaran, berguna di akhir. **Induksi** — benar di anak tangga terbawah, dan setiap anak tangga mengantar ke berikutnya (*Interesting Sequence*, Subbab 1.10).

Keempat teknik muncul di bab ini. Buku: Subbab 1.3.1.

Menyatakan Persoalan dengan Cermat



Gambar 1.2 Persoalan pengurutan: sebuah barisan masuk, sebuah permutasi dari barisan itu keluar, dan kebenaran berarti dua syarat terpisah harus sama-sama terpenuhi.

*Gambar 1.2 adalah Fase 1 yang diterapkan pada persoalan yang paling banyak dikaji dalam ilmu komputer; perhatikan bahwa ia menuntut **dua** syarat, bukan satu. Buku: Subbab 1.3.2 dan Gambar 1.2.*

Persoalan Pengurutan

> **PERSOALAN PENGURUTAN.** *Masukan:* sebuah barisan n bilangan $a_1, a_2, \ldots, a_n$. *Keluaran:* sebuah barisan $b_1, b_2, \ldots, b_n$ sedemikian sehingga (i) $b_1 \le b_2 \le \cdots \le b_n$, dan (ii) $b_1, b_2, \ldots, b_n$ adalah permutasi dari $a_1, a_2, \ldots, a_n$.

n adalah banyaknya bilangan; subskrip pada a_1 adalah label, bukan perkalian; **permutasi** adalah penataan ulang — tidak ada yang ditambahkan, tidak ada yang dibuang.

Masukan $[2, 5, 4, 10, 7]$. Keluaran $[2, 4, 5, 7, 10]$ **benar**: berurutan, dan persis bilangan-bilangan yang diberikan.

Keluaran $[1, 2, 3, 4, 5]$ terurut sempurna dan **salah**: 1 dan 3 tidak pernah ada, 7 dan 10 lenyap. Pemeriksa yang hanya bertanya “apakah terurut?” akan menerimanya.

Keluaran $[2, 5, 4, 10, 7]$ memenuhi (ii) dan gagal pada (i). Juga salah. Rumusan persoalanlah, bukan pengujian, yang mendefinisikan kebenaran.

Buku: Subbab 1.3.2.

Insertion Sort dalam C++

Cara hampir semua orang mengurutkan kartu di tangan: ambil kartu berikutnya, geser ke kiri melewati setiap kartu yang lebih besar, lalu jatuhkan ke celahnya. Kode 1.1 adalah seluruhnya.

Kode 1.1 Insertion sort dalam C++, utuh dan dapat dijalankan, mengurutkan barisan dari Gambar 1.2.

```cpp
#include <cstdio>

void insertionSort(int a[], int n) {
    for (int j = 1; j < n; j++) {          // a[0..j-1] sudah terurut
        int key = a[j];                    // kartu yang hendak kita letakkan
        int i = j - 1;
        while (i >= 0 && a[i] > key) {     // geser kartu yang lebih besar satu ke kanan
            a[i + 1] = a[i];
            i = i - 1;
        }
        a[i + 1] = key;                    // celah yang ditinggalkan itulah tempat key
    }
}

int main() {
    int a[] = {2, 5, 4, 10, 7};
    int n = 5;
    insertionSort(a, n);
    for (int i = 0; i < n; i++) printf("%d%c", a[i], i + 1 == n ? '\n' : ' ');
    return 0;
}
```

```
2 4 5 7 10
```

Keluarannya cocok dengan sisi kanan Gambar 1.2. Satu jawaban benar tidak membuktikan apa-apa, maka berikutnya kita buktikan. Buku: Subbab 1.3.3 dan Kode 1.1.

Invarian Gelung dan Pembuktiannya

DEFINISI. Sebuah **invarian gelung** adalah pernyataan tentang peubah-peubah program yang (a) benar sebelum gelung berjalan — *inisialisasi*; (b) tetap benar setelah setiap putaran jika benar sebelumnya — *pemeliharaan*; dan (c) berguna ketika gelung berhenti — *terminasi*.

INVARIAN GELUNG (insertion sort). Di awal setiap putaran dengan pencacah j, sub-larik $a[0 \ldots j-1]$ memuat persis elemen-elemen yang semula menempati posisi 0 sampai $j-1$, dan dalam keadaan terurut.

Inisialisasi. $j = 1$: $a[0 \ldots 0]$ hanya satu elemen — terurut dengan sendirinya, dan belum ada yang berpindah.

Pemeliharaan. Pergeseran tidak menghilangkan apa pun (nilai yang tertimpa sudah disalin ke kanan, atau itulah `key` yang disimpan), dan `key` mendarat sesudah setiap elemen yang $\leq$ dirinya dan sebelum setiap elemen yang $>$ dirinya, sehingga $a[0 \ldots j]$ terurut.

Terminasi. $j = n$: $a[0 \ldots n-1]$ memuat persis elemen-elemen aslinya, berurutan — syarat (ii) dan (i). ∎

Bilangan $2, 5, 4, 10, 7$ *tidak muncul di mana pun dalam pembuktian: ia mencakup setiap masukan berukuran berapa pun. Gagasannya berasal dari Floyd dan Hoare; Gries menjadikannya sebuah disiplin. Buku: Subbab 1.3.3.*

Invarian yang Dibuat Kasatmata

Kode 1.2 Versi penelusuran dari insertion sort yang mencetak invarian gelung sesudah setiap putaran.

```cpp
#include <cstdio>

int a[] = {2, 5, 4, 10, 7};
const int n = 5;

void show(const char* label, int j) {
    printf("%s", label);
    for (int i = 0; i < n; i++) {
        if (i == j) printf(" |");          // garis penanda batas terurut/belum
        printf(" %d", a[i]);
    }
    printf("\n");
}

int main() {
    show("start        ", 1);
    for (int j = 1; j < n; j++) {
        int key = a[j];
        int i = j - 1;
        while (i >= 0 && a[i] > key) { a[i + 1] = a[i]; i = i - 1; }
        a[i + 1] = key;
        printf("after j = %d: ", j);
        for (int t = 0; t < n; t++) {
            if (t == j + 1) printf(" |");
            printf(" %d", a[t]);
        }
        printf("\n");
    }
```

```
    return 0;
}
```

```
start         2 | 5 4 10 7
after j = 1:  2 5 | 4 10 7
after j = 2:  2 4 5 | 10 7
after j = 3:  2 4 5 10 | 7
after j = 4:  2 4 5 7 10
```

Kode 1.2 menggambar sebuah garis di antara bagian yang terurut dan bagian yang belum tersentuh. Bacalah sisi kiri setiap garis: selalu terurut, selalu penghuni aslinya. Buku: Subbab 1.3.3 dan Kode 1.2.

Sasaran Kedua — Efisiensi dan Anggarannya

Bubble sort memerlukan sekitar $n^2 = 10^{12}$ operasi untuk $n = 10^6$ (sejam); *merge sort* sekitar $n \log_2 n \approx 2 \times 10^7$ (dua puluh milidetik). Keduanya benar.

WAWASAN KUNCI — ANGGARAN OPERASI. Komputer modern mengerjakan kira-kira 10^8 sampai 10^9 operasi sederhana per detik. Lebih dari sekitar 10^8 operasi pada masukan terbesar yang diizinkan hampir pasti melampaui batas waktu.

Tabel 1.2 Mengubah sebuah batasan menjadi keputusan rancangan. Angka-angka di kolom tengah dihitung oleh sebuah program C++ pendek; kolom terakhir menerapkan anggaran 10^8 pada angka-angka itu.

n Terbesar	n	$n\log_2 n$	n^2	Apa yang sanggup Anda pakai
10^3	10^3	$\approx 10^4$	10^6	Bahkan $O(n^3)$ masih aman
10^5	10^5	$\approx 1.7 \times 10^6$	10^{10}	$O(n \log n)$ boleh, $O(n^2)$ tidak
10^6	10^6	$\approx 2.0 \times 10^7$	10^{12}	$O(n \log n)$ boleh, $O(n^2)$ tanpa harapan
10^8	10^8	$\approx 2.7 \times 10^9$	10^{16}	Hanya $O(n)$, dan harus hati-hati
10^{18}	—	—	—	Hanya $O(\log n)$ atau $O(1)$

Lihat baris terakhir Tabel 1.2: pada $n = 10^{18}$ Anda bahkan tidak sempat *membaca* n butir, jadi penyelesaiannya harus memanfaatkan struktur. Benar tanpa efisien berarti *Time Limit Exceeded*; efisien tanpa benar berarti *Wrong Answer*. Anda memerlukan keduanya.

Notasi Big-O didefinisikan dengan cermat di Bab 3; untuk sekarang $O(f(n))$ menyatakan bagaimana waktu tumbuh bersama masukan. Buku: Subbab 1.3.4–1.3.6 dan Tabel 1.2–1.3.

Paradigma 1 — Inkremental

Langkahnya. Olah masukan satu elemen demi satu elemen sambil menyimpan sebuah keadaan berjalan; ketika masukan habis, keadaan itulah jawabannya. Kode 1.3 adalah contoh paling sederhana.

Kode 1.3 Paradigma inkremental dalam wujudnya yang paling sederhana: satu peubah berjalan, satu kali pindai.

```
#include <cstdio>

int main() {
    int a[] = {3, 9, 2, 9, 4, 1};
    int n = 6;
    int best = a[0];                        // maksimum dari prefiks satu elemen a[0..0]
    for (int i = 1; i < n; i++)
        if (a[i] > best) best = a[i];       // perluas prefiks sebanyak satu elemen
    printf("maximum = %d\n", best);
    return 0;
}
```

```
maximum = 9
```

Invarian: `best` *memuat nilai terbesar di antara* $a[0 \ldots i-1]$. ***Brute force*** adalah proses inkremental yang mencoba setiap kemungkinan — sering menjadi pikiran pertama yang tepat, karena jelas benarnya.

Columns dan Journey persis berbentuk gelung seperti ini. Buku: Subbab 1.4.1 dan Kode 1.3.

Paradigma 2 — *Greedy*

Langkahnya. Di setiap tahap ambil pilihan yang tampak terbaik *saat ini*, dan jangan pernah meninjaunya kembali.

Mendaki terus dalam kabut membawa Anda ke *sebuah* puncak, belum tentu yang tertinggi. Sikap rakus itu menggoda dan sering keliru.

> **SIFAT PILIHAN RAKUS (*greedy-choice property*).** Mengambil pilihan yang terbaik secara lokal tidak pernah menutup kemungkinan penyelesaian yang terbaik secara global. Dengan sifat ini algoritma *greedy* ringkas dan cepat; tanpanya, ia ringkas, cepat, dan salah.

Membuktikan sifat itu tidak boleh dilewati (Bab 9). Hari ini ia dibuktikan tiga kali: *Columns*, *Journey*, dan angka-angka *Minimum and Maximum Numbers*. Pohon perentang minimum Kruskal adalah contoh klasik yang masih akan datang.

Buku: Subbab 1.4.2.

Paradigma 3 — *Divide and Conquer*

Langkahnya. Pecah menjadi sub-persoalan yang lebih kecil *dari jenis yang sama*, selesaikan secara rekursif, gabungkan jawabannya. Kode 1.4 memperlihatkan ketiga langkahnya.

Kode 1.4 Merge sort: belah jadi dua, urutkan tiap belahan, gabungkan. Paradigma *divide and conquer* dalam dua puluh baris.

```cpp
#include <cstdio>

int tmp[64];

void mergeSort(int a[], int lo, int hi) {   // mengurutkan a[lo..hi]
    if (lo >= hi) return;                   // deret satu elemen sudah pasti terurut
    int mid = lo + (hi - lo) / 2;
    mergeSort(a, lo, mid);
    mergeSort(a, mid + 1, hi);
    int i = lo, j = mid + 1, k = lo;
    while (i <= mid && j <= hi) tmp[k++] = (a[i] <= a[j]) ? a[i++] : a[j++];
    while (i <= mid) tmp[k++] = a[i++];
    while (j <= hi)  tmp[k++] = a[j++];
    for (int t = lo; t <= hi; t++) a[t] = tmp[t];
}

int main() {
    int a[] = {2, 5, 4, 10, 7, 1, 8, 3};
    int n = 8;
    mergeSort(a, 0, n - 1);
    for (int i = 0; i < n; i++) printf("%d%c", a[i], i + 1 == n ? '\n' : ' ');
    return 0;
}
```

```
1 2 3 4 5 7 8 10
```

Bagi: hitung `mid`. Kuasai: dua panggilan rekursif. Gabungkan: gelung-gelung penggabungan. Bab 3 menghitung waktu jalannya dari sebuah rekurensi; Bab 7 mengembangkan paradigma ini. Buku: Subbab 1.4.3 dan Kode 1.4.

Paradigma 4 — *Dynamic Programming*

Langkahnya. Sub-persoalannya **bertumpang tindih**, maka selesaikan masing-masing sekali, simpan, dan cari kembali sesudahnya — **memoisasi**. $F(0)=0$, $F(1)=1$, $F(n)=F(n-1)+F(n-2)$: rekursi naif menghitung ulang nilai yang sama berlipat eksponensial.

Kode 1.5 Rekursi naif dibandingkan rekursi bermemoisasi untuk bilangan Fibonacci. Pencacah pemanggilan membuat selisihnya mustahil dibantah.

```cpp
#include <cstdio>
#include <map>
using namespace std;

long long calls = 0;                        // berapa kali fibNaive dimasuki

long long fibNaive(int n) {
    calls++;
    if (n <= 1) return n;
    return fibNaive(n - 1) + fibNaive(n - 2);
}

map<int, long long> memo;

long long fibMemo(int n) {
    if (n <= 1) return n;
    if (memo.count(n)) return memo[n];      // sudah pernah dihitung: pakai ulang
    return memo[n] = fibMemo(n - 1) + fibMemo(n - 2);
}

int main() {
    calls = 0;
    long long v1 = fibNaive(30);            // selesaikan pemanggilan SEBELUM membaca `calls`
    long long c1 = calls;
    printf("fibNaive(30) = %lld, calls made = %lld\n", v1, c1);

    long long v2 = fibMemo(30);
    int stored = (int)memo.size();
```

```
    printf("fibMemo(30)  = %lld, map entries = %d\n", v2, stored);
    printf("fibMemo(90)  = %lld\n", fibMemo(90));
    return 0;
}
```

```
fibNaive(30) = 832040, calls made = 2692537
fibMemo(30)  = 832040, map entries = 29
fibMemo(90)  = 2880067194370816120
```

Kode 1.5: 2.692.537 panggilan melawan 29 nilai tersimpan. Subbab 1.11 menerapkan gagasan yang sama pada masukan sebesar 10^{18}. Buku: Subbab 1.4.4 dan Kode 1.5.

Paradigma 5 — Matematis

Langkahnya. Ganti pencarian dengan sebuah rumus. **Faktor persekutuan terbesar** $\gcd(a,b)$ adalah bilangan terbesar yang membagi keduanya. Identitas Euklides, Persamaan (1.1):

$$\gcd(a,b)=\gcd(b,a \bmod b)\,(b>0),\quad \gcd(a,0)=a, \tag{1.1}$$

Mengapa: dengan $a=qb+r$, setiap pembagi bersama a,b membagi $r=a-qb$, dan setiap pembagi bersama b,r membagi a. Pembaginya sama, yang terbesar pun sama. Setiap dua langkah membuat bilangan yang lebih besar kurang dari separuhnya, jadi cukup $O(\log \min(a,b))$ langkah.

Kode 1.6 Faktor persekutuan terbesar dihitung dua kali: dengan pencarian menyeluruh, dan dengan algoritma Euclid.

```cpp
#include <cstdio>

long long gcdSlow(long long a, long long b) {   // coba setiap calon pembagi
    long long g = 1;
    for (long long d = 1; d <= a && d <= b; d++)
        if (a % d == 0 && b % d == 0) g = d;
    return g;
}

long long gcdEuclid(long long a, long long b) { // Euclid: gcd(a,b) = gcd(b, a mod b)
    while (b != 0) { long long r = a % b; a = b; b = r; }
    return a;
}

int main() {
    printf("gcdSlow(1071, 462)   = %lld\n", gcdSlow(1071, 462));
    printf("gcdEuclid(1071, 462) = %lld\n", gcdEuclid(1071, 462));
    printf("gcdEuclid(30, 8)     = %lld\n", gcdEuclid(30, 8));
    printf("gcdEuclid(15, 12)    = %lld\n", gcdEuclid(15, 12));
    return 0;
}
```

```
gcdSlow(1071, 462)   = 21
gcdEuclid(1071, 462) = 21
gcdEuclid(30, 8)     = 2
gcdEuclid(15, 12)    = 3
```

Dengan tangan: $1071 \bmod 462 = 147$*;* $462 \bmod 147 = 21$*;* $147 \bmod 21 = 0$*, jadi* $\gcd = 21$ *dalam tiga langkah, sedangkan pencarian lambat pada Kode 1.6 memerlukan* 462*. Ingat angka* 2 *dan* 3 *untuk Subbab 1.8. Buku: Subbab 1.4.5, Persamaan (1.1), dan Kode 1.6.*

Lima Paradigma dalam Satu Tabel

Tabel 1.3 Lima paradigma perancangan, satu langkah yang menjadi cirinya, dan di mana masing-masing muncul dalam bab ini.

Paradigma	Langkah khasnya	Benar ketika	Muncul di bab ini
Inkremental / *brute force*	Perluas jawaban sebagian satu elemen	Hampir selalu; bisa jadi terlalu lambat	*Columns*, *Journey*
Greedy	Ambil pilihan terbaik setempat, jangan ditinjau ulang	Sifat pilihan *greedy* berlaku	*Columns*, *Journey*, *Min/Max Numbers*
Divide and conquer	Belah, rekursi, gabungkan	Sub-persoalannya saling bebas	Merge sort (Kode 1.4)
Dynamic programming	Belah, rekursi, **simpan**	Sub-persoalannya bertumpang tindih	*Recursive Function 1*
Matematis / teori bilangan	Ganti pencarian dengan sebuah rumus	Anda dapat membuktikan rumusnya	*Unique Numbers*, *Interesting Sequence*

Saat buntu, tanyakan berurutan: **bisakah saya sekadar memindai? bisakah saya rakus dan membuktikannya? bisakah saya memecahnya? apakah pecahannya berulang? adakah rumusnya?** Salah satu dari kelimanya hampir selalu berlaku — Tabel 1.3 adalah daftar periksanya.

Buku: Subbab 1.4.6 dan Tabel 1.4.

Kerangka Lima Fase

Tabel 1.4 Lima fase itu. Tidak satu pun di sini yang boleh dilewati, dan urutannya tidak bisa ditawar.

Fase	Nama	Apa yang Anda hasilkan
1	Memahami	Pernyataan ulang dengan kata-kata Anda sendiri, masukan/keluaran/batasan yang persis, dan satu contoh kecil yang dikerjakan dengan tangan
2	Mengabstraksi	Inti matematis persoalan, dengan ceritanya dilucuti habis
3	Merancang	Algoritmanya dalam bahasa sehari-hari atau pseudokode, beserta struktur data yang diperlukannya
4	Menganalisis	Bukti kebenaran, kompleksitas waktu, kompleksitas ruang, dan pemeriksaan terhadap batasannya
5	Mengimplementasikan	Kode yang bersih, terjemahan langsung dari Fase 3

FASE YANG TIDAK DAPAT ANDA TUNJUKKAN HASILNYA ADALAH FASE YANG ANDA LEWATI. Tabel 1.4 menyebutkan apa yang wajib *dihasilkan* setiap fase: tanpa contoh yang dikerjakan dengan tangan berarti tanpa Fase 1; tanpa angka kompleksitas berarti tanpa Fase 4.

Setiap persoalan di buku ini — termasuk setiap persoalan di keempat bab evaluasi — diselesaikan dengan kelima fase ini. Buku: Subbab 1.5 dan Tabel 1.5.

Kelima Fase dalam Praktik

Memahami. Nyatakan ulang persoalannya tanpa melihat; tuliskan masukan, keluaran, dan batasannya; **kerjakan contoh kecil dengan tangan**. Jika hasilnya tidak cocok dengan contoh uji, Anda salah membaca soal — berhenti dan cari sebabnya.

Mengabstraksi. Kupas ceritanya. Buktikan sebuah **batas bawah** L, lalu **bangun** sesuatu yang mencapai L: jawabannya tepat L.

Merancang. Pilih sebuah paradigma; tulis langkah bernomor dalam bahasa sehari-hari atau pseudokode; sebutkan struktur datanya. Belum ada C++.

Menganalisis. Bukti kebenaran; waktu dan ruang dalam Big-O; masukan terbesar dicocokkan dengan 10^8 operasi dan batas memori (10^8 buah `int` berarti 400 MB). Jika tidak muat, kembali ke Fase 2.

Mengimplementasikan. Terjemahan langsung dan ringkas dari Fase 3. Kode yang rumit menandakan rancangan yang belum selesai.

Keenam program berikut masing-masing hanya delapan sampai dua puluh baris — karena pemikirannya dikerjakan lebih dulu. Buku: Subbab 1.5.1–1.5.5.

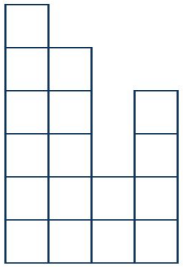
Persoalan 1 — *Columns* (E-Olymp 10281)

Columns (E-Olymp 10281)

Batas waktu 1 dtk Batas memori 128 MB

Pernyataan

Diberikan n kolom kubus; kolom ke-i tingginya a_i. Tentukan banyak warna terkecil yang cukup untuk mewarnai semua kubus sehingga setiap deretan kubus mendatar dalam satu baris (tanpa celah) dan setiap kolom berisi warna yang berbeda-beda.



Masukan

Baris pertama berisi satu bilangan bulat n $(1 \leq n \leq 1000)$, banyaknya kolom. Baris kedua berisi n bilangan bulat $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq 1000)$, tinggi setiap kolom.

Gambar 1.3 *Columns*, soal E-Olymp nomor 10281, dirumuskan ulang dari jurinya: rumusan soalnya, gambar empat kolom kubus, dan bentuk masukannya.

Biarkan kelas membaca Gambar 1.3 sebelum ada yang berbicara. Paradigma: greedy / perhitungan langsung. Buku: Subbab 1.6.1 dan Gambar 1.3.

Columns — Memahami

Ada n kolom kubus, kolom ke-i setinggi a_i, dengan $1 \le n, a_i \le 1000$. Warnai setiap kubus dengan warna sesedikit mungkin: **setiap kolom** dan **setiap *substring*** harus berwarna berbeda semuanya.

SATU-SATUNYA KATA YANG DICIPTAKAN SOAL INI. Sebuah ***substring*** adalah deret mendatar kubus-kubus yang bersebelahan pada satu ketinggian **tanpa sela**. Kolom yang terlalu pendek untuk mencapai ketinggian itu membelah barisnya menjadi dua *substring* yang boleh mengulang warna.

DIKERJAKAN DENGAN TANGAN — tinggi $[6,5,2,4]$, baris teratas dahulu. Baris 6: `[X]`. Baris 5: `[X][X]`. Baris 4 dan 3: `[X][X]` sela `[X]`. Baris 2 dan 1: `[X][X][X][X]`. Kolom tertinggi menumpuk **6** kubus; baris terbawah adalah deret utuh sepanjang **4**. Enam adalah tuntutan yang lebih besar.

Setengah menit dengan pensil, dan gambar itu memuat seluruh penyelesaiannya. Buku: Subbab 1.6.1.

Columns — Contoh Uji dari Juri

Columns (E-Olymp 10281)

Batas waktu 1 dtk Batas memori 128 MB

Keluaran

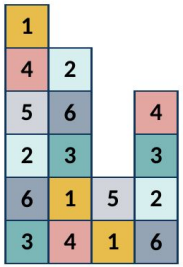
Cetak satu bilangan: banyak warna minimum yang diperlukan.

Contoh

Masukan #1	Jawaban #1
4 6 5 2 4	6

Catatan

Salah satu pewarnaan contoh. Baris ketiga dari bawah memuat warna 3 dua kali; hal itu sah karena ruang kosong di kolom 3 memisahkan kedua kubus tersebut.



Gambar 1.4 Contoh uji *Columns* dan satu pengecatan sah dengan enam warna. Perhatikan baris ketiga dari bawah: baris itu memuat warna 3 dua kali, dan itu sah karena ada ruang kosong yang memisahkan kedua kubusnya.

Baris ketiga dari bawah pada Gambar 1.4 berbunyi 2, 3, *sela,* 3*: warna* 3 *muncul dua kali dalam satu baris, dan itu sah, karena selanya membelah baris tersebut. Buku: Subbab 1.6.1 dan Gambar 1.4.*

Columns — Batas Bawah dan Konstruksi

Tegak: kolom tertinggi memuat $H = \max_i a_i$ kubus yang harus berbeda, jadi $C \geq H$. **Mendatar:** setiap $a_i \geq 1$, jadi baris terbawah adalah satu *substring* sepanjang n, dan $C \geq n$. Bersama-sama, Persamaan (1.2):

$$C \geq \max\left(n, \max_i a_i\right). \tag{1.2}$$

Konstruksi. Dengan $C = \max(n, \max_i a_i)$, kolom $j = 0, \ldots, n-1$ dan baris $r = 0, 1, \ldots$ dari bawah, warnai menurut Persamaan (1.3):

$$\text{warna}(j, r) = ((j + r) \bmod C) + 1. \tag{1.3}$$

Satu kolom memberi $a_j \leq C$ nilai $j + r$ yang **berurutan**; satu deret tanpa sela memberi paling banyak $n \leq C$ nilai berurutan. Bilangan bulat berurutan, paling banyak C buah, bersisa berbeda-beda modulo C. Kedua aturan terpenuhi.

WAWASAN KUNCI. jawaban $= \max(n, \max(a_1, a_2, \ldots, a_n))$ — batas bawah yang tercapai adalah jawaban yang tepat.

Pemeriksa brute force di bab ini (Kode 1.7) menguji setiap kolom dan setiap deret untuk contoh uji itu dan mencetak `VERIFIED: no clash anywhere`*. Buku: Subbab 1.6.2 dan Persamaan (1.2)–(1.3).*

Columns — Merancang, Menganalisis, Mengimplementasikan

Rancangan. $ans \leftarrow n$; untuk setiap tinggi, $ans \leftarrow \max(ans, a_i)$; cetak. Tanpa larik. **Analisis.** Invarian: setelah i tinggi, $ans = \max(n, a_1, \ldots, a_i)$. Waktu $O(n)$, ruang $O(1)$.

Kode 1.7 Penyelesaian lengkap untuk *Columns*, E-Olymp 10281.

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

int n, height, ans;

int main() {
    scanf("%d", &n);
    ans = n;                        // baris terbawah saja sudah memaksa n warna
    for (int i = 0; i < n; i++) {
        scanf("%d", &height);
        ans = max(height, ans);     // kolom setinggi ini memaksa sebanyak height warna
    }
    printf("%d\n", ans);
    return 0;
}
```

```
6
```

Pada contoh uji `4` / `6 5 2 4`, Kode 1.7 mencetak `6`; pada `5` / `1 1 1 1 1` ia mencetak `5`, karena baris terbawah yang berisi lima kubus menuntut lima warna. Buku: Subbab 1.6.3–1.6.5 dan Kode 1.8.

Persoalan 2 — *Journey* (E-Olymp 10280)

Ada n kota pada sebuah **garis lurus**, kota i di x_i; jaraknya $|x_i - x_j|$ (nilai mutlak). Cossack Vus mengunjungi setiap kota lalu kembali ke titik awalnya; ia memilih sendiri titik awal dan rutenya. Laporkan jarak minimumnya. $1 \le n \le 100$, $1 \le x_i \le 1000$.

> **DIKERJAKAN DENGAN TANGAN — kedua contoh uji.** Kota di 1 dan 4: berjalan 3 ke kanan, 3 kembali, total **6**. Kota di $1, 3, 1$: dua kota berbagi koordinat 1; berjalan 2 ke kanan, 2 kembali, total **4**.

Kedengarannya seperti *traveling salesman problem*, salah satu persoalan tersulit dalam ilmu komputer. Pada sebuah garis tidak demikian: gerakannya hanya ke kiri dan ke kanan.

Paradigma: greedy / pengamatan matematis. Buku: Subbab 1.7.1 dan Gambar 1.5–1.6.

Journey — Batas dan Pembuktiannya

Misalkan $R = x_{\max} - x_{\min}$, yaitu **rentang**. Rute tertutup adalah sebuah putaran, jadi bacalah mulai saat ia berdiri di $x_{\min}$: ia **berangkat** ke $x_{\max}$ (sekurang-kurangnya R) lalu **kembali** ke $x_{\min}$ (sekurang-kurangnya R). Jadi setiap rute berbiaya sekurang-kurangnya $2R$.

Dengan kontradiksi: rute yang lebih pendek daripada $2R$ memiliki potongan berangkat atau kembali yang lebih pendek daripada R, padahal masing-masing menghubungkan dua titik yang berjarak tepat R. Mustahil.

Tercapai: dari $x_{\min}$ sapu ke kanan sampai $x_{\max}$ lalu kembali — $2R$, melewati setiap kota. Dari titik awal s mana pun: $(s - x_{\min}) + (x_{\max} - x_{\min}) + (x_{\max} - s)$; suku-suku s saling meniadakan, menghasilkan Persamaan (1.4):

$$-x_{\min} + x_{\max} - x_{\min} + x_{\max} = 2(x_{\max} - x_{\min}) = 2R. \tag{1.4}$$

WAWASAN KUNCI. jawaban $= 2 \times (\max_i x_i - \min_i x_i)$. Contoh uji: $2(4-1) = 6$ dan $2(3-1) = 4$.

Banyaknya kota tidak berpengaruh; hanya kedua titik ekstrem yang penting. Buku: Subbab 1.7.2 dan Persamaan (1.4).

Journey — Programnya

Rancangan. Mulai dengan $d_{\min}=2000$ (di atas setiap koordinat) dan $d_{\max}=0$ (di bawah setiap koordinat); perbarui keduanya untuk setiap koordinat; cetak $2(d_{\max}-d_{\min})$. Invarian: keduanya adalah ekstrem dari koordinat yang sudah dibaca. Waktu $O(n)$, ruang $O(1)$.

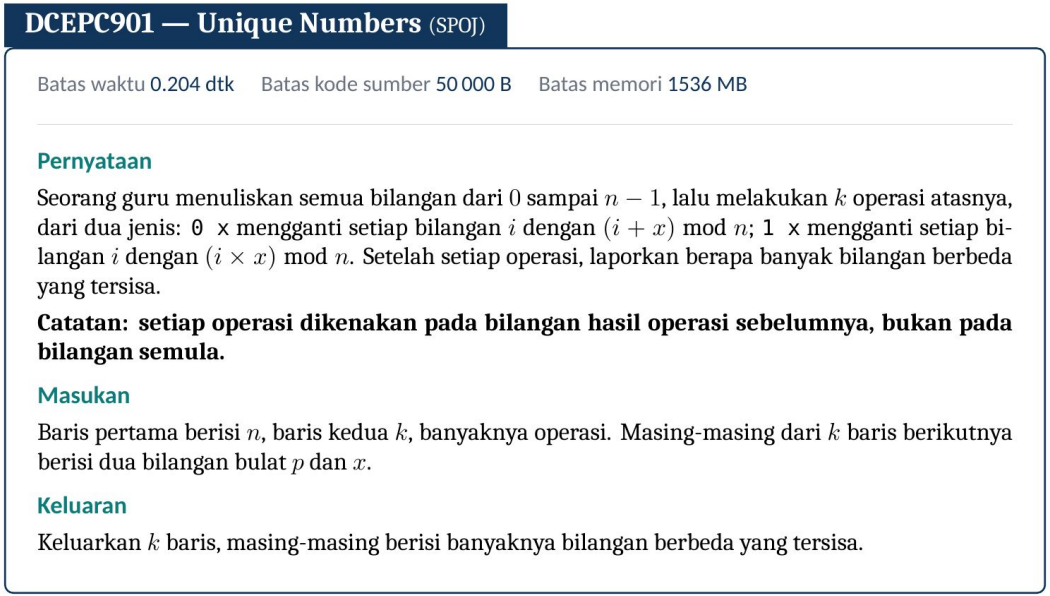
Kode 1.8 Penyelesaian lengkap untuk *Journey*, E-Olymp 10280.

```cpp
#include <cstdio>
using namespace std;

int main() {
    int n, dist;
    scanf("%d", &n);
    int dmin = 2000, dmax = 0;     // 2000 di atas semua koordinat sah, 0 di bawah semuanya
    for (int i = 0; i < n; i++) {
        scanf("%d", &dist);
        dmin = dmin < dist ? dmin : dist;
        dmax = dmax < dist ? dist : dmax;
    }
    printf("%d\n", 2 * (dmax - dmin));
    return 0;
}
```

`dmin < dist ? dmin : dist` adalah **operator kondisional**: nilai sebelum `:` jika ujinya benar, nilai sesudahnya jika tidak — di sini, yang lebih kecil dari keduanya.

Kode 1.8 mencetak `6` pada contoh uji 1 dan `4` pada contoh uji 2. Buku: Subbab 1.7.3–1.7.5 dan Kode 1.9.

Persoalan 3 — *Unique Numbers* (SPOJ DCEPC901)



DCEPC901 — Unique Numbers (SPOJ)

Batas waktu 0.204 dtk Batas kode sumber 50 000 B Batas memori 1536 MB

Pernyataan

Seorang guru menuliskan semua bilangan dari 0 sampai $n-1$, lalu melakukan k operasi atasnya, dari dua jenis: `0 x` mengganti setiap bilangan i dengan $(i+x)$ mod n; `1 x` mengganti setiap bilangan i dengan $(i \times x)$ mod n. Setelah setiap operasi, laporkan berapa banyak bilangan berbeda yang tersisa.

Catatan: setiap operasi dikenakan pada bilangan hasil operasi sebelumnya, bukan pada bilangan semula.

Masukan

Baris pertama berisi n, baris kedua k, banyaknya operasi. Masing-masing dari k baris berikutnya berisi dua bilangan bulat p dan x.

Keluaran

Keluarkan k baris, masing-masing berisi banyaknya bilangan berbeda yang tersisa.

Gambar 1.5 *Unique Numbers*, soal SPOJ DCEPC901, dengan dua jenis operasinya dan catatan penting bahwa operasinya beruntun saling menyusul.

Perhatikan kalimat bercetak tebal pada halaman juri di Gambar 1.5: operasinya bersambung. Paradigma: teori bilangan, faktor persekutuan terbesar. Buku: Subbab 1.8.1 dan Gambar 1.7.

Unique Numbers — Memahami

Mulai dengan $0, 1, \ldots, n-1$. Terapkan k operasi (p, x), masing-masing pada hasil operasi sebelumnya: $p=0$ mengganti setiap i dengan $(i+x) \bmod n$; $p=1$ mengganti setiap i dengan $(i \times x) \bmod n$.

Setelah setiap operasi, cetak banyaknya nilai yang **berbeda** yang tersisa. Batasan: $1 \le n \le 10^8$, $1 \le k \le 20$, $0 \le x \le 10^8$.

Contoh uji: $n=30$, operasi `1 8`, `0 3`, `1 12`; keluaran yang diharapkan 15, 15, 5.

> **BACALAH** $n \le 10^8$ **DENGAN** Tabel 1.2**.** Daftarnya memerlukan 400 MB dan 20 kali penelusuran berarti 2×10^9 langkah. Penyelesaian yang dimaksud tidak pernah membangun daftar itu.

Buku: Subbab 1.8.1 dan Gambar 1.8.

Unique Numbers — Mengabstraksi

Penjumlahan tidak pernah mengubah banyaknya. Jika $(a+x)\equiv(b+x)\pmod n$ maka $a\equiv b$, dan dengan $a,b\in\{0,\ldots,n-1\}$ itu memaksa $a=b$. Pemetaannya injektif, jadi pada himpunan berhingga ia bijektif: sekadar mengocok.

Perkalian menyusutkan semestanya. Dengan $g=\gcd(n,x)$, setiap keluaran $i\,x \bmod n$ adalah kelipatan g; menurut Bézout ($u\,x+v\,n=g$) setiap kelipatan g tercapai. Jadi, menurut Persamaan (1.5):

$$\left|\{(i\times x)\bmod n : 0\le i\le n-1\}\right|=\frac{n}{\gcd(n,x)}, \tag{1.5}$$

Rangkaikan. Yang tersisa, $0,g,2g,\ldots$, berperilaku seperti $\{0,\ldots,n'-1\}$ yang baru dengan $n'=n/g$, maka ganti n dengan n' dan lanjutkan.

> **WAWASAN KUNCI.** Simpan n sebagai peubah berjalan: pada perkalian, $n\leftarrow n/\gcd(n,x)$; pada penjumlahan, tidak berbuat apa-apa; cetak n.

Kita tidak pernah melacak bilangan-bilangannya, hanya banyaknya. Buku: Subbab 1.8.2 dan Persamaan (1.5).

Unique Numbers — Perkalian Modulo 15

op	x	$(i \times x) \bmod 15$														
i		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3	0	3	6	9	12	0	3	6	9	12	0	3	6	9	12
i		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	5	0	5	10	0	5	10	0	5	10	0	5	10	0	5	10
i		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	0	2	4	6	8	10	12	14	1	3	5	7	9	11	13
i		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11

Gambar 1.6 Perkalian modulo 15. Setiap pasangan baris menunjukkan masukan $0..14$ di atas dan keluaran $(i \times x) \bmod 15$ di bawah, untuk $x = 3, 5, 2, 4$ berturut-turut.

Cocokkan Gambar 1.6 dengan Persamaan (1.5): $x = 3$ memberi lima nilai ($15/3$); $x = 5$ tiga ($15/5$); $x = 2$ dan $x = 4$ kelima belasnya ($\gcd = 1$). Empat konfirmasi dengan tangan sebelum ada kode. Buku: Subbab 1.8.2 dan Gambar 1.9.

Unique Numbers — Dikerjakan, Lalu Dikodekan

DIKERJAKAN DENGAN TANGAN — contoh uji, $n=30$. `1 8`: $\gcd(30,8)$: $30 \bmod 8=6$, $8 \bmod 6=2$, $6 \bmod 2=0$, jadi 2; $n=15$, **cetak 15**. `0 3`: sebuah bijeksi, **cetak 15**. `1 12`: $\gcd(15,12)=3$; $n=5$, **cetak 5**.

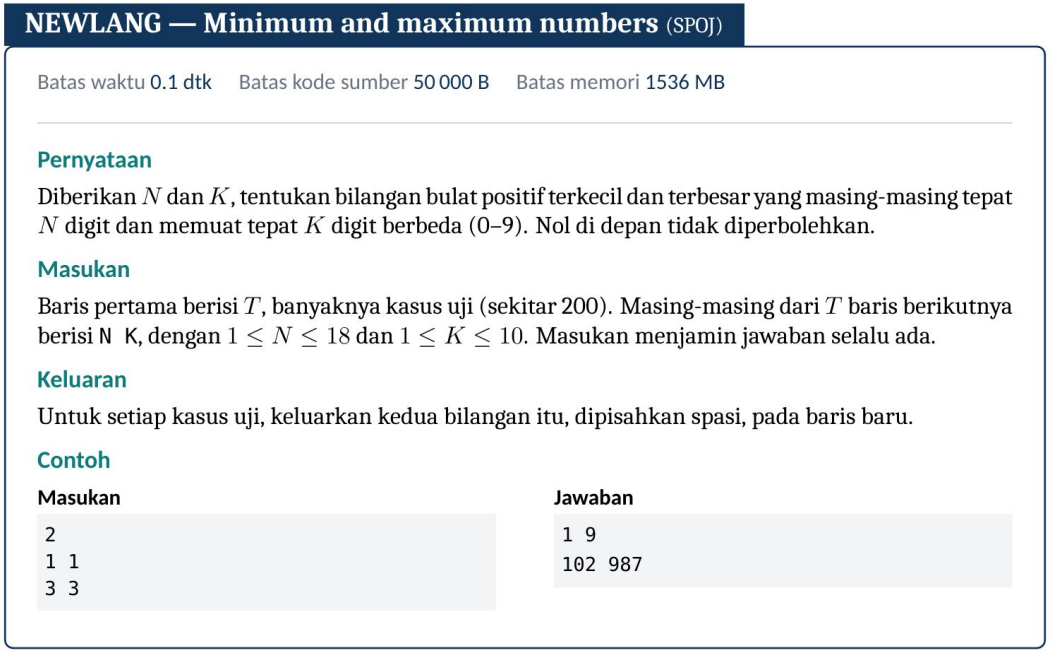
Kode 1.9 Penyelesaian lengkap untuk *Unique Numbers*, SPOJ DCEPC901.

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

int main() {
    int n, k, p, x;
    scanf("%d %d", &n, &k);
    while (k--) {
        scanf("%d %d", &p, &x);
        if (p & 1) n /= __gcd(n, x);   // hanya perkalian yang menciutkan semesta
        printf("%d\n", n);
    }
    return 0;
}
```

JEBAKAN YANG PERLU DIKETAHUI. $x=0$ diizinkan. Menurut kesepakatan $\gcd(n,0)=n$, sehingga $n \leftarrow 1$ — tepat satu nilai yang tersisa, dan banyaknya tidak akan pernah naik lagi.

Kode 1.9 mencetak `15`, `15`, `5` *pada contoh uji, serta* `1`, `1` *pada* `30 / 2 / 1 0 / 0 5`. `__gcd` *adalah Euklides versi GNU (*`std::gcd` *di C++17);* `p & 1` *menguji keganjilan. Waktu* $O(k \log n)$, *ruang* $O(1)$. *Buku: Subbab 1.8.3–1.8.5 dan Kode 1.10.*

Persoalan 4 — *Minimum and Maximum Numbers* (SPOJ NEWLANG)



NEWLANG — Minimum and maximum numbers (SPOJ)

Batas waktu 0.1 dtk Batas kode sumber 50 000 B Batas memori 1536 MB

Pernyataan

Diberikan N dan K, tentukan bilangan bulat positif terkecil dan terbesar yang masing-masing tepat N digit dan memuat tepat K digit berbeda (0–9). Nol di depan tidak diperbolehkan.

Masukan

Baris pertama berisi T, banyaknya kasus uji (sekitar 200). Masing-masing dari T baris berikutnya berisi `N K`, dengan $1 \le N \le 18$ dan $1 \le K \le 10$. Masukan menjamin jawaban selalu ada.

Keluaran

Untuk setiap kasus uji, keluarkan kedua bilangan itu, dipisahkan spasi, pada baris baru.

Contoh

Masukan	Jawaban
2 1 1 3 3	1 9 102 987

Gambar 1.7 *Minimum and maximum numbers*, soal SPOJ NEWLANG, dengan batasan dan contoh ujinya.

Gambar 1.7. Paradigma: konstruksi greedy, dengan pembuktian satu kalimat bahwa pilihan rakusnya aman. Buku: Subbab 1.9.1 dan Gambar 1.10.

Min dan Max — Memahami, dan Mengapa *Greedy* Aman

Diberikan N dan K ($1 \le N \le 18$, $1 \le K \le 10$), cetak bilangan N angka terkecil dan terbesar yang memakai **tepat K angka berbeda**, tanpa nol di depan. Contoh uji: `1 1` memberi `1 9`; `3 3` memberi `102 987`.

Keberbedaan menyangkut *himpunan* angkanya: 121 memakai $\{1,2\}$ — dua. Periksa: 100 dan 101 hanya memakai $\{0,1\}$, jadi 102 yang pertama; 988–999 paling banyak memuat dua angka berbeda, jadi 987 yang terakhir.

> **MENGAPA *GREEDY* AMAN DI SINI.** Angka terdepan berbobot 10^{N-1}, lebih besar daripada semua angka di kanannya ($10^{N-1}-1$ paling banyak). Jadi angka sah terbaik untuk posisi 1 tidak pernah keliru, dan hal yang sama berlaku untuk posisi 2 di antara $2..N$, dan seterusnya.

$K=1$**:** satu angka bukan nol yang diulang — $\underbrace{11\cdots1}_{N}$ dan $\underbrace{99\cdots9}_{N}$.

Kita mencetak angka-angkanya sebagai karakter, jadi besarnya bilangan tidak pernah menjadi soal. Buku: Subbab 1.9.1–1.9.2.

Min dan Max — Dua Konstruksinya

Minimum, $K \geq 2$**:** awali dengan 1, lalu 0 sebanyak mungkin — $(N-1)-(K-2) = N-K+1$ — lalu $2, 3, \ldots, K-1$ menaik, Persamaan (1.6):

$$\text{minimum} \;=\; 1 \;\underbrace{00\cdots0}_{N-K+1}\; 2\,3\,\cdots\,(K-1). \tag{1.6}$$

Maksimum, $K \geq 2$**:** $N-K+1$ angka sembilan, lalu $8, 7, \ldots, 10-K$ menurun, Persamaan (1.7):

$$\text{maksimum} \;=\; \underbrace{99\cdots9}_{N-K+1}\; 8\,7\,\cdots\,(10-K). \tag{1.7}$$

DIKERJAKAN DENGAN TANGAN — $N=5$, $K=3$. Minimum: 1, tiga 0, lalu 2: **10002**. Maksimum: tiga 9, lalu $8, 7$: **99987**. Keduanya lima angka dengan tepat tiga angka berbeda.

Banyak angka: $1+(N-K+1)+(K-2) = N$ *dan* $(N-K+1)+(K-1) = N$*; angka berbeda* $\{0, \ldots, K-1\}$ *dan* $\{10-K, \ldots, 9\}$*. Buku: Subbab 1.9.2 dan Persamaan (1.6)–(1.7).*

Min dan Max — Tabelnya

Minimum

$K \backslash N$	1	2	3	4	5	6	7	8	9	10
1	1	11	111	1111	11111	111111	1111111	11111111	111111111	1111111111
2	–	10	100	1000	10000	100000	1000000	10000000	100000000	1000000000
3	–	–	102	1002	10002	100002	1000002	10000002	100000002	1000000002
4	–	–	–	1023	10023	100023	1000023	10000023	100000023	1000000023
5	–	–	–	–	10234	100234	1000234	10000234	100000234	1000000234

Maksimum

$K \backslash N$	1	2	3	4	5	6	7	8	9	10
1	9	99	999	9999	99999	999999	9999999	99999999	999999999	9999999999
2	–	98	998	9998	99998	999998	9999998	99999998	999999998	9999999998
3	–	–	987	9987	99987	999987	9999987	99999987	999999987	9999999987
4	–	–	–	9876	99876	999876	9999876	99999876	999999876	9999999876
5	–	–	–	–	98765	998765	9998765	99998765	999998765	9999998765

Gambar 1.8 Nilai minimum (atas, berarsir kuning) dan maksimum (bawah, berarsir kelabu) untuk $N = 1..10$ dan $K = 1..5$. Setiap baris adalah satu nilai K; bacalah melintang untuk N yang membesar.

Baris $K = 5$ pada Gambar 1.8 untuk $N = 10$: $1\,000000\,234$ dan $999999\,8765$ — enam nol atau enam sembilan, persis seperti ramalan Persamaan (1.6) dan Persamaan (1.7). Buku: Subbab 1.9.2 dan Gambar 1.11.

Min dan Max — Programnya

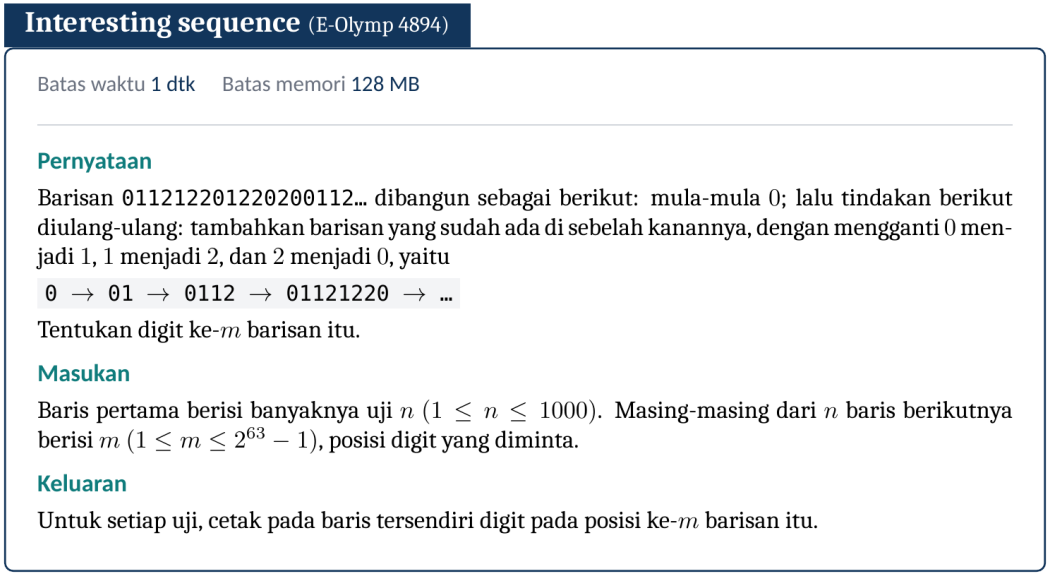
Kode 1.10 Penyelesaian lengkap untuk *Minimum and maximum numbers*, SPOJ NEWLANG.

```cpp
#include <cstdio>

int main() {
    int t, n, k;
    scanf("%d", &t);
    while (t--) {
        scanf("%d %d", &n, &k);
        if (k == 1) {
            for (int i = 1; i <= n; i++) printf("1");
            printf(" ");
            for (int i = 1; i <= n; i++) printf("9");
            printf("\n");
            continue;
        }
        printf("1");
        for (int i = 1; i <= n - k + 1; i++) printf("0");
        for (int i = 1; i <= k - 2; i++) printf("%d", i + 1);
        printf(" ");
        for (int i = 1; i <= n - k + 1; i++) printf("9");
        for (int i = 1; i <= k - 1; i++) printf("%d", 9 - i);
        printf("\n");
    }
    return 0;
}
```

```
1 9
102 987
```

`i + 1` *untuk* $i = 1..K-2$ *mencetak* $2..K-1$ *menaik;* `9 - i` *untuk* $i = 1..K-1$ *mencetak* $8..10-K$ *menurun; pada* $K = 2$ *gelung pertama berjalan nol kali. Kode 1.10 juga mencetak* `10002 99987` *untuk* `5 3`*. Waktu* $O(T\,N)$*. Buku: Subbab 1.9.3–1.9.4 dan Kode 1.11.*

Persoalan 5 — *Interesting Sequence* (E-Olymp 4894)



Interesting sequence (E-Olymp 4894)

Batas waktu 1 dtk Batas memori 128 MB

Pernyataan

Barisan `011212201220200112`… dibangun sebagai berikut: mula-mula 0; lalu tindakan berikut diulang-ulang: tambahkan barisan yang sudah ada di sebelah kanannya, dengan mengganti 0 menjadi 1, 1 menjadi 2, dan 2 menjadi 0, yaitu

$0 \to 01 \to 0112 \to 01121220 \to \ldots$

Tentukan digit ke-m barisan itu.

Masukan

Baris pertama berisi banyaknya uji n ($1 \le n \le 1000$). Masing-masing dari n baris berikutnya berisi m ($1 \le m \le 2^{63} - 1$), posisi digit yang diminta.

Keluaran

Untuk setiap uji, cetak pada baris tersendiri digit pada posisi ke-m barisan itu.

Gambar 1.9 *Interesting sequence*, soal E-Olymp nomor 4894. Barisannya dibangun dengan berulang kali menempelkan salinan dirinya sendiri yang sudah digeser.

Frasa yang perlu diperhatikan pada Gambar 1.9 adalah "tempelkan barisan itu pada dirinya sendiri": berlipat dua, yang berarti biner. Buku: Subbab 1.10.1 dan Gambar 1.12.

Interesting Sequence — Memahami

Mulai dengan `0`. Ulangi selamanya: tempelkan salinan barisan itu dengan $0 \to 1$, $1 \to 2$, $2 \to 0$.

> **DENGAN TANGAN.** `0` → `01` → `0112` → `01121220` → `0112122012202001`. Ditambah dua karakter tahap berikutnya, `0112122012202001120` — persis awalan yang tertulis pada soal.

Tugas: diberikan m, cetak angka ke-m (dihitung dari 1). Sampai 1000 kueri, $1 \le m \le 2^{63} - 1 \approx 9.22 \times 10^{18}$.

Contoh uji: $m = 17,239,385$ memberi $1, 0, 2$. Menurut Tabel 1.2, tidak ada yang sanggup menghitung sampai m: pasti ada rumusnya.

Buku: Subbab 1.10.1 dan Gambar 1.13.

Interesting Sequence — Mengabstraksi

Beri nama transformasinya: $0 \to 1 \to 2 \to 0$ tidak lain adalah $v \mapsto (v+1) \bmod 3$.

Rekursi pada indeks p (dari 0): setelah k tahap ada 2^k angka. Jika $p < 2^k$, angkanya ada di paruh pertama dan tidak berubah; jika $p = 2^k + q$, angkanya adalah angka di q ditambah satu, modulo 3.

Dalam **biner**, "apakah $p \geq 2^k$?" menanyakan apakah bit terdepannya 1, dan $q = p - 2^k$ menghapus bit itu. Setiap bit 1 menambah satu; setiap bit 0 tidak menambah apa-apa. Maka Persamaan (1.8):

$$\text{nilai}(p) = (\text{banyaknya bit 1 dalam bentuk biner } p) \bmod 3. \tag{1.8}$$

WAWASAN KUNCI. Angka-angkanya dinomori dari 1, jadi $\text{jawaban} = \text{popcount}(m-1) \bmod 3$.

Banyaknya bit 1 *disebut population count, atau popcount. Buku: Subbab 1.10.2 dan Persamaan (1.8).*

Interesting Sequence — Bukti dengan Induksi

Misalkan S_k barisan setelah k tahap dan $P(p)$ *popcount* dari p. Klaim: $S_k[p] = P(p) \bmod 3$ untuk $0 \le p < 2^k$.

Basis. $S_0 = [0]$ dan $P(0) = 0$. Benar.

Langkah, paruh pertama. Untuk $p < 2^k$, $S_{k+1}[p] = S_k[p] = P(p) \bmod 3$.

Langkah, paruh kedua. Untuk $p = 2^k + q$ dengan $q < 2^k$, bit berbobot 2^k pada q bernilai 0, sehingga $P(p) = P(q) + 1$, dan $S_{k+1}[p] = (P(q) \bmod 3 + 1) \bmod 3 = P(p) \bmod 3$. ∎

Bab ini juga membangun 128 suku sungguhan secara brute force (Kode 1.12) dan tidak menemukan satu pun ketidakcocokan dengan Persamaan (1.8). Buku: Subbab 1.10.2.

Interesting Sequence — Tabelnya Adalah Barisannya

N	Biner				#bit-1	(#bit-1) mod 3
0	0	0	0	0	0	**0**
1	0	0	0	1	1	**1**
2	0	0	1	0	1	**1**
3	0	0	1	1	2	**2**
4	0	1	0	0	1	**1**
5	0	1	0	1	2	**2**
6	0	1	1	0	2	**2**
7	0	1	1	1	3	**0**
8	1	0	0	0	1	**1**
9	1	0	0	1	2	**2**
10	1	0	1	0	2	**2**
11	1	0	1	1	3	**0**
	MSB			LSB		

Gambar 1.10 Indeks 0 sampai 11 dalam biner, dengan banyaknya bit 1 dan cacah itu modulo 3. Kolom terakhir adalah barisannya sendiri.

Kolom terakhir Gambar 1.10 berbunyi $0,1,1,2,1,2,2,0,1,2,2,0$ *— dua belas angka pertama yang kita bangun dengan tangan. Buku: Subbab 1.10.2 dan Gambar 1.14.*

Interesting Sequence — Dikerjakan, Lalu Dikodekan

DIKERJAKAN DENGAN TANGAN — contoh uji. $m=17$: $p=16=10000_2$, satu bit, **1**. $m=239$: $p=238=128+64+32+8+4+2=11101110_2$, enam bit, **0**. $m=385$: $p=384=256+128=110000000_2$, dua bit, **2**.

Kode 1.11 Penyelesaian lengkap untuk *Interesting sequence*, E-Olymp 4894.

```cpp
#include <iostream>
using namespace std;
typedef unsigned long long LL;

int main() {
    LL n, x;
    cin >> n;
    while (n--) {
        cin >> x;
        LL tmp = __builtin_popcountll(x - 1);   // m berindeks-1 menjadi m-1 berindeks-0
        cout << tmp % 3 << endl;
    }
    return 0;
}
```

```
1
0
2
```

`unsigned long long` *mencapai* $2^{64}-1$*;* *sebuah* `int` *akan meluap.* `__builtin_popcountll` *adalah satu instruksi mesin (*`std::popcount` *di C++20).* *Kode 1.11 mencetak* `0 1 1 2 1 2 2 0` *untuk* $m = 1..8$ *— barisan tahap 3. Buku: Subbab 1.10.3–1.10.4 dan Kode 1.13.*

Persoalan 6 — *Recursive Function 1* (E-Olymp 10296)

Recursive function 1 (E-Olymp 10296)

Batas waktu 1 dtk Batas memori 128 MB

Pernyataan

Tentukan nilai fungsi

$$f(n) = \begin{cases} 1, & n = 0, \\ f\left(\lfloor \frac{n}{2} \rfloor\right) + f\left(\lfloor \frac{n}{3} \rfloor\right), & n > 0. \end{cases}$$

Masukan

Satu bilangan bulat positif n $(1 \le n \le 10^{18})$.

Keluaran

Cetak nilai $f(n)$.

Contoh

Masukan #1	Jawaban #1
5	5

Gambar 1.11 *Recursive function 1*, soal E-Olymp nomor 10296. Definisinya, masukannya yang berupa satu bilangan bulat, dan contoh uji $f(5)=5$.

Gambar 1.11. Paradigma: dynamic programming lewat memoisasi — sekaligus pelajaran tentang memilih wadah. Buku: Subbab 1.11.1 dan Gambar 1.15.

Recursive Function 1 — Memahami

Fungsinya, Persamaan (1.9), untuk satu masukan n dengan $1 \le n \le 10^{18}$ dan batas waktu satu detik:

$$f(n) = \begin{cases} 1, & n = 0, \\ f\left(\lfloor \frac{n}{2} \rfloor\right) + f\left(\lfloor \frac{n}{3} \rfloor\right), & n > 0. \end{cases} \tag{1.9}$$

$\lfloor x \rfloor$, yaitu **lantai** (*floor*), adalah bilangan bulat terbesar yang tidak melebihi x: $\lfloor 7/2 \rfloor = 3$, $\lfloor 1/2 \rfloor = 0$. Di C++, `n / 2` pada nilai tak negatif sudah sama dengan $\lfloor n/2 \rfloor$.

DIKERJAKAN DENGAN TANGAN — $n=5$. $f(5)=f(2)+f(1)$; $f(2)=f(1)+f(0)$; $f(1)=f(0)+f(0)=2$; jadi $f(2)=3$ dan $f(5)=3+2=5$. Perhatikan bahwa $f(1)$ diperlukan **dua kali**.

Buku: Subbab 1.11.1 dan Persamaan (1.9).

Recursive Function 1 — Menghitung Pekerjaannya

Panggilan naif. Pada pohon panggilan setiap simpul dalam punya dua anak dan setiap daun $f(0)$ mengembalikan 1, jadi $f(n)$ adalah banyaknya daun dan banyaknya panggilan adalah Persamaan (1.10):

$$\text{panggilan}(n) = f(n) + (f(n) - 1) = 2f(n) - 1. \tag{1.10}$$

Untuk $n = 5$: $2 \cdot 5 - 1 = 9$ panggilan, cocok dengan penelusuran tangan. Untuk $n = 10^{18}$, $f \approx 2.24 \times 10^{14}$, jadi sekitar 4.5×10^{14} panggilan — kira-kira **lima puluh dua hari**.

Sub-persoalan yang berbeda berbentuk Persamaan (1.11), dengan $a \le 60$ dan $b \le 38$: paling banyak $61 \times 39 = 2{,}379$.

$$\left\lfloor \frac{n}{2^a 3^b} \right\rfloor \text{ untuk bilangan bulat } a, b \ge 0. \tag{1.11}$$

Penelusuran berbasis `set` dari 10^{18} (Kode 1.14) menemukan tepat **1.012**. Larik berindeks n memerlukan delapan juta terabita; `std::map` hanya menyimpan kunci yang dijumpai.

Algoritmanya tidak berubah; struktur datanya yang berubah. Buku: Subbab 1.11.2 dan Persamaan (1.10)–(1.11).

Recursive Function 1 — Isi Tembolok

DIKERJAKAN DENGAN TANGAN — $f(12)$. $f(12)=f(6)+f(4)$; $f(6)=f(3)+f(2)$; $f(3)=f(1)+f(1)$; $f(1)=2$ (**simpan**); $f(3)=4$ (**simpan**); $f(2)=2+1=3$ (**simpan**); $f(6)=7$ (**simpan**); $f(4)=f(2)+f(1)=5$, keduanya dari tembolok; $f(12)=7+5=12$.

Tabel 1.5 Tabel memoisasi yang terbentuk sewaktu menghitung $f(12)$, dalam urutan entrinya terisi.

u	$\lfloor u/2 \rfloor$	$\lfloor u/3 \rfloor$	$f(u)$	Kapan ia masuk cache
0	—	—	1	ditanam sebelum program berjalan
1	0	0	1+1=2	pertama
2	1	0	2+1=3	ketiga
3	1	1	2+2=4	kedua
4	2	1	3+2=5	kelima
6	3	2	4+3=7	keempat
12	6	4	7+5=12	terakhir

Tabel 1.5 adalah penelusuran yang sama dalam wujud tabel yang dipegang program: tujuh nilai berbeda, yang terdalam lebih dulu. $f(12)=12$ *hanyalah kebetulan:* $f(11)=9$. *Buku: Subbab 1.11.2 dan Tabel 1.6.*

Recursive Function 1 — Programnya

Kode 1.12 Penyelesaian lengkap untuk *Recursive function 1*, E-Olymp 10296.

```cpp
#include <map>
#include <cstdio>
using namespace std;
typedef long long ll;

map<ll, ll> mp;

ll rec(ll u) {
    if (mp.count(u)) return mp[u];      // sudah terpecahkan: serahkan jawaban tersimpan
    ll nval = rec(u / 2) + rec(u / 3);
    mp[u] = nval;
    return nval;
}

int main() {
    mp[0] = 1;                          // kasus dasar, ditanam ke dalam cache
    ll u;
    scanf("%lld", &u);
    printf("%lld\n", rec(u));
    return 0;
}
```

Tidak ada `if (u == 0) return 1;`: dengan mengisi `mp[0] = 1` lebih dulu, uji tembolok ikut menangkap kasus dasarnya.

Benar menurut induksi kuat, karena $\lfloor u/2 \rfloor, \lfloor u/3 \rfloor < u$. **Waktu** $O(D \log D)$ dengan $D = 1{,}012$, yaitu $O(\log^2 n \cdot \log \log n)$. **Muat:** $f(10^{18}) = 224{,}412{,}958{,}156{,}019$, jauh di dalam jangkauan `long long`.

Kode 1.12 mencetak `5` pada contoh uji, `12` untuk `12`, dan `224412958156019` seketika untuk 10^{18}. Buku: Subbab 1.11.3–1.11.4 dan Kode 1.15.

Keenam Persoalan dalam Sekilas Pandang

Tabel 1.6 Keenam persoalan bab ini, paradigmanya, dan satu kalimat yang menyelesaikan masing-masing.

Persoalan	Paradigma	Teknik intinya	Waktu	Jawabannya dalam satu kalimat
Columns	*Greedy*	Maksimum berjalan	$O(n)$	$\max(\text{cacah kolom}, \text{tinggi tertinggi})$
Journey	*Greedy*	Rentang lariknya	$O(n)$	$2 \times (\max - \min)$ pada sebuah garis
Unique Numbers	Teori bilangan	Reduksi dengan gcd	$O(k \log n)$	Sebuah perkalian menciutkan semesta sebesar gcd-nya
Min/Max Numbers	*Greedy* konstruktif	Penempatan angka secara rakus	$O(N)$	Angka ekstrem di ujung yang paling berarti
Interesting Sequence	Trik bit	Popcount modulo 3	$O(1)$	Pelipatduaan berarti biner; hitung bit 1-nya
Recursive Function 1	Memoisasi	*Dynamic programming* berbasis map	$O(\log^2 n)$	Sekitar seribu sub-persoalan walaupun $n = 10^{18}$

Pada Tabel 1.6 tidak ada yang lebih buruk daripada $O(n)$ dan tidak ada program yang lebih dari dua puluh baris — namun masing-masing menuntut wawasan sejati di Fase 2. Frasa pada soal menunjuk ke tekniknya: “minimum/maksimum” → satu kali lintas; “sisa bagi” → gcd; “berlipat dua setiap tahap” → biner; “$f(n)$ dari $f(n/a)$” → memoisasi dalam `map`.

Tabel gejala dan diagnosis di bab ini (Tabel 1.8) memuat delapan frasa semacam itu. Buku: Subbab 1.12–1.12.1 dan Tabel 1.7–1.8.

Juri, Penelitian, dan Peta Jalan

Sebuah ***online judge*** mengompilasi program Anda, menjalankannya pada uji rahasia, dan mengembalikan vonis: *Accepted*, *Wrong Answer*, *Time Limit Exceeded*, *Memory Limit Exceeded*, *Runtime Error*, *Compilation Error*. Kita memakai **E-Olymp** dan **SPOJ**; UVa dan beecrowd juga patut dikenal.

Pada *Wrong Answer*, kembali ke Fase 2; pada *Time Limit Exceeded*, kembali ke Fase 4. Jangan mengocok kode lalu mengirim ulang.

Soal juri juga penelitian: makalah ITS tentang SPOJ PERIOD5 (ekstrapolasi periodik, peringkat ke-2) dan HIR (pengenalan angka di bawah 0,1 detik dan 150 kB, dua entri teratas).

Buku acuan: **CLRS** edisi ke-4 membuktikan; **CP4** melatih. Enam belas bab; Bab 4, 8, 12, dan 16 adalah **Evaluasi dan Pembahasan I–IV**, yang menyajikan persoalan atas tiga bab sebelumnya dan membahas masing-masing secara tuntas.

Buku: Subbab 1.13–1.15, Gambar 1.16–1.24, dan Tabel 1.9–1.12.

Meja Kerja Anda — Tiga Sistem

Dua hal: sebuah **kompilator** (`g++` dari GCC di Windows dan Linux, `clang++` dari Clang di macOS, keduanya C++17) dan sebuah **editor** (VS Code). Kompilator dulu, lalu editor, lalu hubungkan keduanya — Tabel 1.7.

Tabel 1.7 Meja kerja di tiga sistem operasi, dari komputer kosong sampai program yang berjalan. Distribusi Linux lain memakai `sudo dnf install gcc-c++ gdb` (Fedora) atau `sudo pacman -S gcc gdb` (Arch) pada langkah 1.

Langkah	Windows 10/11	MacOS	Linux (Ubuntu/Debian)
1. Pasang kompilator	Pasang MSYS2 dari msys2.org, buka terminal **MSYS2 UCRT64**, jalankan `pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain`	Buka **Terminal**, jalankan `xcode-select --install`	Buka terminal, jalankan `sudo apt update && sudo apt install build-essential gdb`
2. Agar dapat ditemukan	Tambahkan `C:\msys64\ucrt64\bin` ke variabel `Path` pengguna, lalu buka terminal baru	Tidak perlu apa-apa	Tidak perlu apa-apa
3. Periksa	`g++ --version`	`clang++ --version`	`g++ --version`
4. Pasang VS Code	Penginstal dari code.visualstudio.com; centang “Add to PATH”	Unduh, seret *Visual Studio Code* ke *Applications*	`sudo snap install code --classic`, atau berkas `.deb` dari situsnya
5. Tambahkan dukungan C++	Tampilan Extensions (`Ctrl+Shift+X`): pasang **C/C++ Extension Pack** (Microsoft)	Sama, dengan `Cmd+Shift+X`	Sama, dengan `Ctrl+Shift+X`
6. Kompilasi berkas	`g++ -std=c++17 -O2 -Wall -o hello hello.cpp`	`clang++ -std=c++17 -O2 -Wall -o hello hello.cpp`	`g++ -std=c++17 -O2 -Wall -o hello hello.cpp`
7. Jalankan	`.\hello.exe`	`./hello`	`./hello`

Simpan Tabel 1.7 sebagai rujukan satu halaman. Buku: Subbab 1.16 dan Tabel 1.13.

Memasang Kompilator, Langkah demi Langkah

Windows. Pasang MSYS2 dari msys2.org ke `C:\msys64`. Buka **MSYS2 UCRT64** dan jalankan `pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain` (Enter, lalu `Y`). Tambahkan `C:\msys64\ucrt64\bin` ke **Path** pengguna. Di terminal *baru*, `g++ --version` harus mencetak nomor versi.

macOS. Di Terminal jalankan `xcode-select --install` lalu klik **Install**; periksa dengan `clang++ --version`. (`g++` juga berjalan di Mac; diam-diam ia menjalankan Clang.)

Linux. `sudo apt update && sudo apt install build-essential gdb`, lalu `g++ --version`. Fedora: `dnf install gcc-c++ gdb`; Arch: `pacman -S gcc gdb`.

VS Code, di setiap sistem. Pasang dari code.visualstudio.com (di Windows centang **Add to PATH**); di Extensions (`Ctrl+Shift+X`, di Mac `Cmd+Shift+X`) pasang **C/C++ Extension Pack** dari Microsoft; buka folder `DAA` dengan satu subfolder untuk setiap bab.

Jika Windows berkata `g++` "tidak dikenali", entri Path belum tersimpan atau terminalnya masih yang lama: buka terminal baru. Buku: Subbab 1.16.1–1.16.4.

Program Pertama Anda

Kode 1.13 Program pertama: bukti bahwa kompilator, editor, dan terminal sudah bekerja bersama.

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string name;
    cout << "What is your name? ";
    if (!(cin >> name)) name = "friend";      // tidak ada masukan? sapa seorang teman
    cout << "Hello, " << name << "! Your C++17 compiler works." << endl;
    cout << "__cplusplus = " << __cplusplus << endl;
    return 0;
}
```

```
g++ -std=c++17 -O2 -Wall -o hello hello.cpp
```

```
What is your name? Irfan
Hello, Irfan! Your C++17 compiler works.
__cplusplus = 201703
```

`-std=c++17` memilih versi bahasanya, `-O2` meminta optimasi, `-Wall` menyalakan peringatan, `-o hello` menamai programnya; jika kompilator diam, berarti berhasil. `201703` membuktikan bahwa Kode 1.13 dikompilasi sebagai C++17. Buku: Subbab 1.16.5 dan Kode 1.16.

Memberi Masukan, dan Menyaksikan Program Berpikir

Juri memberi masukan lewat *standard input*. Simpan contoh ujinya di `input.txt` lalu **alihkan**: `./columns < input.txt` (macOS, Linux, Command Prompt) atau `Get-Content input.txt | .\columns.exe` (PowerShell).

Debugging. Klik di kiri nomor baris untuk memasang ***breakpoint***; tekan `F5` dan pilih **C++ (GDB/LLDB)**. Panel **Variables** memperlihatkan setiap nilai; `F10` maju satu baris, `F11` masuk ke fungsi, `F5` melanjutkan.

> **KETIKA ADA YANG SALAH.** "`iostream`: No such file" — kompilatornya dipasang dari terminal MSYS2 yang keliru; ulangi di **UCRT64**. Jendela berkedip lalu tertutup — jalankan dari terminal. Garis merah bertebaran padahal programnya terkompilasi — jalankan **C/C++: Select IntelliSense Configuration**.

Menyaksikan sebuah invarian gelung tetap berlaku di panel Variables, putaran demi putaran, adalah salah satu cara terbaik untuk memahaminya. Buku: Subbab 1.16.6–1.16.7.

Profitina dalam Praktik

Profitina (`app.profitina.com`) adalah platform visibilitas AI (GEO). Mendeteksi halaman yang nyaris kembar dengan membandingkan semua pasangan berbiaya $\Theta(P^2)$ — 2.5×10^9 pada $P = 50{,}000$ halaman; sidik jari menjadikannya $O(P)$. Aplikasi memberi saran; pengguna yang memutuskan.

Profitina Laundry, studi kasus pengajaran: mobil boks yang singgah di titik-titik antar di Jalan Kertajaya adalah *Journey*; label pada rak pengering adalah *Columns*. Kode 1.14 menyelesaikan keduanya.

Kode 1.14 Perencana Profitina Laundry: perjalanan pulang-pergi terpendek bagi mobil antar dan banyaknya label rak yang diperlukan.

```
#include <cstdio>
#include <algorithm>
using namespace std;

int main() {
    int n;
    scanf("%d", &n);                        // banyaknya titik antar di Jalan Kertajaya

    int x, lo = 1000000000, hi = -1000000000;
    for (int i = 0; i < n; i++) {           // penanda meter tiap titik antar
        scanf("%d", &x);
        lo = min(lo, x);
        hi = max(hi, x);
    }

    int h, labels = n;                      // rak terbawah saja sudah perlu n label
    for (int i = 0; i < n; i++) {           // berapa keranjang dikirim balik tiap titik
        scanf("%d", &h);
        labels = max(labels, h);            // tumpukan setinggi itu perlu label sebanyak itu
    }

    printf("van route            : %d meters\n", 2 * (hi - lo));
    printf("distinct rack labels : %d\n", labels);
    return 0;
}
```

```
van route            : 1810 meters
distinct rack labels : 9
```

Masukan `5` / `120 340 75 980 410` / `3 7 2 9 4`*: rentang* $980-75=905$*, pulang-pergi* 1810*; label* $\max(5,9)=9$*. Buku: Subbab 1.17 dan Kode 1.17.*

Kesalahan yang Sering Terjadi

Menulis kode sebelum memahami, dan **menguji alih-alih membuktikan**: tutup editornya, kerjakan Fase 1 dan 2, lalu buktikan setiap gelung, pilihan rakus, dan rekursi.

Mengabaikan batasan dan **mengoptimalkan hal yang keliru**: masukkan masukan terbesar ke Tabel 1.2 lebih dulu; mengganti `cin` dengan `scanf` tidak menyelamatkan $O(n^2)$ pada $n = 10^6$.

Melupakan kasus tepi yang sah: $x = 0$ pada *Unique Numbers*, $n = 1$ atau koordinat bersama pada *Journey*, $K = 1$ pada *Min dan Max*.

Luapan bilangan bulat (sebuah `int` berhenti di dekat 2.1×10^9) dan **wadah yang keliru** (larik padahal hanya seribu keadaan yang muncul).

“Terurut” bukan berarti “benar”, **istilah yang didefinisikan tetapi salah dibaca** seperti *substring*, dan **juri yang diperlakukan sebagai kompilator**.

Buku: Subbab 1.18.

Ringkasan — Lima Hal untuk Diingat

1. **Algoritma** adalah gagasannya, **program** adalah gagasan itu dalam sebuah bahasa, **struktur data** membuat langkah-langkahnya murah. Algoritmalah bagian yang bertahan.
2. **Benar** berarti betul untuk *setiap* masukan yang diizinkan, dan hanya pembuktian yang dapat menegakkannya: langsung, kontradiksi, invarian gelung, atau induksi.
3. Sekitar 10^8 operasi per detik: **batasan menentukan rancangan** sebelum satu baris kode pun ditulis.
4. Lima paradigma — inkremental, *greedy*, *divide and conquer*, *dynamic programming*, matematis — dan lima fase, dengan kode di urutan terakhir.
5. Langkah inti Fase 2: buktikan sebuah **batas bawah**, lalu **bangun** sesuatu yang mencapainya.

Columns $\max(n, \max a_i)$*; Journey* $2(x_{\max} - x_{\min})$*; Unique Numbers* $n \leftarrow n/\gcd(n, x)$*; Min dan Max angka yang dipilih secara rakus; Interesting Sequence* $\text{popcount}(m-1) \bmod 3$*; Recursive Function 1 sebuah* `map` *memo. Buku: Subbab 1.19.*

Pertemuan Berikutnya

Bab 2 — Memulai: model RAM, *insertion sort* yang ditelusuri dan dihitung baris demi baris, serta pencarian linear dan biner.

Invarian gelung kembali secara utuh — inisialisasi, pemeliharaan, terminasi — dan “berapa lama waktunya?” menjadi sebuah rumus, bukan tebakan.

Sebelum kita bertemu: pasang meja kerjanya dan jalankan Kode 1.13; kirimkan *Columns* dan *Journey* ke E-Olymp setelah mengerjakan kelima fasenya di atas kertas.

Soal Latihan 1.1–1.16 melatih setiap gagasan hari ini, lengkap dengan penyelesaiannya di Subbab 1.21.

Buku: Subbab 1.20–1.21, lalu Bab 2.

Perancangan dan Analisis Algoritma (EF234405)

Kuliah 2 — Memulai: Model RAM, Insertion Sort, dan Pencarian

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*, Bab 2

Dosen: Irfan Subakti — `yifana@gmail.com`

Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya

Hari ini "berapa lama waktunya?" berhenti menjadi tebakan. Buku: Bab 2, serta Cormen et al., Introduction to Algorithms, edisi ke-4, Bab 2.

Agenda

1. Lima kata yang dipakai dengan cermat: problem, instans, algoritma, program, struktur data — dan sekelumit sejarah gagasannya.
2. **Model RAM**: apa yang dihitung satu langkah, dan apa yang tidak.
3. **Insertion Sort** secara tuntas: ditelusuri, dikodekan, dibuktikan benar dengan invarian gelung, dan dihitung baris demi baris.
4. Kasus terbaik, terburuk, dan rata-rata, diuji pada mesin sungguhan; pertumbuhan fungsi.
5. ***Linear search*** dan ***binary search***, dengan invariannya, $\Theta(n)$ dan $\Theta(\lg n)$-nya, serta dua jebakan selisih satu.

Buku: Bab 2, Subbab 2.1–2.19.

Di Mana Kita Berhenti

Bab 1 adalah bab *berbuat*: enam soal juri, kerangka lima fase, C++ yang *Accepted*.

Yang belum kita lakukan adalah **membenarkannya**. "Ini $O(n)$, jadi lolos" memakai notasi yang belum kita definisikan.

Hari ini: sepakati kata-katanya, sepakati sebuah **mesin**, lalu ambil satu algoritma dan lakukan *segalanya* padanya — uraikan, telusuri, buktikan, hitung, turunkan. Lalu hal yang sama, lebih cepat, untuk pencarian.

> **WAWASAN KUNCI.** Bab 1 memberi *kebiasaan* memecahkan persoalan; bab ini memberi *kosakata dan aritmetika* untuk menjelaskan mengapa sebuah penyelesaian baik. Kebiasaan tanpa aritmetika hanyalah tebak-tebakan.

Bab 3 melunasinya dengan O, Θ, dan Ω yang formal. Buku: Subbab 2.1.

Tentang Apa Mata Ajar Ini

Lintasan terpendek dari Keputih ke Rungkut, jadwal gerbang pesawat, pencarian web, aliran udara di bilah turbin, kompresi video: persoalan nyata yang dimodelkan secara komputasional.

Tiga kegiatan: *merancang* algoritma (tindakan kreatif), *menganalisisnya* (tindakan matematis), *memilih struktur data* (agar algoritmanya bekerja murah).

Bukan pokok bahasannya: bahasa pemrograman (C++ hanyalah notasi kita), arsitektur komputer (diabstraksikan oleh model RAM), rekayasa perangkat lunak, teori kompleksitas secara mendalam.

Pusat perhatian kita adalah **algoritma**: gagasan abstrak tentang cara memecahkan persoalan, lepas dari bahasa dan mesinnya.

Buku: Subbab 2.2.

Kata itu, dan Algoritma yang Pertama

Kata ini berasal dari **al-Khwarizmi** (± 780–850), yang bukunya tentang hitung desimal dilatinkan menjadi *Algorismus*. Sumbangannya adalah sebuah *prosedur* yang begitu jelas sehingga dapat diikuti tanpa mengetahui mengapa ia berhasil.

Algoritma tak remeh tertua yang masih dipakai: **Euklides**, ± 300 SM — ganti (a, b) dengan $(b, a \bmod b)$ sampai $b = 0$.

> **DENGAN TANGAN —** $\gcd(48, 18)$. $48 \bmod 18 = 12 \Rightarrow (18, 12)$; $18 \bmod 12 = 6 \Rightarrow (12, 6)$; $12 \bmod 6 = 0 \Rightarrow (6, 0)$; laporkan 6. Tanpa komputer, tanpa bahasa — dan ia berjalan tanpa perubahan di setiap mesin yang pernah dibuat.

Buku: Subbab 2.3.1–2.3.2.

Al-Khwarizmi

Gambar 2.1 Muhammad ibn Musa al-Khwarizmi (sekitar 780–850 M), yang nama Latinnya, *Algorismus*, menjadi kata *algoritma*.

Gambar 2.1: kita meminjam sebuah nama yang umurnya lebih dari seribu tahun. Buku: Subbab 2.3.1 dan Gambar 2.1.

Mesin, dan Apa Itu Komputasi

Babbage merancang Analytical Engine (*mill*, *store*, kartu berlubang). **Ada Lovelace** menulis algoritma pertama yang dimaksudkan untuk mesin — bilangan Bernoulli — dan meramalkan mesin yang mengolah lambang apa pun.

Turing (1936) mendefinisikan keterhitungan dengan pita, kepala baca-tulis, dan tabel aturan, lalu membuktikan bahwa sebagian persoalan yang dirumuskan dengan baik sama sekali *tidak* memiliki algoritma.

Kalkulus lambda Church mendefinisikan kelas fungsi terhitung yang persis sama — tesis Church–Turing.

Von Neumann (1945): program tinggal di memori yang sama dengan datanya. Model RAM adalah idealisasi rancangan ini.

Algoritma lebih awet daripada perangkat keras: Euklides telah melampaui Kekaisaran Romawi dan disket. Buku: Subbab 2.3.3–2.3.4 dan Gambar 2.2–2.6.

Problem dan Instans

DEFINISI. Sebuah **problem algoritmik** menetapkan hubungan masukan–keluaran yang diinginkan: *apa* yang harus dihitung, bukan *bagaimana*. Sebuah **instans** adalah satu masukan sah tertentu.

PROBLEM PENGURUTAN. *Masukan:* barisan $\langle a_1, a_2, \ldots, a_n \rangle$. *Keluaran:* permutasi $\langle a'_1, a'_2, \ldots, a'_n \rangle$ dengan $a'_1 \le a'_2 \le \cdots \le a'_n$.

n adalah **ukuran masukan**; $\langle \cdots \rangle$ adalah barisan (urutan penting, pengulangan boleh); a'_1 adalah elemen *keluaran* yang pertama.

$\langle 5, 2, 4, 6, 1, 3 \rangle$ adalah sebuah instans. **Kebenaran menyangkut problemnya; waktu jalan biasanya menyangkut instansnya**: $\langle 1, \ldots, 6 \rangle$ dan $\langle 6, \ldots, 1 \rangle$ sama ukurannya, tetapi biayanya bagi Insertion Sort sangat berbeda.

Buku: Subbab 2.4.1–2.4.2.

Algoritma: Lima Sifat

> **DEFINISI.** Sebuah **algoritma** adalah prosedur komputasi yang terdefinisi dengan baik, yang menerima satu atau beberapa nilai sebagai **masukan** dan menghasilkan satu atau beberapa nilai sebagai **keluaran**.

Kriteria Knuth: **masukan** (nol atau lebih nilai), **keluaran** (sekurang-kurangnya satu), **kepastian** (“bandingkan $A[i]$ dengan k *e* y”, bukan “aduk sampai tampak pas”).

Keberhinggaan pada *setiap* masukan sah — `while (true) i = i + 1;` pasti tetapi tak berguna. **Keterlaksanaan** — setiap langkah dapat dikerjakan dengan pensil dan kertas.

> **CONTOH.** “Bangkitkan setiap urutan; berhenti pada yang pertama kali terurut.” Kelima sifat terpenuhi, jadi ia *adalah* algoritma — dan untuk $n = 20$ ia mungkin mencoba $20! \approx 2.4 \times 10^{18}$ urutan. Menjadi algoritma adalah tolok ukur yang rendah.

Buku: Subbab 2.4.3.

Program, Struktur Data, Benar, Efisien

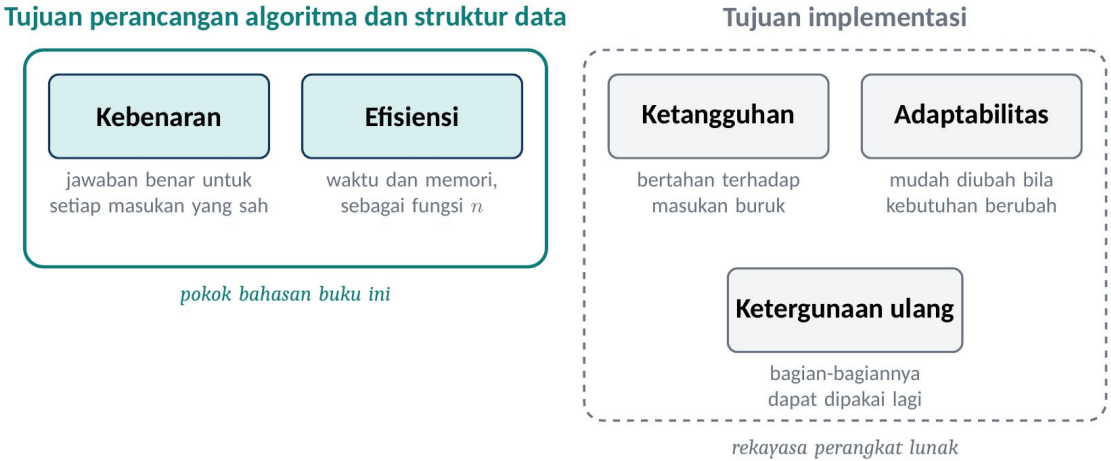
Sebuah **program** mengimplementasikan algoritma dalam sebuah bahasa untuk sebuah mesin. Kita menganalisis algoritma, sehingga analisisnya berlaku untuk setiap implementasi sekaligus.

Sebuah **struktur data** menata data agar sebagian operasi menjadi murah: larik membaca posisi 7 seketika tetapi mahal untuk menyisipkan di sana; senarai berantai sebaliknya.

> **DEFINISI.** Algoritma disebut **benar** jika pada **setiap** instans yang sah ia berhenti dengan keluaran yang memenuhi spesifikasinya. “Benar pada semua instans kecuali satu” berarti salah.
>
> **DEFINISI. Efisiensi** adalah sumber daya yang dipakai *sebagai fungsi dari n*: terutama waktu (langkah primitif), lalu ruang (sel memori).

Seluruh buku dalam satu kalimat: di antara tak berhingga banyaknya algoritma yang benar, temukan yang efisien. Buku: Subbab 2.4.4–2.4.6 dan Gambar 2.7.

Lima Tujuan Perancangan



Gambar 2.2 Kelima tujuan itu, dalam dua kelompok. Kebenaran dan efisiensi adalah tujuan perancangan dan menjadi pokok bahasan buku ini; kekukuhan, keteradaptasian, dan keterpakaian ulang adalah tujuan implementasi yang termasuk wilayah rekayasa perangkat lunak.

Gambar 2.2: kebenaran dan efisiensi adalah tujuan perancangan dan pokok buku ini; ketangguhan, kemampuan beradaptasi, dan kebergunaan ulang adalah tujuan implementasi. Kebenaran dibuktikan (invarian gelung, induksi), bukan diuji. Buku: Subbab 2.5 dan Gambar 2.8.

Model RAM

Aturan. Sel memori bernomor, **sel mana pun** dibaca atau ditulis dalam satu langkah (*akses acak*); satu instruksi dalam satu waktu; setiap instruksi **sederhana** berbiaya satu langkah; nilai berukuran wajar (satu *word* memuat sekurang-kurangnya $\lg n$ bit).

Satu langkah: aritmetika (`+ - * / %`, lantai, perbandingan, operasi bit); perpindahan data, termasuk **pengindeksan** `A[i]`; kendali (uji, lompatan, panggilan, kembali).

```
x = y + z;        // satu penjumlahan, ditambah perpindahannya -- dihitung konstan
if (a > b) ...    // satu pembandingan
t = A[i];         // satu akses larik
i = i - 1;        // satu pengurangan dan satu penyimpanan
```

$\lg n$ *berarti* $\log_2 n$*; Subbab 2.12 mendefinisikannya dengan cermat. Buku: Subbab 2.6.1–2.6.2.*

Yang *Bukan* Satu Langkah — dan Mengapa Model Ini

Gelung bukan satu langkah: badannya berjalan n kali. **Panggilan bukan satu langkah:** `sort(v)` hanya satu baris, tetapi mungkin $n \lg n$ kerja. **Data tak terbatas bukan satu langkah:** menyalin string sepanjang m berbiaya m; 2^k berbiaya sekitar $\lg k$ perkalian.

RENUNGKAN. Apakah kerja baris ini bergantung pada n? Jika tidak, ia satu langkah. Jika ya, bukalah dan hitunglah isinya.

Model ini sengaja "berbohong" (tembolok, *pipeline*, banyak inti). Kita memakainya karena ia **tidak bergantung pada mesin**, **dapat meramalkan** — urutan peringkat algoritmanya benar begitu n besar — dan **cukup sederhana untuk dihitung**.

Sebuah model adalah keputusan tentang apa yang diabaikan. Buku: Subbab 2.6.3–2.6.4.

Cara Menganalisis Waktu Jalan

> **DEFINISI. Waktu jalan** $T(n)$ adalah banyaknya langkah primitif RAM yang dieksekusi pada masukan berukuran n — cacah operasi, tidak pernah detik.

Metode: (1) beri setiap baris biaya konstan $c_1, c_2, \ldots$; (2) hitung berapa kali setiap baris berjalan; (3) jumlahkan biaya × cacah. Seluruh keterampilannya ada di langkah 2.

Definisikan n dulu: banyaknya elemen untuk pengurutan dan pencarian; dimensi untuk matriks (n^2 bilangan); $|V|$ dan $|E|$ untuk graf; banyaknya **bit** untuk teori bilangan.

0,8 detik bercerita tentang laptop Anda; “sekitar $n^2/2$ operasi” akan tetap benar lima puluh tahun lagi.

Buku: Subbab 2.7–2.7.2.

Contoh Lengkap yang Pertama

Pseudokode 2.1 Mencari yang terbesar di antara n bilangan dengan satu kali sapuan.

```
FIND-MAX(A, n)
1  max = A[1]
2  for i = 2 to n
3      if A[i] > max
4          max = A[i]
5  return max
```

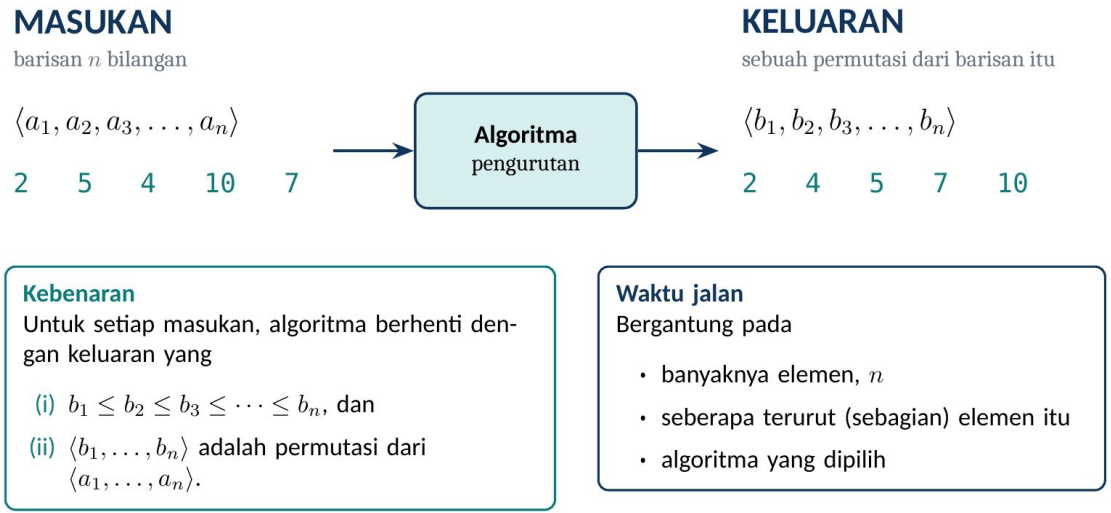
Cacah untuk Pseudokode 2.1: baris 1 sekali; uji baris 2 sebanyak n kali ($n-1$ berhasil dan satu gagal di akhir); baris 3 sebanyak $n-1$ kali; baris 4 sebanyak k kali dengan $0 \le k \le n-1$; baris 5 sekali. Menurut Persamaan (2.1):

$$T(n) = c_1 + c_2 n + c_3 (n-1) + c_4 k + c_5 . \tag{2.1}$$

Kedua nilai ekstrem k menghasilkan bentuk $an+b$: $T(n)$ linear apa pun masukannya. k mengubah konstantanya, bukan bentuknya.

Pola itu — data menggeser konstanta, bukan pangkat n — akan terus berulang sepanjang semester. Buku: Subbab 2.7.3, Pseudokode 2.1, dan Persamaan (2.1).

Problem Pengurutan, secara Formal



Gambar 2.3 Problem pengurutan. Keluaran $\langle b_1, \ldots, b_n \rangle$ harus sekaligus *terurut* ($b_1 \leq b_2 \leq \cdots \leq b_n$) dan merupakan *permutasi* dari masukannya — dua syarat, dan algoritma yang hanya memenuhi salah satunya adalah algoritma yang salah. Waktu jalannya bergantung pada ada berapa elemen, seberapa hampir-terurut elemen itu sudah, dan algoritma mana yang kita pilih.

Kotak kiri bawah Gambar 2.3 menyimpan jebakannya: terurut dan permutasi. Mengeluarkan $\langle 1,2,3,4,5 \rangle$ untuk setiap masukan memang terurut, tetapi salah. Buku: Subbab 2.8 dan Gambar 2.9.

Cara Pemain Kartu

Gambar 2.4 Cara pemain kartu. Tangan kiri memegang deretan terurut 2–4–5–10; tangan kanan baru saja mengambil kartu 7 dan akan menggesernya ke kiri melewati kartu 10 untuk mencapai tempatnya.

Pada Gambar 2.4 deretan yang terurut bertambah satu kartu setiap langkah (metode inkremental), dan tempatnya dicari dengan memindai dari kanan ke kiri, berhenti pada kartu pertama yang tidak lebih besar — penghentian dini itulah sebabnya kasus terbaiknya begitu bagus. Buku: Subbab 2.8.1 dan Gambar 2.10.

Insertion Sort dalam Pseudokode

Pseudokode 2.2 Insertion Sort dalam pseudokode yang dipakai di sepanjang buku ini, dengan indeks berbasis-1.

```
INSERTION-SORT(A)
1  for j = 2 to A.length          // mulai dari elemen ke-2
2      key = A[j]                 // ambil kartunya
3      i = j - 1                  // mulai membandingkan dari sebelah kiri
4      while i > 0 and A[i] > key // geser elemen yang lebih besar ke kanan
5          A[i + 1] = A[i]
6          i = i - 1
7      A[i + 1] = key             // sisipkan kartunya
```

Baris 2 menyimpan `key` sebelum baris 5 menimpa $A[j]$ — hilangkan baris itu dan larik akan penuh dengan duplikat.

Baris 4: `i > 0` menghentikan kita di ujung kiri; `A[i] > key` memberi jalan keluar dini. Urutannya penting: `&&` berhenti di syarat pertama yang salah, jadi $A[0]$ tidak pernah dibaca.

Baris 7 menulis di $A[i{+}1]$, bukan $A[i]$: gelungnya keluar satu langkah *melewati* elemen terakhir yang digeser, jadi $i{+}1$ adalah lubangnya.

Pseudokode 2.2 menghitung dari 1, seperti CLRS, karena matematikanya menjadi lebih bersih. Buku: Subbab 2.8.2 dan Pseudokode 2.2.

Menelusuri dengan Tangan

Tabel 2.1 Insertion Sort pada $A = \langle 5, 2, 4, 6, 1, 3 \rangle$, satu iterasi gelung luar untuk tiap baris. Batas antara awalan yang terurut dan akhiran yang belum tersentuh bergeser satu tempat ke kanan pada setiap baris.

Langkah	Key	Tindakan	Hasil
$j = 2$	2	$2 < 5$, jadi geser 5 ke kanan. Sisipkan 2 di posisi 1.	$[2, 5, 4, 6, 1, 3]$
$j = 3$	4	$4 < 5$, geser 5 ke kanan. $4 > 2$, berhenti. Sisipkan 4 di posisi 2.	$[2, 4, 5, 6, 1, 3]$
$j = 4$	6	$6 > 5$, tak ada pemindahan. Sudah pada tempatnya.	$[2, 4, 5, 6, 1, 3]$
$j = 5$	1	$1 < 6$, geser 6. $1 < 5$, geser 5. $1 < 4$, geser 4. $1 < 2$, geser 2. Sisipkan 1 di posisi 1.	$[1, 2, 4, 5, 6, 3]$
$j = 6$	3	$3 < 6$, geser 6. $3 < 5$, geser 5. $3 < 4$, geser 4. $3 > 2$, berhenti. Sisipkan 3 di posisi 3.	$[1, 2, 3, 4, 5, 6]$

Periksa setiap baris Tabel 2.1 dengan larik di depan Anda: awalan yang terurut bertambah satu di setiap baris.

Buku: Subbab 2.8.3 dan Tabel 2.1.

Baris j = 5, Langkah demi Langkah

Mulai dari $[2,4,5,6,1,3]$; `key` $=1$, $i=4$.

Uji: $4>0$ dan $6>1$ — geser: $[2,4,5,6,6,3]$, $i=3$. Uji: $5>1$ — geser: $[2,4,5,5,6,3]$, $i=2$.

Uji: $4>1$ — geser: $[2,4,4,5,6,3]$, $i=1$. Uji: $2>1$ — geser: $[2,2,4,5,6,3]$, $i=0$.

Uji: $0>0$ bernilai **salah** — kita berhenti pada syarat pertama dan tidak pernah membaca $A[0]$. Baris 7: $A[1] \leftarrow 1$, menghasilkan $[1,2,4,5,6,3]$.

Uji baris 4 dijalankan **lima** kali: empat berhasil, satu gagal. Ingat “lima” — nanti ia disebut t_5.

Enam belas baris untuk memindahkan satu kartu. Buku: Subbab 2.8.3.

Insertion Sort dalam C++

Kode 2.1 Insertion Sort dalam C++, dengan pengindeksan berbasis-1 agar kodenya persis sejajar dengan pseudokodenya.

```cpp
// Insertion Sort, ditulis agar sejajar baris demi baris dengan pseudokodenya.
// Lariknya berbasis-1: A[0] sengaja dibiarkan tak terpakai supaya indeks
// C++-nya persis sama dengan indeks yang dipakai pseudokode di buku ini.
#include <iostream>
#include <vector>
using namespace std;

void insertionSort(vector<int>& A, int n) {
    for (int j = 2; j <= n; ++j) {
        int key = A[j];          // kartu yang kita ambil
        int i = j - 1;           // mulai membandingkan dengan kartu di kirinya
        while (i > 0 && A[i] > key) {
            A[i + 1] = A[i];     // geser kartu yang lebih besar satu tempat ke kanan
            i = i - 1;
        }
        A[i + 1] = key;          // jatuhkan kartunya ke lubang yang tadi dibuka
    }
}

void print(const vector<int>& A, int n) {
    for (int i = 1; i <= n; ++i) cout << A[i] << (i < n ? ' ' : '\n');
}

int main() {
    vector<int> A = {0, 5, 2, 4, 6, 1, 3};   // A[0] tak terpakai
    int n = 6;
    cout << "before: "; print(A, n);
    insertionSort(A, n);
```

```
    cout << "after:  "; print(A, n);
    return 0;
}
```

```
before: 5 2 4 6 1 3
after:  1 2 3 4 5 6
```

Kode 2.1 membiarkan `A[0]` tak terpakai agar C++-nya sejajar baris demi baris dengan pseudokodenya; jawabannya sama dengan Tabel 2.1. Buku: Subbab 2.8.4 dan Kode 2.1.

Mesin Menelusuri Dirinya Sendiri

Kode 2.2 Insertion Sort yang dilengkapi pencatat agar mencetak telusurnya sendiri, satu baris untuk tiap iterasi gelung luar.

```cpp
// Insertion Sort dengan telusur tercetak: satu baris untuk tiap iterasi gelung
// luar, menirukan dengan mesin persis telusur tangan yang dikerjakan di teks.
#include <iostream>
#include <vector>
using namespace std;

void show(const vector<int>& A, int n) {
    cout << '[';
    for (int i = 1; i <= n; ++i) cout << A[i] << (i < n ? "," : "");
    cout << ']';
}

int main() {
    vector<int> A = {0, 5, 2, 4, 6, 1, 3};
    int n = 6;
    cout << "start        "; show(A, n); cout << '\n';
    for (int j = 2; j <= n; ++j) {
        int key = A[j];
        int i = j - 1;
        int shifts = 0;
        while (i > 0 && A[i] > key) { A[i + 1] = A[i]; --i; ++shifts; }
        A[i + 1] = key;
        cout << "j=" << j << " key=" << key
             << " shifts=" << shifts
             << " insert at " << (i + 1) << "  ";
        show(A, n);
        cout << '\n';
    }
```

```
    return 0;
}
```

```
start        [5,2,4,6,1,3]
j=2 key=2 shifts=1 insert at 1  [2,5,4,6,1,3]
j=3 key=4 shifts=1 insert at 2  [2,4,5,6,1,3]
j=4 key=6 shifts=0 insert at 4  [2,4,5,6,1,3]
j=5 key=1 shifts=4 insert at 1  [1,2,4,5,6,3]
j=6 key=3 shifts=3 insert at 3  [1,2,3,4,5,6]
```

Setiap baris keluaran Kode 2.2 cocok dengan Tabel 2.1, termasuk cacah geserannya $1,1,0,4,3$*. Telusuri dengan tangan dan instrumentasikan; bila keduanya tidak cocok, salah satunya berisi kutu. Buku: Subbab 2.8.4 dan Kode 2.2.*

Apa Itu Invarian Gelung

DEFINISI. Sebuah **invarian gelung** (*loop invariant*) adalah pernyataan tentang keadaan komputasi yang benar tepat sebelum setiap iterasi gelung dan benar kembali tepat setelah gelung selesai.

Inisialisasi — benar sebelum iterasi pertama. **Pemeliharaan** — jika benar sebelum suatu iterasi, masih benar sebelum iterasi berikutnya.

Terminasi — saat keluar, invarian ditambah alasan gelung berhenti memberi sifat yang kita inginkan.

Dua yang pertama persis induksi (kasus dasar, langkah induktif); terminasi adalah saat kita menguangkan invariannya.

MENGAPA LANGKAH KETIGA TIDAK BOLEH DILEWATI. "j adalah bilangan bulat" memang invarian dan tak berguna. Invarian yang baik adalah yang kebenarannya *saat keluar* justru hal yang hendak Anda buktikan.

Buku: Subbab 2.9.1.

Invariannya, dan Inisialisasi

> **INVARIAN GELUNG (Insertion Sort).** Pada awal tiap iterasi gelung `for` di baris 1, sublarik $A[1\ldots j-1]$ terdiri atas elemen-elemen yang semula berada di $A[1\ldots j-1]$, tetapi dalam keadaan terurut.

Dua klausa, keduanya diperlukan: **terurut**, dan **elemen yang sama**. Tanpa yang kedua, algoritma yang menulis angka nol pun akan memenuhinya.

Periksa pada telusurnya: di $j=5$, $A[1..4]=[2,4,5,6]$ terurut, dan $\lfloor 2,4,5,6 \rfloor$ adalah penghuni asli $5,2,4,6$.

Inisialisasi. $j=2$: $A[1\ldots 1]$ hanya satu elemen — terurut secara hampa (tidak ada pasangan bersebelahan yang bisa gagal), dan belum ada yang berpindah.

"Setiap unicorn di ruangan ini berwarna ungu" benar karena alasan yang sama: himpunan kosong tidak punya contoh tandingan. Buku: Subbab 2.9.2–2.9.3.

Pemeliharaan: Invarian di Dalam Invarian

> **INVARIAN GELUNG DALAM.** Pada setiap uji baris 4: (a) $A[1\ldots i]$ tidak berubah; (b) $A[i{+}2\ldots j]$ memuat $A[i{+}1\ldots j-1]$ yang lama, bergeser satu tempat ke kanan; (c) $A[i{+}1]$ adalah *lubang*; (d) setiap nilai di $A[i{+}2\ldots j]$ lebih besar daripada `key`.

Berlaku saat masuk ($i=j-1$; rentang di (b) dan (d) kosong; $A[j]$ adalah lubang karena `key` sudah menyimpannya) dan **setiap geseran melestarikannya** (baris 5 mengisi lubang, baris 6 memindahkannya ke kiri).

Keluar dengan $i=0$**:** semua isi $A[2..j]$ melebihi `key`; `key` masuk ke $A[1]$. **Keluar dengan** $A[i]\le key$**:** blok kiri $\le key$, blok kanan $> key$, `key` masuk ke lubang $A[i{+}1]$.

Dalam kedua kasus $A[1\ldots j]$ terurut dan tidak ada nilai yang tercipta atau hilang: invarian luar berlaku untuk $j{+}1$.

Ini menjawab soal selisih satu: kunci masuk di $i{+}1$ karena baris 6 selalu menggeser i satu tempat melewati elemen terakhir yang dipindahkan. Buku: Subbab 2.9.4.

Terminasi

Gelungnya pasti berhenti: j mulai dari 2, bertambah satu, dan diuji terhadap batas tetap n. Ia keluar pada $j = n+1$.

Sulihkan $j = n+1$: $A[1..n]$ memuat elemen-elemen asli $A[1..n]$ dalam keadaan terurut.

Itulah seluruh lariknya, dan persis kedua klausa spesifikasi pengurutan. Insertion Sort benar. ▰

> **RENUNGKAN.** Invarian saja tidak membuktikan apa pun tentang jawabannya; invarian *yang dievaluasi pada syarat keluar*-lah yang membuktikannya.

Tulislah invariannya sebelum gelungnya, dan gelungnya biasanya benar pada percobaan pertama. Buku: Subbab 2.9.5.

Tabel Biaya dan Cacah

DEFINISI. t_j adalah banyaknya **uji** `while` di baris 4 dijalankan untuk nilai j itu. Badannya (baris 5–6) berjalan $t_j - 1$ kali.

Tabel 2.2 Biaya tiap baris Insertion Sort dan berapa kali baris itu berjalan. Lambang t_j adalah berapa kali pengujian baris 4 dinilai untuk nilai j tersebut.

Baris	Biaya	Cacah eksekusi
1: `for j = 2 to A.length`	c_1	n
2: `key = A[j]`	c_2	$n-1$
3: `i = j - 1`	c_3	$n-1$
4: `while i > 0 and A[i] > key`	c_4	$\sum_{j=2}^{n} t_j$
5: `A[i+1] = A[i]`	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6: `i = i - 1`	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7: `A[i+1] = key`	c_7	$n-1$

Baris 1 berjalan n kali: $n-1$ kali masuk ditambah satu uji yang gagal. Pada telusurnya, $t_5 = 5$ dan baris 5–6 berjalan 4 kali — `shifts=4`.

Cacah yang tidak dapat Anda pertanggungjawabkan adalah cacah yang Anda tebak; Tabel 2.2 mempertanggungjawabkan setiap barisnya. $\sum_{j=2}^{n} t_j$ *berarti* $t_2 + t_3 + \cdots + t_n$*. Buku: Subbab 2.10.1 dan Tabel 2.2.*

Rumus Umumnya

Menjumlahkan biaya × cacah di sepanjang Tabel 2.2 memberi waktu jalan untuk masukan **apa pun**, Persamaan (2.2):

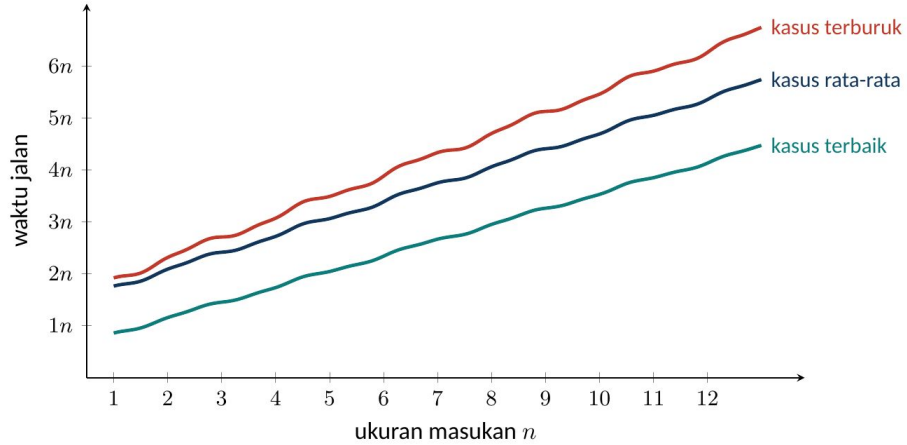
$$T(n)=c_1 n+c_2(n-1)+c_3(n-1)+c_4\sum_{j=2}^{n} t_j+c_5\sum_{j=2}^{n}(t_j-1)+c_6\sum_{j=2}^{n}(t_j-1)+c_7(n-1). \tag{2.2}$$

Nilai c_k bergantung pada kompilator dan prosesor — kita tidak pernah memerlukannya. n adalah ukurannya. Nilai t_j bergantung pada **larik mana yang diberikan**.

> **WAWASAN KUNCI.** $T(n)$ tidak ditentukan oleh n saja: rumusnya berlubang seukuran $\sum t_j$ yang hanya dapat diisi oleh masukannya. Karena itu ada tiga kasus.

Buku: Subbab 2.10.2 dan Persamaan (2.2).

Terbaik, Terburuk, Rata-rata



Gambar 2.5 Untuk suatu ukuran masukan terdapat banyak kemungkinan masukan dan karena itu banyak pula kemungkinan waktu jalan. Kasus terburuk adalah selubung atasnya, kasus terbaik selubung bawahnya, dan kasus rata-rata terletak di antara keduanya.

Pada Gambar 2.5, untuk setiap ukuran ada sebuah pita waktu jalan: kasus terburuk di atas, kasus terbaik di bawah, rata-rata di antaranya. Buku: Subbab 2.11 dan Gambar 2.11.

Kasus Terbaik: Sudah Terurut

Masukan $\langle 1,2,3,4,5,6\rangle$: $A[j-1]\le A[j]$, jadi uji baris 4 langsung gagal. $t_j=1$ untuk setiap j; baris 5–6 tidak pernah berjalan.

$\sum_{j=2}^{n} t_j=n-1$ dan $\sum_{j=2}^{n}(t_j-1)=0$. Sulihkan lalu kumpulkan:

$T(n)=(c_1+c_2+c_3+c_4+c_7)n-(c_2+c_3+c_4+c_7)$, yaitu Persamaan (2.3):

$$T(n)=an+b. \tag{2.3}$$

WAWASAN KUNCI. Linear — $\Theta(n)$ dalam notasi Bab 3. Tidak mungkin lebih cepat: ia harus melihat setiap elemen untuk tahu bahwa lariknya terurut.

Buku: Subbab 2.11.1 dan Persamaan (2.3).

Kasus Terburuk: Dua Jumlahan

Masukan $\langle 6,5,4,3,2,1\rangle$: setiap elemen di kiri j lebih besar, jadi uji berhasil $j-1$ kali lalu gagal sekali: $t_j = j$ (periksa $t_2 = 2$).

Siasat Gauss: tulis $S = 2+3+\cdots+n$ maju dan mundur; masing-masing dari $n-1$ kolom berjumlah $n+2$, jadi $2S = (n-1)(n+2)$ dan Persamaan (2.4):

$$\sum_{j=2}^{n} t_j = \frac{n(n+1)}{2} - 1. \tag{2.4}$$

Dengan $t_j - 1 = j - 1$, banyaknya geseran adalah $1+2+\cdots+(n-1)$, Persamaan (2.5):

$$\sum_{j=2}^{n} (t_j - 1) = \sum_{j=2}^{n} (j-1) = 1+2+3+\cdots+(n-1) = \frac{n(n-1)}{2}, \tag{2.5}$$

Periksa dengan $n = 6$*:* $2+3+4+5+6 = 20 = 21-1$ *dan* $1+2+3+4+5 = 15 = 6\cdot 5/2$*. Buku: Subbab 2.11.2 dan Persamaan (2.4)–(2.5).*

Kasus Terburuk: Kuadratik

Sulihkan Persamaan (2.4) dan Persamaan (2.5) ke dalam Persamaan (2.2), jabarkan $\frac{n(n\pm1)}{2}=\frac{n^2}{2}\pm\frac{n}{2}$, lalu kumpulkan menurut pangkat n:

$$T(n)=an^2+bn+c \tag{2.6}$$

Di sini $a=(c_4+c_5+c_6)/2>0$: pada Persamaan (2.6) kasus terburuknya **kuadratik**, $\Theta(n^2)$ — diturunkan, bukan dinyatakan begitu saja.

> **WAWASAN KUNCI.** Sejuta elemen yang terurut terbalik memerlukan sekitar $10^{12}/2$ geseran: lebih dari delapan menit pada 10^9 operasi per detik, untuk daftar yang diselesaikan `std::sort` dalam kurang dari sedetik.

Mesin mengubah a; ia tidak pernah mengubah n^2-nya. Itulah sebabnya analisis ini dapat dikerjakan di atas kertas, sebelum perangkat kerasnya ada. Buku: Subbab 2.11.2 dan Persamaan (2.6).

Kasus Rata-rata

Andaikan semua $n!$ urutan sama mungkinnya. `key` mendarat di salah satu dari j tempat dengan peluang sama, jadi t_j sama mungkinnya bernilai $1, 2, \ldots, j$: rata-ratanya $\frac{j+1}{2}$ — kira-kira separuh panjang awalan.

Maka $\sum_{j=2}^{n} \frac{j+1}{2} = \frac{n^2+3n-4}{4} \approx \frac{n^2}{4}$ (periksa $n=6$: 12.5), dan geserannya juga sekitar $\frac{n^2}{4}$ — separuh kasus terburuk.

Koefisien n^2 menjadi separuhnya, memberi Persamaan (2.7):

$$T(n) \approx \frac{a}{2} n^2 + b'n + c' \tag{2.7}$$

> **WAWASAN KUNCI.** Orde pertumbuhannya sama dengan kasus terburuk. Jika delapan menit terlalu lambat, empat menit pun terlalu lambat. Hanya laju pertumbuhan baru — $n \lg n$, Merge Sort di Bab 3 — yang menyelesaikannya.

Buku: Subbab 2.11.3 dan Persamaan (2.7).

Menghitung pada Mesin Sungguhan

Kode 2.3 Menghitung $\sum_{j=2}^{n} t_j$ untuk masukan menaik, menurun, dan acak, bersanding dengan nilai kasus terburuk yang diramalkan.

```cpp
// Menghitung kerja yang benar-benar dilakukan Insertion Sort, pada tiga kasus
// klasiknya. Yang kita hitung persis besaran yang disebut t_j di tabel biaya:
// berapa kali PENGUJIAN gelung while dinilai untuk tiap j.
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>
#include <iomanip>
using namespace std;

long long sortAndCountTests(vector<int> A) {      // A diterima per nilai: sebuah salinan
    int n = (int)A.size() - 1;                    // A[0] tak terpakai
    long long tests = 0;
    for (int j = 2; j <= n; ++j) {
        int key = A[j], i = j - 1;
        ++tests;                                  // pengujian yang bisa langsung gagal
        while (i > 0 && A[i] > key) {
            A[i + 1] = A[i]; --i;
            ++tests;                              // tiap pengujian ulang ikut dihitung
        }
        A[i + 1] = key;
    }
    return tests;
}

int main() {
    cout << left << setw(6) << "n" << setw(12) << "best"
         << setw(12) << "worst" << setw(12) << "random"
```

```
         << setw(14) << "n(n+1)/2-1" << '\n';
    mt19937 rng(12345);                              // benih tetap: dapat diulang
    for (int n : {8, 16, 32, 64, 128}) {
        vector<int> asc(n + 1), desc(n + 1), rnd(n + 1);
        for (int i = 1; i <= n; ++i) { asc[i] = i; desc[i] = n - i + 1; }
        rnd = asc;
        shuffle(rnd.begin() + 1, rnd.end(), rng);
        cout << left << setw(6) << n
             << setw(12) << sortAndCountTests(asc)
             << setw(12) << sortAndCountTests(desc)
             << setw(12) << sortAndCountTests(rnd)
             << setw(14) << ((long long)n * (n + 1) / 2 - 1) << '\n';
    }
    return 0;
}
```

```
n     best        worst       random      n(n+1)/2-1
8     7           35          25          35
16    15          135         59          135
32    31          527         340         527
64    63          2079        1045        2079
128   127         8255        4453        8255
```

Kode 2.3: kasus terbaik tepat $n-1$; kasus terburuk identik dengan $n(n+1)/2-1$; yang acak berada di sekitar separuh kasus terburuk ($4453/8255=0.539$). Buku: Subbab 2.11.4 dan Kode 2.3.

Stopwatch dan Uji Pelipatduaan

Kode 2.4 Mengukur waktu kasus terburuk Insertion Sort pada lima ukuran masukan, masing-masing dua kali lipat ukuran sebelumnya. Kolom `ratio` adalah pengganda biaya yang teramati ketika n dilipatduakan.

```cpp
// Apakah kuadratiknya benar-benar muncul pada mesin sungguhan? Lipatduakan n,
// lalu perhatikan. Insertion Sort dijalankan pada masukan terurut terbalik,
// yaitu kasus terburuknya, sehingga cacah pengujian gelung while persis
// n(n+1)/2 - 1 setiap kalinya.
#include <iostream>
#include <vector>
#include <chrono>
#include <iomanip>
using namespace std;
using namespace std::chrono;

double timeWorstCase(int n) {
    vector<int> A(n + 1);
    for (int i = 1; i <= n; ++i) A[i] = n - i + 1;   // n, n-1, ..., 1
    auto t0 = steady_clock::now();
    for (int j = 2; j <= n; ++j) {
        int key = A[j], i = j - 1;
        while (i > 0 && A[i] > key) { A[i + 1] = A[i]; --i; }
        A[i + 1] = key;
    }
    auto t1 = steady_clock::now();
    return duration<double, milli>(t1 - t0).count();
}

int main() {
    cout << setw(8) << "n" << setw(14) << "time (ms)"
         << setw(12) << "ratio" << setw(18) << "n(n+1)/2-1" << '\n';
    double prev = 0;
```

```
    for (int n : {5000, 10000, 20000, 40000, 80000}) {
        double ms = timeWorstCase(n);
        cout << setw(8) << n << setw(14) << fixed << setprecision(2) << ms;
        if (prev > 0) cout << setw(12) << setprecision(2) << (ms / prev);
        else          cout << setw(12) << "-";
        cout << setw(18) << ((long long)n * (n + 1) / 2 - 1) << '\n';
        prev = ms;
    }
    return 0;
}
```

```
       n     time (ms)       ratio        n(n+1)/2-1
    5000          4.10           -          12502499
   10000         16.09        3.93          50004999
   20000         64.06        3.98         200009999
   40000        256.48        4.00         800019999
   80000       1034.66        4.03        3200039999
```

*Milidetik dari Kode 2.4 di komputer Anda akan berbeda; kolom **rasio**-lah yang berlaku di mana-mana, berkisar di angka empat, karena Persamaan (2.8) berikut. Buku: Subbab 2.11.4 dan Kode 2.4.*

Mengapa Empat — dan Mengapa Kasus Terburuk

Untuk $T(n) = a n^2 + b n + c$ suku n^2 mendominasi, sehingga Persamaan (2.8):

$$\frac{T(2n)}{T(n)} \approx \frac{a(2n)^2}{a n^2} = \frac{4 a n^2}{a n^2} = 4. \tag{2.8}$$

Lipatduakan n: algoritma linear menjadi dua kali, yang kuadratik menjadi empat kali, yang logaritmik bertambah sebuah konstanta. Jika Anda yakin $O(n \lg n)$ tetapi melihat 4, ada yang keliru.

Mengapa kasus terburuk: ia sebuah **jaminan**; sebagian sistem (alat pacu jantung, rem) menuntutnya; ia sering terjadi (kunci yang tidak ada, data terurut, lawan yang sengaja); rata-ratanya sering berorde sama; dan rata-rata memerlukan distribusi yang jarang Anda ketahui.

> **INTUISI.** “Tiba paling lambat 5 hari” adalah janji yang bisa Anda rencanakan; “biasanya 3 hari” hanyalah harapan. Analisis kasus terburuk menjual jaminan.

Buku: Subbab 2.11.4–2.11.5 dan Persamaan (2.8).

Logaritma dan Tabel Pertumbuhan

$\lg n = \log_2 n$ menjawab: *berapa kali n dapat dibagi dua sebelum mencapai 1?* $\lg 8 = 3$, $\lg 1024 = 10$, $\lg 10^6 \approx 19.93$.

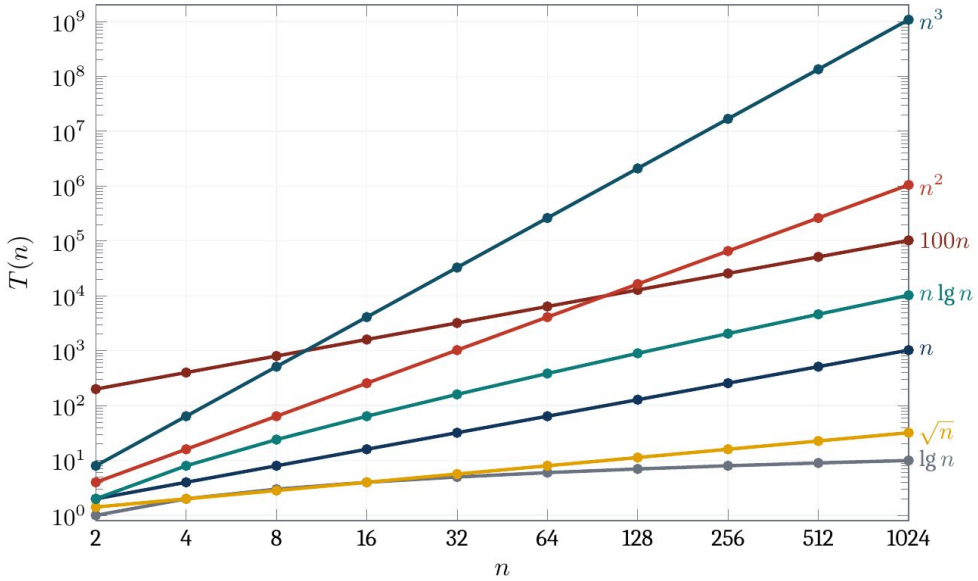
Tabel 2.3 Bagaimana lima waktu jalan yang umum itu tumbuh. Nilai terakhirnya begitu besar sehingga sering ditulis begitu saja sebagai ∞; nilai terhitungnya berdigit lebih dari tiga ratus ribu.

n	$\lg n$	n	$n \lg n$	n^2	2^n
10	3.3	10	33	100	1,024
100	6.6	100	664	10,000	1.27×10^{30}
1,000	10.0	1,000	9,966	1,000,000	1.07×10^{301}
10^6	19.9	10^6	1.99×10^7	10^{12}	9.90×10^{301029}

Pada $n = 10^6$ dan 10^9 operasi per detik: $\lg n$ — 20 operasi; n — satu milidetik; $n \lg n$ — dua puluh milidetik; n^2 — enam belas menit empat puluh detik; 2^n — tidak ada satuan waktunya.

Setiap isi Tabel 2.3 dihitung oleh Kode 2.5 di bab ini, yang mencetak 2^{10^6} tanpa pernah menyimpannya. Buku: Subbab 2.12.1–2.12.2 dan Tabel 2.3.

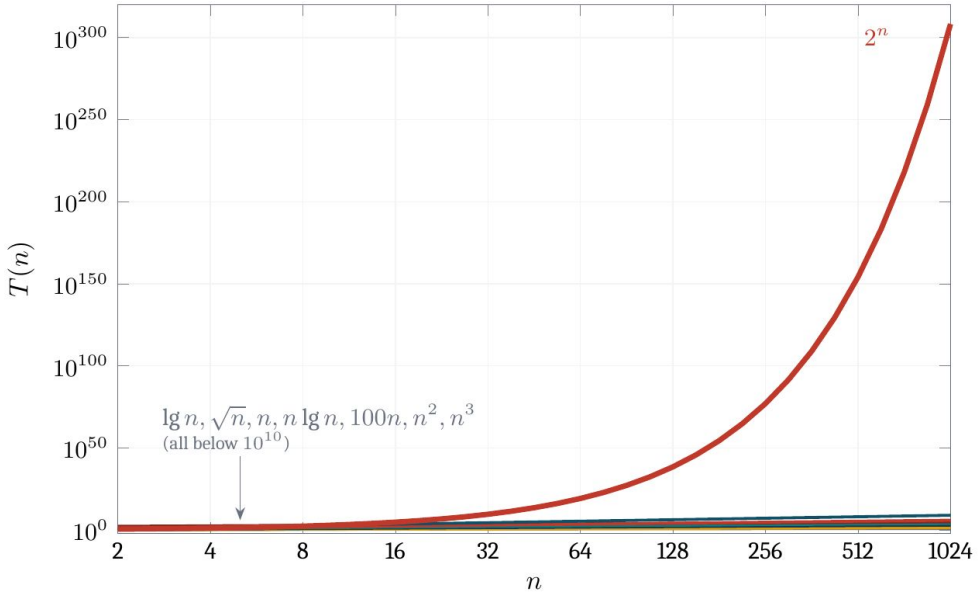
Pertumbuhan pada Sumbu Log–Log



Gambar 2.6 Pertumbuhan $\lg n$, $\sqrt{n}$, n, $n \lg n$, $100\,n$, n^2, dan n^3 dengan kedua sumbu logaritmik. Pada peta semacam itu fungsi n^k menjadi garis lurus yang kemiringannya k, sehingga makin curam berarti makin buruk.

Pada Gambar 2.6 setiap n^k menjadi garis lurus dengan kemiringan k. $100\,n$ bermula lima puluh kali di atas n^2, keduanya berpotongan di dekat $n=100$, lalu n^2 menjauh selamanya: konstanta besar menyembunyikan laju pertumbuhan yang buruk hanya untuk n kecil. Buku: Subbab 2.12.3 dan Gambar 2.12.

Tambahkan yang Eksponensial



Gambar 2.7 Laju pertumbuhan yang sama dengan 2^n ditambahkan. Sumbu tegaknya kini merentang sampai sekitar 10^{308}; setiap kurva polinomial terhimpit ke dasar bagan, tak terbedakan dari sumbunya.

Pada Gambar 2.7 setiap polinomial terimpit ke sumbu: di hadapan 2^n, semua polinomial cepat. Itulah sebabnya garis pemisah utama ditarik antara polinomial dan eksponensial. Buku: Subbab 2.12.3 dan Gambar 2.13.

Hierarki dan Anggarannya

$$O(1) < O(\lg n) < O(\sqrt{n}) < O(n) < O(n \lg n) < O(n^2) < O(n^3) < O(2^n) < O(n!) \tag{2.9}$$

Bacalah Persamaan (2.9) dengan contohnya: `A[i]`; *binary search*; pembagian percobaan; satu kali lintas; Merge Sort; kasus terburuk Insertion Sort; perkalian matriks naif; semua himpunan bagian (paling banyak $n \approx 30$); semua urutan ($n \approx 12$).

Tabel 2.4 Ukuran masukan terbesar yang dapat ditangani tiap kelas kompleksitas dalam batas satu sampai dua detik. Bacalah mundur untuk memilih algoritma: pernyataan soal memberi Anda n, dan tabel ini memberi Anda kompleksitas yang harus Anda capai.

n Maks	Kompleksitas yang boleh	Operasi	Contoh
$n \le 10$	$O(n!), O(2^n)$	$\sim 10^6$	Gaya kasar / runut balik
$n \le 1{,}000$	$O(n^2)$	$\sim 10^6$	Insertion Sort, Bubble Sort
$n \le 100{,}000$	$O(n \lg n)$	$\sim 1.7 \times 10^6$	Merge Sort, Quicksort
$n \le 10^6$	$O(n)$ atau $O(n \lg n)$	$\sim 10^7$	Sapuan linear, counting sort
$n \le 10^{18}$	$O(\lg n)$ atau $O(1)$	~ 60	Binary search, rumus bentuk tertutup

Pakailah Tabel 2.4 secara terbalik: rumusan soal memberi n, tabel memberi kompleksitas yang harus dicapai. $O(n^2)$ pada $n = 10^6$ lima ratus kali melampaui anggaran — tidak ada optimasi kecil yang dapat menutupnya. Buku: Subbab 2.12.4–2.12.5, Persamaan (2.9), dan Tabel 2.4.

Problem Pencarian

Gambar 2.8 Problem pencarian. Pada instans $A = \langle 2, 5, 4, 10, 7 \rangle$ dengan $q = 5$ jawabannya adalah indeks 2; dengan $q = 9$ jawabannya NIL, sebab 9 tidak muncul.

Gambar 2.8: kembalikan sebuah ***indeks*** j *dengan* $A[j] = q$*, atau NIL. NIL harus berupa nilai yang tidak mungkin menjadi indeks sungguhan — 0 untuk indeks berbasis 1,* -1 *untuk berbasis 0. Buku: Subbab 2.13.1 dan Gambar 2.14.*

Linear Search dan Invariannya

Pseudokode 2.3 *Linear search*: periksa elemen dari kiri ke kanan sampai q ditemukan atau lariknya habis.

```
LINEAR-SEARCH(A, q)
1  j = 1
2  while j <= A.length and A[j] != q
3      j = j + 1
4  if j <= A.length
5      return j
6  else
7      return NIL
```

INVARIAN GELUNG (linear search). Pada awal tiap iterasi, tidak ada satu pun di antara $A[1], \ldots, A[j-1]$ yang sama dengan q.

Inisialisasi: rentang kosong. **Pemeliharaan:** badan gelung hanya berjalan bila $A[j] \neq q$. **Terminasi:** $A[j] = q$ (kembalikan j) atau $j = n+1$ dan tidak ada elemen yang sama dengan q (kembalikan NIL). ∎

Pada Pseudokode 2.3, `j <= A.length` harus diuji lebih dulu agar kita tidak membaca melewati ujung larik. Buku: Subbab 2.13.2–2.13.3 dan Pseudokode 2.3.

Linear Search: Analisis dan Kode

Terburuk (tidak ada atau paling akhir): $\Theta(n)$. **Terbaik** ($A[1]=q$): $\Theta(1)$. **Rata-rata**, q ada dan seragam letaknya, Persamaan (2.10):

$$\frac{1}{n}(1+2+\cdots+n)=\frac{1}{n}\cdot\frac{n(n+1)}{2}=\frac{n+1}{2}, \tag{2.10}$$

Kode 2.5 Linear search dalam C++, pada data yang sama dengan Gambar 2.8.

```cpp
// Linear search pada larik yang TIDAK TERURUT, persis seperti pseudokodenya.
// Indeksnya berbasis-1; A[0] tak terpakai. NIL diwakili oleh 0, yang tidak
// pernah menjadi indeks yang sah, jadi ia tak dapat dikelirukan dengan
// jawaban yang sungguhan.
#include <iostream>
#include <vector>
using namespace std;

const int NIL = 0;

int linearSearch(const vector<int>& A, int n, int q) {
    int j = 1;
    while (j <= n && A[j] != q) ++j;    // berhenti pada kecocokan pertama, atau lewati ujungnya
    if (j <= n) return j;               // j indeks sungguhan: q kita temukan di sana
    return NIL;                         // kita melewati ujungnya: q tidak ada
}

int main() {
    vector<int> A = {0, 2, 5, 4, 10, 7};   // data pada gambar problem pencarian
    int n = 5;
    for (int q : {5, 9}) {
        int j = linearSearch(A, n, q);
        cout << "q=" << q << " -> ";
        if (j == NIL) cout << "NIL\n"; else cout << j << '\n';
    }
    return 0;
}
```

```
q=5 -> 2
q=9 -> NIL
```

$\Theta(n)$ optimal pada data tak terurut: posisi yang belum diperiksa bisa saja memuat q. Untuk lebih cepat, ubahlah datanya. Kode 2.5 mereproduksi Gambar 2.8. Buku: Subbab 2.13.4–2.13.6, Persamaan (2.10), dan Kode 2.6.

Problem Pencarian pada Data Terurut

MASUKAN

- barisan bilangan A yang *terurut* (tidak turun) (datanya)
- satu bilangan q (kuerinya)

KELUARAN

indeks j dengan $A[j] = q$,
atau NIL bila q tidak muncul

$\langle a_1, a_2, a_3, \ldots, a_n \rangle;\ q$		j
2 4 5 7 10; 5	→	3
2 4 5 7 10; 9	→	NIL

Gambar 2.9 Problem pencarian terurut. Satu kata yang ditambahkan — *terurut* — pada spesifikasi masukan itulah yang memungkinkan percepatan sebesar eksponensial.

Satu kata tambahan, terurut, pada Gambar 2.9: kini satu perbandingan $A[5] > q$ menyingkirkan posisi 5, 6, 7, … sekaligus. Binary search pada data tak terurut diam-diam memberi jawaban salah. Buku: Subbab 2.14.1 dan Gambar 2.15.

Binary Search

Bandingkan q dengan elemen **tengah**: sama — ketemu; $A[mid]>q$ — buang paruh kanan; $A[mid]<q$ — buang paruh kiri. Seperti membuka buku telepon di sekitar tengahnya.

Pseudokode 2.4 *Binary search* pada larik terurut: bandingkan dengan elemen tengah dan buang paruh yang tidak mungkin memuat q.

```
BINARY-SEARCH(A, q)
1  left = 1
2  right = A.length
3  while left <= right
4      mid = (left + right) / 2        // pembagian bilangan bulat
5      if A[mid] == q then return mid
6      else if A[mid] > q then right = mid - 1
7      else left = mid + 1
8  return NIL
```

INVARIAN GELUNG (binary search). Pada awal tiap iterasi, jika q muncul dalam A, ia muncul di dalam jendela $[left, right]$.

Pada Pseudokode 2.4, $(left, right)$ adalah jendela tempat q mungkin masih bersembunyi; jendela kosong berarti tidak ada. Buku: Subbab 2.14.2–2.14.3 dan Pseudokode 2.4.

Binary Search Benar dan Berhenti

Inisialisasi: jendelanya seluruh larik. **Pemeliharaan:** jika $A[mid] > q$, setiap $A[k]$ dengan $k \geq mid$ melebihi q, jadi q hanya mungkin di $[left, mid-1]$; simetris untuk $<$.

Terminasi: saat $left > right$ jendelanya kosong, jadi q tidak ada dan NIL benar; jalan keluar lainnya mengembalikan mid dengan $A[mid] = q$.

Kemajuan: $left \leq mid \leq right$, jadi $mid - 1$ dan $mid + 1$ pasti menyusutkan jendelanya. ▰

JEBAKAN. Tulis `right = mid`, maka jendela $[1,2]$ dengan $A[1] > q$ menjadi $[1,1]$, lalu $[1,1]$ selamanya: programnya macet.

Buku: Subbab 2.14.3.

Sebuah Telusur, dan Mengapa Logaritmik

Tabel 2.5 Binary search untuk $q=7$ dalam $A=\langle 2,4,5,7,10\rangle$. Dua pembandingan sudah cukup, sedangkan linear search memerlukan empat.

Langkah	Left	Right	Mid	Tindakan
1	1	5	3	$A[3]=5<7 \Rightarrow$ cari di paruh kanan: $left=4$
2	4	5	4	$A[4]=7=7 \Rightarrow$ ketemu! Kembalikan 4

Setelah k perbandingan jendelanya memuat paling banyak $n/2^k$ elemen; ia habis begitu $n/2^k \le 1$, Persamaan (2.11):

$$\frac{n}{2^k} \le 1 \Leftrightarrow n \le 2^k \Leftrightarrow \lg n \le k. \tag{2.11}$$

Jadi $T(n)=\Theta(\lg n)$: **20** perbandingan untuk 10^6 elemen, **60** untuk 10^{18}. Setiap pelipatduaan n menambah tepat satu perbandingan.

Pada Tabel 2.5, satu perbandingan menyingkirkan tiga dari lima posisi. Buku: Subbab 2.14.4–2.14.5, Tabel 2.5, dan Persamaan (2.11).

Binary Search dalam C++

Kode 2.6 Binary search dalam C++, bentuk iteratif, dilengkapi pencatat agar mencetak jendelanya pada tiap langkah.

```cpp
// Binary search pada larik TERURUT, dengan telusur left, right, mid yang
// dicetak. Indeks berbasis-1; NIL adalah 0. Titik tengahnya dihitung sebagai
// left + (right - left) / 2, tidak pernah (left + right) / 2 -- lihat teksnya.
#include <iostream>
#include <vector>
using namespace std;

const int NIL = 0;

int binarySearch(const vector<int>& A, int n, int q, bool trace) {
    int left = 1, right = n, step = 0;
    while (left <= right) {                 // jendela kosong berarti "tidak ada"
        int mid = left + (right - left) / 2;
        if (trace)
            cout << "  step " << ++step << ": left=" << left
                 << " right=" << right << " mid=" << mid
                 << " A[mid]=" << A[mid] << '\n';
        if (A[mid] == q) return mid;
        else if (A[mid] > q) right = mid - 1;   // q hanya mungkin di sebelah kiri
        else                 left  = mid + 1;   // q hanya mungkin di sebelah kanan
    }
    return NIL;
}

int main() {
    vector<int> A = {0, 2, 4, 5, 7, 10};    // terurut; A[0] tak terpakai
    int n = 5;
    cout << "searching for 7, with trace:\n";
```

```
    int j = binarySearch(A, n, 7, true);
    cout << "  result = " << j << "\n\n";
    for (int q : {5, 9}) {
        int r = binarySearch(A, n, q, false);
        cout << "q=" << q << " -> ";
        if (r == NIL) cout << "NIL\n"; else cout << r << '\n';
    }
    return 0;
}
```

```
searching for 7, with trace:
  step 1: left=1 right=5 mid=3 A[mid]=5
  step 2: left=4 right=5 mid=4 A[mid]=7
  result = 4

q=5 -> 3
q=9 -> NIL
```

Langkah-langkah yang dilacak Kode 2.6 sama dengan Tabel 2.5; dua baris terakhir mereproduksi Gambar 2.9. Perhatikan titik tengahnya: `left + (right - left) / 2`. Buku: Subbab 2.14.6 dan Kode 2.7.

Jebakan Limpah pada Titik Tengah

Kode 2.7 Mengapa titik tengah harus dihitung sebagai `left + (right - left) / 2`.

```cpp
// Jebakan limpah pada titik tengah, diperagakan alih-alih sekadar ditegaskan.
// Kedua ungkapannya sama secara matematis. Dalam aritmetika 32 bit hanya satu
// di antaranya yang aman, sebab left + right dapat melampaui INT_MAX.
#include <iostream>
#include <climits>
using namespace std;

int main() {
    int left  = 1'500'000'000;
    int right = 2'000'000'000;
    cout << "INT_MAX           = " << INT_MAX << '\n';
    cout << "left              = " << left  << '\n';
    cout << "right             = " << right << '\n';
    cout << "true midpoint     = " << 1'750'000'000 << '\n';

    // Limpah bertanda adalah perilaku tak terdefinisi, jadi penjumlahannya kita
    // kerjakan dalam tipe yang lebih lebar lalu kita potong menjadi int -- itu
    // menirukan apa yang dilakukan mesin tanpa memberi kompilator kebebasan
    // menganggap bahwa hal itu tidak pernah terjadi.
    long long sum = (long long)left + right;
    int wrapped = (int)(unsigned int)sum;          // pola bit yang akan dipegang int
    cout << "(left+right)      = " << sum << " which does not fit in int\n";
    cout << "(left+right)/2    = " << wrapped / 2 << "   <-- wrong, negative\n";
    cout << "left+(right-left)/2 = " << left + (right - left) / 2 << "   <-- right\n";
    return 0;
}
```

```
INT_MAX           = 2147483647
left              = 1500000000
right             = 2000000000
true midpoint     = 1750000000
(left+right)      = 3500000000 which does not fit in int
(left+right)/2    = -397483648   <-- wrong, negative
left+(right-left)/2 = 1750000000   <-- right
```

Kutu ini bersembunyi dua dasawarsa di Programming Pearls dan bertahun-tahun di pustaka Java. Kode 2.7 menunjukkan sebabnya: $right - left$ *selalu kecil,* $left + right$ *belum tentu. Buku: Subbab 2.14.6 dan Kode 2.8.*

Binary Search sebagai Rekursi

Kode 2.8 Binary search sebagai prosedur divide-and-conquer rekursif, yang mencetak jendelanya pada tiap aras rekursi.

```cpp
// Binary search yang ditulis secara rekursif: bentuk divide-and-conquer-nya
// dibuat tampak. Rekursinya berjalan pada jendela [left..right]; jendela
// kosong adalah kasus dasarnya.
#include <iostream>
#include <vector>
using namespace std;

const int NIL = 0;

int bsRec(const vector<int>& A, int left, int right, int q, int depth) {
    cout << string(2 * depth, ' ') << "window [" << left << ".." << right << "]";
    if (left > right) { cout << " is empty -> NIL\n"; return NIL; }   // kasus dasar
    int mid = left + (right - left) / 2;
    cout << " mid=" << mid << " A[mid]=" << A[mid] << '\n';
    if (A[mid] == q) return mid;
    if (A[mid] > q) return bsRec(A, left, mid - 1, q, depth + 1);
    return                bsRec(A, mid + 1, right, q, depth + 1);
}

int main() {
    vector<int> A = {0, 1, 3, 5, 7, 9, 11, 13, 15};   // 8 elemen terurut
    int n = 8;
    for (int q : {13, 6}) {
        cout << "looking for " << q << ":\n";
        int r = bsRec(A, 1, n, q, 1);
        cout << "result: ";
        if (r == NIL) cout << "NIL\n\n"; else cout << r << "\n\n";
    }
```

```
    return 0;
}
```

```
looking for 13:
  window [1..8] mid=4 A[mid]=7
    window [5..8] mid=6 A[mid]=11
      window [7..8] mid=7 A[mid]=13
result: 7

looking for 6:
  window [1..8] mid=4 A[mid]=7
    window [1..3] mid=2 A[mid]=3
      window [3..3] mid=3 A[mid]=5
        window [4..3] is empty -> NIL
result: NIL
```

Pada Kode 2.8, satu panggilan pada separuh jendela ditambah kerja konstan menghasilkan rekurensi Persamaan (2.12), yang diselesaikan secara mekanis di Bab 3. Buku: Subbab 2.14.7 dan Kode 2.9.

Sebuah Rekurensi, dan Harga Pengurutan

$$T(n) = T(n/2) + \Theta(1), \tag{2.12}$$

Persamaan (2.12): waktu untuk n elemen sama dengan waktu untuk $n/2$ ditambah satu perbandingan. Penyelesaiannya $\Theta(\lg n)$.

Imbal balik untuk k pencarian: *linear* berbiaya $\Theta(k\,n)$; urutkan lalu cari berbiaya $\Theta(n \lg n + k \lg n)$.

Pada $n = 10^6$: satu pencarian — *linear* menang; seratus — *binary* menang lima kali lipat; sepuluh ribu — lima ratus kali lipat. Titik impasnya di sekitar $k \approx \lg n \approx 20$.

> **WAWASAN KUNCI.** Bayar sekali untuk membangun struktur, lalu bertanyalah dengan murah selamanya — tawar-menawar di balik tabel *hash*, pohon pencarian, *heap*, dan jumlah awalan.

Buku: Subbab 2.14.7–2.14.8 dan Persamaan (2.12).

Menengok Kembali Paradigma Perancangan

Inkremental (Insertion Sort): perluas penyelesaian untuk $j-1$ butir menjadi j. Sederhana, bagus untuk data kecil atau hampir terurut, *online*; pengurutan di pustaka memakainya di bawah sekitar enam belas elemen. Kelemahan: sering $\Theta(n^2)$.

Bagi dan kuasai (*binary search*, Merge Sort): bagi, kuasai, gabungkan. Sering $\Theta(n \lg n)$ atau $\Theta(\lg n)$; mudah diparalelkan; berbiaya beban rekursi dan kadang memori tambahan $\Theta(n)$.

Ad hoc: siasat untuk satu persoalan — $\max(n, \max_i a_i)$ pada *Columns*, *popcount* pada *Interesting Sequence*. Paling cepat jika ditemukan; hanya ditemukan dengan mengamati kasus-kasus kecil.

Pelajari paradigmanya agar Anda selalu punya langkah; teruslah mencari siasatnya agar sesekali Anda punya langkah yang lebih baik. Buku: Subbab 2.15.

Bab 1, Ditinjau Ulang

Tabel 2.6 Keenam problem Bab 1, ditelaah ulang dengan perkakas analisis Bab 2. Setiap klaim kompleksitas yang dibuat secara tak resmi di Bab 1 kini menjadi sesuatu yang dapat Anda turunkan.

Problem Bab 1	Paradigma	Konsep Bab 2 yang dipakai	Kompleksitas
Columns	Greedy / langsung	Sapuan inkremental sekali jalan	$O(n)$ — linear, sebaik yang mungkin
Journey	Greedy	Minimum dan maksimum dalam satu sapuan	$O(n)$ — batas bawah untuk membaca masukannya
Unique Numbers	Teori bilangan	GCD Euclid, $O(\lg n)$ per pemanggilan	$O(k \lg n)$ kasus terburuk
Min/Max Numbers	Konstruksi greedy	Cetak digitnya secara inkremental	$O(N)$ per kasus uji
Interesting Sequence	Manipulasi bit	Instruksi CPU bawaan $O(1)$	$O(1)$ — waktu konstan
Recursive Function 1	Memoisasi / DP	Akses map $O(\lg D)$ per pencarian, D kunci berbeda	$O(\lg^2 n \cdot \lg \lg n)$

Columns dan *Journey* tidak dapat mengalahkan $O(n)$: algoritma apa pun harus membaca n bilangan. *Recursive Function 1*: $D = O(\lg^2 n)$ kunci, masing-masing dicari dalam $O(\lg \lg n)$ — melawan $2f(n) - 1 \approx 4.5 \times 10^{14}$ panggilan naif.

Setiap klaim informal Bab 1 pada Tabel 2.6 kini dapat Anda turunkan. Buku: Subbab 2.16 dan Tabel 2.6.

Profitina dalam Praktik

Ukuran masukan dulu: pemeriksaan atas semua pasangan halaman tak terasa pada 40 halaman (1.600 perbandingan) dan memakan menit-menit CPU pada 40.000 halaman (1,6 miliar) — maka setiap sinyal punya anggaran kompleksitas yang dinyatakan.

Data hampir terurut: skor mingguan tiba hampir berurutan, jadi menyisipkan setiap titik berjalan dekat kasus terbaik $\Theta(n)$. **Penjadwal:** urutkan sekali, lakukan *binary search* berkali-kali. **Kasus terburuk adalah janji layanan** — dan situs yang dipindai adalah masukan yang tidak bersahabat.

Profitina Laundry: sepuluh tiket yang tiba hampir berurutan hanya berbiaya **3 geseran** dari kasus terburuk 45; mencari tiket dengan *binary search* berbiaya **4 tilikan**, ada atau tidak ada, karena $\lceil \lg(n+1) \rceil = 4$.

Aplikasi memberi saran; pengguna yang memutuskan. Program buku loket selengkapnya adalah Kode 2.10. Buku: Subbab 2.17.

Kesalahan yang Sering Terjadi

Menyalahkan bahasanya, menganggap gelung satu langkah, menyebut waktu jam dinding sebagai kompleksitas. Hitunglah operasinya; pakai uji pelipatduaan.

Insertion Sort: kehilangan kuncinya; menyisipkan di $A[i]$, bukan $A[i+1]$; menguji $A[i] > key$ sebelum $i > 0$.

***Binary search*:** `mid = (left + right) / 2`; `right = mid`; menjalankannya pada data tak terurut; nilai NIL yang bisa menjadi indeks sungguhan.

Hitung-hitungan: mengurutkan demi satu kali pencarian; menyetel konstanta padahal laju pertumbuhannya yang salah (Tabel 2.3).

Membuktikan dengan menguji. Sepuluh uji yang lolos hanya membuktikan sepuluh masukan. Nyatakan invariannya dan periksa ketiga bagiannya.

Buku: Subbab 2.18.

Ringkasan — Lima Hal untuk Diingat

1. Problem, instans, algoritma, program, dan struktur data adalah lima hal yang berbeda; kebenaran menyangkut semua instans, waktu jalan menyangkut satu instans.

2. Dalam model RAM, $T(n)$ = jumlah biaya × cacah atas semua baris. Gelung tidak pernah satu langkah.

3. Insertion Sort benar menurut invariannya; ia $\Theta(n)$ pada kasus terbaik ($t_j = 1$), $\Theta(n^2)$ pada kasus terburuk ($t_j = j$) dan rata-rata.

4. Laju pertumbuhan mendominasi: $O(1) < O(\lg n) < O(n) < O(n \lg n) < O(n^2) < O(2^n)$; kita menganalisis kasus terburuk karena ia sebuah jaminan.

5. *Linear search* optimal pada data tak terurut dengan $\Theta(n)$; *binary search* $\Theta(\lg n)$ pada data terurut — kecepatan yang dibeli dengan struktur.

"Algoritma harus dilihat agar dipercaya" (Knuth). Telusurilah setiap algoritma dengan tangan. Buku: Subbab 2.19 dan Gambar 2.16.

Pertemuan Berikutnya

Bab 3 — Notasi Asimtotik, Rekurensi, dan Teorema Master.

O, Ω, Θ didefinisikan dengan konstanta yang eksplisit lalu dibuktikan; jumlahan; rekurensi diselesaikan dengan substitusi, pohon rekursi, dan Teorema Master.

Merge Sort diturunkan dari nol: $T(n) = 2T(n/2) + \Theta(n)$ menghasilkan $\Theta(n \lg n)$, akhirnya mengalahkan kuadratik hari ini.

Sebelum kita bertemu: Soal Latihan 2.1, 2.5, 2.10, dan 2.12 — menelusuri, membuktikan, menghitung, dan mematahkan sebuah *binary search*. Penyelesaiannya ada di Subbab 2.21.

Buku: Subbab 2.20–2.21, lalu Bab 3.

Perancangan dan Analisis Algoritma (EF234405)

Kuliah 3 — Notasi Asimtotik, Rekurensi, dan Teorema Master

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*, Bab 3

Dosen: Irfan Subakti — `yifana@gmail.com`

Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya

Hari ini setiap "kira-kira n^2" menjadi pernyataan dengan konstanta yang dapat Anda tuliskan. Buku: Bab 3, serta Cormen et al., Introduction to Algorithms, edisi ke-4, Bab 3–4.

Agenda

1. Kebenaran parsial dan total; mengapa konstanta dibuang.

2. Kelima notasi O, Ω, Θ, o, ω — pembuktian *dan* penyangkalan — serta aljabarnya.

3. Fungsi-fungsi yang sering muncul dan jumlahan dari gelung: aritmetika, geometri, harmonik, teleskopis.

4. Rekurensi, diselesaikan dengan tiga cara: substitusi, pohon rekursi, dan **Teorema Master** beserta celah-celahnya.

5. Merge Sort diturunkan dari nol: algoritma, penelusuran, invarian, rekurensi, program.

Buku: Bab 3, Subbab 3.1–3.18.

Dari Mana Kita Melanjutkan

Model RAM: hitung instruksi, bukan detik. Detik milik sebuah laptop pada suatu hari Selasa; cacah instruksi milik algoritmanya.

Invarian gelung: inisialisasi, pemeliharaan, terminasi — induksi dalam busana lain. Insertion Sort sudah dibuktikan benar dengannya.

Tiga lubang tersisa. *Bahasa*: apa arti "kira-kira n^2"? *Aritmetika*: bentuk tertutup untuk jumlahan yang dihasilkan gelung. *Rekursi*: Merge Sort memanggil dirinya sendiri, jadi ia memerlukan rekurensi.

Bab ini menambal ketiganya. Buku: Subbab 3.1 dan 3.1.2.

Kebenaran Parsial dan Total

> **DEFINISI.** Sebuah algoritma **benar secara parsial** jika, *setiap kali ia berhenti*, keluarannya benar. Ia **benar secara total** jika ia selalu berhenti pada setiap masukan yang sah *dan* keluarannya kemudian benar.

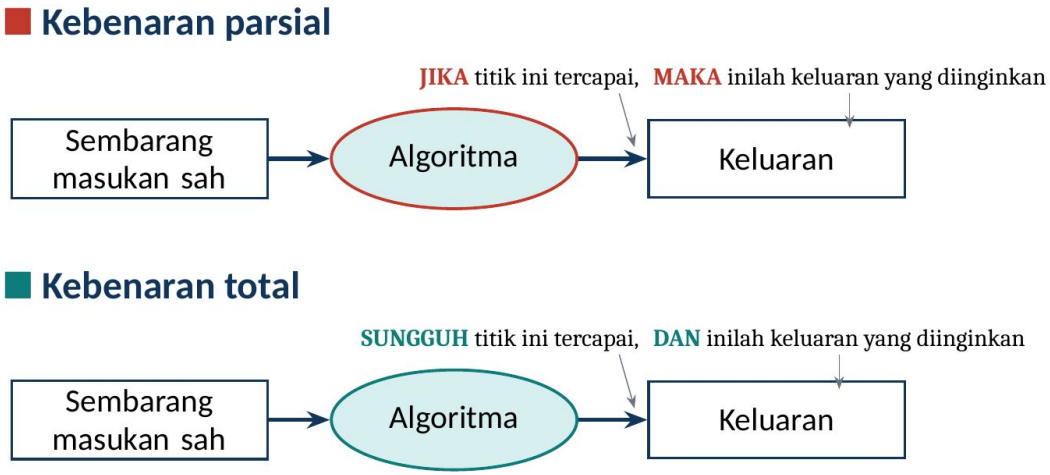
Kebenaran total = kebenaran parsial + terminasi. "*Jika* koki selesai memasak, hidangannya lezat" berbeda dengan "koki selesai memasak, *dan* hidangannya lezat".

Tidak ada program yang dapat memeriksa kebenaran untuk kita: menentukan apakah sebuah algoritma berhenti adalah **Halting Problem**, yang dibuktikan Turing tak terputuskan pada 1936.

Terminasi biasanya jelas (pencacah naik ke sebuah batas; rekursi menyusutkan masukannya); pekerjaan beratnya adalah kebenaran parsial.

Buku: Subbab 3.1.1.

Dua Jaminan, dalam Gambar



Gambar 3.1 Kebenaran parsial dibandingkan kebenaran total. Pada diagram atas, jaminannya bersyarat — **JIKA** kendali sampai ke kotak keluaran, **MAKA** keluarannya adalah yang dikehendaki. Pada diagram bawah, jaminannya tanpa syarat — kendali **SUNGGUH** sampai ke kotak keluaran, **DAN** keluarannya adalah yang dikehendaki.

*Gambar 3.1: diagram atas menjanjikan **JIKA** kendali mencapai keluaran, **MAKA** keluarannya benar; diagram bawah menjanjikan kendali **MEMANG** mencapainya, **DAN** keluarannya benar. Buku: Subbab 3.1.1 dan Gambar 3.1.*

Mengapa Perlu Bahasa untuk Pertumbuhan

A mengurutkan $1,000$ rekaman dalam 0.8 detik, B dalam 0.1 detik. Pada $1,000,000$: A $(n \lg n)$ memerlukan 0.8×2000 detik ≈ 27 menit; B (n^2) memerlukan 0.1×10^6 detik ≈ 28 jam.

> **WAWASAN KUNCI.** Buang **faktor konstanta** $(3n^2 \to n^2)$ dan **suku berorde lebih rendah** $(5n^2+3n+7 \to n^2)$: yang ditentukan mesin dibuang; yang ditentukan algoritma disimpan.

Tabel 3.1 Kelas-kelas pertumbuhan, apa yang terjadi pada masing-masing ketika ukuran masukan digandakan, dan di mana tiap kelas muncul dalam buku ini.

Pertumbuhan	Nama	Menggandakan n ...	Contoh dari buku ini
$\Theta(1)$	konstan	tidak mengubah apa pun	membaca $A[i]$
$\Theta(\lg n)$	logaritmik	menambah satu langkah	pencarian biner
$\Theta(n)$	linear	menggandakan kerjanya	memindai array satu kali
$\Theta(n \lg n)$	linearitmik	sedikit lebih dari menggandakannya	Merge Sort, Heapsort
$\Theta(n^2)$	kuadratik	mengalikan kerja dengan 4	Insertion Sort, kasus terburuk
$\Theta(n^3)$	kubik	mengalikan kerja dengan 8	perkalian matriks naif
$\Theta(2^n)$	eksponensial	*mengkuadratkan* kerjanya	pencarian himpunan bagian secara brutal
$\Theta(n!)$	faktorial	lebih buruk lagi	permutasi secara brutal

Hafalkan kolom pelipatduaan pada Tabel 3.1. Untuk n kecil konstanta tetap berkuasa: Insertion Sort mengalahkan Merge Sort pada sepuluh elemen, karena itu pengurutan di pustaka beralih kepadanya di sekitar enam belas. Kode 3.1 mencetak angka mentahnya. Buku: Subbab 3.2–3.2.1 dan Tabel 3.1.

Big-O: Batas Atas

n adalah ukuran masukan; $f(n)$ fungsi yang dikaji; $g(n)$ fungsi sederhana pembandingnya; c pengali positif yang **tetap**; n_0 ambang yang di bawahnya kita tidak menjanjikan apa pun.

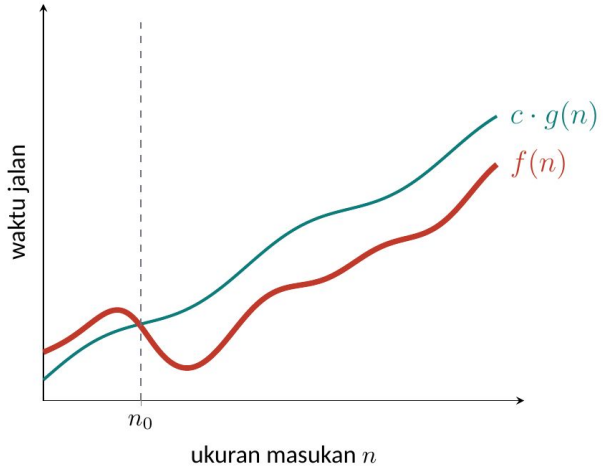
DEFINISI (Big-O). $f(n) = O(g(n))$ jika terdapat konstanta positif c dan n_0 sedemikian sehingga Persamaan (3.1) berlaku:

$$0 \le f(n) \le c \cdot g(n) \text{ untuk semua } n \ge n_0. \tag{3.1}$$

Dengan kata-kata: **pada akhirnya f tetap berada di bawah suatu kelipatan tetap dari g, selamanya.** Anda boleh memilih kelipatannya dan mengabaikan masukan kecil; Anda tidak boleh mengganti pilihan ketika n membesar.

Big-O menyatakan batas atas kasus terburuk. Buku: Subbab 3.3 dan Persamaan (3.1).

Big-O, dalam Gambar



Gambar 3.2 Big-O sebagai batas atas. Di luar ambang n_0, waktu jalan $f(n)$ tidak pernah naik melampaui $c \cdot g(n)$. Apa yang terjadi di sebelah kiri n_0 tidak relevan bagi klaim ini.

Pada Gambar 3.2, di kiri n_0 kurva f boleh berkeliaran di atas $c \cdot g$; mulai n_0 ia terkurung di bawahnya. Buku: Subbab 3.3 dan Gambar 3.2.

Membuktikan Klaim Big-O

KLAIM: $7n-3=O(n)$. Pengurangan membuat nilainya lebih kecil, jadi $7n-3\le 7n$ untuk setiap n. Saksinya: $c=7$, $n_0=1$. ▰ Pasangan apa pun yang berhasil sudah merupakan bukti lengkap; $c=100$, $n_0=50$ pun bisa.

KLAIM: $3n^2+5n+7=O(n^2)$. Untuk $n\ge 1$, $n\le n^2$ dan $1\le n^2$, jadi $3n^2+5n+7\le 15n^2$: $c=15$, $n_0=1$.

Menekan c: $c=4$ memerlukan $5n+7\le n^2$, salah pada $n=6$ ($37>36$), benar mulai $n=7$. Jadi $c=4$, $n_0=7$ juga berhasil — konstanta lebih kecil dibeli dengan ambang lebih besar.

Bukti klaim O selalu berbentuk sama: tunjukkan saksinya, periksa pertidaksamaannya. Buku: Subbab 3.3.1–3.3.2.

Menyangkal Klaim Big-O

Andaikan, untuk mencapai kontradiksi, $n^2 \leq c \cdot n$ untuk semua $n \geq n_0$.

Bagi dengan $n \geq 1$: maka $n \leq c$ untuk semua $n \geq n_0$ — bilangan bulat n tidak pernah melampaui sebuah bilangan tetap.

Ambil $n = \max(n_0, \lfloor c \rfloor + 1)$: maka $n > c$. Kontradiksi, jadi $n^2 \neq O(n)$. ∎

> **WAWASAN KUNCI.** Untuk *membuktikan* $f = O(g)$, tunjukkan satu pasangan (c, n_0). Untuk *menyangkalnya*, tunjukkan bahwa *setiap* pasangan gagal. Membuktikan bersifat eksistensial; menyangkal bersifat universal.

Buku: Subbab 3.3.3.

Membiarkan Mesin Memburu Saksinya

Kode 3.1 Memburu secara numerik konstanta c dan n_0 dalam definisi Big-O.

