MENUJU JUARA 1 NASIONAL

EDISI PERTAMA

IRFAN SUBAKTI

Profitina

profitina.com

DAFTAR ISI

SESI 1 · SESI STRATEGI

Playbook Juara: Strategi, Tips & Trik Hari-H

Medali tidak diputuskan saat lomba — ia diputuskan oleh cara Anda berlatih dan oleh disiplin eksekusi pada hari-H. Sesi ini membekali Anda dengan prosedur operasi standar seorang juara: dari menit pertama membaca soal sampai submit terakhir, semuanya terukur, semuanya disengaja.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Menjalankan protokol 10 menit pertama: memetakan semua soal sebelum menulis satu baris kode pun

(2) Mengalokasikan waktu 5 jam OSN dengan aturan potong-rugi (cut-loss) yang tegas

(3) Memburu subtask: mengubah soal 100 poin menjadi tangga 15-30-55 yang bisa dipanjat

(4) Men-debug dengan protokol, bukan panik: uji kecil, uji tepi, uji brute-force pembanding

(5) Membaca soal sepanjang dua halaman dan mengekstrak model formalnya dalam 5 menit

1.1 Sebelum Lomba: Amunisi yang Disiapkan, Bukan Diimprovisasi

Juara nasional tidak menulis segment tree dari nol di ruang lomba — ia menuangkannya dari otot ingatan yang sudah dilatih ratusan kali. Siapkan dan hafalkan template pribadi: fast I/O, DSU, BFS/DFS, Dijkstra, segment tree, BIT, kombinatorika modular. Latih menulis ulang masing-masing di bawah 4 menit tanpa referensi.

- Tidur cukup dua malam terakhir; performa otak pada jam ke-4 lomba ditentukan oleh tidur, bukan kopi.
- Makan pagi berkarbohidrat kompleks; bawa air. Dehidrasi 2% menurunkan konsentrasi terukur.
- Kenali editor dan compiler di komputer lomba: flag `-O2 -std=c++17`, batas stack, cara kerja judge lokal.
- Datang lebih awal; ritual kecil yang sama setiap lomba menurunkan kecemasan.

Trik juara: notebook kesalahan

Setiap selesai latihan, tulis SATU kalimat: bug apa yang membuang waktu terbanyak hari ini. Sebelum lomba, baca ulang daftar itu — Anda sedang membaca daftar jebakan pribadi Anda.

1.2 Menit 0-10: Protokol Pemetaan Soal

Jangan sentuh keyboard. Baca SEMUA soal dahulu. Untuk tiap soal tulis di kertas: (1) model formalnya satu kalimat, (2) batasan N dan implikasi kompleksitasnya, (3) tebakan kelas algoritma, (4) skor subtask. Urutkan rencana pengerjaan dari expected-points-per-minute tertinggi (Tabel 1.1).

Batasan N	Kompleksitas aman	Kelas algoritma tipikal
N ≤ 20	O(2^N · N)	bitmask, meet in the middle, backtracking
N ≤ 500	$O(N^3)$	DP tabel, Floyd-Warshall
N ≤ 5.000	$O(N^2)$	DP kuadratik, brute pintar
N ≤ 200.000	O(N log N)	sort, segment tree/BIT, D&C, binary search jawaban
N ≤ 10^7	O(N)	sieve, prefix, two pointers
N ≤ 10^{18}	O(log N)	matriks, fast doubling, digit DP

Tabel 1.1 — Membaca batasan = membaca niat pembuat soal.

Kesalahan fatal #1 pemula

Jatuh cinta pada soal pertama dan menghabiskan 2 jam di sana. Aturan juara: jika 45 menit tanpa kemajuan terukur, TINGGALKAN — ambil subtask, pindah. Kembali hanya jika soal lain sudah diamankan.

1.3 Berburu Subtask: Matematika Medali

OSN dinilai per subtask, gaya IOI. Juara 1 nasional hampir tidak pernah orang yang menyelesaikan semua soal sempurna — ia orang yang tidak pernah meninggalkan poin murah di atas meja. 100 poin dari empat soal × 25-40 poin subtask mengalahkan 100 poin dari satu soal sempurna plus tiga nol.

- Subtask 'N kecil' hampir selalu = brute force yang Anda tulis dalam 10 menit. Tulis. Submit. Amankan.
- Brute force yang sama menjadi alat verifikasi solusi penuh Anda nanti — dua manfaat satu kode.
- Simpan solusi yang sudah submit; jangan menimpa kode AC dengan eksperimen.
- 15 menit terakhir: berhenti menulis solusi baru, pastikan semua yang setengah jadi di-submit sebagai subtask.

1.4 Protokol Debug: Dingin, Cepat, Sistematis

- Reproduksi dengan kasus TERKECIL yang gagal — kecilkan input sampai bug telanjang.
- Uji tepi otomatis: N=1, semua sama, semua negatif, kosong, maksimum. Ini membunuh 60% WA.
- Cross-check: brute force + generator acak + script pembanding. Tiga puluh baris yang menyelamatkan medali.
- Overflow: setiap perkalian dua int yang bisa melebihi 2×10^9 → `long long`. Tulis ini di kertas Anda.
- Array off-by-one: indeks dari 1 atau 0 — pilih SATU konvensi seumur hidup dan jangan ganti saat lomba.

```
// Kerangka stress-test — hafalkan polanya / Stress-test skeleton — memorize the pattern
// g++ -O2 sol.cpp -o sol && g++ -O2 brute.cpp -o brute && g++ -O2 gen.cpp -o gen
for (int i = 0; i < 1000; i++) {
  system("./gen > in.txt");
  system("./sol   < in.txt > out1.txt");
  system("./brute < in.txt > out2.txt");
  if (system("diff -q out1.txt out2.txt")) { puts("BEDA! / DIFF!"); break; }
}
```

Trik psikologis saat panik

Tarik napas, tulis di kertas apa yang PASTI benar (sudah teruji) dan apa yang BELUM. Panik datang dari mencampur keduanya. Juara memisahkannya dengan tinta.