```cpp
// witness.cpp - Mencari konstanta c dan n0 yang menjadi saksi sebuah klaim O,
// dan menyaksikan pencarian itu gagal ketika klaimnya salah.
// Kompilasi: g++ -std=c++17 -O2 -o witness witness.cpp
#include <iostream>
#include <functional>
#include <string>
#include <cmath>
using namespace std;

// Coba tiap c di 1..cMax; untuk masing-masingnya, cari n0 terkecil sedemikian
// sehingga f(n) <= c*g(n) berlaku untuk tiap n di [n0, nMax]. Laporkan keberhasilan pertama.
bool findWitness(const string& claim,
                 function<double(double)> f,
                 function<double(double)> g,
                 int cMax, int nMax) {
    for (int c = 1; c <= cMax; ++c) {
        int n0 = -1;
        // Pindai turun dari nMax: n terakhir yang melanggar batasnya menetapkan n0.
        for (int n = nMax; n >= 1; --n) {
            if (f(n) > c * g(n)) { n0 = n + 1; break; }
            if (n == 1) n0 = 1;
        }
        if (n0 != -1 && n0 <= nMax) {
            cout << claim << " : witnessed by c = " << c
                 << ", n0 = " << n0 << "\n";
            return true;
        }
    }
```

```
    cout << claim << " : NO witness with c <= " << cMax
         << " on n <= " << nMax << " -- the claim looks false\n";
    return false;
}

int main() {
    findWitness("7n - 3 = O(n)      ", [](double n){ return 7*n - 3; },
                                       [](double n){ return n; }, 20, 100000);
    findWitness("3n^2+5n+7 = O(n^2) ", [](double n){ return 3*n*n + 5*n + 7; },
                                       [](double n){ return n*n; }, 20, 100000);
    findWitness("2n+10 = O(n)       ", [](double n){ return 2*n + 10; },
                                       [](double n){ return n; }, 20, 100000);
    findWitness("n^2 = O(n)         ", [](double n){ return n*n; },
                                       [](double n){ return n; }, 20, 100000);
    findWitness("2^n = O(n^10)      ", [](double n){ return (n<1000)?pow(2.0,n):1e300; },
                                       [](double n){ return pow(n,10); }, 20, 200);
    return 0;
}
```

```
7n - 3 = O(n)       : witnessed by c = 7, n0 = 1
3n^2+5n+7 = O(n^2)  : witnessed by c = 4, n0 = 7
2n+10 = O(n)        : witnessed by c = 3, n0 = 10
n^2 = O(n)          : NO witness with c <= 20 on n <= 100000 -- the claim looks false
2^n = O(n^10)       : NO witness with c <= 20 on n <= 200 -- the claim looks false
```

Kode 3.1 menegaskan $c=4, n_0=7$. Namun program hanya memeriksa berhingga banyak n: ia dapat mengisyaratkan bahwa sebuah klaim salah, tidak pernah membuktikannya. Buku: Subbab 3.3.4 dan Kode 3.2.

Big-Omega: Batas Bawah

DEFINISI (Big-Ω). $f(n) = \Omega(g(n))$ jika terdapat konstanta positif c dan n_0 sedemikian sehingga Persamaan (3.2) berlaku:

$$0 \le c \cdot g(n) \le f(n) \text{ untuk semua } n \ge n_0. \tag{3.2}$$

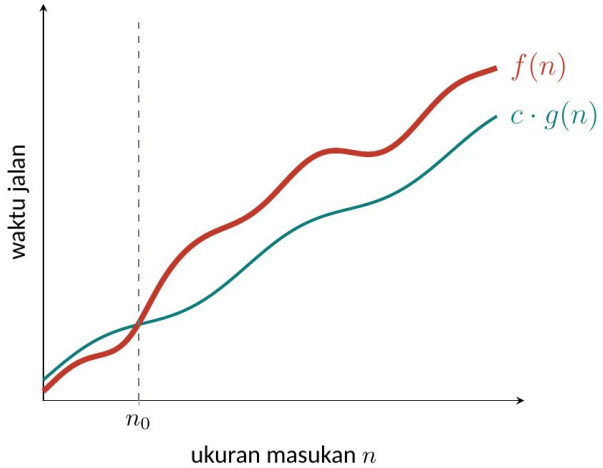
$7n - 3 = \Omega(n)$**:** $c = 1$ memerlukan $n \le 7n - 3$, yaitu $n \ge 1/2$, jadi $n_0 = 1$. $c = 6$ memerlukan $n \ge 3$. $c = 7$ tidak pernah berhasil.

$1/2\,n^2 - 3n = \Omega(n^2)$**:** $c = 1/4$ memerlukan $3n \le 1/4\,n^2$, yaitu $n \ge 12$.

Ω menyatakan kasus terbaik dan, lebih penting lagi, **batas bawah sebuah persoalan**: mencari di larik tak terurut adalah $\Omega(n)$ bagi *setiap* algoritma.

Buku: Subbab 3.4 dan Persamaan (3.2).

Big-Omega, dalam Gambar



Gambar 3.3 Big-Ω sebagai batas bawah. Di luar n_0, waktu jalan $f(n)$ tidak pernah jatuh di bawah $c \cdot g(n)$.

Pada Gambar 3.3, f boleh turun di bawah $c \cdot g$ pada awalnya, lalu naik melampauinya di n_0 dan tidak pernah kembali. Buku: Subbab 3.4 dan Gambar 3.3.

Big-Theta: Batas Ketat

DEFINISI (Big-Θ). $f(n) = \Theta(g(n))$ jika terdapat konstanta positif c_1, c_2, dan n_0 sedemikian sehingga Persamaan (3.3) berlaku:

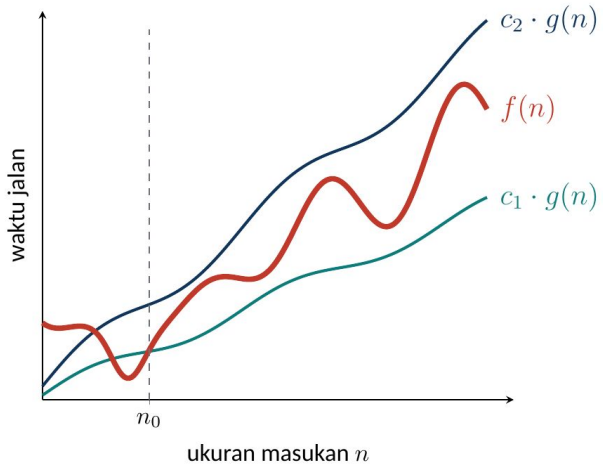
$$0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n) \text{ untuk semua } n \ge n_0. \tag{3.3}$$

TEOREMA. $f(n) = \Theta(g(n))$ jika dan hanya jika $f(n) = O(g(n))$ dan $f(n) = \Omega(g(n))$. (Ambil c_2 dari batas O, c_1 dari batas Ω, dan n_0 ambang yang lebih besar.)

Pada akhirnya f terjepit di antara dua kelipatan tetap dari g: laju pertumbuhannya sama, sampai konstanta. Dalam praktik: buktikan batas atas, buktikan batas bawah, gabungkan.

Buku: Subbab 3.5 dan Persamaan (3.3).

Big-Theta, dalam Gambar



Gambar 3.4 Big-Θ sebagai batas ketat. Setelah n_0, fungsi $f(n)$ terkurung di lorong antara $c_1 \cdot g(n)$ dan $c_2 \cdot g(n)$, sehingga f dan g tumbuh dengan laju yang sama.

Pada Gambar 3.4, melewati n_0 fungsinya berjalan di lorong antara $c_1 g(n)$ dan $c_2 g(n)$. Buku: Subbab 3.5 dan Gambar 3.4.

Sebuah Pembuktian Big-Theta secara Tuntas

KLAIM: $1/2\,n^2 - 3n = \Theta(n^2)$. *Atas:* $-3n \le 0$, jadi $1/2\,n^2 - 3n \le 1/2\,n^2$: $c_2 = 1/2$ mulai $n \ge 1$. *Bawah:* $c_1 n^2 \le 1/2\,n^2 - 3n \Leftrightarrow n \ge \dfrac{3}{\frac{1}{2} - c_1}$; dengan $c_1 = 1/4$, $n \ge 12$. *Gabungkan:* $c_1 = 1/4$, $c_2 = 1/2$, $n_0 = 12$. ∎

RENUNGKAN. $7n = O(n^5)$ benar tetapi tak berguna. Laporkan batas paling ketat yang dapat Anda buktikan: "Insertion Sort $\Theta(n^2)$ pada kasus terburuk" menyatakan bahwa batas itu tidak dapat diperbaiki lagi.

Buku: Subbab 3.5.

Little-o dan Little-omega

DEFINISI (little-o). $f(n) = o(g(n))$ jika untuk **setiap** konstanta positif c terdapat n_0 dengan Persamaan (3.4); setara dengan $f(n)/g(n) \to 0$.

$$0 \le f(n) < c \cdot g(n) \text{ untuk semua } n \ge n_0. \tag{3.4}$$

Big-O berkata "**terdapat** sebuah c"; little-o berkata "**untuk setiap** c". Little-ω adalah bayangan cerminnya: $f/g \to \infty$, dan $f = \omega(g) \Leftrightarrow g = o(f)$.

$2n = o(n^2)$: untuk c berapa pun, ambil $n_0 = \lceil 2/c \rceil + 1$. $2n^2 \neq o(n^2)$: $c = 1$ saja sudah gagal. $\lg n = o(n)$; $n^{100} = o(2^n)$.

Satu c yang gagal sudah cukup untuk menggugurkan klaim little-o. Buku: Subbab 3.6 dan Persamaan (3.4).

Analogi Bilangan

Tabel 3.2 Kelima notasi asimtotik dan perbandingan bilangan yang ditiru masing-masingnya.

Notasi	Arti bagi pertumbuhan	Analogi bilangan
$f(n)=O(g(n))$	f tumbuh tidak lebih cepat daripada g	$f \le g$
$f(n)=\Omega(g(n))$	f tumbuh setidaknya secepat g	$f \ge g$
$f(n)=\Theta(g(n))$	f tumbuh dengan laju yang sama dengan g	$f = g$
$f(n)=o(g(n))$	f tumbuh benar-benar lebih lambat daripada g	$f < g$
$f(n)=\omega(g(n))$	f tumbuh benar-benar lebih cepat daripada g	$f > g$

$O(g(n))$ sebenarnya sebuah **himpunan**; $f(n)=O(g(n))$ berarti $f(n)\in O(g(n))$. Tanda sama dengan di sini satu arah: $2n=O(n^2)$, tidak pernah "$O(n^2)=2n$".

Di dalam rumus, $\Theta(n)$ berarti *suatu fungsi tanpa nama* dari kelas itu: $T(n)=2T(n/2)+\Theta(n)$.

Tabel 3.2 adalah pelayan yang baik tetapi tuan yang buruk: ia patah pada ketrikotomian (dua halaman lagi). Buku: Subbab 3.6.1–3.6.2 dan Tabel 3.2.

Aljabar Notasi-Notasi Itu

Ketransitifan (kelimanya): $f \le c_1 g$ dan $g \le c_2 h$ memberi $f \le (c_1 c_2) h$ melewati $\max(n_1, n_2)$.

Kerefleksifan: $f = O(f) = \Omega(f) = \Theta(f)$; tidak pernah $f = o(f)$, sama seperti $x < x$ salah.

Kesetangkupan hanya untuk Θ; **kesetangkupan transpos**: $f = O(g) \Leftrightarrow g = \Omega(f)$ dan $f = o(g) \Leftrightarrow g = \omega(f)$.

Aturan jumlah $f_1 + f_2 = O(\max(g_1, g_2))$; **aturan hasil kali** $f_1 f_2 = O(g_1 g_2)$; **aturan konstanta** $k f_1 = O(g_1)$.

Aturan jumlah adalah asal "buang suku berorde rendah"; aturan konstanta adalah asal "buang konstanta". Buku: Subbab 3.7.1–3.7.4.

Mengapa Suku Dibuang — dan Di Mana Analogi Patah

Setiap polinomial adalah Θ dari pangkat tertingginya. Untuk $p(n)=a_d n^d+\cdots+a_0$ dengan $a_d>0$: atas $p(n)\le A\,n^d$ dengan $A=\sum|a_i|$; bawah $p(n)\ge a_d n^d - B\,n^{d-1}\ge \frac{a_d}{2}n^d$ begitu $n\ge 2B/a_d$.

$50\,n\log n\to\Theta(n\log n)$; $7n-3\to\Theta(n)$; $8n^2\log n+5n^2+n\to\Theta(n^2\log n)$.

> **MITOS YANG SERING DIPERCAYA.** “Dua fungsi apa pun selalu dapat dibandingkan.” Salah: n dan $n^{1+\sin n}$ (pangkatnya berayun antara 0 dan 2) bukan O maupun Ω satu terhadap yang lain. **Ketrikotomian gagal.**

Waktu jalan yang nyata tersusun dari polinomial, logaritma, dan eksponensial, yang semuanya dapat dibandingkan — jadi analogi itu berlaku sehari-hari. Buku: Subbab 3.7.5–3.7.6.

Lantai, Logaritma, Stirling

$x-1<\lfloor x\rfloor\le x\le\lceil x\rceil<x+1$ dan $\lceil n/2\rceil+\lfloor n/2\rfloor=n$ — sebabnya kedua paruh Merge Sort selalu berjumlah n.

$\lg n=\log_2 n$, $\ln n=\log_e n$, $\lg^k n=(\lg n)^k$ — **bukan** $\lg\lg n$. Perubahan basis menjadikan setiap logaritma kelipatan konstan dari logaritma lain, Persamaan (3.5):

$$\log_2 n=\Theta(\log_{10} n)=\Theta(\ln n)=\Theta(\log_b n)\ \text{untuk sembarang nilai tetap}\ b>1. \tag{3.5}$$

Setiap eksponensial mengalahkan setiap polinomial: $n^d=o(a^n)$ untuk $a>1$. Stirling, Persamaan (3.6):

$$n!=\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right). \tag{3.6}$$

Jadi $n!=o(n^n)$, $n!=\omega(2^n)$, dan $\lg(n!)=\Theta(n\lg n)$ — alasan tidak ada pengurutan berbasis perbandingan yang mengalahkan $n\lg n$ (Bab 5). Basis hanya penting di dalam pangkat seperti $n^{\log_b a}$. Buku: Subbab 3.8.2–3.8.7, Tabel 3.3, dan Persamaan (3.5)–(3.6).

Identitas Logaritma

Tabel 3.3 Kesamaan logaritma yang dipakai dalam bab ini. Dua yang terakhir adalah yang membuat Teorema Master bekerja.

Kesamaan	Mengapa ia benar
$\log_b(xy) = \log_b x + \log_b y$	pangkat bertambah ketika perpangkatan dikalikan
$\log_b(x/y) = \log_b x - \log_b y$	pangkat berkurang ketika perpangkatan dibagi
$\log_b(x^a) = a \log_b x$	$(b^y)^a = b^{ya}$
$\log_b x = \dfrac{\log_c x}{\log_c b}$	penggantian basis; $\log_c b$ adalah konstanta
$b^{\log_b x} = x$	definisinya, dibaca mundur
$x^{\log_b y} = y^{\log_b x}$	ambil $\log_b$ kedua ruas: keduanya memberi $\log_b x \cdot \log_b y$
$a^{\log_b n} = n^{\log_b a}$	baris sebelumnya, dengan $x = a$, $y = n$
$\log_b a = \dfrac{1}{\log_a b}$	penggantian basis dengan $c = a$

Jantung Teorema Master: $a^{\log_b n} = n^{\log_b a}$. Ambil $\log_b$ kedua ruas: masing-masing menghasilkan $(\log_b n)(\log_b a)$, dan $\log_b$ adalah fungsi satu-satu. ∎

Tabel 3.3: dua baris terakhir itulah yang dipakai Teorema Master. Buku: Subbab 3.8.6 dan Tabel 3.3.

Fungsi-Fungsi yang Sering Muncul, Berurutan

Tabel 3.4 Fungsi-fungsi yang sering muncul, dalam urutan pertumbuhan menaik. Setiap baris adalah $o(\cdot)$ dari setiap baris di bawahnya.

Fungsi	Kelas pertumbuhan	Nilai pada n=1024
1	konstan	1
$\lg^* n$	log teriterasi	4
$\lg \lg n$	log log	≈ 3.3
$\lg n$	logaritmik	10
$\sqrt{n} = n^{1/2}$	akar	32
n	linear	1024
$n \lg n$	linearitmik	10240
n^2	kuadratik	1048576
n^3	kubik	$\approx 1.07 \times 10^9$
2^n	eksponensial	besar secara astronomis
$n!$	faktorial	lebih besar lagi

Tabel 3.4 (Lanjutan) *Fungsi-fungsi yang sering muncul, dalam urutan pertumbuhan menaik. Setiap baris adalah $o(\cdot)$ dari setiap baris di bawahnya.*

Fungsi	Kelas pertumbuhan	Nilai pada n=1024
n^n	superfaktorial	lebih besar lagi daripada itu

$\lg^* n$ mencacah berapa kali lg diambil sebelum mencapai 1: dari 65536 ke 2^{65536} ia hanya naik satu. Fibonacci tumbuh seperti ϕ^n dengan $\phi \approx 1.618$ — alasan Fibonacci rekursif naif tidak dapat dipakai.

Setiap baris Tabel 3.4 adalah $o(\cdot)$ dari setiap baris di bawahnya; Kode 3.3 memeriksa $\lg^*$, Stirling, dan Fibonacci secara numerik. Buku: Subbab 3.8.8–3.8.9 dan Tabel 3.4.

Deret Aritmetika

DEFINISI (deret aritmetika). Persamaan (3.7):

$$\sum_{i=1}^{n} i = 1+2+3+\cdots+n = \frac{n(n+1)}{2} = \Theta(n^2). \tag{3.7}$$

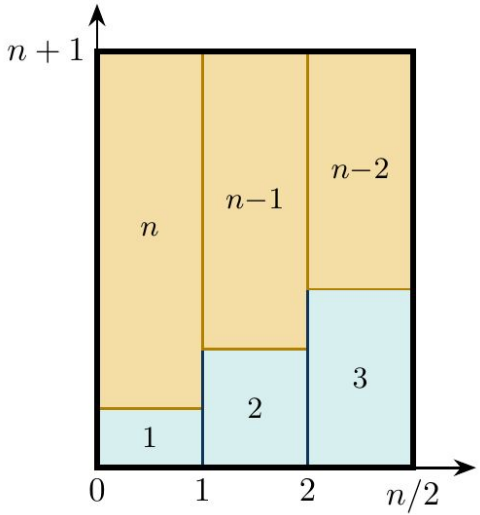
Siasat Gauss: tulis $S = 1+2+\cdots+n$ maju dan mundur lalu jumlahkan kolom demi kolom. Masing-masing dari n kolom berjumlah $n+1$, jadi $2S = n(n+1)$.

Dengan induksi: $S(1) = 1 = \frac{1\cdot 2}{2}$; dan $S(n) = S(n-1) + n = \frac{(n-1)n}{2} + n = \frac{n(n+1)}{2}$.

Ia $\Theta(n^2)$ ($\frac{n^2}{2} \le S \le n^2$), dan ia *adalah* kasus terburuk Insertion Sort: $\sum_{j=2}^{n} (j-1) = \frac{(n-1)n}{2}$.

Buku: Subbab 3.9.1–3.9.2, Persamaan (3.7), dan Gambar 3.5–3.8.

Pemasangan Gauss, dalam Gambar



Gambar 3.5 Pemasangan ala Gauss dibuat geometris. Memasangkan batang pertama dengan yang terakhir, yang kedua dengan yang kedua dari belakang, dan seterusnya, mengemas ulang tangga itu menjadi persegi panjang selebar $n/2$ dan setinggi $n+1$, yang luasnya $n(n+1)/2$.

Gambar 3.5 melipat tangga menjadi persegi panjang selebar $n/2$ dan setinggi $n+1$: luasnya $n(n+1)/2$. Buku: Subbab 3.9.1 dan Gambar 3.6.

Deret Geometri

DEFINISI (deret geometri). Untuk $a \neq 1$ real, Persamaan (3.8):

$$\sum_{i=0}^{n} a^i = 1 + a + a^2 + \cdots + a^n = \frac{1 - a^{n+1}}{1 - a}. \tag{3.8}$$

Penurunannya: $S - aS = 1 - a^{n+1}$, karena setiap suku di tengah saling meniadakan. (Untuk $a = 1$ jumlahnya $n+1$.)

$a < 1$**:** konvergen ke $\frac{1}{1-a}$ — $\Theta(1)$, suku **pertama** mendominasi. $a > 1$**:** $\Theta(a^n)$, suku **terakhir** mendominasi. $a = 1$**:** $\Theta(n)$, setiap suku sama besar sumbangannya.

WAWASAN KUNCI. Mengenali rezim mana yang sedang Anda hadapi adalah seluruh keterampilan analisis pohon rekursi — dan itulah persis tiga kasus Teorema Master.

Buku: Subbab 3.9.3 dan Persamaan (3.8).

Harmonik, Teleskopis, Integral

Harmonik, Persamaan (3.9), dengan $\gamma \approx 0.5772$; mengelompokkan suku dalam blok yang panjangnya berlipat dua, masing-masing sekurang-kurangnya $1/2$, memberi $H_n \geq 1/2 \lg n$:

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \ln n + \gamma + O\left(\frac{1}{n}\right) = \Theta(\lg n). \tag{3.9}$$

Teleskopis: $\sum_{i=1}^{n} (a_i - a_{i-1}) = a_n - a_0$, misalnya $\sum_{i=1}^{n} \frac{1}{i(i+1)} = 1 - \frac{1}{n+1}$.

Batas integral untuk f naik, Persamaan (3.10) (terbalik untuk f turun); ia memberi $\lg(n!) = \Theta(n \lg n)$ tanpa Stirling:

$$\int_{m-1}^{n} f(x)\,dx \leq \sum_{i=m}^{n} f(i) \leq \int_{m}^{n+1} f(x)\,dx. \tag{3.10}$$

Buku: Subbab 3.9.4–3.9.6 dan Persamaan (3.9)–(3.10).

Kartu Rujukan Jumlahan

Tabel 3.5 Jumlahan-jumlahan yang dipakai di sepanjang buku ini, dengan bentuk tertutup dan kelas asimtotiknya.

Jumlahan	Bentuk tertutup	Asimtotik
$\sum_{i=1}^{n} 1$	n	$\Theta(n)$
$\sum_{i=1}^{n} i$	$\frac{n(n+1)}{2}$	$\Theta(n^2)$
$\sum_{i=1}^{n} i^2$	$\frac{n(n+1)(2n+1)}{6}$	$\Theta(n^3)$
$\sum_{i=1}^{n} i^k$	—	$\Theta(n^{k+1})$ untuk $k \geq 0$
$\sum_{i=0}^{n} a^i, a \neq 1$	$\frac{1-a^{n+1}}{1-a}$	$\Theta(1)$ jika $a<1$; $\Theta(n)$ jika $a=1$; $\Theta(a^n)$ jika $a>1$
$\sum_{i=0}^{\infty} a^i$ dengan $\lvert a \rvert<1$	$\frac{1}{1-a}$	$\Theta(1)$
$\sum_{i=1}^{n} \frac{1}{i}$	$\ln n+\gamma+O(1/n)$	$\Theta(\lg n)$

Tabel 3.5 (Lanjutan) *Jumlahan-jumlahan yang dipakai di sepanjang buku ini, dengan bentuk tertutup dan kelas asimtotiknya.*

Jumlahan	Bentuk tertutup	Asimtotik
$\sum_{i=1}^{n} \lg i$	$\lg(n!)$	$\Theta(n \lg n)$
$\sum_{i=1}^{n} (a_i - a_{i-1})$	$a_n - a_0$	teleskopis

Tabel 3.5 adalah kartu yang patut disimpan. Kode 3.4 memeriksa empat barisnya secara brute force — angka 5050 *milik Gauss muncul, dan setelah sejuta suku* H_n *baru mencapai* 14.39*. Buku: Subbab 3.9 dan Tabel 3.5.*

Rekurensi

DEFINISI. Sebuah **rekurensi** menggambarkan sebuah fungsi lewat nilainya pada masukan yang lebih kecil, ditambah **syarat batas** untuk masukan terkecil.

Kebanyakan rekurensi kita berbentuk bagi dan kuasai Persamaan (3.11): $a \geq 1$ subpersoalan, masing-masing berukuran n/b dengan $b > 1$, ditambah $f(n)$ untuk membagi dan menggabungkan.

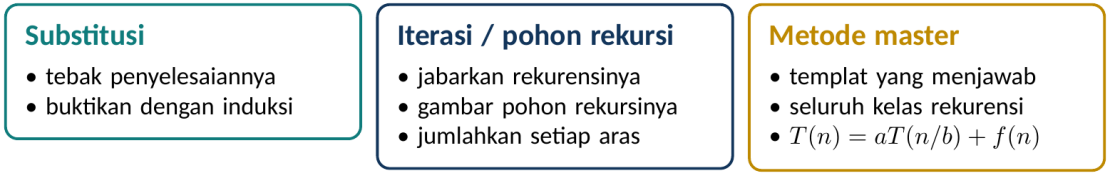
$$T(n) = aT(n/b) + f(n), \tag{3.11}$$

Merge Sort $a = b = 2$; *binary search* $a = 1, b = 2$; Strassen $a = 7, b = 2$ — a dan b saling bebas.

Kita boleh mengabaikan **lantai dan langit-langit**, menghilangkan **syarat batas** ($T(n) = \Theta(1)$ untuk n kecil), dan wajib memperhatikan $\leq$ / $\geq$ / $=$ (O / Ω / Θ).

Rekurensi Merge Sort sebagai definisi dua kasus adalah Gambar 3.10. Buku: Subbab 3.10.1–3.10.3 dan Persamaan (3.11).

Tiga Metode



Gambar 3.6 Tiga metode untuk menyelesaikan rekurensi: substitusi (tebak, lalu sahkan dengan induksi), iterasi atau pohon rekursi (bentangkan rekurensinya dan gambar), dan metode master (sebuah cetakan yang menjawab sekelas rekurensi sekaligus).

Gambar 3.6: ***pohon rekursi*** *menemukan jawaban,* ***substitusi*** *membuktikannya,* ***metode master*** *melompati keduanya bila bentuknya cocok. Coba Teorema Master dulu; jika gagal, gambar pohonnya, lalu buktikan dengan substitusi. Buku: Subbab 3.10.4 dan Gambar 3.11.*

Substitusi: Batas Atas Kubik

Tebak $T(n) \le c\,n^3$; **andaikan** berlaku untuk semua $k < n$, khususnya $k = n/2$.

$$T(n) = 4\,T(n/2) + n \le 4\,c\,\frac{n^3}{8} + n = \frac{c\,n^3}{2} + n = c\,n^3 - \left(\frac{c\,n^3}{2} - n\right).$$

$\le c\,n^3$ asalkan $\dfrac{c\,n^3}{2} \ge n$, yaitu $c \ge 2/n^2$: $c \ge 2$ berlaku untuk setiap $n \ge 1$.

Basis: $T(1) = 1 \le c$. Jadi $T(n) \le 2\,n^3$, dan $T(n) = O(n^3)$. ∎

Langkah yang perlu dipelajari adalah menulis $c\,n^3 - (\text{kelonggaran})$*: sasarannya di kiri, semua yang harus disingkirkan dalam satu kurung. Buku: Subbab 3.11.1.*

Percobaan yang Gagal, dan Perbaikannya

Tebak $T(n) \le c\,n^2$: $4\,c\frac{n^2}{4} + n = c\,n^2 + n$, dan $c\,n^2 + n \le c\,n^2$ memerlukan $n \le 0$. Mustahil untuk setiap c.

JEBAKAN. "$c\,n^2 + n$ adalah $O(n^2)$, selesai" itu salah: induksinya harus menghasilkan kembali hipotesisnya *persis*, dengan c yang sama. Sisa-sisa itu menumpuk di sepanjang tingkat.

Perkuat: $T(n) \le c_1 n^2 - c_2 n$. Maka $4\left(c_1\frac{n^2}{4} - c_2\frac{n}{2}\right) + n = c_1 n^2 - c_2 n - (c_2 n - n) \le c_1 n^2 - c_2 n$ bila $c_2 \ge 1$.

WAWASAN KUNCI. Tertinggal (batas) $+$ (sisa positif)? **Kurangkan** suku berorde sama dengan sisa itu dari tebakannya. Hipotesis yang lebih kuat juga merupakan andaian yang lebih kuat.

Kode 3.5 di bab ini menghitung $T(n)$ secara persis: $T(n) = 2n^2 - n$ pada setiap pangkat dua, jadi $c_1 = 2$, $c_2 = 1$ adalah yang terbaik; rekurensi Merge Sort memberi tepat $n \lg n + n$. Buku: Subbab 3.11.2–3.11.3, 3.11.5, dan Gambar 3.12–3.13.

Kasus Basis dan Penggantian Peubah

Geser kasus basisnya. Untuk $T(n)=2T(\lfloor n/2 \rfloor)+n$, $T(1)=1$, tebakan $c\,n\lg n$ gagal di $n=1$ ($\lg 1=0$). Buktikan basisnya di $n=2,3$: $4\le 2c$ dan $5\le 3c\lg 3$, keduanya benar untuk $c\ge 2$.

Ganti peubah. $T(n)=2T(\sqrt{n})+\lg n$: ambil $m=\lg n$ dan $S(m)=T(2^m)$, sehingga $S(m)=2S(m/2)+m=\Theta(m\lg m)$.

Kembali ke n: $T(n)=\Theta(\lg n\cdot\lg\lg n)$.

Batas asimtotik hanya perlu berlaku untuk $n\ge n_0$; itulah yang mengizinkan kita menggeser basisnya. Buku: Subbab 3.11.4.

Iterasi: Mengurai Rekurensi

$$T(n) = n + 3T(n/4) = n + \frac{3n}{4} + 9T(n/16) = n + \frac{3n}{4} + \frac{9n}{16} + 27T(n/64) = \cdots$$

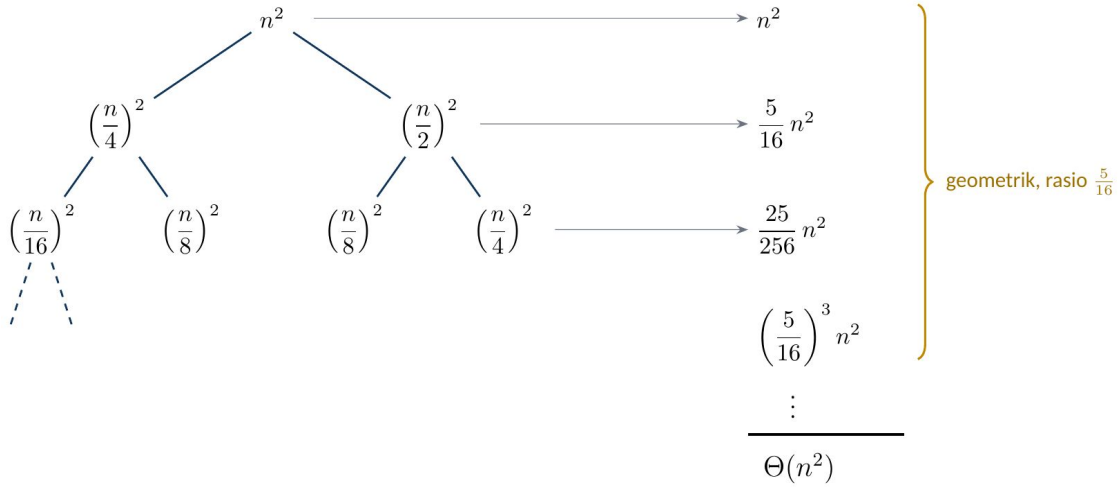
$$T(n) \le n \sum_{i \ge 0} (3/4)^i + \Theta(n^{\log_4 3}) = 4n + \Theta(n^{0.793}).$$

Daun-daunnya berbiaya $n^{\log_4 3} = o(n)$, jadi $T(n) = O(n)$ — dan $\Theta(n)$, karena akarnya saja sudah berbiaya n.

WAWASAN KUNCI. Rasio di bawah 1 ⇒ deretnya konvergen ⇒ **akar** mendominasi ⇒ $T(n) = \Theta(f(n))$.

Buku: Subbab 3.12.1 dan Gambar 3.14.

Pohon yang Tingkatnya Menyusut



Gambar 3.7 Pohon rekursi yang sudah lengkap untuk $T(n)=T(n/4)+T(n/2)+n^2$. Total tiap tingkatnya $n^2, \frac{5}{16}n^2, \frac{25}{256}n^2, \ldots$ membentuk deret geometri berasio $5/16<1$, yang berjumlah $\Theta(n^2)$.

Gambar 3.7: untuk $T(n)=T(n/4)+T(n/2)+n^2$ setiap tingkat berbiaya $(1/4)^2+(1/2)^2=\frac{5}{16}$ dari tingkat di atasnya. Buku: Subbab 3.12.2 dan Gambar 3.16.

Menjumlahkan Pohon Itu

Tingkat 0: n^2. Tingkat 1: $\frac{n^2}{16}+\frac{n^2}{4}=\frac{5n^2}{16}$. Tingkat 2: $\frac{n^2+4n^2+4n^2+16n^2}{256}=\left(\frac{5}{16}\right)^2 n^2$.

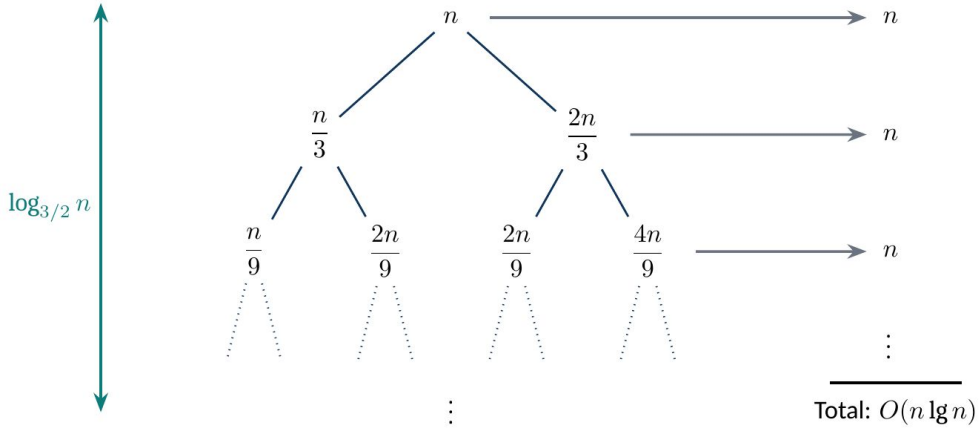
$T(n)\le n^2\sum_{i\ge 0}\left(\frac{5}{16}\right)^i=\frac{16}{11}n^2\approx 1.4545\,n^2$, dan $T(n)\ge n^2$ dari akarnya saja.

Jadi $T(n)=\Theta(n^2)$: semua yang di bawah akar hanya menambah $45\ \%$. Subpersoalannya tidak sama besar — Teorema Master tidak dapat menyentuhnya; pohonnya bisa.

Kode 3.6 di bab ini menjumlahkan tujuh tingkat secara numerik: $327680/1048576=5/16$ *tepat. Buku: Subbab 3.12.2 dan Gambar 3.15.*

Pohon yang Tingkatnya Sama

$$T(n) = T(n/3) + T(2n/3) + n$$



Gambar 3.8 Pohon rekursi untuk $T(n)=T(n/3)+T(2n/3)+n$. Tiap tingkat berjumlah tepat n, dan lintasan akar-ke-daun yang terpanjang memiliki $\log_{3/2} n$ tingkat, sehingga totalnya $O(n \lg n)$.

Gambar 3.8: untuk $T(n)=T(n/3)+T(2n/3)+n$ setiap tingkat berjumlah n ($\frac{n}{9}+\frac{2n}{9}+\frac{2n}{9}+\frac{4n}{9}=n$). Lintasan terpanjang memiliki $\log_{3/2} n$ tingkat, yang terpendek $\log_3 n$, jadi $T(n)=\Theta(n \lg n)$ — paling banyak sekitar $1.71\, n \lg n$. Buku: Subbab 3.12.3 dan Gambar 3.17.

Cacah Daun, dan Tiga Bentuk Pohon

Untuk $aT(n/b)+f(n)$: tingkat i memuat a^i simpul berukuran n/b^i; kedalamannya $\log_b n$; jadi banyaknya daun adalah

Persamaan (3.12) — rumus terpenting di bab ini:

$$a^{\log_b n} = n^{\log_b a}, \tag{3.12}$$

Menjumlahkan biaya tingkat dan daun memberi Persamaan (3.13):

$$\begin{array}{lcl} \text{biaya aras } i & = & a^i f\left(\dfrac{n}{b^i}\right), \quad \text{dan} \quad T(n) = \\ & & \underbrace{\displaystyle\sum_{i=0}^{\log_b n - 1} a^i f\left(\dfrac{n}{b^i}\right)}_{\text{pembagian dan penggabungan, semua aras}} + \underbrace{\Theta(n^{\log_b a})}_{\text{semua daun}}. \end{array} \tag{3.13}$$

Tabel 3.6 Tiga bentuk yang dapat diambil sebuah pohon rekursi, dan apa yang diakibatkan masing-masingnya. Ketiganya berpadanan persis dengan Kasus 3, 2, dan 1 Teorema Master.

Jenis	Rasio antartingkat	Dikuasai oleh	Hasil
Menurun	rasio < 1	akarnya (tingkat teratas)	$\Theta(f(n))$
Setara	rasio $= 1$	semua tingkat sama rata	$\Theta(f(n)\log n)$
Menaik	rasio > 1	daunnya (tingkat terbawah)	$\Theta(n^{\log_b a})$

Merge Sort: $n^{\log_2 2} = n$ *daun; binary search:* $n^{\log_2 1} = 1$ *daun. Tabel 3.6 adalah Teorema Master yang menyamar. Buku: Subbab 3.12.4–3.12.5, Persamaan (3.12)–(3.13), dan Tabel 3.6.*

Teorema Master

TEOREMA MASTER. Misalkan $a \geq 1$ dan $b > 1$ konstanta, $f(n)$ positif secara asimtotik, dan $T(n) = aT(n/b) + f(n)$ (dengan n/b dibaca $\lfloor n/b \rfloor$ atau $\lceil n/b \rceil$). **Kasus 1.** Jika $f(n) = O(n^{\log_b a - \varepsilon})$ untuk suatu $\varepsilon > 0$, maka $T(n) = \Theta(n^{\log_b a})$. **Kasus 2.** Jika $f(n) = \Theta(n^{\log_b a})$, maka $T(n) = \Theta(n^{\log_b a} \lg n)$. **Kasus 3.** Jika $f(n) = \Omega(n^{\log_b a + \varepsilon})$ untuk suatu $\varepsilon > 0$, **dan** $af(n/b) \leq cf(n)$ untuk suatu $c < 1$ dan semua n yang cukup besar, maka $T(n) = \Theta(f(n))$.

SATU GAGASANNYA. Bandingkan biaya akar $f(n)$ dengan biaya daun $n^{\log_b a}$. Yang lebih besar secara polinomial menang; bila seri, bayarlah untuk semua $\lg n$ tingkat.

a adalah faktor percabangan, b faktor penyusutan, $f(n)$ kerja setempat, $n^{\log_b a}$ cacah daun. Buku: Subbab 3.13.1–3.13.2.

Epsilon dan Keteraturan

ε berarti "secara polinomial". Tulis f dan $n^{\log_b a}$ sebagai pangkat n lalu kurangkan pangkatnya. Faktor $n^{0.0001}$ dihitung; faktor $\lg n$ **tidak**, karena $\lg n = o(n^{\varepsilon})$ untuk setiap $\varepsilon > 0$.

Keteraturan, Persamaan (3.14): total kerja anak-anak paling banyak sebagian tetap dari kerja induknya, sehingga biaya tingkat menyusut secara geometris:

$$a f\left(\frac{n}{b}\right) \le c f(n) \text{ untuk suatu konstanta } c < 1 \text{ dan semua nilai besar } n. \tag{3.14}$$

Otomatis untuk polinomial: $f(n) = n^k$ dengan $k > \log_b a$ memberi $c = a/b^k < 1$.

Ia bisa gagal: $T(n) = T(n/2) + n(2 - \cos n)$. Di tempat $\cos n \approx 1$ dan $\cos(n/2) \approx -1$, syaratnya memaksa $c \ge 1.5$.

Buku: Subbab 3.13.3–3.13.4 dan Persamaan (3.14).

Kartu Ringkasan, dan Lima Langkah

Tabel 3.7 Tiga kasus Teorema Master untuk $T(n) = aT(n/b) + f(n)$, beserta gagasan yang diungkapkan masing-masingnya.

Kasus	Syarat atas $f(n)$	Kesimpulan	Gagasannya
1	$f(n) = O(n^{\log_b a - \varepsilon})$ untuk suatu $\varepsilon > 0$	$T(n) = \Theta(n^{\log_b a})$	daun berkuasa
2	$f(n) = \Theta(n^{\log_b a})$	$T(n) = \Theta(n^{\log_b a} \lg n)$	tiap tingkat berbiaya sama
3	$f(n) = \Omega(n^{\log_b a + \varepsilon})$ untuk suatu $\varepsilon > 0$, **dan** $a f(n/b) \le c f(n)$ dengan $c < 1$	$T(n) = \Theta(f(n))$	akar berkuasa

1. Ambil a, b, $f(n)$; periksa $a \ge 1$, $b > 1$, f positif. **2.** Hitung $n^{\log_b a}$. **3.** Bandingkan pangkatnya. **4.** Tetapkan kasusnya — atau kenali celahnya. **5.** Pada Kasus 3, periksa keteraturannya.

Tabel 3.7 adalah kartu yang perlu dihafal. Buku: Subbab 3.13.5 dan 3.13.7, serta Tabel 3.7.

Di Mana Teorema Itu Diam

Celah 1–2: $T(n)=2T(n/2)+n/\lg n$ — lebih kecil daripada n, tetapi tidak secara polinomial. Jawaban sebenarnya $\Theta(n\lg\lg n)$.

Celah 2–3: $T(n)=2T(n/2)+n\lg n$. Bukan Kasus 2 ($\lg n\to\infty$), bukan Kasus 3 ($\lg n\geq c\,n^{\varepsilon}$ gagal). Tingkat i berbiaya $n(\lg n-i)$, sehingga Persamaan (3.15):

$$T(n)=\sum_{i=0}^{\lg n-1} n(\lg n-i)=n\sum_{j=1}^{\lg n} j=n\cdot\frac{\lg n(\lg n+1)}{2}=\Theta(n\lg^2 n). \tag{3.15}$$

JEBAKAN. "Kasus 3, jadi $\Theta(n\lg n)$" salah dua kali: kasusnya tidak berlaku, dan jawabannya adalah $\Theta(n\lg^2 n)$.

Di luar jangkauan: f tidak positif ($4T(n/2)-n^2$); a tidak konstan ($2^nT(n/2)$); bagian tidak sama ($T(n/3)+T(2n/3)$); pengurangan ($T(n-1)+n=\Theta(n^2)$ lewat teleskop).

Buku: Subbab 3.13.6 dan Persamaan (3.15).

Dikerjakan: Empat Rekurensi

$T(n/2)+1$: $n^{\log_2 1}=1=f$. Kasus 2: $\Theta(\lg n)$ — *binary search*.

$9T(n/3)+n$: $n^{\log_3 9}=n^2$; $1<2$, $\varepsilon=1$. Kasus 1: $\Theta(n^2)$ — daun menenggelamkan akar.

$2T(n/2)+\Theta(n)$: seri tepat dengan n. Kasus 2: $\Theta(n\lg n)$ — Merge Sort.

$3T(n/4)+n\lg n$: $\log_4 3\approx 0.7925$; dengan $\varepsilon=0.2$, $n\lg n/n^{0.9925}\to\infty$. Keteraturan: $3\cdot\frac{n}{4}\lg\frac{n}{4}\le 3/4\,n\lg n$. Kasus 3: $\Theta(n\lg n)$.

Buku: Subbab 3.14.1–3.14.4 dan Pseudokode 3.1.

Dikerjakan: a dan b Sama, Tiga Kasus

$a=4, b=2, n^{\log_2 4}=n^2$: $+n$ Kasus 1, $\Theta(n^2)$; $+n^2$ Kasus 2, $\Theta(n^2 \lg n)$; $+n^3$ Kasus 3 dengan $c=1/2$, $\Theta(n^3)$.

$8T(n/2)+n^2$**:** Kasus 1, $\Theta(n^3)$ — perkalian matriks bagi dan kuasai, tidak lebih baik daripada tiga gelung.

Strassen $7T(n/2)+n^2$**:** $\log_2 7 \approx 2.807$, Kasus 1: $\Theta(n^{\lg 7})$. Satu perkalian lebih sedikit memangkas pangkatnya.

$T(n/2)+n$**:** Kasus 3, $c=1/2$: $\Theta(n)$ $(n+\frac{n}{2}+\cdots<2n)$. $2T(n/2)+n^2/\lg n$**:** keteraturan menjadi perhitungan sungguhan — rasionya $\frac{\lg n}{2(\lg n-1)} \le \frac{2}{3}$ untuk $n \ge 16$ — menghasilkan $\Theta(n^2/\lg n)$.

Buku: Subbab 3.14.5–3.14.20.

Kedua Puluhnya Sekilas Pandang

Tabel 3.8 Kedua puluh soal terkerjakan §3.14 dalam satu pandangan, lengkap dengan kasus yang menaungi masing-masingnya.

Rekurensi	$\log_b a$	$f(n)$	Kasus	$T(n)$
$T(n)=T(n/2)+1$	0	1	2	$\Theta(\lg n)$
$T(n)=9T(n/3)+n$	2	n	1	$\Theta(n^2)$
$T(n)=2T(n/2)+n$	1	n	2	$\Theta(n\lg n)$
$T(n)=3T(n/4)+n\lg n$	0.7925	$n\lg n$	3	$\Theta(n\lg n)$
$T(n)=4T(n/2)+n^3$	2	n^3	3	$\Theta(n^3)$
$T(n)=2T(n/2)+n\lg n$	1	$n\lg n$	celah	$\Theta(n\lg^2 n)$
$T(n)=4T(n/2)+n$	2	n	1	$\Theta(n^2)$
$T(n)=4T(n/2)+n^2$	2	n^2	2	$\Theta(n^2\lg n)$
$T(n)=8T(n/2)+n^2$	3	n^2	1	$\Theta(n^3)$
$T(n)=7T(n/2)+n^2$	2.807	n^2	1	$\Theta(n^{\lg 7})$
$T(n)=8T(n/2)+n^3$	3	n^3	2	$\Theta(n^3\lg n)$
$T(n)=2T(n/4)+\sqrt{n}$	0.5	$\sqrt{n}$	2	$\Theta(\sqrt{n}\lg n)$

Tabel 3.8 (Lanjutan) *Kedua puluh soal terkerjakan §3.14 dalam satu pandangan, lengkap dengan kasus yang menaungi masing-masingnya.*

Rekurensi	$\log_b a$	$f(n)$	Kasus	$T(n)$
$T(n)=3T(n/3)+n/2$	1	$n/2$	2	$\Theta(n\lg n)$
$T(n)=16T(n/4)+n^2$	2	n^2	2	$\Theta(n^2\lg n)$
$T(n)=T(n/2)+n$	0	n	3	$\Theta(n)$
$T(n)=2T(n/2)+n/\lg n$	1	$n/\lg n$	celah	$\Theta(n\lg\lg n)$
$T(n)=5T(n/2)+n^2$	2.322	n^2	1	$\Theta(n^{\lg 5})$
$T(n)=T(2n/3)+1$	0	1	2	$\Theta(\lg n)$
$T(n)=2T(n/2)+n^2/\lg n$	1	$n^2/\lg n$	3	$\Theta(n^2/\lg n)$
$T(n)=T(n/2)+n(2-\text{co}$	0	berayun	tidak ada	keteraturan gagal

Tabel 3.8: pemecah Teorema Master di bab ini (Kode 3.7) mereproduksi setiap baris dengan mesin dan mengenali kedua celah sebagai celah. Buku: Subbab 3.14.21 dan Tabel 3.8.

Merge Sort: Cetak Biru dan Algoritma

DEFINISI. Bagi dan kuasai: BAGI menjadi subpersoalan yang lebih kecil dari jenis yang sama; KUASAI secara rekursif (langsung bila kecil); GABUNGKAN penyelesaiannya.

Untuk pengurutan: **bagi** di titik tengah ($\Theta(1)$ dengan indeks); **kuasai** tiap paruh (satu elemen sudah terurut); **gabungkan** dengan **menggabung** dua paruh yang terurut.

Pseudokode 3.1 Merge Sort pada irisan $A[p..r]$, dengan MERGE diuraikan dalam kata-kata; Kode 3.2 memberikan kode lengkapnya.

```
MERGE-SORT(A, p, r)
    if p < r then
        q = floor((p + r) / 2)
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q+1, r)
        MERGE(A, p, q, r)

MERGE(A, p, q, r)
    Ambil yang lebih kecil di antara dua elemen terdepan
    A[p..q] dan A[q+1..r], taruh ke dalam hasil. Ulangi
    sampai keduanya kosong. Salin hasilnya kembali ke A[p..r].
```

Pada Pseudokode 3.1, `if p < r` adalah kasus basis yang menyamar: satu elemen atau tanpa elemen berarti tidak ada yang perlu dikerjakan. Buku: Subbab 3.15.1–3.15.2 dan Pseudokode 3.2.

Penelusuran Lengkap

```
Tingkat 0: [1, 5, 2, 4 | 6, 3, 2, 6]
Tingkat 1: [1, 5 | 2, 4]   [6, 3 | 2, 6]
Tingkat 2: [1|5] [2|4]     [6|3] [2|6]
Tingkat 3: [1][5] [2][4]   [6][3] [2][6]      <- tunggalan: kasus-kasus basisnya
Tingkat 3 -> 2: [1]+[5]=[1,5]   [2]+[4]=[2,4]   [6]+[3]=[3,6]   [2]+[6]=[2,6]
Tingkat 2 -> 1: [1,5]+[2,4]=[1,2,4,5]           [3,6]+[2,6]=[2,3,6,6]
Tingkat 1 -> 0: [1,2,4,5]+[2,3,6,6] = [1, 2, 2, 3, 4, 5, 6, 6]
```

Penggabungan terakhir: $1 < 2$, ambil 1; seri di 2 — ambil yang **kiri** (stabilitas); lalu $2, 3, 4, 5$; daftar kiri habis, salin $6, 6$.

Buku: Subbab 3.15.3.

MERGE Itu Benar

> **INVARIAN GELUNG.** Pada awal setiap iterasi: (a) $A[p..k-1]$ memuat $k-p$ elemen terkecil dari $L \cup R$, terurut; (b) $L[i]$ dan $R[j]$ adalah elemen terkecil L dan R yang belum disalin kembali.

Inisialisasi: $k = p$, awalannya kosong; $i = j = 0$ menunjuk minimum dari L dan R yang terurut.

Pemeliharaan: jika $L[i] \le R[j]$, $L[i]$ mengalahkan semua sisa di kedua daftar, jadi menambahkannya menjaga (a); i maju, menjaga (b). Simetris untuk kasus sebaliknya.

Terminasi: satu daftar habis; sisa daftar lainnya terurut dan tidak lebih kecil daripada yang sudah ditempatkan; gelung penyapu menambahkannya. Paling banyak $r - p + 1$ iterasi: **benar secara total**, $\Theta(n)$. ∎

MERGE-SORT lalu mengikuti dengan induksi kuat pada panjang irisan. Buku: Subbab 3.15.4.

Rekurensinya, dengan Tiga Cara

$$T(n) = \begin{cases} \Theta(1) & \text{jika } n = 1, \\ 2\,T(n/2) + \Theta(n) & \text{jika } n > 1, \end{cases} \tag{3.16}$$

Pohon: setiap tingkat Persamaan (3.16) berbiaya $c\,n$, dan ada $\lg n+1$ tingkat: $c\,n\lg n+c\,n=\Theta(n\lg n)$.

Master: $a=b=2$, $n^{\log_2 2}=n=f(n)$: Kasus 2, $\Theta(n\lg n)$.

Substitusi: $2\,c\frac{n}{2}\lg\frac{n}{2}+n=c\,n\lg n-c\,n+n\le c\,n\lg n$ bila $c\ge 1$.

Tiga metode, satu jawaban — dan penyelesaian persisnya dengan $T(1)=1$ adalah $n\lg n+n$. Buku: Subbab 3.15.5 dan Persamaan (3.16).

Merge Sort dalam C++

Kode 3.2 Merge Sort dalam C++, dengan penelusuran tercetak dan cacah pembandingan terukur.

```cpp
// mergesort.cpp - Merge Sort selengkapnya: langkah penggabungan, pengurutan rekursif,
// penelusuran tercetak atas fase bagi dan fase padu, serta cacah
// pembandingan yang sungguh dilakukan, terhadap ramalan n lg n.
// Kompilasi: g++ -std=c++17 -O2 -o mergesort mergesort.cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include <random>
#include <cmath>
using namespace std;

long long comparisons = 0;   // tiap pembandingan kunci-ke-kunci yang dilakukan penggabungan

static string show(const vector<int>& a, int p, int r) {
    string s = "[";
    for (int i = p; i <= r; ++i) { s += to_string(a[i]); if (i < r) s += ", "; }
    return s + "]";
}

// MERGE: A[p..q] dan A[q+1..r] masing-masing terurut; sesudahnya A[p..r] terurut.
// Invarian loop: di puncak tiap iterasi, A[p..k-1] memuat k-p
// elemen terkecil dari L dan R dalam keadaan terurut, dan L[i] serta R[j] adalah
// elemen terkecil array masing-masing yang belum disalin kembali.
void merge(vector<int>& A, int p, int q, int r) {
    vector<int> L(A.begin() + p, A.begin() + q + 1);
    vector<int> R(A.begin() + q + 1, A.begin() + r + 1);
    size_t i = 0, j = 0;
```

```
    int k = p;
    while (i < L.size() && j < R.size()) {
        ++comparisons;
        if (L[i] <= R[j]) A[k++] = L[i++];   // <= menjaga pengurutannya tetap stabil
        else              A[k++] = R[j++];
    }
    while (i < L.size()) A[k++] = L[i++];
    while (j < R.size()) A[k++] = R[j++];
}
```

Kode 3.2 (Lanjutan) *Merge Sort dalam C++, dengan penelusuran tercetak dan cacah pembandingan terukur.*

```
void mergeSort(vector<int>& A, int p, int r, int depth, bool trace) {
    if (p >= r) return;                          // satu elemen: sudah terurut
    int q = p + (r - p) / 2;                     // floor((p+r)/2) tanpa limpahan
    if (trace) cout << string(2 * depth, ' ') << "divide " << show(A, p, r)
                    << " -> " << show(A, p, q) << " " << show(A, q + 1, r) << "\n";
    mergeSort(A, p, q, depth + 1, trace);
    mergeSort(A, q + 1, r, depth + 1, trace);
    merge(A, p, q, r);
    if (trace) cout << string(2 * depth, ' ') << "merge  -> " << show(A, p, r) << "\n";
}

int main() {
    vector<int> A = {1, 5, 2, 4, 6, 3, 2, 6};
    cout << "input  " << show(A, 0, (int)A.size() - 1) << "\n\n";
    mergeSort(A, 0, (int)A.size() - 1, 0, true);
    cout << "\nsorted " << show(A, 0, (int)A.size() - 1) << "\n";
    cout << "comparisons for n = 8: " << comparisons << "\n\n";

    cout << "n          comparisons      n lg n    ratio\n";
    cout << "-------------------------------------------\n";
    mt19937 rng(12345);
    for (int n : {16, 64, 256, 1024, 4096, 65536}) {
        vector<int> B(n);
        for (int& x : B) x = (int)(rng() % 1000000);
        comparisons = 0;
        mergeSort(B, 0, n - 1, 0, false);
        double nlgn = n * log2((double)n);
        cout << left << setw(11) << n << setw(17) << comparisons
```

```
            << setw(12) << fixed << setprecision(0) << nlgn
            << setprecision(3) << comparisons / nlgn << "\n";
    }
    return 0;
}
```

Kode 3.2 menggabung dengan `<=` agar tetap stabil dan menghitung `q = p + (r - p) / 2` untuk menghindari limpahan. Buku: Subbab 3.15.6 dan Kode 3.8.

Yang Dicetak Program Itu

```
input  [1, 5, 2, 4, 6, 3, 2, 6]

divide [1, 5, 2, 4, 6, 3, 2, 6] -> [1, 5, 2, 4] [6, 3, 2, 6]
  divide [1, 5, 2, 4] -> [1, 5] [2, 4]
    divide [1, 5] -> [1] [5]
    merge  -> [1, 5]
    divide [2, 4] -> [2] [4]
    merge  -> [2, 4]
  merge  -> [1, 2, 4, 5]
  divide [6, 3, 2, 6] -> [6, 3] [2, 6]
    divide [6, 3] -> [6] [3]
    merge  -> [3, 6]
    divide [2, 6] -> [2] [6]
    merge  -> [2, 6]
  merge  -> [2, 3, 6, 6]
merge  -> [1, 2, 2, 3, 4, 5, 6, 6]

sorted [1, 2, 2, 3, 4, 5, 6, 6]
comparisons for n = 8: 16

n          comparisons      n lg n    ratio
-------------------------------------------
16         47               64          0.734
64         307              384         0.799
256        1725             2048        0.842
1024       8962             10240       0.875
4096       43970            49152       0.895
65536      965956           1048576     0.921
```

Rasionya terhadap $n \lg n$ *bertahan antara* 0.73 *dan* 0.93 *ketika* n *membesar empat ribu kali —* $\Theta(n \lg n)$ *yang terukur. Merge Sort* $\Theta(n \lg n)$ *pada setiap kasus dan stabil, tetapi memerlukan memori tambahan* $\Theta(n)$*. Buku: Subbab 3.15.6–3.15.7.*

Profitina dalam Praktik

Pemindaian adalah rekurensi: pecah penjelajahan situs menjadi a serpihan berukuran n/b, dan kasus Master menentukan apakah serpihan tambahan membantu (Kasus 1) atau penguraian halamanlah hambatannya (Kasus 3).

Pengurutan harus $n \lg n$, dan stabil: tren, deduplikasi, dan peringkat semuanya mengurutkan; skor yang seri harus tetap dalam urutan yang dapat diulang — `<=` pada Merge Sort.

Profitina Laundry: urutkan dua belas keranjang menurut waktu ambil dan hitung **inversi** dalam penggabungan yang sama — setiap kali paruh kanan menang, tambahkan $|L| - i$. Hasilnya: **32** pasangan tidak berurutan dari 66 yang mungkin, dalam $\Theta(n \lg n)$.

Aplikasi memberi saran; pemilik merek yang memutuskan; data pribadi tunduk pada UU PDP 27/2022. Program penghitung inversi selengkapnya adalah Kode 3.9. Buku: Subbab 3.16.

Kesalahan yang Sering Terjadi

O **dipakai untuk** Θ, dan membaca $=$ secara terbalik. Nyatakan batas paling ketat; baca $=$ sebagai "adalah anggota dari".

Sisa positif dalam substitusi, dan **kasus basis yang terlupa**. Hasilkan kembali hipotesisnya persis; perkuat dengan mengurangkan; geser basisnya ke atas.

"Lebih besar" dikira "lebih besar secara polinomial", dan **melewatkan keteraturan**. Bandingkan pangkat; faktor logaritma berarti celah.

Teorema Master pada bentuk yang keliru — bagian tidak sama, $T(n-1)$, a yang tidak konstan.

$\lg^2 n$ **dikira** $\lg \lg n$ (256 melawan 4 pada $n = 65536$); **mencemaskan basis logaritma di dalam** Θ; **menganggap semua fungsi dapat dibandingkan**; **memercayai asimtotik pada masukan mungil**.

Buku: Subbab 3.17.

Ringkasan — Lima Hal untuk Diingat

1. Kebenaran total = kebenaran parsial + terminasi; invarian adalah induksi pada gelung.

2. O, Ω, Θ, o, ω adalah klaim tentang konstanta yang dapat Anda sebutkan: tunjukkan untuk membuktikan, gugurkan setiap pilihan untuk menyangkal.

3. Aritmetika $\Theta(n^2)$; geometri didominasi suku pertama, terakhir, atau semuanya sama rata; harmonik $\Theta(\lg n)$.

4. Pohon untuk menebak, substitusi untuk membuktikan, Teorema Master bila bentuknya cocok: bandingkan $f(n)$ dengan $n^{\log_b a}$, dan kenali celah-celahnya.

5. Merge Sort: $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)$ pada setiap kasus; benar menurut invarian; stabil; tidak *in place*.

Tabel 3.9 dan 3.10 di buku meringkas seluruh bab, dan setiap rekurensi sejauh ini, masing-masing dalam satu halaman. Buku: Subbab 3.18.

Pertemuan Berikutnya

Bab 4 — Evaluasi dan Pembahasan I, atas Bab 1–3.

Dua soal SPOJ, LDCSUMAB dan TOHU, dibawa dari rumusan soal sampai rumus yang terbukti dan programnya; lalu dua belas soal terbahas tentang invarian, pencacahan, pertumbuhan, dan rekurensi.

Sebelum kita bertemu: Soal Latihan 3.1, 3.4, 3.8, 3.13, dan 3.16 — membuktikan, mengurutkan, menerapkan teorema, mengenali yang bukan celah, memperkuat tebakan.

Lalu cobalah kedua soal SPOJ itu sendiri sebelum membaca penyelesaiannya.

Buku: Subbab 3.19–3.20, lalu Bab 4.

Perancangan dan Analisis Algoritma (EF234405)

Kuliah 4 — Evaluasi dan Pembahasan I: Fondasi, Analisis, dan Problem Pertama

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*, Bab 4

Dosen: Irfan Subakti — `yifana@gmail.com`

Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya

Hari ini tidak ada teori baru: empat belas persoalan atas Bab 1–3, masing-masing diselesaikan dari nol dengan setiap angka dituliskan. Buku: Bab 4.

Agenda

1. **Metode tujuh langkah** untuk membaca persoalan algoritma apa pun.
2. **LDCSUMAB** (SPOJ): membalik peringkat jumlah 2^m+2^n — sebuah rumus, sebuah bukti, tiga program.
3. **TOHU** (SPOJ): barisan tersembunyi yang dipulihkan dengan menyelisihkan dan dijumlahkan secara teleskopik.
4. Persoalan Terkerjakan **A–L**: invarian dan penelusuran; pencacahan dan pertumbuhan; rekurensi; bentuk tertutup dan Fibonacci yang tersembunyi; *brute force* melawan urutkan-lalu-cari, diukur waktunya.
5. Keempat belas persoalan dibandingkan, Profitina, kesalahan yang sering terjadi, ringkasan.

Sebelum setiap penyelesaian, tutuplah dan cobalah sendiri sepuluh menit; varian uji diri di buku menguji apakah Anda sungguh memahaminya. Buku: Subbab 4.1–4.13.

Posisi Kita Saat Ini

Bab 1: algoritma, kebenaran dan efisiensi, lima paradigma, lima fase *Memahami, Mengabstraksi, Merancang, Menganalisis, Mengimplementasikan*.

Bab 2: model RAM; invarian gelung (inisialisasi, pemeliharaan, terminasi); Insertion Sort dalam tiga kasus; *linear search* dan pencarian biner.

Bab 3: O, Ω, Θ, o, ω dengan konstanta eksplisit; fungsi-fungsi baku; jumlahan; substitusi, pohon rekursi, dan Teorema Master.

> **CARA MEMAKAI BAB INI.** Baca pernyataannya, tutup penyelesaiannya, coba sendiri sepuluh menit. Penyelesaian yang pernah Anda gumuli dipahami dengan cara yang tidak pernah dicapai penyelesaian yang hanya Anda baca.

Buku: Subbab 4.1.

Tujuh Langkah: Memahami

Ketujuh langkah adalah kelima fase yang dibuat konkret: Langkah 1–2 *Memahami*, 3–5 *Mengabstraksi*, 6 *Merancang* beserta kebenarannya, 7 *Mengimplementasikan* setelah biaya diperiksa.

Langkah 1 — salin formatnya apa adanya: format masukan, format keluaran (baris, ketelitian, susunan kata), setiap batasan, contoh masukan dan keluaran karakter demi karakter.

Banyak vonis *Wrong Answer* adalah kesalahan format: TOHU meminta tepat sebelas angka di belakang koma; cetak sepuluh, maka algoritma yang sempurna pun dinyatakan salah.

Langkah 2 — namai yang diketahui dan yang ditanya, masing-masing dengan lambang. LDCSUMAB: *diketahui* $k \le 10^9$, sampai $T \le 10$ kali; *ditanya* $m<n$ dengan 2^m+2^n bilangan ke-k terkecil yang berbentuk demikian.

Tanpa lambang Anda tidak dapat beraljabar, dan aljabarlah yang menyelesaikan sebagian besar bab ini. Buku: Subbab 4.2–4.2.2.

Langkah 3 — Batasan Menjadi Anggaran Waktu

Mesin *judge* mengerjakan sekitar 10^8–10^9 operasi sederhana per detik, dan batas waktunya satu sampai tiga detik: **program Anda boleh mengerjakan sekitar 10^8 operasi seluruhnya.**

Tabel 4.1 Anggaran kompleksitas. Bacalah batas pada n, bacalah kompleksitas yang terjangkau, lalu tolak setiap gagasan yang melampauinya sebelum Anda menulis satu baris kode pun.

Batas pada n	Yang terjangkau per kasus uji	Bentuk yang lazim
$n \le 10$	$O(n!), O(2^n \cdot n)$	gaya kasar, permutasi
$n \le 20$	$O(2^n)$	pencacahan himpunan bagian, DP bitmask
$n \le 500$	$O(n^3)$	Floyd–Warshall, rantai matriks
$n \le 5000$	$O(n^2)$	DP berpasangan, tabel DP sederhana
$n \le 10^6$	$O(n)$ atau $O(n \log n)$	satu lintasan, pengurutan, saringan, tumpukan
$n \le 10^{12}$	$O(\sqrt{n})$	blok pembagi, pembagian coba-coba
$n \le 10^{18}$	$O(\log n)$	pencarian biner, perpangkatan cepat
n tak terbatas	$O(1)$	sebuah bentuk tertutup

Bila soal menyebutkan kompleksitasnya — TOHU menulis $O(1)$ — ia sedang memberi tahu bentuk jawabannya: sebuah bentuk tertutup.

Tabel 4.1 adalah model RAM yang dijadikan alat perencanaan: tolak setiap gagasan yang melampaui anggaran sebelum menulis satu baris pun. Buku: Subbab 4.2.3 dan Tabel 4.1.

Langkah 4 sampai 7

Langkah 4 — hitung lima kasus pertama dengan tangan. Suku-suku TOHU $2/3, 1/24, 1/60, 1/120$; nilai n pada LDCSUMAB $1, 2, 2, 3, 3, 3, 4, 4, 4, 4$.

Langkah 5 — sebutkan bentuknya: bentuk tertutup dari sebuah jumlah; membalik rumus pencacahan; gelung dengan invarian; menghitung langkah dengan jumlahan; rekurensi; *brute force*, lalu yang lebih baik.

Langkah 6 — tuliskan invarian, rekurensi, atau rumusnya, lalu buktikan. Pola yang dibaca dari tabel masih dugaan sampai saat itu.

Langkah 7 — baru sekarang, programnya. Periksa limpahan (`int` $\approx 2.1 \times 10^9$, `long long` $\approx 9.22 \times 10^{18}$), ketelitian (`double` 15–16 angka, `float` 7), dan kecepatan baca (`scanf`/`printf`).

> **WAWASAN KUNCI: TABEL ITULAH ALGORITMANYA.** Sebagian besar penyelesaian di sini terlihat pada tabel lima sampai sepuluh kasus yang dihitung dengan tangan. Tidak ada pola? Carilah invarian, urutan, atau argumen pertukaran.

Buku: Subbab 4.2.4–4.2.7.

Persoalan 1 — LDCSUMAB

LDCSUMAB — BRVT - Math Team Collection 2013 (SPOJ)

Kumpulan soal tutorial, 15861

Pernyataan

Tinjau barisan semua bilangan bulat yang dapat ditulis sebagai $2^m + 2^n$ dengan bilangan bulat $0 \leq m < n$, terurut naik. Diberikan bilangan bulat positif k, tentukan m dan n untuk suku ke-k barisan itu.

Masukan

Baris pertama berisi bilangan bulat T ($T \leq 10$), banyaknya kasus uji. Masing-masing dari T baris berikutnya berisi satu bilangan bulat k ($k \leq 10^9$).

Keluaran

Untuk setiap kasus uji, cetak kedua bilangan m dan n ($m < n$) dari suku ke-k.

Contoh

Masukan	Keluaran
3 1 3 5	0 1 1 2 1 3

Gambar 4.1 Himpunan persoalan SPOJ (*tutorial*) 15861, *LDCSUMAB — BRVT - Math Team Collection 2013*, dirumuskan ulang dari *judge*-nya: pernyataan, format masukan dan keluaran, serta uji contohnya.

Gambar 4.1: persoalan tutorial SPOJ yang hanya memakai tiga gagasan — deret aritmetika, fungsi pembulatan ke atas, dan pencarian biner. Buku: Subbab 4.3 dan Gambar 4.1.

LDCSUMAB — Persoalannya dan Anggarannya

> **PERSOALAN.** Daftarkan semua bilangan bulat $2^m + 2^n$ dengan $0 \le m < n$ secara menaik: $3, 5, 6, 9, 10, 12, 17, \ldots$ Diberikan posisi k ($1 \le k \le 10^9$, paling banyak 10 pertanyaan), cetak `m n` untuk bilangan ke-k. Contoh `3 / 1 / 3 / 5` memberi `0 1 / 1 2 / 1 3`.

Urutan penting: m dahulu, lalu n — terbalik berarti *Wrong Answer* sejak baris pertama.

Taksir biaya gagasannya. Bangun daftarnya: $\Theta(k)$, 10^{10} untuk sepuluh pertanyaan — ditolak. Susuri bloknya: sekitar $\sqrt{2k} \approx 44,721$ langkah — muat. Pencarian biner: sekitar 16 langkah. Bentuk tertutup: $O(1)$.

Anggaran memberi tahu gagasan mana yang terlarang, bukan mana yang terbaik; kita menurunkan rumusnya karena itulah yang paling banyak mengajarkan. Buku: Subbab 4.3.1–4.3.2.

LDCSUMAB — Sepuluh Pertama dengan Tangan

Tabel 4.2 Sepuluh bilangan pertama berbentuk $2^m + 2^n$, dihitung dengan tangan. Kolom n berbunyi satu angka 1, dua angka 2, tiga angka 3, empat angka 4; di dalam tiap blok, m menghitung $0, 1, 2, \ldots$.

k	Nilai	m	n	Posisi di dalam blok n
1	3	0	1	ke-1 dari 1
2	5	0	2	ke-1 dari 2
3	6	1	2	ke-2 dari 2
4	9	0	3	ke-1 dari 3
5	10	1	3	ke-2 dari 3
6	12	2	3	ke-3 dari 3
7	17	0	4	ke-1 dari 4
8	18	1	4	ke-2 dari 4
9	20	2	4	ke-3 dari 4
10	24	3	4	ke-4 dari 4

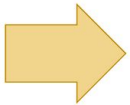
Calon dengan $n \le 4$ ada sepuluh; yang terkecil dengan $n = 5$ adalah 33, lebih besar daripada semuanya — jadi inilah tepat sepuluh yang pertama.

Bacalah Tabel 4.2 per kolom: blok n memuat n entri; di dalamnya $m = 0, 1, \ldots, n-1$; blok-blok berakhir di $1, 3, 6, 10$ — bilangan segitiga.

Buku: Subbab 4.3.3 dan Tabel 4.2.

Pengamatan yang Sama, dalam Gambar

k	m	n	$2^m + 2^n$
1	0	1	3
2	0	2	5
3	1	2	6
4	0	3	9
5	1	3	10
6	2	3	12
7	0	4	17
8	1	4	18
9	2	4	20
10	3	4	24



n	k
1	**1**
2	**2**
2	**3**
3	**4**
3	**5**
3	**6**
4	**7**
4	**8**
4	**9**
4	**10**

$$k = \frac{n(n+1)}{2}$$

$$n^2 + n - 2k = 0$$

$$n = \frac{-1 + \sqrt{1 + 8k}}{2}$$

Gambar 4.2 Sepuluh suku pertama barisan LDCSUMAB (kiri), dirangkum menjadi peta dari k ke n (tengah), dan pembalikannya $k = \frac{n(n+1)}{2}$, $n^2 + n - 2k = 0$, $n = \frac{-1 + \sqrt{1+8k}}{2}$ (kanan).

Gambar 4.2: daftarnya, pemetaan dari k ke n, dan pembalikan yang akan kita turunkan. Buku: Subbab 4.3.3 dan Gambar 4.2.

Urutannya Dibuktikan, Bloknya Dihitung

Klaim 1 — blok tidak pernah berselang-seling. Nilai terbesar blok n adalah $2^{n-1}+2^n=3\cdot 2^{n-1}$; yang terkecil di blok $n+1$ adalah $1+4\cdot 2^{n-1}$, yang lebih besar.

Klaim 2 — di dalam blok, nilainya naik bersama m. Jadi daftarnya adalah pasangan-pasangan dalam urutan leksikografis, n dahulu, lalu m.

Blok 1 sampai n memuat, menurut deret aritmetika Subbab 3.9.1, Persamaan (4.1) — bilangan segitiga T_n, dengan $T_0=0$ dan $T_n=T_{n-1}+n$:

$$1+2+3+\cdots+n=\frac{n(n+1)}{2} \tag{4.1}$$

Posisi k berada di blok n tepat ketika $T_{n-1}<k\le T_n$: **n adalah bilangan bulat terkecil dengan $T_n\ge k$.**

Tabel sepuluh baris adalah bukti petunjuk; kedua klaim menjadikannya berlaku selamanya. Keluarga ini disebut ranking dan unranking. Buku: Subbab 4.3.4 dan Persamaan (4.1).

Menyelesaikan untuk n dan m

$\frac{n(n+1)}{2} = k \Rightarrow n^2 + n - 2k = 0 \Rightarrow n = \frac{-1 \pm \sqrt{1+8k}}{2}$; ambil tanda plus. Bilangan bulat terkecil yang tidak kurang dari akar itu adalah Persamaan (4.2):

$$n = \left\lceil \frac{-1+\sqrt{1+8k}}{2} \right\rceil \tag{4.2}$$

Pembulatan ke atas, bukan ke bawah: pada $k = 5$ akarnya sekitar 2.70 dan posisi 5 berada di blok 3; pada k segitiga akarnya sudah bulat.

Di dalam blok n, posisi k membawa $m = k - T_{n-1} - 1$; dengan $r = T_n$ itu adalah Persamaan (4.3):

$$m = k - (r - n) - 1 = n + k - (r + 1) \tag{4.3}$$

Buku: Subbab 4.3.4 dan Persamaan (4.2)–(4.3).

Mengerjakan Kasusnya dengan Tangan

$k=1$: $\sqrt{9}=3$, $n=1$, $r=1$, $m=1+1-2=0$ → `0 1`. $k=3$: $\sqrt{25}=5$, $n=2$, $r=3$, $m=1$ → `1 2`.

$k=5$: $\sqrt{41}\approx 6.4031$, akar ≈ 2.7016, $n=3$, $r=6$, $m=3+5-7=1$ → `1 3`, yaitu nilai 10.

$k=6$ **(batas blok):** $\sqrt{49}=7$, $n=3$, $m=2$ → `2 3`. $k=7$ **(blok baru):** akar ≈ 3.2749, $n=4$, $r=10$, $m=0$ → `0 4`.

$k=10^9$: $\sqrt{8,000,000,001}\approx 89,442.719$, jadi $n=44,721$; periksa $T_{44720}=999,961,560<10^9\le T_{44721}=1,000,006,281$; $m=38,439$ → `38439 44721`.

Buku: Subbab 4.3.4.

Rumusnya Benar untuk Setiap k

TEOREMA. Untuk setiap bilangan bulat $k \geq 1$, misalkan n bilangan bulat terkecil dengan $T_n \geq k$ dan $m = k - T_{n-1} - 1$. Maka $0 \leq m < n$, dan $2^m + 2^n$ adalah bilangan bulat ke-k terkecil yang berbentuk $2^a + 2^b$ dengan $0 \leq a < b$.

Menurut Klaim 1–2, pasangan dengan eksponen paling besar $n-1$ mengisi T_{n-1} posisi pertama, yang paling besar n mengisi T_n posisi pertama.

Menurut pilihan n, $T_{n-1} < k \leq T_n$: eksponen di posisi k tepat n, dan di dalam bloknya posisi itu membawa $m = k - T_{n-1} - 1$.

$T_{n-1} < k$ memberi $m \geq 0$; $k \leq T_n = T_{n-1} + n$ memberi $m \leq n - 1$. ∎

Buku: Subbab 4.3.5.

Menjadikan Akar Kuadratnya Aman

Pseudokode 4.1 Menjawab satu pertanyaan LDCSUMAB: terkaan pertama titik-mengambang untuk bloknya, dua gelung pengoreksi, dan posisi di dalam blok.

```
LDCSUMAB-QUERY(k)
1  n = floor((-1 + sqrt(1 + 8k)) / 2)     // terkaan pertama, dari rumus kuadrat
2  while n(n+1)/2 < k                      // terkaan terlalu kecil: naik
3      n = n + 1
4  while n >= 1 and (n-1)n/2 >= k          // terkaan terlalu besar: turun
5      n = n - 1
6  r = n(n+1)/2                            // posisi terakhir blok n
7  m = n + k - (r + 1)
8  return (m, n)
```

Setelah baris 2–3: $T_n \geq k$ (gelungnya hanya berhenti ketika ujinya gagal, dan T_n tumbuh tanpa batas).

Baris 4–5 menjaga invarian $T_n \geq k$**:** badannya hanya berjalan bila $T_{n-1} \geq k$. Saat keluar $T_{n-1} < k \leq T_n$ ($n = 0$ mustahil, karena $T_0 = 0 < k$).

Baris 6–7 menghitung Persamaan (4.3), yang dibuktikan benar oleh Teorema.

Pseudokode 4.1 memakai akar floating point hanya sebagai tebakan awal; dua gelung kecil memakukan bilangan bulat yang tepat. Buku: Subbab 4.3.5 dan Pseudokode 4.1.

LDCSUMAB — Biaya dan Limpahan

Pada batas blok $k = T_n$, $1 + 8k = (2n+1)^2$ — kuadrat ganjil yang tepat. `double` menyimpan bilangan bulat sampai 2^{53} dengan tepat dan `sqrt` dibulatkan dengan benar, jadi untuk $k \le 10^9$ tebakan awalnya sudah tepat dan **kedua gelung tidak berjalan**.

Mengapa tetap dipertahankan? Jika batasnya menjadi $k \le 10^{18}$, $1 + 8k$ melampaui 2^{53} dan tebakannya bisa meleset satu; kedua gelung itu menghapus ketergantungan itu tanpa biaya.

Biaya: $O(1)$ per pertanyaan, $\Theta(T)$ seluruhnya, ruang $O(1)$. **Limpahan:** $n(n+1) \approx 2.0 \times 10^9$ tinggal sedikit di bawah batas `int` — pakailah `long long`.

Bab ini juga menampilkan penyelesaian ringkas dengan `ceil` di samping tabelnya (Gambar 4.3). Buku: Subbab 4.3.6.

LDCSUMAB — Programnya

Kode 4.1 LDCSUMAB — membalikkan peringkat barisan 2^m+2^n, dalam $O(1)$ per pertanyaan sesudah akar kuadratnya.

```cpp
#include <cstdio>
#include <cmath>
using namespace std;
typedef long long LL;

int main() {
    int T;
    if (scanf("%d", &T) != 1) return 0;
    while (T--) {
        LL k;
        if (scanf("%lld", &k) != 1) break;
        // n terkecil dengan n(n+1)/2 >= k, dicari lewat rumus kuadrat lalu dikoreksi
        LL n = (LL)floor((-1.0 + sqrt(1.0 + 8.0 * (double)k)) / 2.0);
        while (n * (n + 1) / 2 < k) ++n;          // koreksi ke atas  (menjaga galat sqrt)
        while (n >= 1 && (n - 1) * n / 2 >= k) --n;  // koreksi ke bawah
        LL r = n * (n + 1) / 2;                   // berapa pasangan yang memakai eksponen <= n
        LL m = n + k - (r + 1);
        printf("%lld %lld\n", m, n);
    }
    return 0;
}
```

```
0 1
1 2
1 3
2 3
0 4
998 999
0 1000
38439 44721
```

Kode 4.1 pada contoh uji, lalu pada $6, 7, 499500, 499501, 10^9$*:* $499,500 = T_{999}$ *menutup blok* 999*; posisi berikutnya membuka blok* 1000*. Kompilasi dengan* `g++ -std=c++17 -O2 -Wall`*, jalankan dengan* `./ldcsumab < sample.txt`*. Buku: Subbab 4.3.7 dan Kode 4.1.*

Jalan Kedua: Pencarian Biner atas Nomor Blok

"Apakah $T_n \geq k$?" dijawab *tidak* untuk n kecil dan *ya* mulai suatu titik: pertanyaan monoton, dijawab dengan pencarian biner. $T_{50000} = 1,250,025,000 \geq 10^9$, jadi carilah di $[1, 50000]$.

Invarian: jawabannya ada di $[lo, hi]$. Jika $T_{mid} \geq k$ jadikan $hi = mid$, jika tidak $lo = mid + 1$; berhenti ketika $lo = hi$.

> **DENGAN TANGAN — $k = 5$ di $[1, 8]$.** $mid = 4$: $T_4 = 10 \geq 5$, $hi = 4$. $mid = 2$: $T_2 = 3 < 5$, $lo = 3$. $mid = 3$: $T_3 = 6 \geq 5$, $hi = 3$. Kini $lo = hi = 3$, dan $m = 5 - T_2 - 1 = 1 \rightarrow$ 1 3.

Sama sekali tanpa floating point, dan argumen kebenarannya cukup satu invarian. Buku: Subbab 4.3.8.

Program Pencarian Biner

Kode 4.2 LDCSUMAB dengan pencarian biner atas nomor blok, $\Theta(\lg 50000) \approx 16$ langkah per pertanyaan, tanpa titik-mengambang sama sekali.

```cpp
#include <cstdio>
using namespace std;
typedef long long LL;

// n >= 1 terkecil dengan n(n+1)/2 >= k, lewat pencarian biner atas jawabannya.
// Invarian: jawabannya terletak di [lo, hi].
LL blockOf(LL k) {
    LL lo = 1, hi = 50000;                  // 50000*50001/2 > 10^9, jadi hi cukup besar
    while (lo < hi) {
        LL mid = lo + (hi - lo) / 2;        // tak pernah melimpah, tidak seperti (lo + hi) / 2
        if (mid * (mid + 1) / 2 >= k) hi = mid;   // mid cukup besar:  jawaban <= mid
        else lo = mid + 1;                         // mid terlalu kecil: jawaban >  mid
    }
    return lo;
}

int main() {
    int T;
    if (scanf("%d", &T) != 1) return 0;
    while (T--) {
        LL k;
        if (scanf("%lld", &k) != 1) break;
        LL n = blockOf(k);
        LL m = k - (n - 1) * n / 2 - 1;     // posisi di dalam blok n, dihitung dari 0
        printf("%lld %lld\n", m, n);
    }
    return 0;
}
```

Kode 4.2 mencetak baris-baris yang sama dengan Kode 4.1 pada setiap masukan yang dicoba; biayanya sekitar 16 *iterasi per pertanyaan. Perhatikan titik tengah yang aman dari limpahan (Subbab 2.14.6). Buku: Subbab 4.3.8 dan Kode 4.2.*

Memeriksa terhadap *Brute Force*

Kode 4.3 Pemeriksaan silang gaya kasar untuk LDCSUMAB: daftarnya dibangun dan diurutkan secara eksplisit, lalu dibandingkan dengan rumus pada kedelapan ratus dua puluh posisi.

```cpp
#include <cstdio>
#include <cmath>
#include <vector>
#include <tuple>
#include <algorithm>
using namespace std;
typedef long long LL;

LL formulaN(LL k) {                          // metode bentuk tertutup
    LL n = (LL)floor((-1.0 + sqrt(1.0 + 8.0 * (double)k)) / 2.0);
    while (n * (n + 1) / 2 < k) ++n;
    while (n >= 1 && (n - 1) * n / 2 >= k) --n;
    return n;
}

int main() {
    // Gaya kasar: bangun setiap nilai 2^m + 2^n dengan 0 <= m < n <= 40 lalu urutkan menurut nilai.
    vector<tuple<LL, int, int>> seq;
    for (int n = 1; n <= 40; ++n)
        for (int m = 0; m < n; ++m)
            seq.emplace_back((1LL << m) + (1LL << n), m, n);
    sort(seq.begin(), seq.end());
    printf(" k  value  m  n\n");
    for (int k = 1; k <= 10; ++k)
        printf("%2d  %5lld  %d  %d\n", k, get<0>(seq[k - 1]), get<1>(seq[k - 1]), get<2>(seq[k - 1]));
    int bad = 0;
    for (LL k = 1; k <= 820; ++k) {          // 820 = 40*41/2 suku, semuanya dengan n <= 40
        LL n = formulaN(k), m = n + k - (n * (n + 1) / 2 + 1);
```

```
        if (m != get<1>(seq[k - 1]) || n != get<2>(seq[k - 1])) ++bad;
    }
    printf("terms compared: 820, mismatches: %d\n", bad);
    return 0;
}
```

```
 k  value  m  n
 1      3  0  1
 2      5  0  2
 3      6  1  2
 4      9  0  3
 5     10  1  3
 6     12  2  3
 7     17  0  4
 8     18  1  4
 9     20  2  4
10     24  3  4
terms compared: 820, mismatches: 0
```

Kode 4.3 membangun semua $820 = T_{40}$ *nilai dengan* $n \le 40$*, mengurutkannya, lalu membandingkannya dengan rumus: tidak ada ketidakcocokan, dan sepuluh baris pertamanya adalah Tabel 4.2. Buku: Subbab 4.3.9 dan Kode 4.3.*

Persoalan 2 — TOHU: Persoalannya dan Anggarannya

Sebuah barisan $a_1, a_2, \ldots$ hanya diketahui lewat jumlah berbobotnya, Persamaan (4.4):

$$a_1 + 2\,a_2 + 3\,a_3 + \cdots + k\,a_k = \frac{k+1}{k+2} \text{ untuk setiap } k \geq 1 \tag{4.4}$$

> **PERSOALAN.** Untuk setiap pertanyaan n, paling banyak $20{,}000$ pertanyaan, cetak $S_n = a_1 + \cdots + a_n$ dengan sebelas angka di belakang koma. Contoh `2 / 2 / 5` memberi `0.70833333333 / 0.73809523810`. Soalnya mencatat: *kompleksitas waktu* $O(1)$.

Dua yang tidak diketahui: suku-suku tersembunyi a_k, lalu jumlah biasanya. **Anggaran:** menjumlahkan n suku per pertanyaan berbiaya $\Theta(T\,n)$, sampai 2×10^{13} — maka pasti ada bentuk tertutupnya. Sebelas angka menyingkirkan `float`.

TOHU melatih perkakas penjumlahan agung lain dari Bab 3: jumlah teleskopik. Buku: Subbab 4.4.1–4.4.2 dan Persamaan (4.4).

TOHU — Lima Pertama dengan Tangan

Dengan $W_k = \dfrac{k+1}{k+2}$ dan $W_0 = 0$, tiap langkah menambah satu suku baru: $k\,a_k = W_k - W_{k-1}$.

Tabel 4.3 Lima suku pertama barisan TOHU, dipulihkan dengan tangan dari jumlah-jumlah berbobotnya. Mulai $k=2$, a_k adalah kebalikan hasil kali tiga bilangan bulat berurutan; a_1 mematahkan pola itu.

k	$W_k=(k+1)/(k+2)$	$k\,a_k=W_k-W_{k-1}$	a_k	a_k Difaktorkan	S_k	S_k Sebagai desimal
1	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{2}{3}$	—	$\frac{2}{3}$	0.66666666667
2	$\frac{3}{4}$	$\frac{1}{12}$	$\frac{1}{24}$	$\frac{1}{2\cdot3\cdot4}$	$\frac{17}{24}$	0.70833333333
3	$\frac{4}{5}$	$\frac{1}{20}$	$\frac{1}{60}$	$\frac{1}{3\cdot4\cdot5}$	$\frac{29}{40}$	0.72500000000
4	$\frac{5}{6}$	$\frac{1}{30}$	$\frac{1}{120}$	$\frac{1}{4\cdot5\cdot6}$	$\frac{11}{15}$	0.73333333333
5	$\frac{6}{7}$	$\frac{1}{42}$	$\frac{1}{210}$	$\frac{1}{5\cdot6\cdot7}$	$\frac{31}{42}$	0.73809523810

Baris S_2 dan S_5 adalah keluaran contohnya. Mulai $k=2$, $a_k=\dfrac{1}{k(k+1)(k+2)}$ — tetapi $a_1=\dfrac{2}{3}$, bukan $\dfrac{1}{6}$. **Tabel itu juga memperlihatkan di mana polanya patah.**

Tabel 4.3: setiap pengurangan dikerjakan atas penyebut bersama, misalnya $2\,a_2=\dfrac{9}{12}-\dfrac{8}{12}$. *Buku: Subbab 4.4.3 dan Tabel 4.3.*

Memulihkan Suku-Sukunya dengan Menyelisihkan

Tulis Persamaan (4.4) untuk k dan untuk $k-1$ (dengan $k \geq 2$), yang satu di atas yang lain, lalu kurangkan: semuanya saling meniadakan kecuali $k\,a_k = \frac{k+1}{k+2} - \frac{k}{k+1}$.

Atas penyebut $(k+1)(k+2)$: $\frac{(k^2+2k+1)-(k^2+2k)}{(k+1)(k+2)} = \frac{1}{(k+1)(k+2)}$. Bagi dengan k untuk mendapatkan Persamaan (4.5):

$$a_k = \frac{1}{k(k+1)(k+2)} \text{ untuk } k \geq 2, a_1 = \frac{2}{3} \tag{4.5}$$

Mengapa $k=1$ berbeda: rumus jumlah berbobot sebelumnya akan mengklaim $W_0 = 1/2$, padahal jumlah kosong bernilai 0. Jadi $1/6$ dari pola itu kurang $1/2$: $1/6 + 1/2 = 2/3$.

Buku: Subbab 4.4.4 dan Persamaan (4.5).

Menjumlahkan secara Teleskopik

$\frac{1}{k(k+1)} - \frac{1}{(k+1)(k+2)} = \frac{2}{k(k+1)(k+2)}$, jadi setiap suku terbelah menjadi Persamaan (4.6):

$$\frac{1}{k(k+1)(k+2)} = \frac{1}{2}\left[\frac{1}{k(k+1)} - \frac{1}{(k+1)(k+2)}\right] \tag{4.6}$$

Dijumlahkan dari $k=2$ sampai n, setiap suku negatif meniadakan suku positif berikutnya: $\sum_{k=2}^{n} a_k = \frac{1}{12} - \frac{1}{2(n+1)(n+2)}$.

Ditambah $a_1 = \frac{2}{3} = \frac{8}{12}$ menghasilkan Persamaan (4.7):

$$S_n = \frac{3}{4} - \frac{1}{2(n+1)(n+2)} \tag{4.7}$$

Periksa: $n=2$: $\frac{3}{4} - \frac{1}{24} = \frac{17}{24}$ ✓. $n=5$: $\frac{3}{4} - \frac{1}{84} = \frac{31}{42} \approx 0.73809523810$ ✓.

Buku: Subbab 4.4.5 dan Persamaan (4.6)–(4.7).

Jalan Kedua: Pecahan Parsial

Carilah A, B, C dengan $\frac{1}{k(k+1)(k+2)} = \frac{A}{k} + \frac{B}{k+1} + \frac{C}{k+2}$.

Kalikan: $1 = (A+B+C)k^2 + (3A+2B+C)k + 2A$. Samakan koefisiennya: $2A = 1$, $2A+B = 0$, $A+B+C = 0$.

Jadi $A = 1/2$, $B = -1$, $C = 1/2$: $\frac{1}{k(k+1)(k+2)} = \frac{1}{2k} - \frac{1}{k+1} + \frac{1}{2(k+2)}$.

Dikelompokkan sebagai $1/2[1/k - 1/k+1] - 1/2[1/k+1 - 1/k+2]$, ia kembali menjadi Persamaan (4.6): kedua jalan sampai pada Persamaan (4.7).

Jalan pertama lebih pendek; jalan kedua berlaku untuk fungsi rasional apa pun. Buku: Subbab 4.4.5.

TOHU — Bukti dengan Induksi

TEOREMA. Untuk setiap $n \geq 1$, $S_n = \frac{3}{4} - \frac{1}{2(n+1)(n+2)}$.

Basis, $n=1$**:** $S_1 = \frac{2}{3}$, dan $\frac{3}{4} - \frac{1}{12} = \frac{8}{12} = \frac{2}{3}$ ✓ — a_1 yang istimewa itu terserap ke dalam konstanta $\frac{3}{4}$.

Langkah, $n \geq 2$**:** $S_n = S_{n-1} + a_n = \frac{3}{4} - \frac{1}{2n(n+1)} + \frac{1}{n(n+1)(n+2)}$.

Atas $2n(n+1)(n+2)$: $-\frac{(n+2)-2}{2n(n+1)(n+2)} = -\frac{1}{2(n+1)(n+2)}$. ∎

Ketika n membesar, S_n naik mendekati $\frac{3}{4}$ tanpa pernah mencapainya: setiap jawaban terletak di $\left[\frac{2}{3}, \frac{3}{4}\right)$. Buku: Subbab 4.4.6.

TOHU — Biaya, Ketelitian, Limpahan

Waktu: segelintir operasi per pertanyaan — $\Theta(1)$, $\Theta(T)$ seluruhnya, sekitar 10^5 operasi.

Ketelitian: sebelas desimal dari bilangan di bawah 0.75 memerlukan sekitar sebelas angka bermakna; `double` punya lima belas sampai enam belas, `float` tujuh dan gagal pada contoh ujinya.

Limpahan: `2*(n+1)*(n+2)` dalam `int` melampaui $2{,}147{,}483{,}647$ pada $n = 32{,}767$. Ubah setiap faktor ke `double` lebih dulu; pada $n = 10^9$ koreksinya $\approx 5 \times 10^{-19}$ lenyap dan program mencetak `0.75000000000`.

Buku: Subbab 4.4.7.

TOHU — Programnya

Pseudokode 4.2 Algoritma TOHU: satu evaluasi bentuk tertutup Persamaan (4.7) per pertanyaan.

```
TOHU()
1  read T
2  for q = 1 to T
3      read n
4      print 3/4 - 1 / (2(n+1)(n+2)) with 11 decimal places
```

Kode 4.4 TOHU — Help Tohu, dalam $O(1)$ per kasus uji, tanpa kasus khusus untuk $n=1$.

```cpp
#include <cstdio>
using namespace std;

int main() {
    int T;
    if (scanf("%d", &T) != 1) return 0;
    while (T--) {
        long long n;
        if (scanf("%lld", &n) != 1) break;
        // S(n) = 3/4 - 1 / (2 (n+1)(n+2))   -- berlaku untuk setiap n >= 1
        double denom = 2.0 * (double)(n + 1) * (double)(n + 2);
        printf("%.11f\n", 0.75 - 1.0 / denom);
    }
    return 0;
}
```

```
0.66666666667
0.70833333333
0.72500000000
0.73333333333
0.74621212121
0.75000000000
```

Pseudokode 4.2 dan Kode 4.4, pada $n = 1, 2, 3, 4, 10, 10^9$*: empat baris pertama adalah kolom desimal Tabel 4.3; baris kelima adalah* $\frac{197}{264}$*. Contoh ujinya memberi* `0.70833333333` *dan* `0.73809523810`*. Buku: Subbab 4.4.8, Pseudokode 4.2, dan Kode 4.4.*

Program Benar yang Lain

```c
#include <stdio.h>
int main() {
    int t, n;
    double formula;
    scanf("%d", &t);
    while (t--) {
        scanf("%d", &n);
        if (n==1) {
            printf("%.11lf\n",2.0/3.0);
        } else {
            formula = ((double) 1.0/12.0) - 1/ ((double) 2 * (n+1) * (n+2));
            formula = formula + ((double) 2.0/3.0);
            printf("%.11lf\n", formula);
        }
    }
    return 0;
}
```

Gambar 4.3 Sebuah program TOHU alternatif dengan cabang terpisah `if (n==1) printf("%.11lf\n", 2.0/3.0);` dan cabang umum yang menghitung $\frac{1}{12} - \frac{1}{2(n+1)(n+2)} + \frac{2}{3}$.

Gambar 4.3 mengistimewakan $n=1$ *— tidak merugikan tetapi tidak perlu, karena kasus basis induksinya menunjukkan bahwa Persamaan (4.7) sudah memberi* $\frac{2}{3}$*. Cast* `(double) 2 * (n+1) * (n+2)` *menghindari limpahan dengan benar, tetapi rapuh. Buku: Subbab 4.4.8 dan Gambar 4.4.*

Pecahan Eksak, Tanpa Mengandaikan Rumus

Kode 4.5 TOHU dari prinsip pertama: tiap a_k dipulihkan secara eksak dari syarat jumlah berbobot, tanpa mengandaikan rumus apa pun, lalu dibandingkan dengan bentuk tertutupnya.

```cpp
#include <cstdio>
#include <numeric>
using namespace std;
typedef long long LL;

struct Frac {                                   // pecahan eksak p/q, selalu dalam bentuk paling sederhana
    LL p, q;
    Frac(LL a = 0, LL b = 1) { LL g = gcd(a, b); p = a / g; q = b / g; }
};
Frac operator+(Frac a, Frac b) { return Frac(a.p * b.q + b.p * a.q, a.q * b.q); }
Frac operator-(Frac a, Frac b) { return Frac(a.p * b.q - b.p * a.q, a.q * b.q); }

int main() {
    // Pulihkan a_k dari syarat pendefinisinya, secara eksak, tanpa mengandaikan rumus apa pun:
    // k*a_k = (k+1)/(k+2) - (jumlah j*a_j untuk j < k).
    Frac weighted(0, 1), S(0, 1);
    printf(" k   a_k        S_k (exact)     S_k (decimal)   closed form\n");
    for (LL k = 1; k <= 10; ++k) {
        Frac kak = Frac(k + 1, k + 2) - weighted;        // inilah k * a_k
        Frac ak(kak.p, kak.q * k);                       // bagi dengan k
        weighted = weighted + kak;
        S = S + ak;
        double closed = 0.75 - 1.0 / (2.0 * (k + 1) * (k + 2));
        printf("%2lld  %lld/%-6lld  %6lld/%-8lld  %.11f  %.11f\n",
               k, ak.p, ak.q, S.p, S.q, (double)S.p / S.q, closed);
    }
    return 0;
```

```
}
```

```
 k   a_k        S_k (exact)     S_k (decimal)   closed form
 1  2/3            2/3         0.66666666667  0.66666666667
 2  1/24          17/24        0.70833333333  0.70833333333
 3  1/60          29/40        0.72500000000  0.72500000000
 4  1/120         11/15        0.73333333333  0.73333333333
 5  1/210         31/42        0.73809523810  0.73809523810
 6  1/336         83/112       0.74107142857  0.74107142857
 7  1/504        107/144       0.74305555556  0.74305555556
 8  1/720         67/90        0.74444444444  0.74444444444
 9  1/990         41/55        0.74545454545  0.74545454545
10  1/1320       197/264       0.74621212121  0.74621212121
```

Kode 4.5 memulihkan setiap a_k dari Persamaan (4.4) saja, dengan aritmetika eksak memakai `std::gcd`: tidak berguna sebagai penyelesaian ($\Theta(n)$ per pertanyaan), tetapi sangat baik sebagai pemeriksa. Buku: Subbab 4.4.9 dan Kode 4.5.

Kesalahan pada Kedua Soal *Judge*

LDCSUMAB: menganggap blok tidak berselang-seling tanpa bukti; pembulatan ke bawah alih-alih ke atas, atau $>$ alih-alih $\geq$ — ujilah $k = 1, 3, 6, 10$ dan $2, 4, 7, 11$; mencetak `n m`; $n(n+1)$ dalam `int`; membangun daftarnya.

TOHU: menggelung $k = 1..n$ per pertanyaan; menyelisihkan terbalik sehingga mendapat $-k\,a_k$.

TOHU: memercayai polanya pada $k = 1$, yang menghasilkan $\frac{1}{4} - \frac{1}{2(n+1)(n+2)}$ — salah tepat $\frac{1}{2}$.

TOHU: `float` atau `%.10f`; `(n+1)*(n+2)` dalam `int`; menulis `3/4`, yang di C++ bernilai `0`.

Buku: Subbab 4.3.10 dan 4.4.10.

Persoalan Terkerjakan A — Aturan Horner

Pseudokode 4.3 Aturan Horner: mengevaluasi $a_0+a_1x+\cdots+a_nx^n$ mulai dari koefisien tertinggi ke bawah.

```
HORNER(A, n, x)
1  p = 0
2  for i = n downto 0
3      p = A[i] + x * p
4  return p
```

Gagasan: $a_0+a_1x+a_2x^2+a_3x^3=a_0+x(a_1+x(a_2+x\,a_3))$ — satu perkalian per tingkat.

INVARIAN GELUNG (aturan Horner). Pada awal setiap iterasi baris 2–3, dengan indeks i, Persamaan (4.8) berlaku:

$$p=\sum_{k=0}^{n-(i+1)} A[k+i+1]x^k \tag{4.8}$$

Inisialisasi: $i=n$, jumlahnya berjalan sampai -1 — kosong, jadi 0, yaitu nilai p dari baris 1. ✓

Dengan kata-kata: p adalah polinomial berkoefisien $A[i+1],\ldots,A[n]$, yang terendah lebih dulu. Pseudokode 4.3 mengklaim $n+1$ perkalian. Buku: Subbab 4.5.1, Pseudokode 4.3, dan Persamaan (4.8).

Horner — Pemeliharaan, Terminasi, Penelusuran

Pemeliharaan: $p' = A[i] + x\sum_{k} A[k+i+1]x^{k} = \sum_{j=0}^{n-i} A[j+i]x^{j}$ (ambil $j = k+1$); dengan $i' = i-1$ itu kembali Persamaan (4.8). ✓

Terminasi: pada $i = -1$, $p = \sum_{k=0}^{n} a_k x^k = P(x)$. ∎ Biaya $\Theta(n)$ melawan $\frac{(n+1)(n+2)}{2} = \Theta(n^2)$ untuk cara naif.

Tabel 4.4 Aturan Horner pada $2x^3 - 6x^2 + 2x - 1$ di $x = 3$. Sesudah tiap iterasi, p sama dengan polinomial parsial yang diramalkan invariannya; sesudah iterasi terakhir, nilainya sama dengan $P(3) = 5$.

Iterasi i	$A[i]$	p Sebelum	Baris 3: $A[i] + 3p$	Invarian untuk indeks berikutnya, dievaluasi langsung
3	2	0	$2 + 3 \cdot 0 = 2$	$A[3] = 2$
2	-6	2	$-6 + 3 \cdot 2 = 0$	$A[2] + 3A[3] = -6 + 6 = 0$
1	2	0	$2 + 3 \cdot 0 = 2$	$A[1] + 3A[2] + 9A[3] = 2 - 18 + 18 = 2$
0	-1	2	$-1 + 3 \cdot 2 = 5$	$A[0] + 3A[1] + 9A[2] + 27A[3] = -1 + 6 - 54 + 54 = 5$

Tabel 4.4 menghitung invariannya secara terpisah di kolom terakhir: ia berlaku di setiap langkah. Buku: Subbab 4.5.1 dan Tabel 4.4.

Horner — Programnya

Kode 4.6 Aturan Horner terhadap evaluasi naif, dengan penelusuran gelung dan hitungan perkalian.

```cpp
#include <cstdio>
#include <vector>
using namespace std;
typedef long long LL;

// HORNER(A, n, x): p = 0; for i = n downto 0: p = A[i] + x*p
LL horner(const vector<LL>& A, LL x, LL& mults) {
    LL p = 0;
    int n = (int)A.size() - 1;
    for (int i = n; i >= 0; --i) {
        // invarian: p = sum_{k=0}^{n-(i+1)} A[k+i+1] * x^k
        p = A[i] + x * p;
        ++mults;
        printf("  i = %d   p = A[%d] + %lld * (previous p) = %lld\n", i, i, x, p);
    }
    return p;
}

// Evaluasi naif: hitung setiap pangkat x^k dari nol.
LL naive(const vector<LL>& A, LL x, LL& mults) {
    LL sum = 0;
    for (int k = 0; k < (int)A.size(); ++k) {
        LL power = 1;
        for (int t = 0; t < k; ++t) { power *= x; ++mults; }   // k perkalian
        sum += A[k] * power; ++mults;                          // satu lagi untuk A[k]*x^k
    }
    return sum;
}
```

```
int main() {
    vector<LL> A = {-1, 2, -6, 2};          // P(x) = 2x^3 - 6x^2 + 2x - 1, A[k] = koefisien x^k
    LL x = 3, mh = 0, mn = 0;
    printf("Horner on P(x) = 2x^3 - 6x^2 + 2x - 1 at x = 3\n");
    LL h = horner(A, x, mh);
    LL v = naive(A, x, mn);
    printf("Horner: P(3) = %lld using %lld multiplications\n", h, mh);
    printf("Naive : P(3) = %lld using %lld multiplications\n", v, mn);
```

Kode 4.6 (Lanjutan) *Aturan Horner terhadap evaluasi naif, dengan penelusuran gelung dan hitungan perkalian.*

```
    // Hitungan operasi untuk derajat lebih besar (semua koefisien 1, x = 1, jadi tanpa limpahan).
    printf("\n degree n   Horner mults   naive mults\n");
    for (int n : {3, 10, 100, 1000}) {
        vector<LL> B(n + 1, 1);
        LL a = 0, b = 0;
        LL p = 0; for (int i = n; i >= 0; --i) { p = B[i] + 1 * p; ++a; }
        naive(B, 1, b);
        printf(" %8d   %12lld   %11lld\n", n, a, b);
    }
    return 0;
}
```

```
Horner on P(x) = 2x^3 - 6x^2 + 2x - 1 at x = 3
  i = 3   p = A[3] + 3 * (previous p) = 2
  i = 2   p = A[2] + 3 * (previous p) = 0
  i = 1   p = A[1] + 3 * (previous p) = 2
  i = 0   p = A[0] + 3 * (previous p) = 5
Horner: P(3) = 5 using 4 multiplications
Naive : P(3) = 5 using 10 multiplications

 degree n   Horner mults   naive mults
        3              4            10
       10             11            66
      100            101          5151
     1000           1001        501501
```

Kode 4.6: $1,001$ *perkalian melawan* $501,501$ *pada derajat* 1000. *Uji diri:* `r = 1; for i = 1 to n: r = r * x` *berinvarian* $r = x^{i-1}$. *Buku: Subbab 4.5.1 dan Kode 4.6.*

Persoalan Terkerjakan B — Menelusuri Insertion Sort

Tabel 4.5 *Insertion sort* pada $\langle 8,3,6,3,9,1 \rangle$. Total pergeserannya, 9, sama dengan banyaknya inversi; t_j menghitung berapa kali uji `while` pada baris 4 dijalankan dan selalu satu lebih banyak daripada pergeserannya.

j	Kunci	Larik sesudah iterasi	Perbandingan	Pergeseran	t_j
2	3	$\langle 3,8,6,3,9,1 \rangle$	1	1	2
3	6	$\langle 3,6,8,3,9,1 \rangle$	2	1	2
4	3	$\langle 3,3,6,8,9,1 \rangle$	3	2	3
5	9	$\langle 3,3,6,8,9,1 \rangle$	1	0	1
6	1	$\langle 1,3,3,6,8,9 \rangle$	5	5	6
		total	**12**	**9**	**14**

Pada $\langle 8,3,6,3,9,1 \rangle$: di $j=4$ uji $3>3$ bernilai **salah**, sehingga kedua angka 3 tetap dalam urutan asalnya — stabilitas yang dibeli oleh `>` yang tegas.

Inversi larik aslinya: $4+1+2+1+1=9$ — tepat sama dengan banyaknya geseran, selalu, karena setiap geseran menghapus satu inversi dan tidak menciptakan inversi baru.

t_j = geseran $+1$ (satu uji yang gagal mengakhiri setiap gelung): $\sum t_j = 14$.

Tabel 4.5: 12 perbandingan, 9 geseran. Kode 4.7 di bab ini mencetak penelusuran yang sama. Uji diri: $\langle 4,1,3,9,7 \rangle$ memerlukan 3 geseran dan memiliki 3 inversi. Buku: Subbab 4.5.2 dan Tabel 4.5.

Persoalan Terkerjakan C — Menelusuri Pencarian Biner

Tabel 4.6 Pencarian biner pada larik 15 elemen untuk kunci yang ada (44) dan kunci yang tidak ada (20). Jendelanya menyusut $15 \to 7 \to 3 \to 1 \to 0$; empat probe cukup dalam setiap kasus.

q	Probe	*left*	*right*	*mid*	$A[mid]$	Keputusan	Ukuran jendela sesudahnya
44	1	1	15	8	31	terlalu kecil: $left = 9$	7
44	2	9	15	12	51	terlalu besar: $right = 11$	3
44	3	9	11	10	40	terlalu kecil: $left = 11$	1
44	4	11	11	11	44	ditemukan, kembalikan 11	—
20	1	1	15	8	31	terlalu besar: $right = 7$	7
20	2	1	7	4	15	terlalu kecil: $left = 5$	3
20	3	5	7	6	22	terlalu besar: $right = 5$	1
20	4	5	5	5	19	terlalu kecil: $left = 6$	0

Larik $\langle 3, 8, 11, 15, 19, 22, 27, 31, 36, 40, 44, 51, 57, 62, 70 \rangle$: $q = 44$ ditemukan di 11; $q = 20$ berakhir dengan $left = 6 > right = 5$ — ia seharusnya di antara 19 dan 22, tempat tidak ada posisi sama sekali.

Dengan $15 = 2^4 - 1$ elemen jendelanya menyusut $15 \to 7 \to 3 \to 1 \to 0$: paling banyak $\lfloor \lg 15 \rfloor + 1 = 4$ tilikan.

Tabel 4.6; Kode 4.8 di bab ini mencetak satu baris per tilikan. Uji diri: $q=70$ *dan* $q=3$ *juga memerlukan 4 tilikan. Buku: Subbab 4.5.3 dan Tabel 4.6.*

Persoalan Terkerjakan D — Mencacah Gelung Rangkap Tiga

Pseudokode 4.4 Gelung rangkap tiga yang baris terdalamnya hendak dihitung secara tepat.

```
LOOPS(n)
1  count = 0
2  for i = 1 to n
3      for j = i to n
4          for k = i to j
5              count = count + 1
6  return count
```

Gelung k: $j-i+1$ nilai. Gelung j: $\sum_{t=1}^{n-i+1} t = \frac{(n-i+1)(n-i+2)}{2}$. Gelung i, dengan $m=n-i+1$: $\frac{1}{2}\left(\sum m^2+\sum m\right)$.

$\sum m^2 = \frac{n(n+1)(2n+1)}{6}$ lewat teleskop $(m+1)^3-m^3=3m^2+3m+1$. Bersama-sama, Persamaan (4.9):

$$\text{eksekusi baris } 5 = \sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=i}^{j} 1 = \frac{n(n+1)(n+2)}{6} \tag{4.9}$$

Pseudokode 4.4: kerjakan dari dalam ke luar; setiap gelung menjadi sebuah jumlahan. Buku: Subbab 4.6.1, Pseudokode 4.6, dan Persamaan (4.9).

Cacahnya, Diperiksa

Tabel 4.7 Menghitung eksekusi terdalam LOOPS untuk $n=4$ dengan tangan. Subtotalnya adalah bilangan segitiga $10, 6, 3, 1$, dan jumlahnya $\frac{4\cdot 5\cdot 6}{6}=20$.

i	Sumbangan $j-i+1$ untuk $j=i,\ldots,4$	Subtotal
1	$1+2+3+4$	10
2	$1+2+3$	6
3	$1+2$	3
4	1	1
	total	**20**

$\frac{4\cdot 5\cdot 6}{6}=20$ ✓. Baris-baris Tabel 4.7 adalah bilangan segitiga; jumlah-jumlahnya $1, 4, 10, 20, 35, \ldots$ adalah bilangan *tetrahedral* — jeruk yang disusun menjadi limas segitiga.

$\frac{n^3+3n^2+2n}{6}=\Theta(n^3)$ dengan konstanta $\frac{1}{6}$: seperenam dari n^3 gelung rangkap tiga biasa.

UJI DIRI. `for i = 1 to n: for j = i+1 to n` berjalan $\frac{n(n-1)}{2}$ kali; `i = n; while i > 1: i = floor(i/2)` berjalan $\lfloor \lg n \rfloor$ kali ($10 \to 5 \to 2 \to 1$).

Kode 4.9 di bab ini menegaskan rumusnya sampai $n = 500$ *(*$20,958,500$*). Buku: Subbab 4.6.1 dan Tabel 4.7.*

Persoalan Terkerjakan E — Konstanta yang Eksplisit

Definisinya, Persamaan (4.10); O hanyalah separuh kanannya:

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ untuk semua } n \ge n_0 \tag{4.10}$$

(a) $5n^2 - 3n + 12 = \Theta(n^2)$**:** $c_2 = 5$ memerlukan $n \ge 4$; $c_1 = 4$ memerlukan $(n - 3/2)^2 + \frac{39}{4} \ge 0$, selalu benar. $n_0 = 4$ sungguh bekerja: $f(3) = 48 > 45$.

(b) $2^{n+1} = O(2^n)$**:** $c = 2$. **(c)** $2^{2n} = O(2^n)$ **salah:** setelah dibagi, $2^n \le c$ selamanya — patah pada $n = \max(n_0, \lfloor \lg c \rfloor + 1)$. Konstanta di dalam pangkat mengubah basisnya.

(d) $\lg(n!) = \Theta(n \lg n)$**:** $n! \le n^n$ memberi $c_2 = 1$; separuh atas faktornya memberi $n! \ge (n/2)^{n/2}$, jadi $c_1 = 1/4$ mulai $n_0 = 4$.

Kode 4.10 di bab ini memeriksa konstanta-konstanta ini untuk setiap n sampai 10^6 tanpa satu pelanggaran pun. Buku: Subbab 4.6.2 dan Persamaan (4.10).

Persoalan Terkerjakan F — Tiga Belas Fungsi Diurutkan

Sederhanakan dulu: $2^{\lg n} = n$ dan $4^{\lg n} = n^2$ (lewat $a^{\lg n} = n^{\lg a}$); $\lg(n!) = \Theta(n \lg n)$ menurut Persoalan E.

Tabel 4.8 Ketiga belas fungsi pada Persoalan Terkerjakan F dalam urutan laju pertumbuhan yang menaik. Fungsi pada baris yang sama saling Θ; kolom terakhir membenarkan setiap langkah naik.

Peringkat	Fungsi	Kelas yang disederhanakan	Alasan ia mengalahkan baris di atasnya
1	100	$\Theta(1)$	—
2	$\lg n$	$\Theta(\lg n)$	tumbuh tanpa batas
3	$\lg^2 n$	$\Theta(\lg^2 n)$	rasio $\lg n \to \infty$
4	$\sqrt{n}$	$\Theta(n^{1/2})$	polilog $= o(n^{\varepsilon})$
5	$2^{\lg n}$	$= n$	rasio $\sqrt{n} \to \infty$
6	$n \lg n, \lg(n!)$	$\Theta(n \lg n)$	rasio $\lg n \to \infty$
7	$n^2, 4^{\lg n}$	$= n^2$	rasio $n/\lg n \to \infty$
8	n^3	$\Theta(n^3)$	rasio $n \to \infty$
9	$(3/2)^n$	eksponensial	eksponensial mengalahkan polinomial
10	2^n	eksponensial	rasio $(4/3)^n \to \infty$
11	$n!$	$\Theta(\sqrt{n}(n/e)^n)$	rasio $\geq \frac{1}{2}(3/2)^{n-2} \to \infty$

Pada n kecil urutannya menipu: pada $n=16$ konstanta 100 mengalahkan n, dan $(3/2)^{16}<16^3$; baru pada $n=2^{32}$ setiap baris melampaui baris di atasnya.

Setiap langkah Tabel 4.8 dibenarkan oleh rasio yang menuju 0 atau ∞; Kode 4.11 di bab ini mencetak $\lg f(n)$ pada empat ukuran. Uji diri: $n^{1/\lg n}=2$, dan $n^{\lg 7}$ terletak di antara n^2 dan n^3. Buku: Subbab 4.6.3 dan Tabel 4.8.

Persoalan Terkerjakan G — Substitusi, Diperkuat

(a) $T(n)=T(\lfloor n/2 \rfloor)+n$, $T(1)=1$: tebak $T(n)\le 2n$; $T(n)\le 2\lfloor n/2 \rfloor+n\le 2n$ ✓; dan $T(n)\ge n$. Jadi $\Theta(n)$ — deret geometri $n+n/2+\cdots<2n$.

(b) Rekurensi Persamaan (4.11):

$$U(n)=2U(\lfloor n/2 \rfloor)+1, U(1)=1 \tag{4.11}$$

Tebak $U(n)\le cn$: kita mendapat $cn+1$ — **gagal untuk setiap** c, dan "$cn+1$ tetap $O(n)$" adalah persis kekeliruan Subbab 3.11.3.

Tebak $U(n)\le cn-d$: $2(c\lfloor n/2 \rfloor-d)+1\le cn-2d+1\le cn-d$ bila $d\ge 1$; basisnya memerlukan $c\ge 2$. Jadi $U(n)\le 2n-1$.

Klaim yang tampak lebih lemah, $U(n)\le 2n$, benar tetapi tidak dapat dibuktikan dengan cara ini; klaim yang lebih kuat justru terbukti. Sebenarnya $U(n)=2^{\lfloor \lg n \rfloor+1}-1$. Buku: Subbab 4.7.1 dan Persamaan (4.11).

Substitusi, Diperiksa Mesin

Kode 4.7 Kedua rekurensi pada Persoalan Terkerjakan G dievaluasi secara eksak, dengan batas-batas yang telah dibuktikan diperiksa untuk setiap $n \leq 10^6$.

```cpp
#include <cstdio>
#include <vector>
using namespace std;
typedef long long LL;

int main() {
    const int N = 1000000;
    vector<LL> T(N + 1), U(N + 1);
    T[1] = 1; U[1] = 1;
    for (int n = 2; n <= N; ++n) {
        T[n] = T[n / 2] + n;          // T(n) = T(floor(n/2)) + n
        U[n] = 2 * U[n / 2] + 1;      // U(n) = 2U(floor(n/2)) + 1
    }
    printf("  n   T(n)  2n   U(n)  2n-1\n");
    for (int n = 1; n <= 10; ++n)
        printf("%3d  %5lld %3d  %5lld %5d\n", n, T[n], 2 * n, U[n], 2 * n - 1);
    int badT = 0, badU = 0, badUn = 0, firstBadUn = 0;
    for (int n = 1; n <= N; ++n) {
        if (!(n <= T[n] && T[n] <= 2LL * n)) ++badT;
        if (!(U[n] <= 2LL * n - 1)) ++badU;
        if (U[n] > n) { ++badUn; if (!firstBadUn) firstBadUn = n; }
    }
    printf("n <= T(n) <= 2n     fails for %d values of n up to 10^6\n", badT);
    printf("U(n) <= 2n - 1      fails for %d values of n up to 10^6\n", badU);
    printf("U(n) <= n (c = 1)   fails for %d values, the first at n = %d\n", badUn, firstBadUn);
    return 0;
}
```

```
  n   T(n)  2n   U(n)  2n-1
  1      1   2      1     1
  2      3   4      3     3
  3      4   6      3     5
  4      7   8      7     7
  5      8  10      7     9
  6     10  12      7    11
  7     11  14      7    13
  8     15  16     15    15
  9     16  18     15    17
 10     18  20     15    19
n <= T(n) <= 2n     fails for 0 values of n up to 10^6
U(n) <= 2n - 1      fails for 0 values of n up to 10^6
U(n) <= n (c = 1)   fails for 999981 values, the first at n = 2
```

Kode 4.7: U tepat bertemu $2n-1$ pada pangkat dua, jadi $c=2$ tidak dapat diperbaiki. Uji diri: $T(n)=T(\lfloor n/2 \rfloor)+1$ memberi $T(n)\le \lg n+1$. Buku: Subbab 4.7.1 dan Kode 4.12.

Persoalan Terkerjakan H — Pohon Rekursi

$$T(n) = 3T(n/4) + n^2, T(1) = 1 \tag{4.12}$$

```
tingkat 0                        n^2                                  total  n^2
                    /             |             \
tingkat 1        (n/4)^2        (n/4)^2        (n/4)^2                total  (3/16) n^2
                 /  |  \        /  |  \        /  |  \
tingkat 2   sembilan simpul, masing-masing (n/16)^2                   total  (3/16)^2 n^2
              ...
tingkat i   3^i simpul, masing-masing (n/4^i)^2                       total  (3/16)^i n^2
              ...
tingkat k   3^k = n^(log_4 3) daun, masing-masing T(1) = 1            total  n^(log_4 3)
```

Tabel 4.9 Pohon rekursi $T(n)=3T(n/4)+n^2$, tingkat demi tingkat. Tiap tingkat berbiaya $\frac{3}{16}$ dari tingkat di atasnya, sehingga akarnya mendominasi.

Tingkat i	Simpul	Ukuran tiap simpul	Biaya tiap simpul	Biaya tingkat
0	1	n	n^2	n^2
1	3	$n/4$	$n^2/16$	$\frac{3}{16}n^2$
2	9	$n/16$	$n^2/256$	$\left(\frac{3}{16}\right)^2 n^2$
i	3^i	$n/4^i$	$n^2/16^i$	$\left(\frac{3}{16}\right)^i n^2$
$k=\log_4 n$	$3^k=n^{\log_4 3}$	1	1	$n^{\log_4 3}$

Untuk Persamaan (4.12) dengan $n=4^k$: tingkat i memuat 3^i simpul yang masing-masing berbiaya $n^2/16^i$, jadi tingkat itu berbiaya $(3/16)^i n^2$; Tabel 4.9 memperlihatkan akar yang mendominasi. Buku: Subbab 4.7.2, Persamaan (4.12), dan Tabel 4.9.

Menjumlahkan Pohonnya, Lalu Dua Penegasan

Tingkat-tingkat dalam: $n^2 \cdot \frac{1 - 3^k/n^2}{13/16} = \frac{16}{13}(n^2 - 3^k)$; tambahkan 3^k daunnya, menghasilkan Persamaan (4.13):

$$T(n) = \frac{16n^2 - 3^{k+1}}{13}, n = 4^k \tag{4.13}$$

Periksa dengan tangan: $T(4) = 3 + 16 = 19 = \frac{256 - 9}{13}$; $T(16) = 3 \cdot 19 + 256 = 313 = \frac{4096 - 27}{13}$. Jadi $n^2 \le T(n) < \frac{16}{13} n^2$: $\Theta(n^2)$.

Substitusi: $3 \cdot \frac{16}{13} \cdot \frac{n^2}{16} + n^2 = \frac{16}{13} n^2$ ✓. **Master:** $n^{\log_4 3} \approx n^{0.792}$, $f = n^2$, keteraturan $c = \frac{3}{16}$ — Kasus 3.

Pohon menemukan, substitusi membuktikan; konstanta keteraturannya tepat sama dengan rasio antartingkat. Kode 4.13 di bab ini memperlihatkan $T(n)/n^2 \to 1.230769 = \frac{16}{13}$. Buku: Subbab 4.7.2 dan Persamaan (4.13).

Persoalan Terkerjakan I — Empat Rekurensi, Satu Celah

Tabel 4.10 Empat rekurensi melalui Teorema Master. Yang terakhir terletak di celah antara Kasus 2 dan 3 dan dipecahkan dengan membuka rekurensinya.

Rekurensi	$\log_b a$	Bandingkan $f(n)$	Kasus	Penyelesaian	Eksak pada pangkat b
$8T(n/4)+n$	1.5	$n=O(n^{1.5-0.5})$	1	$\Theta(n^{1.5})$	$2n^{1.5}-n$
$27T(n/3)+n^3$	3	$n^3=\Theta(n^3)$	2	$\Theta(n^3\lg n)$	$n^3(\log_3 n+1)$
$5T(n/4)+n^2$	≈ 1.161	$n^2=\Omega(n^{1.161+0.839}), c=\frac{5}{16}$	3	$\Theta(n^2)$	$\frac{16n^2-5n^{\log_4 5}}{11}$
$T(n/2)+\lg n$	0	$\lg n$: tidak secara polinomial di atas 1	celah	$\Theta(\lg^2 n)$ dengan membuka rekurensi	$\frac{\lg n(\lg n+1)}{2}+1$

(a) $\log_4 8=1.5$, $\varepsilon=0.5$: Kasus 1. **(b)** $\log_3 27=3$: Kasus 2. **(c)** $\log_4 5\approx 1.161$, keteraturan $c=\frac{5}{16}$: Kasus 3.

(d) $T(n)=T(n/2)+\lg n$: $n^0=1$, dan $\lg n$ lebih besar tetapi tidak *secara polinomial* — **celah**. Uraikan: $T(2^k)=k+(k-1)+\cdots+1+1=\frac{k(k+1)}{2}+1=\Theta(\lg^2 n)$; pada $n=16$, 11 ✓.

Tabel 4.10. Di luar teorema juga: $2T(n-1)+1=\Theta(2^n)$, $1/2\,T(n/2)+n$ *(*$a<1$*),* $2^n T(n/2)+n$*. Buku: Subbab 4.7.3 dan Tabel 4.10.*

Keempatnya, Diperiksa Mesin

Kode 4.8 Keempat rekurensi pada Persoalan Terkerjakan I dievaluasi secara eksak lalu dibagi dengan batas yang diramalkan.

```cpp
#include <cstdio>
#include <cmath>
using namespace std;
typedef long double LD;

// Tiap rekurensi dievaluasi secara eksak pada pangkat-pangkat b, dengan T(1) = 1,
// lalu dibagi dengan batas yang diramalkan analisisnya. Rasio yang mengendap
// ke sebuah konstanta menegaskan batas Theta-nya.
LD Ta(LD n) { return n <= 1 ? 1 : 8 * Ta(n / 4) + n; }          // Theta(n^1.5)
LD Tb(LD n) { return n <= 1 ? 1 : 27 * Tb(n / 3) + n * n * n; } // Theta(n^3 lg n)
LD Tc(LD n) { return n <= 1 ? 1 : 5 * Tc(n / 4) + n * n; }      // Theta(n^2)
LD Td(LD n) { return n <= 1 ? 1 : Td(n / 2) + log2l(n); }       // Theta(lg^2 n)

int main() {
    printf("(a) T(n)=8T(n/4)+n      T(n)/n^1.5\n");
    for (int k = 2; k <= 14; k += 4) { LD n = powl(4, k); printf("    n=4^%-2d  %.6Lf\n", k, Ta(n) /
powl(n, 1.5L)); }
    printf("(b) T(n)=27T(n/3)+n^3   T(n)/(n^3 log3 n)\n");
    for (int k = 2; k <= 14; k += 4) { LD n = powl(3, k); printf("    n=3^%-2d  %.6Lf\n", k, Tb(n) / (n *
n * n * k)); }
    printf("(c) T(n)=5T(n/4)+n^2    T(n)/n^2\n");
    for (int k = 2; k <= 14; k += 4) { LD n = powl(4, k); printf("    n=4^%-2d  %.6Lf\n", k, Tc(n) / (n *
n)); }
    printf("(d) T(n)=T(n/2)+lg n    T(n)/lg^2 n\n");
    for (int k = 4; k <= 64; k *= 2) { LD n = powl(2, k); printf("    n=2^%-2d  %.6Lf\n", k, Td(n) /
((LD)k * k)); }
    return 0;
}
```

```
(a) T(n)=8T(n/4)+n      T(n)/n^1.5
    n=4^2   1.750000
    n=4^6   1.984375
    n=4^10  1.999023
    n=4^14  1.999939
(b) T(n)=27T(n/3)+n^3   T(n)/(n^3 log3 n)
    n=3^2   1.500000
    n=3^6   1.166667
    n=3^10  1.100000
    n=3^14  1.071429
(c) T(n)=5T(n/4)+n^2    T(n)/n^2
    n=4^2   1.410156
    n=4^6   1.454122
    n=4^10  1.454541
    n=4^14  1.454545
(d) T(n)=T(n/2)+lg n    T(n)/lg^2 n
    n=2^4   0.687500
    n=2^8   0.578125
    n=2^16  0.535156
    n=2^32  0.516602
    n=2^64  0.508057
```

Pada Kode 4.8 rasionya mengendap di 2, 1, $\frac{16}{11}$, *dan* $\frac{1}{2}$. *Rasio yang bergeser seperti* $1+1/k$, *seperti pada (b) dan (d), menandakan faktor logaritmik. Buku: Subbab 4.7.3 dan Kode 4.14.*

Persoalan Terkerjakan J — Kupon Laundry

LAUNDRT. Pada akhir hari ke-k beredar $T_k = 1+2+\cdots+k$ kupon; bonus hari ke-k adalah $1/T_k$. Diberikan $n \le 10^9$ (sampai $20,000$ kali), cetak $V_n = \sum_{k=1}^{n} 1/T_k$ dengan 11 desimal, dalam $O(1)$.

Dengan tangan: $V_1, \ldots, V_5 = 1, 4/3, 3/2, 8/5, 5/3 = 2/2, 4/3, 6/4, 8/5, \frac{10}{6}$ — polanya $\frac{2n}{n+1}$.

Bukti: $\frac{1}{T_k} = \frac{2}{k(k+1)} = 2\left(\frac{1}{k} - \frac{1}{k+1}\right)$, yang meneleskop menjadi Persamaan (4.14):

$$V_n = 2\sum_{k=1}^{n}\left(\frac{1}{k} - \frac{1}{k+1}\right) = 2\left(1 - \frac{1}{n+1}\right) = \frac{2n}{n+1} \tag{4.14}$$

$V_n \to 2$: *total bonusnya terbatas, berapa lama pun promosinya berjalan. Buku: Subbab 4.8.1 dan Persamaan (4.14).*

Kupon Laundry — Programnya

Pseudokode 4.5 Algoritma Kupon Laundry: satu evaluasi Persamaan (4.14) per pertanyaan.

```
LAUNDRT()
1  read T
2  for q = 1 to T
3      read n
4      print 2n / (n + 1) with 11 decimal places
```

Kode 4.9 LAUNDRT — Kupon Laundry, diselesaikan dalam $O(1)$ secara teleskopik.

```cpp
#include <cstdio>
using namespace std;

int main() {
    int T;
    if (scanf("%d", &T) != 1) return 0;
    while (T--) {
        long long n;
        if (scanf("%lld", &n) != 1) break;
        // sum_{k=1..n} 1/T_k  dengan T_k = k(k+1)/2   sama dengan   2n/(n+1)
        printf("%.11f\n", 2.0 * (double)n / ((double)n + 1.0));
    }
    return 0;
}
```

```
1.00000000000
1.33333333333
1.50000000000
1.81818181818
1.99999999800
```

Pseudokode 4.5 dan *Kode 4.9* pada $n = 1, 2, 3, 10, 10^9$. *Jebakannya:* `2 * n / (n + 1)` *dalam bilangan bulat bernilai* `1` *untuk setiap* $n \geq 1$ — *ubah tipenya sebelum membagi. Uji diri:* $\sum_{k=1}^{n} \frac{1}{k(k+2)} = \frac{3}{4} - \frac{2n+3}{2(n+1)(n+2)}$. *Buku: Subbab 4.8.1, Pseudokode 4.7, dan Kode 4.15.*

Persoalan Terkerjakan K — Menubin Lajur 2 × N

> **STRIPTILE.** Tutuplah lorong $2 \times N$ dengan ubin 1×2. Ada berapa cara, modulo 10^9+7? Sampai 10^5 pertanyaan, $0 \le N \le 10^{18}$, sasaran $O(\log N)$.

Dengan tangan: $D_0=1$ (tidak memasang apa pun), $D_1=1$, $D_2=2$ (VV, HH), $D_3=3$, $D_4=5$.

Kolom paling kiri ditutup satu ubin tegak (menyisakan $N-1$) atau paruh kiri dua ubin mendatar yang bertumpuk (menyisakan $N-2$) — saling lepas dan lengkap, sehingga Persamaan (4.15):

$$D_N = D_{N-1} + D_{N-2}, D_0 = D_1 = 1 \tag{4.15}$$

Itulah Fibonacci yang bergeser: $D_N = F_{N+1}$. Tabel berisi 10^{18} entri bukan tabel, melainkan angan-angan. Buku: Subbab 4.8.2 dan Persamaan (4.15).

Pelipatduaan Cepat

$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^n = \begin{pmatrix}F_{n+1} & F_n\\ F_n & F_{n-1}\end{pmatrix}$ menurut induksi; mengkuadratkan matriks untuk k menghasilkan matriks untuk $2k$, sehingga Persamaan (4.16):

$$F_{2k} = F_k(2F_{k+1} - F_k), F_{2k+1} = F_k^2 + F_{k+1}^2 \tag{4.16}$$

Pseudokode 4.6 Pelipatduaan cepat: pasangan Fibonacci (F_n, F_{n+1}) modulo p dalam $\Theta(\log n)$ perkalian.

```
FIB(n)                          // mengembalikan pasangan (F(n), F(n+1)) modulo p
1  if n == 0
2      return (0, 1)
3  (a, b) = FIB(floor(n / 2))    // a = F(k), b = F(k+1), dengan k = floor(n/2)
4  c = a * (2b - a) mod p        // F(2k)
5  d = (a*a + b*b) mod p         // F(2k+1)
6  if n is odd
7      return (d, c + d mod p)   // (F(2k+1), F(2k+2))
8  else return (c, d)            // (F(2k), F(2k+1))
```

Rumus Binet memerlukan $\sqrt{5}$ modulo p; pelipatduaan cepat hanya memerlukan bilangan bulat. Pseudokode 4.6 membagi dua n di setiap panggilan: $T(n) = T(n/2) + \Theta(1) = \Theta(\log n)$. Buku: Subbab 4.8.2, Persamaan (4.16), dan Pseudokode 4.8.

Pelipatduaan Cepat dengan Tangan

Tabel 4.11 Pelipatduaan cepat menghitung $(F_6, F_7) = (8, 13)$ dalam empat pemanggilan. Tiap baris memeriksa satu kejadian identitas pelipatduaan.

Pemanggilan	$k = \lfloor n/2 \rfloor$	$(a, b) = (F_k, F_{k+1})$	$c = a(2b - a)$	$d = a^2 + b^2$	Mengembalikan
FIB(0)	—	—	—	—	$(0, 1)$
FIB(1)	0	$(0, 1)$	$0 \cdot (2 - 0) =$	$0 + 1 = 1$	ganjil: $(1, 0 + 1) = (1, 1)$
FIB(3)	1	$(1, 1)$	$1 \cdot (2 - 1) =$	$1 + 1 = 2$	ganjil: $(2, 1 + 2) = (2, 3)$
FIB(6)	3	$(2, 3)$	$2 \cdot (6 - 2) =$	$4 + 9 = 13$	genap: $(8, 13)$

Jadi $D_5 = F_6 = 8$, cocok dengan $D_4 + D_3 = 5 + 3$.

Kedalamannya $\lfloor \log_2 N \rfloor + 2 \le 61$ untuk $N \le 10^{18}$: sekitar 6×10^6 operasi untuk 10^5 pertanyaan — jauh di dalam anggaran.

Setiap baris Tabel 4.11 memeriksa satu contoh identitas pelipatduaan. Buku: Subbab 4.8.2 dan Tabel 4.11.

STRIPTILE — Programnya

Kode 4.10 STRIPTILE — penubinan lajur $2 \times N$ dengan pelipatduaan cepat, $\Theta(\log N)$ per pertanyaan.

```cpp
#include <cstdio>
#include <utility>
using namespace std;
typedef long long LL;
const LL MOD = 1000000007LL;

// pelipatduaan cepat: mengembalikan (F(n) mod p, F(n+1) mod p) dengan F(0)=0, F(1)=1
pair<LL, LL> fib(LL n) {
    if (n == 0) return {0, 1};
    auto p = fib(n >> 1);
    LL a = p.first, b = p.second;             // a = F(k), b = F(k+1), k = n/2
    LL c = a * ((2 * b % MOD - a % MOD + MOD) % MOD) % MOD;   // F(2k)   = F(k)(2F(k+1)-F(k))
    LL d = (a * a % MOD + b * b % MOD) % MOD;                  // F(2k+1) = F(k)^2 + F(k+1)^2
    if (n & 1) return {d, (c + d) % MOD};
    return {c, d};
}

int main() {
    int T;
    if (scanf("%d", &T) != 1) return 0;
    while (T--) {
        LL n;
        if (scanf("%lld", &n) != 1) break;
        printf("%lld\n", fib(n + 1).first);    // T(n) = F(n+1)
    }
    return 0;
}
```

```
1
1
2
3
5
8
1346269
680057396
```

Kode 4.10 pada $N = 0, 1, 2, 3, 4, 5, 30, 10^{18}$. *Reduksi setiap hasil kali modulo* p *saat itu juga (dua residu muat dalam* `long long`*, tiga tidak), dan tambahkan* `MOD` *sebelum* `%` *karena* `2b - a` *bisa negatif. Jebakan:* F_N *alih-alih* F_{N+1}*; melupakan* $D_0 = 1$. *Buku: Subbab 4.8.2 dan Kode 4.16.*

Persoalan Terkerjakan L — Adakah Pasangan Berjumlah x?

***Brute force*:** cobalah semua $\binom{n}{2}$ pasangan — jelas benar, $\Theta(n^2)$ pada kasus terburuk.

Pseudokode 4.7 Algoritma urutkan-lalu-cari untuk persoalan jumlah pasangan, dalam $\Theta(n \lg n)$.

```
PAIR-SUM(a[0..n-1], x)
1   sort a into increasing order
2   for i = 0 to n - 1
3       want = x - a[i]
4       lo = i + 1, hi = n            // cari di a[i+1 .. n-1]
5       while lo < hi                 // invarian: a[lo-1] < want <= a[hi], bila keduanya ada
6           mid = lo + floor((hi - lo) / 2)
7           if a[mid] < want then lo = mid + 1
8           else hi = mid
9       if lo < n and a[lo] == want
10          return TRUE
11  return FALSE
```

Benar: jawaban TRUE menunjuk dua posisi berbeda yang berjumlah x; jika pasangan $p<q$ ada setelah pengurutan, pencarian dari $i=p$ menemukan $a[q]$ sebagai nilai pertama yang ≥ sasarannya.

Dengan tangan: $\langle 1,3,3,6,8,9 \rangle$, $x=12 \rightarrow 3+9$, TRUE; $x=16 \rightarrow$ FALSE (8 tidak boleh dipasangkan dengan dirinya sendiri).

Pseudokode 4.7: urutkan sekali ($\Theta(n \lg n)$), lalu n pencarian biner masing-masing $O(\lg n)$. Buku: Subbab 4.9.1 dan Pseudokode 4.9.

Adu Cepat, Diukur Waktunya

Kode 4.11 Gaya kasar melawan urutkan-dan-cari untuk persoalan jumlah pasangan, diukur waktunya pada masukan yang ukurannya berlipat dua.

```cpp
#include <cstdio>
#include <vector>
#include <random>
#include <algorithm>
#include <chrono>
using namespace std;
typedef long long LL;

// Gaya kasar: coba setiap pasangan i < j.  Theta(n^2) perbandingan pada kasus terburuk.
bool pairBrute(const vector<LL>& a, LL x) {
    size_t n = a.size();
    for (size_t i = 0; i < n; ++i)
        for (size_t j = i + 1; j < n; ++j)
            if (a[i] + a[j] == x) return true;
    return false;
}

// Lebih baik: urutkan sekali, lalu cari biner pasangan x - a[i] bagi setiap elemen.
// Theta(n lg n) untuk pengurutan ditambah n pencarian masing-masing Theta(lg n).
bool pairSorted(vector<LL> a, LL x) {             // menerima salinan: larik milik pemanggil tak
tersentuh
    sort(a.begin(), a.end());
    size_t n = a.size();
    for (size_t i = 0; i < n; ++i) {
        LL want = x - a[i];
        size_t lo = i + 1, hi = n;                // cari pasangannya di a[i+1 .. n-1]
        while (lo < hi) {
            size_t mid = lo + (hi - lo) / 2;
```

```
            if (a[mid] < want) lo = mid + 1; else hi = mid;
        }
        if (lo < n && a[lo] == want) return true;
    }
    return false;
}

double millis(chrono::steady_clock::time_point s) {
    return chrono::duration<double, milli>(chrono::steady_clock::now() - s).count();
}
```

Kode 4.11 (Lanjutan) *Gaya kasar melawan urutkan-dan-cari untuk persoalan jumlah pasangan, diukur waktunya pada masukan yang ukurannya berlipat dua.*

```
int main() {
    mt19937_64 rng(2026);                              // benih tetap: data yang sama pada setiap eksekusi
    printf("     n   brute (ms)   sorted (ms)   same answer?\n");
    for (int n = 2000; n <= 64000; n *= 2) {
        vector<LL> a(n);
        for (int i = 0; i < n; ++i) a[i] = 2LL * i + 1;    // n bilangan ganjil yang berbeda
        shuffle(a.begin(), a.end(), rng);
        LL x = 1;                                          // ganjil + ganjil adalah genap, jadi tak ada pasangan
berjumlah 1:
                                                           // kedua metode harus bekerja penuh
        auto s1 = chrono::steady_clock::now(); bool r1 = pairBrute(a, x);  double t1 = millis(s1);
        auto s2 = chrono::steady_clock::now(); bool r2 = pairSorted(a, x); double t2 = millis(s2);
        printf("%6d   %10.2f   %11.3f   %s\n", n, t1, t2, r1 == r2 ? "yes" : "NO");
    }
    vector<LL> small = {8, 3, 6, 3, 9, 1};
    printf("small example {8,3,6,3,9,1}: x=12 -> %d %d, x=16 -> %d %d\n",
           pairBrute(small, 12), pairSorted(small, 12), pairBrute(small, 16), pairSorted(small, 16));
    return 0;
}
```

```
     n   brute (ms)   sorted (ms)   same answer?
  2000         0.63         0.125   yes
  4000         2.46         0.230   yes
  8000         9.87         0.480   yes
 16000        41.79         0.994   yes
 32000       173.16         2.452   yes
 64000       670.66         4.631   yes
small example {8,3,6,3,9,1}: x=12 -> 1 1, x=16 -> 0 0
```

Kode 4.11 memakai n bilangan ganjil yang berbeda dan $x = 1$, sehingga tidak ada pasangan dan kedua metode mengerjakan seluruh kerjanya; benih acaknya tetap. Milidetik Anda akan berbeda; rasionya tidak. Buku: Subbab 4.9.1 dan Kode 4.17.

Membaca Rasionya

Tabel 4.12 Waktu terukur dari Kode 4.11. Melipatduakan n mengalikan waktu gaya kasar dengan sekitar 4, sebagaimana diramalkan $\Theta(n^2)$, dan waktu urutkan-dan-cari dengan sekitar 2, sebagaimana diramalkan $\Theta(n \lg n)$; keunggulannya sendiri berlipat dua pada setiap baris.

n	Gaya kasar (ms)	Rasio terhadap baris sebelumnya	Urutkan dan cari (ms)	Rasio terhadap baris sebelumnya	Kasar / lebih baik
2,000	0.63	—	0.125	—	5.0
4,000	2.46	3.90	0.230	1.84	10.7
8,000	9.87	4.01	0.480	2.09	20.6
16,000	41.79	4.23	0.994	2.07	42.0
32,000	173.16	4.14	2.452	2.47	70.6
64,000	670.66	3.87	4.631	1.89	144.8

Melipatduakan n: *brute force* $\times 3.87$–4.23 (diramalkan 4); urutkan-lalu-cari $\times$ sekitar 2 (diramalkan sedikit di atas 2). Keunggulannya tumbuh dari $5\times$ menjadi $145\times$.

Ekstrapolasi ke 10^6: *brute force* sekitar $164,000$ ms, hampir tiga menit; urutkan-lalu-cari sekitar 90 ms.

Mengapa hanya $145\times$, padahal cacah operasinya berbeda sekitar $1,000\times$? Gelung dalam *brute force* adalah hal termurah yang dikerjakan prosesor; pengurutan dan pencarian melompat-lompat dan bercabang. Konstanta penting untuk detik, bukan untuk pertumbuhan.

Rasio-rasio pada Tabel 4.12 itulah yang diramalkan analisisnya. Uji diri: setelah pengurutan, dua penunjuk dari kedua ujung selesai dalam $\Theta(n)$ langkah. Buku: Subbab 4.9.1 dan Tabel 4.12.

Keempat Belas Persoalan Dibandingkan

Tabel 4.13 Keempat belas persoalan bab ini sekilas: keluarga tempat masing-masing bernaung, alat dari Bab 1 sampai Bab 3 yang menyelesaikannya, dan biaya penyelesaiannya.

Persoalan	Subbab	Keluarga (Subbab 4.2.5)	Alat utama, dan di mana diajarkan	Biaya penyelesaiannya
LDCSUMAB	4.3	membalikkan rumus pencacahan	deret aritmetika, pembulatan ke atas (§3.9.1, §3.8.2); pencarian biner (§2.14)	$O(1)$ per pertanyaan
TOHU	4.4	bentuk tertutup dari sebuah jumlah	penyelisihan, teleskopik, induksi (§3.9.5)	$O(1)$ per pertanyaan
A — aturan Horner	4.5.1	gelung dengan invarian	bukti invarian gelung (§2.9)	$\Theta(n)$
B — penelusuran *insertion sort*	4.5.2	gelung dengan invarian	penelusuran tepat; t_j (§2.8, §2.10)	$\Theta(n+\text{inversi})$
C — penelusuran pencarian biner	4.5.3	gelung dengan invarian	invarian jendela; $\lfloor \lg n \rfloor + 1$ probe (§2.14)	$\Theta(\lg n)$
D — gelung rangkap tiga	4.6.1	mencacah langkah dengan penjumlahan	jumlah bersarang, $\sum m^2$ (§3.9)	$\frac{n(n+1)(n+2)}{6}$ langkah
E — konstanta eksplisit	4.6.2	pembuktian asimtotik	definisi O, Θ (§3.3, §3.5)	—
F — mengurutkan fungsi	4.6.3	pembuktian asimtotik	limit rasio (§3.8)	—
G — substitusi	4.7.1	rekurensi	hipotesis yang diperkuat (§3.11)	$\Theta(n)$

Tabel 4.13 (Lanjutan) *Keempat belas persoalan bab ini sekilas: keluarga tempat masing-masing bernaung, alat dari Bab 1 sampai Bab 3 yang menyelesaikannya, dan biaya penyelesaiannya.*

Persoalan	Subbab	Keluarga (Subbab 4.2.5)	Alat utama, dan di mana diajarkan	Biaya penyelesaiannya
H — pohon rekursi	4.7.2	rekurensi	jumlah tingkat geometris (§3.12)	$\Theta(n^2)$
I — Teorema Master	4.7.3	rekurensi	tiga kasus dan celahnya (§3.13)	beragam
J — Kupon Laundry	4.8.1	bentuk tertutup dari sebuah jumlah	teleskopik (§3.9.5)	$O(1)$ per pertanyaan
K — lajur $2 \times N$	4.8.2	rekurensi	Fibonacci, pelipatduaan cepat (§3.8.9)	$\Theta(\log N)$ per pertanyaan
L — jumlah pasangan	4.9	gaya kasar, lalu yang lebih baik	pengurutan dan pencarian biner (§2.14, §3.15)	$\Theta(n \lg n)$

POLA DI BALIK POLA. Gerakan 1: hitung kasus-kasus kecil dengan tangan dan tabulasikan. Gerakan 2: susun ulang perhitungannya sehingga kerja per jawaban menurun — meneleskopkan, membalik pencacahan, membawa invarian, membagi dua, atau mengurutkan agar pencarian menggantikan pemindaian. Lalu buktikan bahwa jawabannya tetap benar.

Pada Tabel 4.13, setiap persoalan punya cara gamblang yang salah atau terlalu lambat, setiap persoalan luluh oleh sebuah tabel, dan setiap jawaban diperiksa dua kali. Buku: Subbab 4.10 dan Tabel 4.13.

Profitina dalam Praktik

Anggaran sebagai aturan rancangan: halaman × sinyal × situs yang jatuh tempo harus muat dalam waktu yang tersedia. **Proyeksi dari bentuk tertutup:** pemindaian ulang selama suatu kurun dihitung dalam $\Theta(1)$, bukan hari demi hari.

Halaman nyaris kembar adalah Persoalan L untuk dokumen: kunci kanonik, lalu urutkan atau *hash* — sekitar $\Theta(n \lg n)$ alih-alih $\Theta(n^2)$. **Memeriksa dua kali** adalah gerbang rilis: lebih dari seribu uji otomatis harus lolos sebelum rilis mana pun.

Profitina Laundry: bentangan hari berurutan yang paling menguntungkan, lewat jumlah awalan $P[j]$ dan invarian pada minimum yang paling dini — satu lintasan $\Theta(n)$. Dua pekan di Keputih: **hari 2 sampai 7, Rp 430.000** ($340{,}000 - (-90{,}000)$).

Platform memberi saran; pelanggan yang memutuskan. Program lengkapnya, dengan peragaan aturan pemecah seri, adalah Kode 4.18. Buku: Subbab 4.11.

Kesalahan yang Sering Terjadi

Kode lebih dulu; melewatkan kasus kecil; memercayai pola tanpa bukti (a_1 pada TOHU). Kerjakan Langkah 1–6 dulu dan buktikan apa yang diisyaratkan tabel.

Mengabaikan anggaran — rujuklah Tabel 4.1. **Selisih satu di batas** — pembulatan ke atas vs ke bawah, `mid - 1`, F_N vs F_{N+1}.

Invarian tanpa terminasi; konstanta yang bergeser dalam substitusi; konstanta yang bergantung pada n.

Teorema Master di celahnya ($T(n/2)+\lg n$) atau di luar bentuknya ($2T(n-1)+1$).

Aritmetika: pembagian bilangan bulat (`3/4` bernilai `0`), limpahan diam-diam, `sqrt` dipercaya untuk bilangan bulat, `float` untuk sebelas desimal, waktu dibaca tanpa rasio.

Buku: Subbab 4.12.

Ringkasan — Lima Hal untuk Diingat

1. Tujuh langkah: format; diketahui dan ditanya; anggaran; kasus kecil; bentuk; buktikan; baru kemudian kode.

2. LDCSUMAB: urutannya dibuktikan, bloknya dihitung dengan T_n, dibalik dengan rumus kuadrat dan pembulatan ke atas, diamankan dengan sebuah invarian.

3. TOHU: menyelisihkan memulihkan a_k (dengan a_1 yang istimewa), teleskop menghasilkan $S_n = \frac{3}{4} - \frac{1}{2(n+1)(n+2)}$.

4. Persoalan A–L: invarian, cacah yang tepat, konstanta eksplisit, tiga metode rekurensi dan sebuah celah, teleskop, pelipatduaan cepat, adu cepat yang diukur.

5. Tabulasikan kasus-kasus kecil, lalu susun ulang perhitungannya — dan buktikan bahwa perhitungan yang disusun ulang itu tetap benar.

Buku: Subbab 4.13.

Pertemuan Berikutnya

Bab 5 — Heapsort dan Quicksort: dua pengurutan $\Theta(n \lg n)$ pada masukan yang lazim, dengan alasan yang sama sekali berbeda.

Analisisnya memakai persis apa yang kita latih hari ini: invarian gelung, jumlahan, dan rekurensi.

Sebelum kita bertemu: Soal Latihan 4.1–4.12 — di antaranya *ranking* untuk LDCSUMAB (4.9) dan varian TOHU yang a_1-nya tidak istimewa (4.10).

Penyelesaian lengkapnya ada di Subbab 4.15; cobalah masing-masing sepuluh menit lebih dulu.

Buku: Subbab 4.14–4.15, lalu Bab 5.

Kuliah 5 — Pengurutan dalam Praktik: Heapsort dan Quicksort

Perancangan dan Analisis Algoritma (EF234405)

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*, Bab 5

Dosen: Irfan Subakti — yifana@gmail.com

Bawalah larik $[4, 1, 3, 2, 16, 9, 10, 14, 8, 7]$*: hari ini kita membangun sebuah heap darinya, mengurutkannya, lalu mengurutkan larik kedua dengan cara yang sama sekali berbeda. Buku: Bab 5, Subbab 5.1–5.15.*

Yang Dibahas Pertemuan Ini

1. **Celahnya** — setiap pengurutan cepat yang kita kenal memerlukan larik kedua; setiap pengurutan di tempat yang kita kenal berperilaku kuadratik.
2. **Binary heap** — pohon tanpa penunjuk yang tersembunyi di dalam larik datar; `MAX-HEAPIFY`, `BUILD-MAX-HEAP` dalam $O(n)$, dan `HEAPSORT`, masing-masing ditelusuri di papan tulis.
3. **Antrean prioritas** — `MAXIMUM`, `EXTRACT-MAX`, `INCREASE-KEY`, `INSERT`.
4. **Quicksort** — pemartisian Lomuto dan Hoare, invarian empat wilayah, dan satu penelusuran lengkap.
5. **Analisis** — kasus terbaik, terburuk, $9:1$, rata-rata ($\approx 1.39\, n \lg n$), pengacakan, tumpukan panggilan, dan lantai $\Omega(n \lg n)$.
6. **Memilih pengurutan** — kestabilan, perilaku *cache*, dan Profitina Laundry.

Urutan pertemuan ini mengikuti bab ini subbab demi subbab. Buku: Bab 5, Subbab 5.1–5.15.

Dari Mana Kita Melanjutkan: Lubang dalam Tabel

Bab 2 memberi kita Insertion Sort dan invarian gelung; Bab 3 memberi kita O, Ω, Θ, rekurensi, Teorema Master, dan Merge Sort dengan $\Theta(n \lg n)$. Tabel 5.1 adalah seluruh persediaan kita — bacalah hanya kolom **kasus terburuk** dan **di tempat**.

Tabel 5.1 Empat algoritma pengurutan dari Bab 2 dan 3. Perhatikan bahwa yang cepat justru satu-satunya yang tidak bekerja di tempat.

Algoritma	Kasus terburuk	Kasus terbaik	Di tempat?	Ruang tambahan	Sifat utama
Insertion Sort	$\Theta(n^2)$	$\Theta(n)$	Ya	$O(1)$	Bagus sekali untuk data yang hampir terurut
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	Ya	$O(1)$	Selalu tepat $n-1$ penukaran
Bubble Sort	$\Theta(n^2)$	$\Theta(n)$	Ya	$O(1)$	Sederhana tetapi lambat
Merge Sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$	Tidak	$O(n)$	Cepat, tetapi butuh larik kedua

IMPIAN ITU. Sebuah pengurutan yang berjalan dalam waktu $O(n \lg n)$ **dan** di tempat, dengan memori tambahan $O(1)$. Tak satu pun di Tabel 5.1 memenuhi keduanya. **Heapsort** mengisi lubang itu dengan kasus terburuk $\Theta(n \lg n)$ dan ruang tambahan $O(1)$; **Quicksort** mengisinya dengan $\Theta(n \lg n)$ secara rata-rata, ruang tambahan $O(\lg n)$, dan kecepatan terbaik dalam praktik.

Di tempat *(in-place) berarti menata ulang di dalam larik yang diberikan dengan sejumlah tetap variabel tambahan.* `MERGE` *tidak sanggup melakukannya. Buku: Subbab 5.1.1 dan 5.1.2.*

Mengapa $n \lg n$, dan Wawasan dari Selection Sort

Ambil $n = 10^7$: $n^2 = 10^{14}$, sedangkan $n \lg n \approx 2.3 \times 10^8$, dengan rasio sekitar $430{,}000$. Pada 10^9 operasi per detik, itu berarti seperempat detik lawan **dua puluh tujuh jam**.

Selection Sort mengulang dua langkah: **Langkah A** memindai $A[1 \ldots i]$ untuk mencari yang terbesar, $\Theta(i)$; **Langkah B** menukarnya ke tempatnya, $\Theta(1)$. Totalnya $\Theta(n) \times (n-1) = \Theta(n^2)$, dan **seluruhnya** berasal dari Langkah A.

Andaikan menemukan dan membuang maksimum hanya berongkos $O(\lg n)$, totalnya menjadi $O(\lg n) \times (n-1) = O(n \lg n)$ — tanpa mengubah bentuk algoritmanya.

> **GAGASAN KUNCI.** Kadang cara mempercepat sebuah algoritma adalah mengganti **struktur data** tempat ia menyimpan datanya. Algoritma yang sama, wadah yang berbeda: kuadratik berubah menjadi $n \lg n$.

Mengapa sasarannya $n \lg n$, bukan n? Subbab 5.11 membuktikan bahwa tidak ada pengurutan berbasis pembandingan yang dapat lebih baik. Buku: Subbab 5.1.3 dan 5.2.

Sebuah Pohon yang Tersembunyi di Dalam Larik

Pohon biner disebut **hampir lengkap** bila setiap level penuh kecuali mungkin level terakhir, dan level terakhir terisi **dari kiri**. Bentuknya ditentukan oleh n semata, sehingga tidak pernah perlu disimpan.

Beri nomor simpul $1, 2, \ldots, n$ menurut urutan level dan letakkan simpul i di $A[i]$ ($A[0]$ tidak dipakai). Level d dimulai di indeks 2^d; simpul $i = 2^d + k$ beranak $2^{d+1} + 2k = 2i$ dan $2i+1$. Penurunan itu menghasilkan Persamaan (5.1), dan membaliknya menghasilkan Persamaan (5.2):

$$Left(i) = 2i, Right(i) = 2i + 1. \tag{5.1}$$

$$Parent(j) = \lfloor j/2 \rfloor. \tag{5.2}$$

Tabel 5.2 Berkeliling di pohon yang tersembunyi itu dengan aritmetika belaka. Setiap satu di antaranya adalah sebuah instruksi mesin tunggal.

Hubungan	Rumus	Contoh, $i=3$	Trik perangkat keras
Induk simpul i	$\lfloor i/2 \rfloor$	$Parent(3) =$	Geser kanan satu bit
Anak kiri simpul i	$2i$	$Left(3) = 6$	Geser kiri satu bit
Anak kanan simpul i	$2i+1$	$Right(3) = 7$	Geser kiri, lalu nyalakan bit terendah

Tabel 5.2 adalah seluruh sistem navigasinya: tidak ada penunjuk yang diikuti, dan itulah sebabnya heap cepat dalam praktik. Buku: Subbab 5.3.1 dan 5.3.2.

Rumus-rumusnya, Digambar

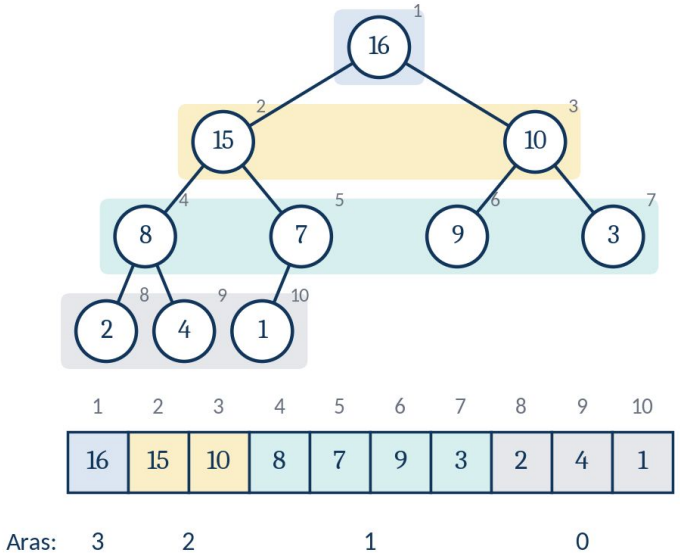
Parent(i)
kembalikan $\lfloor i/2 \rfloor$

Left(i)
kembalikan $2i$

Right(i)
kembalikan $2i + 1$

Sifat max-heap:

$A[\text{Parent}(i)] \geq A[i]$



Gambar 5.1 Tiga rumus navigasi, pohon yang mereka perikan, dan larik yang memuatnya. Pita-pita warna mengelompokkan simpul menurut kedalamannya dan menandai potongan larik yang bersesuaian di bawahnya; baris berlabel “Level” dihitung dari bawah ke atas, sehingga pita akar bernomor 3 dan baris terbawah bernomor 0.

Gambar 5.1 menggambar larik $[16, 15, 10, 8, 7, 9, 3, 2, 4, 1]$ *— juga sebuah max-heap yang sah, yang menunjukkan bahwa bilangan yang sama dapat membentuk banyak heap. Buku: Subbab 5.3.2.*

Sifat Max-Heap

DEFINISI. Sebuah **binary max-heap** adalah larik $A[1 \ldots n]$, dipandang sebagai pohon biner hampir lengkap, yang setiap simpulnya paling sedikit sama besar dengan masing-masing anaknya:

$$A[Parent(i)] \geq A[i] \text{ untuk setiap } i = 2, 3, \ldots, n. \tag{5.3}$$

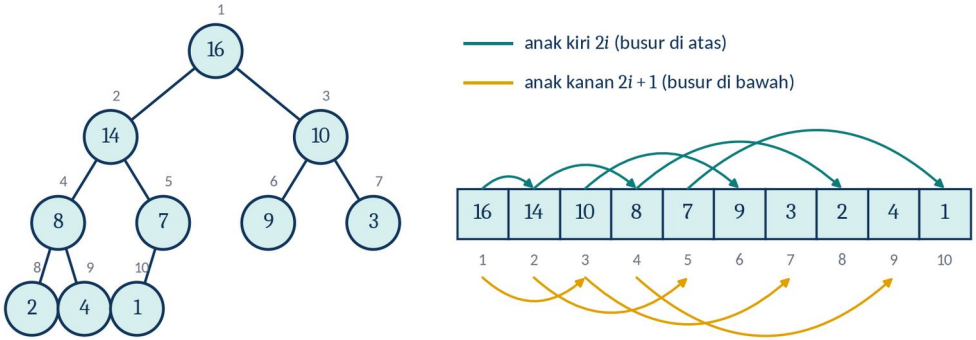
Sebuah **min-heap** membaliknya: $A[Parent(i)] \leq A[i]$ untuk setiap $i > 1$.

Periksa $A = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]$ terhadap Persamaan (5.3): simpul 1: $16 \geq 14, 10$ ✔️; simpul 2: $14 \geq 8, 7$ ✔️; simpul 3: $10 \geq 9, 3$ ✔️; simpul 4: $8 \geq 2, 4$ ✔️; simpul 5: $7 \geq 1$ ✔️ (tanpa anak kanan, $11 > 10$). Simpul 6–10 tidak beranak.

JEBAKAN YANG SERING TERJADI. *Heap* **tidak** terurut: 9 di indeks 6 muncul sesudah 8 di indeks 4. Satu-satunya jaminan global adalah bahwa $A[1]$ merupakan maksimum — dengan merangkai $A[Parent(i)] \geq A[i]$ hingga ke akar. ∎

Sifat ini murni lokal: tidak berkata apa pun tentang saudara atau paman. Kelemahan itulah yang membuat heap murah untuk diperbaiki. Buku: Subbab 5.3.3–5.3.5.

Sepuluh Bilangan yang Sama, Dilihat Dua Kali



Gambar 5.2 Sepuluh bilangan yang sama dilihat dua kali. Di kiri, pohon biner hampir lengkapnya, dengan indeks larik tiap simpul tercetak di atasnya. Di kanan, larik $A = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]$, dengan sebuah busur digambar dari tiap simpul ke masing-masing anaknya — ke anak kiri di atas larik, ke anak kanan di bawahnya. Busur-busur itulah yang dihitung oleh aritmetika indeks; tidak satu pun busur semacam itu pernah disimpan.

Pada Gambar 5.2, busur-busurnya adalah apa yang dihitung oleh aritmetika indeks; tidak ada busur yang pernah disimpan. Buku: Subbab 5.3.4.

Tinggi, Daun, dan Banyaknya Simpul

TEOREMA. *Heap* berisi n elemen memiliki tinggi $h = \lfloor \lg n \rfloor$. *Bukti.* Level $0 \ldots h-1$ penuh ($2^h - 1$ simpul) dan level h memuat antara 1 dan 2^h simpul, sehingga diperoleh Persamaan (5.4); mengambil lg menghasilkan $h \le \lg n < h+1$. ∎

$$2^h \le n < 2^{h+1}. \tag{5.4}$$

Daun. Simpul i mempunyai anak kiri jika dan hanya jika $2i \le n$, jadi simpul internalnya $1 \ldots \lfloor n/2 \rfloor$ dan daunnya $\lfloor n/2 \rfloor + 1 \ldots n$ — sebanyak $\lceil n/2 \rceil$, sedikit lebih dari separuh.

Lema. Paling banyak $\lceil n/2^{h+1} \rceil$ simpul bertinggi h: kupas daunnya lalu ulangi. Separuh simpul adalah daun, seperempat bertinggi 1, dan hanya akar yang bertinggi $\lfloor \lg n \rfloor$.

Untuk $n = 10$, $h = 3$; untuk $n = 10^6$, $h = 19$. Pegang lema ini: ia adalah seluruh isi bukti $O(n)$ nanti. Buku: Subbab 5.3.6–5.3.9.

`MAX-HEAPIFY`: Satu-satunya Perbaikan

Prasyarat: subpohon di $Left(i)$ dan $Right(i)$ sudah *max-heap*, tetapi $A[i]$ mungkin terlalu kecil. *Pascasyarat:* subpohon di i menjadi *max-heap*. Nilai yang bermasalah **tenggelam**, seperti yang diperlihatkan Pseudokode 5.1.

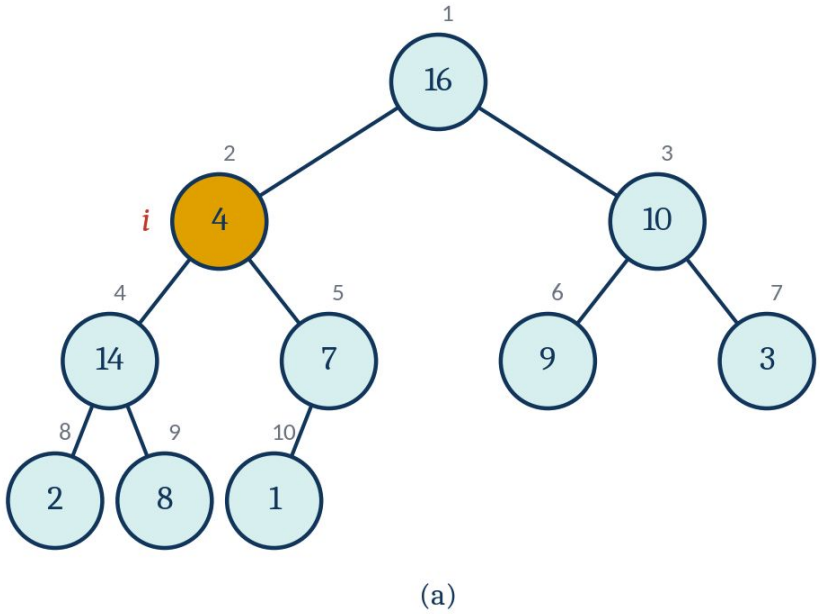
Pseudokode 5.1 `MAX-HEAPIFY`, yang juga disebut `HEAPIFY`: biarkan nilai di simpul i tenggelam sampai ia mendominasi kedua anaknya.

```
HEAPIFY(A, i):
 1.  l ← LEFT(i)                          // l = 2i
 2.  r ← RIGHT(i)                         // r = 2i + 1
 3.  largest ← i
 4.  if l ≤ heap-size and A[l] > A[i]
 5.      then largest ← l
 6.  if r ≤ heap-size and A[r] > A[largest]
 7.      then largest ← r
 8.  if largest ≠ i
 9.      swap A[i] ↔ A[largest]
10.      HEAPIFY(A, largest)              // turun secara rekursif
```

TIGA RINCIAN. (1) Uji `l ≤ heap-size` **sebelum** membaca $A[l]$. (2) Pakai `heap-size`, bukan panjang larik — Heapsort bergantung padanya. (3) Baris 6 membandingkan dengan `A[largest]`, bukan `A[i]`.

Dengan kata-kata sederhana: lihat tiga nilai, temukan yang terbesar; jika bukan simpul itu sendiri, tukar dan ulangi satu level ke bawah. Buku: Subbab 5.4.1 dan 5.4.2.

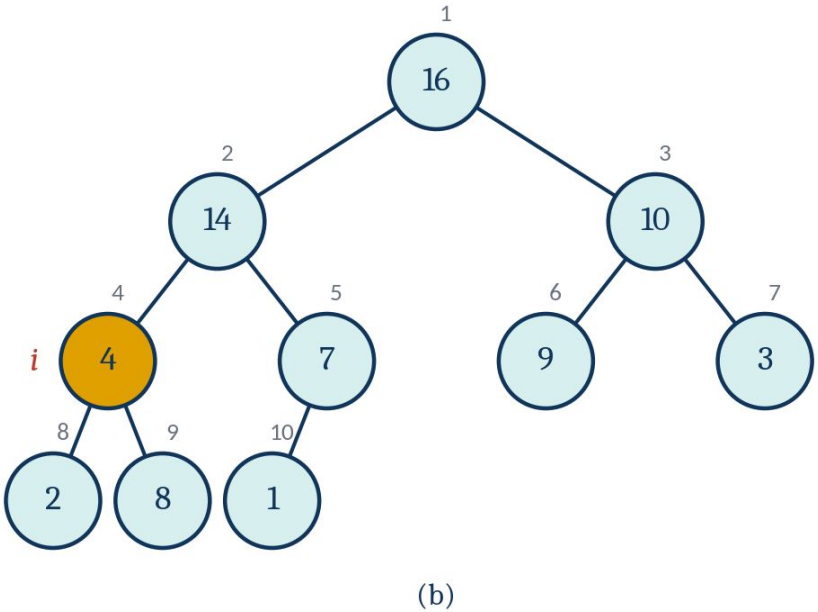
`MAX-HEAPIFY(A, 2)` — Langkah (a)



Gambar 5.3 `MAX-HEAPIFY(A, 2)`, langkah (a). Pemanggilannya dimulai. Simpul $i = 2$, yang diarsir, memuat nilai 4. Anak-anaknya adalah $A[4] = 14$ dan $A[5] = 7$. Kedua subpohon di bawahnya sudah merupakan *max-heap* yang sah; hanya simpul 2 yang salah tempat.

Gambar 5.3: $A = [16, 4, 10, 14, 7, 9, 3, 2, 8, 1]$. $l = 4$, $r = 5$; $14 > 4$ *sehingga* $\text{largest} = 4$; $7 > 14$*? tidak. Tukar* $A[2] \leftrightarrow A[4]$ *lalu berekursi di* 4. *Buku: Subbab 5.4.3.*

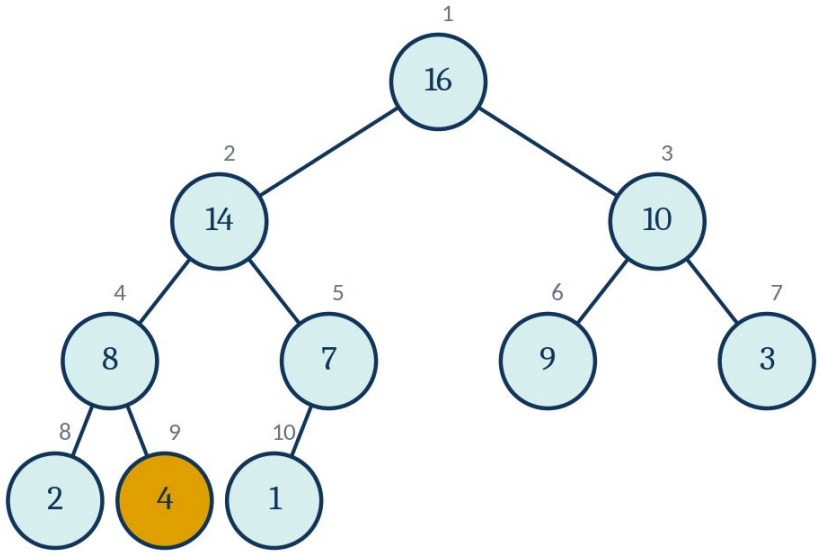
`MAX-HEAPIFY(A, 2)` — Langkah (b)



Gambar 5.4 Langkah (b). Sesudah penukaran pertama. Nilai 14 sudah mengambil tempat 4 di indeks 2, dan si pengganggu 4 kini berada di indeks $i=4$, yang diarsir. Anak-anaknya adalah $A[8]=2$ dan $A[9]=8$ — dan $8>4$, jadi penenggelamannya belum selesai.

Gambar 5.4: di $i=4$, $l=8$, $r=9$; $2>4$*? tidak;* $8>4$*? ya, jadi* largest$=9$. *Tukar* $A[4] \leftrightarrow A[9]$ *lalu berekursi di* 9. *Buku: Subbab 5.4.3.*

`MAX-HEAPIFY(A, 2)` — Langkah (c)



(c)

Gambar 5.5 Langkah (c). Perbaikannya rampung. Nilai 4 sudah berhenti di indeks 9, sebuah daun, dan seluruh pohonnya sekali lagi menjadi *max-heap*. Nilai itu menempuh lintasan 2 → 4 → 9, yang merupakan satu lintasan akar-ke-daun dan tidak lebih.

Gambar 5.5: di $i=9$, $18>10$, jadi tidak ada anak; tidak ada penukaran; kembali. Hasilnya $[16,14,10,8,7,9,3,2,4,1]$ — tiga panggilan pada satu lintasan menurun. Buku: Subbab 5.4.3.

Benar, dan $O(\lg n)$

Kebenaran, dengan induksi pada tinggi i. Daun langsung kembali. Selain itu, $A[\text{largest}]$ adalah yang terbesar dari tiga nilai; sesudah penukaran, subpohon yang tak tersentuh tetap *heap*, dan subpohon yang lain memenuhi tepat prasyarat `MAX-HEAPIFY` pada tinggi yang lebih rendah. ∎

Biaya, dalam satu baris. $\Theta(1)$ per level, paling banyak $\lfloor \lg n \rfloor$ level: $O(\lg n)$.

Biaya, lewat Bab 3. Bila level terbawah terisi tepat separuh, subpohon kiri memuat $\frac{2 \cdot 2^{h-1} - 1}{3 \cdot 2^{h-1} - 1} < \frac{2}{3}$ dari simpulnya, sehingga kita memperoleh Persamaan (5.5):

$$T(n) \le T(2n/3) + \Theta(1). \tag{5.5}$$

$a = 1$, $b = 3/2$, $n^{\log_b a} = n^0 = 1$, $f(n) = \Theta(1)$: Kasus 2 dengan $k = 0$, jadi $T(n) = \Theta(\lg n)$ dan, karena rekurensinya berupa pertidaksamaan, $O(\lg n)$.

Sejuta elemen: paling banyak sembilan belas langkah. Semiliar: paling banyak dua puluh sembilan. Buku: Subbab 5.4.4 dan 5.4.5.

`BUILD-MAX-HEAP`: Tiga Baris

Pseudokode 5.2 `BUILD-MAX-HEAP`: perbaiki setiap simpul dalam, dari yang terakhir kembali naik sampai ke akar.

```
BUILD-MAX-HEAP(A):
 1.  heap-size ← n
 2.  for i ← ⌊n/2⌋ downto 1
 3.      MAX-HEAPIFY(A, i)
```

Mengapa mulai di $\lfloor n/2 \rfloor$? Simpul $\lfloor n/2 \rfloor + 1 \ldots n$ adalah daun, yang sudah merupakan *heap* satu elemen. **Mengapa menghitung mundur?** Anak $2i$ dan $2i+1$ lebih besar daripada i, jadi setiap anak selesai sebelum induknya — Pseudokode 5.2 selalu memenuhi prasyarat `MAX-HEAPIFY`.

INVARIAN GELUNG. Pada awal setiap iterasi, setiap simpul $i+1, \ldots, n$ adalah akar sebuah *max-heap*. *Inisialisasi:* semuanya daun. *Pemeliharaan:* kedua anak i adalah akar *heap*, jadi panggilan itu menjadikan i akar *heap* pula. *Terminasi:* $i = 0$, jadi simpul 1 — seluruh larik — adalah *max-heap*. ∎

*Menghitung **naik** melanggar prasyaratnya dan memberi jawaban yang salah, bukan sekadar lambat. Buku: Subbab 5.5.1 dan 5.5.2.*

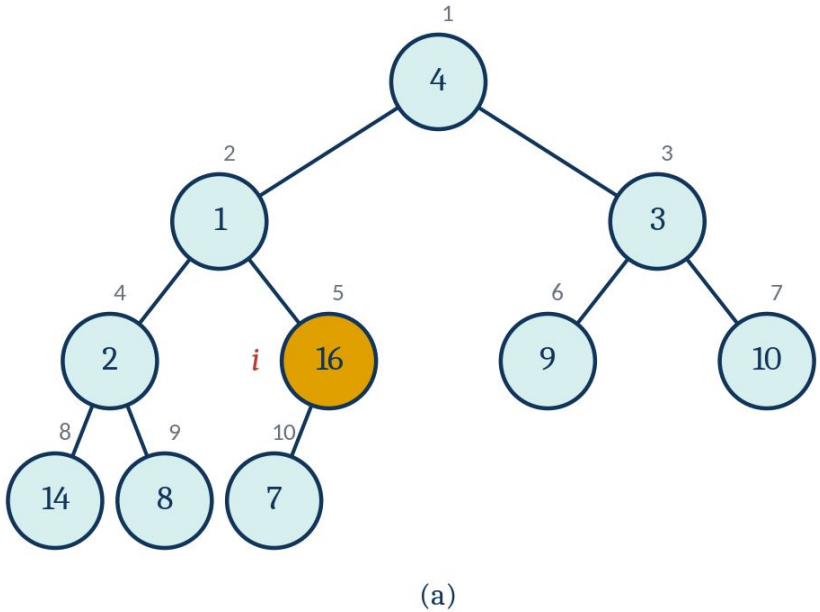
Masukannya: Sama Sekali Tanpa Urutan

	1	2	3	4	5	6	7	8	9	10
A	4	1	3	2	16	9	10	14	8	7

Gambar 5.6 Masukan bagi `BUILD-MAX-HEAP`: larik $A = [4, 1, 3, 2, 16, 9, 10, 14, 8, 7]$, tanpa urutan tertentu.

Gambar 5.6: $n = 10$, jadi gelung berjalan untuk $i = 5, 4, 3, 2, 1$; simpul 6–10 (berisi $9, 10, 14, 8, 7$) adalah daun dan dilewati. Buku: Subbab 5.5.3.

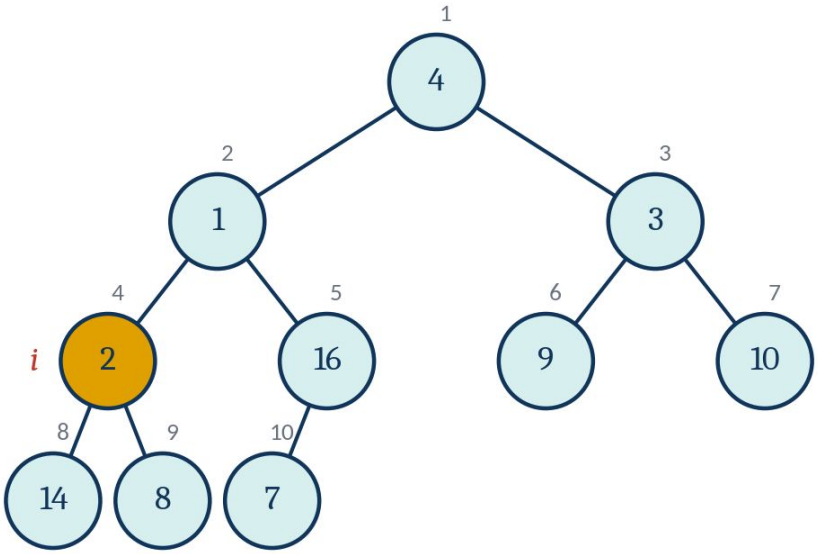
Pembangunan, Langkah (a): $i=5$



Gambar 5.7 `BUILD-MAX-HEAP`, langkah (a): $i=5$. Lariknya digambar sebagai pohon hampir lengkap. Simpul 5, yang diarsir, memuat 16. Satu-satunya anaknya adalah $A[10]=7$. Gelungnya mulai di sini karena simpul 5 adalah simpul terakhir yang masih punya anak sama sekali.

Gambar 5.7: $A[5]=16$*, satu-satunya anak* $A[10]=7$*;* $16>7$*, jadi tidak ada penukaran. Lariknya tidak berubah. Buku: Subbab 5.5.3.*

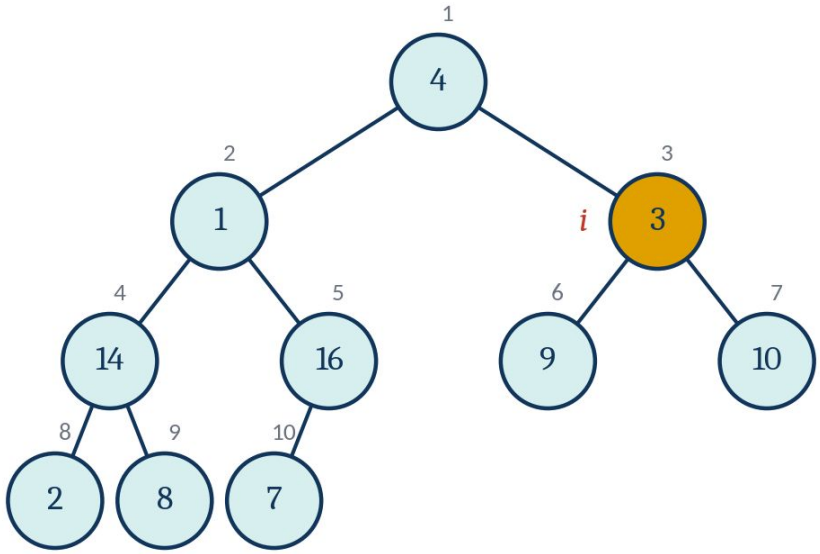
Pembangunan, Langkah (b): $i=4$



Gambar 5.8 Langkah (b): $i=4$. Simpul 4, yang diarsir, memuat 2, dan kedua anaknya $A[8]=14$ dan $A[9]=8$ sama-sama lebih besar. Nilai 2 harus tenggelam.

Gambar 5.8: yang terbesar dari $\{2, 14, 8\}$ *adalah* 14 *di indeks* 8*; tukar, menghasilkan* $[4, 1, 3, 14, 16, 9, 10, 2, 8, 7]$*; simpul* 8 *adalah daun. Buku: Subbab 5.5.3.*

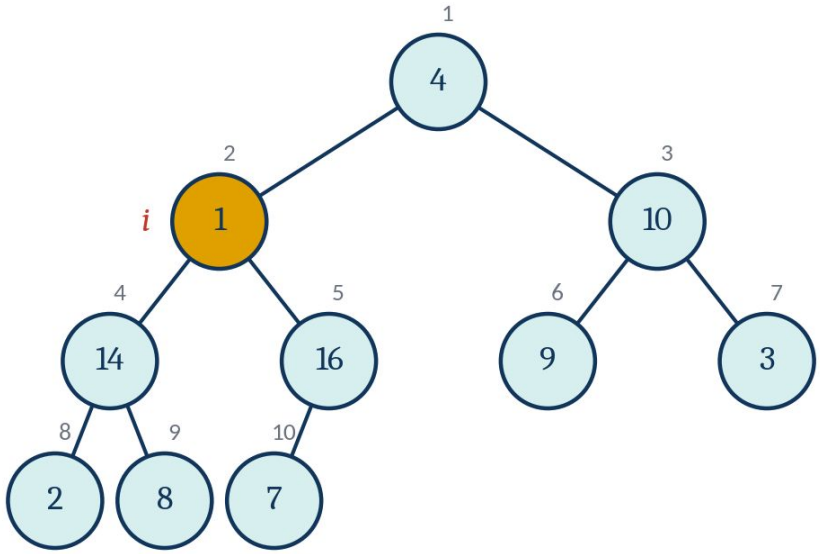
Pembangunan, Langkah (c): $i=3$



Gambar 5.9 Langkah (c): $i=3$. Simpul 3, yang diarsir, memuat 3; anak-anaknya adalah $A[6]=9$ dan $A[7]=10$. Anak kanannya yang lebih besar, jadi penukarannya ke kanan.

Gambar 5.9: yang terbesar dari $\{3,9,10\}$ adalah 10 di indeks 7 — penukarannya ke kanan: $[4,1,10,14,16,9,3,2,8,7]$. Buku: Subbab 5.5.3.

Pembangunan, Langkah (d): $i = 2$

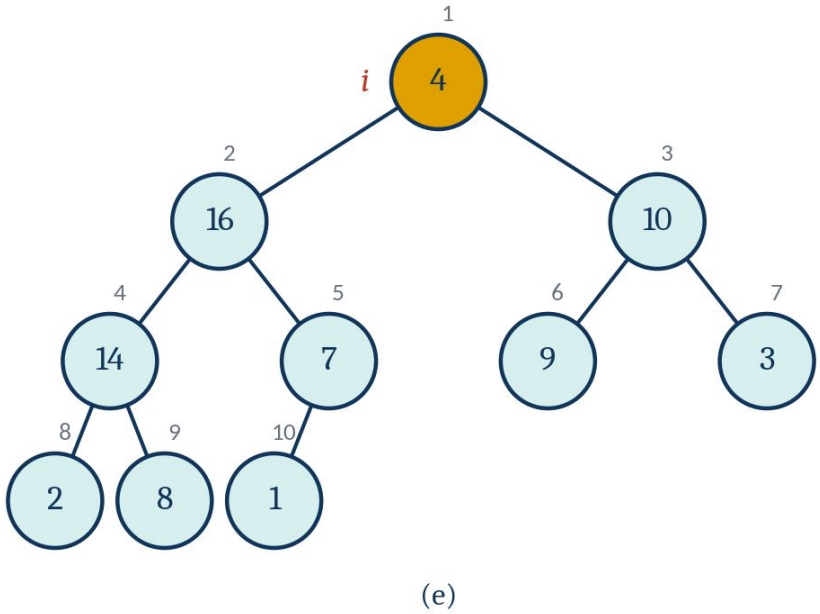


(d)

Gambar 5.10 Langkah (d): $i = 2$. Simpul 2, yang diarsir, memuat 1, nilai terkecil dalam lariknya. Ia harus tenggelam jauh: dua tingkat, ternyata.

Gambar 5.10: 1 bertukar dengan 16 (indeks 5), lalu dengan 7 (indeks 10): [4,16,10,14,7,9,3,2,8,1]. Dua penukaran, dua level. Buku: Subbab 5.5.3.

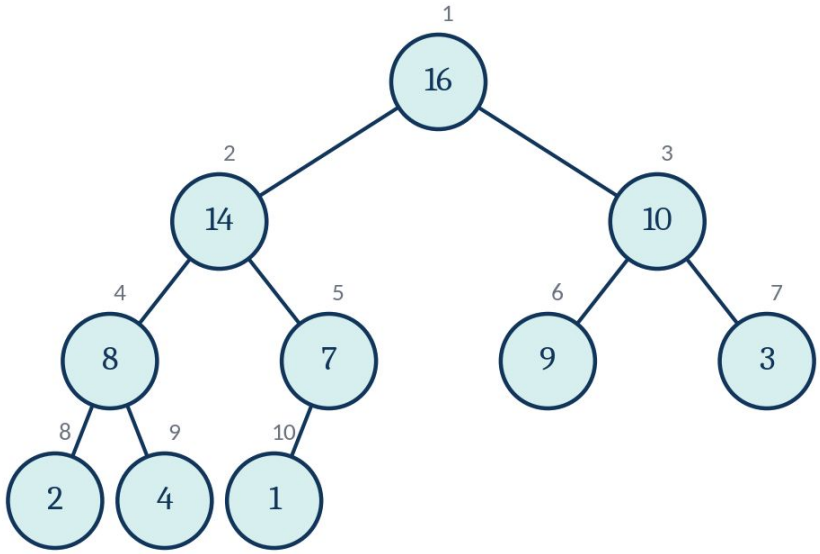
Pembangunan, Langkah (e): $i=1$



Gambar 5.11 Langkah (e): $i=1$. Iterasi terakhir, di akarnya. Simpul 1, yang diarsir, memuat 4. Segala yang di bawahnya sudah merupakan *heap* yang sah, jadi satu pemanggilan `MAX-HEAPIFY` ini merampungkan pekerjaannya.

Gambar 5.11: 4 *tenggelam* 1 → 2 → 4 → 9*, bertukar dengan* 16, 14*, dan* 8*. Tiga penukaran dalam satu panggilan ini saja. Buku: Subbab 5.5.3.*

Pembangunan, Langkah (f): Heap yang Sudah Jadi



(f)

Gambar 5.12 Langkah (f): max-heap yang sudah jadi $A = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]$. Setiap induk mendominasi kedua anaknya, dan maksimumnya, 16, sudah tiba di akar.

Gambar 5.12: total tujuh penukaran untuk sepuluh elemen, dan panggilan terakhir melakukan tiga di antaranya. Ketimpangan itulah benih analisisnya. Buku: Subbab 5.5.3.

Batas Longgar $O(n \lg n)$, Batas Ketat $O(n)$

Longgar: $\lfloor n/2 \rfloor$ panggilan × masing-masing $O(\lg n) = O(n \lg n)$. Benar, tetapi tidak ketat: panggilan pada simpul bertinggi h berongkos $O(h)$, bukan $O(\lg n)$.

Langkah 1–4: panggilan pada tinggi h berongkos $O(h)$; paling banyak $\lceil n/2^{h+1} \rceil$ simpul bertinggi h; tingginya berkisar $0 \ldots \lfloor \lg n \rfloor$. Menjumlahkan per tinggi menghasilkan Persamaan (5.6):

$$T(n) = \sum_{h=0}^{\lfloor \lg n \rfloor} \underbrace{\left\lceil \frac{n}{2^{h+1}} \right\rceil}_{\text{berapa simpul bertinggi } h} \times \underbrace{O(h)}_{\text{biaya masing-masing}} . \tag{5.6}$$

Langkah 5–8: $\lceil x \rceil \le 2x$ untuk $x \ge 1/2$, jadi $\lceil n/2^{h+1} \rceil \le n/2^h$; perpanjang jumlahannya sampai ∞; nilainya 2; maka $T(n) = O\left(n \sum_h h/2^h\right) = O(2n) = O(n)$, dan bersama $\Omega(n)$, $\Theta(n)$. ∎

Panggilan yang mahal jarang, panggilan yang murah umum: satu simpul bertinggi $\lg n$*, separuh simpul bertinggi* 0*. Buku: Subbab 5.5.4 dan 5.5.5.*

Mengapa $\sum_{h\geq 0} h/2^h = 2$

Mulailah dari $\sum_{i\geq 0} x^i = \frac{1}{1-x}$ untuk $|x|<1$. Turunkan suku demi suku: $\sum_{i\geq 1} i\,x^{i-1} = \frac{1}{(1-x)^2}$. Kalikan dengan x untuk memperoleh Persamaan (5.7):

$$\sum_{i=1}^{\infty} i\,x^i = \frac{x}{(1-x)^2}. \tag{5.7}$$

Masukkan $x = 1/2$: $\frac{1/2}{(1/2)^2} = 2$. Suku $h=0$ bernilai 0, sehingga Persamaan (5.8) menyusul:

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = 2. \ \blacksquare \tag{5.8}$$

Diukur: dari $n = 10^3$ sampai $n = 10^7$, `BUILD-MAX-HEAP` melakukan sekitar 0.74 penukaran per elemen pada setiap ukuran, sementara $\lg n$ naik dari 9 ke 23.

Delapan suku yang dijumlahkan dengan tangan sudah memberi 1.9296875*. Buku: Subbab 5.5.6 dan 5.5.7.*

Heapsort: Selection Sort yang Dibangun Ulang

Pseudokode 5.3 HEAPSORT: bangun sebuah *max-heap*, lalu berulang kali pindahkan akarnya ke ujung *heap* yang menyusut dan perbaiki akarnya.

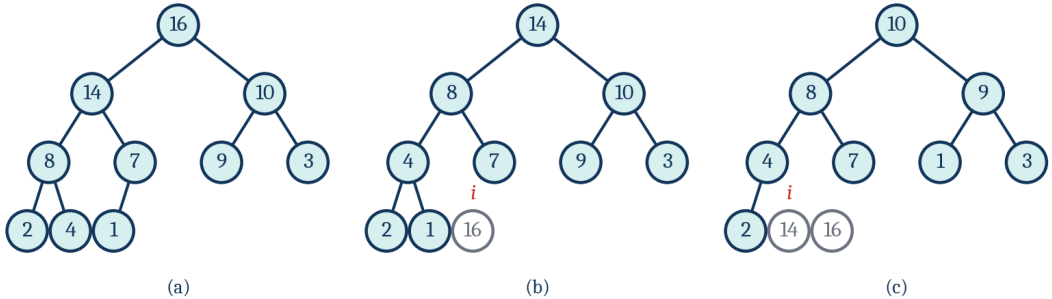
```
HEAPSORT(A):
 1.  BUILD-MAX-HEAP(A)                    // O(n)
 2.  for i ← n downto 2                   // n − 1 kali
 3.      swap A[1] ↔ A[i]                 // O(1)
 4.      heap-size ← heap-size − 1        // O(1)
 5.      MAX-HEAPIFY(A, 1)                // O(lg n)
```

Sesudah baris 1 dari Pseudokode 5.3, $A[1]$ adalah maksimum; baris 3 menaruhnya di $A[i]$ untuk selamanya; baris 4 menyusutkan *heap*; baris 5 memperbaiki akar. Batas antara *heap* $A[1 \ldots \text{heap-size}]$ dan ekor yang terurut bergeser satu langkah ke kiri pada setiap iterasi.

INVARIAN GELUNG. Pada awal setiap iterasi, $A[1 \ldots i]$ adalah *max-heap* yang memuat i elemen terkecil, dan $A[i{+}1 \ldots n]$ memuat $n - i$ elemen terbesar dalam keadaan terurut. Saat terminasi ($i = 1$) seluruh larik terurut. ∎

Heapsort berasal dari Williams (1964); pembangunan waktu linearnya dari Floyd, pada tahun yang sama. Buku: Subbab 5.6.1 dan 5.6.3.

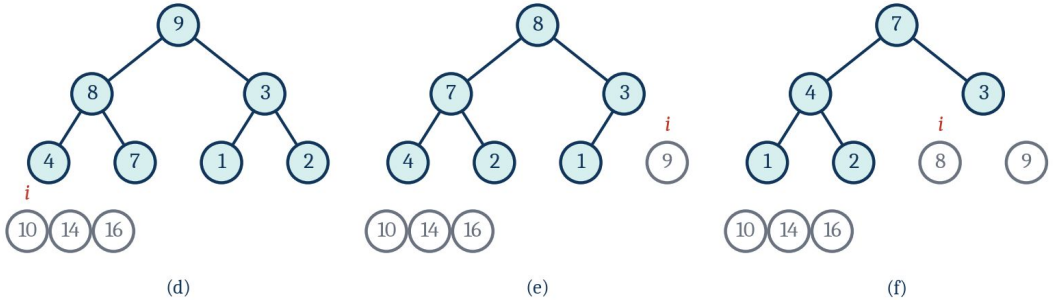
Heapsort, Langkah (a)–(c)



Gambar 5.13 Heapsort, langkah (a)–(c). Simpul yang diarsir masih berada di dalam *heap*; yang tidak diarsir sudah ditempatkan pada posisi terurutnya yang final dan tidak lagi menjadi bagiannya. **(a)** *heap* sesudah `BUILD-MAX-HEAP`. **(b)** sesudah menukarkan 16 ke posisi 10 dan memperbaiki akarnya. **(c)** sesudah menukarkan 14 ke posisi 9 dan memperbaikinya.

Gambar 5.13: $i=10$ *menghasilkan* $[14,8,10,4,7,9,3,2,1 \mid 16]$; $i=9$ *menghasilkan* $[10,8,9,4,7,1,3,2 \mid 14,16]$. *Garis tegak menandai ekor yang terurut. Buku: Subbab 5.6.2.*

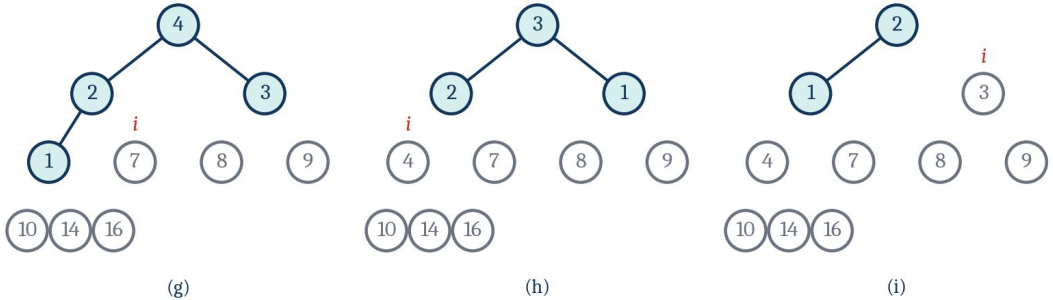
Heapsort, Langkah (d)–(f)



Gambar 5.14 Heapsort, langkah (d)–(f). *Heap*-nya menyusut satu simpul setiap kali dan ekor terurutnya, yang digambar tanpa arsiran dan dijajarkan di sepanjang bagian bawah, tumbuh menjemputnya. **(d)** 10 ditempatkan. **(e)** 9 ditempatkan. **(f)** 8 ditempatkan.

Gambar 5.14: $[9,8,3,4,7,1,2|10,14,16]$, *lalu* $[8,7,3,4,2,1|9,\ldots]$, *lalu* $[7,4,3,1,2|8,\ldots]$. *Buku: Subbab 5.6.2.*

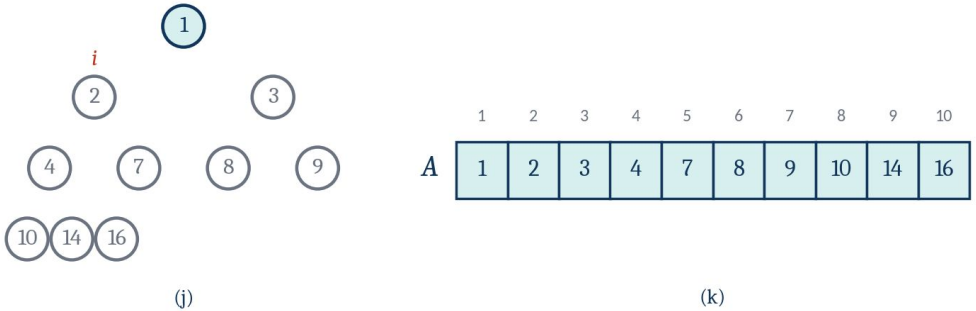
Heapsort, Langkah (g)–(i)



Gambar 5.15 Heapsort, langkah (g)–(i). Dengan *heap* yang tinggal empat, tiga, lalu dua simpul, pekerjaan per iterasinya sudah menjadi sepele: tingginya 2, lalu 1, lalu 1.

Gambar 5.15: $[4,2,3,1|7,\ldots]$, *lalu* $[3,2,1|4,\ldots]$, *lalu* $[2,1|3,\ldots]$. *Buku: Subbab 5.6.2.*

Heapsort, Langkah (j)–(k)



Gambar 5.16 Heapsort, langkah (j) dan (k). (j) sesudah penukaran terakhir, *heap*-nya memuat satu elemen. **(k)** lariknya yang final $A = [1, 2, 3, 4, 7, 8, 9, 10, 14, 16]$, dalam urutan tidak menurun.

Gambar 5.16: gelung berhenti di $i = 2$; heap satu elemen sudah terurut. Hasilnya $[1, 2, 3, 4, 7, 8, 9, 10, 14, 16]$ ✔. Buku: Subbab 5.6.2.

Biaya Heapsort, dan Mengapa Ia Tidak Stabil

Tabel 5.3 Biaya Heapsort, baris demi baris. Suku yang mendominasi adalah $n-1$ pemanggilan `MAX-HEAPIFY`.

Langkah	Biaya tiap eksekusi	Banyaknya eksekusi	Total
`BUILD-MAX-HEAP` (baris 1)	$O(n)$	1	$O(n)$
tukar + susut (baris 3–4)	$\Theta(1)$	$n-1$	$\Theta(n)$
`MAX-HEAPIFY` (baris 5)	$O(\lg n)$	$n-1$	$O(n \lg n)$
Total			$O(n \lg n)$

Tabel 5.3: $O(n)+\Theta(n)+O(n\lg n)=O(n\lg n)$. Kasus terburuknya $\Theta(n\lg n)$ menurut Subbab 5.11; dengan kunci yang berbeda-beda setiap kasusnya $\Theta(n\lg n)$; bila semua kunci sama, $\Theta(n)$.

TIDAK STABIL. Urutkan $\langle 1,a\rangle, \langle 1,b\rangle$ menurut bilangannya: pembangunan tidak mengubahnya, lalu $i=2$ menukarnya menjadi $\langle 1,b\rangle, \langle 1,a\rangle$. Kestabilan hilang pada larik sepanjang dua.

Heapsort, diringkas: $\Theta(n\lg n)$, ruang tambahan $O(1)$, tidak stabil — impian Subbab 5.1.2. Buku: Subbab 5.6.4 dan 5.6.5.

Heapsort dalam C++

Kode 5.1 mencetak larik sesudah setiap pengambilan, dengan garis tegak di antara *heap* yang menyusut dan ekor terurut yang tumbuh.

Kode 5.1 Heapsort lengkap yang dapat dijalankan, yang menarasikan setiap pengeluarannya.

```cpp
// HEAPSORT, mencetak lariknya sesudah tiap pengeluaran, dengan ekor terurutnya ditandai |.
#include <iostream>
#include <vector>
#include <string>
using namespace std;

vector<int> A;
int heapSize;                                   // seberapa banyak bagian A yang masih berupa heap
int n;                                          // seberapa banyak bagian A yang ada sama sekali

void printState(const string& label) {
    cout << label << "[";
    for (int i = 1; i <= n; ++i) {
        if (i == heapSize + 1 && heapSize < n) cout << "| ";
        cout << A[i] << (i < n ? " " : "");
    }
    cout << "]\n";
}

void maxHeapify(int i) {
    int l = 2 * i, r = 2 * i + 1, largest = i;
    if (l <= heapSize && A[l] > A[largest]) largest = l;
    if (r <= heapSize && A[r] > A[largest]) largest = r;
    if (largest != i) { swap(A[i], A[largest]); maxHeapify(largest); }
}

void buildMaxHeap() { for (int i = heapSize / 2; i >= 1; --i) maxHeapify(i); }
```

```
void heapsort() {
    heapSize = n;
    buildMaxHeap();
    printState("after BUILD-MAX-HEAP:  ");
    for (int i = n; i >= 2; --i) {
        cout << "  swap A[1]=" << A[1] << " with A[" << i << "]=" << A[i];
        swap(A[1], A[i]);
        --heapSize;                              // elemen yang baru diparkir kini dibekukan
        maxHeapify(1);
        cout << ", shrink to " << heapSize << ", heapify -> ";
        printState("");
    }
}
```

Kode 5.1 (Lanjutan) *Heapsort lengkap yang dapat dijalankan, yang menarasikan setiap pengeluarannya.*

```
int main() {
    A = {0, 4, 1, 3, 2, 16, 9, 10, 14, 8, 7};
    n = 10; heapSize = n;
    printState("input:                 ");
    heapsort();
    heapSize = n;                              // seluruh lariknya sudah terurut; buang pemisahnya
    printState("\nsorted:                ");

    bool ok = true;                            // pemeriksaan hasilnya secara mandiri
    for (int i = 2; i <= n; ++i) if (A[i - 1] > A[i]) ok = false;
    cout << "non-decreasing: " << (ok ? "yes" : "no") << "\n";
    return 0;
}
```

```
input:                [4 1 3 2 16 9 10 14 8 7]
after BUILD-MAX-HEAP:  [16 14 10 8 7 9 3 2 4 1]
  swap A[1]=16 with A[10]=1, shrink to 9, heapify -> [14 8 10 4 7 9 3 2 1 | 16]
  swap A[1]=14 with A[9]=1, shrink to 8, heapify -> [10 8 9 4 7 1 3 2 | 14 16]
  swap A[1]=10 with A[8]=2, shrink to 7, heapify -> [9 8 3 4 7 1 2 | 10 14 16]
  swap A[1]=9 with A[7]=2, shrink to 6, heapify -> [8 7 3 4 2 1 | 9 10 14 16]
  swap A[1]=8 with A[6]=1, shrink to 5, heapify -> [7 4 3 1 2 | 8 9 10 14 16]
  swap A[1]=7 with A[5]=2, shrink to 4, heapify -> [4 2 3 1 | 7 8 9 10 14 16]
  swap A[1]=4 with A[4]=1, shrink to 3, heapify -> [3 2 1 | 4 7 8 9 10 14 16]
  swap A[1]=3 with A[3]=1, shrink to 2, heapify -> [2 1 | 3 4 7 8 9 10 14 16]
  swap A[1]=2 with A[2]=1, shrink to 1, heapify -> [1 | 2 3 4 7 8 9 10 14 16]

sorted:               [1 2 3 4 7 8 9 10 14 16]
non-decreasing: yes
```

Setiap larik antara cocok dengan panel (b)–(j) pada keempat gambar. Seperti itulah wujud program yang benar. Buku: Subbab 5.6.6.

Antrean Prioritas: Kegunaan Sejati Heap

DEFINISI. Sebuah **antrean prioritas maks** menyimpan himpunan S berisi elemen berkunci dan mendukung $Insert(S,x)$, $Maximum(S)$, $Extract\text{-}Max(S)$, serta $Increase\text{-}Key(S,x,k)$ dengan k paling sedikit sama dengan kunci saat ini. Antrean prioritas min adalah cerminannya.

Pseudokode 5.4 `HEAP-MAXIMUM`: kunci terbesar sebuah *max-heap* selalu berada di akarnya.

```
HEAP-MAXIMUM(A):
 1.  if heap-size < 1 then error "heap underflow"
 2.  return A[1]
```

Tabel 5.4 Biaya keempat operasi antrean prioritas di bawah tiga implementasi. *Heap* adalah satu-satunya yang tidak punya entri $O(n)$.

Operasi	Antrean berbasis heap	Larik tak terurut	Larik terurut
`INSERT`	$O(\lg n)$	$O(1)$	$O(n)$
`MAXIMUM`	$O(1)$	$O(n)$	$O(1)$
`EXTRACT-MAX`	$O(\lg n)$	$O(n)$	$O(1)$
`INCREASE-KEY`	$O(\lg n)$	$O(1)$	$O(n)$

Pseudokode 5.4 hanyalah satu pembacaan larik. Tabel 5.4: sejuta penyisipan ditambah sejuta pengambilan berongkos sekitar 10^{12} operasi pada larik tak terurut, tetapi 4×10^{7} pada *heap*.

Kegunaannya: penjadwalan sistem operasi, simulasi kejadian, Dijkstra dan Prim, pengodean Huffman, penjadwalan perayapan. Di C++, `std::priority_queue<int>` *adalah max-heap;* `PriorityQueue` *di Java dan* `heapq` *di Python adalah min-heap (Python 3.14 menambahkan fungsi* `*_max` *terpisah untuk max-heap). Buku: Subbab 5.7.1–5.7.3, 5.7.7, dan 5.7.9.*

EXTRACT-MAX

Pseudokode 5.5 `HEAP-EXTRACT-MAX` dalam pseudokode. Simpan akarnya; pindahkan elemen terakhir ke akar; susutkan *heap*-nya satu; panggil `HEAPIFY` pada akarnya untuk memulihkan keteraturan; kembalikan nilai yang tadi disimpan.

```
HEAP-EXTRACT-MAX(A):                    // keluarkan dan kembalikan elemen terbesar A
 1.  max ← A[1]
 2.  A[1] ← A[heap-size]
 3.  heap-size ← heap-size − 1
 4.  HEAPIFY(A, 1)                       // bentuk ulang heap-nya
 5.  return max
```

Pseudokode 5.5 pada $[16,14,10,8,7,9,3,2,4,1]$*: simpan* 16*;* $A[1] \leftarrow 1$*; susutkan menjadi* 9*;* 1 *tenggelam melewati* 14*,* 8*,* 4*; kembalikan* 16*, menyisakan* $[14,8,10,4,7,9,3,2,1]$*.* $O(\lg n)$*. Buku: Subbab 5.7.4.*

`INCREASE-KEY`: Nilainya Mengapung Naik

Pseudokode 5.6 `HEAP-INCREASE-KEY`: naikkan kunci di posisi i, lalu apungkan ia melewati setiap induk yang lebih kecil.

```
HEAP-INCREASE-KEY(A, i, key):
 1.  if key < A[i] then error "new key is smaller than current key"
 2.  A[i] ← key
 3.  while i > 1 and A[PARENT(i)] < A[i]
 4.      swap A[i] ↔ A[PARENT(i)]
 5.      i ← PARENT(i)
```

Pseudokode 5.7 `MAX-HEAP-INSERT`: tambahkan satu slot berisi $-\infty$ di ujung, lalu naikkan ke kunci barunya.

```
MAX-HEAP-INSERT(A, key):
 1.  heap-size ← heap-size + 1
 2.  A[heap-size] ← −∞
 3.  HEAP-INCREASE-KEY(A, heap-size, key)
```

CONTOH YANG DIKERJAKAN. *Heap* $[10,8,9,4,7,3,2,1]$; naikkan posisi 8 dari 1 menjadi 20. Induk $A[4]=4<20$: tukar, $i=4$. Induk $A[2]=8<20$: tukar, $i=2$. Induk $A[1]=10<20$: tukar, $i=1$. Berhenti. Hasilnya $[20,10,9,8,7,3,2,4]$.

Pseudokode 5.6 adalah cerminan `MAX-HEAPIFY`: satu jalan dari daun ke akar, $O(\lg n)$. Pseudokode 5.7 menambahkan $-\infty$ lalu menaikkannya — penyisipan tidak memerlukan kode baru. Buku: Subbab 5.7.5 dan 5.7.6.

`INSERT` Adalah `INCREASE-KEY` yang Menyamar

Pseudokode 5.8 `HEAP-INSERT` dalam pseudokode. Tumbuhkan *heap*-nya satu slot di ujung, lalu apungkan kunci barunya ke atas, dengan memindahkan setiap induk yang kalah turun ke tempat anaknya, sampai sebuah induk menang atau akarnya tercapai.

```
HEAP-INSERT(A, key):
 1.  heap-size ← heap-size + 1
 2.  i ← heap-size
 3.  while i > 1 and A[PARENT(i)] < key
 4.      A[i] ← A[PARENT(i)]              // geser induk yang kalah ke bawah
 5.      i ← PARENT(i)
 6.  A[i] ← key                           // tulis kuncinya sekali saja, di akhir
```

Pseudokode 5.8 menggeser setiap induk yang kalah ke bawah dan menulis kuncinya sekali saja, di akhir: hasil yang sama dengan separuh jumlah operasi tulis ke memori. Buku: Subbab 5.7.6.

Quicksort: Bagi dan Kuasai, Kerjanya Dipindahkan

DEFINISI. Quicksort memilih sebuah **pivot**, menata ulang $A[p \ldots r]$ sehingga setiap elemen yang tidak lebih besar daripada pivot berada sebelum setiap elemen yang tidak lebih kecil daripadanya, lalu mengurutkan kedua bagiannya secara rekursif. Tidak ada langkah penggabungan.

Tabel 5.5 Merge Sort dan Quicksort adalah bayangan cermin satu sama lain. Masing-masing membelanjakan $\Theta(n)$ per tingkat; keduanya berbeda dalam *di mana* mereka membelanjakannya, dan dalam apakah belahannya terjamin.

	Merge Sort	Quicksort
Bagi	sepele — hitung titik tengahnya, $\Theta(1)$	mahal — pemartisian, $\Theta(n)$
Kuasai	dua pemanggilan rekursif	dua pemanggilan rekursif
Padu	mahal — pemaduan, $\Theta(n)$	sepele — tidak ada, $\Theta(1)$
Ukuran belahan	selalu tepat separuh-separuh	apa pun yang diberikan datanya
Memori tambahan	$\Theta(n)$ untuk penyangga pemaduannya	tidak ada

Tabel 5.5: Merge Sort *membeli* belahan yang seimbang dengan penyangga penggabungan; Quicksort menolak membayar dan menerima belahan apa pun yang diberikan datanya.

Tony Hoare menemukan Quicksort pada 1959 dan menerbitkannya pada 1962. Buku: Subbab 5.8.1 dan 5.8.2.

PARTITION Lomuto: Empat Wilayah

Pseudokode 5.9 PARTITION versi Lomuto: satu sapuan dari kiri ke kanan yang menumbuhkan wilayah elemen yang tidak lebih besar daripada *pivot*-nya.

```
PARTITION(A, p, r):
 1.  x ← A[r]                        // pivotnya adalah elemen terakhir
 2.  i ← p − 1                       // i menandai ujung wilayah "≤ x"
 3.  for j ← p to r − 1
 4.      if A[j] ≤ x
 5.          i ← i + 1
 6.          swap A[i] ↔ A[j]
 7.  swap A[i+1] ↔ A[r]              // jatuhkan pivotnya ke celahnya
 8.  return i + 1
```

INVARIAN GELUNG. Pada awal setiap iterasi: $A[k] \le x$ untuk $p \le k \le i$; $A[k] > x$ untuk $i+1 \le k \le j-1$; $A[r] = x$. Indeks $j \dots r-1$ belum diperiksa. Saat terminasi, baris 7 menjatuhkan pivot ke celahnya dan menegakkan Persamaan (5.9).

$$A[k] \le A[q] \text{ untuk } p \le k < q,\ A[k] > A[q] \text{ untuk } q < k \le r \tag{5.9}$$

Pseudokode 5.9 melakukan satu lintasan sebanyak $r-p$ iterasi: $\Theta(n)$. Buku: Subbab 5.8.3.

Satu Pemartisian Lomuto, dengan Tangan

$A = [2, 8, 7, 1, 3, 5, 6, 4]$, pivot $x = A[8] = 4$, $i = 0$. Tabel 5.6 menjalankan gelungnya.

Tabel 5.6 `PARTITION` versi Lomuto pada $[2, 8, 7, 1, 3, 5, 6, 4]$ dengan *pivot* 4. Awalan yang digarisbawahi adalah wilayah 1 — elemen-elemen yang sudah diketahui ≤ 4.

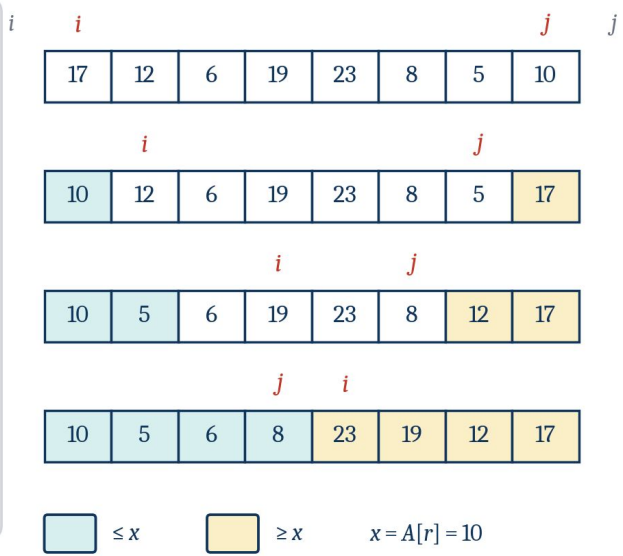
j	$A[j]$	$A[j] \le 4$?	Tindakan	Larik sesudahnya
1	2	ya	$i \to 1$; tukar $A[1] \leftrightarrow A[1]$	$[\underline{2}, 8, 7, 1, 3, 5, 6, $
2	8	tidak	tidak ada	$[\underline{2}, 8, 7, 1, 3, 5, 6, $
3	7	tidak	tidak ada	$[\underline{2}, 8, 7, 1, 3, 5, 6, $
4	1	ya	$i \to 2$; tukar $A[2] \leftrightarrow A[4]$	$[\underline{2, 1}, 7, 8, 3, 5, 6, $
5	3	ya	$i \to 3$; tukar $A[3] \leftrightarrow A[5]$	$[\underline{2, 1, 3}, 8, 7, 5, 6, $
6	5	tidak	tidak ada	$[\underline{2, 1, 3}, 8, 7, 5, 6, $
7	6	tidak	tidak ada	$[\underline{2, 1, 3}, 8, 7, 5, 6, $

Lalu $i = 3$, sehingga baris 7 menukar $A[4] \leftrightarrow A[8]$: $A = [2, 1, 3, 4, 7, 5, 6, 8]$, $q = 4$. Kiri $[2, 1, 3] \le 4$ ✔️, kanan $[7, 5, 6, 8] > 4$ ✔️.

Pivot 4 *kini berada di posisi terurut akhirnya dan tidak akan pernah berpindah lagi. Buku: Subbab 5.8.4.*

PARTITION Hoare, Ditelusuri

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  j ← r + 1
4  while TRUE
5      repeat j ← j − 1
6          until A[j] ≤ x
7      repeat i ← i + 1
8          until A[i] ≥ x
9      if i < j
10         exchange A[i] ↔ A[j]
11     else return j
```



Gambar 5.17 PARTITION versi Hoare, dengan penelusuran lengkap pada $[17, 12, 6, 19, 23, 8, 5, 10]$. Hijau toska menandai elemen yang sudah diketahui $\le x$, kuning tua yang sudah diketahui $\ge x$. Penunjuk i dan j berangkat dari tepat di luar lariknya lalu merapat; ketika keduanya bersilangan, pemartisiannya selesai dan j dikembalikan.

Gambar 5.17, pivot $x = A[8] = 10$: penukarannya menghasilkan $[10, 12, \ldots, 17]$, $[10, 5, 6, 19, 23, 8, 12, 17]$, $[10, 5, 6, 8, 23, 19, 12, 17]$; lalu $i = 5 > j = 4$: kembalikan 4. Buku: Subbab 5.8.5 dan 5.8.6.

Skema Hoare, dan Jebakan Pivotnya

Pseudokode 5.10 `PARTITION` versi Hoare: dua penunjuk merapat dari kedua ujung dan menukarkan pasangan yang salah tempat sampai keduanya bersilangan.

```
PARTITION(A, p, r):                    // Hoare, 1961
 1.  x ← A[r]                          // NILAI pivotnya (slot pivotnya tidak berpindah)
 2.  i ← p − 1
 3.  j ← r + 1
 4.  while TRUE
 5.      repeat j ← j − 1 until A[j] ≤ x
 6.      repeat i ← i + 1 until A[i] ≥ x
 7.      if i < j then swap A[i] ↔ A[j]
 8.      else return j
```

Lomuto mengembalikan **posisi akhir pivot**: berekursi pada $(p, q-1)$ dan $(q+1, r)$. Pseudokode 5.10 milik Hoare mengembalikan sebuah **batas**: berekursi pada (p, q) dan $(q+1, r)$.

JEBAKAN PIVOT. Dengan $x = A[r]$ dan rekursi (p, q), maksimum di $A[r]$ membuat kedua penunjuk berhenti di r, sehingga $q = r$ dan panggilan kiri menerima sublarik yang sama selamanya. Ambil $x = A[p]$, seperti CLRS: dengan $x = A[1] = 17$ pemartisian mengembalikan $j = 5$, kiri $[10, 12, 6, 5, 8]$, kanan $[23, 19, 17]$.

Skema Hoare melakukan sekitar tiga kali lebih sedikit penukaran dan tangguh terhadap kunci kembar; pelajari Lomuto lebih dulu. Buku: Subbab 5.8.5 dan 5.8.6.

Quicksort Itu Sendiri — Empat Baris

Pseudokode 5.11 `QUICKSORT` dengan pemartisian Lomuto: partisikan, lalu urutkan kedua sisi *pivot*-nya secara rekursif.

```
QUICKSORT(A, p, r):
 1.  if p < r
 2.      q ← PARTITION(A, p, r)
 3.      QUICKSORT(A, p, q − 1)          // Lomuto: pivot di q sudah final
 4.      QUICKSORT(A, q + 1, r)
```

Panggilan awal Pseudokode 5.11 adalah `QUICKSORT(A, 1, n)`. Bila $p \geq r$, sublariknya berisi paling banyak satu elemen dan sudah terurut; setiap panggilan menerima sublarik yang benar-benar lebih pendek, sehingga rekursinya berakhir.

KESALAHAN SATU-ANGKA. Dengan pemartisian Lomuto, menulis `QUICKSORT(A, p, q)` berekursi selamanya ketika pivot adalah maksimum. **Lomuto mengecualikan pivot dari kedua panggilan; Hoare tidak mengecualikannya dari satu pun.**

Tulislah pemartisian dan kedua panggilan rekursifnya bersamaan, jangan pernah terpisah. Buku: Subbab 5.8.7.

Quicksort, Ditelusuri Tuntas

Kode 5.2 mengurutkan $[2,8,7,1,3,5,6,4]$ dan mencetak larik sesudah setiap pemartisian; kurung siku menandai sublarik yang sedang dikerjakan dan indentasi menunjukkan kedalamannya.

Kode 5.2 Quicksort dengan pemartisian Lomuto, yang menarasikan rekursinya. Indentasinya menunjukkan kedalaman pemanggilannya.

```cpp
// QUICKSORT dengan pemartisian Lomuto, mencetak seluruh lariknya sesudah tiap pemartisian.
#include <iostream>
#include <vector>
#include <string>
using namespace std;

vector<int> A;
int calls = 0;

void showWindow(int p, int r, const string& note) {
    cout << "  ";
    for (int k = 0; k < (int)A.size(); ++k) {
        if (k == p) cout << "[";
        cout << A[k];
        if (k == r) cout << "]"; else cout << " ";
    }
    cout << "  " << note << "\n";
}

int partition(int p, int r) {
    int x = A[r], i = p - 1;
    for (int j = p; j <= r - 1; ++j)
        if (A[j] <= x) swap(A[++i], A[j]);
    swap(A[i + 1], A[r]);
    return i + 1;
}

void quicksort(int p, int r, int depth) {
```

```
    if (p >= r) {                                   // nol atau satu elemen: tidak ada yang perlu
dikerjakan
        if (p == r) cout << string(2 * depth + 2, ' ') << "QUICKSORT(" << p << "," << r
                         << "): one element, return\n";
        return;
    }
    ++calls;
    cout << string(2 * depth + 2, ' ') << "QUICKSORT(" << p << "," << r
         << "): pivot = A[" << r << "] = " << A[r] << "\n";
    int q = partition(p, r);
    cout << string(2 * depth + 2, ' ');
    showWindow(p, r, "after PARTITION, pivot " + to_string(A[q]) + " is final at index " + to_string(q));
    quicksort(p, q - 1, depth + 1);
    quicksort(q + 1, r, depth + 1);
}
```

Kode 5.2 (Lanjutan) *Quicksort dengan pemartisian Lomuto, yang menarasikan rekursinya. Indentasinya menunjukkan kedalaman pemanggilannya.*

```
int main() {
    A = {2, 8, 7, 1, 3, 5, 6, 4};
    cout << "input: ";
    for (int x : A) cout << x << " ";
    cout << "\n\n";
    quicksort(0, (int)A.size() - 1, 0);
    cout << "\nsorted: ";
    for (int x : A) cout << x << " ";
    cout << "\npartition calls: " << calls << "\n";
    return 0;
}
```

```
input: 2 8 7 1 3 5 6 4

  QUICKSORT(0,7): pivot = A[7] = 4
    [2 1 3 4 7 5 6 8]  after PARTITION, pivot 4 is final at index 3
    QUICKSORT(0,2): pivot = A[2] = 3
      [2 1 3]4 7 5 6 8   after PARTITION, pivot 3 is final at index 2
      QUICKSORT(0,1): pivot = A[1] = 1
        [1 2]3 4 7 5 6 8   after PARTITION, pivot 1 is final at index 0
        QUICKSORT(1,1): one element, return
    QUICKSORT(4,7): pivot = A[7] = 8
      1 2 3 4 [7 5 6 8]  after PARTITION, pivot 8 is final at index 7
      QUICKSORT(4,6): pivot = A[6] = 6
        1 2 3 4 [5 6 7]8   after PARTITION, pivot 6 is final at index 5
        QUICKSORT(4,4): one element, return
        QUICKSORT(6,6): one element, return

sorted: 1 2 3 4 5 6 7 8
partition calls: 5
```

Lima pemartisian menempatkan lima pivot secara permanen; tiga elemen lainnya jatuh sebagai kasus dasar satu elemen. Pivot tidak pernah berpindah lagi. Buku: Subbab 5.8.8.

Segalanya Bergantung pada Belahannya

`PARTITION` berongkos $\Theta(n)$ apa pun yang terjadi. Dengan k elemen di kiri, rekurensi umumnya adalah Persamaan (5.10):

$$T(n) = T(k) + T(n-k-1) + \Theta(n), \tag{5.10}$$

Kasus terbaik, pivot selalu median, menghasilkan Persamaan (5.11) — rekurensi Merge Sort; Kasus 2 Teorema Master memberi $\Theta(n \lg n)$.

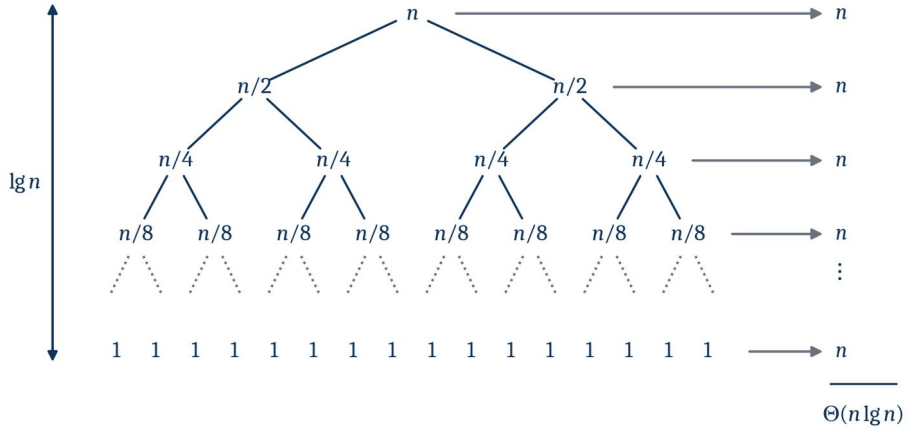
$$T(n) = 2T(n/2) + \Theta(n). \tag{5.11}$$

Kasus terburuk, $k = 0$ setiap kali, menghasilkan Persamaan (5.12), yang bila diurai menjadi $c \cdot n(n+1)/2 = \Theta(n^2)$.

$$T(n) = T(0) + T(n-1) + \Theta(n) = T(n-1) + \Theta(n), \tag{5.12}$$

Substitusi dengan tebakan $T(n) \le c\,n^2$ memastikan kasus terburuk tidak pernah lebih buruk daripada $\Theta(n^2)$. Buku: Subbab 5.9.1–5.9.3.

Kasus Terbaik: Setiap Level Berongkos n



Gambar 5.18 Pohon rekursi kasus terbaik bagi Quicksort. Ukuran submasalah tiap tingkat membelah dua — n, lalu $n/2$ dua kali, lalu $n/4$ empat kali — tetapi pekerjaan *total* pada tiap tingkatnya tetap n, sebab memartisi 2^d subarray berukuran $n/2^d$ berongkos $2^d \times n/2^d = n$. Ada $\lg n + 1$ tingkat, jadi totalnya $\Theta(n \lg n)$.

Gambar 5.18: 2^d sublarik berukuran $n/2^d$ berongkos n per level, sebanyak $\lg n + 1$ level. Gambar yang sama menjelaskan setiap kasus lainnya. Buku: Subbab 5.9.2.

Kasus Terburuk, Diurai

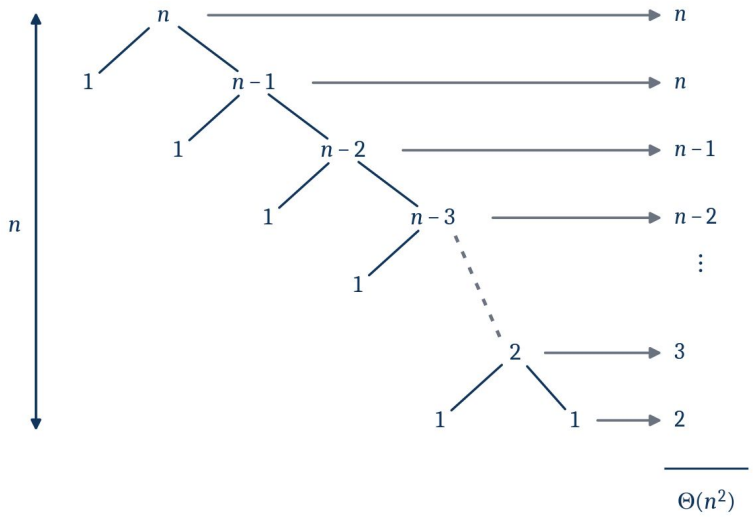
$$
\begin{aligned}
T(n) &= T(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \sum_{k=1}^{n} \Theta(k) \\
&= \Theta\left(\sum_{k=1}^{n} k \right) \\
&= \Theta(n^2)
\end{aligned}
$$

Gambar 5.19 Rekurensi kasus terburuknya dibuka gulungannya:

$$T(n)=T(1)+T(n-1)+\Theta(n)=T(n-1)+\Theta(n)=\sum_{k=1}^{n}\Theta(k)=\Theta\left(\sum_{k=1}^{n}k\right)=\Theta(n^2).$$

Gambar 5.19 memakai deret aritmetika $\sum_{k=1}^{n} k=n(n+1)/2$ *dari Bab 3. Buku: Subbab 5.9.3.*

Kasus Terburuk: Pohon yang Bukan Pohon



Gambar 5.20 Pohon rekursi kasus terburuknya. Alih-alih bercabang, pohonnya merosot menjadi satu rantai tunggal sepanjang n — setiap simpul memperanakkan satu anak berukuran satu lebih kecil dan satu anak berukuran 0. Biaya tingkat-tingkatnya adalah $n, n, n-1, n-2, \ldots, 2$, dan menjumlahkannya memberikan $\Theta(n^2)$.

*Gambar 5.20: dengan pivot elemen terakhir ala Lomuto, hal ini terjadi pada masukan yang **sudah terurut** — kasus terbaik Insertion Sort adalah kasus terburuk Quicksort naif. Buku: Subbab 5.9.3.*

Belahan $1:9$ pun Tidak Mengapa

Belahan yang selalu $10\ \%\ /\ 90\ \%$ menghasilkan Persamaan (5.13):

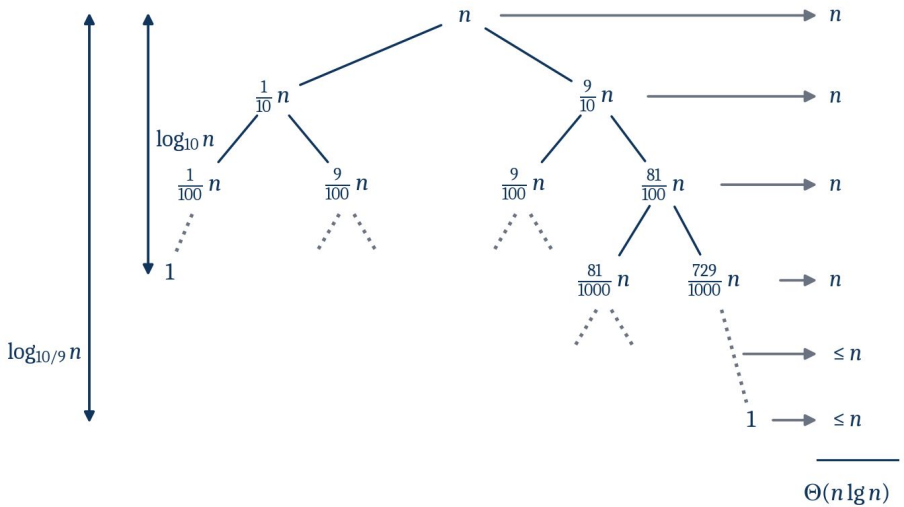
$$T(n) = T(n/10) + T(9n/10) + \Theta(n). \tag{5.13}$$

Setiap level yang lengkap berongkos tepat n. Lintasan terpendek berakhir sesudah $\log_{10} n \approx 0.30 \lg n$ level, yang terpanjang sesudah $\log_{10/9} n \approx 6.58 \lg n$. Keduanya $\Theta(n \lg n)$.

> **FAKTA YANG MENENTUKAN. Setiap** belahan dengan rasio konstan menghasilkan $\Theta(n \lg n)$. Hanya belahan degeneratif $1:(n-1)$ — dengan *banyak* elemen yang tetap, bukan *pecahan* yang tetap, di sisi kecil — yang menghasilkan $\Theta(n^2)$.

Untuk $n = 10^6$: belahan $99:1$ setiap kali berongkos sekitar 1.4×10^9 operasi; belahan degeneratif berongkos 5×10^{11}. Buku: Subbab 5.9.4.

Pohon Rekursi 1:9



Gambar 5.21 Pohon rekursi bagi belahan tetap $1:9$. Tulang punggung kirinya, yang ukurannya menyusut sepersepuluh setiap tingkat, menyentuh dasar sesudah $\log_{10} n$ tingkat. Tulang punggung kanannya, yang ukurannya hanya menyusut dengan faktor $10/9$, bertahan sampai kedalaman $\log_{10/9} n$. Di antara kedua kedalaman itu, beberapa tingkat tidak lengkap dan berongkos kurang dari n; di atas kedalaman yang lebih dangkal, setiap tingkat berongkos tepat n. Totalnya karena itu berada di antara $n \log_{10} n$ dan $n \log_{10/9} n$, dan keduanya $\Theta(n \lg n)$.

Gambar 5.21: ketimpangan hanya berongkos faktor konstan sekitar dua puluh dua, tidak lebih. Buku: Subbab 5.9.4.

Keberuntungan yang Berselang-seling

$$L(n) = 2U(n/2) + \Theta(n) \qquad \text{beruntung}$$

$$U(n) = L(n-1) + \Theta(n) \qquad \text{sial}$$

maka diperoleh

$$L(n) = 2(L(n/2-1) + \Theta(n/2)) + \Theta(n)$$

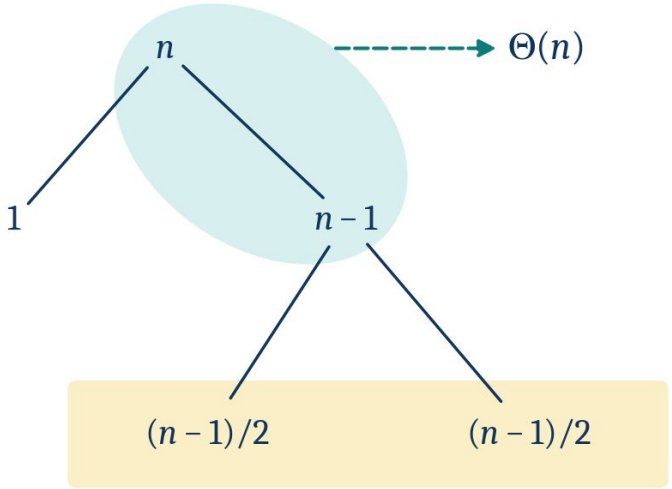
$$= 2L(n/2-1) + \Theta(n)$$

$$= \Theta(n \lg n)$$

Gambar 5.22 Analisis nasib berselang-seling. Dengan $L(n)=2U(n/2)+\Theta(n)$ untuk tingkat yang mujur dan $U(n)=L(n-1)+\Theta(n)$ untuk tingkat yang sial, mensubstitusikan yang kedua ke dalam yang pertama memberikan $L(n)=2(L(n/2-1)+\Theta(n/2))+\Theta(n)=2L(n/2-1)+\Theta(n)=\Theta(n\lg n)$.

Gambar 5.22: biarkan keberuntungan berselang-seling — $L(n)$ untuk level yang beruntung (seimbang), $U(n)$ untuk level yang sial ($1 : n-1$). Menyubstitusikan rekurensi yang satu ke yang lain menghasilkan lagi rekurensi Merge Sort, jadi belahan buruk diserap oleh belahan baik yang menyusulnya. Buku: Subbab 5.9.5.

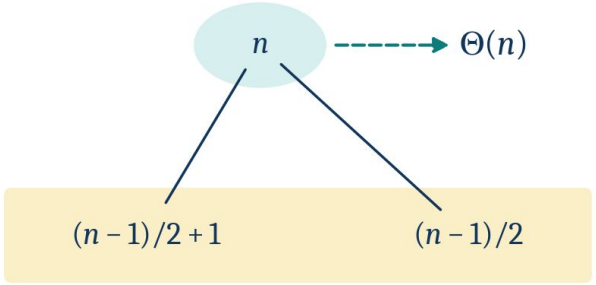
Sial, lalu Beruntung



Gambar 5.23 Sebuah belahan sial disusul sebuah belahan mujur. Pemartisian sial atas n elemen menghasilkan potongan berukuran 1 dan potongan berukuran $n-1$; pemartisian mujur atas potongan itu menghasilkan dua potongan berukuran $(n-1)/2$. Daerah yang diarsir menunjukkan biaya gabungannya, $\Theta(n)+\Theta(n-1)=\Theta(n)$.

Gambar 5.23: dua level kerja $\Theta(n)$ berakhir pada dua bagian berukuran sekitar $(n-1)/2$. Buku: Subbab 5.9.5.

Satu Belahan Beruntung, sebagai Pembanding



Gambar 5.24 Sebuah belahan mujur tunggal, sebagai pembanding: satu pemartisian atas n elemen, berongkos $\Theta(n)$, menghasilkan dua potongan berukuran kira-kira $(n-1)/2$.

Gambar 5.24 berakhir di tempat yang sama dengan Gambar 5.23 dengan ongkos yang sama sampai faktor konstan: belahan yang sial tidak menggandakan waktu jalan, ia hanya menambah satu level. Buku: Subbab 5.9.5.

Nilai Harapan Banyaknya Pembandingan

Namai ulang elemennya $z_1 < z_2 < \cdots < z_n$ dan misalkan $Z_{ij} = \{z_i, \ldots, z_j\}$. **Dua elemen dibandingkan paling banyak sekali** — hanya ketika salah satunya adalah pivot.

z_i dan z_j dibandingkan tepat ketika pivot pertama yang diambil dari Z_{ij} adalah z_i atau z_j. Dengan pivot acak seragam, diperoleh Persamaan (5.14):

$$\Pr\{z_i \text{ dibandingkan dengan } z_j\} = \frac{2}{j - i + 1}. \tag{5.14}$$

Menjumlahkan dengan kelinearan nilai harapan, $E[X] < 2(n-1)H_n = O(n \lg n)$; nilai tepatnya adalah Persamaan (5.15):

$$E[X] = 2(n+1)H_n - 4n \approx 2n \ln n - 2.85n \approx 1.386 n \lg n. \tag{5.15}$$

Diukur pada $n = 4000$: masukan terurut dengan pivot tetap membuat tepat $n(n-1)/2 = 7,998,000$ pembandingan; dengan pivot acak, $57,639$ — terhadap prediksi $54,989$. Buku: Subbab 5.9.5 dan 5.9.6.

Quicksort Teracak: Dua Baris

Pseudokode 5.12 `RANDOMIZED-PARTITION` dan `RANDOMIZED-QUICKSORT`: tukarkan sebuah elemen acak seragam ke slot *pivot*, lalu partisikan seperti biasa.

```
RANDOMIZED-PARTITION(A, p, r):
 1.  i ← RANDOM(p, r)                    // indeks acak seragam dalam [p, r]
 2.  swap A[r] ↔ A[i]
 3.  return PARTITION(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r):
 1.  if p < r
 2.      q ← RANDOMIZED-PARTITION(A, p, r)
 3.      RANDOMIZED-QUICKSORT(A, p, q − 1)
 4.      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Pseudokode 5.12 membuat waktu jalan tidak bergantung pada urutan masukan: tidak ada masukan tertentu yang memicu kasus terburuk. Kasus terburuk masih dapat terjadi — karena nasib sial — dengan peluang sekitar Persamaan (5.16), kira-kira 10^{-128} untuk $n = 100$.

$$\frac{2}{n} \cdot \frac{2}{n-1} \cdots \frac{2}{2} = \frac{2^{n-1}}{n!}. \tag{5.16}$$

Pertahanan lain: median-dari-tiga, dua pivot (Java), **Introsort** (`std::sort` di C++: Quicksort yang beralih ke Heapsort bila kedalaman melampaui $2 \lg n$, diakhiri Insertion Sort), dan pemartisian tiga arah untuk banyak kunci kembar.

Kita tidak membuat algoritmanya lebih cepat; kita membuatnya dapat diramalkan. Buku: Subbab 5.10.1–5.10.3.

Berekursilah pada Sisi yang Lebih Kecil

Quicksort naif pada masukan terurut berekursi sedalam n dan dapat menghabiskan tumpukan panggilan sebelum waktu $\Theta(n^2)$-nya sempat terasa.

Pseudokode 5.13 `QUICKSORT-TAIL`: berekursi hanya pada potongan yang lebih kecil dan menggelung pada yang lebih besar, sehingga kedalaman tumpukannya tetap $O(\lg n)$.

```
QUICKSORT-TAIL(A, p, r):
 1.  while p < r
 2.      q ← PARTITION(A, p, r)
 3.      if (q − p) < (r − q)                    // potongan KIRI yang lebih kecil
 4.          QUICKSORT-TAIL(A, p, q − 1)         //   berekursi padanya ...
 5.          p ← q + 1                           //   ... lalu menggelung ke kanan
 6.      else                                    // potongan KANAN yang lebih kecil
 7.          QUICKSORT-TAIL(A, q + 1, r)         //   berekursi padanya ...
 8.          r ← q − 1                           //   ... lalu menggelung ke kiri
```

TEOREMA. Pseudokode 5.13 memakai ruang tumpukan $O(\lg n)$. *Bukti.* Setiap panggilan rekursif sungguhan dilakukan pada bagian yang lebih kecil, paling besar separuh ukuran pemanggilnya, sehingga kedalaman d memenuhi $n/2^d \geq 1$, yaitu $d \leq \lg n$. ∎

Diukur pada $n = 64,000$: kedalaman naif $64,000$ pada masukan terurut, kedalaman versi rekursi ekor 2; pada masukan acak 39 lawan $11 \leq \lg n = 15$. Buku: Subbab 5.10.4.

Lantai $\Omega(n \lg n)$

Sebuah **pengurutan berbasis pembandingan** mengenal masukannya hanya lewat pertanyaan "apakah $a_i \le a_j$?". Untuk n yang tetap, jalannya adalah sebuah **pohon keputusan**: simpul internalnya pembandingan, daunnya permutasi keluaran.

Ia memerlukan paling sedikit $n!$ daun (satu untuk setiap urutan), dan pohon bertinggi h memiliki paling banyak 2^h daun — Persamaan (5.17):

$$n! \le \ell \le 2^h. \tag{5.17}$$

Jadi $h \ge \lg(n!)$, dan rumus Stirling menghasilkan Persamaan (5.18):

$$\lg(n!) \ge \lg\left(\left(\frac{n}{e}\right)^n\right) = n\lg n - n\lg e = n\lg n - 1.443\,n = \Omega(n\lg n). \tag{5.18}$$

Counting Sort dan Radix Sort mengalahkan batas ini justru karena bukan pengurutan berbasis pembandingan. Untuk $n=3$: $\lg 6 = 2.585$, jadi diperlukan tiga pembandingan. Buku: Subbab 5.11.

Memilih Pengurutan: Perbandingan yang Jujur

Tabel 5.7 Setiap algoritma pengurutan dalam buku ini sejauh ini, dengan batas Θ atas banyaknya pembandingan (kasus terbaik Heapsort mengandaikan kunci yang berbeda-beda). Kolom ruangnya mencacah memori tambahan di luar larik masukannya.

Algoritma	Terburuk	Rata-rata	Terbaik	Ruang tambahan	Stabil?	Di tempat?	CLRS	Bab
Insertion Sort	n^2	n^2	n	1	Ya	Ya	Bab 2	2
Selection Sort	n^2	n^2	n^2	1	Tidak	Ya	Bab 2	2
Bubble Sort	n^2	n^2	n	1	Ya	Ya	Bab 2	2
Merge Sort	$n \lg n$	$n \lg n$	$n \lg n$	n	Ya	Tidak	Bab 2	3
Heapsort	$n \lg n$	$n \lg n$	$n \lg n$	1	Tidak	Ya	Bab 6	5
Quicksort	n^2	$n \lg n$	$n \lg n$	$\lg n$	Tidak	Ya	Bab 7	5

Kestabilan (lihat Tabel 5.7): Insertion Sort dan Merge Sort mempertahankan `1b 1e`, `2d 2g`, `3a 3c 3f`; Heapsort membalik setiap kelompok; Quicksort memberi `3c 3f 3a`. Pemecah seri berupa indeks asal membuat pengurutan apa pun menjadi stabil.

Cache: Quicksort memindai memori secara berurutan; Heapsort melompat $i \to 2i \to 4i$. Diukur pada $n = 10^6$: Quicksort 66.79 ms, Merge Sort 97.02 ms, Heapsort 122.72 ms — padahal pada $n = 10^5$, yang muat di *cache*, Heapsort yang tercepat.

Pakailah pengurutan dari pustaka kecuali ada alasan untuk tidak; Heapsort bila perlu jaminan, Merge Sort bila perlu kestabilan, Insertion Sort untuk larik mungil atau hampir terurut, dan heap untuk "yang terbesar yang tersisa". Buku: Subbab 5.12.

Profitina dalam Praktik: Antrean Laundry

Antrean perayapan Profitina adalah antrean prioritas maks yang berkunci skor urgensi (`EXTRACT-MAX`, `INCREASE-KEY`, `INSERT`); penjadwal pemindaian ulangnya antrean prioritas min; dasbornya mengurutkan.

Profitina Laundry menjawab “cucian mana berikutnya?” dengan skor Persamaan (5.19):

$$\text{urgency}(o) = (48 - \text{dueHour}) \times 10 + 40 \cdot [o \,\text{kilat}] - \text{kg}. \tag{5.19}$$

`PL-0344` jatuh tempo pukul 13:00, paling awal, tetapi dicuci **ketiga**: pesanan ekspres `PL-0342` bernilai $340 + 40 - 3 = 377 > 346$. Satu jam bernilai 10 poin dan ekspres bernilai 40.

Heap tidak memutuskan apa pun; fungsi skorlah yang memutuskan. Pesanan membawa kode tiket anonim, tidak pernah nama (UU PDP 27/2022). Aplikasi memberi saran; pengguna yang memutuskan. Buku: Subbab 5.13.

Kesalahan yang Sering Terjadi

Mengira *heap* itu terurut, atau mencampuradukkan `heap-size` dengan panjang larik.

Membaca $A[l]$ sebelum uji batas, atau membandingkan anak kanan dengan $A[i]$, bukan dengan $A[\text{largest}]$.

Membangun dengan n kali penyisipan ($O(n \lg n)$) alih-alih `BUILD-MAX-HEAP` ($O(n)$); menjalankan gelung pembangunan secara naik.

Batas rekursi yang tidak cocok — Lomuto $(p, q-1), (q+1, r)$; Hoare $(p, q), (q+1, r)$ dengan $x = A[p]$.

Pivot tetap pada data nyata; melupakan tumpukan panggilan; mengira pengacakan menghapus kasus terburuk; mengira Quicksort atau Heapsort stabil; menghitung dengan `int`, bukan `long long`.

Buku memuat lima belas kesalahan, masing-masing dengan perbaikannya. Buku: Subbab 5.14.

Ringkasan: Lima Hal untuk Diingat

1. *Heap* adalah pohon hampir lengkap di dalam larik: $Parent(i) = \lfloor i/2 \rfloor$, anak $2i$ dan $2i+1$, tinggi $\lfloor \lg n \rfloor$; sifat *max-heap* bersifat lokal, dan $A[1]$ adalah maksimum.

2. `MAX-HEAPIFY` berongkos $O(\lg n)$; `BUILD-MAX-HEAP` $\Theta(n)$ karena $\sum_{h \geq 0} h/2^h = 2$; Heapsort $\Theta(n \lg n)$, di tempat, tidak stabil.

3. Antrean prioritas berbasis *heap* mengerjakan `MAXIMUM` dalam $O(1)$ dan tiga operasi lainnya dalam $O(\lg n)$.

4. Ongkos Quicksort bergantung pada belahannya: $\Theta(n \lg n)$ untuk rasio konstan apa pun, $\Theta(n^2)$ hanya untuk $1 : (n-1)$; dengan pivot acak, nilai harapannya $\approx 1.39\, n \lg n$ pembandingan.

5. Tidak ada pengurutan berbasis pembandingan yang mengalahkan $\Omega(n \lg n)$; di antara yang optimal, perilaku *cache* dan jaminanlah yang menentukan.

Mengganti struktur data dapat mengubah kelas kompleksitas: Selection Sort dan Heapsort adalah algoritma yang sama. Buku: Subbab 5.15.

Pertemuan Berikutnya

Bab 6 — Pohon Pencarian Biner dan Pohon AVL. *Heap* hanya menemukan maksimum; pohon pencarian menemukan kunci *apa saja*.

Pencarian, penyisipan, dan penghapusan dalam $O(h)$ — serta rotasi AVL yang menjaga $h = O(\lg n)$.

Sebelum itu: kerjakan Soal Latihan 5.5, 5.6, 5.11, dan 5.14 dengan tangan.

Bab 11 dan Bab 13 akan memetik hasil antrean prioritas hari ini: Prim dan Dijkstra adalah Subbab 5.7 yang dibungkuskan pada sebuah graf. Buku: Subbab 5.15.1 dan 5.16.

Kuliah 6 — Pohon Pencarian Biner dan Pohon AVL

Perancangan dan Analisis Algoritma (EF234405)

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*, Bab 6

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini: sebuah struktur yang menjaga kunci tetap terurut sementara kunci ditambah dan dibuang sesuka hati — dan sebuah pohon yang memperbaiki bentuknya sendiri sesudah setiap perubahan. Buku: Bab 6, Subbab 6.1–6.24.

Yang Dibahas Pertemuan Ini

1. **Sifat pohon pencarian biner** — dan mengapa ia bukan sifat *heap*.
2. **Penelusuran dan kueri** — inorder itu terurut (beserta buktinya); pencarian, minimum, maksimum, penerus, pendahulu, semuanya $O(h)$.
3. **Penyisipan dan tiga kasus penghapusan**, ditelusuri di papan tulis.
4. **Ketika h mencapai n** — dan obat AVL: invariannya, serta bukti Fibonacci untuk $h \leq 1.44 \lg n$.
5. **Rotasi dan empat kasus** — contoh penyisipan lima langkah dan contoh penghapusan.
6. **Memilih** — pohon AVL, pohon merah-hitam, atau tabel hash; papan pengambilan Profitina Laundry.

Setiap operasi hari ini berongkos $O(h)$; paruh kedua membahas satu pertanyaan: bagaimana mencegah h tumbuh? Buku: Subbab 6.1.

Dari Mana Kita Melanjutkan

Bab 3 memberi kita notasi asimtotik dan rekurensi. Hari ini kita memakainya pada sebuah *struktur data*, yang operasi-operasinya saling terkait: cepat-lambatnya satu operasi bergantung pada bentuk yang ditinggalkan operasi lainnya.

Larik terurut mencari dalam $\Theta(\lg n)$ tetapi menyisipkan dalam $\Theta(n)$. **Senarai berantai** menyambung dalam $\Theta(1)$ tetapi menemukan tempatnya dalam $\Theta(n)$.

Pohon pencarian biner menolak memilih: setiap pembandingan membuang satu sisi utuh, seperti pencarian biner, dan penyisipan hanya mengubah dua atau tiga penunjuk, seperti pada senarai.

Koleksi yang terus-menerus dicari dan terus-menerus diubah memerlukan keduanya sekaligus. Buku: Subbab 6.1.

Kosakata Pohon

Akar, induk, anak, daun, simpul internal, saudara; leluhur dan keturunan; **subpohon berakar di** x adalah x beserta semua keturunannya. **Kedalaman** menghitung sisi dari akar; **tinggi** menghitung sisi turun sampai daun terdalam.

> **PANCANGKAN KESEPAKATANNYA.** Satu simpul bertinggi 0 dan pohon kosong bertinggi -1. Sebagian gambar menghitung *simpul* (daun bertinggi 1); kedua kesepakatan berselisih satu di mana-mana, dan faktor keseimbangan — yang berupa selisih — tetap sama.

Rantai memberi $h \le n-1$; pohon penuh memuat paling banyak $2^{h+1}-1$ simpul, sehingga $h \ge \lceil \lg(n+1) \rceil - 1$. Bersama-sama keduanya memberi Persamaan (6.1):

$$\lceil \lg(n+1) \rceil - 1 \le h \le n-1 . \tag{6.1}$$

Persamaan (6.1) adalah seluruh bab dalam satu baris: ujung kirinya sekitar 20 *langkah untuk sejuta kunci, ujung kanannya sejuta langkah. Buku: Subbab 6.2.*

Sifat Pohon Pencarian Biner

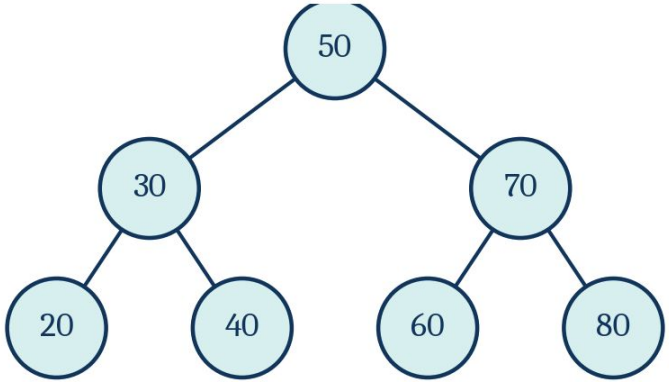
> **DEFINISI.** Sebuah pohon biner adalah **pohon pencarian biner** bila, untuk setiap simpul x, setiap kunci di subpohon kiri x **lebih kecil** daripada kunci x, dan setiap kunci di subpohon kanannya **lebih besar**. Tidak ada dua simpul yang memuat kunci yang sama.

Kata yang bekerja keras adalah **setiap** dan **subpohon**: bukan hanya anak-anaknya, tetapi setiap kunci di bawahnya pada sisi itu.

Jebakan klasik. Ganti 40 dengan 55 pada pohon di halaman berikut: simpul 30 masih lolos (55>30), tetapi simpul 50 gagal, karena 55 duduk di subpohon kirinya. Sebuah pohon dapat lolos setiap uji induk–anak dan tetap bukan pohon pencarian biner.

Mengapa berguna: satu pembandingan di x membuang satu sisi utuh tanpa melihatnya. Itulah pencarian biner yang dijalankan pada penunjuk. Buku: Subbab 6.3.

Sebuah Pohon Pencarian Biner



Gambar 6.1 Sebuah pohon pencarian biner. Akarnya adalah 50; segala isi subpohon kiri $(30, 20, 40)$ berada di bawah 50, dan segala isi subpohon kanan $(70, 60, 80)$ berada di atasnya. Uji yang sama lolos di setiap simpul lainnya.

Gambar 6.1, diperiksa: di 50, $\lfloor 30, 20, 40 \rfloor < 50 < \lfloor 70, 60, 80 \rfloor$; *di* 30, $20 < 30 < 40$; *di* 70, $60 < 70 < 80$; *daun selalu lolos. Buku: Subbab 6.3.*

Sifat Pohon Pencarian Biner ≠ Sifat Heap

Max-heap hanya menuntut induk ≥ anak — **vertikal dan lokal**. Sifat pohon pencarian biner Persamaan (6.2) bersifat **horizontal dan global**:

$$\text{setiap kunci di kiri}(x) < \text{kunci}(x) < \text{setiap kunci di kanan}(x). \tag{6.2}$$

Tabel 6.1 Sifat pohon pencarian biner dibandingkan sifat tumpukan-maks. Keduanya aturan yang berbeda, keduanya mendukung operasi yang berbeda, dan keduanya disimpan dengan cara yang berbeda.

Pertanyaan	Pohon pencarian biner	Tumpukan-maks
Aturannya	subpohon kiri $<$ simpul $<$ subpohon kanan	induk $\geq$ tiap anak
Jangkauan aturannya	semua keturunan	anak langsung saja
Apakah kiri/kanan penting?	ya, keduanya bermakna berlawanan	tidak, kedua anak bisa dipertukarkan
Di mana kunci terkecilnya?	simpul paling kiri, sejauh $O(h)$	entah di mana di antara simpul daun, $O(n)$ untuk menemukannya
Di mana kunci terbesarnya?	simpul paling kanan, sejauh $O(h)$	di akar, $O(1)$
Biaya mencari kunci sembarang k	$O(h)$	$\Theta(n)$ — tidak lebih baik daripada memindai
Apa yang dicetak penelusuran inorder?	kuncinya dalam urutan terurut	tidak ada yang bermakna
Bentuk pohonnya	apa pun yang dihasilkan urutan penyisipan	selalu pohon lengkap, tinggi tepat $\lfloor \lg n \rfloor$
Penyimpanan yang lazim	simpul dan penunjuk	larik biasa, tanpa penunjuk sama sekali

Tabel 6.1: heap untuk berulang kali mengambil nilai ekstrem; pohon pencarian untuk mencari. Salah memilih berongkos faktor n. Buku: Subbab 6.4.

Simpulnya dalam C++

Kode 6.1 mendeklarasikan simpul — satu kunci dan dua penunjuk — lalu merangkai ketujuh simpul Gambar 6.1 dengan tangan.

Kode 6.1 Simpul sebuah pohon pencarian biner, dan pohon Gambar 6.1 yang dirakit dengan tangan dari tujuh simpul. Kompilasi dengan `g++ -std=c++17 -O2 -o node node.cpp`.

```cpp
// Simpul pohon pencarian biner, dan pohon pada gambar pertama yang dirakit dengan tangan.
#include <iostream>

struct Node {
    int   key;
    Node* left;
    Node* right;
    Node(int k) : key(k), left(nullptr), right(nullptr) {}
};

void inorder(const Node* x) {                  // subpohon kiri, simpul, subpohon kanan
    if (!x) return;                            // pohon kosong: tidak ada yang dicetak
    inorder(x->left);
    std::cout << x->key << " ";
    inorder(x->right);
}

int main() {
    Node* root = new Node(50);                 // seluruh pohon adalah penunjuk ke akarnya
    root->left  = new Node(30);
    root->right = new Node(70);
    root->left->left   = new Node(20);
    root->left->right  = new Node(40);
    root->right->left  = new Node(60);
    root->right->right = new Node(80);

    std::cout << "root " << root->key << ", left child " << root->left->key
              << ", right child " << root->right->key << "\n";
```

```
    std::cout << "left child of 20 is " << (root->left->left->left ? "a node" : "nullptr")
              << " (the empty tree)\n";
    std::cout << "inorder: ";
    inorder(root);
    std::cout << "\n";

    Node* nodes[] = {root->left->left, root->left->right, root->right->left,
                     root->right->right, root->left, root->right, root};
    for (Node* p : nodes) delete p;            // anak dihapus sebelum induknya
    return 0;
}
```

```
root 50, left child 30, right child 70
left child of 20 is nullptr (the empty tree)
inorder: 20 30 40 50 60 70 80
```

Pohon kosong adalah `nullptr`: tinggi -1*, tanpa kunci, dan kasus dasar hampir setiap rekursi di bab ini. Buku: Subbab 6.5.*

Tiga Penelusuran

DEFINISI. Untuk menelusuri subpohon di x: **inorder** — kiri, x, kanan; **preorder** — x, kiri, kanan; **postorder** — kiri, kanan, x. Pohon kosong tidak melakukan apa-apa.

Pada Gambar 6.1: inorder $20, 30, 40, 50, 60, 70, 80$; preorder $50, 30, 20, 40, 70, 60, 80$; postorder $20, 40, 30, 60, 80, 70, 50$.

Tabel 6.2 Ketiga penelusuran mendalam-dahulu dan pekerjaan yang dikerjakan masing-masingnya.

Penelusuran	Urutan	Gunanya untuk apa
inorder	kiri, simpul, kanan	mendaftar kunci dalam urutan terurut; kueri rentang; memeriksa bahwa sebuah pohon sungguh pohon pencarian biner
preorder	simpul, kiri, kanan	menyalin atau menyerialkan pohon — menyisipkan ulang barisan preorder membangun kembali bentuk yang persis sama
postorder	kiri, kanan, simpul	memusnahkan pohon (bebaskan anak-anaknya sebelum induknya); menghitung nilai di sebuah simpul yang bergantung pada kedua subpohonnya, misalnya tinggi

Tabel 6.2: setiap penelusuran mengunjungi tiap simpul sekali, $\Theta(n)$ apa pun bentuknya — satu-satunya operasi di sini yang ongkosnya tidak bergantung pada h. Buku: Subbab 6.6.

Mengapa Penelusuran Inorder Terurut

Klaim. Penelusuran inorder pada pohon pencarian biner mengeluarkan kuncinya dalam urutan naik tegas. **Bukti** dengan induksi pada n. Untuk $n=0$ keluarannya kosong.

Untuk $n \geq 1$ dengan akar x: subpohon L dan R adalah pohon pencarian biner dengan simpul lebih sedikit, jadi penelusurannya naik menurut hipotesis. Keluarannya: penelusuran(L), kunci(x), penelusuran(R).

Pada sambungannya: setiap kunci L lebih kecil daripada kunci(x), dan setiap kunci R lebih besar. Jadi seluruh keluarannya naik. ▰

> **PENGURUTAN GRATIS.** Sisipkan n kunci, telusuri secara inorder: *tree sort*, $O(nh)$ — $\Theta(n \lg n)$ bila pohonnya tetap pendek, $\Theta(n^2)$ bila tidak.

Tinggi pohon kini menjadi pembeda antara algoritma pengurutan yang unggul dan yang buruk. Buku: Subbab 6.6.1.

Pencarian, Minimum, Maksimum

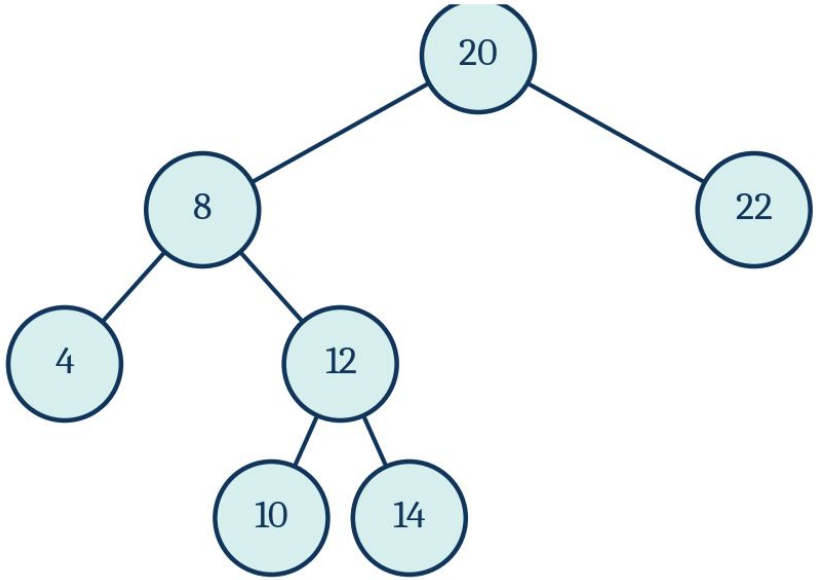
TREE-SEARCH(x, k)**:** kosong — gagal; sama — ketemu; lebih kecil — ke kiri; selain itu ke kanan. Setiap pembandingan menyingkirkan satu sisi utuh.

Pada pohon di halaman berikut: SEARCH(10) mengunjungi $20 \to 8 \to 12 \to 10$ dan berhasil. SEARCH(13) mengunjungi $20 \to 8 \to 12 \to 14$ lalu jatuh sebagai anak kiri 14 — tepat di tempat 13 seharusnya berada.

MINIMUM: terus ke kiri $(20 \to 8 \to 4)$. **MAXIMUM:** terus ke kanan $(20 \to 22)$. Minimum tidak harus berupa daun.

Setiap langkah turun satu level, jadi ketiganya berongkos $O(h)$*. Buku: Subbab 6.7 dan 6.8.*

Sebuah Pohon untuk Mencari Penerus



Gambar 6.2 Sebuah pohon untuk berlatih mencari penerus inorder. Menelusurinya secara inorder memberi $4, 8, 10, 12, 14, 20, 22$, dan membaca daftar itu adalah cara termudah untuk memeriksa klaim penerus atau pendahulu mana pun.

Gambar 6.2: penelusuran inorder $4, 8, 10, 12, 14, 20, 22$ *adalah cara termudah memeriksa klaim penerus atau pendahulu mana pun. Buku: Subbab 6.7–6.9.*

Penerus: Dua Kasus

Kasus 1 — x punya subpohon kanan: penerusnya adalah minimum subpohon itu. Untuk $x=8$: $12 \to 10$, jadi penerusnya 10 ✓.

Kasus 2 — tanpa subpohon kanan: naiklah; selama Anda datang dari anak kanan, teruslah naik; induk pertama yang dicapai dari anak kiri adalah penerusnya. Untuk $x=14$: $14 \to 12 \to 8$ (dari kanan), lalu $8 \to 20$ (dari kiri): 20 ✓. Untuk $x=22$: keluar dari puncak — tidak punya penerus.

Kasus 2 perlu bergerak ke atas: tambahkan penunjuk `parent`, atau bawa kandidat terbaik turun dari akar selama pencarian biasa.

Pendahulu adalah cerminannya. Bagaimanapun, satu lintasan, turun atau naik: $O(h)$. Buku: Subbab 6.9.

Penyisipan: Pencarian yang Gagal, lalu Sebuah Daun

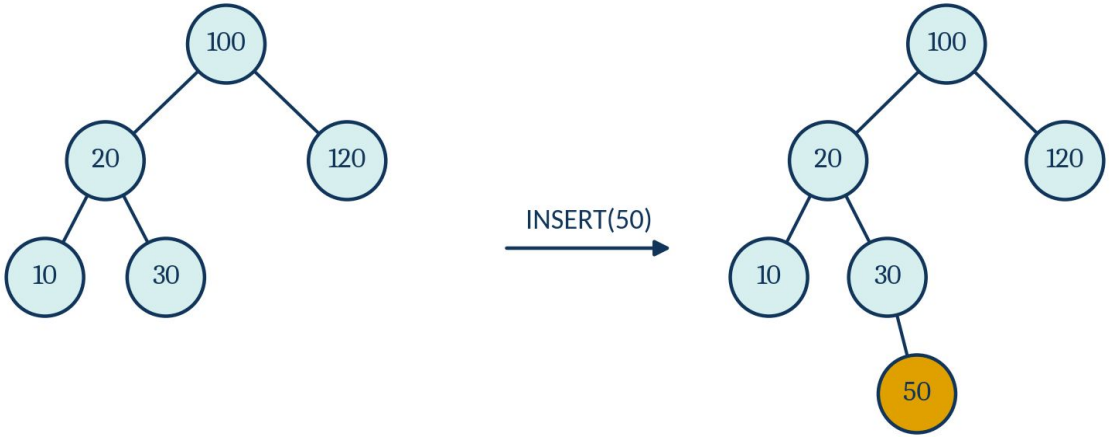
TREE-INSERT(T, k)**:** turun seperti SEARCH; ketika anak yang dituju kosong, gantungkan simpul baru berisi k di sana. Simpul baru **selalu berupa daun**.

`Insert(50)` di halaman berikut: 50<100, ke kiri; 50>20, ke kanan; 50>30, ke kanan; kosong — 50 menjadi anak kanan 30.

Periksa: $\lfloor 50 \rfloor > 30$ ✓, $\lfloor 30, 50 \rfloor > 20$ ✓, $\lfloor 20, 10, 30, 50 \rfloor < 100$ ✓. Sifatnya bertahan karena lintasannya dipilih oleh pembandingan yang sama.

Ongkos $O(h)$*. Urutan itu penting:* $1, 2, 3, 4, 5$ *menghasilkan rantai setinggi* 4*;* $3, 1, 4, 2, 5$ *menghasilkan tinggi* 2*. Buku: Subbab 6.10.*

`Insert(50)`



Gambar 6.3 Sebuah contoh penyisipan. Kunci 50 dijahitkan turun menyusuri pohon — ke kiri di 100, ke kanan di 20, ke kanan di 30 — dan mendarat sebagai anak kanan baru milik 30, ditandai dengan warna kuning tua. Tidak ada hal lain di pohon itu yang bergeser.

Gambar 6.3: tidak ada bagian lain pohon yang berpindah; kita hanya menempelkan. Buku: Subbab 6.10.

Penghapusan: Tiga Kasus

Tabel 6.3 Ketiga kasus penghapusan pada pohon pencarian biner, dan kerja yang dituntut masing-masingnya di luar pencarian awal.

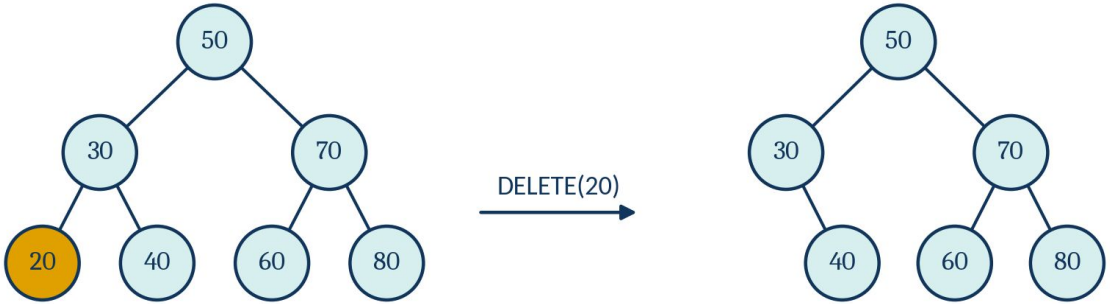
Kasus	Anak simpul yang hendak dilenyapkan	Apa yang dilakukan	Biaya
1	tidak punya	lepaskan ia dari induknya lalu bebaskan	$O(1)$ sesudah pencariannya
2	tepat satu	naikkan seluruh subpohon anak itu ke tempatnya	$O(1)$ sesudah pencariannya
3	dua	salin kunci penerus inordernya ke dalamnya, lalu hapus simpul penerus itu (yang merupakan kasus 1 atau 2)	satu penyusuran $O(h)$ tambahan

Tabel 6.3: kasusnya ditentukan oleh banyaknya anak simpul yang akan dihapus — nol, satu, atau dua.

Tiga halaman berikut menghapus 20, lalu 30, lalu 50 dari pohon berakar 50, beranak 30 dan 70, dengan daun $20, 40, 60, 80$.

Menemukan simpulnya berongkos $O(h)$; kasus 3 menambah satu penurunan $O(h)$ lagi. Penghapusan secara keseluruhan $O(h)$. Buku: Subbab 6.11.

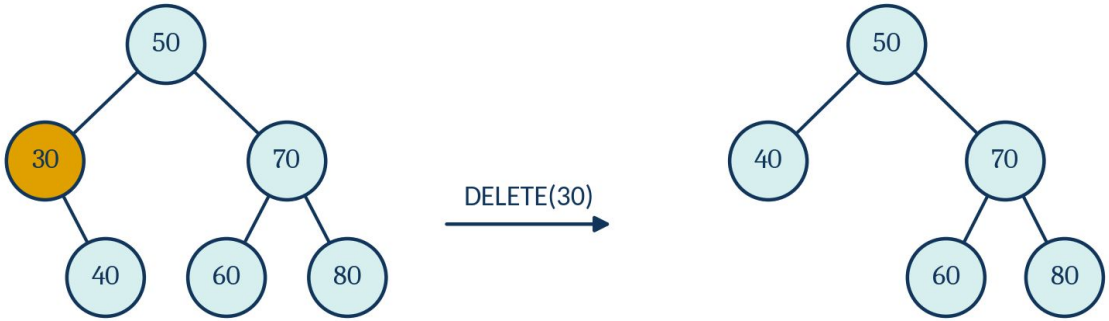
Kasus 1 — Sebuah Daun



Gambar 6.4 Kasus 1 penghapusan: simpul yang hendak dihapus, 20, adalah simpul daun. Ia cukup dibuang; sisa pohonnya tidak tersentuh.

Gambar 6.4: cari $50 \rightarrow 30 \rightarrow 20$*; 20 tidak beranak; kosongkan anak kiri* 30*. Inorder:* $30, 40, 50, 60, 70, 80$*. Buku: Subbab 6.11.1.*

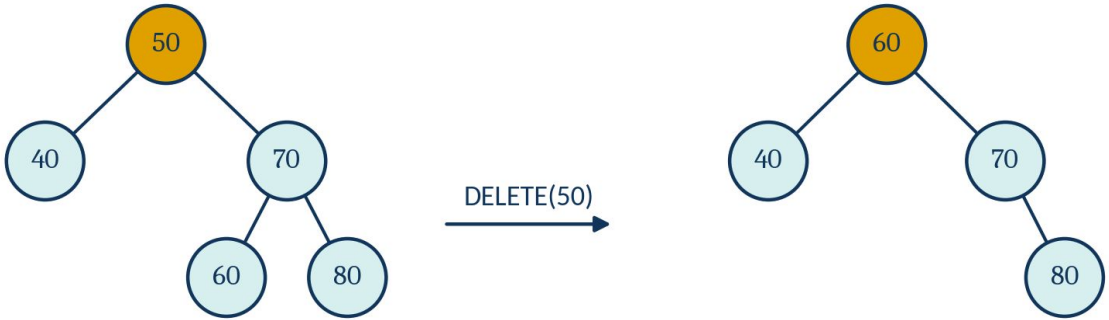
Kasus 2 — Satu Anak



Gambar 6.5 Kasus 2 penghapusan: simpul 30 hanya punya anak kanan, yaitu 40. Menghapus 30 menaikkan 40 ke tempatnya.

Gambar 6.5: 30 hanya punya anak kanan 40; naikkan 40 ke tempat 30. Setiap kunci di subpohon yang dinaikkan sudah berada di sisi yang benar terhadap setiap leluhurnya. Inorder: 40 , 50 , 60 , 70 , 80. Buku: Subbab 6.11.2.

Kasus 3 — Dua Anak



Gambar 6.6 Kasus 3 penghapusan: simpul 50 punya dua anak. Penerus inordernya adalah 60, kunci terkecil di subpohon kanannya. Nilai 60 disalin naik ke akar dan simpul 60 yang asli lalu dihapus, yang merupakan penghapusan kasus 1.

Gambar 6.6: penerus 50 *adalah* 70 → 60*; salin* 60 *ke akar, lalu hapus daun* 60 *yang lama (kasus 1). Inorder:* 40 , 60 , 70 , 80*. Buku: Subbab 6.11.3.*

Mengapa Kasus 3 Sah, dan Mengapa Ia Berhenti

Sah. Kunci subpohon kiri $< \text{key}(x) < \text{key}(s)$; dan $\text{key}(s)$ adalah yang terkecil di antara kunci subpohon kanan, sehingga semua kunci kanan yang tersisa lebih besar daripadanya. Sifatnya berlaku di x.

Berhenti. Penerusnya adalah simpul paling kiri di subpohon kanan, jadi ia tidak punya anak kiri: menghapusnya adalah kasus 1 atau kasus 2, tidak pernah kasus 3 lagi.

Pendahulu juga bisa (maksimum subpohon kiri, tanpa anak kanan); di sini ia akan menaikkan 40 alih-alih 60.

Dalam kode, kasus 3 menyalin kunci, bukan simpul: data satelitnya harus ikut berpindah. Buku: Subbab 6.11.3 dan 6.12.

Ketika Pohonnya Rusak: h Bisa Mencapai n

Sisipkan $10, 20, 30, 40, 50$ dengan urutan itu: setiap kunci ke kanan, dan hasilnya rantai setinggi 4 — senarai berantai yang berbaju pohon.

Masukan terurut adalah masukan yang paling umum: tanggal, nomor faktur, cap waktu, kunci primer.

Diukur pada $n = 16,000$: masukan terurut menghasilkan tinggi $15,999$ dan 7999.5 penjajakan per penyisipan; masukan acak menghasilkan tinggi 32 dan 16.03 penjajakan. Selisih sekitar **lima ratus kali**, hanya karena urutan kedatangan.

Urutan penyisipan acak memberi tinggi harapan $\Theta(\lg n)$, tetapi data nyata jarang acak. Kita menginginkan jaminan, bukan harapan. Buku: Subbab 6.13.

Pohon AVL: Keseimbangan sebagai Invarian

DEFINISI. Faktor keseimbangan sebuah simpul adalah Persamaan (6.3), dengan subpohon kosong bertinggi -1. **Pohon AVL** adalah pohon pencarian biner yang memenuhi Persamaan (6.4).

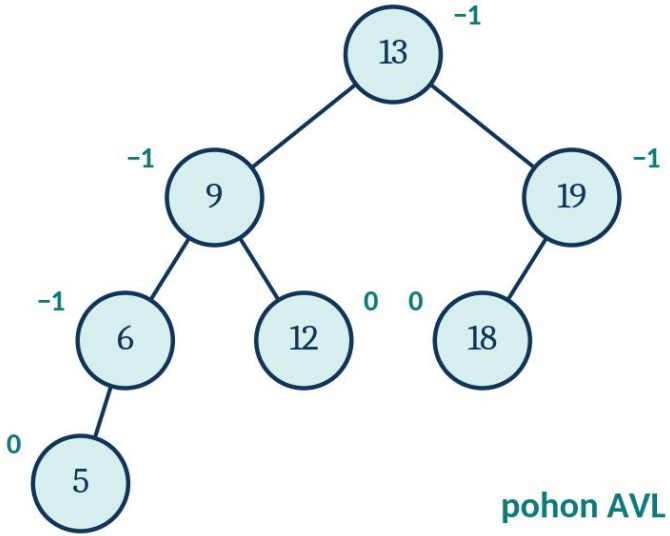
$$bf(x) = h(\text{kanan}(x)) - h(\text{kiri}(x)), \tag{6.3}$$

$$bf(x) \in \{-1, 0, +1\} \text{ untuk setiap simpul } x. \tag{6.4}$$

Negatif berarti **berat kiri**, positif **berat kanan**. Invariannya menyangkut tinggi, bukan banyaknya simpul: pohon AVL menyeimbangkan *kedalaman*, bukan populasi.

Adelson-Velsky dan Landis, 1962: sesudah setiap perubahan, temukan simpul yang timpang dan luruskan dengan penataan lokal $O(1)$. Buku: Subbab 6.14.

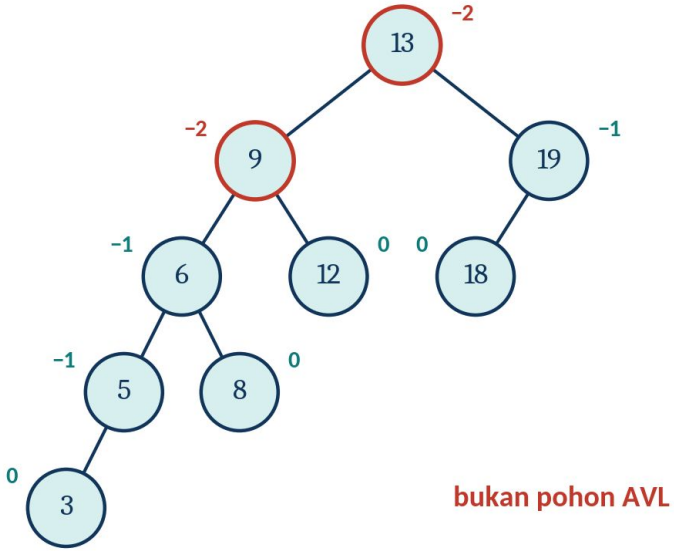
Sebuah Pohon AVL



Gambar 6.7 Sebuah pohon AVL. Kedua subpohon tiap simpul berbeda tinggi paling banyak satu.

Gambar 6.7: daun $5, 12, 18$ *bernilai* 0*; simpul* 6*:* $-1-0=-1$*; simpul* 9*:* $0-1=-1$*; simpul* 19*:* -1*; akar* 13*:* $1-2=-1$*. Semuanya sah. Buku: Subbab 6.14.*

Dan Satu yang Bukan



Gambar 6.8 Sebuah pohon pencarian biner yang **bukan** pohon AVL. Menambahkan simpul 8 dan 3 mendorong dua faktor keseimbangan keluar dari rentang: milik 9 dan 13, keduanya -2.

Gambar 6.8: dengan 8 *di bawah* 6 *dan* 3 *di bawah* 5*, simpul* 6 *bertinggi* 2*, sehingga simpul* 9 *ber-*$bf = 0 - 2 = -2$*. Masih pohon pencarian biner yang sah — inorder* $3, 5, 6, 8, 9, 12, 13, 18, 19$ *— tetapi bukan AVL. Buku: Subbab 6.14.*

Mengapa Pohon AVL Selalu Pendek

Balikkan pertanyaannya: misalkan $N(h)$ adalah **banyak simpul paling sedikit** pada pohon AVL bertinggi h. $N(0)=1$, $N(1)=2$.

Untuk $h \geq 2$: satu subpohon bertinggi $h-1$; invariannya mengizinkan yang lain sependek $h-2$; buat keduanya minimal dan tambahkan akar. Itulah Persamaan (6.5):

$$N(h)=N(h-1)+N(h-2)+1, N(0)=1, N(1)=2. \tag{6.5}$$

DENGAN TANGAN. $N(2)=2+1+1=4$; $N(3)=4+2+1=7$; $N(4)=12$; $N(5)=20$; $N(6)=20+12+1=33$. Jadi 32 simpul tidak akan pernah membentuk pohon AVL bertinggi 6.

Pohon yang mencapai minimum ini disebut pohon Fibonacci — dan nama itu adalah petunjuk. Buku: Subbab 6.15.

Hubungannya dengan Fibonacci

Misalkan $M(h) = N(h) + 1$: suku $+1$ lenyap dan $M(h) = M(h-1) + M(h-2)$ dengan $M(0) = 2 = F(3)$ dan $M(1) = 3 = F(4)$. Maka berlaku Persamaan (6.6):

$$N(h) + 1 = F(h+3). \tag{6.6}$$

Tabel 6.4 Simpul paling sedikit yang bisa dimuat pohon AVL bertinggi h, dan bilangan Fibonacci yang sama dengannya begitu ditambah satu.

h	$N(h)$	$N(h)+1$	$F(h+3)$
0	1	2	2
1	2	3	3
2	4	5	5
3	7	8	8
4	12	13	13
5	20	21	21
6	33	34	34
7	54	55	55
8	88	89	89
9	143	144	144
10	232	233	233
11	376	377	377
12	609	610	610

Tabel 6.4: $N(4)+1=13=F(7)$ ✓, $N(6)+1=34=F(9)$ ✓. *Buku: Subbab 6.15.1.*

Dari Fibonacci ke $1.44 \lg n$

Rumus Binet Persamaan (6.7); karena $\left|\hat{\varphi}^m\right|/\sqrt{5} < 1/2$, rumus itu memberi Persamaan (6.8):

$$F(m) = \frac{\varphi^m - \hat{\varphi}^m}{\sqrt{5}}, \varphi = \frac{1+\sqrt{5}}{2} \approx 1.61803, \hat{\varphi} = \frac{1-\sqrt{5}}{2} \approx -0.61803. \tag{6.7}$$

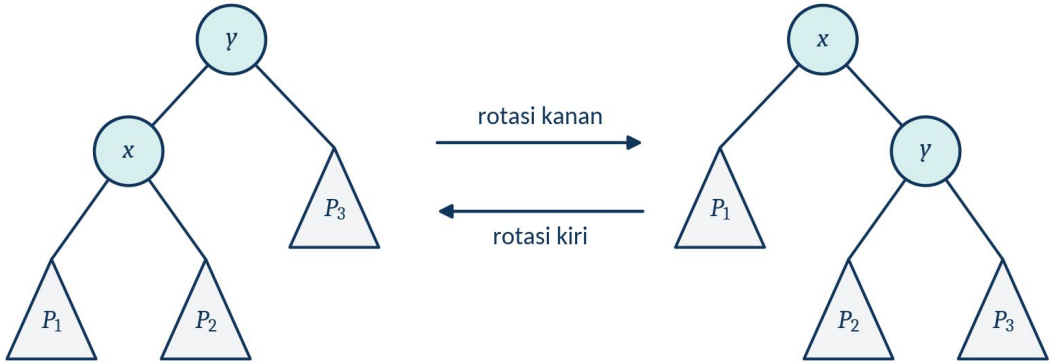
$$F(m) + 1 \geq \frac{\varphi^m}{\sqrt{5}}. \tag{6.8}$$

Lalu $n+2 \geq F(h+3)+1 \geq \varphi^{h+3}/\sqrt{5}$; ambil lg dan bagi dengan $\lg \varphi \approx 0.69424$ untuk memperoleh Persamaan (6.9):

$$h \leq 1.4405 \lg(n+2) - 1.3277 \tag{6.9}$$

Tidak pernah lebih dari sekitar 44% lebih tinggi daripada pohon yang seimbang sempurna: semiliar kunci, paling banyak 41 *level. Versi* -0.3277 *yang sering dikutip menghitung simpul, bukan sisi. Buku: Subbab 6.15.2.*

Rotasi: Satu-satunya Perkakas



bacaan inorder kedua pohon: $\text{kunci}(P_1) < x < \text{kunci}(P_2) < y < \text{kunci}(P_3)$

Gambar 6.9 Kedua rotasi. Kelima benda yang sama — simpul x dan y serta subpohon P_1, P_2, P_3 — muncul di kedua gambarnya. Rotasi kanan mengubah gambar kiri menjadi gambar kanan; rotasi kiri mengubah gambar kanan kembali menjadi gambar kiri. Keduanya balikan yang persis satu sama lain.

Gambar 6.9: rotasi kanan terhadap y mengangkat x; rotasi kiri terhadap x mengangkat y. Keduanya saling membalik dan berongkos $\Theta(1)$. Buku: Subbab 6.16.

Rotasi Mempertahankan Penelusuran Inorder

Pada kedua gambar, sifat pohon pencarian biner memastikan Persamaan (6.10):

$$\text{kunci}(P_1) < x < \text{kunci}(P_2) < y < \text{kunci}(P_3). \tag{6.10}$$

TEOREMA. Rotasi mengubah *bentuk* pohon tetapi tidak mengubah *urutan inorder*-nya; jadi rotasi pada pohon pencarian biner selalu menghasilkan pohon pencarian biner.

Tinggi: rotasi kanan mengangkat P_1 satu level dan menurunkan P_3 satu level; P_2 hanya berganti induk. Rotasi memindahkan tinggi dari satu sisi ke sisi lain.

Hanya P_2 yang berganti induk; subpohon-subpohon itu hanya dialihkan penunjuknya, tidak pernah ditelusuri. Buku: Subbab 6.16 dan 6.16.3.

Rotasi Kanan dalam C++

Kode 6.2 melakukan pembaruan penunjuk secara berurutan dan merotasi rantai $30 \to 20 \to 10$; setiap simpul dicetak sebagai `key[height]`.

Kode 6.2 Rotasi kanan beserta pembaruan penunjuknya yang persis, diterapkan pada rantai yang condong ke kiri $30 \to 20 \to 10$. Kompilasi dengan `g++ -std=c++17 -O2 -o rotright rotright.cpp`.

```cpp
// Rotasi kanan beserta pembaruan penunjuknya yang persis, diterapkan pada rantai yang condong ke kiri.
#include <iostream>
#include <string>
#include <algorithm>

struct Node {
    int   key;
    int   height;                 // daun bertinggi 0, pohon kosong -1
    Node* left;
    Node* right;
    Node(int k) : key(k), height(0), left(nullptr), right(nullptr) {}
};

int  h(const Node* x) { return x ? x->height : -1; }
void refresh(Node* x) { x->height = 1 + std::max(h(x->left), h(x->right)); }

Node* rotateRight(Node* y) {
    Node* x  = y->left;         // 1. ingat simpul yang akan naik
    Node* P2 = x->right;        // 2. ingat subpohon yang akan berganti induk
    x->right = y;               // 3. y menjadi anak kanan x
    y->left  = P2;              // 4. P2 menjadi anak kiri y
    refresh(y);                 // 5. y kini lebih rendah: hitung ulang tingginya LEBIH DAHULU
    refresh(x);                 // 6. baru x, yang bergantung pada tinggi baru y
    return x;                   // 7. x adalah akar baru subpohon ini
}

std::string shape(const Node* x) {          // kunci[tinggi]{kiri,kanan}; "." berarti kosong
    if (!x) return ".";
```

```
    std::string s = std::to_string(x->key) + "[" + std::to_string(x->height) + "]";
    if (x->left || x->right) s += "{" + shape(x->left) + "," + shape(x->right) + "}";
    return s;
}

int main() {
    Node* a = new Node(30); Node* b = new Node(20); Node* c = new Node(10);
    a->left = b; b->left = c;                    // rantai 30 -> 20 -> 10, condong ke kiri
    refresh(c); refresh(b); refresh(a);
    std::cout << "before: " << shape(a) << "\n";
    Node* root = rotateRight(a);
    std::cout << "after : " << shape(root) << "\n";
    delete a; delete b; delete c;
    return 0;
}
```

```
before: 30[2]{20[1]{10[0],.},.}
after : 20[1]{10[0],30[0]}
```

*Perbarui y **sebelum** x: tinggi x dihitung dari tinggi y. Membalik urutannya adalah kesalahan AVL yang paling umum, dan ia tidak pernah membuat program macet. Buku: Subbab 6.16.1 dan 6.16.2.*

Penyisipan AVL: Empat Langkah, Empat Kasus

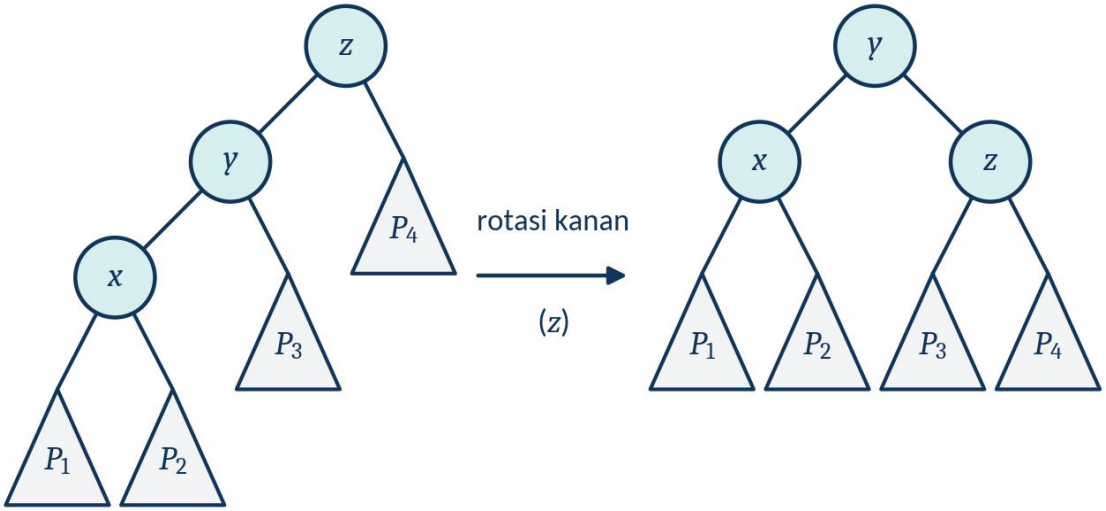
PENYISIPAN AVL. (1) Sisipkan s seperti pada pohon pencarian biner biasa. (2) Naiklah dari s ke simpul pertama yang faktor keseimbangannya keluar dari $\{-1, 0, +1\}$; sebut z. (3) Misalkan y anak z pada lintasan itu dan x cucunya. (4) Seimbangkan ulang di z.

LL: y di kiri z, x di kiri y. **LR:** y kiri, x kanan. **RR:** y kanan, x kanan. **RL:** y kanan, x kiri.

Pada keempat gambar, $P_1, \ldots, P_4$ adalah subpohon yang tingginya, sesudah penyisipan, k atau $k-1$; penyisipan itu menaikkan x dari tinggi k ke $k+1$.

"Kiri-Kiri" berarti pohonnya condong ke kiri dua kali berturut-turut. Buku: Subbab 6.17.

Kasus LL



Gambar 6.10 Kasus Kiri-Kiri, yang juga muncul pada penghapusan. Satu rotasi kanan terhadap z memulihkan keseimbangannya; y menjadi akar baru subpohonnya, dengan x dan z sebagai anak-anaknya.

Gambar 6.10: $bf(z) = k - (k+2) = -2$ *dan* $bf(y) = k - (k+1) = -1$. *Satu rotasi kanan terhadap* z. *Buku: Subbab 6.17.1.*

Perbaikan LL

Pseudokode 6.1 Perbaikan Kiri-Kiri: satu rotasi kanan terhadap z, dituliskan sebagai keempat penugasan penunjuknya.

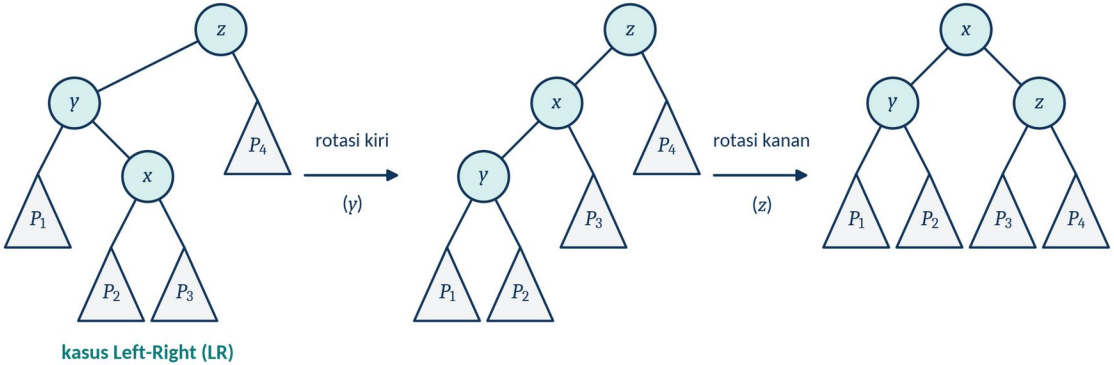
```
y        = z->left
P3       = y->right
y->right = z
z->left  = P3
refresh(z); refresh(y);         // z dahulu, karena ia kini lebih rendah
```

Pseudokode 6.1 adalah rotasi kanan pada Kode 6.2 dengan peran yang dinamai ulang. Sesudahnya $bf(z)=0$, $bf(y)=0$, dan x tetap -1.

PENGAMATAN YANG MENENTUKAN. Subpohon di z setinggi $k+2$ sebelum penyisipan, $k+3$ sesudahnya, dan kembali $k+2$ sesudah rotasi. Leluhurnya tidak melihat perubahan apa pun.

Pengamatan inilah alasan penyisipan tidak pernah memerlukan lebih dari satu penyeimbangan ulang. Buku: Subbab 6.17.1.

Kasus LR



Gambar 6.11 Kasus Kiri-Kanan, yang juga muncul pada penghapusan. Dua rotasi diperlukan: pertama rotasi kiri terhadap y, yang mengubah persoalannya menjadi kasus LL, lalu rotasi kanan terhadap z.

Gambar 6.11: $bf(z) = -2$ *tetapi* $bf(y) = +1$ *— kiri, lalu kanan. Rotasi kiri terhadap* y *lebih dulu, lalu rotasi kanan terhadap* z*. Buku: Subbab 6.17.2.*

Perbaikan LR

Mengapa satu rotasi gagal: rotasi kanan terhadap z saja akan menyisakan $bf(y) = (k+2) - k = +2$ — masih rusak, hanya condong ke arah lain. Satu rotasi tidak dapat membereskan zig-zag.

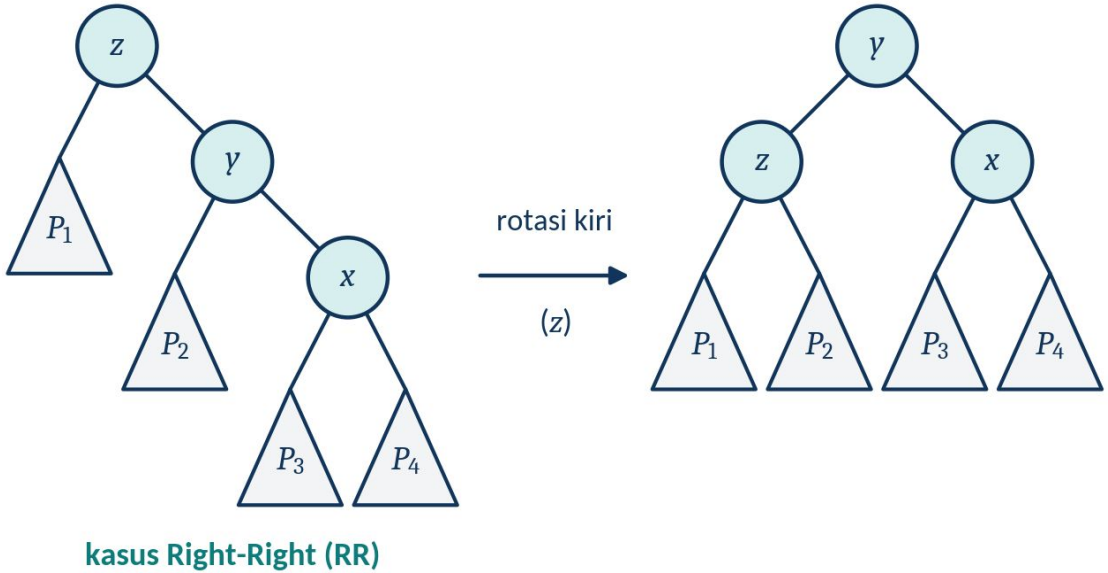
Pseudokode 6.2 Perbaikan Kiri-Kanan: rotasi kiri terhadap y mengubah zig-zagnya menjadi kasus Kiri-Kiri, lalu rotasi kanan terhadap z merampungkannya.

```
z->left = rotateLeft(z->left);   // langkah 1: mengubah LR menjadi LL
return rotateRight(z);           // langkah 2: perbaikan LL yang biasa
```

Pseudokode 6.2: langkah 1 mengubah zig-zag $z \to y \to x$ menjadi rantai lurus $z \to x \to y$ (kasus LL); langkah 2 adalah perbaikan LL biasa. Hasilnya: x di atas, y dan z di bawahnya, tinggi $k+2$.

Ketiga faktor keseimbangannya berakhir di $\{-1, 0, +1\}$. Buku: Subbab 6.17.2.

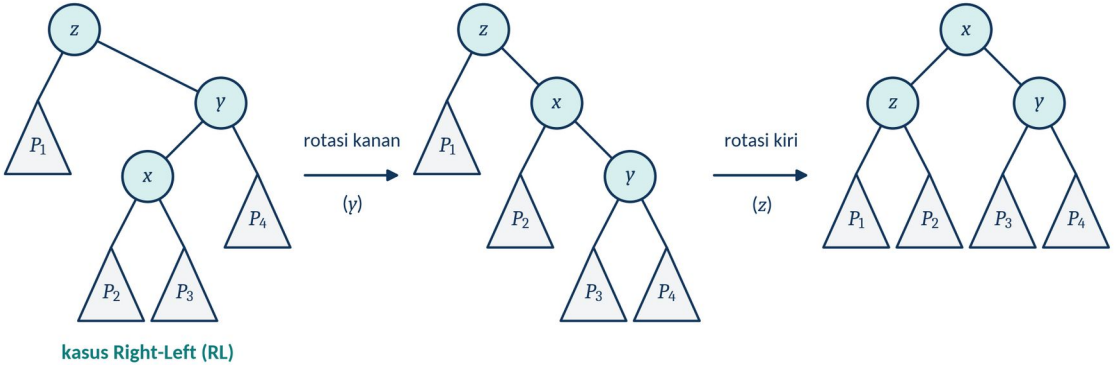
Kasus RR



Gambar 6.12 Kasus Kanan-Kanan, yang juga muncul pada penghapusan. Bayangan cermin yang persis dari Gambar 6.10: satu rotasi kiri terhadap z.

Gambar 6.12: $bf(z) = +2$, $bf(y) = +1$ — *dua kali condong ke kanan. Satu rotasi kiri terhadap* z*. Buku: Subbab 6.17.3.*

Kasus RL



Gambar 6.13 Kasus Kanan-Kiri, yang juga muncul pada penghapusan. Cermin dari Gambar 6.11: pertama rotasi kanan terhadap y, yang mengubah zig-zagnya menjadi kasus RR, lalu rotasi kiri terhadap z.

Gambar 6.13: $bf(z) = +2$, $bf(y) = -1$ — *kanan, lalu kiri. Rotasi kanan terhadap* y*, lalu rotasi kiri terhadap* z*. Buku: Subbab 6.17.4.*

Perbaikan RR dan RL

Pseudokode 6.3 Perbaikan Kanan-Kanan: satu rotasi kiri terhadap z, bayangan cermin Pseudokode 6.1.

```
y       = z->right
P2      = y->left
y->left  = z
z->right = P2
refresh(z); refresh(y);
```

Pseudokode 6.4 Perbaikan Kanan-Kiri: rotasi kanan terhadap y, lalu rotasi kiri terhadap z.

```
z->right = rotateRight(z->right);   // langkah 1: mengubah RL menjadi RR
return rotateLeft(z);               // langkah 2: perbaikan RR yang biasa
```

Pseudokode 6.3 mencerminkan perbaikan LL; Pseudokode 6.4 mencerminkan perbaikan LR. Pada setiap kasus subpohonnya kembali setinggi $k+2$.

Empat kasus, paling banyak dua rotasi, masing-masing hanya mengubah $O(1)$ penunjuk. Buku: Subbab 6.17.3 dan 6.17.4.

Membedakan Keempat Kasus

Tabel 6.5 Keempat kasus penyeimbangan ulang, dikenali lewat dua faktor keseimbangan, beserta rotasi yang dituntut masing-masingnya.

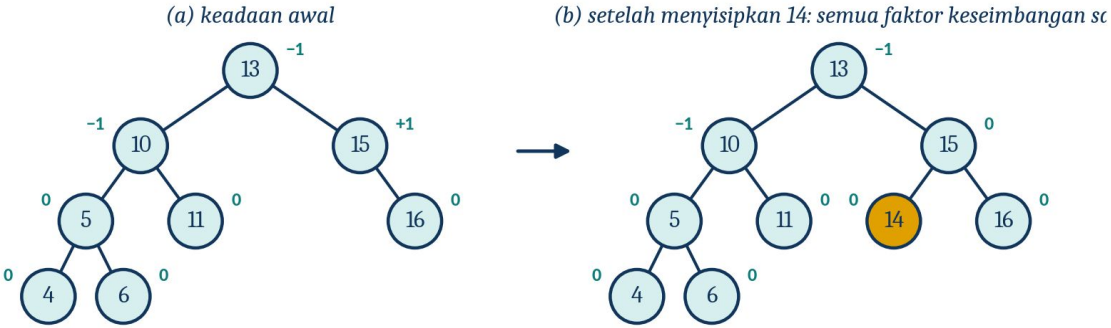
bf(z)	bf(y), dengan y adalah anak z yang lebih tinggi	Kasus	Rotasi	Gambar
< -1 (berat kiri)	negatif	Kiri-Kiri	rotasi kanan terhadap z	Gambar 6.10
< -1 (berat kiri)	positif	Kiri-Kanan	kiri terhadap y, lalu kanan terhadap z	Gambar 6.11
$> +1$ (berat kanan)	positif	Kanan-Kanan	rotasi kiri terhadap z	Gambar 6.12
$> +1$ (berat kanan)	negatif	Kanan-Kiri	kanan terhadap y, lalu kiri terhadap z	Gambar 6.13

Tabel 6.5 hanya memerlukan dua faktor keseimbangan: tanda di z menentukan sisinya, tanda di y menentukan rotasi tunggal atau ganda.

BAGAIMANA JIKA $bf(y)=0$? Mustahil sesudah penyisipan, lazim sesudah penghapusan. Perlakukan sebagai kasus rotasi **tunggal**: tulis `if (bf(z->left) <= 0)`, bukan `< 0`. Rotasi ganda akan menyisakan y pada ± 2.

Diucapkan: LL adalah $(-)$ lalu $(-)$, LR $(-)$ lalu $(+)$, RR $(+)$ lalu $(+)$, RL $(+)$ lalu $(-)$. Buku: Subbab 6.17.5.

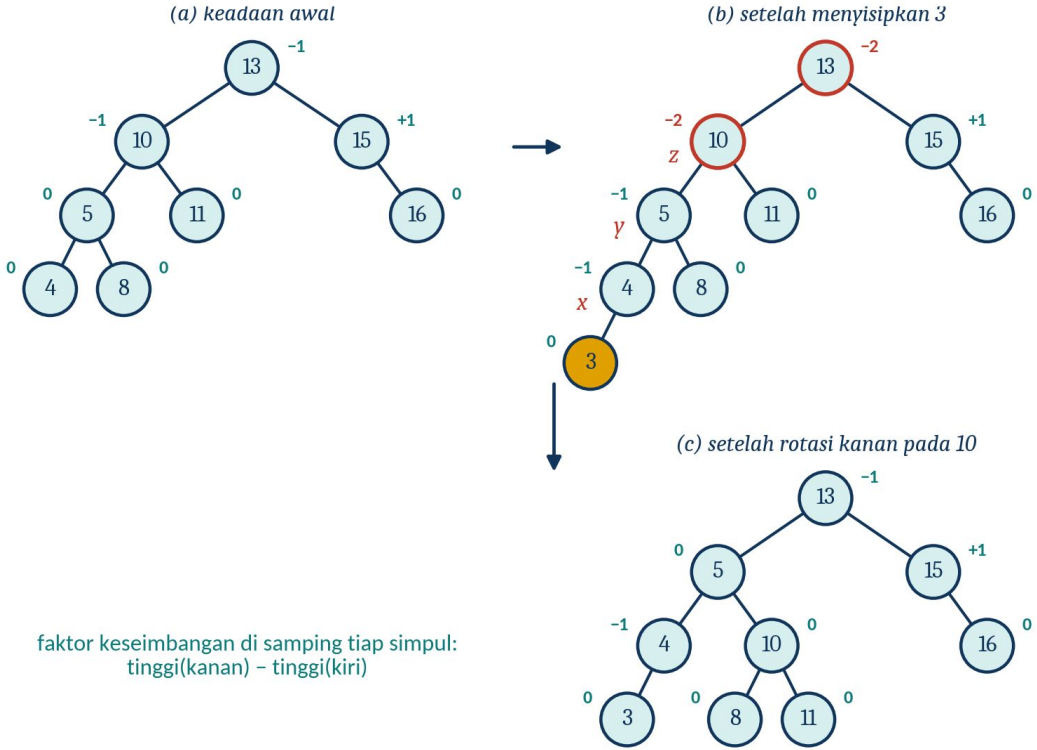
Contoh Penyisipan, Langkah 1: Sisipkan 14



Gambar 6.14 Menyisipkan 14 ke dalam sebuah pohon AVL. Setiap faktor keseimbangannya tetap berada di dalam $\{-1, 0, +1\}$, jadi tidak ada rotasi yang dilakukan.

Gambar 6.14: 14 *ke kanan di* 13*, ke kiri di* 15*.* $bf(15)$ *membaik dari* $+1$ *menjadi* 0*;* $bf(13) = 1 - 2 = -1$*. Tanpa rotasi — kasus yang paling umum. Buku: Subbab 6.18.1.*

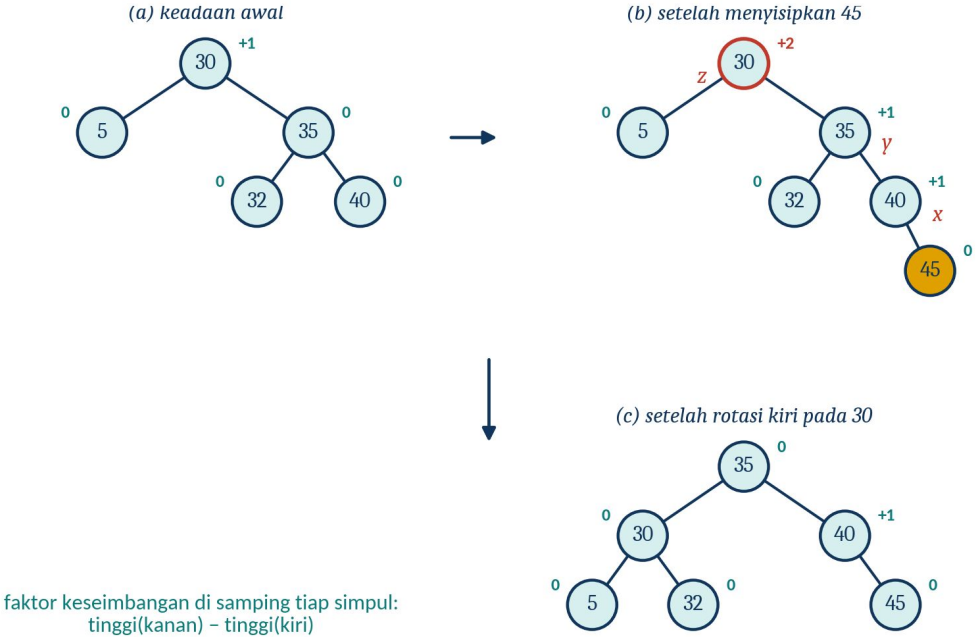
Langkah 2: Sisipkan 3 — Kiri-Kiri



Gambar 6.15 Menyisipkan 3 menghasilkan faktor keseimbangan −2 di simpul 10. Kasusnya adalah Kiri-Kiri, dan satu rotasi kanan dengan 10 sebagai porosnya memperbaikinya.

Gambar 6.15: 3 tergantung di bawah 4; $bf(4) = -1$, $bf(5) = -1$, $bf(10) = 0 - 2 = -2$. Jadi $z = 10$, $y = 5$, $x = 4$: LL. Rotasi kanan terhadap 10; 8 berganti induk. Tinggi pohon tetap 3. Buku: Subbab 6.18.2.

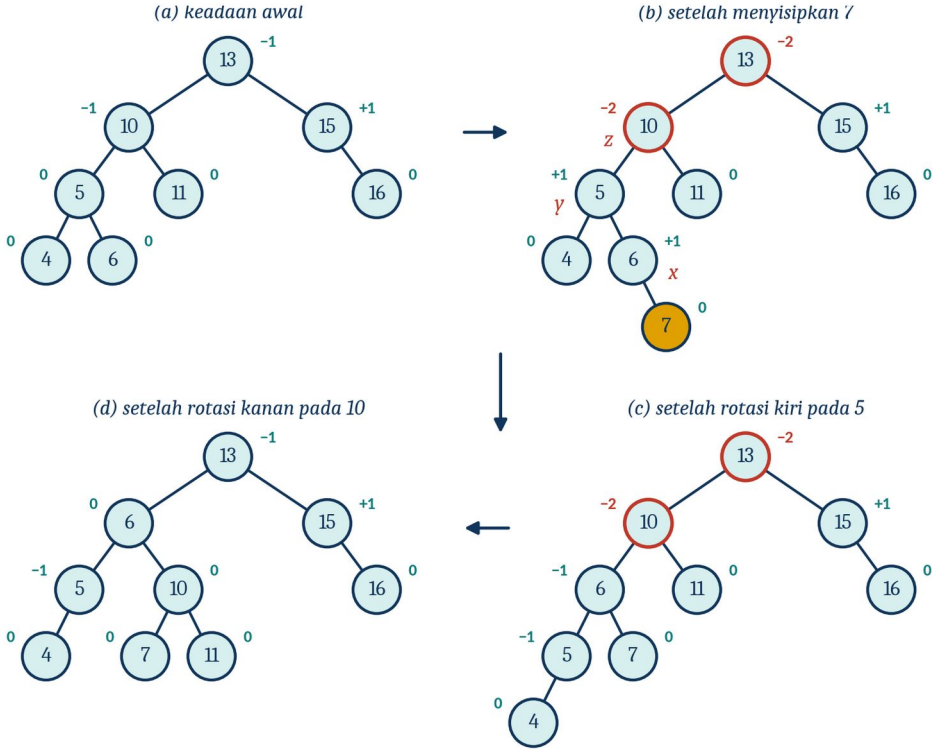
Langkah 3: Sisipkan 45 — Kanan-Kanan



Gambar 6.16 Menyisipkan 45 membuat simpul 30 berat kanan sebanyak dua. Kasusnya adalah Kanan-Kanan, dan satu rotasi kiri dengan 30 sebagai porosnya memperbaikinya.

Gambar 6.16: $bf(40) = +1$, $bf(35) = +1$, $bf(30) = 2 - 0 = +2$. *Jadi* $z = 30$, $y = 35$, $x = 40$: *RR. Rotasi kiri terhadap* 30; 32 *berganti induk. Akar baru* 35. *Buku: Subbab 6.18.3.*

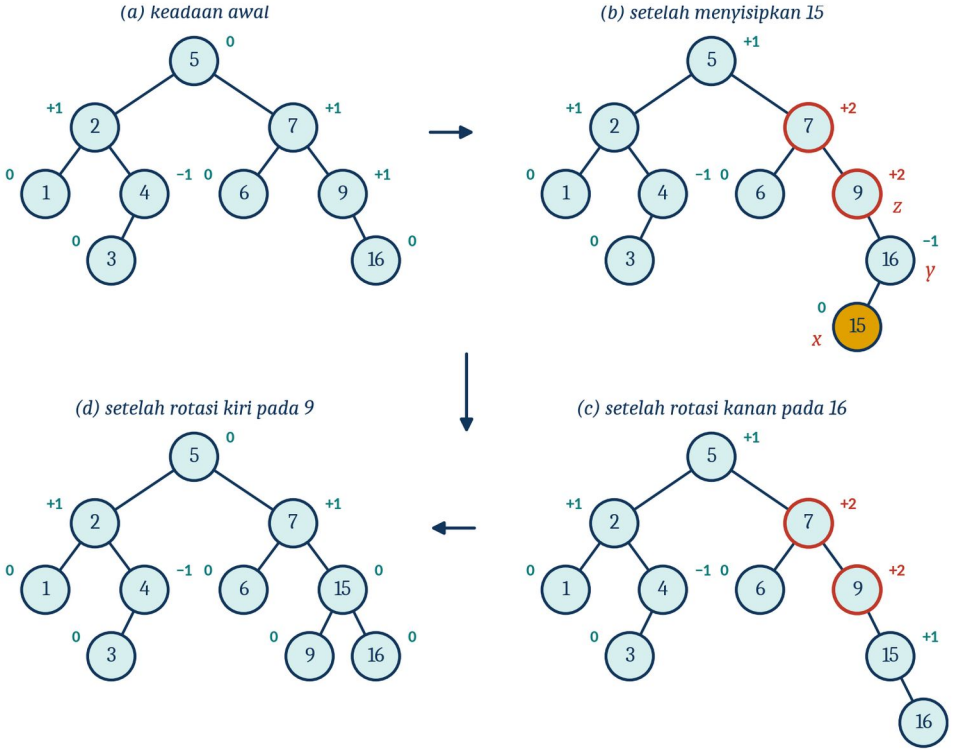
Langkah 4: Sisipkan 7 — Kiri-Kanan



Gambar 6.17 Menyisipkan 7 menciptakan zig-zag — ke kiri di 10, lalu ke kanan di 5. Dua rotasi diperlukan: kiri dengan 5 sebagai poros, lalu kanan dengan 10 sebagai poros.

Gambar 6.17: 7 tergantung di kanan 6; $bf(5) = +1$, $bf(10) = -2$. Jadi $z = 10$, $y = 5$, $x = 6$: LR. Kiri terhadap 5, lalu kanan terhadap 10: 6 berakhir di atas dengan anak 5 dan 10. Buku: Subbab 6.18.4.

Langkah 5: Sisipkan 15 — Kanan-Kiri



Gambar 6.18 Menyisipkan 15 membuat simpul 9 berat kanan sebanyak dua sementara simpul 16 condong ke kiri. Kasusnya adalah Kanan-Kiri: rotasi kanan dengan 16 sebagai poros, lalu rotasi kiri dengan 9 sebagai poros.

Gambar 6.18: 15 tergantung di kiri 16; $bf(16) = -1$, $bf(9) = 1 - (-1) = +2$. Jadi $z = 9$, $y = 16$, $x = 15$: RL. Kanan terhadap 16, lalu kiri terhadap 9: 15 berakhir di atas dengan anak 9 dan 16. Buku: Subbab 6.18.5.

Implementasi Rujukan dalam Java, Bagian 1

```java
package AVLTree;

/**
 *
 * AVLTree class extended from Tree class.
 * Added a variable "height" for performing height-balanced efficiently
 */

public class AVLTree extends Tree {

    protected final boolean empty;
    protected final int value;
    protected final AVLTree left, right;
    protected final int height;

    /**
     * Creates a new tree whose root holds value and whose
     * left and right subtrees are left and right
     */
    public AVLTree(int value, AVLTree left, AVLTree right) {
        this.empty = false;
        this.value = value;
        this.left = left;
        this.right = right;
        this.height = 1 + Math.max(left.getHeight(), right.getHeight());
    }
```

Gambar 6.19 Kelas `AVLTree.java`, bagian pertama. Kelasnya memperluas kelas `Tree` biasa dan menambahkan satu medan — `height` — "for performing height-balanced efficiently". Konstruktornya menetapkan `height = 1 + Math.max(left.getHeight(), right.getHeight())`.

Gambar 6.19: pohon AVL adalah pohon dengan satu medan tambahan. `height = 1 + max(left, right)` persis sama dengan `refresh` pada kode C++. Buku: Subbab 6.19 dan 6.19.1.

Implementasi Rujukan dalam Java, Bagian 2

```
26
27      /**
28       * Creates an empty tree
29       */
30      public AVLTree() {
31          this.empty = true;
32          this.value = 0;
33          this.left = null;
34          this.right = null;
35          this.height = 0;
36      }
37
38      /**
39       * Creates a tree with a single node
40       */
41      public AVLTree(int value) {
42          this.empty = false;
43          this.value = value;
44          this.left = new AVLTree();
45          this.right = new AVLTree();
46          this.height = 1;
47      }
48
49      /**
50       * Gets the height
51       */
52      private int getHeight() {
53          return height;
54      }
55
56  }
```

Gambar 6.20 Kelas `AVLTree.java`, bagian kedua. Pohon kosong adalah sebuah objek dengan `height = 0`; pohon bersimpul tunggal punya `height = 1`.

Gambar 6.20 menghitung simpul: pohon kosong 0*, satu simpul* 1*. C++ kita memakai* -1 *dan* 0*; faktor keseimbangannya tetap sama. Buku: Subbab 6.19.1.*

Penghapusan AVL: Dua Perbedaan

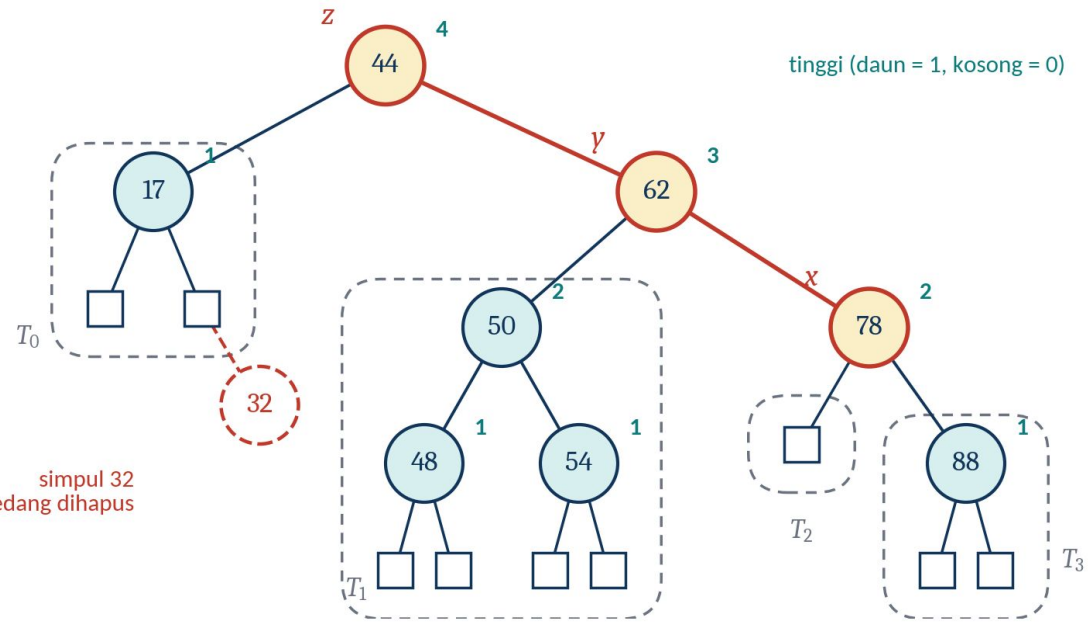
Hapus seperti pada pohon pencarian biner biasa; naik ke simpul tak seimbang pertama z; seimbangkan ulang dengan empat kasus yang sama.

Perbedaan 1: y adalah anak z **yang lebih tinggi**, dan x anak y yang lebih tinggi — sisi yang menyusut adalah sisi pendek, jadi sisi tinggi harus dicari langsung. Bila seri, ambil x pada sisi yang sama dengan y.

Perbedaan 2: teruskan. Sesudah penghapusan, subpohon yang dirotasi bisa menjadi lebih pendek daripada semula, induknya bisa menjadi tak seimbang, dan perbaikannya merambat — sampai $O(\lg n)$ rotasi.

Diukur pada pohon Fibonacci: satu penyisipan selalu memerlukan 1 *penyeimbangan ulang; satu penghapusan memerlukan* $2, 3, 5, 6, 8$ *pada tinggi* $4, 7, 10, 13, 16$. *Keduanya tetap* $O(\lg n)$. *Buku: Subbab 6.20–6.20.2.*

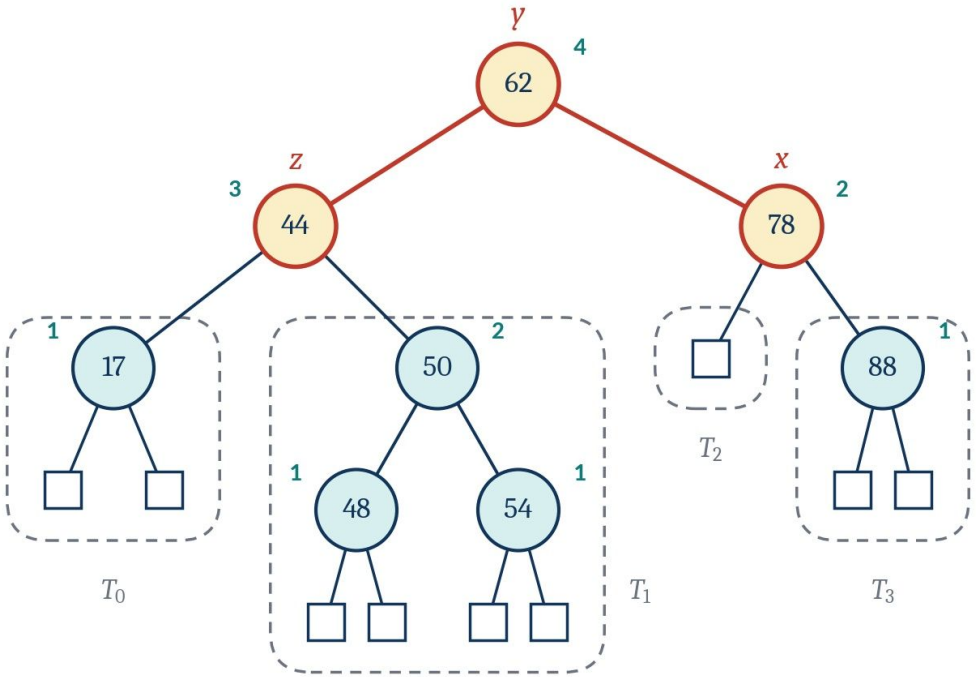
Contoh Penghapusan — Sebelum



Gambar 6.21 Menghapus dari pohon AVL: simpul 32 hendak dihapus. Tinggi dituliskan di samping simpulnya, dan leluhur tak seimbang yang pertama z, anaknya yang lebih tinggi y, dan anak y yang lebih tinggi x ditandai dengan warna merah. Kotak-kotak kosongnya adalah subpohon nol, dan T_0 sampai T_3 menamai keempat subpohon yang akan dipindahkan rotasinya.

Gambar 6.21 menghitung simpul (daun bernilai 1*). Hapus daun* 32*; di* 44 *kedua sisinya* 1 *dan* 3*, jadi* $z = 44$*,* $y = 62$*;* 50 *dan* 78 *seri di* 2*, jadi* $x = 78$*: Kanan-Kanan. Buku: Subbab 6.20.3.*

Contoh Penghapusan — Sesudah



Gambar 6.22 Pohonnya sesudah rotasi kiri. Simpul 62 adalah akar barunya; 44 turun menjadi anak kirinya dan mengambil subpohon T_1 (yang berakar di 50) sebagai anak kanannya; 78 tidak berubah sebagai anak kanan 62.

Gambar 6.22: satu rotasi kiri terhadap 44; 50 *(*T_1*) pindah ke kanan* 44*. Kini setiap simpul berselisih paling banyak satu, dan di sini satu rotasi sudah cukup. Buku: Subbab 6.20.3.*

Pohon AVL, Pohon Merah-Hitam, atau Tabel Hash?

Pohon merah-hitam menyimpan satu bit warna; jumlah simpul hitam yang sama pada setiap lintasan dan tidak adanya sisi merah–merah memberi Persamaan (6.11), dibandingkan sekitar $1.44 \lg n$ untuk AVL.

$$h \leq 2 \lg (n+1) \tag{6.11}$$

Penyisipan merah-hitam memerlukan ≤ 2 rotasi dan penghapusannya ≤ 3; penghapusan AVL bisa memerlukan $\Theta(\lg n)$. **Beban baca yang dominan cocok untuk AVL; beban tulis yang dominan cocok untuk merah-hitam** (`std::map`).

Tabel hash memberi $O(1)$ secara harapan tetapi tanpa urutan: tanpa penerus, tanpa minimum, tanpa kueri rentang. Diukur pada $200,000$ kunci: pencarian 2.65 ms (hash) lawan 36.59 ms (AVL); jumlah rentang 1000 kunci 0.51 ms (hash) lawan 0.01 ms (AVL).

Diberi kunci menaik, pohon AVL mencapai tinggi 17*; pohon pencarian biner biasa dengan kunci acak mencapai* 41*. Buku: Subbab 6.21.*

Berapa Ongkos Setiap Operasi

Tabel 6.6 Biaya kasus terburuk untuk pohon pencarian biner biasa dan untuk pohon AVL dengan n simpul.

Operasi	BST biasa, kasus terburuk	Pohon AVL, kasus terburuk	Mengapa
SEARCH	$\Theta(n)$	$\Theta(\lg n)$	satu lintasan akar-ke-daun
MINIMUM, MAXIMUM	$\Theta(n)$	$\Theta(\lg n)$	satu lintasan, selalu ke arah yang sama
SUCCESSOR, PREDECESSOR	$\Theta(n)$	$\Theta(\lg n)$	satu lintasan turun, atau satu lintasan naik
INSERT	$\Theta(n)$	$\Theta(\lg n)$, ≤ 2 rotasi	susur turun, tempel, seimbangkan ulang sekali
DELETE	$\Theta(n)$	$\Theta(\lg n)$, $O(\lg n)$ rotasi	susur turun, sambung, seimbangkan ulang sampai ke atas
Penelusuran inorder	$\Theta(n)$	$\Theta(n)$	tiap simpul dikunjungi sekali
Kueri rentang yang mengembalikan k kunci	$\Theta(n)$	$\Theta(\lg n+k)$	susur turun sekali, lalu susuri rentangnya
Bangun dari n kunci	$\Theta(n^2)$	$\Theta(n \lg n)$	n penyisipan
Tinggi	sampai $n-1$	$\leq 1.4405 \lg(n+2) - 1.327$	rekurensi simpul-minimum

Tabel 6.6 adalah ringkasan untuk dihafalkan: setiap kueri menelusuri satu lintasan, jadi tinggi $O(\lg n)$ yang terjamin membuat setiap operasi $O(\lg n)$.

Penyisipan menyeimbangkan ulang sekali; penghapusan mungkin di setiap level. Buku: Subbab 6.24.

Profitina dalam Praktik: Papan Pengambilan

Di Profitina, penjadwal pemindaian ulang bertanya “pemindaian mana yang jatuh tempo berikutnya?” (MINIMUM) dan “apa saja yang jatuh tempo antara 02:00 dan 04:00?” (kueri rentang); ekspor CSV mingguan adalah penelusuran inorder; deduplikasi URL hanya perlu kesamaan, jadi memakai tabel hash.

Profitina Laundry menyimpan papan pengambilannya dalam pohon AVL berkunci (menit yang dijanjikan, tiket): sisipkan saat cucian diterima, penerus untuk “apa yang jatuh tempo berikutnya?”, rentang untuk “apa yang jatuh tempo antara 10:00 dan 14:30?”, hapus saat diambil.

Sembilan pesanan duduk dalam pohon setinggi 3 — yang terpendek yang mungkin untuk sembilan simpul — walaupun tiket datang dengan urutan waktu yang menaik.

Platform memberi saran; pelanggan yang memutuskan. Penghapusan tuntas menurut UU PDP 27/2022 memerlukan indeks yang dapat menghapus dengan murah. Buku: Subbab 6.22 dan 6.22.1.

Memeriksa Pohon Pencarian Biner dengan Benar

Kesalahan yang paling umum hanya memeriksa anak-anaknya. Kode 6.3 membawa sebuah selang terbuka turun ke bawah pohon, dan menunjukkan pohon yang mengecoh uji lokal.

Kode 6.3 Cara yang salah dan cara yang benar untuk memverifikasi sifat pohon pencarian biner. Kompilasi dengan `g++ -std=c++17 -O2 -o isbst isbst.cpp`.

```cpp
// Apakah pohon biner ini sebuah pohon pencarian biner?
// Uji lokal yang menggoda -- "anak kiri lebih kecil, anak kanan lebih besar" -- itu
// salah. Uji yang benar membawa rentang yang diizinkan turun menyusuri pohon.
#include <iostream>
#include <climits>

struct Node {
    int key;
    Node* left;
    Node* right;
    Node(int k, Node* l = nullptr, Node* r = nullptr) : key(k), left(l), right(r) {}
};

// SALAH: hanya membandingkan sebuah simpul dengan kedua anaknya.
bool looksLocallyFine(Node* x) {
    if (!x) return true;
    if (x->left  && x->left->key  >= x->key) return false;
    if (x->right && x->right->key <= x->key) return false;
    return looksLocallyFine(x->left) && looksLocallyFine(x->right);
}

// BENAR: setiap kunci di subpohon kiri harus di bawah x, setiap kunci di
// subpohon kanan di atasnya, jadi turunkan selang terbuka (lo, hi) ke bawah.
bool isBST(Node* x, long long lo, long long hi) {
    if (!x) return true;
    if (x->key <= lo || x->key >= hi) return false;
    return isBST(x->left, lo, x->key) && isBST(x->right, x->key, hi);
}
```

```
int main() {
    /*        20
     *       /  \
     *      10   30
     *          /  \
     *         5    40      <-- 5 ada di subpohon KANAN milik 20: bukan BST
     */
    Node* bad = new Node(20, new Node(10), new Node(30, new Node(5), new Node(40)));

    /*        20
     *       /  \
     *      10   30
     *     /       \
     *    5         40
     */
    Node* good = new Node(20, new Node(10, new Node(5)), new Node(30, nullptr, new Node(40)));
```

Kode 6.3 (Lanjutan) *Cara yang salah dan cara yang benar untuk memverifikasi sifat pohon pencarian biner. Kompilasi dengan g++ -std=c++17 -O2 -o isbst isbst.cpp.*

```
    std::cout << "bad  tree: local test says "
              << (looksLocallyFine(bad) ? "BST" : "not a BST")
              << ", correct test says "
              << (isBST(bad, LLONG_MIN, LLONG_MAX) ? "BST" : "not a BST") << "\n";
    std::cout << "good tree: local test says "
              << (looksLocallyFine(good) ? "BST" : "not a BST")
              << ", correct test says "
              << (isBST(good, LLONG_MIN, LLONG_MAX) ? "BST" : "not a BST") << "\n";
    return 0;
}
```

```
bad  tree: local test says BST, correct test says not a BST
good tree: local test says BST, correct test says BST
```

Selangnya bertipe `long long` agar nilai penjaganya berada tegas di luar setiap kunci `int`. Buku: Subbab 6.23, kesalahan 1.

Kesalahan yang Sering Terjadi

Memeriksa sifat pohon pencarian biner hanya terhadap anak-anaknya, atau mencampuradukkannya dengan sifat *heap*.

Memperbarui tinggi dengan urutan yang salah sesudah rotasi; **mencampur** kesepakatan tinggi -1 dan 0 untuk pohon kosong.

Berhenti sesudah satu penyeimbangan ulang pada penghapusan; memakai `< 0` alih-alih `<= 0` untuk uji rotasi tunggal.

Menganggap pohon pencarian biner biasa pasti seimbang karena biasanya begitu; menyalin kunci tanpa data satelitnya pada kasus 3.

Memakai pohon ketika tabel hash sudah cukup — pohon hanya sepadan ongkosnya bila urutan itu penting.

Seluruhnya sepuluh kesalahan, masing-masing dengan perbaikannya. Buku: Subbab 6.23.

Ringkasan: Lima Hal untuk Diingat

1. Subpohon kiri $<$ simpul $<$ subpohon kanan, untuk *setiap* kunci di bawahnya — bukan hanya anak-anaknya.

2. Inorder itu terurut (dibuktikan dengan induksi); setiap kueri dan pembaruan menelusuri satu lintasan: $O(h)$.

3. Penghapusan: lepaskan daun; naikkan anak tunggal; salin penerus, lalu hapus penerus itu.

4. AVL: $bf \in \{-1, 0, +1\}$; $N(h)+1 = F(h+3)$ memberi $h \le 1.4405 \lg(n+2) - 1.3277$.

5. Rotasi mempertahankan urutan inorder; empat kasus; penyisipan menyeimbangkan ulang sekali, penghapusan mungkin di setiap level.

Pohon pencarian biner biasa bergantung pada keberuntungan Anda; pohon AVL tidak. Buku: Subbab 6.24.

Pertemuan Berikutnya

Bab 7 — dari divide and conquer ke dynamic programming. Fibonacci lagi — kali ini sebagai persoalan terkecil yang mematahkan divide and conquer.

Memoization, tabel bottom-up, perkalian rantai matriks, dan longest common subsequence.

Sebelum itu: ulangi kelima langkah penyisipan dan contoh penghapusan dengan tangan, menghitung setiap faktor keseimbangan.

Bab 8 kemudian mengerjakan soal-soal dari Bab 5 sampai 7, termasuk pohon-pohon ini. Buku: Subbab 6.24.1 dan 6.25.

Kuliah 7 — Dari Divide and Conquer ke Dynamic Programming

Perancangan dan Analisis Algoritma (EF234405)

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*, Bab 7

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita belajar menilai, dengan bukti, apakah sebuah rekursi menghabiskan waktunya menghitung ulang jawaban yang sudah ia miliki — dan bagaimana menjadikannya polinomial. Buku: Bab 7, Subbab 7.1–7.13.

Yang Dibahas Pertemuan Ini

1. ***Divide and conquer*, dinyatakan ulang** — beserta satu anggapannya: potongan-potongannya semua berbeda.
2. **Fibonacci** — $2F_{n+1} - 1$ panggilan untuk $n + 1$ pertanyaan, dan empat perbaikan: *memoization*, *bottom-up*, jendela geser, pangkat matriks.
3. **Persoalan optimasi** — *optimal substructure* dan bukti potong-dan-tempel.
4. **Perkalian rantai matriks** — rekurensinya, tabel yang diisi dengan tangan, pengurungan yang dibaca kembali, $\Theta(n^3)$.
5. **Resep empat langkah**, dan ***longest common subsequence*** — tabel, panah, rekonstruksi, ruang $\Theta(\min(m, n))$.
6. **Submasalah yang tumpang tindih lawan yang saling bebas** — diukur; pencocok nota Profitina Laundry.

Dua contoh yang dikerjakan tuntas hari ini: rantai matriks $d = \langle 10,20,3,5,30 \rangle$ *serta string "ACGGTTA" dan "CGTAT". Buku: Bab 7.*

Dari Mana Kita Melanjutkan

Bab 3 memberi kita rekurensi umum *divide and conquer* Persamaan (7.1), dan tiga cara menyelesaikannya: substitusi, pohon rekursi, Teorema Master.

$$T(n) = a\,T(n/b) + f(n), \qquad a \geq 1,\ b > 1. \tag{7.1}$$

> **DEFINISI. Submasalah** adalah instans yang lebih kecil dari persoalan yang *sama*, yang dihasilkan di tengah jalan. **Pohon rekursi** berakar pada panggilan semula, dan anak-anak setiap simpul adalah panggilan langsungnya; setiap simpul diberi label submasalahnya.

Satu pengamatan: kadang label yang sama muncul pada lebih dari satu simpul. Bila tidak, *divide and conquer* yang tepat; bila ya — dan berulang secara dahsyat — *dynamic programming* yang tepat.

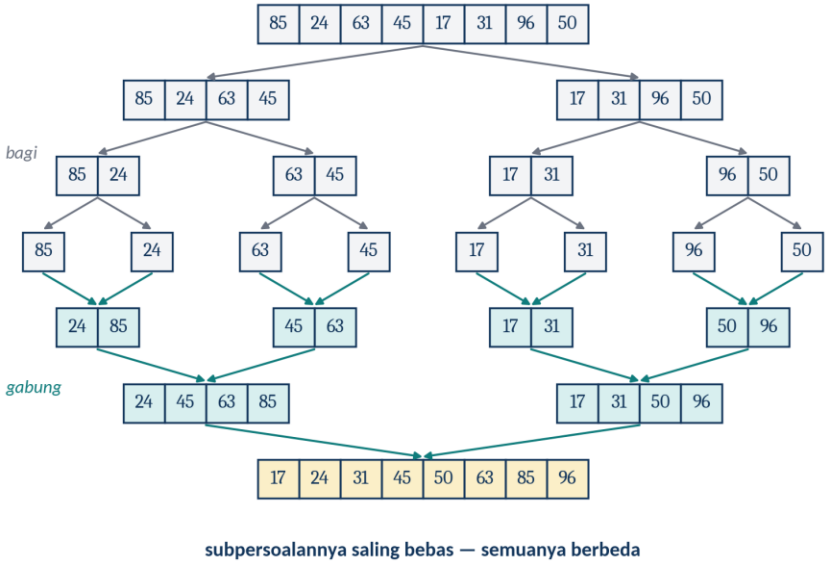
Bab 3 diam-diam menganggap potongannya berbeda. Bila anggapan itu gagal, Teorema Master tetap menjawab dengan benar — dengan sesuatu seperti $\Theta(1.618^n)$. Buku: Subbab 7.1 dan 7.1.1.

Metode Divide and Conquer

Gambar 7.1 Metode *divide and conquer* untuk perancangan algoritma: bagilah persoalan menjadi submasalah yang **saling lepas**, kuasai submasalah itu dengan rekursi, lalu gabungkan solusinya menjadi solusi persoalan asli.

***Gambar 7.1**: **bagi** menjadi submasalah yang saling lepas, **kuasai** secara rekursif (dengan kasus dasar), **gabungkan** jawabannya. Buku: Subbab 7.2.1.*

Merge Sort: Saling Bebas, Semuanya Berbeda



Gambar 7.2 Merge Sort sebagai *divide and conquer*. Larik dibelah dua, tiap paruh diurutkan secara rekursif, lalu kedua paruh terurut itu digabungkan. Kalimat yang penting tertulis di bawah gambarnya: **submasalahnya saling bebas, semuanya berbeda.**

Gambar 7.2*: tidak ada label yang berulang di mana pun pada pohonnya, sehingga* $T(n) = 2T(n/2) + \Theta(n)$ *menyatakan kebenaran seutuhnya. Divide and conquer cocok bila persoalannya dapat dibelah, potongannya saling lepas, dan penggabungannya murah — Merge Sort, pencarian biner, Strassen. Buku: Subbab 7.2.2 dan 7.2.3.*

Ketika Divide and Conquer Memboroskan Kerja

> **JEBAKANNYA.** Rekursi tidak lambat. *Perhitungan ulang*-lah yang lambat. Rekursi polos tidak punya ingatan: ditanya hal yang sama sejuta kali, ia melakukan sejuta perhitungan penuh.

Contoh terkecilnya adalah Fibonacci, yang didefinisikan oleh Persamaan (7.2):

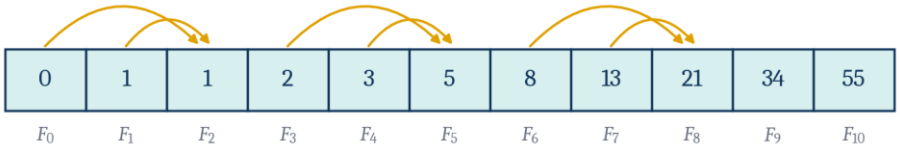
$$F_0 = 0, \qquad F_1 = 1, \qquad F_n = F_{n-1} + F_{n-2} \text{ untuk } n \geq 2. \tag{7.2}$$

Dengan tangan: $F_2 = 1, F_3 = 2, F_4 = 3, F_5 = 5, F_6 = 8, F_7 = 13, F_8 = 21, F_9 = 34, F_{10} = 55$.

Dua baris pertama adalah kasus dasar; baris ketiga adalah kasus rekursif. Buku: Subbab 7.2.4 dan 7.3.1.

Bilangan Fibonacci

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \quad (n \geq 2)$$



setiap bilangan adalah jumlah dua bilangan sebelumnya

prosedur rekursif yang lugas itu lambat — mengapa, dan selambat apa?

Gambar 7.3 Bilangan Fibonacci, beserta peringatan yang menyusul persis sesudahnya: *prosedur rekursif yang lurus-lurus saja itu lambat*. Mengapa, dan selambat apa, adalah urusan empat subbab berikutnya.

Gambar 7.3 *diakhiri peringatan: prosedur rekursif yang lurus-lurus saja itu lambat. Mengapa, dan seberapa lambat, dibahas berikut ini. Buku: Subbab 7.3.1.*

Rekursi yang Lurus-lurus Saja, Dihitung

Pseudokode 7.1 Rekursi yang lurus-lurus saja untuk F_n, disalin langsung dari definisinya.

```
long long fib(int n) {
    if (n <= 1) return n;            // kasus basis: F(0)=0 dan F(1)=1
    return fib(n - 1) + fib(n - 2);  // kasus rekursif
}
```

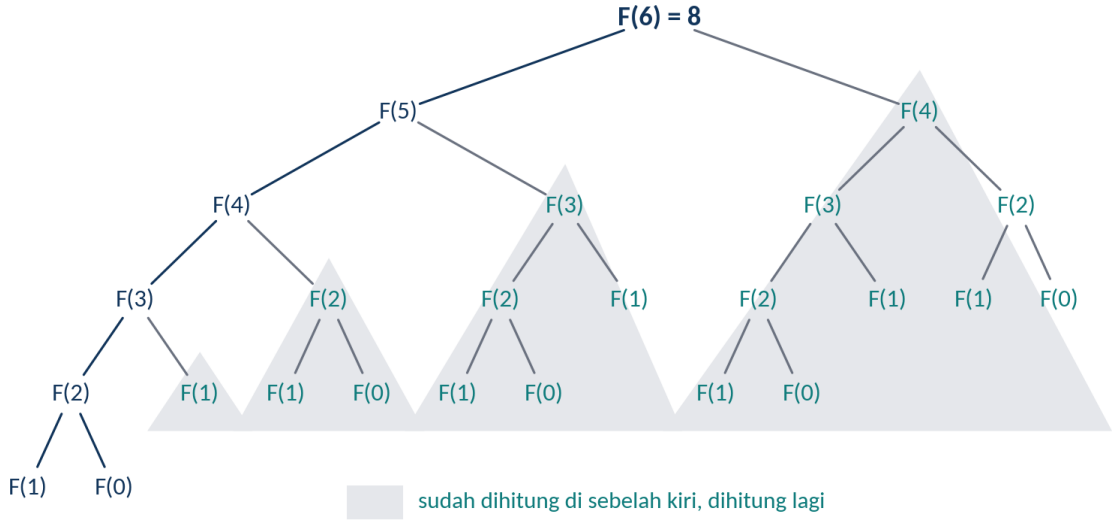
Misalkan $C(n)$ menghitung panggilan yang dibuat **Pseudokode 7.1**: $C(0) = C(1) = 1$ dan $C(n) = C(n-1) + C(n-2) + 1$.

KLAIM. $C(n) = 2F_{n+1} - 1$. *Bukti* dengan induksi kuat: $C(n) = (2F_n - 1) + (2F_{n-1} - 1) + 1 = 2F_{n+1} - 1$. ■

Daun: F_{n+1}; penjumlahan: $F_{n+1} - 1$. Untuk $n = 6$: 25 panggilan, 13 daun, 12 penjumlahan.

Prosedurnya benar. Ia juga lambat dengan cara yang membuat program tak berguna. Buku: Subbab 7.3.2 dan 7.3.4.

Pohon Rekursi `fib(6)`



Kita terus-menerus menghitung nilai yang sama berulang-ulang.

Gambar 7.4 Pohon rekursi `fib(6)` yang lurus-lurus saja. Segitiga kelabu menandai subpohon yang sudah pernah dihitung di suatu tempat di sebelah kirinya dan kini sedang dihitung ulang dari awal. Seluruh paruh kanan pohon itu — seluruh subpohon $F(4)$ — adalah duplikat subpohon yang sudah selesai dikerjakan di sebelah kiri.

***Gambar 7.4**: $F(4)$ dua kali, $F(3)$ tiga kali, $F(2)$ lima kali, $F(1)$ delapan kali. Kita terus menghitung nilai yang sama berulang-ulang. Buku: Subbab 7.3.3.*

Mengapa Biayanya $\Theta(\phi^n)$

$\phi = (1+\sqrt{5})/2 \approx 1.618$ dan $\psi = (1-\sqrt{5})/2 \approx -0.618$ adalah akar-akar $x^2 = x + 1$; mencocokkan kasus dasarnya menghasilkan rumus Binet Persamaan (7.3):

$$F_n = \frac{\phi^n - \psi^n}{\sqrt{5}}. \tag{7.3}$$

Karena $|\psi^n/\sqrt{5}| < 1/2$, F_n adalah bilangan bulat terdekat dengan $\phi^n/\sqrt{5}$, dan kita memperoleh Persamaan (7.4):

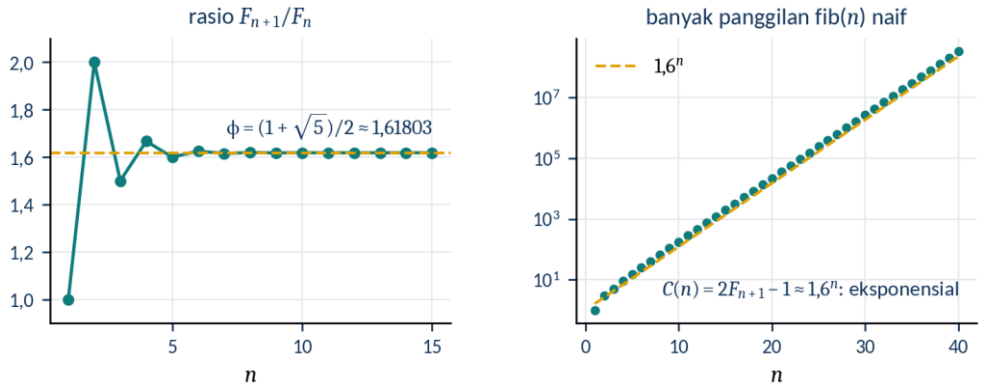
$$C(n) = 2F_{n+1} - 1 = \Theta(\phi^n) = \Theta(1.618^n). \tag{7.4}$$

Hanya ada $n+1$ pertanyaan yang berbeda, tetapi $F(k)$ ditanyakan sebanyak Persamaan (7.5):

$$\text{panggilan ke } F(k) \;=\; F_{n-k+1} \quad (k \geq 1), \qquad \text{panggilan ke } F(0) \;=\; F_{n-1}. \tag{7.5}$$

Komputer seribu kali lebih cepat hanya membeli sekitar empat belas nilai n tambahan. Buku: Subbab 7.3.5 dan 7.3.6.

Rasio Emas, Secara Garis Besar



Gambar 7.5 Argumennya secara garis besar. Kiri: bilangan Fibonacci yang berurutan mendekati rasio emas $\phi \approx 1.61803$, sehingga $F_n \approx 1.6^n$. Kanan: banyaknya panggilan $2F_{n+1} - 1$ prosedur naif, pada skala logaritmik, sejajar dengan 1.6^n — pohonnya punya kira-kira 1.6^n daun, dan waktu jalannya eksponensial.

***Gambar 7.5**: bilangan Fibonacci yang berurutan mendekati ϕ, sehingga pohonnya memiliki sekitar 1.6^n daun. Buku: Subbab 7.3.5.*

Mengukurnya

Kode 7.1 menghitung setiap panggilan, mencatat berapa kali setiap $F(k)$ diminta, dan mengukur waktu rekursinya.

Kode 7.1 Rekursi Fibonacci naif, lengkap dengan alat ukurnya. Simpan sebagai `fib_naive.cpp`, kompilasi dengan `g++ -std=c++17 -O2 -o fib_naive fib_naive.cpp`, lalu jalankan `./fib_naive`.

```cpp
// fib_naive.cpp - rekursi yang lurus-lurus saja, dengan alat ukur terpasang.
// Mencacah tiap panggilan, mencacah seberapa sering tiap F(k) dihitung ulang, dan mengukur waktunya.
#include <iostream>
#include <iomanip>
#include <vector>
#include <chrono>
using namespace std;
using namespace std::chrono;

long long calls = 0;                 // jumlah seluruh pemanggilan fib()
vector<long long> perLevel;          // perLevel[k] = berapa kali F(k) dipanggil

long long fib(int n) {
    ++calls;
    if (n < (int)perLevel.size()) ++perLevel[n];
    if (n <= 1) return n;            // kasus basis F(0)=0, F(1)=1
    return fib(n - 1) + fib(n - 2);
}

int main() {
    // Bagian 1: profil panggilan fib(6), pohon yang digambar di Subbab 7.3.3.
    perLevel.assign(7, 0);
    calls = 0;
    long long v = fib(6);
    cout << "fib(6) = " << v << ", total calls = " << calls << "\n";
    cout << "calls to F(k), k = 0..6:";
    for (int k = 0; k <= 6; ++k) cout << " F(" << k << ")=" << perLevel[k];
    cout << "\n\n";
```

```
    // Bagian 2: pertumbuhan cacah panggilan dan waktu jam dindingnya.
    cout << left << setw(5) << "n" << setw(14) << "F(n)"
         << setw(14) << "calls" << setw(12) << "ms"
         << setw(10) << "calls/prev" << "\n";
    long long prevCalls = 0;
    for (int n = 20; n <= 42; n += 2) {
        perLevel.assign(0, 0);
        calls = 0;
        auto t0 = steady_clock::now();
        long long f = fib(n);
        auto t1 = steady_clock::now();
        double ms = duration<double, milli>(t1 - t0).count();
        cout << left << setw(5) << n << setw(14) << f << setw(14) << calls
             << setw(12) << fixed << setprecision(2) << ms;
        if (prevCalls) cout << setw(10) << setprecision(3) << (double)calls / prevCalls;
        cout << "\n";
        prevCalls = calls;
```

Kode 7.1 (Lanjutan) *Rekursi Fibonacci naif, lengkap dengan alat ukurnya. Simpan sebagai fib_naive.cpp, kompilasi dengan g++ -std=c++17 -O2 -o fib_naive fib_naive.cpp, lalu jalankan ./fib_naive.*

```
    }
    return 0;
}
```

```
fib(6) = 8, total calls = 25
calls to F(k), k = 0..6: F(0)=5 F(1)=8 F(2)=5 F(3)=3 F(4)=2 F(5)=1 F(6)=1

n    F(n)          calls         ms          calls/prev
20   6765          21891         0.05
22   17711         57313         0.12        2.618
24   46368         150049        0.34        2.618
26   121393        392835        0.86        2.618
28   317811        1028457       2.21        2.618
30   832040        2692537       5.83        2.618
32   2178309       7049155       16.27       2.618
34   5702887       18454929      40.88       2.618
36   14930352      48315633      108.97      2.618
38   39088169      126491971     280.85      2.618
40   102334155     331160281     735.49      2.618
42   267914296     866988873     1943.29     2.618
```

Profil `fib(6)` *adalah* 25 *panggilan yang terbagi* 5,8,5,3,2,1,1*; rasio dua langkah adalah* $2.618 = \phi^2$*;* F_{42} *memakan* 866 *juta panggilan dan hampir dua detik. Buku: Subbab 7.3.7.*

Perbaikannya, Secara Singkat

Gambar 7.6 Perbaikannya, secara ringkas: hitunglah F_n dalam waktu linear dengan *mengingat solusi submasalah yang sudah terselesaikan* — *dynamic programming* — membangun solusinya secara *bottom-up*, menukar ruang dengan waktu. Pasangan yang disorot itulah pengamatan yang akan menjadi §7.4.3: hanya dua nilai yang perlu diingat pada saat mana pun.

***Gambar 7.6**: ingat solusi submasalah yang sudah diselesaikan — dynamic programming, dinamai oleh Richard Bellman pada 1950-an — dan tukar ruang dengan waktu. Buku: Subbab 7.4.*

Perbaikan 1: Memoization

DEFINISI. ***Memoization*** menyimpan tabel jawaban yang sudah dihitung, memeriksanya di awal setiap panggilan, dan mengembalikan jawaban tersimpan alih-alih berekursi. Struktur rekursinya tetap; hanya pengulangannya yang hilang.

Pseudokode 7.2 Fibonacci dengan *memoization*: rekursi yang sama, yang memeriksa dan mengisi sebuah tabel jawaban.

```
vector<long long> memo;                    // memo[k] = F(k), atau -1 bila belum diketahui
long long fibMemo(int n) {
    if (memo[n] != -1) return memo[n];      // <-- sudah terselesaikan: jangan berekursi
    long long r = (n <= 1) ? n : fibMemo(n - 1) + fibMemo(n - 2);
    memo[n] = r;                            // <-- catat sebelum kembali
    return r;
}
```

TRIK PEMBUKUANNYA. Hitung entri tabelnya dan kalikan dengan biaya mengisi satu entri, *mengabaikan* panggilan rekursif di dalamnya (panggilan itu dibebankan ke entrinya sendiri): $n + 1$ entri $\times\, O(1) = \Theta(n)$.

Pseudokode 7.2 *melenyapkan setiap segitiga abu-abu dari pohon rekursi. Memoization berasal dari memo, catatan untuk diri sendiri — bukan "menghafal". Buku: Subbab 7.4.1.*

Perbaikan 2 dan 3: Bottom-Up, lalu Dua Variabel

Pseudokode 7.3 Fibonacci secara *bottom-up*: tabelnya diisi dari indeks terkecil ke atas.

```
long long fibTable(int n) {
    vector<long long> F(n + 1);
    F[0] = 0; if (n >= 1) F[1] = 1;
    for (int i = 2; i <= n; ++i) F[i] = F[i-1] + F[i-2];
    return F[n];
}
```

Pseudokode 7.3: hitung $F_0, F_1, F_2, \ldots$ secara berurutan, sehingga semua yang diperlukan sebuah entri sudah tersedia. Waktu dan ruang $\Theta(n)$, tanpa rekursi. **Urutan itulah seluruh seninya.**

Pseudokode 7.4 Fibonacci dalam ruang tetap: jendela geser berisi dua nilai menggantikan tabelnya.

```
long long fibIter(int n) {
    if (n <= 1) return n;
    long long prev = 0, cur = 1;            // prev = F(i-2), cur = F(i-1)
    for (int i = 2; i <= n; ++i) {
        long long next = prev + cur;         // F(i)
        prev = cur;                          // geser jendelanya satu langkah ke kanan
        cur  = next;
    }
    return cur;
}
```

Pseudokode 7.4: F_i hanya menjangkau dua tempat ke belakang, jadi jendela geser berisi dua nilai menggantikan tabelnya — waktu $\Theta(n)$, ruang $\Theta(1)$.

Trik jendela yang sama pada tabel dua dimensi memberi LCS dalam ruang $\Theta(min(m,n))$ nanti hari ini. Buku: Subbab 7.4.2 dan 7.4.3.

Jendelanya, dan Perbaikan 4: Pangkat Matriks

Tabel 7.1 Jendela geser `fibIter` sewaktu menghitung F_6. Sesudah iterasi terakhir `cur` memuat $8 = F_6$, dan tidak ada satu larik pun yang pernah dialokasikan.

i	`prev` sebelum	`cur` sebelum	`next` = prev+cur	`prev` sesudah	`cur` sesudah
2	0	1	1	1	1
3	1	1	2	1	2
4	1	2	3	2	3
5	2	3	5	3	5
6	3	5	8	5	8

Tabel 7.1 menelusuri jendelanya untuk F_6. Satu langkah rekurensi adalah pemetaan linear, yang menghasilkan Persamaan (7.6):

$$Q^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}, \tag{7.6}$$

Pengkuadratan berulang menghitung Q^n dalam $O(\lg n)$ operasi aritmetika — dihitung menurut model RAM; F_n yang sangat besar memerlukan bilangan bulat besar, sehingga programnya bekerja modulo 1 000 000 007.

F_n adalah entri kanan atas Q^n. Buku: Subbab 7.4.3 dan 7.4.4.

Keempatnya, Diukur

Tabel 7.2 Lima cara menghitung F_n dari bab ini, lengkap dengan biaya masing-masing. Baris pertama adalah yang harus dihindari; baris terakhir adalah yang menang ketika n besar.

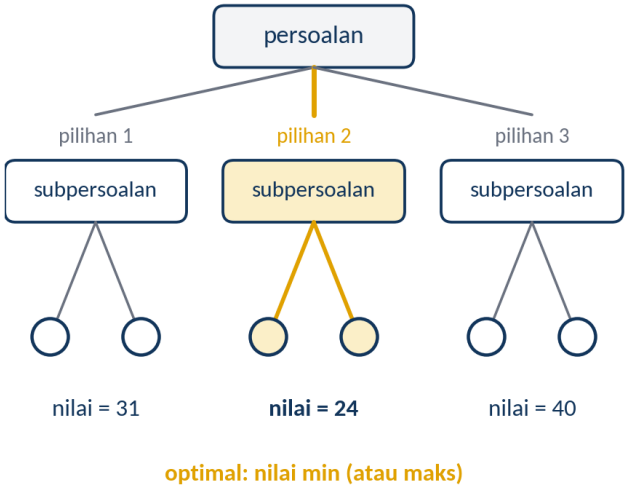
Metode	Waktu	Ruang	Catatan
Rekursi naif	$\Theta(\phi^n)$	tumpukan $\Theta(n)$	Tak terpakai melewati $n \approx 45$
Memoization (*top-down*)	$\Theta(n)$	$\Theta(n)$ + tumpukan $\Theta(n)$	Penulisan ulang termudah; terbatas tumpukan
Tabel *bottom-up*	$\Theta(n)$	$\Theta(n)$	Tanpa rekursi, tanpa risiko tumpukan
Jendela geser	$\Theta(n)$	$\Theta(1)$	Tak bisa memulihkan F_i antara
Pangkat matriks	$\Theta(\lg n)$	$\Theta(1)$	Perlu wawasan aljabar linear

Tabel 7.2, diukur: pada $n = 40$ *memoization* memakan 0.83 µs dibandingkan 735,490 µs untuk rekursi naif — sekitar 886,000 kali lebih cepat, hanya dari dua baris tambahan.

Memoization paling lambat di antara ketiga metode linear (ongkos panggilan) dan tumpukan panggilannya meluap pada $n = 1{,}000{,}000$; jendela geser mengalahkan tabel (tanpa lalu lintas memori); pangkat matriks memakan 0.13 µs pada $n = 10^6$.

Asimtotik mengatakan ketiga metode linear itu setara; mesin mengatakan lain. Kedua pernyataan itu benar. Buku: Subbab 7.4.5.

Persoalan Optimasi



- penyelesaian adalah untaian pilihan
- algoritma mengembalikan nilai optimal — dan, bila diminta, pilihan-pilihan yang mencapainya

Gambar 7.7 Persoalan optimasi secara garis besar: ada banyak solusi, satu di antaranya optimal, sebuah solusi adalah *rangkaian pilihan*, dan algoritmanya menghitung nilai optimal serta — bila kita memintanya — solusi optimalnya sendiri.

***Gambar 7.7**: banyak solusi layak, masing-masing bernilai; temukan sebuah solusi optimal. Solusi adalah untaian pilihan, dan algoritmanya mengembalikan nilai optimal dan, bila diminta, solusinya. Buku: Subbab 7.5 dan 7.5.1.*

Optimal Substructure, dan Potong-dan-Tempel

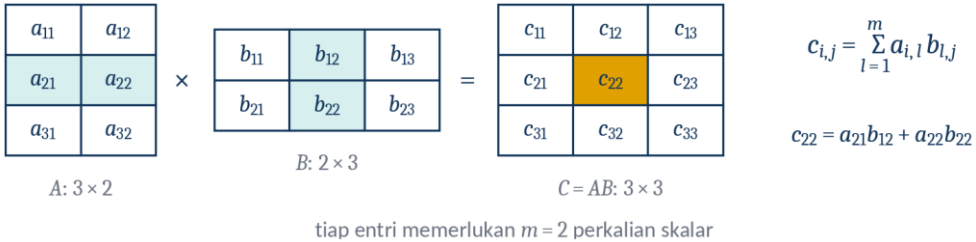
DEFINISI. Sebuah persoalan memiliki ***optimal substructure*** bila solusi optimalnya memuat solusi optimal bagi submasalah penyusunnya. (Bukan kebalikannya: potongan optimal dari belahan yang *salah* menyatu menjadi sampah — maka kita mencoba setiap belahan.)

POTONG-DAN-TEMPEL. Andaikan suatu potongan S_i tidak optimal: potong ia dan tempelkan S_i' yang lebih baik; hasilnya tetap **sah** dan lebih baik secara tegas daripada S — kontradiksi.

Tanpa substruktur: lintasan sederhana terpanjang. Pada bujur sangkar $q - r - t - s - q$, lintasan $q \to t$ terpanjang adalah $q \to r \to t$, tetapi lintasan $q \to r$ terpanjang adalah $q \to s \to t \to r$; menempelkannya mengulang simpul, jadi penempelannya tidak sah.

Buktikan optimal substructure — dan periksa bahwa penempelannya sah — sebelum menulis tabel apa pun. Lintasan sederhana terpanjang bersifat NP-hard. Buku: Subbab 7.5.2–7.5.4.

Biaya Satu Hasil Kali Matriks



Gambar 7.8 Mengalikan matriks $n \times m$ bernama A dengan matriks $m \times k$ bernama B, di sini dengan $n = 3, m = 2, k = 3$. Baris A yang diarsir dan kolom B yang diarsir bergabung menghasilkan satu entri terarsir c_{22} pada hasilnya, lewat rumus $c_{i,j} = \sum_{l=1}^{m} a_{i,l} \cdot b_{l,j}$.

***Gambar 7.8**: setiap entri hasil yang berjumlah $n \times k$ memerlukan m perkalian skalar. Buku: Subbab 7.6.1.*

Aturan Biaya, dan Mengapa Pengelompokan Penting

Entri (i, j) dari AB adalah Persamaan (7.7). Jadi hasil kali $n \times m$ dengan $m \times k$ berongkos $n \cdot m \cdot k$ perkalian skalar.

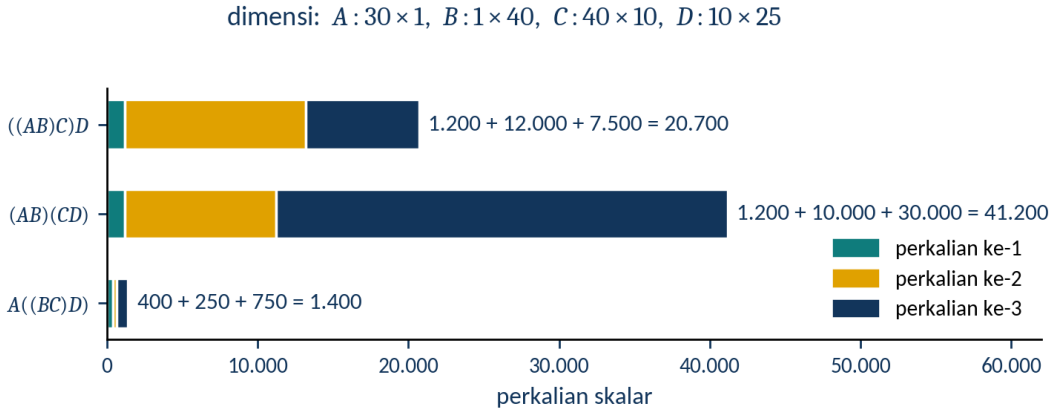
$$c_{i,j} = \sum_{l=1}^{m} a_{i,l} \cdot b_{l,j}. \tag{7.7}$$

A: 10×100, B: 100×5, C: 5×50. $(AB)C$ berongkos $5\,000 + 2\,500 = 7\,500$; $A(BC)$ berongkos $25\,000 + 50\,000 = 75\,000$. Jawaban sama, kerja sepuluh kali lipat.

> **PERSOALANNYA.** Diberikan dimensi $A_1, \ldots, A_n$, temukan pengurungan penuh yang meminimumkan total perkalian skalar. Keluarannya sebuah rencana — seperangkat kurung.

Perkalian matriks asosiatif dalam jawabannya, tidak dalam biayanya. Buku: Subbab 7.6.1 dan 7.6.2.

Sebuah Contoh Empat Matriks



Gambar 7.9 Sebuah contoh empat matriks. Untuk A (30×1), B (1×40), C (40×10), D (10×25), tiga di antara lima pengurungan yang mungkin berbiaya masing-masing 20 700, 41 200, dan 1 400; setiap batang memecah totalnya menjadi tiga perkaliannya. Baris terakhirnya menetapkan notasi yang dipakai mulai dari sini: kita mengelompokkan tanda kurung secara optimal atas $A_1 \times A_2 \times \cdots \times A_n$, dengan A_i berupa matriks $d_{i-1} \times d_i$.

***Gambar 7.9**: $d = \langle 30,1,40,10,25 \rangle$ dan A_i berukuran $d_{i-1} \times d_i$. CLRS menulis barisan yang sama sebagai p dan tabelnya sebagai m. Buku: Subbab 7.6.3.*

Tiga Pengurungan, Dihitung Harganya

$((AB)C)D$: $1\,200 + 12\,000 + 7\,500 = 20\,700$.

$(AB)(CD)$: $1\,200 + 10\,000 + 30\,000 = 41\,200$.

$A((BC)D)$: $400 + 250 + 750 = 1\,400$ — dua puluh sembilan kali lebih murah daripada yang terburuk.

> **MENGAPA YANG KETIGA MENANG.** B berukuran 1×40: memakainya lebih awal membuat setiap matriks antara hanya setinggi satu baris. Pengurungan yang baik menyuruh dimensi kecil yang bekerja.

Setiap angka cocok dengan gambarnya. Buku: Subbab 7.6.3.

Mengapa Tidak Mencoba Semuanya?

Perkalian terluar membelah rantai di suatu k, dan kedua sisinya dikurung secara bebas, yang menghasilkan Persamaan (7.8):

$$P(1) = 1, \qquad P(n) = \sum_{k=1}^{n-1} P\,(k)\,P(n-k) \quad (n \geq 2). \tag{7.8}$$

$P(2) = 1, P(3) = 2, P(4) = 5, P(5) = 14$: bilangan Catalan, $P(n+1) = C_n$ dengan Persamaan (7.9), tumbuh seperti $\Omega(4^n/n^{3/2})$.

$$C_n = \frac{1}{n+1}\binom{2n}{n}, \tag{7.9}$$

Pada $n = 60$, brute force akan berjalan lebih lama daripada umur alam semesta. Buku: Subbab 7.6.4.

Brute Force sebagai Kebenaran Acuan

Kode 7.2 mendaftar dan menghitung harga setiap pengurungan rantai $d = \langle 10,20,3,5,30 \rangle$ yang akan kita selesaikan dengan tangan berikut ini.

Kode 7.2 Pencacahan *brute force* atas setiap pengelompokan tanda kurung, dipakai sebagai kebenaran acuan untuk memeriksa hasil program dinamisnya. Kompilasi dengan `g++ -std=c++17 -O2 -o brute brute.cpp`.

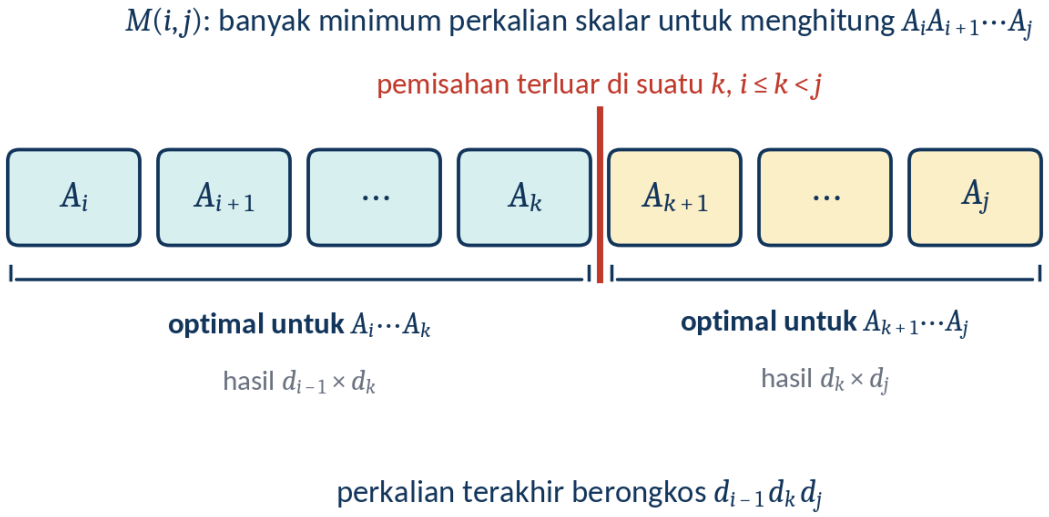
```cpp
// brute.cpp - cacah SETIAP pengelompokan tanda kurung sebuah deretan dan hitung harganya,
// supaya jawaban dynamic programming-nya dapat diperiksa terhadap kebenarannya.
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include <climits>
using namespace std;

vector<long long> d;

// Setiap cara membelah A_i..A_j, dengan biayanya dan pengelompokan yang dicetak.
vector<pair<long long,string>> allWays(int i, int j) {
    if (i == j) return {{0, "A" + to_string(i)}};
    vector<pair<long long,string>> out;
    for (int k = i; k < j; ++k)
        for (auto& L : allWays(i, k))
            for (auto& R : allWays(k + 1, j))
                out.push_back({L.first + R.first + d[i-1]*d[k]*d[j],
                               "(" + L.second + R.second + ")"});
    return out;
}

int main() {
    d = {10, 20, 3, 5, 30};
    int n = (int)d.size() - 1;
    auto ways = allWays(1, n);
    cout << "d = [10, 20, 3, 5, 30]: " << ways.size() << " parenthesizations\n\n";
```

```
    cout << left << setw(22) << "grouping" << right << setw(10) << "cost" << "\n";
    long long best = LLONG_MAX, worst = 0; string bestS, worstS;
    for (auto& w : ways) {
        cout << left << setw(22) << w.second << right << setw(10) << w.first << "\n";
        if (w.first < best)  { best  = w.first; bestS  = w.second; }
        if (w.first > worst) { worst = w.first; worstS = w.second; }
    }
    cout << "\ncheapest : " << bestS  << " at " << best  << "\n";
    cout << "dearest  : " << worstS << " at " << worst << "\n";
    cout << "ratio    : " << fixed << setprecision(2) << (double)worst / best << " times\n";
    return 0;
}
```

```
d = [10, 20, 3, 5, 30]: 5 parenthesizations

grouping                cost
(A1(A2(A3A4)))          8250
(A1((A2A3)A4))          9300
((A1A2)(A3A4))          1950
((A1(A2A3))A4)          2800
(((A1A2)A3)A4)          2250

cheapest : ((A1A2)(A3A4)) at 1950
dearest  : (A1((A2A3)A4)) at 9300
ratio    : 4.77 times
```

Lima pengurungan, seperti yang diramalkan $P(4) = 5$. Ingat 1 950 *dan* $((A_1A_2)(A_3A_4))$*: kita akan menurunkan keduanya tanpa mendaftar apa pun. Buku: Subbab 7.6.4.*

Dua Pengamatan Kunci



Gambar 7.10 Dua pengamatan kuncinya. Pertama, kurung terluar pasti membelah deretannya pada *suatu* posisi k dengan $i \le k < j$. Kedua, berapa pun nilai k itu, pengelompokan tanda kurung di kedua sisinya haruslah sendirinya optimal — *optimal substructure*, dan itulah pembenaran bagi rekurensinya.

***Gambar 7.10**: kurung terluar membelah di suatu k dengan $i \le k < j$, dan kedua sisinya sendiri harus optimal. Buku: Subbab 7.6.5.*

Submasalah dan Rekurensinya

Misalkan $M(i,j)$ adalah biaya minimum $A_i \cdots A_j$ — subrantai yang **bersebelahan**, karena hanya matriks yang berdampingan yang dapat dikurung. Hasil kali terakhir berongkos $d_{i-1}d_k d_j$ bagaimanapun kedua belahannya dikurung; potong-dan-tempel menunjukkan kedua belahan itu optimal. Maka Persamaan (7.10):

$$M(i,j) = \begin{cases} 0 & \text{jika } i = j, \\ \min\limits_{i \le k < j} \left\{ M(i,k) + M(k+1,j) + d_{i-1}\, d_k\, d_j \right\} & \text{jika } i < j. \end{cases} \tag{7.10}$$

TIGA JEBAKAN INDEKS. (1) $i \le k < j$, bukan $k \le j$. (2) Sisi kanan dimulai di $k+1$. (3) Tripelnya d_{i-1}, d_k, d_j — karena itu ada $n+1$ dimensi yang diindeks dari 0.

Dengan kata-kata: coba setiap tempat perkalian terakhir bisa terjadi; jumlahkan biaya kiri, biaya kanan, dan hasil kali terakhir; ambil yang terkecil. Buku: Subbab 7.6.5.

Rekurensi Rantai Matriks

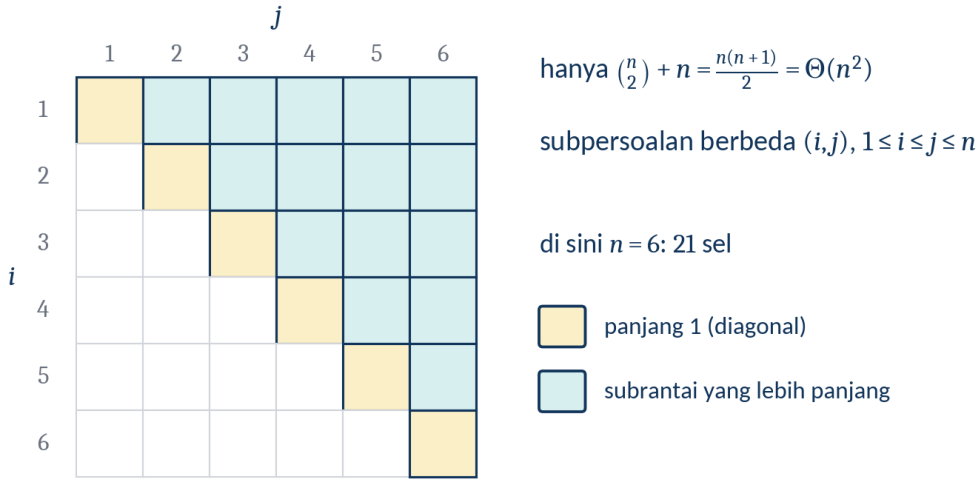
$$M(i,i) = 0$$

$$M(i,j) = \min_{i \le k < j} \{ M(i,k) + M(k+1,j) + d_{i-1}\, d_k\, d_j \}$$

Gambar 7.11 Rekurensi deretan matriks. Kasus basisnya adalah $M(i,i) = 0$; kasus umumnya meminimumkan, atas setiap titik belah k, biaya subderetan kiri ditambah biaya subderetan kanan ditambah $d_{i-1}d_k d_j$ untuk perkalian terakhirnya.

***Gambar 7.11**: kasus dasar $M(i,i) = 0$; satu matriks tidak memerlukan perkalian. Buku: Subbab 7.6.5.*

Hanya $\Theta(n^2)$ Submasalah



Gambar 7.12 Hitungan submasalahnya: hanya ada $\binom{n}{2} + n = \Theta(n^2)$ submasalah (i, j) yang berbeda dengan $1 \le i \le j \le n$. Segitiganya menggambarkan submasalah itu untuk $n = 6$: 21 sel pada dan di atas diagonal. Jurang antara $\Theta(n^2)$ pertanyaan dan kunjungan rekursif yang banyaknya eksponensial itulah celah yang dimanfaatkan *dynamic programming*.

***Gambar 7.12**: pasangan (i, j) dengan $1 \le i \le j \le n$. Untuk $n = 60$ itu 1 830 pertanyaan dibandingkan sekitar 4.1×10^{32} pengurungan. Buku: Subbab 7.6.6.*

Matrix-Chain-Order

Pseudokode 7.5 Matrix-Chain-Order dalam pseudokode. Baris 1–2 mengisi diagonalnya dengan nol. Baris 3 adalah yang paling penting: gelung terluarnya berjalan atas **panjang** deretan l, bukan atas i atau j, supaya setiap subderetan terselesaikan sebelum subderetan lebih panjang mana pun yang bersandar padanya. Baris 7–11 mencoba setiap belahan k dan menyimpan baik biaya terbaik di M maupun belahan pemenangnya di c.

```
MATRIX-CHAIN-ORDER(d[0..n]):
 1.  for i ← 1 to n
 2.      M[i,i] ← 0
 3.  for l ← 2 to n                              // l = panjang subrantai
 4.      for i ← 1 to n − l + 1
 5.          j ← i + l − 1
 6.          M[i,j] ← ∞
 7.          for k ← i to j − 1
 8.              q ← M[i,k] + M[k+1,j] + d[i−1]·d[k]·d[j]
 9.              if q < M[i,j] then
10.                  M[i,j] ← q
11.                  c[i,j] ← k
12.  return M, c
```

Pseudokode 7.5*: isi diagonal dengan nol, lalu menggelunglah atas* ***panjang*** *rantai l, lalu atas i dengan* $j = i + l - 1$*, mencoba setiap k dan mencatat pemenangnya di c. Buku: Subbab 7.6.7.*

Mengapa Menggelung atas Panjang

Banyaknya submasalah adalah Persamaan (7.11):

$$\binom{n}{2} + n = \frac{n(n-1)}{2} + n = \frac{n(n+1)}{2} = \Theta(n^2), \tag{7.11}$$

URUTANNYA. $M[i,j]$ memerlukan $M[i,k]$ dan $M[k+1,j]$, keduanya subrantai yang **lebih pendek**. Kerjakan subrantai menurut panjang yang menaik, dan semua yang diperlukan sebuah sel sudah siap. Menggelung atas i atau j dalam urutan alami membaca sel yang belum ditulis.

Tabel c tidak berongkos apa-apa untuk disimpan dan merupakan satu-satunya cara memulihkan pengurungannya, bukan hanya harganya.

Baris 6 mengisi $M[i,j]$ dengan ∞ agar kandidat pertama selalu menang. Buku: Subbab 7.6.6 dan 7.6.7.

Tabel dengan Tangan: $l = 2$ dan $l = 3$

$d = \langle 10,20,3,5,30 \rangle$: A_1 berukuran 10×20, A_2 20×3, A_3 3×5, A_4 5×30. Diagonal: semuanya 0.

$l = 2$: $M[1,2] = 10 \cdot 20 \cdot 3 = 600$; $M[2,3] = 20 \cdot 3 \cdot 5 = 300$; $M[3,4] = 3 \cdot 5 \cdot 30 = 450$.

$M[1,3]$: $k = 1$: $0 + 300 + 1\,000 = 1\,300$; $k = 2$: $600 + 0 + 150 = 750$. Jadi 750, $c = 2$.

$M[2,4]$: $k = 2$: $0 + 450 + 1\,800 = 2\,250$; $k = 3$: $300 + 0 + 3\,000 = 3\,300$. Jadi 2 250, $c = 2$.

Pada $M[1,3]$, membayar 600 lebih awal menyusutkan 20 menjadi 3, sehingga hasil kali terakhir hanya berongkos 150. Membayar lebih awal untuk menyusutkan dimensi adalah temanya. Buku: Subbab 7.6.8.

Tabel dengan Tangan: $l = 4$

$M[1,4]$: $k = 1$: $0 + 2\,250 + 6\,000 = 8\,250$; $k = 2$: $600 + 450 + 900 = 1\,950$; $k = 3$: $750 + 0 + 1\,500 = 2\,250$. Jadi $1\,950$, $c[1,4] = 2$ — jawaban *brute force*. **Tabel 7.3** dan **Tabel 7.4** adalah tabel-tabel yang sudah jadi.

Tabel 7.3 Tabel M yang sudah selesai untuk $d = \langle 10,20,3,5,30 \rangle$. Hanya segitiga atasnya yang bermakna, sebab sebuah subderetan haruslah punya $i \leq j$. Jawaban persoalan aslinya ada di pojok kanan atas, $M[1,4] = 1950$.

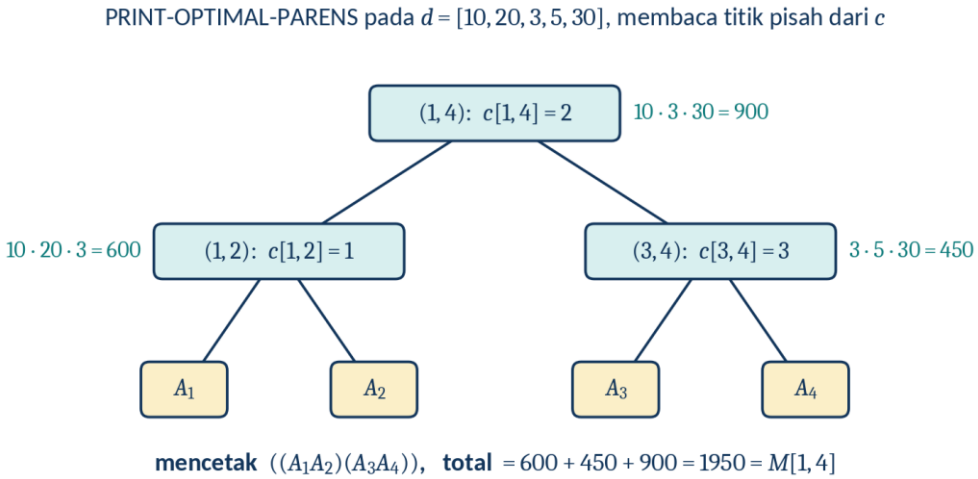
	j=1	j=2	j=3	j=4
$i = 1$	0	600	750	**1950**
$i = 2$	—	0	300	2250
$i = 3$	—	—	0	450
$i = 4$	—	—	—	0

Tabel 7.4 Tabel c yang sudah selesai: $c[i, j]$ mencatat titik belah k yang mencapai minimum pada $M[i, j]$. Ia satu-satunya catatan tentang *bagaimana* optimumnya dicapai, dan §7.6.9 membaca pengurungannya dari sana.

	j=1	j=2	j=3	j=4
$i = 1$	—	1	2	**2**
$i = 2$	—	—	2	2
$i = 3$	—	—	—	3
$i = 4$	—	—	—	—

Tabelnya terisi diagonal demi diagonal, ke atas dan ke kanan — persis yang dihasilkan gelung atas l. Buku: Subbab 7.6.8.

Nilai dan Pembagiannya



Gambar 7.13 Sesudah algoritmanya berjalan, $M[1, n]$ memuat nilai optimalnya dan c memuat pembagian optimalnya; sebuah prosedur rekursif sederhana **Print-Optimal-Parens**(c, i, j) menyusuri c dan mencetak pengurungannya. Pohonnya memperlihatkan penyusuran itu pada contoh yang baru saja kita selesaikan dengan tangan, $d = [10,20,3,5,30]$, dengan ongkos tiap perkalian di samping panggilannya.

***Gambar 7.13**: $M[1, n]$ menyimpan nilai optimal dan c menyimpan pembagian optimalnya; Print-Optimal-Parens menelusuri c. Buku: Subbab 7.6.9.*

Membaca Kembali Pengurungannya

Pseudokode 7.6 Print-Optimal-Parens: membaca pengurungan optimal dari tabel c.

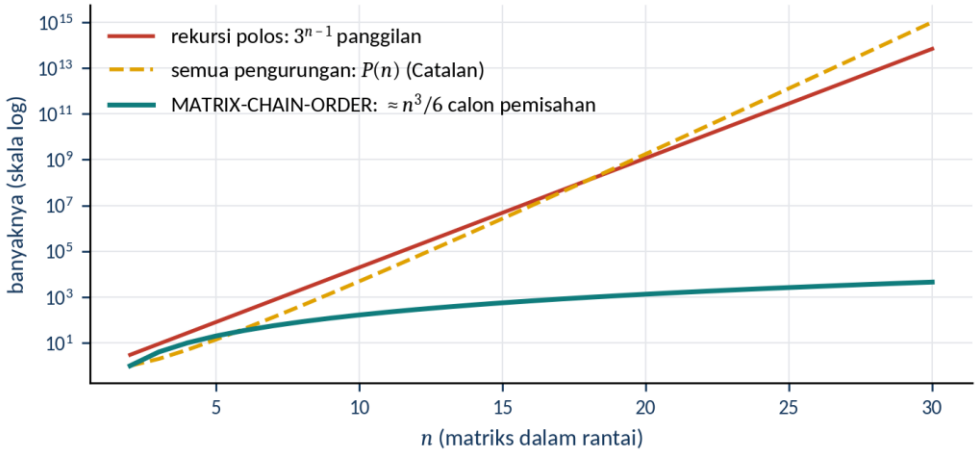
```
string printOptimalParens(int i, int j) {
    if (i == j) return "A" + to_string(i);          // matriks tunggal dicetak sebagai dirinya sendiri
    int k = c[i][j];                                 // di mana optimumnya membelah deretan ini
    return "(" + printOptimalParens(i, k)
               + printOptimalParens(k + 1, j) + ")";
}
```

Pseudokode 7.6 pada tabel c: `parens(1,4)` membelah di 2 menjadi `parens(1,2)` dan `parens(3,4)`; keduanya membelah di 1 dan 3; hasilnya $((A_1A_2)(A_3A_4))$.

Periksa dengan aturan biaya saja: $600 + 450 + 10 \cdot 3 \cdot 30 = 600 + 450 + 900 = 1\,950$ ✓ — tiga jalan yang saling bebas menuju satu angka.

Simpan pilihannya, bukan hanya nilainya: itulah langkah 4 resepnya. Buku: Subbab 7.6.9.

Waktu Jalannya



Gambar 7.14 Waktu jalan **Matrix-Chain-Order**: dengan mudah terlihat $O(n^3)$, dan sesungguhnya juga $\Omega(n^3)$, sehingga $\Theta(n^3)$. Grafiknya, pada skala logaritmik, menyandingkan banyaknya calon pemisahan itu dengan 3^{n-1} panggilan rekursi polos dan $P(n)$ pengurungan pencarian menyeluruh: *dari waktu eksponensial ke polinomial*, seluruh pokok bab ini dalam lima kata.

***Gambar 7.14**: $O(n^3)$ dan sesungguhnya juga $\Omega(n^3)$ — dari waktu eksponensial ke polinomial. Buku: Subbab 7.6.10.*

$\Theta(n^3)$, Dihitung dengan Tepat

Untuk setiap l, gelung tengah berjalan $n-l+1$ kali dan gelung dalam $l-1$ kali. Substitusi $t=l-1$ lalu penjumlahan menghasilkan Persamaan (7.12):

$$= \frac{3n^2(n-1)-(n-1)n(2n-1)}{6} = \frac{n(n-1)[3n-(2n-1)]}{6} = \frac{n(n-1)(n+1)}{6} = \frac{n^3-n}{6}. \qquad (7.12)$$

Untuk $n=4$: $(64-4)/6=10$ — persis sepuluh kandidat belahan pada penelusuran dengan tangan $(3+2\cdot 2+3)$.

Ruang: dua tabel $n \times n$, $\Theta(n^2)$. Pada $n=25$: sekitar 1.29×10^{12} pengurungan dibandingkan 2 600 kandidat belahan.

Kode 7.2 *dan tabel yang diisi dengan tangan saling cocok; program lengkap di buku mencetak kesepuluh kandidat belahan dan kedua tabelnya. Buku: Subbab 7.6.10 dan 7.6.11.*

Lookup-Chain: Mempertahankan Rekursinya

Pseudokode 7.7 Lookup-Chain: rekurensi deretan matriks yang ditulis sebagai rekursi, dengan baris 1 memotong langsung setiap submasalah yang sudah terselesaikan. Seluruh entri M diinisialisasi ke ∞ sebelumnya, dan ∞ sekaligus berperan sebagai penanda "belum terhitung".

```
LOOKUP-CHAIN(d, i, j):
 1.  if M[i,j] < ∞ then
 2.      return M[i,j]                          // sudah pernah diselesaikan: pakai lagi
 3.  if i = j then
 4.      M[i,j] ← 0
 5.  else for k ← i to j − 1
 6.      q ← LOOKUP-CHAIN(d, i, k) + LOOKUP-CHAIN(d, k+1, j) + d[i−1]·d[k]·d[j]
 7.      if q < M[i,j] then
 8.          M[i,j] ← q
 9.  return M[i,j]
```

***Pseudokode 7.7**: rekurensinya sebagai rekursi, dengan setiap entri diawali ∞, yang sekaligus berarti "belum dihitung". Buku: Subbab 7.7.1.*

Memoization Lawan Bottom-Up

Pseudokode 7.8 Baris pertama **Lookup-Chain**: kembalikan jawaban tersimpan alih-alih menghitungnya ulang.

```
1  if M[i,j] < ∞ then
2      return M[i,j]
```

Pseudokode 7.8 adalah satu-satunya pembeda dari bencana eksponensial. *Sentinel* harus **mustahil** menjadi jawaban sungguhan.

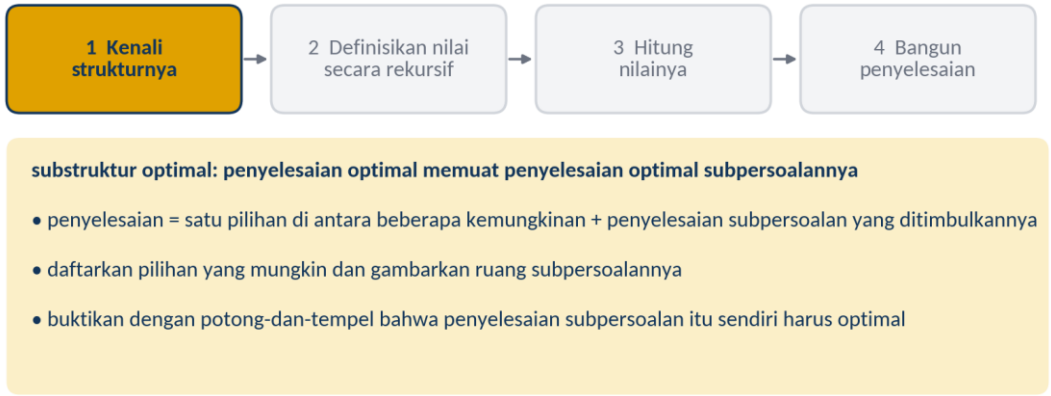
Tabel 7.5 *Memoization* melawan *bottom-up*. Keduanya memberi waktu jalan asimtotik yang sama; segala yang berbeda ada di dalam tabel ini.

	***Memoization* (*top-down*)**	**Tabel *bottom-up***
Urutan penghitungan	Ditemukan sendiri oleh rekursinya	Anda harus menentukannya dan mengkodekannya ke dalam gelung
Bentuk kode	Rekurensinya, nyaris disalin begitu saja	Gelung yang bisa jadi tidak menyerupai rekurensinya
Submasalah yang diselesaikan	Hanya yang benar-benar terjangkau	Semuanya, terjangkau atau tidak
Faktor tetapan	Lebih tinggi (panggilan, bingkai, akses tersebar)	Lebih rendah (gelung rapat, akses menyapu)
Tumpukan	Kedalamannya sebanding dengan kedalaman rekursi; bisa meluap	Tidak ada
Penghematan ruang	Sulit — seluruh tabelnya harus bertahan	Mudah — baris/diagonal lama boleh dibuang

Tabel 7.5, diukur: pada $n = 17$ rekursi polos membuat $3^{16} = 43{,}046{,}721$ panggilan (109 ms), *memoization* 1 633 (0.005 ms). Pada $n = 200$ *bottom-up* sekitar empat kali lebih cepat daripada *memoization*.

Bila ragu, tulis versi memoization lebih dulu, lalu ubah menjadi bottom-up dan periksa bahwa keduanya sepakat. Buku: Subbab 7.7.2 dan 7.7.3.

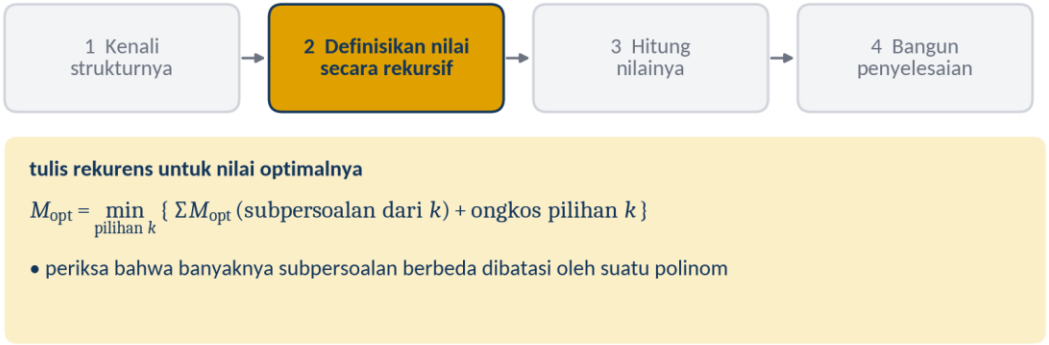
Resepnya, Langkah 1



Gambar 7.15 Langkah 1 resepnya: tunjukkan adanya *optimal substructure*. Sebuah solusi adalah satu pilihan dari sejumlah kemungkinan ditambah solusi bagi submasalah yang diciptakan pilihan itu; lalu tunjukkan, dengan potong-dan-tempel, bahwa solusi submasalah itu sendiri haruslah optimal.

***Gambar 7.15**: namai pilihannya; namai submasalah yang diciptakannya; buktikan dengan potong-dan-tempel bahwa solusi submasalah itu harus optimal. Buku: Subbab 7.8.1.*

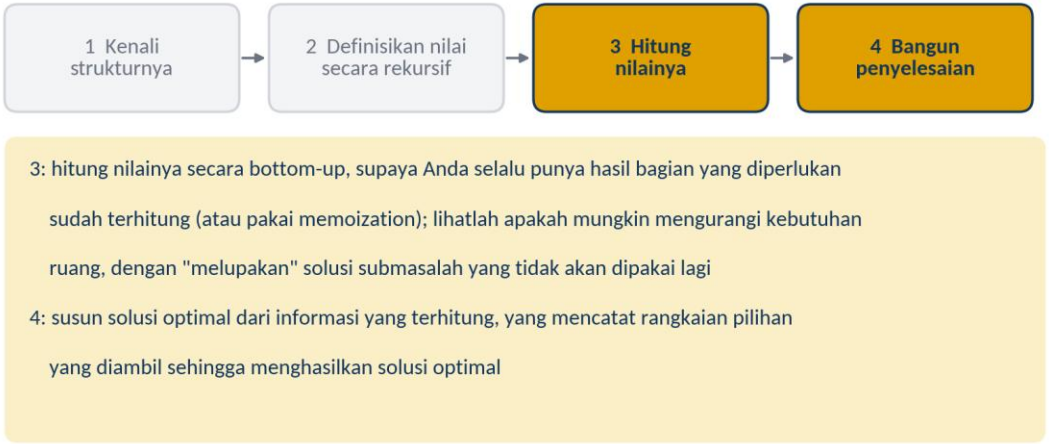
Resepnya, Langkah 2



Gambar 7.16 Langkah 2: tuliskan rekurensinya. Bentuk umumnya adalah $M_{opt} = \min_{\text{pilihan } k}\{(\text{jumlah } M_{opt} \text{ atas submasalah yang dihasilkan } k) + (\text{biaya memilih } k)\}$ — lalu tunjukkan bahwa banyaknya instans submasalah yang berbeda terbatasi oleh sebuah polinomial.

Gambar 7.16*: tulis rekurensinya, dan periksa bahwa banyaknya submasalah yang berbeda polinomial. Buku: Subbab 7.8.2.*

Resepnya, Langkah 3 dan 4



Gambar 7.17 Langkah 3 dan 4: hitung nilai optimalnya secara *bottom-up* supaya hasil bagian yang diperlukan selalu sudah siap (atau pakai *memoization*), carilah peluang mengurangi ruangnya dengan melupakan submasalah yang tidak akan dipakai lagi, dan akhirnya susunlah sebuah solusi optimal dari rangkaian pilihan yang tercatat.

***Gambar 7.17**: hitung secara bottom-up (atau dengan memoization), lupakan yang tidak lagi diperlukan, dan susun solusinya dari pilihan-pilihan yang tercatat. Buku: Subbab 7.8.3.*

Empat Langkah, dalam Satu Halaman

Setiap *dynamic programming* di buku ini adalah wujud dari templat Persamaan (7.13):

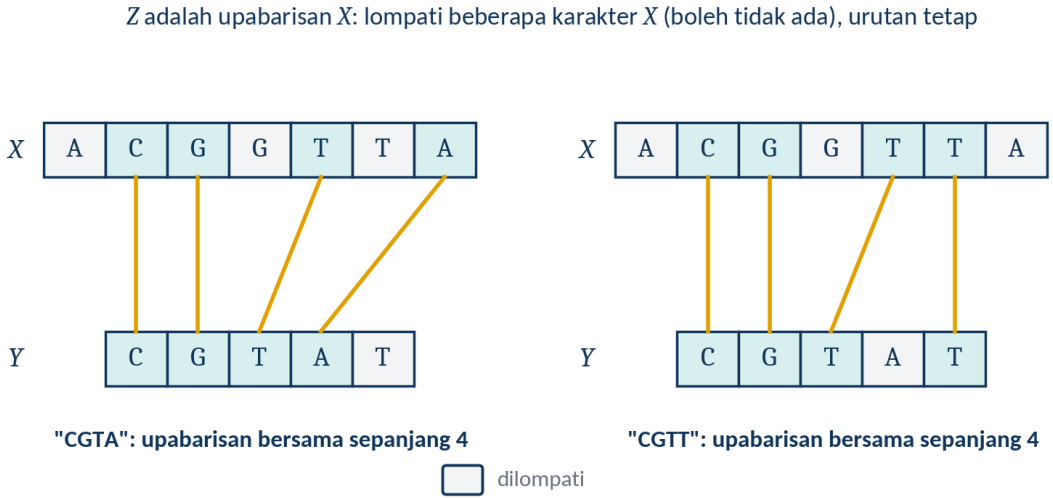
$$M_{\text{opt}}(\text{persoalan}) = \min_{\text{semua pilihan } k} \left\{ \sum_{\text{submasalah } s \text{ yang diciptakan } k} M_{\text{opt}}(s) + \text{biaya}(k) \right\}. \quad (7.13)$$

EMPAT LANGKAHNYA. 1. Cirikan — pilihan, submasalah, potong-dan-tempel. **2. Definisikan secara rekursif** — templat, kasus dasar, jumlah polinomial. **3. Hitung** — menurut urutan ketergantungan, atau dengan *memoization*. **4. Susun** — dari pilihan yang tercatat.

KEGAGALAN YANG PALING SERING. Menggambar tabel sebelum menamai submasalahnya. Selesaikan dulu kalimat "submasalahnya adalah: diberikan …, temukan yang terbaik …".

Rantai matriks cocok persis: pilihannya k, submasalahnya (i, k) dan $(k+1, j)$, biayanya $d_{i-1} d_k d_j$. Buku: Subbab 7.8.4.

Longest Common Subsequence



Gambar 7.18 Definisinya: Z adalah *subsequence* dari X bila Z dapat dibangkitkan dengan melewatkan sejumlah (boleh nol) karakter X. Untuk $X = $ “ACGGTTA” dan $Y = $ “CGTAT”, LCS(X, Y) adalah “CGTA” atau “CGTT”. Menyelesaikan persoalannya berarti menemukan *lewatan-lewatannya*.

***Gambar 7.18**: dipakai untuk membandingkan DNA, memeriksa ejaan, `diff`, mendeteksi plagiarisme — dan laundry kita hari ini. Buku: Subbab 7.9.1 dan 7.9.2.*

Subsequence, Didefinisikan dan Diperiksa

DEFINISI. Z adalah ***subsequence*** dari X bila diperoleh dengan mencoret sebagian karakter dan mempertahankan sisanya **menurut urutannya**. **LCS** dari X dan Y adalah *subsequence* bersama yang terpanjang — *sebuah*, bukan *satu-satunya*.

BUKAN SUBSTRING. *Substring* harus bersambung; *subsequence* tidak. "AGT" adalah *subsequence* dari "ACGGTTA" tetapi bukan *substring*-nya.

X = "ACGGTTA", Y = "CGTAT": "CGTA" dan "CGTT" merupakan *subsequence* bersama (panjang 4); "CGTAT" bukan. *Brute force* akan menguji 2^m *subsequence* — 2^{1000} untuk $m = 1000$.

Kita memerlukan dynamic programming. Buku: Subbab 7.9.2 dan 7.9.3.

Optimal Substructure pada LCS

Gambar 7.19 *Optimal substructure* milik LCS. Bekerja dari *ujung* X_m dan Y_n: jika karakter terakhirnya sama, karakter itu bergabung ke dalam jawabannya dan kita berekursi pada kedua untai yang masing-masing dipendekkan satu; jika berbeda, kita harus melewatkan satu huruf dari X atau satu huruf dari Y, dan kita memutuskan yang mana dengan membandingkan kedua panjang LCS yang dihasilkan. Pembenarannya adalah argumen potong-dan-tempel.

Gambar 7.19*: bekerjalah dari ujung string; bila karakter terakhirnya sama, ia masuk ke jawaban; bila tidak, lompati salah satunya. Buku: Subbab 7.9.4.*

Tiga Kasus, Dibuktikan

Misalkan X_i adalah prefiks $x_1 \cdots x_i$, dan $Z = z_1 \cdots z_r$ sembarang LCS dari X dan Y.

(1) Bila $x_m = y_n$ maka $z_r = x_m$ dan Z_{r-1} adalah LCS dari X_{m-1}, Y_{n-1} — kalau tidak, menambahkan x_m, atau menempelkan W yang lebih panjang, akan mengalahkan Z.

(2) Bila $x_m \neq y_n$ dan $z_r \neq x_m$, maka Z adalah LCS dari X_{m-1}, Y. **(3)** Secara simetris dengan Y_{n-1}.

Bila $x_m \neq y_n$, z_r tidak mungkin sama dengan keduanya, jadi (2) atau (3) berlaku: kasusnya lengkap, dan keadaan seri menjelaskan mengapa LCS bisa lebih dari satu.

Setiap buktinya adalah argumen potong-dan-tempel. Buku: Subbab 7.9.4.

Rekurensi LCS

$$c[i,j] = \begin{cases} 0 & \text{jika } i = 0 \text{ atau } j = 0 \\ c[i-1, j-1] + 1 & \text{jika } i,j > 0 \text{ dan } x_i = y_j \\ \max(c[i,j-1],\ c[i-1,j]) & \text{jika } i,j > 0 \text{ dan } x_i \neq y_j \end{cases}$$

Gambar 7.20 Rekurensi LCS: nol bila salah satu prefiksnya kosong; satu lebih banyak daripada tetangga diagonalnya ketika kedua karakternya cocok; selain itu yang lebih besar di antara tetangga di atas dan tetangga di kirinya.

Gambar 7.20*: nol pada prefiks kosong; diagonal ditambah satu bila cocok; selain itu yang lebih besar dari atas dan kiri. Buku: Subbab 7.9.5.*

Rekurensinya, Dituliskan

Dengan $c[i, j]$ panjang LCS dari X_i dan Y_j, rekurensinya adalah Persamaan (7.14); jawabannya $c[m, n]$.

$$c[i,j] = \begin{cases} 0 & \text{jika } i = 0 \text{ atau } j = 0, \\ c[i-1, j-1] + 1 & \text{jika } i, j > 0 \text{ dan } x_i = y_j, \\ \max\{c[i, j-1],\ c[i-1, j]\} & \text{jika } i, j > 0 \text{ dan } x_i \neq y_j. \end{cases} \tag{7.14}$$

DUA PENGINDEKSAN SEKALIGUS. Tabelnya mulai dari 0 (prefiks kosong); stringnya mulai dari 1. Di C++, karakter pada posisi tabel (i, j) adalah `X[i-1]` lawan `Y[j-1]`.

Ada $(m+1)(n+1) = \Theta(mn)$ submasalah — polinomial, seperti yang dituntut langkah 2. Buku: Subbab 7.9.5.

LCS-Length

Pseudokode 7.9 LCS-Length dalam pseudokode. Baris 1–4 mengisi baris nol dan kolom nolnya. Baris 5–15 menyapu tabelnya baris demi baris, mengisi $c[i,j]$ menurut rekurensinya dan mencatat di $b[i,j]$ kasus mana di antara ketiganya yang dipakai: `"copy"`, `"skipx"`, atau `"skipy"`.

```
LCS-LENGTH(X, Y, m, n):
 1.  for i ← 1 to m
 2.      c[i,0] ← 0
 3.  for j ← 0 to n
 4.      c[0,j] ← 0
 5.  for i ← 1 to m
 6.      for j ← 1 to n
 7.          if x_i = y_j then
 8.              c[i,j] ← c[i-1,j-1] + 1
 9.              b[i,j] ← "copy"
10.          else if c[i-1,j] ≥ c[i,j-1] then
11.              c[i,j] ← c[i-1,j]
12.              b[i,j] ← "skipx"
13.          else
14.              c[i,j] ← c[i,j-1]
15.              b[i,j] ← "skipy"
16.  return c, b
```

***Pseudokode 7.9**: baris demi baris, mencatat "copy" (↖), "skipx" (↑), atau "skipy" (←) di b. Baris 10 membandingkan dengan ≥, sehingga keadaan seri jatuh ke skipx. Buku: Subbab 7.9.6.*

Contoh yang Akan Diisi

Gambar 7.21 Tabel yang sudah terisi untuk $X = $ “ACGGTTA” dan $Y = $ “CGTAT”: tiap sel memuat $c[i, j]$ beserta panahnya $b[i, j]$, dan sel yang diarsir adalah penelusuran balik yang mengeja “CGTT”. Garis di sampingnya mengajukan pertanyaan §7.9.10: seberapa banyak ruang yang diperlukan jika kita tidak merekonstruksi subbarisannya sendiri?

***Gambar 7.21**: X = A, C, G, G, T, T, A ($m = 7$) dan Y = C, G, T, A, T ($n = 5$). Buku: Subbab 7.9.7.*

Tabel c, Sel demi Sel

Baris 1 (A): hanya $y_4 =$ A yang cocok, jadi $c[1,4] = 1$. Baris 2 (C): $c[2,1] = 1$ dari kecocokan dengan y_1. Baris 3 (G): $c[3,2] = c[2,1] + 1 = 2$. Baris 5 (T): $c[5,3] = 3$. Baris 6 (T): $c[6,5] = c[5,4] + 1 = 4$. Baris 7 (A): $c[7,4] = c[6,3] + 1 = 4$.

Tabel 7.6 Tabel c untuk \$X = \$ “ACGGTTA” dan \$Y = \$ “CGTAT”, diisi sel demi sel di §7.9.7. Setiap entri adalah panjang sebuah LCS dari kedua prefiks yang melabeli baris dan kolomnya. Jawabannya, $c[7,5] = 4$, adalah sel yang dicetak tebal.

	j=0	j=1 (C)	j=2 (G)	j=3 (T)	j=4 (A)	j=5 (T)
$i = 0$	0	0	0	0	0	0
$i = 1$ **(A)**	0	0	0	0	1	1
$i = 2$ **(C)**	0	1	1	1	1	1
$i = 3$ **(G)**	0	1	2	2	2	2
$i = 4$ **(G)**	0	1	2	2	2	2
$i = 5$ **(T)**	0	1	2	3	3	3
$i = 6$ **(T)**	0	1	2	3	3	4
$i = 7$ **(A)**	0	1	2	3	4	**4**

Tabel 7.6*: entrinya tidak pernah turun ke kanan maupun ke bawah, dan tetangganya berselisih paling banyak* 1*. Jawabannya* $c[7,5] = 4$*. Buku: Subbab 7.9.7.*

Panah-panahnya, dan Jalan Kembali

Tabel 7.7 Tabel panah b untuk instans yang sama. ↖ adalah `"copy"` (karakternya cocok), ↑ adalah `"skipx"` (nilainya diambil dari atas), ← adalah `"skipy"` (nilainya diambil dari kiri). Mengikuti panahnya mundur dari pojok kanan bawah akan merekonstruksi sebuah LCS.

	j=1 (C)	j=2 (G)	j=3 (T)	j=4 (A)	j=5 (T)
$i = 1$ **(A)**	↑	↑	↑	↖	←
$i = 2$ **(C)**	↖	←	←	↑	↑
$i = 3$ **(G)**	↑	↖	←	←	←
$i = 4$ **(G)**	↑	↖	↑	↑	↑
$i = 5$ **(T)**	↑	↑	↖	←	↖
$i = 6$ **(T)**	↑	↑	↖	↑	↖
$i = 7$ **(A)**	↑	↑	↑	↖	↑

Dari (7,5) pada **Tabel 7.7**: ↑ ke (6,5); ↖ mengeluarkan T; ← ke (5,3); ↖ mengeluarkan T; ↖ mengeluarkan G; ↑ ke (2,1); ↖ mengeluarkan C; berhenti pada $j = 0$.

Keluar terbalik sebagai T, T, G, C, jadi LCS(X, Y)= “CGTT”. Aturan seri ≥ memilihnya alih-alih “CGTA”; keduanya benar.

Jalan kembali ini memakan $O(m + n)$ — tak berarti dibandingkan pengisian $\Theta(mn)$. Buku: Subbab 7.9.8 dan 7.9.9.

Waktu $\Theta(mn)$, Ruang $\Theta(\min(m,n))$

Waktu: $m \cdot n$ sel, masing-masing $O(1)$ — tanpa gelung minimisasi, sehingga LCS kuadratik sedangkan rantai matriks kubik.

Ruang: $c[i,j]$ hanya membaca baris $i-1$ dan baris i, jadi dua baris sudah cukup; letakkan string yang lebih pendek pada kolom untuk memperoleh $\Theta(\min(m,n))$.

> **HASIL SOAL RUANGNYA.** *Panjangnya* hanya memerlukan ruang $\Theta(\min(m,n))$; *subsequence*-nya memerlukan tabel b (atau penghalusan Hirschberg). Diukur pada $m = n = 8000$: 488 MiB dan 1 931 ms untuk tabel penuh, 0.061 MiB dan 153 ms untuk dua baris.

Dua untai DNA sepanjang sejuta karakter: empat terabita untuk tabel penuh, sekitar delapan megabita untuk dua baris. Buku: Subbab 7.9.10 dan 7.9.11.

Submasalah yang Tumpang Tindih Lawan yang Saling Bebas

Dynamic programming memerlukan **kedua** ciri: ***optimal substructure*** (rekurensinya ada) dan ***overlapping subproblems*** (sedikit submasalah berbeda, dikunjungi berkali-kali).

Rasio panggilan terhadap submasalah berbeda yang diukur: Merge Sort 1.00 pada setiap ukuran; Fibonacci naif 7.44, 187.82, 6002; rantai matriks polos 2.70, 60.75, 2271.12.

Tabel 7.8 *Divide and conquer* melawan *dynamic programming*, berdampingan. Baris pertamanyalah yang menentukan mana yang sedang Anda hadapi; segala yang lain mengikutinya.

	Divide and conquer	***Dynamic programming***
Submasalah	Saling lepas, semuanya berbeda	Tumpang tindih, sedikit yang berbeda
Arah	Hanya *top-down*	*Top-down* dengan memo, atau *bottom-up*
Memori tambahan	Tidak ada di luar tumpukan	Sebuah tabel jawaban submasalah
Bentuk khas	$T(n) = aT(n/b) + f(n)$	nilai(s) = optimum atas pilihan
Diselesaikan oleh	Teorema Master (§3.13)	Mencacah sel × biaya per sel
Contoh	Merge Sort, pencarian biner, Strassen	Fibonacci, deretan matriks, LCS
Modus kegagalan	—	Ledakan eksponensial bila tidak di-*memoization*

Tabel 7.8*: baris pertama menentukan apa yang sedang Anda hadapi. Rasio mendekati* 1 *berarti biarkan saja; rasio yang terus tumbuh berarti pakai memoization atau tabel. Buku: Subbab 7.10.*

Profitina dalam Praktik: Mencocokkan Nota

Di Profitina, deteksi halaman hampir-kembar memakai kemiripan ala LCS dalam ruang dua baris; tampilan "apa yang berubah" pada usulan perbaikan teks adalah jalan kembali lewat panah; transformasi berantai adalah rantai matriks yang pendek.

Profitina Laundry mencocokkan nota tulisan tangan dengan daftar harga memakai Persamaan (7.15), sesudah diubah ke huruf kecil dan spasinya dibuang:

$$\text{kemiripan}(u, v) = \frac{|\text{LCS}(u, v)|}{\max(|u|, |v|)}, \tag{7.15}$$

`bedcover besar` bernilai 1.000; `cuci kring expres` 0.882; `cuci kering reg` 0.765. "titip motor sebentar" hanya bernilai 0.389, dan di bawah 0.600 aplikasinya menolak menebak lalu bertanya kepada orang.

Aplikasi memberi saran; pengguna yang memutuskan. Tidak ada medan data pribadi yang masuk ke perhitungan kemiripan (UU PDP 27/2022). Buku: Subbab 7.11 dan 7.11.1.

Kesalahan yang Sering Terjadi

Menyalahkan rekursi, bukan pengulangannya; **mengisi tabel dengan urutan yang salah** (menggelunglah atas panjang rantai).

Salah rentang k $(i \leq k < j)$; **mencampuradukkan *subsequence* dengan *substring***.

***Sentinel* yang bisa sama dengan jawaban sungguhan**; **melupakan tabel pilihan**, lalu memerlukan solusinya.

Salah satu indeks antara indeks string dan indeks tabel; **menganggap** *optimal substructure* alih-alih membuktikannya.

Rekursi *memoization* yang dalam meluapkan tumpukan panggilan; melaporkan LCS yang terbalik; menyimpan tabel yang tidak diperlukan; memakai *memoization* padahal tidak ada yang tumpang tindih.

Seluruhnya dua belas kesalahan, masing-masing dengan perbaikannya. Buku: Subbab 7.12.

Ringkasan: Lima Hal untuk Diingat

1. *Divide and conquer* menghendaki submasalah yang saling lepas; bila berulang, rekursi polos menjadi eksponensial — Fibonacci membuat $2F_{n+1} - 1 = \Theta(\phi^n)$ panggilan.

2. *Memoization* dan tabulasi *bottom-up* adalah dua wujud *dynamic programming*; analisislah dengan banyak sel × biaya per sel.

3. *Dynamic programming* memerlukan *optimal substructure* (potong-dan-tempel) dan *overlapping subproblems* (jumlah yang polinomial).

4. Rantai matriks: $M(i,j) = \min_{i \le k < j}\{M(i,k) + M(k+1,j) + d_{i-1}d_k d_j\}$, diisi menurut panjang, $\Theta(n^3)$; jawaban untuk $\langle 10,20,3,5,30 \rangle$ adalah 1 950.

5. LCS: diagonal $+1$ bila cocok, selain itu tetangga yang lebih besar; waktu $\Theta(mn)$; "CGTT" lewat panah; dua baris bila hanya panjangnya yang diperlukan.

Resep empat langkah berlaku juga untuk persoalan yang belum pernah Anda lihat. Buku: Subbab 7.13.

Pertemuan Berikutnya

Bab 8 — Evaluasi dan Pembahasan II. Dua puluh lima soal dari Bab 5 sampai 7, dikerjakan tuntas: *heap*, pemartisian, pohon pencarian, pohon AVL dan merah-hitam, rantai matriks, dan LCS.

Sebelum itu: Soal Latihan 7.7 (rantai enam matriks) dan 7.11 (tabel LCS), dengan tangan.

Kelak, Bab 9: algoritma *greedy* — *optimal substructure* ditambah sifat pilihan-rakus.

Bawalah Bab 5, 6, dan 7: setiap soal pada pertemuan berikutnya memakai salah satunya. Buku: Subbab 7.13.1 dan 7.14.

Kuliah 8 — Evaluasi dan Pembahasan II: Pengurutan, Pohon Pencarian, dan Pemrograman Dinamis

Perancangan dan Analisis Algoritma (EF234405)

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*, Bab 8

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini tanpa teori baru: kita memakai Bab 5, 6, dan 7 bersama-sama pada soal yang cukup besar sehingga beberapa gagasan harus bekerja sama. Buku: Bab 8, Subbab 8.1–8.21.

Yang Dibahas Pertemuan Ini

1. **Pengurutan** — Soal 8.1, *max-heap* yang dibangun dengan tangan; Soal 8.2, partisi Quicksort, kasus terbaik dan terburuk dihitung dengan tepat.
2. **Pohon pencarian** — Soal 8.3 sampai 8.6: `isBST`, `printDescending`, `max`, `isHeightBalanced`.
3. **Pemeliharaan AVL** — Soal 8.7 sampai 8.10: `insertHB`, kedua penyeimbang, `deleteHB`.
4. **Pohon merah-hitam** — Soal 8.11 sampai 8.15: empat pemeriksa dan dua rotasi.
5. **Pemrograman dinamis** — Soal 8.16, sebuah rantai matriks; Soal 8.17, SORTED lawan STORE.
6. **Pemeriksaan cepat, pohon statistik-urutan** (Soal 8.18 sampai 8.25) dan indeks Profitina Laundry yang mengaudit dirinya sendiri.

Setiap soal dinyatakan, dianalisis, diselesaikan selangkah demi selangkah, lalu diperiksa oleh program yang keluarannya dicetak di buku. Buku: Bab 8.

Di Mana Kita Berdiri

Bab 5: heapsort dan Quicksort. Bab 6: pohon pencarian biner dan pohon AVL. Bab 7: *memoization* dan pemrograman dinamis. Tabel 8.1 memetakan setiap soal hari ini ke subbab dan teorinya.

Tabel 8.1 Soal-soal bab ini, subbab yang menyelesaikan masing-masing, dan bab yang teorinya dipakai. Nomor soal dan nomor subbab adalah dua barisan yang berbeda; tabel ini menerjemahkan yang satu ke yang lain.

Soal	Subbab	Yang ditanyakan	Teori dari
8.1	8.3	Membangun *max-heap* dengan tangan, lalu mengurutkannya dengan *heapsort*	Bab 5
8.2	8.4	Melacak partisi Quicksort; kasus terbaik dan terburuk	Bab 5
8.3	8.6	`isBST`: apakah pohon biner ini pohon pencarian?	Bab 6
8.4	8.7	`printDescending`: mencetak BST dari yang terbesar, tanpa membangun daftar	Bab 6
8.5	8.8	`max`: kunci terbesar lewat satu lintasan saja	Bab 6
8.6	8.9	`isHeightBalanced`: syarat AVL dalam satu lintasan	Bab 6, 7
8.7	8.10	`insertHB`: penyisipan yang menjaga pohon tetap seimbang	Bab 6
8.8	8.11	`rebalanceForLeft`: kasus LL dan LR	Bab 6
8.9	8.12	`rebalanceForRight`: kasus RR dan RL	Bab 6
8.10	8.13	`deleteHB`: penghapusan yang menjaga pohon tetap seimbang	Bab 6
8.11–8.15	8.14	Pohon merah-hitam: empat pemeriksa dan dua rotasi	Bab 6
8.16	8.15	Perkalian rantai matriks, tabelnya diisi dengan tangan	Bab 7
8.17	8.16	*Longest common subsequence*, tabelnya dan jalan mundurnya	Bab 7
8.18–8.25	8.18	Soal lanjutan pada pohon statistik-urutan	Bab 5, 6, 7

> **SATU GAGASANNYA.** Bawalah informasi turun melalui rekursi, atau kembalikan ia ke atas, sehingga tidak ada satu pekerjaan pun yang dikerjakan dua kali.

Nomor soal dan nomor subbab adalah dua barisan yang berbeda; tabel ini menerjemahkan keduanya. Buku: Subbab 8.1.

Cara Mendekati Soal Bertingkat

Baca semua bagian lebih dulu dan gambarkan ketergantungannya. Soal 8.7 sampai 8.10 paling baik diselesaikan dengan urutan 8.8, 8.9, 8.7, 8.10 — setiap rutin cukup ditulis sekali.

Sebelum menulis kode, tuliskan lima hal: tanda tangan yang tepat; kasus dasar dalam kata-kata; invarian atau kontrak dalam satu kalimat; contoh gagal yang terkecil; klaim kompleksitas beserta alasannya dalam setengah kalimat.

Uji kasus kecil lebih dulu — kosong, satu elemen, dua elemen, kunci yang tidak ada — dan tuliskan apa yang belum ditangani.

> **ASAS PENCERMINAN.** Mencerminkan sebuah pohon menukar kiri dan kanan, `<` dan `>`, minimum dan maksimum, serta kedua rotasi. Kenali pasangan cerminnya dan pekerjaannya berkurang separuh.

Inilah kerangka lima fase Bab 1, dipadatkan untuk satu soal. Buku: Subbab 8.2.

Soal 8.1 — Membangun Max-Heap

$A = [5, 13, 2, 25, 7, 17, 20, 8, 4, 11]$: gambar pohonnya, jalankan `BUILD-MAX-HEAP` dengan setiap penukarannya, hitung pembandingan dan penukaran, urutkan hasilnya dengan heapsort, dan jelaskan batas $O(n)$.

Pseudokode 8.1 `MAX-HEAPIFY`: nilai di simpul i tenggelam sampai tak ada anak yang lebih besar daripadanya.

```
MAX-HEAPIFY(A, i)
1  l = LEFT(i)
2  r = RIGHT(i)
3  if l ≤ A.heap-size and A[l] > A[i]
4      largest = l
5  else largest = i
6  if r ≤ A.heap-size and A[r] > A[largest]
7      largest = r
8  if largest ≠ i
9      exchange A[i] with A[largest]
10     MAX-HEAPIFY(A, largest)
```

Pseudokode 8.2 `BUILD-MAX-HEAP`: memperbaiki simpul-simpul dalam dari bawah ke atas.

```
BUILD-MAX-HEAP(A, n)
1  A.heap-size = n
2  for i = ⌊n/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

Pseudokode 8.1 membiarkan satu nilai tenggelam; Pseudokode 8.2 memanggilnya pada simpul $\lfloor n/2 \rfloor, \ldots, 1$, menghitung *mundur* agar kedua subpohon sudah menjadi *heap* lebih dulu.

Di sini $n = 10$*: simpul internal* 1–5*, daun* 6–10*. Akar memuat* 5 *sedangkan* 25 *berada dua level di bawahnya. Buku: Subbab 8.3.1–8.3.3.*

Soal 8.1 — Penelusurannya, Satu Panggilan per Baris

$i=5$: 11>7, tukar. $i=4$: 25 sudah mendominasi. $i=3$: 17>2, lalu 20>17 — bandingkan dengan *juara saat ini* — tukar dengan 20. $i=2$: tukar 13 dengan 25. $i=1$: 5 tenggelam tiga level, melewati 25, 13, dan 8.

Tabel 8.2 `BUILD-MAX-HEAP` pada $A=[5,13,2,25,7,17,20,8,4,11]$, satu baris per panggilan `MAX-HEAPIFY`. Nilai di akar tenggelam tiga aras pada panggilan terakhir.

Panggilan	Nilai di simpul *i*	Perbandingan	Pertukaran	*A* Setelah panggilan
`MAX-HEAPIFY(A, 5)`	7	1	1	[5,13,2,25,11,17,20,8,4,7
`MAX-HEAPIFY(A, 4)`	25	2	0	[5,13,2,25,11,17,20,8,4,7
`MAX-HEAPIFY(A, 3)`	2	2	1	[5,13,20,25,11,17,2,8,4,7
`MAX-HEAPIFY(A, 2)`	13	4	1	[5,25,20,13,11,17,2,8,4,7
`MAX-HEAPIFY(A, 1)`	5	6	3	[25,13,20,8,11,17,2,5,4,7
Jumlah		**15**	**6**	

Tabel 8.2: 15 pembandingan dan 6 penukaran menghasilkan $[25,13,20,8,11,17,2,5,4,7]$. Periksa setiap induk: $25 \geq 13,20$; $13 \geq 8,11$; $20 \geq 17,2$; $8 \geq 5,4$; $11 \geq 7$ ✓.

Periksa diri: $[3,9,2,1,4,5]$ *menjadi* $[9,4,5,1,3,2]$ *sesudah* 7 *pembandingan dan* 3 *penukaran. Buku: Subbab 8.3.4 dan 8.3.8.*

Soal 8.1 — Heapsort, dan Mengapa $O(n)$

Pengambilan: $[20,13,17,8,11,7,2,5,4 \mid 25]$, lalu $[17,13,7,8,11,4,2,5 \mid 20,25]$, lalu $[13,11,7,8,5,4,2 \mid 17,20,25]$; sesudah sembilan kali, $[2,4,5,7,8,11,13,17,20,25]$.

Simpul bertinggi h berongkos $O(h)$, dan paling banyak $\lceil n/2^{h+1} \rceil$ simpul bertinggi h, sehingga diperoleh Persamaan (8.1):

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h} \right) = O(2n) = O(n), \tag{8.1}$$

Pada *heap* sempurna, total penenggelamannya tepat Persamaan (8.2) — diukur: $32,767$ kunci menaik memerlukan $32,752$ penukaran.

$$\sum_{\ell=0}^{k-1} 2^{\ell}(k-1-\ell) = 2^k - k - 1 = n - \lg(n+1). \tag{8.2}$$

Belokan keliru yang sering terjadi: menghitung gelung secara naik; membandingkan anak kanan dengan $A[i]$; lupa menyusutkan heap pada heapsort. Buku: Subbab 8.3.5–8.3.9.

Soal 8.2 — Partisi Quicksort

$A = [7, 2, 9, 4, 1, 8, 3, 6, 5]$: telusuri partisi Lomuto, lalu Quicksort seluruhnya; temukan masukan paling lambat dan paling cepat dan hitung pembandingannya dengan tepat.

Pseudokode 8.3 `PARTITION` Lomuto: satu sapuan dari kiri ke kanan yang menumbuhkan daerah nilai $\leq x$.

```
PARTITION(A, p, r)
1  x = A[r]                        // pivotnya
2  i = p − 1                       // A[p..i] memuat nilai ≤ x yang sudah ditemukan
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]     // jatuhkan pivot ke celahnya
8  return i + 1
```

Pseudokode 8.4 `QUICKSORT`: partisi, lalu rekursi pada kedua sisi pivot.

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

Pseudokode 8.3 menjaga empat wilayah — $\leq x$, $> x$, belum diperiksa, pivot; Pseudokode 8.4 tidak pernah menyentuh pivotnya lagi. Buku: Subbab 8.4.1 dan 8.4.2.

Soal 8.2 — Partisinya, Langkah demi Langkah

Tabel 8.3 `PARTITION` Lomuto pada $[7,2,9,4,1,8,3,6,5]$ dengan pivot $x=5$. Delapan perbandingan, empat pertukaran di dalam gelung dan satu di akhir; pivotnya berakhir di indeks $q=5$.

j	$A[j]$	$A[j]\le 5$?	i Sesudahnya	Pertukaran	A Setelah langkah itu
1	7	tidak	0	—	$[7,2,9,4,1,8,3,6,5]$
2	2	ya	1	$A[1]\leftrightarrow A[$	$[2,7,9,4,1,8,3,6,5]$
3	9	tidak	1	—	$[2,7,9,4,1,8,3,6,5]$
4	4	ya	2	$A[2]\leftrightarrow A[$	$[2,4,9,7,1,8,3,6,5]$
5	1	ya	3	$A[3]\leftrightarrow A[$	$[2,4,1,7,9,8,3,6,5]$
6	8	tidak	3	—	$[2,4,1,7,9,8,3,6,5]$
7	3	ya	4	$A[4]\leftrightarrow A[$	$[2,4,1,3,9,8,7,6,5]$
8	6	tidak	4	—	$[2,4,1,3,9,8,7,6,5]$

Tabel 8.3 (Lanjutan) *PARTITION Lomuto pada [7,2,9,4,1,8,3,6,5] dengan pivot x=5. Delapan perbandingan, empat pertukaran di dalam gelung dan satu di akhir; pivotnya berakhir di indeks q=5.*

j	*A*[*j*]	*A*[*j*]≤5?	*i* Sesudahnya	Pertukaran	*A* Setelah langkah itu
baris 7				$A[5] \leftrightarrow A[$	$[2,4,1,3,5,8,7,6,9]$

Tabel 8.3: pivot 5 mendarat di $q=5$ sesudah tepat $r-p=8$ pembandingan — jumlah yang tidak bergantung pada datanya.

Rekursinya: $[1..4]$ pivot 3 (3 pembandingan), $[1..2]$ pivot 1 (1), $[6..9]$ pivot 9 (3), $[6..8]$ pivot 6 (2), $[7..8]$ pivot 8 (1). Totalnya $8+3+1+3+2+1=18$.

Periksa diri: mempartisi $[3,8,1,6,2,7,5,4]$ *menghasilkan* $[3,1,2,4,8,7,5,6]$, $q=4$, *sesudah* 7 *pembandingan. Buku: Subbab 8.4.3, 8.4.4, dan 8.4.8.*

Soal 8.2 — Terburuk dan Terbaik, Dihitung dengan Tepat

Terburuk: kunci yang menaik membuat pivot selalu maksimum: $C(n)=C(n-1)+(n-1)=\frac{n(n-1)}{2}$, yaitu Persamaan (8.3).

$$T(n)=T(n-1)+\Theta(n)\Rightarrow T(n)=\Theta(n^2). \tag{8.3}$$

Terbaik: pivot median pada $n=2^k-1$ memberi $C(n)=2C\left(\frac{n-1}{2}\right)+(n-1)$, yaitu Persamaan (8.4), dengan jumlah tepat Persamaan (8.5) yang dibuktikan dengan induksi.

$$T(n)=2T(n/2)+\Theta(n)\Rightarrow T(n)=\Theta(n\lg n), \tag{8.4}$$

$$C(n)=(n+1)\lg(n+1)-2n. \tag{8.5}$$

Diukur pada $n=1023$: terurut $522,753$ pembandingan, terbaik $8,194$, acak $10,637$. Mengacak pivot menjadikan $\Theta(n\lg n)$ sebagai ongkos harapan pada setiap masukan. Buku: Subbab 8.4.5–8.4.9.

Tipe Data yang Dipakai Soal-Soal Pohon

DEFINISI. `MyTree` adalah pohon kosong ($\varnothing$) atau tripel (v, L, R). Tingginya mengikuti Persamaan (8.6), dan faktor keseimbangannya Persamaan (8.7) — negatif berarti berat kiri.

$$h(\varnothing) = -1, h((v, L, R)) = 1 + \max(h(L), h(R)). \tag{8.6}$$

$$bf((v, L, R)) = h(R) - h(L), \tag{8.7}$$

Kode 8.1 Tipe `MyTree` dalam C++, dengan satu-satunya konstruktor `node()` yang menjaga tinggi tersimpannya tetap jujur, dan sebuah pencetak bentuk yang menuliskan seluruh pohon dalam satu baris.

```cpp
// mytree.cpp -- tipe data MyTree yang menjadi dasar setiap soal pohon dalam bab ini.
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

struct MyTreeNode;
using MyTree = MyTreeNode*;              // penunjuk null ITULAH pohon kosong

struct MyTreeNode {
    int    value;                        // bilangan bulat yang disimpan di simpul ini
    MyTree left, right;                  // kedua subpohon
    int    h;                            // tinggi tersimpan; pohon kosong bertinggi -1
};

MyTree emptyTree()          { return nullptr; }
bool   isEmpty(MyTree t)    { return t == nullptr; }
int    height(MyTree t)     { return isEmpty(t) ? -1 : t->h; }
int    rootValue(MyTree t)  { return t->value; }
MyTree leftOf(MyTree t)     { return t->left; }
MyTree rightOf(MyTree t)    { return t->right; }

// node(v,l,r): satu-satunya konstruktor. Ia menghitung ulang tingginya, sehingga
// pohon yang dirakit hanya lewat node() tak pernah menyimpan tinggi yang basi.
MyTree node(int v, MyTree l, MyTree r) {
    MyTree t = new MyTreeNode;
    t->value = v; t->left = l; t->right = r;
    t->h = 1 + max(height(l), height(r));
```

```
    return t;
}
MyTree leaf(int v) { return node(v, emptyTree(), emptyTree()); }

int balanceFactor(MyTree t) { return height(t->right) - height(t->left); }

// shape(t) mencetak nilai(faktorKeseimbangan){kiri,kanan}; '.' adalah pohon kosong.
string shape(MyTree t) {
    if (isEmpty(t)) return ".";
    string s = to_string(t->value) + "(" + to_string(balanceFactor(t)) + ")";
    if (isEmpty(t->left) && isEmpty(t->right)) return s;      // daun tak perlu kurung kurawal
    return s + "{" + shape(t->left) + "," + shape(t->right) + "}";
}

int countNodes(MyTree t) {
    return isEmpty(t) ? 0 : 1 + countNodes(t->left) + countNodes(t->right);
}
```

Kode 8.1 (Lanjutan) *Tipe MyTree dalam C++, dengan satu-satunya konstruktor node() yang menjaga tinggi tersimpannya tetap jujur, dan sebuah pencetak bentuk yang menuliskan seluruh pohon dalam satu baris.*

```
void inorder(MyTree t) {
    if (isEmpty(t)) return;
    inorder(t->left);
    cout << t->value << ' ';
    inorder(t->right);
}

int main() {
    // Pohon contoh delapan simpul dari Bab 6, dirakit dengan tangan.
    MyTree t = node(13,
                    node(10, node(5, leaf(4), leaf(6)), leaf(11)),
                    node(15, emptyTree(), leaf(16)));

    cout << "shape   : " << shape(t) << "\n";
    cout << "nodes   : " << countNodes(t) << "\n";
    cout << "height  : " << height(t) << "\n";
    cout << "inorder : "; inorder(t); cout << "\n";
    cout << "empty   : height " << height(emptyTree())
         << ", isEmpty = " << (isEmpty(emptyTree()) ? "true" : "false") << "\n";
    return 0;
}
```

```
shape   : 13(-1){10(-1){5(0){4(0),6(0)},11(0)},15(1){.,16(0)}}
nodes   : 8
height  : 3
inorder : 4 5 6 10 11 13 15 16
empty   : height -1, isEmpty = true
```

Kode 8.1: `node()` adalah satu-satunya konstruktor dan selalu menghitung ulang tinggi, sehingga rotasi yang ditulis sebagai `node(...)` baru tidak mungkin meninggalkan tinggi yang basi. Buku: Subbab 8.5.

Soal 8.3 — `isBST`

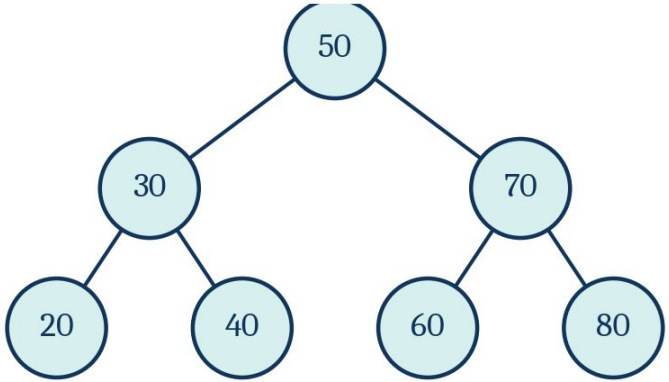
Tulis `bool isBST(MyTree t)` secara rekursif, dengan pembantu `isBST(t, lowerBound, upperBound)`: apakah t pohon pencarian biner yang semua nilainya berada tegas di antara kedua batas itu?

Pemeriksaan lokal — membandingkan setiap simpul dengan kedua anaknya — pendek, $\Theta(n)$, dan **salah**. Pohon: 20 dengan kiri $10\,[5, 25]$ dan kanan 30. Setiap uji induk–anak lolos, tetapi $25 > 20$ duduk di subpohon kiri 20; penelusuran inorder $5, 10, 25, 20, 30$ tidak naik.

JEBAKANNYA. Pemeriksaan lokal adalah syarat *perlu* tetapi bukan syarat *cukup*. Tiga simpul sudah cukup untuk menyingkapnya: 20, anak kiri 10, dan anak kanan 10 yaitu 25.

Bentuk jamak dalam definisinya — cabang-cabang, bukan anak-anak — adalah inti seluruh soal. Buku: Subbab 8.6.1–8.6.3.

Definisinya, Dibaca dengan Cermat



Gambar 8.1 Sebuah pohon pencarian biner yang menggambarkan definisinya: “Left branch(es) contains the nodes whose values $<$ the value of the node” — cabang-cabang, jamak, bukan anak.

Gambar 8.1: setiap simpul di mana pun pada subpohon kiri lebih kecil — bukan hanya anak kirinya. Buku: Subbab 8.6.2.

Soal 8.3 — Menurunkan Jawaban yang Benar

Pecah sifatnya di akar, Persamaan (8.8); kedua kuantornya adalah kendala rentang $(-\infty, v)$ dan $(v, +\infty)$.

$$\mathrm{BST}(t) \Leftrightarrow (\forall u \in L : u < v) \wedge (\forall u \in R : u > v) \wedge \mathrm{BST}(L) \wedge \mathrm{BST}(R). \tag{8.8}$$

Perumum menjadi $P(t, \ell, u)$: $\ell < v < u$, $P(L, \ell, v)$, dan $P(R, v, u)$; $P(\varnothing, \ell, u)$ benar; $BST(t) \Leftrightarrow P(t, -\infty, +\infty)$.

Tabel 8.4 Rekursi yang benar ditelusuri pada pohon jebakan $20\,[10\,[5, 25], 30]$. Batas $u = 20$ mencapai simpul 25 lewat *dua* aras rekursi; itulah keterangan yang dibuang oleh pemeriksaan lokal.

Panggilan	Simpul	ℓ	u	$\ell < v < u$?	Hasil
1	20	$-\infty$	$+\infty$	$-\infty < 20 < +\infty$ ✓	berekursi
2	10	$-\infty$	20	$-\infty < 10 < 20$ ✓	berekursi
3	5	$-\infty$	10	$-\infty < 5 < 10$ ✓	kedua upapohonnya kosong → **true**
4	25	10	20	$25 < 20$? **tidak**	**false**
5	—	—	—	—	panggilan 2 mengembalikan **false**
6	—	—	—	—	panggilan 1 mengembalikan **false**; simpul 30 tak pernah dikunjungi

Tabel 8.4: simpul 25 *ditolak oleh batas* 20 *yang diwarisi dari kakeknya. Setiap simpul memiliki selang pada ruang kunci — langkah yang sama dengan partisi Quicksort. Buku: Subbab 8.6.4 dan 8.6.5.*

Soal 8.3 — Programnya

Kode 8.2 memuat jawaban selang berbatas, pemeriksaan lokal sebagai pembanding, alternatif penelusuran inorder, dan diagnostik yang menyebut simpul pelanggar pertama.

Kode 8.2 Soal 8.3 selengkapnya: jawaban selang-terbatas, pemeriksaan lokal yang gagal, cara kedua lewat penelusuran inorder, dan sebuah diagnostik yang menyebut simpul pertama yang bersalah.

```cpp
// isbst.cpp -- Soal 8.3: isBST(), dan jawaban keliru yang menggoda.
#include <iostream>
#include <string>
#include <climits>
#include <algorithm>
using namespace std;

struct MyTreeNode;
using MyTree = MyTreeNode*;
struct MyTreeNode { int value; MyTree left, right; int h; };

MyTree emptyTree()       { return nullptr; }
bool   isEmpty(MyTree t) { return t == nullptr; }
int    height(MyTree t)  { return isEmpty(t) ? -1 : t->h; }
MyTree node(int v, MyTree l, MyTree r) {
    MyTree t = new MyTreeNode; t->value = v; t->left = l; t->right = r;
    t->h = 1 + max(height(l), height(r)); return t;
}
MyTree leaf(int v) { return node(v, emptyTree(), emptyTree()); }
string shape(MyTree t) {
    if (isEmpty(t)) return ".";
    string s = to_string(t->value);
    if (isEmpty(t->left) && isEmpty(t->right)) return s;
    return s + "{" + shape(t->left) + "," + shape(t->right) + "}";
}

// ---- Jawaban yang KELIRU: hanya memeriksa induk terhadap kedua anaknya. ----
bool isBSTLocal(MyTree t) {
```

```
    if (isEmpty(t)) return true;
    if (!isEmpty(t->left)  && t->left->value  >= t->value) return false;
    if (!isEmpty(t->right) && t->right->value <= t->value) return false;
    return isBSTLocal(t->left) && isBSTLocal(t->right);
}

// ---- Jawaban yang benar: bawa selang terbuka turun menyusuri rekursi. ----
// Setiap nilai di subpohon berakar t harus memenuhi lowerBound < v < upperBound.
// batas bertipe long long dipakai agar INT_MIN dan INT_MAX sah sebagai nilai pohon.
static bool isBST(MyTree t, long long lowerBound, long long upperBound) {
    if (isEmpty(t)) return true;                                  // kasus dasar
    if (t->value <= lowerBound || t->value >= upperBound) return false;
    return isBST(t->left,  lowerBound,   t->value)                // kiri: (lo, v)
        && isBST(t->right, t->value,     upperBound);             // kanan: (v, hi)
}
bool isBST(MyTree t) {
    return isBST(t, (long long)INT_MIN - 1, (long long)INT_MAX + 1);
}
```

Kode 8.2 (Lanjutan) *Soal 8.3 selengkapnya: jawaban selang-terbatas, pemeriksaan lokal yang gagal, cara kedua lewat penelusuran inorder, dan sebuah diagnostik yang menyebut simpul pertama yang bersalah.*

```
// ---- Cara kedua: penelusuran inorder harus menaik secara tegas. ----
static bool inorderIncreasing(MyTree t, long long& prev) {
    if (isEmpty(t)) return true;
    if (!inorderIncreasing(t->left, prev)) return false;
    if (t->value <= prev) return false;        // satu-satunya perbandingan yang menentukan
    prev = t->value;
    return inorderIncreasing(t->right, prev);
}
bool isBSTByWalk(MyTree t) {
    long long prev = (long long)INT_MIN - 1;
    return inorderIncreasing(t, prev);
}

// ---- Versi diagnostik: sebutkan simpul dan batas yang dilanggarnya. ----
static bool audit(MyTree t, long long lo, long long hi, string& why) {
    if (isEmpty(t)) return true;
    if (t->value <= lo) {
        why = "node " + to_string(t->value) + " must be > " + to_string(lo);
        return false;
    }
    if (t->value >= hi) {
        why = "node " + to_string(t->value) + " must be < " + to_string(hi);
        return false;
    }
    return audit(t->left, lo, t->value, why) && audit(t->right, t->value, hi, why);
}
```

```
void report(const char* name, MyTree t) {
    string why = "-";
    bool ok = audit(t, (long long)INT_MIN - 1, (long long)INT_MAX + 1, why);
    cout << name << "\n";
    cout << "  shape        : " << shape(t) << "\n";
    cout << "  isBSTLocal   : " << (isBSTLocal(t)  ? "true " : "false") << "   (the wrong method)\n";
    cout << "  isBST        : " << (isBST(t)       ? "true " : "false") << "\n";
    cout << "  isBSTByWalk  : " << (isBSTByWalk(t) ? "true " : "false") << "\n";
    cout << "  first fault  : " << (ok ? "none" : why) << "\n\n";
}

int main() {
    // BST yang sungguh-sungguh sah.
    report("good : root 20, left 10{5,15}, right 30{25,35}",
           node(20, node(10, leaf(5), leaf(15)), node(30, leaf(25), leaf(35))));
```

Kode 8.2 (Lanjutan) *Soal 8.3 selengkapnya: jawaban selang-terbatas, pemeriksaan lokal yang gagal, cara kedua lewat penelusuran inorder, dan sebuah diagnostik yang menyebut simpul pertama yang bersalah.*

```
    // Contoh sanggahan yang klasik: tiap pasangan induk-anak baik-baik saja, tetapi 25
    // berada di subpohon KIRI milik 20, padahal 25 > 20.
    report("trap : root 20, left 10{5,25}, right 30",
           node(20, node(10, leaf(5), leaf(25)), leaf(30)));

    // Nilai kembar. Definisi BST yang dipakai di sini melarang kunci yang sama.
    report("dup  : root 20, left 20, right 30",
           node(20, leaf(20), leaf(30)));

    // Satu simpul tunggal berisi INT_MIN: pengujian nilai penjaga.
    report("edge : the single node INT_MIN", leaf(INT_MIN));

    // Pohon kosong adalah BST menurut kesepakatan.
    report("empty: the empty tree", emptyTree());
    return 0;
}
```

```
good : root 20, left 10{5,15}, right 30{25,35}
  shape        : 20{10{5,15},30{25,35}}
  isBSTLocal   : true    (the wrong method)
  isBST        : true
  isBSTByWalk  : true
  first fault  : none

trap : root 20, left 10{5,25}, right 30
  shape        : 20{10{5,25},30}
  isBSTLocal   : true    (the wrong method)
  isBST        : false
  isBSTByWalk  : false
  first fault  : node 25 must be < 20

dup  : root 20, left 20, right 30
  shape        : 20{20,30}
  isBSTLocal   : false   (the wrong method)
  isBST        : false
  isBSTByWalk  : false
  first fault  : node 20 must be < 20

edge : the single node INT_MIN
  shape        : -2147483648
  isBSTLocal   : true    (the wrong method)
  isBST        : true
  isBSTByWalk  : true
  first fault  : none
```

```
empty: the empty tree
  shape        : .
  isBSTLocal   : true    (the wrong method)
  isBST        : true
  isBSTByWalk  : true
  first fault  : none
```

Pada pohon jebakan, pemeriksaan lokal berkata true, `isBST` berkata false, dan diagnostiknya melaporkan "node 25 must be < 20". Buku: Subbab 8.6.5.

Soal 8.3 — Perangkap `int`, dan Metode Kedua

Menggoda tetapi salah: mengisi batas `int` dengan `INT_MIN` dan `INT_MAX`. Pohon satu simpul berisi `INT_MIN` gagal pada `INT_MIN <= INT_MIN` dan ditolak. Perbaikannya: batas `long long`, batas `std::optional`, atau penelusuran inorder.

Metode kedua: sebuah pohon adalah pohon pencarian biner jika dan hanya jika penelusuran inordernya naik tegas. Bagikan `prev` lewat referensi dan bandingkan dengan `<=`.

Bukan sifat *heap*: aturan *heap* adalah induk terhadap anak — sungguh lokal — sehingga pemeriksaan lokal memang pemeriksa *heap* yang benar. Aturan pohon pencarian biner bersifat global.

Belokan keliru, menurut seringnya: pemeriksaan lokal; batas tak tegas; batas tertukar; tanpa kasus dasar; pembantu yang terbuka; sentinel `int`. Buku: Subbab 8.6.6–8.6.9.

Soal 8.4 — `printDescending`, Tanpa Daftar

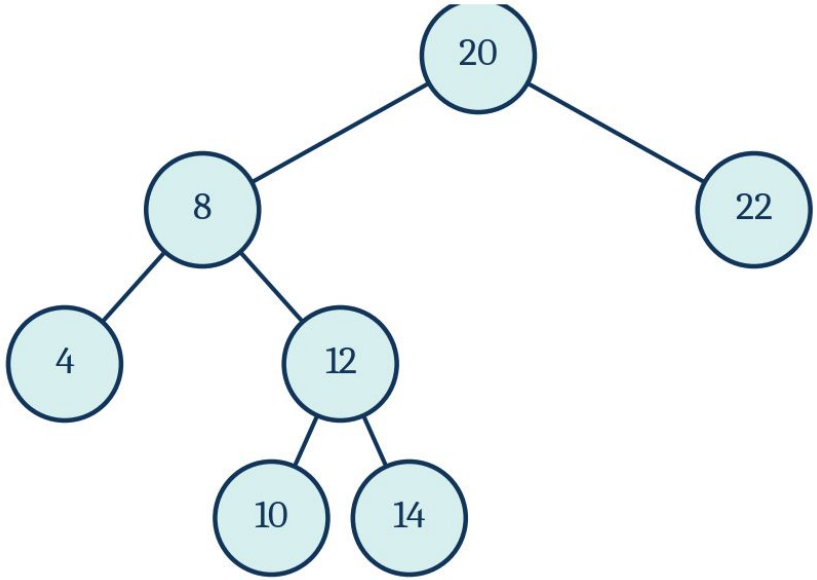
Cetak pohon pencarian biner dari yang terbesar, **tanpa membuat daftar terpisah berisi nilai-nilainya**. Inorder (kiri, simpul, kanan) menaik, jadi cerminannya — **kanan, simpul, kiri** — menurun.

$printDescending(\varnothing)$ tidak melakukan apa-apa; selain itu $printDescending(R)$, cetak v, $printDescending(L)$. Empat baris, waktu $\Theta(n)$, tumpukan $O(h)$.

Pada pohon delapan simpul: `16 15 13 11 10 6 5 4`. Versi iteratif mendorong setiap tulang punggung kanan ke tumpukan berisi paling banyak $h+1$ simpul yang menunggu — tumpukan rekursi yang dibuat eksplisit, bukan daftar.

Penelusuran adalah tiga pernyataan dalam suatu urutan; "menurun" berarti memindahkan satu baris. Uji pada $10\,[5,15]$*: jawabannya harus* `15 10 5`*. Buku: Subbab 8.7.*

Penerus Inorder



Gambar 8.2 Penerus inorder. Merangkai para penggantinya — 8, 10, 12, 14, 20 — menjejakkan urutan terurut yang dihasilkan penelusuran inorder. Membalik rangkaiannya memberikan urutan menurun.

Gambar 8.2: merangkai penerus-penerusnya menelusuri urutan terurut; membalik rangkaian itu memberi urutan menurun. Buku: Subbab 8.7.2.

Soal 8.5 — `max` Lewat Satu Lintasan

Kembalikan nilai terbesar **tanpa mengunjungi semua simpul**. Setiap nilai di subpohon kiri lebih kecil daripada v, jadi buang subpohon kiri tanpa diperiksa; bila subpohon kanan tidak kosong, jawabannya maksimum subpohon itu, selain itu v.

Jalannya mengikuti **tulang punggung kanan**: paling banyak $h{+}1$ simpul, jadi $O(h)$. Diukur pada pohon seimbang 15 simpul: 4 simpul dibandingkan 15 untuk pemindaian penuh.

Nyatakan batasnya agar benar: $O(h)$ selalu; $O(\lg n)$ hanya pada pohon seimbang. Pada rantai sepuluh simpul yang condong ke kiri, `min` mengunjungi kesepuluhnya.

Pohon kosong tidak punya maksimum: lemparkan pengecualian, atau dokumentasikan prasyaratnya — tetapi jangan pernah mengembalikan `INT_MIN`, yang merupakan nilai sah. Buku: Subbab 8.8.

Soal 8.6 — Syarat AVL

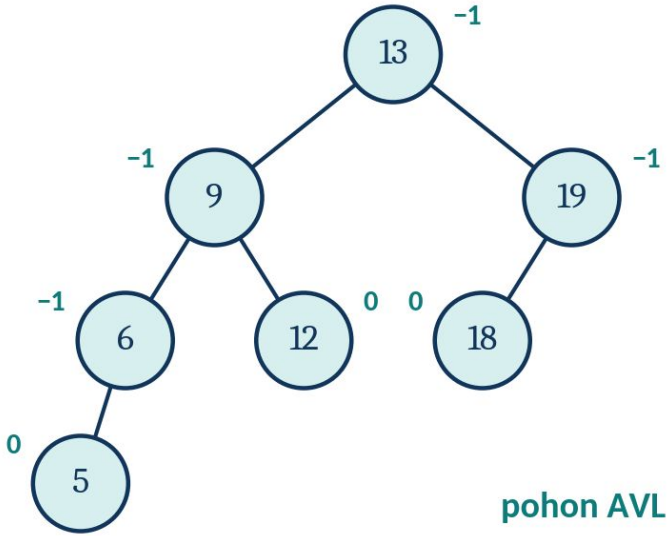
DEFINISI. Sebuah pohon **seimbang-tinggi** bila Persamaan (8.9) berlaku, dengan pohon kosong bertinggi -1. Pohon kosong seimbang-tinggi.

$$\left|bf(x)\right| \leq 1 \text{ untuk setiap simpul } x \text{ dari } t, \tag{8.9}$$

Tulis `bool isHeightBalanced(MyTree t)`, dengan sasaran satu lintasan saja. Dua halaman berikut menunjukkan satu pohon dari masing-masing jenis.

Syaratnya menyangkut setiap simpul; memeriksa akar saja adalah kesalahan pemeriksaan lokal lagi. Buku: Subbab 8.9.1 dan 8.9.2.

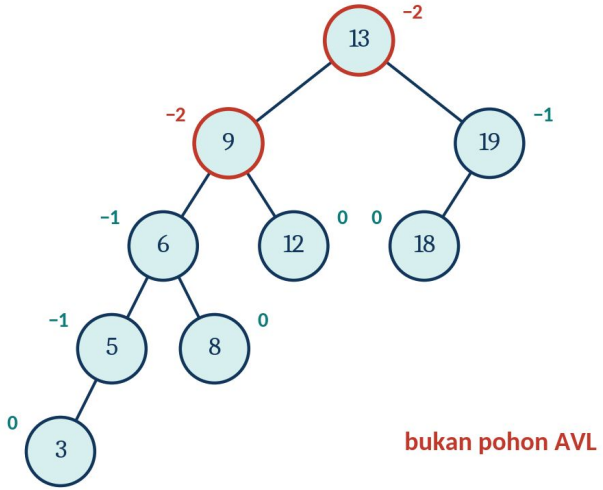
Sebuah Pohon AVL



Gambar 8.3 Pohon yang **merupakan** pohon AVL. Kedua subpohon tiap simpulnya berselisih tinggi paling banyak satu.

Gambar 8.3: simpul 6 *ber-*$bf = -1$*, simpul* 9 − 1*, simpul* 19 − 1*, akar* − 1*; setiap daun* 0*. Buku: Subbab 8.9.2.*

Dan Satu yang Bukan



Gambar 8.4 Pohon yang **bukan** pohon AVL. Menggantungkan simpul 3 di bawah 5 mendorong dua faktor keseimbangan, milik 9 dan 13, menjadi -2, di luar jangkauan yang sah.

Gambar 8.4: dengan 3 *di bawah* 5*, simpul* 6 *bertinggi* 2*, sehingga simpul* 9 *ber-*$bf = 0 - 2 = -2$*. Buku: Subbab 8.9.2 dan 6.14.*

Soal 8.6 — Berapa Ongkos Jawaban yang Jelas

Menghitung kedua tinggi subpohon di setiap simpul memang benar, tetapi membuang setiap tinggi itu. Pada pohon sempurna ia memeriksa $\sum_x size(x)$ simpul, tepat Persamaan (8.10):

$$W = (n+1)\lg(n+1) - n. \tag{8.10}$$

$n=7$: 17; $n=15$: 49; $n=127$: 769 — diukur dengan tepat. Itu $\Theta(n \lg n)$, faktor $\lg n$ di atas satu lintasan.

> **BUKAN $\Theta(n^2)$ DI SINI.** Pohon yang lolos itu pendek, dan pohon yang gagal keluar lebih awal. Rutin yang sungguh kuadratik adalah yang memberi label *setiap* simpul pada rantai: $8256 = \frac{128 \cdot 129}{2}$ untuk $n=128$.

Pelajaran Bab 7 berlaku: jangan pernah menghitung ulang apa yang sudah Anda ketahui. Buku: Subbab 8.9.3.

Soal 8.6 — Satu Lintasan, dengan Penjaga

Kembalikan tingginya dan pakai -2 — di bawah setiap tinggi sungguhan — untuk berarti “tidak seimbang di bawah sini”, yang memberi Persamaan (8.11); pohonnya seimbang jika dan hanya jika $ch(t) \neq -2$.

$$\mathrm{ch}(\emptyset) = -1; \qquad \mathrm{ch}((v, L, R)) = \begin{cases} -2 & \text{jika } \mathrm{ch}(L) = -2 \\ -2 & \text{jika } \mathrm{ch}(R) = -2 \\ -2 & \text{jika } |\mathrm{ch}(R) - \mathrm{ch}(L)| > 1 \\ 1 + \max(\mathrm{ch}(L), \mathrm{ch}(R)) & \text{jika tidak.} \end{cases} \tag{8.11}$$

Tabel 8.5 Pemeriksaan simpul terukur berhadapan dengan bentuk tertutupnya. Pemeriksaan keseimbangan yang naif cocok persis dengan $(n+1)\lg(n+1)-n$ pada pohon sempurna, cara satu-lintasan cocok dengan n, dan rutin pelabelan tingginya cocok persis dengan $n(n+1)/2$ pada rantai.

Pohon	n	Lambat, terukur	$(n+1)\lg(n+1)-n$	Cepat, terukur	Pelabelan, terukur	$n(n+1)/2$
sempurna	7	17	17	7	17	—
sempurna	15	49	49	15	49	—
sempurna	31	129	129	31	129	—
sempurna	63	321	321	63	321	—
sempurna	127	769	769	127	769	—
rantai	16	16	—	16	136	136
rantai	64	64	—	64	2080	2080
rantai	128	128	—	128	8256	8256

Tabel 8.5 memastikan hitungannya. Teorema Master sepakat: satu lintasan adalah Persamaan (8.12), kasus 1, $\Theta(n)$; pemeriksaan lambat adalah Persamaan (8.13), kasus 2, $\Theta(n\lg n)$.

$$T(n)=2T(n/2)+\Theta(1). \tag{8.12}$$

$$T(n)=2T(n/2)+\Theta(n), \tag{8.13}$$

Memilih -1 sebagai penanda adalah kesalahan klasik: itu tinggi pohon kosong. Buku: Subbab 8.9.4–8.9.7.

Soal 8.7 — `insertHB`

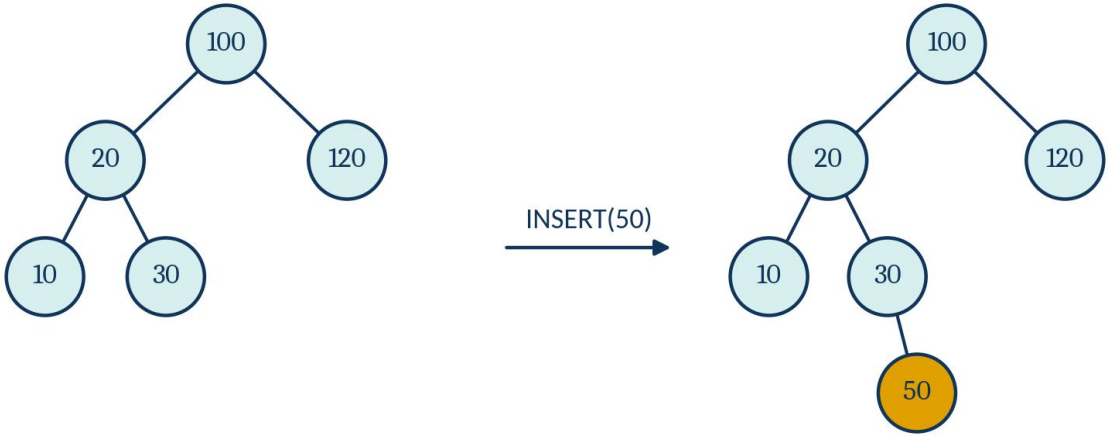
KONTRAK (`rebalanceForLeft(t)`). t tidak kosong; kedua subpohonnya seimbang; subpohon kiri mungkin bertambah tinggi satu takik, sehingga $bf(t) \in \{-2,-1,0,+1\}$. Ia mengembalikan pohon seimbang dengan nilai yang sama, dalam $O(1)$. `rebalanceForRight` adalah cerminannya.

Penyisipan biasa membangun ulang lintasannya; bungkuslah setiap simpul yang dibangun ulang dengan penyeimbang — sisi yang mungkin bertambah tinggi — dan Anda mendapatkan Persamaan (8.14):

$$\text{insertHB}(n,(v,L,R)) = \begin{cases} \text{rebalanceForLeft}(v,\text{insertHB}(n,L),R) & n < v \\ \text{rebalanceForRight}(v,L,\text{insertHB}(n,R)) & n > v \\ (v,L,R) & n = v \end{cases} \tag{8.14}$$

Tanda tangannya mengembalikan `MyTree` karena penyeimbangan dapat mengganti akar subpohon. Ongkos $O(h)=O(\lg n)$, paling banyak dua rotasi per penyisipan. Buku: Subbab 8.10.1–8.10.3.

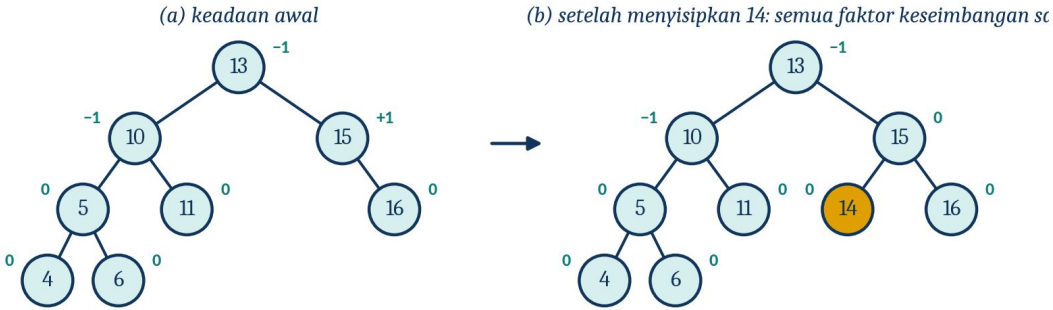
Kunci Baru Selalu Menjadi Daun



Gambar 8.5 Penyisipan ke dalam BST biasa: elemen baru selalu disisipkan sebagai daun. Penurunannya adalah pencariannya, dan pencariannya adalah penempatannya.

Gambar 8.5: penurunannya adalah pencarian, dan pencarian itulah penempatannya. Buku: Subbab 8.10.3.

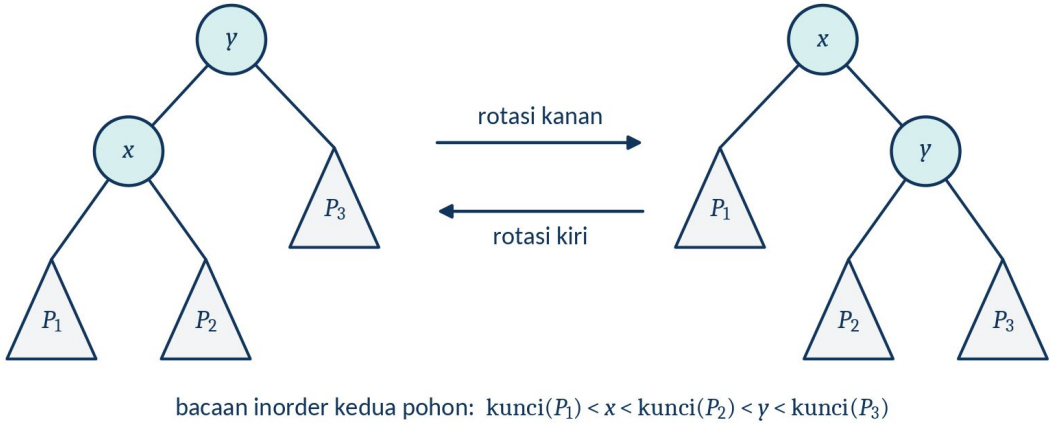
Menyisipkan 14: Tanpa Rotasi



Gambar 8.6 Menyisipkan 14 ke dalam pohon delapan simpul. Penyisipannya terjadi, faktor keseimbangannya dihitung ulang menaiki lintasannya, dan tak satu pun keluar dari jangkauan yang sah — sehingga `insertHB` melakukan nol rotasi. Inilah kasus yang lazim.

Gambar 8.6: setiap faktor keseimbangan pada lintasannya tetap di $\{-1, 0, +1\}$, jadi `insertHB` melakukan nol rotasi — kasus yang paling umum. Buku: Subbab 8.10.4.

Soal 8.8 — Satu-satunya Perkakas: Rotasi



Gambar 8.7 Kedua rotasi. Pada kedua gambar, bacaan inordernya adalah P_1, x, P_2, y, P_3, sebagaimana dinyatakan baris di bawah gambarnya. Rotasi kanan mengubah gambar kiri menjadi gambar kanan; rotasi kiri adalah kebalikannya yang persis.

Gambar 8.7: kedua bentuk dibaca P_1, x, P_2, y, P_3 secara berurutan, jadi keduanya pohon pencarian biner dengan nilai yang sama. Buku: Subbab 8.11.2.

Soal 8.8 — `rebalanceForLeft`

Dalam gaya fungsional, rotasi kanan adalah satu ekspresi, Persamaan (8.15); `node` membereskan urutan tinggi secara otomatis.

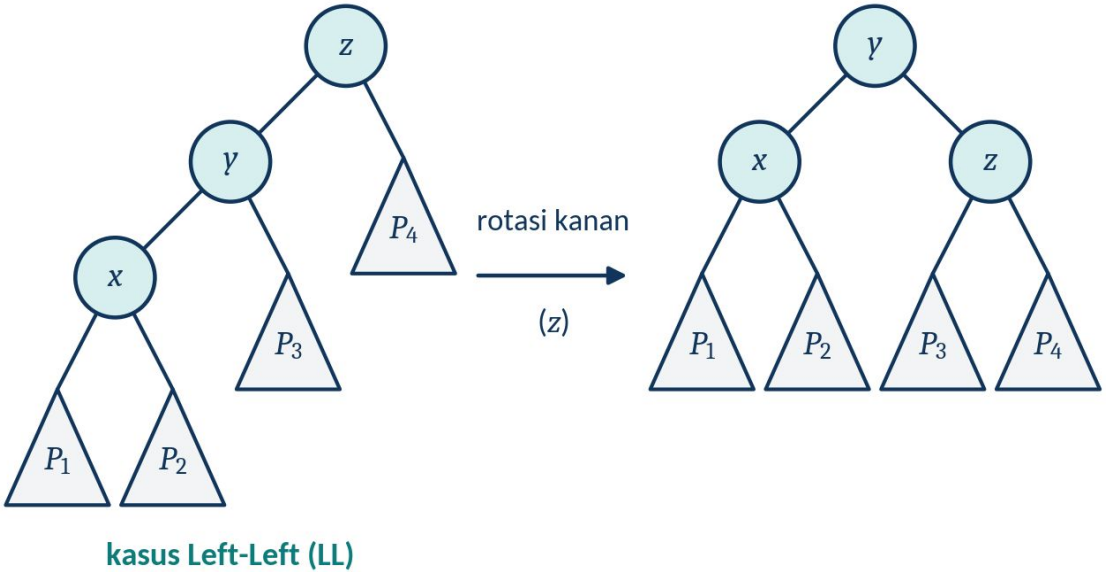
$$rotateRight((y,(x,P_1,P_2),P_3)) = (x,P_1,(y,P_2,P_3)), \tag{8.15}$$

Dengan $bf(t) = -2$: **LL** bila $bf(L) \le 0$ — satu rotasi kanan; **LR** bila $bf(L) > 0$ — rotasikan L ke kiri, lalu t ke kanan. Itulah Persamaan (8.16):

$$\text{rebalanceForLeft}(t) = \begin{cases} t & \text{bf}(t) \ge -1 \quad \text{(tak ada yang perlu dikerjakan)} \\ \text{rotateRight}(t) & \text{bf}(t) = -2 \text{ dan } \text{bf}(L) \le 0 \quad \text{(LL)} \\ \text{rotateRight}(v, \text{rotateLeft}(L), R) & \text{bf}(t) = -2 \text{ dan } \text{bf}(L) > 0 \quad \text{(LR)} \end{cases} \tag{8.16}$$

Kasus LR terkecil: 10*, kiri* 5*, anak kanan* 5 *yaitu* 7*. Rotasi kanan saja hanya memindahkan ketimpangan ke sisi lain; kiri-lalu-kanan memberi* $7\,[5,10]$*. Buku: Subbab 8.11.2–8.11.4.*

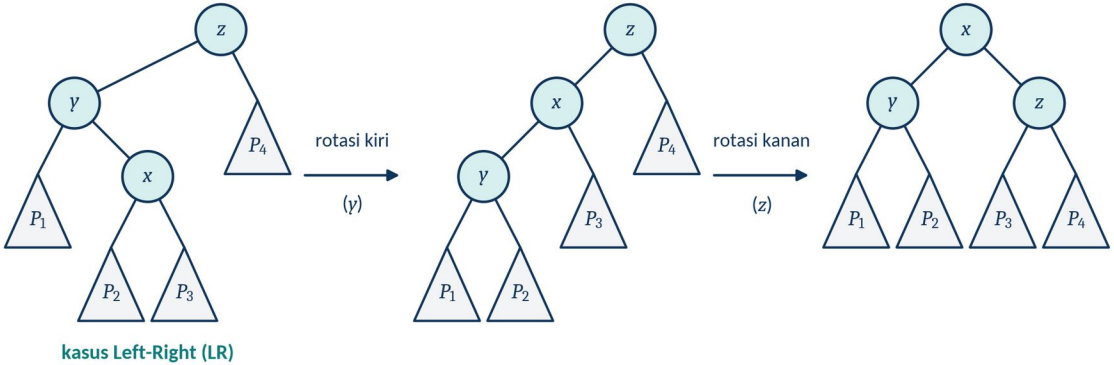
Kasus LL



Gambar 8.8 Kasus Left-Left: z adalah simpul yang tak seimbang, y anak kirinya, x cucu kirinya. Satu rotasi kanan terhadap z mengangkat y ke puncak. P_1 sampai P_4 adalah keempat upapohonnya.

Gambar 8.8: dengan $h(P_4)=k$ dan $h(y)=k+2$, satu rotasi kanan menjaga $bf(z)$ dan $bf(y)$ tetap dalam jangkauan, baik $h(P_3)$ bernilai k maupun $k+1$. Buku: Subbab 8.11.3.

Kasus LR



Gambar 8.9 Kasus Left-Right. Satu rotasi kanan tunggal terhadap z akan meninggalkan pohonnya sama tak seimbangnya ke arah sebaliknya. Pertama satu rotasi kiri terhadap y meluruskan zig-zagnya menjadi condongan kiri-kiri; lalu satu rotasi kanan terhadap z menuntaskannya.

Gambar 8.9: bagian yang tinggi adalah subpohon kanan y, yang tidak terangkat oleh rotasi kanan terhadap z; luruskan zig-zagnya lebih dulu. Buku: Subbab 8.11.4.

Butir Tersubtil: ≤, Bukan <

Sesudah penyisipan, $bf(L) = 0$ tidak mungkin terjadi. Sesudah **penghapusan** bisa: 10 dengan $L = 5\{3,7\}$ dan $R = 20$; hapus 20 maka $bf(10) = -2$ sedangkan $bf(5) = 0$.

Dengan `<= 0`, satu rotasi kanan memberi $5\{3, 10\{7, .\}\}$, seimbang. Dengan `< 0`, rotasi ganda melakukan kerja yang tidak perlu dan, secara umum, menyisakan ± 2 di akar yang baru.

> ≤ **ITU BUKAN HIASAN.** Pembantu yang sama melayani penghapusan, dan itulah yang memaksakan `<= 0`. Versi `< 0` lolos setiap uji penyisipan dan gagal pada penghapusan.

Bila hanya satu pertanyaan diajukan tentang `rebalanceForLeft`, pertanyaannya adalah yang ini. Buku: Subbab 8.11.5.

Soal 8.9 — `rebalanceForRight`, Cerminannya

Cerminkan Soal 8.8 simbol demi simbol: `>= -1` menjadi `<= 1`; `leftOf` menjadi `rightOf`; `bf(l) <= 0` menjadi `bf(r) >= 0`; `rotateRight` dan `rotateLeft` bertukar. Empat perubahan, bukan tiga.

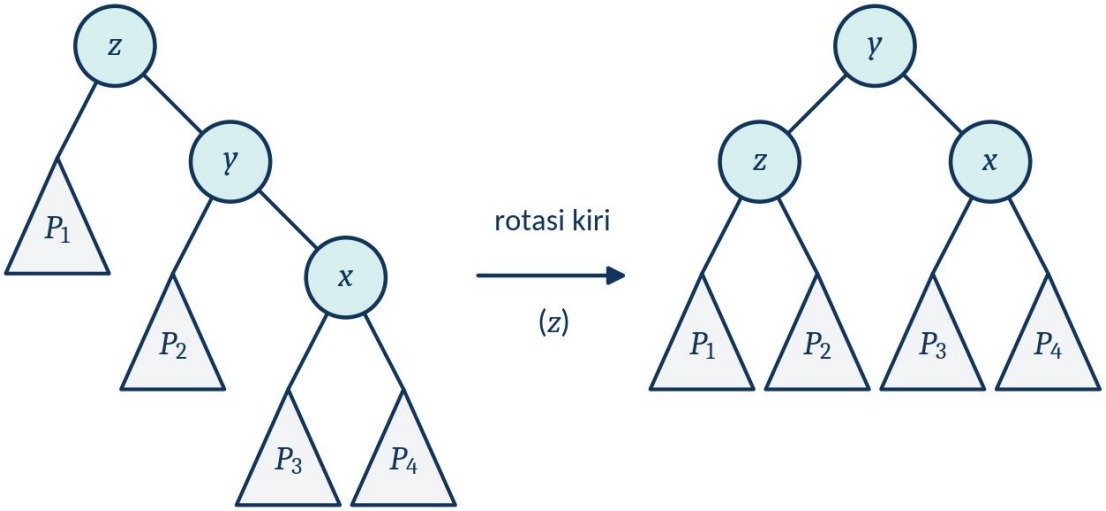
Tabel 8.6 Keempat kasus rotasi, sebagaimana diuji di dalam kode. Perhatikan bahwa kasus nolnya tergabung dengan rotasi *tunggal* di kedua sisinya — ≤ 0 di kiri, ≥ 0 di kanan — karena alasan yang diberikan §8.11.5.

bf(t)	bf Anak yang tinggi	Kasus	Perbaikan
-2	$bf(L) \leq 0$	**LL**	satu `rotateRight(t)`
-2	$bf(L) > 0$	**LR**	`rotateLeft(L)`, lalu `rotateRight(t)`
$+2$	$bf(R) \geq 0$	**RR**	satu `rotateLeft(t)`
$+2$	$bf(R) < 0$	**RL**	`rotateRight(R)`, lalu `rotateLeft(t)`

Tabel 8.6 adalah seluruh tabel keputusannya: kasus nol ikut rotasi *tunggal* pada kedua sisi.

Belokan keliru: penjaga luar terbalik; `<` untuk `<=`; merotasi terhadap simpul yang salah pada kasus ganda; menganggap penyeimbang selalu punya pekerjaan. Buku: Subbab 8.12.

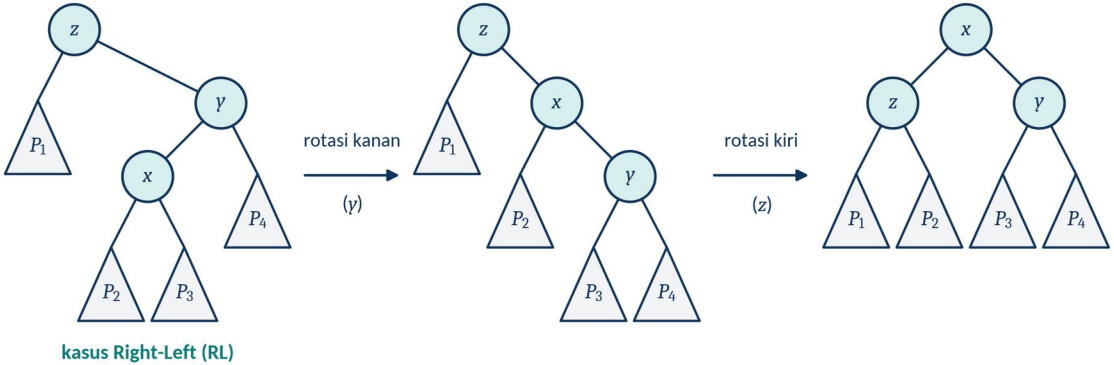
Kasus RR



Gambar 8.10 Kasus Right-Right: y adalah anak kanan z dan x adalah anak kanan y. Satu rotasi kiri terhadap z membetulkannya. Ini cermin persis dari Gambar 8.8.

Gambar 8.10: satu rotasi kiri terhadap z. Buku: Subbab 8.12.3.

Kasus RL



Gambar 8.11 Kasus Right-Left: zig-zagnya kanan lalu kiri. Satu rotasi kanan terhadap y meluruskannya, lalu satu rotasi kiri terhadap z menuntaskannya. Cermin persis dari Gambar 8.9.

Gambar 8.11: rotasi kanan terhadap y, lalu rotasi kiri terhadap z. Buku: Subbab 8.12.3.

Soal 8.10 — `deleteHB`

Pada simpul sasaran: subpohon kiri kosong mengembalikan R (kasus 1 dan 2); subpohon kanan kosong mengembalikan L; selain itu ganti v dengan $m = \max(L)$ — Soal 8.5 — lalu hapus m dari L.

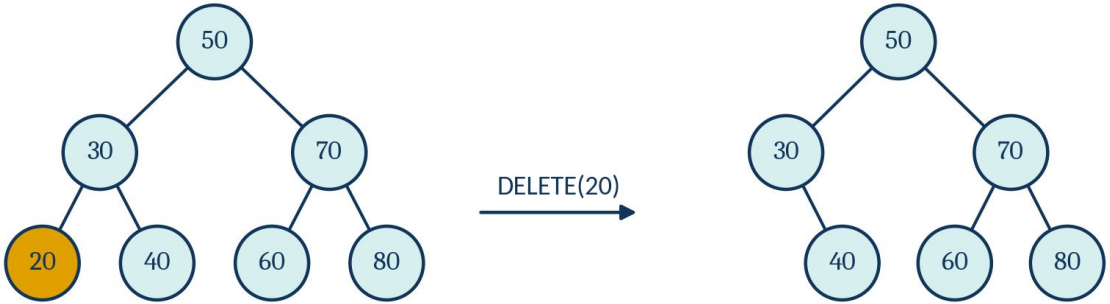
SISIP KIRI ⇒ `rebalanceForLeft`. HAPUS KIRI ⇒ `rebalanceForRight`. Menghapus dari satu sisi membuat sisi *yang lain* relatif lebih tinggi.

$$\text{deleteHB}((v, L, R), x) = \begin{cases} \text{rebalanceForRight}(v, \text{deleteHB}(L, x), R) & x < v \\ \text{rebalanceForLeft}(v, L, \text{deleteHB}(R, x)) & x > v \\ \text{ketiga kasus pada §8.13.2} & x = v \end{cases} \tag{8.17}$$

Dibandingkan Persamaan (8.14), Persamaan (8.17) berbentuk sama dengan penyeimbang yang dipertukarkan. Penyisipan menyeimbangkan ulang paling banyak sekali; penghapusan dapat merambat hingga $\Theta(\lg n)$ kali.

Seimbangkan ulang juga sesudah penggantian pada kasus dua anak, dan kembalikan pohon tanpa perubahan bila x tidak ada. Buku: Subbab 8.13.1–8.13.5.

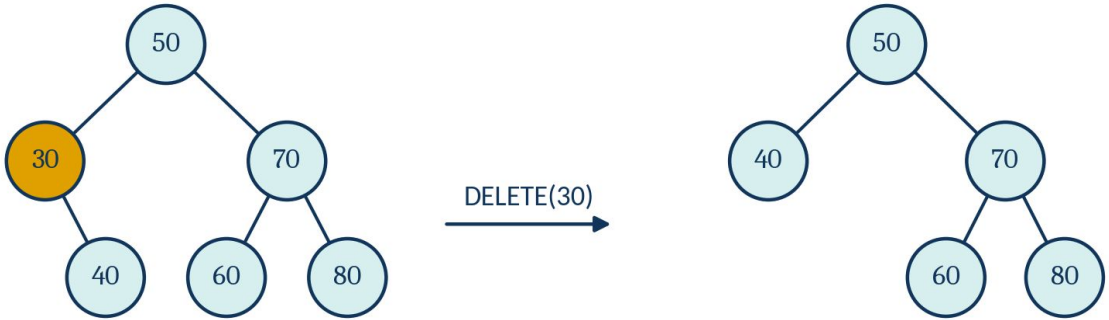
Penghapusan Kasus 1 — Sebuah Daun



Gambar 8.12 Kasus penghapusan pertama: simpul yang akan dihapus adalah daun. Cukup singkirkan ia.

Gambar 8.12: cukup buang simpulnya. Buku: Subbab 8.13.2.

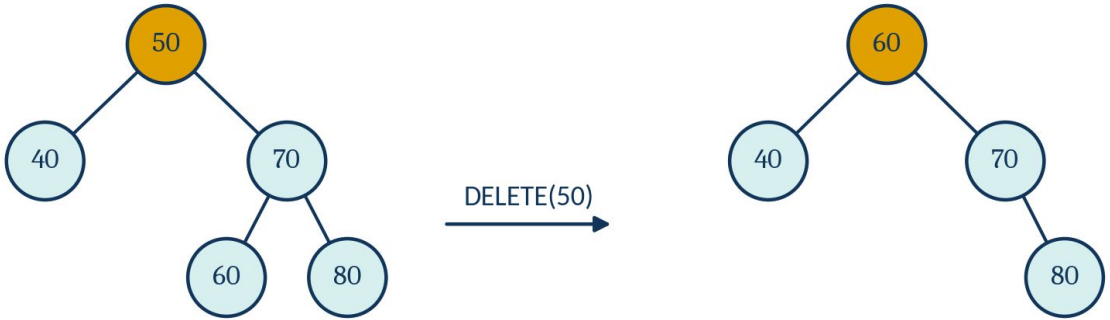
Penghapusan Kasus 2 — Satu Anak



Gambar 8.13 Kasus kedua: simpulnya punya tepat satu anak. “Copy the child to the node will be deleted, then delete the child” — setara dengan itu, dalam gaya fungsional, kembalikanlah anaknya sebagai pengganti simpulnya.

Gambar 8.13: dalam gaya fungsional, kembalikan anaknya sebagai pengganti simpul itu. Buku: Subbab 8.13.2.

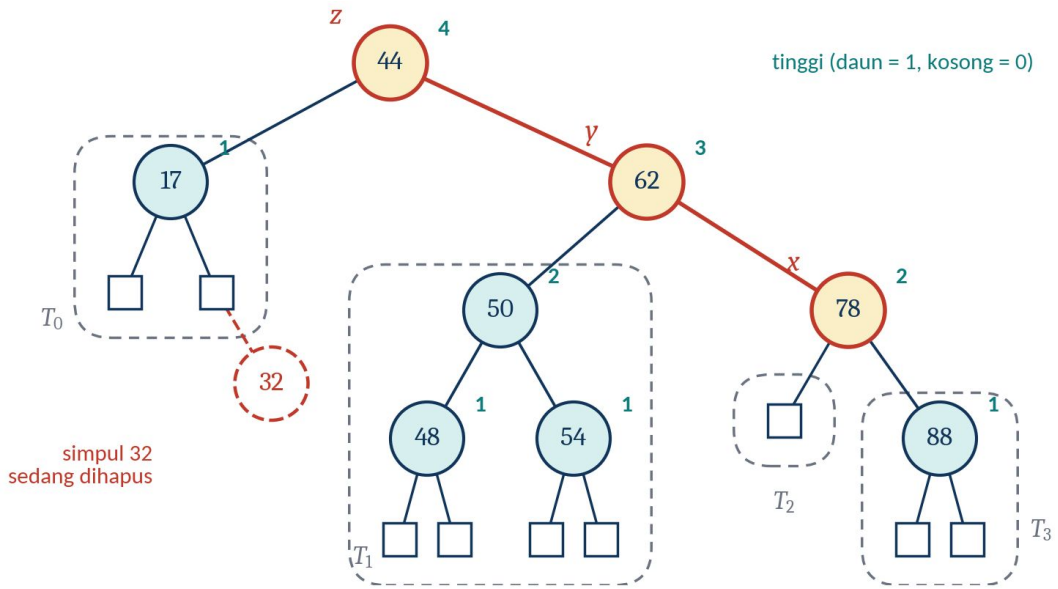
Penghapusan Kasus 3 — Dua Anak



Gambar 8.14 Kasus ketiga: simpulnya punya dua anak. Carilah pengganti inordernya (atau, secara simetris, pendahulunya), salinlah nilai itu ke dalam simpulnya, lalu hapuslah penggantinya dari subpohon kanannya — sebuah soal yang lebih kecil berupa kasus 1 atau kasus 2.

Gambar 8.14: salin penerus (atau pendahulunya) lalu hapus ia dari subpohonnya — soal kasus 1 atau kasus 2. Buku: Subbab 8.13.2.

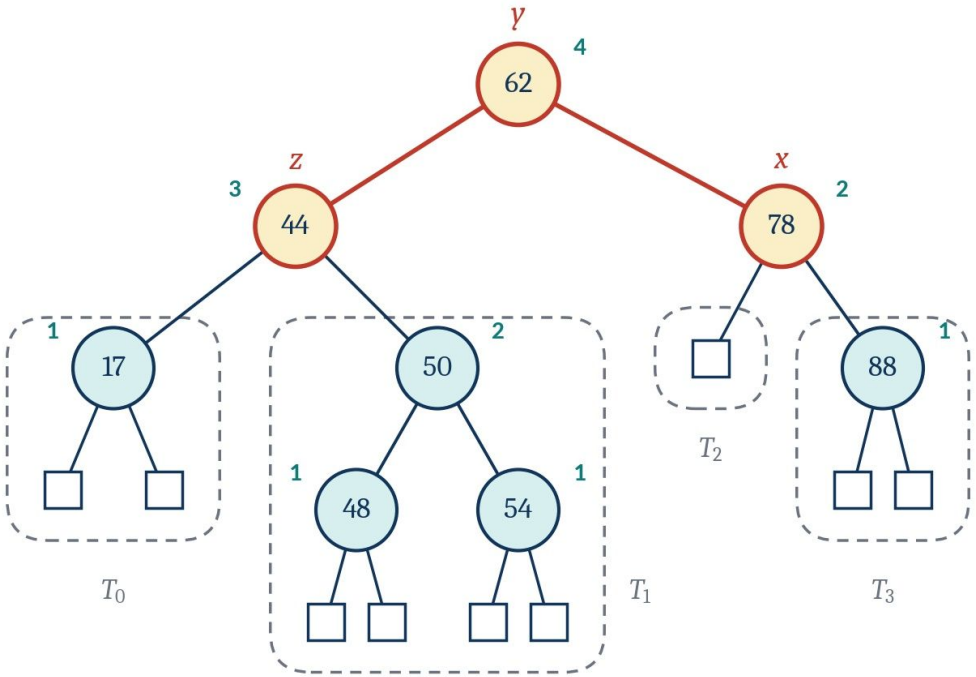
Contoh Penghapusan — Sebelum



Gambar 8.15 Sebelum penghapusan: simpul 32 hendak dihapus, dan leluhur tak seimbang yang pertama $z = 44$, anaknya yang lebih tinggi $y = 62$, serta anak y yang lebih tinggi $x = 78$ sudah ditandai. Tinggi pada gambarnya menghitung simpul, sehingga sebuah daun bernilai 1 — satu lebih besar daripada kesepakatan kita di mana-mana.

Gambar 8.15: menghapus 32 *memendekkan sisi kanan* 17 *(*`rebalanceForLeft` *tidak melakukan apa-apa), lalu sisi kiri* 44*, sehingga* `rebalanceForRight` *di* 44 *menemukan* $bf(44) = +2$ *dan* $bf(62) = 0 \geq 0$*: baris RR. Buku: Subbab 8.13.5.*

Contoh Penghapusan — Sesudah



Gambar 8.16 Setelah satu rotasi kiri terhadap 44. Simpul 62 menjadi akar yang baru, 44 turun ke kirinya sambil memegang subpohon berakar 50, dan 78 tak berubah di kanan.

Gambar 8.16: satu rotasi kiri terhadap 44*; programnya mencetak* `62(-1){44(1){17(0),50(0){48(0),54(0)}},78(1){.,88(0)}}`*. Buku: Subbab 8.13.5.*

Soal 8.7 sampai 8.10, Diperiksa sebagai Satu Program

Sisipkan 14, 3, 45, 7, 15 lalu hapus 32: jumlah rotasinya $0, 1, 1, 2, 2, 1$ — tidak ada, LL, RR, LR, RL, RR — dan setiap bentuknya cocok dengan pohon yang dikerjakan dengan tangan di Bab 6.

Hapus 62 (dua anak): pendahulunya, 54, menggantikannya, tanpa rotasi. Hapus 99 (tidak ada): pohonnya tidak berubah.

Uji beban: menyisipkan 1 sampai 1000 secara berurutan menghasilkan tinggi 9, di dalam pemeriksaan program $1.4405 \lg(n+2) - 0.3277 = 14.03$; menghapus keseribunya mengosongkan pohon dengan 491 rotasi.

Argumen kebenaran yang dapat diperiksa siapa pun dalam tiga puluh detik lebih berharga daripada prosa sebanyak apa pun. Buku: Subbab 8.13.6 dan 8.13.7.

Soal 8.11 sampai 8.15 — Pohon Merah-Hitam

Tabel 8.7 Kelima soal merah-hitam dalam subbab ini. Kolom terakhir mengatakan apakah sifatnya dapat diperiksa hanya dengan melihat sebuah simpul dan anak-anaknya — pertanyaan yang menentukan Soal 8.3.

Soal	Rutin	Yang harus dikerjakannya	Lokal atau global?
8.11	`countBlackNodes(t)`	Menghitung simpul hitam, tanpa menghitung `NIL`	cacahan global, tak mungkin dipangkas
8.12	`validateProperty4(t)`	Memeriksa bahwa tak ada simpul merah yang beranak merah	lokal: induk dan anak-anaknya
8.13	`validateBlackHeight(t)`	Memeriksa bahwa setiap lintasan menurun membawa simpul hitam yang sama banyak	global, diselesaikan dalam satu lintasan dengan penjaga
8.14	`isValidRBTree(t)`	Memeriksa kelima sifat merah-hitam dan sifat pohon pencarian	perakitan dari yang lain
8.15	`leftRotate(t, x)`, `rightRotate(t, y)`	Merotasi di tempat, sambil membetulkan semua penunjuk induk	bedah lokal atas enam penunjuk

Tabel 8.7: sifat 4 (tidak ada simpul merah beranak merah) adalah aturan induk–anak, sehingga pemeriksaan **lokal** kali ini benar; sifat 5 (tinggi-hitam yang seragam) memakai penjaga -1, karena tinggi-hitam selalu ≥ 0.

`isValidRBTree` tidak perlu menguji sifat 1 dan 3 — warna bertipe `bool` dan `NIL` hitam yang dipakai bersama sudah menjaminnya.

Pada pohon uji — akar hitam 26*; merah* 17 *di atas hitam* 14 *dan* 21*; hitam* 41 *di atas merah* 47 *— ada empat simpul hitam dan tinggi-hitamnya* 2*; kedua puluh pemeriksaan lolos. Buku: Subbab 8.14.1–8.14.6.*

Merah-Hitam Berdampingan dengan AVL

Tabel 8.8 Soal-soal AVL dan soal-soal merah-hitam berdampingan. Pemeriksaan pohon pencarian dan rotasinya pada hakikatnya sama; soal-soal di tengahnya berbeda sebab invarian keseimbangannya berbeda.

	Soal-soal AVL	Soal-soal merah-hitam
Pemeriksaan pohon pencarian	`isBST` (Soal 8.3)	`isBST` yang sama, logika identik
Pencacahan	`printDescending` (Soal 8.4)	`countBlackNodes` (Soal 8.11)
Sebuah pemeriksaan lokal	`max` (Soal 8.5)	`validateProperty4` (Soal 8.12)
Pemeriksaan satu lintasan dengan penjaga	`isHeightBalanced` (Soal 8.6)	`validateBlackHeight` (Soal 8.13)
Perakitan	`insertHB` (Soal 8.7)	`isValidRBTree` (Soal 8.14)
Rotasi	`rebalanceForLeft`, `rebalanceForRight` (8.8, 8.9)	`leftRotate`, `rightRotate` (8.15)
Penghapusan	`deleteHB` (Soal 8.10)	—
Invarian keseimbangan	$\lvert bf(x)\rvert \le 1$	warna, ditambah tinggi-hitam yang seragam
Gaya	fungsional: bangun ulang lalu kembalikan	imperatif: ubah di tempat, dengan penunjuk `parent`

Tabel 8.8: satu keluarga. Rotasi imperatif pada Soal 8.15 punya **enam** penunjuk yang harus dibetulkan — tautan ke bawah *dan* tautan induk; melupakan yang ke atas menghasilkan pohon yang tercetak benar tetapi berjalan naik ke tempat yang salah.

Rotasi fungsional pada Soal 8.8 sama sekali tidak memerlukan penunjuk induk, karena akar subpohon yang baru dikembalikan. Buku: Subbab 8.14.7 dan 8.14.8.

Soal 8.16 — Rantai Matriks dengan Tangan

A_1: 4×10, A_2: 10×3, A_3: 3×12, A_4: 12×20, A_5: 20×7, jadi $d = \langle 4, 10, 3, 12, 20, 7 \rangle$ dan rekurensinya Persamaan (8.18).

$$M[i,j] = \begin{cases} 0 & \text{jika } i = j, \\ \min\limits_{i \le k < j} \{M[i,k] + M[k+1,j] + d_{i-1}\, d_k\, d_j\} & \text{jika } i < j. \end{cases} \tag{8.18}$$

Pseudokode 8.5 `MATRIX-CHAIN-ORDER`: setiap subrantai diselesaikan sekali, yang terpendek lebih dahulu, dan pemisah pemenangnya dicatat.

```
MATRIX-CHAIN-ORDER(d, n)
1  for i = 1 to n
2      M[i, i] = 0
3  for l = 2 to n                          // l adalah panjang rantai
4      for i = 1 to n − l + 1
5          j = i + l − 1
6          M[i, j] = ∞
7          for k = i to j − 1
8              q = M[i, k] + M[k + 1, j] + d[i − 1] · d[k] · d[j]
9              if q < M[i, j]
10                 M[i, j] = q
11                 c[i, j] = k
12 return M and c
```

Pseudokode 8.5 mengisi tabel menurut panjang rantai yang menaik, sehingga setiap sel yang diperlukan sebuah sel sudah tersedia. Buku: Subbab 8.15.1–8.15.3.

Soal 8.16 — Mengisi Tabelnya

Panjang 2: $M[1,2]=120$, $M[2,3]=360$, $M[3,4]=720$, $M[4,5]=1680$.

Panjang 3: $M[1,3]=\min(840,264)=264$; $M[2,4]=\min(1320,2760)=1320$; $M[3,5]=\min(1932,1140)=1140$.

Panjang 4: $M[1,4]=\min(2120,1080,1224)=1080$; $M[2,5]=\min(1350,2880,2720)=1350$.

Panjang 5: $M[1,5]=\min(1630,1344,2280,1640)=1344$ pada $k=2$.

Pada $M[1,3]$, mengalikan A_1A_2 lebih dulu menyusutkan 10 menjadi 3, sehingga setiap perkalian sesudahnya lebih murah. Buku: Subbab 8.15.4–8.15.7.

Soal 8.16 — Tabel dan Pengurungannya

Tabel 8.9 Tabel biaya M untuk $d = \langle 4, 10, 3, 12, 20, 7 \rangle$. Setiap diagonal diisi sebelum diagonal berikutnya; jawabannya ada di pojok kanan atas, $M[1,5] = 1\,344$.

$M[i,j]$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0	120	264	1 080	**1 344**
$i=2$		0	360	1 320	1 350
$i=3$			0	720	1 140
$i=4$				0	1 680
$i=5$					0

Tabel 8.10 Tabel pemisah c untuk rantai yang sama. Kedua entri tebal adalah entri yang dibaca oleh rekonstruksi di §8.15.8.

$c[i,j]$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	1	2	2	**2**
$i=2$		2	2	2
$i=3$			3	**4**
$i=4$				4

Dari Tabel 8.9 dan Tabel 8.10: $c[1,5]=2$, $c[3,5]=4$, jadi $((A_1A_2)((A_3A_4)A_5))$ dengan $120+720+420+84=1344$ ✓, dibandingkan 1784 dari kiri ke kanan.

Rekursi polos membuat $R(n)=3^{n-1}$ panggilan (81 untuk $n=5$); *memoization* membuat Persamaan (8.19) (41 untuk $n=5$).

$$1+\sum_{\delta=1}^{n-1} 2\delta(n-\delta)=1+\frac{n^3-n}{3}. \tag{8.19}$$

Periksa diri: $d=\langle 2,3,6,4,5\rangle$ *berongkos* 124 *dengan* $(((A_1A_2)A_3)A_4)$. *Buku: Subbab 8.15.8–8.15.12.*

Soal 8.17 — SORTED Lawan STORE

Isi c dan b dengan Persamaan (8.20), keadaan seri ke atas; jalan mundur; putuskan apakah jawabannya tunggal; nyatakan ongkosnya.

$$c[i,j] = \begin{cases} 0 & \text{jika } i = 0 \text{ atau } j = 0, \\ c[i-1, j-1] + 1 & \text{jika } i, j > 0 \text{ dan } x_i = y_j, \\ \max\{c[i-1, j],\ c[i, j-1]\} & \text{jika } i, j > 0 \text{ dan } x_i \neq y_j. \end{cases} \tag{8.20}$$

Pseudokode 8.6 `LCS-LENGTH`: tabel panjang dan tabel panah, diisi baris demi baris, dengan seri diputuskan ke atas.

```
LCS-LENGTH(X, Y, m, n)
1  for i = 0 to m
2      c[i, 0] = 0
3  for j = 0 to n
4      c[0, j] = 0
5  for i = 1 to m
6      for j = 1 to n
7          if x_i == y_j
8              c[i, j] = c[i − 1, j − 1] + 1;  b[i, j] = "↖"
9          elseif c[i − 1, j] ≥ c[i, j − 1]
10             c[i, j] = c[i − 1, j];          b[i, j] = "↑"
11         else
12             c[i, j] = c[i, j − 1];          b[i, j] = "←"
13 return c and b
```

Pseudokode 8.7 `PRINT-LCS`: ikuti panah-panahnya mundur; setiap panah diagonal adalah satu karakter jawaban.

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2      return
3  if b[i, j] == "↖"
4      PRINT-LCS(b, X, i − 1, j − 1)
5      print x_i
6  elseif b[i, j] == "↑"
7      PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```

Pseudokode 8.6 mencatat ↖, ↑, *atau* ←; *Pseudokode 8.7 mencetak sesudah panggilan rekursifnya, sehingga karakternya keluar menurut urutan. Buku: Subbab 8.16.1 dan 8.16.2.*

Soal 8.17 — Tabelnya, dan Jalan Mundurnya

Tabel 8.11 Tabel c dan b untuk \$X = \$ SORTED dan \$Y = \$ STORE. Sel-sel tebal adalah keempat panah diagonal yang dilewati jalan mundur di §8.16.4.

	j=0	j=1 (S)	j=2 (T)	j=3 (O)	j=4 (R)	j=5 (E)
$i=0$	0	0	0	0	0	0
$i=1$ (S)	0	**1** ↖	1 ←	1 ←	1 ←	1 ←
$i=2$ (O)	0	1 ↑	1 ↑	**2** ↖	2 ←	2 ←
$i=3$ (R)	0	1 ↑	1 ↑	2 ↑	**3** ↖	3 ←
$i=4$ (T)	0	1 ↑	2 ↖	2 ↑	3 ↑	3 ↑
$i=5$ (E)	0	1 ↑	2 ↑	2 ↑	3 ↑	**4** ↖
$i=6$ (D)	0	1 ↑	2 ↑	2 ↑	3 ↑	4 ↑

Tabel 8.11: $c[6,5]=4$. Dari $(6,5)$: ↑; ↖ mengambil E; ↑; ↖ mengambil R; ↖ mengambil O; ←; ↖ mengambil S. Dibalik: **SORE**.

Tunggal? T muncul sebelum O dan R pada STORE tetapi sesudahnya pada SORTED, jadi *subsequence* bersama yang memuat T paling panjang 3 (STE); D tidak ada pada STORE. Jadi SORE satu-satunya.

$\Theta(mn)$ *untuk tabelnya (*30 *sel), paling banyak* $m+n$ *langkah untuk jalan mundurnya (*7 *di sini). Periksa diri: HEAPS dan SHAPE memberi HAP. Buku: Subbab 8.16.3–8.16.9.*

Pemeriksaan Cepat

Mengapa `BUILD-MAX-HEAP` $\Theta(n)$? Sebagian besar simpul berada di bawah; $\sum_h \lceil n/2^{h+1} \rceil O(h) = O(n)$.

Kasus terburuk Quicksort dengan pivot elemen terakhir? Masukan terurut: tepat $\frac{n(n-1)}{2}$ pembandingan.

Batas bawah pengurutan berbasis pembandingan? $\Omega(n \lg n)$: $n!$ daun memaksa tinggi $\lg(n!)$.

Rotasi per penyisipan AVL? Per penghapusan? Paling banyak dua; hingga $\Theta(\lg n)$.

Kapan pemrograman dinamis alih-alih *divide and conquer*? Submasalah yang tumpang tindih ditambah *optimal substructure*.

Dua puluh lima pemeriksaan cepat mencakup Bab 1 sampai 7; yang terlewat menunjukkan subbab yang perlu dibaca ulang. Buku: Subbab 8.17.

Soal 8.18 sampai 8.25 — Pohon Statistik-Urutan

Setiap simpul juga menyimpan `h` dan `size`, keduanya dipelihara oleh konstruktor tunggal `node(v, l, r)`. Tabel 8.12 memasangkan setiap soal dengan soal yang diperluasnya.

Tabel 8.12 Soal 8.18 sampai 8.25 dibandingkan dengan soal-soal yang diperluasnya. Bentuknya sama; hanya invariannya yang berubah.

Soal	Memperluas	Yang baru
8.18	8.3 `isBST`	dua medan turunan juga harus diperiksa
8.19	8.4 `printDescending`	pemangkasan, bukan sekadar urutan penelusuran
8.20	8.5 `max`	ukuran subpohon mengemudikan penurunannya
8.21	8.6 `isHeightBalanced`	gagasan keseimbangan yang *berbeda*
8.22	8.7 `insertHB`	medan `size` harus bertahan melewati rotasi
8.23	8.8 `rebalanceForLeft`	strukturnya tak berubah
8.24	8.9 `rebalanceForRight`	strukturnya tak berubah
8.25	8.10 `deleteHB`	strukturnya tak berubah

Bentuknya sama dengan Soal 8.3 sampai 8.10, invariannya berbeda: mengingat saja tidak cukup, memahami yang cukup. Buku: Subbab 8.18.1.

Pohon Statistik-Urutan, Diselesaikan

8.18: kembalikan tinggi dan ukuran yang sebenarnya lalu bandingkan dengan medan tersimpan; pohon yang dirusak tertangkap sebagai “node 50 stores size 99 but the truth is 11”.

8.19: inorder dengan dua uji pemangkasan, $O(h+k)$. **8.20:** dengan $\ell = size(\text{kiri})$, ke kiri bila $k \le \ell$, berhenti bila $k = \ell + 1$, selain itu cari $k - \ell - 1$ di kanan; `select(6) = 45` sesudah 4 penjajakan.

8.21: pohon AVL **tidak** harus seimbang-bobot: subpohon kiri sempurna berisi 7 simpul lawan subpohon kanan minimal berisi 2 memberi $7 > 2 \cdot 2 + 1$.

8.22–8.25: sama persis dengan 8.7–8.10, karena `node` juga membereskan `size`. Uji beban: 2000 penyisipan menaik menghasilkan tinggi 10.

Letakkan perbaikan invarian di dalam konstruktor, maka algoritmanya tidak lagi peduli berapa banyak invariannya. Buku: Subbab 8.18.2–8.18.4.

Profitina dalam Praktik: Indeks yang Mengaudit Dirinya

Di Profitina, antrean pemindaian adalah antrean prioritas (Soal 8.1 dalam skala besar), membandingkan versi halaman adalah Soal 8.17, dan indeks waktu-jatuh-tempo harus mengaudit dirinya dengan `isBST` dan `isHeightBalanced`.

Profitina Laundry menyimpan jadwal pengambilan dalam pohon AVL. Sesudah suntingan langsung yang keliru, auditnya berbunyi `search-tree=BROKEN, height-balanced=OK` dan menyebut catatannya: “order PL-0431 (minute 500) is filed where only minutes in (-1,20) belong”.

Pemulihan tidak boleh bergantung pada invarian yang rusak: penelusuran inorder tetap mencapai setiap simpul, jadi baca keluar, urutkan menurut kunci yang benar, bangun ulang dengan `insertHB`. Tagihan: Rp 7.000 per kg, minimum dua kilogram, Rp 350.000 untuk hari itu.

Jangan percaya medan yang tersimpan; hitung ulang. Platform memberi saran; pelanggan yang memutuskan; penghapusan tuntas menurut UU PDP 27/2022 memerlukan indeks yang dapat menghapus. Buku: Subbab 8.19 dan 8.19.1.

Kesalahan yang Sering Terjadi

Membaca soal hanya sekali — *rekursif*, *tanpa daftar terpisah*, *satu lintasan*, *satu kali jalan*, *pakai kedua pembantu* adalah isi yang sesungguhnya.

Menguji jalur mulus saja — pohon kosong, satu simpul, dua simpul, dan kunci yang tidak ada menemukan sebagian besar galat.

Memercayai medan tersimpan; **memeriksa secara lokal apa yang berlaku global** (sifat pohon pencarian biner) atau secara global apa yang berlaku lokal.

Memanggil penyeimbang yang dinamai menurut sisi tempat Anda berekursi saat menghapus; mengatakan $O(\lg n)$ padahal maksudnya $O(h)$.

Batas gelung atau urutan pengisian yang salah; **menyimpan nilai tetapi membuang pilihannya** (M tanpa c, c tanpa b).

Pencacah hanya berongkos tiga baris dan mengubah pendapat menjadi bukti. Buku: Subbab 8.20.

Ringkasan: Lima Hal untuk Diingat

1. Bawa informasi turun atau kembalikan ke atas, sehingga tidak ada kerja yang diulang: batas untuk `isBST`, tinggi untuk `isHeightBalanced`, subpohon yang dibangun ulang untuk `insertHB`, tabel untuk pemrograman dinamis.

2. Soal 8.1: 15 pembandingan, 6 penukaran, $O(n)$. Soal 8.2: $\frac{n(n-1)}{2}$ terburuk, $(n+1)\lg(n+1)-2n$ terbaik.

3. Pemeriksaan pohon pencarian biner memerlukan selang; pemeriksaan *heap* tidak. `max` menelusuri tulang punggung kanan dalam $O(h)$.

4. AVL: satu konstruktor, rotasi sebagai ekspresi, `<= 0` untuk rotasi tunggal, dan penyeimbang **yang berlawanan** saat menghapus.

5. Rantai matriks: 1344 dengan $((A_1 A_2)((A_3 A_4) A_5))$. LCS dari SORTED dan STORE: SORE.

Setiap penelusuran dengan tangan di sini direproduksi oleh program, dan setiap hitungan bentuk tertutup dibandingkan dengan hasil ukur. Buku: Subbab 8.21.

Pertemuan Berikutnya

Bab 9 — algoritma *greedy*: pemilihan kegiatan, pengodean Huffman, dan teori matroid yang menyatakan kapan sifat rakus itu aman.

Sebelum itu: Soal Latihan 1, 6, 11, 13, dan 14 dari Bab 8, dengan tangan.

Terus bertanya: *invarian apa yang dijaga kode saya, dan bagaimana saya memeriksanya?*

Algoritma greedy benar tepat ketika suatu invarian pertukaran bertahan pada setiap langkah. Buku: Subbab 8.21.1 dan 8.22.

BAB 1

Kuliah 9 — Algoritma Rakus

Perancangan dan Analisis Algoritma (EF234405)

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*

Dosen: Irfan Subakti — yifana@gmail.com

Buku: Bab 9, Algoritma Rakus. Setiap halaman handout ini diakhiri dengan penunjuk ke buku, yang menjadi rujukan utama; handout ini adalah bab tersebut dalam bentuk ringkas.

Pertemuan Hari Ini

1. Apa itu algoritma rakus — dan mengapa aturan rakus biasanya *keliru* sampai ia dibuktikan (Subbab 9.2–9.3).

2. Pemilihan aktivitas secara tuntas: empat aturan, substruktur optimal, **argumen penukaran**, penelusuran, dan waktu jalannya (Subbab 9.4).

3. Unsur-unsur strategi rakus, dan dua masalah ransel yang memisahkan kerakusan dari pemrograman dinamis (Subbab 9.5–9.6).

4. Kode Huffman: pohonnya, biaya $B(T)$, penelusuran per penggabungan, dan buktinya (Subbab 9.7).

5. Matroid — jawaban terdalam atas pertanyaan *kapan kerakusan berhasil?* — dan penjadwalan dengan tenggat (Subbab 9.8); lalu katalognya, Profitina Laundry, dan kesalahan yang sering terjadi (Subbab 9.9–9.11).

Buku: Bab 9, Subbab 9.1 sampai 9.12.

Dari Mana Kita Melanjutkan

Bab 7 memberi kita **pemrograman dinamis**: cirikan solusi optimalnya, definisikan nilainya secara rekursif, hitung menurut urutan ketergantungan, lalu rekonstruksi dari pilihan yang dicatat.

Bab itu memberi kita **substruktur optimal** beserta **argumen potong-dan-tempel** yang membuktikannya (Subbab 7.5).

Bab itu juga memberi tagihannya: rantai matriks dengan waktu $\Theta(n^3)$ dan ruang $\Theta(n^2)$; subbarisan persekutuan terpanjang dengan $\Theta(mn)$ — tabel sebesar 488 MiB pada $m=n=8000$.

Mengapa begitu mahal? Di setiap sel, program dinamis *tidak tahu pilihan mana yang benar*, sehingga ia mencoba semuanya.

> **PERTANYAAN HARI INI.** *Bagaimana jika pilihannya sudah jelas?* Jika satu kandidat tampak benar begitu saja, tak perlu tabel dan tak perlu minimisasi — cukup satu pengurutan dan satu sapuan, biasanya $\Theta(n \lg n)$.

Buku: Subbab 9.1 dan Subbab 9.1.1.

Apa Itu Algoritma Rakus

> **DEFINISI.** Sebuah **algoritma rakus** menyelesaikan persoalan optimasi dengan serangkaian pilihan. Pada tiap titik keputusan ia menjatuhkan pilihan **yang tampak terbaik pada saat itu**, dinilai dengan aturan tetap yang hanya memeriksa keadaan saat ini — tak pernah memeriksa solusi submasalah yang dilahirkan pilihan itu, tak pernah memeriksa pilihan di masa depan. Ia tak pernah meninjau ulang pilihannya.

“Terbaik pada saat itu.” Aturannya *lokal*: koin terbesar yang muat; aktivitas yang paling awal selesai; gabungkan dua lambang yang paling jarang.

“Tak pernah memeriksa submasalah.” Pemrograman dinamis *menyelesaikan submasalah sebelum menjatuhkan pilihan pertama*; algoritma rakus *menjatuhkan pilihan pertama sebelum menyelesaikan submasalah apa pun*.

“Tak pernah meninjau ulang.” Tanpa runut balik: dari situlah kecepatannya, dan karena itu pula kekeliruan di langkah 3 tak pernah dapat dibatalkan di langkah 40.

Buku: Subbab 9.2.1.

Kerakusan Adalah Harapan, Bukan Metode

ALGORITMA RAKUS BIASANYA KELIRU. Ambillah sembarang persoalan optimasi dan sebuah aturan lokal yang masuk akal, maka hasilnya, secara umum, tidak optimal. Tak ada apa pun dalam *bentuk* algoritma rakus yang membuatnya benar.

BUKTINYA ADALAH SELURUH PEKERJAANNYA. Menulis algoritmanya itu sepele — urutkan, lalu ambil berurutan. Memutuskan urutan *yang mana*, dan *membuktikan* bahwa urutan itu aman, itulah seluruh pekerjaannya. Algoritma rakus *dengan* buktinya biasanya adalah algoritma benar yang tercepat dan tersederhana bagi persoalan itu.

Program dinamis yang mencoba setiap titik belah tak pernah berjudi; algoritma rakus berjudi **di setiap langkah**, dan buktilah yang mengubah perjudian itu menjadi kepastian.

Buku: Subbab 9.2.2; rencana babnya ada di Subbab 9.2.3.

Uang Kembalian: Aturan Rakusnya

PERSOALAN UANG KEMBALIAN. Pecahan koin $d_1, d_2, \ldots, d_k$ (bilangan bulat positif, persediaan tak terbatas) dan sebuah nilai A: bayarlah tepat A dengan **koin sesedikit mungkin**.

ATURAN RAKUS UNTUK UANG KEMBALIAN. Selama masih ada utang, serahkan **koin terbesar yang tidak melebihi sisa yang terutang**.

Rupiah, terutang 78 000: ambil 50 000 (sisa 28 000), 20 000 (sisa 8 000), 5 000 (sisa 3 000), 2 000 (sisa 1 000), 1 000 (sisa 0).

Lima koin, $50\,000+20\,000+5\,000+2\,000+1\,000=78\,000$, dan tak ada empat koin yang sanggup melakukannya.

Buku: Subbab 9.3.1 dan 9.3.2.

Aturan yang Sama, Patah

Koin $\{1, 3, 4\}$, terutang $A = 6$. Sisa 6: ambil 4, sisa 2. Sisa 2: 4 dan 3 terlalu besar, ambil 1; lalu 1 lagi. Kerakusan membayar $4+1+1$ — **tiga koin**.

Kasir yang cermat membayar $3+3$ — **dua koin**. Lima puluh persen lebih banyak koin daripada yang perlu, untuk nilai enam.

Koin $\{1, 10, 25\}$, $A = 30$: kerakusan membayar $25+1+1+1+1+1$ (enam koin); optimumnya $10+10+10$ (tiga).

> **BENTUK SETIAP KEGAGALAN RAKUS.** Pilihan yang menarik secara lokal merusak submasalah yang ditinggalkannya lebih besar daripada keuntungan lokalnya. Mengambil 4 menyisakan 2, yang memerlukan dua koin; mengambil 3 akan menyisakan 3, yang cukup satu koin.

Buku: Subbab 9.3.3.

Sistem Koin Kanonis, Diaudit

Sebuah sistem koin disebut **kanonis** jika aturan rakus optimal bagi setiap nilai: $\{1,5,10,25\}$ kanonis, $\{1,3,4\}$ tidak. Tabel 9.1 mengaudit enam sistem untuk setiap nilai dari 1 sampai 200.

Tabel 9.1 Aturan rakus yang sama pada enam sistem koin, diaudit oleh Kode 9.1 terhadap optimum pemrograman dinamis untuk setiap nilai dari 1 sampai 200. Aturannya tak berubah; hanya struktur masukannya yang berubah.

Sistem koin	Kanonis ?	Contoh sanggahan terkecil	Rakus / optimal di sana	Keliru pada
$\{1,3,4\}$	tidak	$A=6$	3 keping lawan 2	49 dari 200
$\{1,10,25\}$	tidak	$A=30$	6 keping lawan 3	70 dari 200
$\{1,5,10,25\}$	ya	—	—	0 dari 200
$\{1,2,5,10,20,50,10$	ya	—	—	0 dari 200
$\{1,2,4,8,16,32,64\}$	ya	—	—	0 dari 200
$\{1,5,10,20,50,100\}$	ya	—	—	0 dari 200

PELAJARANNYA DALAM SATU KALIMAT. Strateginya bukan intinya; buktinyalah intinya. Pada $\{1,10,25\}$ kerakusan keliru pada **70 dari 200 nilai pertama** — aturan yang gagal gagal secara sistematis, bukan sesekali.

Buku: Subbab 9.3.4 dan 9.3.5.

Program Auditnya

Kode 9.1 menghitung jawaban rakus dan optimum sejatinya — dengan tabel satu dimensi `best[a] = 1 + min over coins c of best[a-c]` dari Bab 7 — untuk enam sistem dan setiap nilai sampai 200.

Kode 9.1 Mengaudit aturan rakus uang kembalian terhadap optimum sejatinya, untuk enam sistem koin dan setiap nilai sampai 200. Kompilasi dengan `g++ -std=c++17 -O2 -o coins coins.cpp` lalu jalankan `./coins`.

```cpp
// coins.cpp — Benarkah aturan rakus "koin terbesar yang muat" memberi koin paling sedikit?
// Untuk tiap sistem koin kita hitung, bagi setiap nilai uang, cacah koin versi
// rakus dan optimum sejatinya (dengan pemrograman dinamis), lalu melaporkan nilai
// terkecil tempat keduanya berselisih.
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include <algorithm>
#include <climits>
using namespace std;

// Rakus: berulang kali ambil koin terbesar yang tak melebihi sisa yang terutang.
// Mengembalikan koin yang dipakai, atau vektor kosong bila rakus macet.
vector<int> greedyChange(vector<int> coin, int amount) {
    sort(coin.rbegin(), coin.rend());           // terbesar dahulu
    vector<int> used;
    for (int c : coin)
        while (amount >= c) { amount -= c; used.push_back(c); }
    if (amount != 0) return {};                 // rakus macet: tak ada kembalian pas
    return used;
}

// Optimum dengan pemrograman dinamis: best[a] = koin tersedikit yang jumlahnya tepat a.
// choice[a] mengingat satu koin pada penyelesaian optimal agar dapat kita cetak.
pair<int, vector<int>> optimalChange(const vector<int>& coin, int amount) {
    const int INF = INT_MAX / 2;
    vector<int> best(amount + 1, INF), choice(amount + 1, -1);
```

```
    best[0] = 0;
    for (int a = 1; a <= amount; ++a)
        for (int c : coin)
            if (c <= a && best[a - c] + 1 < best[a]) {
                best[a] = best[a - c] + 1;
                choice[a] = c;
            }
    if (best[amount] >= INF) return {-1, {}};
    vector<int> used;
    for (int a = amount; a > 0; a -= choice[a]) used.push_back(choice[a]);
    return {best[amount], used};
}
```

Kode 9.1 (Lanjutan) *Mengaudit aturan rakus uang kembalian terhadap optimum sejatinya, untuk enam sistem koin dan setiap nilai sampai 200. Kompilasi dengan g++ -std=c++17 -O2 -o coins coins.cpp lalu jalankan ./coins.*

```
string show(const vector<int>& v) {
    string s;
    for (size_t i = 0; i < v.size(); ++i) { if (i) s += " + "; s += to_string(v[i]); }
    return s.empty() ? "(none)" : s;
}

void audit(const string& name, vector<int> coin, int limit) {
    cout << "System " << name << " = {";
    for (size_t i = 0; i < coin.size(); ++i) { if (i) cout << ", "; cout << coin[i]; }
    cout << "}\n";
    int firstBad = -1;
    int badCount = 0;
    for (int a = 1; a <= limit; ++a) {
        auto g = greedyChange(coin, a);
        auto o = optimalChange(coin, a);
        if (o.first < 0) continue;              // nilai ini sama sekali tak terbayarkan
        int gc = g.empty() && a > 0 ? -1 : (int)g.size();
        if (gc != o.first) { ++badCount; if (firstBad < 0) firstBad = a; }
    }
    if (firstBad < 0) {
        cout << "  greedy is optimal for every amount from 1 to " << limit << "\n\n";
        return;
    }
    auto g = greedyChange(coin, firstBad);
    auto o = optimalChange(coin, firstBad);
    cout << "  greedy FAILS.  smallest counterexample: amount = " << firstBad << "\n";
    cout << "    greedy : " << show(g) << "   = " << g.size() << " coins\n";
    cout << "    optimal: " << show(o.second) << "   = " << o.first << " coins\n";
```

```
    cout << "  greedy is wrong for " << badCount << " of the " << limit
         << " amounts tested\n\n";
}

int main() {
    audit("A (textbook trap)",  {1, 3, 4},                 200);
    audit("B (no nickel)",      {1, 10, 25},               200);
    audit("C (US coins)",       {1, 5, 10, 25},            200);
    audit("D (rupiah, x1000)",  {1, 2, 5, 10, 20, 50, 100}, 200);
    audit("E (powers of 2)",    {1, 2, 4, 8, 16, 32, 64},  200);
    audit("F (rupiah minus 2)", {1, 5, 10, 20, 50, 100},   200);
    return 0;
}
```

Keluarannya, yang menjadi sumber angka-angka Tabel 9.1:

```
System A (textbook trap) = {1, 3, 4}
  greedy FAILS.  smallest counterexample: amount = 6
    greedy : 4 + 1 + 1   = 3 coins
    optimal: 3 + 3   = 2 coins
  greedy is wrong for 49 of the 200 amounts tested

System B (no nickel) = {1, 10, 25}
  greedy FAILS.  smallest counterexample: amount = 30
    greedy : 25 + 1 + 1 + 1 + 1 + 1   = 6 coins
    optimal: 10 + 10 + 10   = 3 coins
  greedy is wrong for 70 of the 200 amounts tested

System C (US coins) = {1, 5, 10, 25}
  greedy is optimal for every amount from 1 to 200

System D (rupiah, x1000) = {1, 2, 5, 10, 20, 50, 100}
  greedy is optimal for every amount from 1 to 200

System E (powers of 2) = {1, 2, 4, 8, 16, 32, 64}
  greedy is optimal for every amount from 1 to 200

System F (rupiah minus 2) = {1, 5, 10, 20, 50, 100}
  greedy is optimal for every amount from 1 to 200
```

Buku: Subbab 9.3.5. Soal Latihan 7.18 adalah gejala yang sama dengan kantong laundry 3, 5, dan 8 kg.

Pemilihan Aktivitas: Persoalannya

DEFINISI. $S = \{a_1, \ldots, a_n\}$ adalah aktivitas yang menghendaki satu sumber daya. Aktivitas a_i menempati selang **setengah terbuka** $[s_i, f_i)$, $0 \le s_i < f_i < \infty$. Aktivitas a_i dan a_j **selaras** jika $s_i \ge f_j$ atau $s_j \ge f_i$. Pilihlah himpunan berukuran **maksimum** berisi aktivitas yang saling selaras.

Setengah terbuka membuat aktivitas yang berakhir pukul 10.00 selaras dengan aktivitas yang mulai pukul 10.00 — sumber dayanya berpindah tangan, tak ada yang dipesan ganda.

Yang dimaksimumkan adalah **cacah** aktivitas — bukan lama pemakaian, bukan nilai. Bila nilai disertakan, aturan rakus hari ini menjadi keliru (Soal Latihan 9.15).

Asumsi tetap Persamaan (9.1): aktivitas datang terurut menurut waktu selesai (bila belum, urutkan dengan biaya $O(n \lg n)$).

$$f_1 \le f_2 \le f_3 \le \cdots \le f_{n-1} \le f_n. \tag{9.1}$$

Buku: Subbab 9.4.1.

Contoh Berjalannya

Gambar 9.1 adalah contoh kasus bagi seluruh Subbab 9.4: $a_1 = [1, 4)$, $a_2 = [3, 5)$, $a_3 = [0, 6)$, ..., $a_{11} = [12, 16)$.

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Gambar 9.1 Sebelas aktivitas contoh berjalan, dengan waktu mulai s_i dan waktu selesai f_i, sudah terurut sehingga $f_1 \leq f_2 \leq \cdots \leq f_{11}$. Tabel yang satu ini menjadi contoh berjalan bagi seluruh §9.4.

Buku: Subbab 9.4.2 dan Gambar 9.1.

Maksimal Bukan Maksimum

Tabel 9.2 Contoh kasus Gambar 9.1 dengan durasi tiap aktivitas dihitung. Aktivitas a_{10} adalah raksasanya: dua belas satuan waktu, hampir sepanjang hari.

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16
durasi $f_i - s_i$	3	2	6	2	6	4	4	3	4	12	4

Tabel 9.2 menambahkan durasinya. $\{a_3, a_9, a_{11}\}$ selaras ($8 \geq 6$, $12 \geq 12$, $12 \geq 6$) dan tak ada yang dapat ditambahkan — namun isinya hanya tiga aktivitas.

$\{a_1, a_4, a_8, a_{11}\}$ dan $\{a_2, a_4, a_9, a_{11}\}$ sama-sama berisi **empat**, jumlah terbanyak yang mungkin — optimumnya **tidak tunggal**.

MAKSIMAL BUKAN MAKSIMUM. *Maksimal*: tak ada yang dapat ditambahkan. *Maksimum*: tak ada himpunan sah yang lebih besar. Setiap algoritma rakus menghasilkan yang maksimal tanpa susah payah; seluruh kesulitannya ada pada yang maksimum.

Buku: Subbab 9.4.2.

Empat Aturan Rakus Kandidat

Mulai terawal, **durasi terpendek**, **konflik tersedikit**, **selesai terawal**. Pada contoh berjalannya, dua aturan yang keliru tetap memperoleh empat:

```
=== Four greedy rules on the running example ===
  earliest finish time  -> {a1, a4, a8, a11}, size 4
  earliest start time   -> {a3, a7, a11}, size 3
  shortest duration     -> {a2, a4, a8, a11}, size 4
  fewest conflicts      -> {a1, a4, a8, a11}, size 4
```

Satu contoh kasus tidak membuktikan apa-apa. Pencarian menyeluruh menemukan contoh penyangkal terkecil bagi ketiga aturan yang keliru:

```
=== Smallest counterexample for each incorrect rule ===
  earliest start time on an instance with n = 3:
      activity :       a1       a2       a3
      [s, f)   :    [0,8)    [1,4)    [0,1)
      the rule picks 1 activities {a1}
      brute force says the optimum is 2  ->  the rule is WRONG

  shortest duration  on an instance with n = 3:
      activity :       a1       a2       a3
      [s, f)   :    [5,8)    [1,5)    [4,6)
      the rule picks 1 activities {a3}
      brute force says the optimum is 2  ->  the rule is WRONG

  fewest conflicts   on an instance with n = 9:
      activity :       a1       a2       a3       a4       a5       a6       a7       a8       a9
      [s, f)   :    [4,7)   [9,13)    [2,5)    [1,4)    [1,2)   [6,10)    [0,2)   [7,15)  [10,11)
      the rule picks 3 activities {a1, a2, a4}
      brute force says the optimum is 4  ->  the rule is WRONG
```

Buku: Subbab 9.4.3.

Mematahkan Durasi Terpendek dengan Tangan

$a_1 = [5, 8)$ (durasi 3), $a_2 = [1, 5)$ (durasi 4), $a_3 = [4, 6)$ (durasi 2). Aturannya mengambil yang terpendek, a_3.

a_1 bertindih dengan a_3 pada $[5, 6)$; a_2 bertindih dengannya pada $[4, 5)$. Keduanya dibuang: **satu** aktivitas.

Padahal $a_2 = [1, 5)$ dan $a_1 = [5, 8)$ selaras $(5 \geq 5)$: **dua** tersedia. Aktivitas pendek itu duduk di tengah dan meracuni kedua tetangganya.

> **MENGAPA SELESAI-TERAWAL BERBEDA.** Pilihlah aktivitas yang menyisakan sumber daya bagi sebanyak mungkin aktivitas lain. Yang paling awal selesai membebaskannya sejak saat paling dini; dengan masukan terurut, itu sekadar a_1. Itu gerak hati — kini kita perlu dua bukti: substruktur optimal dan sifat pilihan rakus.

Buku: Subbab 9.4.3; penelusuran konflik tersedikit dengan sembilan aktivitas ada di Soal Latihan 9.4.

Substruktur Optimal

Submasalah S_{ij} pada Persamaan (9.2): aktivitas yang mulai setelah a_i selesai dan selesai sebelum a_j mulai. Penjaga a_0 ($f_0 = 0$) dan a_{n+1} ($s_{n+1} = \infty$) membuat $S_{0,n+1} = S$.

$$S_{ij} = \{a_k \in S : f_i \leq s_k \text{ dan } f_k \leq s_j\}, \tag{9.2}$$

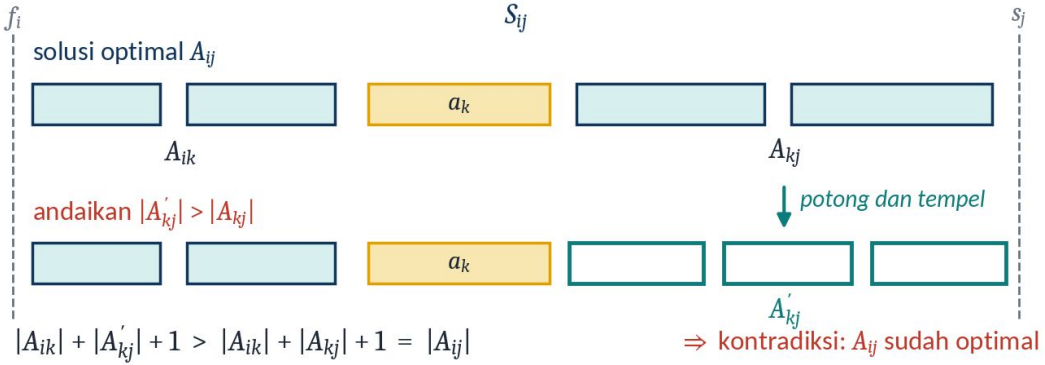
TEOREMA 9.1. Misalkan A_{ij} himpunan selaras berukuran maksimum di S_{ij} dengan $a_k \in A_{ij}$, dan misalkan $A_{ik} = A_{ij} \cap S_{ik}$, $A_{kj} = A_{ij} \cap S_{kj}$. Maka $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$, ukurannya terbelah seperti Persamaan (9.3), dan kedua potongannya berukuran maksimum di submasalahnya masing-masing.

$$|A_{ij}| = |A_{ik}| + |A_{kj}| + 1, \tag{9.3}$$

Buku: Subbab 9.4.4, Teorema 9.1.

Potong-dan-Tempel, Digambar

Gambar 9.2 menuntaskan Teorema 9.1 dengan kontradiksi dan menggambarkan rekurensi yang mengikutinya.



rekurensi:

$$c[i,j] = \begin{cases} 0 & \text{jika } S_{ij} = \varnothing \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{jika } S_{ij} \neq \varnothing \end{cases}$$

Gambar 9.2 Perampungan Teorema 9.1 lewat pengontradiksian, sekaligus menggambarkan rekurensi pemrograman dinamis bagi $c[i, j]$ yang mengikutinya. Andaikan ada himpunan A'_{kj} yang lebih baik, menyulihkannya ke dalam A_{ij} akan memberi lebih dari $|A_{ij}|$ aktivitas yang saling selaras — bertentangan dengan keoptimalan A_{ij}.

Buku: Subbab 9.4.4 dan Gambar 9.2.

Buktinya, Satu Baris per Langkah

1–2. Andaikan A_{kj} bukan maksimum: ada himpunan selaras $A'_{kj} \subseteq S_{kj}$ dengan $|A'_{kj}| > |A_{kj}|$. **3.** Potong dan tempel: $B = A_{ik} \cup \{a_k\} \cup A'_{kj}$.

4. Sahkah B? Semua yang di A_{ik} selesai paling lambat s_k; semua yang di A'_{kj} mulai paling cepat f_k; dan $s_k < f_k$. Jadi setiap pasangan selaras.

5–7. $|B| = |A_{ik}| + |A'_{kj}| + 1 > |A_{ij}|$, sebuah himpunan sah di S_{ij} yang lebih besar daripada yang maksimum — kontradiksi. **8.** Secara simetris untuk A_{ik}. ∎

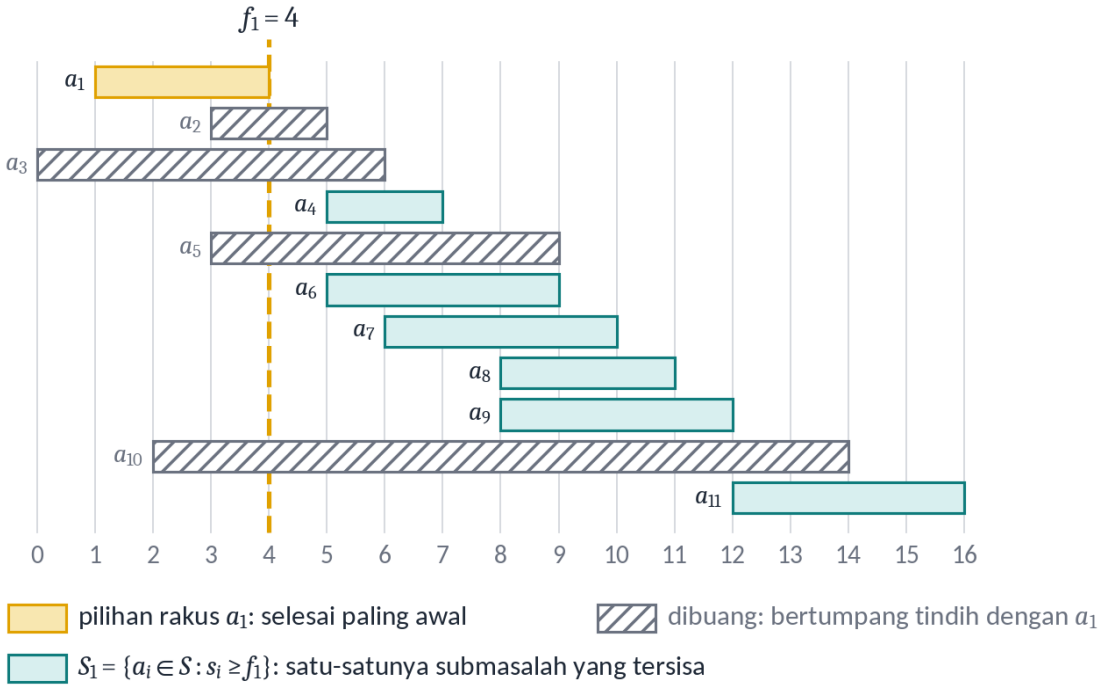
Teorema ini memberi program dinamis Persamaan (9.4): waktu $\Theta(n^3)$ dan ruang $\Theta(n^2)$ — bentuk rantai matriks lagi. Program ini akan segera kita buang.

$$c[i,j] = \begin{cases} 0 & \text{jika } S_{ij} = \emptyset, \\ \max\limits_{a_k \in S_{ij}} \{\, c[i,k] + c[k,j] + 1 \,\} & \text{jika } S_{ij} \neq \emptyset. \end{cases} \tag{9.4}$$

Buku: Subbab 9.4.4 dan 9.4.5.

Hanya Satu Submasalah yang Tersisa

Gambar 9.3 menyatakan penyusutannya. Definisikan keluarga berindeks tunggal Persamaan (9.5): aktivitas yang mulai setelah a_k selesai.



Gambar 9.3 Penyusutan menjadi satu submasalah: begitu pilihan rakus a_1 dijatuhkan, tersisalah satu submasalah S_1, dan solusi optimal bagi persoalan aslinya adalah a_1 bersama sebuah solusi optimal bagi S_1.

$$S_k = \left\{ a_i \in S : s_i \geq f_k \right\}, \tag{9.5}$$

Buku: Subbab 9.4.6 dan Gambar 9.3.

Tak Ada yang Menyelinap Sebelum a_1

Klaim. Setiap aktivitas yang selaras dengan a_1 mulai paling cepat f_1, sehingga setelah a_1 dipilih, persoalan sisanya tepat S_1.

Bukti. Jika a_i selaras dengan a_1, maka $s_i \geq f_1$ atau $s_1 \geq f_i$. Yang kedua memberi $f_i \leq s_1 < f_1$ — aktivitas yang selesai sebelum a_1, mustahil karena f_1 adalah waktu selesai minimum. ▰

Jadi setelah pilihan rakus hanya ada **satu** submasalah — bukan dua seperti pada Teorema 9.1, bukan n kandidat seperti pada Persamaan (9.4).

Buku: Subbab 9.4.6.

Sifat Pilihan Rakus

TEOREMA 9.2. Misalkan S_k tak kosong dan a_m punya waktu selesai terawal di S_k. Maka a_m termuat di dalam **suatu** himpunan berukuran maksimum berisi aktivitas yang saling selaras dari S_k.

Bukti (penukaran). Misalkan A_k **sembarang** himpunan bagian selaras berukuran maksimum dari S_k, dan a_j aktivitasnya yang pertama selesai. Jika $a_j = a_m$, selesai.

Jika tidak, **tukarkan**: $A'_k = (A_k \setminus \{a_j\}) \cup \{a_m\}$. Satu dibuang, satu ditambah, dan $a_m \notin A_k$, sehingga $|A'_k| = |A_k|$ — **tidak lebih kecil**.

Sah? Setiap $a_i \in A_k$ yang lain memenuhi $f_j \le s_i$; dan $f_m \le f_j$ karena a_m selesai paling awal di seluruh S_k. Jadi $f_m \le f_j \le s_i$: selaras. A'_k optimal dan memuat a_m. ∎

Buku: Subbab 9.4.7, Teorema 9.2.

Argumen Penukaran sebagai Cetakan

(1) Ambil **sembarang** solusi optimal A — sembarang saja, bukan yang enak dipakai.

(2) Temukan unsur A yang memainkan peran yang hendak dimainkan pilihan rakus (di sini: aktivitas A yang pertama selesai).

(3) Tukarkan pilihan rakus ke posisi itu; buktikan hasilnya **layak** dan **tidak lebih buruk**.

(4) Simpulkan bahwa ada solusi optimal yang memuat pilihan rakus — pilihan itu **aman**. Lalu substruktur optimal mengulangi argumennya pada sisanya.

AKIBAT 9.3. Jika a_m selesai paling awal di S_k dan A^* himpunan bagian selaras berukuran maksimum dari S_m, maka $\{a_m\} \cup A^*$ himpunan bagian selaras berukuran maksimum dari S_k — izin untuk mengambil a_m, berekursi sekali, lalu merekatkan.

Buku: Subbab 9.4.7 dan 9.4.8, Akibat 9.3.

Algoritma Rakus Rekursif

Gambar 9.4 memberikan **Recursive-Activity-Selector**; panggilan pertamanya dengan $k=0$ dan penjaga $f_0=0$.

```
    RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)
1   m = k + 1
2   while m ≤ n and s[m] < f[k]          // cari aktivitas pertama di S_k yang selesai
3       m = m + 1
4   if m ≤ n
5       return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)
6   else return ∅
```

Gambar 9.4 Recursive-Activity-Selector dalam pseudokode. Gelung **while** pada baris 2 mencari aktivitas pertama di S_k yang selesai; pemanggilannya dilakukan dengan $k=0$, memakai aktivitas fiktif a_0 dengan $f_0=0$.

Buku: Subbab 9.4.9 dan Gambar 9.4.

Membacanya Baris demi Baris

Baris 1 mulai di $m = k + 1$. **Baris 2–3** melompati setiap a_m dengan $s_m < f_k$ — setiap aktivitas yang tidak berada di S_k.

Baris 4 menanyakan apakah ada yang ditemukan. **Baris 5** mengambil a_m (Teorema 9.2) dan berekursi pada satu-satunya submasalah sisa S_m (Akibat 9.3). **Baris 6** mengembalikan $\varnothing$.

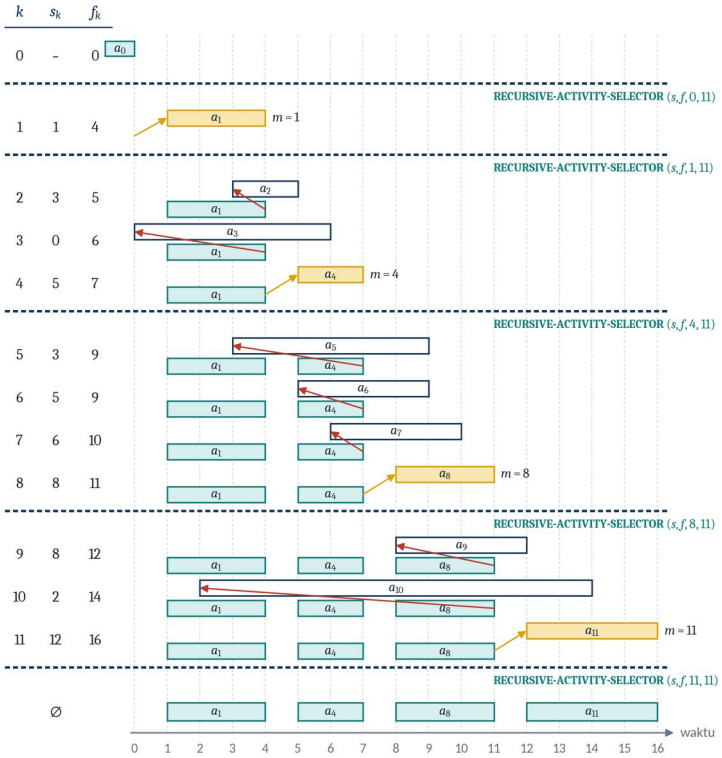
Dengan $f_0 = 0$, uji $s_m < 0$ bernilai salah bagi setiap aktivitas nyata, sehingga pilihan pertamanya a_1.

> **INI SEBENARNYA BUKAN REKURSI.** Pemanggilan rekursifnya adalah *tail call* (panggilan ekor): tak ada yang dihitung sesudahnya kecuali merekatkan a_m. Itu gelung yang menyamar, dan karena itulah Subbab 9.4.11 menuliskannya ulang sebagai gelung.

Buku: Subbab 9.4.9.

Penelusurannya, Digambar

Gambar 9.5 adalah penelusuran lengkap atas sebelas aktivitas pada Gambar 9.1 — gambar yang paling kaya informasi di bab ini.


k
s_k
f_k
0 - 0 a_0
RECURSIVE-ACTIVITY-SELECTOR $(s, f, 0, 11)$
1 1 4 a_1 $m = 1$
RECURSIVE-ACTIVITY-SELECTOR $(s, f, 1, 11)$
2 3 5 a_2 a_1
3 0 6 a_3 a_1
4 5 7 a_4 $m = 4$ a_1
RECURSIVE-ACTIVITY-SELECTOR $(s, f, 4, 11)$
5 3 9 a_5 a_1 a_4
6 5 9 a_6 a_1 a_4
7 6 10 a_7 a_1 a_4
8 8 11 a_8 $m = 8$ a_1 a_4
RECURSIVE-ACTIVITY-SELECTOR $(s, f, 8, 11)$
9 8 12 a_9 a_1 a_4 a_8
10 2 14 a_{10} a_1 a_4 a_8
11 12 16 a_{11} $m = 11$ a_1 a_4 a_8
RECURSIVE-ACTIVITY-SELECTOR $(s, f, 11, 11)$
∅ a_1 a_4 a_8 a_{11}
waktu
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Gambar 9.5 Penelusuran atas **Recursive-Activity-Selector** pada sebelas aktivitas Gambar 9.1. Tiap baris adalah satu iterasi gelung **while**: batang atasnya adalah aktivitas a_k (amber bila diambil), batang-batang toska adalah aktivitas yang sudah terpilih, dan anak panahnya menandai pembandingan s_k terhadap f milik pilihan terakhir. Garis-garis putus-putus tebalnya memisahkan pemanggilan-pemanggilan rekursif, masing-masing diberi label argumennya; baris terbawahnya memperlihatkan jawaban $\{a_1, a_4, a_8, a_{11}\}$ yang menempati sumber dayanya dari waktu 1 sampai waktu 16.

Buku: Subbab 9.4.10 dan Gambar 9.5.

Contoh Terkerjakan 1: Penelusuran dalam Kata

Panggilan 1 ($k=0$): $m=1$, apakah $1<0$? Tidak. Ambil a_1. **Panggilan 2** ($f_1=4$): $m=2$, $3<4$ lompati; $m=3$, $0<4$ lompati; $m=4$, $5<4$? Tidak — ambil a_4.

Panggilan 3 ($f_4=7$): a_5 ($3<7$), a_6 ($5<7$), a_7 ($6<7$) dilompati; $m=8$, $8<7$? Tidak — ambil a_8.

Panggilan 4 ($f_8=11$): a_9 ($8<11$), a_{10} ($2<11$) dilompati; $m=11$, $12<11$? Tidak — ambil a_{11}. **Panggilan 5**: $m=12>n$, kembalikan $\varnothing$.

```
=== RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, 11) ===
RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, 11)  -> m = 1, take a1
  RECURSIVE-ACTIVITY-SELECTOR(s, f, 1, 11)  -> m = 4, take a4
    RECURSIVE-ACTIVITY-SELECTOR(s, f, 4, 11)  -> m = 8, take a8
      RECURSIVE-ACTIVITY-SELECTOR(s, f, 8, 11)  -> m = 11, take a11
        RECURSIVE-ACTIVITY-SELECTOR(s, f, 11, 11)  -> no activity left, return {}
  result = {a1, a4, a8, a11}, size 4
```

HITUNGLAH KERJANYA. m mengambil nilai 1; $2,3,4$; $5,6,7,8$; $9,10,11$; 12 — setiap indeks diperiksa **tepat sekali**. Rekursinya mengerjakan total $\Theta(n)$.

Buku: Subbab 9.4.10.

Algoritma Rakus Iteratif

Gambar 9.6 adalah **Greedy-Activity-Selector**: k menyimpan pilihan terakhir, sehingga $f[k]$ adalah saat sumber daya kembali bebas.

```
GREEDY-ACTIVITY-SELECTOR (s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

Gambar 9.6 Greedy-Activity-Selector, bentuk iteratif dari Gambar 9.4. Peubah k selalu memegang indeks pilihan yang paling mutakhir, sehingga $f[k]$ adalah waktu saat sumber dayanya berikutnya menjadi bebas.

Buku: Subbab 9.4.11 dan Gambar 9.6.

Invarian dan Waktu Jalan

INVARIAN GELUNG. Pada awal tiap iterasi gelung **for**, A adalah himpunan bagian selaras berukuran maksimum dari $\{a_1, \ldots, a_{m-1}\}$ dan a_k adalah aktivitasnya dengan waktu selesai terakhir. Inisialisasi ($\{a_1\}$), pemeliharaan (Teorema 9.2 dan Akibat 9.3), terminasi pada $m = n+1$.

Kedua bentuknya $\Theta(n)$ pada masukan terurut; bersama pengurutan, **Teorema 9.4**: waktu $\Theta(n \lg n)$ dan ruang tambahan $\Theta(1)$. Tabel 9.3 mengukurnya.

Tabel 9.3 Pengurutan ditambah satu sapuan rakus, diukur pada aktivitas acak. Kolom ketiga adalah waktu yang berlalu dibagi $n \lg n$; nilainya tetap sekitar 4 di sepanjang rentang n selebar 32 kali, dan beginilah rupa $\Theta(n \lg n)$ pada mesin sungguhan.

n	Waktu (ms)	$t/(n \lg n) \times 10^6$	Aktivitas terpilih
250 000	18.07	4.03	122 426
500 000	36.84	3.89	177 718
1 000 000	91.81	4.61	252 379
2 000 000	163.89	3.91	357 471
4 000 000	373.26	4.25	504 635
8 000 000	762.48	4.16	713 102

Dibandingkan program dinamisnya pada $n = 1000$: sekitar sepuluh ribu operasi alih-alih 167 juta. **Inilah yang dibeli oleh sebuah bukti.**

Buku: Subbab 9.4.11–9.4.13, Teorema 9.4.

Strategi Rakus dalam Enam Langkah

Tabel 9.4 Enam langkah strategi rakus, dipetakan ke pekerjaan yang dikerjakan di §9.4. Tak ada yang dilewati dan tak ada yang ditambahkan.

Langkah	Tempatnya dikerjakan di §9.4	Apa yang dihasilkannya
1. substruktur optimal	§9.4.4, Teorema 9.1	$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$, potongannya optimal
2. solusi rekursif	§9.4.5	$c[i,j] = \max_k \{c[i,k] + c[k,j] + 1\}$
3. hanya tersisa satu submasalah	§9.4.6	setelah mengambil a_1, tinggal S_1
4. pilihan rakusnya aman	§9.4.7, Teorema 9.2	argumen penukaran
5. algoritma rekursif	§9.4.9	**Recursive-Activity-Selector**
6. algoritma iteratif	§9.4.11	**Greedy-Activity-Selector**

Tabel 9.4 memetakan keenam langkah itu ke pekerjaan yang baru saja kita lakukan. Bentuk singkatnya, tiga hal yang harus **dibuktikan**: bentuklah persoalannya sebagai *pilihan ditambah satu submasalah*; buktikan pilihan rakus itu **aman**; buktikan **substruktur optimal** sehingga pilihan ditambah subsolusi optimal adalah optimal.

Buku: Subbab 9.5.1.

Dua Bahan, Satu Resep

> **SIFAT PILIHAN RAKUS.** Untuk setiap contoh kasus, pilihan yang dijatuhkan aturan rakus termuat di dalam **suatu** solusi optimal. **SUBSTRUKTUR OPTIMAL.** Solusi optimal memuat solusi optimal bagi submasalah-submasalahnya.

Keduanya diperlukan, dan yang satu tidak menyiratkan yang lain. Ransel 0-1 punya substruktur optimal tetapi tidak punya sifat pilihan rakus; lintasan sederhana terpanjang tak punya keduanya.

Resepnya. Nyatakan tujuannya; daftarkan setiap aturan yang masuk akal; **coba patahkan masing-masing pada contoh kecil dengan gaya kasar (*brute force*)**; coba argumen penukaran pada yang bertahan; buktikan substruktur optimal; baru kemudian tulis kodenya — dan simpan program gaya kasarnya sebagai uji regresi.

Buku: Subbab 9.5.2–9.5.4.

Dua Masalah Ransel

Ransel 0-1: barang diambil utuh atau ditinggalkan. **Ransel pecahan:** boleh diambil sembarang pecahan $x_i \in [0,1]$. Keduanya punya substruktur optimal (potong-dan-tempel).

Aturan rakus yang wajar mengurutkan barang menurut kerapatan nilai Persamaan (9.6) dan mengambil yang terapat lebih dulu. Tabel 9.5 adalah contoh kasusnya, $W = 50$ kg.

$$\text{kerapatan}(i) = \frac{v_i}{w_i}, \tag{9.6}$$

Tabel 9.5 Contoh kasus ransel berisi tiga barang, dengan kapasitas $W = 50$ kg. Barangnya sudah tersusun menurun menurut nilai per kilogram, jadi barang 1 adalah pilihan pertama algoritma rakusnya.

Barang i	Bobot w_i (kg)	Nilai v_i	Kerapatan v_i/w_i
1	10	60	6.00
2	20	100	5.00
3	30	120	4.00

Buku: Subbab 9.6.1 dan 9.6.2.

Pecahan: Kerakusan Berhasil

Barang 1 (6,00/kg), seluruhnya: nilai 60, sisa ruang 40. Barang 2 (5,00/kg), seluruhnya: nilai 160, sisa 20. Barang 3: ambil $x_3 = 20/30 = 2/3$, nilai 80. **Total 240**, ransel tepat penuh.

Teorema 9.5 (penukaran). Misalkan y sembarang solusi optimal dan j indeks pertama tempat ia berbeda dari solusi rakus; maka $y_j < x_j$. Pindahkan bobot $\delta = (x_j - y_j)\, w_j$ ke barang j dari barang-barang belakangan; nilainya berubah sebesar Persamaan (9.7), yang ≥ 0 karena barang belakangan tidak lebih rapat.

$$\text{nilai}(y') - \text{nilai}(y) = \delta \cdot \frac{v_j}{w_j} - \sum_{i>j} \delta_i \cdot \frac{v_i}{w_i}, \tag{9.7}$$

DI MANA BUKTINYA MEMAKAI KETERBAGIAN. Kita harus menambahkan *tepat* δ dan membuang *tepat* δ. Di dunia 0-1 barang datang dalam bongkahan, dan boleh jadi tak ada bongkahan yang ukurannya pas.

Buku: Subbab 9.6.3, Teorema 9.5.

Contoh Terkerjakan 2: Kerakusan 0-1 Gagal

Aturan sama, barang utuh: barang 1 (sisa 40), barang 2 (sisa 20), barang 3 berbobot 30 — lewati. Hasil rakus: **160**, dengan 20 kg kosong. Tabel 9.6 menghargai kedelapan himpunan bagian.

Tabel 9.6 Setiap himpunan bagian contoh kasus di Tabel 9.5, lengkap dengan harganya. Optimumnya $\{2,3\}$ bernilai 220; aturan rakusnya mengembalikan $\{1,2\}$ bernilai 160.

Himpunan bagian	Bobot (kg)	Nilai	Muat di 50 kg?
$\{\}$	0	0	ya
$\{1\}$	10	60	ya
$\{2\}$	20	100	ya
$\{1,2\}$	30	160	ya — inilah jawaban rakusnya
$\{3\}$	30	120	ya
$\{1,3\}$	40	180	ya
$\{2,3\}$	50	220	ya — **inilah optimumnya**
$\{1,2,3\}$	60	280	tidak

Optimum $\{2, 3\}$: 50 kg, **220**. Kekurangannya $220 - 160 = 60$, yaitu $60/220 \approx 27{,}3\ \%$. Optimumnya *mengorbankan barang terapat* agar kapasitasnya terbagi pas.

Buku: Subbab 9.6.4.

Seberapa Parah, Seberapa Sering

```
=== Stress test: how often is the greedy 0-1 answer wrong? ===
  n =   5: greedy was suboptimal in   4241 of 20000 random instances (21.2%),  mean value/optimum =
0.9789,  worst = 0.4577
  n =  10: greedy was suboptimal in   7033 of 20000 random instances (35.2%),  mean value/optimum =
0.9868,  worst = 0.7821
  n =  20: greedy was suboptimal in   9666 of 20000 random instances (48.3%),  mean value/optimum =
0.9930,  worst = 0.9250

=== Fractional greedy is never wrong: same stress test ===
  n =   5: fractional greedy fell below the 0-1 optimum in 0 of 20000 instances
  n =  10: fractional greedy fell below the 0-1 optimum in 0 of 20000 instances
  n =  20: fractional greedy fell below the 0-1 optimum in 0 of 20000 instances
```

Kerakusan 0-1 **sering** keliru (48,3% contoh kasus pada $n=20$) dan biasanya keliru **sedikit** (rasio rata-rata 0,993) — tetapi kasus terburuk pada $n=5$ adalah 0,4577. “Ia berhasil pada data uji saya” bukanlah bukti. Tabel 9.7 menyandingkan kedua teknik.

Tabel 9.7 Rakus lawan pemrograman dinamis. Baris yang menentukan adalah yang kedua: sifat pilihan rakus adalah tuntutan tambahan kerakusan, dan justru itulah yang tak dimiliki ransel 0-1.

	Rakus	Pemrograman dinamis
Memerlukan substruktur optimal	ya	ya
Memerlukan sifat pilihan rakus	**ya**	tidak
Memerlukan submasalah yang bertumpang tindih	tidak	ya (§7.10.1)
Kapan pilihan pertamanya dijatuhkan?	**sebelum** submasalah mana pun diselesaikan	**sesudah** submasalahnya diselesaikan
Ada berapa submasalah sesudah sebuah pilihan?	tepat satu	umumnya beberapa, dengan beberapa calon pilihan
Arahnya	*top-down*, satu pilihan demi satu pilihan	*bottom-up*, atau *top-down* dengan *memoization*
Apakah ia meninjau ulang?	tak pernah	secara tersirat, dengan mengambil maks atau min atas semua pilihan
Biaya khasnya, pemilihan aktivitas	$\Theta(n \lg n)$	waktu $\Theta(n^3)$, ruang $\Theta(n^2)$
Ransel pecahan	**optimal** (Teorema 9.5)	berhasil tetapi tak perlu
Ransel 0-1	**keliru** — sampai 54% di bawah optimum	optimal, $\Theta(nW)$

Buku: Subbab 9.6.5 dan 9.6.6; program lengkapnya ada di Subbab 9.6.7.

Kode Huffman: Persoalannya

Sebuah berkas berisi 100 000 karakter atas a–f. Panjang tetap memerlukan $\lceil \lg 6 \rceil = 3$ bit per karakter: 300 000 bit. Kode panjang-berubah memberi karakter yang sering kata sandi pendek. Gambar 9.7 adalah contoh kasusnya.

	a	b	c	d	e	f
Frekuensi (dalam ribuan)	45	13	12	16	9	5
Kata kode panjang tetap	000	001	010	011	100	101
Kata kode panjang berubah	0	101	100	111	1101	1100

Gambar 9.7 Sebuah persoalan penyandian karakter. Sebuah berkas berisi 100 000 karakter hanya memuat a–f, dengan frekuensi sebagaimana ditampilkan (dalam ribuan). Kode panjang tetapnya berbiaya 300 000 bit; kode panjang berubah yang ditampilkan berbiaya 224 000.

Buku: Subbab 9.7.1 dan 9.7.2, serta Gambar 9.7.

Menghitung Bitnya

Tabel 9.8 Kedua kode Gambar 9.7, dengan biaya masing-masing dihitung karakter demi karakter. Dalam ribuan bit: 300 bagi kode panjang tetap, 224 bagi yang panjang berubah.

Karakter	Frekuensi (ribuan)	Kata sandi tetap	Bit	Kata sandi berubah	Bit	Frekuensi × panjang
a	45	000	3	0	1	$45 \times 1 = 45$
b	13	001	3	101	3	$13 \times 3 = 39$
c	12	010	3	100	3	$12 \times 3 = 36$
d	16	011	3	111	3	$16 \times 3 = 48$
e	9	100	3	1101	4	$9 \times 4 = 36$
f	5	101	3	1100	4	$5 \times 4 = 20$
total	**100**		**300**			**224**

Tabel 9.8: $45+39+36+48+36+20=224$ ribu bit melawan 300 — hemat 76 000 bit, $25,33\ \%$, atau $2,24$ bit per karakter.

Asal penghematannya: “a” menyusut dari 3 bit menjadi 1, hemat 90 000; e dan f tumbuh dari 3 menjadi 4, biaya 14 000; bersihnya 76 000.

Buku: Subbab 9.7.2.

Kode Bebas Awalan

Kode $a=0$, $b=01$, $c=1$ **taksa**: `001` dapat dibaca “aac” atau “ab”. Masalahnya: `0` adalah **awalan** dari `01`.

DEFINISI. Sebuah kode disebut **bebas awalan** (*prefix-free*) jika tak ada kata sandi yang menjadi awalan kata sandi lain.

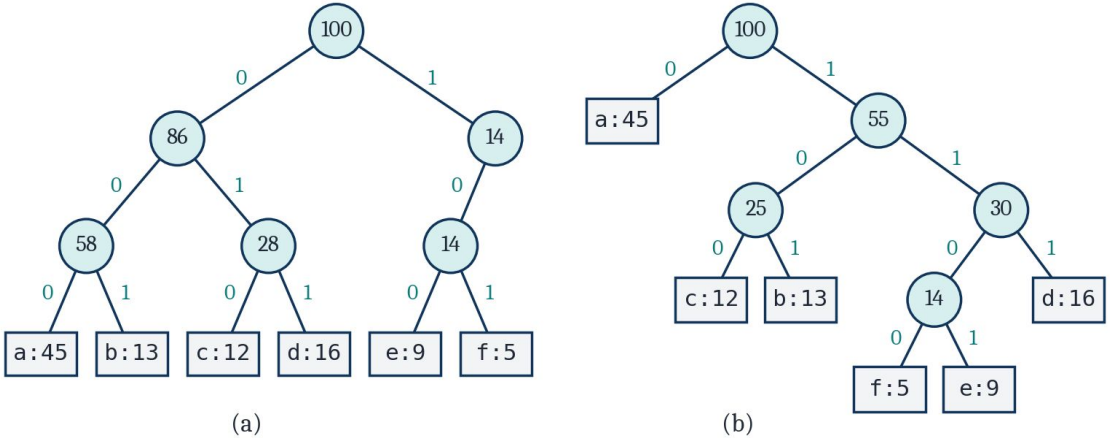
Penguraiannya kemudian rakus, dari kiri ke kanan, tanpa menengok ke depan: `0` → a; `1`, `10`, `101` → b; `1`, `10`, `100` → c; dan seterusnya.

Membatasi diri pada kode bebas awalan tidak merugikan: setiap kode yang terurai secara tunggal punya kode bebas awalan dengan panjang kata sandi yang sama.

Buku: Subbab 9.7.3.

Kode sebagai Pohon Biner

Sisi kiri `0`, sisi kanan `1`, karakter di daun: daun tak punya keturunan, jadi kodenya bebas awalan **menurut konstruksinya**. Gambar 9.8 menggambar kedua kodenya.



Gambar 9.8 Pohon-pohon bagi kedua skema penyandian Gambar 9.7. Tiap daun membawa sebuah karakter dan frekuensinya; tiap simpul dalam membawa jumlah frekuensi di dalam upapohonnya. **(a)** kode panjang tetap $a = 000, \ldots, f = 101$; **(b)** kode awalan optimal $a = 0$, $b = 101$, ..., $f = 1100$.

Buku: Subbab 9.7.4 dan Gambar 9.8.

Biaya Sebuah Pohon

Lema 9.6. Kode optimal adalah pohon **penuh**: $|C|$ daun dan $|C|-1$ simpul dalam (simpul beranak tunggal dapat dilewati, memendekkan kata sandinya).

Dengan $d_T(c)$ kedalaman daun c, biaya Persamaan (9.8) adalah total cacah bitnya. Pohon (b): $45(1)+13(3)+12(3)+16(3)+9(4)+5(4)=224$.

$$B(T)=\sum_{c\in C} c.freq\cdot d_T(c), \tag{9.8}$$

RUMUS SETARANYA. $B(T)$ juga sama dengan **jumlah label simpul dalam**: $100+55+25+30+14=224$. Karakter berkedalaman d terletak di bawah tepat d simpul dalam. Jadi biaya pohonnya adalah total penggabungan yang membangunnya.

Buku: Subbab 9.7.4 dan 9.7.5, Lema 9.6.

Algoritma Huffman

Gambar 9.9: Q adalah antrean prioritas-min menurut frekuensi; tiap iterasi dari $|C|-1$ iterasinya menggabungkan dua benda yang paling jarang menjadi simpul berlabel jumlah keduanya.

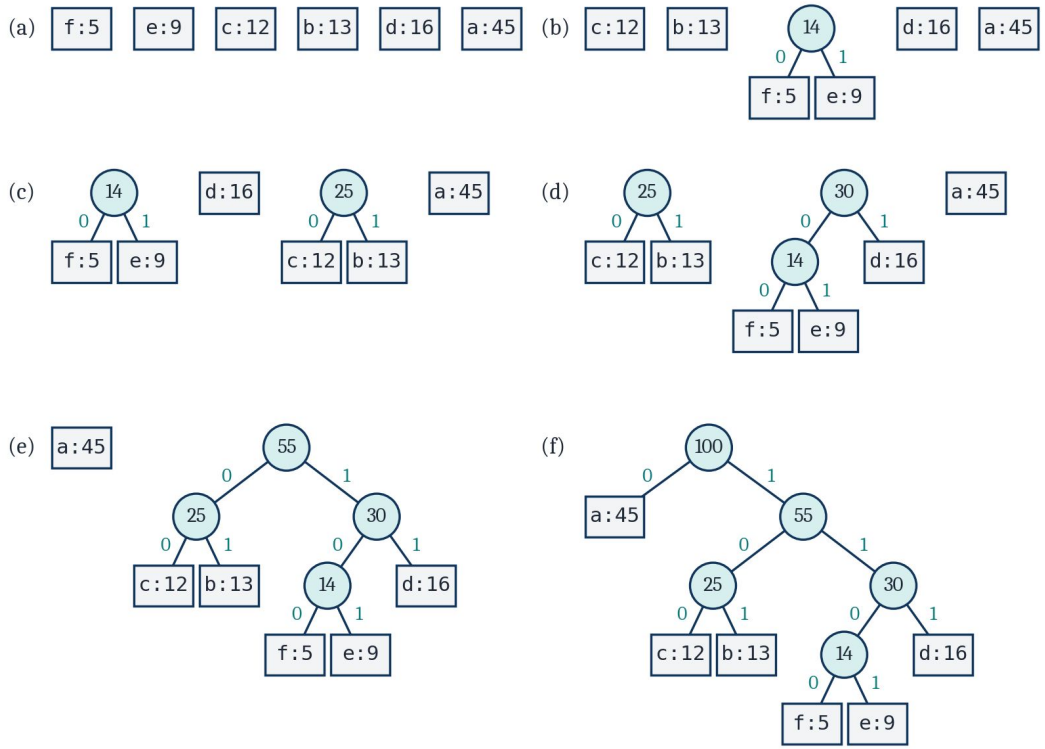
```
    HUFFMAN (C)
1   n = |C|
2   Q = C
3   for i = 1 to n − 1
4       allocate a new node z
5       z.left = x = EXTRACT-MIN (Q)
6       z.right = y = EXTRACT-MIN (Q)
7       z.freq = x.freq + y.freq
8       INSERT (Q, z)
9   return EXTRACT-MIN (Q)          // kembalikan akar pohonnya
```

Gambar 9.9 Huffman(C) dalam pseudokode. Q adalah antrean prioritas-min yang berkunci frekuensi; gelungnya berjalan $|C|-1$ kali, tiap kali menggantikan dua benda yang paling jarang muncul dengan simpul baru yang frekuensinya adalah jumlah keduanya.

Buku: Subbab 9.7.6 dan Gambar 9.9.

Penelusurannya, Digambar

Gambar 9.10 memperlihatkan antrean sebelum tiap penggabungan; panel (f) adalah pohon (b) pada Gambar 9.8.



Gambar 9.10 Penelusuran enam panel atas **Huffman** pada frekuensi Gambar 9.7. Tiap panelnya memperlihatkan antreannya terurut menaik menurut frekuensi; pada tiap langkah dua pohon berfrekuensi terendah digabungkan. Daun berupa persegi panjang, simpul dalam berupa lingkaran. **(a)** enam daun awalnya; **(b)**–**(e)** tahap-tahap antaranya; **(f)** pohon akhirnya.

Buku: Subbab 9.7.7 dan Gambar 9.10.

Contoh Terkerjakan 3: Lima Penggabungan

Tabel 9.9 Lima penggabungan **Huffman** atas frekuensi Gambar 9.7, dengan antreannya sesudah masing-masing. Jumlah berjalan label barunya adalah biaya $B(T)$ dalam ribuan bit, dan ia berakhir pada 224 — bilangan yang sama yang diperoleh Tabel 9.8 dengan menjumlahkan atas daun.

Penggabungan	Benda yang dikeluarkan	Label baru	Antrean sesudahnya	Jumlah berjalan
—	—	—	f:5 e:9 c:12 b:13 d:16 a:45	0
1	f:5 dan e:9	14	c:12 b:13 (14) d:16 a:45	14
2	c:12 dan b:13	25	(14) d:16 (25) a:45	39
3	(14) dan d:16	30	(25) (30) a:45	69
4	(25) dan (30)	55	a:45 (55)	124
5	a:45 dan (55)	100	(100)	**224**

Tabel 9.9: $f{:}5{+}e{:}9{=}14$; $c{:}12{+}b{:}13{=}25$; $(14){+}d{:}16{=}30$; $(25){+}(30){=}55$; $a{:}45{+}(55){=}100$. Lima penggabungan untuk enam karakter.

Total berjalan label-label barunya $14{+}25{+}30{+}55{+}100{=}224$ — persis Persamaan (9.8) yang dihitung atas daun.

Buku: Subbab 9.7.7.

Mengapa Huffman Optimal: Penukarannya

LEMA 9.7. Jika x, y berfrekuensi terendah, ada pohon optimal yang di dalamnya x dan y adalah daun bersaudara berkedalaman maksimum.

Ambil sembarang pohon optimal T dan saudara terdalam a, b. Tukar x dengan a: hanya dua kedalaman yang berubah, dan biayanya turun sebesar Persamaan (9.9).

$$B(T) - B(T') = (a.freq - x.freq)(d_T(a) - d_T(x)). \tag{9.9}$$

Kedua faktornya ≥ 0 (x paling jarang, a paling dalam), jadi penukarannya tak dapat menaikkan biaya; tukar y dengan b dengan cara yang sama. **Frekuensi rendah dan posisi dalam memang berjodoh.**

Buku: Subbab 9.7.8, Lema 9.7.

Substruktur, Teorema, Waktu Jalan

Lema 9.8. Ganti x, y dengan satu lambang z ber-$z.freq = x.freq + y.freq$. Pohon optimal T' apa pun bagi abjad yang lebih kecil, dengan z dimekarkan, optimal bagi C, karena Persamaan (9.10).

$$B(T) = B(T') + x.freq + y.freq. \tag{9.10}$$

Teorema 9.9. Huffman menghasilkan kode bebas awalan yang optimal — induksi atas $|C|$, dengan Lema 9.7 mengabsahkan Lema 9.8.

Teorema 9.10. Dengan *heap* biner-min: $O(n)$ untuk membangun, $n-1$ iterasi masing-masing $O(\lg n)$ — $O(n \lg n)$. Masukan yang sudah terurut: $O(n)$ dengan dua antrean (Soal Latihan 9.12).

Buku: Subbab 9.7.9–9.7.11; penyandi dan pengurai sandinya ada di Subbab 9.7.12.

Penyandinya Bekerja

```
=== Part 4: fixed-length against Huffman ===
  6 distinct characters need ceil(lg 6) = 3 bits each
  fixed-length total  = 3 x 100000 = 300000 bits
  Huffman total       = 224000 bits
  saving              = 76000 bits = 25.33%
  average codeword length = 2.24 bits per character

=== Part 5: encode and decode real text ===
  input length      : 126 characters
  distinct symbols  : 20
  codebook (symbol -> codeword), shortest first:
    '_'=101 'a'=111 'r'=001 'b'=0100 'd'=1100 'e'=1001
    'n'=1000 's'=1101 't'=0000 'h'=01101 'i'=00011 'l'=01110
    'u'=01010 'y'=01100 'c'=010111 'g'=000101 'o'=011111 'p'=010110
    'v'=000100 'w'=011110
  fixed-length cost : 1008 bits (8 bits per char)
  Huffman cost      : 489 bits in 62 bytes
  compression       : 51.49% smaller
  round trip        : IDENTICAL
  first 64 bits     : 1110100001111010111111110011101000011111011110100001111010111111
  decoded prefix    : "abracadabra abracadabra a laundry in sur..."

=== Part 6: the one-symbol edge case ===
  1000 copies of 'z': codeword "0", 1000 bits, round trip IDENTICAL
```

Kalimat berisi 126 karakter: 1008 bit polos, 489 bit tersandi — **51,5% lebih kecil** — dan perjalanan pulang-pergunya identik. Lambang yang jarang (“v”, “w”) mendapat kata sandi terpanjang, sebagaimana dituntut Lema 9.7.

Kasus tepinya menggigit: berkas berisi satu lambang berulang memberi kata sandi kosong; programnya memasangkan lambang tunggal itu dengan lambang boneka sehingga biayanya satu bit.

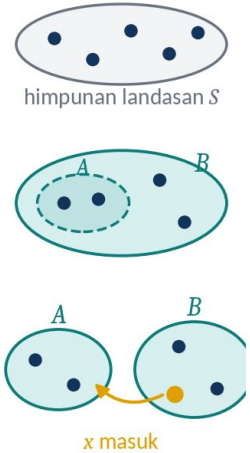
Buku: Subbab 9.7.12.

Matroid: Saat Kerakusan Otomatis Benar

Gambar 9.11 menyatakan definisinya: himpunan dasar S yang berhingga, keluarga I berisi himpunan bagian *bebas* yang **turun-temurun**, dan **sifat penukaran**.

Matroid adalah pasangan $M = (S, \mathcal{I})$ yang memenuhi

1. S adalah himpunan berhingga — himpunan landasan.
2. $\mathcal{I}$ adalah keluarga tak kosong himpunan bagian S, yakni himpunan-himpunan bebas, dan bersifat turun-temurun: $B \in \mathcal{I}$ dan $A \subseteq B$ berakibat $A \in \mathcal{I}$ (jadi $\varnothing \in \mathcal{I}$).
3. Sifat penukaran: jika $A, B \in \mathcal{I}$ dan $|A| < |B|$, maka ada $x \in B - A$ dengan $A \cup \{x\} \in \mathcal{I}$.



Gambar 9.11 Definisi sebuah matroid: himpunan landasan berhingga S, keluarga turun-temurun I berisi himpunan bagian yang *bebas*, dan sifat penukarannya.

Buku: Subbab 9.8.1 dan 9.8.2, serta Gambar 9.11.

Definisinya dengan Bahasa Sederhana

Turun-temurun: membuang sesuatu tak pernah merusak kumpulan yang sah; khususnya $\varnothing$ bebas. Bacalah *bebas* sebagai “sah”.

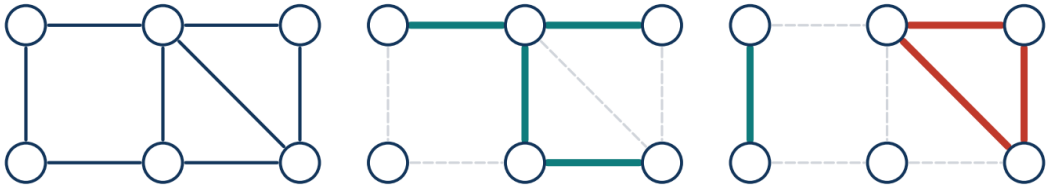
Penukaran: jika kumpulan sah Anda lebih kecil daripada milik orang lain, Anda dapat mengambil *suatu* barang milik mereka dan tetap sah.

> **SYARAT CUKUP, BUKAN SYARAT PERLU.** Teori matroid meliputi pohon merentang minimum dan penjadwalan tugas, tetapi **tidak** meliputi pemilihan aktivitas atau penyandian Huffman. Bukan-contoh: $A = \{a_3\} = \{[0, 6)\}$, $B = \{a_1, a_4\}$; tak satu pun unsur B dapat ditambahkan ke A.

Buku: Subbab 9.8.1–9.8.3.

Matroid Grafik

Gambar 9.12: himpunan dasarnya E, dan sehimpunan sisi bebas tepat bila ia **asiklik**. Sifat penukaran berlaku karena hutan yang lebih kecil punya komponen lebih banyak, sehingga hutan yang lebih besar punya sisi yang menghubungkan dua di antaranya.



Gambar 9.12 Matroid grafik $M_G = (S_G, I_G)$ dari sebuah graf tak berarah $G = (V, E)$: himpunan landasannya adalah E, dan sehimpunan sisi bebas tepat ketika ia nirsiklus — yaitu ketika ia membentuk sebuah hutan. **(a)** Sebuah graf G; **(b)** himpunan bebas A berisi empat sisi tanpa siklus; **(c)** himpunan tak bebas B, yang sisi-sisi merahnya menutup sebuah siklus. M_G berkaitan erat dengan persoalan pohon merentang minimum, yang dibahas Bab 11 secara terperinci.

Buku: Subbab 9.8.3 dan Gambar 9.12.

Algoritma Rakus pada Matroid

LEMA 9.12. Semua himpunan bebas yang maksimal dalam sebuah matroid berukuran sama. Jadi di dalam matroid **maksimal = maksimum**, dan kemaksimalan yang diperoleh kerakusan tanpa susah payah sudah cukup.

Pseudokode 9.1: urutkan menurun menurut bobot, pertahankan setiap unsur yang menjaga kebebasan. Biayanya Persamaan (9.11), dengan $f(n)$ biaya satu uji kebebasan.

Pseudokode 9.1 Greedy(M, w), algoritma rakus matroid berbobot. §9.8.8 memberikannya sebagai program utuh yang dapat dikompilasi, dengan orakel kebebasan yang abstrak.

```
GREEDY(M, w)
1  A = {}
2  sort M.S into monotonically decreasing order by weight w
3  for each x in M.S, taken in that order
4      if A union {x} is in M.I          // apakah himpunan yang diperbesar tetap bebas?
5          A = A union {x}
6  return A
```

$$O(n \lg n + n f(n)). \tag{9.11}$$

Lema 9.13 (pilihan rakus, lewat penukaran), **Lema 9.14** (kontraksi), **Teorema 9.15**: Greedy optimal pada **setiap** matroid berbobot — buktinya ditulis sekali, untuk semuanya.

Buku: Subbab 9.8.4 dan 9.8.5, Teorema 9.15.

Tugas Berwaktu-Satuan dengan Tenggat

n tugas berwaktu-satuan; tugas a_i bertenggat d_i dan berpenalti w_i bila terlambat. Yang penting hanya **himpunan** tugas yang tepat waktu (bentuk awal-dahulu), dan menurut Persamaan (9.12) kita boleh memaksimumkan alih-alih meminimumkan.

$$\text{minimumkan} \sum_{\text{terlambat}} w_i \Leftrightarrow \text{maksimumkan} \sum_{\text{awal}} w_i. \tag{9.12}$$

Himpunan A **bebas** jika semua tugasnya dapat tepat waktu. Dengan $N_t(A)$ dari Persamaan (9.13), **Lema 9.16**: A bebas $\Leftrightarrow N_t(A) \le t$ untuk semua t $\Leftrightarrow$ urutan tenggat tak membuat satu pun tugas A terlambat.

$$N_t(A) = \left| \{ a_i \in A : d_i \le t \} \right| \tag{9.13}$$

Teorema 9.17. (S, I) adalah matroid — jadi kerakusan optimal tanpa bukti tambahan.

Buku: Subbab 9.8.6, Lema 9.16 dan Teorema 9.17.

Contoh Kasusnya

Gambar 9.13: $d = \langle 4, 2, 4, 3, 1, 4, 6 \rangle$, $w = \langle 70, 60, 50, 40, 30, 20, 10 \rangle$, total 280, sudah terurut menurun menurut penalti.

Tugas

a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Gambar 9.13 Sebuah contoh kasus bagi persoalan menjadwalkan tugas berwaktu-satuan dengan tenggat dan penalti pada satu pemroses: tujuh tugas $a_1, \ldots, a_7$ dengan tenggat d_i dan penalti w_i.

Buku: Subbab 9.8.7 dan Gambar 9.13.

Contoh Terkerjakan 4: Greedy Menjadwalkan

Tabel 9.10 Greedy pada tujuh tugas Gambar 9.13, satu baris per unsur yang ditinjau dalam urutan penalti yang menurun. Bobot awal 230, penalti yang dibayar 50.

Ditinjau	d	w	Uji kebebasan	Vonis	A Sesudahnya
a_1	4	70	$N_4 = 1 \le 4$	simpan	$\{a_1\}$
a_2	2	60	$N_2 = 1 \le 2$, $N_4 = 2 \le 4$	simpan	$\{a_1, a_2\}$
a_3	4	50	$N_4 = 3 \le 4$	simpan	$\{a_1, a_2, a_3\}$
a_4	3	40	$N_3 = 2 \le 3$, $N_4 = 4 \le 4$	simpan	$\{a_1, a_2, a_3, a_4\}$
a_5	1	30	$N_4 = 5 > 4$	**tolak**	$\{a_1, a_2, a_3, a_4\}$
a_6	4	20	$N_4 = 5 > 4$	**tolak**	$\{a_1, a_2, a_3, a_4\}$
a_7	6	10	$N_6 = 5 \le 6$	simpan	$\{a_1, a_2, a_3, a_4, a_7\}$

Tabel 9.10: a_5 dan a_6 ditolak karena lima tugas akan jatuh tempo sampai waktu 4. Himpunan tepat waktu $\{a_1, a_2, a_3, a_4, a_7\}$ berbobot 230; penaltinya $30 + 20 = 50$.

Jadwal menurut urutan tenggat: $\langle a_2, a_4, a_1, a_3, a_7, a_5, a_6 \rangle$. Gaya kasar atas $2^7 = 128$ himpunan bagian setuju; demikian pula 200 000 contoh kasus acak.

Buku: Subbab 9.8.7; dengan orakel keasiklikan, kode yang sama menjadi algoritma Kruskal (Subbab 9.8.8).

Katalog Algoritma Rakus

Tabel 9.11 Algoritma rakus dalam buku ini. Lajur ketiganya adalah aturan lokalnya; lajur keempatnya adalah kewajiban pembuktiannya. Perhatikan betapa banyak entri di lajur keempat yang berbunyi “sebuah argumen penukaran”.

Algoritma	Persoalan	Pilihan rakusnya	Mengapa ia aman	Bab
Greedy-Activity-Selector	himpunan aktivitas selaras terbesar	aktivitas yang paling awal selesai	Teorema 9.2, argumen penukaran	9 (di sini)
Fractional-Knapsack	muatan terbagi yang paling bernilai	barang dengan nilai per kg tertinggi	Teorema 9.5, argumen penukaran	9 (di sini)
Huffman	kode bebas awalan yang optimal	gabungkan dua lambang yang paling jarang	Lema 9.7 dan 9.8	9 (di sini)
Greedy(M, w)	himpunan bebas berbobot maksimum	unsur terberat yang menjaga kebebasan	Teorema 9.15	9 (di sini)
Penjadwalan tugas	total penalti terkecil	tugas berpenalti tertinggi yang menjaga kebebasan	Teorema 9.17 + 9.15	9 (di sini)
Kruskal	pohon merentang minimum	sisi teringan yang tak menutup siklus	matroid grafik, Teorema 9.15	11
Prim	pohon merentang minimum	sisi teringan yang meninggalkan pohon yang sudah terbangun	sifat potongan, sebuah argumen penukaran	11
Dijkstra	lintasan terpendek sumber tunggal, bobot taknegatif	simpul belum-dikunjungi dengan jarak sementara terkecil	argumen penukaran atas lintasan	13
Pengurutan topologis (Kahn)	urutan linear yang menghormati sebuah DAG	sembarang simpul yang tak lagi punya pendahulu	setiap DAG punya simpul semacam itu	11

Tabel 9.11: Kruskal adalah Greedy pada matroid grafik; Prim dan Dijkstra bukan algoritma matroid, tetapi buktinya adalah argumen penukaran dengan empat gerakan yang sama.

Buku: Subbab 9.9.

Profitina dalam Praktik

Penjadwal pemindaian ulang — pemindaian berebut pekerja: pemilihan aktivitas, urutkan menurut waktu *selesai*, bukan siapa datang dulu dilayani dulu.

Anggaran perayapan — halaman tak dapat diambil setengah: ransel **0-1**, tempat kerakusan keliru; sertakan batas pecahan di samping setiap jawaban rakus.

Penyimpanan analitik — kode status yang timpang: kompresi Huffman dengan perjalanan pulang-pergi yang terverifikasi.

Profitina Laundry: pemesanan pengering — selesai terawal melayani 4, datang-dulu-dilayani-dulu 3; log status termampatkan 82,74%; mobil antar (0-1) kehilangan 68 dari 683 (10,0%) karena kerakusan.

Buku: Subbab 9.10 dan Subbab 9.10.1. Aplikasi memberi saran; pengguna yang memutuskan.

Kesalahan yang Sering Terjadi

1. Menyebut algoritma rakus benar tanpa argumen penukaran. 2. Menguji pada satu contoh kasus — dua aturan keliru tetap benar pada contoh berjalannya.

3. Mengurutkan dengan kunci yang salah (mulai, durasi). **4. Selang tertutup** — ujilah `s[m] >= f[k]`, jangan pernah `>`.

5. Aturan kerapatan pada ransel 0-1. 6. Menganggap rata-rata sebagai batas (rata-rata 0,993, terburuk 0,458).

7. Kode yang tidak bebas awalan. 8. Berkas Huffman berlambang tunggal. 9. Lupa menyimpan pohonnya. 10. *Heap*-maks padahal Huffman memerlukan *heap*-min.

11. Mengira persoalannya matroid tanpa memeriksa sifat penukaran. **12. Meminimumkan dengan Greedy** tanpa transformasi bobot yang sah. **13. Maksimal disangka maksimum. 14. Menambahkan nilai pada pemilihan aktivitas.**

Buku: Subbab 9.11.

Ringkasan: Lima Hal untuk Diingat

1. Algoritma rakus cepat dan biasanya keliru; **buktinya adalah seluruh pekerjaannya** — aturannya tak pernah berubah, hanya strukturnya (Tabel 9.1).

2. Dua bahan: **sifat pilihan rakus** (dibuktikan dengan argumen penukaran) dan **substruktur optimal** (potong-dan-tempel).

3. Pemilihan aktivitas: **selesai terawal**, $\Theta(n \lg n)$; jawaban pada contoh berjalannya adalah $\langle a_1, a_4, a_8, a_{11} \rangle$.

4. Ransel pecahan: kerakusan optimal (240); 0-1: rakus 160 melawan 220. Huffman: gabungkan dua yang paling jarang; $B(T) = 224$, jumlah penggabungannya.

5. Matroid: maksimal sama dengan maksimum, sehingga Greedy optimal pada setiap matroid berbobot — grafik (Kruskal) dan penjadwalan tugas (penalti 50).

Buku: Subbab 9.12.

Pertemuan Berikutnya

Bab 10 — Algoritma Graf Dasar. Masukan tidak lagi berupa barisan, melainkan graf: simpul, sisi, dan tiga cara menyimpannya.

Penelusuran melebar (sebuah antrean) dan **penelusuran mendalam** (sebuah tumpukan), keduanya $O(V+E)$, beserta enam persoalan yang diselesaikannya.

Bab 11 lalu kembali ke kerakusan: Kruskal dan Prim — dan Subbab 9.8.8 sudah menjalankan Kruskal pada grafnya. Jawabannya, bobot 37, menunggu untuk diperiksa.

Sebelum pertemuan: kerjakan Soal Latihan 9.1, 9.5, 9.10, dan 9.17, lalu bacalah Subbab 10.1 sampai 10.5.

Buku: Subbab 9.12.1 dan Bab 10.

BAB 1

Kuliah 10 — Algoritma Graf Dasar: Penelusuran Melebar dan Penelusuran Mendalam

Perancangan dan Analisis Algoritma (EF234405)

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*

Dosen: Irfan Subakti — yifana@gmail.com

Buku: Bab 10, Algoritma Graf Dasar. Setiap halaman handout ini diakhiri dengan penunjuk ke buku, yang menjadi rujukan utama.

Pertemuan Hari Ini

1. Apa itu graf, untuk apa graf itu, dan seluruh kosakatanya — derajat, lintasan, siklus, komponen, pohon (Subbab 10.2–10.4).

2. Tiga cara menyimpan graf, dan ongkos masing-masing (Subbab 10.5).

3. Penelusuran melebar: algoritmanya, penelusuran lengkap, programnya, $O(V+E)$, dan bukti bahwa ia menemukan lintasan terpendek (Subbab 10.6–10.7).

4. Penelusuran mendalam: cap waktu, hutannya, teorema tanda kurung dan teorema lintasan putih, empat golongan sisi (Subbab 10.8).

5. Enam hal yang dapat dikerjakan kedua penelusuran, teorema DAG, dan pengurutan topologis; lalu Profitina, kesalahan yang sering terjadi, dan ringkasannya (Subbab 10.9–10.12).

Buku: Bab 10, Subbab 10.1 sampai 10.12.

Dari Mana Kita Melanjutkan

Sembilan bab tentang **barisan**: larik terurut, *heap*, pohon pencarian, tabel pemrograman dinamis — dalam setiap kasus strukturnya *kita sendiri yang memilih*.

Bab 9 mengubah gaya argumennya: pilihan rakus, dibuktikan aman lewat penukaran.

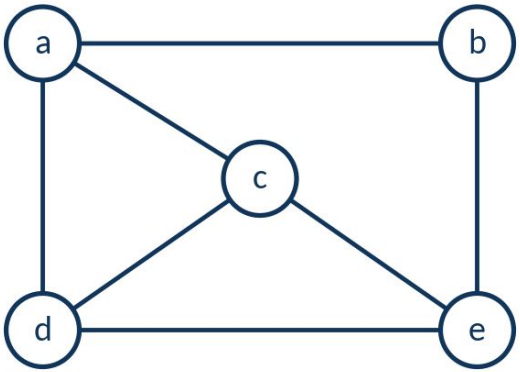
Mulai sekarang masukan **datang dengan struktur** yang tak dapat kita atur ulang: jaringan jalan, tautan antarhalaman web, prasyarat dalam sebuah *build*. Objeknya adalah **graf**.

Yang terbawa: notasi asimtotik (kini dengan dua ukuran, V dan E), antrean dan tumpukan, invarian gelung. Yang baru: penelusuran dapat mengunjungi simpul yang sama dua kali, sehingga setiap algoritma membawa **warna** untuk mengingat apa yang sudah dilihatnya.

Buku: Subbab 10.1 dan Subbab 10.1.1.

Apa Itu Graf

DEFINISI. Sebuah **graf** $G=(V,E)$ adalah himpunan V berisi **simpul** dan himpunan $E \subseteq V \times V$ berisi **sisi**. Pasangan terurut: **graf berarah**; pasangan tak terurut: **graf tak berarah**. Gambar 10.1 adalah contoh yang dipakai di sepanjang bab.



$$V = \{a, b, c, d, e\}$$

$$E = \{(a, b), (a, c), (a, d), (b, e), (c, d), (c, e), (d, e)\}$$

Gambar 10.1 Definisi graf, beserta graf contoh yang dipakai sepanjang bab ini: $V=\{a,b,c,d,e\}$ dan $E=\{(a,b),(a,c),(a,d),(b,e),(c,d),(c,e),(d,e)\}$. Lima simpul, tujuh sisi. Perhatikan bahwa gambarnya dan kedua daftar himpunan itu membawa keterangan yang persis sama; gambarnya hanya lebih enak dipandang.

Buku: Subbab 10.2.1–10.2.3 dan Gambar 10.1.

Membaca Graf, Memodelkan Persoalan

Lima simpul, tujuh sisi; pasangan yang tidak ada adalah (a,e), (b,c), (b,d) — dari $\binom{5}{2}=10$ pasangan yang mungkin, tujuh hadir. **Gambar bukanlah grafnya.**

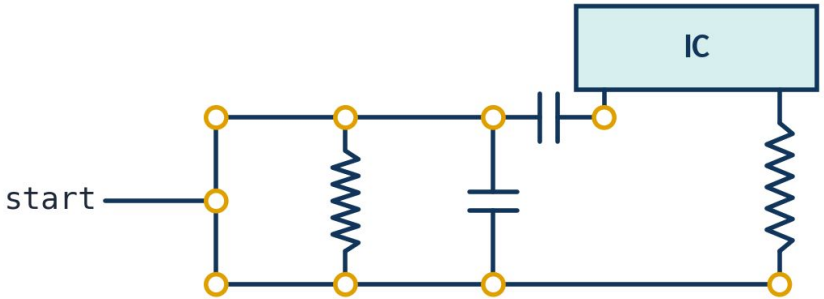
Gelung-diri adalah sisi (v,v); graf tanpa gelung-diri dan tanpa sisi berulang disebut **sederhana**; sisi sejajar menghasilkan **multigraf**.

> **MEMODELKAN DALAM TIGA PERTANYAAN.** *Apa bendanya?* Menjadi V. *Kapan dua benda terhubung?* Aturan itu menjadi E. *Apakah hubungannya punya arah?* Baru sesudah itu pilihlah algoritmanya.

Penerapannya: rangkaian listrik (Gambar 10.2), jaringan penerbangan dan jalan (Gambar 10.3), kompilator (urutan tugas), grafika (*flood fill* adalah BFS), labirin (DFS), peta, pengelompokan.

Buku: Subbab 10.2.3, 10.2.4, dan 10.3.

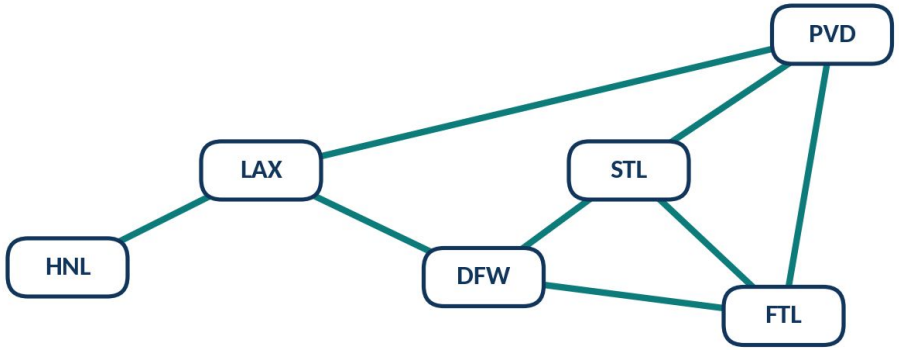
Rangkaian Listrik adalah Graf



Gambar 10.2 Sebuah rangkaian elektronik sebagai graf. Simpulnya adalah titik-titik solder tempat komponen bertemu; sisinya adalah komponennya — resistor, kapasitor — yang menghubungkan titik-titik itu. Pertanyaan “berapa lintasan dengan hambatan terkecil dari `start` ke blok bertanda IC?” adalah pertanyaan lintasan terpendek pada graf ini.

Buku: Subbab 10.3 dan Gambar 10.2.

Jaringan Penerbangan adalah Graf

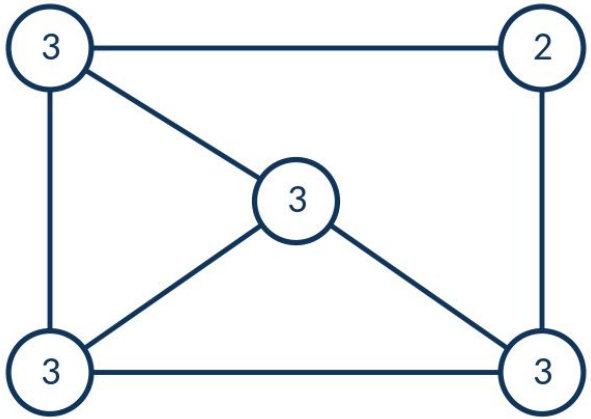


Gambar 10.3 Sebuah jaringan penerbangan sebagai graf: simpulnya adalah bandara (HNL, LAX, DFW, STL, FTL, PVD), sisinya adalah rute yang diterbangi di antaranya. “Bisakah saya pergi dari Honolulu ke Providence, dan dengan berapa kali ganti pesawat?” persis merupakan pertanyaan yang dijawab penelusuran melebar di §10.7.

Buku: Subbab 10.3 dan Gambar 10.3. “Lompatan tersedikit” adalah bab ini; “jarak terpendek” adalah Bab 13.

Derajat, Digambarkan

$Adj[v]$ adalah himpunan **tetangga** v; **derajatnya** $\deg(v) = |Adj[v]|$. Gambar 10.4 melabeli tiap simpul dengan derajatnya.



Gambar 10.4 Derajat. Setiap simpul graf bab ini diberi label derajatnya alih-alih namanya. Dibaca berlawanan arah jarum jam dari kiri atas, labelnya adalah $3, 2, 3, 3, 3$; jumlahnya 14, yaitu dua kali banyaknya sisi.

Buku: Subbab 10.4.1 dan 10.4.2, serta Gambar 10.4.

Lema Jabat Tangan

$Adj[a]=\{b,c,d\}$, $Adj[b]=\{a,e\}$, $Adj[c]=\{a,d,e\}$, $Adj[d]=\{a,c,e\}$, $Adj[e]=\{b,c,d\}$: derajatnya $3+2+3+3+3=14=2|E|$.

$$\sum_{v \in V} \deg(v) = 2|E|. \tag{10.1}$$

Bukti Persamaan (10.1)**:** cacahlah ujung-sisi dua kali — menurut sisi ($2|E|$) dan menurut simpul ($\sum_v \deg(v)$). Akibatnya: banyaknya simpul berderajat ganjil selalu genap; **seluruh daftar ketetanggaan bersama-sama memuat** $2|E|$ **entri**; derajat rata-ratanya $2|E|/|V|$.

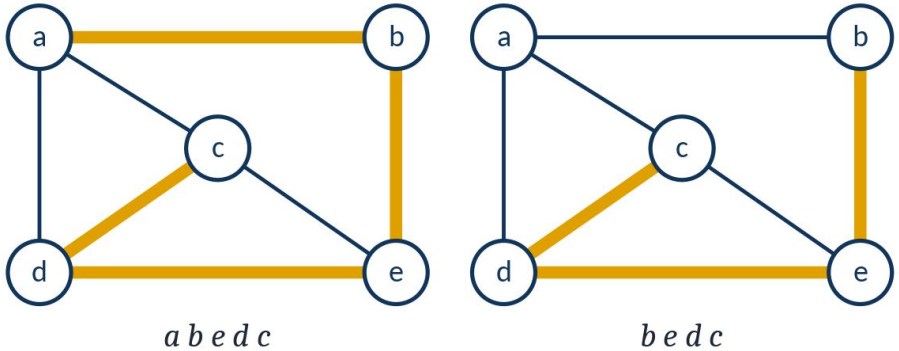
Graf berarah: derajat keluar $\deg^+$ dan derajat masuk $\deg^-$ masing-masing berjumlah $|E|$, seperti dinyatakan Persamaan (10.2). Derajat masuk 0 disebut **sumber**, derajat keluar 0 disebut **muara**.

$$\sum_{v \in V} \deg^+(v) = |E|, \sum_{v \in V} \deg^-(v) = |E|, \text{sehingga} \sum_{v \in V} (\deg^+(v) + \deg^-(v)) = 2|E|. \tag{10.2}$$

Buku: Subbab 10.4.3 dan 10.4.4.

Lintasan dan Panjangnya

Lintasan adalah $v_1, \ldots, v_k$ dengan simpul berurutan yang bertetangga; **panjangnya** $k-1$ sisi; ia **sederhana** jika tak ada simpul berulang. Gambar 10.5 memperlihatkan dua.

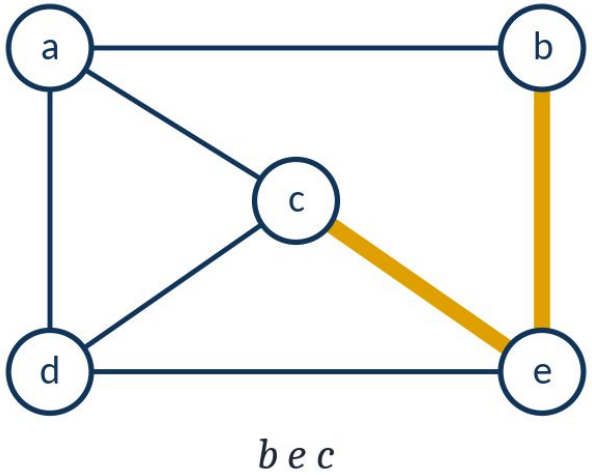


Gambar 10.5 Dua lintasan pada graf bab ini, disorot. Di sebelah kiri, $a\,b\,e\,d\,c$, berpanjang 4; di sebelah kanan, $b\,e\,d\,c$, berpanjang 3. Keduanya sederhana: tidak ada simpul yang dipakai dua kali.

Buku: Subbab 10.4.5 dan Gambar 10.5.

Lintasan Terpendek dan Lintasan Sederhana

Gambar 10.6: $b\,e\,c$ berpanjang 2 dan terpendek, karena tidak ada sisi (b, c). Kita menulis $\delta(u, v)$ untuk sisi tersedikit pada lintasan mana pun (∞ jika tak terjangkau).



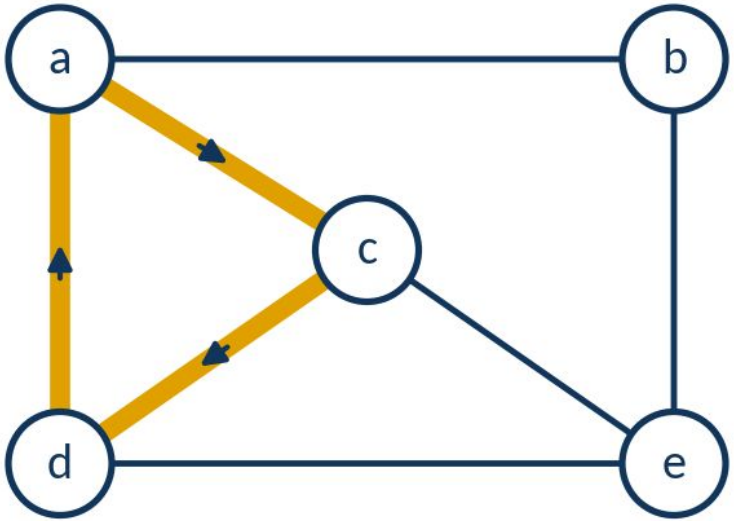
Gambar 10.6 Lintasan ketiga, $b\,e\,c$, berpanjang 2. Inilah lintasan terpendek dari b ke c pada graf ini — tidak ada sisi (b, c), jadi tidak ada lintasan berpanjang 1.

WAWASAN KUNCI. Jika ada lintasan dari u ke v, ada pula lintasan **sederhana** yang tidak lebih panjang — potong putaran di antara dua kemunculan sebuah simpul. Jadi tak ada lintasan terpendek yang lebih dari $|V| - 1$ sisi.

Buku: Subbab 10.4.5 dan Gambar 10.6.

Sebuah Siklus

Siklus adalah lintasan sederhana yang simpul terakhirnya sama dengan simpul pertamanya; graf tanpa siklus disebut **tak bersiklus**. Gambar 10.7 menyorot segitiga $a\,c\,d\,a$.

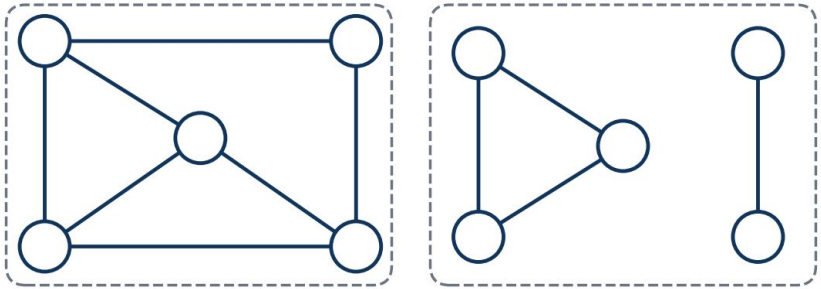


Gambar 10.7 Sebuah siklus pada graf bab ini, disorot: $a\,c\,d\,a$. Berangkatlah dari a, berjalanlah ke c, berjalanlah ke d, berjalanlah kembali ke a — tiga sisi, dan Anda berada di tempat Anda memulai tanpa pernah mengulang satu simpul pun di perjalanan.

Buku: Subbab 10.4.6 dan Gambar 10.7. Graf berarah tak bersiklus disebut DAG (Subbab 10.9.5).

Terhubung dan Tak Terhubung

Graf tak berarah disebut **terhubung** jika setiap pasangan simpul dihubungkan oleh sebuah lintasan. Gambar 10.8 membandingkan kedua keadaan.

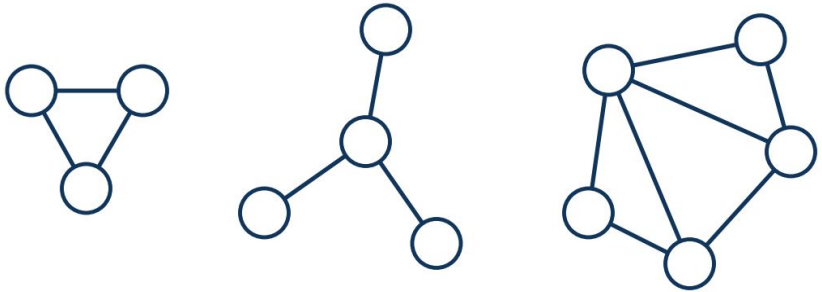


Gambar 10.8 Terhubung dan tidak terhubung. Di sebelah kiri, setiap simpul dapat dicapai dari setiap simpul lain. Di sebelah kanan simpul-simpul yang sama digambar dengan lebih sedikit sisi, dan dua simpul di ujung kanan terputus dari segitiga di sebelah kiri: tidak ada lintasan yang menghubungkannya, jadi grafnya tak terhubung.

Buku: Subbab 10.4.7 dan Gambar 10.8.

Komponen Terhubung

Komponen adalah subgraf terhubung yang *maksimal*. Setiap simpul berada di tepat satu komponen; tak ada sisi yang menghubungkan dua komponen; jadi penelusuran yang dimulai di dalam sebuah komponen tak pernah dapat keluar. Gambar 10.9 punya tiga.

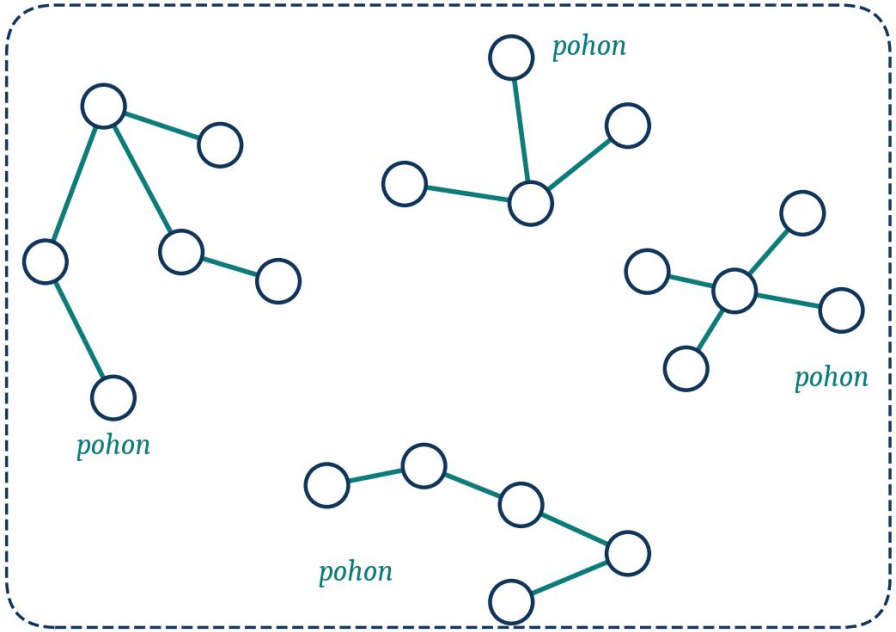


Gambar 10.9 Sebuah graf dengan tiga komponen terhubung: sebuah segitiga di sebelah kiri, sebuah bintang di tengah, dan sebuah keping berisi lima simpul di sebelah kanan. Tidak ada sisi yang menyeberang di antara ketiganya, dan di dalam masing-masingnya setiap simpul mencapai setiap simpul lain.

Buku: Subbab 10.4.8 dan Gambar 10.9.

Pohon dan Hutan

Pohon (bebas) terhubung dan tak bersiklus; **hutan** tak bersiklus. Untuk $|V|$ simpul, sembarang dua dari *terhubung*, *tak bersiklus*, $|E|=|V|-1$ menyiratkan yang ketiga. Gambar 10.10 memperlihatkan keduanya.



Gambar 10.10 Pohon dan hutan. Masing-masing dari keempat graf itu sendiri-sendiri adalah sebuah *pohon*: terhubung, dan tanpa siklus di mana pun di dalamnya. Keempatnya bersama-sama membentuk satu graf yang tak bersiklus tetapi tidak terhubung — sebuah *hutan*, dengan empat komponen.

Buku: Subbab 10.4.11 dan Gambar 10.10.

Kosakata Lainnya

Terhubung kuat (berarah): setiap pasangan terurut terhubung ke dua arah. Graf berarah Subbab 10.8 punya komponen $\{v, x, y\}$, $\{u\}$, $\{w\}$, $\{z\}$.

Subgraf: himpunan bagian simpul beserta himpunan bagian sisi di antaranya; **merentang** jika mempertahankan setiap simpul. Kedua penelusuran menghasilkan subgraf merentang.

Graf lengkap K_n: setiap pasangan bertetangga, dengan cacah sisi pada Persamaan (10.3). **Bipartit**: dua kubu, setiap sisi menyeberang — tepat ketika **tak ada siklus ganjil**.

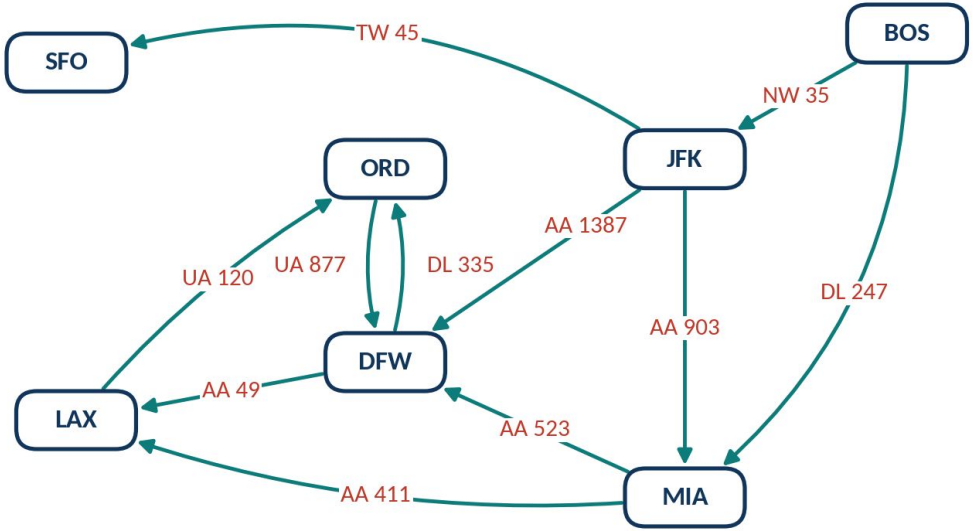
$$|E| = \binom{n}{2} = \frac{n(n-1)}{2}. \tag{10.3}$$

Berbobot: $w : E \to R$. **Jarang** ($|E|$ dekat $|V|$) lawan **padat** ($|E|$ dekat $|V|^2$): pada 10^6 simpul, matriks berisi 10^{12} entri. BFS meminimumkan **cacah sisi**, bukan bobot.

Buku: Subbab 10.4.9–10.4.16; seluruh kosakatanya terangkum dalam satu tabel di Subbab 10.4.16.

Menyimpan Graf: Contoh Berjalannya

Setiap representasi dinilai dengan empat pertanyaan — **P1** apakah (u, v) sebuah sisi? **P2** berapa $\deg(v)$? **P3** daftarkan $Adj[v]$? **P4** daftarkan semua sisi? — dan dengan ruangnya. Gambar 10.11 adalah contoh berarahnya.

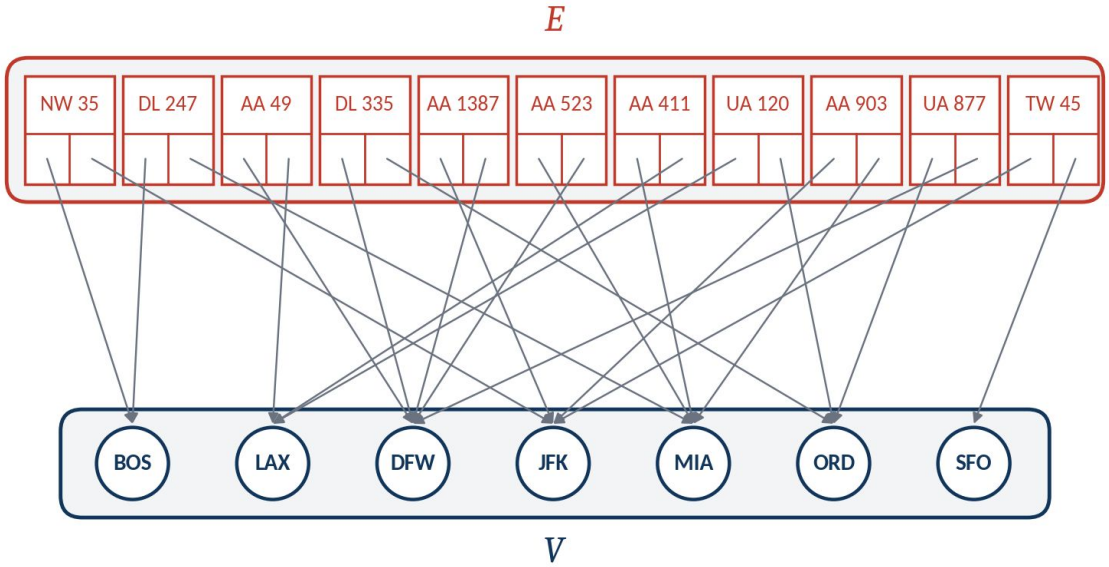


Gambar 10.11 Contoh berjalan untuk penyimpanan graf: graf berarah berisi penerbangan antara tujuh bandara (BOS, JFK, ORD, SFO, LAX, DFW, MIA), dengan tiap busur diberi label nomor penerbangan. Ada sebelas busur. Karena busurnya berarah, penerbangan AA 49 dari DFW ke LAX adalah sisi yang berbeda dari penerbangan DL 335 dari DFW ke ORD, dan tak satu pun di antaranya menyiratkan adanya penerbangan ke arah sebaliknya.

Buku: Subbab 10.5.1 dan Gambar 10.11.

Daftar Sisi

Gambar 10.12: sebuah wadah simpul dan sebuah wadah sisi, tiap sisi menunjuk ke ujung-ujungnya. Ruang $\Theta(V+E)$; P4 dalam $\Theta(E)$ — tetapi P1, P2, dan P3 semuanya memindai seluruh wadah.

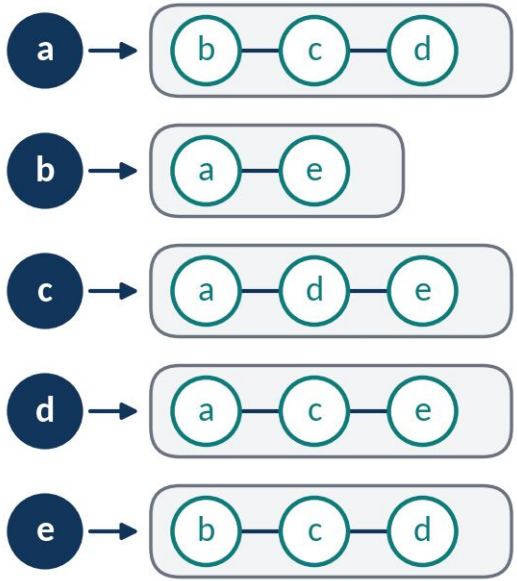


Gambar 10.12 Daftar sisi, dibangun pada graf penerbangan Gambar 10.11. Wadah bawah memuat ketujuh simpulnya; wadah atas memuat kesebelas sisinya, dan setiap objek sisi menunjuk ke bawah pada kedua ujungnya. Untuk menemukan penerbangan yang meninggalkan DFW, Anda harus memeriksa setiap satu dari sebelas objek sisi itu, sebab tidak ada apa pun di dalam strukturnya yang mengelompokkannya menurut bandara.

Buku: Subbab 10.5.2 dan Gambar 10.12. Daftar sisi persis yang diperlukan algoritma Kruskal (Bab 11).

Daftar Ketetanggaan

Gambar 10.13: satu daftar per simpul, $2|E|=14$ entri untuk graf tak berarah. Ruang $\Theta(V+E)$; P3 dalam $\Theta(\deg v)$, optimal. **Pilihan bawaan.**

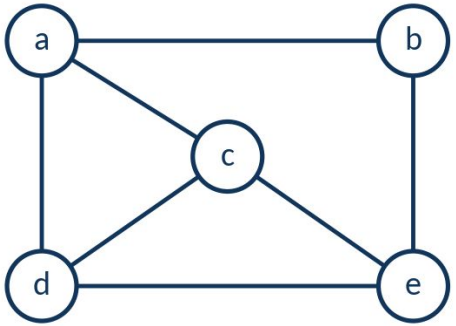


Gambar 10.13 Daftar ketetanggaan graf bab ini. Lima daftar, satu per simpul. Daftar simpul a memuat b, c, d; daftar simpul b memuat a, e; dan seterusnya. Daftar-daftar itu bersama-sama memuat 14 entri, yaitu $2|E|$ — lema jabat tangan, dibuat kasatmata.

Buku: Subbab 10.5.3 dan Gambar 10.13.

Matriks Ketetanggaan

Gambar 10.14: $M[i][j]$ bernilai benar tepat bila (i, j) sebuah sisi. Simetris untuk graf tak berarah, jumlah baris = derajat. P1 dalam $\Theta(1)$ — tetapi ruangnya $\Theta(V^2)$ dan P3 memakan $\Theta(V)$ bahkan untuk simpul terpencil.



	a	b	c	d	e
a	F	T	T	T	F
b	T	F	F	F	T
c	T	F	F	T	T
d	T	F	T	F	T
e	F	T	T	T	F

Gambar 10.14 Matriks ketetanggaan graf bab ini, digambar berdampingan dengan grafnya sendiri. Baris c terbaca T, F, F, T, T: simpul c bertetangga dengan a, d, dan e tetapi tidak dengan b, dan entri diagonalnya F sebab tidak ada gelung-diri. Matriksnya setangkup terhadap diagonalnya sebab grafnya tak berarah, dan ia punya 25 entri untuk graf yang hanya punya 7 sisi.

Buku: Subbab 10.5.4 dan Gambar 10.14.

Ketiganya Berdampingan

Tabel 10.1 Biaya setiap operasi dasar pada masing-masing dari ketiga representasi. Tidak ada representasi yang menang di setiap baris, dan itulah sebabnya ketiganya bertahan hidup. Baris terakhirlah yang menentukan isi bab ini: penelusuran atas daftar ketetanggaan bersifat linear; penelusuran yang sama atas matriks bersifat kuadratik.

Pertanyaan	Daftar sisi	Daftar ketetanggaan	Matriks ketetanggaan
P1: apakah (u, v) sebuah sisi?	$\Theta(E)$	$O(\deg u)$, terburuk $O(V)$	$\Theta(1)$
P2: berapa $\deg(v)$?	$\Theta(E)$	$\Theta(1)$	$\Theta(V)$
P3: daftarkan $Adj[v]$	$\Theta(E)$	$\Theta(\deg v)$	$\Theta(V)$
P4: daftarkan semua sisi	$\Theta(E)$	$\Theta(V+E)$	$\Theta(V^2)$
Tambahkan sebuah sisi	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Hapus sisi (u, v)	$\Theta(E)$	$O(\deg u + \deg v)$	$\Theta(1)$
Ruang	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(V^2)$
Biaya satu BFS atau DFS penuh	—	$\Theta(V+E)$	$\Theta(V^2)$

Tabel 10.1: tak ada representasi yang menang di setiap baris. **Penelusuran, Prim, Dijkstra → daftar ketetanggaan. Graf padat, uji sisi berwaktu tetap, algoritma semua-pasangan → matriks. Kruskal, membaca dari berkas → daftar sisi.**

Buku: Subbab 10.5.5–10.5.8; program yang membangun ketiganya ada di Subbab 10.5.8.

Menelusuri Graf, dan Penelusuran Melebar

TIGA WARNA. Putih: belum ditemukan. **Kelabu**: sudah ditemukan, sisinya belum semua diperiksa — **perbatasan** penelusuran. **Hitam**: selesai. Warna hanya bergerak putih → kelabu → hitam, sehingga tiap simpul masuk ke wadah paling banyak sekali.

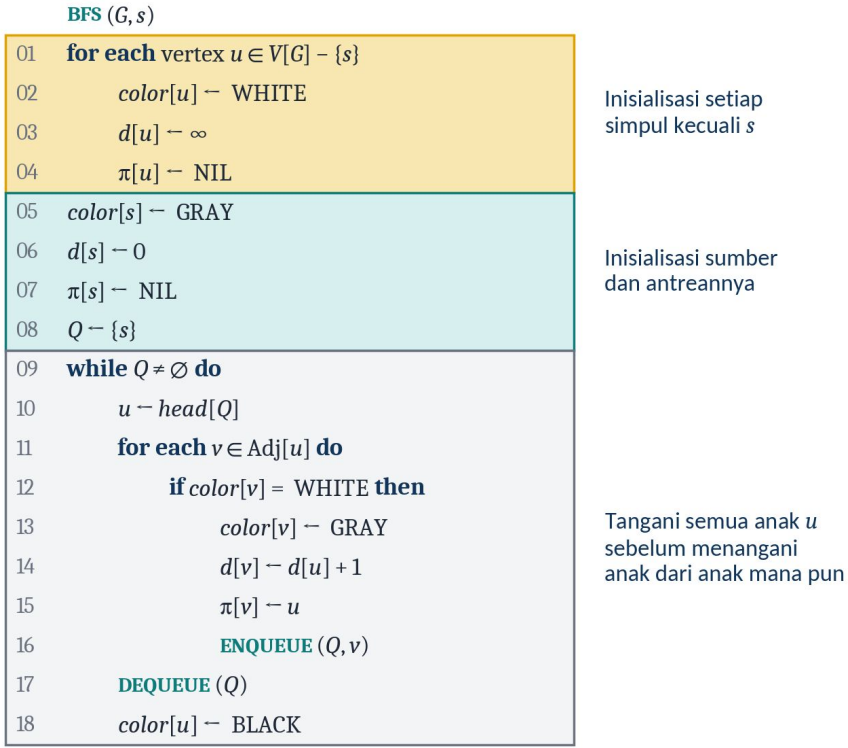
Jelajahi tempat yang **paling awal** diketahui → sebuah **antrean** → **penelusuran melebar**. Yang **paling akhir** diketahui → sebuah **tumpukan** → **penelusuran mendalam**.

BFS adalah labirin dengan benang yang diulur **satu sisi setiap kali**: ia menghitung `color[v]`, jarak `d[v]`, dan pendahulu `π[v]`. **Sentuhan pertama menang**: jarak sebuah simpul ditetapkan saat ia ditemukan.

Buku: Subbab 10.6 dan Subbab 10.7.1–10.7.2.

Algoritma BFS, Berkode Warna

Gambar 10.15: blok amber menginisialisasi, blok toska menyiapkan sumber dan antreannya, blok abu-abu adalah penelusurannya.



Gambar 10.15 Penelusuran melebar. Blok amber (baris 01–04) menginisialisasi setiap simpul selain sumbernya; blok toska (05–08) menginisialisasi sumbernya dan antreannya; blok abu-abu (09–18) adalah penelusurannya sendiri, dan catatan di sebelah kanan menyatakan apa yang ia capai — *tangani semua anak u sebelum menangani anak mana pun dari anak-anaknya*, dan itulah definisi melebar.

Buku: Subbab 10.7.3 dan Gambar 10.15.

BFS dalam Pseudokode

Pseudokode 10.1 Penelusuran melebar dari sumber s, dalam notasi buku ini, dengan nomor baris yang sama seperti Gambar 10.15.

```
BFS(G, s)
 1  for each vertex u in G.V - {s}
 2      color[u] = WHITE
 3      d[u] = INFINITY
 4      pi[u] = NIL
 5  color[s] = GRAY
 6  d[s] = 0
 7  pi[s] = NIL                 // s, bukan u: keseleo yang lazim
 8  Q = {s}
 9  while Q is not empty
10      u = head[Q]
11      for each v in Adj[u]
12          if color[v] == WHITE
13              color[v] = GRAY
14              d[v] = d[u] + 1
15              pi[v] = u       // v, bukan u: keseleo yang lazim
16              ENQUEUE(Q, v)
17      DEQUEUE(Q)
18      color[u] = BLACK
```

DUA KESELEO YANG PERLU DIHINDARI. Pada Pseudokode 10.1, baris 07 mengisi $\pi[s]$ dan baris 15 mengisi $\pi[v] \leftarrow u$. Menulis $\pi[u] \leftarrow u$ tetap mengunjungi setiap simpul dan menghitung setiap jarak — lalu mengembalikan lintasan yang ngawur. Inilah kutu nyata yang paling sering pada kode BFS.

Buku: Subbab 10.7.3.

Contoh Terkerjakan 1: Grafnya

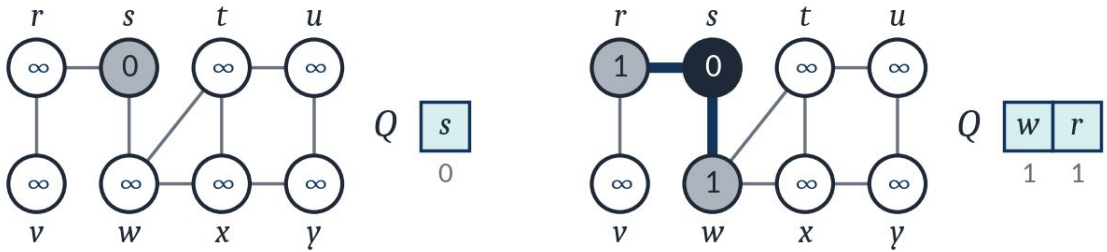
$V = \{r, s, t, u, v, w, x, y\}$, $E = \{rs, rv, sw, tu, tw, tx, uy, wx, xy\}$ — delapan simpul, sembilan sisi, jumlah derajat $18 = 2 \times 9$. Sumbernya s.

Urutan ketetanggaan (menentukan pohonnya, bukan jaraknya): $Adj[s] = [w, r]$, $Adj[r] = [s, v]$, $Adj[w] = [s, t, x]$, $Adj[t] = [u, w, x]$, $Adj[x] = [t, w, y]$, $Adj[v] = [r]$, $Adj[u] = [t, y]$, $Adj[y] = [u, x]$.

Lima halaman berikut memperlihatkan keadaannya: Gambar 10.16, Gambar 10.17, Gambar 10.18, Gambar 10.19, dan Gambar 10.20. Lingkaran putih = putih, kelabu = kelabu, terisi = hitam; angkanya d; sisi tebal adalah sisi pohon.

Buku: Subbab 10.7.4.

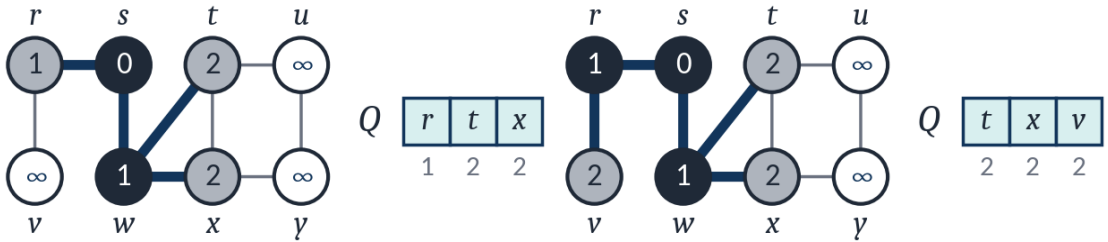
Inisialisasi dan Langkah 1



Gambar 10.16 Contoh BFS, dua keadaan yang pertama. **Kiri:** sesudah inisialisasi. Hanya s yang kelabu dan hanya s yang punya angka, 0; setiap simpul lain menampilkan ∞. Antreannya memuat s. **Kanan:** sesudah langkah 1. Kedua tetangga s kelabu dengan jarak 1, kedua sisi pohon s–w dan s–r digambar tebal, s hitam, dan antreannya memuat w lalu r — dalam urutan itu, yang menetapkan urutan $A\,d\,j[s]$ sebagai $[w, r]$.

Buku: Subbab 10.7.4 dan Gambar 10.16.

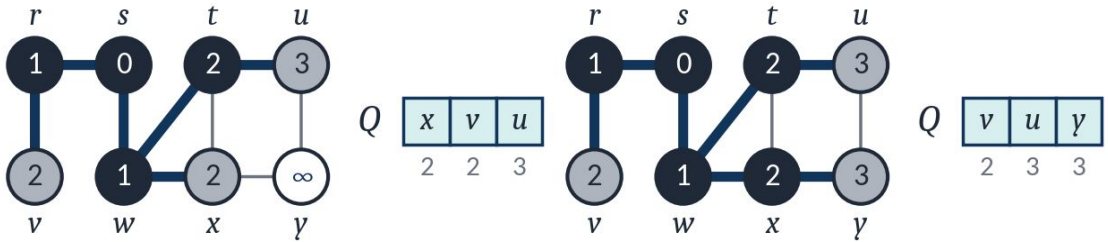
Langkah 2 dan 3



Gambar 10.17 Langkah 2 dan 3. **Kiri:** w telah dikeluarkan dari antrean dan dihitamkan, menemukan t dan x pada jarak 2; antreannya r, t, x. **Kanan:** r telah dikeluarkan dan dihitamkan, menemukan v pada jarak 2; antreannya t, x, v — tiga simpul yang semuanya berjarak 2, yaitu cincin kedua yang lengkap.

Buku: Subbab 10.7.4 dan Gambar 10.17. Sesudah langkah 3 antreannya memuat seluruh cincin berjarak 2.

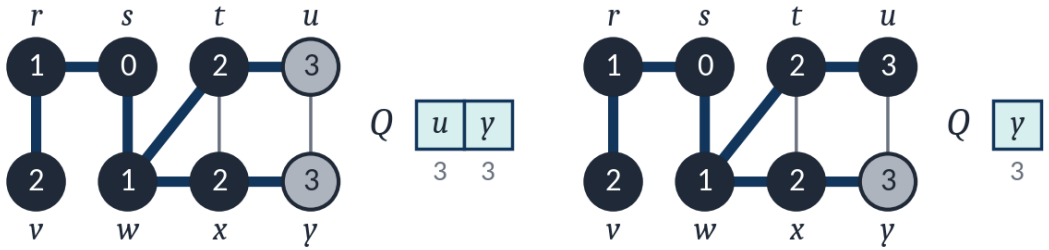
Langkah 4 dan 5



Gambar 10.18 Langkah 4 dan 5. **Kiri:** t dikeluarkan, menemukan u pada jarak 3; sisi (t, x) diperiksa lalu ditolak, sebab x sudah kelabu dengan $d[x]=2$. **Kanan:** x dikeluarkan, menemukan y pada jarak 3; antreannya v, u, y.

Buku: Subbab 10.7.4 dan Gambar 10.18. Pada langkah 4, x kelabu: sentuhan pertama menang.

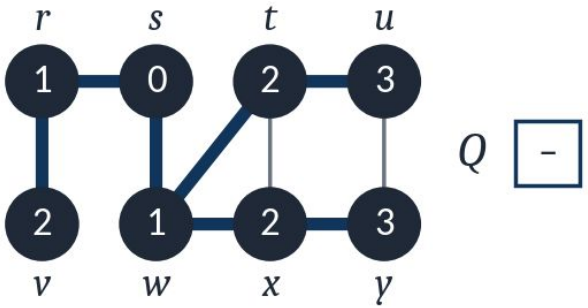
Langkah 6 dan 7



Gambar 10.19 Langkah 6 dan 7. **Kiri:** v dikeluarkan; satu-satunya tetangganya, r, sudah hitam, jadi tidak ada yang ditemukan dan v sekadar berubah hitam. **Kanan:** u dikeluarkan; kedua tetangganya sudah terlihat. Antreannya tinggal y seorang.

Buku: Subbab 10.7.4 dan Gambar 10.19.

Hasilnya



Gambar 10.20 Hasilnya. Setiap simpul hitam, antreannya kosong, tiap simpul membawa jaraknya dari s, dan ketujuh sisi tebalnya adalah pohon melebarnya. Ada delapan simpul dan tujuh sisi pohon, persis sebagaimana dituntut hitungan Gambar 10.10 atas sebuah pohon.

Buku: Subbab 10.7.4 dan Gambar 10.20.

Membaca Jawabannya

Tabel 10.2 BFS lengkap §10.7.4, satu baris per putaran gelung **while**. Setiap simpul dimasukkan ke antrean tepat sekali dan dikeluarkan tepat sekali, dan itulah sebabnya ada delapan langkah untuk delapan simpul. Perhatikan bahwa jarak di dalam antreannya tidak pernah menurun dari kiri ke kanan dan tidak pernah merentang lebih dari satu nilai — sifat yang dibuktikan di Lema 10.3.

Langkah	Dikeluarkan	Adj Yang diperiksa	Ditemukan, beserta d	Dilewati, dan sebabnya	Antrean sesudah langkah itu
—	—	—	$s{:}\,d{=}0$	—	$(s{:}0)$
1	s	w, r	$w{:}1, r{:}1$	—	$(w{:}1, r{:}1)$
2	w	s, t, x	$t{:}2, x{:}2$	s hitam	$(r{:}1, t{:}2, x{:}2)$
3	r	s, v	$v{:}2$	s hitam	$(t{:}2, x{:}2, v{:}2)$
4	t	u, w, x	$u{:}3$	w hitam, x kelabu	$(x{:}2, v{:}2, u{:}3)$
5	x	t, w, y	$y{:}3$	t, w hitam	$(v{:}2, u{:}3, y{:}3)$
6	v	r	—	r hitam	$(u{:}3, y{:}3)$
7	u	t, y	—	t hitam, y kelabu	$(y{:}3)$
8	y	u, x	—	keduanya hitam	$()$

Tabel 10.3 Keluaran BFS dari s: larik jarak d, larik pendahulu π, dan lintasan terpendek ke tiap simpul yang dibaca dari π dengan menyusuri pendahulunya kembali ke s lalu membaliknya. Panjang tiap lintasan sama dengan nilai d simpulnya, sebagaimana dituntut Teorema 10.4.

v	r	s	t	u	v	w	x	y
$d[v]$	1	0	2	3	2	1	2	3
$\pi[v]$	s	NIL	w	t	r	s	w	x
lintasan terpendek dari s	$s\,r$	s	$s\,w\,t$	$s\,w\,t\,u$	$s\,r\,v$	$s\,w$	$s\,w\,x$	$s\,w\,x\,y$

Tabel 10.2 adalah penelusurannya; Tabel 10.3 jawabannya. Lintasan ke y: $\pi[y]=x$, $\pi[x]=w$, $\pi[w]=s$; dibalik, $s\,w\,x\,y$, panjangnya $3=d[y]$.

Buku: Subbab 10.7.5.

Programnya

Kode 10.1 mencetak antreannya sesudah setiap langkah; barisan antreannya — (w, r), (r, t, x), (t, x, v), (x, v, u), (v, u, y), (u, y), (y), kosong — cocok dengan tabel dan gambarnya.

Kode 10.1 Penelusuran melebar dalam C++, yang mencetak antreannya pada setiap langkah. Daftar ketetanggaannya dituliskan secara eksplisit alih-alih dibangun dengan `addEdge`, supaya *urutannya* persis sama dengan Gambar 10.16 sampai Gambar 10.19 dan penelusuran yang tercetak dapat dibandingkan dengan Tabel 10.2 baris demi baris.

```cpp
// Penelusuran melebar pada graf Bab 10, dengan antreannya dicetak
// pada setiap langkah, persis seperti yang ditunjukkan gambar Subbab 10.7.4.
//   V = {r,s,t,u,v,w,x,y}
//   E = {rs, rv, sw, tu, tw, tx, uy, wx, xy}
#include <iostream>
#include <vector>
#include <deque>
#include <string>
#include <algorithm>
#include <climits>
using namespace std;

enum Color { WHITE, GRAY, BLACK };
static const char* CNAME[] = {"white", "gray ", "black"};
static const int INF = INT_MAX;

struct Graph {
    int n;
    vector<vector<int>> adj;
    explicit Graph(int n_) : n(n_), adj(n_) {}
    void addEdge(int u, int v) { adj[u].push_back(v); adj[v].push_back(u); }
};

static const string NAME = "rstuvwxy";
static string qToString(const deque<int>& Q, const vector<int>& d) {
    if (Q.empty()) return "( empty )";
    string s = "(";
    for (size_t i = 0; i < Q.size(); ++i) {
```

```
        if (i) s += " ";
        s += NAME[Q[i]];
        s += ":";
        s += to_string(d[Q[i]]);
    }
    return s + ")";
}

// Mengembalikan d[] dan pi[]; mencetak antreannya sesudah tiap masuk dan keluar antrean.
void bfs(const Graph& G, int s, vector<int>& d, vector<int>& pi, bool trace) {
    vector<Color> color(G.n, WHITE);
    d.assign(G.n, INF);
    pi.assign(G.n, -1);
```

Kode 10.1 (Lanjutan) *Penelusuran melebar dalam C++, yang mencetak antreannya pada setiap langkah. Daftar ketetanggaannya dituliskan secara eksplisit alih-alih dibangun dengan addEdge, supaya urutannya persis sama dengan Gambar 10.16 sampai Gambar 10.19 dan penelusuran yang tercetak dapat dibandingkan dengan Tabel 10.2 baris demi baris.*

```
    color[s] = GRAY; d[s] = 0; pi[s] = -1;   // sumbernya: kelabu, jarak 0
    deque<int> Q; Q.push_back(s);
    int step = 0;
    if (trace) cout << "  init      enqueue " << NAME[s] << "   Q = " << qToString(Q,d) << "\n";

    while (!Q.empty()) {
        int u = Q.front();                    // u = head[Q]
        Q.pop_front();                        // Dequeue(Q)
        ++step;
        if (trace) cout << "  step " << step << "    dequeue " << NAME[u]
                        << "   Q = " << qToString(Q,d) << "\n";
        for (int v : G.adj[u]) {              // pindai Adj[u] sekali, dan hanya sekali
            if (color[v] == WHITE) {
                color[v] = GRAY;
                d[v]  = d[u] + 1;             // satu sisi lebih jauh daripada u
                pi[v] = u;
                Q.push_back(v);
                if (trace) cout << "            discover " << NAME[v]
                                << " via " << NAME[u] << ", d=" << d[v]
                                << "  Q = " << qToString(Q,d) << "\n";
            } else if (trace) {
                cout << "            skip " << NAME[v] << " (already "
                     << CNAME[color[v]] << ")\n";
            }
        }
        color[u] = BLACK;                     // u sudah selesai
        if (trace) cout << "           " << NAME[u] << " -> black\n";
    }
```

```
}

vector<int> pathTo(int v, const vector<int>& pi) {
    vector<int> p;
    for (int x = v; x != -1; x = pi[x]) p.push_back(x);
    reverse(p.begin(), p.end());
    return p;
}
```

Kode 10.1 (Lanjutan) *Penelusuran melebar dalam C++, yang mencetak antreannya pada setiap langkah. Daftar ketetanggaannya dituliskan secara eksplisit alih-alih dibangun dengan addEdge, supaya urutannya persis sama dengan Gambar 10.16 sampai Gambar 10.19 dan penelusuran yang tercetak dapat dibandingkan dengan Tabel 10.2 baris demi baris.*

```
int main() {
    Graph G(8);
    // r=0 s=1 t=2 u=3 v=4 w=5 x=6 y=7 ; urutan ketetanggaan yang dipakai gambarnya
    G.adj[1] = {5,0};        // s: w lalu r  -- urutan yang ditunjukkan antrean pada gambar
    G.adj[0] = {1,4};        // r: s, v
    G.adj[5] = {1,2,6};      // w: s, t, x
    G.adj[2] = {3,5,6};      // t: u, w, x
    G.adj[6] = {2,5,7};      // x: t, w, y
    G.adj[4] = {0};          // v: r
    G.adj[3] = {2,7};        // u: t, y
    G.adj[7] = {3,6};        // y: u, x

    vector<int> d, pi;
    cout << "BFS from s\n";
    bfs(G, 1, d, pi, true);

    cout << "\nresult\n  vertex :";
    for (int i = 0; i < 8; ++i) cout << "  " << NAME[i];
    cout << "\n  d      :";
    for (int i = 0; i < 8; ++i) cout << "  " << d[i];
    cout << "\n  pi     :";
    for (int i = 0; i < 8; ++i) cout << "  " << (pi[i] < 0 ? '-' : NAME[pi[i]]);
    cout << "\n";

    cout << "\nbreadth-first tree edges (pi[v], v)\n  ";
    for (int v = 0; v < 8; ++v)
        if (pi[v] >= 0) cout << "(" << NAME[pi[v]] << "," << NAME[v] << ") ";
    cout << "\n";
```

```
    cout << "\nshortest paths read out of pi\n";
    for (int v = 0; v < 8; ++v) {
        cout << "  s to " << NAME[v] << " (" << d[v] << (d[v]==1 ? " edge" : " edges") << "): ";
        for (int x : pathTo(v, pi)) cout << NAME[x];
        cout << "\n";
    }
    return 0;
}
```

Buku: Subbab 10.7.6, tempat keluaran lengkapnya dicetak.

Ongkos, dan Pohon Melebar

Tiap simpul masuk antrean paling banyak sekali (diuji putih, lalu dicat kelabu); tiap daftar dipindai sekali, saat simpulnya dikeluarkan; panjang seluruh daftar $\Theta(E)$ menurut Persamaan (10.1). Maka Persamaan (10.4) — linear terhadap masukannya, jadi optimal.

$$T(n) = \underbrace{O(V)}_{\text{inisialisasi}} + \underbrace{O(V)}_{\text{operasi antrean}} + \underbrace{O(E)}_{\text{memindai daftarnya}} = O(V+E). \tag{10.4}$$

$O(V+E)$ **ADALAH KLAIM TENTANG DAFTAR KETETANGGAAN.** Pada matriks, baris 11 memakan $\Theta(V)$ per simpul dan BFS menjadi $\Theta(V^2)$ — sekitar 136 kali lebih lambat pada 8 000 simpul.

Subgraf pendahulu Persamaan (10.5) adalah **pohon melebar**: pohon merentang atas komponen s, berisi $|V_\pi| - 1$ sisi, dan setiap lintasan pohonnya adalah lintasan terpendek. Di sini 7 sisi pohon; (t, x) dan (u, y) ditolak.

$$V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}, E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}. \tag{10.5}$$

Buku: Subbab 10.7.7 dan 10.7.8.

Mengapa BFS Menemukan Lintasan Terpendek (1)

Lema 10.1 (tanpa algoritma): untuk setiap sisi (u, v), berlaku Persamaan (10.6). Satu langkah lagi tak dapat membawa Anda dua langkah lebih jauh dari rumah.

$$\delta(s, v) \leq \delta(s, u) + 1. \tag{10.6}$$

Lema 10.2: BFS tak pernah menaksir terlalu rendah, $d[v] \geq \delta(s, v)$ — induksi atas pemasukan ke antrean, memakai Lema 10.1.

Lema 10.3: antrean $\langle v_1, \ldots, v_r \rangle$ memenuhi Persamaan (10.7) — terurut menurut jarak, paling banyak dua nilai berbeda. Jadi simpul dikeluarkan dalam urutan d yang tak menurun.

$$d[v_1] \leq d[v_2] \leq \cdots \leq d[v_r] \leq d[v_1] + 1. \tag{10.7}$$

Buku: Subbab 10.7.9, Lema 10.1–10.3.

Mengapa BFS Menemukan Lintasan Terpendek (2)

Teorema 10.4. Andaikan ada d yang salah; ambil v dengan $\delta(s,v)$ terkecil di antara yang salah. Menurut Lema 10.2, Persamaan (10.8). Misalkan u mendahului v pada lintasan terpendek; u benar, sehingga Persamaan (10.9).

$$d[v] > \delta(s,v). \tag{10.8}$$

$$d[v] > \delta(s,v) = \delta(s,u) + 1 = d[u] + 1. \tag{10.9}$$

Saat u dikeluarkan, v putih (maka $d[v] = d[u]+1$), hitam ($d[v] \le d[u]$), atau kelabu ($d[v] \le d[u]+1$) — setiap kasus bertentangan dengan Persamaan (10.9). ∎

Akibat 10.5: setiap sisi menghubungkan tingkat yang selisihnya paling banyak satu — fakta yang menjadi dasar uji kebipartitan.

Buku: Subbab 10.7.9, Teorema 10.4 dan Akibat 10.5.

Penelusuran Mendalam

Labirin dengan benang **dan sekaleng cat**: berjalanlah sejauh mungkin, lalu mundur ke persimpangan terakhir yang masih punya lorong belum dijelajahi. Benangnya adalah **jalan pulang** — sebuah tumpukan.

> **WAWASAN KUNCI.** Simpul kelabu membentuk **rantai linear**: lintasan dari akar ke tempat Anda berdiri. “Apakah v kelabu?” berarti “apakah v leluhur tempat ini?” — itulah sebabnya simpul kelabu di depan berarti ada siklus.

DFS **tidak memerlukan sumber**: gelung luarnya memulai pohon baru pada setiap simpul putih, sehingga ia menjelajahi seluruh graf dan menghasilkan **hutan**. Gambar 10.21 memberikan kedua prosedurnya.

```
DFS (G)
1  for each vertex u ∈ V[G]
2       do color[u] ← WHITE
3  time ← 0
4  for each vertex u ∈ V[G]
5       do if color[u] = WHITE
6            then DFS-VISIT (u)
```

```
DFS-VISIT (u)
1  color[u] ← GRAY          ▷ Simpul putih u ditemukan.
2  d[u] ← time              ▷ Catat waktu penemuan.
3  time ← time + 1          ▷ Majukan jam global.
4  for each v ∈ Adj[u]      ▷ Jelajahi semua sisi (u, v).
5       do if color[v] = WHITE
6            then DFS-VISIT (v)
7  color[u] ← BLACK         ▷ Hitamkan u; ia selesai.
8  f[u] ← time              ▷ Catat waktu selesai.
9  time ← time + 1          ▷ Majukan jam global.
```

Gambar 10.21 Penelusuran mendalam. Prosedur luar DFS(G) memutihkan setiap simpul, menjalankan jamnya, lalu memanggil DFS-Visit pada setiap simpul yang masih putih; tiap panggilan semacam itu mengakarkan satu pohon baru dalam hutan mendalamnya. Prosedur dalam DFS-Visit(u) adalah penelusurannya sendiri: cat u kelabu, cap waktu penemuannya, rekursilah ke setiap tetangga yang putih, lalu cat u hitam dan cap waktu selesainya.

Buku: Subbab 10.8.1 dan 10.8.2, serta Gambar 10.21.

DFS dalam Pseudokode, dan Dua Cap Waktu

Pseudokode 10.2 Penelusuran mendalam: prosedur luarnya memulai pohon baru pada setiap simpul yang masih putih, dan DFS-Visit menjalankan penelusurannya sambil mencap waktu penemuan dan waktu selesai.

```
DFS(G)
 1  for each vertex u in G.V
 2      color[u] = WHITE
 3      pi[u] = NIL
 4  time = 0
 5  for each vertex u in G.V
 6      if color[u] == WHITE
 7          DFS-VISIT(u)

DFS-VISIT(u)
 1  color[u] = GRAY              // simpul putih u baru saja ditemukan
 2  time = time + 1
 3  d[u] = time                  // cap waktu penemuannya
 4  for each v in Adj[u]         // jelajahi setiap sisi (u, v)
 5      if color[v] == WHITE
 6          pi[v] = u
 7          DFS-VISIT(v)
 8  color[u] = BLACK             // u sudah selesai
 9  time = time + 1
10  f[u] = time                  // cap waktu selesainya
```

Pseudokode 10.2 menaikkan jamnya lebih dulu (seperti pada contoh terkerjakan). Tiap simpul mendapat **waktu penemuan** $d[u]$ dan **waktu selesai** $f[u]$, dengan $d[u] < f[u]$; jamnya berakhir tepat pada $2|V|$.

Buku: Subbab 10.8.2 dan 10.8.3.

Contoh Terkerjakan 2: Enam Belas Langkah

Busur $u \to v$, $u \to x$, $v \to y$, $w \to y$, $w \to z$, $x \to v$, $y \to x$, $z \to z$; gelung luar u, v, w, x, y, z.

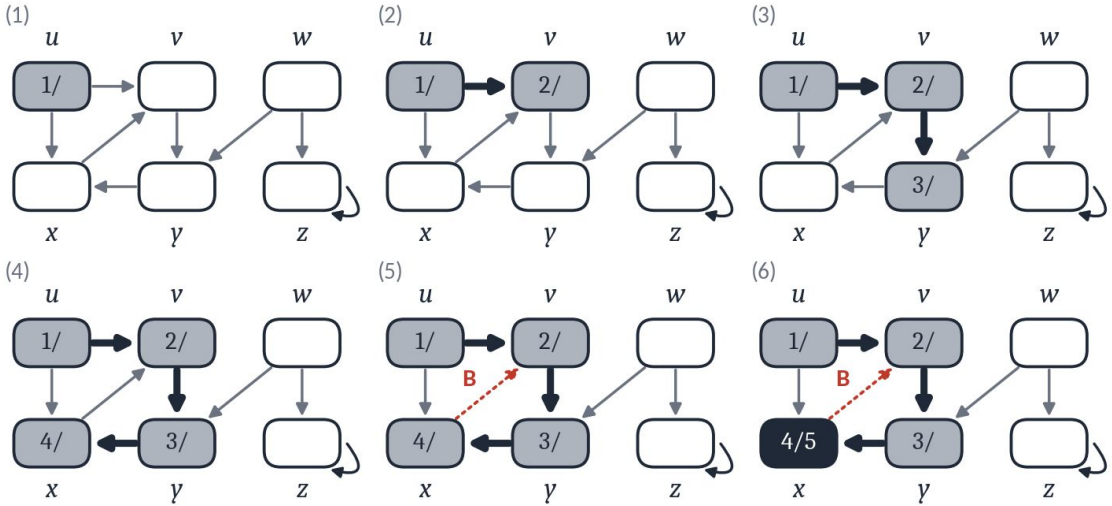
u $d=1$; pohon $u \to v$ ($d=2$); pohon $v \to y$ (3); pohon $y \to x$ (4); $x \to v$ menemukan v **kelabu**: sisi **balik**. x selesai pada 5, y pada 6, v pada 7.

$u \to x$: x hitam dan $d[u] < d[x]$ — **maju**. u selesai pada 8. Pohon baru di w ($d=9$): $w \to y$ hitam, $d[w] > d[y]$ — **silang**.

Pohon $w \to z$ (10); $z \to z$ menemukan z kelabu — **balik** (gelung-diri tak perlu kasus khusus). z selesai pada 11, w pada $12 = 2|V|$. Ketiga gambarnya adalah Gambar 10.22, Gambar 10.23, dan Gambar 10.24.

Buku: Subbab 10.8.4.

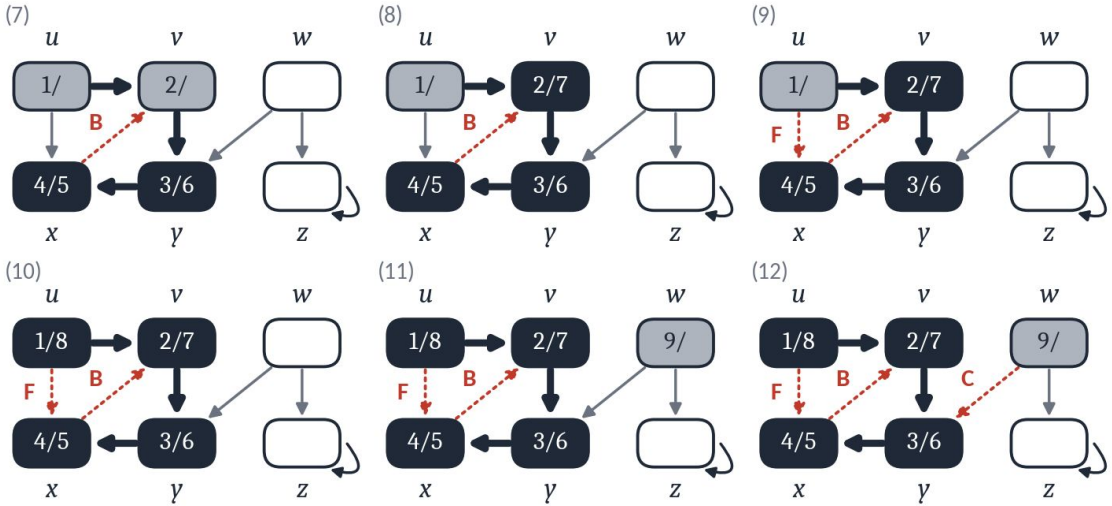
Langkah 1 sampai 6



Gambar 10.22 Langkah 1 sampai 6 contoh DFS, dalam urutan baca: u berubah kelabu dengan $d=1$; v ditemukan lewat sisi pohon dengan $d=2$; y dengan $d=3$; x dengan $d=4$; sisi (x, v) diperiksa, didapati menunjuk ke simpul kelabu, dan diberi label **B** untuk sisi balik, digambar sebagai panah putus-putus; dan akhirnya x berubah hitam dengan $f=5$, ditampilkan sebagai $4/5$.

Buku: Subbab 10.8.4 dan Gambar 10.22.

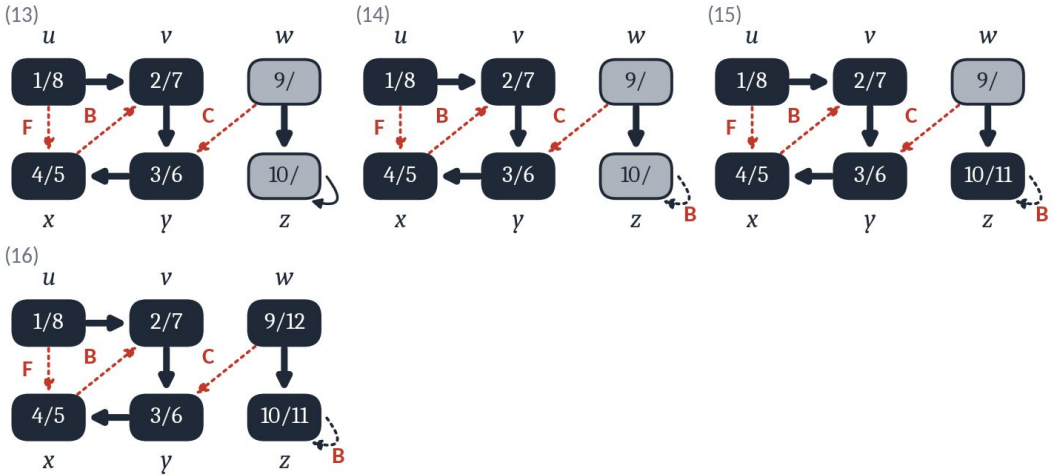
Langkah 7 sampai 12



Gambar 10.23 Langkah 7 sampai 12. Rekursinya terurai, menghitamkan y pada $3/6$ dan v pada $2/7$; kembali di daftar ketetanggaan u, sisi (u, x) diperiksa dan diberi label **F** untuk sisi maju; u selesai pada $1/8$, melengkapkan pohon pertamanya; gelung luarnya lalu mengakarkan pohon kedua di w dengan $d = 9$; dan sisi (w, y) diberi label **C** untuk sisi silang, sebab y milik pohon pertama.

Buku: Subbab 10.8.4 dan Gambar 10.23.

Langkah 13 sampai 16



Gambar 10.24 Langkah 13 sampai 16. z ditemukan dari w lewat sisi pohon dengan $d = 10$; gelung-diri (z, z) diperiksa dan diberi label **B**, sebab pada saat itu z kelabu; z selesai pada $10/11$; dan w selesai pada $9/12$. Setiap simpul kini hitam dan jamnya telah mencapai $12 = 2|V|$.

Buku: Subbab 10.8.4 dan Gambar 10.24.

Jawabannya

Tabel 10.4 Pencap waktu dan pendahulu sesudah DFS atas graf §10.8.4. Kedua akarnya adalah u dan w, dua simpul yang tidak punya pendahulu. Kedua belas capnya adalah bilangan bulat 1 sampai 12, masing-masing dipakai sekali.

v	u	v	w	x	y	z
$d[v]$	1	2	9	4	3	10
$f[v]$	8	7	12	5	6	11
$\pi[v]$	NIL	u	NIL	y	v	w

Tabel 10.5 Setiap busur contohnya, digolongkan, beserta uji warna yang membuat keputusannya. Kedelapan busurnya muncul: tiga sisi pohon di pohon pertama, satu di pohon kedua, dua sisi balik, satu maju, satu silang.

Busur	Golongan	Uji yang menentukannya
$u \to v$	pohon	v berwarna WHITE
$v \to y$	pohon	y berwarna WHITE
$y \to x$	pohon	x berwarna WHITE
$x \to v$	balik	v berwarna GRAY — seorang leluhur
$u \to x$	maju	x berwarna BLACK dan $d[u] < d[x]$ — seorang keturunan
$w \to y$	silang	y berwarna BLACK dan $d[w] > d[y]$ — bukan keduanya
$w \to z$	pohon	z berwarna WHITE
$z \to z$	balik	z berwarna GRAY — dirinya sendiri, dan kelabu

Tabel 10.4 dan Tabel 10.5: dua akar, empat sisi pohon $(6-2)$, dua balik, satu maju, satu silang.

Buku: Subbab 10.8.4.

Hutan, Tumpukan, Ongkos

Hutan mendalam Persamaan (10.10) merentang **seluruh** V: di sini pohon $u \to v \to y \to x$ dan $w \to z$.

$$E_\pi = \{(\pi[v], v) \in E : v \in V \text{ dan } \pi[v] \neq \text{NIL}\}. \tag{10.10}$$

Tanpa rekursi: tumpukan bingkai $(u, next)$ — posisi lanjutnya penting, sebab tanpanya kembali ke sebuah simpul akan memulai ulang daftarnya. Diperlukan pada graf yang dalam, tempat rekursi meluapkan tumpukan panggilan.

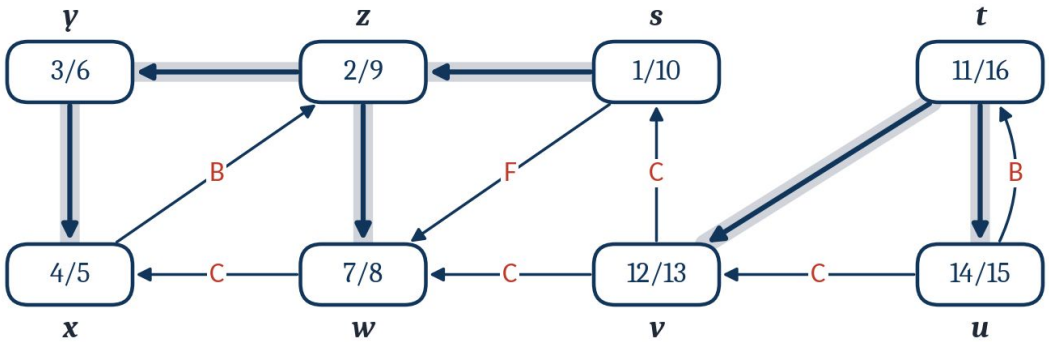
Ongkos Persamaan (10.11): dua gelung $\Theta(V)$, satu DFS-Visit per simpul, setiap daftar dijalani sekali — **analisis agregat**.

$$T(n) = \underbrace{\Theta(V)}_{\text{dua gelung luarnya}} + \underbrace{\Theta(V)}_{|V| \text{ panggilan, overhead tetap tiap panggilan}} + \underbrace{\Theta(E)}_{\text{menyusuri semua da}} \tag{10.11}$$
$$\Theta(V + E).$$

Buku: Subbab 10.8.5–10.8.7.

Contoh yang Lebih Besar, Bercap Waktu

Gambar 10.25: tiga belas busur pada delapan simpul, dua pohon, dan keempat golongan sisi.

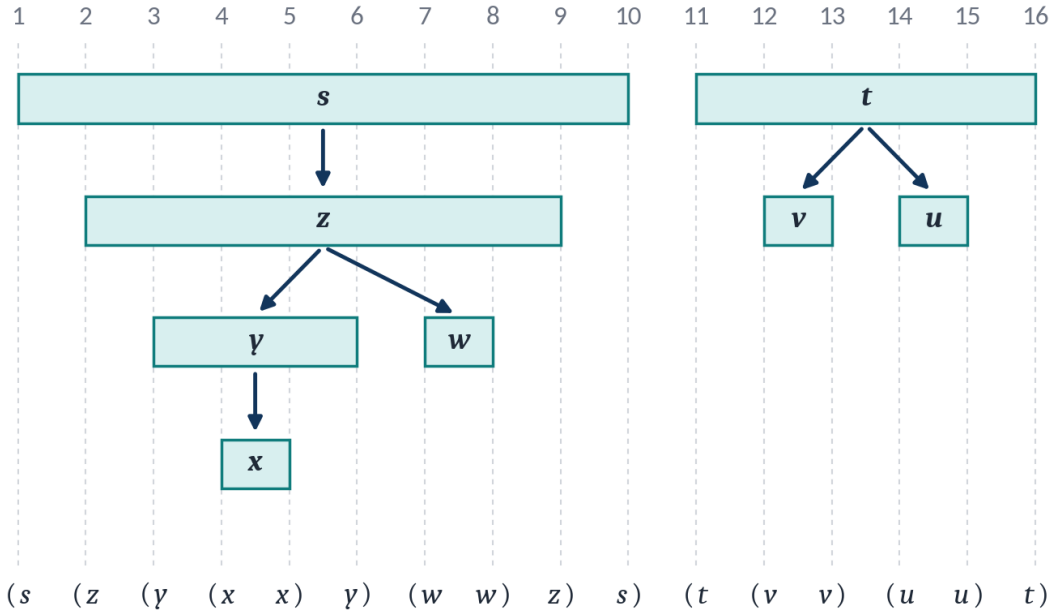


Gambar 10.25 Contoh DFS yang lebih besar, berpencap waktu. Tiap kotak simpul memuat d/f: s adalah $1/10$, z adalah $2/9$, y adalah $3/6$, x adalah $4/5$, w adalah $7/8$, t adalah $11/16$, v adalah $12/13$, u adalah $14/15$. Panah kelabu tebal adalah sisi pohon; panah tipisnya diberi label **B** (balik), **F** (maju), dan **C** (silang). Gambar tunggal ini membawa pencap waktunya, hutannya, dan penggolongan sisinya yang lengkap sekaligus.

Buku: Subbab 10.8.8 dan Gambar 10.25.

Struktur Tanda Kurung

Gambar 10.26: satu batang per simpul yang merentang $[d, f]$. Contoh pertama: $(u(v(y(x\,x)\,y)\,v)\,u)(w(z\,z)\,w)$.



Gambar 10.26 Struktur tanda kurung Gambar 10.25. Di atas, hutan mendalamnya digambar ulang dengan sisi pohon menunjuk ke bawah. Di bawah, jamnya dari 1 sampai 16 dengan satu batang mendatar per simpul, merentang $[d, f]$, dan penandakurungannya ditulis di sepanjang bagian bawah: $(s(z(y(xx)y)(ww)z)s)(t(vv)(uu)t)$. Setiap pasang batangnya entah bersarang atau saling lepas; tidak ada yang bertindihan sebagian.

Buku: Subbab 10.8.8 dan Gambar 10.26.

Teorema Tanda Kurung dan Lintasan Putih

TEOREMA 10.6 (TANDA KURUNG). Untuk sembarang u, v, selang $[d[u], f[u]]$ dan $[d[v], f[v]]$ entah **saling lepas** (tak satu pun keturunan yang lain) atau **bersarang** (yang di dalam adalah keturunan) — tak pernah bertindih sebagian.

Akibat 10.7: v keturunan sejati u tepat ketika Persamaan (10.12) — bentuk pohon menjadi perbandingan empat bilangan bulat.

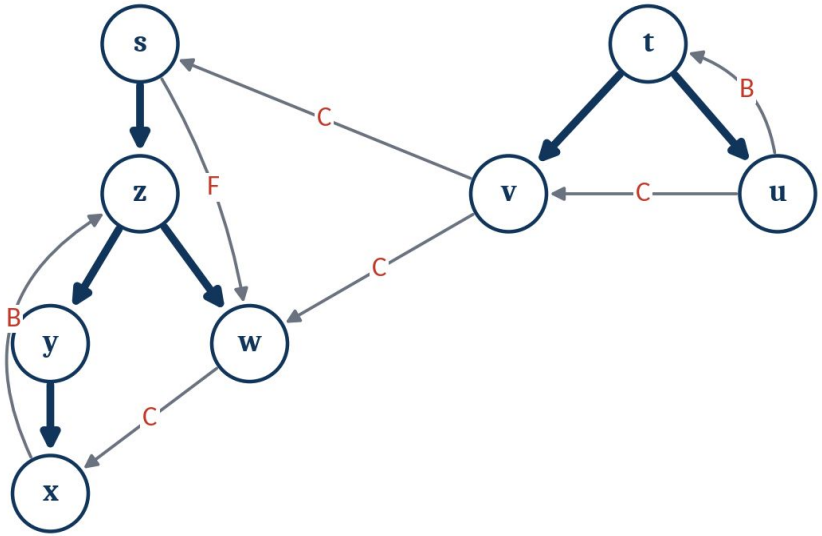
$$d[u] < d[v] < f[v] < f[u], \tag{10.12}$$

TEOREMA 10.8 (LINTASAN PUTIH). v menjadi keturunan u jika dan hanya jika, pada saat $d[u]$, ada lintasan simpul **putih** dari u ke v.

Buku: Subbab 10.8.8 dan 10.8.9, Teorema 10.6 dan 10.8.

Menggolongkan Sisi

Menurut warna v saat (u, v) diperiksa: **pohon** (putih), **balik** (kelabu), **maju** (hitam, $d[u] < d[v]$), **silang** (hitam, $d[u] > d[v]$). Gambar 10.27 membuatnya kasatmata.



Gambar 10.27 Graf Gambar 10.25 digambar ulang sebagai hutan mendalamnya, dengan sisi pohon tebal dan menunjuk ke bawah. Kini penggolongannya menjadi kasatmata: $x \to z$ dan $u \to t$ melengkung kembali **ke atas** menuju seorang leluhur (**B**); $s \to w$ menunjuk **ke bawah** melewati satu aras antara menuju seorang keturunan (**F**); $w \to x$, $v \to w$, $v \to s$, dan $u \to v$ berjalan **ke samping** di antara cabang atau di antara kedua pohonnya (**C**).

Buku: Subbab 10.8.10 dan Gambar 10.27.

Empat Golongan, dan Graf Tak Berarah

Tabel 10.6 Keempat golongan sisi: graf mana yang dapat memuatnya, uji warna yang mengenalinya, pola pencap waktu yang dihasilkannya, dan untuk apa masing-masingnya berguna. Pada graf tak berarah hanya dua lajur pertamanya yang mungkin, dan itulah Teorema 10.9.

	Pohon	Balik	Maju	Silang
Graf tak berarah	ya	ya	tak pernah	tak pernah
Graf berarah	ya	ya	ya	ya
Warna v saat (u, v) diperiksa	putih	kelabu	hitam	hitam
Uji pencap waktu	—	$d[v] < d[u] < f[u] < f[v$	$d[u] < d[v] < f[v] < f[u$	$d[v] < f[v] < d[u] < f[u$
Apa yang ia beri tahu Anda	hutannya	ada siklus	tidak banyak	tidak banyak

Tabel 10.6. **Teorema 10.9:** pada graf **tak berarah** setiap sisi adalah sisi pohon atau sisi balik — jadi sisi *bukan pohon* mana pun berarti ada siklus.

Buku: Subbab 10.8.10, Teorema 10.9.

Empat Hal yang Dapat Dikerjakan Penelusuran

Komponen: BFS dari tiap simpul yang belum berlabel; total $O(V+E)$, lalu "terhubungkah?" cukup satu perbandingan. Graf Gambar 10.9: tiga komponen berukuran 3, 4, 5.

Siklus: tak berarah — tetangga bukan-putih selain induknya (lewati induk **tepat sekali**); berarah — busur menuju simpul **kelabu**. Menguji `visited` alih-alih kelabu melaporkan sisi silang dan sisi maju sebagai siklus.

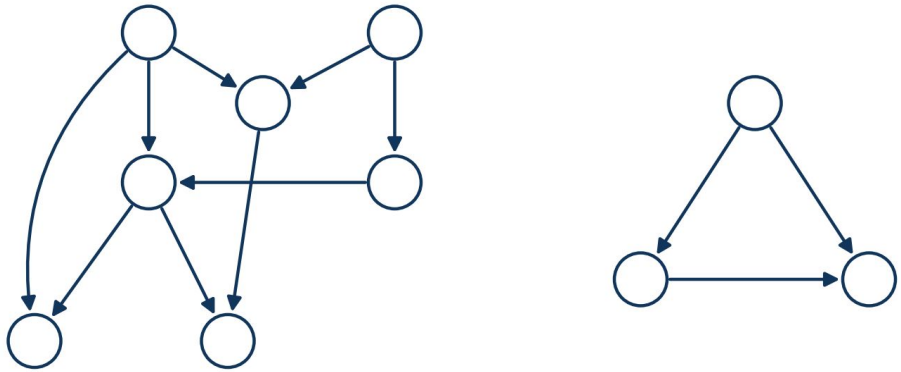
Kebipartitan: warnai menurut paritas jarak BFS; sisi di antara dua simpul setingkat memperlihatkan siklus ganjil.

Lintasan terpendek tak berbobot: BFS itu sendiri; berhenti lebih awal di simpul tujuan; banyak sumber dimulai bersama di antrean; jika berbobot, pakailah Dijkstra (Bab 13).

Buku: Subbab 10.9.1–10.9.4.

Graf Berarah Tak Bersiklus

DAG adalah graf berarah tanpa siklus berarah — model bagi urutan prasyarat. Gambar 10.28 memperlihatkan dua.



Gambar 10.28 Dua graf berarah tak bersiklus. Pada masing-masingnya, ikutilah panahnya dari mana saja dan Anda tidak akan pernah dapat kembali ke tempat Anda memulai: tidak ada siklus berarah. DAG adalah model alami bagi sehimpunan prasyarat — "ini sebelum itu" — dan himpunan prasyarat yang tak bersiklus persis merupakan himpunan yang dapat dipenuhi.

Buku: Subbab 10.9.5 dan Gambar 10.28.

Teorema DAG dan Pengurutan Topologis

TEOREMA 10.10. Graf berarah tak bersiklus **jika dan hanya jika** DFS tidak menghasilkan sisi balik. (Sisi balik + lintasan pohon = siklus; sebaliknya simpul pertama sebuah siklus yang ditemukan melihat lintasan putih yang mengelilinginya.)

Pseudokode 10.3 Pengurutan topologis dengan penelusuran mendalam: simpul-simpul dalam urutan waktu selesai yang menurun.

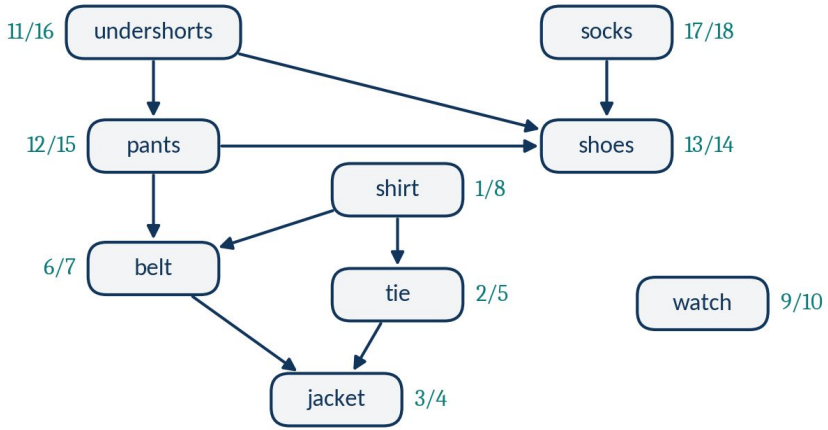
```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times f[v] for each vertex v
2  as each vertex is finished, insert it onto the FRONT of a linked list
3  return the linked list of vertices
```

Pseudokode 10.3 mendaftarkan simpul dalam **urutan waktu selesai yang menurun**. **Teorema 10.11:** pada DAG, $f[v] < f[u]$ untuk setiap busur (u, v) — v kelabu mustahil, v putih selesai di dalam panggilan u, v hitam sudah selesai. Ongkos $O(V+E)$, tanpa pengurutan.

Buku: Subbab 10.9.5 dan 10.9.6, Teorema 10.10 dan 10.11.

Berpakaian, Bercap Waktu

Gambar 10.29: panah $x \to y$ berarti x harus dikenakan sebelum y.

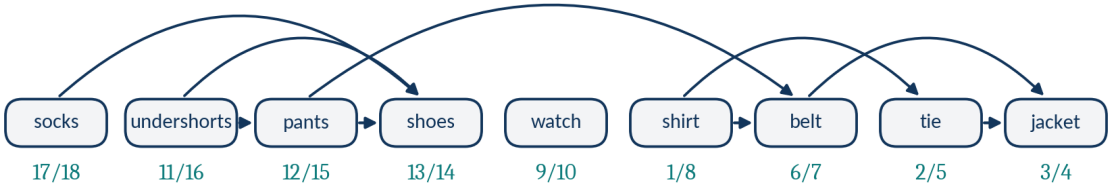


Gambar 10.29 DAG berpakaian, diberi pencap waktu oleh sebuah DFS. Sembilan busana; panah dari x ke y berarti x harus dikenakan sebelum y. Capnya adalah shirt $1/8$, tie $2/5$, jacket $3/4$, belt $6/7$, watch $9/10$, undershorts $11/16$, pants $12/15$, shoes $13/14$, socks $17/18$ — delapan belas cap untuk sembilan simpul, sebagaimana dituntut $2|V|$. Si watch sama sekali tidak punya panah: ia boleh dikenakan kapan saja.

Buku: Subbab 10.9.6 dan Gambar 10.29.

Setiap Panah Menunjuk ke Kanan

Gambar 10.30: kaus kaki, celana dalam, celana, sepatu, jam tangan, kemeja, ikat pinggang, dasi, jas.



Gambar 10.30 Urutan topologisnya: kesembilan simpul yang sama ditata dari kiri ke kanan dalam urutan waktu selesai yang menurun — socks 18, undershorts 16, pants 15, shoes 14, watch 10, shirt 8, belt 7, tie 5, jacket 4 — dan kini **setiap panah menunjuk ke kanan**. Itulah yang disebut pengurutan topologis.

Buku: Subbab 10.9.6 dan Gambar 10.30. Bab 11 menambahkan algoritma Kahn.

BFS atau DFS?

Tabel 10.7 Melebar melawan mendalam, pada sebelas sumbu. Keduanya berongkos sama dan berbeda pada satu baris kode; segala sesuatu lain di dalam tabel ini mengikuti dari satu baris itu.

	Penelusuran melebar	Penelusuran mendalam
Wadah	antrean (FIFO)	tumpukan (LIFO), biasanya tumpukan panggilannya
Mengambil sumber?	ya, satu per panggilan	tidak — ia menelusuri seluruh grafnya
Menghasilkan	sebuah pohon, dari satu komponen	sebuah hutan, merentang seluruh V
Merekam	$d[v]$ dan $\pi[v]$	$d[u]$, $f[u]$, dan $\pi[u]$
Biaya	$O(V+E)$	$\Theta(V+E)$
Ruang tambahan	$O(V)$ untuk antreannya	$O(V)$ untuk tumpukannya — atau tumpukan panggilannya
Menemukan lintasan terpendek?	**ya**, ketika sisinya tak berbobot	tidak
Urutan kunjungan	menurut jarak yang menaik dari sumbernya	sedalam mungkin, lalu mundur
Sisi bukan pohon	hanya antara aras k dan k, $k+1$	balik, maju, dan silang
Wajar untuk	lompatan paling sedikit, benda terdekat, *flood fill*, kerja aras demi aras, kebipartitan	siklus, pengurutan topologis, komponen terhubung kuat, titik artikulasi, labirin, runut balik
Risiko dalam praktik	antrean besar pada graf yang lebar	luapan tumpukan pada graf yang dalam

Tabel 10.7: **jarak, lompatan, terdekat, tingkat → BFS. Urutan, siklus, ketergantungan → DFS.** Terukur: menggandakan grafnya menggandakan waktu daftar ketetanggaan dan melipatempatkan waktu matriks (0,778; 3,105; 12,263; 47,902 ms).

Buku: Subbab 10.9.7 dan 10.9.8.

Profitina dalam Praktik

Pemindaiannya adalah BFS atas graf tautan berarah: putih = terlihat di perbatasan, kelabu = sudah diambil, hitam = sudah dinilai. BFS mencapai halaman penting lebih dulu, dan $d[v]$ **adalah kedalaman klik**, sinyal GEO yang nyata.

Komponen menemukan gugus yatim; **deteksi siklus** menjaga DAG ketergantungan sinyal; **pengurutan topologis** mengurutkan sinyalnya; daftar ketetanggaan, karena graf situsnya jarang.

Profitina Laundry: BFS dari Keputih menempatkan enam zona dalam dua lompatan; Nginden Jangkungan berjarak empat lompatan; Dukuh Setro **tak terjangkau**, bukan jauh. Menambahkan *notify* → *invoice* membuat proses cucinya bersiklus, dan programnya menyebut siklusnya `pack -> notify -> invoice -> pack`.

Buku: Subbab 10.10 dan Subbab 10.10.1. Produk memberi saran; pengguna yang memutuskan.

Kesalahan yang Sering Terjadi

1. Tidak menandai simpul yang sudah terlihat — programnya tak pernah berhenti. **2. Menandai saat dikeluarkan** alih-alih saat dimasukkan. **3. BFS pada graf berbobot.**

4. `visited` **padahal diperlukan tiga warna** (siklus berarah). **5. Melupakan induk** pada deteksi siklus tak berarah. **6. Mengeluarkan dari antrean dengan** `vector::erase(begin())`**.**

7. Berekursi pada graf yang dalam. 8. Melanjutkan daftar ketetanggaan dari awal. 9. Hanya menambahkan satu arah dari sisi tak berarah. 10. Matriks $V \times V$ untuk graf jarang.

11. Mengira satu BFS meliputi seluruh graf. 12. Menganggap ∞ sebagai "jauh". 13. Mengharapkan pohon atau urutan yang tunggal. 14. Mencampuradukkan d pada BFS (jarak) dengan d pada DFS (waktu). 15. Mengklaim $O(V+E)$ di atas matriks.

Buku: Subbab 10.11.

Ringkasan: Lima Hal untuk Diingat

1. Graf adalah $G=(V,E)$; **lema jabat tangan** $\sum_{v} \deg(v)=2|E|$ adalah sumber setiap E dalam waktu jalannya.

2. Simpanlah sebagai **daftar ketetanggaan** kecuali ada alasan lain: ruang $\Theta(V+E)$, pendaftaran tetangga yang optimal.

3. BFS = antrean: $d[v]=\delta(s,v)$ dalam $O(V+E)$, dibuktikan oleh Lema 10.1–10.3 dan Teorema 10.4.

4. DFS = tumpukan: cap waktu, teorema tanda kurung dan lintasan putih, serta empat golongan sisi (hanya pohon dan balik bila tak berarah).

5. Komponen, siklus, kebipartitan, lompatan terpendek, teorema DAG, dan pengurutan topologis — semuanya $O(V+E)$, semuanya penelusuran ditambah beberapa baris.

Buku: Subbab 10.12.

Pertemuan Berikutnya

Bab 11 — Pohon Merentang Minimum, DAG, dan Pengurutan Topologis. Teorema DAG dibuktikan lagi, dan algoritma derajat-masuk Kahn di samping metode DFS.

Bobot pun hadir: pohon merentang termurah, **sifat potongan**, dan dua algoritma rakus — **Prim** (BFS dengan antrean prioritas) dan **Kruskal** (sapuan atas daftar sisi).

Sebelum pertemuan: kerjakan Soal Latihan 10.1, 10.6, 10.9, dan 10.10, lalu bacalah Subbab 11.1 sampai 11.4.

Buku: Subbab 10.12.1 dan Bab 11.

Bab 1

Kuliah 11 — Pohon Merentang Minimum, DAG, dan Pengurutan Topologis

Perancangan dan Analisis Algoritma (EF234405)

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*

Dosen: Irfan Subakti — yifana@gmail.com

Buku: Bab 11, Pohon Merentang Minimum, DAG, dan Pengurutan Topologis. Setiap halaman handout ini diakhiri dengan penunjuk ke buku, yang menjadi rujukan utama.

Pertemuan Hari Ini

1. Graf berarah asiklik dan teorema DAG, dibuktikan ke dua arah (Subbab 11.2).

2. Pengurutan topologis dengan dua cara — menurut waktu selesai DFS dan menurut derajat masuk Kahn — ditelusuri pada DAG berpakaian (Subbab 11.3).

3. Pohon merentang, persoalan pohon merentang minimum, substruktur optimal, dan **sifat potongan** (Subbab 11.4–11.8).

4. **Prim–Jarník** dan **Kruskal**, masing-masing ditelusuri langkah demi langkah pada graf sembilan simpul yang sama, beserta hutan himpunan saling lepas yang membuat Kruskal cepat (Subbab 11.9–11.10).

5. Prim lawan Kruskal, bobot kembar dan ketunggalan, Profitina, kesalahan yang sering terjadi (Subbab 11.11–11.15).

Buku: Bab 11, Subbab 11.1 sampai 11.15.

Dari Mana Kita Melanjutkan

Dari Bab 10: graf, daftar ketetanggaan, dan pencarian mendalam-dahulu (DFS) beserta jamnya — waktu penemuan $d[u]$, waktu selesai $f[u]$, warna putih, kelabu, hitam, pendahulu $\pi[u]$.

Empat golongan sisi: **pohon** (putih), **balik** (kelabu, ke leluhur), **maju** dan **silang** (hitam). Hari ini bertumpu pada sisi balik.

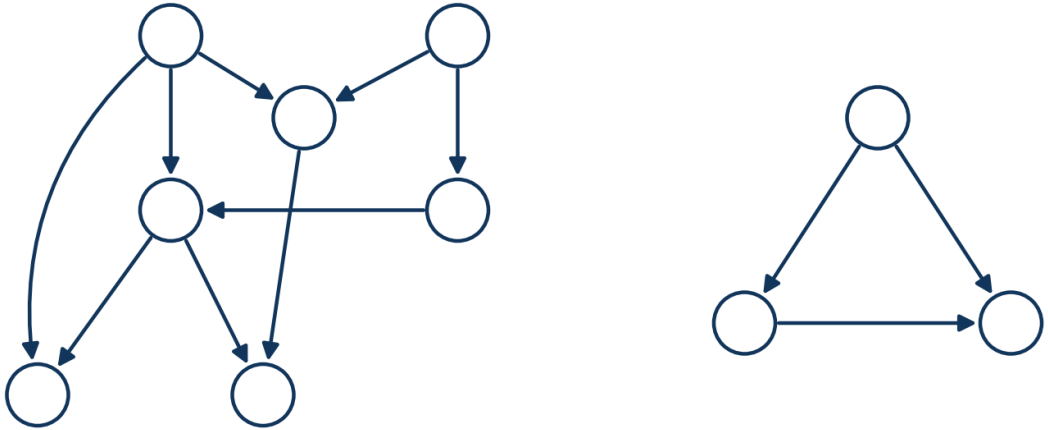
Dari Bab 9: disiplin rakus — **sifat pilihan rakus** dan **substruktur optimal** harus sama-sama dibuktikan sebelum kerakusan diizinkan.

Yang masih kurang: Bab 10 tidak pernah *mengukur* sebuah graf. Berikan bobot pada setiap sisi, lalu tanyakan himpunan sisi termurah yang menjaga semuanya tetap terhubung.

Buku: Subbab 11.1 dan Subbab 11.1.1.

Graf Berarah Asiklik

DAG adalah graf berarah tanpa siklus. **Gambar 11.1** memperlihatkan dua; yang kanan tampak seperti segitiga, tetapi anak panahnya tidak berputar mengelilinginya ke satu arah.



Gambar 11.1 Dua graf berarah asiklik. **Kiri:** ikuti anak panahnya dari mana saja dan anak panahnya selalu habis. **Kanan:** tiga sisi yang membentuk segitiga tetapi tetap asiklik, sebab anak panahnya tidak berputar mengelilinginya ke satu arah; membalik sisi dari simpul atas ke simpul kanan bawah akan menutup sebuah siklus berarah.

Buku: Subbab 11.2.1 dan 11.2.2, serta Gambar 11.1. DAG sehari-hari: ketergantungan build, prasyarat dalam kurikulum, jadwal tugas, kode paralel.

Teorema DAG, Dibuktikan

TEOREMA 11.1. Graf berarah G asiklik **jika dan hanya jika** pencarian mendalam-dahulu atas G tidak menghasilkan sisi balik. Jadi deteksi siklus berongkos $O(V + E)$.

Sisi balik ⇒ siklus. Jika (u, v) mendapati v kelabu, v ada pada tumpukan rekursi, leluhur u: lintasan pohon $v \rightsquigarrow u$ ditambah (u, v) menutup sebuah siklus.

Siklus ⇒ sisi balik. Misalkan v simpul **pertama** siklus itu yang ditemukan dan u pendahulunya pada siklus. Pada $d[v]$ sisa siklusnya putih — lintasan putih menuju u — sehingga u menjadi keturunan v, dan saat (u, v) diperiksa, v masih **kelabu**. ■

KATA "PERTAMA" MEMIKUL BEBAN, dan teoremanya tentang graf **berarah**: pada graf tak berarah, sisi yang baru saja Anda lewati tampak seperti sisi balik.

Buku: Subbab 11.2.3–11.2.5, Teorema 11.1.

Pengurutan Topologis

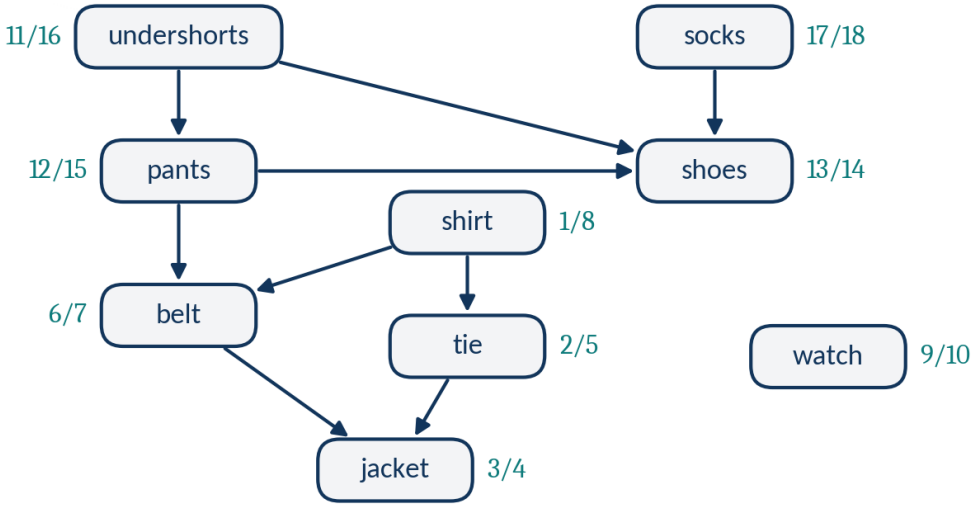
> **DEFINISI. Pengurutan topologis** sebuah DAG adalah pengurutan linear atas **seluruh** simpulnya sedemikian sehingga untuk setiap sisi (u, v), u muncul **sebelum** v — bentangkan simpulnya dan setiap anak panah menunjuk ke kanan.

Hampir tak pernah tunggal: pasangan yang tak terkendala boleh berurutan ke mana saja. Sebuah rantai hanya punya satu urutan; n simpul terpencil punya $n!$. **Periksalah sifatnya, jangan untai tersimpan.**

Contoh berjalannya **Gambar 11.2**: sembilan pakaian, sembilan sisi — undershorts → pants, undershorts → shoes, pants → shoes, pants → belt, shirt → belt, shirt → tie, belt → jacket, tie → jacket, socks → shoes. `watch` tak terkendala.

Buku: Subbab 11.3.1–11.3.3.

Berpakaian



Gambar 11.2 DAG pendahuluan untuk berpakaian, dengan cap waktu pencarian mendalam-dahulu $d[v]/f[v]$ tertulis di samping tiap simpul. Sembilan simpul, sembilan sisi. `watch` sama sekali tidak punya sisi: tidak ada yang mengekang kapan ia dikenakan.

Buku: Subbab 11.3.3 dan Gambar 11.2.

Algoritma 1: Waktu Selesai

Pseudokode 11.1 Pengurutan topologis dengan pencarian mendalam-dahulu, yang mencatat tiap simpul pada saat simpul itu selesai.

```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times f[v] for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

Pseudokode 11.1 menjalankan satu DFS dan menaruh tiap simpul di **depan** sebuah daftar saat simpul itu **selesai** — waktu selesai menurun, tanpa pengurutan apa pun.

Gerak hatinya: sebuah simpul baru selesai setelah semua yang terjangkau darinya selesai, jadi “paling akhir selesai” berarti “paling hulu”.

Buku: Subbab 11.3.4.

Contoh Terkerjakan 1: Setiap Detak Jam

Tabel 11.1 Setiap detak jam mendalam-dahulu pada **Gambar 11.2**. Jamnya maju tepat satu pada tiap penemuan dan tiap penyelesaian, sehingga ia berakhir di $2|V| = 18$. Bandingkan pasangan d/f-nya dengan angka yang tertulis di samping simpul-simpulnya pada **Gambar 11.2**: keduanya cocok persis.

Jam	Peristiwa	Keadaan
1	temukan `shirt`	d[shirt] = 1; `shirt` kelabu
2	dari `shirt`, `tie` putih: temukan ia	d[tie] = 2
3	dari `tie`, `jacket` putih: temukan ia	d[jacket] = 3
4	`jacket` tidak punya sisi keluar: selesaikan ia	f[jacket] = 4; senarai = `jacket`
5	kembali di `tie`, tidak ada sisi lagi: selesaikan ia	f[tie] = 5; senarai = `tie, jacket`
6	kembali di `shirt`, `belt` putih: temukan ia	d[belt] = 6
7	dari `belt`, `jacket` **hitam** — tidak ada yang perlu dikerjakan; selesaikan `belt`	f[belt] = 7; senarai = `belt, tie, jacket`
8	kembali di `shirt`, tidak ada sisi lagi: selesaikan ia	f[shirt] = 8
9	gelung luar: `watch` putih, temukan ia	d[watch] = 9
10	`watch` tidak punya sisi keluar: selesaikan ia	f[watch] = 10
11	gelung luar: `undershorts` putih, temukan ia	d[undershorts] = 11
12	dari `undershorts`, `pants` putih: temukan ia	d[pants] = 12

Jam	Peristiwa	Keadaan
13	dari `pants`, `shoes` putih: temukan ia	$d[\text{shoes}] = 13$
14	`shoes` tidak punya sisi keluar: selesaikan ia	$f[\text{shoes}] = 14$
15	kembali di `pants`, `belt` hitam: selesaikan `pants`	$f[\text{pants}] = 15$

Tabel 11.1 (Lanjutan) *Setiap detak jam mendalam-dahulu pada Gambar 11.2. Jamnya maju tepat satu pada tiap penemuan dan tiap penyelesaian, sehingga ia berakhir di 2V=18. Bandingkan pasangan d/f-nya dengan angka yang tertulis di samping simpul-simpulnya pada Gambar 11.2: keduanya cocok persis.*

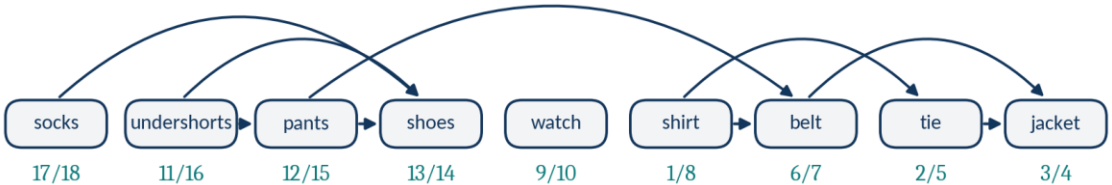
Jam	Peristiwa	Keadaan
16	kembali di `undershorts`, `shoes` hitam: selesaikan ia	$f[\text{undershorts}] = 16$
17	gelung luar: `socks` putih, temukan ia	$d[\text{socks}] = 17$
18	`shoes` hitam; selesaikan `socks`	$f[\text{socks}] = 18$

Tabel 11.1: gelung luar shirt, tie, jacket, belt, watch, undershorts, pants, shoes, socks. Jamnya berakhir pada $2|V| = 18$.

Buku: Subbab 11.3.5.

Setiap Anak Panah Menunjuk ke Kanan

Gambar 11.3: waktu selesai 18,16,15,14,10,8,7,5,4 — socks, undershorts, pants, shoes, watch, shirt, belt, tie, jacket.



Gambar 11.3 Hasilnya: kesembilan simpul **Gambar 11.2** dibentangkan dalam urutan waktu penyelesaian yang menurun. Setiap anak panah kini menunjuk ke kanan, dan itu persis definisi pengurutan topologis. Pasangan d/f dicetak di bawah tiap simpul supaya urutannya dapat diperiksa terhadap **Tabel 11.1**.

Buku: Subbab 11.3.5 dan Gambar 11.3.

Mengapa Waktu Selesai Berhasil

TEOREMA 11.2. Pada DAG, $f[v] < f[u]$ untuk setiap sisi (u, v); jadi urutan f yang menurun adalah pengurutan topologis.

Saat (u, v) diperiksa, u kelabu. v **kelabu:** sisi balik, mustahil pada DAG. v **putih:** v menjadi anak dan selesai di dalam panggilan u. v **hitam:** v sudah selesai sedangkan u belum. ■

Ongkos $\Theta(V + E)$: satu DFS ditambah $O(1)$ per penyisipan di depan — optimal, sebab masukannya berukuran $\Theta(V + E)$.

JEBAKAN. Sisipkan di **depan** dan pada waktu **selesai**. Menempel di belakang memberi urutan terbalik; mencatat pada waktu penemuan memberi `shirt, tie, jacket`, padahal `jacket` punya dua pendahulu yang belum terdaftar.

Buku: Subbab 11.3.6 dan 11.3.7, Teorema 11.2; programnya ada di Subbab 11.3.8.

Algoritma 2: Derajat Masuk Kahn

Pseudokode 11.2 Pengurutan topologis Kahn: berulang kali keluarkan simpul yang tidak punya pendahulu tersisa; kekurangan pada akhirnya membuktikan adanya siklus.

```
KAHN(G)
 1  compute indeg[v] for every vertex v
 2  R = { v : indeg[v] == 0 }              // R adalah himpunan "siap"
 3  L = empty list
 4  while R is not empty
 5      remove some vertex u from R
 6      append u to L
 7      for each v in Adj[u]
 8          indeg[v] = indeg[v] - 1        // u sudah selesai, jadi v menunggu satu lebih sedikit
 9          if indeg[v] == 0
10              add v to R
11  if |L| < |V|
12      report "G has a cycle"
13  return L
```

Tabel 11.2 Algoritma Kahn pada **Gambar 11.2**, satu pengeluaran tiap baris. Sembilan simpul keluar dari sembilan langkah, jadi tidak ada siklus. Bandingkan hasilnya dengan **Gambar 11.3**: kedua urutan itu tidak berbagi satu posisi pun, dan keduanya benar.

Langkah	Keluarkan	Derajat masuk yang jatuh ke 0	Himpunan siap sesudahnya
1	`shirt`	`tie`	watch, undershorts, socks, tie
2	`tie`	—	watch, undershorts, socks
3	`watch`	—	undershorts, socks
4	`undershorts`	`pants`	socks, pants
5	`pants`	`belt`	socks, belt
6	`belt`	`jacket`	socks, jacket
7	`jacket`	—	socks
8	`socks`	`shoes`	shoes
9	`shoes`	—	(kosong)

Pseudokode 11.2 mengeluarkan simpul mana pun yang berderajat masuk 0. Derajat masuk: shirt 0, tie 1, jacket 2, belt 2, watch 0, undershorts 0, pants 1, shoes 3, socks 0. **Tabel 11.2** memberi shirt, tie, watch, undershorts, pants, belt, jacket, socks, shoes — tak satu posisi pun sama dengan urutan DFS, dan keduanya benar.

Buku: Subbab 11.3.9 dan 11.3.10.

Kahn: Kebenaran, Ongkos, Pilihan

Benar: sebuah simpul ditambahkan hanya saat pencacahnya mencapai 0, yaitu setelah semua pendahulunya. **Lengkap:** bila ada simpul tersisa, masing-masing punya pendahulu yang juga tersisa, dan berjalan mundur di dalam himpunan berhingga pasti berulang — sebuah siklus.

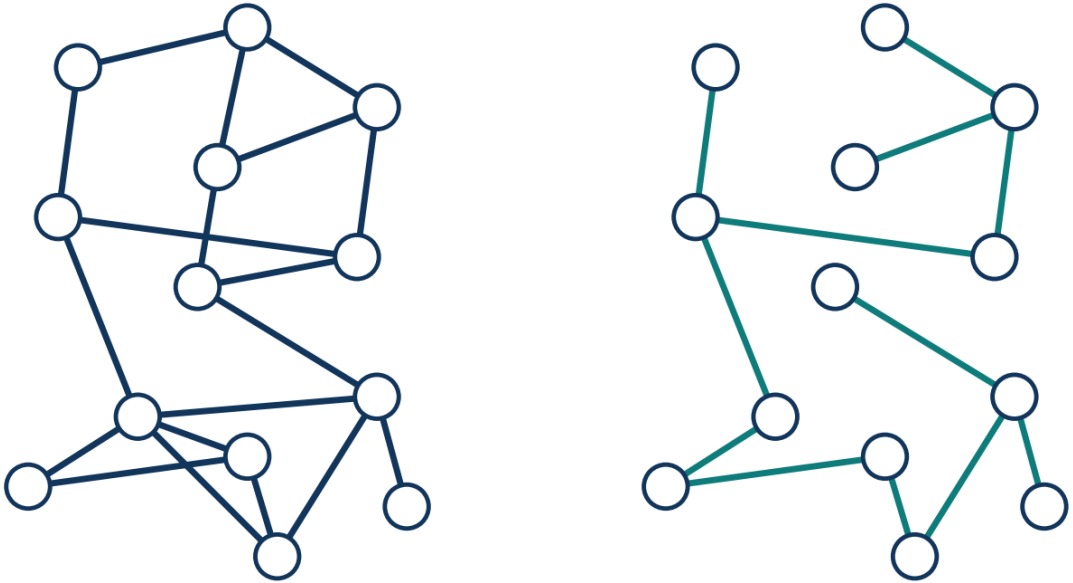
Jadi uji $|L| < |V|$ adalah pendeteksi siklus yang **tepat**; himpunan yang macet adalah diagnosisnya. Ongkos $\Theta(V + E)$ (dengan *heap* agar keluarannya rapi, $O(V \lg V + E)$).

Pakai yang mana? DFS: dua baris tambahan jika Anda sudah punya DFS, dan ia melaporkan sisi penyebabnya. Kahn: tanpa rekursi, kendali penuh atas urutan, gelombang paralel, dan langsung dapat diperluas ke penjadwalan.

Buku: Subbab 11.3.11–11.3.14.

Pohon Merentang

Pohon merentang dari graf terhubung adalah upagraf yang berupa pohon dan memuat **semua** simpul. **Teorema 11.3:** pohon dengan n simpul punya tepat $n - 1$ sisi; menambahkan sisi mana pun menciptakan tepat satu siklus. **Gambar 11.4**: 13 simpul, 12 sisi.



Gambar 11.4 Sebuah pohon merentang. **Kiri:** graf G. **Kanan:** sebuah pohon merentang G — setiap simpul G hadir, grafnya masih terhubung, dan tidak ada siklus yang tersisa. Hitunglah sisi di sebelah kanan: dua belas, untuk tiga belas simpul.

Buku: Subbab 11.4.1 dan 11.4.2 serta Gambar 11.4. Cayley: K_n punya n^{n-2} pohon merentang (Subbab 11.4.3).

Persoalan Pohon Merentang Minimum

Fungsi bobot $w: E \to \mathbb{R}$; bobot sebuah pohon adalah Persamaan (11.1). **Pohon merentang minimum** berbobot terkecil di antara semua pohon merentang.

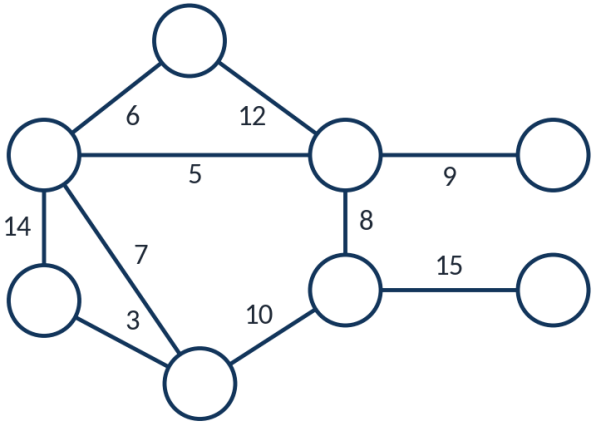
$$w(T) = \sum_{\{u,v\} \in T} w\,(u, v). \tag{11.1}$$

Sembilan desa dan air perpipaan: jawabannya harus merentang, harus asiklik (siklus memuat pipa yang dapat dibuang), dan harus termurah. K_9 saja sudah punya $9^7 = 4\,782\,969$ pohon merentang — gaya kasar tak ada harapan.

> **MST BUKAN LINTASAN TERPENDEK.** MST meminimumkan **total**; ia tidak meminimumkan jarak antara pasangan tertentu (Bab 13). **Gambar 11.5** adalah contoh pertamanya.

Buku: Subbab 11.5.1 dan 11.5.2.

Graf Berbobot Pertama

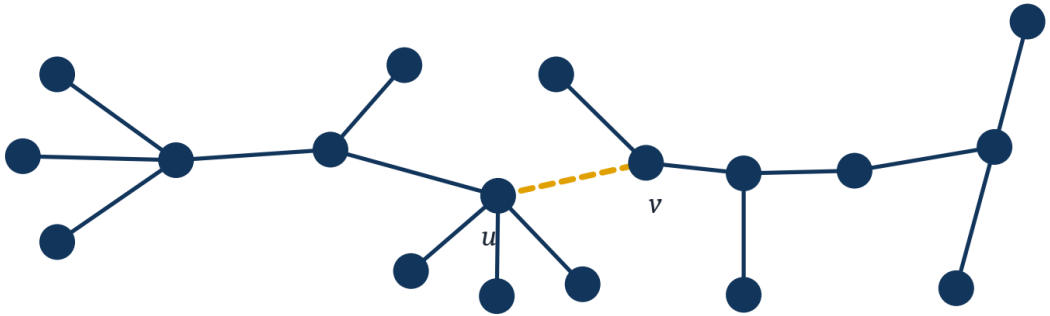


Gambar 11.5 Sebuah graf berbobot: graf tak berarah yang terhubung dengan sebuah bobot pada tiap sisinya. Sebagian sisi yang mahal tak terhindarkan (yang 15 adalah satu-satunya jalan menuju simpulnya); sebagian yang tampak murah justru berlebih. Membedakan keduanya adalah seluruh persoalannya.

Buku: Subbab 11.5.3 dan 11.5.4 serta Gambar 11.5. Mengambil $|V| - 1$ sisi termurah secara membabi buta dapat menutup siklus.

Substruktur Optimal: Pembelahannya

Buang sebuah sisi (u, v) dari MST T (**Gambar 11.6**): tersisa tepat dua potong, T_1 dan T_2.

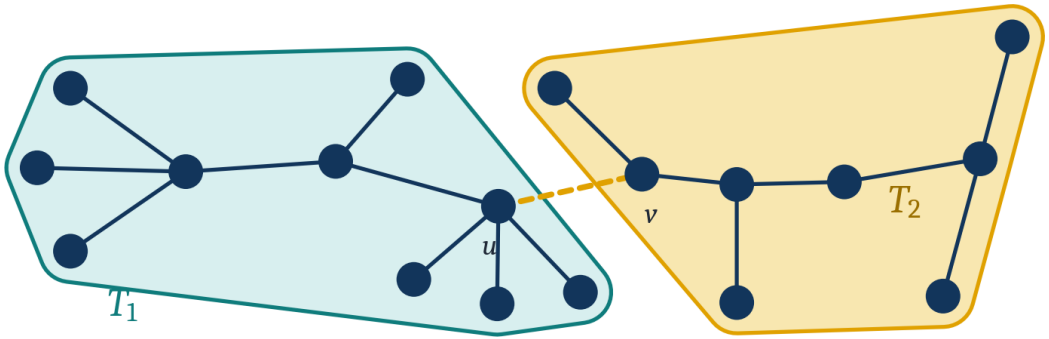


Gambar 11.6 Sebuah pohon merentang minimum T, dengan salah satu sisinya (u, v) digambar sebagai garis putus-putus. Menghapus sisi itu membelah T menjadi dua bagian.

Buku: Subbab 11.6.1 dan Gambar 11.6.

Kedua Potongnya, Diberi Nama

Gambar 11.7 menamai keduanya; bobotnya menjumlah seperti Persamaan (11.2).



Gambar 11.7 Gambar 11.6 dengan anotasi: kedua bagian yang tersisa sesudah sisi putus-putus (u, v) dihapus dinamai T_1 (kiri) dan T_2 (kanan). Klaim Teorema 11.4 adalah bahwa masing-masingnya sendiri merupakan pohon merentang minimum bagi upagraf G yang diinduksi oleh himpunan simpulnya sendiri.

$$w(T) = w(u, v) + w(T_1) + w(T_2). \tag{11.2}$$

Buku: Subbab 11.6.1 dan Gambar 11.7.

Teorema 11.4, lewat Potong-dan-Tempel

> **TEOREMA 11.4.** T_1 adalah pohon merentang minimum bagi upagraf G_1 yang diinduksi simpul-simpulnya; demikian pula T_2 bagi G_2.

Andaikan tidak: ada pohon merentang $T_1{}'$ bagi G_1 yang memenuhi Persamaan (11.3). Rekatkan $T' = T_1{}' \cup \{(u, v)\} \cup T_2$: ia merentang, terhubung, dan bersisi $|V| - 1$, jadi ia pohon merentang.

$$w(T_1{}') < w(T_1). \tag{11.3}$$

Maka $w(T') = w(u, v) + w(T_1{}') + w(T_2) < w(T)$, bertentangan dengan keminimuman T. ■

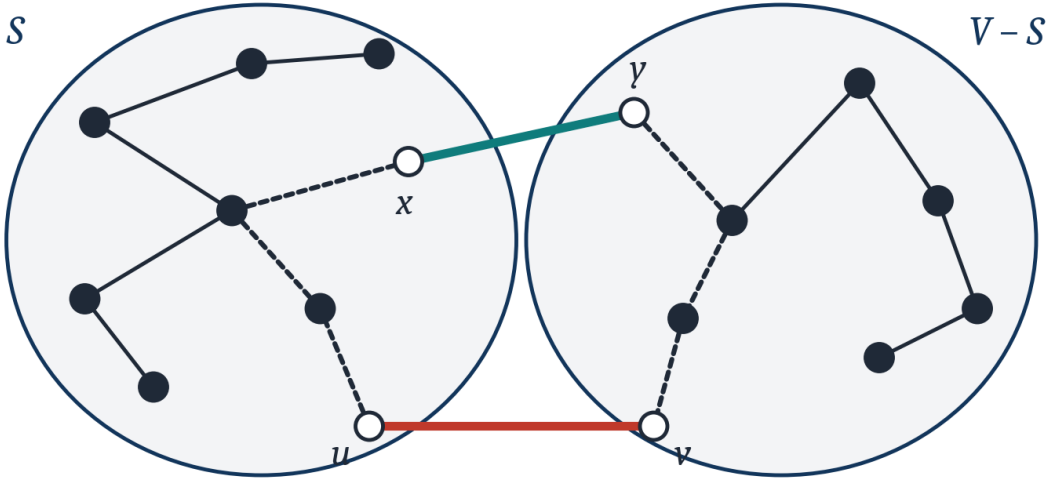
Yang tidak diberikannya: teorema ini mengandaikan MST sudah di tangan. Ia mengizinkan kerakusan; ia tidak mengatakan **sisi mana** yang diambil lebih dulu.

Buku: Subbab 11.6.2 dan 11.6.3, Teorema 11.4.

Sifat Potongan

DEFINISI. Potongan $(S, V - S)$ membelah simpul menjadi dua bagian tak kosong. Sebuah sisi **melintasinya** jika tepat satu ujungnya di S. Potongan **menghormati** A jika tak ada sisi A yang melintasinya. Sisi lintas berbobot minimum disebut **sisi ringan**.

TEOREMA 11.5. Untuk sembarang potongan $(S, V - S)$, sisi berbobot minimum yang melintasinya termasuk dalam **suatu** pohon merentang minimum. **Gambar 11.8** menggambarkan buktinya.



Gambar 11.8 Argumen pertukaran yang membuktikan Teorema 11.5. S dan $V - S$ adalah kedua gumpalannya. Sisi merah (u, v) di bawah adalah sisi ringannya, yang diandaikan *tidak* berada di dalam pohonnya. Garis putus-putus adalah lintasan tunggal yang disediakan pohonnya dari u ke v; sisi toska (x, y) adalah sisi pertama lintasan itu yang melintasi potongannya. Menukar (x, y) dengan (u, v) adalah langkah yang diambil buktinya.

Buku: Subbab 11.7.1–11.7.3, Teorema 11.5, dan Gambar 11.8.

Penukarannya, Langkah demi Langkah

Misalkan T sebuah MST tanpa sisi ringan (u, v). Lintasan pohon yang tunggal dari u ke v bermula di S dan berakhir di luarnya, sehingga salah satu sisinya, (x, y), melintasi potongan itu, dan berlaku Persamaan (11.4).

$$w(u, v) \leq w(x, y). \tag{11.4}$$

$T' = (T \setminus \{(x, y)\}) \cup \{(u, v)\}$ adalah pohon merentang dengan $w(T') \leq w(T)$, jadi ia MST — dan ia memuat (u, v). ■

Akibat 11.6: jika A termuat dalam suatu MST dan potongannya **menghormati** A, sisi ringannya **aman**: $A \cup \{(u, v)\}$ tetap termuat dalam suatu MST (sisi yang ditukar keluar melintasi potongan, jadi bukan anggota A).

Pemeriksaan pada graf sembilan simpul: $S = \{a\}$ memberi $\{a, b\}$ (4); $S = \{g, h\}$ memberi $\{f, g\}$ (2); $S = \{d\}$ memberi $\{c, d\}$ (7) — semuanya ada di kedua MST.

Buku: Subbab 11.7.3–11.7.6, Akibat 11.6.

Algoritma MST Generik

Pseudokode 11.3 Algoritma pohon merentang minimum generik: tumbuhkan himpunan sisi A satu sisi aman setiap kalinya.

```
GENERIC-MST(G, w)
1  A = {}                                  // memuat sisi-sisi yang termasuk dalam sebuah MST
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A union {(u, v)}
5  return A
```

Pseudokode 11.4 Algoritma generik dengan kata "aman" diganti Akibat 11.6: pilih potongan yang menghormati A lalu ambil sisi ringannya.

```
MORE-SPECIFIC-MST(G, w)
1    A = {}                                // memuat sisi-sisi yang termasuk dalam sebuah MST
2    while A does not form a spanning tree
3.1      make a cut (S, V-S) of G that respects A
3.2      take the min-weight edge (u, v) connecting S to V-S
4        A = A union {(u, v)}
5    return A
```

Pseudokode 11.3 menambahkan sisi **aman** setiap kali; invarian *"A adalah himpunan bagian suatu MST"* berlaku di awal, terpelihara menurut definisi aman, dan pada $|V| - 1$ sisi memberi sebuah MST.

Pseudokode 11.4 menyisakan tepat satu keputusan — **potongan yang mana**. Prim: S = pohon sejauh ini (digerakkan simpul, antrean prioritas). Kruskal: S = satu komponen hutan (digerakkan sisi, diurutkan sekali).

Buku: Subbab 11.8.

Prim–Jarník

Satu awan yang tumbuh. Untuk v di luar pohon, $key[v]$ adalah sisi termurah dari v ke pohon dan $\pi[v]$ ujung lainnya; kunci terkecil adalah sisi ringan potongannya.

Pseudokode 11.5 Algoritma Prim–Jarník: $key[v]$ adalah bobot sisi termurah dari v ke dalam pohon yang sedang tumbuh, dan $\pi[v]$ adalah ujung lainnya.

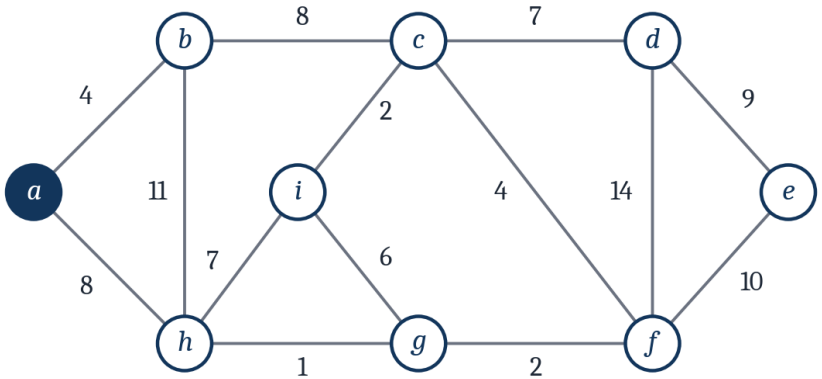
```
MST-PRIM(G, w, r)
01  Q = G.V                          // Q: simpul-simpul yang belum masuk T
02  for each u in Q
03      key[u] = INFINITY
04  key[r] = 0
05  pi[r] = NIL
06  while Q is not empty
07      u = EXTRACT-MIN(Q)            // u menjadi bagian dari T
08      for each v in Adj[u]
09          if v in Q and w(u, v) < key[v]
10              pi[v] = u
11              key[v] = w(u, v)
```

PRIM BUKAN DIJKSTRA. **Pseudokode 11.5** menguji $w(u,v) < key[v]$ — satu sisi, tanpa panjang yang terakumulasi. Menulis `key[u] + w(u,v)` menghasilkan pohon lintasan terpendek.

Buku: Subbab 11.9.1–11.9.4; antrean prioritasnya menyediakan `BuildPQ`, `ExtractMin`, dan `DecreaseKey`.

Prim, Langkah 1: Akarnya

Gambar 11.9 sekaligus graf bagi setiap penelusuran: sembilan simpul, empat belas sisi. Keluarkan a; kini $Q = \{b: 4/a,\ h: 8/a,\ \text{sisanya } \infty\}$.

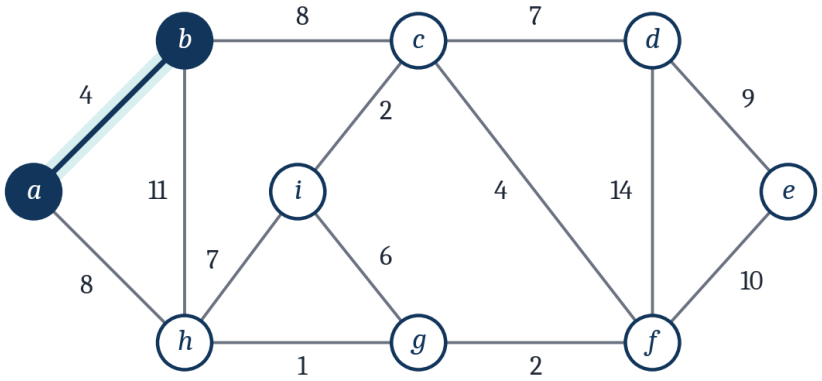


Gambar 11.9 Langkah 1 penelusuran Prim: pohonnya baru berupa akar a. Ini sekaligus grafnya sendiri — sembilan simpul, empat belas sisi, bobot 1 sampai 14 — dan inilah graf yang dipakai untuk setiap penelusuran di sisa bab ini.

Buku: Subbab 11.9.5 dan Gambar 11.9.

Prim, Langkah 2

Gambar 11.10: keluarkan b: 4, ikatkan (a, b), total 4. $key[c] \leftarrow 8/b$; h tetap $8/a$ karena $11 \not< 8$.

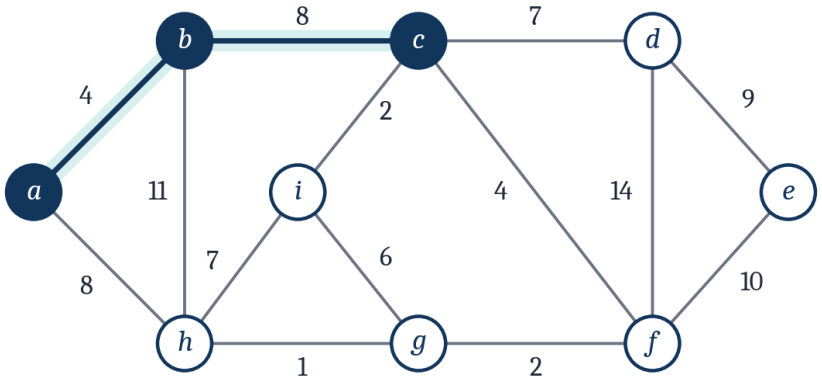


Gambar 11.10 Langkah 2: sisi ringan bagi potongan $S = \{a\}$ adalah (a, b) berbobot 4, sehingga b bergabung ke pohonnya.

Buku: Subbab 11.9.5 dan Gambar 11.10.

Prim, Langkah 3: Seri

Gambar 11.11: c dan h seri pada 8; penelusurannya mengambil c — ikatkan (b, c), total 12. Kini i: 2/c, f: 4/c, d: 7/c.

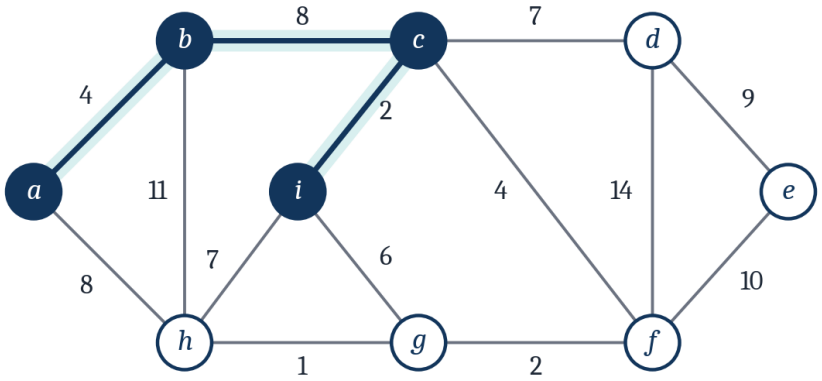


Gambar 11.11 Langkah 3: dengan $S = \{a, b\}$ sisi-sisi yang melintas adalah $(b, c) = 8$, $(a, h) = 8$, dan $(b, h) = 11$. Dua di antaranya seri sebagai yang teringan; penelusurannya mengambil (b, c).

Buku: Subbab 11.9.5 dan Gambar 11.11.

Prim, Langkah 4

Gambar 11.12: keluarkan i: 2, ikatkan (c, i), total 14. $g \leftarrow 6/i$ dan h **membaik** dari $8/a$ menjadi $7/i$.

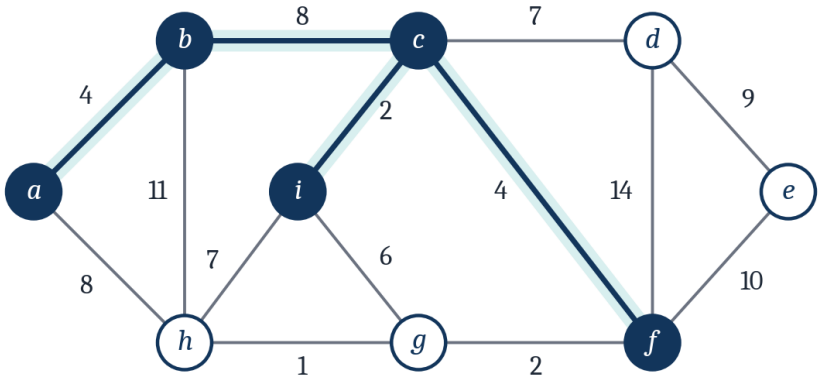


Gambar 11.12 Langkah 4: (c, i) berbobot 2 adalah jalan keluar termurah dari $\{a, b, c\}$, sehingga i bergabung. Ini juga langkah saat kunci h *diperbaiki*, dari 8 menjadi 7.

Buku: Subbab 11.9.5 dan Gambar 11.12.

Prim, Langkah 5

Gambar 11.13: keluarkan f: 4, ikatkan (c, f), total 18. g membaik menjadi $2/f$; $e \leftarrow 10/f$.

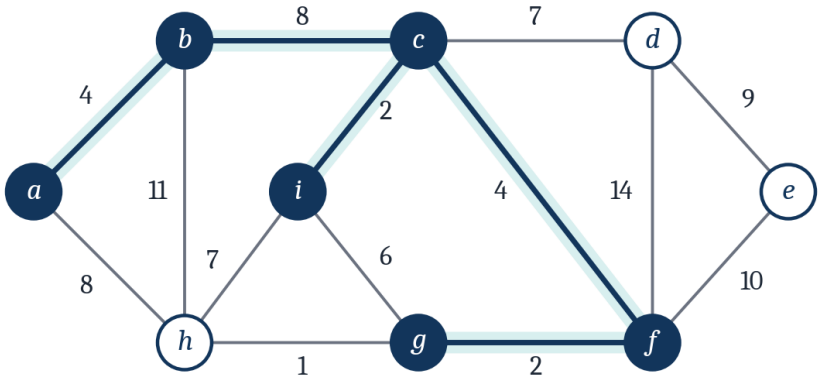


Gambar 11.13 Langkah 5: (c, f) berbobot 4 memasukkan f. Pohonnya kini $\{a, b, c, i, f\}$ dan kunci g turun dari 6 menjadi 2.

Buku: Subbab 11.9.5 dan Gambar 11.13.

Prim, Langkah 6

Gambar 11.14: keluarkan g: 2, ikatkan (f, g), total 20. h membaik lagi, menjadi $1/g$.

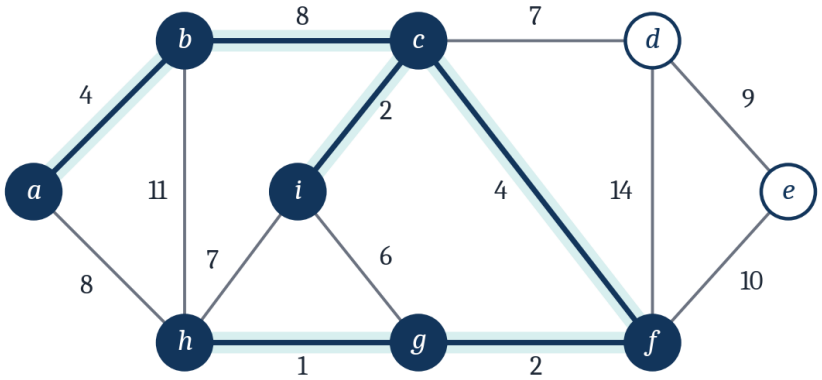


Gambar 11.14 Langkah 6: (f, g) berbobot 2 memasukkan g. Simpul h kini sudah tiga kali kuncinya diturunkan — 8, lalu 7, lalu 1 — dan persis untuk itulah `DecreaseKey` ada.

Buku: Subbab 11.9.5 dan Gambar 11.14.

Prim, Langkah 7

Gambar 11.15: keluarkan h: 1, ikatkan (g, h), total 21 — sisi teringan di seluruh graf, baru diambil pada langkah 7.

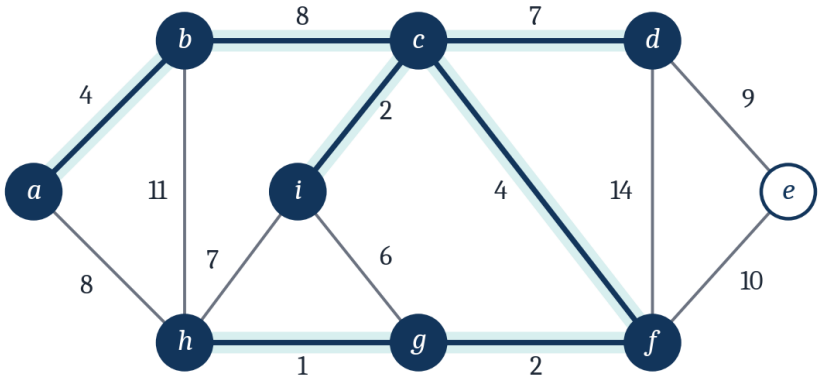


Gambar 11.15 Langkah 7: sisi termurah di seluruh grafnya, (g, h) berbobot 1, baru diambil sekarang — pada langkah ketujuh. Prim mengambil sisi menurut urutan awannya yang tumbuh menjangkau mereka, bukan menurut urutan bobot.

Buku: Subbab 11.9.5 dan Gambar 11.15.

Prim, Langkah 8

Gambar 11.16: keluarkan d: 7, ikatkan (c, d), total 28; e membaik dari $10/f$ menjadi $9/d$.

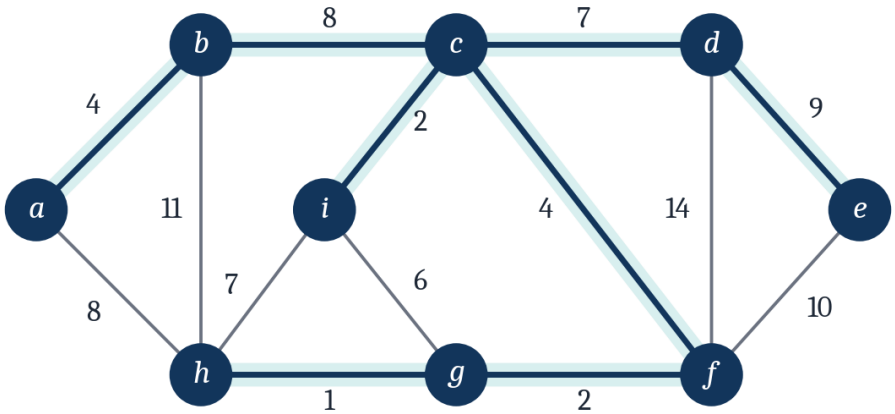


Gambar 11.16 Langkah 8: (c, d) berbobot 7 memasukkan d, dan kunci e membaik dari 10 menjadi 9.

Buku: Subbab 11.9.5 dan Gambar 11.16.

Prim, Langkah 9: Pohonnya

Gambar 11.17: keluarkan e: 9, ikatkan (d, e), total **37**. Q kosong.



Gambar 11.17 Langkah 9 — gambar terakhirnya. Kesembilan simpulnya hitam dan kedelapan sisi yang tersorot membentuk pohon merentang minimumnya: (a, b), (b, c), (c, i), (c, f), (f, g), (g, h), (c, d), (d, e), dengan bobot total 37.

Buku: Subbab 11.9.5 dan Gambar 11.17.

Contoh Terkerjakan 2: Prim dalam Satu Halaman

Tabel 11.3 Jalan lengkap Prim–Jarník dari **Gambar 11.9** sampai **Gambar 11.17**, dengan antrean prioritasnya dicetak sesudah tiap langkah sebagai `simpul:kunci/π`. Sembilan pengeluaran, delapan sisi, bobot total 37. Barisan bobot yang dikomitmenkan adalah 4, 8, 2, 4, 2, 1, 7, 9 — tidak terurut, sebab Prim tidak mengurutkan.

Langkah	Keluarkan	*key*	Sisi yang dikomitmenkan	Total	*Q* Sesudahnya
1	a	0	— (akar)	0	b:4/a, h:8/a, sisanya ∞
2	b	4	(a, b)	4	c:8/b, h:8/a, sisanya ∞
3	c	8	(b, c)	12	i:2/c, f:4/c, d:7/c, h:8/a, e:∞, g:∞
4	i	2	(c, i)	14	f:4/c, g:6/i, d:7/c, h:7/i, e:∞
5	f	4	(c, f)	18	g:2/f, d:7/c, h:7/i, e:10/f
6	g	2	(f, g)	20	h:1/g, d:7/c, e:10/f
7	h	1	(g, h)	21	d:7/c, e:10/f
8	d	7	(c, d)	28	e:9/d
9	e	9	(d, e)	37	(kosong)

Tabel 11.3: bobot yang diikatkan 4, 8, 2, 4, 2, 1, 7, 9 — tidak berurutan, sebab Prim hanya memandang tepi awannya sendiri.

Buku: Subbab 11.9.5.

Mengapa Prim Benar, dan Ongkosnya

Benar: sisi-sisi yang diikatkan membentuk pohon atas $S = V \setminus Q$; potongan $(S, V - S)$ menghormatinya; sisi milik simpul yang dikeluarkan adalah sisi ringan. Akibat 11.6 mengerjakan sisanya.

Ongkos: $|V|$ pengeluaran dan $O(E)$ perubahan kunci, Persamaan (11.5); dengan *heap* biner, Persamaan (11.6).

$$\text{Waktu} \;=\; |V| \cdot T(\text{ExtractMin}) \;+\; O(E) \cdot T(\text{ModifyKey}). \tag{11.5}$$

$$\text{Waktu} = O(V \lg V) + O(E \lg V) = O((V + E)\lg V) = O(E \lg V), \tag{11.6}$$

Tabel 11.4 Waktu jalan Prim dengan tiga antrean prioritas. Baris *Fibonacci heap* adalah alasan struktur itu ada: ia membuat `DecreaseKey` menjadi gratis dalam arti teramortisasi, dan `DecreaseKey` adalah operasi yang paling sering dilakukan Prim.

Q	T(ExtractMin)	T(DecreaseKey)	Total
larik tak terurut	$O(V)$	$O(1)$	$O(V^2)$
tumpukan biner	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$ teramortisasi	$O(1)$ teramortisasi	$O(V \lg V + E)$

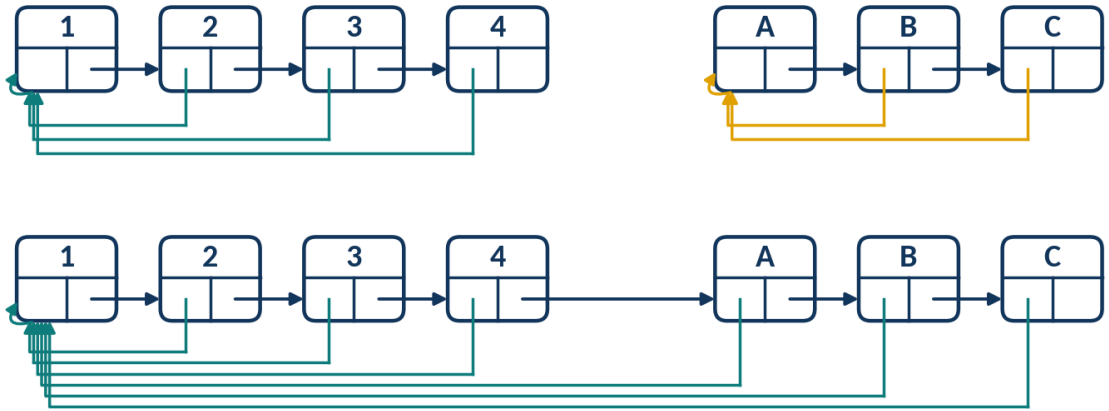
Tabel 11.4: larik untuk graf padat, *heap* biner untuk graf jarang, *heap* Fibonacci untuk batas terbaik. `std::priority_queue` tidak punya `DecreaseKey`: versi malasnya memasukkan salinan baru dan melewati salinan basi, $O(E \lg E)$.

Buku: Subbab 11.9.6–11.9.10.

Kruskal dan Himpunan Saling Lepas

Urutkan semua sisi; terima setiap sisi yang ujung-ujungnya berada di **pohon yang berbeda** dalam hutannya. Pertanyaan "sepohon?" diajukan $|E|$ kali: **himpunan saling lepas** — `MakeSet`, `Union`, `FindSet`.

Percobaan 1, **daftar** (**Gambar 11.18**): `FindSet` $O(1)$; `Union` menunjuk ulang daftar yang lebih pendek. Setiap perpindahan sekurang-kurangnya menggandakan himpunan sebuah simpul, sehingga totalnya Persamaan (11.7).



Gambar 11.18 Himpunan saling lepas sebagai senarai. **Atas:** dua himpunan, $\{1,2,3,4\}$ dan $\{A, B, C\}$; di dalam masing-masingnya, tiap elemen membawa penunjuk balik ke elemen pertama, yang merupakan wakilnya. **Bawah:** hasil penggabungan keduanya — senarai yang lebih pendek sudah ditempelkan dan setiap elemennya diarahkan ulang ke wakil yang baru, yaitu 1.

$$\sum_{v \in V} t\,(v) \;\leq\; |V|\lg|V| \;=\; O(V\lg V). \tag{11.7}$$

Buku: Subbab 11.10.1–11.10.3 dan Gambar 11.18.

Pohon, Gabung menurut Peringkat, Pemampatan Lintasan

Percobaan 2, **pohon**: `parent[]`, akarnya menjadi wakil. Penggabungan naif dapat membangun lintasan sedalam $n - 1$.

Pseudokode 11.6 Membuat himpunan bersinggel, dan gabung menurut peringkat: akar berperingkat lebih kecil digantungkan di bawah akar berperingkat lebih besar.

```
MAKE-SET(x)
1  parent[x] = x
2  rank[x] = 0

UNION(x, y)
1  rx = FIND-SET(x)
2  ry = FIND-SET(y)
3  if rx == ry
4      return                          // sudah menyatu
5  if rank[rx] < rank[ry]
6      swap rx and ry
7  parent[ry] = rx
8  if rank[rx] == rank[ry]
9      rank[rx] = rank[rx] + 1
```

Pseudokode 11.7 Pencarian dengan pemampatan lintasan: dalam perjalanan kembali dari akar, setiap simpul pada lintasannya diarahkan ulang ke akar.

```
FIND-SET(x)
1  if parent[x] != x
2      parent[x] = FIND-SET(parent[x])
3  return parent[x]
```

Pseudokode 11.6: gantungkan peringkat kecil di bawah peringkat besar. **Lema 11.7:** peringkat k berarti sekurang-kurangnya 2^k elemen, jadi tingginya $\leq \lg n$. **Pseudokode 11.7**: tunjuk ulang setiap simpul pada lintasannya ke akar.

Buku: Subbab 11.10.4–11.10.6, Lema 11.7.

Contoh Terkerjakan 3: Kedua Siasat Bekerja

Tabel 11.5 Sebuah hutan himpunan saling lepas dengan gabung menurut peringkat dan pemampatan lintasan, dicetak sesudah tiap operasi. Entri yang berubah dicetak tebal. Dua hal yang perlu diperhatikan: peringkat sebuah akar hanya naik ketika dua peringkat yang sama bertemu, dan dalam barisan ini ketujuh penggabungan sungguhan semuanya mempertemukan peringkat yang sama — menjadi 1 pada baris 2–5, menjadi 2 pada baris 6–7, dan menjadi 3 pada baris 8; dan `Union(1,5)` yang "tanpa efek" itu toh tetap mengubah `parent[5]` dari 4 menjadi 0, sebab kedua pencarian yang dilakukannya memampatkan lintasannya dalam perjalanan. Sesudah sapuan pencarian terakhirnya, setiap elemen menunjuk langsung ke akarnya dan peringkatnya — yang memang hanya pernah menjadi batas *atas* bagi tingginya — dibiarkan basi, dan itu tidak berbahaya.

Operasi	`parent` (elemen 0…7)	`rank`	Himpunan
sesudah `MakeSet`	0 1 2 3 4 5 6 7	0 0 0 0 0 0 0 0	8
`Union(0,1)`	0 **0** 2 3 4 5 6 7	**1** 0 0 0 0 0 0 0	7
`Union(2,3)`	0 0 2 **2** 4 5 6 7	1 0 **1** 0 0 0 0 0	6
`Union(4,5)`	0 0 2 2 4 **4** 6 7	1 0 1 0 **1** 0 0 0	5
`Union(6,7)`	0 0 2 2 4 4 6 **6**	1 0 1 0 1 0 **1** 0	4
`Union(0,2)`	0 0 **0** 2 4 4 6 6	**2** 0 1 0 1 0 1 0	3
`Union(4,6)`	0 0 0 2 4 4 **4** 6	2 0 1 0 **2** 0 1 0	2
`Union(0,4)`	0 0 0 2 **0** 4 4 6	**3** 0 1 0 2 0 1 0	1

Operasi	parent (elemen 0…7)	rank	Himpunan
`Union(1,5)` (tanpa efek)	0 0 0 2 0 **0** 4 6	3 0 1 0 2 0 1 0	1

Tabel 11.5 (Lanjutan) *Sebuah hutan himpunan saling lepas dengan gabung menurut peringkat dan pemampatan lintasan, dicetak sesudah tiap operasi. Entri yang berubah dicetak tebal. Dua hal yang perlu diperhatikan: peringkat sebuah akar hanya naik ketika dua peringkat yang sama bertemu, dan dalam barisan ini ketujuh penggabungan sungguhan semuanya mempertemukan peringkat yang sama — menjadi 1 pada baris 2–5, menjadi 2 pada baris 6–7, dan menjadi 3 pada baris 8; dan Union(1,5) yang "tanpa efek" itu toh tetap mengubah parent[5] dari 4 menjadi 0, sebab kedua pencarian yang dilakukannya memampatkan lintasannya dalam perjalanan. Sesudah sapuan pencarian terakhirnya, setiap elemen menunjuk langsung ke akarnya dan peringkatnya — yang memang hanya pernah menjadi batas atas bagi tingginya — dibiarkan basi, dan itu tidak berbahaya.*

Operasi	parent (elemen 0…7)	rank	Himpunan
sesudah `find(0)`…`find(7)`	**0 0 0 0 0 0 0 0**	3 0 1 0 2 0 1 0	1

Tabel 11.5: peringkat naik hanya bila dua peringkat sama bertemu; `Union(1,5)` yang "tak berbuat apa-apa" tetap memampatkan; sesudah semua `find`, setiap elemen menunjuk ke 0 dan peringkatnya dibiarkan basi — tidak berbahaya.

Buku: Subbab 11.10.6; programnya ada di Subbab 11.10.8.

Nyaris Konstan

Dengan kedua siasat, m operasi atas n elemen berongkos Persamaan (11.8), dengan α fungsi Ackermann balikan (Tarjan; buktinya analisis teramortisasi, Bab 14).

$$O(m \cdot \alpha(n)), \tag{11.8}$$

Tabel 11.6 Bagaimana fungsi Ackermann balikan tumbuh. Banyaknya atom di alam semesta teramati kira-kira 2^{266}, sehingga untuk setiap persoalan himpunan saling lepas yang akan pernah dijalankan di atas perangkat keras fisik, $\alpha(n) \leq 4$.

n	$\alpha(n)$
$0 \leq n \leq 2$	0
$n = 3$	1
$4 \leq n \leq 7$	2
$8 \leq n \leq 2047$	3
$2048 \leq n \leq A_4(1)$, bilangan yang jauh lebih besar daripada $2^{2^{2048}}$	4

Tabel 11.6: $\alpha(n) \leq 4$ untuk setiap n yang pernah dijalankan pada perangkat keras. Dan $\Omega(\alpha(n))$ tak terhindarkan bagi struktur apa pun (Fredman dan Saks).

Buku: Subbab 11.10.7.

Algoritma Kruskal

Pseudokode 11.8 Algoritma Kruskal: pindai sisi-sisinya dari yang teringan sampai yang terberat dan simpan setiap sisi yang menghubungkan dua pohon hutan yang berbeda.

```
MST-KRUSKAL(G, w)
01  A = {}
02  for each vertex v in G.V
03      MAKE-SET(v)
04  sort the edges of G.E into non-decreasing order by weight w
05  for each edge (u, v) in G.E, taken in non-decreasing order by weight
06      if FIND-SET(u) != FIND-SET(v)
07          A = A union {(u, v)}
08          UNION(u, v)
09  return A
```

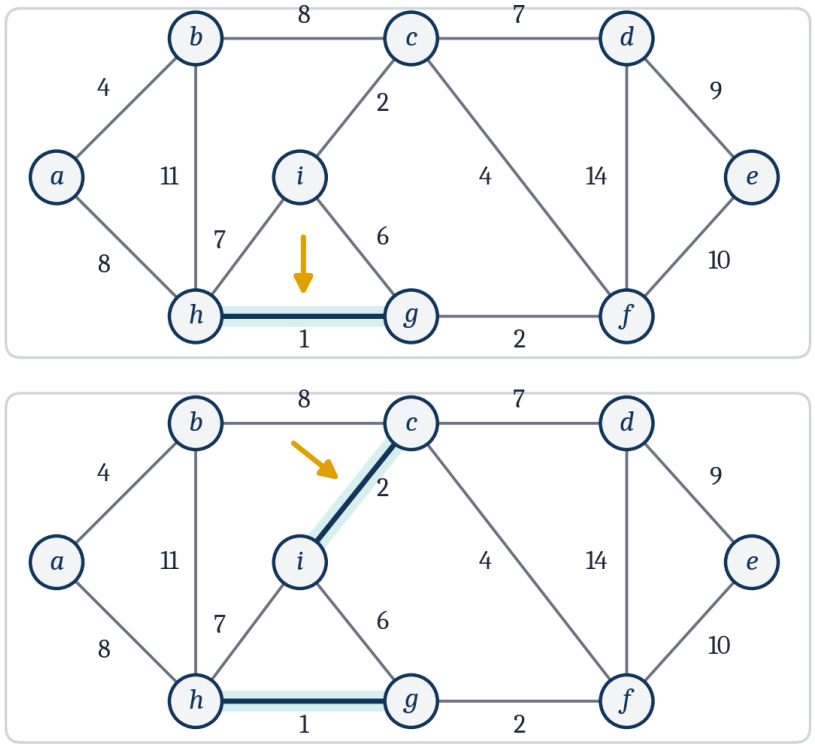
Pseudokode 11.8 pada graf sembilan simpul, terurut: $(g,h)1$, $(c,i)2$, $(f,g)2$, $(a,b)4$, $(c,f)4$, $(g,i)6$, $(c,d)7$, $(h,i)7$, $(a,h)8$, $(b,c)8$, $(d,e)9$, $(e,f)10$, $(b,h)11$, $(d,f)14$.

Tujuh halaman berikut mengambil keempat belas sisi dua demi dua; sebuah anak panah menandai sisi yang sedang ditimbang.

Buku: Subbab 11.10.9 dan 11.10.10.

Kruskal, Sisi 1 dan 2

Gambar 11.19: (g, h) 1: terima (total 1). (c, i) 2: terima (total 3). Kruskal mulai dari sisi termurah di seluruh graf.

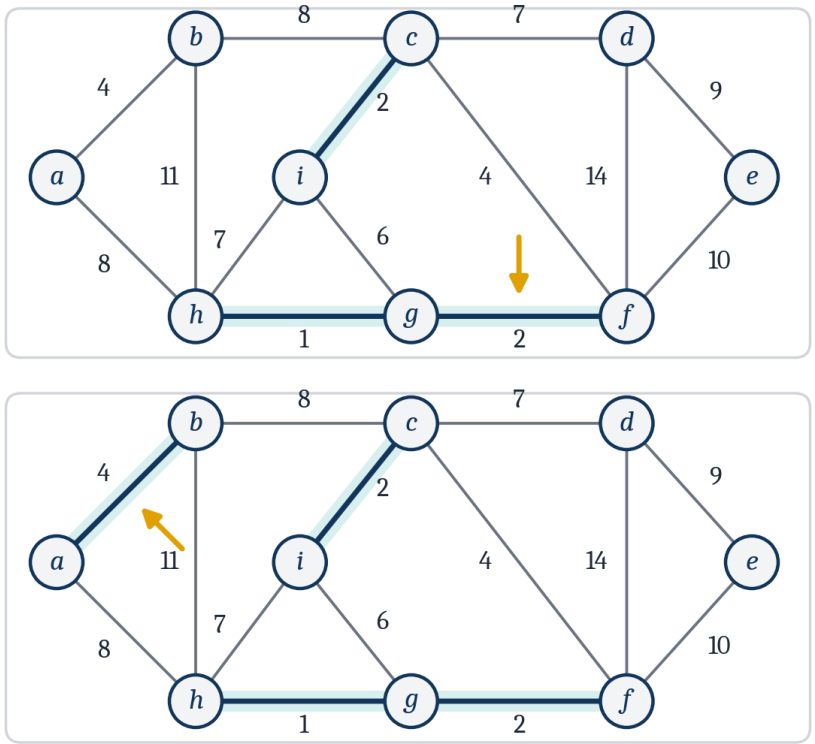


Gambar 11.19 Dua langkah Kruskal yang pertama. **Atas:** (g, h) berbobot 1 — sisi termurah di seluruh grafnya — diterima. **Bawah:** (c, i) berbobot 2 diterima. Tidak seperti Prim, Kruskal memulai dengan sisi termurah secara global, di mana pun ia kebetulan berada.

Buku: Subbab 11.10.10 dan Gambar 11.19.

Kruskal, Sisi 3 dan 4

Gambar 11.20: (f, g) 2: terima, $\{f, g, h\}$ (total 5). (a, b) 4: terima (total 9). Empat pohon tumbuh bersamaan.

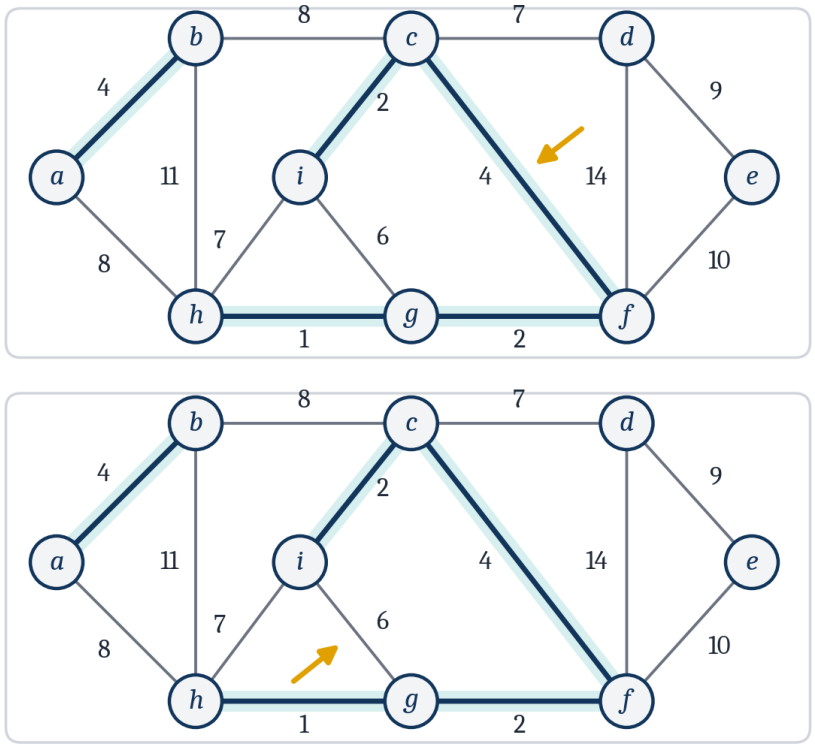


Gambar 11.20 Atas: (f, g) berbobot 2 diterima, meleburkan $\{f\}$ dengan $\{g, h\}$. **Bawah:** (a, b) berbobot 4 diterima. Perhatikan bahwa hutannya kini punya empat pohon terpisah yang tumbuh serentak — sesuatu yang tidak pernah dilakukan Prim.

Buku: Subbab 11.10.10 dan Gambar 11.20.

Kruskal, Sisi 5 dan 6

Gambar 11.21: (c, f) 4: terima, $\{c, f, g, h, i\}$ (total 13). (g, i) 6: `Find` memberi c untuk keduanya — **tolak**; sisi itu akan menutup $g - f - c - i$.

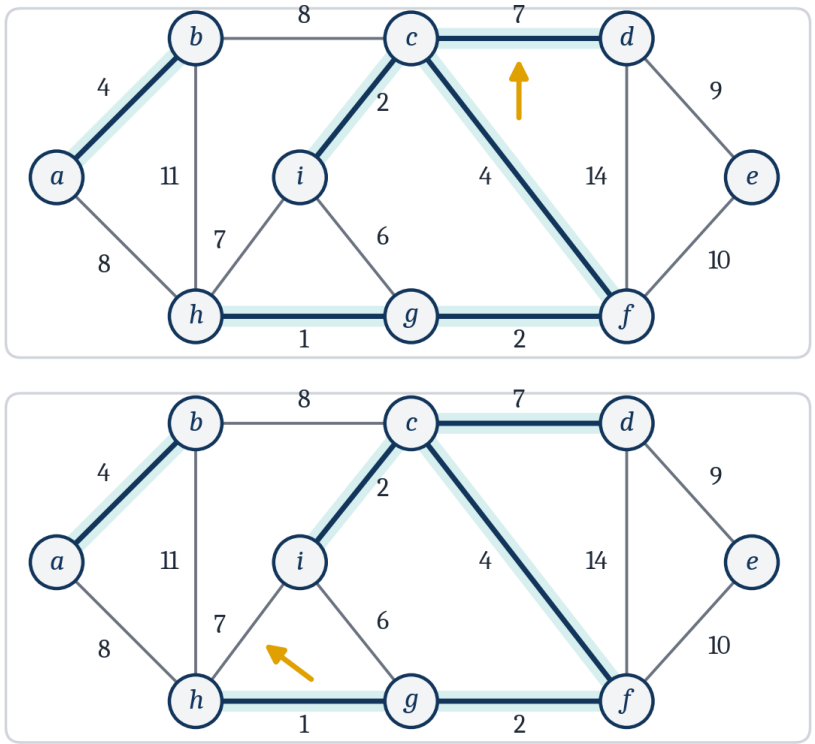


Gambar 11.21 Atas: (c, f) berbobot 4 diterima, meleburkan dua pohon menjadi $\{c, f, g, h, i\}$. **Bawah:** penolakan yang pertama — (g, i) berbobot 6 *tidak* tersorot, sebab g dan i sudah berada di satu pohon dan sisi itu akan menutup sebuah siklus.

Buku: Subbab 11.10.10 dan Gambar 11.21.

Kruskal, Sisi 7 dan 8

Gambar 11.22: (c, d) 7: terima (total 20). (h, i) 7: tolak. Bobot sama, nasib berlawanan — partisinyalah yang memutuskan.

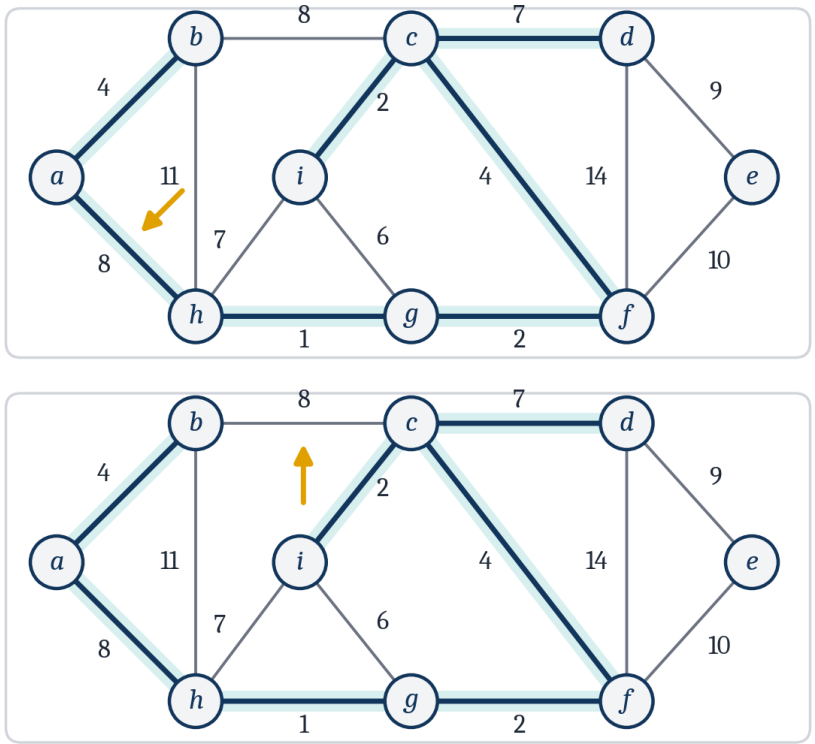


Gambar 11.22 Atas: (c, d) berbobot 7 diterima. **Bawah:** (h, i), yang juga berbobot 7, ditolak. Dua sisi berbobot sama, nasib berlawanan — keputusannya sama sekali tidak berkaitan dengan bobotnya dan sepenuhnya berkaitan dengan partisinya.

Buku: Subbab 11.10.10 dan Gambar 11.22.

Kruskal, Sisi 9 dan 10

Gambar 11.23: (a, h) 8: terima (total 28). (b, c) 8: tolak. **Di sinilah Kruskal berpisah jalan dengan Prim**, yang mengambil (b, c).

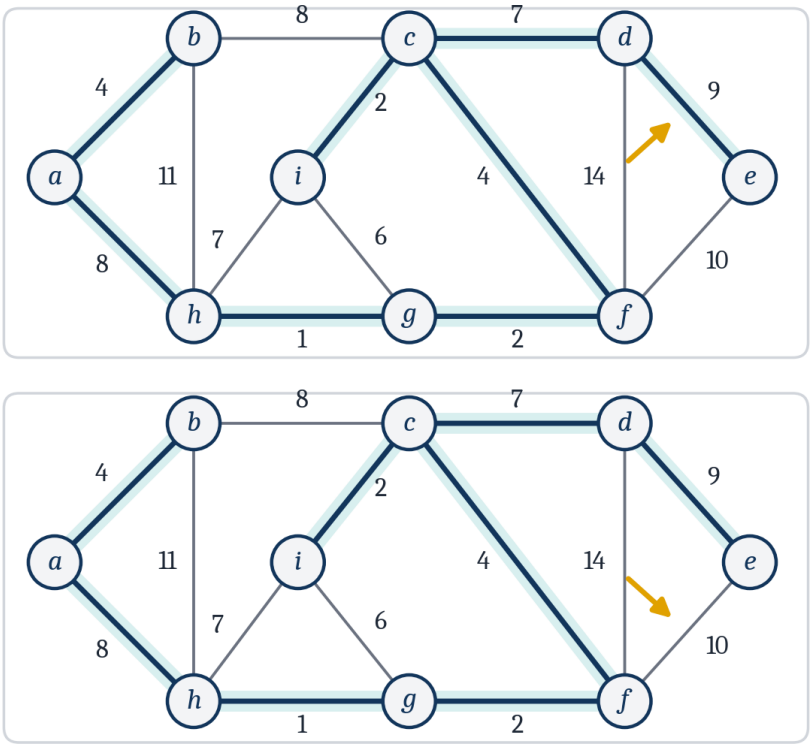


Gambar 11.23 Atas: (a, h) berbobot 8 diterima. **Bawah:** (b, c) berbobot 8 ditolak. **Inilah langkah saat jawaban Kruskal menyimpang dari jawaban Prim:** Prim mengambil (b, c) dan tidak pernah menimbang (a, h); Kruskal melakukan kebalikannya. Kedua pohonnya berbobot 37 (lihat §11.12).

Buku: Subbab 11.10.10 dan Gambar 11.23.

Kruskal, Sisi 11 dan 12

Gambar 11.24: (d, e) 9: terima — sisi kedelapan, total **37**. (e, f) 10: tolak.

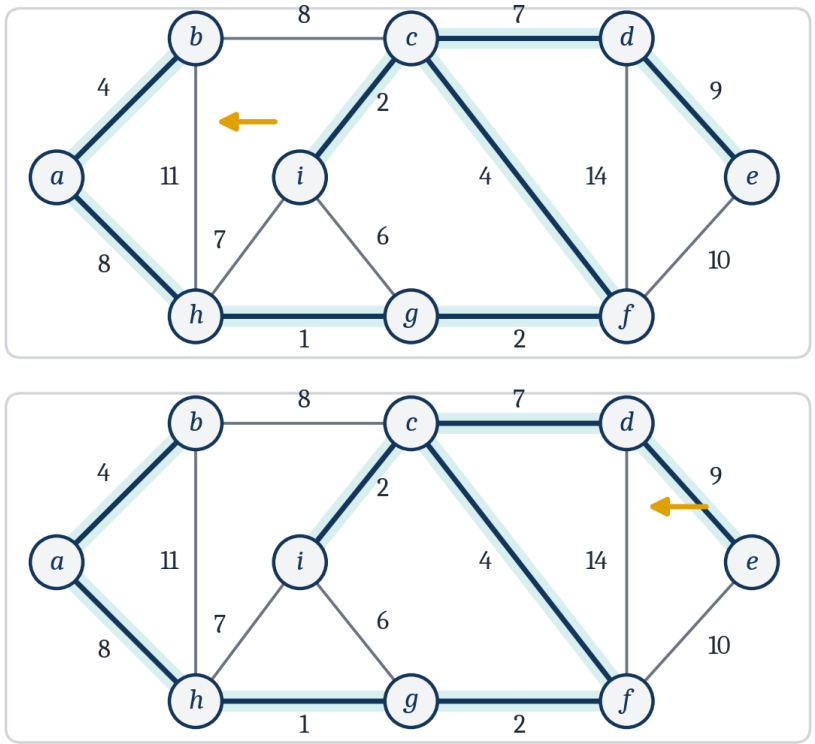


Gambar 11.24 Atas: (d, e) berbobot 9 diterima — sisi kedelapan sekaligus yang terakhir, yang membawa simpul terpencil e masuk ke pohonnya. **Bawah:** (e, f) berbobot 10 ditolak.

Buku: Subbab 11.10.10 dan Gambar 11.24.

Kruskal, Sisi 13 dan 14

Gambar 11.25: (b, h) 11 dan (d, f) 14: tolak. Implementasi praktis berhenti setelah $|V| - 1$ sisi diterima.



Gambar 11.25 Dua panel terakhir: (b, h) berbobot 11 dan (d, f) berbobot 14 sama-sama ditolak, dan kedelapan sisi yang tersorot adalah pohon merentang minimum yang sudah jadi, dengan bobot total 37 — bobot yang sama dengan jawaban Prim pada **Gambar 11.17**, lewat himpunan sisi yang berbeda.

Buku: Subbab 11.10.10 dan Gambar 11.25.

Contoh Terkerjakan 4: Kruskal dalam Satu Halaman

Tabel 11.7 Jalan lengkap Kruskal dari **Gambar 11.19** sampai **Gambar 11.25**. Empat belas sisi ditimbang, delapan diterima, enam ditolak, bobot total 37. Kolom `Find` menunjukkan wakil yang sungguh-sungguh dikembalikan hutan himpunan saling lepasnya; perhatikan bagaimana c menjadi wakil komponen yang tumbuh itu lalu tetap demikian.

#	Sisi	w	`Find` tiap ujungnya	Keputusan	Total	Hutan sesudahnya
1	(g, h)	1	g, h	terima	1	a,b,c,d,e,f,**gh**,i
2	(c, i)	2	c, i	terima	3	a,b,**ci**,d,e,f,gh
3	(f, g)	2	f, g	terima	5	a,b,ci,d,e,**fgh**
4	(a, b)	4	a, b	terima	9	**ab**,ci,d,e,fgh
5	(c, f)	4	c, g	terima	13	ab,**cfghi**,d,e
6	(g, i)	6	c, c	**tolak**	13	ab,cfghi,d,e
7	(c, d)	7	c, d	terima	20	ab,**cdfghi**,e
8	(h, i)	7	c, c	**tolak**	20	ab,cdfghi,e

Tabel 11.7 (Lanjutan) *Jalan lengkap Kruskal dari Gambar 11.19 sampai Gambar 11.25. Empat belas sisi ditimbang, delapan diterima, enam ditolak, bobot total 37. Kolom Find menunjukkan wakil yang sungguh-sungguh dikembalikan hutan himpunan saling lepasnya; perhatikan bagaimana c menjadi wakil komponen yang tumbuh itu lalu tetap demikian.*

#	Sisi	w	`Find` tiap ujungnya	Keputusan	Total	Hutan sesudahnya
9	(a, h)	8	a, c	terima	28	**abcdfghi**,e
10	(b, c)	8	c, c	**tolak**	28	abcdfghi,e
11	(d, e)	9	c, e	terima	37	**abcdefghi**
12	(e, f)	10	c, c	**tolak**	37	abcdefghi
13	(b, h)	11	c, c	**tolak**	37	abcdefghi
14	(d, f)	14	c, c	**tolak**	37	abcdefghi

Tabel 11.7: empat belas sisi, delapan diterima ($|V| - 1$), enam ditolak, total 37.

Buku: Subbab 11.10.10.

Mengapa Kruskal Benar, dan Ongkosnya

Benar: A selalu berupa hutan. Untuk (u, v) yang diterima, ambil potongan S = komponen u: ia menghormati A, (u, v) melintasinya, dan sisi lintas yang lebih murah pasti sudah ditimbang lebih dulu dan kini berada di dalam sebuah komponen. Jadi (u, v) ringan — **pengurutannya membayar dirinya sendiri.**

Ongkos Persamaan (11.9), didominasi pengurutan; dan menurut Persamaan (11.10), $\lg E = \Theta(\lg V)$, sehingga Kruskal dan Prim ber-*heap* berongkos sama secara asimtotik.

$$O(V) + \Theta(E \lg E) + O(E\,\alpha(V)) = O(E \lg E). \tag{11.9}$$

$$|V| - 1 \leq |E| \leq \binom{|V|}{2} < |V|^2. \tag{11.10}$$

Buku: Subbab 11.10.11 dan 11.10.12.

Program Kruskal

Kode 11.1 mencetak partisinya sebelum setiap keputusan; setiap baris cocok dengan **Tabel 11.7**, dan ia berakhir dengan satu komponen.

Kode 11.1 Algoritma Kruskal dengan hutan himpunan saling lepas yang benar, menelusuri partisinya sebelum tiap keputusan. Kompilasilah dengan `g++ -std=c++17 -O2 -o kruskal kruskal.cpp`.

```cpp
// Algoritma Kruskal dengan hutan himpunan saling lepas (gabung menurut peringkat +
// pemampatan lintasan), pada graf berbobot Subbab 11.9.5. Tiap sisi yang ditimbang
// dilaporkan, diterima atau ditolak, berikut komponennya sebelum keputusannya.
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <algorithm>
#include <map>
using namespace std;

class DisjointSets {
public:
    explicit DisjointSets(int n) : parent(n), rank_(n, 0) {
        for (int i = 0; i < n; ++i) parent[i] = i;
    }
    int findSet(int x) {                              // dengan pemampatan lintasan
        if (parent[x] != x) parent[x] = findSet(parent[x]);
        return parent[x];
    }
    bool unite(int x, int y) {                        // gabung menurut peringkat
        int rx = findSet(x), ry = findSet(y);
        if (rx == ry) return false;
        if (rank_[rx] < rank_[ry]) swap(rx, ry);
        parent[ry] = rx;
        if (rank_[rx] == rank_[ry]) ++rank_[rx];
        return true;
    }
```

```
private:
    vector<int> parent, rank_;
};

struct Edge {
    int u, v, w;
    bool operator<(const Edge& o) const { return w < o.w; }   // menurut bobot tak-menurun
};
```

Kode 11.1 (Lanjutan) *Algoritma Kruskal dengan hutan himpunan saling lepas yang benar, menelusuri partisinya sebelum tiap keputusan. Kompilasilah dengan g++ -std=c++17 -O2 -o kruskal kruskal.cpp.*

```
// Mencetak partisi saat ini sebagai himpunan blok, agar penelusurannya menunjukkan mengapa
// sebuah sisi diterima atau ditolak.
string partitionOf(DisjointSets& D, const vector<string>& name) {
    map<int, string> blocks;
    for (int v = 0; v < (int)name.size(); ++v) blocks[D.findSet(v)] += name[v];
    string s = "{";
    bool first = true;
    for (auto& kv : blocks) { if (!first) s += ","; s += kv.second; first = false; }
    return s + "}";
}

int main() {
    vector<string> name = { "a","b","c","d","e","f","g","h","i" };
    enum Vertex { Va, Vb, Vc, Vd, Ve, Vf, Vg, Vh, Vi };
    vector<Edge> edges = {
        {Va,Vb,4},  {Va,Vh,8},  {Vb,Vc,8},  {Vb,Vh,11}, {Vc,Vd,7},
        {Vc,Vf,4},  {Vc,Vi,2},  {Vd,Ve,9},  {Vd,Vf,14}, {Ve,Vf,10},
        {Vf,Vg,2},  {Vg,Vh,1},  {Vg,Vi,6},  {Vh,Vi,7}
    };
    const int n = (int)name.size();

    // Baris 04: urutkan sisi-sisi E menurut bobot tak-menurun.
    // stable_sort menjaga bobot yang sama pada urutan penulisannya di atas,
    // sehingga jalannya dapat diulang dan dapat dibandingkan dengan gambarnya.
    stable_sort(edges.begin(), edges.end());

    cout << "edges in sorted order:\n ";
    for (const Edge& e : edges) cout << " (" << name[e.u] << name[e.v] << ")" << e.w;
```

```
    cout << "\n\n";

    DisjointSets D(n);
    vector<Edge> mst;                                   // hutan yang sedang dibangun
    int total = 0, step = 0;
    for (const Edge& e : edges) {
        cout << "forest " << partitionOf(D, name) << "\n";
        int ru = D.findSet(e.u), rv = D.findSet(e.v);
        bool accept = (ru != rv);
        cout << "  edge " << setw(2) << ++step << ": (" << name[e.u] << "," << name[e.v]
             << ") w=" << setw(2) << e.w
             << "  Find(" << name[e.u] << ")=" << name[ru]
             << " Find(" << name[e.v] << ")=" << name[rv]
             << (accept ? "  DIFFERENT -> accept" : "  SAME -> reject");
        if (accept) { D.unite(e.u, e.v); mst.push_back(e); total += e.w;
                      cout << ", total=" << total; }
        cout << "\n";
    }
```

Kode 11.1 (Lanjutan) *Algoritma Kruskal dengan hutan himpunan saling lepas yang benar, menelusuri partisinya sebelum tiap keputusan. Kompilasilah dengan g++ -std=c++17 -O2 -o kruskal kruskal.cpp.*

```
    cout << "\nMST edges, in the order Kruskal accepted them:\n";
    for (const Edge& e : mst)
        cout << "  (" << name[e.u] << "," << name[e.v] << ") w=" << e.w << "\n";
    cout << "  edge count = " << mst.size() << "  (|V|-1 = " << n - 1 << ")\n";
    cout << "  total weight = " << total << "\n";
    cout << "  components at the end = "
         << [&]{ int c = 0; for (int v = 0; v < n; ++v) if (D.findSet(v) == v) ++c; return c; }()
         << "\n";
    return 0;
}
```

Buku: Subbab 11.10.13.

Prim lawan Kruskal

Bobot sama, 37; pohon berbeda — tujuh sisi sama, (b, c) milik Prim lawan (a, h) milik Kruskal, keduanya berbobot 8.

Terukur: pada 1000 graf acak keduanya tak pernah berbeda bobot; graf jarang (200 000 simpul) Kruskal 26,3 ms lawan Prim 104,9 ms; graf padat (2000 simpul, 10^6 sisi) Prim 39,9 ms lawan Kruskal 58,7 ms. **Tabel 11.8** adalah tabel keputusannya.

Tabel 11.8 Memilih di antara kedua algoritmanya. Dua baris pertamanya meliputi sebagian besar kasus nyata; selebihnya adalah kasus-kasus yang jawabannya tidak gamblang.

Bila…	Pakai	Sebab
E dekat ke V (jarang: jaringan jalan, jaring renggang)	**Kruskal**	pengurutannya atas sedikit butir dan ramah-*cache*
E dekat ke $\|V\|^2$ (padat: matriks jarak semua pasangan)	**Prim**, tumpukan biner	ia tidak pernah mengurutkan banyak sisi yang tak akan dipakainya
$\|E\| = \Theta(V^2)$ *dan* V kecil	**Prim**, larik tak terurut	$O(V^2)$ dengan konstanta mungil, dan sepele untuk ditulis
sisi-sisinya sudah datang terurut, atau bobotnya bilangan bulat kecil	**Kruskal**	buang pengurutannya; sisanya $O(E\,\alpha(V))$, nyaris linear
grafnya diberikan sebagai matriks ketetanggaan	**Prim**, larik	memindai satu baris matriks *adalah* `ExtractMin`-nya
grafnya tidak muat di memori dan datang sebagai aliran sisi	**Kruskal**	ia satu sapuan atas sisi terurut; pengurutan eksternal persoalan yang sudah selesai
Anda harus menangani graf **tak terhubung** dan menginginkan *hutan* merentang minimum	**Kruskal**	ia sama sekali tidak perlu diubah; ia cukup berhenti sebelum $\|V\| - 1$ sisi
Anda memerlukan pohonnya berakar di simpul tertentu, lengkap dengan induknya	**Prim**	π adalah pohon berakarnya, cuma-cuma
$\|E\| = \Theta(V \lg V)$ atau lebih padat dan V berjuta-juta	**Prim**, *Fibonacci heap*	$O(V \lg V + E)$ adalah batas terbaik yang diketahui

Buku: Subbab 11.11.

Bobot Kembar, dan Kapan MST Tunggal

Teorema 11.8: bobot yang berbeda-beda menjamin MST tunggal. Kebalikannya salah (segitiga berbobot 1, 1, 5).

Kriteria: T satu-satunya MST jika dan hanya jika menghapus sisi pohon mana pun membuat pohon merentang terbaiknya benar-benar lebih berat. **Tabel 11.9** menjalankannya: hanya (a, h) yang dapat dibuang tanpa ongkos — tepat dua MST.

Tabel 11.9 Uji ketunggalan §11.12.3 diterapkan pada graf contoh sembilan simpul, dihitung dengan memanggil Kruskal dari **Kode 11.1** sekali untuk tiap sisi pohonnya dengan sisi itu dihapus. Tepat satu barisnya berbobot sama, sehingga MST-nya tidak tunggal — dan pohon alternatif yang dihasilkan baris ketujuhnya adalah $\{(a,b)4,(b,c)8,(c,d)7,(c,f)4,(c,i)2,(d,e)9,(f,g)2,(g,h)1\}$, yang persis merupakan jawaban Prim dari **Gambar 11.17**.

Sisi yang dihapus	Bobot terbaik tanpanya	Kesimpulan
(g,h) 1	43	lebih berat — ada di setiap MST
(c,i) 2	41	lebih berat — ada di setiap MST
(f,g) 2	41	lebih berat — ada di setiap MST
(a,b) 4	41	lebih berat — ada di setiap MST
(c,f) 4	39	lebih berat — ada di setiap MST
(c,d) 7	40	lebih berat — ada di setiap MST
(a,h) 8	**37**	**sama** — ada MST lain tanpanya
(d,e) 9	38	lebih berat — ada di setiap MST

UNTUK RANGKAIAN UJI. Jangan pernah menegaskan himpunan sisi tertentu; tegaskanlah $|V|-1$ sisi, keterhubungan, dan **bobot** acuannya.

Buku: Subbab 11.12, Teorema 11.8.

Profitina dalam Praktik

Alur pemindaian adalah DAG: diperiksa siklusnya saat mulai, diurutkan dengan **Kahn**, yang himpunan siapnya adalah antrean kerja paralel yang alami.

Halaman nyaris kembar: MST atas jarak antarhalaman, dipotong di atas sebuah ambang — pengelompokan *single-linkage*.

Profitina Laundry, pertanyaan 1: sepuluh cabang, delapan belas calon tautan; sembilan dipasang, **7,9 km** lawan 22,7 km (hemat 65,2%); Prim setuju.

Pertanyaan 2: sembilan tahap diurutkan secara topologis; lintasan kritis intake → sort → wash → dry → press → fold → handover memakan **108 menit**, sehingga pengering yang lebih cepat membantu, sedangkan timbangan kedua tidak.

Buku: Subbab 11.13 dan Subbab 11.13.3. Aplikasi memberi saran; pengguna yang memutuskan.

Kesalahan yang Sering Terjadi

1. MST dikira pohon lintasan terpendek. 2. `key[u] + w(u,v)` pada Prim. 3. Melupakan uji `v in Q`.

4. Tidak mengurutkan (atau mengurutkan menurun) pada Kruskal. 5. Larik biasa untuk uji komponen. 6. Hanya salah satu dari gabung menurut peringkat dan pemampatan lintasan.

7. Menegaskan MST tertentu. 8. Menempel di belakang pada pengurutan topologis DFS. 9. Mencatat pada waktu penemuan. 10. Mengira urutan topologisnya tunggal.

11. Uji DAG pada graf tak berarah. 12. Prim pada graf tak terhubung. 13. Jumlah bobot meluap pada 32 bit — pakailah `long long`.

Buku: Subbab 11.14.

Ringkasan: Lima Hal untuk Diingat

1. Graf berarah asiklik jika dan hanya jika DFS tak menemukan sisi balik; urutan topologis = f menurun, atau derajat masuk Kahn — keduanya $\Theta(V + E)$, jarang tunggal.

2. Pohon merentang bersisi $|V| - 1$; MST punya substruktur optimal (potong-dan-tempel).

3. Sifat potongan: sisi ringan pada potongan mana pun yang menghormati A adalah aman — setiap algoritma MST adalah cara memilih potongannya.

4. Prim menumbuhkan satu pohon dengan key dan π, $O(E \lg V)$; **Kruskal** mengurutkan lalu menggabung dengan hutan himpunan saling lepas, $O(E \lg E)$, operasi himpunannya $O(\alpha(n))$.

5. Bobotnya selalu sama, pohonnya sama hanya bila MST-nya tunggal; bobot yang berbeda-beda sudah cukup, tetapi tidak perlu.

Buku: Subbab 11.15.

Pertemuan Berikutnya

Bab 12 — Evaluasi dan Pembahasan III. Tanpa teori baru: satu proyek perancangan dikerjakan dari awal sampai akhir, dan dua puluh tiga persoalan terkerjakan lanjutan atas Bab 9, 10, dan 11.

Setiap antrean dicetak, setiap cap waktu dicatat, setiap penggabungan direkam — dan satu varian *uji diri* sesudah setiap persoalan.

Sebelum pertemuan: kerjakan Soal Latihan 11.1, 11.5, 11.6, dan 11.16, lalu baca ulang sifat potongan (Subbab 11.7).

Buku: Subbab 11.15.1 dan Bab 12.

Bab 1

Kuliah 12 — Evaluasi dan Pembahasan III: Algoritma Greedy dan Graf

Perancangan dan Analisis Algoritma (EF234405)

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*

Dosen: Irfan Subakti — yifana@gmail.com

Buku: Bab 12, Evaluasi dan Pembahasan III: Algoritma Greedy dan Graf. Setiap halaman handout ini diakhiri dengan penunjuk ke buku, yang menjadi rujukan utama.

Pertemuan Hari Ini

1. Tanpa teori baru. Bab 9, 10, dan 11 dimantapkan dengan mengerjakan persoalan, setiap nilai antara dituliskan (Subbab 12.1).

2. Satu proyek perancangan, dikerjakan dari awal sampai akhir — Rute Laundry: persoalan, model, pilihan algoritma, pseudokode, kompleksitas, bukti dengan tangan, implementasi, evaluasi, kesimpulan (Subbab 12.2).

3. Persoalan Terkerjakan A–F tentang algoritma greedy dan G–L tentang menelusuri graf (Subbab 12.3 dan 12.4).

4. Persoalan Terkerjakan M–Q tentang DAG dan pengurutan topologis, serta R–W tentang pohon merentang minimum (Subbab 12.5 dan 12.6).

5. Latihan uji diri singkat, Profitina dalam praktik, kesalahan yang sering terjadi, dan ringkasannya (Subbab 12.7–12.10). Setiap persoalan diakhiri varian *uji diri*.

Buku: Bab 12, Subbab 12.1 sampai 12.10.

Posisi Kita Saat Ini

Bab 9: algoritma greedy, argumen penukaran, pemilihan aktivitas, kedua *knapsack*, kode Huffman, matroid.

Bab 10: kosakata dan penyimpanan graf; BFS dalam cincin-cincin jarak; DFS dengan cap waktu dan empat golongan sisi; komponen, siklus, kebipartitan.

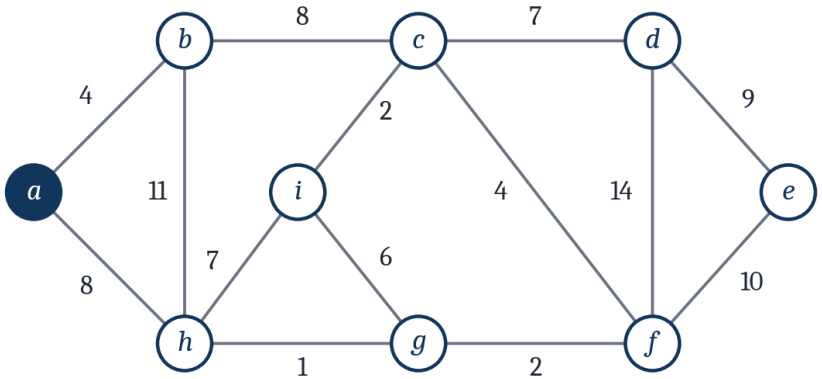
Bab 11: DAG dan teorema DAG; pengurutan topologis menurut waktu selesai dan menurut Kahn; sifat potongan; Prim–Jarník dan Kruskal dengan hutan himpunan saling lepas.

> **CARA MEMAKAI PERTEMUAN INI.** Bacalah tiap persoalan, tutup penyelesaiannya, dan cobalah sendiri selama sepuluh menit sebelum melanjutkan. Menyebut alatnya sudah separuh pekerjaan — peta pengulangan di Subbab 12.1.2 menyebutkannya untuk setiap gagasan.

Buku: Subbab 12.1.1–12.1.3.

Graf Tolok Ukur

Gambar 12.1: MST-nya berbobot **37**, dan ada dua pohon berbeda dengan bobot itu — pemeriksaan tiga puluh detik bagi implementasi Prim atau Kruskal mana pun.



Gambar 12.1 Graf berbobot dengan sembilan simpul dan empat belas sisi yang dipakai untuk setiap penelusuran pohon merentang minimum di Bab 11. Pohon merentang minimumnya berbobot 37, dan ada dua pohon berbeda dengan bobot itu — dan itulah sebabnya §11.12 ada.

Buku: Subbab 12.1.2 dan Gambar 12.1.

Proyek Perancangan dalam Sembilan Tahap

Tabel 12.1 Kesembilan tahap sebuah proyek perancangan, pertanyaan yang dijawab masing-masing tahap, dan bukti yang dihasilkan masing-masingnya. Tahap 1 sampai 6 berlangsung sebelum ada kode apa pun.

Tahap	Pertanyaan yang dijawabnya	Apa yang harus dihasilkannya
1. Persoalan	*Apa persisnya yang ditanyakan?*	Pertanyaan-pertanyaannya, dengan kata-kata sederhana, beserta masukan dan keluarannya yang disebut namanya
2. Model	*Objek matematis apakah ini?*	Datanya dinyatakan sebagai graf, himpunan, atau barisan, dengan setiap lambang didefinisikan
3. Pemilihan algoritma	*Algoritma yang mana, dan mengapa bukan yang lain?*	Algoritmanya disebut namanya, dan sekurang-kurangnya satu alternatif ditolak beserta alasannya
4. Pseudokode	*Apa yang akan dikerjakan program, baris demi baris?*	Prosedur bernomor yang cukup tepat untuk ditelusuri dengan tangan
5. Kompleksitas	*Berapa ongkosnya, sebelum kita menjalankannya?*	Waktu jalan setiap fase, diramalkan lebih dahulu
6. Pembuktian dengan tangan	*Apakah rancangannya berhasil pada data yang sesungguhnya?*	Algoritmanya ditelusuri dengan tangan pada masukan yang sebenarnya, dengan setiap langkah dituliskan
7. Implementasi	*Apakah ia berjalan, dan apakah ia algoritma yang kita rancang?*	Program lengkap yang keluarannya dapat dibandingkan dengan penelusuran tangan
8. Evaluasi	*Bagaimana kita tahu ia benar, dan bagaimana kita tahu ia cepat?*	Bukti kebenaran pada banyak masukan, dan waktu terukur yang dibandingkan dengan ramalannya

Tahap	Pertanyaan yang dijawabnya	Apa yang harus dihasilkannya
9. Kesimpulan	*Apa yang kita pelajari, dan apa berikutnya?*	Angka-angka utamanya, satu keterbatasan nyata, satu langkah berikutnya yang spesifik

Tabel 12.1: **enam dari sembilan tahap berlangsung sebelum ada kode**, dan tahap 8 tidak boleh dilewati.

> **UJI PEMIKUL BEBAN.** *Seandainya masukannya membesar seribu kali lipat, apakah metode naifnya masih berhasil?* Jika ya, algoritmanya hiasan. Sprite dalam labirin 5×5 gagal dalam uji ini; rute penjemputan di jalan-jalan sebuah kota (2^E himpunan bagian) lolos.

Buku: Subbab 12.2.1 dan 12.2.2.

Tahap 1 dan 2: Persoalan dan Model

Rute Laundry, gerai di Sukolilo. **P1:** zona mana yang dapat dijanjikan penjemputan dua lompatan, dan mana yang tak terjangkau? **P2:** jalan mana yang membentuk jaringan antar-jemput termurah? **P3:** dalam urutan apa tahap-tahapnya boleh berjalan, dan secepat apa sebuah kantong dapat diselesaikan?

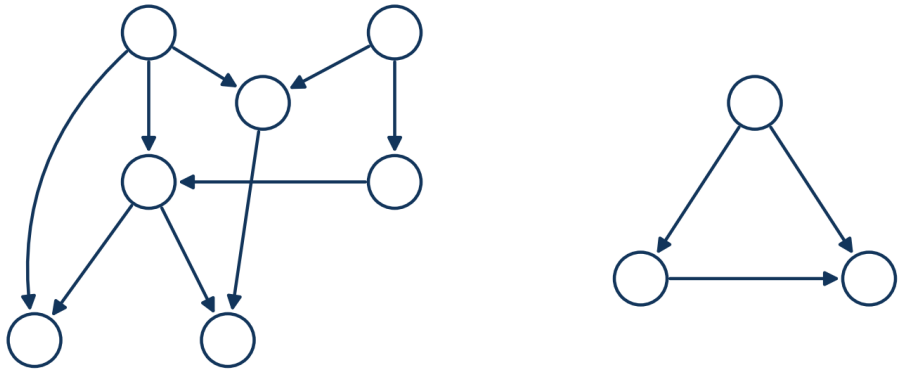
Model: **graf tak berarah** berisi 14 zona dan 17 jalan pada **Tabel 12.2** (tak berbobot untuk P1, berbobot untuk P2); **graf berarah** berisi 10 tahap untuk P3, yang harus asiklik (**Gambar 12.2**).

Tabel 12.2 Ketujuh belas jalan pada contoh Rute Laundry, dengan jarak dalam ratusan meter. Empat belas zona; enam belas jalan terletak di dalam satu komponen dan yang ketujuh belas, Bulak–Kenjeran, adalah pulau tersendiri.

Jalan	*w*	Jalan	*w*	Jalan	*w*
Sukolilo–Menur	8	Nginden–Manyar	6	Mulyosari–Kalijudan	5
Sukolilo–Keputih	6	Klampis–Manyar	3	Kalijudan–DukuhSetro	8
Menur–Gebang	7	Klampis–Mulyosari	13	Manyar–DukuhSetro	9
Menur–Semolowaru	9	Keputih–Kejawan	12	Kejawan–Mulyosari	7
Keputih–Gebang	5	Gebang–Klampis	10	Bulak–Kenjeran	4
Semolowaru–Nginden	4	Semolowaru–Klampis	11		

Buku: Subbab 12.2.3 dan 12.2.4.

Asiklik atau Tidak



Gambar 12.2 Dua graf berarah taksiklik. Di kiri, ikutilah anak panahnya dari mana pun dan Anda akan kehabisan anak panah. Di kanan, tiga anak panah membentuk segitiga yang tetap taksiklik, sebab arahnya tidak berputar searah mengelilinginya; balikkan anak panah bawahnya dan ketiganya akan menutup sebuah siklus berarah, dan spesifikasi alur kerja yang berbentuk seperti itu tidak dapat dijalankan dalam urutan apa pun.

Buku: Subbab 12.2.4 dan Gambar 12.2.

Tahap 3: Algoritma Dipilih, dan yang Ditolak

P1: BFS dari gerai, $\Theta(V + E)$. *Ditolak:* Dijkstra (antrean prioritas untuk bobot satuan) dan Floyd–Warshall semua-pasangan (satu sumber, bukan V).

P2: Kruskal di atas hutan himpunan saling lepas — datanya tiba sebagai **daftar sisi**. *Ditolak:* menghitung semua pohon merentang, sebab menurut rumus Cayley Persamaan (12.1) K_{12} saja punya $12^{10} = 61\,917\,364\,224$; dan Prim, sama benarnya tetapi perlu struktur ketetanggaan lebih dulu.

$$\tau(K_n) = n^{n-2}, \tag{12.1}$$

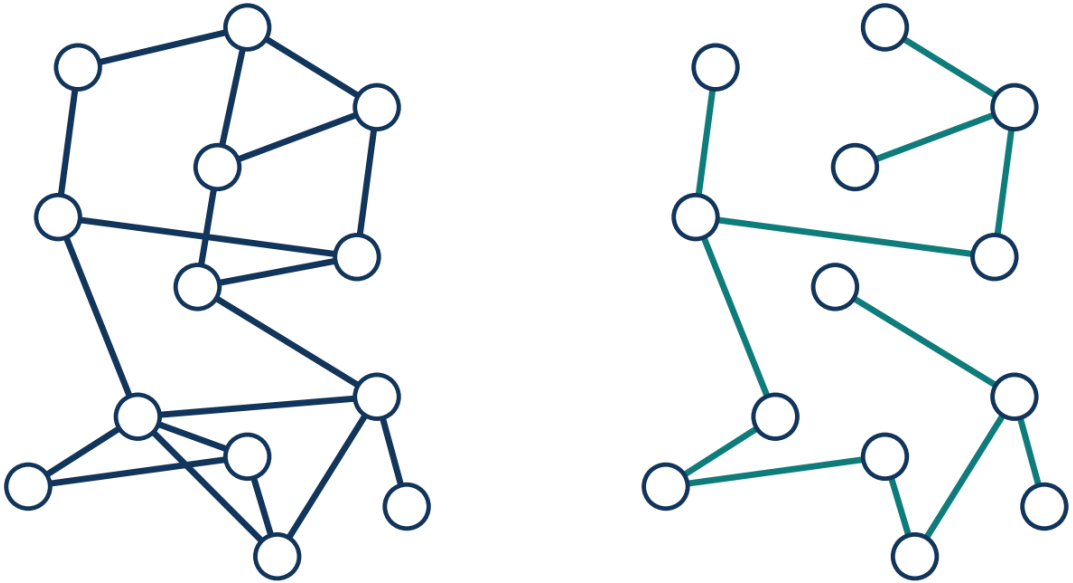
P3: Kahn, yang mendeteksi spesifikasi bersiklus secara gratis, dan dalam lintasan yang sama lintasan kritisnya menurut Persamaan (12.2): sebuah tahap selesai sepanjang durasinya sendiri setelah prasyarat terakhirnya selesai.

$$ef[v] = dur[v] + \max(\{0\} \cup \{\, ef[u] : u \to v \,\}). \tag{12.2}$$

Buku: Subbab 12.2.5.

Tulang Punggungnya Adalah Pohon Merentang

Gambar 12.3: setiap simpul hadir, tetap terhubung, tanpa siklus, $|V| - 1$ sisi — tulang punggung antar-jemput P2, dipilih seringan mungkin.



Gambar 12.3 Sebuah pohon merentang. **Kiri:** sebuah graf. **Kanan:** sebuah pohon merentangnya — semua simpul hadir, masih terhubung, tanpa siklus; tiga belas simpul dan dua belas sisi, yaitu aturan $|V| - 1$ pada §11.4.2. Tulang punggung antar-jemput pertanyaan 2 persis benda ini, dipilih seringan mungkin.

Buku: Subbab 12.2.5 dan Gambar 12.3.

Tahap 4 dan 5: Pseudokode dan Ongkos

Pseudokode 12.1 Rancangan Rute Laundry: pencarian melebar dan komponen untuk wilayah layanan, Kruskal di atas hutan himpunan saling lepas untuk tulang punggungnya, dan algoritma Kahn beserta lintasan kritisnya untuk alur kerjanya.

```
RUTE-LAUNDRY(zones, roads, stages, precedences, outlet)
 1  d, pi  = BFS(zones, roads, outlet)            // pertanyaan 1
 2  comp   = COMPONENTS(zones, roads)             // keterjangkauan
 3  sort roads by weight, ascending               // pertanyaan 2
 4  MAKE-SET(z) for every zone z in comp[outlet]
 5  T = empty set
 6  for each road (u, v, w) in weight order
 7      if FIND-SET(u) != FIND-SET(v)
 8          T = T + {(u, v)};  UNION(u, v)
 9  order, ef = KAHN(stages, precedences)         // pertanyaan 3
10  if |order| < |stages| then report a cyclic specification
11  return d, pi, comp, T, order, max(ef)
```

Pseudokode 12.1 adalah seluruh rancangannya dalam sebelas baris. Diramalkan sebelum ada kode: daftar ketetanggaan, BFS, dan komponen $\Theta(n+m)$; pengurutan $\Theta(m \lg m)$; m `Find` dan $n-1$ `Union` $O(m\,\alpha(n))$; Kahn $\Theta(s+p)$ — totalnya Persamaan (12.3), dan **pengurutanlah leher botolnya**.

$$T(n, m, s, p) = \Theta(m \lg m + n + s + p), \tag{12.3}$$

Buku: Subbab 12.2.6 dan 12.2.7.

Tahap 6: P1 Dibuktikan dengan Tangan

Tabel 12.3 Cincin-cincin melebar di sekeliling gerai Sukolilo, dihitung dengan tangan. Enam zona terletak di dalam janji dua lompatan. Dua zona tak terjangkau, dan itu harus dilaporkan programnya sebagai fakta usaha alih-alih sebagai galat.

d	Zona
0	Sukolilo
1	Menur, Keputih
2	Gebang, Semolowaru, Kejawan
3	Nginden, Klampis, Mulyosari
4	Manyar, Kalijudan
5	DukuhSetro
∞	Bulak, Kenjeran

Tabel 12.3: BFS dari Sukolilo, satu cincin setiap kali. Di Keputih, Gebang sudah kelabu — **sentuhan pertama menang**. Dua belas zona dikeluarkan; Bulak dan Kenjeran tak pernah tersentuh.

Enam zona berada dalam janji dua lompatan; dua **tak terjangkau** — fakta bisnis (perlu gerai kedua), bukan galat.

Buku: Subbab 12.2.8.

Tahap 6: P2 dan P3 Dibuktikan dengan Tangan

Tabel 12.4 Kruskal pada jalan-jalan Rute Laundry, dengan tangan. Sebelas sisi diterima untuk dua belas zona — persis $|V| - 1$ — dengan total 69 ratus meter, atau 6,9 km. Lima jalan ditolak sebab kedua ujungnya sudah tersambung.

#	Jalan	w	Vonis	Total berjalan
1	Klampis–Manyar	3	terima	3
2	Semolowaru–Nginden	4	terima	7
3	Keputih–Gebang	5	terima	12
4	Mulyosari–Kalijudan	5	terima	17
5	Sukolilo–Keputih	6	terima	23
6	Nginden–Manyar	6	terima	29
7	Menur–Gebang	7	terima	36
8	Kejawan–Mulyosari	7	terima	43
9	Sukolilo–Menur	8	**tolak** — kedua ujungnya sudah dalam satu pohon	43
10	Kalijudan–DukuhSetro	8	terima	51
11	Menur–Semolowaru	9	terima	60
12	Manyar–DukuhSetro	9	terima	69

#	Jalan	*w*	Vonis	Total berjalan
13–16	Gebang–Klampis 10, Semolowaru–Klampis 11, Keputih–Kejawan 12, Klampis–Mulyosari 13		semuanya ditolak	69

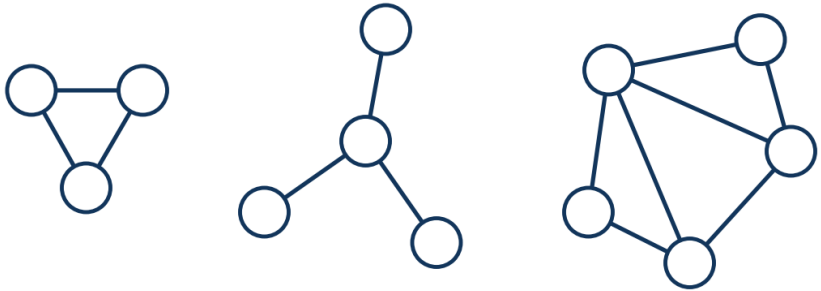
Tabel 12.4: sebelas jalan untuk dua belas zona, **69** ratus meter; keenam belas jalan akan berongkos 123, jadi tulang punggungnya menghemat $54/123 = 43{,}9\%$. Baris 9 ditolak pada bobot 8 sementara baris 10, juga 8, diterima — partisinyalah yang memutuskan.

P3: Kahn mengeluarkan intake, weigh, sort, invoice, pretreat, wash, dry, press, fold, pack — sepuluh dari sepuluh, jadi asiklik. Menurut Persamaan (12.2): pack selesai pada $\max(115{,}9) + 6 = 121$; lintasan kritis intake → weigh → sort → pretreat → wash → dry → press → fold → pack, **121 menit**; invoice punya kelonggaran $115 - 9 = 106$ menit.

Buku: Subbab 12.2.8.

Tahap 7: Implementasinya

Program lengkapnya cocok dengan penelusuran tangan baris demi baris, termasuk kedua nilai tak hingga, dan melaporkan **dua komponen**; yang kedua, Bulak dengan Kenjeran, adalah alasan janji itu tak boleh diiklankan untuk seluruh kota (**Gambar 12.4**).



Gambar 12.4 Graf dengan tiga komponen terhubung: sebuah segitiga, sebuah bintang, dan sepotong bersimpul lima. Tak ada sisi yang melintas di antaranya. Jaringan jalan Rute Laundry punya dua potongan semacam itu, dan yang kedua — Bulak dengan Kenjeran — adalah sebabnya janji dua lompatan tidak boleh diiklankan untuk seluruh kota.

Buku: Subbab 12.2.9 dan Gambar 12.4. `find` *memampatkan dalam perjalanan pulang; sisi yang ditolak tetap mengubah hutannya;* `emitted 10 of 10` *adalah sertifikat keasiklikan.*

Tahap 8: Evaluasinya

```
=== Test 1: Kruskal and Prim must agree, on 2000 random graphs ===
mismatches: 0 out of 2000

=== Test 2: the BFS postcondition, on 2000 random graphs ===
edges spanning more than one BFS level: 0

=== Test 3: does it scale as the analysis says? ===
    zones     roads      BFS ms  Kruskal ms   BFS/(V+E) us Kruskal/(ElgE) us
     2000      8000       0.084       0.446         0.0047            0.0043
     4000     16000       0.158       0.921         0.0044            0.0041
     8000     32000       0.355       1.771         0.0049            0.0037
    16000     64000       0.742       3.731         0.0052            0.0037
    32000    128000       1.886       7.174         0.0065            0.0033
```

Uji 1 adalah pemeriksaan silang (dua program yang salah bisa saja sepakat). **Uji 2** adalah pascasyarat, bukti yang lebih baik: untuk setiap sisi, Persamaan (12.4). **Uji 3** membagi waktunya dengan ramalannya: Kruskal datar, BFS naik 38% — meleset dari *cache*, bukan analisis yang rusak.

$$|\, d[u] - d[v] \,| \leq 1. \tag{12.4}$$

Buku: Subbab 12.2.10.

Tahap 9: Kesimpulannya

> **KESIMPULAN, EMPAT KALIMAT.** Tiga persoalan graf pada satu himpunan data, diselesaikan dengan BFS, Kruskal, dan Kahn dengan ongkos ramalan $\Theta(m \lg m)$. Enam zona dalam dua lompatan, dua tak terjangkau, tulang punggung 11 jalan sepanjang 6,9 km lawan 12,3 km (43,9% lebih pendek), lintasan kritis 121 menit dengan kelonggaran 106 menit. Keterbatasan: lompatan bukanlah waktu tempuh. Langkah berikutnya: Dijkstra (Bab 13) dengan perangkat evaluasi yang sama.

Lima jebakan: algoritma hiasan; rancangan yang ditulis belakangan; evaluasi yang tidak ada; pengukuran yang tidak dibaca; masukan degenerate yang tidak diperiksa.

Uji diri. Sukolilo–Menur turun menjadi 5: ia masuk, Menur–Gebang (7) keluar, tulang punggungnya $69 - 7 + 5 = 67$. Invoice memakan 110 menit: waktu penyelesaian tetap 121, tetapi kini ada dua lintasan kritis dan tanpa kelonggaran.

Buku: Subbab 12.2.11 dan 12.2.12.

Persoalan A: Uang Kembalian

Pseudokode 12.2 Aturan rakus penukaran uang: berulang-ulang serahkan koin terbesar yang tidak melebihi sisa jumlah yang masih terutang.

```
GREEDY-CHANGE(coins, x)          // koin terurut sehingga c_1 > c_2 > ... > c_k = 1
1  count = 0
2  for i = 1 to k
3      while x >= c_i
4          x = x - c_i
5          count = count + 1
6  return count
```

Token $\{1,7,10\}$: **Pseudokode 12.2** membayar 14 sebagai $10+1+1+1+1$ — lima token; $7+7$ cukup dua. Minimum sejatinya diperoleh dari rekurensi Persamaan (12.5).

$$M(0)=0, \qquad M(x)=1+\min_{\substack{c\in\text{koin}\\ c\le x}} M(x-c) \quad (x\ge 1). \tag{12.5}$$

Kerakusan kalah tepat pada 14, 15, 16, 24, 25, 26 sampai 30. **Bahan yang hilang:** sifat pilihan greedy — satu-satunya pembayaran optimal untuk 14 tidak memuat 10.

Untuk {1,5,10} **greedy optimal:** pembayaran optimal memuat paling banyak empat token 1 dan paling banyak satu token 5, jadi nilai 5 dan 1 paling banyak 9, memaksa $\lfloor x/10 \rfloor$ token 10 — tepat pilihan greedy. **Uji diri:** {1,5,8} pertama kali gagal pada 10 (8 + 1 + 1 lawan 5 + 5).

Buku: Subbab 12.3.2.

Persoalan A: Programnya

Kode 12.1 mencetak hasil greedy di samping optimum pemrograman dinamis untuk setiap nilai dari 1 sampai 30, lalu mengaudit {1,5,8} (47 kekalahan pada 1..100) dan {1,5,10,25} (tanpa kekalahan).

Kode 12.1 Penukaran uang secara rakus dibandingkan dengan optimum pemrograman dinamis, untuk setiap jumlah dari 1 sampai 30 dengan token {1,7,10}, disusul dua audit ringkasan. Kompilasilah dengan `g++ -std=c++17 -O2 -Wall -o change change.cpp`.

```cpp
// Persoalan Terkerjakan A: menukar uang dengan koin {1, 7, 10}.
// Untuk setiap jumlah dari 1 sampai 30 program ini membandingkan aturan greedy
// ("koin terbesar yang muat") dengan optimum sejati hasil pemrograman dinamis,
// mencetak kedua multihimpunan koinnya, dan melaporkan jumlah terkecil tempat greedy kalah.
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <climits>
using namespace std;

string showCoins(const vector<int>& c) {
    string s;
    for (size_t i = 0; i < c.size(); ++i) { if (i) s += "+"; s += to_string(c[i]); }
    return s;
}

vector<int> greedy(const vector<int>& coins, int x) {      // koin dalam urutan menaik
    vector<int> used;
    for (int k = (int)coins.size() - 1; k >= 0; --k)
        while (x >= coins[k]) { x -= coins[k]; used.push_back(coins[k]); }
    return used;
}

// best[x] = koin tersedikit untuk membayar x; last[x] = satu koin dari pembayaran optimal
void optimal(const vector<int>& coins, int maxX, vector<int>& best, vector<int>& last) {
    best.assign(maxX + 1, INT_MAX); last.assign(maxX + 1, 0);
    best[0] = 0;
```

```
    for (int x = 1; x <= maxX; ++x)
        for (int c : coins)
            if (c <= x && best[x - c] != INT_MAX && best[x - c] + 1 < best[x]) {
                best[x] = best[x - c] + 1; last[x] = c;
            }
}

vector<int> rebuild(const vector<int>& last, int x) {
    vector<int> used;
    while (x > 0) { used.push_back(last[x]); x -= last[x]; }
    return used;
}
```

Kode 12.1 (Lanjutan) *Penukaran uang secara rakus dibandingkan dengan optimum pemrograman dinamis, untuk setiap jumlah dari 1 sampai 30 dengan token \{1, 7, 10\}, disusul dua audit ringkasan. Kompilasilah dengan g++ -std=c++17 -O2 -Wall -o change change.cpp.*

```
void audit(const vector<int>& coins, int maxX, bool table) {
    vector<int> best, last;
    optimal(coins, maxX, best, last);
    cout << "coins {";
    for (size_t i = 0; i < coins.size(); ++i) cout << (i ? ", " : "") << coins[i];
    cout << "}, amounts 1.." << maxX << "\n";
    if (table) cout << "  x  greedy              #g  optimum             #o\n";
    int firstBad = 0, bad = 0;
    for (int x = 1; x <= maxX; ++x) {
        vector<int> g = greedy(coins, x), o = rebuild(last, x);
        bool lose = g.size() > o.size();
        if (lose) { ++bad; if (!firstBad) firstBad = x; }
        if (table)
            cout << setw(3) << x << "  " << left << setw(18) << showCoins(g) << right
                 << setw(4) << g.size() << "  " << left << setw(18) << showCoins(o)
                 << right << setw(4) << o.size() << (lose ? "   <- greed loses" : "") << "\n";
    }
    cout << "  amounts on which greed loses: " << bad << " of " << maxX;
    if (firstBad) cout << "; the smallest is " << firstBad << "\n";
    else cout << "; greed is optimal on every amount tested\n";
}

int main() {
    audit({1, 7, 10}, 30, true);
    cout << "\n";
    audit({1, 5, 8}, 100, false);                          // sistem untuk bagian uji diri
    audit({1, 5, 10, 25}, 100, false);                     // sebuah sistem kanonis
    return 0;
```

```
}
```

Buku: Subbab 12.3.2, tempat keluaran lengkapnya dicetak. Kozen dan Zaks: kegagalan pertama terletak di bawah jumlah dua koin terbesar.

Persoalan B: Pemilihan Aktivitas

Tabel 12.5 Kesepuluh pekerjaan pengering pada Persoalan Terkerjakan B, didaftar menurut urutan waktu selesai yang tidak menurun.

i	1	2	3	4	5	6	7	8	9	10
s_i	0	2	1	5	4	8	7	10	9	12
f_i	3	4	6	7	9	11	12	13	14	16

Pseudokode 12.3 Aturan waktu-selesai-terawal untuk pemilihan aktivitas, sebagai satu sapuan atas pekerjaan yang sudah terurut menurut waktu selesai.

```
GREEDY-ACTIVITY-SELECTOR(s, f, n)   // pekerjaan terurut sehingga f_1 <= f_2 <= ... <= f_n
1  A = {a_1}
2  F = f_1
3  for i = 2 to n
4      if s_i >= F
5          A = A + {a_i}
6          F = f_i
7  return A
```

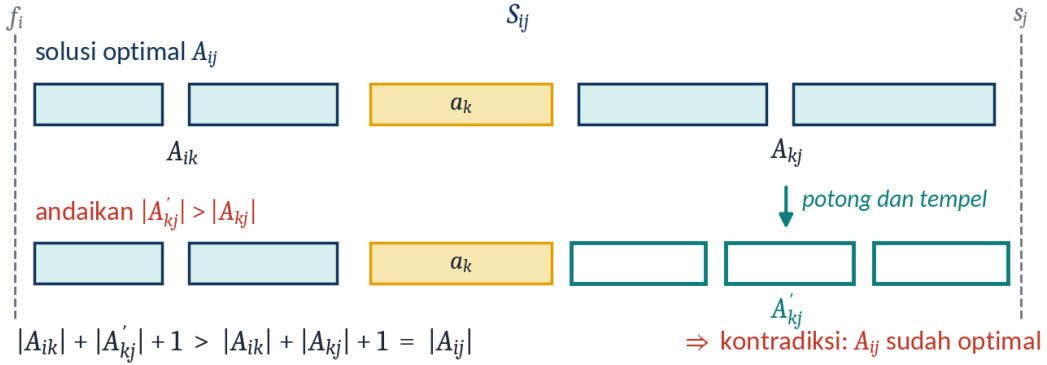
Tabel 12.5 dengan **Pseudokode 12.3**, F mulai dari 0: ambil a_1 ($F = 3$); lewati a_2, a_3; ambil a_4 ($F = 7$); lewati a_5; ambil a_6 ($F = 11$); lewati a_7, a_8, a_9; ambil a_{10}. $\{a_1, a_4, a_6, a_{10}\}$**, ukuran 4.** Mulai-terawal memberi $\{a_1, a_5, a_9\}$, ukuran 3.

Buku: Subbab 12.3.3.

Persoalan B: Penukarannya

Ambil **sembarang** A_k maksimum dengan aktivitas yang pertama selesai $a_j \neq a_m$, lalu bentuk Persamaan (12.6); setiap pekerjaan lain mulai paling cepat $f_j \geq f_m$, sehingga tetap selaras dan ukurannya tak berubah. **Gambar 12.5** menggambarkan kembaran potong-dan-tempelnya.

$$A'_k = (A_k \setminus \{a_j\}) \cup \{a_m\}. \tag{12.6}$$



rekurensi:

$$c[i,j] = \begin{cases} 0 & \text{jika } S_{ij} = \varnothing \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{jika } S_{ij} \neq \varnothing \end{cases}$$

Gambar 12.5 Teorema pemilihan aktivitas dirampungkan lewat kontradiksi, bersama rekurensi pemrograman dinamis untuk $c[i,j]$ yang dihasilkan separuh *optimal substructure* argumennya. Penyulihan yang digambar di atasnya — himpunan A'_{kj} yang lebih besar ditempelkan menggantikan A_{kj} — adalah kembaran potong-dan-tempel dari pertukaran pada Persamaan (12.6): di sana satu pekerjaan ditukar dengan yang rakus dan ukurannya tetap; di sini potongan yang lebih besar akan memperbesar keseluruhannya, dan keoptimalan melarang hal itu.

Buku: Subbab 12.3.3 dan Gambar 12.5. Tulislah "suatu himpunan bagian berukuran maksimum": $\{a_2, a_4, a_6, a_{10}\}$ juga optimal.

Persoalan B: Uji Diri

Mulai-terakhir-dahulu pada pekerjaan pengering memberi $\{a_2, a_4, a_6, a_{10}\}$ — optimal, dengan mencerminkan waktu. Pada sebelas aktivitas **Gambar 12.6**, selesai-terawal memberi $\{a_1, a_4, a_8, a_{11}\}$.

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Gambar 12.6 Kesebelas aktivitas §9.4.2 beserta waktu mulai dan waktu selesainya, sudah terurut sehingga $f_1 \leq f_2 \leq \cdots \leq f_{11}$.

Buku: Subbab 12.3.3 dan Gambar 12.6.

Persoalan C: Keterlambatan Maksimum

Enam pekerjaan, $p = \langle 3,2,1,4,3,2 \rangle$, $d = \langle 6,8,9,9,14,15 \rangle$; minimumkan Persamaan (12.7).

$$L_{\max} = \max_{1 \le i \le n} (C_i - d_i) \;\rightarrow\; \min. \tag{12.7}$$

Tabel 12.6 Tenggat terawal dahulu pada keenam pekerjaan. Hanya J_4 yang terlambat, sebesar satu satuan, jadi $L_{\max} = 1$.

Urutan	Pekerjaan	p_i	C_i	d_i	$L_i = C_i - d_i$
1	J_1	3	3	6	-3
2	J_2	2	5	8	-3
3	J_3	1	6	9	-3
4	J_4	4	10	9	$+1$
5	J_5	3	13	14	-1
6	J_6	2	15	15	0

Tabel 12.6: tenggat-terawal-dahulu memberi $L_{\max} = 1$; waktu-proses-terpendek-dahulu memberi 6. **Bukti:** menukar pasangan bersebelahan yang terbalik ($d_i > d_j$) tak dapat menaikkan $L_{\max}$, menurut Persamaan (12.8); urutkan jadwal optimal mana pun ala *bubble sort* ke urutan tenggat.

$$L'_i = t + p_i + p_j - d_i \ < \ t + p_i + p_j - d_j = L_j. \tag{12.8}$$

Buku: Subbab 12.3.4. Gaya kasar: optimumnya 1, dicapai oleh 20 dari 720 urutan.

Persoalan D: Sebuah Kode Huffman

Pseudokode 12.4 Algoritma Huffman: $n - 1$ penggabungan atas kedua pohon terjarang, tiap penggabungan menciptakan satu simpul dalam.

```
HUFFMAN(C)                  // C: simbol-simbolnya, masing-masing dengan frekuensi c.freq
1  n = |C|
2  Q = C                    // antrean prioritas-min berkunci freq
3  for i = 1 to n - 1
4      allocate a new node z
5      x = EXTRACT-MIN(Q)
6      y = EXTRACT-MIN(Q)
7      z.left = x;  z.right = y
8      z.freq = x.freq + y.freq
9      INSERT(Q, z)
10 return EXTRACT-MIN(Q)    // akar pohonnya
```

Tabel 12.7 Keenam penggabungan algoritma Huffman pada frekuensi lognya. Enam penggabungan untuk tujuh simbol, sebagaimana dituntut $|C| - 1$. Label simpul terakhirnya, 100, adalah frekuensi totalnya — sebuah pemeriksaan yang berguna.

Gabung	Antrean sebelumnya (terurut)	Dibuang	Simpul baru
1	1, 4, 8, 12, 15, 22, 38	Q 1, X 4	5
2	5, 8, 12, 15, 22, 38	5, P 8	13
3	12, 13, 15, 22, 38	F 12, 13	25
4	15, 22, 25, 38	D 15, W 22	37
5	25, 37, 38	25, 37	62
6	38, 62	R 38, 62	100

R 38, W 22, D 15, F 12, P 8, X 4, Q 1 (ribuan). Panjang tetap: $\lceil \lg 7 \rceil = 3$ bit, 300 kbit. **Pseudokode 12.4** melakukan enam penggabungan, **Tabel 12.7**: 5, 13, 25, 37, 62, 100.

Buku: Subbab 12.3.5.

Persoalan D: Buku Kode dan $B(T)$

Tabel 12.8 Buku kode Huffman untuk lognya, beserta ongkos tiap simbol. Kedua simbol terjarang, Q dan X, memperoleh kedua kata kode terpanjang dan hanya berbeda pada bit terakhirnya — keduanya bersaudara, dan itulah yang dilakukan penggabungan pertama terhadap keduanya.

Simbol	Frek (k)	Kata kode	Panjang	Frek × panjang
R	38	`0`	1	38
W	22	`111`	3	66
D	15	`110`	3	45
F	12	`100`	3	36
P	8	`1011`	4	32
X	4	`10101`	5	20
Q	1	`10100`	5	5
total	**100**			**242**

Tabel 12.8; atas daun, Persamaan (12.9) memberi 242 kbit; atas simpul dalam, Persamaan (12.10) memberi hasil yang sama. Hemat $58/300 = 19{,}33\%$.

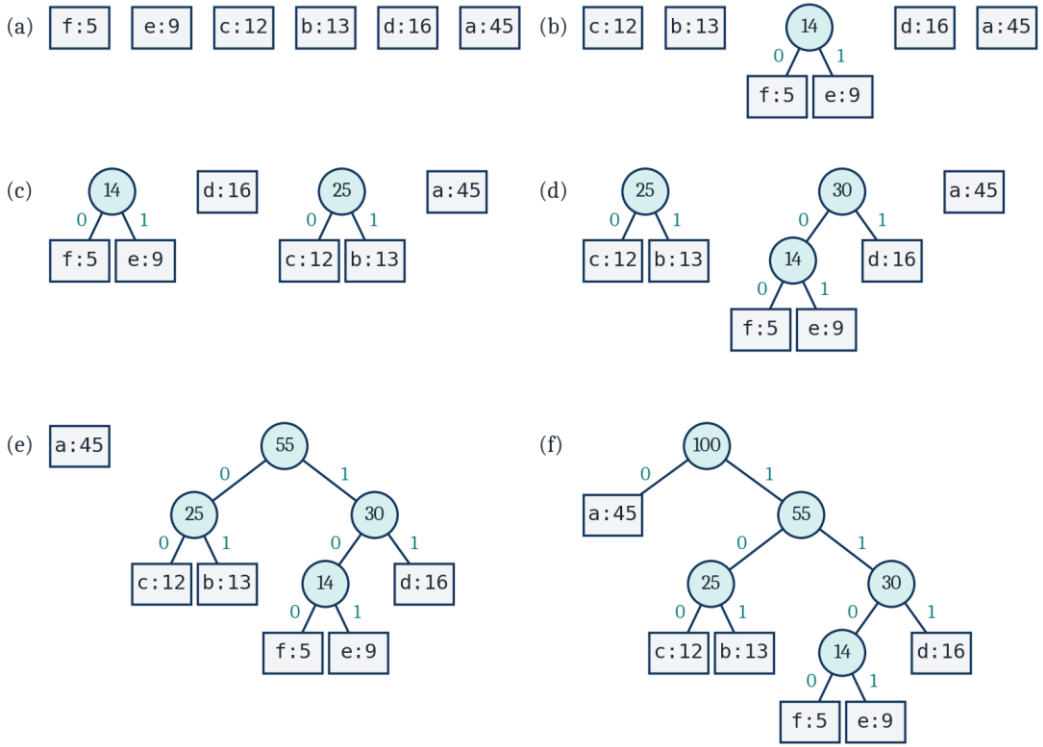
$$B(T) = \sum_{c \in C} c\,.\,freq \cdot d_T(c), \tag{12.9}$$

$$B(T) = \sum_{z \text{ dalam}} z\,.\,freq = 5 + 13 + 25 + 37 + 62 + 100 = 242 \text{ kbit}. \tag{12.10}$$

Buku: Subbab 12.3.5. Uji diri: `10110100111110` *terurai menjadi PRFWD.*

Proses yang Sama, Digambar

Gambar 12.7 adalah urutan penggabungan pada berkas enam lambang di Subbab 9.7.



Gambar 12.7 Penelusuran enam panel atas algoritma Huffman pada data bersimbol enam §9.7. Tiap panel memperlihatkan antreannya terurut menaik menurut frekuensi; pada tiap langkah kedua pohon berfrekuensi terendah digabungkan, daunnya digambar sebagai persegi panjang dan simpul dalamnya sebagai lingkaran.

Buku: Subbab 12.3.5 dan Gambar 12.7.

Panjang Tetap lawan Panjang Berubah

Gambar 12.8: perhitungan yang sama pada enam lambang — 300 000 lawan 224 000 bit. Huffman optimal di antara kode bebas awalan *per lambang*; pengodean aritmetik dapat lebih baik.

	a	b	c	d	e	f
Frekuensi (dalam ribuan)	45	13	12	16	9	5
Kata kode panjang tetap	000	001	010	011	100	101
Kata kode panjang berubah	0	101	100	111	1101	1100

Gambar 12.8 Persoalan pengodean karakter §9.7: berkas berisi 100.000 karakter atas abjad a–f, dengan kode panjang-tetap berongkos 300.000 bit dan kode panjang-berubah berongkos 224.000. Bangun argumennya persis seperti Persoalan Terkerjakan D; hanya abjadnya yang berganti.

Buku: Subbab 12.3.5 dan Gambar 12.8.

Persoalan E: Kedua *Knapsack*

Tabel 12.9 Keempat barang pada Persoalan Terkerjakan E.

Barang	1	2	3	4
nilai v_i	7	6	9	9
berat w_i	3	3	5	5

Tabel 12.9, 10 kg. Kerapatan 2,333; 2,000; 1,800; 1,800. **Pecahan:** barang 1 dan 2 utuh, 0,8 barang 3: **20,2**. **Greedy 0-1:** barang 1 dan 2 — **13**, 4 kg terbuang.

Optimum 0-1 menurut Persamaan (12.11), diisikan ke **Tabel 12.10**: **18**, barang 3 dan 4. Greedy kehilangan $5/18 = 27{,}8\%$ — persoalan 0-1 tak punya sifat pilihan greedy.

$$c[i,w] = \begin{cases} 0 & i = 0 \text{ atau } w = 0, \\ c[i-1,w] & w_i > w, \\ \max(c[i-1,w],\ v_i + c[i-1,\ w-w_i]) & \text{selainnya.} \end{cases} \tag{12.11}$$

Tabel 12.10 Tabel *knapsack* 0-1 $c[i, w]$ untuk Persoalan Terkerjakan E. Lima puluh lima sel, masing-masing satu perbandingan tunggal, jadi seluruh tabelnya berongkos $\Theta(nW) = \Theta(4 \times 10)$. Jawabannya adalah pojok kanan bawahnya, 18.

$w=$	0	1	2	3	4	5	6	7	8	9	10
$i = 0$	0	0	0	0	0	0	0	0	0	0	0
$i = 1$	0	0	0	7	7	7	7	7	7	7	7
$i = 2$	0	0	0	7	7	7	13	13	13	13	13
$i = 3$	0	0	0	7	7	9	13	13	16	16	16
$i = 4$	0	0	0	7	7	9	13	13	16	16	**18**

Buku: Subbab 12.3.6. Uji diri (8 kg): pecahan 16,6, greedy 0-1 13, optimum 16.

Persoalan F: Matroid yang Bekerja

Tujuh pesanan setrika sejam-an, tenggat $\langle 2,4,1,3,2,4,6 \rangle$, pengembalian dana $\langle 55,45,40,35,30,20,10 \rangle$. Uji kebebasan Persamaan (12.12); sifat turun-temurun dan penukaran berlaku (dengan $A = \{a_1, a_3\}$, $B = \{a_3, a_4, a_5\}$, a_4 dapat ditambahkan, a_5 tidak).

$$A \text{ bebas} \Leftrightarrow N_t(A) \leq t \quad \text{untuk } t = 0,1,\ldots,n. \tag{12.12}$$

Pseudokode 12.5 Algoritma rakus matroid untuk tugas berwaktu satuan dengan tenggat dan penalti: pertahankan pesanan termahal tetap tepat waktu kapan pun kebebasannya mengizinkan.

```
GREEDY-SCHEDULE(a, d, w, n)
1  sort the orders so that w_1 >= w_2 >= ... >= w_n
2  A = empty set
3  for i = 1 to n
4      if A + {a_i} satisfies N_t <= t for every t
5          A = A + {a_i}
6  schedule A in order of deadline, then the other orders in any order
7  return the schedule, and the refund sum of the orders not in A
```

Tabel 12.11 Algoritma rakus matroid pada ketujuh pesanan. Cacah yang tercetak tebal melanggar aturan $N_t \leq t$. Dua pesanan ditolak, dan pengembalian uang yang dibayar adalah jumlah keduanya, $30 + 20 = 50$.

Timbang	N_1	N_2	N_3	N_4	N_5	N_6	N_7	Vonis	A Sesudahnya
a_1 (55)	0	1	1	1	1	1	1	pertahankan	$\{a_1\}$
a_2 (45)	0	1	1	2	2	2	2	pertahankan	$\{a_1, a_2\}$
a_3 (40)	1	2	2	3	3	3	3	pertahankan	$\{a_1, a_2, a_3\}$
a_4 (35)	1	2	3	4	4	4	4	pertahankan	$\{a_1, a_2, a_3, a_4\}$
a_5 (30)	1	**3**	**4**	**5**	5	5	5	**tolak**	tak berubah
a_6 (20)	1	2	3	**5**	5	5	5	**tolak**	tak berubah
a_7 (10)	1	2	3	4	4	5	5	pertahankan	$\{a_1, a_2, a_3, a_4, a_7\}$

Pseudokode 12.5, ditelusuri pada **Tabel 12.11**: pertahankan a_1–a_4 dan a_7; tolak a_5 dan a_6. Setrika a_3, a_1, a_4, a_2, a_7; pengembalian dana **50** dari 235 — dikonfirmasi atas semua 128 himpunan bagian.

Buku: Subbab 12.3.7.

Contoh Kasus Subbab 9.8.7

Gambar 12.9 adalah contoh kasus tujuh tugas yang lain; ia juga membayar 50. **Uji diri:** pindahkan tenggat a_5 ke 5 — hanya a_6 yang terlambat, pengembalian dananya 20.

Tugas

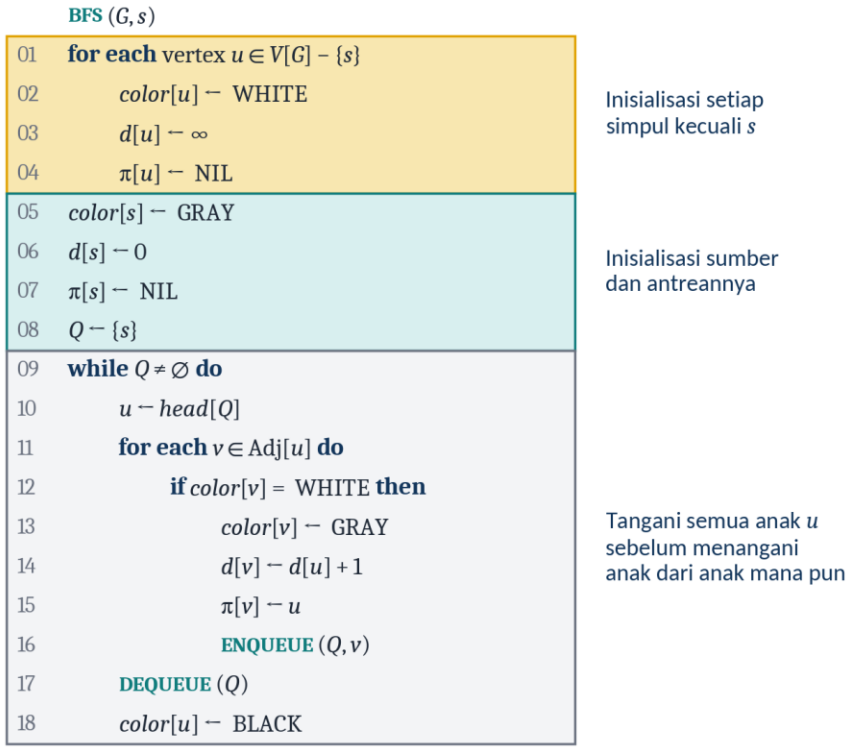
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Gambar 12.9 Contoh §9.8.7 untuk penjadwalan tugas berwaktu satuan dengan tenggat dan penalti: tujuh tugas dengan tenggat d_i dan penalti w_i. Algoritma rakus matroid menjadwalkan lebih awal sebuah himpunan berbobot 230 dan membayar penalti 50.

Buku: Subbab 12.3.7 dan Gambar 12.9; Persoalan B sampai E sebagai satu program ada di Subbab 12.3.8.

Persoalan G: BFS, Ditelusuri

Jalankan **Gambar 12.10** persis — termasuk sisi-sisi menuju simpul kelabu, yang harus diperiksa lalu diabaikan.



Gambar 12.10 Pencarian melebar. Blok ambernya menyiapkan setiap simpul selain sumbernya; blok toskanya menyiapkan sumbernya dan antreannya; blok abu-abunya adalah pencariannya. Catatan di sebelah kanan menyatakan invariannya dengan kata-kata — *tangani semua anak u sebelum anak mana pun dari anak-anaknya* — dan kalimat itulah seluruh bedanya dari pencarian mendalam.

Buku: Subbab 12.4.2 dan Gambar 12.10.

Persoalan G: Penelusuran pada G_1

G_1: A–I, sisi $AB, AC, BD, BE, CE, CF, DG, EG, EH, FH, GI, HI$; daftar menurut abjad. Derajatnya berjumlah 24, sebagaimana dituntut Persamaan (12.13).

$$\sum_{v \in V} \deg(v) = 2\,|E|, \tag{12.13}$$

Tabel 12.12 Pencarian melebar atas G_1 dari A, satu baris per putaran gelung **while**. Sembilan simpul, sembilan langkah: setiap simpul diantrekan sekali dan dikeluarkan sekali. Jarak di dalam antreannya tak pernah menurun dari kiri ke kanan dan tak pernah merentang lebih dari satu nilai — yaitu invarian antrean §10.7.9.

Langkah	Keluarkan	Ditemukan, dengan d	Dilewati, dan sebabnya	Antrean sesudahnya
1	A	B: 1, C: 1	—	B: 1 C: 1
2	B	D: 2, E: 2	A hitam	C: 1 D: 2 E: 2
3	C	F: 2	A hitam; E **kelabu**	D: 2 E: 2 F: 2
4	D	G: 3	B hitam	E: 2 F: 2 G: 3
5	E	H: 3	B, C hitam; G **kelabu**	F: 2 G: 3 H: 3
6	F	—	C hitam; H kelabu	G: 3 H: 3
7	G	I: 4	D, E hitam	H: 3 I: 4
8	H	—	E, F hitam; I kelabu	I: 4
9	I	—	G, H hitam	(kosong)

Tabel 12.13 Keluaran pencarian melebar atas G_1: jarak dan pendahulu. Membaca π mundur dari simpul mana pun lalu membaliknya memberikan sebuah lintasan terpendek, dan panjangnya selalu sama dengan d simpul itu.

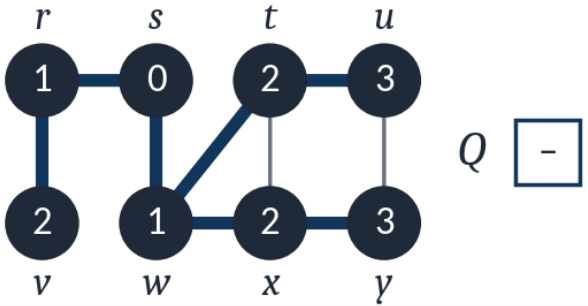
v	A	B	C	D	E	F	G	H	I
$d[v]$	0	1	1	2	2	2	3	3	4
$\pi[v]$	–	A	A	B	B	C	D	E	G

Tabel 12.12 dan **Tabel 12.13**: lintasan terpendek ke I adalah $A\ B\ D\ G\ I$, panjang 4; delapan sisi pohon, $|V| - 1$. Dari E setiap simpul berada dalam dua sisi.

Buku: Subbab 12.4.2.

Penelusuran yang Tuntas

Gambar 12.11: setiap simpul hitam, antrean kosong, $|V| - 1$ sisi pohon yang tebal.



Gambar 12.11 Pencarian melebar yang sudah selesai: setiap simpul hitam, antreannya kosong, tiap simpul membawa jaraknya dari s, dan ketujuh sisi tebalnya membentuk pohon melebar atas delapan simpul.

Buku: Subbab 12.4.2 dan Gambar 12.11. Jarak tidak bergantung pada urutan daftar; pohonnya bergantung.

Persoalan H: DFS pada Digraf

G_2: $A \to B$, $A \to C$, $A \to D$, $B \to D$, $C \to D$, $C \to E$, $D \to A$, $E \to F$, $F \to E$, $G \to C$, $G \to F$. Terapkan **Gambar 12.12** sebelas kali.

```
DFS (G)
1  for each vertex u ∈ V[G]
2       do color[u] ← WHITE
3  time ← 0
4  for each vertex u ∈ V[G]
5       do if color[u] = WHITE
6            then DFS-VISIT (u)

DFS-VISIT (u)
1  color[u] ← GRAY          ▷ Simpul putih u ditemukan.
2  d[u] ← time              ▷ Catat waktu penemuan.
3  time ← time + 1          ▷ Majukan jam global.
4  for each v ∈ Adj[u]      ▷ Jelajahi semua sisi (u, v).
5       do if color[v] = WHITE
6            then DFS-VISIT (v)
7  color[u] ← BLACK         ▷ Hitamkan u; ia selesai.
8  f[u] ← time              ▷ Catat waktu selesai.
9  time ← time + 1          ▷ Majukan jam global.
```

Gambar 12.12 Pencarian mendalam. Prosedur luarnya memutihkan setiap simpul, memulai jamnya dari nol, dan memanggil DFS-Visit pada setiap simpul yang masih putih — tiap panggilan semacam itu mengakari satu pohon hutannya. Prosedur dalamnya mengelabui u, membubuhkan $d[u]$, berekursi ke setiap tetangga putih, lalu menghitamkan u dan membubuhkan $f[u]$.

Buku: Subbab 12.4.3 dan Gambar 12.12.

Persoalan H: Cap Waktu dan Golongan

Tabel 12.14 Cap waktu dan pendahulu sesudah pencarian mendalam atas G_2. Jamnya berhenti di $14 = 2|V|$, dan keempat belas capnya adalah bilangan bulat 1 sampai 14, masing-masing terpakai sekali — itulah pemeriksaan pertama yang harus dijalankan pada jawaban DFS mana pun.

v	A	B	C	D	E	F	G
d/f	1/12	2/5	6/11	3/4	7/10	8/9	13/14
π	–	A	A	B	C	E	–

Tabel 12.15 Setiap busur G_2 digolongkan, beserta uji yang menentukannya. Lima pohon, dua mundur, satu maju, tiga silang — sebelas seluruhnya. Dua sisi mundur berarti grafnya tidak taksiklik.

Busur	Warna ujung jauhnya saat diperiksa	Golongan
$A \to B$	putih	pohon
$A \to C$	putih	pohon
$A \to D$	hitam, $d[A] < d[D]$	**maju**
$B \to D$	putih	pohon
$C \to D$	hitam, $d[C] > d[D]$	**silang**
$C \to E$	putih	pohon
$D \to A$	**kelabu**	**mundur**
$E \to F$	putih	pohon

Tabel 12.15 (Lanjutan) *Setiap busur G2 digolongkan, beserta uji yang menentukannya. Lima pohon, dua mundur, satu maju, tiga silang — sebelas seluruhnya. Dua sisi mundur berarti grafnya tidak taksiklik.*

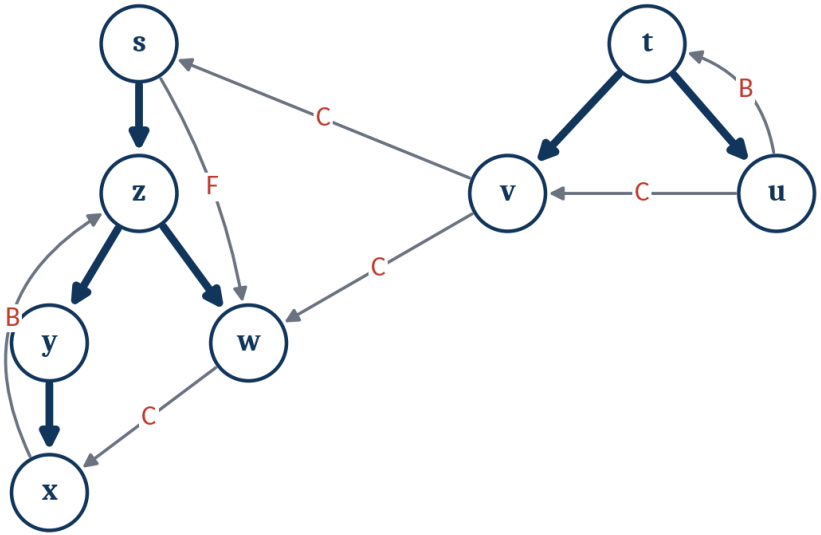
Busur	Warna ujung jauhnya saat diperiksa	Golongan
$F \to E$	**kelabu**	**mundur**
$G \to C$	hitam, $d[G] > d[C]$	silang
$G \to F$	hitam, $d[G] > d[F]$	silang

Tabel 12.14 dan **Tabel 12.15**: **kelabu berarti mundur, putih berarti pohon, hitam terbelah menjadi maju atau silang menurut waktu penemuan.** Dua sisi mundur — $A \to B \to D \to A$ dan $E \rightleftarrows F$ — jadi tidak asiklik.

Buku: Subbab 12.4.3.

Mundur, Maju, Silang — Terlihat

Gambar 12.13: sisi mundur melengkung ke atas, sisi maju menunjuk ke bawah melewati satu tingkat, sisi silang berjalan menyamping.

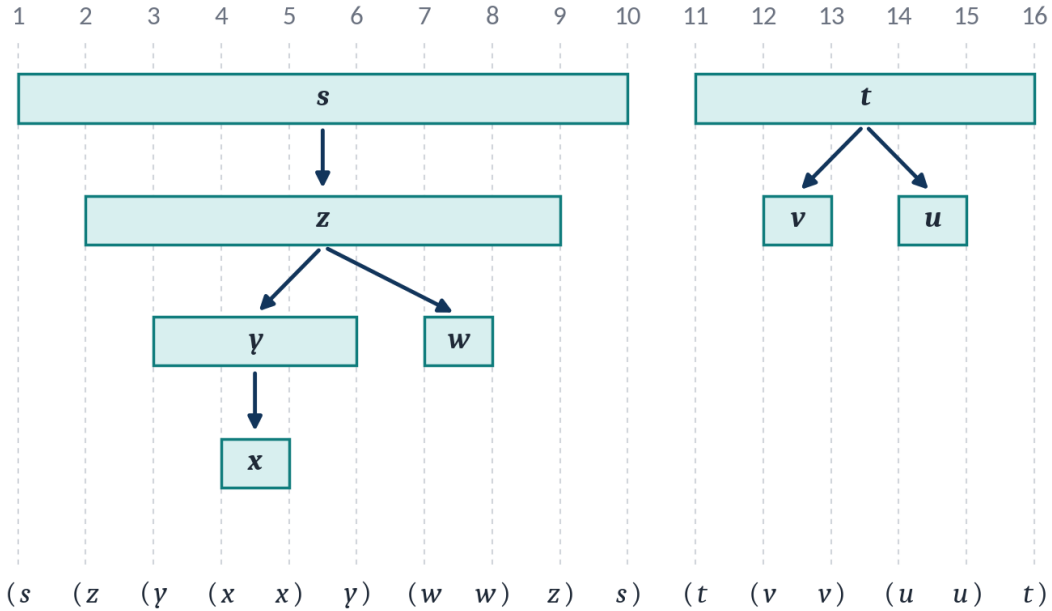


Gambar 12.13 Sebuah contoh yang lebih besar digambar ulang sebagai hutan mendalamnya, sisi pohonnya tebal dan menurun. Kini penggolongannya menjadi geometris: **B** melengkung kembali ke atas menuju leluhur, **F** menunjuk ke bawah melewati satu tingkat antara menuju keturunan, dan **C** berjalan menyamping di antara cabang atau di antara pohon.

Buku: Subbab 12.4.3 dan Gambar 12.13.

Struktur Tanda Kurung

G_2: $(A\ (B\ (D\ D)\ B)\ (C\ (E\ (F\ F)\ E)\ C)\ A)\ (G\ G)$. B adalah [2,5], C adalah [6,11]: saling lepas, jadi tak berkerabat. **Gambar 12.14** memperlihatkan batang-batangnya.

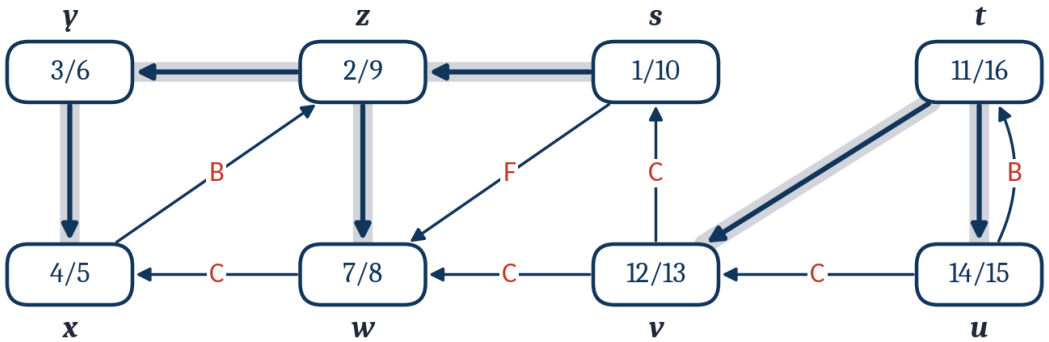


Gambar 12.14 Struktur kurung. Di atas, hutan mendalamnya dengan sisi pohon menunjuk ke bawah; di bawah, jamnya dari 1 sampai 16 dengan satu batang per simpul yang merentang $[d, f]$, dan pengurungannya ditulis di sepanjang dasarnya. Setiap pasangan batang tersarang atau lepas — tak pernah tumpang tindih sebagian.

Buku: Subbab 12.4.3 dan Gambar 12.14. Uji diri: dengan $\text{Adj}[A] = [D, C, B]$ *sisi majunya lenyap dan muncul sisi silang keempat.*

Bentuk yang Patut Ditiru

Gambar 12.15: satu kotak per simpul berisi d/f, sisi pohon tebal, **B**, **F**, **C** pada sisi lainnya — tuliskan jawaban Anda sendiri dengan cara ini.



Gambar 12.15 Pencarian mendalam bersimpul delapan, bercap waktu: s adalah 1/10, z adalah 2/9, y adalah 3/6, x adalah 4/5, w adalah 7/8, t adalah 11/16, v adalah 12/13, u adalah 14/15. Satu gambar membawa cap waktunya, hutannya, dan penggolongan sisinya selengkapnya sekaligus — bentuk yang layak ditiru ketika Anda menuliskan sebuah pencarian sendiri.

Buku: Subbab 12.4.3 dan Gambar 12.15.

Persoalan I: DFS, Tak Berarah

Setiap sisi berada di dua daftar. Di u dengan induk p: putih → pohon; kelabu dan bukan p → **mundur**; hitam → sudah digolongkan dari ujung lainnya.

Tabel 12.16 Pencarian mendalam atas graf tak berarah G_1 dari A. Jamnya berakhir di $18 = 2|V|$. Selang-selangnya tersarang sempurna, sebab pohon mendalamnya di sini satu lintasan tunggal yang melewati kesembilan simpul.

v	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*
d/f	1/18	2/17	6/13	3/16	5/14	7/12	4/15	8/11	9/10
π	–	*A*	*E*	*B*	*G*	*C*	*D*	*F*	*H*

Tabel 12.16: pohon $AB, BD, DG, GE, EC, CF, FH, HI$; mundur EB, CA, HE, IG. Sisi maju atau silang tak mungkin muncul, dan empat sisi mundur cocok dengan empat siklus bebas, $12 - 9 + 1$.

Buku: Subbab 12.4.4. Uji diri: dari E sisi mundurnya CE, HE, DB, GE — tetap empat.

Persoalan J: Komponen

G_3 pada 1..12: segitiga {1,2,3}, bintang {4,5,6,7}, siklus lima {8, … ,12}. Satu BFS per simpul tak berlabel, $\Theta(V + E)$ untuk seluruh larik pada **Tabel 12.17**.

Tabel 12.17 Label komponen setiap simpul G_3. Tiga komponen. Begitu larik ini ada, keterhubungan menjadi satu perbandingan bilangan bulat tunggal, dan itulah sebabnya lariknya — dan bukan pencariannya — yang menjadi jawabannya.

Simpul	1	2	3	4	5	6	7	8	9	10	11	12
komponen	0	0	0	1	1	1	1	2	2	2	2	2

Pertanyaan menjadi perbandingan: 1–3 ya, 1–4 tidak, 5–7 ya, 6–11 tidak, 8–12 ya. Peringkat sirkuit Persamaan (12.14) memberi $11 - 12 + 3 = 2$ siklus bebas.

$$r(G) = |E| - |V| + k. \tag{12.14}$$

Buku: Subbab 12.4.5. Menambahkan {3,4} membiarkan peringkatnya tetap 2: sisi itu sebuah jembatan.

Persoalan K dan L: Warna dan Siklus

K. Warnai G_1 menurut $d[v]$ mod 2: kubu $\{A, D, E, F, I\}$ dan $\{B, C, G, H\}$; kedua belas sisi menyeberang — bipartit. Tambahkan EF (keduanya genap): tidak bipartit; sertifikatnya $E\ B\ A\ C\ F\ E$ (panjang 5) atau $E\ C\ F\ E$ (panjang 3).

L, tak berarah: lewati **persis satu** kemunculan induknya; tetangga bukan-putih lainnya berarti siklus. Lintasan itu asiklik; persegi dan G_1 tidak.

L, berarah: hanya kepala yang **kelabu** yang dihitung. G_2 punya dua siklus; $A \to B$, $A \to C$, $B \to C$ adalah DAG, meskipun C "terlihat dua kali".

JEBAKANNYA. Uji berarah pada graf tak berarah melaporkan siklus pada setiap graf yang punya sisi; aturan induk tak bermakna pada graf berarah.

Buku: Subbab 12.4.6 dan 12.4.7; Persoalan G, H, J, K, dan L sebagai satu program ada di Subbab 12.4.8.

Persoalan M dan N: DAG, Lalu Urutan

Tabel 12.18 Kedelapan mata ajar digraf prasyarat yang dipakai dalam Persoalan M sampai Q, beserta durasi dan prasyaratnya. Sembilan busur seluruhnya: $A \to D$, $B \to D$, $B \to E$, $C \to G$, $D \to E$, $D \to F$, $E \to H$, $F \to H$, $G \to H$.

Kode	Mata ajar	Pekan	Prasyarat
A	Dasar Pemrograman	3	—
B	Matematika Diskret	2	—
C	Kalkulus	4	—
D	Struktur Data	5	A, B
E	Algoritma	6	B, D
F	Basis Data	3	D
G	Statistika	2	C
H	Pembelajaran Mesin	4	E, F, G

Tabel 12.19 Cap waktu pencarian mendalam pada DAG prasyarat. Tiga akar — A, B, dan C, ketiga mata ajar tanpa prasyarat — jadi hutannya punya tiga pohon dan $8 - 3 = 5$ sisi pohon.

v	A	B	C	D	E	F	G	H
d/f	1/10	11/12	13/16	2/9	3/6	7/8	14/15	4/5

Tabel 12.18, ditelusuri pada **Tabel 12.19**: lima busur pohon dan empat silang, **tanpa busur mundur** — sebuah DAG. $B \to D$ silang, bukan maju: D berada di pohon A.

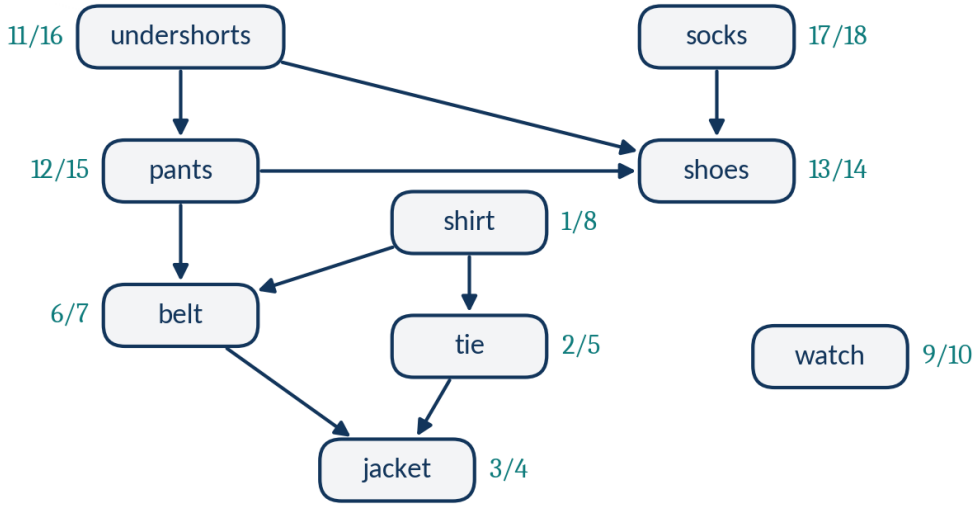
f menurun: C, G, B, A, D, F, E, H; kesembilan busur menunjuk ke depan, sebab Persamaan (12.15). Mengurutkan menurut d gagal pada $B \to A$ yang ditelusuri dari A.

$$f[v] < f[u] \qquad \text{untuk setiap busur } u \to v \text{ pada sebuah DAG,} \tag{12.15}$$

Buku: Subbab 12.5.1–12.5.3.

Contoh Kasus Klasik

Gambar 12.16: DAG berpakaian beserta cap waktunya.

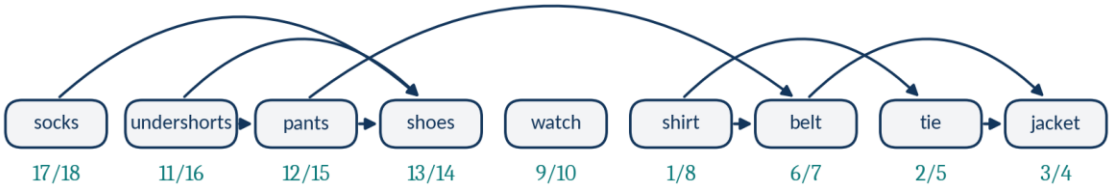


Gambar 12.16 DAG urutan untuk berpakaian, dengan cap waktu pencarian mendalam di samping tiap simpul: kemeja 1/8, dasi 2/5, jas 3/4, ikat pinggang 6/7, arloji 9/10, celana dalam 11/16, celana 12/15, sepatu 13/14, kaus kaki 17/18. Sembilan simpul, sembilan busur, delapan belas cap.

Buku: Subbab 12.5.3 dan Gambar 12.16.

Urutan Topologisnya

Gambar 12.17: waktu selesai menurun, setiap anak panah menunjuk ke kanan.



Gambar 12.17 Hasilnya: kesembilan busana yang sama ditata dari kiri ke kanan menurut waktu selesai yang menurun — kaus kaki 18, celana dalam 16, celana 15, sepatu 14, arloji 10, kemeja 8, ikat pinggang 7, dasi 5, jas 4 — dan kini setiap anak panah menunjuk ke kanan. Itulah *arti* pengurutan topologis.

Buku: Subbab 12.5.3 dan Gambar 12.17.

Persoalan O: Algoritma Kahn

Pseudokode 12.6 Algoritma Kahn: berulang-ulang keluarkan simpul yang tak lagi punya prasyarat tersisa, lalu buang busur-busur keluarnya.

```
KAHN(G)
 1  for each vertex v:  indeg[v] = 0
 2  for each arc (u, v):  indeg[v] = indeg[v] + 1
 3  Q = an empty FIFO queue;  L = an empty list
 4  for each vertex v with indeg[v] == 0:  ENQUEUE(Q, v)
 5  while Q is not empty
 6      u = DEQUEUE(Q);  append u to L
 7      for each v in Adj[u]
 8          indeg[v] = indeg[v] - 1
 9          if indeg[v] == 0:  ENQUEUE(Q, v)
10  if |L| < |V| then report that G has a cycle
11  return L
```

Tabel 12.20 Algoritma Kahn pada DAG prasyarat, satu baris per penghapusan. Entri yang berubah tercetak tebal. Delapan mata ajar dikeluarkan dalam delapan putaran, jadi grafnya taksiklik dan urutannya sah.

Putaran	Keluarkan	Derajat masuk sesudahnya (A…H)	Antrean sesudahnya
1	A	0 0 0 **1** 2 1 1 3	B C
2	B	0 0 0 **0** **1** 1 1 3	C D
3	C	0 0 0 0 1 1 **0** 3	D G
4	D	0 0 0 0 **0** **0** 0 3	G E F
5	G	0 0 0 0 0 0 0 **2**	E F
6	E	0 0 0 0 0 0 0 **1**	F
7	F	0 0 0 0 0 0 0 **0**	H
8	H	semua nol	(kosong)

Derajat masuk 0,0,0,2,2,1,1,3 (jumlahnya $9 = |E|$). **Pseudokode 12.6**, ditelusuri pada **Tabel 12.20**: A, B, C, D, G, E, F, H — satu posisi sama dengan Persoalan N, dan keduanya sah.

Buku: Subbab 12.5.4. Dengan antrean prioritas-min urutannya menjadi menurut abjad.

Persoalan P dan Q: Mencacah, Lintasan Kritis

P. 84 dari $8! = 40\,320$ permutasi sah. Tanpa mendaftar, memoisasikan Persamaan (12.16) atas himpunan bagian: $O(2^n \cdot n)$. Urutannya tunggal jika dan hanya jika ada lintasan Hamilton: diperlukan **tiga** busur tambahan ($A \to B, E \to C, G \to F$).

$$N(\varnothing) = 1, \qquad N(S) = \sum_{\substack{v \in S \\ v \text{ tak punya prasyarat di } S}} N\,(S \setminus \{v\}). \tag{12.16}$$

Q. Menurut Persamaan (12.2) dalam urutan topologis: A 3, B 2, C 4, D 8, G 6, E 14, F 11, H **18**. Lintasan kritis $A \to D \to E \to H$; satu per satu akan memakan 29 minggu. Menambahkan $H \to A$: Kahn hanya mengeluarkan 3, DFS menemukan 1 sisi mundur.

Buku: Subbab 12.5.5 dan 12.5.6; Persoalan M sampai Q sebagai satu program ada di Subbab 12.5.7.

Persoalan R: Kruskal dengan Penelusuran Himpunan

Tabel 12.21 Graf berbobot G pada Persoalan R, S, U, V, dan W: delapan simpul dan tiga belas sisi. Dua sisi, AB dan FG, sama-sama berbobot 3.

Sisi	*AB*	*AC*	*BC*	*BD*	*CD*	*CE*	*DE*	*DF*	*EF*	*EG*	*FG*	*FH*	*GH*
w	3	5	1	6	4	9	2	8	7	10	3	12	11

Tabel 12.22 Kruskal pada G, dengan hutan himpunan saling lepasnya sesudah setiap keputusan. Tujuh sisi diterima, lima ditolak, satu tak pernah diperiksa. Baris 11 layak dipelajari: sisinya *ditolak*, namun `parent[G]` berubah dari `F` menjadi `B`, sebab kedua panggilan `Find` memampatkan lintasannya dalam perjalanan menuju jawabannya.

#	Sisi	w	Find(a)	Find(b)	Vonis	parent	rank
1	BC	1	B	C	ACCEPT	ABBDEFGH	01000000
2	DE	2	D	E	ACCEPT	ABBDDFGH	01010000
3	AB	3	A	B	ACCEPT	BBBDDFGH	01010000
4	FG	3	F	G	ACCEPT	BBBDDFFH	01010100
5	CD	4	B	D	ACCEPT	BBBBDFFH	02010100
6	AC	5	B	B	reject	BBBBDFFH	02010100
7	BD	6	B	B	reject	BBBBDFFH	02010100
8	EF	7	B	F	ACCEPT	BBBBBBFH	02010100

Tabel 12.22 (Lanjutan) *Kruskal pada G, dengan hutan himpunan saling lepasnya sesudah setiap keputusan. Tujuh sisi diterima, lima ditolak, satu tak pernah diperiksa. Baris 11 layak dipelajari: sisinya ditolak, namun parent[G] berubah dari F menjadi B, sebab kedua panggilan Find memampatkan lintasannya dalam perjalanan menuju jawabannya.*

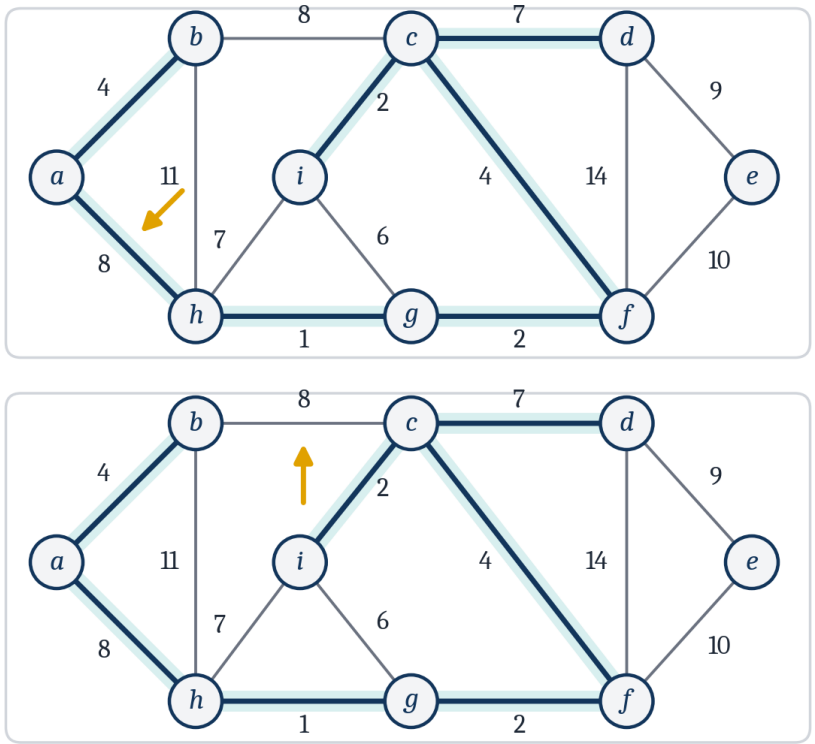
#	Sisi	w	Find(a)	Find(b)	Vonis	parent	rank
9	DF	8	B	B	reject	BBBBBBFH	02010100
10	CE	9	B	B	reject	BBBBBBFH	02010100
11	EG	10	B	B	reject	BBBBBBBH	02010100
12	GH	11	B	H	ACCEPT	BBBBBBBB	02010100

Tabel 12.21, seri dipecahkan menurut abjad; **Tabel 12.22** memperlihatkan `parent` dan `rank` sesudah tiap keputusan. Baris 11 ditolak, namun `parent[G]` berubah — pemampatan. $T = \{BC, DE, AB, FG, CD, EF, GH\}$, $w(T) = 31$; berhenti pada $|V| - 1$.

Buku: Subbab 12.6.1 dan 12.6.2. `rank` *adalah batas atas ketinggian, bukan ketinggiannya.*

Tempat Kruskal dan Prim Berpisah

Gambar 12.18: pada bobot 8, (a, h) diterima dan (b, c) ditolak — Prim melakukan kebalikannya. Kedua pohon berbobot 37.



Gambar 12.18 Kruskal pada graf bersimpul sembilan Bab 11 di bobot 8: (a, h) diterima, (b, c) ditolak. Prim, pada graf yang sama, mengerjakan sebaliknya — ia mengambil (b, c) dan tak pernah menimbang (a, h). Kedua pohonnya berbobot 37. Bobot yang sama adalah tempat kedua algoritma itu bebas berselisih.

Buku: Subbab 12.6.2 dan Gambar 12.18. Uji diri: dengan $w(FG) = 13$ pohonnya berbobot 38.

Persoalan S: Prim dengan Larik Kunci

Untuk v di luar pohon, Persamaan (12.17); keluarkan kunci terkecil, ikatkan sisinya, turunkan kunci tetangganya.

$$key[v] = \min\{\, w(u,v) : u \in T,\ \{u,v\} \in E \,\}, \tag{12.17}$$

Tabel 12.23 Prim–Jarník dari A, satu baris per pengambilan. Delapan pengambilan, tujuh sisi ditetapkan, total 31. Setiap kunci yang turun adalah sebuah `DecreaseKey`: C dari $5/A$ ke $1/B$ pada langkah 2, D dari $6/B$ ke $4/C$ pada langkah 3, E dari $9/C$ ke $2/D$ pada langkah 4, F dari $8/D$ ke $7/E$ pada langkah 5, G dari $10/E$ ke $3/F$ pada langkah 6, dan H dari $12/F$ ke $11/G$ pada langkah 7.

Langkah	Ambil	Sisi yang ditetapkan	Total	Yang menunggu sesudahnya, sebagai v : key/π
1	A	(akar)	0	$B:3/A$ $C:5/A$ $D:\infty$ $E:\infty$ $F:\infty$ $G:\infty$ $H:\infty$
2	B	$AB(3)$	3	$C:1/B$ $D:6/B$ $E:\infty$ $F:\infty$ $G:\infty$ $H:\infty$
3	C	$BC(1)$	4	$D:4/C$ $E:9/C$ $F:\infty$ $G:\infty$ $H:\infty$
4	D	$CD(4)$	8	$E:2/D$ $F:8/D$ $G:\infty$ $H:\infty$
5	E	$DE(2)$	10	$F:7/E$ $G:10/E$ $H:\infty$
6	F	$EF(7)$	17	$G:3/F$ $H:12/F$
7	G	$FG(3)$	20	$H:11/G$
8	H	$GH(11)$	31	(kosong)

Tabel 12.23: dari A pohonnya tumbuh sebagai lintasan, bobot 3,1,4,2,7,3,11 — tujuh sisi yang sama dengan Kruskal, total **31**. Dari H: pohon yang sama dari ujung lainnya.

Buku: Subbab 12.6.3.

Persoalan T: Prim dengan *Heap* Biner

Pseudokode 12.7 Algoritma Prim dengan tumpukan biner dan `DecreaseKey` malas: kunci yang membaik didorong sebagai entri baru, dan entri yang basi dibuang ketika sampai di puncak.

```
LAZY-PRIM(G, w, r)
 1  for each vertex v:  key[v] = infinity;  pi[v] = NIL;  intree[v] = FALSE
 2  key[r] = 0;  H = a min-heap containing (0, r)
 3  while H is not empty
 4      (k, u) = EXTRACT-MIN(H)
 5      if intree[u] or k != key[u]
 6          continue                          // entri basi: buang
 7      intree[u] = TRUE                      // tetapkan sisi (pi[u], u)
 8      for each v in Adj[u]
 9          if not intree[v] and w(u, v) < key[v]
10              key[v] = w(u, v);  pi[v] = u
11              INSERT(H, (key[v], v))        // entri lama tetap tertinggal
```

Tabel 12.24 Prim dengan tumpukan biner dan `DecreaseKey` malas pada graf bersimpul sembilan, dari a. Sembilan pengambilan sungguhan menetapkan delapan sisi dengan bobot total 37; empat pengeluaran lainnya mendapati entri basi — salinan lama g, h (dua kali), dan e yang kuncinya sudah turun sejak itu — dan membuangnya.

Langkah	Keluarkan	Tetapkan	Total	Kunci yang turun	Tumpukan sesudahnya
1	$a:0$	(akar)	0	$b:\infty \to 4$, $h:\infty \to 8$	$b:4$ $h:8$
2	$b:4$	$ab(4)$	4	$c:\infty \to 8$	$c:8$ $h:8$
3	$c:8$	$bc(8)$	12	$d \to 7$, $f \to 4$, $i \to 2$	$i:2$ $f:4$ $d:7$ $h:8$
4	$i:2$	$ci(2)$	14	$g \to 6$, $h:8 \to 7$	$f:4$ $g:6$ $d:7$ $h:7$ $h:8$
5	$f:4$	$cf(4)$	18	$e \to 10$, $g:6 \to 2$	$g:2$ $g:6$ $d:7$ $h:7$ $h:8$ $e:10$
6	$g:2$	$fg(2)$	20	$h:7 \to 1$	$h:1$ $g:6$ $d:7$ $h:7$ $h:8$ $e:10$
7	$h:1$	$gh(1)$	21	—	$g:6$ $d:7$ $h:7$ $h:8$ $e:10$
—	$g:6$	basi, dibuang	21	—	$d:7$ $h:7$ $h:8$ $e:10$
8	$d:7$	$cd(7)$	28	$e:10 \to 9$	$h:7$ $h:8$ $e:9$ $e:10$
—	$h:7$, $h:8$	keduanya basi	28	—	$e:9$ $e:10$

Tabel 12.24 (Lanjutan) *Prim dengan tumpukan biner dan DecreaseKey malas pada graf bersimpul sembilan, dari a. Sembilan pengambilan sungguhan menetapkan delapan sisi dengan bobot total 37; empat pengeluaran lainnya mendapati entri basi — salinan lama g, h (dua kali), dan e yang kuncinya sudah turun sejak itu — dan membuangnya.*

Langkah	Keluarkan	Tetapkan	Total	Kunci yang turun	Tumpukan sesudahnya
9	e: 9	$de(9)$	37	—	e: 10
—	e: 10	basi, dibuang	37	—	(kosong)

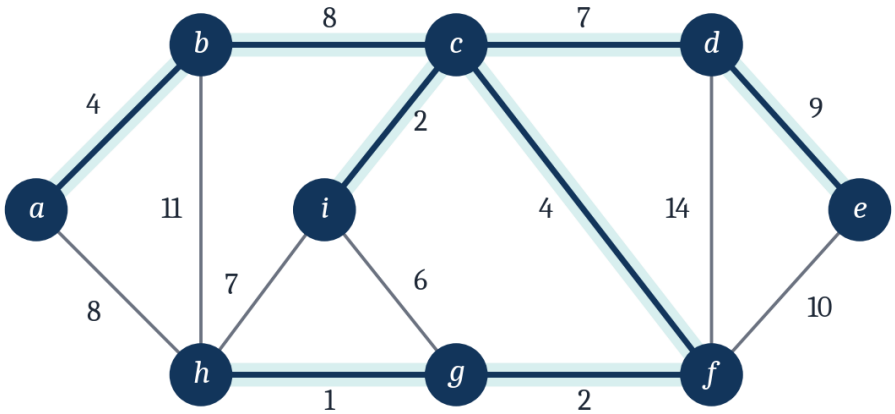
Pseudokode 12.7 pada graf sembilan simpul dari a, **Tabel 12.24**: 13 dorongan, 13 pengambilan, **4 basi**; bobot pohonnya 37. Paling banyak $|E| + 1$ dorongan, sehingga Persamaan (12.18).

$$O(E \lg E) = O(E \lg V), \tag{12.18}$$

Buku: Subbab 12.6.4. Dari i bobotnya lagi-lagi 37, tetapi pohonnya memakai ha alih-alih bc.

Pohon dari a

Gambar 12.19: $ab, bc, ci, cf, fg, gh, cd, de$ — bobot 37, delapan sisi untuk sembilan simpul.



Gambar 12.19 Pohon merentang minimum yang sudah jadi pada graf bersimpul sembilan: (a, b), (b, c), (c, i), (c, f), (f, g), (g, h), (c, d), (d, e), berbobot total 37. Delapan sisi untuk sembilan simpul. Kruskal mencapai himpunan delapan sisi yang berbeda — juga berbobot 37 — sebab dua sisi berbobot 8 seri.

Buku: Subbab 12.6.4 dan Gambar 12.19.

Persoalan U: Sifat Potongan dan Sifat Siklus

Sifat potongan: sisi ringan pada potongan yang dihormati A adalah aman (**Gambar 12.20**). Untuk $S = \{A, B, C\}$ sisi-sisi lintasnya BD 6, CD 4, CE 9; sisi ringannya CD ada di dalam pohon.

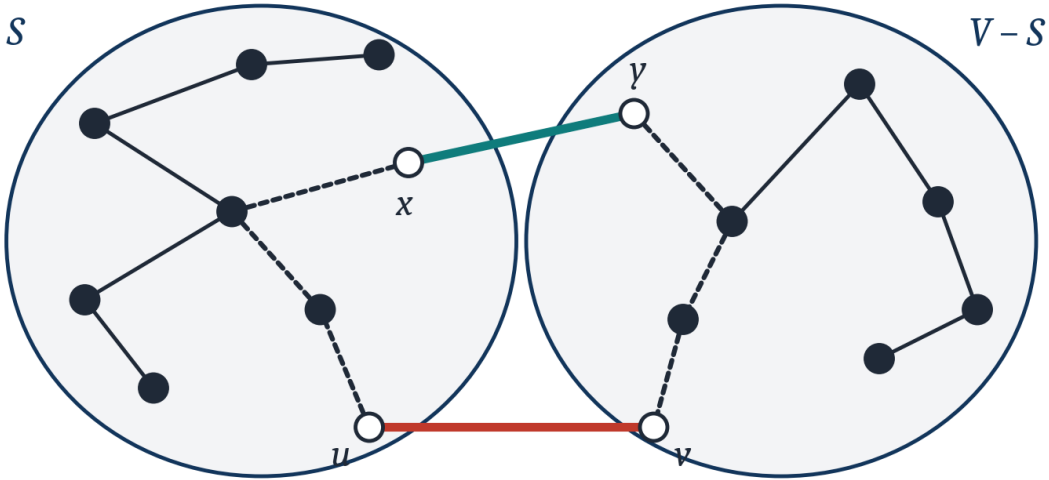
Sifat siklus: sisi terberat yang **tunggal** pada suatu siklus tidak termasuk MST mana pun. **Tabel 12.25**: setiap sisi bukan-pohon benar-benar terberat pada siklusnya — jadi pohon Persoalan R adalah satu-satunya MST.

Tabel 12.25 Setiap sisi bukan-pohon G, siklus yang ditutupnya bersama pohon merentang minimumnya, dan sisi pohon terberat pada siklus itu. Keenamnya tegas terberat, jadi tak satu pun dapat muncul di pohon merentang minimum mana pun — dan itulah bukti lengkap bahwa pohon Persoalan R satu-satunya.

Sisi bukan-pohon	w	Siklus yang ditutupnya	Sisi pohon terberat di atasnya	Vonis
AC	5	A–B–C–A	3 (AB)	tegas terberat ⇒ tak di MST mana pun
BD	6	B–C–D–B	4 (CD)	tak di MST mana pun
CE	9	C–D–E–C	4 (CD)	tak di MST mana pun
DF	8	D–E–F–D	7 (EF)	tak di MST mana pun
EG	10	E–F–G–E	7 (EF)	tak di MST mana pun
FH	12	F–G–H–F	11 (GH)	tak di MST mana pun

Buku: Subbab 12.6.5. Uji diri: untuk $S = \{A, B, C, D\}$ sisi ringannya DE (2).

Penukarannya, Sekali Lagi



Gambar 12.20 Pertukaran yang membuktikan sifat potongan. S dan $V - S$ adalah kedua gumpalannya; sisi merah (u, v) di bawah adalah sisi ringannya, yang diandaikan tidak berada di pohon; garis bertitik adalah lintasan pohon yang tunggal dari u ke v; dan sisi toska (x, y) adalah sisi pertama lintasan itu yang melintasi potongannya. Menukar (x, y) keluar demi (u, v) adalah seluruh buktinya.

Buku: Subbab 12.6.5 dan Gambar 12.20.

Persoalan V: Peringkat dan Pemampatan

Tabel 12.26 Tujuh penyatuan menurut peringkat. Peringkat sebuah akar naik hanya ketika dua peringkat yang sama bertemu — ketujuh penyatuan di sini kebetulan semuanya pertemuan semacam itu, dan itulah sebabnya strukturnya tetap seimbang sempurna.

Operasi	Akibat	parent	rank
`Union(A,B)`	peringkat seri di 0; B di bawah A, `rank[A]`$$1	`AACDEFGH`	`10000000`
`Union(C,D)`	peringkat seri di 0; D di bawah C	`AACCEFGH`	`10100000`
`Union(E,F)`	peringkat seri di 0; F di bawah E	`AACCEEGH`	`10101000`
`Union(G,H)`	peringkat seri di 0; H di bawah G	`AACCEEGG`	`10101010`
`Union(A,C)`	peringkat seri di 1; C di bawah A, `rank[A]`$$2	`AAACEEGG`	`20101010`
`Union(E,G)`	peringkat seri di 1; G di bawah E, `rank[E]`$$2	`AAACEEEG`	`20102010`
`Union(A,E)`	peringkat seri di 2; E di bawah A, `rank[A]`$$3	`AAACAEEG`	`30102010`

Tabel 12.26: setiap penggabungan mempertemukan peringkat yang sama, sehingga pohonnya tetap seimbang setinggi 3. Setelah `Find` pada setiap elemen, `parent = AAAAAAAA` dan peringkatnya tetap basi.

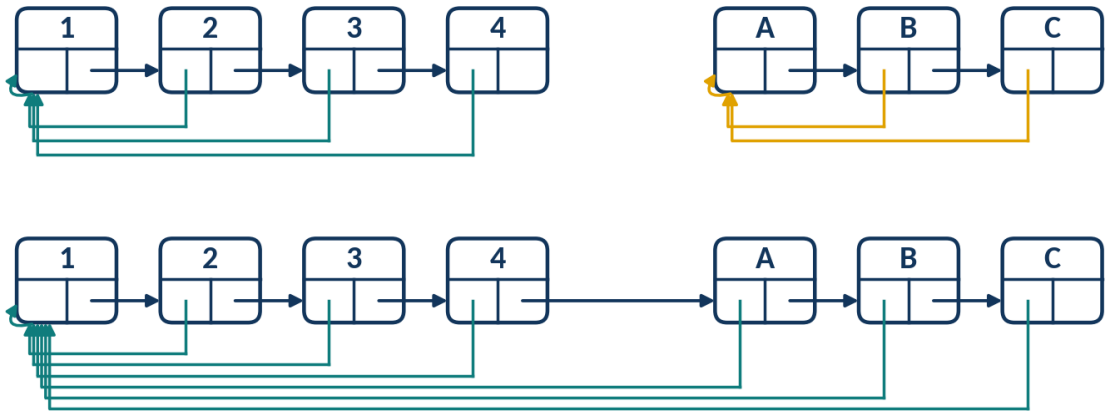
Batas Persamaan (12.19) membatasi tingginya pada $\lfloor \lg 8 \rfloor = 3$; rantai naif mencapai 7. Pada 100 000 elemen berantai: 4 999 950 000 langkah penunjuk tanpa kedua siasat, 99 999 dengan keduanya.

$$size(x) \geq 2^{rank(x)} \qquad \text{untuk setiap akar } x. \tag{12.19}$$

Buku: Subbab 12.6.6. Uji diri: barisan perantaian dengan gabung menurut peringkat memberi tinggi 1.

Daftar, sebagai Pembanding

Gambar 12.21: penggabungan daftar menunjuk ulang setiap elemen daftar yang lebih pendek; hutannya hanya mengubah satu penunjuk.



Gambar 12.21 Himpunan saling lepas sebagai daftar. **Atas:** dua himpunan, tiap unsur membawa penunjuk balik ke wakilnya. **Bawah:** hasil menyatukan keduanya — daftar yang lebih pendek disambungkan dan setiap unsurnya diarahkan ulang. Bandingkan ongkosnya dengan **Tabel 12.26**, yang di dalamnya sebuah penyatuan mengubah persis satu penunjuk.

Buku: Subbab 12.6.6 dan Gambar 12.21.

Persoalan W: Tunggalkah MST-nya?

Tabel 12.27 Uji ketunggalan pada G: setiap sisi pohon dihapus bergiliran lalu Kruskal dijalankan ulang. Setiap baris tegas lebih berat daripada 31, jadi pohon merentang minimum G **tunggal**.

Sisi dihapus	w	Pohon merentang terbaik yang baru	Vonis
BC	1	35	lebih berat 4
DE	2	37	lebih berat 6
AB	3	33	lebih berat 2
FG	3	38	lebih berat 7
CD	4	33	lebih berat 2
EF	7	32	lebih berat 1
GH	11	32	lebih berat 1

Tabel 12.27: setiap penghapusan benar-benar lebih berat — **tunggal**, walaupun AB dan FG sama-sama berbobot 3. Bobot berbeda-beda itu cukup, bukan perlu.

Turunkan $w(AC)$ menjadi 3: dua MST berbobot 31, dengan AB atau AC. **Uji diri:** turunkan $w(FH)$ menjadi 11 sebagai gantinya — lagi-lagi dua.

Buku: Subbab 12.6.7; Persoalan R, S, U, V, dan W sebagai satu program ada di Subbab 12.6.8.

Latihan Uji Diri Singkat

Greedy. Dua bahannya? Pilihan greedy dan substruktur optimal. Maksimal atau maksimum? Greedy memberi yang maksimal tanpa susah payah. *Knapsack* mana yang tak punya pilihan greedy? 0-1.

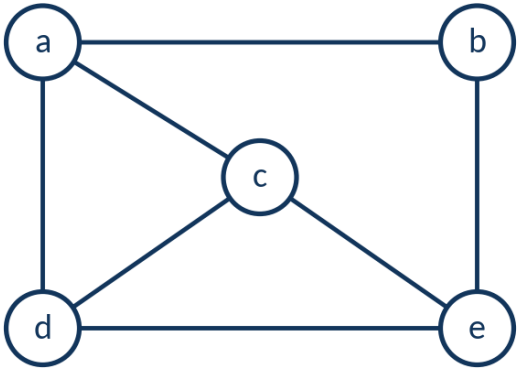
Penelusuran. Lema jabat tangan? $\sum_v \deg(v) = 2|E|$. Mengapa jarak BFS terpendek? Urutan pengeluaran yang tak menurun. Golongan sisi pada DFS tak berarah? Hanya pohon dan mundur.

DAG. Teorema DAG? Tanpa sisi mundur. Mengapa f menurun, bukan d? $f[v] < f[u]$ untuk setiap busur.

MST. Sisi aman? Menjaga A tetap di dalam suatu MST. MST tunggal? Setiap sisi bukan-pohon benar-benar terberat pada siklusnya. **Gambar 12.22** memuat seluruh kosakatanya.

Buku: Subbab 12.7, tiga puluh latihan beserta rujukan subbabnya.

Kosakata dalam Satu Gambar



$$V = \{a, b, c, d, e\}$$

$$E = \{(a, b), (a, c), (a, d), (b, e), (c, d), (c, e), (d, e)\}$$

Gambar 12.22 Definisi graf, dengan contoh bersimpul lima yang dipakai di sepanjang Bab 10: $V = \{a, b, c, d, e\}$ dan $E = \{(a, b), (a, c), (a, d), (b, e), (c, d), (c, e), (d, e)\}$. Gambar dan kedua daftar himpunannya membawa informasi yang identik; gambarnya hanya lebih enak dipandang. Derajat 3,2,3,3,3 berjumlah $14 = 2 \times 7$ — latihan 11 dalam satu gambar.

Buku: Subbab 12.7 dan Gambar 12.22.

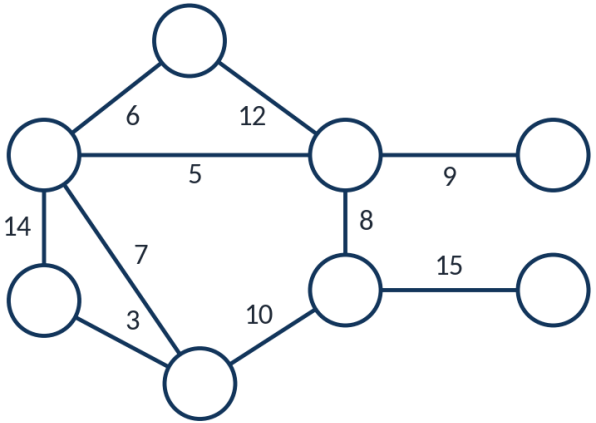
Profitina dalam Praktik

Perayapan = BFS (kedalaman klik $d[v]$; $d[v] = \infty$ berarti yatim). **Halaman nyaris kembar = MST yang sisi beratnya dipotong** (**Gambar 12.23**). **Rencana perbaikan = DAG**, diurutkan secara topologis. **Pemindaian ulang** = tenggat-terawal-dahulu.

Audit situs web Profitina Laundry: `/promo-lama` yatim; tiga gugus nyaris kembar (ambang 30); perbaikannya memakan **16 jam** pada lintasan kritis lawan 25 jam bila satu tugas setiap kali.

Buku: Subbab 12.8 dan Subbab 12.8.1. Aplikasi memberi saran; pengguna yang memutuskan.

Graf Kemiripan Berbentuk Seperti Ini



Gambar 12.23 Sebuah graf berbobot dari Bab 11. Sebagian sisi yang mahal tak terhindarkan — yang berbobot 15 adalah satu-satunya jalan masuk ke simpulnya — dan sebagian yang tampak murah justru mubazir. Graf kemiripan halaman berwatak persis seperti ini, dan algoritma yang sama memisahkan kedua jenis sisinya.

Buku: Subbab 12.8 dan Gambar 12.23.

Kesalahan yang Sering Terjadi

Rancangan: algoritma hiasan; rancangan ditulis sesudah programnya; tanpa evaluasi; waktu dicetak tetapi tak dibaca.

Greedy: memercayai aturan karena pernah berhasil sekali; menulis "solusi optimalnya" alih-alih "*suatu* solusi optimal"; membaca penukaran matroid sebagai "setiap unsur berhasil"; kode Huffman yang tak diperiksa.

Penelusuran: hanya hasil akhir penelusuran; uji berarah pada graf tak berarah; sisi maju atau silang pada DFS tak berarah; "maju" karena sasarannya datang belakangan.

DAG dan MST: mengurutkan menurut waktu penemuan; entri *heap* basi tak dilewati; bobot kembar dikira berarti tidak tunggal; Prim dan Kruskal diharapkan mengembalikan sisi yang sama; `rank` dibaca sebagai tinggi.

Buku: Subbab 12.9.

Ringkasan: Lima Hal untuk Diingat

1. Proyek perancangan berjalan dalam sembilan tahap, enam di antaranya sebelum kode; uji pemikul beban memutuskan apakah persoalannya layak diberi algoritma.

2. Rute Laundry: 6 dari 14 zona dalam dua lompatan, 2 tak terjangkau; tulang punggung 6,9 km (hemat 43,9%); lintasan kritis 121 menit — semuanya dibuktikan dengan tangan lebih dulu.

3. Evaluasi menjawab *benarkah, apakah ia berskala, apa manfaatnya* — pascasyarat mengalahkan pemeriksaan silang; bagilah waktu dengan ongkos ramalannya.

4. Aturan greedy memerlukan bukti penukaran atau contoh penyangkal; penelusuran memerlukan setiap nilai antara.

5. Penelusuran, urutan topologis, dan MST jarang tunggal — periksalah sifat dan bobotnya, bukan untai dan himpunan sisinya.

Buku: Subbab 12.10.

Pertemuan Berikutnya

Bab 13 — Lintasan Terpendek Sumber Tunggal. Bobot menjadi waktu tempuh dan pertanyaannya menjadi **lintasan** termurah.

BFS bila setiap sisi berongkos sama; **Dijkstra** untuk ongkos taknegatif; **Bellman–Ford** untuk ongkos negatif; dan pada DAG satu sapuan topologis — sapuan yang menghasilkan lintasan-lintasan kritis hari ini.

Sebelum pertemuan: kerjakan Soal Latihan 12.3, 12.5, 12.9, dan 12.12, lalu bacalah Subbab 13.1 sampai 13.4.

Buku: Subbab 12.10.1 dan Bab 13.

Kuliah 13 — Lintasan Terpendek Sumber Tunggal

Perancangan dan Analisis Algoritma (EF234405)

Bab 13 buku ini dalam bentuk ringkas, untuk dipakai di kelas.

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*.

Pengajar: Irfan Subakti — yifana@gmail.com

Hari ini kita berhenti mencacah sisi dan mulai menjumlahkannya: tiga algoritma, satu primitif, dan satu pengandaian yang, ketika dilanggar, merusak segalanya tanpa bersuara. Buku: Bab 13, paragraf pembuka.

Agenda

1. **Persoalannya, dinyatakan dengan cermat.** Bobot sebuah lintasan, bilangan $\delta(u, v)$, dan keempat ragam pertanyaannya.
2. **Dua fakta struktural.** Substruktur optimal dan ketaksamaan segitiga; setiap algoritma hari ini memanfaatkan keduanya.
3. **Bobot negatif.** *Sisi* negatif tidak berbahaya; *siklus* negatif merusak persoalannya.
4. **Relaksasi**, satu-satunya primitif, dan kelima sifat yang membuatnya aman.
5. **Dijkstra**, ditelusuri panel demi panel, dibuktikan benar, dan dihitung waktunya; **6. Bellman–Ford**, ditelusuri sapuan demi sapuan, beserta pendeteksi siklus negatifnya.
7. **DAG** dalam waktu linear; **8. memulihkan rutenya**; algoritma mana untuk graf mana; Profitina Laundry.

Urutan halaman hari ini persis mengikuti buku, dan soal latihan di akhir bab adalah pekerjaan rumahnya. Buku: Subbab 13.2 sampai 13.12.

Dari Mana Kita Melanjutkan

Dari Bab 10, grafnya. Sebuah graf berarah $G=(V,E)$; sebuah lintasan $p=\langle v_1, v_2, \ldots, v_k \rangle$ dengan $(v_i, v_{i+1}) \in E$; senarai ketetanggaan $Adj[u]$. Di dalam notasi asimtotik kita melepas palangnya: $O(V+E)$ berarti $O(|V|+|E|)$.

Dari Bab 10, penelusuran melebar. Ia menemukan *banyaknya sisi* terkecil dari s ke setiap simpul dalam $O(V+E)$: persoalan hari ini pada kasus khusus ketika setiap sisi bernilai sama.

Dari Bab 11, bobotnya. Sebuah fungsi bobot $w : E \to R$. Perhatikan R, bukan $R^{\geq 0}$: bilangan negatif diizinkan, dan Subbab 13.5 menagih ongkosnya kepada kita.

Dari Bab 11, antrean prioritas (`Insert`, `ExtractMin`, `DecreaseKey`, sebagai larik tak terurut, *binary heap*, atau *Fibonacci heap*) dan **pengurutan topologis**, yang dipakai Subbab 13.9 secara langsung.

> **Pohon merentang minimum bukan pohon lintasan terpendek.** Bab 11 mengukur *seluruh jejaring*: subgraf terhubung termurah, tanpa titik berangkat. Hari ini kita mengukur sebuah *perjalanan*: rute termurah dari satu simpul tertentu. Keduanya biasanya pohon yang berbeda (Soal Latihan 13.4 dan 13.5).

Buku: Subbab 13.1.

Dari Cacah Sisi ke Bobot Total

DEFINISI. Bobot lintasan $p=\langle v_1, v_2, \ldots, v_k \rangle$ adalah jumlah bobot sisi-sisinya:

$$w(p)=\sum_{i=1}^{k-1} w(v_i, v_{i+1}). \tag{13.1}$$

Pada Persamaan (13.1) indeks i berjalan sampai $k-1$, satu nilai untuk tiap **sisi**, bukan untuk tiap simpul. Lintasan berisi satu simpul tidak punya sisi, sehingga jumlahnya kosong dan $w(p)=0$; kesepakatan itulah yang membuat $\delta(s,s)=0$ keluar dengan benar.

DEFINISI. Bobot lintasan terpendek dari u ke v adalah

$$\delta(u,v) = \begin{cases} \min\{\, w(p) : p \text{ adalah lintasan dari } u \text{ ke } v \,\} & \text{bila lintasan semacam itu ada,} \\ \infty & \text{bila tidak.} \end{cases} \tag{13.2}$$

Sebuah **lintasan terpendek** adalah sembarang lintasan p dari u ke v dengan $w(p)=\delta(u,v)$.

Tiga jebakan pada Persamaan (13.2). δ adalah sebuah *bilangan*, lintasan terpendek adalah sebuah *barisan*. Ia **sebuah** lintasan terpendek, bukan **satu-satunya**: bisa ada beberapa yang bobotnya sama. Dan $\delta(u,v)=\infty$ bukan kegagalan, melainkan jawaban *tidak ada rute*.

> **CONTOH.** Pada graf contoh Dijkstra, $p = \langle s, x, u, v \rangle$ berbobot $w(p) = 5+3+1 = 9$, sedangkan $q = \langle s, u, v \rangle$ berbobot $w(q) = 10+1 = 11$. Lintasan yang lebih murah memakai *lebih banyak* sisi: lebih sedikit sisi dan lebih sedikit biaya adalah dua hal yang berbeda.

Di dalam kode, ∞ adalah `const long long INF = LLONG_MAX / 4;` *dan tiap relaksasi dikawal oleh* `d[u] < INF`*; itulah sebabnya tidak satu pun program hari ini meluap. Buku: Subbab 13.2.1 sampai 13.2.3.*

$\delta(s,v)$ dengan Cara yang Jujur

Sebelum algoritma apa pun, hitunglah $\delta(s,v)$ pada graf contoh Dijkstra dengan *mendaftarkan setiap lintasan sederhana dari s ke v lalu menjumlahkan*. Sisi-sisinya: $w(s,u)=10$, $w(s,x)=5$, $w(u,v)=1$, $w(u,x)=2$, $w(x,u)=3$, $w(x,v)=9$, $w(x,y)=2$, $w(v,y)=4$, $w(y,v)=6$, $w(y,s)=7$.

Tabel 13.1 Setiap lintasan sederhana dari s ke v di dalam graf contoh Dijkstra, ditimbang satu sisi setiap kalinya; minimumnya, 9, adalah $\delta(s,v)$, yang dicapai oleh $\langle s,x,u,v\rangle$.

Lintasan dari s ke v	Bobot sisi	Total
$\langle s,u,v\rangle$	10+1	11
$\langle s,x,v\rangle$	5+9	14
$\langle s,x,u,v\rangle$	5+3+1	9
$\langle s,x,y,v\rangle$	5+2+6	13
$\langle s,u,x,v\rangle$	10+2+9	21
$\langle s,u,x,y,v\rangle$	10+2+2+6	20

Mengapa tepat enam baris pada Tabel 13.1? Dari s hanya ada jalan keluar ke u dan ke x; menelusuri setiap cabang sampai mencapai v atau kehabisan simpul yang belum dikunjungi menghasilkan tepat keenam rute sederhana ini.

Benar, tetapi mustahil dipakai secara umum: graf dengan n simpul dapat memiliki lebih dari $(n-2)!$ lintasan sederhana antara sepasang simpul. Pada tiga puluh simpul, cara ini mendaftarkan lebih banyak lintasan daripada banyaknya atom dalam sebutir pasir.

Ingatlah bilangan 9*: Dijkstra akan menghasilkannya dalam lima pengeluaran. Buku: Subbab 13.2.4.*

Empat Persoalan, Satu Algoritma

Tabel 13.2 Keempat persoalan lintasan terpendek. Tiga dari empatnya tereduksi ke yang kedua, dan itulah sebabnya yang kedua itulah yang kita selesaikan.

Ragam	Yang ditetapkan	Yang ditanyakan	Cara kita menyelesaikannya
Pasangan tunggal	sumber s, target t	$\delta(s,t)$ dan satu lintasan	selesaikan sumber tunggal dari s, boleh berhenti lebih awal
Sumber tunggal	sumber s	$\delta(s,v)$ untuk semua v	bab ini: Dijkstra, Bellman–Ford, relaksasi DAG
Tujuan tunggal	target t	$\delta(u,t)$ untuk semua u	balikkan setiap sisi, lalu selesaikan sumber tunggal dari t
Semua pasangan	tidak ada	$\delta(u,v)$ untuk semua u, v	Bab 15: Floyd–Warshall, Johnson; atau $\lvert V \rvert$ kali menjalankan bab ini

Pasangan tunggal tidak lebih mudah. Setiap metode yang dikenal adalah metode sumber tunggal yang boleh berhenti ketika targetnya keluar dari antrean, dan pada kasus terburuk target itu keluar paling akhir. Belum dikenal metode yang lebih baik secara asimtotik.

Tujuan tunggal adalah bayangan cermin. Bangunlah transpos G^T dengan $E^T = \{(v,u) : (u,v) \in E\}$, pertahankan bobotnya, dan jalankan dari t. Membalik sisi berongkos $\Theta(V+E)$.

Semua pasangan adalah $\lvert V \rvert$ kali menjalankan sumber tunggal, atau pokok bahasan Bab 15. Jadi Tabel 13.2 berkata: tiga dari keempat ragam tereduksi ke sumber tunggal atau dibangun darinya.

Buku: Subbab 13.2.5 dan 13.2.6.

Di Mana Lintasan Terpendek Muncul

Perutean jaringan. OSPF bersifat *link-state*: setiap router membangun peta lalu menjalankan **Dijkstra**. RIP bersifat *distance-vector*: router saling bertukar tabel jarak dengan tetangganya, yang tidak lain adalah **Bellman–Ford** yang dijalankan tersebar, satu sapuan per putaran pertukaran.

Perencanaan gerak robot. Diskretkan lantainya; beri setiap sisi bobot waktu atau energi. Bobotnya secara alami tak negatif, sehingga Dijkstra berlaku.

Peta dan lalu lintas. Bobotnya adalah waktu tempuh yang berubah sepanjang hari, dan kuerinya pasangan tunggal. Sistem produksi memakai A* dan *contraction hierarchies*, yang kebenarannya diargumentasikan dari sifat-sifat Subbab 13.6.

Penjadwalan dengan prasyarat. Tugas-tugas membentuk DAG, bobotnya durasi, dan lintasan *terpanjang* adalah lintasan kritisnya (Subbab 13.9.7).

Arbitrase mata uang. Beri setiap sisi penukaran bobot $-\ln(\text{kurs})$; siklus negatif adalah rangkaian penukaran yang menghasilkan lebih banyak uang daripada modal awalnya. Pendeteksi siklus Bellman–Ford *adalah* pendeteksi arbitrase.

Teori umumnya, lintasan terpendek sebagai aliran berbiaya minimum yang paling sederhana, ada pada Ahuja, Magnanti, dan Orlin; uraian yang lebih ringan ada pada Kleinberg dan Tardos. Buku: Subbab 13.2.7.

Substruktur Optimal

TEOREMA 13.1. *Sublintasan dari lintasan terpendek adalah lintasan terpendek.* Bila $p = \langle v_1, \ldots, v_k \rangle$ adalah lintasan terpendek dari v_1 ke v_k, maka untuk sembarang $1 \le i \le j \le k$ sublintasan $p_{ij} = \langle v_i, \ldots, v_j \rangle$ adalah lintasan terpendek dari v_i ke v_j.

Bukti, dengan menggunting dan menempel. Pecahlah p pada kedua simpul yang dipilih:

$$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk}). \tag{13.3}$$

Bila ada p'_{ij} dengan $w(p'_{ij}) < w(p_{ij})$, guntinglah p_{ij} lalu tempelkan p'_{ij} (sah, ujung-ujungnya cocok). Menurut Persamaan (13.3) lintasan barunya berbobot $w(p_{1i}) + w(p'_{ij}) + w(p_{jk}) < w(p)$, bertentangan dengan keminimuman p. ∎

Langkah menyambung patut dicurigai. Ia berhasil di sini karena dua lintasan yang berbagi ujung selalu dapat disambungkan. Ia gagal untuk lintasan *sederhana* terpanjang, karena sambungannya dapat mengunjungi simpul dua kali; itulah sebabnya persoalan itu NP-hard sedangkan yang ini tidak.

Apa yang kita peroleh. Izin untuk membangun lintasan terpendek sedikit demi sedikit. Setiap algoritma hari ini menjawab satu pertanyaan: *dalam urutan apa simpul-simpul harus ditinjau, agar ketika saya memakai $\delta(s, u)$ nilainya sudah diketahui?* Dijkstra: *menurut jarak yang menaik.* Bellman–Ford: *saya tidak tahu, maka coba semuanya $|V| - 1$ kali.* Algoritma DAG: *menurut urutan topologis.*

Bila jalan termurah dari Depot ke Keputih melewati ITS, maka bagian Depot–ITS-nya adalah jalan termurah dari Depot ke ITS. Buku: Subbab 13.3.

Ketaksamaan Segitiga

TEOREMA 13.2 (bentuk simpul). Untuk sembarang tiga simpul u, x, v,

$$\delta(u,v) \le \delta(u,x) + \delta(x,v). \tag{13.4}$$

Bukti Persamaan (13.4)**.** Bila salah satu suku di ruas kanan adalah ∞, ketaksamaan berlaku dengan sendirinya. Bila tidak, misalkan p_1 lintasan terpendek $u \rightsquigarrow x$ dan p_2 lintasan terpendek $x \rightsquigarrow v$; sambungannya q adalah lintasan dari u ke v dengan $w(q) = \delta(u,x) + \delta(x,v)$. Minimum atas sebuah himpunan tidak melebihi anggota mana pun, sehingga $\delta(u,v) \le w(q)$. ∎

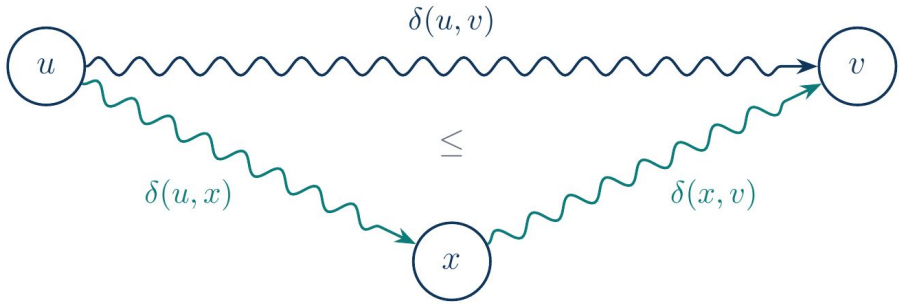
AKIBAT 13.3 (bentuk sisi). Untuk setiap sisi $(u,v) \in E$ dan setiap sumber s,

$$\delta(s,v) \le \delta(s,u) + w(u,v). \tag{13.5}$$

Bukti Persamaan (13.5)*:* Teorema 13.2 pada (s,u,v) memberi $\delta(s,v) \le \delta(s,u) + \delta(u,v)$, dan sisi tunggal $\langle u,v \rangle$ adalah sebuah lintasan, sehingga $\delta(u,v) \le w(u,v)$. ∎ Dalam kata-kata: jarak sebenarnya ke v tidak pernah dapat dikalahkan dengan mencapai u secara optimal lalu menempuh satu sisi lagi.

Seluruh Subbab 13.6 dibangun di atas bentuk sisi ini; algoritma-algoritmanya berusaha membuat estimasinya memenuhi ketaksamaan itu dengan kesamaan. Buku: Subbab 13.4.1 sampai 13.4.3.

Ketaksamaan Segitiga, Digambar



Gambar 13.1 Ketaksamaan segitiga: anak panah berombak di atas adalah lintasan terpendek dari u ke v, pasangan di bawah adalah lintasan terpendek dari u ke x disusul lintasan terpendek dari x ke v. Rute bawah itu adalah *sebuah* lintasan dari u ke v, sehingga ia tidak mungkin lebih murah daripada yang terbaik.

Pada Gambar 13.1 garis berombak adalah notasi baku untuk “suatu lintasan, tidak penting yang mana”. Bukti satu barisnya: rute terbaik $u \rightsquigarrow v$ tidak lebih panjang daripada rute lain mana pun, khususnya rute yang dipaksa melewati x.

Buku: Subbab 13.4.2.

Bentuk Sisinya, Diperiksa pada Sepuluh Sisi

Tabel 13.3 Bentuk sisi ketaksamaan segitiga diperiksa pada setiap sisi graf contoh Dijkstra, dengan $\delta(s,s)=0$, $\delta(s,x)=5$, $\delta(s,y)=7$, $\delta(s,u)=8$, $\delta(s,v)=9$.

Sisi (u,v)	$w(u,v)$	$\delta(s,u)+w(u,v)$	$\delta(s,v)$	Berlaku?
(s,u)	10	$0+10=10$	8	$8\le 10$ ya
(s,x)	5	$0+5=5$	5	kesamaan
(u,v)	1	$8+1=9$	9	kesamaan
(u,x)	2	$8+2=10$	5	ya
(x,u)	3	$5+3=8$	8	kesamaan
(x,v)	9	$5+9=14$	9	ya
(x,y)	2	$5+2=7$	7	kesamaan
(v,y)	4	$9+4=13$	7	ya
(y,v)	6	$7+6=13$	9	ya
(y,s)	7	$7+7=14$	0	ya

Baris-baris Tabel 13.3 yang menunjukkan *kesamaan* — (s,x), (x,u), (u,v), (x,y) — persis keempat sisi pohon lintasan terpendeknya, sebagaimana diterangkan Subbab 13.6.9. Mencapai kesamaan itulah seluruh pekerjaannya.

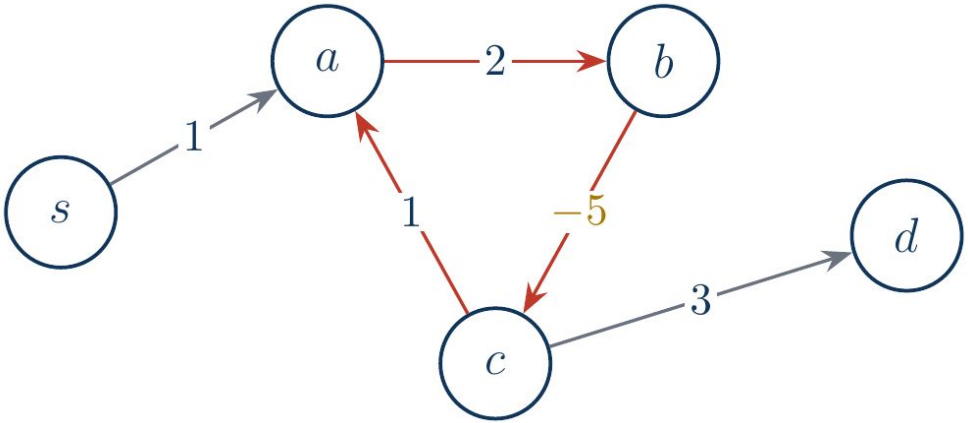
Buku: Subbab 13.4.4.

Sebuah Sisi Negatif; Sebuah Siklus Negatif

Sisi negatif tidaklah aneh. Ruas kurir yang berakhir di titik antar mitra yang memberi rabat berbiaya bersih negatif. Teorema 13.1 dan 13.2 dibuktikan *tanpa* mengandaikan $w \geq 0$: buktinya hanya menjumlahkan dan membandingkan.

> **PERINGATAN.** Bila sebuah siklus berbobot negatif terjangkau dari s dan v terjangkau dari siklus itu, maka $\delta(s, v)$ *tidak ada* sebagai bilangan berhingga. Menurut kesepakatan kita menulis $\delta(s, v) = -\infty$.

Mengapa. Dengan x pada siklus c, lintasan $p : s \rightsquigarrow x$ dan lintasan $q : x \rightsquigarrow v$, jalan yang mengitari c tepat k kali berbobot $w(p) + k\,w(c) + w(q)$. Karena $w(c) < 0$, nilainya turun di bawah bilangan mana pun yang Anda sebutkan: tidak ada minimum.



Gambar 13.2 Sebuah graf dengan siklus negatif yang terjangkau. Berkeliling siklus merah $a \to b \to c \to a$ berongkos $2+(-5)+1=-2$, sehingga tiap putaran tambahan membuat "lintasannya" lebih murah dua; setiap simpul yang terjangkau dari siklus itu memiliki $\delta(s,\cdot)=-\infty$.

Pada Gambar 13.2, $\delta(s,a)=\delta(s,b)=\delta(s,c)=\delta(s,d)=-\infty$ sedangkan $\delta(s,s)=0$. Siklus negatif yang **tak terjangkau** tidak merugikan, dan simpul yang tidak terjangkau *dari* siklus itu mempertahankan jarak berhingganya. Persoalannya tidak sukar di sini; ia *salah rumus*.

Graf contoh Bellman–Ford mempunyai tiga sisi negatif dan jawaban yang terdefinisi dengan baik. Buku: Subbab 13.5.1 dan 13.5.2.

Lintasan Terpendek Tidak Punya Siklus

TEOREMA 13.4. Bila tidak ada siklus berbobot negatif yang terjangkau dari s, maka untuk setiap v yang terjangkau terdapat lintasan terpendek dari s ke v yang **sederhana**, sehingga memiliki paling banyak $|V|-1$ sisi.

Bukti. Misalkan p lintasan terpendek yang mengulang simpul x, sehingga bentuknya $s \rightsquigarrow x \overset{c}{\rightsquigarrow} x \rightsquigarrow v$. Membuang c menyisakan p' dengan $w(p') = w(p) - w(c)$.

c terjangkau dari s, sehingga menurut hipotesis $w(c) \geq 0$ dan $w(p') \leq w(p)$; dan p terpendek, sehingga $w(p') \geq w(p)$. Jadi lintasan yang lebih pendek itu *juga* terpendek. Setiap pembuangan mengurangi sekurang-kurangnya satu sisi, sehingga prosesnya berakhir pada lintasan terpendek sederhana, dengan paling banyak $|V|-1$ sisi. ∎

Ingatlah yang satu ini. *Setiap lintasan terpendek tidak lebih panjang daripada $n-1$ sisi.* Inilah alasan Bellman–Ford menjalankan tepat $|V|-1$ sapuan, dan alasan sapuan ke-$|V|$-nya adalah *pendeteksi*, bukan kerja yang berguna.

CONTOH. Kedua graf contoh memiliki lima simpul, sehingga lintasan terpendek dari s paling banyak empat sisi. Pada graf Dijkstra yang terpanjang adalah $\langle s, x, u, v \rangle$, tiga sisi; pada graf Bellman–Ford adalah $\langle s, y, x, t, z \rangle$, tepat empat. Itulah sebabnya keempat sapuan diperlukan di sana.

Buku: Subbab 13.5.3.

Algoritma Mana Bertahan terhadap Apa

Tabel 13.4 Apa yang dikerjakan masing-masing dari ketiga algoritma itu pada tiap rezim bobot. Kata yang perlu diperhatikan adalah *diam-diam*.

Keadaan	Dijkstra	Bellman–Ford	Relaksasi DAG
Semua bobot ≥ 0	benar	benar	benar (bila asiklik)
Ada bobot < 0, tanpa siklus negatif	**salah, diam-diam**	benar	benar (bila asiklik)
Siklus negatif terjangkau dari s	salah, diam-diam	mendeteksinya, mengembalikan `false`	tidak mungkin muncul: DAG sama sekali tidak bersiklus
Siklus negatif tak terjangkau dari s	salah bila ada sisi negatif yang terjangkau	benar untuk bagian yang terjangkau	tidak mungkin muncul

Kata yang perlu diperhatikan pada Tabel 13.4 adalah **diam-diam**: Dijkstra tidak macet dan tidak mengeluh pada sisi negatif; ia mengembalikan bilangan salah yang tampak masuk akal. Subbab 13.7.3 membangun graf terkecil semacam itu.

Baris 2 menentukan benar-tidaknya program Anda. Baris 3 menentukan apakah pertanyaannya punya jawaban sama sekali.

Kebiasaan yang berguna: periksalah `w >= 0` untuk setiap sisi sebelum menjalankan Dijkstra yang Anda serahkan; jawaban yang salah berubah menjadi kegagalan yang nyaring. Buku: Subbab 13.5.4.

Relaksasi: Kedua Lariknya

DEFINISI. Untuk setiap simpul v kita menyimpan dua nilai. $d[v]$, **estimasi lintasan terpendek**, selalu merupakan *batas atas* bagi $\delta(s, v)$: bobot rute terbaik dari s ke v yang sejauh ini ditemukan. $\pi[v]$, **pendahulu**, adalah simpul asal kita tiba di v sepanjang rute itu, atau `NIL`.

Bilangan d menjawab *seberapa jauh*; pendahulu π menjawab *lewat mana*, dan Subbab 13.10 mengubahnya menjadi daftar simpul.

PERINGATAN. $d[v]$ adalah sebuah *estimasi*. Selama berjalan ia bisa terlalu besar; ia tidak pernah terlalu kecil. Kode yang memperlakukan $d[v]$ sebagai jawaban di tengah gelung adalah galat paling umum di seluruh pokok bahasan ini.

GAGASAN KUNCI. Relaksasi adalah **satu-satunya** cara d dan π berubah sesudah inisialisasi. Dijkstra, Bellman–Ford, dan algoritma DAG hanya berbeda dalam **urutan merelaksasi sisi, dan berapa kali**, sehingga segala yang dibuktikan tentang d dan π berlaku bagi ketiganya sekaligus.

Buku: Subbab 13.6.1.

Inisialisasikan, Lalu Relaksasikan

Pseudokode 13.1 Inisialisasi yang dipakai bersama oleh setiap algoritma sumber tunggal: semua estimasi tak hingga, tanpa pendahulu, dan sumbernya berjarak nol.

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  untuk tiap simpul v di V[G]
2      d[v] <- tak hingga
3      pi[v] <- NIL
4  d[s] <- 0
```

Pseudokode 13.2 Merelaksasi sisi (u, v): satu pembandingan, dan dua pemberian nilai bila rute lewat u benar-benar lebih baik.

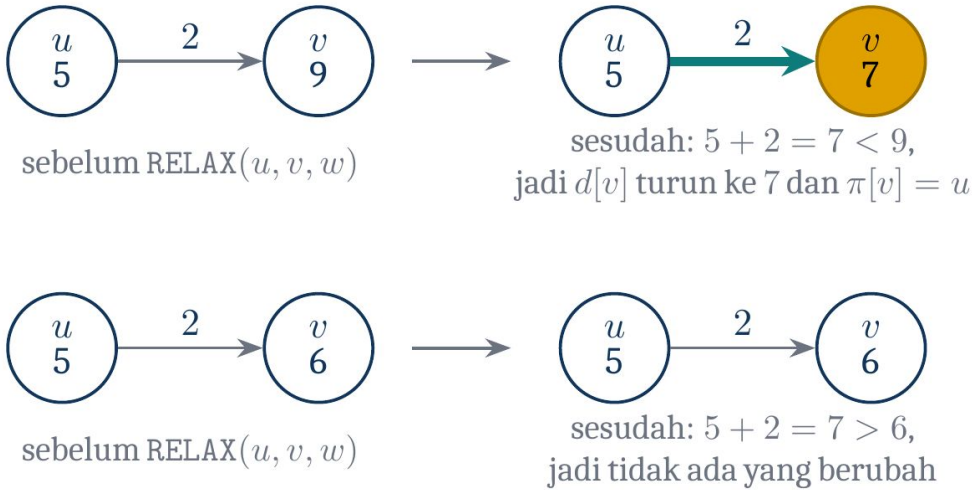
```
RELAX(u, v, w)
1  bila d[v] > d[u] + w(u,v)
2      d[v]  <- d[u] + w(u,v)
3      pi[v] <- u
```

Pseudokode 13.1 berongkos $\Theta(V)$ dan berkata: *saya tahu satu rute, rute kosong dari s ke dirinya sendiri, dan rute itu gratis; tentang selebihnya saya sama sekali tidak tahu.* Inilah baris-baris pertama ketiga algoritmanya.

Pseudokode 13.2 adalah satu pembandingan dan dua pemberian nilai. Baris 1 hanya menyala bila nilai barunya **lebih kecil**, sehingga $d[v]$ hanya pernah turun, seperti pegas tegang yang dibiarkan mengendur.

Tiga baris yang sama inilah baris "relaksasi sisi" yang disorot pada pseudokode Dijkstra: satu-satunya baris yang mengubah jawabannya. Buku: Subbab 13.6.2 dan 13.6.3.

Kedua Hasil Sebuah Relaksasi



Gambar 13.3 Kedua hasil `RELAX(u,v,w)` dengan $w(u,v)=2$. Atas: $d[u]+w=5+2=7<d[v]=9$, sehingga $d[v]$ turun menjadi 7 dan $\pi[v]$ menjadi u. Bawah: $d[u]+w=7$ tidak kurang daripada $d[v]=6$, sehingga tidak terjadi apa-apa.

Bagian atas Gambar 13.3. Kita mengetahui rute ke v berongkos 9 dan rute ke u berongkos 5. Satu sisi lagi berbobot 2 memberi 7, yang lebih baik: ambillah, dan ingatlah bahwa kita tiba lewat u.

Bagian bawah. Rute lewat u akan berongkos 7; kita sudah punya yang berongkos 6. Pertahankan, dan biarkan $\pi[v]$ apa adanya.

Relaksasi yang tidak mengubah apa pun bukan kerja yang sia-sia: ia menegaskan bahwa estimasi yang sekarang sudah mengalahkan rute ini. Buku: Subbab 13.6.4.

Batas Atas dan Tanpa-Lintasan

LEMA 13.5 (Sifat batas atas). Sesudah inisialisasi dan sesudah *sembarang* barisan relaksasi, $d[v] \geq \delta(s,v)$ untuk setiap v. Begitu $d[v]$ mencapai $\delta(s,v)$, nilainya tidak pernah berubah lagi.

Bukti, dengan induksi atas banyaknya relaksasi. *Basis:* $d[v] = \infty \geq \delta(s,v)$ untuk $v \neq s$, dan $d[s] = 0 = \delta(s,s)$ karena tidak ada siklus negatif yang melewati s. *Langkah:* hanya $d[v]$ yang berubah, menjadi $d[u] + w(u,v) \geq \delta(s,u) + w(u,v) \geq \delta(s,v)$, menurut hipotesis lalu menurut Persamaan (13.5). ∎

Separuh pembekuannya. Perubahan berikutnya akan menurunkan $d[v]$ secara tegas ke bawah $\delta(s,v)$, dan itu mustahil. **Estimasinya tidak pernah terlalu optimistis, dan begitu benar ia tetap benar.**

LEMA 13.6 (Sifat tanpa-lintasan). Bila tidak ada lintasan dari s ke v, maka $d[v] = \delta(s,v) = \infty$ setiap saat.

Agar $d[v]$ menjadi berhingga, suatu relaksasi (u,v) harus mendapati $d[u]$ berhingga; bila ditelusuri mundur, rantainya berakhir di s, dan rantai itu *adalah* lintasan dari s ke v. Di dalam kode, lema ini adalah pengawal `d[u] < INF`.

Buku: Subbab 13.6.5 dan 13.6.6.

Kekonvergenan dan Relaksasi Lintasan

LEMA 13.7 (Sifat kekonvergenan). Misalkan $s \rightsquigarrow u \to v$ adalah lintasan terpendek, sehingga $\delta(s,v) = \delta(s,u) + w(u,v)$. Bila pada suatu saat $d[u] = \delta(s,u)$, maka kapan pun *sesudah* (u,v) direlaksasi, $d[v] = \delta(s,v)$.

Bukti. Bila $d[v]$ sudah sama dengan $\delta(s,v)$, Lema 13.5 membekukannya. Bila tidak, $d[v] > \delta(s,v) = \delta(s,u) + w(u,v) = d[u] + w(u,v)$, sehingga ujinya menyala dan menetapkan $d[v]$ tepat menjadi $\delta(s,v)$. ■ Inilah lema pekerja keras di setiap bukti kebenaran hari ini.

LEMA 13.8 (Sifat relaksasi lintasan). Misalkan $p = \langle v_0, \ldots, v_k \rangle$ lintasan terpendek dari $s = v_0$. Bila sisi-sisi p direlaksasi **secara berurutan**, dengan sembarang banyaknya relaksasi lain yang diselipkan di mana saja, maka sesudahnya $d[v_k] = \delta(s, v_k)$ untuk selamanya.

Sketsa bukti. Induksi atas i: menurut substruktur optimal $\delta(s,v_i) = \delta(s,v_{i-1}) + w(v_{i-1}, v_i)$, sehingga Lema 13.7 menyala pada tiap sisi secara bergiliran. **Selipan itu gratis**: relaksasi lain hanya menurunkan estimasi, tidak pernah ke bawah δ. Itulah yang membuat Bellman–Ford berhasil.

Hanya urutan relatifnya yang penting. Kita memerlukan (v_0, v_1) sebelum (v_1, v_2) sebelum (v_2, v_3), tidak harus berturut-turut, dan kita tidak pernah perlu tahu p yang mana.

Buku: Subbab 13.6.7 dan 13.6.8.

Subgraf Pendahulu Adalah Sebuah Pohon

DEFINISI. Subgraf pendahulu $G_\pi = (V_\pi, E_\pi)$ didefinisikan oleh

$$V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}, E_\pi = \{(\pi[v], v) : v \in V_\pi \setminus \{s\}\}. \quad (13.6)$$

Sebuah **pohon lintasan terpendek** berakar di s memuat tepat simpul-simpul yang terjangkau dari s, merupakan pohon berakar dengan akar s, dan lintasan tunggalnya dari s ke tiap v adalah lintasan terpendek di G.

Dalam kata-kata, Persamaan (13.6) mempertahankan setiap simpul yang punya pendahulu, ditambah s, lalu menggambar satu sisi ke setiap simpul itu dari pendahulunya.

LEMA 13.9 (Sifat subgraf pendahulu). Bila tidak ada siklus negatif yang terjangkau dari s, dan pada suatu saat $d[v] = \delta(s, v)$ untuk setiap v, maka G_π adalah pohon lintasan terpendek yang berakar di s.

Sketsa. *Asiklik:* menjumlahkan $d[c_i] \geq d[c_{i-1}] + w(c_{i-1}, c_i)$ mengelilingi siklus di G_π menghapus suku-suku d dan memaksa $w(\text{siklus}) \leq 0$; negatif bertentangan dengan hipotesis, dan nol disingkirkan pada Soal Latihan 13.10. *Himpunan simpulnya tepat:* $\pi[v] \neq$ NIL tepat ketika v terjangkau. *Lintasannya tepat:* menyusuri π mundur berteleskop menjadi $\delta(s, v)$. ∎

Lema ini adalah izin bagi Subbab 13.10: ikutilah π mundur dan Anda sungguh memperoleh lintasan terpendek, asalkan algoritmanya sudah berhenti. Buku: Subbab 13.6.9.

Kelima Sifatnya dalam Satu Halaman

Tabel 13.5 Kelima sifat relaksasi, dibuktikan sekali lalu dipakai oleh ketiga algoritmanya.

Sifat	Pernyataan	Apa yang diizinkannya
Batas atas (13.5)	$d[v] \geq \delta(s,v)$ selalu; begitu sama, membeku	tidak memercayai d terlalu dini; berhenti ketika tak ada yang berubah
Tanpa-lintasan (13.6)	tak ada rute $\Rightarrow d[v] = \delta(s,v) = \infty$	pengawal `d[u] < INF` di setiap program
Kekonvergenan (13.7)	$d[u] = \delta(s,u)$ dan (u,v) pada lintasan terpendek, relaksasi $(u,v) \Rightarrow d[v] = \delta(s,v)$	induksi di setiap bukti kebenaran
Relaksasi lintasan (13.8)	relaksasi sisi lintasan terpendek secara berurutan $\Rightarrow$ ujungnya terselesaikan	$\lvert V \rvert - 1$ sapuan Bellman–Ford; sapuan tunggal algoritma DAG
Subgraf pendahulu (13.9)	bila semua $d = \delta$, G_π adalah pohon lintasan terpendek	memulihkan rute dari π

Tabel 13.5 adalah halaman yang layak dipotret. Tidak ada algoritma hari ini yang memerlukan argumen kebenaran yang tidak tereduksi ke tabel ini.

Ketika diminta membuktikan kebenaran sebuah algoritma, sebutkan sifat yang Anda pakai pada tiap langkah: “menurut sifat kekonvergenan”, jangan pernah “jelas”.

Buku: Subbab 13.6.10.

Dijkstra: Gagasannya

Bobot sisi tak negatif. Diterbitkan Edsger Dijkstra pada 1959, dan konon dirancang dalam sekitar dua puluh menit di sebuah kafe di Amsterdam. Batasan ini nyata, dan Subbab 13.7.3 memperlihatkan di mana ia menggigit.

Rakus, seperti Prim. Kunci Prim bagi v adalah $\min\{w(u,v)\}$, ongkos *satu sisi*; kunci Dijkstra adalah $\min\{d[u]+w(u,v) : u \in S\}$, ongkos *seluruh perjalanan*. Bentuknya sama, kuncinya berbeda.

Seperti BFS. Bila setiap bobot sama dengan 1, "simpul belum selesai yang terdekat" adalah "yang paling sedikit sisinya", dan antrean prioritas menjadi antrean FIFO Bab 10.

Metodenya. Simpan himpunan S berisi simpul yang *sudah selesai* dan antrean prioritas Q berkunci $d[v]$. Berulang kali keluarkan simpul belum selesai yang terdekat, masukkan ke S, lalu relaksasi setiap sisi yang keluar darinya.

> **GAGASAN KUNCI.** Dijkstra memproses simpul menurut jarak sebenarnya dari s yang menaik. Begitu sebuah simpul meninggalkan Q, nilai d-nya **final**. Hal itu aman hanya karena tidak ada bobot sisi yang negatif.

Buku: Subbab 13.7.1.

Pseudokode Dijkstra

DIJKSTRA(G, w, s)

```
1  untuk tiap v ∈ V
2      lakukan d[v] ← ∞
3  d[s] ← 0
4  S ← ∅      ▷ himpunan simpul yang sudah tuntas
5  Q ← V
6  selama Q ≠ ∅
7      lakukan u ← EXTRACT-MIN(Q)
8          S ← S ∪ {u}
9          untuk tiap v ∈ Adj[u]
10             lakukan jika d[v] > d[u] + w(u, v)
11                 maka d[v] ← d[u] + w(u, v)
```

relaksasi sisi

Gambar 13.4 Algoritma Dijkstra: baris 1–3 menginisialisasi, baris 4–5 membuat himpunan selesai S yang kosong dan antrean Q, dan baris 6–8 berulang kali menyatakan simpul terdekat yang belum selesai sebagai selesai. Baris 9–11 yang disorot adalah RELAX, yang diterapkan pada setiap sisi yang keluar dari simpul itu.

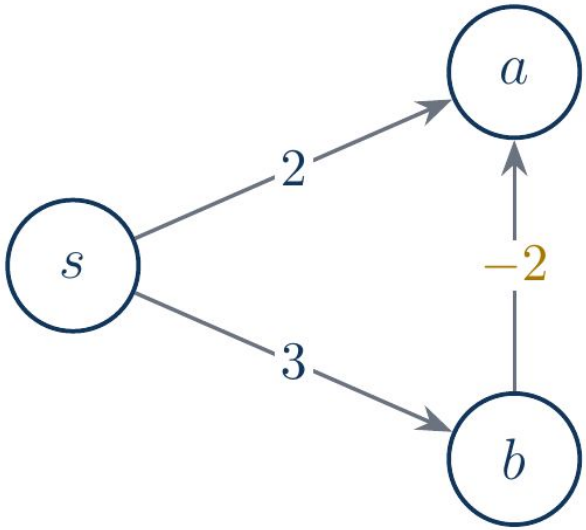
Pada Gambar 13.4: **baris 1–3** menetapkan $d[v] \leftarrow \infty$, lalu $d[s] \leftarrow 0$; **baris 4** $S \leftarrow \varnothing$; **baris 5** $Q \leftarrow V$, berkunci d. **Baris 6–8**: selama Q tidak kosong, keluarkan minimum u dan masukkan ke S; badannya berjalan $|V|$ kali dan pengeluaran pertamanya adalah s.

Baris 9–11 merelaksasi setiap sisi yang keluar dari u. *Menetapkan $d[v]$ memperbarui Q (memerlukan `Decrease-Key`).* Gambarnya tidak pernah menetapkan π; setiap implementasi menambahkan $\pi[v] \leftarrow u$ di samping baris 11.

Buku: Subbab 13.7.2.

Mengapa Bobotnya Harus Tak Negatif

Kerakusan Dijkstra bertumpu pada satu pengandaian: **menempuh satu sisi tambahan tidak pernah dapat membuat perjalanan lebih murah.** Bila ada $w<0$, jalan memutar yang panjang dapat berakhir dengan rabat yang besar.



Gambar 13.5 Sebuah graf yang di atasnya Dijkstra diam-diam mengembalikan jawaban yang salah. Tidak ada siklus, sehingga tidak ada siklus negatif, dan jarak sebenarnya adalah $\delta(s,a)=1$ dan $\delta(s,b)=3$. Dijkstra memfinalkan a pada 2 — sebelum ia pernah menengok sisi (b,a) — dan tidak pernah meninjaunya kembali.

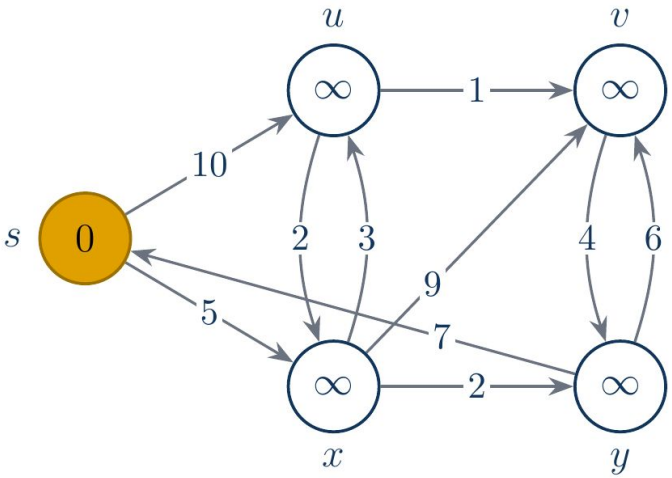
Penelusuran Gambar 13.5**.** (1) Keluarkan s pada 0; relaksasi memberi $d[a]=2$, $d[b]=3$. (2) Keluarkan a pada 2, sebab $2<3$: a **final.** (3) Keluarkan b pada 3; relaksasi (b,a) menawarkan $3+(-2)=1<2$, tetapi a sudah meninggalkan antrean. Dilaporkan 2; kebenarannya 1.

Menambahkan konstanta c pada setiap bobot tidak memperbaikinya: lintasan k sisi bertambah kc, sehingga lintasan bersisi banyak lebih dihukum dan lintasan termurahnya dapat berganti (Soal Latihan 13.6).

Program pada Subbab 13.7.3 mencetak angka 2 dari Dijkstra di samping angka 1 yang benar dari Bellman–Ford: tanpa macet, tanpa peringatan, tanpa kode galat. Buku: Subbab 13.7.3.

Panel (a) — Sesudah Inisialisasi

Simpul s, u, v, x, y; sepuluh sisi berarah: $w(s,u)=10$, $w(s,x)=5$, $w(u,v)=1$, $w(u,x)=2$, $w(x,u)=3$, $w(x,v)=9$, $w(x,y)=2$, $w(v,y)=4$, $w(y,v)=6$, $w(y,s)=7$. **Putih** = masih di Q; **kuning** = baru dikeluarkan; **biru tua** = di S; anak panah tebal adalah sisi-sisi G_π.



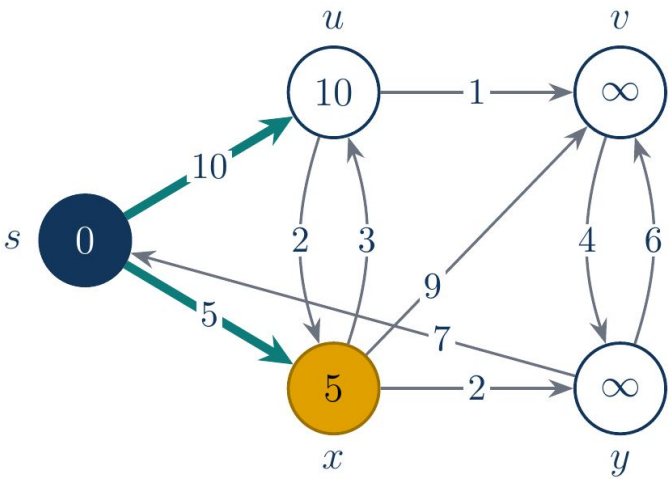
Gambar 13.6 Contoh Dijkstra, panel (a): grafnya tepat sesudah `INITIALIZE-SINGLE-SOURCE`. $d[s]=0$ (kuning, hendak dikeluarkan) dan setiap estimasi lainnya adalah ∞. $Q=\{s=0, u=\infty, v=\infty, x=\infty, y=\infty\}$.

Antrean prioritasnya, yang tersirat tetapi tidak digambar pada Gambar 13.6: $Q=\{s=0, u=\infty, v=\infty, x=\infty, y=\infty\}$ dan $S=\varnothing$.

Buku: Subbab 13.7.4.

Langkah 1 — Keluarkan s, $d=0$

(s,u), $w=10$: $\infty>0+10$, sehingga $d[u]\leftarrow 10$, $\pi[u]\leftarrow s$. (s,x), $w=5$: $\infty>0+5$, sehingga $d[x]\leftarrow 5$, $\pi[x]\leftarrow s$.



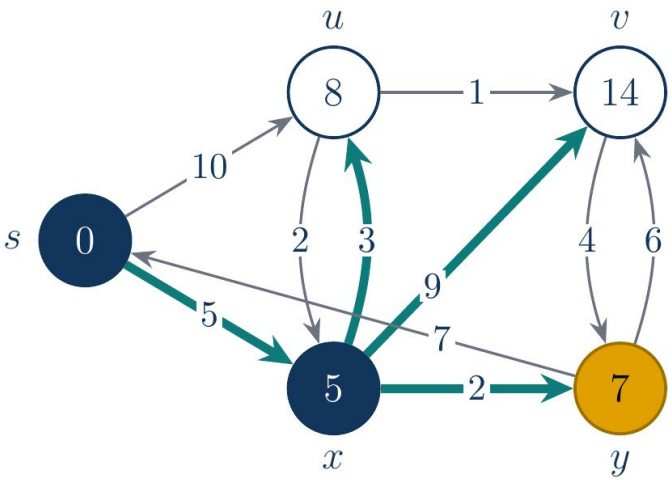
Gambar 13.7 Panel (b): s biru tua (sudah selesai), kedua sisi yang keluar darinya sudah direlaksasi, dan x — kuning — adalah simpul berikutnya yang akan dikeluarkan sebab $5<10$. Perhatikan bahwa u sudah punya estimasi 10; estimasi itu akan diperbaiki sebelum u pernah dikeluarkan.

Kini $Q=\{x=5, u=10, v=\infty, y=\infty\}$ dan $S=\{s\}$, keadaan pada Gambar 13.7.

Buku: Subbab 13.7.4, Langkah 1.

Langkah 2 — Keluarkan x, $d=5$

(x,u), $w=3$: $10>5+3=8$, sehingga $d[u]\leftarrow 8$, $\pi[u]\leftarrow x$ — **`Decrease-Key` yang pertama**. (x,v), $w=9$: $d[v]\leftarrow 14$, $\pi[v]\leftarrow x$. (x,y), $w=2$: $d[y]\leftarrow 7$, $\pi[y]\leftarrow x$.



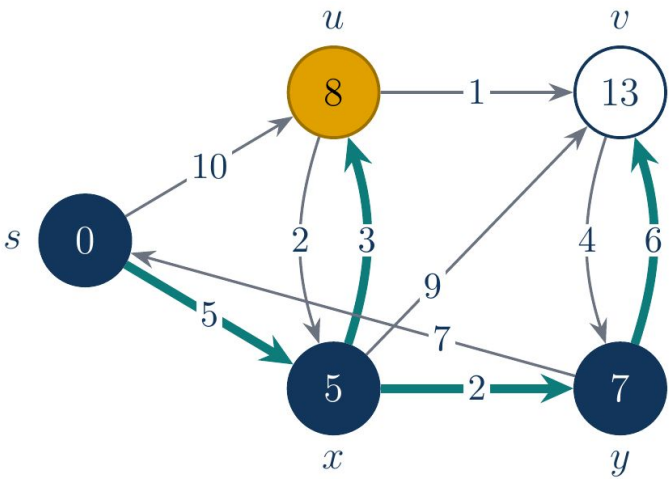
Gambar 13.8 Panel (c): s dan x biru tua. Estimasi bagi u turun dari 10 menjadi 8 dan sisi pendahulunya berayun dari (s,u) ke (x,u). y kuning: pada 7 ia kunci terkecil yang tersisa.

Kini $Q=\{y=7,u=8,v=14\}$ dan $S=\{s,x\}$: Gambar 13.8.

Buku: Subbab 13.7.4, Langkah 2.

Langkah 3 — Keluarkan y, $d=7$

(y, v), $w=6$: $14>7+6=13$, sehingga $d[v] \leftarrow 13$, $\pi[v] \leftarrow y$ — satu `Decrease-Key` lagi. (y, s), $w=7$: $7+7=14>0$, tidak terjadi apa-apa.



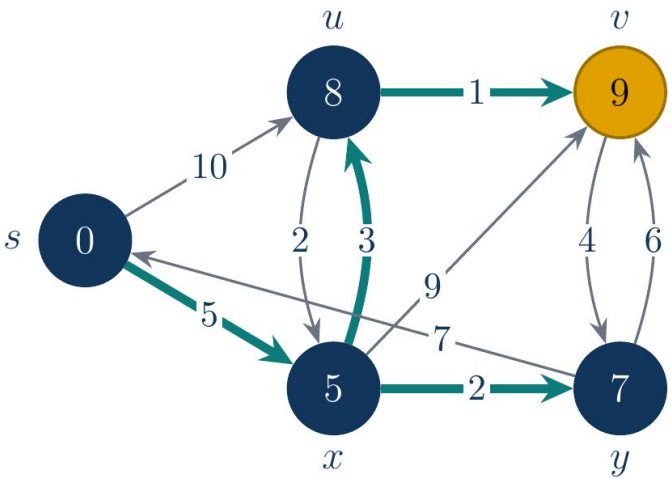
Gambar 13.9 Panel (d): s, x, dan y biru tua. Estimasi v membaik dari 14 menjadi 13 dan anak panah pendahulunya yang tebal berayun dari x ke y. u, kuning pada 8, adalah yang berikutnya.

Kini $Q=\{u=8, v=13\}$ dan $S=\{s, x, y\}$: Gambar 13.9.

Buku: Subbab 13.7.4, Langkah 3.

Langkah 4 — Keluarkan u, $d=8$

(u,v), $w=1$: $13>8+1=9$, sehingga $d[v]\leftarrow 9$, $\pi[v]\leftarrow u$ — `Decrease-Key` yang ketiga; v telah bergerak $\infty \to 14 \to 13 \to 9$.
(u,x), $w=2$: $8+2=10>5$, tidak terjadi apa-apa.



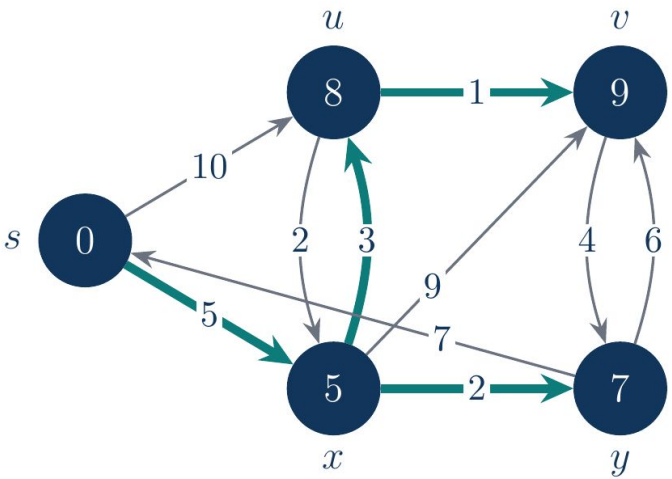
Gambar 13.10 Panel (e): empat simpul biru tua, v kuning pada 9, dan anak panah pendahulunya kini datang dari u. Setiap sisi di dalam grafnya sudah direlaksasi sekurang-kurangnya sekali kecuali kedua sisi yang meninggalkan v.

Kini $Q=\{v=9\}$ dan $S=\{s,x,y,u\}$: Gambar 13.10.

Buku: Subbab 13.7.4, Langkah 4.

Langkah 5 — Keluarkan v, $d=9$

(v, y), $w=4$: $9+4=13>7$, tidak terjadi apa-apa. Q kosong; algoritmanya berhenti.



Gambar 13.11 Panel (f): setiap simpul biru tua, algoritmanya selesai. Keempat sisi tebal (s, x), (x, u), (u, v), dan (x, y) adalah subgraf pendahulu G_π, pohon lintasan terpendek yang berakar di s.

Pada Gambar 13.11, membaca sebuah rute tinggal soal menyusuri anak panah tebal mundur dari simpul mana pun ke s. $d[v]=9$, bilangan yang ditemukan Tabel 13.1 dengan mendaftarkan enam lintasan.

Buku: Subbab 13.7.4, Langkah 5.

Seluruh Jalannya dalam Dua Tabel

Tabel 13.6 Jalannya Dijkstra selengkapnya pada graf contoh, satu baris per putaran gelung `while`.

Langkah	u Yang dikeluarkan	$d[u]$	Sisi yang direlaksasi (hasil)	Q Sesudah langkah	S Sesudahnya
0	—	—	(inisialisasi)	$s=0, u=\infty, v=\infty, x=$	$\varnothing$
1	s	0	$(s,u)\to 10$; $(s,x)\to 5$	$x=5, u=10, v=\infty, y=$	$\{s\}$
2	x	5	$(x,u)\to 8$; $(x,v)\to 14$; $(x,y)\to 7$	$y=7, u=8, v=14$	$\{s,x\}$
3	y	7	$(y,v)\to 13$; (y,s) tak berubah	$u=8, v=13$	$\{s,x,y\}$
4	u	8	$(u,v)\to 9$; (u,x) tak berubah	$v=9$	$\{s,x,y,u\}$
5	v	9	(v,y) tak berubah	kosong	semua

Tabel 13.7 Keluaran Dijkstra pada graf contoh: jarak, pendahulu, dan rute yang diperoleh dengan menyusuri π mundur.

v	s	u	v	x	y
$d[v]=\delta(s,v)$	0	8	9	5	7
$\pi[v]$	NIL	x	u	s	x
lintasan terpendek	s	s,x,u	s,x,u,v	s,x	s,x,y

Pada Tabel 13.6 kunci di dalam Q hanya pernah turun, sedangkan kunci yang dikeluarkan, $0, 5, 7, 8, 9$, tidak pernah turun. Tabel 13.7 cocok dengan Tabel 13.1.

Buku: Subbab 13.7.5.

Programnya

`priority_queue` pustaka baku tidak memiliki `DecreaseKey`, sehingga Kode 13.1 memakai **penghapusan malas**: ketika $d[v]$ turun, dorong salinan kedua v dengan kunci yang lebih kecil dan buang salinan basi saat dikeluarkan. Antreannya memuat hingga $|E|$ entri, sehingga ongkosnya $O(E \lg E) = O(E \lg V)$.

Kode 13.1 Algoritma Dijkstra dengan antrean prioritas *binary heap* pada graf contoh, yang mencetak antreannya pada setiap langkah. Kompilasilah dengan `g++ -std=c++17 -O2 -o dijkstra dijkstra.cpp`.

```cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <queue>
#include <algorithm>
#include <climits>

using namespace std;

const long long INF = LLONG_MAX / 4;          // "tak hingga" yang masih aman kita tambahi

struct Edge { int to; long long w; };

// Graf contoh Dijkstra: simpul s,u,v,x,y.
const vector<string> NAME = {"s", "u", "v", "x", "y"};
enum { S = 0, U = 1, V = 2, X = 3, Y = 4 };

// Cetak antrean prioritasnya tanpa merusaknya: kita menyalinnya lalu menguras salinannya.
// Entri basi (simpul yang sudah di dalam S, atau kunci yang dikalahkan DecreaseKey
// berikutnya) dilewati, sehingga yang tercetak adalah isi logis Q.
string queueContents(priority_queue<pair<long long,int>,
                                    vector<pair<long long,int>>,
                                    greater<pair<long long,int>>> q,
                     const vector<bool>& done, const vector<long long>& d) {
    vector<pair<long long,int>> live;
    vector<bool> seen(done.size(), false);
    while (!q.empty()) {
```

```
        auto [key, v] = q.top(); q.pop();
        if (done[v] || seen[v] || key > d[v]) continue;
        seen[v] = true;
        live.push_back({key, v});
    }
    sort(live.begin(), live.end());
    string out = "{";
    for (size_t i = 0; i < live.size(); ++i) {
        if (i) out += ", ";
        out += NAME[live[i].second] + "=" +
               (live[i].first >= INF ? string("INF") : to_string(live[i].first));
    }
    return out + "}";
}
```

Kode 13.1 (Lanjutan) *Algoritma Dijkstra dengan antrean prioritas binary heap pada graf contoh, yang mencetak antreannya pada setiap langkah. Kompilasilah dengan g++ -std=c++17 -O2 -o dijkstra dijkstra.cpp.*

```
int main() {
    int n = 5;
    vector<vector<Edge>> adj(n);
    auto add = [&](int a, int b, long long w) { adj[a].push_back({b, w}); };
    add(S, U, 10); add(S, X, 5);
    add(U, V, 1);  add(U, X, 2);
    add(X, U, 3);  add(X, V, 9); add(X, Y, 2);
    add(V, Y, 4);
    add(Y, V, 6);  add(Y, S, 7);

    vector<long long> d(n, INF);
    vector<int> pi(n, -1);
    vector<bool> done(n, false);                    // done[v] == "v berada di dalam S"
    d[S] = 0;

    priority_queue<pair<long long,int>, vector<pair<long long,int>>,
                   greater<pair<long long,int>>> Q;
    for (int v = 0; v < n; ++v) Q.push({d[v], v});

    cout << "initial Q = " << queueContents(Q, done, d) << "\n\n";

    int step = 0;
    while (!Q.empty()) {
        auto [key, u] = Q.top(); Q.pop();
        if (done[u]) continue;                      // salinan basi: abaikan saja
        if (key > d[u]) continue;                   // digantikan oleh kunci yang lebih kecil
        done[u] = true;
        cout << "step " << ++step << ": EXTRACT-MIN gives " << NAME[u]
```

```
             << " with d=" << (d[u] >= INF ? string("INF") : to_string(d[u])) << "\n";
        if (d[u] >= INF) { cout << "    unreachable; no edges relaxed\n\n"; continue; }
        for (const Edge& e : adj[u]) {
            long long cand = d[u] + e.w;
            cout << "    relax (" << NAME[u] << "," << NAME[e.to] << ") w=" << e.w
                 << ": d[" << NAME[e.to] << "]="
                 << (d[e.to] >= INF ? string("INF") : to_string(d[e.to]))
                 << " vs " << d[u] << "+" << e.w << "=" << cand << " -> ";
            if (cand < d[e.to]) {
                d[e.to] = cand; pi[e.to] = u;
                Q.push({cand, e.to});
                cout << "improved, d[" << NAME[e.to] << "]=" << cand
                     << ", pi[" << NAME[e.to] << "]=" << NAME[u] << "\n";
            } else {
                cout << "no change\n";
            }
        }
        cout << "    Q = " << queueContents(Q, done, d) << "\n\n";
```

Kode 13.1 (Lanjutan) *Algoritma Dijkstra dengan antrean prioritas binary heap pada graf contoh, yang mencetak antreannya pada setiap langkah. Kompilasilah dengan g++ -std=c++17 -O2 -o dijkstra dijkstra.cpp.*

```
    }

    cout << "vertex |  d  | pi | shortest path from s\n";
    cout << "-------+-----+----+----------------------\n";
    for (int v = 0; v < n; ++v) {
        vector<string> path;
        for (int t = v; t != -1; t = pi[t]) path.push_back(NAME[t]);
        reverse(path.begin(), path.end());
        string p;
        for (size_t i = 0; i < path.size(); ++i) { if (i) p += " -> "; p += path[i]; }
        cout << "   " << NAME[v] << "   |" << setw(4) << d[v] << " |  "
             << (pi[v] == -1 ? string("-") : NAME[pi[v]]) << " | " << p << "\n";
    }
    return 0;
}
```

Buku: Subbab 13.7.6.

Apa yang Dicetaknya

```
initial Q = {s=0, u=INF, v=INF, x=INF, y=INF}

step 1: EXTRACT-MIN gives s with d=0
    relax (s,u) w=10: d[u]=INF vs 0+10=10 -> improved, d[u]=10, pi[u]=s
    relax (s,x) w=5: d[x]=INF vs 0+5=5 -> improved, d[x]=5, pi[x]=s
    Q = {x=5, u=10, v=INF, y=INF}

step 2: EXTRACT-MIN gives x with d=5
    relax (x,u) w=3: d[u]=10 vs 5+3=8 -> improved, d[u]=8, pi[u]=x
    relax (x,v) w=9: d[v]=INF vs 5+9=14 -> improved, d[v]=14, pi[v]=x
    relax (x,y) w=2: d[y]=INF vs 5+2=7 -> improved, d[y]=7, pi[y]=x
    Q = {y=7, u=8, v=14}

step 3: EXTRACT-MIN gives y with d=7
    relax (y,v) w=6: d[v]=14 vs 7+6=13 -> improved, d[v]=13, pi[v]=y
    relax (y,s) w=7: d[s]=0 vs 7+7=14 -> no change
    Q = {u=8, v=13}

step 4: EXTRACT-MIN gives u with d=8
    relax (u,v) w=1: d[v]=13 vs 8+1=9 -> improved, d[v]=9, pi[v]=u
    relax (u,x) w=2: d[x]=5 vs 8+2=10 -> no change
    Q = {v=9}

step 5: EXTRACT-MIN gives v with d=9
    relax (v,y) w=4: d[y]=7 vs 9+4=13 -> no change
    Q = {}

vertex |  d  | pi | shortest path from s
```

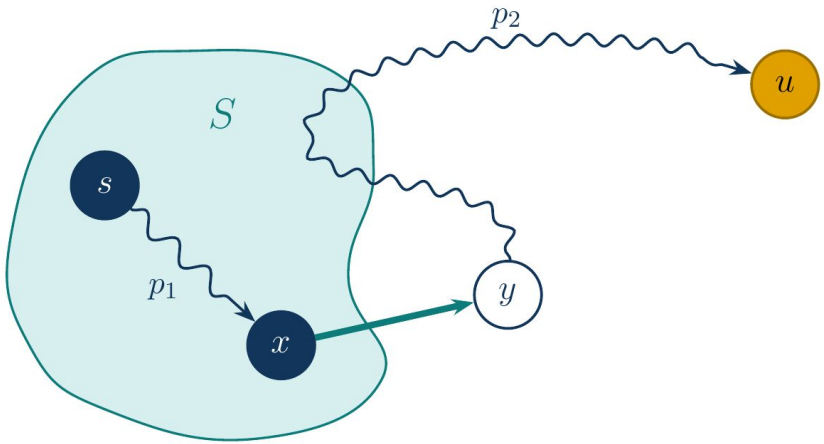
```
-------+-----+----+----------------------
   s   |   0 |  - | s
   u   |   8 |  x | s -> x -> u
   v   |   9 |  u | s -> x -> u -> v
   x   |   5 |  s | s -> x
   y   |   7 |  x | s -> x -> y
```

Bandingkan baris demi baris dengan keenam panel dan dengan Tabel 13.6: tiga momen `Decrease-Key` yang sama dan tiga relaksasi yang sama-sama tidak mengubah apa pun.

Buku: Subbab 13.7.7.

Kebenaran: Klaim dan Gambarnya

TEOREMA 13.10. Jalankan Dijkstra dengan $w(u,v) \geq 0$ untuk setiap sisi. Di awal setiap putaran gelung `while`, $d[v] = \delta(s,v)$ untuk setiap $v \in S$. Karena itu pada saat berhenti $d[v] = \delta(s,v)$ untuk setiap v, dan G_π adalah pohon lintasan terpendek yang berakar di s.



Gambar 13.12 Gambar di balik bukti kebenaran Dijkstra: wilayah hijau-biru muda adalah himpunan selesai S, dan u adalah simpul pertama yang konon dikeluarkan dengan estimasi yang salah. Lintasan terpendek yang sebenarnya meninggalkan S lewat sisi (x, y), sehingga berbentuk $s \overset{p_1}{\rightsquigarrow} x \to y \overset{p_2}{\rightsquigarrow} u$.

Buktinya dengan kontradiksi dan berlangsung di dalam Gambar 13.12: *setiap kali u dimasukkan ke S, $d[u]=\delta(s,u)$, dan kesamaan itu terjaga sesudahnya.*

Buku: Subbab 13.7.8.

Buktinya, dalam Enam Langkah

Persiapan. Misalkan u simpul **pertama** yang dikeluarkan dengan $d[u] \neq \delta(s,u)$; menurut Lema 13.5,

$$d[u] > \delta(s,u). \tag{13.7}$$

1. $u \neq s$, sebab $d[s] = 0 = \delta(s,s)$. **2.** u terjangkau, sebab bila tidak $d[u] = \infty = \delta(s,u)$, bertentangan dengan Persamaan (13.7).

3. Uraikan pada batas S**:** y adalah simpul pertama lintasan terpendek p yang tidak di S, dan $x \in S$ pendahulunya pada p. **4.** $d[y] = \delta(s,y)$**:** x dikeluarkan lebih dahulu, sehingga $d[x] = \delta(s,x)$; (x,y) direlaksasi saat itu; menurut substruktur optimal $\delta(s,y) = \delta(s,x) + w(x,y)$; Lema 13.7 berlaku.

5. $\delta(s,u) = \delta(s,y) + w(p_2)$ dan setiap sisi p_2 tak negatif, sehingga

$$\delta(s,y) \leq \delta(s,u). \tag{13.8}$$

> **Langkah 5 adalah satu-satunya langkah yang memakai** $w \geq 0$**.** Bila p_2 dapat berbobot kurang dari nol, rute sebenarnya ke u dapat melewati simpul yang *lebih jauh* daripada u sendiri, persis seperti pada Gambar 13.5.

6. Kontradiksi. Menurut langkah 4, Persamaan (13.8), dan Persamaan (13.7): $d[y] = \delta(s,y) \leq \delta(s,u) < d[u]$. Keduanya berada di Q dan algoritmanya mengeluarkan **minimum**, yang menuntut $d[u] \leq d[y]$. ∎

Buku: Subbab 13.7.8.

Kunci yang Dikeluarkan Tidak Pernah Turun

AKIBAT 13.11. Barisan kunci yang dikembalikan oleh panggilan `Extract-Min` berturut-turut pada Dijkstra tidak turun.

Bukti, dua pengeluaran berurutan u lalu u'. Ketika u dikeluarkan, setiap kunci di Q bernilai $\geq d[u]$. Relaksasi (u, v) menetapkan $d[v] \leftarrow d[u] + w(u, v) \geq d[u]$, karena $w \geq 0$. Jadi setiap kunci tetap $\geq d[u]$, dan $d[u'] \geq d[u]$. ▰

Pada penelusuran kita kunci yang dikeluarkan adalah $0, 5, 7, 8, 9$: monoton, sesuai janji.

Keluar lebih awal. Untuk persoalan pasangan tunggal Anda boleh berhenti begitu t dikeluarkan: setiap kunci sesudahnya sekurang-kurangnya sebesar itu, sehingga tak ada relaksasi berikutnya yang dapat menurunkan $d[t]$.

Dengan sisi negatif, relaksasi dapat menanam kunci di bawah $d[u]$ dan pengeluarannya berjalan mundur, seperti pada graf biaya Profitina Laundry di Subbab 13.12.2. Buku: Subbab 13.7.8.

Waktu Jalannya

Hitunglah operasi antreannya: $|V|$ kali `Insert`, $|V|$ kali `Extract-Min` (satu per putaran), dan paling banyak $|E|$ kali `Decrease-Key` (tiap sisi diperiksa sekali, ketika pangkalnya dikeluarkan):

$$T = |V| \cdot T_{Extract\text{-}Min} + |E| \cdot T_{Decrease\text{-}Key}. \tag{13.9}$$

Tabel 13.8 Waktu jalan Dijkstra untuk tiga implementasi antrean prioritas, diperoleh dengan menyulihkan ongkos tiap antrean ke dalam hitungan operasinya.

Q Diimplementasikan sebagai	*T*(Extract-Min)	*T*(Decrease-Key)	Total
larik tak terurut	$\Theta(V)$	$\Theta(1)$	$\Theta(V^2)$
binary heap	$\Theta(\lg V)$	$\Theta(\lg V)$	$O((V+E)\lg V) = O(E$
Fibonacci heap	$O(\lg V)$ teramortisasi	$\Theta(1)$ teramortisasi	$O(V \lg V + E)$

Tabel 13.8 menyulihkan ke dalam Persamaan (13.9). **Larik** untuk graf *padat*: tanpa beban *heap*, dan $\Theta(V^2)$ tidak dapat dikalahkan bila ada $\Theta(V^2)$ sisi yang harus dibaca. ***Binary heap*** untuk graf *jarang*. ***Fibonacci heap***: batas terbaik, tetapi konstantanya besar; pakailah hanya sesudah mengukur.

Diukur (Subbab 13.7.10): pada graf jarang ($m = 8n$) *heap* empat sampai dua belas kali lebih cepat; pada graf padat ($m = n^2/4$) keduanya berdekatan. Tiga implementasi sepakat pada 2000 graf acak.

Titik silang antara larik dan heap berada kira-kira pada $E = \Theta(V^2 / \lg V)$. *Buku: Subbab 13.7.9 dan 13.7.10.*

BFS, Prim, dan Dijkstra

Tabel 13.9 Penelusuran melebar, Prim, dan Dijkstra dibandingkan: kerangka yang sama dengan tiga kunci yang berbeda.

	BFS (Bab 10)	Prim–Jarník (Bab 11)	Dijkstra (Bab 13)
Pertanyaan yang dijawab	sisi paling sedikit dari s	pohon merentang termurah	rute termurah dari s
Struktur perbatasan	antrean FIFO	antrean prioritas	antrean prioritas
Kunci v	(tidak ada; urutan kedatangan)	$\min\{w(u,v) : u \in \text{pohon}\}$	$\min\{d[u]+w(u,v) : u \in S\}$
Aturan pembaruan	masukkan antrean bila putih	$key[v] \leftarrow w(u,v)$	$d[v] \leftarrow d[u]+w(u,v)$
Perlu $w \geq 0$?	tidak berbobot sama sekali	tidak — bobot apa pun boleh	**ya**
Arah graf	keduanya	tak berarah	berarah (atau keduanya)
Keluaran	pohon BFS + cacah lompatan	pohon merentang minimum	pohon lintasan terpendek + jarak
Waktu (*binary heap*)	$O(V+E)$	$O(E \lg V)$	$O(E \lg V)$

Baris Tabel 13.9 yang memisahkan Prim dari Dijkstra adalah kuncinya: Prim mengukur **satu sisi**, Dijkstra mengukur **seluruh perjalanan**.

Itu pula sebabnya Prim tahan terhadap bobot negatif sedangkan Dijkstra tidak: Prim tidak pernah *menjumlahkan* dua bobot, sehingga tidak pernah memerlukan ketaksamaan segitiga. Kerangkanya sama, tulangnya berbeda.

Buku: Subbab 13.7.11.

Bellman–Ford: Mengapa dan Bagaimana

Algoritma Dijkstra tidak berhasil bila ada sisi negatif, sebab *kita tidak dapat bersikap rakus dengan mengandaikan panjang lintasan hanya akan bertambah di masa depan.* Bellman (1958), Ford (1956), dan Moore (1957) menemukan obat yang sama: **tinggalkan kerakusan**, relaksasi *setiap* sisi $|V|-1$ kali, dan biarkan Lema 13.8 mengerjakan sisanya.

BELLMAN-FORD(G, w, s)

01 **untuk tiap** $v \in V[G]$
02 $\quad d[v] \leftarrow \infty$
03 $\quad \pi[v] \leftarrow$ NIL
04 $d[s] \leftarrow 0$
05 **untuk** $i \leftarrow 1$ **sampai** $|V[G]| - 1$ **lakukan**
06 $\quad$ **untuk tiap sisi** $(u, v) \in E[G]$ **lakukan**
07 $\quad\quad$ RELAX(u, v, w)
08 **untuk tiap sisi** $(u, v) \in E[G]$ **lakukan**
09 $\quad$ **jika** $d[v] > d[u] + w(u, v)$ **maka kembalikan** *false*
10 **kembalikan** *true*

Gambar 13.13 Bellman–Ford: baris 01–04 menginisialisasi, baris 05–07 menjalankan $|V| - 1$ sapuan yang masing-masing merelaksasi setiap sisi, dan baris 08–09 adalah pendeteksinya, yang mengembalikan `false` bila masih ada sisi yang dapat direlaksasi. Baris 10 mengembalikan `true` ketika jawabannya layak dipercaya.

Gambar 13.13 tidak memiliki antrean maupun pengurutan, karena algoritmanya tidak mengambil keputusan apa pun. Baris 01–04 persis `INITIALIZE-SINGLE-SOURCE`.

Sebagai ganti kelambatannya, ia benar pada sisi negatif dan dapat mendeteksi siklus negatif alih-alih mengembalikan omong kosong. Buku: Subbab 13.8.1 dan 13.8.2.

Mengapa $|V|-1$ Sapuan Sudah Cukup

TEOREMA 13.12. Tanpa siklus negatif yang terjangkau, sesudah sapuan ke-i berlaku $d[v]=\delta(s,v)$ untuk setiap v yang lintasan terpendeknya memakai paling banyak i sisi. Karena itu sesudah $|V|-1$ sapuan, $d[v]=\delta(s,v)$ untuk setiap v.

Bukti. Menurut Teorema 13.4 ada lintasan terpendek *sederhana* $\langle v_0,\ldots,v_k\rangle$ dengan $k\le|V|-1$. Sapuan ke-i merelaksasi setiap sisi, khususnya (v_{i-1},v_i). Sesudah k sapuan sisi-sisinya telah direlaksasi secara berurutan, dan Lema 13.8 memberi $d[v_k]=\delta(s,v_k)$. ∎

Bentuk cacah sisi (Teorema 13.12′). Misalkan $\delta_i(s,u)$ bobot termurah atas lintasan *paling banyak i sisi*. Lintasan terbaik semacam itu memakai $\le i-1$ sisi, atau berakhir dengan sisi (z,u):

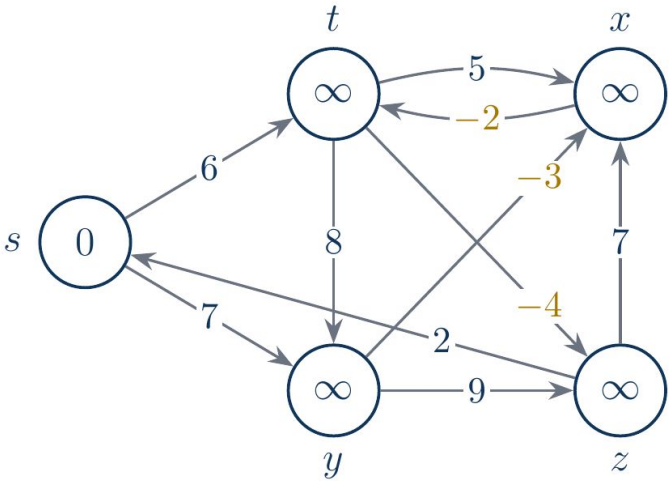
$$\delta_i(s,u)=\min\Big(\delta_{i-1}(s,u),\ \min_{(z,u)\in E}\big(\delta_{i-1}(s,z)+w(z,u)\big)\Big). \tag{13.10}$$

Sapuan ke-i merelaksasi setiap sisi, sehingga menangkap kedua kasus Persamaan (13.10): sesudah sapuan ke-i, $\delta(s,u)\le d[u]\le\delta_i(s,u)$. Karena $\delta_{n-1}=\delta$ (Teorema 13.4), kedua ujungnya bertemu.

Mengapa ketaksamaan? Satu sapuan dapat membawa sebuah estimasi melintasi beberapa sisi sekaligus bila daftar sisinya kebetulan berurutan baik; $|V|-1$ adalah kasus terburuknya. Buku: Subbab 13.8.3.

Bellman–Ford: Panel (a)

Simpul s, t, x, y, z; sepuluh sisi, **tiga negatif**, direlaksasi dalam urutan ini: $w(t,x)=5$, $w(t,y)=8$, $w(t,z)=-4$, $w(x,t)=-2$, $w(y,x)=-3$, $w(y,z)=9$, $w(z,x)=7$, $w(z,s)=2$, $w(s,t)=6$, $w(s,y)=7$. Sisi yang keluar dari s datang **paling akhir**: sengaja kasus terburuk.



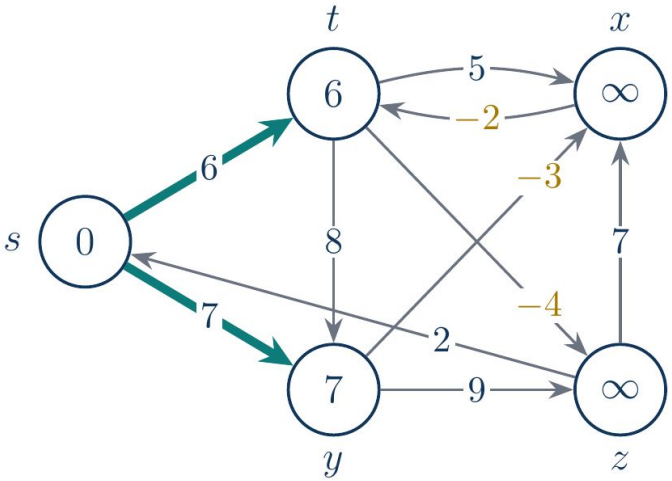
Gambar 13.14 Contoh Bellman–Ford, panel (a): inisialisasi. Tiga dari kesepuluh sisinya berbobot negatif, dan itulah persis sebabnya Dijkstra tidak boleh dipakai di sini. Namun tidak ada *siklus* negatif: $t \to z \to s \to t$ berongkos $-4+2+6=4>0$, dan $t \to x \to t$ berongkos $5+(-2)=3>0$.

Periksalah kedua siklus yang disebut pada Gambar 13.14 sebelum mulai: sisi negatif tidak apa-apa; siklus negatif akan membuat pertanyaannya tak bermakna.

Buku: Subbab 13.8.4.

Sapuan 1 — Hanya Sisi s yang Menggigit

Delapan sisi pertama tidak berbuat apa-apa: pangkalnya masih bernilai ∞. Lalu (s,t), $w=6$: $d[t] \leftarrow 6$, $\pi[t] \leftarrow s$; dan (s,y), $w=7$: $d[y] \leftarrow 7$, $\pi[y] \leftarrow s$.



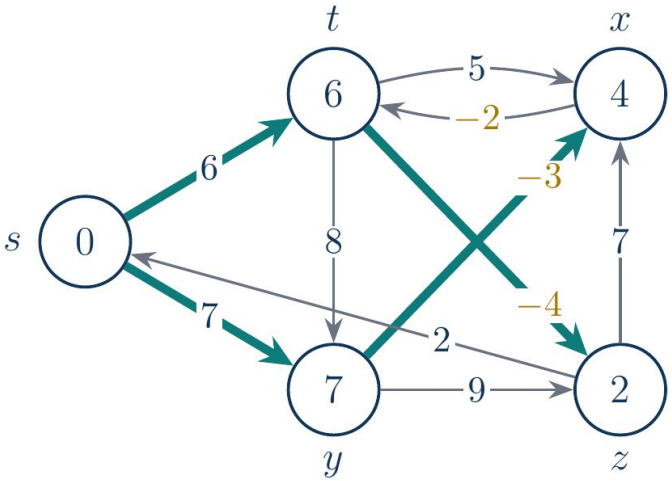
Gambar 13.15 Panel (b): sesudah sapuan 1. Hanya kedua sisi yang meninggalkan s yang berpengaruh, sebab hanya merekalah yang pangkalnya punya estimasi berhingga ketika mereka diperiksa.

Sesudah sapuan 1: $d = (s:0, t:6, x:\infty, y:7, z:\infty)$, seperti pada Gambar 13.15.

Buku: Subbab 13.8.4, Sapuan 1.

Sapuan 2 — Dua Sisi Negatif Membayar

(t, x): $d[x] \leftarrow 6+5=11$. (t, y): $14>7$, tidak. (t, z): $d[z] \leftarrow 6-4=2$. (x, t): $11-2=9>6$, tidak. (y, x): $11>7-3=4$, sehingga $d[x] \leftarrow 4$, $\pi[x] \leftarrow y$: **diperbaiki dua kali dalam satu sapuan**. Selebihnya tidak mengubah apa pun.



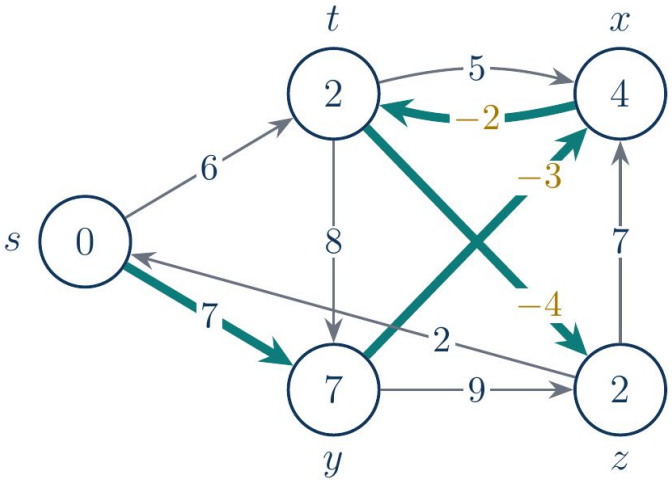
Gambar 13.16 Panel (c): sesudah sapuan 2. Setiap simpul kini punya estimasi berhingga. $d[x]$ mendarat di 4 lewat y dan sisi negatif (y, x), dan $d[z]$ di 2 lewat t dan sisi negatif (t, z). Dua dari kelima estimasinya masih salah.

Sesudah sapuan 2: $d=(0, 6, 4, 7, 2)$, seperti pada Gambar 13.16.

Buku: Subbab 13.8.4, Sapuan 2.

Sapuan 3 — Satu Sisi, Tiga Lompatan Dalam

Tepat satu relaksasi berhasil: (x, t), $w = -2$: $6 > 4 - 2 = 2$, sehingga $d[t] \leftarrow 2$, $\pi[t] \leftarrow x$. Rute $s \to y \to x \to t$ berongkos $7 + (-3) + (-2) = 2$ dan mengalahkan sisi langsung berbobot 6.



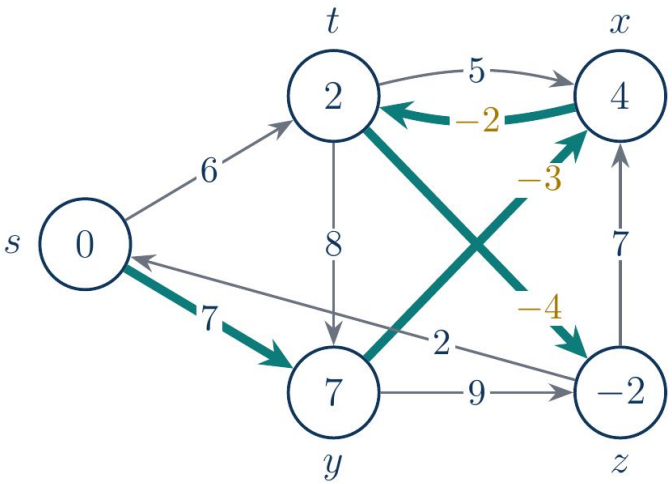
Gambar 13.17 Panel (d): sesudah sapuan 3. Estimasi bagi t turun dari 6 menjadi 2, sebab rute $s \to y \to x \to t$ berongkos $7 + (-3) + (-2) = 2$, yang mengalahkan sisi langsung berbobot 6. Anak panah pendahulunya yang tebal berayun dari s ke x.

Sesudah sapuan 3: $d = (0, 2, 4, 7, 2)$. Perbaikan pada Gambar 13.17 tak terlihat sampai sapuan 3 sebab ia memerlukan tiga sisi.

Buku: Subbab 13.8.4, Sapuan 3.

Sapuan 4 — Jawaban Akhirnya

Lagi-lagi tepat satu relaksasi berhasil: (t, z), $w = -4$: $2 > 2 - 4 = -2$, sehingga $d[z] \leftarrow -2$; $\pi[z]$ tetap t.



Gambar 13.18 Panel (e), sesudah sapuan 4: $\delta(s, z) = -2$ lewat rute empat sisi $s \to y \to x \to t \to z$, berbobot $7 + (-3) + (-2) + (-4)$. Empat sisi adalah sebanyak-banyaknya sisi lintasan sederhana di dalam graf lima simpul, sehingga keempat sapuan $|V| - 1 = 4$ diperlukan.

Sapuan pendeteksi pada Gambar 13.18: (y, x) memberi $4 = d[x]$, (x, t) memberi $2 = d[t]$, (t, z) memberi $-2 = d[z]$, (z, s) memberi $0 = d[s]$. Tidak ada yang membaik; algoritmanya mengembalikan `true`.

Buku: Subbab 13.8.4, Sapuan 4 dan sapuan pendeteksi.

Jalannya, Sapuan demi Sapuan

Tabel 13.10 Jalannya Bellman–Ford selengkapnya pada graf contoh, satu baris per sapuan atas kesepuluh sisinya; hanya nilai pada akhir tiap sapuan yang diperlihatkan.

Sesudah	$d[s]$	$d[t]$	$d[x]$	$d[y]$	$d[z]$	Yang berubah
inisialisasi	0	∞	∞	∞	∞	—
sapuan 1	0	6	∞	7	∞	$(s,t),(s,y)$
sapuan 2	0	6	4	7	2	$(t,x)\to 11,(t,z)\to 2,(y,x)\to 4$
sapuan 3	0	2	4	7	2	$(x,t)\to 2$
sapuan 4	0	2	4	7	-2	$(t,z)\to -2$
deteksi	0	2	4	7	-2	tidak ada — mengembalikan `true`

Tabel 13.11 Keluaran Bellman–Ford pada graf contoh. Tiga dari keempat rute tak sepelenya memakai y dan sisi negatif (y,x).

v	s	t	x	y	z
$\delta(s,v)$	0	2	4	7	-2
$\pi[v]$	NIL	x	y	s	t
lintasan terpendek	s	s,y,x,t	s,y,x	s,y	s,y,x,t,z

Pada Tabel 13.10, $d[x]$ menjadi 11 lalu 4 *di dalam sapuan 2*. Tabel 13.11 adalah struktur yang mudah terlewat oleh metode rakus yang memfinalkan simpul terlalu dini.

Buku: Subbab 13.8.5.

Programnya

Kode 13.2 mencetak setiap sapuan. Separuh keduanya menurunkan $w(z,s)$ dari 2 menjadi -8, sehingga tercipta siklus $s \to t \to z \to s$ berbobot $6+(-4)+(-8)=-6$, dan pendeteksinya menyala.

Kode 13.2 Bellman–Ford dengan pendeteksian siklus negatif pada graf contoh, yang mencetak setiap sapuannya; separuh kedua `main` menciptakan siklus negatif dan memperlihatkan pendeteksinya menyala. Kompilasilah dengan `g++ -std=c++17 -O2 -o bellman bellman.cpp`.

```cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <algorithm>
#include <climits>

using namespace std;

const long long INF = LLONG_MAX / 4;

struct Edge { int u, v; long long w; };

struct Graph {
    vector<string> name;
    vector<Edge>   edges;
    int n() const { return (int)name.size(); }
    int id(const string& s) const {
        return (int)(find(name.begin(), name.end(), s) - name.begin());
    }
};

string show(long long x) { return x >= INF ? "INF" : to_string(x); }

void printRow(const string& tag, const Graph& g, const vector<long long>& d) {
    cout << setw(9) << left << tag << right;
    for (int v = 0; v < g.n(); ++v)
        cout << setw(6) << (g.name[v] + "=" + show(d[v]));
```

```
    cout << "\n";
}

// Mengembalikan true ketika tidak ada siklus negatif yang terjangkau dari s.
// d dan pi terisi bagaimanapun juga.
bool bellmanFord(const Graph& g, int s, vector<long long>& d, vector<int>& pi,
                 bool verbose) {
    d.assign(g.n(), INF);
    pi.assign(g.n(), -1);
    d[s] = 0;
    if (verbose) printRow("init", g, d);
```

Kode 13.2 (Lanjutan) *Bellman–Ford dengan pendeteksian siklus negatif pada graf contoh, yang mencetak setiap sapuannya; separuh kedua main menciptakan siklus negatif dan memperlihatkan pendeteksinya menyala. Kompilasilah dengan g++ -std=c++17 -O2 -o bellman bellman.cpp.*

```
    for (int pass = 1; pass <= g.n() - 1; ++pass) {
        bool changed = false;
        for (const Edge& e : g.edges) {
            if (d[e.u] < INF && d[e.u] + e.w < d[e.v]) {   // RELAX(u,v,w)
                d[e.v] = d[e.u] + e.w;
                pi[e.v] = e.u;
                changed = true;
                if (verbose)
                    cout << "   pass " << pass << ": relax ("
                         << g.name[e.u] << "," << g.name[e.v] << ") w=" << e.w
                         << " -> d[" << g.name[e.v] << "]=" << d[e.v] << "\n";
            }
        }
        if (verbose) printRow("pass " + to_string(pass), g, d);
        if (!changed) {                     // tak ada yang bergerak: sapuan berikutnya tak mungkin
menggerakkan apa pun
            if (verbose) cout << "   (no change in this pass; the rest are skipped)\n";
            break;
        }
    }

    for (const Edge& e : g.edges)           // sapuan ke-|V|: pendeteksinya
        if (d[e.u] < INF && d[e.u] + e.w < d[e.v]) {
            if (verbose)
                cout << "   edge (" << g.name[e.u] << "," << g.name[e.v]
                     << ") still relaxes: " << d[e.u] << "+" << e.w
                     << " < " << d[e.v] << "  => NEGATIVE CYCLE\n";
            return false;
```

```
        }
    return true;
}

int main() {
    // Graf contoh Bellman-Ford, sisi dalam urutan yang dipakai pada penelusuran.
    Graph g;
    g.name = {"s", "t", "x", "y", "z"};
    auto E = [&](const char* a, const char* b, long long w) {
        g.edges.push_back({g.id(a), g.id(b), w});
    };
    E("t","x", 5); E("t","y", 8); E("t","z",-4);
    E("x","t",-2);
    E("y","x",-3); E("y","z", 9);
    E("z","x", 7); E("z","s", 2);
    E("s","t", 6); E("s","y", 7);
```

Kode 13.2 (Lanjutan) *Bellman–Ford dengan pendeteksian siklus negatif pada graf contoh, yang mencetak setiap sapuannya; separuh kedua main menciptakan siklus negatif dan memperlihatkan pendeteksinya menyala. Kompilasilah dengan g++ -std=c++17 -O2 -o bellman bellman.cpp.*

```
    cout << "=== the example graph ===\n";
    vector<long long> d; vector<int> pi;
    bool ok = bellmanFord(g, g.id("s"), d, pi, true);
    cout << "returns " << (ok ? "true" : "false") << "\n\n";

    cout << "vertex |  d  | pi | shortest path from s\n";
    cout << "-------+-----+----+---------------------\n";
    for (int v = 0; v < g.n(); ++v) {
        vector<string> path;
        for (int t = v; t != -1; t = pi[t]) path.push_back(g.name[t]);
        reverse(path.begin(), path.end());
        string p;
        for (size_t i = 0; i < path.size(); ++i) { if (i) p += " -> "; p += path[i]; }
        cout << "   " << g.name[v] << "   |" << setw(4) << d[v] << " |  "
             << (pi[v] == -1 ? string("-") : g.name[pi[v]]) << " | " << p << "\n";
    }

    // Graf yang sama, tetapi w(z,s) diturunkan dari 2 menjadi -8. Kini s->t->z->s
    // berongkos 6 + (-4) + (-8) = -6 < 0: sebuah siklus negatif.
    cout << "\n=== the same graph with w(z,s) = -8 ===\n";
    for (Edge& e : g.edges)
        if (e.u == g.id("z") && e.v == g.id("s")) e.w = -8;
    ok = bellmanFord(g, g.id("s"), d, pi, false);
    cout << "BELLMAN-FORD returns " << (ok ? "true" : "false") << "\n";
    return 0;
}
```

Pengawal `d[e.u] < INF` *adalah Lema 13.6 dalam bentuk kode, dan* `if (!changed) break;` *adalah jalan keluar dini yang gratis; sapuan pendeteksinya tetap dijalankan sesudahnya. Buku: Subbab 13.8.6.*

Apa yang Dicetaknya

```
=== the example graph ===
init        s=0 t=INF x=INF y=INF z=INF
   pass 1: relax (s,t) w=6 -> d[t]=6
   pass 1: relax (s,y) w=7 -> d[y]=7
pass 1      s=0   t=6 x=INF   y=7 z=INF
   pass 2: relax (t,x) w=5 -> d[x]=11
   pass 2: relax (t,z) w=-4 -> d[z]=2
   pass 2: relax (y,x) w=-3 -> d[x]=4
pass 2      s=0   t=6   x=4   y=7   z=2
   pass 3: relax (x,t) w=-2 -> d[t]=2
pass 3      s=0   t=2   x=4   y=7   z=2
   pass 4: relax (t,z) w=-4 -> d[z]=-2
pass 4      s=0   t=2   x=4   y=7  z=-2
returns true

vertex |  d  | pi | shortest path from s
-------+-----+----+----------------------
   s   |   0 |  - | s
   t   |   2 |  x | s -> y -> x -> t
   x   |   4 |  y | s -> y -> x
   y   |   7 |  s | s -> y
   z   |  -2 |  t | s -> y -> x -> t -> z

=== the same graph with w(z,s) = -8 ===
BELLMAN-FORD returns false
```

Setiap bilangan cocok dengan Tabel 13.10 dan kelima panel, termasuk $d[x]=11$ yang sesaat di dalam sapuan 2.

Buku: Subbab 13.8.7.

Pendeteksinya, dan Mengapa Ia Berhasil

TEOREMA 13.13. Sesudah $|V|-1$ sapuan, Bellman–Ford mengembalikan `false` **jika dan hanya jika** grafnya memuat siklus berbobot negatif yang terjangkau dari s.

Tanpa siklus ⇒ `true`. Sesudah $|V|-1$ sapuan $d=\delta$, sehingga menurut Persamaan (13.5) $d[v]\le d[u]+w(u,v)$ untuk setiap sisi. Tidak ada sisi yang menyala.

Ada siklus ⇒ `false`. Bila ia mengembalikan `true`, maka untuk setiap sisi dengan $d[u]$ berhingga

$$d[v]\le d[u]+w(u,v). \tag{13.11}$$

Jumlahkan Persamaan (13.11) mengelilingi siklus negatif c yang terjangkau: kedua jumlah d berjalan atas simpul yang sama dan saling menghapus, tersisa $0\le w(c)<0$. Mustahil. ∎

`false` **berarti "ada siklus negatif yang terjangkau dari** s**"**, bukan "grafnya punya siklus negatif di suatu tempat". Untuk menguji seluruh graf, tambahkan satu simpul dengan sisi berbobot nol ke setiap simpul lain lalu jalankan darinya (Soal Latihan 13.14). Untuk menemukan siklusnya, mundurlah $|V|$ langkah sepanjang π lalu kelilingi (Soal Latihan 13.15).

Waktu jalan.

$$T_{\mathrm{BF}} = \underbrace{\Theta(V)}_{\text{inisialisasi}} + \underbrace{(|V|-1)\,|E|}_{\text{sapuan}} + \underbrace{|E|}_{\text{pendeteksi}} = \Theta(VE). \tag{13.12}$$

Menurut Persamaan (13.12): graf jarang memberi $\Theta(V^2)$, graf padat memberi $\Theta(V^3)$, harga Floyd–Warshall untuk *semua* pasangan.

Penyempurnaan berbasis antrean yang dikenal sebagai SPFA sering jauh lebih cepat dalam praktik, tetapi kasus terburuknya tetap $\Theta(V\,E)$: jangan pernah menyebut batas yang lebih baik. Buku: Subbab 13.8.8 dan 13.8.9.

Lintasan Terpendek di dalam DAG

Mencari lintasan terpendek di dalam DAG jauh lebih mudah, sebab urutan untuk melakukan relaksasinya mudah ditemukan — pengurutan topologis! Ketika kita tiba di u, setiap sisi yang dapat memperbaiki $d[u]$ sudah direlaksasi.

```
DAG-SHORTEST-PATHS(G, w, s)
01  untuk tiap v ∈ V[G]
02      d[v] ← ∞
03  d[s] ← 0
04  urutkan simpul-simpul V[G] secara topologis
05  untuk tiap simpul u, diambil dalam urutan topologis, lakukan
06      untuk tiap simpul v ∈ Adj[u] lakukan
07          RELAX(u, v, w)
```

Gambar 13.19 `DAG-Shortest-Paths`: baris 01–03 menginisialisasi, baris 04 mengurutkan simpul secara topologis, dan baris 05–07 merelaksasi sisi keluar tiap simpul tepat sekali, dalam urutan itu. Tidak ada antrean dan tidak ada pengulangan.

Pada Gambar 13.19, baris 04 menggantikan antrean prioritas Dijkstra, dan $|V|-1$ sapuan pada Gambar 13.13 lenyap.

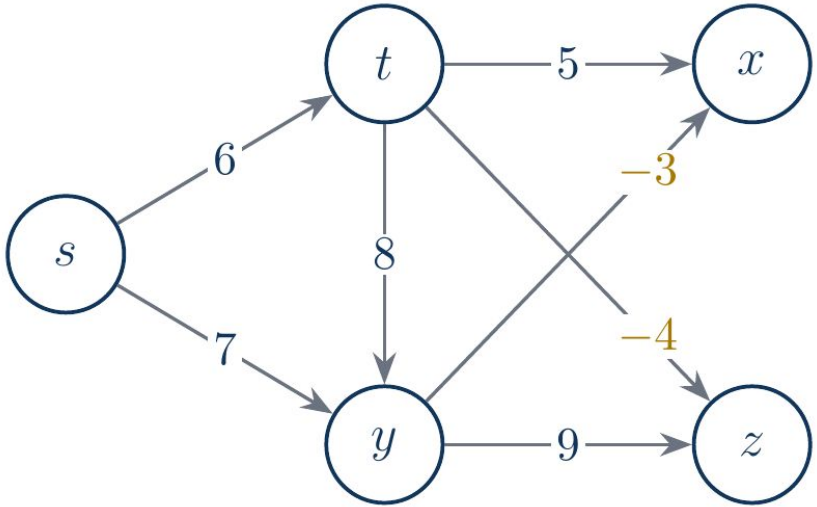
TEOREMA 13.14. Pada DAG algoritma ini berakhir dengan $d[v]=\delta(s,v)$ untuk setiap v. *Bukti:* dalam urutan topologis, simpul-simpul sembarang lintasan muncul sesuai urutan lintasannya, sehingga sisi-sisi lintasan terpendek

direlaksasi secara berurutan dan Lema 13.8 berlaku. **Tidak diperlukan pengandaian apa pun tentang tanda bobot.** ▰

Waktu jalannya $\Theta(V+E)$, linear dalam ukuran masukan: tanpa faktor lg *karena tidak ada antrean, tanpa faktor $|V|$ karena tidak ada pengulangan. Buku: Subbab 13.9.1 sampai 13.9.3 dan 13.9.6.*

Sebuah DAG Dikerjakan dengan Tangan

Ambillah graf Bellman–Ford pada Gambar 13.14 dan buang ketiga sisi *mundurnya*, (x,t), (z,x), dan (z,s). Pengurutan Kahn memberi urutan s,t,y,x,z.



Gambar 13.20 Graf contoh Bellman–Ford dengan ketiga sisi mundurnya dibuang: $s \to t\,(6)$, $s \to y\,(7)$, $t \to x\,(5)$, $t \to y\,(8)$, $t \to z\,(-4)$, $y \to x\,(-3)$, $y \to z\,(9)$. Grafnya sebuah DAG, dan ia masih memuat dua sisi negatif.

Tabel 13.12 `DAG-Shortest-Paths` pada contoh DAG, satu baris per simpul dalam urutan topologis.

Simpul yang diambil	$d[s]$	$d[t]$	$d[x]$	$d[y]$	$d[z]$	Sisi yang direlaksasi
(inisialisasi)	0	∞	∞	∞	∞	—
s	0	6	∞	7	∞	$(s,t)\to 6, (s,y)\to 7$
t	0	6	11	7	2	$(t,x)\to 11$, (t,y) tidak, $(t,z)\to 2$
y	0	6	4	7	2	$(y,x)\to 4$, (y,z) tidak
x	0	6	4	7	2	tidak ada
z	0	6	4	7	2	tidak ada

Tabel 13.12 pada Gambar 13.20: **tujuh** relaksasi, dibandingkan empat puluh bagi Bellman–Ford pada Tabel 13.10. Jawabannya berbeda ($\delta(s,t)=6$ di sini, 2 di sana) karena membuang (x,t) menghilangkan rute yang murah: dua graf, bukan dua jawaban.

Buku: Subbab 13.9.4.

Program DAG, dan Lintasan Terpanjang

Kode 13.3 `DAG-Shortest-Paths`: pengurutan topologis Kahn disusul satu sapuan relaksasi; uji `order.size() != n` adalah pendeteksi siklus yang cuma-cuma. Kompilasilah dengan `g++ -std=c++17 -O2 -o dag dag.cpp`.

```cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <queue>
#include <algorithm>
#include <climits>

using namespace std;

const long long INF = LLONG_MAX / 4;

struct Edge { int to; long long w; };

int main() {
    // Graf contoh Bellman-Ford dengan ketiga sisi mundurnya
    // (x,t), (z,x), dan (z,s) dihapus. Yang tersisa adalah sebuah DAG.
    vector<string> name = {"s", "t", "x", "y", "z"};
    int n = (int)name.size();
    auto id = [&](const string& s) {
        return (int)(find(name.begin(), name.end(), s) - name.begin());
    };
    vector<vector<Edge>> adj(n);
    auto E = [&](const char* a, const char* b, long long w) {
        adj[id(a)].push_back({id(b), w});
    };
    E("s","t", 6); E("s","y", 7);
    E("t","x", 5); E("t","y", 8); E("t","z",-4);
```

```
    E("y","x",-3); E("y","z", 9);

    // --- langkah 1: pengurutan topologis (algoritma derajat masuk Kahn, Bab 11) ---
    vector<int> indeg(n, 0);
    for (int u = 0; u < n; ++u) for (const Edge& e : adj[u]) ++indeg[e.to];
    queue<int> ready;
    for (int v = 0; v < n; ++v) if (indeg[v] == 0) ready.push(v);
    vector<int> order;
    while (!ready.empty()) {
        int u = ready.front(); ready.pop();
        order.push_back(u);
        for (const Edge& e : adj[u]) if (--indeg[e.to] == 0) ready.push(e.to);
    }
    if ((int)order.size() != n) { cout << "not a DAG\n"; return 1; }
```

Kode 13.3 (Lanjutan) *DAG-Shortest-Paths: pengurutan topologis Kahn disusul satu sapuan relaksasi; uji order.size() != n adalah pendeteksi siklus yang cuma-cuma. Kompilasilah dengan g++ -std=c++17 -O2 -o dag dag.cpp.*

```
    cout << "topological order:";
    for (int v : order) cout << " " << name[v];
    cout << "\n\n";

    // --- langkah 2: satu sapuan relaksasi, simpul dalam urutan topologis ---
    vector<long long> d(n, INF);
    vector<int> pi(n, -1);
    int s = id("s");
    d[s] = 0;

    for (int u : order) {
        cout << "take " << name[u] << " (d="
             << (d[u] >= INF ? string("INF") : to_string(d[u])) << ")\n";
        if (d[u] >= INF) { cout << "    unreachable; skip its edges\n"; continue; }
        for (const Edge& e : adj[u]) {
            long long cand = d[u] + e.w;
            cout << "    relax (" << name[u] << "," << name[e.to] << ") w=" << e.w
                 << ": " << d[u] << (e.w < 0 ? "+(" + to_string(e.w) + ")"
                                             : "+" + to_string(e.w))
                 << "=" << cand << " vs d["
                 << name[e.to] << "]="
                 << (d[e.to] >= INF ? string("INF") : to_string(d[e.to])) << " -> ";
            if (cand < d[e.to]) { d[e.to] = cand; pi[e.to] = u; cout << "improved\n"; }
            else                { cout << "no change\n"; }
        }
    }

    cout << "\nvertex |  d  | pi | shortest path from s\n";
```

```
    cout <<   "-------+-----+----+--------------------\n";
    for (int v = 0; v < n; ++v) {
        vector<string> path;
        for (int t = v; t != -1; t = pi[t]) path.push_back(name[t]);
        reverse(path.begin(), path.end());
        string p;
        for (size_t i = 0; i < path.size(); ++i) { if (i) p += " -> "; p += path[i]; }
        cout << "   " << name[v] << "   |" << setw(4) << d[v] << " |  "
             << (pi[v] == -1 ? string("-") : name[pi[v]]) << " | " << p << "\n";
    }
    return 0;
}
```

```
topological order: s t y x z

take s (d=0)
    relax (s,t) w=6: 0+6=6 vs d[t]=INF -> improved
    relax (s,y) w=7: 0+7=7 vs d[y]=INF -> improved
take t (d=6)
    relax (t,x) w=5: 6+5=11 vs d[x]=INF -> improved
    relax (t,y) w=8: 6+8=14 vs d[y]=7 -> no change
    relax (t,z) w=-4: 6+(-4)=2 vs d[z]=INF -> improved
take y (d=7)
    relax (y,x) w=-3: 7+(-3)=4 vs d[x]=11 -> improved
    relax (y,z) w=9: 7+9=16 vs d[z]=2 -> no change
take x (d=4)
take z (d=2)

vertex |  d  | pi | shortest path from s
-------+-----+----+----------------------
   s   |   0 |  - | s
   t   |   6 |  s | s -> t
   x   |   4 |  y | s -> y -> x
   y   |   7 |  s | s -> y
   z   |   2 |  t | s -> t -> z
```

Kode 13.3 menolak menjawab pada graf yang pengandaiannya gagal. **Lintasan terpanjang didapat cuma-cuma**: negasikan setiap bobot, jalankan gelung yang sama, lalu negasikan jawabannya. Itulah **metode lintasan kritis** dalam penjadwalan proyek.

Buku: Subbab 13.9.5 dan 13.9.7.

Memulihkan Rutenya dari π

Pseudokode 13.3 Mencetak lintasan terpendek dari s ke v secara rekursif lewat larik pendahulu.

```
PRINT-PATH(G, s, v)
1  bila v = s
2      cetak s
3  bila tidak, bila pi[v] = NIL
4      cetak "tidak ada lintasan dari s ke v"
5  bila tidak
6      PRINT-PATH(G, s, pi[v])
7      cetak v
```

Pseudokode 13.3 mencetak dalam urutan yang benar tanpa pembalikan eksplisit: baris 6 mencetak segala sesuatu sebelum v, lalu baris 7 mencetak v. Kedalaman rekursinya sepanjang lintasannya, hingga $|V|-1$, sehingga versi iteratif lebih aman pada graf besar.

Pada graf Dijkstra ia mencetak `s`, `s -> x -> u`, `s -> x -> u -> v`, `s -> x`, `s -> x -> y`: baris terbawah Tabel 13.7.

Tiga jebakan. Membaca π sebelum algoritmanya berhenti (di tengah jalan ia menggambarkan lintasan nyata berbobot 13); tidak pernah menetapkan π sama sekali; dan menyusuri π sesudah Bellman–Ford mengembalikan `false`, ketika π dapat memuat siklus, sehingga kawallah penyusurannya dengan hitungan langkah.

Buku: Subbab 13.10.

Algoritma Mana untuk Graf Mana

Tabel 13.13 Algoritma sumber tunggal mana yang dipakai. Kedua pertanyaan pertama adalah tentang kebenaran dan tidak menerima kompromi; hanya yang ketiga yang tentang kecepatan.

	Dijkstra	Bellman–Ford	Relaksasi DAG	BFS (Bab 10)
Graf haruslah	graf berarah apa pun	graf berarah apa pun	**asiklik**	graf apa pun
Bobot yang diizinkan	hanya $w \geq 0$	w real apa pun	w real apa pun	tidak ada (semua sama)
Siklus negatif	tak terdeteksi, jawaban salah	**terdeteksi**, mengembalikan `false`	mustahil	mustahil
Waktu, jarang ($E = \Theta(V)$)	$O(V \lg V)$	$\Theta(V^2)$	$\Theta(V)$	$\Theta(V)$
Waktu, padat ($E = \Theta(V^2)$)	$\Theta(V^2)$ dengan larik	$\Theta(V^3)$	$\Theta(V^2)$	$\Theta(V^2)$
Struktur tambahan	antrean prioritas	tidak ada	pengurutan topologis	antrean FIFO
Pemakaian khas	jaringan jalan, perutean OSPF	arbitrase mata uang, perutean RIP	penjadwalan proyek, *pipeline*	cacah lompatan, labirin tak berbobot

Prosedur di balik Tabel 13.13**.** 1. Semua bobot sama? **BFS.** 2. Asiklik? **Relaksasi DAG**, bobot negatif pun boleh. 3. Semua bobot ≥ 0? **Dijkstra**: *heap* bila jarang, larik bila padat. 4. Selain itu **Bellman–Ford**, *dan periksa nilai kembaliannya*. 5. Semua pasangan? **Bab 15.**

400.000 persimpangan, 1.000.000 ruas jalan, menit tempuh: Dijkstra dengan *heap*. *5.000 tugas build dengan durasi*: relaksasi DAG. *40 mata uang dengan* $-\ln(\text{kurs})$: Bellman–Ford, dijalankan justru untuk melihat apakah ia mengembalikan `false`.

Buku: Subbab 13.11.

Profitina dalam Praktik

Di dalam platform GEO. Perbatasan perayapan sebuah pemindaian mengambil halaman berikutnya yang termurah menurut ongkos yang terakumulasi sepanjang lintasan tautan: antrean prioritas Dijkstra, dengan bobot tak negatif. *Pipeline* pemindaiannya adalah DAG, dan durasi minimumnya adalah sebuah lintasan terpanjang.

Profitina Laundry, Pertanyaan 1. Tujuh perhentian dan sebelas ruas satu arah berbobot menit berkendara: Dijkstra. Depot → Dharmahusada secara langsung memakan 11 menit; lewat Kertajaya $7+3=10$.

Pertanyaan 2. Mitra di ITS memberi rabat, sehingga ruas (ITS, Keputih) berbiaya -15 ribu rupiah: Bellman–Ford. Keputih berongkos 19 untuk dicapai, sedangkan ITS, *yang terletak di jalan menuju ke sana*, berongkos 34: biaya yang turun ketika Anda melangkah lebih jauh.

Pertanyaan 3. Naikkan rabatnya menjadi -80 dan putaran Depot → Kertajaya → Manyar → ITS → Keputih → Depot berongkos $14+8+12-80+30=-16$: pendeteksinya menyala.

Algoritma memberi saran; manusia yang memutuskan. Buku: Subbab 13.12.

Kesalahan yang Sering Terjadi

1. Dijkstra pada graf bersisi negatif: programnya tidak mengeluh (Gambar 13.5). **2. Menambahkan konstanta pada setiap bobot**: lintasan k sisi bertambah $k\,c$. **3. Mengacaukan sisi negatif dengan siklus negatif** (Gambar 13.2).

4. Menjalankan Bellman–Ford $|V|$ atau $|E|$ sapuan; Teorema 13.4 menetapkannya $|V|-1$. **5. Melewatkan sapuan pendeteksi**, yang berongkos hanya satu bagian dari $|V|$ bagian totalnya.

6. Merelaksasi dari simpul yang belum terjangkau: `INF + (-4) < INF`. **7. Luapan karena penanda yang terlalu besar**: `INT_MAX + w` melingkar menjadi negatif. Pakailah `LLONG_MAX/4` dan kawal lebih dahulu.

8. Memperbarui d tanpa π. 9. Membaca π sebelum berhenti. 10. Memalsukan `DecreaseKey` secara keliru: dorong salinan kedua *dan buang salinan yang basi*.

11. Mengira pohon lintasan terpendek adalah MST. 12. Algoritma DAG pada graf bersiklus (Kode 13.3 menolak). **13. Menyebut $O(E)$ untuk SPFA. 14. Memperlakukan $\delta(s,v)=\infty$ sebagai galat**: artinya *tak terjangkau*.

Buku: Subbab 13.13.

Ringkasan — Lima Hal

1. **Persoalannya.** $\delta(s, v)$ adalah minimum $w(p)$ atas semua lintasan. Dari keempat ragam kita menyelesaikan **sumber tunggal**, karena yang lain tereduksi kepadanya atau dibangun darinya.
2. **Dua fakta struktural.** *Substruktur optimal*, dengan menggunting dan menempel; *ketaksamaan segitiga* $\delta(s, v) \le \delta(s, u) + w(u, v)$ untuk setiap sisi.
3. **Sisi negatif tidak berbahaya; siklus negatif mematikan.** Tanpa siklus negatif yang terjangkau, lintasan terpendek dapat dipilih sederhana, jadi paling banyak $|V| - 1$ sisi.
4. **Relaksasi adalah satu-satunya primitif**, dibuat aman oleh lima sifat: batas atas, tanpa-lintasan, kekonvergenan, relaksasi lintasan, subgraf pendahulu.
5. **Tiga algoritma.** *Dijkstra*, $w \ge 0$, $O(E \lg V)$ dengan *heap*. *Bellman–Ford*, bobot apa pun, $\Theta(V E)$, dengan pendeteksi. *Relaksasi DAG*, $\Theta(V + E)$. Rutenya dibaca mundur dari π sesudah algoritmanya selesai.

Buku: Subbab 13.14.

Pertemuan Berikutnya

Bab 14 — Analisis Amortisasi. Kita menepi sejenak dari graf untuk melunasi utang: "$\Theta(1)$ teramortisasi" pada baris *Fibonacci heap* di Tabel 13.8 sudah kita pakai tanpa bukti.

Tiga metode, *agregat*, *akuntansi*, dan *potensial*, diterapkan pada tumpukan dengan `Multipop`, pencacah biner, dan tabel dinamis; dan hal yang menjebak semua orang: **amortisasi bukan kasus rata-rata.**

Bab 15 — Lintasan Terpendek Semua Pasangan menjawab baris keempat Tabel 13.2: Floyd–Warshall, perkalian matriks dengan min dan $+$, dan pembobotan ulang Johnson, yang membuat Dijkstra sah pada graf bersisi negatif.

Pekerjaan rumah. Soal Latihan 13.1–13.16; mulailah dengan 13.7 (Dijkstra dengan tangan dari sumber x) dan 13.13 (melabeli setiap simpul $+\infty$, berhingga, atau $-\infty$).

Cormen dkk., Introduction to Algorithms, edisi ke-4, Bab 22, membahas bahan yang sama dengan konvensi gambar yang sama. Buku: Subbab 13.14.1 dan 13.15.

Kuliah 14 — Analisis Amortisasi

Perancangan dan Analisis Algoritma (EF234405)

Bab 14 buku ini dalam bentuk ringkas, untuk dipakai di kelas.

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*.

Pengajar: Irfan Subakti — yifana@gmail.com

Hari ini: struktur yang operasinya kadang murah dan sesekali sangat mahal, dan batas atas seluruh barisan yang jauh lebih baik daripada menjumlahkan kasus-kasus terburuk. Buku: Bab 14, paragraf pembuka.

Agenda

1. **Pertanyaannya.** Mengapa “operasi tunggal terburuk dikali n” dapat meleset sebesar faktor n, dan apa arti *biaya amortisasi*.
2. **Bukan kasus rata-rata.** Tidak ada peluang sama sekali; batasnya berlaku untuk setiap barisan, termasuk barisan pilihan lawan.
3. **Tiga metode** — agregat, akuntansi, potensial — masing-masing diterapkan pada tumpukan dengan `Multipop` dan pencacah biner, ditambah Soal 17.1-3, 17.2-2, dan 17.3-6 yang dikerjakan tuntas.
4. **Tabel dinamis.** Pelipatan saat penyisipan, dan mengapa menyeparuhkan pada separuh penuh menimbulkan *thrashing* sedangkan menyeparuhkan pada seperempat penuh tidak.
5. **Union–find**, janji dari Bab 11 ditepati: apa itu $\alpha(n)$ dan mengapa nilainya tidak pernah melebihi 4. Lalu memilih metode, Profitina, kesalahan, ringkasan.

Halaman-halamannya mengikuti urutan buku; soal latihan di akhir bab adalah pekerjaan rumahnya. Buku: Subbab 14.2 sampai 14.10.

Dari Mana Kita Melanjutkan

Sebuah utang dari Bab 11. Hutan himpunan saling lepas dengan gabung menurut peringkat dan pemampatan lintasan disebut berbiaya $O(m\,\alpha(n))$ untuk m operasi, dengan catatan bahwa yang dapat dibatasi hanyalah *rata-rata atas seluruh barisan*. Subbab 14.8 melunasi utang itu.

Surat utang kedua. `DecreaseKey` *Fibonacci heap* tercantum "$O(1)$ teramortisasi" pada tabel antrean prioritas di Bab 11 dan 13. Di akhir hari ini Anda akan tahu persis apa yang dinyatakan baris itu.

Yang Anda perlukan. Tumpukan, larik, O, Θ, Ω dari Bab 3, dan lambang penjumlahan. Selebihnya dibangun hari ini.

> **Perubahan kebiasaan.** Sepanjang hari ini, n adalah **banyaknya operasi dalam barisan**, bukan banyaknya elemen. Bila kita memerlukan banyaknya elemen, kita menyebutnya s (item di tumpukan) atau num (item di tabel).

Buku: Subbab 14.1.

Sebuah Batas yang Benar, dan Tidak Berguna

Tumpukan dengan tiga operasi. `Push(S,x)` dan `Pop(S)` masing-masing berbiaya 1. `Multipop(S,k)` mengambil k item teratas, atau semuanya bila kurang dari k; dengan s item biayanya $\min(s,k)$.

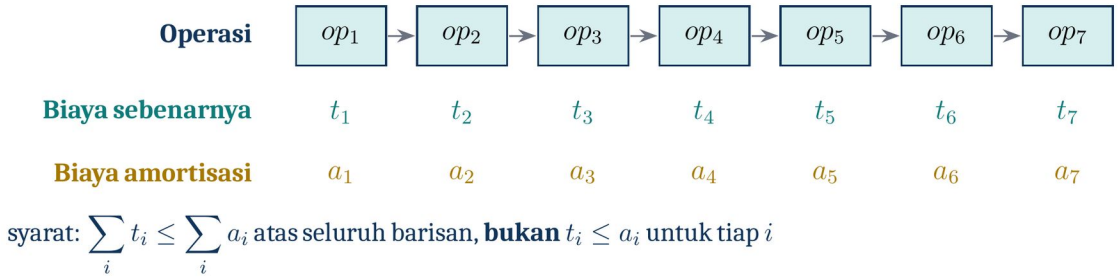
Analisis pertama yang dilakukan semua orang. Tumpukan tidak pernah memuat lebih dari n item, sehingga satu operasi paling mahal n; n operasi paling mahal $n \times n = n^2$. **Setiap katanya benar**, padahal jawaban sebenarnya $O(n)$.

Barisan pertama. Push $n-1$ item, lalu `Multipop(S,n)`: $(n-1)+(n-1)=2n-2$. **Barisan kedua.** Push $n/2$ item, lalu $n/2$ kali `Multipop(S,n)`: yang pertama mengosongkan tumpukan, sisanya berbiaya 0; totalnya n.

> **Di mana batas kasar itu keliru.** Ia memperlakukan setiap operasi seolah-olah semuanya dapat menjadi yang terburuk *pada saat yang sama*. `Multipop` yang mengambil sejuta item memerlukan sejuta `Push` sebelumnya dari anggaran yang sama: **operasi yang mahal dibayar oleh operasi murah yang mendahuluinya.**

Untuk $n = 1.000.000$ batas kasar itu menyebut satu triliun langkah; kenyataannya kurang dari dua juta. Buku: Subbab 14.2.1 dan 14.2.2.

Tiga Biaya, Satu Pertidaksamaan



Gambar 14.1 Operasi $o\,p_1, \ldots, o\,p_7$ beserta **biaya sebenarnya** t_i dan **biaya amortisasi** a_i yang kita tetapkan untuknya. Syaratnya adalah $\sum t_i \le \sum a_i$ atas seluruh barisan, dan sama sekali **bukan** $t_i \le a_i$ untuk tiap i.

Pada Gambar 14.1 baris tengah adalah **biaya sebenarnya** (ditulis t_i di sana, c_i mulai sekarang); baris bawah adalah **biaya amortisasi** a_i, bilangan yang *kita karang*, dengan satu aturan:

$$\sum_{i=1}^{n} t_i \le \sum_{i=1}^{n} a_i \text{ atas seluruh barisan, mulai dari } o\,p_1. \tag{14.1}$$

Tidak disyaratkan: $t_i \le a_i$ untuk tiap i. Satu operasi boleh jauh melampaui jatahnya; hanya total berjalan pada Persamaan (14.1) yang harus berlaku. Sebagian operasi menabung; operasi berikutnya membelanjakan tabungan itu.

"Amortisasi" berasal dari dunia keuangan: menyebar satu pembayaran besar menjadi banyak cicilan kecil. Buku: Subbab 14.2.3.

Definisinya

DEFINISI. Misalkan c_i **biaya sebenarnya** operasi ke-i dari n operasi. Bilangan $a_1, \ldots, a_n$ adalah **biaya amortisasi** yang sah bila

$$\sum_{i=1}^{k} c_i \le \sum_{i=1}^{k} a_i \text{ untuk setiap } k = 1, 2, \ldots, n. \tag{14.2}$$

Bila setiap $a_i \le \hat{c}$, operasinya berbiaya **amortisasi** $\hat{c}$, dan sembarang n operasi berbiaya paling banyak $n\hat{c}$.

Persamaan (14.2) meminta pertidaksamaan itu pada **setiap awalan** k, bukan hanya pada $k = n$. Itulah yang dihasilkan metode akuntansi dan potensial, dan artinya program boleh dihentikan kapan saja tanpa merusak batasnya.

PERTANYAAN BAB INI. *Diberikan barisan n operasi, berapa biaya total kasus terburuk seluruh barisan itu?* Total dibagi n adalah **biaya amortisasi per operasi**.

Buku: Subbab 14.2.3.

Amortisasi Bukan Kasus Rata-Rata

Analisis kasus rata-rata mengandaikan sebaran peluang atas masukan; jawabannya sebuah *ekspektasi*. $O(n \lg n)$ milik quicksort teracak adalah ekspektasi, dan sebagian jalannya memakan $\Theta(n^2)$.

Analisis amortisasi tidak mengandaikan **apa pun**: tanpa sebaran, tanpa keacakan, tanpa masukan khas. Ia adalah **batas kasus terburuk atas total**, berlaku untuk setiap barisan, termasuk yang dipilih lawan yang sudah membaca kode sumber Anda.

> **Salah kaprah yang harus dimatikan.** "$O(1)$ teramortisasi" *tidak* berarti "$O(1)$ hampir setiap saat". Artinya **tidak ada barisan n operasi yang totalnya melebihi $O(n)$.** Yang tidak dijaminnya adalah bahwa suatu operasi *tertentu* cepat.

Cara mengujinya. *Dapatkah lawan yang mengetahui seluruh implementasi saya mengalahkan batas ini?* Untuk batas ekspektasi, sering kali dapat. Untuk batas amortisasi, tidak pernah.

Buku menyatakannya dua kali dengan sengaja; kali kedua terasa berbeda sesudah Anda melihat metode agregat bekerja. Buku: Subbab 14.2.4 dan 14.4.

Ketiga Metodenya

Tiga cara menemukan biaya amortisasi

1. **Metode agregat.** Batasi biaya total $T(n)$ dari sembarang n operasi; tiap operasi ditagih $T(n)/n$.
2. **Metode akuntansi.** Beri operasi tagihan yang berbeda-beda; tagih sebagian lebih mahal agar membayar di muka, sebagai kredit tersimpan, untuk yang lain.
3. **Metode potensial.** Simpan pembayaran di muka itu sebagai "energi potensial" Φ struktur datanya sendiri: biaya amortisasi = biaya sebenarnya + perubahan potensial.

Gambar 14.2 Ketiga teknik yang umum: **metode agregat** membagi biaya total $T(n)$ dengan n; **metode akuntansi** menagih sebagian operasi terlalu mahal supaya membayar di muka untuk yang lain; **metode potensial** menyimpan pembayaran di muka itu sebagai "energi potensial" strukturnya sendiri.

Agregat: hitung total $T(n)$ dengan argumen pencacahan dan beri setiap operasi $T(n)/n$. **Akuntansi:** tagih tiap jenis operasi jumlah tetap dan simpan kelebihannya sebagai **kredit** pada objek tertentu. **Potensial:** simpan pembayaran di muka sebagai satu bilangan Φ yang dihitung dari seluruh struktur.

Ketiganya, seperti tersirat pada Gambar 14.2, memberi jawaban yang sama pada contoh-contoh hari ini; bedanya ada pada banyaknya kerja dan banyaknya yang diceritakan.

Buku: Subbab 14.2.5.

Metode Agregat: Tumpukan

Dua langkah. (1) Tunjukkan bahwa sembarang n operasi memakan $T(n)$ pada kasus terburuk. (2) Tetapkan biaya amortisasi tiap operasi sebesar $T(n)/n$. Satu-satunya kelemahannya: semua operasi mendapat bilangan yang sama.

masing-masing $O(1)$

```
PUSH(S, x)
1  top[S] ← top[S] + 1
2  S[top[S]] ← x
```

```
POP(S)
1  jika STACK-EMPTY(S)
2       maka galat "underflow"
3       selain itu top[S] ← top[S] − 1
4       kembalikan S[top[S] + 1]
```

biaya = banyaknya putaran gelung $= \min(s, k)$

```
MULTIPOP(S, k)
1  selama tidak STACK-EMPTY(S) dan k ≠ 0
2       lakukan POP(S)
3            k ← k − 1
```

Gambar 14.3 Operasi-operasi tumpukan: `Push` dan `Pop` masing-masing menyentuh satu slot larik dan satu indeks, sehingga keduanya $O(1)$. `Multipop(S,k)` memanggil `Pop` sampai tumpukannya kosong atau k pop sudah dilakukan, sehingga biayanya adalah banyaknya putaran gelungnya.

Pada Gambar 14.3, `Multipop` berputar selama tumpukan tidak kosong *dan* $k \neq 0$, sehingga dengan s item ia berputar $\min(s, k)$ kali: biaya $O(\min(s, k))$, **bukan** $O(1)$.

Buku: Subbab 14.3.1 dan 14.3.2.

Di-pop Paling Banyak Sekali

Klaim. Mulai dari tumpukan kosong, banyaknya pengambilan tidak pernah melebihi banyaknya push. **Bukti.** Sebuah item masuk hanya lewat satu `Push`; ia keluar hanya lewat `Pop`, dan satu putaran `Multipop` *adalah* sebuah `Pop`; sekali diambil ia hilang. Jadi tiap item di-pop **paling banyak sekali** sesudah di-push. ∎

Totalnya. Dengan p push dan q pengambilan, $q \le p \le n$, sehingga

$$T(n) = p \times 1 + q \times 1 \le n + n = 2n, \quad \frac{T(n)}{n} \le 2 = O(1).$$

> **“Tiap elemen di-pop paling banyak sekali sesudah di-push.”** Kalimat itulah seluruh analisisnya. Bentuk umumnya: *temukan satuan kerja yang dibebankan pada sebuah objek, lalu tunjukkan bahwa tiap objek dibebani sebanyak terbatas kali atas seluruh barisan.*

Ketiga operasi berbiaya amortisasi $O(1)$, termasuk `Multipop`, yang biaya sebenarnya bisa $\Theta(n)$. Itu bukan kontradiksi; itulah definisi yang sedang bekerja.

Pembukuan di dalam `Multipop` mengubah $2n$ menjadi cn untuk suatu konstanta kecil c; angka bersihnya 2. Buku: Subbab 14.3.2.

Tumpukannya, dalam C++

Kode 14.1 mencacah setiap item yang meninggalkan tumpukan pada tiga barisan: barisan lawan, barisan yang murah, dan campuran acak-semu.

Kode 14.1 Sebuah tumpukan dengan `Multipop`, diinstrumentasi. Kompilasi dengan `g++ -std=c++17 -O2 -o multipop multipop.cpp`.

```cpp
// Sebuah tumpukan dengan PUSH, POP dan MULTIPOP, diinstrumentasi sehingga
// setiap slot larik yang tersentuh ikut dihitung.  Program ini lalu melaporkan
// biaya amortisasi terukur -- total kerja dibagi banyaknya
// operasi -- untuk tiga barisan yang berbeda.
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
using namespace std;

class CountingStack {
public:
    long long work = 0;        // tulis/baca slot dasar, yaitu biaya sebenarnya
    long long nPush = 0, nPop = 0, nMultipop = 0, nOps = 0;

    void push(int x) { data.push_back(x); ++work; ++nPush; ++nOps; }

    // Satu POP sebagai operasi tersendiri.
    void pop() { ++nPop; ++nOps; popOne(); }

    // MULTIPOP(S,k): buang min(s,k) item teratas.  Dihitung sebagai SATU
    // operasi, tetapi biaya sebenarnya adalah banyaknya item yang dibuangnya.
    int multipop(int k) {
        ++nMultipop; ++nOps;
        int removed = 0;
        while (!data.empty() && k != 0) { popOne(); --k; ++removed; }
        return removed;
    }
```

```
    size_t size() const { return data.size(); }
    bool empty() const { return data.empty(); }

private:
    vector<int> data;
    // Satu pembuangan dasar.  MULTIPOP dibangun dari ini, sehingga
    // `work` menghitung tepat satu satuan per item yang meninggalkan tumpukan.
    void popOne() { if (!data.empty()) { data.pop_back(); ++work; } }
};
```

Kode 14.1 (Lanjutan) *Sebuah tumpukan dengan Multipop, diinstrumentasi. Kompilasi dengan g++ -std=c++17 -O2 -o multipop multipop.cpp.*

```
void report(const string& name, const CountingStack& S) {
    cout << left << setw(30) << name
         << " ops=" << setw(8) << S.nOps
         << " (push " << setw(7) << S.nPush
         << " pop " << setw(7) << S.nPop
         << " multipop " << setw(7) << S.nMultipop << ")"
         << "  actual cost=" << setw(9) << S.work
         << "  cost/op=" << fixed << setprecision(6)
         << (double)S.work / (double)S.nOps << "\n";
}

int main() {
    const int n = 100000;

    // Barisan A: kesukaan si lawan.  Dorong n/2 item, lalu minta
    // MULTIPOP mengosongkan seluruh tumpukan dalam satu operasi.  Ulangi.
    {
        CountingStack S;
        while (S.nOps + 1 < n) {
            for (int i = 0; i < n / 2 && S.nOps + 1 < n; ++i) S.push(i);
            S.multipop(n);                          // satu operasi yang sangat mahal
        }
        report("A: fill, then MULTIPOP(n)", S);
    }

    // Barisan B: satu push, satu multipop, terus-menerus -- tiap MULTIPOP murah.
    {
        CountingStack S;
```

```
        while (S.nOps + 2 <= n) { S.push(1); S.multipop(n); }
        report("B: push/MULTIPOP alternating", S);
    }

    // Barisan C: campuran pseudo-acak, deterministik agar dapat diulang.
    {
        CountingStack S;
        unsigned long long seed = 12345;
        auto rnd = [&seed]() { seed = seed * 6364136223846793005ULL + 1442695040888963407ULL;
                               return (unsigned)(seed >> 33); };
        while (S.nOps < n) {
            unsigned r = rnd() % 100;
            if (r < 60) S.push((int)r);
            else if (r < 90) { if (!S.empty()) S.pop(); else S.push(0); }
            else S.multipop((int)(rnd() % 1000));
        }
        report("C: random 60/30/10 mix", S);
    }
```

Kode 14.1 (Lanjutan) *Sebuah tumpukan dengan Multipop, diinstrumentasi. Kompilasi dengan g++ -std=c++17 -O2 -o multipop multipop.cpp.*

```
    cout << "\nThe bound proved in the text is: actual cost < 2n, i.e. cost/op < 2.\n";
    cout << "With n = " << n << " that ceiling is " << 2 * n << ".\n";
    return 0;
}
```

Buku: Subbab 14.3.2, “Mengukurnya”.

Tumpukannya, Diukur

```
A: fill, then MULTIPOP(n)      ops=100000   (push 99998   pop 0       multipop 2      )  actual
cost=199996     cost/op=1.999960
B: push/MULTIPOP alternating   ops=100000   (push 50000   pop 0       multipop 50000  )  actual
cost=100000     cost/op=1.000000
C: random 60/30/10 mix         ops=100000   (push 64514   pop 25192   multipop 10294  )  actual
cost=129027     cost/op=1.290270

The bound proved in the text is: actual cost < 2n, i.e. cost/op < 2.
With n = 100000 that ceiling is 200000.
```

Barisan A mengosongkan tumpukan dalam satu satu `Multipop` raksasa dan mencapai 199.996 terhadap langit-langit 200.000 : empat satuan di bawah batas yang dibuktikan, dan tidak lebih.

Barisan C, campuran biasa, berbiaya $1,29$ per operasi. Tidak ada kasus yang mencapai 2, dan tidak ada yang mendekati $n = 100.000$ per operasi yang diizinkan batas $O(n^2)$.

Buku: Subbab 14.3.2, “Mengukurnya”.

Pencacah Biner k-Bit

Larik $A[0..k-1]$ berisi bit, $A[0]$ paling tidak signifikan, menyatakan $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$. Untuk menambah satu: ubah 1 di ekor menjadi 0 lalu setel 0 yang pertama, persis seperti 1999+1 di atas kertas.

```
INCREMENT(A)
1  i ← 0
2  selama i < length[A] dan A[i] = 1
3      lakukan A[i] ← 0
4          i ← i + 1
5  jika i < length[A]
6      maka A[i] ← 1
```

$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	
0	0	0	0	0	1	1	1	$= 7$
0	0	0	0	1	0	0	0	$= 8$

larik bit $A[0..k-1]$, $length[A] = k$;
$A[0]$ adalah bit paling tidak signifikan

Gambar 14.4 `Increment(A)`: baris 2–4 berjalan naik dari $A[0]$ sambil mengubah tiap 1 di ekornya menjadi 0, dan baris 6 menyetel 0 pertama yang dicapainya. Bila setiap bitnya 1, pencacahnya diam-diam berputar kembali ke nol, persis seperti register mesin k-bit.

Penelusuran Gambar 14.4, $k=4$, dari 0111 (tujuh): nolkan $A[0]$, $A[1]$, $A[2]$; berhenti pada $A[3]=0$; setel $A[3]$. Hasilnya 1000 (delapan), empat bit ditulis. **Biaya satu `Increment` adalah banyaknya bit yang diubahnya.**

Buku: Subbab 14.3.3.

Tabel 8-Bit

Nilai pencacah	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Biaya total
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

Gambar 14.5 Pencacah 8-bit sesudah tiap satu dari 16 increment; sel yang diarsir pada sebuah baris adalah bit yang dibalik oleh increment berikutnya, sehingga banyaknya adalah biaya sebenarnya increment itu. Total berjalannya sesudah 16 increment adalah 31, di bawah $2 \times 16 = 32$.

Pada Gambar 14.5, $A[0]$ diarsir di *setiap* baris: ia berbalik n kali dalam n increment. Baris yang lebar ($7 \to 8$ mengarsir empat sel, $15 \to 16$ lima) jarang, dan makin jarang makin lebar.

Batas kasar menyebut $O(nk)$. Koreksinya satu pengamatan: *tidak semua bit berubah setiap kali* `Increment` *dipanggil.*

Buku: Subbab 14.3.3, "Tabel 8-bit".

Seberapa Sering Bit i Berbalik?

$A[0]$ berbalik pada setiap increment: n kali. $A[1]$ hanya bila $A[0]$ bernilai 1, setiap increment kedua: $\lfloor n/2 \rfloor$. $A[2]$ setiap increment keempat: $\lfloor n/4 \rfloor$.

Secara umum, untuk $i = 0, 1, \ldots, \lfloor \lg n \rfloor$, bit $A[i]$ berubah $\lfloor n/2^i \rfloor$ kali, sebab ia berbalik tepat ketika i bit terendah nilai yang dinaikkan semuanya 1, yaitu satu nilai dalam setiap blok 2^i. Bit yang lebih tinggi tidak pernah bergerak.

Totalnya atas n increment:

$$T(n) = \sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor . \tag{14.3}$$

Kita menulis $\lg$ *untuk* $\log_2$*, sebagaimana ditetapkan Bab 3. Buku: Subbab 14.3.3, "Seberapa sering bit* i *berbalik?".*

Menjumlahkannya

$$\sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor \;<\; n \sum_{i=0}^{\infty} \frac{1}{2^i} \;=\; 2n$$

Gambar 14.6 Analisis agregat pencacah dalam satu tampilan: buang lantainya, perpanjang jumlahnya sampai tak hingga, keluarkan n, lalu jumlahkan deret geometrinya menjadi 2.

Gambar 14.6 membatasi Persamaan (14.3): $\lfloor x \rfloor \le x$; perpanjangan sampai tak hingga menambah suku positif; keluarkan n; dan $1+1/2+1/4+\cdots=2$ (misalkan S jumlahnya; $1/2\,S=S-1$, jadi $S=2$). Maka $T(n)<2n$ dan biaya amortisasinya $<2=O(1)$.

> **Dalam bahasa sehari-hari.** Untuk membalik k bit Anda harus lebih dahulu menyetel k bit itu satu per satu selama 2^k-1 increment sebelumnya. Rata-rata kurang dari dua bit berbalik per increment, **berapa pun besarnya** k: pencacah 64-bit dan 8-bit sama biayanya.

Diukur atas sejuta increment, totalnya $1.999.993$*, tujuh di bawah* $2n$*. Buku: Subbab 14.3.3, “Menjumlahkannya” dan “Mengukurnya”.*

Soal 17.1-3

SOAL 17.1-3. Operasi ke-i dari n operasi berbiaya i bila i tepat sebuah pangkat 2, dan 1 bila tidak. Gunakan metode agregat untuk menentukan biaya amortisasi per operasi.

- Sebuah barisan n operasi dijalankan pada sebuah struktur data.
- Biaya operasi ke-i adalah i bila i tepat sebuah pangkat 2, dan 1 bila bukan.
- Gunakan **metode agregat** untuk menentukan biaya amortisasi per operasi.

Operasi i	**Biaya** c_i
1	$1 = 2^0$
2	$2 = 2^1$
3	1
4	$4 = 2^2$
5	1
Total	**9**

Gambar 14.7 Lima operasi pertama Soal 17.1-3 berbiaya $1, 2, 1, 4, 1$: pangkat dua $1, 2, 4$ menagih indeksnya sendiri dan operasi lain menagih 1. Totalnya 9, atau $1, 8$ per operasi.

Seperti diperlihatkan Gambar 14.7, fungsi biayanya adalah

$$c_i = \begin{cases} i & \text{bila } i = 2^j \text{ untuk suatu bilangan bulat } j \geq 0, \\ 1 & \text{bila tidak.} \end{cases} \tag{14.4}$$

Pecahlah Persamaan (14.4) menjadi dasar 1 per operasi dan tambahan $2^j - 1$ dari pangkat-pangkat dua, $j = 0, \ldots, \lfloor \lg n \rfloor$:

$$\text{Total} = \sum_{j=0}^{\lfloor \lg n \rfloor} \left(2^j - 1\right) + \sum_{i=1}^{n} 1. \tag{14.5}$$

Tabel pelipatan di Subbab 14.7 berperilaku hampir persis seperti karikatur ini. Buku: Subbab 14.3.4.

Soal 17.1-3, Diselesaikan

$$
\begin{aligned}
\textit{Total} &= \sum_{j=0}^{\lfloor \lg n \rfloor} (2^j - 1) \;+\; \sum_{i=1}^{n} 1 && \text{pangkat dua membayar } 2^j\text{, bukan 1} \\
&= \left(2^{\lfloor \lg n \rfloor + 1} - 1\right) - \left(\lfloor \lg n \rfloor + 1\right) + n && \text{deret geometri: } \sum_{j=0}^{m} x^j = \frac{x^{m+1} - 1}{x - 1} \\
&= 2^{\lfloor \lg n \rfloor + 1} + n - \lfloor \lg n \rfloor - 2 \\
&\le 2n + n - \lfloor \lg n \rfloor - 2 && \text{sebab } 2^{\lfloor \lg n \rfloor} \le 2^{\lg n} = n \\
&= 3n - \lfloor \lg n \rfloor - 2.
\end{aligned}
$$

Biaya total n operasi adalah $O(n)$, sehingga biaya amortisasi per operasi adalah $O(1)$.

Gambar 14.8 Penyelesaian lengkap Soal 17.1-3, dari jumlah yang dipecah sampai $\text{Total} \le 3n - \lfloor \lg n \rfloor - 2$, lengkap dengan rumus deret geometri dan kesamaan logaritma yang diperlukannya.

Deret geometri (kalikan dengan x lalu kurangkan):

$$\sum_{k=0}^{m} x^k = \frac{x^{m+1} - 1}{x - 1} \quad (x \neq 1). \tag{14.6}$$

Dengan $x = 2$, Persamaan (14.6) mengubah Persamaan (14.5) menjadi total yang eksak

$$\text{Total} = 2^{\lfloor \lg n \rfloor + 1} + n - \lfloor \lg n \rfloor - 2. \tag{14.7}$$

Pada Persamaan (14.7), $2^{\lfloor \lg n \rfloor + 1} \le 2^{\lg n + 1} = 2n$, sehingga $\text{Total} \le 3n - \lfloor \lg n \rfloor - 2 \le 3n - 2$, seperti kesimpulan Gambar 14.8: biaya amortisasi $< 3 = O(1)$.

Batasnya ketat tepat ketika n pangkat dua, dan program di Subbab 14.3.4 mengonfirmasi rumus eksaknya pada setiap titik periksa. Buku: Subbab 14.3.4.

Yang Tidak Dapat Dilakukan Metode Agregat

Satu bilangan untuk semua operasi. Pada tumpukan, `Push`, `Pop`, dan `Multipop` semuanya bernilai 2, padahal `Push` jelas berbiaya tepat 1.

Ia memerlukan argumen pencacahan global. Untuk tumpukan dan pencacah argumen itu ada; untuk *splay tree* atau *Fibonacci heap* tidak ada yang diketahui, dan metodenya gagal begitu saja.

Ia tidak mengatakan apa-apa tentang *kapan* biayanya muncul. Batas amortisasi menyangkut *throughput*, bukan latensi: tenggat waktu nyata memerlukan jaminan lain.

> **Mengapa 3 dan tidak lebih buruk, pada Soal 17.1-3.** Tambahannya $1, 2, 4, \ldots$ berlipat dua, sehingga jumlahnya kurang dari dua kali yang terbesar, di bawah $2n$; satu satuan untuk tiap operasi menjadikannya $3n$. Biaya yang berlipat dua memberi biaya amortisasi konstan.

Buku: Subbab 14.3.4 dan 14.3.5.

Metode Akuntansi

Aturannya. Setiap operasi diberi jatah $\hat{c}_i$ yang kita pilih. Ia membelanjakan biaya sebenarnya c_i; kelebihannya disimpan sebagai **kredit** pada objek tertentu; kekurangannya dibayar dari kredit yang disimpan sebelumnya.

INVARIAN KREDIT. Untuk setiap $k = 1, 2, \ldots, n$,

$$C_k = \sum_{i=1}^{k} \hat{c}_i - \sum_{i=1}^{k} c_i \geq 0. \tag{14.8}$$

Strukturnya tidak pernah berutang.

Bila disusun ulang, Persamaan (14.8) tidak lain adalah Persamaan (14.2). Ia harus berlaku pada **setiap** k, karena barisannya dapat berhenti di mana saja.

Jebakannya. "Tagihannya jelas cukup besar" tidak membuktikan apa pun. Anda harus **menunjuk letak** kreditnya: objek mana yang memegang tiap satuan, sehingga saldonya tampak sebagai sebuah cacah dan tidak mungkin negatif.

Bayangkan setiap langkah dasar berongkos satu rupiah. Buku: Subbab 14.5.1 dan 14.5.2.

Tumpukannya, dengan Akuntansi

Operasi	Biaya sebenarnya	Biaya amortisasi (tagihan)
`push`	1	2
`pop`	1	0
`multipop`	$\min(s, k)$	0

- Setiap tagihan adalah konstanta, sehingga biaya amortisasi masing-masing dari ketiga operasi itu $O(1)$.
- Kreditnya sama dengan banyaknya item di tumpukan, yang tidak pernah negatif; jadi n operasi berbiaya total $O(n)$.

Gambar 14.9 Tagihan akuntansi untuk tumpukan: `Push` (biaya sebenarnya 1) ditagih 2, sedangkan `Pop` (biaya 1) dan `Multipop` (biaya $\min(s, k)$) ditagih 0. Ketiga tagihannya konstanta, sehingga n operasi berbiaya $O(n)$.

Aturan kredit untuk Gambar 14.9: setiap item di tumpukan membawa tepat 1 rupiah. `Push` membelanjakan 1 dan meninggalkan 1 pada item barunya. `Pop` dan setiap putaran `Multipop` membelanjakan rupiah yang dibawa item yang diambil.

Mengapa kreditnya tidak pernah negatif. Total kredit sama dengan banyaknya item di tumpukan, dan sebuah cacah tidak mungkin negatif. ∎

> *Setiap item membayar, pada saat di-push, kedatangannya sekaligus kepergiannya kelak.* Tagihan terbesar 2, sehingga n operasi berbiaya paling banyak $2n$: jawaban yang sama dengan metode agregat.

Buku: Subbab 14.5.3.

Pencacahnya, dengan Akuntansi

Tagihannya. Menyetel sebuah bit menjadi 1: **2**. Menolkan sebuah bit: **0**.

Aturan kredit: setiap bit yang sedang bernilai 1 membawa tepat 1 rupiah. Satu penyetelan-ke-1 membelanjakan 1 dan menyimpan 1 pada bit yang disetelnya. Masing-masing dari t penolan dibayar oleh rupiah yang ada pada bit 1 itu.

Jadi setiap `Increment` ditagih paling banyak 2, berapa pun bit yang dibaliknya. Increment dari $2^{20}-1$ ke 2^{20} membalik dua puluh satu bit dan tetap ditagih 2.

Tidak pernah negatif: kreditnya sama dengan banyaknya bit 1. **Luapan:** semua k penolan dibayar oleh k rupiah pada k bit 1, sehingga kreditnya cukup turun ke nol. ▰

Diinstrumentasi, kolom kreditnya sama dengan cacah item dan cacah bit 1 di setiap langkah, dan nilai terendahnya atas 200.000 operasi tumpukan dan 1.000.000 increment adalah 0. Buku: Subbab 14.5.4.

Soal 17.2-2

Kerjakan ulang Soal 17.1-3 dengan metode akuntansi. Tagih **setiap** operasi 3. Operasi murah menyimpan 2; operasi berbiaya i menarik $i-3$.

Tagih setiap operasi 3. Kredit yang tersisa sesudah operasi ke-i, ditulis Total_i, adalah

$$\mathit{Total}_i = \sum_{j=1}^{i} 3 \;-\; \mathit{ActualCost}_i = 3i - \left(2^{\lfloor \lg i \rfloor + 1} + i - \lfloor \lg i \rfloor - 2\right).$$

Kasus 1: $i = 2^k$ dengan $k \geq 0$. Maka $2^{\lfloor \lg i \rfloor + 1} = 2i$ dan

$$\mathit{Total}_i = 3i - (2i + i - k - 2) = k + 2 \;\geq\; 2.$$

Kasus 2: $i = 2^k + j$ dengan $k \geq 0$ dan $0 \leq j < 2^k$. Maka $2^{\lfloor \lg i \rfloor + 1} = 2^{k+1} = 2(i - j)$ dan

$$\mathit{Total}_i = 3i - \left(2(i-j) + i - k - 2\right) = 2j + k + 2 \;\geq\; 2.$$

Pada kedua kasus kreditnya tidak negatif, sehingga biaya amortisasi total $3n$ menjadi batas atas biaya sebenarnya total: n operasi berbiaya $O(n)$, dan biaya amortisasi per operasi adalah $O(1)$.

Gambar 14.10 Penyelesaian Soal 17.2-2 dengan metode akuntansi: tiap operasi ditagih 3, dan kredit sesudah operasi i adalah $3\,i$ dikurangi biaya sebenarnya sejauh itu, dipecah menjadi dua kasus menurut letak i di antara pangkat-pangkat dua.

Gambar 14.10: kredit sesudah i operasi adalah $3i - \text{ActualCost}_i$, dan Persamaan (14.7) dengan i di tempat n memberi ActualCost_i. Jadi

$$\text{Total}_i = 3i - \left(2^{\lfloor \lg i \rfloor + 1} + i - \lfloor \lg i \rfloor - 2\right).$$

Buku: Subbab 14.5.5.

Kedua Kasusnya

Kasus 1: $i = 2^k$**.** Maka $2^{\lfloor \lg i \rfloor + 1} = 2i$ dan $\text{Total}_i = 3i - (3i - k - 2) = k + 2 \geq 2$. Inilah saat paling ketat, tepat sesudah operasi termahal sejauh ini.

Kasus 2: $i = 2^k + j$**,** $0 \leq j < 2^k$**.** Maka $2^{\lfloor \lg i \rfloor + 1} = 2(i - j)$ dan

$$\text{Total}_i = 3i - (3i - 2j - k - 2) = 2j + k + 2.$$

(14.9)

Kasus 1 adalah $j = 0$ pada Persamaan (14.9). Pada kedua kasus kreditnya sekurang-kurangnya 2, sehingga tagihan 3 itu sah: n operasi berbiaya paling banyak $3n = O(n)$, amortisasi $O(1)$ per operasi.

Metode agregat memberi $3n - \lfloor \lg n \rfloor - 2$*; metode akuntansi memberi* $3n$ *yang sedikit lebih lemah dengan argumen per operasi yang jauh lebih pendek. Buku: Subbab 14.5.5, kedua kasus dan kesimpulannya.*

Soal 17.2-2, Diuji

Kode 14.2 Soal 17.2-2: saldo kreditnya dilacak dan diuji terhadap $2j+k+2$. Kompilasi dengan `g++ -std=c++17 -O2 -o ex1722 ex1722.cpp`.

```cpp
// Soal 17.2-2: kerjakan ulang Soal 17.1-3 dengan metode akuntansi.
// Tiap operasi ditagih 3.  Program ini melacak saldo banknya sesudah
// tiap operasi dan mencocokkannya dengan bentuk tertutup dalam teks,
//     bila i = 2^k + j  dengan 0 <= j < 2^k, maka credit_i = 2j + k + 2.
#include <iostream>
#include <iomanip>
using namespace std;

bool isPowerOfTwo(long long i) { return i > 0 && (i & (i - 1)) == 0; }
long long actualCost(long long i) { return isPowerOfTwo(i) ? i : 1; }

int floorLg(long long n) { int k = 0; while ((1LL << (k + 1)) <= n) ++k; return k; }

int main() {
    const long long N = 1000000;
    long long credit = 0, minCredit = 0, actualTotal = 0;
    bool formulaHolds = true;
    long long firstBad = 0;

    cout << "  i  actual  charged  credit   2j+k+2\n";
    for (long long i = 1; i <= N; ++i) {
        long long c = actualCost(i);
        actualTotal += c;
        credit += 3 - c;                        // tagih 3, bayarkan c
        int k = floorLg(i);
        long long j = i - (1LL << k);           // sehingga i = 2^k + j, 0 <= j < 2^k
        long long predicted = 2 * j + k + 2;
        if (credit != predicted && formulaHolds) { formulaHolds = false; firstBad = i; }
```

```
        if (credit < minCredit) minCredit = credit;
        if (i <= 17)
            cout << setw(3) << i << setw(8) << c << setw(9) << 3
                 << setw(8) << credit << setw(9) << predicted << "\n";
    }
    cout << "\nclosed form 2j+k+2 agrees for every i up to " << N << ": "
         << (formulaHolds ? "yes" : "no, first failure at i = ") ;
    if (!formulaHolds) cout << firstBad;
    cout << "\nlowest credit ever seen = " << minCredit
         << "  (the accounting method needs this to be >= 0)\n";
    cout << "total actual cost = " << actualTotal
         << ", total charged = " << 3 * N
         << ", so 3n is indeed an upper bound.\n";
    return 0;
}
```

```
  i  actual  charged  credit   2j+k+2
  1       1        3       2        2
  2       2        3       3        3
  3       1        3       5        5
  4       4        3       4        4
  5       1        3       6        6
  6       1        3       8        8
  7       1        3      10       10
  8       8        3       5        5
  9       1        3       7        7
 10       1        3       9        9
 11       1        3      11       11
 12       1        3      13       13
 13       1        3      15       15
 14       1        3      17       17
 15       1        3      19       19
 16      16        3       6        6
 17       1        3       8        8

closed form 2j+k+2 agrees for every i up to 1000000: yes
lowest credit ever seen = 0  (the accounting method needs this to be >= 0)
total actual cost = 2048555, total charged = 3000000, so 3n is indeed an upper bound.
```

Kode 14.2 mengonfirmasi Persamaan (14.9) untuk sejuta operasi. Saldonya naik 2 pada tiap operasi murah dan jatuh ke $4, 5, 6$ pada $i = 4, 8, 16$: tepat $k+2$.

Buku: Subbab 14.5.5, “Diperiksa dengan sebuah program”.

Metode Potensial

Metode akuntansi menyimpan kredit *pada objek*. Metode potensial menyimpannya sebagai **satu bilangan untuk seluruh struktur**, cocok untuk tabungan yang merupakan sifat *bentuk* strukturnya.

PENYIAPANNYA. D_0 adalah keadaan awal dan D_i keadaan sesudah operasi i; c_i biaya sebenarnya operasi i; Φ adalah **fungsi potensial** pilihan kita, yang memetakan tiap keadaan ke sebuah bilangan real; dan **biaya amortisasinya** adalah

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}). \tag{14.10}$$

Bacalah Persamaan (14.10) **dengan kata-kata:** biaya sebenarnya ditambah seberapa besar operasi itu menaikkan potensial. Operasi yang membuat struktur lebih berbahaya ditagih atas bahaya itu; operasi yang menjinakkannya mendapat pengembalian.

$\Phi(D_i) - \Phi(D_{i-1})$ *adalah perubahan potensial, sering ditulis* $\Delta\Phi_i$. *Buku: Subbab 14.6.1.*

Jumlah Teleskopik

Bila biaya amortisasi dijumlahkan, suku-suku potensialnya saling meniadakan berpasangan (teleskopik):

$$\begin{aligned}\sum_{i=1}^{n} a_i &= \sum_{i=1}^{n} \big(c_i + \Phi(D_i) - \Phi(D_{i-1})\big) \\ &= \Big(\sum_{i=1}^{n} c_i\Big) + \big(\cancel{\Phi(D_1)} - \Phi(D_0)\big) + \big(\cancel{\Phi(D_2)} - \cancel{\Phi(D_1)}\big) + \cdots + \big(\Phi(D_n) - \cancel{\Phi(D_{n-1})}\big) \\ &= \Big(\sum_{i=1}^{n} c_i\Big) + \Phi(D_n) - \Phi(D_0).\end{aligned}$$

Jadi bila $\Phi(D_n) \geq \Phi(D_0)$, biaya amortisasi total $\sum a_i$ menjadi batas atas biaya sebenarnya total $\sum c_i$.

Gambar 14.11 Menjumlahkan biaya amortisasi atas $i = 1, \ldots, n$ membuat suku-suku potensialnya saling meniadakan berpasangan, menyisakan yang pertama dan yang terakhir. Perhitungan inilah yang membuat seluruh metodenya bekerja.

Pada Gambar 14.11, tuliskan suku potensialnya untuk $n = 4$: $(\Phi_1 - \Phi_0) + (\Phi_2 - \Phi_1) + (\Phi_3 - \Phi_2) + (\Phi_4 - \Phi_3)$. Semua yang di tengah saling meniadakan, seperti teleskop yang dilipat:

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0). \tag{14.11}$$

Buku: Subbab 14.6.2.

Satu Syaratnya

Dengan menyusun ulang Persamaan (14.11), total sebenarnya tidak melebihi total karangan tepat ketika

$$\Phi(D_n) \geq \Phi(D_0), \tag{14.12}$$

dan karena barisannya dapat berhenti di mana saja, kita mensyaratkan Persamaan (14.12) pada setiap i.

> **Invarian kredit yang menyamar.** Menurut teleskop, $\sum_{i \leq k} \hat{c}_i - \sum_{i \leq k} c_i = \Phi(D_k) - \Phi(D_0)$, sehingga Persamaan (14.8) dan "potensial tidak pernah turun di bawah nilai awalnya" adalah **pernyataan yang sama**.

Jalan pintas praktis. Atur $\Phi(D_0) = 0$ dan $\Phi(D) \geq 0$ untuk setiap D yang dapat dicapai; nyatakan kedua fakta itu secara eksplisit. **Mencari** Φ: ia harus mengukur *seberapa banyak masalah yang sedang ditimbun strukturnya*.

Buku: Subbab 14.6.3.

Tumpukannya, dengan Potensial

- Potensial: $\Phi(D_i)$ = banyaknya objek di tumpukan sesudah operasi ke-i.
- Tumpukannya mula-mula kosong, jadi $\Phi(D_0) = 0$; tumpukan tidak pernah memuat objek sebanyak bilangan negatif, jadi $\Phi(D_i) \geq 0 = \Phi(D_0)$.
- `Push` ke tumpukan berisi s objek: $a_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + (s+1) - s = 2$.
- `Multipop`(S, k) dengan $k' = \min(k, s)$: $a_i = k' + (s - k') - s = 0$; demikian pula `Pop` berbiaya $1 - 1 = 0$.

Gambar 14.12 Tumpukan dengan metode potensial: dengan Φ = banyaknya elemen di tumpukannya, `Push` berbiaya amortisasi $1+(s+1)-s=2$ dan `Multipop` berbiaya amortisasi $k'-k'=0$, dengan $k'=\min(k,s)$.

Φ = banyaknya item. $\Phi(D_0)=0$ dan $\Phi\geq 0$. Dengan s item sebelum operasi, seperti pada Gambar 14.12: `Push` $1+(s+1)-s=2$; `Pop` $1+(s-1)-s=0$; `Multipop` dengan $k'=\min(k,s)$: $k'+(s-k')-s=0$.

> Potensialnya mencacah *kerja masa depan yang ditimbun*. `Multipop` mengerjakan k' satuan sekarang tetapi menghapus k' satuan kerja timbunan, sehingga ia tidak ditagih apa pun: ia **membelanjakan tabungan**, bukan menciptakan kewajiban.

Metode ketiga, $2n$ yang sama: jawabannya adalah sifat strukturnya, bukan sifat metodenya. Buku: Subbab 14.6.4.

Pencacahnya, dengan Potensial

- Potensial: b_i = banyaknya bit 1 sesudah `Increment` ke-i.
- Bila `Increment` ke-i menolkan t_i bit, biaya sebenarnya paling besar $t_i + 1$, dan $b_i \leq b_{i-1} - t_i + 1$.
- Perubahan potensial: $\Phi(D_i) - \Phi(D_{i-1}) \leq (b_{i-1} - t_i + 1) - b_{i-1} = 1 - t_i$.
- Biaya amortisasi: $a_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \leq (t_i + 1) + (1 - t_i) = 2$.
- Jadi n increment yang dimulai dari nol berbiaya $O(n)$.

Gambar 14.13 Pencacah dengan metode potensial: dengan b_i = banyaknya bit 1 sesudah `Increment` ke-i dan t_i = banyaknya bit yang dinolkan increment itu, perubahan potensialnya paling besar $1-t_i$ dan biaya amortisasinya paling besar $(t_i+1)+(1-t_i)=2$.

$\Phi(D_i)=b_i$, banyaknya bit 1; $b_0=0$. Increment ke-i menolkan t_i bit: $c_i \leq t_i+1$ dan $b_i \leq b_{i-1}-t_i+1$, sehingga $\Delta\Phi_i \leq 1-t_i$ dan, seperti pada Gambar 14.13,

$$\hat{c}_i \leq (t_i+1)+(1-t_i)=2.$$

t_i-nya saling meniadakan. Berapa pun bit yang dinolkan, potensialnya turun persis sebanyak itu. Diukur pada 16 increment: $\sum c_i=31$, $\Phi(D_n)=1$, dan $31+1=32=\sum \hat{c}_i$.

Buku: Subbab 14.6.5.

Soal 17.3-6: Antrean dari Dua Tumpukan

Implementasikan antrean dengan dua tumpukan sehingga `Enqueue` dan `Dequeue` berbiaya amortisasi $O(1)$. Tumpukan masukan S_{IN} dan tumpukan keluaran S_{OUT}: `Enqueue` mem-push ke S_{IN}; `Dequeue` mem-pop dari S_{OUT}, lebih dahulu menguras seluruh S_{IN} ke dalamnya **bila S_{OUT} kosong**.

```
ENQUEUE(e)
1  PUSH(S_IN, e)
```

```
DEQUEUE()
1  jika STACK-EMPTY(S_OUT) maka
2      selama tidak STACK-EMPTY(S_IN) lakukan
3          PUSH(S_OUT, POP(S_IN))
4  jika STACK-EMPTY(S_OUT) maka
5      galat "underflow"
6  selain itu
7      kembalikan POP(S_OUT)
```

Gambar 14.14 `Enqueue` dan `Dequeue` untuk antrean dua-tumpukan: `Enqueue` adalah satu push, dan `Dequeue` mem-pop dari S_{OUT} sesudah lebih dahulu menguras seluruh S_{IN} ke dalamnya, **hanya bila S_{OUT} kosong**. Pengujian *underflow*-nya datang sesudah pemindahannya.

Dua perincian pada Gambar 14.14. Pindahkan hanya bila S_{OUT} kosong, atau urutannya terbalik berkali-kali. Uji *underflow sesudah* pemindahan: antrean kosong hanya bila kedua tumpukan kosong.

Membalik tumpukan mengubahnya menjadi antrean: item tertua, di dasar S_{IN}, berakhir di puncak S_{OUT}. Buku: Subbab 14.6.6.

Potensialnya, dan Ketiga Kasusnya

- Potensial: $\Phi(D_i) = 2 \times \textit{elements}_i$, dengan $\textit{elements}_i$ banyaknya item di tumpukan **masukan** S_{IN} sesudah operasi ke-i.
- `Enqueue`: $a_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 2 = 3$.
- `Dequeue` tanpa pemindahan (S_{OUT} tidak kosong): $a_i = 1 + 0 = 1$.
- `Dequeue` dengan pemindahan $e = \textit{elements}_{i-1}$ item: $a_i = (2e + 1) + 0 - 2e = 1$.
- Jadi `Enqueue` maupun `Dequeue` berbiaya amortisasi $O(1)$.

Gambar 14.15 Analisis potensial antrean dua-tumpukan. Dengan $\Phi(D_i) = 2 \times \text{elements}_i$, dengan elements_i mencacah item di tumpukan **masukan** saja, `Enqueue` berbiaya amortisasi 3 dan kedua jenis `Dequeue` berbiaya 1.

$\Phi = 2 \times |S_{IN}|$: sebuah pemindahan menyentuh setiap item S_{IN} dua kali, dan item di S_{OUT} sudah membayar. Dengan $e = |S_{IN}|$ sebelum operasi, Gambar 14.15 memberi: `Enqueue` $1+2=3$; `Dequeue` tanpa pemindahan $1+0=1$; `Dequeue` dengan pemindahan $(2e+1)+0-2e=1$.

> **Setiap item menempuh tepat satu perjalanan**: di-push ke S_{IN}, di-pop, di-push ke S_{OUT}, di-pop. `Enqueue` membayar di muka tiga langkah itu; `Dequeue` yang mengeluarkan item itu membayar langkah terakhirnya.

Buku: Subbab 14.6.6, "Fungsi potensialnya".

Antrean Dua-Tumpukan dalam C++

Kode 14.3 Soal 17.3-6: antrean dua-tumpukan, dengan fungsi potensialnya dihitung sesudah tiap operasi. Kompilasi dengan `g++ -std=c++17 -O2 -o queue2 queue2.cpp`.

```cpp
// Soal 17.3-6: antrean FIFO yang dibangun dari dua tumpukan LIFO, dengan
// fungsi potensial Phi(D) = 2 * (banyaknya item di tumpukan masukan)
// dihitung sesudah tiap operasi, sehingga biaya amortisasinya
//     a_i = c_i + Phi(D_i) - Phi(D_{i-1})
// dapat dicetak di samping biaya sebenarnya dan diuji terhadap klaim
// bahwa ia tidak pernah melampaui 3.
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
using namespace std;

class TwoStackQueue {
public:
    long long lastActual = 0, lastAmortized = 0;
    long long totalActual = 0, totalAmortized = 0;
    long long worstActual = 0, worstAmortized = 0;

    void enqueue(int x) {
        long long before = potential();
        sIn.push_back(x);                    // ENQUEUE(e, S_IN): push(S_IN, e)
        account(1, before);
    }

    // Mengembalikan nilai yang dikeluarkan; menyetel `ok` ke false bila antreannya kosong.
    int dequeue(bool& ok) {
        long long before = potential();
        long long c = 0;
```

```
        if (sOut.empty()) {                    // pindahkan semuanya ke seberang
            // Tiap elemen yang pindah berbiaya 2: satu POP dari S_IN, satu PUSH ke S_OUT.
            while (!sIn.empty()) { sOut.push_back(sIn.back()); sIn.pop_back(); c += 2; }
        }
        if (sOut.empty()) { ok = false; account(c, before); return 0; }
        int v = sOut.back(); sOut.pop_back(); ++c;
        ok = true;
        account(c, before);
        return v;
    }

    long long potential() const { return 2 * (long long)sIn.size(); }
    size_t size() const { return sIn.size() + sOut.size(); }
    size_t inSize() const { return sIn.size(); }
    size_t outSize() const { return sOut.size(); }
```

Kode 14.3 (Lanjutan) *Soal 17.3-6: antrean dua-tumpukan, dengan fungsi potensialnya dihitung sesudah tiap operasi. Kompilasi dengan g++ -std=c++17 -O2 -o queue2 queue2.cpp.*

```
private:
    vector<int> sIn, sOut;
    void account(long long c, long long before) {
        lastActual = c;
        lastAmortized = c + potential() - before;
        totalActual += lastActual;
        totalAmortized += lastAmortized;
        if (lastActual > worstActual) worstActual = lastActual;
        if (lastAmortized > worstAmortized) worstAmortized = lastAmortized;
    }
};

int main() {
    TwoStackQueue Q;
    cout << "op          returned  |S_IN|  |S_OUT|   Phi   actual   amortized\n";
    auto line = [&](const string& what, const string& ret) {
        cout << left << setw(12) << what << setw(10) << ret << right
             << setw(6) << Q.inSize() << setw(9) << Q.outSize()
             << setw(7) << Q.potential()
             << setw(8) << Q.lastActual << setw(12) << Q.lastAmortized << "\n";
    };
    for (int i = 1; i <= 5; ++i) { Q.enqueue(i); line("ENQUEUE " + to_string(i), "-"); }
    bool ok;
    for (int i = 0; i < 3; ++i) { int v = Q.dequeue(ok); line("DEQUEUE", to_string(v)); }
    Q.enqueue(6); line("ENQUEUE 6", "-");
    for (int i = 0; i < 3; ++i) { int v = Q.dequeue(ok); line("DEQUEUE", to_string(v)); }

    // Jalan panjang yang memusuhi: n/2 enqueue, lalu n/2 dequeue, berulang.
```

```
    TwoStackQueue R;
    const int n = 200000;
    long long ops = 0;
    while (ops < n) {
        for (int i = 0; i < 1000 && ops < n; ++i) { R.enqueue(i); ++ops; }
        for (int i = 0; i < 1000 && ops < n; ++i) { R.dequeue(ok); ++ops; }
    }
    cout << "\n" << n << " operations in blocks of 1000:\n";
    cout << "  worst single actual cost   = " << R.worstActual << "\n";
    cout << "  worst single amortized cost= " << R.worstAmortized << "\n";
    cout << "  total actual cost          = " << R.totalActual << "\n";
    cout << "  total amortized cost       = " << R.totalAmortized << "\n";
    cout << "  actual cost per operation  = " << fixed << setprecision(6)
         << (double)R.totalActual / ops << "\n";
    cout << "The single worst operation costs " << R.worstActual
         << ", yet the average is at most 3: that gap is what amortization buys.\n";
    return 0;
}
```

Kode 14.3 mengimplementasikan pseudokode antrean dua-tumpukan baris demi baris. Buku: Subbab 14.6.6, “Programnya”.

Apa yang Dicetak Antreannya

```
op          returned  |S_IN|  |S_OUT|   Phi   actual   amortized
ENQUEUE 1   -              1        0      2        1           3
ENQUEUE 2   -              2        0      4        1           3
ENQUEUE 3   -              3        0      6        1           3
ENQUEUE 4   -              4        0      8        1           3
ENQUEUE 5   -              5        0     10        1           3
DEQUEUE     1              0        4      0       11           1
DEQUEUE     2              0        3      0        1           1
DEQUEUE     3              0        2      0        1           1
ENQUEUE 6   -              1        2      2        1           3
DEQUEUE     4              1        1      2        1           1
DEQUEUE     5              1        0      2        1           1
DEQUEUE     6              0        0      0        3           1

200000 operations in blocks of 1000:
  worst single actual cost   = 2001
  worst single amortized cost= 3
  total actual cost          = 400000
  total amortized cost       = 400000
  actual cost per operation  = 2.000000
The single worst operation costs 2001, yet the average is at most 3: that gap is what amortization buys.
```

Kolom `returned` berbunyi `1,2,3,4,5,6`: masuk pertama, keluar pertama. Baris keenam adalah `Dequeue` yang mahal: lima item dipindahkan dengan biaya 2 masing-masing ditambah satu pop, $c_i = 11$, dan biaya amortisasinya 1 karena Φ turun dari 10 ke 0.

Dalam skala besar: satu operasi berbiaya 2001, dan rata-rata atas 200.000 operasi adalah 2.

Buku: Subbab 14.6.6, “Programnya”.

Tabel Dinamis: Persoalannya

Struktur yang Anda pakai setiap hari: `std::vector`, `list` Python, `ArrayList` Java. Memori datang dalam blok tetap, sehingga untuk tumbuh Anda mengalokasikan blok yang lebih besar, menyalin, lalu membebaskan blok lama.

Dua tujuan. *Kecepatan:* $O(1)$ teramortisasi per penyisipan dan penghapusan. *Ruang:* bagian yang tak terpakai tidak pernah melebihi suatu pecahan konstan dari alokasinya.

Kosakata. `TABLE-INSERT`, `TABLE-DELETE`; $num[T]$ item tersimpan, $size[T]$ slot teralokasi; **faktor muatan** $\alpha(T) = num[T]/size[T]$, dengan $\alpha = 1$ untuk tabel kosong berukuran 0. **Kebijakan pertama: bila** $\alpha(T) = 1$**, lipatduakan tabelnya.**

> **Benturan notasi.** Di sini α adalah faktor muatan antara 0 dan 1; di Subbab 14.8 α adalah fungsi Ackermann balikan. Keduanya tidak berkaitan.

Buku: Subbab 14.7.1.

TABLE-INSERT

```
TABLE-INSERT(T, x)
 1  jika size[T] = 0
 2      maka alokasikan table[T] dengan 1 slot
 3           size[T] ← 1
 4  jika num[T] = size[T]
 5      maka alokasikan new-table dengan 2 · size[T] slot
 6           sisipkan semua item table[T] ke new-table
 7           bebaskan table[T]
 8           table[T] ← new-table
 9           size[T] ← 2 · size[T]
10  sisipkan x ke table[T]
11  num[T] ← num[T] + 1
```

Gambar 14.16 `TABLE-INSERT(T,x)`: baris 1–3 menangani penyisipan pertama ke dalam tabel tanpa slot, baris 4–9 memperluas (alokasikan dua kali lipat, salin, bebaskan blok lama), dan baris 10–11 menyimpan x lalu mencacahnya.

Biaya Gambar 14.16, satu satuan per penyisipan dasar: $c_i = 1$ bila masih ada tempat; bila tabelnya penuh, semua item disalin dan satu ditulis, sehingga $c_i = i$.

Syarat perluasan. Tabel penuh pada ukuran $1, 2, 4, 8, \ldots$, sehingga penyisipan ke-i memperluas tepat ketika $i - 1$ pangkat 2: penyisipan ke-$2, 3, 5, 9, 17, \ldots$

Batas kasarnya lagi-lagi $O(n^2)$. Buku: Subbab 14.7.2.

Agregat: Jumlahnya

$$\begin{aligned}\sum_{i=1}^{n} c_i &\leq n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j \\ &< n + 2n \\ &= 3n\end{aligned}$$

Gambar 14.17 Analisis agregat `TABLE-INSERT`. Satu satuan untuk tiap penyisipan, ditambah biaya perluasannya $2^0 + 2^1 + \cdots + 2^{\lfloor \lg n \rfloor}$, yang kurang dari $2n$; totalnya di bawah $3n$.

Dasar n, ditambah 2^j untuk perluasan pada penyisipan ke-(2^j+1), $j \leq \lfloor \lg n \rfloor$:

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j. \tag{14.13}$$

Menurut Persamaan (14.6), jumlah pada Persamaan (14.13) adalah $2^{\lfloor \lg n \rfloor + 1} - 1 < 2n$, sehingga totalnya di bawah $3n$, sebagaimana dinyatakan Gambar 14.17.

Mengapa melipatduakan, bukan menambah. Pelipatan menjadikan penyalinannya deret *geometri*, di bawah dua kali ukuran akhir tabelnya. Tumbuh dengan 100 slot tetap menjadikannya deret *aritmetika*, $\Theta(n^2)$ totalnya. **Pelipatan bukan perincian implementasi; ia adalah seluruh gagasannya.**

Buku: Subbab 14.7.3.

Akuntansi: Tiga Satuan per Item

Tagih setiap penyisipan 3. Satu satuan menulis itemnya; dua disimpan padanya. Setiap item membayar (1) penyisipannya sendiri, (2) pemindahannya sendiri pada perluasan berikutnya, (3) pemindahan satu item *lebih tua* yang sudah pernah dipindahkan.

Mengapa butir (3). Ketika tabel penuh berukuran s dilipatduakan, $s/2$ itemnya *baru* sejak perluasan terakhir dan masih memegang tabungannya: $2 \times s/2 = s$ satuan, tepat cukup untuk menyalin semua s item.

Invariannya. Kredit yang tersimpan adalah $2\times$ (item yang disisipkan sejak perluasan terakhir), sebuah cacah, jadi tidak pernah negatif. Maka n penyisipan berbiaya paling banyak $3n$.

Buku: Subbab 14.7.4.

Potensial: $\Phi = 2num - size$

Potensialnya.

$$\Phi(T) = 2 \cdot num[T] - size[T]. \tag{14.14}$$

Tepat sesudah perluasan tabelnya separuh penuh dan Persamaan (14.14) bernilai 0; tiap penyisipan menaikkannya 2; saat penuh nilainya sama dengan $size$, tepat yang harus disalin perluasan berikutnya.

Syarat-syaratnya. Tabel kosong memberi $\Phi = 0$; tabelnya selalu sekurang-kurangnya separuh penuh, sehingga $2num \geq size$ dan $\Phi \geq 0$.

Potensial: $\Phi(T) = 2 \cdot num[T] - size[T]$.

Tanpa perluasan pada penyisipan ke-i ($size_i = size_{i-1}$, biaya sebenarnya 1):

$$a_i = 1 + (2\,num_i - size_i) - (2\,num_{i-1} - size_{i-1}) = 1 + 2 = 3.$$

Dengan perluasan pada penyisipan ke-i ($size_i/2 = size_{i-1} = num_i - 1$, biaya sebenarnya num_i):

$$\begin{aligned} a_i &= num_i + (2\,num_i - size_i) - (2\,num_{i-1} - size_{i-1}) \\ &= num_i + \big(2\,num_i - 2(num_i - 1)\big) - \big(2(num_i - 1) - (num_i - 1)\big) \\ &= num_i + 2 - (num_i - 1) = 3. \end{aligned}$$

Pada kedua kasus biaya amortisasi `TABLE-INSERT` adalah 3, yakni $O(1)$.

Gambar 14.18 Kedua kasus analisis potensial `TABLE-INSERT` menghasilkan tepat 3. Pada kasus tanpa perluasan, biaya sebenarnya 1 dan potensialnya naik 2; pada kasus dengan perluasan, biaya sebenarnya num_i dan potensialnya ambruk sebesar $num_i - 3$.

Gambar 14.18 merangkum kedua kasusnya; halaman berikut menuliskannya lengkap.

Potensialnya telah menabung tepat sebesar biaya bencana yang dapat dilihatnya datang. Buku: Subbab 14.7.5.

Kedua Kasus Menghasilkan Tepat 3

Tanpa perluasan: $size_i = size_{i-1}$, $num_{i-1} = num_i - 1$, $c_i = 1$:

$$\hat{c}_i = 1 + (2num_i - size_i) - (2(num_i - 1) - size_i) = 1 + 2 = 3.$$

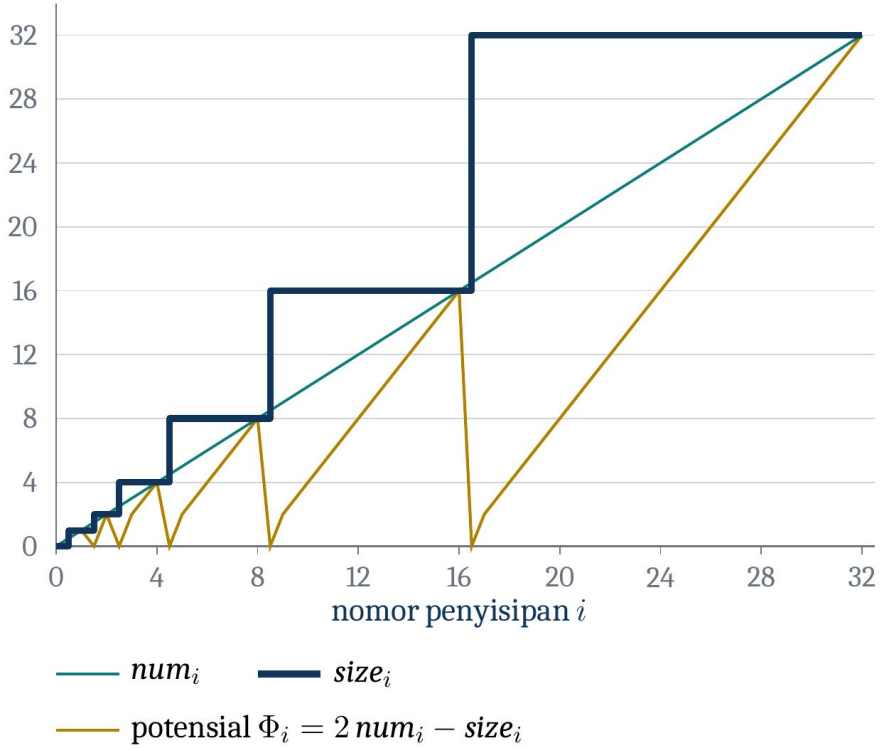
Dengan perluasan: $size_i/2 = size_{i-1} = num_i - 1$ dan $c_i = num_i$:

$$\hat{c}_i = num_i + (2num_i - 2(num_i - 1)) - (2(num_i - 1) - (num_i - 1)) = num_i + 2 - (num_i - 1) = 3.$$

Keduanya **kesamaan**, bukan batas yang masih longgar: biaya amortisasi `TABLE-INSERT` tepat 3.

Buku: Subbab 14.7.5.

Gambarnya Adalah Buktinya



Gambar 14.19 Sebuah barisan penyisipan: diagonal tipis adalah num, tangga tebal adalah $size$, dan gigi gergajinya adalah $\Phi = 2\,num - size$, yang jatuh ke 0 pada tiap perluasan lalu mendaki dengan kemiringan 2 sampai menemui $size$ lagi.

Telusurilah gigi gergaji pada Gambar 14.19: tepat sebelum perluasan $\Phi = size$; perluasan melipatduakan $size$ dan Φ jatuh ke 0; lalu ia mendaki lagi dengan kemiringan 2. **Potensial adalah rekening tabungan**, diisi terus-menerus dan dikosongkan seketika oleh perluasan.

Buku: Subbab 14.7.5, "Gambarnya".

Tabelnya dalam C++

Kode 14.4 `TABLE-INSERT` dengan pelipatan, diinstrumentasi dengan fungsi potensialnya. Kompilasi dengan `g++ -std=c++17 -O2 -o dyntable dyntable.cpp`.

```cpp
// TABLE-INSERT dengan pelipatan, diinstrumentasi tiga cara sekaligus:
//   * biaya sebenarnya c_i dari tiap penyisipan,
//   * total berjalannya, yang dibatasi metode agregat oleh 3n,
//   * potensial Phi(T) = 2*num - size dan biaya amortisasi
//     a_i = c_i + Phi(T_i) - Phi(T_{i-1}), yang dibuktikan teks bernilai 3.
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cstring>
using namespace std;

class DynamicTable {
public:
    ~DynamicTable() { free(slot); }

    // Mengembalikan biaya sebenarnya penyisipan ini: 1 bila masih ada ruang,
    // bila tidak num_i (satu tulis untuk x ditambah num_{i-1} salinan).
    long long insert(int x) {
        long long c = 0;
        if (size_ == 0) { slot = (int*)malloc(sizeof(int)); size_ = 1; }
        if (num_ == size_) {
            int* fresh = (int*)malloc(sizeof(int) * 2 * size_);
            memcpy(fresh, slot, sizeof(int) * size_);   // num_ == size_ salinan
            c += size_;
            free(slot); slot = fresh; size_ *= 2;
            ++expansions;
        }
        slot[num_++] = x; c += 1;
```

```
        totalActual += c; ++nOps;
        return c;
    }

    long long potential() const { return 2 * num_ - size_; }
    long long num() const { return num_; }
    long long size() const { return size_; }
    long long actual() const { return totalActual; }
    long long ops() const { return nOps; }
    long long expansionCount() const { return expansions; }

private:
    int* slot = nullptr;
    long long num_ = 0, size_ = 0, totalActual = 0, nOps = 0, expansions = 0;
};
```

Kode 14.4 (Lanjutan) *TABLE-INSERT dengan pelipatan, diinstrumentasi dengan fungsi potensialnya. Kompilasi dengan g++ -std=c++17 -O2 -o dyntable dyntable.cpp.*

```
int main() {
    DynamicTable T;
    cout << " i   num  size  actual c_i   Phi before  Phi after  amortized a_i  running total\n";
    for (int i = 1; i <= 17; ++i) {
        long long before = T.potential();
        long long c = T.insert(i);
        long long after = T.potential();
        cout << setw(3) << i << setw(6) << T.num() << setw(6) << T.size()
             << setw(12) << c << setw(13) << before << setw(11) << after
             << setw(15) << c + after - before << setw(15) << T.actual() << "\n";
    }

    DynamicTable U;
    const long long n = 1000000;
    for (long long i = 0; i < n; ++i) U.insert((int)i);
    cout << "\nn = " << n << " insertions into a table that starts empty:\n";
    cout << "  expansions performed   = " << U.expansionCount() << "\n";
    cout << "  total actual cost      = " << U.actual() << "\n";
    cout << "  aggregate bound 3n     = " << 3 * n << "\n";
    cout << "  measured cost per insert = " << fixed << setprecision(6)
         << (double)U.actual() / (double)U.ops() << "\n";
    cout << "  final num = " << U.num() << ", final size = " << U.size()
         << ", final potential = " << U.potential() << "\n";
    return 0;
}
```

Kode 14.4 memakai `malloc`, `memcpy`, dan `free` yang sungguhan, sehingga penyalinannya kerja nyata. Buku: Subbab 14.7.5, "Programnya".

Apa yang Dicetak Tabelnya

```
 i   num  size  actual c_i   Phi before  Phi after  amortized a_i  running total
  1     1     1           1            0          1              2              1
  2     2     2           2            1          2              3              3
  3     3     4           3            2          2              3              6
  4     4     4           1            2          4              3              7
  5     5     8           5            4          2              3             12
  6     6     8           1            2          4              3             13
  7     7     8           1            4          6              3             14
  8     8     8           1            6          8              3             15
  9     9    16           9            8          2              3             24
 10    10    16           1            2          4              3             25
 11    11    16           1            4          6              3             26
 12    12    16           1            6          8              3             27
 13    13    16           1            8         10              3             28
 14    14    16           1           10         12              3             29
 15    15    16           1           12         14              3             30
 16    16    16           1           14         16              3             31
 17    17    32          17           16          2              3             48

n = 1000000 insertions into a table that starts empty:
  expansions performed   = 20
  total actual cost      = 2048575
  aggregate bound 3n     = 3000000
  measured cost per insert = 2.048575
  final num = 1000000, final size = 1048576, final potential = 951424
```

Kolom amortisasinya dinding angka 3 (kecuali penyisipan awal). Kolom biaya sebenarnya mahal tepat pada $i = 2, 3, 5, 9, 17$. Pada $i = 16$ potensialnya mencapai 16, yang akan disalin perluasan berikutnya, dan pada $i = 17$ ambruk menjadi 2.

Sejuta penyisipan: 2.048.575 satuan terhadap batas terbukti 3.000.000, dengan hanya 20 perluasan.

Buku: Subbab 14.7.5, “Programnya”.

Penghapusan, dan Menyeparuhkan pada Separuh

Tanpa penyusutan, tabel yang tumbuh sampai sejuta lalu menyusut ke satu tetap memegang sejuta slot: tujuan ruangnya gagal. **Kebijakan A:** bila penghapusan membuat tabel kurang dari separuh penuh, separuhkan ukurannya.

Blok *thrashing* dari tabel penuh, $num = size = s$: `INSERT` (perluasan, biaya $s+1$); `DELETE` ($num = s$, tepat separuh, tanpa penyusutan, biaya 1); `DELETE` (di bawah separuh, penyusutan, biaya s); `INSERT` (biaya 1). Kembali ke titik awal.

Empat operasi berbiaya $(s+1)+1+s+1 = 2s+3$. Isi sampai $n/4$, lalu ulangi: sekitar $\frac{3n}{16}$ blok masing-masing $\frac{n}{2}$, jadi $\frac{3n^2}{32} = \Theta(n^2)$. **Kebijakan A berbiaya amortisasi $\Theta(n)$.**

> **Mengapa gagal: tidak ada kelonggaran.** Sesudah perluasan faktor muatannya tepat 1/2, persis titik pemicu penyusutan. **Batas amortisasi memerlukan ruang di antara titik-titik pemicunya.**

Buku: Subbab 14.7.6.

Perbaikannya: Separuhkan pada Seperempat Penuh

Kebijakan B. Lipatduakan bila penuh; separuhkan hanya bila penghapusan membuat tabel **kurang dari seperempat penuh**. Dari muatan 1/2, harus lewat $size/4$ penghapusan sebelum penyusutan dan $size/2$ penyisipan sebelum perluasan.

```
THRASHING TEST.  Fill the table to exactly full, then repeat the
block  INSERT, DELETE, DELETE, INSERT  until n/2 operations are done.

n =    1024
  A: shrink below 1/2  table=     256 ops=      512 cost=       65920 cost/op=     128.75  expansions=
128 contractions=128
  B: shrink below 1/4  table=     256 ops=      512 cost=         768 cost/op=       1.50  expansions=
1 contractions=0
n =    4096
  A: shrink below 1/2  table=    1024 ops=     2048 cost=     1050112 cost/op=     512.75  expansions=
512 contractions=512
  B: shrink below 1/4  table=    1024 ops=     2048 cost=        3072 cost/op=       1.50  expansions=
1 contractions=0
n =   16384
  A: shrink below 1/2  table=    4096 ops=     8192 cost=    16783360 cost/op=    2048.75  expansions=
2048 contractions=2048
  B: shrink below 1/4  table=    4096 ops=     8192 cost=       12288 cost/op=       1.50  expansions=
1 contractions=0
n =   65536
  A: shrink below 1/2  table=   16384 ops=    32768 cost=   268460032 cost/op=    8192.75  expansions=
8192 contractions=8192
  B: shrink below 1/4  table=   16384 ops=    32768 cost=       49152 cost/op=       1.50  expansions=
1 contractions=0
n =  262144
  A: shrink below 1/2  table=   65536 ops=   131072 cost=  4295065600 cost/op=   32768.75  expansions=
32768 contractions=32768
  B: shrink below 1/4  table=   65536 ops=   131072 cost=      196608 cost/op=       1.50  expansions=
1 contractions=0
```

```
Potential and worst case on a long mixed run (400000 inserts, then 400000 deletes):
  shrink below 1/2   cost=  1848555  cost/op=2.3107  worst single op=  262145  lowest potential=0  final
num=0 size=1
  shrink below 1/4   cost=  1586412  cost/op=1.9830  worst single op=  262145  lowest potential=0  final
num=0 size=2
```

Terukur: biaya per operasi Kebijakan A menjadi empat kali setiap kali n menjadi empat kali ($128{,}75$ sampai $32768{,}75$); Kebijakan B berbiaya $1{,}50$ pada setiap skala, dengan satu perluasan dan tanpa penyusutan. Faktor muatannya tetap di $[1/4, 1]$.

Histeresis. Bila sebuah struktur menata ulang dirinya pada suatu ambang, ambang untuk tumbuh dan ambang untuk menyusut tidak boleh berimpit, sama seperti termostat yang tidak menyala-padam setiap detik.

Keluaran program Subbab 14.7.6 yang membandingkan kedua kebijakan. Buku: Subbab 14.7.6, "Perbaikannya" dan "Mengukur kedua kebijakannya".

Potensial Gabungannya

Dengan $\alpha = num/size$:

$$\Phi(T) = \begin{cases} 2\,num[T] - size[T] & \text{bila } \alpha(T) \geq \frac{1}{2}, \\ \frac{size[T]}{2} - num[T] & \text{bila } \alpha(T) < \frac{1}{2}. \end{cases} \tag{14.15}$$

Persamaan (14.15) bernilai **nol pada separuh penuh** (kedua cabang sepakat), **tak negatif di mana-mana**, dan **naik menuju kedua bencana**: ke $size$ ketika $\alpha \to 1$ (biaya perluasan) dan ke $size/4$ ketika $\alpha \to 1/4$ (biaya penyusutan).

TEOREMA (tabel dinamis). Dengan pelipatan saat penuh dan penyeparuhan di bawah seperempat penuh, mulai dari tabel kosong, sembarang n operasi `TABLE-INSERT` dan `TABLE-DELETE` memakan $O(n)$ secara total; masing-masing $O(1)$ teramortisasi, dan paling banyak tiga perempat ruangnya pernah terbuang.

Analisis delapan kasusnya adalah Soal Latihan 14.12, dengan penyelesaian lengkap di buku. Buku: Subbab 14.7.7.

Union–Find: Janjinya Ditepati

Strukturnya. Himpunan sebagai pohon; tiap akar adalah wakilnya; `MakeSet`, `FindSet` (ikuti induk sampai akar), dan `Union`. Biaya `FindSet(x)` adalah kedalaman x, jadi pertanyaannya: **seberapa dalam pohonnya dapat menjadi?**

Gabung menurut peringkat. Gantungkan akar berperingkat lebih kecil di bawah yang lebih besar. Pohon berperingkat r memuat sekurang-kurangnya 2^r simpul (dua pohon berperingkat $r-1$ membentuk satu berperingkat r), sehingga $r \le \lg n$: setiap `FindSet` paling mahal $\lg n$, batas kasus terburuk yang sesungguhnya.

Pemampatan lintasan. Dalam perjalanan kembali dari akar, arahkan setiap simpul pada lintasan itu langsung ke akar. Satu pencarian masih bisa berbiaya $\Theta(\lg n)$, tetapi hanya sekali per lintasan: kerjanya adalah *investasi*.

> $O(\alpha(n))$ **adalah batas amortisasi.** Satu `FindSet` dapat berbiaya $\Theta(\lg n)$; banyak `FindSet` tidak dapat, karena yang pertama meratakan lintasannya. Seperti biasa, batasnya menyangkut jumlah.

Tarjan membuktikan batas $O(m\,\alpha(n))$ pada 1975 dengan fungsi potensial yang canggih; kita menyatakannya lalu mengukurnya. Buku: Subbab 14.8.1 sampai 14.8.4.

Apa Sebenarnya α Itu

Keluarga Ackermann, dengan $A_{k-1}^{(j+1)}$ berarti menerapkan A_{k-1} pada keluarannya sendiri $j+1$ kali:

$$A_k(j) = \begin{cases} j+1 & \text{bila } k = 0, \\ A_{k-1}^{(j+1)}(j) & \text{bila } k \geq 1. \end{cases} \tag{14.16}$$

Balikan dari Persamaan (14.16):

$$\alpha(n) = \min\left\{k : A_k(1) \geq n\right\}. \tag{14.17}$$

Dari definisinya: $A_0(1)=2$, $A_1(1)=3$, $A_2(1)=7$, $A_3(1)=A_2(7)=2^8 \cdot 8 - 1 = 2047$.

Tabel 14.1 Nilai-nilai $A_k(1)$ yang mendefinisikan α. Lompatan dari $k=3$ ke $k=4$ adalah lompatan dalam jenis: 2047 adalah bilangan yang dapat Anda tuliskan, sedangkan $A_4(1)$ tidak akan muat di alam semesta teramati.

k	$A_k(1)$
0	2
1	3
2	7
3	2047
4	jauh lebih besar daripada 2^{2048}

Membaca Persamaan (14.17) dari Tabel 14.1: $A_4(1)=A_3(2047)$, dan penerapan pertama dari 2048 kali penerapan A_2-nya saja sudah memiliki sekitar 620 angka.

Buku: Subbab 14.8.5.

Mengapa $\alpha(n) \le 4$, dan Pengukurannya

Tabel 14.2 Fungsi Ackermann balikan. Agar $\alpha(n)$ mencapai 5, Anda memerlukan lebih dari $A_4(1) > 2^{2048}$ elemen.

Rentang n	$\alpha(n)$
$n \le 2$	0
$n \le 3$	1
$n \le 7$	2
$n \le 2047$	3
$n \le A_4(1)$, yang melebihi 2^{2048}	4

Alam semesta teramati memuat sekitar $10^{80} \approx 2^{266}$ atom, jauh di bawah $A_4(1)$. Jadi menurut Tabel 14.2, $\alpha(n) \le 4$ untuk setiap masukan yang secara fisik mungkin ada. Fredman dan Saks membuktikan bahwa $\Omega(\alpha(n))$ teramortisasi tak terhindarkan bagi struktur berpenunjuk.

Diukur pada $n = 32768$**.** Hutan naif rata-rata 10.922 lompatan per operasi (tinggi 32.767). Gabung menurut peringkat saja dipaksa sampai tinggi 15, tepat $\lg n$, dengan 2,5 lompatan; peringkat ditambah pemampatan lintasan meratakannya ke tinggi 1 dengan **1,000** lompatan per operasi.

Perlakukan `FindSet` *dan* `Union` *sebagai konstan dalam taksiran rekayasa, sambil tahu bahwa batas jujurnya* $O(\alpha(n))$*. Buku: Subbab 14.8.6 dan 14.8.7.*

Memilih Metode

Tabel 14.3 Ketiga metodenya dibandingkan. Ketiganya setara kekuatannya untuk contoh-contoh di dalam bab ini; bedanya terletak pada berapa banyak kerja yang dituntut masing-masing dan berapa banyak yang diceritakannya kepada Anda.

	Agregat	Akuntansi	Potensial
Yang harus ditemukan	argumen pencacahan untuk total $T(n)$	tagihan per jenis operasi, dan aturan letak kreditnya	fungsi Φ atas keadaan struktur
Yang diperoleh	satu biaya amortisasi, sama untuk setiap operasi	biaya amortisasi berbeda untuk tiap jenis operasi	biaya amortisasi berbeda untuk tiap operasi, yang boleh bergantung pada keadaan
Yang harus dibuktikan	argumen pencacahannya	kredit tidak pernah negatif	$\Phi(D_i) \geq \Phi(D_0)$ untuk semua i
Paling cocok bila	ada argumen bersih "tiap objek disentuh paling banyak k kali"	pembayaran di muka melekat secara alami pada objek	pembayaran di muka adalah sifat bentuk strukturnya
Menjangkau	struktur sederhana saja	struktur yang cukup rumit	*splay tree*, *Fibonacci heap*, union–find

Urutan serangan menurut Tabel 14.3**.** Cobalah agregat lebih dahulu; pakailah akuntansi bila jenis operasinya sungguh berbeda; pakailah potensial bila masalahnya ada pada *bentuk* struktur, atau bila dua yang lain gagal.

Ketiganya bukan teorema yang berbeda. Ketiganya membuktikan $\sum c_i \leq \sum \hat{c}_i$. Bila dua metode memberi jawaban berbeda, salah satunya memuat kekeliruan hitung.

Buku: Subbab 14.9.

Profitina dalam Praktik

Perbatasan perayapan adalah tabel dinamis: perayapan m halaman melakukan penyalinan $O(m)$, bukan $O(m^2)$. **Penyangga sinyal adalah antrean dua-tumpukan**: produsen tidak pernah terhambat, `Enqueue` teramortisasi 3. **Pengelompokan halaman hampir-kembar adalah union–find.**

Profitina Laundry. Buku register penerimaan adalah tabel dinamis, tumpukan cucian yang menunggu adalah tumpukan yang dimuat ke mesin dengan `Multipop`, dan antrean pengambilan adalah antrean dua-tumpukan.

```
ordinary Tuesday           garments=  20000 events=   68324 work=   145646 work/event=   2.1317
expansions=15
worst day (one big load)   garments=  20000 events=   80001 work=   172767 work/event=   2.1596
expansions=15

Intake register alone: 20000 garments logged.
  most expensive single log entry = 16385 slot writes (a table expansion)
  total work                      = 52767
  amortized cost per garment      = 2.6383 slot writes
  aggregate bound proved in 14.7  = 3 per garment, i.e. 60000 in total
  final potential 2*num - size    = 7232
```

Hari terburuk berbiaya **2,16** satuan per peristiwa, Selasa biasa **2,13**: urutan peristiwa tidak berpengaruh, hanya banyaknya. Peristiwa tunggal terburuk adalah satu perluasan sebesar **16.385** operasi tulis slot, tidak berbahaya sebagai salinan memori tetapi risiko latensi yang nyata di bawah tenggat keras.

Aplikasinya memberi saran; penggunanya yang memutuskan. Buku: Subbab 14.10.

Kesalahan yang Sering Terjadi

1. "Amortisasi" untuk "kasus rata-rata": tanyakan di mana sebaran peluangnya. **2. "Amortisasi" untuk "biasanya cepat"**: satu operasi masih dapat berbiaya $\Theta(n)$; tenggat waktu memerlukan batas kasus terburuk.

3. Mengamortisasi melintasi pengaturan ulang: batasnya mencakup satu barisan tak terputus pada satu struktur. **4. Melupakan** $\Phi(D_0)$: tuliskan "$\Phi(D_0)=0$" dan "$\Phi \geq 0$" sebelum menghitung biaya amortisasi apa pun.

5. Menegaskan kredit tetap tak negatif tanpa menyebut letaknya. **6. Menyeparuhkan pada separuh penuh** (Gambar 14.19 hanya bekerja dengan kelonggaran). **7. Tumbuh dengan konstanta** alih-alih dengan faktor konstan.

8. Melupakan awal yang tidak kosong: tambahkan $\Phi(D_0)$ ke totalnya. **9. Mengacaukan kedua** α**. 10. Menulis** $O(1)$ **untuk** $O(\alpha(n))$ di dalam bukti.

Buku: Subbab 14.11.

Ringkasan — Lima Hal

1. **Pertanyaannya.** Berapa biaya *seluruh barisan*? Terburuk-tunggal-kali-n dapat melebihinya sebesar faktor n. Biaya amortisasi sah bila setiap awalan biaya sebenarnya tertutupi.
2. **Bukan kasus rata-rata.** Tanpa peluang; batasnya berlaku untuk setiap barisan, termasuk barisan lawan.
3. **Tiga metode, satu jawaban.** Agregat (tumpukan $2n$, pencacah $< 2n$, Soal 17.1-3 $\leq 3n$); akuntansi (kredit pada item, pada bit 1, $2j + k + 2$); potensial ($\hat{c}_i = c_i + \Delta\Phi_i$, sah bila $\Phi(D_i) \geq \Phi(D_0)$).
4. **Tabel dinamis.** Pelipatan memberi tepat 3 per penyisipan dengan $\Phi = 2\,num - size$; susutkan pada seperempat penuh, jangan pada separuh penuh.
5. **Union–find.** Gabung menurut peringkat membatasi tinggi dengan $\lg n$; pemampatan lintasan menjadikannya $O(\alpha(n))$ teramortisasi, dan $\alpha(n) \leq 4$ untuk setiap n yang mungkin ada.

Buku: Subbab 14.12.

Pertemuan Berikutnya

Bab 15 — Lintasan Terpendek Semua Pasangan. Kembali ke graf: $\delta(u, v)$ untuk setiap pasangan sekaligus, dengan relaksasi berulang yang mirip perkalian matriks.

Lintasan terpendek sebagai perkalian matriks atas $(\min, +)$, pengkuadratan berulang, pemrograman dinamis $\Theta(V^3)$ Floyd–Warshall, penutupan transitif, dan pembobotan ulang Johnson untuk graf jarang.

Kebiasaan yang perlu dijaga: *tanyakan berapa biaya seluruh barisan, bukan biaya langkah terburuknya*. Setiap `push_back`, setiap *rehash*, setiap penggabungan *Fibonacci heap* bersandar pada batas amortisasi.

Pekerjaan rumah. Soal Latihan 14.1–14.17; mulailah dengan 14.4 (pangkat 3), 14.9 (faktor pertumbuhan 3), dan 14.17 (increment dan decrement).

Buku: Subbab 14.12.1 dan 14.13.

Kuliah 15 — Lintasan Terpendek Semua Pasangan

Perancangan dan Analisis Algoritma (EF234405)

Bab 15 buku ini dalam bentuk ringkas, untuk dipakai di kelas.

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*.

Pengajar: Irfan Subakti — yifana@gmail.com

Hari ini: rute termurah antara setiap pasangan terurut simpul, dalam satu tabel, lewat empat algoritma yang ongkosnya dapat kita bandingkan dengan angka, bukan dengan pendapat. Buku: Bab 15, paragraf pembuka.

Agenda

1. **Persoalannya.** Matriks masukan W, matriks keluaran D dan Π, dan mencetak sebuah rute.
2. **Harga yang naif.** $|V|$ kali Dijkstra atau Bellman–Ford, dan mengapa Dijkstra tetap tidak boleh dipakai.
3. **Lintasan terpendek sebagai perkalian matriks** atas $(\min, +)$: SLOW-APSP dalam $\Theta(n^4)$, lalu pemangkatan berulang dalam $\Theta(n^3 \lg n)$.
4. **Floyd–Warshall** dalam $\Theta(n^3)$: rekurensi, urutan gelung, pendahulu, dua putaran dengan tangan, keamanan di tempat, uji siklus negatif.
5. **Penutupan transitif** dengan menukar dua operator; **6. pembobotan ulang Johnson** untuk graf jarang; lalu algoritma mana dan kapan, Profitina, kesalahan, ringkasan.

Halaman-halamannya mengikuti urutan buku; soal latihan di akhir bab adalah pekerjaan rumahnya. Buku: Subbab 15.2 sampai 15.10.

Dari Mana Kita Melanjutkan

Bab 13, sumber tunggal. $w(p)=\sum_{i=1}^{k} w(v_{i-1},v_i)$; $\delta(u,v)$ bobot terkecil sebuah lintasan, ∞ bila tidak ada; estimasi $d[v]$ dan pendahulu $\pi[v]$, yang hanya diturunkan oleh **relaksasi**.

Tiga strategi. Dijkstra (bobot ≥ 0): $O(V^2+E)$ dengan larik, $O(E \lg V)$ dengan tumpukan biner, $O(V \lg V+E)$ dengan *Fibonacci heap*. Bellman–Ford (bobot apa pun): $O(V\,E)$, dengan uji siklus negatif. Algoritma DAG: $\Theta(V+E)$.

> **TEOREMA 13.1, DIINGAT KEMBALI.** Sublintasan dari lintasan terpendek adalah lintasan terpendek. Inilah seluruh alasan pemrograman dinamis berlaku bagi lintasan terpendek, dan hari ini ia dipakai dalam dua samaran yang berbeda.

Sisi negatif diterima, siklus negatif tidak. Contoh berjalan hari ini mempunyai dua sisi negatif dan tanpa siklus negatif; Subbab 15.6.14 memperlihatkan cara memeriksanya tanpa biaya.

Buku: Subbab 15.1.1.

Pertanyaan yang Baru

DEFINISI (lintasan terpendek semua pasangan). Diberikan $G=(V,E)$ dengan $w:E\to R$, temukan untuk **setiap pasangan terurut** $u,v\in V$ sebuah lintasan terpendek dari u ke v beserta bobotnya $\delta(u,v)$. Keluarannya sebuah tabel: baris u, kolom v.

Terurut: di dalam graf berarah, $u\to v$ dan $v\to u$ adalah perjalanan yang berbeda. Tabelnya memuat $|V|^2$ entri dan tidak simetris.

Mengapa seluruh tabel? Di mana depot diletakkan; **diameter** sebuah jaringan (nilai $\delta(u,v)$ terbesar); dua halaman yang terjauh dalam cacah klik; penutupan transitif; dan praproses untuk pencarian $O(1)$.

Mulai sekarang $n=|V|$, dan kita melepas palang di dalam notasi asimtotik: $O(V^3)$ berarti $O(|V|^3)$.

Buku: Subbab 15.1.2.

Masukannya: Matriks Ketetanggaan

Beri nomor simpul-simpulnya $1, 2, \ldots, n$ dan simpan bobot sisi graf berarah $G = (V, E)$ dalam matriks $n \times n$ $W = (w_{ij})$:

$$w_{ij} = \begin{cases} 0 & \text{jika } i = j \\ \text{bobot sisi } (i,j) & \text{jika } i \neq j \text{ dan } (i,j) \in E \\ \infty & \text{jika } i \neq j \text{ dan } (i,j) \notin E \end{cases}$$

Sisi berbobot negatif diperbolehkan; siklus berbobot negatif tidak.

Gambar 15.1 Masukan setiap algoritma di bab ini: matriks ketetanggaan $n \times n$, $W = (w_{ij})$, dari Persamaan (15.1). Sisi berbobot negatif diperbolehkan; siklus berbobot negatif tidak.

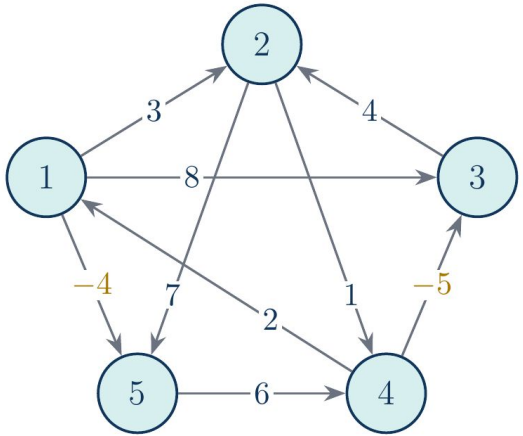
Dituliskan lengkap, matriks pada Gambar 15.1 adalah

$$w_{ij} = \begin{cases} 0 & \text{bila } i = j, \\ \text{bobot sisi berarah } (i,j) & \text{bila } i \neq j \text{ dan } (i,j) \in E, \\ \infty & \text{bila } i \neq j \text{ dan } (i,j) \notin E. \end{cases} \tag{15.1}$$

Pada Persamaan (15.1), $w_{ii}=0$ memikul beban dua kali hari ini; ∞ tidak pernah memenangkan min dan meracuni setiap jumlah, sehingga program-programnya memakai `INF = 1000000000LL` dan mengawal setiap penjumlahan dari `INF + INF`.

Buku: Subbab 15.2.1.

Contoh Berjalan



Gambar 15.2 Contoh berjalan: sebuah graf berarah bersimpul lima dengan sembilan sisi, dua di antaranya negatif ($1 \to 5$ berbobot -4 dan $4 \to 3$ berbobot -5). Ia tidak punya siklus berbobot negatif (§15.6.14 memeriksanya dengan mesin), sehingga setiap $\delta(i, j)$ berhingga.

Menerapkan Persamaan (15.1) pada Gambar 15.2:

$$W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}.$$

Baris 1: diam di tempat 0, ke 2 dengan 3, ke 3 dengan 8, tidak ada sisi ke 4, ke 5 dengan -4. W tidak simetris: $w_{15} = -4$ tetapi $w_{51} = \infty$.

Menyimpan W berongkos $\Theta(n^2)$ berapa pun banyaknya sisi, dan itu tepat karena keluarannya memang $\Theta(n^2)$. Buku: Subbab 15.2.1.

Keluarannya: Dua Matriks

Jarak. $D=(d_{ij})$ dengan, pada saat berhenti,

$$d_{ij}=\delta(i,j) \text{ untuk semua } i,j\in V.$$

(15.2)

DEFINISI. Matriks pendahulu $\Pi=(\pi_{ij})$ memuat $\pi_{ij}=$ NIL bila $i=j$ atau tidak ada lintasan, dan selainnya **pendahulu** j **pada suatu lintasan terpendek dari** i: simpul *kedua dari akhir*, bukan simpul kedua.

Baris i dari Π menyandikan **subgraf pendahulu** $G_{\pi,i}$, dengan $V_{\pi,i}=\{j:\pi_{ij}\neq\text{NIL}\}\cup\{i\}$ dan $E_{\pi,i}=\{(\pi_{ij},j):j\in V_{\pi,i}-\{i\}\}$: pohon lintasan terpendek berakar di i. Kita membangun n pohon sekaligus.

D **memberi tahu *berapa*;** Π **memberi tahu *bagaimana*.** D adalah daftar harga, Π setumpuk n peta jalan, masing-masing dilipat menjadi satu baris. Keduanya $\Theta(n^2)$, jadi kita menyimpan keduanya.

Metode matriks pada Subbab 15.4 menamai matriksnya $L^{(m)}$*;* L *yang terakhir adalah* D *yang sama seperti pada Persamaan (15.2). Buku: Subbab 15.2.2.*

Mencetak Lintasan dari Π

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
1  jika i == j
2      cetak i
3  jika tidak, jika π_ij == NIL
4      cetak "tidak ada lintasan dari" i "ke" j
5  selain itu PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
6      cetak j
```

Gambar 15.3 PRINT-ALL-PAIRS-SHORTEST-PATH(Π , i , j) memulihkan rute yang sesungguhnya dari matriks pendahulunya, mencetaknya maju sekalipun Π menyimpannya mundur.

Pada Gambar 15.3 panggilan rekursifnya datang *sebelum* pencetakan: $i = j$ cetak i; π_{ij} = **NIL** katakan tidak ada lintasan; **selainnya** cetak rute dari i ke π_{ij}, lalu cetak j.

Linear dalam panjang lintasannya, paling banyak n; mencetak semua n^2 lintasan berongkos $O(n^3)$, sama dengan menghitungnya.

Pemeriksaan ukuran: *pada* $n = 1000$ *jawabannya sejuta bilangan; pada* $n = 100.000$ *keluarannya saja delapan puluh gigabita dan semua pasangan adalah pertanyaan yang keliru. Buku: Subbab 15.2.3 dan 15.2.4.*

Algoritma yang Kentara, dan Harganya

Pendekatan naif: jalankan algoritma sumber tunggal sekali per simpul lalu tumpuk jawabannya. Benar, sederhana, dan kadang-kadang yang tercepat yang dikenal.

Tabel 15.1 Biaya menyelesaikan lintasan terpendek semua pasangan dengan menjalankan algoritma sumber tunggal sekali per simpul. Baris terakhirnyalah yang menyakitkan, dan baris itulah yang diserang sisa bab ini.

Keadaan	Algoritma sumber tunggal, dijalankan V kali	Biaya total	Pada graf padat ($E=\Theta(V^2)$)
semua bobot ≥ 0, antrean larik	Dijkstra	$O(V^3+V E)$	$O(V^3)$
semua bobot ≥ 0, tumpukan biner	Dijkstra	$O(V E \lg V)$	$O(V^3 \lg V)$
semua bobot ≥ 0, *Fibonacci heap*	Dijkstra	$O(V^2 \lg V+V$	$O(V^3)$
ada bobot < 0	Bellman–Ford	$O(V^2 E)$	$O(V^4)$

Pada Tabel 15.1 tumpukan biner unggul hanya bila $E=o(V^2/\lg V)$; baris Fibonacci adalah batas yang diwarisi Johnson. Bencananya ada di baris terakhir: $O(V^4)$, sekitar 6×10^{10} relaksasi pada $n=500$.

Floyd–Warshall akan menyamai $O(V^3)$ dengan sisi negatif diizinkan; Johnson akan mengalahkannya pada graf jarang. Buku: Subbab 15.3.1, 15.3.2, dan 15.3.4.

Mengapa Tidak Tetap Memakai Dijkstra?

Dijkstra memfinalkan simpul berestimasi terkecil, sebab melangkah *lebih jauh* hanya dapat lebih mahal. Sisi negatif mematahkan itu: melangkah lebih jauh dapat membuat rute lebih murah. Program pada Subbab 15.3.3 mengaudit Dijkstra dari setiap simpul terhadap Floyd–Warshall:

```
(a) the five-vertex running example:
   entries Dijkstra got wrong: 0 out of 25

(b) a four-vertex graph built to expose the flaw:
   delta(1,2): Dijkstra says 1, the truth is -1  <-- WRONG
   delta(1,4): Dijkstra says 2, the truth is 0  <-- WRONG
   entries Dijkstra got wrong: 2 out of 16

Moral: on (a) Dijkstra happened to be right, and that is worse than
being wrong, because it invites you to trust it on (b).
```

Pada graf (b), dari simpul 1 Dijkstra memfinalkan simpul 2 pada 1 dan merelaksasi $2 \to 4$ menjadi 2; baru sesudahnya ia menemukan $1 \to 3 \to 2$ seharga $2+(-3)=-1$, terlambat untuk mencapai simpul 4. Kebenarannya $\delta(1,4)=0$.

> **Algoritma yang hanya kadang-kadang salah lebih berbahaya daripada yang selalu salah,** sebab pengujian tidak akan menemukannya. Bobotlah ulang lebih dahulu (Subbab 15.8), atau pakailah Floyd–Warshall.

Buku: Subbab 15.3.3.

Langkah 1: Strukturnya, menurut Banyaknya Sisi

Resep pemrograman dinamis Bab 7: cirikan solusi optimal; definisikan nilainya secara rekursif; hitung secara bawah-atas; susun solusinya. Hari ini kita mengikutinya dua kali, dengan dua pencirian yang berbeda.

- Setiap sublintasan dari lintasan terpendek adalah lintasan terpendek (Teorema 13.1).
- Misalkan p lintasan terpendek dari i ke j dengan paling banyak m sisi; tanpa siklus berbobot negatif, m semacam itu berhingga.
- Bila $i = j$, maka p berbobot 0 dan tidak memuat sisi.
- Bila $i \neq j$, pecah p menjadi $i \overset{p'}{\rightsquigarrow} k \to j$, dengan p' memuat paling banyak $m-1$ sisi. Menurut Teorema 13.1, p' lintasan terpendek dari i ke k, jadi $\delta(i,j) = \delta(i,k) + w_{kj}$.

Gambar 15.4 Struktur sebuah lintasan terpendek yang bersisi paling banyak m. Bila $i \neq j$, penggallah sisi terakhirnya dan sisanya haruslah lintasan terpendek pula — inilah Teorema 13.1 sedang bekerja.

Seperti pada Gambar 15.4: bila $i = j$ lintasan terbaiknya kosong, berbobot 0. Bila $i \neq j$, lepaskan sisi terakhir $k \to j$; sisanya p' bersisi paling banyak $m-1$ dan terpendek menurut Teorema 13.1, sehingga $\delta(i,j) = \delta(i,k) + w_{kj}$. Kita tidak tahu k, tetapi calonnya hanya n.

Tutuplah ruas jalan terakhir dengan ibu jari: sisanya haruslah jalan termurah ke tempat ibu jari itu bermula. Buku: Subbab 15.4.1 dan 15.4.2.

Langkah 2: Mendefinisikan $l_{ij}^{(m)}$

- Misalkan $l_{ij}^{(m)}$ bobot minimum dari lintasan mana pun dari i ke j yang memuat **paling banyak** m sisi.
- Untuk $m = 0$ ada lintasan dari i ke j tanpa sisi jika dan hanya jika $i = j$, sehingga

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{jika } i = j \\ \infty & \text{jika } i \neq j \end{cases}$$

- Untuk $m \geq 1$, $l_{ij}^{(m)}$ adalah yang lebih kecil di antara $l_{ij}^{(m-1)}$ (lintasan terbaik dengan paling banyak $m-1$ sisi) dan lintasan terbaik dengan tepat m sisi, yang ditemukan dengan mencoba setiap pendahulu k yang mungkin bagi j.

Gambar 15.5 Definisi $l_{ij}^{(m)}$ beserta kasus dasarnya $l_{ij}^{(0)}$. Perhatikanlah bahwa superskripnya berada di dalam tanda kurung — ia mencacah sisi, ia bukan pangkat.

DEFINISI. $l_{ij}^{(m)}$ adalah **bobot minimum sembarang lintasan dari i ke j yang bersisi paling banyak m**, ∞ bila tidak ada. Superskrip (m) pada Gambar 15.5 adalah *jatah sisi*, bukan pangkat.

Kasus dasar: tanpa sisi Anda tidak dapat bergerak,

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{bila } i = j, \\ \infty & \text{bila } i \neq j. \end{cases} \tag{15.3}$$

Dalam bentuk matriks, Persamaan (15.3) bernilai 0 di diagonal dan ∞ di tempat lain: Subbab 15.4.9 memperlihatkan bahwa ia adalah identitas perkalian baru kita.

Buku: Subbab 15.4.3.

Rekurensinya, dan Penyederhanaannya

Secara rekursif:

$$l_{ij}^{(m)} = \min\left(l_{ij}^{(m-1)}, \min_{1\le k\le n}\{l_{ik}^{(m-1)} + w_{kj}\}\right)$$
$$= \min_{1\le k\le n}\{l_{ik}^{(m-1)} + w_{kj}\} \qquad \text{sebab } w_{jj} = 0 \text{ untuk semua } j$$

Tanpa siklus berbobot negatif, setiap kali $\delta(i,j) < \infty$ ada lintasan terpendek dari i ke j yang sederhana, sehingga memuat paling banyak $n-1$ sisi; lintasan dengan sisi lebih banyak tidak mungkin lebih ringan. Karena itu

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \cdots$$

Gambar 15.6 Rekurensi Persamaan (15.4), penyederhanaannya yang satu baris, dan fakta, Persamaan (15.6), bahwa barisan $l_{ij}^{(m)}$ berhenti berubah pada $m = n-1$.

$$l_{ij}^{(m)} = \min\left(l_{ij}^{(m-1)}, \min_{1\le k\le n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right). \tag{15.4}$$

Pada Persamaan (15.4), yang dirangkum Gambar 15.6, argumen pertama menangani sisi tambahan yang tidak diperlukan; argumen kedua mencoba setiap lompatan terakhir $k \to j$.

Calon $k = j$ memberi $l_{ij}^{(m-1)} + w_{jj} = l_{ij}^{(m-1)}$, sehingga argumen pertama sudah termuat di dalam minimum bagian dalam:

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\}, w_{jj} = 0. \tag{15.5}$$

Satu minimum atas satu indeks adalah bentuk sebuah hasil kali matriks; dengan ∞ *di diagonal* W*, peringkasan menjadi Persamaan (15.5) akan gagal. Periksa:* $l_{24}^{(2)} = \min\{\infty, 1, \infty, 1, 13\} = 1$*. Buku: Subbab 15.4.4 dan 15.4.5.*

Kapan Barisannya Berhenti Berubah?

Setiap lintasan terpendek bersisi paling banyak $n-1$ (Teorema 13.4): buang setiap siklus, yang berbobot ≥ 0, sampai lintasannya sederhana.

Sisi tambahan tidak pernah menolong: tidak ada yang mengalahkan lintasan terpendek. Jadi

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \cdots \tag{15.6}$$

Di mana "tanpa siklus negatif" dipakai. Pada separuh pertama argumen untuk Persamaan (15.6), dan hanya di sana. Dengan siklus negatif, $l_{ij}^{(m)}$ akan terus turun selamanya, dan itu benar, sebab $\delta(i,j) = -\infty$.

Barisan $l_{ij}^{(0)}, l_{ij}^{(1)}, \ldots$ *tidak naik dan konstan sejak* $n-1$. *Buku: Subbab 15.4.6.*

Langkah 3: EXTEND-SHORTEST-PATHS

Mulailah dari $L^{(1)} = W$ (dengan Persamaan (15.3), hanya $k = i$ yang bertahan, memberi w_{ij}) lalu hitung $L^{(2)}, \ldots, L^{(n-1)}$, satu sisi jatah demi satu sisi jatah.

```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  misalkan L' = (l'_ij) sebuah matriks n × n baru
3  untuk i = 1 sampai n
4      untuk j = 1 sampai n
5          l'_ij = ∞
6          untuk k = 1 sampai n
7              l'_ij = min(l'_ij, l_ik + w_kj)
8  kembalikan L'
```

Gambar 15.7 EXTEND-SHORTEST-PATHS(L, W), yang mengambil sebuah matriks berisi rute-terbaik-sejauh-ini lalu memperpanjang setiap satunya dengan satu sisi lagi. Tiga gelung bersarang, sehingga waktunya $\Theta(n^3)$.

Pada Gambar 15.7, baris 3–4 mengunjungi setiap sel, baris 5 menyetelnya ke ∞ (identitas bagi min), dan baris 6–7 memelihara minimum berjalan $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$. Tiga gelung bersarang: $\Theta(n^3)$.

Buku: Subbab 15.4.7.

Ia Adalah Perkalian Matriks

SQUARE-MATRIX-MULTIPLY(A, B)

```
1  n = A.rows
2  misalkan C = (c_ij) sebuah matriks n × n baru
3  untuk i = 1 sampai n
4      untuk j = 1 sampai n
5          c_ij = 0
6          untuk k = 1 sampai n
7              c_ij = c_ij + a_ik · b_kj
8  kembalikan C
```

Gambar 15.8 SQUARE-MATRIX-MULTIPLY(A, B), algoritma sekolahan $\Theta(n^3)$ untuk $C = A \cdot B$. Taruhlah ia berdampingan dengan Gambar 15.7 dan keduanya berbeda di persis dua tempat.

Tabel 15.2 EXTEND-SHORTEST-PATHS adalah SQUARE-MATRIX-MULTIPLY dengan $\min$ menggantikan $+$, $+$ menggantikan $\times$, dan ∞ menggantikan 0 sebagai nilai awalnya; baris 1–4 sama persis.

Baris	SQUARE-MATRIX-MULTIPLY	EXTEND-SHORTEST-PATHS
5	$c_{ij} = 0$	$l'_{ij} = \infty$
6	**for** $k = 1$ **to** n	**for** $k = 1$ **to** n
7	$c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$	$l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$
8	**return** C	**return** L'

Tabel 15.2, dibaca berdampingan dengan Gambar 15.8: $+ \to \min$, $\times \to +$, $0 \to \infty$. Dengan $A \odot B$ untuk hasilnya, $L^{(m)} = L^{(m-1)} \odot W = W^m$, dan $L^{(n-1)} = W^{n-1}$ memuat jawabannya.

Superskrip pada W adalah pangkat sungguhan terhadap $\odot$; superskrip berkurung pada L adalah jatah sisi. Nilainya sama. Buku: Subbab 15.4.8.

Semiring $(\min, +)$

Sebuah **semiring** mempunyai "penjumlahan" $\oplus$ yang asosiatif dan komutatif dengan identitas $\acute{0}$, "perkalian" $\otimes$ yang asosiatif dengan identitas $\acute{1}$, dan hukum distributif. $R \cup \{\infty\}$ dengan $\oplus = \min$, $\otimes = +$ adalah salah satunya: $\acute{0} = \infty$, $\acute{1} = 0$, dan $a + \min(b, c) = \min(a + b, a + c)$.

Tiga akibat gratis. Matriks identitasnya adalah $L^{(0)}$ pada Persamaan (15.3). Hasil kali $\odot$ bersifat **asosiatif**, yang dijual Subbab 15.5. Dan semiring lain menyelesaikan persoalan lain dengan gelung rangkap tiga yang sama.

$(\vee, \wedge)$ memberi keterjangkauan (Subbab 15.7); $(\max, \min)$ memberi lintasan terlebar; $(+, \times)$ atas cacah lintasan mencacah lintasan terpendek (Soal Latihan 15.14).

Buku: Subbab 15.4.9.

SLOW-ALL-PAIRS-SHORTEST-PATHS

```
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)
1  n = W.rows
2  L^(1) = W
3  untuk m = 2 sampai n − 1
4      misalkan L^(m) sebuah matriks n × n baru
5      L^(m) = EXTEND-SHORTEST-PATHS(L^(m−1), W)
6  kembalikan L^(n−1)
```

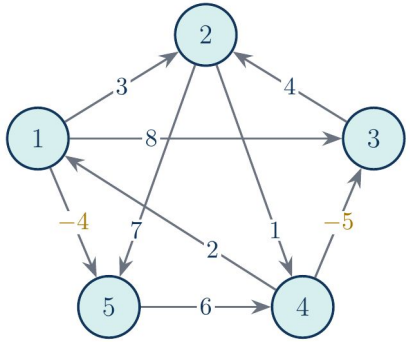
Gambar 15.9 SLOW-ALL-PAIRS-SHORTEST-PATHS(W). Ia melakukan $n-2$ panggilan ke EXTEND-SHORTEST-PATHS, masing-masing berongkos $\Theta(n^3)$, sehingga totalnya $\Theta(n^4)$.

Gambar 15.9: baris 2 menyetel $L^{(1)}=W$; baris 3–5 menghitung $L^{(m)}=L^{(m-1)}\odot W$ untuk $m=2,\ldots,n-1$; baris 6 mengembalikan $L^{(n-1)}$. $(n-2)\cdot\Theta(n^3)=\Theta(n^4)$.

Tidak lebih baik daripada $|V|\times$ Bellman–Ford pada graf padat. Tujuannya adalah *wawasan*, yang diubah menjadi penghematan pada halaman-halaman berikut.

Buku: Subbab 15.4.10.

Setiap Matriks Antaranya



$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Gambar 15.10 Graf bersimpul lima itu beserta barisan $L^{(1)}, L^{(2)}, L^{(3)}, L^{(4)}$ yang dihitung SLOW-ALL-PAIRS-SHORTEST-PATHS. Karena $n=5$, matriks terakhirnya $L^{(4)}=L^{(n-1)}$ memuat bobot lintasan terpendeknya, dan $L^{(5)}=L^{(4)}\odot W$ sama dengan $L^{(4)}$.

Bacalah l_{12} melintasi Gambar 15.10: 3 pada $L^{(1)}$, $L^{(2)}$, $L^{(3)}$, lalu 1 pada $L^{(4)}$, sebab rute termurah $1\to5\to4\to3\to2$, berbobot $-4+6-5+4=1$, memerlukan keempat sisinya. Entri itulah sebabnya algoritma ini tidak boleh berhenti lebih awal.

Program pada Subbab 15.4.11 mereproduksi setiap matriksnya, dan hasil kali kelima tidak mengubah apa pun. Buku: Subbab 15.4.11 dan 15.4.13.

Baris 4 dari $L^{(2)}$, Sel demi Sel

Menurut Persamaan (15.5) dengan $m=2$ dan baris 4 dari W sama dengan $(2,\infty,-5,0,\infty)$, hanya $k\in\{1,3,4\}$ yang dapat menyumbang.

$l_{41}^{(2)}=\min\{2+0,\infty,\infty,0+2,\infty\}=2$. $l_{42}^{(2)}=\min\{2+3,\infty,-5+4,\infty,\infty\}=-1$, lewat $4\to3\to2$: sisi negatifnya membayar.

$l_{43}^{(2)}=\min\{2+8,\infty,-5+0,0+(-5),\infty\}=-5$. $l_{44}^{(2)}=0$. $l_{45}^{(2)}=\min\{2+(-4),\infty,\infty,\infty,\infty\}=-2$, lewat $4\to1\to5$.

Baris 4 dari $L^{(2)}$ adalah $(2,-1,-5,0,-2)$; baris 2 adalah $(3,0,-4,1,7)$. *Setiap hasil kali membiarkan setiap rute bertambah satu sisi lagi.*

Buku: Subbab 15.4.12.

Pemangkatan Berulang

Kita hanya menginginkan matriks terakhir, dan menurut Persamaan (15.6) jatah berapa pun yang $\geq n-1$ memberikannya: kita bebas melampaui. **Asosiativitas** memperbolehkan $W^4 = (W \odot W) \odot (W \odot W)$: dua hasil kali alih-alih tiga.

Yang kita perlukan hanya $L^{(n-1)}$, dan $L^{(m)} = L^{(n-1)}$ untuk setiap $m \geq n-1$. Karena "perkalian" EXTEND-SHORTEST-PATHS bersifat asosiatif, kuadratkan berulang-ulang:

$$\begin{aligned} L^{(1)} &= W, \\ L^{(2)} &= W^2 = W \cdot W, \\ L^{(4)} &= W^4 = W^2 \cdot W^2, \\ L^{(8)} &= W^8 = W^4 \cdot W^4, \\ &\vdots \\ L^{(2^{\lceil \lg(n-1) \rceil})} &= W^{2^{\lceil \lg(n-1) \rceil}} = W^{2^{\lceil \lg(n-1) \rceil - 1}} \cdot W^{2^{\lceil \lg(n-1) \rceil - 1}}. \end{aligned}$$

Karena $2^{\lceil \lg(n-1) \rceil} \geq n-1$, hasil kali terakhir sama dengan $L^{(n-1)}$ sesudah hanya $\lceil \lg(n-1) \rceil$ perkalian.

Gambar 15.11 Hanya $\lceil \lg(n-1) \rceil$ hasil kali yang diperlukan, sebab masing-masing melipatduakan jatah sisinya. Hasil kali terakhirnya $L^{(2^{\lceil \lg(n-1) \rceil})}$ sama dengan $L^{(n-1)}$ menurut Persamaan (15.6).

Gambar 15.11: jatahnya $1, 2, 4, 8, \ldots$; sesudah t pengkuadratan jatahnya 2^t, dan t terkecil dengan $2^t \geq n-1$ adalah $\lceil \lg(n-1) \rceil$. Untuk $n = 1000$: 10 hasil kali alih-alih 998.

Akal yang sama menghitung a^b dalam $\lg b$ perkalian, di sini dalam semiring yang lain. Buku: Subbab 15.5.1 sampai 15.5.4.

FASTER-ALL-PAIRS-SHORTEST-PATHS

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
1  n = W.rows
2  L^(1) = W
3  m = 1
4  selama m < n − 1
5      misalkan L^(2m) sebuah matriks n × n baru
6      L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
7      m = 2m
8  kembalikan L^(m)
```

Gambar 15.12 FASTER-ALL-PAIRS-SHORTEST-PATHS(W). Baris 6 mengkuadratkan matriks yang sedang berjalan alih-alih mengalikannya dengan W, dan itulah seluruh bedanya dari Gambar 15.9.

Tabel 15.3 Kedua jadwalnya pada contoh berjalan bersimpul lima. Versi lambatnya menambahkan satu pada jatah sisinya tiap hasil kali; versi cepatnya melipatduakannya.

Langkah	SLOW-APSP menghitung	Jatah	FASTER-APSP menghitung	Jatah
awal	$L^{(1)} = W$	1	$L^{(1)} = W$	1
1	$L^{(2)} = L^{(1)} \odot W$	2	$L^{(2)} = L^{(1)} \odot L^{(1)}$	2
2	$L^{(3)} = L^{(2)} \odot W$	3	$L^{(4)} = L^{(2)} \odot L^{(2)}$	4 — berhenti
3	$L^{(4)} = L^{(3)} \odot W$	4 — berhenti	—	—
hasil kali	$n - 2 = 3$		$\lceil \lg(n-1) \rceil = 2$	

Gambar 15.12 mengubah dua baris: kuadratkan alih-alih mengalikan dengan W, dan lipatduakan m. Tabel 15.3 membandingkan jadwalnya. Ongkosnya: $\Theta(n^3 \lceil \lg(n-1) \rceil) = \Theta(n^3 \lg n)$.

Buku: Subbab 15.5.5 dan 15.5.6.

Pemangkatan Berulang dalam C++

Kode 15.1 FASTER-ALL-PAIRS-SHORTEST-PATHS, versi pemangkatan berulangnya. Kompilasilah dengan `g++ -std=c++17 -O2 -o faster_apsp faster_apsp.cpp`.

```cpp
// Bab 15, Kode: FASTER-ALL-PAIRS-SHORTEST-PATHS -- pemangkatan berulang.
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
using namespace std;

const long long INF = 1000000000LL;
typedef vector<vector<long long>> Matrix;

Matrix extendShortestPaths(const Matrix& A, const Matrix& B) {
    int n = (int)A.size();
    Matrix C(n, vector<long long>(n, INF));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) {
            C[i][j] = INF;
            for (int k = 0; k < n; ++k)
                if (A[i][k] < INF && B[k][j] < INF)
                    C[i][j] = min(C[i][j], A[i][k] + B[k][j]);
        }
    return C;
}

void printMatrix(const string& name, const Matrix& M) {
    int n = (int)M.size();
    cout << name << " =\n";
    for (int i = 0; i < n; ++i) {
        cout << "   ";
```

```
        for (int j = 0; j < n; ++j) {
            if (M[i][j] >= INF) cout << setw(5) << "inf";
            else                cout << setw(5) << M[i][j];
        }
        cout << '\n';
    }
    cout << '\n';
}
```

Kode 15.1 (Lanjutan) *FASTER-ALL-PAIRS-SHORTEST-PATHS, versi pemangkatan berulangnya. Kompilasilah dengan g++ -std=c++17 -O2 -o faster_apsp faster_apsp.cpp.*

```cpp
int main() {
    const int n = 5;
    Matrix W = {
        {  0,   3,   8, INF,  -4},
        {INF,   0, INF,   1,   7},
        {INF,   4,   0, INF, INF},
        {  2, INF,  -5,   0, INF},
        {INF, INF, INF,   6,   0}
    };

    // FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
    Matrix L = W;            // baris 2: L^(1) = W
    int m = 1;               // baris 3
    int products = 0;
    printMatrix("L^(1)", L);
    while (m < n - 1) {                    // baris 4
        L = extendShortestPaths(L, L);     // baris 6: L^(2m) = L^(m) . L^(m)
        ++products;
        m = 2 * m;                         // baris 7
        printMatrix("L^(" + to_string(m) + ")", L);
    }
    cout << "n = " << n
         << ", matrix products performed = " << products
         << ", final exponent m = " << m << '\n';
    return 0;
}
```

```
L^(1) =
       0    3    8  inf   -4
     inf    0  inf    1    7
     inf    4    0  inf  inf
       2  inf   -5    0  inf
     inf  inf  inf    6    0

L^(2) =
       0    3    8    2   -4
       3    0   -4    1    7
     inf    4    0    5   11
       2   -1   -5    0   -2
       8  inf    1    6    0

L^(4) =
       0    1   -3    2   -4
       3    0   -4    1   -1
       7    4    0    5    3
       2   -1   -5    0   -2
       8    5    1    6    0

n = 5, matrix products performed = 2, final exponent m = 4
```

Kode 15.1 melompat dari jatah 2 ke jatah 4; $L^{(3)}$ tidak pernah dihitung, dan $L^{(4)}$-nya identik dengan milik versi lambat.

Buku: Subbab 15.5.6.

Yang Tidak Diberikan Pemangkatan

Ia tetap $\Theta(n^3 \lg n)$, dan Floyd–Warshall mengerjakan $\Theta(n^3)$ dengan konstanta yang lebih kecil. Nilai pandangan matriks ini adalah *gagasannya*: lintasan terpendek hidup di dalam sebuah semiring.

Strassen tidak menolong. Perkalian matriks cepat bersandar pada **pengurangan**, dan $(\min, +)$ tidak memilikinya: tidak ada yang membatalkan sebuah min. Algoritma semua pasangan sub-kubik yang ada menempuh jalan yang lain.

> **Gagasannya menyebar.** Gelung rangkap tiga yang sama, atas semiring yang berbeda, memberi penutupan transitif, lintasan terlebar, dan cacah lintasan; Subbab 15.7 adalah contoh pertamanya.

Buku: Subbab 15.5.7.

Floyd–Warshall: Cara Memotong yang Lain

Pemrograman dinamis dalam $\Theta(V^3)$ yang mengizinkan sisi negatif tetapi tidak siklus negatif. Subbab 15.4 memotong lintasan menurut **banyaknya sisi**; Floyd dan Warshall memotongnya menurut **simpul mana yang boleh dilewati**.

DEFINISI. Simpul antara sebuah lintasan sederhana $\langle v_1, \ldots, v_l \rangle$ adalah setiap simpul selain v_1 dan v_l.

Misalkan p lintasan terpendek dari i ke j yang semua simpul antaranya terletak di $\{1, 2, \ldots, k\}$. Hubungannya bergantung pada apakah k simpul antara bagi p:

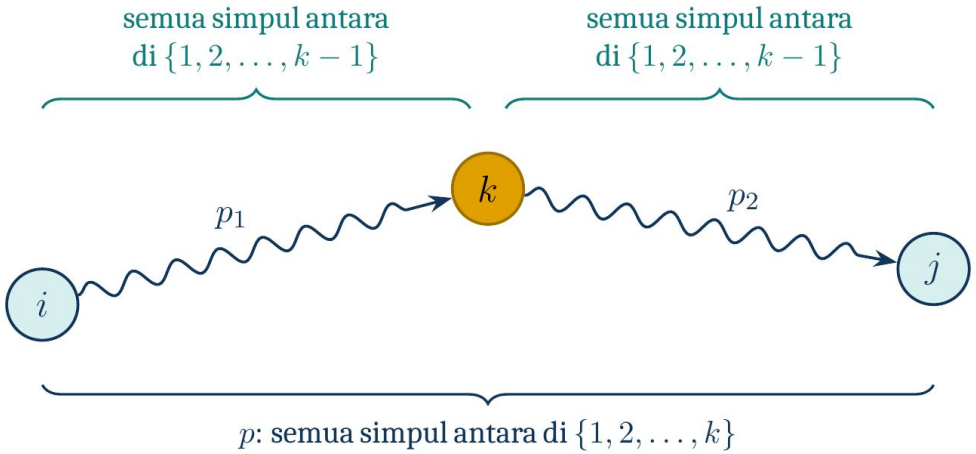
- k **tidak dilewati** p. Maka setiap simpul antara p ada di $\{1, \ldots, k-1\}$, dan p juga lintasan terpendek dari i ke j yang semua simpul antaranya di $\{1, \ldots, k-1\}$.
- k **dilewati** p. Pecah p menjadi $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$; semua simpul antara p_1 maupun p_2 ada di $\{1, \ldots, k-1\}$.

Gambar 15.13 Struktur sebuah lintasan terpendek yang simpul antaranya terletak di $\{1, \ldots, k\}$, dipecah menurut apakah k sendiri terpakai.

Gambar 15.13: misalkan p lintasan terpendek dari i ke j yang semua simpul antaranya di $\{1, \ldots, k\}$. **Kasus 1:** k bukan simpul antara, jadi p sudah berada di $\{1, \ldots, k-1\}$. **Kasus 2:** k simpul antara, jadi $p = i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$.

Buku: Subbab 15.6.1 dan 15.6.2.

Mengapa Kedua Paruhnya Hanya Memakai $\{1,\ldots,k-1\}$



Gambar 15.14 Lintasan p adalah lintasan terpendek dari i ke j dan k adalah simpul antaranya yang bernomor tertinggi. Kedua paruhnya, p_1 dan p_2, lalu punya seluruh simpul antaranya di $\{1,2,\ldots,k-1\}$.

Pada Gambar 15.14, p_1 terpendek menurut Teorema 13.1. Dan k **bukan simpul antara pada** p_1: ia adalah ujung p_1, dan ia tidak dapat muncul lagi di dalamnya karena p sederhana. Hal yang sama berlaku bagi p_2.

> Potonglah rute termurah lewat tujuh simpul pertama di simpul 7: setiap paruhnya termurah, dan tidak satu pun *melewati* simpul 7. **Dua persoalan yang lebih kecil dengan bentuk yang sama**, itulah substruktur optimal.

Buku: Subbab 15.6.3.

Rekurensi Floyd–Warshall

Misalkan $d_{ij}^{(k)}$ bobot lintasan terpendek dari i ke j yang semua simpul antaranya ada di $\{1, 2, \ldots, k\}$.

Untuk $k = 0$ lintasannya sama sekali tidak memiliki simpul antara, sehingga memuat paling banyak satu sisi dan $d_{ij}^{(0)} = w_{ij}$. Secara rekursif,

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{jika } k = 0 \\ \min\left(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{jika } k \geq 1 \end{cases}$$

Setiap lintasan memiliki semua simpul antaranya di $\{1, \ldots, n\}$, sehingga jawabannya $D^{(n)} = \left(d_{ij}^{(n)}\right)$: $d_{ij}^{(n)} = \delta(i, j)$ untuk semua $i, j \in V$.

Gambar 15.15 Definisi $d_{ij}^{(k)}$, kasus dasarnya $d_{ij}^{(0)} = w_{ij}$, dan rekurensi pada Persamaan (15.7). Matriks terakhirnya $D^{(n)} = \left(d_{ij}^{(n)}\right)$ adalah jawabannya.

Dengan $d_{ij}^{(k)}$ bobot terkecil lintasan dari i ke j yang simpul antaranya di $\{1, \ldots, k\}$, seperti pada Gambar 15.15:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{bila } k = 0, \\ \min\left(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{bila } k \geq 1. \end{cases} \tag{15.7}$$

Simpul antara lintasan mana pun terletak di $\{1, \ldots, n\}$, sehingga

$$d_{ij}^{(n)} = \delta(i, j) \text{ untuk semua } i, j \in V. \tag{15.8}$$

> **Kedua superskrip pada suku kedua** Persamaan (15.7) **adalah** $(k-1)$**.** Menulis (k) akan melingkar. Dan Persamaan (15.7) mengambil minimum atas **dua** calon, n kali, dibandingkan n calon pada Persamaan (15.5): itulah seluruh perbedaan kecepatannya.

Persamaan (15.8) memenuhi sasaran Persamaan (15.2). Buku: Subbab 15.6.4.

Algoritmanya

FLOYD-WARSHALL(W)

1 $n = W.rows$
2 $D^{(0)} = W$
3 **untuk** $k = 1$ **sampai** n
4 **misalkan** $D^{(k)} = \left(d_{ij}^{(k)}\right)$ sebuah matriks $n \times n$ baru
5 **untuk** $i = 1$ **sampai** n
6 **untuk** $j = 1$ **sampai** n
7 $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
8 **kembalikan** $D^{(n)}$

Gambar 15.16 FLOYD-WARSHALL(W): lima kata keadaan dan tiga gelung bersarang. Baris 7 adalah rekurensi pada Persamaan (15.7) yang dituliskan lengkap.

Gambar 15.16: baris 2 menyetel $D^{(0)}=W$; baris 3 menjalankan $k=1..n$, memperluas himpunan yang diizinkan; baris 5–6 mengunjungi setiap pasangan; baris 7 menerapkan Persamaan (15.7); baris 8 mengembalikan $D^{(n)}$.

Waktu jalan. Baris 7 berongkos $O(1)$ dan dijalankan n^3 kali: $\Theta(n^3)$, dengan konstanta mungil dan akses memori yang bersinambung sepanjang baris i dan k.

Nilai $d_{ij}^{(k)}$ dihitung menurut k yang menaik, dan urutan itu adalah syarat kebenaran. Buku: Subbab 15.6.5 dan 15.6.12.

Mengapa k Harus Menjadi Gelung Terluar

Setiap nilai yang dibaca pada putaran k harus bersuperskrip $(k-1)$. Dengan k terluar hal itu otomatis. Dengan i terluar dan k terdalam, sel (i, j) dapat membaca d_{kj} dari baris yang belum dikunjungi. Program pada Subbab 15.6.6 memperlihatkan kedua ragam kegagalannya:

```
(a) the wrong order loses the path entirely:
  k outermost (correct):
        0     3     2     1
      inf     0   inf   inf
      inf     1     0   inf
      inf     2     1     0
  i outermost, k innermost (wrong):
        0   inf     2     1
      inf     0   inf   inf
      inf     1     0   inf
      inf     2     1     0
  the two agree: NO
  delta(1,2): correct = 3, wrong = inf (no path found at all)
  passes of the wrong order needed before it matches: 2

(b) the wrong order finds a path, but overcharges:
  k outermost (correct):
        0    10     9     6
      inf     0   inf   inf
      inf     1     0   inf
      inf     4     3     0
  i outermost, k innermost (wrong):
        0    12     9     6
      inf     0   inf   inf
      inf     1     0   inf
      inf     4     3     0
  the two agree: NO
  delta(1,2): correct = 10, wrong = 12
```

```
  passes of the wrong order needed before it matches: 2
```

Graf (a), lintasan $1 \to 4 \to 3 \to 2$: urutan yang salah melaporkan **tidak ada lintasan** dari 1 ke 2, padahal ada yang berbobot 3. Graf (b): ia melaporkan **12** alih-alih **10**, bilangan salah yang masuk akal dan lebih sulit disadari.

> **Galat Floyd–Warshall yang paling umum:** `for (i) for (j) for (k)` alih-alih `for (k) for (i) for (j)`. Ia terkompilasi, berjalan, dan salah. k **di depan.**

Mengulang urutan yang salah memang konvergen di sini sesudah dua sapuan, tetapi banyaknya sapuan bergantung pada penomoran, sehingga jaminan $\Theta(n^3)$ hilang. Buku: Subbab 15.6.6.

Matriks Pendahulunya

$\pi_{ij}^{(k)}$ adalah pendahulu j pada lintasan terpendek dari i dengan simpul antara di $\{1,\ldots,k\}$. **Kasus dasar:** hanya sisi langsung,

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{bila } i = j \text{ atau } w_{ij} = \infty, \\ i & \text{bila } i \neq j \text{ dan } w_{ij} < \infty. \end{cases} \tag{15.9}$$

Kasus rekursif, mulai dari Persamaan (15.9): tidak berubah bila kita tidak lewat k; selainnya pendahulu j pada rute terbaik **dari** k:

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{bila } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{bila } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases} \tag{15.10}$$

Tebakan alami $\pi_{ij}^{(k)}=k$ keliru. Simpul k berada di *tengah* rute. Dalam kode, Persamaan (15.10) adalah `P[i][j] = P[k][j];`, bukan `P[i][j] = k;`.

Buku: Subbab 15.6.7.

Penelusuran Lengkapnya

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Gambar 15.17 Barisan lengkap matriks $D^{(k)}$ dan $\Pi^{(k)}$ yang dihitung Floyd–Warshall untuk graf pada Gambar 15.2. Baris $k=0$ adalah masukannya; baris $k=5$ adalah jawabannya.

Gambar 15.17 memperlihatkan enam matriks jarak dan enam matriks pendahulu. $D^{(5)}$ sama dengan $L^{(4)}$ dari metode matriks: dua algoritma yang berbeda, satu jawaban.

Buku: Subbab 15.6.8.

Floyd–Warshall dalam C++

Kode 15.2 Floyd–Warshall beserta matriks pendahulunya, mencetak setiap $D^{(k)}$ dan $\Pi^{(k)}$ lalu setiap rutenya. Kompilasilah dengan `g++ -std=c++17 -O2 -o floyd_warshall floyd_warshall.cpp`.

```cpp
// Bab 15, Kode: algoritma Floyd-Warshall, lengkap dengan matriks pendahulu
// dan rekonstruksi lintasan, ditelusuri pada graf contoh berjalan.
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
using namespace std;

const long long INF = 1000000000LL;
const int NIL = -1;

typedef vector<vector<long long>> Matrix;
typedef vector<vector<int>>       PiMatrix;

void printD(const string& name, const Matrix& D) {
    int n = (int)D.size();
    cout << name << " =\n";
    for (int i = 0; i < n; ++i) {
        cout << "   ";
        for (int j = 0; j < n; ++j) {
            if (D[i][j] >= INF) cout << setw(6) << "inf";
            else                cout << setw(6) << D[i][j];
        }
        cout << '\n';
    }
}

void printPi(const string& name, const PiMatrix& P) {
```

```
    int n = (int)P.size();
    cout << name << " =\n";
    for (int i = 0; i < n; ++i) {
        cout << "   ";
        for (int j = 0; j < n; ++j) {
            if (P[i][j] == NIL) cout << setw(6) << "NIL";
            else                cout << setw(6) << P[i][j] + 1;   // berbasis 1 untuk pencetakan
        }
        cout << '\n';
    }
}
```

Kode 15.2 (Lanjutan) *Floyd–Warshall beserta matriks pendahulunya, mencetak setiap D^{(k)} dan \Pi^{(k)} lalu setiap rutenya. Kompilasilah dengan g++ -std=c++17 -O2 -o floyd_warshall floyd_warshall.cpp.*

```
// PRINT-ALL-PAIRS-SHORTEST-PATH(Pi, i, j), persis seperti gambar di Subbab 15.2.3.
void printAllPairsShortestPath(const PiMatrix& P, int i, int j) {
    if (i == j) {
        cout << i + 1;
    } else if (P[i][j] == NIL) {
        cout << "no path from " << i + 1 << " to " << j + 1 << " exists";
    } else {
        printAllPairsShortestPath(P, i, P[i][j]);
        cout << " -> " << j + 1;
    }
}

int main() {
    const int n = 5;
    Matrix W = {
        {  0,   3,   8, INF,  -4},
        {INF,   0, INF,   1,   7},
        {INF,   4,   0, INF, INF},
        {  2, INF,  -5,   0, INF},
        {INF, INF, INF,   6,   0}
    };

    // D^(0) = W, dan Pi^(0) dari kasus dasar Pi di Subbab 15.6.7.
    Matrix   D = W;
    PiMatrix P(n, vector<int>(n, NIL));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (i != j && W[i][j] < INF) P[i][j] = i;
```

```
    printD("D^(0)", D);   printPi("Pi^(0)", P);   cout << '\n';

    // FLOYD-WARSHALL(W), dikerjakan di tempat: baris 7 pseudokodenya.
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                if (D[i][k] < INF && D[k][j] < INF && D[i][k] + D[k][j] < D[i][j]) {
                    D[i][j] = D[i][k] + D[k][j];
                    P[i][j] = P[k][j];            // pendahulu j pada lintasan yang menembus k
                }
        printD("D^(" + to_string(k + 1) + ")", D);
        printPi("Pi^(" + to_string(k + 1) + ")", P);
        cout << '\n';
    }
```

Kode 15.2 (Lanjutan) *Floyd–Warshall beserta matriks pendahulunya, mencetak setiap D^{(k)} dan \Pi^{(k)} lalu setiap rutenya. Kompilasilah dengan g++ -std=c++17 -O2 -o floyd_warshall floyd_warshall.cpp.*

```
    // Siklus berbobot negatif akan tampak sebagai entri negatif pada diagonalnya.
    bool negativeCycle = false;
    for (int i = 0; i < n; ++i) if (D[i][i] < 0) negativeCycle = true;
    cout << "negative-weight cycle detected? " << (negativeCycle ? "yes" : "no") << "\n\n";

    cout << "Shortest paths read out of Pi^(5):\n";
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            cout << "  " << i + 1 << " to " << j + 1 << " (weight " << setw(3) << D[i][j] << "): ";
            printAllPairsShortestPath(P, i, j);
            cout << '\n';
        }
    return 0;
}
```

Kode 15.2 memelihara satu matriks `D` dan menimpanya di tempat, yang dibuktikan aman di Subbab 15.6.13. Buku: Subbab 15.6.8.

Apa yang Dicetaknya

```
D^(0) =
       0     3     8   inf    -4
     inf     0   inf     1     7
     inf     4     0   inf   inf
       2   inf    -5     0   inf
     inf   inf   inf     6     0
Pi^(0) =
     NIL     1     1   NIL     1
     NIL   NIL   NIL     2     2
     NIL     3   NIL   NIL   NIL
       4   NIL     4   NIL   NIL
     NIL   NIL   NIL     5   NIL

D^(1) =
       0     3     8   inf    -4
     inf     0   inf     1     7
     inf     4     0   inf   inf
       2     5    -5     0    -2
     inf   inf   inf     6     0
Pi^(1) =
     NIL     1     1   NIL     1
     NIL   NIL   NIL     2     2
     NIL     3   NIL   NIL   NIL
       4     1     4   NIL     1
     NIL   NIL   NIL     5   NIL

D^(2) =
       0     3     8     4    -4
```

```
      inf     0   inf     1     7
      inf     4     0     5    11
        2     5    -5     0    -2
      inf   inf   inf     6     0
Pi^(2) =
      NIL     1     1     2     1
      NIL   NIL   NIL     2     2
      NIL     3   NIL     2     2
        4     1     4   NIL     1
      NIL   NIL   NIL     5   NIL

D^(3) =
        0     3     8     4    -4
      inf     0   inf     1     7
      inf     4     0     5    11
        2    -1    -5     0    -2
      inf   inf   inf     6     0
Pi^(3) =
      NIL     1     1     2     1
      NIL   NIL   NIL     2     2
      NIL     3   NIL     2     2
        4     3     4   NIL     1
      NIL   NIL   NIL     5   NIL
```

```
D^(4) =
        0     3    -1     4    -4
        3     0    -4     1    -1
        7     4     0     5     3
        2    -1    -5     0    -2
        8     5     1     6     0
Pi^(4) =
      NIL     1     4     2     1
        4   NIL     4     2     1
        4     3   NIL     2     1
        4     3     4   NIL     1
        4     3     4     5   NIL

D^(5) =
        0     1    -3     2    -4
        3     0    -4     1    -1
        7     4     0     5     3
        2    -1    -5     0    -2
        8     5     1     6     0
Pi^(5) =
      NIL     3     4     5     1
        4   NIL     4     2     1
        4     3   NIL     2     1
        4     3     4   NIL     1
        4     3     4     5   NIL

negative-weight cycle detected? no
```

```
Shortest paths read out of Pi^(5):
  1 to 2 (weight   1): 1 -> 5 -> 4 -> 3 -> 2
  1 to 3 (weight  -3): 1 -> 5 -> 4 -> 3
  1 to 4 (weight   2): 1 -> 5 -> 4
  1 to 5 (weight  -4): 1 -> 5
  2 to 1 (weight   3): 2 -> 4 -> 1
  2 to 3 (weight  -4): 2 -> 4 -> 3
  2 to 4 (weight   1): 2 -> 4
  2 to 5 (weight  -1): 2 -> 4 -> 1 -> 5
  3 to 1 (weight   7): 3 -> 2 -> 4 -> 1
  3 to 2 (weight   4): 3 -> 2
  3 to 4 (weight   5): 3 -> 2 -> 4
  3 to 5 (weight   3): 3 -> 2 -> 4 -> 1 -> 5
  4 to 1 (weight   2): 4 -> 1
  4 to 2 (weight  -1): 4 -> 3 -> 2
  4 to 3 (weight  -5): 4 -> 3
  4 to 5 (weight  -2): 4 -> 1 -> 5
  5 to 1 (weight   8): 5 -> 4 -> 1
  5 to 2 (weight   5): 5 -> 4 -> 3 -> 2
  5 to 3 (weight   1): 5 -> 4 -> 3
  5 to 4 (weight   6): 5 -> 4
```

Kedua belas matriks cocok dengan Gambar 15.17, dan rute-rutenya dibaca dari $\Pi^{(5)}$ oleh PRINT-ALL-PAIRS-SHORTEST-PATH.

Buku: Subbab 15.6.8 dan 15.6.11.

Putaran $k=1$, Sel demi Sel

Calon $d_{i1}^{(0)}+d_{1j}^{(0)}$ melawan petahana $d_{ij}^{(0)}$. **Baris 1:** calonnya $0+d_{1j}$, seri; **baris k dan kolom k tidak pernah berubah pada putaran k.**

Baris 2, 3, 5: $d_{21}=d_{31}=d_{51}=\infty$, sehingga setiap calon bernilai ∞: **bila $d_{ik}=\infty$, baris i tidak dapat berubah pada putaran k.**

Baris 4 ($d_{41}=2$): $j=2$: $\min(\infty, 2+3)=5$, dan $\pi_{42}^{(1)}=\pi_{12}^{(0)}=1$. $j=3$: $\min(-5, 10)=-5$. $j=5$: $\min(\infty, 2+(-4))=-2$, dan $\pi_{45}^{(1)}=1$.

Dua sel berubah dalam seluruh putaran, keduanya di baris 4: persis $D^{(1)}$ yang tercetak.

Buku: Subbab 15.6.9.

Putaran $k=4$, Tempat Kerja Terjadi

Dari $D^{(3)}$ dengan baris $(0,3,8,4,-4)$, $(\infty,0,\infty,1,7)$, $(\infty,4,0,5,11)$, $(2,-1,-5,0,-2)$, $(\infty,\infty,\infty,6,0)$.

Baris 1 ($d_{14}=4$): $d_{13}=\min(8,4-5)=-1$, $\pi_{13}=4$; d_{12} seri pada 3. **Baris 2** ($d_{24}=1$): $d_{21}=3$, $d_{23}=-4$, $d_{25}=-1$ dengan $\pi_{25}=\pi_{45}=1$.

Baris 3 ($d_{34}=5$): $d_{31}=7$, $d_{35}=3$; d_{33} seri pada 0, jadi siklus $3\to2\to4\to3$ berbobot tepat 0. **Baris 4** adalah baris k: tidak berubah. **Baris 5** ($d_{54}=6$): $d_{51}=8$, $d_{52}=5$ dengan $\pi_{52}=\pi_{42}=3$, $d_{53}=1$.

Sembilan sel berubah; sesudahnya setiap entri berhingga. Simpul 4 adalah poros yang menyambungkan bagian-bagiannya.

Buku: Subbab 15.6.10.

Membaca Rute, dan Bekerja di Tempat

Rute 1 ke 2. $\pi_{12}=3$, $\pi_{13}=4$, $\pi_{14}=5$, $\pi_{15}=1$: penguraiannya mencetak `1 5 4 3 2`, dan $-4+6-5+4=1=d_{12}^{(5)}$.

> **TEOREMA 15.1 (di tempat itu aman).** Menjalankan gelung rangkap tiga pada satu matriks, menimpa d_{ij}, menghasilkan $D^{(n)}$ yang sama dengan memelihara $n+1$ matriks.

Bukti. Hanya baris k dan kolom k yang dibaca pada putaran k dan mungkin sudah berubah. Calon bagi d_{kj} adalah $d_{kk}+d_{kj} \geq d_{kj}$, sebab $d_{kk} \geq 0$ bila tidak ada siklus negatif; begitu pula kolom k. Jadi keduanya tidak berubah, dan setiap nilai yang dibaca masih bernilai putaran $(k-1)$. ∎

Memori turun dari $\Theta(n^3)$ ke $\Theta(n^2)$: pada $n=2000$, 32 megabita alih-alih 64 gigabita.

Buku: Subbab 15.6.11 dan 15.6.13.

Siklus Negatif, Terdeteksi Tanpa Biaya

UJINYA. Sesudah gelung rangkap tiga, graf memuat siklus berbobot negatif **jika dan hanya jika** $d_{ii} < 0$ untuk suatu i.

d_{ii} adalah jalan tertutup termurah dari i. Tanpa siklus negatif nilainya 0; jalan tertutup negatif memuat siklus sederhana negatif; dan begitu k mencapai simpul tertinggi siklus semacam itu, entri diagonalnya menjadi negatif. Program pada Subbab 15.6.14:

```
(a) the running example (edges of weight -4 and -5, but no negative cycle):
   diagonal after the triple loop:    0   0   0   0   0
   negative-weight cycle? no

(b) a graph whose cycle 1 -> 2 -> 3 -> 1 has weight 2 + 3 + (-6) = -1:
   diagonal after the triple loop:   -1  -1  -2
   negative-weight cycle? YES

(c) a graph whose cycle 1 -> 2 -> 1 has weight 1 + (-4) = -3:
   diagonal after the triple loop:   -3  -6
   negative-weight cycle? YES
```

(a) adalah contoh berjalan: semuanya nol, bukti yang diperiksa mesin bahwa hasilnya bermakna. Pada (b) dan (c) hanya **uji tandanya** yang dapat dipercaya; bilangan di luar diagonal hanyalah artefak.

Pemeriksaan silang $|V|\times$ Bellman–Ford, SLOW-APSP, pemangkatan berulang, Floyd–Warshall, dan Johnson pada contoh berjalan memberi satu jawaban. Buku: Subbab 15.6.14 dan 15.6.15.

Penutupan Transitif

DEFINISI. Penutupan transitif G adalah $G^*=(V, E^*)$ dengan $E^*=\{(i,j): \text{ada lintasan dari } i \text{ ke } j \text{ di } G\}$.

Jawaban yang malas: beri setiap sisi bobot 1 lalu jalankan Floyd–Warshall; $d_{ij}<n$ tepat bila ada lintasan. **Jawaban yang lebih baik:** ganti min dengan ∨ dan $+$ dengan ∧.

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{jika } i \neq j \text{ dan } (i,j) \notin E \\ 1 & \text{jika } i = j \text{ atau } (i,j) \in E \end{cases}$$

dan untuk $k \geq 1$

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right).$$

Gambar 15.18 Kasus dasar $t_{ij}^{(0)}$ dan rekurensi bagi $t_{ij}^{(k)}$, yang merupakan rekurensi Floyd–Warshall dengan ∨ menggantikan min dan ∧ menggantikan $+$.

Dituliskan lengkap, Gambar 15.18 adalah

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{bila } i \neq j \text{ dan } (i,j) \notin E, \\ 1 & \text{bila } i = j \text{ atau } (i,j) \in E, \end{cases} \tag{15.11}$$

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right). \tag{15.12}$$

Bacalah Persamaan (15.12) keras-keras: sudah terjangkau, atau terjangkau ke k dan dari k. Pada Persamaan (15.11), $t_{ii}^{(0)}=1$ adalah $w_{ii}=0$ versi boolean. Buku: Subbab 15.7.1 dan 15.7.2.

TRANSITIVE-CLOSURE

TRANSITIVE-CLOSURE(G)

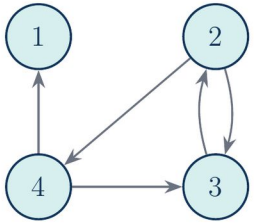
1 $n = |G.V|$
2 **misalkan** $T^{(0)} = \left(t_{ij}^{(0)}\right)$ sebuah matriks $n \times n$ baru
3 **untuk** $i = 1$ **sampai** n
4 **untuk** $j = 1$ **sampai** n
5 **jika** $i == j$ **atau** $(i, j) \in G.E$
6 $t_{ij}^{(0)} = 1$
7 **selain itu** $t_{ij}^{(0)} = 0$
8 **untuk** $k = 1$ **sampai** n
9 **misalkan** $T^{(k)} = \left(t_{ij}^{(k)}\right)$ sebuah matriks $n \times n$ baru
10 **untuk** $i = 1$ **sampai** n
11 **untuk** $j = 1$ **sampai** n
12 $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
13 **kembalikan** $T^{(n)}$

Gambar 15.19 TRANSITIVE-CLOSURE(G). Baris 3–7 membangun $T^{(0)}$ dari himpunan sisinya; baris 8–12 adalah gelung rangkap tiga Floyd–Warshall dengan operator boolean. Ia berjalan dalam waktu $\Theta(n^3)$.

Gambar 15.19 adalah semiring boolean $(\vee, \wedge)$ dengan $\acute{0}$ = FALSE dan $\acute{1}$ = TRUE: *adakah salah satu rute yang tersedia?* dan *dapatkah saya melakukan ini lalu itu?*

Buku: Subbab 15.7.3.

Penutupannya, Dikerjakan



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Gambar 15.20 Sebuah graf berarah dengan sisi $2 \to 3, 2 \to 4, 3 \to 2, 4 \to 1, 4 \to 3$, beserta matriks $T^{(k)}$ yang dihitung algoritma penutupan transitifnya.

Pada Gambar 15.20: putaran $k=1$ tidak mengubah apa pun (simpul 1 tidak punya sisi keluar); putaran 2 menyetel t_{34} lewat $3 \to 2 \to 4$; putaran 3 menyetel t_{42} lewat $4 \to 3 \to 2$; putaran 4 mengisi baris 2 dan 3, karena keduanya mencapai 4 dan 4 mencapai 1.

Baris 1 tetap $(1,0,0,0)$ selamanya: simpul 1 adalah sebuah muara.

Buku: Subbab 15.7.3.

Penutupan Transitif dalam C++

Kode 15.3 Algoritma penutupan transitif selengkapnya. Kompilasilah dengan `g++ -std=c++17 -O2 -o transitive_closure transitive_closure.cpp`.

```cpp
// Bab 15, Kode: TRANSITIVE-CLOSURE pada graf contoh bersimpul empat.
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
using namespace std;

typedef vector<vector<int>> BoolMatrix;   // entri 0/1, sehingga | dan & dapat dipakai langsung

void printT(const string& name, const BoolMatrix& T) {
    int n = (int)T.size();
    cout << name << " =\n";
    for (int i = 0; i < n; ++i) {
        cout << "   ";
        for (int j = 0; j < n; ++j) cout << setw(3) << T[i][j];
        cout << '\n';
    }
    cout << '\n';
}

int main() {
    const int n = 4;
    // Graf contoh: sisi 2->3, 2->4, 3->2, 4->1, 4->3 (simpul bernomor 1..4).
    vector<pair<int,int>> edges = {{2,3},{2,4},{3,2},{4,1},{4,3}};

    // Baris 2-7: bangunlah T^(0).
    BoolMatrix T(n, vector<int>(n, 0));
    for (int i = 0; i < n; ++i) T[i][i] = 1;              // i == j
```

```
    for (auto& e : edges) T[e.first - 1][e.second - 1] = 1; // (i,j) di dalam G.E
    printT("T^(0)", T);

    // Baris 8-12: gelung rangkap tiga yang sama seperti Floyd-Warshall, dengan OR untuk min dan AND
untuk +.
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                T[i][j] = T[i][j] | (T[i][k] & T[k][j]);
        printT("T^(" + to_string(k + 1) + ")", T);
    }

    cout << "Edges of the transitive closure G* (excluding self-loops):\n   ";
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (i != j && T[i][j]) cout << "(" << i + 1 << "," << j + 1 << ") ";
    cout << '\n';
    return 0;
}
```

```
T^(0) =
     1  0  0  0
     0  1  1  1
     0  1  1  0
     1  0  1  1

T^(1) =
     1  0  0  0
     0  1  1  1
     0  1  1  0
     1  0  1  1

T^(2) =
     1  0  0  0
     0  1  1  1
     0  1  1  1
     1  0  1  1

T^(3) =
     1  0  0  0
     0  1  1  1
     0  1  1  1
     1  1  1  1

T^(4) =
     1  0  0  0
     1  1  1  1
     1  1  1  1
```

```
     1  1  1  1

Edges of the transitive closure G* (excluding self-loops):
   (2,1) (2,3) (2,4) (3,1) (3,2) (3,4) (4,1) (4,2) (4,3)
```

Kode 15.3 mereproduksi kelima matriks pada Gambar 15.20.

Buku: Subbab 15.7.3.

Mengapa Boolean Lebih Cepat dalam Praktik

Ruang. Sebuah jarak memerlukan satu kata 32 atau 64 bit; sebuah boolean memerlukan satu bit, faktor 32 atau 64, yang menentukan apakah matriksnya muat di *cache*.

Waktu. Dimampatkan ke `std::bitset`, gelung j terdalam menjadi OR dua baris utuh: 64 putaran dalam satu instruksi. Diukur oleh program pada Subbab 15.7.4:

```
n = 1024, edges = 4096
byte-per-entry version: 231.7 ms
bitset version:         3.2 ms
the two closures agree: yes
reachable pairs found:  1015046 out of 1048576
```

Faktor lebih dari **tujuh puluh** untuk perubahan dua baris. Asimtotik memilih algoritmanya; ia tidak mengatakan apa-apa tentang konstantanya, dan keduanya penting.

Pengukuran waktunya berasal dari satu kali jalan pada satu mesin; rasionya yang terbawa. Buku: Subbab 15.7.4.

Johnson: Keinginan dan Rencananya

Keinginannya: $O(V^2 \lg V + VE)$ pada graf jarang, lebih cepat daripada Floyd–Warshall atau pemangkatan berulang, dengan sisi negatif diizinkan. Batas itu adalah V kali Dijkstra dengan *Fibonacci heap*: *bagaimana membuat $V\times$ Dijkstra sah dengan sisi negatif?*

- Bila setiap bobot sisi tidak negatif, menjalankan Dijkstra sekali dari tiap simpul menyelesaikan semua pasangan; dengan *Fibonacci heap* waktunya $O(V^2 \lg V + VE)$.
- Bila G memiliki sisi berbobot negatif tetapi tanpa siklus berbobot negatif, hitunglah bobot baru $\widehat{w}$ yang tidak negatif agar metode yang sama dapat dipakai.
- Bobot barunya harus memenuhi dua sifat:

1. Untuk semua $u, v \in V$, sebuah lintasan p adalah lintasan terpendek dari u ke v menurut w **jika dan hanya jika** ia lintasan terpendek dari u ke v menurut $\widehat{w}$.
2. Untuk setiap sisi (u, v), bobot baru $\widehat{w}(u, v)$ tidak negatif.

Gambar 15.21 Bila sebuah fungsi bobot baru $\widehat{w}$ melestarikan lintasan terpendek dan tak negatif di mana-mana, maka $V\times$ Dijkstra atas $\widehat{w}$ menjawab pertanyaan aslinya.

Gambar 15.21: **(1)** p terpendek terhadap w jika dan hanya jika terpendek terhadap $\widehat{w}$; **(2)** $\widehat{w}(u,v) \geq 0$ untuk setiap sisi.

Menambahkan konstanta gagal. Dengan $w(1,2)=-2$, $w(2,3)=-2$, $w(1,3)=-3$ rute terbaiknya $1 \to 2 \to 3$ seharga -4. Menambahkan $c=3$ memberi 1, 1, 0: kini sisi langsung yang menang. Sebuah lintasan bertambah c per sisi; perbaikannya harus bergantung pada **ujung-ujungnya**.

Buku: Subbab 15.8.1 dan 15.8.2.

Pembobotan Ulang yang Berhasil

Pilihlah sembarang $h : V \to R$, sebuah *ketinggian* untuk setiap simpul, lalu tetapkan

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v). \tag{15.13}$$

LEMA 15.2. Dengan Persamaan (15.13), untuk sembarang lintasan $p = \langle v_0, \ldots, v_k \rangle$, $\hat{w}(p) = w(p) + h(v_0) - h(v_k)$. Jadi lintasan terpendek terlestarikan, dan siklus negatif tidak muncul maupun hilang.

Bukti. Menjumlahkan Persamaan (15.13) sepanjang p, suku-suku $h(v_{i-1}) - h(v_i)$ **berteleskop** menjadi $h(v_0) - h(v_k)$. Semua lintasan dengan ujung yang sama bertambah konstanta yang sama, sehingga urutannya tidak berubah; siklus mempunyai $v_0 = v_k$ dan bobotnya tetap. ∎

Ini membuktikan sifat 1 untuk setiap h; seninya adalah memilih h yang juga memberi sifat 2. Buku: Subbab 15.8.3.

Memilih h: Simpul Bantu

Sifat 2 adalah $h(v) \le h(u) + w(u,v)$ untuk setiap sisi: **ketaksamaan segitiga**, Akibat 13.3. Jadi ambillah $h(v) = \delta(s,v)$ untuk suatu sumber s yang menjangkau semuanya.

Ciptakan sumber itu. G' menambahkan simpul baru s dan sisi (s,v) berbobot 0 ke setiap v. s **tidak punya sisi masuk**, sehingga ia tidak terletak pada lintasan terpendek mana pun antara simpul asli dan tidak mengubah jarak di antara mereka.

Satu kali Bellman–Ford dari s menghitung $h(v) = \delta(s,v)$, dan sekaligus menjadi uji siklus negatif.

> **Mengapa setiap bobot menjadi tak negatif:** $\delta(s,v) \le \delta(s,u) + w(u,v)$ adalah $h(v) \le h(u) + w(u,v)$, yakni $0 \le \hat{w}(u,v)$. Satu baris, begitu Anda melihat bahwa syaratnya *adalah* ketaksamaan segitiga.

Buku: Subbab 15.8.4.

JOHNSON(G, w)

```
JOHNSON(G, w)
1  bentuk G', dengan G'.V = G.V ∪ {s},
       G'.E = G.E ∪ {(s, v) : v ∈ G.V}, dan w(s, v) = 0 untuk semua v ∈ G.V
2  jika BELLMAN-FORD(G', w, s) == FALSE
3      cetak "graf masukan memuat siklus berbobot negatif"
4  selain itu untuk tiap simpul v ∈ G'.V
5          tetapkan h(v) sebagai nilai δ(s, v) yang dihitung BELLMAN-FORD
6      untuk tiap sisi (u, v) ∈ G'.E
7          ŵ(u, v) = w(u, v) + h(u) − h(v)
8      misalkan D = (d_uv) sebuah matriks n × n baru
9      untuk tiap simpul u ∈ G.V
10         jalankan DIJKSTRA(G, ŵ, u) untuk menghitung δ̂(u, v) bagi semua v ∈ G.V
11         untuk tiap simpul v ∈ G.V
12             d_uv = δ̂(u, v) + h(v) − h(u)
13     kembalikan D
```

Gambar 15.22 JOHNSON(G, w). Baris 1 menambahkan simpul bantunya, baris 2 menjalankan Bellman–Ford sekali, baris 6–7 membobot ulang, baris 9–10 menjalankan Dijkstra $|V|$ kali, dan baris 12 mengubah tiap jawabannya kembali.

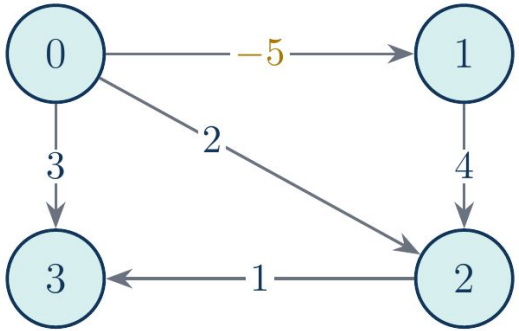
Baris 12 pada Gambar 15.22 **membatalkan pembobotan ulangnya**, menurut Lema 15.2 pada sebuah lintasan terpendek:

$$\hat{\delta}(u,v) = \delta(u,v) + h(u) - h(v), \text{ jadi } d_{uv} = \hat{\delta}(u,v) + h(v) - h(u). \tag{15.14}$$

Ongkos. Bellman–Ford pada G': $O(VE)$. Pembobotan ulang: $O(E)$. V kali Dijkstra dengan *Fibonacci heap*: $O(V^2 \lg V + VE)$, yang dominan. Dengan tumpukan biner: $O(VE \lg V)$, biasanya lebih baik dalam praktik.

Pada Persamaan (15.14) perhatikan tandanya yang terbalik: untuk memperoleh yang asli, kurangkan apa yang tadi ditambahkan. Buku: Subbab 15.8.5.

Pembobotan Ulang Kecil: Grafnya

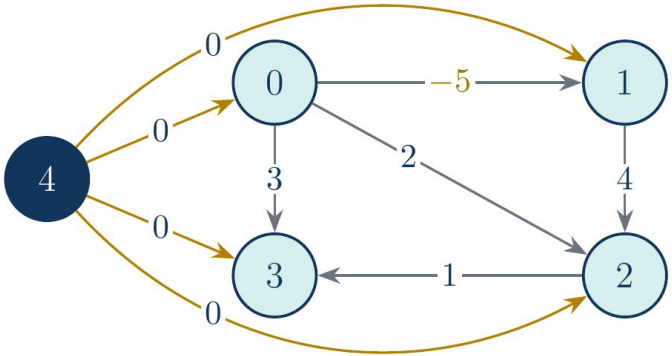


Gambar 15.23 Sebuah graf bersimpul empat dengan satu sisi negatif, $0 \to 1$ berbobot -5. Dijkstra tidak diperbolehkan di sini.

Gambar 15.23 mempunyai simpul $0, 1, 2, 3$ dan sisi $0 \to 1$ (-5), $0 \to 2$ (2), $0 \to 3$ (3), $1 \to 2$ (4), $2 \to 3$ (1).

Buku: Subbab 15.8.6.

Tambahkan s, Jalankan Bellman–Ford

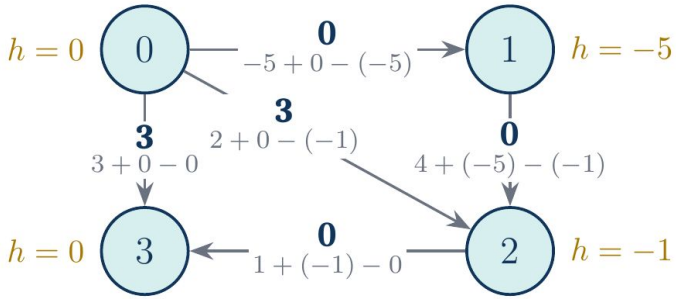


Gambar 15.24 Graf yang sama dengan simpul bantu 4 ditambahkan, tersambung ke tiap simpul asli lewat sebuah sisi berbobot 0. Simpul 4 tidak punya sisi masuk, sehingga ia tidak dapat muncul di dalam lintasan terpendek mana pun antara simpul-simpul asli.

Dari simpul 4 pada Gambar 15.24: $h[0]=0$; $h[1]=0+(-5)=-5$ lewat $4\to0\to1$; $h[2]=-5+4=-1$ lewat $4\to0\to1\to2$; $h[3]=0$. Jadi $h=[0,-5,-1,0]$.

Buku: Subbab 15.8.6.

Bobot Ulang: Setiap Bobot Tak Negatif



Gambar 15.25 Graf bersimpul empat itu sesudah dibobot ulang. Setiap bobotnya kini tak negatif — tiga dari kelimanya persis nol — sehingga algoritma Dijkstra dapat dipakai dari tiap simpul sebagai sumbernya.

Menurut Persamaan (15.13): $(0,1)$: $-5+0+5=0$; $(0,2)$: $2+0+1=3$; $(0,3)$: 3; $(1,2)$: $4-5+1=0$; $(2,3)$: $1-1-0=0$, seperti diperlihatkan Gambar 15.25.

Setiap sisi pada lintasan terpendek dari s menjadi tepat 0. Dan tidak semua bobot mengecil: $(0,2)$ naik dari 2 ke 3. Pembobotan ulang adalah pembagian ulang yang adil, bukan potongan harga.

Sisi-sisi berbobot nol tidak harus membentuk pohon (Soal Latihan 15.17). Buku: Subbab 15.8.6.

Johnson dalam C++

Kode 15.4 Algoritma Johnson selengkapnya: Bellman–Ford dari sebuah simpul bantu, pembobotan ulang, $|V|$ kali jalannya Dijkstra, dan pengubahannya kembali. Kompilasilah dengan `g++ -std=c++17 -O2 -o johnson johnson.cpp`.

```cpp
// Bab 15, Kode: algoritma Johnson untuk graf jarang, dijalankan pada graf contoh
// berjalan. Bellman-Ford memasok h, lalu Dijkstra dijalankan sekali tiap simpul.
#include <iostream>
#include <iomanip>
#include <vector>
#include <queue>
#include <string>
using namespace std;

const long long INF = 1000000000LL;

struct Edge { int u, v; long long w; };

// BELLMAN-FORD pada G', mengembalikan false ketika ada siklus berbobot negatif yang terjangkau.
bool bellmanFord(int n, const vector<Edge>& E, int s, vector<long long>& d) {
    d.assign(n, INF);
    d[s] = 0;
    for (int pass = 1; pass <= n - 1; ++pass)          // |V'| - 1 sapuan
        for (const Edge& e : E)
            if (d[e.u] < INF && d[e.u] + e.w < d[e.v]) d[e.v] = d[e.u] + e.w;
    for (const Edge& e : E)                            // satu sapuan lagi: masih adakah perbaikan?
        if (d[e.u] < INF && d[e.u] + e.w < d[e.v]) return false;
    return true;
}

// DIJKSTRA dengan tumpukan-min biner, hanya untuk bobot tak negatif.
vector<long long> dijkstra(int n, const vector<vector<pair<int,long long>>>& adj, int s) {
    vector<long long> d(n, INF);
```

```
    d[s] = 0;
    priority_queue<pair<long long,int>, vector<pair<long long,int>>, greater<>> Q;
    Q.push({0, s});
    while (!Q.empty()) {
        auto [du, u] = Q.top(); Q.pop();
        if (du > d[u]) continue;                        // entri tumpukan yang sudah basi
        for (auto [v, w] : adj[u])
            if (d[u] + w < d[v]) { d[v] = d[u] + w; Q.push({d[v], v}); }
    }
    return d;
}
```

Kode 15.4 (Lanjutan) *Algoritma Johnson selengkapnya: Bellman–Ford dari sebuah simpul bantu, pembobotan ulang, $|V|$ kali jalannya Dijkstra, dan pengubahannya kembali. Kompilasilah dengan g++ -std=c++17 -O2 -o johnson johnson.cpp.*

```
int main() {
    const int n = 5;                                    // simpul 1..5, disimpan 0..4
    vector<Edge> E = {
        {0,1,3}, {0,2,8}, {0,4,-4},
        {1,3,1}, {1,4,7},
        {2,1,4},
        {3,0,2}, {3,2,-5},
        {4,3,6}
    };

    // Baris 1: G' = G ditambah sumber baru s dengan sisi berbobot 0 ke setiap simpul.
    int s = n;                                          // simpul bantu, indeks 5
    vector<Edge> Eprime = E;
    for (int v = 0; v < n; ++v) Eprime.push_back({s, v, 0});

    // Baris 2-5: Bellman-Ford dari s memberikan h(v) = delta(s, v).
    vector<long long> h;
    if (!bellmanFord(n + 1, Eprime, s, h)) {
        cout << "the input graph contains a negative-weight cycle\n";
        return 0;
    }
    cout << "h(v) = delta(s,v) for v = 1..5:  ";
    for (int v = 0; v < n; ++v) cout << h[v] << ' ';
    cout << "\n\n";

    // Baris 6-7: bobotilah ulang. w-hat(u,v) = w(u,v) + h(u) - h(v).
    vector<vector<pair<int,long long>>> adj(n);
    cout << "reweighted edges:\n";
```

```
    for (const Edge& e : E) {
        long long wh = e.w + h[e.u] - h[e.v];
        adj[e.u].push_back({e.v, wh});
        cout << "   w-hat(" << e.u + 1 << "," << e.v + 1 << ") = "
             << setw(3) << e.w << " + (" << h[e.u] << ") - (" << h[e.v] << ") = " << wh << '\n';
    }
    cout << '\n';

    // Baris 8-12: satu Dijkstra tiap simpul, lalu batalkan pembobotan ulangnya.
    vector<vector<long long>> D(n, vector<long long>(n, INF));
    for (int u = 0; u < n; ++u) {
        vector<long long> dhat = dijkstra(n, adj, u);
        for (int v = 0; v < n; ++v)
            if (dhat[v] < INF) D[u][v] = dhat[v] + h[v] - h[u];
    }
```

Kode 15.4 (Lanjutan) *Algoritma Johnson selengkapnya: Bellman–Ford dari sebuah simpul bantu, pembobotan ulang, $|V|$ kali jalannya Dijkstra, dan pengubahannya kembali. Kompilasilah dengan g++ -std=c++17 -O2 -o johnson johnson.cpp.*

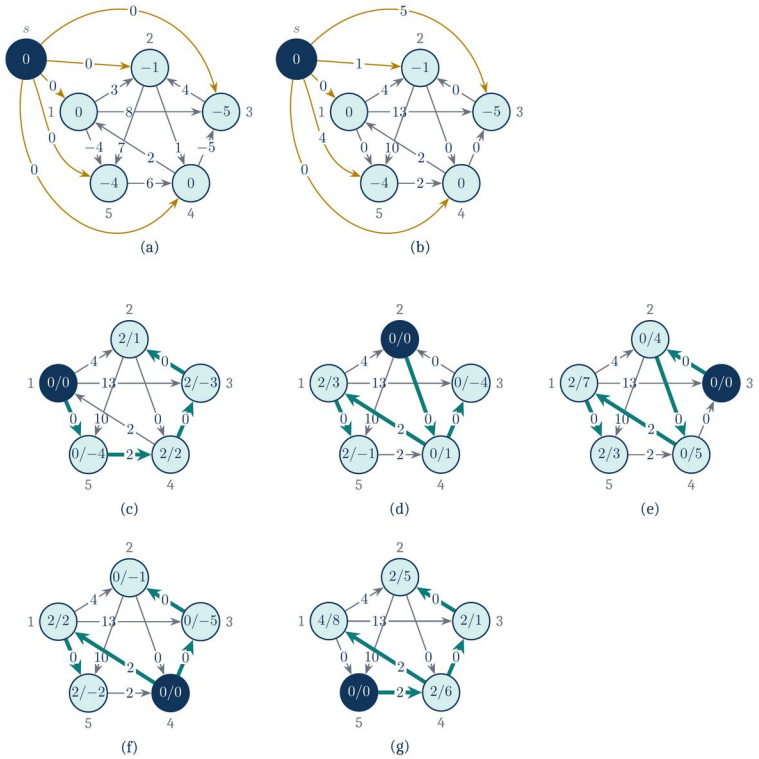
```
    cout << "D = matrix of shortest-path weights:\n";
    for (int i = 0; i < n; ++i) {
        cout << "   ";
        for (int j = 0; j < n; ++j) {
            if (D[i][j] >= INF) cout << setw(6) << "inf";
            else                cout << setw(6) << D[i][j];
        }
        cout << '\n';
    }
    return 0;
}
```

```
h(v) = delta(s,v) for v = 1..5:  0 -1 -5 0 -4

reweighted edges:
   w-hat(1,2) =   3 + (0) - (-1) = 4
   w-hat(1,3) =   8 + (0) - (-5) = 13
   w-hat(1,5) =  -4 + (0) - (-4) = 0
   w-hat(2,4) =   1 + (-1) - (0) = 0
   w-hat(2,5) =   7 + (-1) - (-4) = 10
   w-hat(3,2) =   4 + (-5) - (-1) = 0
   w-hat(4,1) =   2 + (0) - (0) = 2
   w-hat(4,3) =  -5 + (0) - (-5) = 0
   w-hat(5,4) =   6 + (-4) - (0) = 2

D = matrix of shortest-path weights:
        0     1    -3     2    -4
        3     0    -4     1    -1
        7     4     0     5     3
        2    -1    -5     0    -2
        8     5     1     6     0
```

Kode 15.4 pada contoh berjalan: $h = (0, -1, -5, 0, -4)$, sembilan bobot tak negatif dengan empat nol, dan D yang sama dengan Floyd–Warshall dan metode matriks.

Buku: Subbab 15.8.7.

Johnson dalam Satu Gambar



Gambar 15.26 Algoritma Johnson pada graf Gambar 15.2. (a) G' dengan bobot aslinya dan $h(v)=\delta(s,v)$ pada tiap simpul. (b) Sesudah dibobot ulang, setiap bobot tak negatif. (c)–(g) Dijkstra dari tiap simpul bergantian, tiap simpul memperlihatkan $\hat{\delta}(u,v)$ dan $\delta(u,v)$ dipisahkan garis miring.

Periksa bagian (c) pada Gambar 15.26 terhadap Persamaan (15.14): sumber $u=1$, $h(1)=0$; simpul 3 memperlihatkan $2/-3$, dan $\delta(1,3)=2+(-5)-0=-3$, baris 1 kolom 3 dari D.

Jawaban graf terbobot ulang, 2*, tidak menjawab pertanyaan siapa pun; Persamaan (15.14) mengubahnya kembali menjadi jawaban. Buku: Subbab 15.8.7.*

Algoritma yang Mana, dan Kapan

Tabel 15.4 Setiap algoritma di bab ini, beserta ongkosnya dan kasus yang diperuntukkan baginya. Kedua yang dicetak tebal adalah kedua yang sungguh akan Anda pakai.

Algoritma	Menangani sisi negatif?	Waktu jalan	Ruang	Catatan
$V \times$ Dijkstra (larik)	tidak	$O(V^3 + V E)$	$\Theta(V^2)$	pilihan benar paling sederhana pada graf padat tak negatif
$V \times$ Dijkstra (tumpukan biner)	tidak	$O(V E \lg V)$	$\Theta(V + E)$	bagus pada graf jarang tak negatif
$V \times$ Bellman–Ford	ya	$O(V^2 E)$, padat $O(V^4)$	$\Theta(V + E)$	pakailah hanya bila tidak ada pilihan lain
SLOW-APSP ($\odot W$ berulang)	ya	$\Theta(V^4)$	$\Theta(V^2)$	bersifat pengajaran; tak pernah pilihan produksi yang tepat
Pemangkatan berulang	ya	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	gagasan semiringnya, dibuat efisien
Floyd–Warshall	ya	$\Theta(V^3)$	$\Theta(V^2)$	bawaannya; konstanta mungil, sepele dikodekan
Penutupan transitif (boolean)	–	$\Theta(V^3), \Theta(V^3/64)$ termampat	$\Theta(V^2)$ bit	keterjangkauan saja
Johnson	ya	$O(V^2 \lg V + V E)$	$\Theta(V + E)$	bawaannya untuk graf jarang

Dari Tabel 15.4, Johnson mengalahkan Floyd–Warshall secara asimtotik bila $E = o(V^2)$. **Terukur** pada $V = 600$: Johnson menang hingga $|E|/|V| \approx 40$, Floyd–Warshall sejak sekitar 80, titik silang di dekat kerapatan **13 persen**.

Waktu Floyd–Warshall hampir datar terhadap E, sesuai ramalan $\Theta(V^3)$; waktu Johnson tumbuh bersama E. Buku: Subbab 15.9.1 dan 15.9.2.

Sebuah Prosedur Pengambilan Keputusan

Tabel 15.5 Sebuah prosedur pengambilan keputusan bagi persoalan lintasan terpendek semua pasangan. Bekerjalah menurun; "ya" yang pertama adalah jawaban Anda.

Pertanyaan	Bila ya	Bila tidak
Apakah Anda hanya memerlukan keterjangkauan, bukan jarak?	penutupan transitif boolean, dimampatkan ke dalam bit (§15.7)	lanjutkan
Apakah n kecil (katakanlah $n \le 400$), atau apakah kodenya akan dibaca orang?	Floyd–Warshall — lima baris, tanpa struktur data (§15.6)	lanjutkan
Apakah grafnya padat, $E \gtrsim 0.13\,V^2$ menurut pengukuran §15.9.2?	Floyd–Warshall	lanjutkan
Apakah semua bobotnya tak negatif?	$V \times$ Dijkstra dengan tumpukan biner	Johnson: satu Bellman–Ford, bobotilah ulang, lalu $V \times$ Dijkstra (§15.8)
Apakah Anda memerlukan rutenya dan bukan hanya ongkosnya?	peliharalah Π berdampingan dengan D (§15.6.7); ongkosnya ruang $\Theta(n^2)$ dan nol waktu	—
Mungkinkah ada siklus berbobot negatif?	periksalah diagonalnya sesudah Floyd–Warshall (§15.6.14), atau biarlah Bellman–Ford di baris 2 Johnson melaporkannya	—

Telusuri Tabel 15.5 dari atas ke bawah; "ya" yang pertama adalah jawabannya.

Buku: Subbab 15.9.3.

Profitina dalam Praktik

Di dalam platform GEO, situs yang dirayapi adalah graf berarah; **diameter** graf tautan internalnya adalah pertanyaan semua pasangan, dan **laporan halaman yatim** (halaman yang tidak dijangkau baris penutupan mana pun) adalah penutupan transitif. Situs besar ditangani bagian demi bagian.

Profitina Laundry. Delapan wilayah Surabaya dengan beberapa jalan satu arah, sehingga matriksnya tidak simetris. Dua pertanyaan depot: perjalanan **terburuk** yang terkecil (**pusat** graf, eksentrisitas terkecil), dan waktu berkendara pergi-pulang **total** yang terkecil.

Dua jawaban. Klampis: tidak ada pelanggan yang lebih jauh dari **17** menit. Manyar: total berkendara **137** menit, satu kurang daripada Klampis, tetapi perjalanan terburuknya 20. Algoritmanya mengganti perdebatan dengan sebuah tabel; pemiliknya yang memilih.

Keputih mencapai Manyar dalam 9 menit, Manyar memerlukan 12 untuk kembali; Π mengungkap perjalanan terburuk dari Klampis sebagai rute dua ruas Klampis → Keputih → Rungkut.

Platformnya memberi saran; penggunanya yang memutuskan. Buku: Subbab 15.10.

Kesalahan yang Sering Terjadi

1. Urutan gelung yang salah: k di depan. **2. Meratakan superskrip**: $l_{ij}^{(m)}$ menjatah sisi, $d_{ij}^{(k)}$ mengizinkan simpul antara; keduanya algoritma yang berbeda.

3. Menghitung `INF + INF`: kawal penjumlahannya, atau pilih penanda yang cukup kecil sehingga dua kalinya masih muat. **4.** $\pi_{ij} = k$ **alih-alih** π_{kj}: ujilah dengan menjumlahkan bobot sepanjang rute yang dicetak.

5. Memercayai Dijkstra karena graf ujinya berhasil. 6. Lupa membatalkan pembobotan ulang dengan Persamaan (15.14). **7. Menambahkan konstanta pada setiap sisi.**

8. Mengira D **simetris. 9. Semua pasangan padahal cukup satu sumber. 10. Membaca entri di luar diagonal** sebelum memeriksa diagonal atas siklus negatif.

Buku: Subbab 15.11.

Ringkasan — Lima Hal

1. **Persoalannya.** $\delta(i, j)$ untuk setiap pasangan terurut, dari W dengan 0 di diagonal dan ∞ untuk sisi yang tidak ada; keluarannya D dan Π.
2. **Naif.** n kali sumber tunggal: $O(V^3)$ dengan Dijkstra bila $w \geq 0$, $O(V^2 E)$, padat $O(V^4)$, dengan Bellman–Ford.
3. **Perkalian matriks atas** $(\min, +)$. $L^{(m)} = W^m$ menurut jatah sisi: $\Theta(n^4)$ secara lambat, $\Theta(n^3 \lg n)$ dengan pemangkatan berulang.
4. **Floyd–Warshall.** Simpul antara yang diizinkan $\{1, \ldots, k\}$: $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$, $\Theta(n^3)$, k terluar, di tempat, uji diagonal. Versi boolean: penutupan transitif.
5. **Johnson.** $h = \delta(s, \cdot)$ dari sumber bantu, $\hat{w} = w + h(u) - h(v)$, $V\times$ Dijkstra, ubah kembali: $O(V^2 \lg V + V E)$. Floyd–Warshall bila padat atau kecil, Johnson bila jarang.

Buku: Subbab 15.12.

Pertemuan Berikutnya

Bab 16 — Evaluasi dan Pembahasan IV. Soal-soal yang dikerjakan tentang Bab 13 sampai 15 dan tentang seluruh buku.

Setiap algoritma hari ini muncul lagi di sana. Latihlah dua jenis soal: *telusuri Floyd–Warshall pada graf ini, perlihatkan setiap* $D^{(k)}$, dan *mengapa* k *harus menjadi gelung terluar?*

Juga tabel Dijkstra dan Bellman–Ford, pencacah biner dan tabel dinamis dengan tiga cara, serta pembobotan ulang Johnson yang dikerjakan tuntas.

Pekerjaan rumah. Soal Latihan 15.1–15.18; mulailah dengan 15.8 (putaran $k=2$ dengan tangan) dan 15.17 (sisi-sisi berbobot nol sesudah pembobotan ulang).

Cormen dkk., Introduction to Algorithms, edisi ke-4, Bab 23, membahas bahan yang sama. Buku: Subbab 15.12.1 dan 15.13.

Kuliah 16 — Evaluasi dan Pembahasan IV: Lintasan Terpendek, Analisis Amortisasi, dan Keseluruhan Buku

Perancangan dan Analisis Algoritma (EF234405)

Bab 16 buku ini dalam bentuk ringkas, untuk dipakai di kelas.

Buku: Irfan Subakti & Rully Soelaiman, *Perancangan dan Analisis Algoritma*.

Pengajar: Irfan Subakti — yifana@gmail.com

Hari ini tidak ada teori baru: dua puluh satu persoalan terkerjakan yang memantapkan Bab 13 sampai 15 dan memadukan seluruh buku, masing-masing diselesaikan dari nol dan diperiksa dengan program. Buku: Bab 16, paragraf pembuka.

Agenda

1. **Posisi kita saat ini**, seluruh buku pada satu halaman, dan cara menghadapi persoalan berbagian serta bukti yang macet.
2. **Lima persoalan pemadu:** lima belas pernyataan benar-salah (16.1), subbarisan menaik terpanjang (16.2), pohon biner diperluas dengan induksi (16.3), hampiran-2 untuk TSP metrik (16.4), Bellman–Ford berhadapan dengan Dijkstra (16.5).
3. **Tujuh persoalan tentang Bab 13 sampai 15:** Dijkstra, Bellman–Ford dengan siklus negatif, lintasan terpendek DAG (16.6–16.8); pencacah biner dan tabel dinamis dengan tiga cara (16.9–16.10); Floyd–Warshall dan Johnson dengan tangan (16.11–16.12).
4. **Sembilan persoalan lanjutan** (16.13–16.21) yang meninjau kembali bagian lain buku.
5. **Pemantapan:** persoalan-persoalan dibandingkan, latihan uji diri, tabel pemilihan algoritma, Profitina, kesalahan yang sering terjadi, ringkasan dan kata penutup, serta ke mana melangkah selanjutnya.

Setiap persoalan berbentuk sama: persoalan, apa yang diuji, penyelesaian, program, dan varian periksa diri Anda beserta jawabannya. Buku: Subbab 16.1.3.

Posisi Kita Saat Ini

Bab 13: lintasan terpendek sumber tunggal, taksiran $d[v]$ dan pendahulu $\pi[v]$, serta satu-satunya primitif, **relaksasi**:

$$\text{if } d[v] > d[u] + w(u,v) \text{ then } d[v] \leftarrow d[u] + w(u,v), \pi[v] \leftarrow u. \tag{16.1}$$

Dijkstra, Bellman–Ford, dan algoritma DAG hanya berbeda dalam *urutan* penerapan Persamaan (16.1).

Bab 14: analisis amortisasi atas sebuah *barisan* operasi, dengan metode agregat, akuntansi, dan potensial, pada tumpukan dengan `MULTIPOP`, pencacah k-bit, dan tabel yang berlipat dua.

Bab 15: semua pasangan, lewat "perkalian" matriks berulang, rekurensi Floyd–Warshall atas simpul antara yang diizinkan, dan pembobotan ulang Johnson agar Dijkstra menjadi sah.

Ketiga bab mengikuti Cormen dkk., Introduction to Algorithms, edisi ke-4. Buku: Subbab 16.1.1.

Seluruh Buku pada Satu Halaman

Tabel 16.1 Satu gagasan per bab, dan di mana bab ini memakainya. Jika Anda dapat menyebutkan keenam belasnya di luar kepala, seluruh buku ada di tangan Anda.

Bab	Satu gagasannya	Dipakai di sini
1	Sebuah persoalan diselesaikan dalam lima fase: memahami, mengabstraksi, merancang, menganalisis, mengimplementasikan. Menulis kode adalah fase terakhir, bukan yang pertama.	Subbab 16.2
2	Invarian gelung adalah cara Anda mengetahui sebuah gelung benar: inisialisasi, pemeliharaan, terminasi.	Subbab 16.4.4
3	Bandingkan laju pertumbuhan, bukan stopwatch. $T(n)=aT(n/b)+f(n)$ diselesaikan dengan membandingkan $f(n)$ dengan $n^{\log_b a}$.	Persoalan 16.1, pernyataan (11)
4	Kendala adalah anggaran waktu. $n \le 10^5$ berarti $O(n \lg n)$ atau lebih baik; $n \le 500$ berarti $O(n^3)$ sudah memadai.	Subbab 16.4.7
5	Heapsort menukar pemilihan murah dari *heap* dengan $\Theta(n \lg n)$ yang selalu berlaku; quicksort menukar partisi murah dengan $\Theta(n \lg n)$ yang biasanya berlaku. $\Omega(n \lg n)$ adalah lantai pengurutan berbasis perbandingan.	Persoalan 16.1, pernyataan (1) dan (5)
6	Invarian keseimbangan mengubah $O(n)$ menjadi $O(\lg n)$, dan rotasi adalah satu-satunya alat yang dibutuhkan untuk memeliharanya.	Persoalan 16.3
7	Ketika subpersoalan bertumpang tindih dan optimumnya dibangun dari potongan-potongan optimal, ingatlah alih-alih menghitung ulang.	Persoalan 16.2 dan 16.19
8	Jawabannya adalah penurunannya. Kode tanpa invarian adalah tebakan yang kebetulan lolos sebuah uji.	Subbab 16.2

Tabel 16.1 (Lanjutan) *Satu gagasan per bab, dan di mana bab ini memakainya. Jika Anda dapat menyebutkan keenam belasnya di luar kepala, seluruh buku ada di tangan Anda.*

Bab	Satu gagasannya	Dipakai di sini
9	Pilihan yang terbaik secara lokal hanya aman secara global jika sifat pilihan rakus (*greedy-choice property*) sudah dibuktikan.	Persoalan 16.4 dan 16.16
10	Satu kerangka penelusuran, dua wadah: antrean memberi jumlah sisi paling sedikit, tumpukan memberi struktur.	Persoalan 16.1 (pernyataan 13) dan 16.8
11	Sifat potongan: sisi teringan yang menyeberangi potongan mana pun yang dihormati hutan saat ini adalah sisi yang aman.	Persoalan 16.4 dan 16.21
12	Sebuah algoritma baru menjadi artefak rekayasa ketika ia diimplementasikan, diukur, dan dipertahankan.	Subbab 16.16
13	Relaksasi ditambah sebuah urutan. Urutan Dijkstra menurut jarak, urutan Bellman–Ford “semua sisi, $\lvert V\rvert - 1$ kali”, urutan DAG topologis.	Persoalan 16.5 sampai 16.8
14	Biaya sebuah barisan operasi bukanlah langkah tunggal terburuk dikalikan banyaknya langkah.	Persoalan 16.9 dan 16.10
15	Rekurensi yang sama dua kali: atas panjang lintasan, dan atas simpul mana yang boleh menjadi simpul antara. Pembobotan ulang membuat Dijkstra sah pada sisi negatif.	Persoalan 16.11, 16.12, dan 16.17
16	Memilih algoritma adalah keterampilannya. Sisanya pembukuan.	Subbab 16.14

Tabel 16.1 menyebutkan, untuk setiap bab, satu gagasan yang memungkinkan Anda membangun kembali sisanya, dan di mana persoalan hari ini memakainya.

Jika Anda dapat menyebutkan keenam belas gagasan itu di luar kepala, seluruh buku ada di tangan Anda. Buku: Subbab 16.1.2.

Cara Menghadapi Persoalan Berbagian

Urutan tetap di dalam setiap bagian: (1) jawabannya dalam satu kalimat; (2) asumsinya, dengan setiap kata dalam persoalan didefinisikan; (3) penurunan atau buktinya; (4) sebuah contoh terkerjakan, jangan pernah dilewatkan.

Ketika bukti tidak kunjung selesai: nyatakan klaimnya secara formal; sebutkan tekniknya dan besaran yang diinduksikan; kerjakan kasus dasarnya; nyatakan hipotesisnya dengan persis; tuliskan langkah yang dapat dibenarkan dan **beri label pada celahnya**; periksa pada sebuah contoh.

> **Bukti yang macet diperbaiki dengan struktur, bukan dengan panjang.** Enam baris berlabel (klaim, teknik, basis, hipotesis, apa yang menyusul, apa yang masih kurang) lebih berharga daripada tiga halaman aljabar tanpa label.

Lima kebiasaan: jangan menulis “jelas”; jangan menyebut batas O tanpa kasusnya; pseudokode sebelum kode; ilustrasikan algoritmanya, bukan grafnya; periksa setiap bilangan dua kali.

Buku: Subbab 16.2.

Persoalan 16.1 — Lima Belas Pernyataan

Tandai Benar atau Salah dan beri alasannya: (1) *quick-sort* berkompleksitas asimtotik lebih buruk daripada *merge sort*; (2) pencarian biner $O(\log n)$; (3) pencarian linear $O(n)$ pada larik tak terurut tetapi $O(\log n)$ pada larik terurut; (4) pencarian linear pada senarai berantai terurut $O(n)$; (5) untuk satu pencarian, pencarian linear mengalahkan urutkan-lalu-cari-biner.

(6) Bobot busur yang berbeda semua membuat MST tunggal; (7) pencarian biner memerlukan larik terurut; (8) penyisipan ke senarai terurut dapat dilakukan dalam $O(\log n)$; (9) fungsi *hash* yang baik menyebar kunci secara seragam; (10) pencarian pada tabel *hash* dapat $O(1)$.

(11) $O(g(n))$ berarti sekurang-kurangnya $g(n)$ operasi; (12) selalu pilih kompleksitas asimtotik yang lebih baik; (13) DFS dan BFS tidak pernah mengunjungi > 2 simpul dalam urutan yang sama; (14) pohon biner bertinggi H memuat paling banyak $2^{H+1} - 1$ simpul; (15) pada pohon biner lengkap hanya daun yang tidak beranak.

Apa yang diuji: empat pasang gagasan yang lazim tercampur: batas atas lawan cacahan eksak; kasus terburuk lawan kasus rata-rata; apa yang diizinkan bentuk struktur lawan apa yang dikerjakan algoritma; yang dijamin lawan yang lazim.

Buku: Subbab 16.3.1 dan 16.3.2.

Persoalan 16.1 — Jawabannya

Tabel 16.2 Kelima belas jawaban Persoalan 16.1, berikut pembenaran satu baris untuk masing-masingnya. Sembilan benar dan enam salah.

#	Pernyataan, disingkat	Jawaban	Mengapa
1	Quicksort berkompleksitas asimtotik lebih buruk daripada merge sort	**B**	Kasus terburuk $\Theta(n^2)$ berhadapan dengan $\Theta(n \lg n)$
2	Pencarian biner $O(\log n)$	**B**	$T(n) = T(n/2) + \Theta(1)$
3	Pencarian linear: $O(n)$ tak terurut, $O(\log n)$ terurut	**S**	Pencarian linear $\Theta(n)$ apa pun urutannya
4	Pencarian linear pada senarai berantai terurut $O(n)$	**B**	Tanpa akses acak, jadi tanpa pembagian dua
5	Untuk satu pencarian, linear mengalahkan urutkan-lalu-cari-biner	**B**	$O(n)$ berhadapan dengan $\Omega(n \lg n)$
6	Bobot busur berbeda semua ⇒ MST-nya tunggal	**B**	Argumen pertukaran, Subbab 16.3.9
7	Pencarian biner menuntut lariknya terurut	**B**	Separuh yang dibuang harus dipastikan keliru
8	Penyisipan ke senarai terurut dapat dikerjakan dalam $O(\log n)$	**S**	Mencari tempatnya $O(\lg n)$; membuat ruang $\Theta(n)$
9	Fungsi *hash* yang baik menyebarkan kunci secara seragam	**B**	Itulah definisi *hashing* seragam sederhana
10	Fungsi *hash* baik + data wajar ⇒ pencarian $O(1)$	**B**	Harapan $\Theta(1+\alpha)$ dengan $\alpha = O(1)$

Tabel 16.2 (Lanjutan) *Kelima belas jawaban Persoalan 16.1, berikut pembenaran satu baris untuk masing-masingnya. Sembilan benar dan enam salah.*

#	Pernyataan, disingkat	Jawaban	Mengapa
11	$O(g(n))$ berarti paling sedikit $g(n)$ operasi dijalankan	**S**	O batas atas; Ω yang batas bawah
12	Selalu pilih kompleksitas asimtotik yang lebih baik	**S**	Konstanta, memori, ukuran masukan, kestabilan
13	DFS dan BFS tak pernah sama pada graf > 2 simpul	**S**	Sebuah lintasan atau bintang; 26 dari 38 graf bersimpul empat, Subbab 16.3.11
14	Pohon biner bertinggi H punya paling banyak $2^{H+1} - 1$ simpul	**B**	$\sum_{i=0}^{H} 2^i = 2^{H+1} - 1$
15	Pada pohon biner lengkap hanya daun yang tak beranak	**B**	Benar menurut definisi *daun*

Tabel 16.2: sembilan benar, enam salah, dan setiap yang salah itu salah karena alasan yang diajarkan buku ini.

Pertahanan terhadap salah baca: garis bawahi kata kerja dan kata penentunya pada setiap pernyataan (pencarian linear, sekurang-kurangnya, selalu, tidak mungkin) sebelum menjawab. Buku: Subbab 16.3.3 dan 16.3.12.

Pernyataan (1), (3)–(5), (7), (8)

(1) Benar, dibaca sebagai kasus terburuk: *quicksort* $T(n)=T(n-1)+\Theta(n)=\Theta(n^2)$ pada pemecahan buruk lawan $\Theta(n \lg n)$ milik *merge sort*. Pada kasus rata-rata keduanya $\Theta(n \lg n)$ dan pernyataannya salah. “Lebih cepat dalam praktik” adalah klaim yang lain.

(3) Salah, (4) Benar, (7) Benar. Mengurutkan data mengubah apa yang *dapat* dilakukan algoritma, bukan apa yang *dilakukan* pencarian linear: ia tetap $\Theta(n)$. Senarai berantai tidak memberi akses $\Theta(1)$ ke tengah, jadi tidak ada pembelahan. Langkah pembuangan pencarian biner sah hanya bila datanya terurut.

(5) Benar. Pencarian linear berbiaya $\Theta(n)$; mengurutkan lebih dahulu berbiaya $\Omega(n \lg n)+O(\lg n)$. Dengan k pencarian keduanya bersilang di dekat $k \approx \lg n$: sekitar dua puluh pencarian untuk sejuta unsur.

(8) Salah. Menemukan tempatnya $O(\lg n)$ pada larik; menyediakan ruangnya $\Theta(n)$. Pada senarai berantai kedua separuh bertukar peran. Bagaimanapun juga $\Theta(n)$; pohon seimbang ada justru karena ini.

Buku: Subbab 16.3.4 sampai 16.3.7.

Pernyataan (6), (9)–(15)

(6) Benar, dengan pertukaran. Bila $T_1 \neq T_2$ keduanya minimum, misalkan e sisi teringan pada selisih simetrisnya, katakan $e \in T_1$. Menambahkan e ke T_2 menutup sebuah siklus yang memuat suatu $f \in T_2 \setminus T_1$ dengan $w(e) < w(f)$; maka $T_2 \cup \{e\} \setminus \{f\}$ lebih ringan daripada T_2. Kontradiksi. ∎

(9), (10) Benar: *hashing* seragam sederhana memberi ekspektasi $\Theta(1+\alpha)$. **(12) Salah:** konstanta, memori, ukuran masukan, dan kestabilan kadang mengalahkan asimtotik. **(11) Salah:** O adalah batas atas; "sekurang-kurangnya" adalah Ω.

(14) Benar: $\sum_{i=0}^{H} 2^i = 2^{H+1} - 1$. **(15) Benar** menurut definisi daun. **(13) Salah:** sebuah lintasan $1-2-3$ atau sebuah bintang; program pada Subbab 16.3.11 memeriksa setiap graf pada empat simpul berlabel:

```
edges: 0-1 0-2 0-3   order: 0 1 2 3
edges: 0-1 0-2 0-3 1-2   order: 0 1 2 3
edges: 0-2 0-3 1-3   order: 0 2 3 1

connected graphs on 4 labeled vertices reachable from 0: 38
of those, DFS order = BFS order in 26 of them
```

Kebalikan (6) tidak berlaku: bobot yang seri memungkinkan MST kedua tetapi tidak memaksanya. Buku: Subbab 16.3.8 sampai 16.3.11.

Persoalan 16.2 — Subbarisan Menaik Terpanjang

Persoalan. Rancanglah algoritma berwaktu polinomial untuk menemukan subbarisan menaik monoton terpanjang dari n bilangan bulat yang berbeda semua; berikan asumsi, pseudokode, dan analisis kompleksitas waktunya.

Asumsi, dinyatakan dengan benar. Masukan $S[1..n]$; semua nilai **berbeda**; menaik berarti **tegas**; sebuah **subbarisan** mempertahankan urutan tetapi tidak harus bersebelahan, $1 \le i_1 < \cdots < i_k \le n$; jawabannya sepanjang sekurang-kurangnya 1.

> **Seluruh kesulitannya tersembunyi dalam kata *subbarisan*.** Menyelesaikan persoalan yang bersebelahan memberi algoritma $O(n)$ yang benar untuk persoalan yang tidak ditanyakan siapa pun: pada contoh berikut deret menaik terpanjangnya sepanjang 3, bukan 6.

Polinomial adalah lantai, bukan sasaran: $O(n^2)$ adalah jawaban yang wajar, dan $O(n \lg n)$ lebih baik.

Buku: Subbab 16.4.1 sampai 16.4.3.

Rekurensi dan Pseudokodenya

Perkuat subpersoalannya sampai dapat disusun: $L[i]$ adalah panjang subbarisan menaik terpanjang dari $S[1..i]$ yang **berakhir tepat di** $S[i]$. Dengan menggunting dan menempel,

$$L[i] = \begin{cases} \max\limits_{j<i \,\wedge\, S[j]<S[i]} (L[j]+1) & \text{bila } j \text{ semacam itu ada,} \\ 1 & \text{bila tidak.} \end{cases} \tag{16.2}$$

Jawabannya $\max_i L[i]$; $P[i]$ mencatat j yang memaksimumkan.

Pseudokode 16.1 Program dinamis $O(n^2)$ untuk subbarisan menaik terpanjang, disusul penelusuran balik yang memulihkan subbarisannya sendiri.

```
LIS(S, n)
 1  for i = 1 to n
 2      L[i] = 1
 3      P[i] = 0
 4      for j = 1 to i - 1
 5          if S[j] < S[i] and L[j] >= L[i]
 6              L[i] = L[j] + 1
 7              P[i] = j
 8  let k be an index with the largest value L[k] among L[1..n]
 9  i = 1                                          // penelusuran balik
10  j = k
11  repeat
12      T[i] = S[j]
13      i = i + 1;  j = P[j]
14  until j == 0
15  return L[k] and the reverse of T[1..L[k]]
```

Pada Pseudokode 16.1, uji `L[j] >= L[i]` di baris 5 setara dengan `L[j] + 1 > L[i]` atas bilangan bulat, dan mempertahankan j memenuhi yang *terakhir*. Invarian gelung untuk Persamaan (16.2): $L[1..i-1]$ dan $P[1..i-1]$ benar di awal putaran i.

Buku: Subbab 16.4.4 dan 16.4.5.

Sebuah Contoh, dengan Tangan

$S = \langle 9, 2, 5, 3, 7, 11, 8, 10, 13, 6 \rangle$. Misalnya pada $i = 5$ ($S[5] = 7$): $j = 2$ memberi $L = 2$, $j = 3$ memberi $L = 3$, $j = 4$ tidak memperbaiki; jadi $L[5] = 3$, $P[5] = 3$.

Tabel 16.3 Tabel pemrograman dinamis untuk $S = \langle 9, 2, 5, 3, 7, 11, 8, 10, 13, 6 \rangle$. $L[i]$ adalah panjang subbarisan menaik terpanjang yang berakhir di $S[i]$; $P[i]$ adalah indeks unsur sebelumnya.

i	1	2	3	4	5	6	7	8	9	10
$S[i]$	9	2	5	3	7	11	8	10	13	6
$L[i]$	1	1	2	2	3	4	4	5	6	3
$P[i]$	0	0	2	2	3	5	5	7	8	3

Dari Tabel 16.3: $k = 9$ dengan $L[9] = 6$; penelusuran balik $9 \to 8 \to 7 \to 5 \to 3 \to 2$ memberi $\langle 2, 5, 7, 8, 10, 13 \rangle$, panjang 6, pada posisi $2 < 3 < 5 < 7 < 8 < 9$.

Waktu. $\sum_{i=1}^{n} (i - 1) = \frac{n^2 - n}{2}$ putaran dalam, $\Theta(n^2)$; penelusuran baliknya $\Theta(n)$ karena $P[j] < j$. Pada $n = 10^4$ jauh di bawah satu detik; pada $n = 10^6$ berjam-jam.

Buku: Subbab 16.4.6 dan 16.4.7.

Programnya

Kode 16.1 Algoritma $O(n^2)$ Persoalan 16.2 dalam C++, mencetak tabel L dan P.

```cpp
// Persoalan 16.2: subbarisan menaik monoton terpanjang, dengan pemrograman
// dinamis O(n^2) dari Subbab 16.4.5, dan seluruh tabelnya diperlihatkan.
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;

int main() {
    vector<int> S = {9, 2, 5, 3, 7, 11, 8, 10, 13, 6};
    int n = (int)S.size();

    vector<int> L(n + 1, 1), P(n + 1, 0);          // berbasis 1, seperti pseudokodenya

    for (int i = 1; i <= n; ++i) {
        L[i] = 1;
        P[i] = 0;
        for (int j = 1; j <= i - 1; ++j)
            // "L[j] >= L[i]" adalah uji yang sama dengan "L[j]+1 > L[i]"
            if (S[j - 1] < S[i - 1] && L[j] >= L[i]) {
                L[i] = L[j] + 1;
                P[i] = j;
            }
    }

    cout << " i   : ";
    for (int i = 1; i <= n; ++i) cout << setw(4) << i;
    cout << "\n S[i]: ";
    for (int i = 1; i <= n; ++i) cout << setw(4) << S[i - 1];
```

```
    cout << "\n L[i]: ";
    for (int i = 1; i <= n; ++i) cout << setw(4) << L[i];
    cout << "\n P[i]: ";
    for (int i = 1; i <= n; ++i) cout << setw(4) << P[i];
    cout << "\n";

    int k = 1;
    for (int i = 2; i <= n; ++i) if (L[i] > L[k]) k = i;   // L[k] terbesar yang pertama

    vector<int> T;
    for (int j = k; j != 0; j = P[j]) T.push_back(S[j - 1]);

    cout << "\nlength = " << L[k] << "\nsubsequence =";
    for (int i = (int)T.size() - 1; i >= 0; --i) cout << " " << T[i];
    cout << "\n";
    return 0;
}
```

```
 i   :    1   2   3   4   5   6   7   8   9  10
 S[i]:    9   2   5   3   7  11   8  10  13   6
 L[i]:    1   1   2   2   3   4   4   5   6   3
 P[i]:    0   0   2   2   3   5   5   7   8   3

length = 6
subsequence = 2 5 7 8 10 13
```

Kode 16.1 cocok entri demi entri dengan hitungan tangan.

Buku: Subbab 16.4.8.

Cara Kedua: $O(n \lg n)$

Simpan `tail[k]`, **nilai terkecil yang dapat mengakhiri subbarisan menaik sepanjang** $k+1$ sejauh ini. `tail` terurut, dan setiap unsur baru menggantikan tepat satu entri, yang ditemukan dengan pencarian biner: n pencarian biner, $O(n \lg n)$.

Kode 16.2 Cara $O(n \lg n)$ untuk Persoalan 16.2, mencetak larik `tail` sesudah setiap unsur.

```cpp
// Persoalan 16.2, cara kedua: subbarisan menaik terpanjang yang sama, dalam
// O(n lg n), dengan patience sorting berikut pencarian biner.
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    vector<int> S = {9, 2, 5, 3, 7, 11, 8, 10, 13, 6};
    int n = (int)S.size();

    vector<int> tail;            // tail[k] = ekor terkecil yang mungkin bagi subbarisan
                                 // menaik berpanjang k+1 yang terlihat sejauh ini
    vector<int> tailIdx, parent(n, -1);

    for (int i = 0; i < n; ++i) {
        // posisi pertama yang ekornya >= S[i]; menaik tegas, jadi lower_bound
        int k = (int)(lower_bound(tail.begin(), tail.end(), S[i]) - tail.begin());
        if (k > 0) parent[i] = tailIdx[k - 1];
        if (k == (int)tail.size()) { tail.push_back(S[i]); tailIdx.push_back(i); }
        else                       { tail[k] = S[i];       tailIdx[k] = i;       }
        cout << "after S[" << i + 1 << "]=" << S[i] << "  tail =";
        for (int t : tail) cout << " " << t;
        cout << "\n";
    }

    vector<int> out;
    for (int i = tailIdx.back(); i != -1; i = parent[i]) out.push_back(S[i]);
```

```
    reverse(out.begin(), out.end());

    cout << "\nlength = " << tail.size() << "\nsubsequence =";
    for (int v : out) cout << " " << v;
    cout << "\n";
    return 0;
}
```

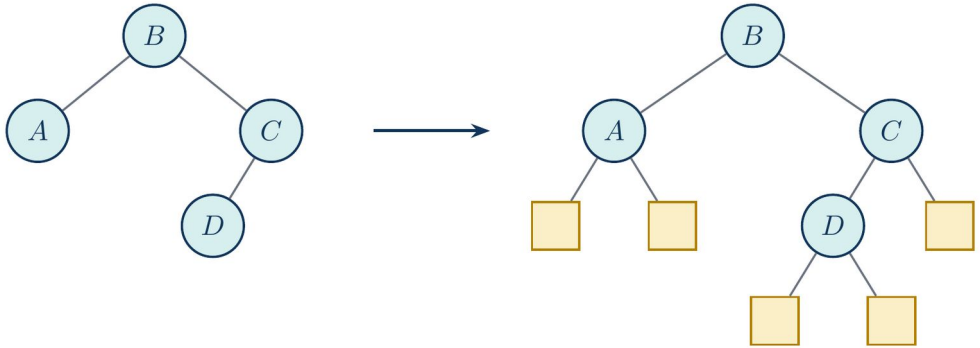
```
after S[1]=9  tail = 9
after S[2]=2  tail = 2
after S[3]=5  tail = 2 5
after S[4]=3  tail = 2 3
after S[5]=7  tail = 2 3 7
after S[6]=11  tail = 2 3 7 11
after S[7]=8  tail = 2 3 7 8
after S[8]=10  tail = 2 3 7 8 10
after S[9]=13  tail = 2 3 7 8 10 13
after S[10]=6  tail = 2 3 6 8 10 13

length = 6
subsequence = 2 3 7 8 10 13
```

Kode 16.2 menemukan $\langle 2,3,7,8,10,13\rangle$: panjangnya sama, subbarisannya lain. Dan `tail` **bukan** jawaban: $2,3,6,8,10,13$ menaruh 6 sebelum 8 padahal 6 datang sesudah 13 di S.

Dua kesalahan klasik: menyelesaikan persoalan yang bersebelahan, dan hanya menghitung $\max_i L[i]$ *tanpa penelusuran balik. Buku: Subbab 16.4.9 dan 16.4.10.*

Persoalan 16.3 — Pohon Biner Diperluas

Gantilah setiap anak yang tidak ada pada pohon biner dengan sebuah **simpul luar**; simpul-simpul aslinya adalah **simpul dalam**; pohon kosong diperluas menjadi satu simpul luar.



Gambar 16.1 Sebuah pohon biner biasa dengan empat simpul (kiri) dan pohon biner diperluas yang diperoleh darinya (kanan). Lingkaran adalah simpul dalam; persegi adalah simpul luar yang menggantikan penunjuk kosong. Empat simpul dalam, lima simpul luar.

Persoalan 16.3. Buktikan dengan induksi bahwa pohon biner diperluas dengan n simpul dalam mempunyai $n+1$ simpul luar. Pada Gambar 16.1: empat simpul dalam, lima simpul luar.

*Jebakannya adalah berinduksi atas tinggi; besaran yang tepat adalah banyaknya simpul dalam, dan bentuk yang tepat adalah induksi **kuat**. Buku: Subbab 16.5.1 dan 16.5.2.*

Buktinya, dan Dua Pemeriksaan

Klaim $X(n)=n+1$. **Basis** $n=0$: pohon kosong diperluas menjadi satu simpul luar. **Hipotesis:** $X(n')=n'+1$ untuk setiap $0\le n'<n$. **Langkah:** buang akarnya; subpohonnya bersimpul dalam k dan $(n-1)-k$, keduanya $<n$, sehingga

$$X(n)=X(k)+X((n-1)-k)=(k+1)+(n-k)=n+1. \tag{16.3}$$

Pada Persamaan (16.3) k **saling meniadakan**, sebab itulah hasilnya berlaku untuk setiap bentuk. Pada Gambar 16.1: $k=1$, $X(1)=2$, $X(2)=3$, total 5.

Alternatif. Buang sebuah simpul dalam yang kedua anaknya simpul luar: dua simpul luar menjadi satu, sehingga $X(n)-1=X(n-1)$. Atau cacah sisinya: $2n$ dari atas, $n+X(n)-1$ dari bawah.

> **Induksi lemah meninggalkan lubang:** subpohon berukuran k dan $n-1-k$ biasanya tidak berukuran $n-1$. Dan definisikan $X(n)$ untuk n simpul **dalam**, tepat sekali dan tepat benar.

Cacah simpul totalnya $2n+1$ selalu ganjil, dan $n+1$ tautan kosongnya adalah yang dimanfaatkan pohon berulir. Buku: Subbab 16.5.3 sampai 16.5.8.

Persoalan 16.4 — Hampiran-2 untuk TSP

Persoalan. Pada graf lengkap dengan jarak metrik: pilih akar r, bangun MST T, daftarkan simpulnya secara **preorder**, kembalikan daftar itu sebagai tur. Buktikan bahwa ia (1) menghasilkan tur, (2) berjalan dalam waktu polinomial, (3) berbatas rasio $\rho(n)=2$.

DEFINISI. Sebuah **tur** adalah siklus Hamilton; TSP mencari yang terpendek. Bobotnya memenuhi **ketaksamaan segitiga** bila

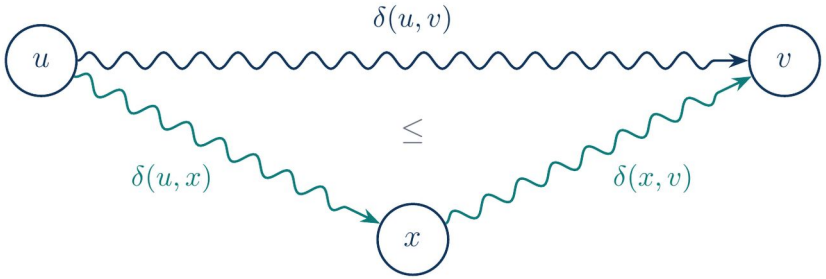
$$w(u,v) \le w(u,x)+w(x,v) \tag{16.4}$$

untuk semua u, v, x. **Batas rasio** $\rho(n)$: $C/C^{*} \le \rho(n)$ pada setiap masukan. **Penelusuran penuh** mendaftarkan simpul setiap kali ditemui; **penelusuran preorder** adalah penelusuran penuh dengan pengulangan dibuang.

Hampiran bukan salah satu bab buku ini, tetapi setiap bahannya ada: MST (Bab 11), penelusuran pohon (Bab 10), dan Persamaan (16.4) (Bab 13).

Buku: Subbab 16.6.1 dan 16.6.2.

Ketaksamaan Segitiga, Sekali Lagi



Gambar 16.2 Ketaksamaan segitiga: pergi dari u ke v secara langsung tidak pernah lebih buruk daripada pergi dari u ke x lalu dari x ke v. Fakta tunggal inilah yang membuat langkah pemintasan pada Persoalan 16.4 sah.

Gambar 16.2 adalah gambar Bab 13, yang di sana dipakai untuk $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$. Di sini ia membuat setiap pintasan penelusuran penuh tidak lebih panjang daripada jalan memutar yang digantikannya.

Jarak garis lurus, jarak berkendara, dan jarak Manhattan bersifat metrik; tabel tarif dengan penerbangan sambungan berpotongan harga mungkin tidak. Buku: Subbab 16.6.2.

Bagian 1 dan 2: Sebuah Tur, dalam Waktu Polinomial

Bagian 1. T merentang V, sehingga penelusuran penuhnya mengunjungi setiap simpul sekurang-kurangnya sekali; mempertahankan kemunculan pertama menyisakan setiap simpul **tepat** sekali. G lengkap, sehingga simpul yang berurutan dan pasangan penutupnya tersambung: sebuah siklus Hamilton. ∎

Tabel 16.4 Biaya setiap langkah algoritma hampiran pada graf lengkap dengan n simpul. Pohon merentang minimumnya mendominasi.

Langkah	Apa yang dikerjakannya	Biaya
1	Memilih akar r	$\Theta(1)$
2	Membangun pohon merentang minimum	$\Theta(n^2)$ dengan Prim dan larik; $O(E \lg V) = O(n^2 \lg n)$ dengan Kruskal atau *binary heap*
3	Penelusuran *preorder* atas T	$\Theta(n)$ — pohonnya punya n simpul dan $n-1$ sisi
4	Mengembalikan L	$\Theta(n)$

Bagian 2. Menurut Tabel 16.4, $\Theta(n^2)$ dengan Prim dan larik, pilihan yang tepat pada graf lengkap, tempat ia mengalahkan versi *heap* $O(n^2 \lg n)$. ∎

Kelengkapan dipakai sekali, di bagian 1; ketaksamaan segitiga sama sekali tidak dipakai di bagian 1. TSP eksak akan memerlukan sekitar 6×10^{16} *tur pada* $n = 20$. *Buku: Subbab 16.6.3 dan 16.6.4.*

Bagian 3: Batas Rasionya 2

Langkah 1, $d(T) \le d(L^*)$**:** membuang satu sisi tur optimal menyisakan pohon merentang T' dengan $d(T') \le d(L^*)$, dan T minimum. **Langkah 2,** $d(W) = 2\,d(T)$**:** penelusuran penuh memakai setiap sisi pohon dua kali.

Langkah 3, $d(L) \le d(W)$**:** setiap pengulangan yang dibuang mengganti $u \to x \to v$ dengan $u \to v$, yang menurut Persamaan (16.4) tidak lebih panjang. **Digabungkan:**

$$d(L) \le d(W) = 2\,d(T) \le 2\,d(L^*), \tag{16.5}$$

jadi menurut Persamaan (16.5) $d(L)/d(L^*) \le 2$. ∎

> **Di mana setiap hipotesis dipakai.** Kelengkapan di bagian 1; ketaknegatifan dan keminimuman T di Langkah 1; ketaksamaan segitiga di Langkah 3 dan tidak di tempat lain.

Tanpa ketaksamaan segitiga tidak ada rasio konstan yang mungkin kecuali $P = NP$: bobot 1 pada sisi H dan $\rho n + 1$ di tempat lain akan memutuskan siklus Hamilton.

Buku: Subbab 16.6.5 dan 16.6.6.

Sebuah Contoh Terkerjakan yang Lengkap

Enam titik jemput Profitina Laundry dengan metrik Manhattan: $A=(0,0)$, $B=(4,0)$, $C=(8,1)$, $D=(6,5)$, $E=(2,6)$, $F=(0,3)$.

Tabel 16.5 Jarak Manhattan dalam blok antara enam titik jemput Profitina Laundry. Matriksnya simetris, diagonalnya nol, dan memenuhi ketaksamaan segitiga — tiga syarat yang menjadikannya metrik.

	A	B	C	D	E	F
A	0	4	9	11	8	3
B	4	0	5	7	8	7
C	9	5	0	6	11	10
D	11	7	6	0	5	8
E	8	8	11	5	0	5
F	3	7	10	8	5	0

Dari Tabel 16.5: Prim dari A mengambil $AF3$, $AB4$, $BC5$, $FE5$, $ED5$: $d(T)=22$. Penelusuran penuh A,B,C,B,A,F,E,D,E,F,A sepanjang $44=2\cdot 22$. Preorder A,B,C,F,E,D: tur $4+5+10+5+5+11=40$.

Gaya kasar atas $5!/2=60$ tur: optimumnya $A\to B\to C\to D\to E\to F\to A$ seharga 28. Periksa: $22\le 28$, $44=44$, $40\le 44$, dan $40/28=1,4286\le 2$.

Batas 2 ketat secara asimtotik untuk algoritma ini; algoritma Christofides mencapai $3/2$. *Buku: Subbab 16.6.7 sampai 16.6.9.*

Persoalan 16.5 — Bellman–Ford Berhadapan dengan Dijkstra

Persoalan. (1) Jelaskan perbedaan dan penerapannya; (2) tuliskan pseudokodenya dan tunjukkan di mana keduanya berbeda; (3) ilustrasikan masing-masing secara singkat. **Apa yang diuji:** bahwa keduanya *gagasan yang sama dengan dua urutan* penerapan Persamaan (16.1).

Tabel 16.6 Dijkstra berhadapan dengan Bellman–Ford. Dua baris terakhir menjelaskan pembagian peran di dunia nyata: *router* memakai Bellman–Ford karena ia lokal; aplikasi peta memakai Dijkstra karena ia cepat dan bobot jalan tidak pernah negatif.

	Dijkstra	Bellman–Ford
Paradigma perancangan	Rakus (Bab 9)	Pemrograman dinamis atas panjang lintasan (Bab 7)
Bobot sisi negatif	**Tidak diizinkan**	Diizinkan
Siklus negatif	Perilaku tak terdefinisi	**Dideteksi**; mengembalikan `false`
Urutan relaksasi	Sisi keluar dari simpul belum-selesai terdekat	Semua sisi, $\lvert V\rvert - 1$ kali
Setiap simpul dituntaskan	Sekali, ketika dikeluarkan dari antrean	Tidak pernah — setiap taksiran dapat turun lagi
Struktur data yang dibutuhkan	Antrean prioritas yang mendukung `Extract-Min` dan `Decrease-Key`	Tidak ada; cukup daftar sisi
Waktu jalan	$\Theta(V^2)$ dengan larik; $O(E \lg V)$ dengan *binary heap*; $O(V \lg V + E)$ dengan *Fibonacci heap*	$\Theta(V E)$
Pada graf jarang, $E = \Theta(V)$	$O(V \lg V)$	$\Theta(V^2)$
Implementasi terdistribusi	Sulit — butuh minimum global	Mudah — tiap simpul hanya butuh tetangganya
Gagal ketika	Ada bobot sisi yang negatif	Sebuah siklus negatif terjangkau (dan ia melaporkannya)

Penerapan, beserta alasannya. Perutean RIP adalah Bellman–Ford tersebar, karena merelaksasi sebuah sisi hanya memerlukan kedua *router* di ujungnya (Tabel 16.6, baris “tersebar”); arbitrase mata uang memakai bobot $-\ln r_{ij}$. Navigasi dan OSPF memakai Dijkstra: bobot tak negatif, graf jarang, kecepatan.

Buku: Subbab 16.7.1 sampai 16.7.3.

Pseudokode Dijkstra

```
DIJKSTRA(G, w, s)
1  untuk tiap v ∈ V
2      lakukan d[v] ← ∞
3  d[s] ← 0
4  S ← ∅      ▷ himpunan simpul yang sudah tuntas
5  Q ← V
6  selama Q ≠ ∅
7      lakukan u ← EXTRACT-MIN(Q)
8          S ← S ∪ {u}
9          untuk tiap v ∈ Adj[u]                      ┐
10             lakukan jika d[v] > d[u] + w(u, v)     │ relaksasi
11                 maka d[v] ← d[u] + w(u, v)         ┘ sisi
```

Gambar 16.3 Algoritma Dijkstra sebagaimana diberikan Bab 13. Baris 1–3 menginisialisasi, baris 7 mengeluarkan simpul belum-selesai terdekat, dan baris 9–11 yang disorot adalah relaksasinya.

Pada Gambar 16.3, baris 4–7 memelihara himpunan selesai S dan antrean prioritas, lalu memilih simpul berikutnya dengan `Extract-Min`: pilihan rakus. Setiap sisi direlaksasi **sekali**.

Buku: Subbab 16.7.4.

Pseudokode Bellman–Ford

```
BELLMAN-FORD(G, w, s)
01  untuk tiap v ∈ V[G]
02      d[v] ← ∞
03      π[v] ← NIL
04  d[s] ← 0
05  untuk i ← 1 sampai |V[G]| − 1 lakukan
06      untuk tiap sisi (u, v) ∈ E[G] lakukan
07          RELAX(u, v, w)
08  untuk tiap sisi (u, v) ∈ E[G] lakukan
09      jika d[v] > d[u] + w(u, v) maka kembalikan false
10  kembalikan true
```

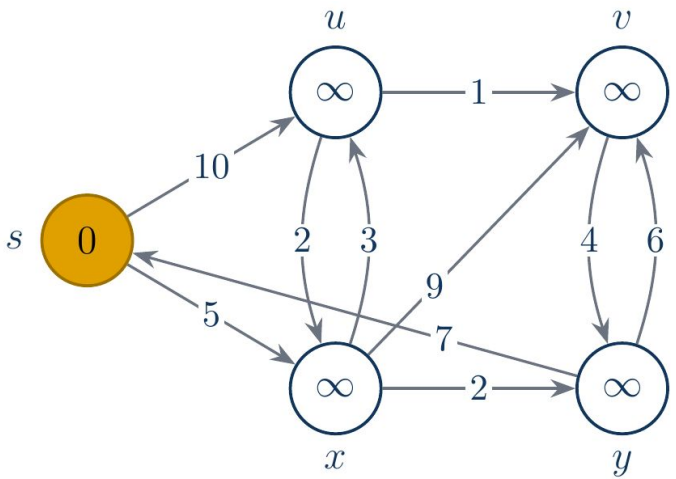
Gambar 16.4 Algoritma Bellman–Ford sebagaimana diberikan Bab 13. Baris 1–4 menginisialisasi persis seperti Dijkstra; baris 5–7 adalah $|V| - 1$ putaran atas semua sisi; baris 8–10 adalah uji siklus negatif yang tidak ada padanannya pada Dijkstra.

Empat perbedaan dengan Gambar 16.3: (1) inisialisasinya sama; (2) Gambar 16.4 tidak memilih apa pun, ia berputar $|V|-1$ kali atas semua sisi; (3) jadi setiap sisi direlaksasi $|V|-1$ kali, $\Theta(VE)$ lawan $O(E \lg V)$; (4) baris 8–10 mendeteksi siklus negatif, yang tidak dapat dilakukan Dijkstra.

Mengapa $|V|-1$ putaran cukup: tanpa siklus negatif, lintasan terpendek bersisi paling banyak $|V|-1$, dan putaran ke-i menuntaskan setiap simpul yang lintasan terpendeknya bersisi paling banyak i.

Buku: Subbab 16.7.4.

Dijkstra, Diilustrasikan: Awalnya

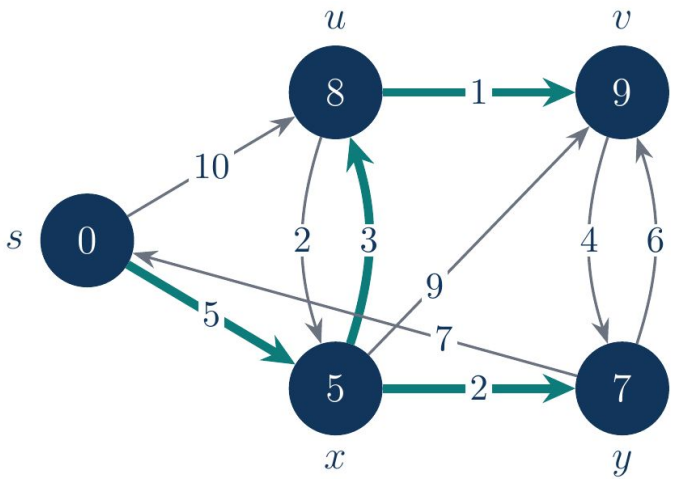


Gambar 16.5 Contoh Dijkstra pada saat inisialisasi: $d[s]=0$, dan $d[u]=d[v]=d[x]=d[y]=\infty$. Bobot sisi $s \to u=10$, $s \to x=5$, $u \to v=1$, $u \to x=2$, $x \to u=3$, $x \to v=9$, $x \to y=2$, $y \to v=6$, $y \to s=7$, $v \to y=4$.

Dari Gambar 16.5: keluarkan s: $d[u]=10$, $d[x]=5$. Keluarkan x: $d[u]=8$, $d[v]=14$, $d[y]=7$. Keluarkan y: $d[v]=13$. Keluarkan u: $d[v]=9$. Keluarkan v: tidak ada yang berubah.

Buku: Subbab 16.7.5.

Dijkstra, Diilustrasikan: Akhirnya

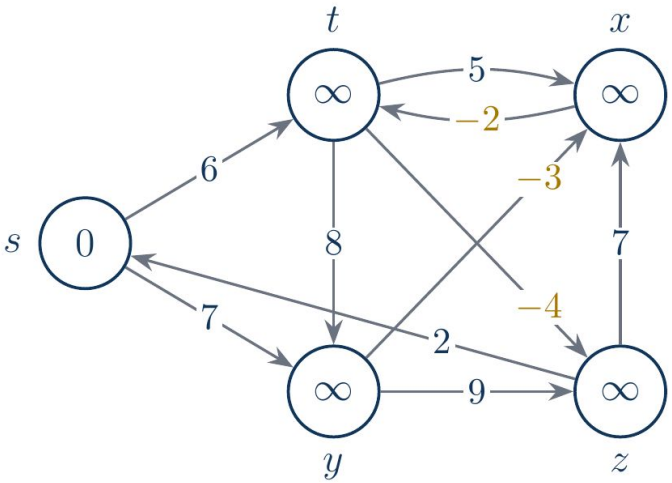


Gambar 16.6 Contoh Dijkstra ketika berhenti: $d[s]=0$, $d[x]=5$, $d[y]=7$, $d[u]=8$, $d[v]=9$. Sisi-sisi tebal adalah pohon lintasan terpendek. Bandingkan bilangannya dengan penelusuran tangan di Subbab 16.7.5.

Gambar 16.6 memperlihatkan setiap simpul sudah tuntas dan pohon lintasan terpendek dicetak tebal. Sebuah ilustrasi harus memperlihatkan taksiran yang berubah, sekurang-kurangnya dua keadaan antara.

Buku: Subbab 16.7.5.

Bellman–Ford, Diilustrasikan: Putaran demi Putaran



Gambar 16.7 Contoh Bellman–Ford pada saat inisialisasi. Bobot: $s \to t = 6$, $s \to y = 7$, $t \to x = 5$, $t \to y = 8$, $t \to z = -4$, $x \to t = -2$, $y \to x = -3$, $y \to z = 9$, $z \to s = 2$, $z \to x = 7$. Tiga dari sepuluh bobotnya negatif, jadi Dijkstra tidak dapat dipakai.

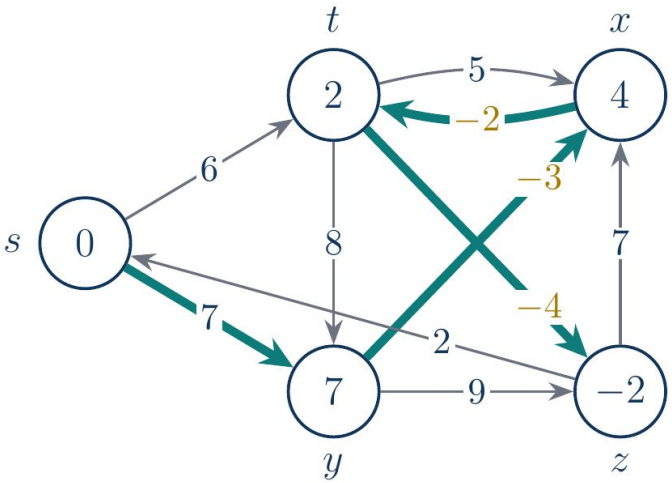
Tabel 16.7 Empat putaran Bellman–Ford pada graf Gambar 16.7. Perhatikan bahwa $d[t]$ *turun* dari 6 ke 2 pada putaran 2 — taksiran yang oleh Dijkstra sudah dituntaskan — dan bahwa putaran 4 tidak mengubah apa pun, yang lalu dikukuhkan oleh pemeriksaan penutupnya.

Sesudah	$d[s]$	$d[t]$	$d[x]$	$d[y]$	$d[z]$
inisialisasi	0	∞	∞	∞	∞
putaran 1	0	6	4	7	2
putaran 2	0	2	4	7	2
putaran 3	0	2	4	7	-2
putaran 4	0	2	4	7	-2

Tabel 16.7 pada Gambar 16.7, dengan relaksasi dalam urutan tercantum: putaran 1 memberi $(0, 6, 4, 7, 2)$ ($d[x]$ menjadi 11 lalu 4 di dalamnya); $d[t]$ turun dari 6 ke 2 pada putaran 2, taksiran yang oleh Dijkstra sudah dibekukan.

Buku: Subbab 16.7.5.

Akhir Bellman–Ford, dan Kegagalan Dijkstra



Gambar 16.8 Bellman–Ford ketika berhenti: $d[s]=0$, $d[t]=2$, $d[x]=4$, $d[y]=7$, $d[z]=-2$. Sisi-sisi tebal adalah pohon lintasan terpendek. Lintasan ke t adalah $s \to y \to x \to t$, berbiaya $7-3-2=2$ — tiga sisi, padahal sisi langsung $s \to t$ berbiaya 6.

Gambar 16.8 adalah keadaan akhirnya. Program pada Subbab 16.7.7 lalu memperlihatkan *mengapa* Dijkstra gagal pada sisi negatif dan Bellman–Ford menolak siklus negatif:

```
graph 1:  s->a = 2,  s->b = 3,  b->a = -2,  a->c = 1
  vertex   Dijkstra   Bellman-Ford
     s          0            0
     a          2            1
     b          3            3
     c          3            2
  the true shortest path to c is s->b->a->c = 3 + (-2) + 1 = 2;
  Dijkstra froze d[a] = 2 before the edge b->a could improve it.

graph 2:  s->a = 2,  a->b = 1,  b->a = -4   (cycle weight -3)
  Bellman-Ford returns false -- a negative-weight cycle is reachable from s
```

Dijkstra membekukan $d[a]=2$ *sebelum* $b \to a$ *dapat memperbaikinya, sehingga* $d[c]$ *tetap 3, bukan 2. Buku: Subbab 16.7.5 sampai 16.7.7.*

Persoalan 16.6 — Dijkstra, Seluruh Tabelnya

Persoalan. Dijkstra dari s pada $s \to a(4)$, $s \to b(1)$, $b \to a(2)$, $a \to c(5)$, $b \to c(8)$, $b \to d(10)$, $c \to d(2)$, $c \to e(6)$, $d \to e(1)$, $a \to d(9)$; berikan d/π sesudah setiap pengeluaran.

Tabel 16.8 Algoritma Dijkstra pada graf Persoalan 16.6, satu baris per `Extract-Min`. Setiap sel adalah $d[v]/\pi[v]$; sel tebal adalah simpul yang dituntaskan pada baris itu.

Sesudah	Dikeluarkan	s	a	b	c	d	e
inisialisasi	—	$0/-$	$\infty/-$	$\infty/-$	$\infty/-$	$\infty/-$	$\infty/-$
1	s	**0/NIL**	$4/s$	$1/s$	$\infty/-$	$\infty/-$	$\infty/-$
2	b	$0/-$	$3/b$	**1/s**	$9/b$	$11/b$	$\infty/-$
3	a	$0/-$	**3/b**	$1/s$	$8/a$	$11/b$	$\infty/-$
4	c	$0/-$	$3/b$	$1/s$	**8/a**	$10/c$	$14/c$
5	d	$0/-$	$3/b$	$1/s$	$8/a$	**10/c**	$11/d$
6	e	$0/-$	$3/b$	$1/s$	$8/a$	$10/c$	**11/d**

Pada Tabel 16.8, iterasi 2 memperbaiki a dari 4 ke 3 lewat $s \to b \to a$; $a \to d$ pada iterasi 3 ($12 \not< 11$) satu-satunya relaksasi yang gagal. Jawabannya $\delta = (0, 3, 1, 8, 10, 11)$; pohonnya satu lintasan $s \to b \to a \to c \to d \to e$.

Pemeriksaan mandiri di d: keenam rutenya berbiaya $11, 13, 12, 11, 11, 10$; minimumnya 10.

Sembilan relaksasi berhasil, empat di antaranya operasi `Decrease-Key` pada implementasi heap. Buku: Subbab 16.8.1.

Persoalan 16.6 dalam C++

Kode 16.3 Algoritma Dijkstra pada graf Persoalan 16.6, mencetak pasangan $d[v]/\pi[v]$ untuk setiap simpul sesudah setiap pengeluaran, lalu setiap lintasan terpendek yang dibaca balik lewat π.

```cpp
// Persoalan 16.6: algoritma Dijkstra dengan seluruh tabel (d, pi) dicetak sesudah
// tiap EXTRACT-MIN, sehingga setiap baris penelusuran tangan dapat diperiksa mesin.
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <queue>
#include <climits>
using namespace std;

const int INF = INT_MAX / 4;
struct Edge { int u, v, w; };

int main() {
    vector<string> nm = {"s", "a", "b", "c", "d", "e"};
    vector<Edge> E = {{0,1,4}, {0,2,1}, {2,1,2}, {1,3,5}, {2,3,8},
                      {2,4,10}, {3,4,2}, {3,5,6}, {4,5,1}, {1,4,9}};
    int n = (int)nm.size();
    vector<vector<pair<int,int>>> adj(n);
    for (auto& e : E) adj[e.u].push_back({e.v, e.w});

    vector<int> d(n, INF), pi(n, -1);
    vector<bool> done(n, false);
    d[0] = 0;
    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<>> Q;
    Q.push({0, 0});

    auto row = [&](const string& tag) {
```

```
        cout << setw(11) << left << tag << right;
        for (int v = 0; v < n; ++v) {
            string cell = (d[v] >= INF) ? "inf" : to_string(d[v]);
            cell += "/";
            cell += (pi[v] < 0) ? "-" : nm[pi[v]];
            cout << setw(8) << cell;
        }
        cout << "\n";
    };
    cout << setw(11) << left << "step" << right;
    for (int v = 0; v < n; ++v) cout << setw(8) << ("d/pi " + nm[v]);
    cout << "\n";
    row("init");
```

Kode 16.3 (Lanjutan) *Algoritma Dijkstra pada graf Persoalan 16.6, mencetak pasangan d[v]/\pi[v] untuk setiap simpul sesudah setiap pengeluaran, lalu setiap lintasan terpendek yang dibaca balik lewat \pi.*

```
    int step = 0;
    while (!Q.empty()) {
        auto [du, u] = Q.top(); Q.pop();
        if (done[u] || du > d[u]) continue;       // entri antrean yang basi
        done[u] = true;
        for (auto [v, w] : adj[u])
            if (d[u] + w < d[v]) { d[v] = d[u] + w; pi[v] = u; Q.push({d[v], v}); }
        row(to_string(++step) + ": " + nm[u]);
    }

    cout << "\nshortest paths from s:\n";
    for (int v = 0; v < n; ++v) {
        vector<int> path;
        for (int x = v; x != -1; x = pi[x]) path.push_back(x);
        cout << "  " << nm[v] << "  d = " << setw(2) << d[v] << "  path:";
        for (int i = (int)path.size() - 1; i >= 0; --i) cout << " " << nm[path[i]];
        cout << "\n";
    }
    return 0;
}
```

```
step         d/pi s  d/pi a  d/pi b  d/pi c  d/pi d  d/pi e
init            0/-   inf/-   inf/-   inf/-   inf/-   inf/-
1: s            0/-     4/s     1/s   inf/-   inf/-   inf/-
2: b            0/-     3/b     1/s     9/b    11/b   inf/-
3: a            0/-     3/b     1/s     8/a    11/b   inf/-
4: c            0/-     3/b     1/s     8/a    10/c    14/c
5: d            0/-     3/b     1/s     8/a    10/c    11/d
6: e            0/-     3/b     1/s     8/a    10/c    11/d

shortest paths from s:
  s  d =  0  path: s
  a  d =  3  path: s b a
  b  d =  1  path: s b
  c  d =  8  path: s b a c
  d  d = 10  path: s b a c d
  e  d = 11  path: s b a c d e
```

Baris demi baris, Kode 16.3 mencetak tabel hitungan tangan.

Periksa diri Anda: dengan $w(s,b)=5$ *urutannya menjadi* s,a,b,c,d,e *dan* b *berakhir sebagai daun. Buku: Subbab 16.8.1.*

Persoalan 16.7(a) — Bellman–Ford, Tanpa Siklus

Persoalan. Sisi-sisi direlaksasi dalam urutan $(b,c,-2)$, $(a,b,4)$, $(s,a,3)$, $(s,b,6)$, $(c,d,3)$, $(a,c,5)$, $(d,b,1)$. Satu-satunya siklus $b \to c \to d \to b$ berbobot $2>0$.

Tabel 16.9 Bellman–Ford pada graf Persoalan 16.7(a), dengan $d[v]/\pi[v]$ sesudah setiap putaran. Taksiran-taksirannya mapan sesudah putaran 2; putaran 3 dan 4 tidak mengubah apa pun dan pemeriksaan penutupnya mengembalikan TRUE.

Sesudah	s	a	b	c	d	Perubahan
inisialisasi	$0/-$	$\infty/-$	$\infty/-$	$\infty/-$	$\infty/-$	—
putaran 1	$0/-$	$3/s$	$6/s$	$8/a$	$\infty/-$	3
putaran 2	$0/-$	$3/s$	$6/s$	$4/b$	$7/c$	2
putaran 3	$0/-$	$3/s$	$6/s$	$4/b$	$7/c$	0
putaran 4	$0/-$	$3/s$	$6/s$	$4/b$	$7/c$	0

Pada putaran 1 di Tabel 16.9, (c,d) diperiksa sebelum (a,c) membuat $d[c]$ berhingga: urutan sisi memakan satu putaran. Lintasan terpendek $s \to b \to c \to d$ (bobot 7) sudah terelaksasi penuh pada putaran 2. Jawabannya $(0,3,6,4,7)$, TRUE.

Putaran tanpa perubahan berarti tidak akan ada lagi yang berubah, sehingga keluar lebih awal sah. Buku: Subbab 16.8.2.

Persoalan 16.7(b) — Siklus Negatif Diperlihatkan

Tetapkan $w(d,b)=-2$: siklus $b \to c \to d \to b$ kini berbobot -1, terjangkau lewat $s \to b$.

Tabel 16.10 Bellman–Ford pada graf Persoalan 16.7(b). Ketiga taksiran pada siklus itu turun 1 pada setiap putaran, dan pemeriksaan penutupnya menemukan bahwa (b,c) masih dapat direlaksasi, sebab $3+(-2)=1<2$.

Sesudah	s	a	b	c	d	Perubahan
inisialisasi	$0/-$	$\infty/-$	$\infty/-$	$\infty/-$	$\infty/-$	—
putaran 1	$0/-$	$3/s$	$6/s$	$8/a$	$\infty/-$	3
putaran 2	$0/-$	$3/s$	$5/d$	$4/b$	$7/c$	3
putaran 3	$0/-$	$3/s$	$4/d$	$3/b$	$6/c$	3
putaran 4	$0/-$	$3/s$	$3/d$	$2/b$	$5/c$	3

Pada Tabel 16.10 taksiran pada siklus turun 1 setiap putaran. **Pemeriksaan penutup:** (b,c) memberi $3+(-2)=1<2$, sehingga algoritmanya mengembalikan FALSE.

Perlihatkan siklusnya. Dari c ikuti π mundur $|V|=5$ langkah, $c \to b \to d \to c \to b \to d$, mendarat pada siklus di d; kelilingi sampai d lagi lalu balikkan: $b \to c \to d \to b$, bobot $-2+3-2=-1$.

Lima langkah cukup karena rantai enam simpul dari lima simpul pasti mengulang salah satunya. Buku: Subbab 16.8.2.

Persoalan 16.8 — Lintasan Terpendek pada DAG

Pseudokode 16.2 Lintasan terpendek sumber tunggal pada graf berarah taksiklik: satu pengurutan topologis, lalu satu relaksasi per sisi.

```
DAG-SHORTEST-PATHS(G, w, s)
1  topologically sort the vertices of G
2  for each vertex v in G.V
3      v.d = infinity;  v.pi = NIL
4  s.d = 0
5  for each vertex u, taken in topologically sorted order
6      for each vertex v in G.Adj[u]
7          RELAX(u, v, w)
```

(a) DFS dari s memberi waktu selesai $e\,6$, $d\,7$, $c\,8$, $a\,9$, $b\,11$, $s\,12$; menurut waktu selesai yang *menurun* urutannya s, b, a, c, d, e, dan kesepuluh sisi mengarah maju. **(b)** Satu putaran Pseudokode 16.2:

Tabel 16.11 Algoritma DAG pada Persoalan 16.8, satu baris per simpul yang diproses dalam urutan topologis. Nilai setiap simpul sudah tuntas pada saat ia diproses, sebab semua sisi masuknya berasal dari simpul yang diproses lebih awal.

Diproses	s	a	b	c	d	e
s	$0/-$	$6/s$	$3/s$	$\infty/-$	$\infty/-$	$\infty/-$
b	$0/-$	$1/b$	$3/s$	$7/b$	$10/b$	$\infty/-$
a	$0/-$	$1/b$	$3/s$	$2/a$	$6/a$	$\infty/-$
c	$0/-$	$1/b$	$3/s$	$2/a$	$-1/c$	$4/c$
d	$0/-$	$1/b$	$3/s$	$2/a$	$-1/c$	$0/d$
e	$0/-$	$1/b$	$3/s$	$2/a$	$-1/c$	$0/d$

Dari Tabel 16.11: $\delta = (s:0, a:1, b:3, c:2, d:-1, e:0)$, dan lintasan ke e adalah $s \to b \to a \to c \to d \to e$ berbobot 0. **(c)** Setiap sisi masuk u direlaksasi sebelum u diproses, sehingga tidak ada yang perlu dipilih dan tidak perlu antrean. Dijkstra kebetulan berhasil di sini, karena untung: nilai pengeluarannya $0, 3, 1, 2, -1, 0$ bahkan tidak menaik.

Buku: Subbab 16.8.3.

Analisis Amortisasi: Identitas-Identitasnya

Biaya amortisasi terhadap potensial Φ:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}), \tag{16.6}$$

dan dengan menjumlahkan Persamaan (16.6), potensialnya berteleskop:

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0), \text{ jadi } \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \text{ bilamana } \Phi(D_n) \geq \Phi(D_0). \tag{16.7}$$

Persoalan 16.9. Pencacah k-bit mulai dari nol dan Pseudokode 16.3 dijalankan n kali; tunjukkan bahwa totalnya $< 2n$ dengan metode agregat, akuntansi, potensial, dan dengan induksi.

Pseudokode 16.3 `INCREMENT` pada pencacah biner k-bit: nolkan bit-bit 1 di ujung, lalu jadikan bit 0 yang pertama bernilai 1.

```
INCREMENT(A, k)
1  i = 0
2  while i < k and A[i] == 1
3      A[i] = 0
4      i = i + 1
5  if i < k
6      A[i] = 1
```

Syarat pada Persamaan (16.7) harus berlaku untuk setiap n, sebab itulah kita mengambil $\Phi(D_0)=0$ dan membuktikan $\Phi \geq 0$. Buku: Subbab 16.9 dan 16.9.1.

Persoalan 16.9 — Metode Agregat

Nilai pencacah	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Biaya total
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

Gambar 16.9 Pencacah biner 8-bit dari 0 sampai 16. Bit yang diarsir adalah yang terbalik dalam perjalanan ke nilai berikutnya, dan kolom terakhir adalah jumlah berjalan pembalikan — 31 sesudah enam belas penambahan, tepat di bawah $2 \times 16 = 32$.

Batas naif nk menagih setiap panggilan seolah-olah yang terburuk. **Cacah per bit** (lihat Gambar 16.9): bit i berbalik $\lfloor n/2^i \rfloor$ kali, sehingga

$$C(n) = \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < \sum_{i=0}^{\infty} \frac{n}{2^i} = 2n. \tag{16.8}$$

Pada $n = 16$, Persamaan (16.8) memberi $16+8+4+2+1 = 31 < 32$.

Buku: Subbab 16.9.1, penyelesaian (a).

Akuntansi, Potensial, dan Induksi

(b) Akuntansi. Iuran 2 per panggilan; satu penyetelan-ke-1 membelanjakan 1 dan meninggalkan 1 pada bitnya; penolan dibayar dengan kredit itu. Tabungannya sama dengan banyaknya bit 1, tidak pernah negatif. **(c) Potensial** $\Phi = b_i$: $\hat{c}_i \le (t_i + 1) + (1 - t_i) = 2$; panggilan 8 berbiaya 4 dan $\hat{c}_8 = 4 + 1 - 3 = 2$.

(d) Induksi memerlukan klaim yang lebih kuat: $C(n) + v(n) \le 2n$, dengan v banyaknya bit 1. Langkah: $C(n) + v(n) = C(n-1) + v(n-1) + 2$. **Metode potensial adalah induksi dengan hipotesis yang diperkuat, ditulis sebagai pembukuan.**

Tabel 16.12 Pencacah biner dengan ketiga metode untuk delapan panggilan pertama. Jumlah berjalan $C(i)$ tetap di bawah $2i$ (agregat), tabungannya tidak pernah negatif dan selalu sama dengan banyaknya bit 1 (akuntansi), dan setiap biaya amortisasi tepat 2 (potensial).

i	Pencacah	t_i	c_i	$C(i)$	$2i$	Tabungan	Φ_i	$c_i = c_i + \Phi_i - \Phi_{i-1}$
1	0001	0	1	1	2	1	1	$1+1-0=2$
2	0010	1	2	3	4	1	1	$2+1-1=2$
3	0011	0	1	4	6	2	2	$1+2-1=2$
4	0100	2	3	7	8	1	1	$3+1-2=2$
5	0101	0	1	8	10	2	2	$1+2-1=2$
6	0110	1	2	10	12	2	2	$2+2-2=2$
7	0111	0	1	11	14	3	3	$1+3-2=2$
8	1000	3	4	15	16	1	1	$4+1-3=2$

Pada Tabel 16.12 kolom *tabungan* dan Φ_i sama: kredit dan potensial adalah bilangan yang sama.

Buku: Subbab 16.9.1, penyelesaian (b) sampai (d).

Persoalan 16.9 dalam C++

Kode 16.4 Pencacah biner Persoalan 16.9 dengan total agregat, tabungan akuntansi, dan biaya amortisasi metode potensial dicetak untuk setiap panggilan, disusul jumlah agregat untuk n yang lebih besar.

```
// Persoalan 16.9: pencacah biner k-bit dianalisis dengan tiga cara sekaligus.
//   agregat   : jumlah berjalan biaya sebenarnya, dibandingkan dengan 2n
//   akuntansi : tarif 2 per INCREMENT; tabungan tidak boleh negatif
//   potensial : Phi = banyaknya bit 1; biaya amortisasi = c + Phi_i - Phi_{i-1}
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

const int K = 8;

int main() {
    int A[K] = {0};
    long long total = 0, bank = 0;
    int phiPrev = 0;
    cout << "  i  counter   t  cost  total   2i  bank  Phi  amortized\n";
    for (int i = 1; i <= 16; ++i) {
        int j = 0, t = 0;
        while (j < K && A[j] == 1) { A[j] = 0; ++j; ++t; }   // nolkan bit 1 di ujung
        if (j < K) A[j] = 1;                                 // jadikan satu bit 0 bernilai 1
        int cost = t + (j < K ? 1 : 0);
        total += cost;
        bank += 2 - cost;                  // ditarif 2, dibelanjakan biaya sebenarnya
        int phi = 0;
        for (int b = 0; b < K; ++b) phi += A[b];
        int amort = cost + phi - phiPrev;
        phiPrev = phi;
        string bits;
```

```
        for (int b = K - 1; b >= 0; --b) bits += char('0' + A[b]);
        cout << setw(3) << i << "  " << bits << setw(3) << t << setw(6) << cost
             << setw(7) << total << setw(5) << 2 * i << setw(6) << bank
             << setw(5) << phi << setw(8) << amort << "\n";
    }
    // rumus agregat, diperiksa untuk n yang lebih besar
    for (long long n : {16LL, 100LL, 1000LL, 1000000LL}) {
        long long c = 0;
        for (long long p = 1; p <= n; p *= 2) c += n / p;   // jumlah floor(n / 2^i)
        cout << "n = " << setw(7) << n << ":  total flips = " << setw(7) << c
             << "  <  2n = " << 2 * n << "\n";
    }
    return 0;
}
```

```
  i  counter   t  cost  total   2i  bank  Phi  amortized
  1  00000001  0     1      1    2     1    1       2
  2  00000010  1     2      3    4     1    1       2
  3  00000011  0     1      4    6     2    2       2
  4  00000100  2     3      7    8     1    1       2
  5  00000101  0     1      8   10     2    2       2
  6  00000110  1     2     10   12     2    2       2
  7  00000111  0     1     11   14     3    3       2
  8  00001000  3     4     15   16     1    1       2
  9  00001001  0     1     16   18     2    2       2
 10  00001010  1     2     18   20     2    2       2
 11  00001011  0     1     19   22     3    3       2
 12  00001100  2     3     22   24     2    2       2
 13  00001101  0     1     23   26     3    3       2
 14  00001110  1     2     25   28     3    3       2
 15  00001111  0     1     26   30     4    4       2
 16  00010000  4     5     31   32     1    1       2
n =      16:  total flips =      31  <  2n = 32
n =     100:  total flips =     197  <  2n = 200
n =    1000:  total flips =    1994  <  2n = 2000
n = 1000000:  total flips = 1999993  <  2n = 2000000
```

Kode 16.4: pada $n=1000$ totalnya 1994, amortisasi $1,994$ per panggilan.

Periksa diri Anda: bila bit i berbiaya 2^i, $n=2^m$ penambahan berbiaya $n(\lg n+1)$, jadi amortisasi menyebar biaya tetapi tidak melenyapkannya. Buku: Subbab 16.9.1.

Persoalan 16.10 — Tabel Dinamis

Pseudokode 16.4 `TABLE-INSERT` untuk tabel yang berlipat dua ketika penuh. Baris 3–6 adalah perluasan yang mahal; baris 7 adalah penyisipan dasarnya.

```
TABLE-INSERT(T, x)
1  if T.size == 0
2      allocate T.table with 1 slot;  T.size = 1
3  if T.num == T.size
4      allocate new-table with 2 * T.size slots
5      insert all items in T.table into new-table
6      free T.table;  T.table = new-table;  T.size = 2 * T.size
7  insert x into T.table
8  T.num = T.num + 1
```

(a) Untuk Pseudokode 16.4, $c_i = i$ bila $i-1$ pangkat 2 ($i = 2, 3, 5, 9, 17, \ldots$), selainnya 1. **(b) Agregat:** $n + \sum_j 2^j < 3n$; pada $n = 17$, $17 + 31 = 48 < 51$.

(c) Akuntansi, iuran 3: satu satuan menulis itemnya, satu membayar pemindahannya kelak, satu membayar pemindahan item lama yang sudah menghabiskan kreditnya. Tepat sebelum perluasan $2m$ item, tabungannya tepat $2m$.

(d) Potensial

$$\Phi(T) = 2 \cdot num[T] - size[T]. \tag{16.9}$$

Persamaan (16.9) bernilai 0 ketika kosong dan ≥ 0 karena tabelnya sekurang-kurangnya separuh penuh.

Buku: Subbab 16.9.2, penyelesaian (a) sampai (d).

Kedua Kasus, dan Sembilan Penyisipan

Tanpa perluasan: $\hat{c}_i = 1 + (2num_i - size_i) - (2(num_i - 1) - size_i) = 3.$ **Dengan perluasan:** $\hat{c}_i = num_i + (2num_i - 2(num_i - 1)) - (2(num_i - 1) - (num_i - 1)) = 3.$ ($\hat{c}_1 = 2.$)

Tabel 16.13 Tabel dinamis Persoalan 16.10 dengan ketiga metode untuk sembilan penyisipan pertama. Totalnya tetap di bawah $3i$, tabungannya tetap positif, dan biaya amortisasinya 3 sejak penyisipan kedua — termasuk pada penyisipan mahal $i = 2, 3, 5, 9$.

i	$size_i$	num_i	c_i	Σc	$3i$	Tabungan	$\Phi_i = 2num_i - size_i$	c_i
1	1	1	1	1	3	2	$2 - 1 = 1$	$1 + 1 - 0 =$
2	2	2	2	3	6	3	$4 - 2 = 2$	$2 + 2 - 1 =$
3	4	3	3	6	9	3	$6 - 4 = 2$	$3 + 2 - 2 =$
4	4	4	1	7	12	5	$8 - 4 = 4$	$1 + 4 - 2 =$
5	8	5	5	12	15	3	$10 - 8 = 2$	$5 + 2 - 4 =$
6	8	6	1	13	18	5	$12 - 8 = 4$	$1 + 4 - 2 =$
7	8	7	1	14	21	7	$14 - 8 = 6$	$1 + 6 - 4 =$
8	8	8	1	15	24	9	$16 - 8 = 8$	$1 + 8 - 6 =$

Tabel 16.13 (Lanjutan) *Tabel dinamis Persoalan 16.10 dengan ketiga metode untuk sembilan penyisipan pertama. Totalnya tetap di bawah 3i, tabungannya tetap positif, dan biaya amortisasinya 3 sejak penyisipan kedua — termasuk pada penyisipan mahal i=2,3,5,9.*

i	$size_i$	num_i	c_i	Σc	$3i$	Tabungan	$\Phi_i = 2num_i - size_i$	c_i
9	16	9	9	24	27	3	$18 - 16 = 2$	$9 + 2 - 8 =$

Pada Tabel 16.13 tabungannya selalu $\Phi_i + 1$: satuan lebihnya adalah kembalian dari penyisipan pertama, yang ditagih 3 tetapi berbiaya amortisasi 2.

Program pada Subbab 16.9.2 mencetak pembukuan yang sama dan total 48, 2023, dan 2048575 untuk $n = 17,\ 1000,\ 10^6$. *Buku: Subbab 16.9.2, penyelesaian (e).*

Sebuah Potensial Keliru yang Menggoda

Cobalah $\Phi = num - size$**.** Kasus 1: $\hat{c}_i = 2$. Kasus 2: $\hat{c}_i = num_i + (2 - num_i) = 2$. Setiap biaya amortisasi paling besar 2, seolah-olah "membuktikan" $2n$. Padahal pada $n = 17$ total sebenarnya **48**, bukan paling banyak 34.

Mengapa gagal: Φ ini **negatif**, $17 - 32 = -15$ pada $n = 17$. Menurut Persamaan (16.7), $\sum c_i = 33 + 15 = 48$: total amortisasinya mengecilkan kenyataan persis sebesar utang yang ditimbun potensialnya.

$\Phi = num$ tak negatif tetapi memberi $\hat{c}_i = num_i + 1$ pada perluasan: tidak konstan, jadi tidak lebih baik daripada batas naif. Potensial yang baik bernilai besar *tepat ketika operasi mahal sudah dekat*.

> **Selalu periksa** $\Phi(D_i) \geq \Phi(D_0)$ **sebelum memercayai sebuah batas amortisasi.** Potensial yang menjadi negatif meminjam dari masa depan.

Periksa diri Anda: tumbuh dengan 4 slot tetap berbiaya 57 untuk 17 penyisipan dan $\Theta(n^2)$ *secara umum. Buku: Subbab 16.9.2, penyelesaian (f).*

Persoalan 16.11 — Floyd–Warshall dengan Tangan

Grafnya untuk Persoalan 16.11 dan 16.12: $1 \to 2(4)$, $1 \to 4(5)$, $2 \to 3(-2)$, $3 \to 1(3)$, $3 \to 4(2)$, $4 \to 2(1)$, $4 \to 3(6)$. Kelima siklusnya semua berbobot positif.

Rekurensinya: lintasan dengan simpul antara di $\{1..k\}$ menghindari k atau melewatinya sekali,

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{bila } k = 0, \\ \min(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{bila } k \geq 1, \end{cases} \tag{16.10}$$

dan bila argumen kedua Persamaan (16.10) menang, $\pi_{ij}^{(k)} = \pi_{kj}^{(k-1)}$.

Tabel 16.14 Matriks-matriks awal Persoalan 16.11: $D^{(0)}$ adalah matriks bobot, dan $\Pi^{(0)}$ mencatat ekor setiap sisi.

$D^{(0)}$	1	2	3	4		$\Pi^{(0)}$	1	2	3	4
1	0	4	∞	5		**1**	NIL	1	NIL	1
2	∞	0	−2	∞		**2**	NIL	NIL	2	NIL
3	3	∞	0	2		**3**	3	NIL	NIL	3
4	∞	1	6	0		**4**	NIL	4	4	NIL

Tabel 16.14 adalah ronde 0. **Jalan pintas:** pada ronde k hanya entri dengan d_{ik} dan d_{kj} keduanya berhingga, serta $i, j \neq k$, yang dapat berubah.

Buku: Subbab 16.10 dan 16.10.1.

Delapan Perubahan, dan Jawabannya

Tabel 16.15 Setiap entri yang berubah selama Floyd–Warshall pada Persoalan 16.11, berikut perbandingan yang mengubahnya. Perhatikan bahwa d_{32} berubah dua kali: rute yang ditemukan pada ronde 1 dikalahkan oleh rute yang ditemukan pada ronde 4.

Ronde k	Entri	Lama	Calon $d_{ik}+d_{kj}$	Baru	π_{ij} Baru
1	d_{32}	∞	$3+4=7$	7	1
2	d_{13}	∞	$4+(-2)=2$	2	2
2	d_{43}	6	$1+(-2)=-1$	-1	2
3	d_{14}	5	$2+2=4$	4	3
3	d_{21}	∞	$-2+3=1$	1	3
3	d_{24}	∞	$-2+2=0$	0	3
3	d_{41}	∞	$-1+3=2$	2	3
4	d_{32}	7	$2+1=3$	3	4

Tabel 16.15: d_{32} berubah dua kali, 7 lewat 3 → 1 → 2 pada ronde 1, lalu 3 lewat 3 → 4 → 2 pada ronde 4. Hasilnya

$$D^{(4)} = \begin{pmatrix} 0 & 4 & 2 & 4 \\ 1 & 0 & -2 & 0 \\ 3 & 3 & 0 & 2 \\ 2 & 1 & -1 & 0 \end{pmatrix}.$$

Lintasan dari $\Pi^{(4)}$: $1 \to 2 \to 3 \to 4$ (bobot 4), $4 \to 2 \to 3 \to 1$ (2), $3 \to 4 \to 2$ (3). **Siklus negatif:** lihatlah diagonalnya; di sini semuanya nol.

Buku: Subbab 16.10.1.

Persoalan 16.11 dalam C++

Kode 16.5 Floyd–Warshall pada graf Persoalan 16.11, mencetak $D^{(k)}$ dan $\Pi^{(k)}$ untuk $k = 0, \ldots, 4$ dan menandai setiap entri yang berubah, disusul uji siklus negatif pada diagonalnya.

```cpp
// Persoalan 16.11: Floyd-Warshall pada graf empat simpul dengan satu sisi negatif,
// mencetak setiap matriks jarak D(0)..D(4) dan setiap matriks pendahulu
// Pi(0)..Pi(4), serta menandai dengan * setiap entri yang berubah pada ronde itu.
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
using namespace std;

const int INF = 1000000;
const int n = 4;
struct Edge { int u, v, w; };

int D[n + 1][n + 1], P[n + 1][n + 1];          // berbasis 1; P = 0 berarti NIL
bool changed[n + 1][n + 1];

void print(int k) {
    cout << "D(" << k << ")                          Pi(" << k << ")\n";
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
            string c = (D[i][j] >= INF) ? "inf" : to_string(D[i][j]);
            if (changed[i][j]) c += "*";
            cout << setw(6) << c;
        }
        cout << "        ";
        for (int j = 1; j <= n; ++j) {
            if (P[i][j] == 0) cout << setw(4) << "NIL"; else cout << setw(4) << P[i][j];
        }
```

```
        cout << "\n";
    }
    cout << "\n";
}

int main() {
    vector<Edge> E = {{1,2,4}, {1,4,5}, {2,3,-2}, {3,1,3},
                      {3,4,2}, {4,2,1}, {4,3,6}};
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j) { D[i][j] = (i == j) ? 0 : INF; P[i][j] = 0; }
    for (auto& e : E) { D[e.u][e.v] = e.w; P[e.u][e.v] = e.u; }
    print(0);
```

Kode 16.5 (Lanjutan) *Floyd–Warshall pada graf Persoalan 16.11, mencetak D^{(k)} dan \Pi^{(k)} untuk k = 0, \ldots, 4 dan menandai setiap entri yang berubah, disusul uji siklus negatif pada diagonalnya.*

```
    for (int k = 1; k <= n; ++k) {
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j) {
                changed[i][j] = false;
                if (D[i][k] < INF && D[k][j] < INF && D[i][k] + D[k][j] < D[i][j]) {
                    D[i][j] = D[i][k] + D[k][j];   // lewat k saja
                    P[i][j] = P[k][j];             // langkah terakhir disalin dari rute k
                    changed[i][j] = true;
                }
            }
        print(k);
    }

    bool negCycle = false;
    for (int i = 1; i <= n; ++i) if (D[i][i] < 0) negCycle = true;
    cout << "negative diagonal entry: " << (negCycle ? "yes" : "no") << "\n";
    return 0;
}
```

```
D(0)                          Pi(0)
     0     4   inf     5          NIL   1 NIL   1
   inf     0    -2   inf          NIL NIL   2 NIL
     3   inf     0     2            3 NIL NIL   3
   inf     1     6     0          NIL   4   4 NIL

D(1)                          Pi(1)
     0     4   inf     5          NIL   1 NIL   1
   inf     0    -2   inf          NIL NIL   2 NIL
     3    7*     0     2            3   1 NIL   3
   inf     1     6     0          NIL   4   4 NIL

D(2)                          Pi(2)
     0     4    2*     5          NIL   1   2   1
   inf     0    -2   inf          NIL NIL   2 NIL
     3     7     0     2            3   1 NIL   3
   inf     1   -1*     0          NIL   4   2 NIL

D(3)                          Pi(3)
     0     4     2    4*          NIL   1   2   3
    1*     0    -2    0*            3 NIL   2   3
     3     7     0     2            3   1 NIL   3
    2*     1    -1     0            3   4   2 NIL

D(4)                          Pi(4)
     0     4     2     4          NIL   1   2   3
     1     0    -2     0            3 NIL   2   3
     3    3*     0     2            3   4 NIL   3
```

```
     2     1    -1     0          3   4   2 NIL

negative diagonal entry: no
```

Entri bertanda bintang yang dicetak Kode 16.5 persis kedelapan baris tabel perubahan: $4^3 = 64$ perbandingan seluruhnya.

Periksa diri Anda: dengan $w(4,2) = -1$ *diagonalnya menjadi* $(0,-1,-1,-2)$*, menandai siklus* $2 \to 3 \to 4 \to 2$*. Buku: Subbab 16.10.1.*

Persoalan 16.12 — Pembobotan Ulang Johnson

(a) Tambahkan q dengan sisi berbobot nol ke setiap simpul; Bellman–Ford dari q memberi $h=(0,0,-2,0)$: hanya simpul 3 yang tercapai lebih murah daripada gratis, lewat $2 \to 3$.

(b) Bobot ulang setiap sisi dengan

$$w'(u,v)=w(u,v)+h(u)-h(v). \tag{16.11}$$

Tabel 16.16 Pembobotan ulang Johnson pada graf Persoalan 16.12. Setiap sisi yang dibobot ulang tak negatif, dan sisi negatif $2 \to 3$ telah menjadi sisi berbobot 0.

Sisi (u,v)	$w(u,v)$	$h(u)$	$h(v)$	$w'(u,v)=w+h(u)-h(v)$
$(1,2)$	4	0	0	$4+0-0=4$
$(1,4)$	5	0	0	$5+0-0=5$
$(2,3)$	-2	0	-2	$-2+0+2=0$
$(3,1)$	3	-2	0	$3-2-0=1$
$(3,4)$	2	-2	0	$2-2-0=0$
$(4,2)$	1	0	0	$1+0-0=1$
$(4,3)$	6	0	-2	$6+0+2=8$

Tabel 16.16 menerapkan Persamaan (16.11): semuanya tak negatif, menurut ketaksamaan segitiga $h(v) \le h(u) + w(u,v)$; nol-nolnya, $(2,3)$ dan $(3,4)$, terletak pada lintasan terpendek dari q. **(c)** Dijkstra dari 4 memberi $\hat{d} = (2,1,1,0)$, dan $\delta(u,v) = \hat{d}(u,v) - h(u) + h(v)$ memberi $(2,1,-1,0)$.

(d) Sepanjang lintasan mana pun koreksinya berteleskop:

$$w'(p) = \sum_{i=1}^{k} \left(w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i) \right) = w(p) + h(v_0) - h(v_k), \tag{16.12}$$

sehingga menurut Persamaan (16.12) setiap lintasan dengan ujung yang sama bergeser sebesar yang sama.

Program pada Subbab 16.10.2 memulihkan persis $D^{(4)}$ Persoalan 16.11: dua gagasan berbeda, enam belas entri yang sepakat. Buku: Subbab 16.10.2.

Persoalan 16.13 dan 16.18 — Tiga Puluh Pernyataan Lagi

16.13, jebakannya. (2) **S**: siklus apa pun, sekalipun sepanjang 2, meniadakan pengurutan topologis. (4) **S**: *knapsack* 0/1 rakus memperoleh 160 padahal $\{2, 3\}$ memberi 220. (7) **S**: DFS tak berarah tidak punya sisi silang. (11) **S**: Φ harus tetap $\geq \Phi(D_0)$ pada *setiap* langkah. (13) **S**: relaksasi tidak pernah menaikkan $d[v]$.

16.13, yang benar dan layak diberi alasan. (1) Kruskal $O(E \lg V)$; (9) Johnson mengalahkan Floyd–Warshall pada graf jarang; (10) biaya amortisasi dapat melebihi kasus terburuk (`PUSH` 2 lawan 1); (14) setiap sisi MST teringan pada suatu potongan; (15) lintasan terpendek tidak pernah mengulang simpul.

16.18, jebakannya. (2) **S**: `BUILD-MAX-HEAP` $\Theta(n)$. (6) **S**: memoisasi dapat melewatkan subpersoalan. (7) **S**: substruktur optimal tanpa sifat pilihan rakus tidak cukup. (10) **S**: Kruskal memelihara hutan. (11) **S**: sisi dari simpul tak terjangkau tidak pernah direlaksasi. (13) **S**: $\Theta(nW)$ pseudo-polinomial.

16.18, yang benar. (1) *heapsort* di tempat, $\Theta(n \lg n)$; (3) AVL $N(h) = 1 + N(h-1) + N(h-2)$; (5) $\Omega(n \lg n)$ perbandingan; (8) $O(m\,\alpha(n))$; (9) Prim menumbuhkan satu pohon; (12) metode-metodenya boleh membagi biaya secara berbeda; (14) lebih dari $V - 1$ sisi memaksa siklus; (15) sifat siklus.

Buku: Subbab 16.11.1 dan 16.11.6.

Persoalan 16.14 — Lintasan Terpanjang pada DAG

Rancangan. Proses simpul dalam urutan topologis dan relaksasi dengan **maks** alih-alih min: Persoalan 16.8 dibalik, sebab algoritma DAG menerima bobot apa pun, termasuk yang dinegasikan.

Pseudokode 16.5 Lintasan terpanjang dari s pada DAG berbobot: algoritma lintasan terpendek DAG dengan perbandingan yang dibalik.

```
LONGEST-PATH-DAG(G, w, s)
1  topologically sort V(G)
2  for each v in V:  dist[v] = -infinity;  par[v] = NIL
3  dist[s] = 0
4  for each u in topological order
5      if dist[u] > -infinity
6          for each edge (u,v) in Adj[u]
7              if dist[u] + w(u,v) > dist[v]
8                  dist[v] = dist[u] + w(u,v);  par[v] = u
9  return dist, par
```

Pseudokode 16.5 berongkos $\Theta(V+E)$. **Contoh terkerjakan** dalam urutan $1,3,2,5,4,6,7$: $dist[5]=15$, $dist[4]=11$, $dist[6]=19$, $dist[7]=22$; lintasan terpanjangnya $1 \to 2 \to 5 \to 6 \to 7$ berbobot $6+9+4+3=22$.

Pada graf berarah umum ia gagal dua kali: tidak ada urutan topologis, dan lintasan sederhana terpanjang bersifat NP-hard (lintasan Hamilton adalah lintasan terpanjang dengan bobot satuan).

Inilah metode lintasan kritis pada Subbab 13.9.7. Buku: Subbab 16.11.2.

Persoalan 16.15 dan 16.16 — Pohon dan Huffman

16.15: pohon dengan n simpul mempunyai $n-1$ sisi. Induksi kuat: pohon dengan $n \geq 2$ simpul mempunyai daun ℓ; membuang ℓ beserta sisinya menyisakan pohon T' bersimpul $n-1$, terhubung dan asiklik; jadi $|E(T)| = (n-2)+1 = n-1$. ∎

16.16 **(1) Bebas awalan:** simbol hanya berada di daun, dan daun tidak punya keturunan, sehingga tidak ada kata kode yang menjadi awalan kata kode lain. **(2)** $n-1$ penggabungan berisi dua `Extract-Min` dan satu `Insert`: $O(n \lg n)$.

(3) Optimal. *Pilihan rakus:* menukar kedua simbol terjarang ke saudara terdalam mengubah biaya sebesar $-(f(a)-f(x))(d(a)-d(x)) \leq 0$. *Substruktur optimal:* menggabungkan keduanya memberi $cost(T) = cost(T') + f(x) + f(y)$. Induksi atas n menuntaskannya. ∎

Fakta pohon itu dipakai diam-diam pada Persoalan 16.4: tur dikurangi satu sisi adalah pohon merentang. Buku: Subbab 16.11.3 dan 16.11.4.

Persoalan 16.17 — Floyd–Warshall Berhadapan dengan Johnson

Perbedaannya. Floyd–Warshall: program dinamis atas simpul antara yang diizinkan, waktu $\Theta(V^3)$, ruang $\Theta(V^2)$, diagonal negatif untuk siklus negatif. Johnson: sebuah reduksi, satu Bellman–Ford dan V kali Dijkstra, $O(V^2 \lg V + V E)$. **Padat atau kecil: Floyd–Warshall; jarang: Johnson.**

Pseudokode 16.6 Floyd–Warshall (kiri) dan Johnson (kanan). Floyd–Warshall tidak pernah mengubah bobotnya; Johnson mengubahnya sekali, di baris 4, sehingga Dijkstra menjadi sah.

```
FLOYD-WARSHALL(W)                    JOHNSON(G, w)
1 D = W                              1 add vertex q joined to all v with w(q,v)=0
2 for k = 1 to n                     2 if BELLMAN-FORD(G', w, q) = false: report neg. cycle
3   for i = 1 to n                   3 h(v) = delta(q,v) for all v
4     for j = 1 to n                 4 w'(u,v) = w(u,v) + h(u) - h(v)  for all (u,v)
5       D[i][j] = min(D[i][j],       5 for each u in V
6                 D[i][k]+D[k][j])   6   run DIJKSTRA(G, w', u) giving dhat(u,v)
7 return D                           7   D[u][v] = dhat(u,v) - h(u) + h(v)
                                     8 return D
```

Pseudokode 16.6: pembobotan ulang Johnson sah menurut Persamaan (16.12). Pada $V = 1000$, $E = 3000$, Floyd–Warshall mengerjakan 10^9 langkah dan Johnson sekitar 10^7. Pada graf lima simpul, program pada Subbab 16.11.5 mencetak sisi terbobot ulang yang tak negatif $4, 13, 0, 0, 10, 0, 2, 0, 2$.

Buku: Subbab 16.11.5.

Persoalan 16.19 — *Knapsack* 0/1

Rekurensinya. Item i ditinggalkan atau diambil:

$$c[i][j] = \begin{cases} 0 & i = 0 \text{ atau } j = 0, \\ c[i-1][j] & w_i > j, \\ \max(c[i-1][j],\ v_i + c[i-1][j-w_i]) & \text{selainnya.} \end{cases} \tag{16.13}$$

Pseudokode 16.7 Program dinamis *knapsack* 0/1: n baris berisi $W+1$ entri, masing-masing dihitung dalam waktu konstan.

```
KNAPSACK(w, v, n, W)
1 for j = 0 to W:  c[0][j] = 0
2 for i = 1 to n
3   c[i][0] = 0
4   for j = 1 to W
5     if w[i] > j: c[i][j] = c[i-1][j]
6     else:        c[i][j] = max(c[i-1][j], v[i] + c[i-1][j-w[i]])
7 return c[n][W]
```

Pseudokode 16.7 mengisi Persamaan (16.13) dalam $\Theta(nW)$. **Apakah polinomial? Tidak.** W memerlukan $\lceil \lg(W+1) \rceil$ bit, sehingga $\Theta(nW)$ eksponensial terhadap *ukuran* masukan: **pseudo-polinomial**. *Knapsack* 0/1 bersifat NP-hard.

Kode 16.6 Program dinamis *knapsack* 0/1 Persoalan 16.19, mencetak seluruh tabel c dan menyusun kembali himpunan yang terpilih dengan menelusurinya mundur.

```cpp
// Persoalan 16.19: knapsack 0/1 dengan pemrograman dinamis, seluruh tabelnya.
// Lima layanan tambahan, masing-masing memakan menit-mesin dan menghasilkan rupiah.
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
using namespace std;

int main() {
    const int n = 5, W = 10;
    string name[n + 1] = {"", "starch", "express", "perfume", "fold", "iron"};
    int w[n + 1] = {0, 3, 4, 5, 2, 6};           // menit-mesin
    int v[n + 1] = {0, 30, 50, 60, 20, 70};      // ribu rupiah

    vector<vector<int>> c(n + 1, vector<int>(W + 1, 0));
    for (int i = 1; i <= n; ++i)
        for (int j = 0; j <= W; ++j)
            c[i][j] = (w[i] > j) ? c[i-1][j]
                                 : max(c[i-1][j], c[i-1][j - w[i]] + v[i]);

    cout << "      j =";
    for (int j = 0; j <= W; ++j) cout << setw(4) << j;
    cout << "\n";
    for (int i = 0; i <= n; ++i) {
        cout << setw(8) << (i == 0 ? string("-") : name[i]) << " ";
        for (int j = 0; j <= W; ++j) cout << setw(4) << c[i][j];
        cout << "\n";
    }
```

```
    cout << "\nbest value = " << c[n][W] << "\nchosen:";
    int j = W, used = 0;
    for (int i = n; i >= 1; --i)
        if (c[i][j] != c[i-1][j]) { cout << " " << name[i]; used += w[i]; j -= w[i]; }
    cout << "\nminutes used = " << used << " of " << W << "\n";
    return 0;
}
```

```
      j =   0   1   2   3   4   5   6   7   8   9  10
        -   0   0   0   0   0   0   0   0   0   0   0
  starch    0   0   0  30  30  30  30  30  30  30  30
 express    0   0   0  30  50  50  50  80  80  80  80
 perfume    0   0   0  30  50  60  60  80  90 110 110
    fold    0   0  20  30  50  60  70  80  90 110 110
    iron    0   0  20  30  50  60  70  80  90 110 120

best value = 120
chosen: iron express
minutes used = 10 of 10
```

Kode 16.6: express dan iron memberi 120 dalam 10 menit; $c[i][j] \neq c[i-1][j]$ berarti item i diambil. Buku: Subbab 16.11.7.

Persoalan 16.20 — Hampiran-2 untuk Penutup Simpul

`APPROX-VERTEX-COVER` mengambil sembarang sisi (u, v) yang belum tertutup, menambahkan **kedua** ujungnya, dan menghapus setiap sisi yang disentuhnya. **(1)** Ia berhenti hanya ketika setiap sisi tertutup. **(2)** Setiap sisi diperiksa sebanyak konstan kali: $O(V+E)$.

(3) Rasio 2. Sisi-sisi terambil A membentuk sebuah **pemasangan**: tidak ada dua yang berbagi ujung. Setiap penutup memuat satu ujung dari masing-masing, semuanya berbeda, sehingga $|C^*| \geq |A|$. Algoritmanya mengembalikan $|C| = 2|A| \leq 2|C^*|$. ∎

Pada contoh programnya: $|C| = 6$, $|C^*| = 4$, rasio $1{,}5$, ukuran pemasangan 3.

> **Kerangka yang sama dengan Persoalan 16.4:** temukan besaran yang membatasi bawah optimumnya ($|A|$ di sini, $d(T)$ di sana) lalu tunjukkan bahwa jawabannya paling besar dua kalinya.

Buku: Subbab 16.11.8.

Persoalan 16.21 — Prim Berhadapan dengan Kruskal

Keduanya menambahkan sisi yang **aman** menurut sifat potongan. **Prim** menumbuhkan satu pohon dengan antrean prioritas: $\Theta(V^2)$ dengan larik, untuk graf padat. **Kruskal** menumbuhkan hutan dengan pengurutan dan union–find: $O(E \lg V)$, untuk graf jarang atau sisi yang sudah terurut.

Pseudokode 16.8 Prim (kiri) dan Kruskal (kanan). Baris 5 masing-masing adalah tempat keduanya berbeda: simpul terdekat ke pohon, berhadapan dengan sisi berikutnya yang menyambungkan dua komponen.

```
PRIM(G, w, r)                          KRUSKAL(G, w)
1 for each v: key[v]=inf, pi[v]=NIL    1 A = empty
2 key[r] = 0                           2 for each v: MAKE-SET(v)
3 Q = V                                3 sort E by weight, ascending
4 while Q not empty                    4 for each (u,v) in E, in order
5   u = EXTRACT-MIN(Q)                 5   if FIND-SET(u) != FIND-SET(v)
6   for each v in Adj[u]               6     A = A + {(u,v)}
7     if v in Q and w(u,v) < key[v]    7     UNION(u,v)
8       pi[v] = u; key[v] = w(u,v)     8 return A
```

Tabel 16.17 Prim dan Kruskal pada graf enam simpul Tabel 16.5. Prim menambahkan $F-E$ lalu $E-D$; Kruskal menambahkan $D-E$ lalu $E-F$. *Urutannya* berbeda; *himpunan* sisinya $\{AF, AB, BC, DE, EF\}$ dan bobotnya 22 tidak.

Prim dari A	Sisi	w	Kum.		Kruskal	Sisi	w	Putusan	Kum.
1	(akar A)	0	0		1	$A-F$	3	terima	3
2	$A-F$	3	3		2	$A-B$	4	terima	7
3	$A-B$	4	7		3	$B-C$	5	terima	12
4	$B-C$	5	12		4	$D-E$	5	terima	17
5	$F-E$	5	17		5	$E-F$	5	terima	22
6	$E-D$	5	22		—	berhenti: 5 sisi			22

Pseudokode 16.8: Prim bertanya *simpul mana yang terdekat ke pohon?*, Kruskal *apakah sisi ini menyambungkan dua komponen?* Tabel 16.17, pada keenam titik laundry Tabel 16.5: urutan berbeda, himpunan sisi sama, bobot 22.

Tiga sisi seri berbobot 5, namun pencarian menyeluruh hanya menemukan satu MST: bobot seri memungkinkan pohon kedua, bukan menghasilkannya. Buku: Subbab 16.11.9.

Persoalan-Persoalan Dibandingkan

Tabel 16.18 Kedua puluh satu persoalan bab ini sekilas: jenis masing-masing, alat yang menyelesaikannya dan di mana buku ini mengajarkannya, dan biaya penyelesaiannya.

Persoalan	Subbab	Jenisnya	Alat utama, dan di mana ia diajarkan	Biaya penyelesaiannya
16.1 lima belas pernyataan	16.3	pengenalan di seluruh buku	O berhadapan dengan Ω, kasus terburuk berhadapan dengan rata-rata (Bab 3, 5, 6, 11)	—
16.2 subbarisan menaik	16.4	rancangan dan analisis	pemrograman dinamis, subpersoalan yang diperkuat (Bab 7)	$\Theta(n^2)$; $O(n \lg n)$
16.3 pohon biner diperluas	16.5	bukti induksi	induksi kuat atas simpul dalam (Bab 3, 6)	—
16.4 TSP metrik	16.6	jaminan tiga bagian	MST, penelusuran *preorder*, ketaksamaan segitiga (Bab 10, 11, 13)	$\Theta(n^2)$, rasio 2
16.5 Bellman–Ford vs Dijkstra	16.7	perbandingan	relaksasi dan urutannya (Bab 13)	$\Theta(VE)$; $O(E \lg V)$
16.6 tabel Dijkstra	16.8.1	penelusuran	`Extract-Min` ditambah relaksasi (§13.7)	$O(E \lg V)$
16.7 siklus negatif	16.8.2	penelusuran dan deteksi	$\lvert V \rvert - 1$ putaran, pemeriksaan penutup, penelusuran π (§13.8)	$\Theta(VE)$
16.8 lintasan terpendek DAG	16.8.3	penelusuran	pengurutan topologis DFS ditambah relaksasi (§10, §13.9)	$\Theta(V+E)$
16.9 pencacah biner	16.9.1	amortisasi, empat cara	agregat, akuntansi, potensial, induksi (§14.3–14.6)	total $< 2n$

Tabel 16.18 (Lanjutan) *Kedua puluh satu persoalan bab ini sekilas: jenis masing-masing, alat yang menyelesaikannya dan di mana buku ini mengajarkannya, dan biaya penyelesaiannya.*

Persoalan	Subbab	Jenisnya	Alat utama, dan di mana ia diajarkan	Biaya penyelesaiannya
16.10 tabel dinamis	16.9.2	amortisasi, tiga cara	potensial $2\,num - size$ (§14.7)	total $< 3n$
16.11 Floyd–Warshall	16.10.1	penelusuran	rekurensi atas simpul antara (§15.6)	$\Theta(V^3)$
16.12 Johnson	16.10.2	penelusuran	pembobotan ulang dengan potensial (§15.8)	$O(V^2 \lg V + V E)$
16.13 lima belas pernyataan	16.11.1	pengenalan di seluruh buku	Bab 9–15	—
16.14 lintasan terpanjang DAG	16.11.2	rancangan dan analisis	urutan topologis, relaksasi pemaksimum (§13.9)	$\Theta(V+E)$
16.15 sisi pohon	16.11.3	bukti induksi	membuang sebuah daun (Bab 10, 11)	—
16.16 Huffman	16.11.4	jaminan tiga bagian	argumen pertukaran, substruktur optimal (Bab 9)	$O(n \lg n)$
16.17 Floyd–Warshall vs Johnson	16.11.5	perbandingan	padat berhadapan dengan jarang (Bab 15)	$\Theta(V^3)$; $O(V^2 \lg V + V E)$
16.18 lima belas pernyataan	16.11.6	pengenalan di seluruh buku	Bab 5–14	—
16.19 *knapsack* 0/1	16.11.7	rancangan dan analisis	pemrograman dinamis; waktu pseudo-polinomial (Bab 7)	$\Theta(nW)$

Tabel 16.18 (Lanjutan) *Kedua puluh satu persoalan bab ini sekilas: jenis masing-masing, alat yang menyelesaikannya dan di mana buku ini mengajarkannya, dan biaya penyelesaiannya.*

Persoalan	Subbab	Jenisnya	Alat utama, dan di mana ia diajarkan	Biaya penyelesaiannya
16.20 penutup simpul	16.11.8	jaminan tiga bagian	pemasangan menjadi batas bawah optimum	$O(V+E)$, rasio 2
16.21 Prim vs Kruskal	16.11.9	perbandingan	sifat potongan; *heap* berhadapan dengan *union–find* (Bab 11, 14)	$\Theta(V^2)$; $O(E \lg V)$

Yang diperlihatkan Tabel 16.18**.** Setiap persoalan mempunyai pendekatan kentara yang keliru; setiap jawaban diperiksa dua kali; dan beberapa gagasan berulang: relaksasi pada tujuh persoalan, hipotesis yang diperkuat pada dua, batas bawah optimum pada dua.

Satu gagasan, dua bab. *h* milik Johnson dan Φ amortisasi sama-sama menambahkan koreksi yang berteleskop: bandingkan Persamaan (16.12) dengan Persamaan (16.7). Hanya kedua ujungnya yang tersisa.

Buku: Subbab 16.12.

Latihan Uji Diri Singkat

Bab 1–4. Kelima fasenya? Memahami, mengabstraksi, merancang, menganalisis, mengimplementasikan. Tiga kewajiban invarian gelung? Inisialisasi, pemeliharaan, terminasi. Definisikan $f(n) = O(g(n))$. Ketiga kasus Teorema Master.

Bab 5–8. Mengapa `BUILD-MAX-HEAP` $\Theta(n)$? Apa yang dibeli pivot acak? Batas bawah pengurutan perbandingan, dan mengapa? Apa yang membuat persoalan cocok untuk pemrograman dinamis?

Bab 9–12. Dua hipotesis bukti rakus? Mengapa Huffman optimal? BFS dan DFS berbeda dalam hal apa, tepatnya? Nyatakan sifat potongan. Prim atau Kruskal?

Bab 13–16. Apa itu relaksasi? Mengapa Dijkstra gagal pada sisi negatif? Mengapa $|V| - 1$ putaran? Syarat sebuah potensial? Mengapa pembobotan ulang Johnson melestarikan lintasan terpendek? Satu pertanyaan sebelum menulis algoritma apa pun: *apa bentuk masukannya, dan bentuk jawabannya?*

Buku memuat empat puluh latihan dengan jawaban satu baris; yang Anda lewatkan menunjuk bab yang perlu dibaca ulang. Buku: Subbab 16.13.

Tabel Pemilihan Algoritma

Tabel 16.19 Tabel pemilihan algoritma. Bacalah kolom kiri sampai sebuah baris cocok dengan persoalan di depan Anda; tiga kolom lainnya adalah jawabannya, pembenaran yang akan diminta dari Anda, dan anggaran yang harus Anda penuhi.

Persoalannya tampak seperti ini	Pakailah	Mengapa	Biaya
Mengurutkan n butir, memori sempit	Heapsort	Di tempat, dan $\Theta(n \lg n)$ tanpa kasus buruk	$\Theta(n \lg n)$, ruang $O(1)$
Mengurutkan n butir, kecepatan rata-rata paling penting	Quicksort teracak	Konstanta kecil, lokalitas baik	Harapan $\Theta(n \lg n)$
Mengurutkan n butir, kunci yang sama harus mempertahankan urutannya	Merge sort	Satu-satunya yang stabil di antara ketiganya	$\Theta(n \lg n)$, ruang $\Theta(n)$
Mengurutkan n bilangan bulat kecil dalam rentang yang diketahui	*Counting sort* atau *radix sort*	Bukan pengurutan berbasis perbandingan, jadi lantai $\Omega(n \lg n)$ tidak berlaku	$\Theta(n+k)$
Mencari dalam larik terurut yang statis	Pencarian biner	Pembagian dua hanya sahih karena lariknya terurut	$\Theta(\lg n)$
Mencari dengan penyisipan dan penghapusan, penelusuran terurut dibutuhkan	Pohon AVL atau merah-hitam	Mempertahankan urutan *dan* pembaruan $O(\lg n)$	$O(\lg n)$
Mencari dengan penyisipan dan penghapusan, urutan tidak pernah dibutuhkan	Tabel *hash*	Harapan $\Theta(1)$; tanpa urutan	Harapan $\Theta(1)$

Tabel 16.19 (Lanjutan) *Tabel pemilihan algoritma. Bacalah kolom kiri sampai sebuah baris cocok dengan persoalan di depan Anda; tiga kolom lainnya adalah jawabannya, pembenaran yang akan diminta dari Anda, dan anggaran yang harus Anda penuhi.*

Persoalannya tampak seperti ini	Pakailah	Mengapa	Biaya
Berulang kali mengambil yang terkecil atau terbesar	*Binary heap*	`Extract-Min` dalam $\Theta(\lg n)$, `Build` dalam $\Theta(n)$	$\Theta(\lg n)$ per operasi
Memilih sebanyak mungkin butir yang tidak bertumpang tindih	Rakus menurut waktu selesai paling awal	Sifat pilihan rakus berlaku; dibuktikan di Bab 9	$\Theta(n \lg n)$
Mengisi kapasitas dengan barang yang dapat dibagi	Rakus menurut kerapatan nilai	*Knapsack* pecahan; argumen pertukaran	$\Theta(n \lg n)$
Mengisi kapasitas dengan barang yang tidak dapat dibagi	Pemrograman dinamis atas kapasitas	Rakus gagal; rekurensi $c[i][j]$	$\Theta(nW)$, pseudo-polinomial
Memampatkan simbol dengan frekuensi yang diketahui	Huffman	Kode awalan optimal	$O(n \lg n)$
Subpersoalan bertumpang tindih, jawaban dibangun dari potongan optimal	Pemrograman dinamis	Ingat, jangan hitung ulang	bergantung persoalan
Subpersoalan saling bebas	Bagi dan kuasai	Tak ada yang perlu diingat	Teorema Master
Menjangkau setiap simpul, sisi paling sedikit dari satu sumber	BFS	Antreannya memberi jarak yang tak menurun	$\Theta(V+E)$
Mendeteksi siklus, mengurutkan tugas, menemukan komponen	DFS	Waktu penemuan/selesai menyingkap struktur	$\Theta(V+E)$
Mengurutkan tugas yang berprasyarat	Pengurutan topologis	Ada tepat untuk DAG	$\Theta(V+E)$

Tabel 16.19 (Lanjutan) *Tabel pemilihan algoritma. Bacalah kolom kiri sampai sebuah baris cocok dengan persoalan di depan Anda; tiga kolom lainnya adalah jawabannya, pembenaran yang akan diminta dari Anda, dan anggaran yang harus Anda penuhi.*

Persoalannya tampak seperti ini	Pakailah	Mengapa	Biaya
Menyambungkan segalanya dengan biaya terkecil	Prim (padat) atau Kruskal (jarang)	Sifat potongan membuat sisi rakusnya aman	$\Theta(V^2)$ / $O(E \lg V)$
Lintasan terpendek, satu sumber, bobot ≥ 0	Dijkstra	Rakus aman ketika memperpanjang tidak dapat memendekkan	$O(E \lg V)$
Lintasan terpendek, satu sumber, sebagian bobot negatif	Bellman–Ford	Merelaksasi semuanya $\lvert V \rvert - 1$ kali; mendeteksi siklus negatif	$\Theta(V E)$
Lintasan terpendek, satu sumber, grafnya DAG	Relaksasi dalam urutan topologis	Satu putaran cukup; juga menyelesaikan lintasan *terpanjang*	$\Theta(V+E)$
Lintasan terpendek antara setiap pasangan, graf padat	Floyd–Warshall	Tiga gelung, ruang $\Theta(V^2)$	$\Theta(V^3)$
Lintasan terpendek antara setiap pasangan, graf jarang	Johnson	Bobot ulang sekali, lalu V kali Dijkstra	$O(V^2 \lg V + V E)$
Barisan operasi, salah satunya sesekali mahal	Analisis amortisasi	Langkah terburuk bukan langkah rata-rata	—
Jawaban eksaknya NP-sulit dan n besar	Hampiran dengan rasio yang dibuktikan	Jaminan mengalahkan heuristik	bergantung persoalan
Sesuatu harus dipelihara selagi butir-butir berdatangan	Hutan himpunan-lepas, *heap*, atau pohon seimbang	Pilih menurut operasi yang paling sering Anda kerjakan	$O(\alpha(n))$ / $O(\lg n)$

Tabel 16.19 adalah halaman yang perlu disimpan. **Aturan 1:** bentuknya datang dari masukan dan keluaran yang diminta, bukan dari kata-kata persoalannya. **Aturan 2:** sebuah baris adalah titik berangkat, bukan vonis: ukurlah sebelum berkomitmen.

“Sambungkan desa-desa dengan murah” dan “rangkai sebuah cip dengan tembaga sesedikit mungkin” satu baris; “kunjungi setiap desa sekali” adalah baris NP-hard. Buku: Subbab 16.14.

Profitina dalam Praktik

Di dalam platform GEO: antrean perayapan adalah antrean prioritas; kedalaman halaman adalah jarak BFS, dipilih agar perayapan yang terpotong anggaran sudah menjangkau halaman-halaman dangkal; irama pemindaian ulang adalah antrean prioritas dan pemilihan aktivitas; riwayat skor mingguan adalah tabel yang berlipat dua.

Memeriksa dua kali adalah gerbang rilis: lebih dari seribu uji otomatis harus lulus, dan sebuah rilis dinyatakan berhasil hanya sesudah pemeriksaan kesehatannya. Platform *memberi saran*; pelanggan *memutuskan*.

Malam terakhir Profitina Laundry: tiga pertanyaan, tiga algoritma. Pekerjaan setrika mana yang muat pada satu mesin? Pemilihan aktivitas rakus. Dalam urutan apa mobil boks menjemput? MST ditambah penelusuran preorder, dengan rasio Persoalan 16.4. Seberapa jauh setiap lingkungan lewat jalan? Dijkstra.

Jadwal rakusnya melewatkan pekerjaan panjang J4 (08.45–10.30) dan menjalankan empat pekerjaan, terbukti optimal. Kisi Manhattan dan jaringan jalan memberi jarak yang sama di sini hanya karena kebetulan: keduanya dua graf yang berbeda.

Platform memberi saran; pelanggan yang memutuskan, dan skor adalah taksiran, bukan jaminan. Buku: Subbab 16.16 dan 16.16.1.

Kesalahan yang Sering Terjadi

Membaca. Menjawab pernyataan yang Anda harapkan; batas O tanpa kasusnya; subbarisan tertukar dengan substring; menghitung nilai program dinamis tetapi tidak solusinya.

Bukti. Induksi lemah padahal perlu induksi kuat; langkah yang ditegaskan tanpa hipotesisnya; batas rasio tanpa batas bawah optimum.

Lintasan terpendek. Menganggap Dijkstra berlaku umum; mengeluarkan simpul yang salah; melupakan pemeriksaan penutup Bellman–Ford; urutan topologis dari waktu selesai yang *menaik*; membiarkan $\infty + w$ meluap; k pada gelung Floyd–Warshall yang salah; tidak membatalkan pembobotan ulang Johnson; menambahkan konstanta pada setiap sisi.

Amortisasi dan selebihnya. Potensial yang menjadi negatif; amortisasi tertukar dengan kasus rata-rata; pseudo-polinomial disebut polinomial; mengilustrasikan graf alih-alih algoritma; membaca Tabel 16.19 sebagai vonis.

Buku: Subbab 16.17.

Ringkasan, dan Sepatah Kata Penutup

Persoalan 16.1–16.5: sembilan benar dan enam salah; $L[i]$ yang *berakhir di* $S[i]$; $X(n) = n+1$ dengan k saling meniadakan; $d(L) \le d(W) = 2\,d(T) \le 2\,d(L^*)$; dua algoritma yang hanya berbeda dalam urutan relaksasi.

Persoalan 16.6–16.12: Dijkstra, Bellman–Ford dengan pemeriksaan siklus negatif dan penelusuran π, satu putaran DAG; pencacah $< 2n$ dan tabel $< 3n$ dengan ketiga metode; Floyd–Warshall dan Johnson mencapai matriks yang sama.

Persoalan 16.13–16.21 meninjau seluruh buku: lintasan terpanjang DAG, sisi pohon, Huffman, pilihan semua pasangan, *knapsack* pseudo-polinomial, penutup simpul, Prim berhadapan dengan Kruskal.

> **Apa yang kini Anda miliki** bukan katalog algoritma, melainkan kemampuan **mengenali bentuk** persoalan yang belum pernah Anda lihat: Persoalan 16.4 berasal dari pokok bahasan yang tidak diajarkan buku ini, dan semua yang Anda perlukan sudah ada di tangan Anda. Peliharalah kebiasaan memeriksa dua kali, dan aritmetika akibat. Selamat berjuang, dan tetaplah luar biasa.

Buku: Subbab 16.18, termasuk “Sepatah kata penutup”.

Ke Mana Melangkah Selanjutnya

Aliran jaringan (Ford–Fulkerson, aliran maksimum potongan minimum; BFS memberi Edmonds–Karp) serta **NP-kelengkapan dan hampiran** (reduksi; batas rasio seperti Persoalan 16.4 dan 16.20).

Pencocokan string (KMP dalam $\Theta(n+m)$, dianalisis seperti pencacah biner; *suffix array*) dan **geometri komputasional** (pasangan terdekat dengan bagi dan kuasai, *Graham's scan*, garis sapu).

Pemrograman linear (potensial Johnson adalah solusi dual) serta **algoritma teracak dan daring** (masukan buruk menjadi nasib buruk; rasio kompetitif).

Kebiasaan mengukur: setiap pernyataan asimtotik dalam buku ini dapat diperiksa di mesin Anda sendiri dalam sepuluh menit.

Pekerjaan rumah. Soal Latihan 1–14 Bab 16; mulailah dengan 4 (contoh TSP metrik yang lain), 12 (pencacah dengan `RESET`), dan 14 (Johnson pada graf Soal Latihan 13).

Buku: Subbab 16.15 dan 16.19.