1.5 Studi Kasus HARD: HOTLINE (SPOJ #13)

Kartu soal

SPOJ HOTLINE | Classical | Hard | Natural-Language Statement Parsing

Sumber: https://www.spoj.com/problems/HOTLINE/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Sebuah dialog terdiri atas kalimat-kalimat bahasa Inggris sederhana: pernyataan (diakhiri '.'), pertanyaan (diakhiri '?'), dengan subjek seperti i, you, everybody, nobody, atau nama. Program harus mengingat semua pernyataan positif dan negatif, mendeteksi kontradiksi, lalu menjawab pertanyaan 'do/does ...?' dan 'who ...?' persis mengikuti aturan tata bahasa dan format jawaban yang ditentukan soal. Satu karakter melenceng = Wrong Answer: inilah ujian membaca paling murni.

- Beberapa dialog; tiap dialog diakhiri baris kosong — format keluaran 'Dialogue #k:' lalu tiap pertanyaan diikuti jawabannya.

- Aturan verba: is/are/am, bentuk -s untuk orang ketiga, don't/doesn't — semuanya harus dinormalisasi.

Fase 1-2 — Pahami dan Modelkan

Susun peta pernyataan: kunci 'subjek|verba|objek' menuju dua kamus (positif dan negatif). Setiap kalimat baru dinormalisasi (verba ke bentuk dasar, don't/doesn't menandai negatif), disimpan tanpa duplikat, dan dicek tabrakannya; setiap pertanyaan dijawab murni dari pernyataan yang sudah lewat.

Fase 3 — Rancang Algoritmanya

1. Pecah tiap baris menurut tanda akhirnya (., ?, !) menjadi kata-kata; normalisasi verba ke bentuk dasar dan deteksi negasi (don't/doesn't, subjek 'nobody').
2. Simpan fakta baru ke peta positif/negatif; begitu ada pasangan yang bertabrakan — termasuk lewat 'everybody'/'nobody' — tandai seluruh dialog kontradiktif ('I am abroad.').
3. Pertanyaan 'do/does X ...?': cari kunci spesifiknya, lalu kunci 'everybody'/'nobody' yang menaunginya; jawab yes/no/maybe dengan penukaran i ↔ you yang benar.
4. Pertanyaan 'who ...?': kumpulkan semua subjek positif yang cocok dalam urutan kemunculan, tangani kasus 'everybody' dan daftar bersambung 'A, B and C'.

Fase 4 — Analisis Kompleksitas

Maksimal 100 pernyataan per dialog; tiap pertanyaan memindai peta berukuran ≤ 100 — total kerja linear, jauh di bawah batas. Seluruh kesulitan soal ini ada di ketepatan aturan, bukan kecepatan.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <cstdio>
#include <cstring>
#include <string>
#include <vector>
#include <map>
#include <set>
using namespace std;
static char buf[120];
bool isIorYou(const string& s){return s=="I"||s=="you";}
string verbS(const string& v,const string& s){return isIorYou(s)?v:v+"s";}
string dontS(const string& s){return isIorYou(s)?"don't":"doesn't";}
string swap_iy(const string& s){if(s=="I")return "you";if(s=="you")return
"I";return s;}
string baseVerb(const string& v){
  if(!v.empty()&&v.back()=='s')return v.substr(0,v.size()-1);
  return v;
}
struct Stmt{string subj;bool neg;string verb,obj;};
void processDialogue(){
  vector<Stmt> stmts;
  map<string,bool> posMap,negMap;
  bool contradiction=false;
```

```
  vector<pair<string,string>> out;
  while(fgets(buf,sizeof(buf),stdin)){
    int len=strlen(buf);
    while(len>0&&(buf[len-1]=='\n'||buf[len-1]=='\r'))buf[--len]=0;
    if(!len)continue;
    char end=0;
    vector<string> words;string word;
    for(int i=0;i<len;i++){
      char c=buf[i];
      if(c=='.'||c=='?'||c=='!'){if(!word.empty())
{words.push_back(word);word.clear();}end=c;break;}
      else if(c==' '){if(!word.empty()){words.push_back(word);word.clear();}}
      else word+=c;
    }
    if(end=='!'){
      for(auto&[q,a]:out)printf("%s\n%s\n\n",q.c_str(),a.c_str());
      printf("%s\n\n",buf);return;
    }
    if(end=='.'&&words.size()>=2){
      string subj=words[0];bool neg=false;string verb,obj;
      if(words[1]=="don't"||words[1]=="doesn't"){
        neg=true;if(words.size()<3)continue;
        verb=words[2];
        for(int i=3;i<(int)words.size();i++){if(!obj.empty())obj+="
";obj+=words[i];}
      }else{
        string vs=words[1];
        if(subj=="everybody"||subj=="nobody")verb=baseVerb(vs);
        else if(isIorYou(subj))verb=vs;
        else verb=baseVerb(vs);
        for(int i=2;i<(int)words.size();i++){if(!obj.empty())obj+="
";obj+=words[i];}
        if(subj=="nobody")neg=true;
      }
      string key=subj+"|"+verb+"|"+obj;
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
    }
  }
}
int main(){
  int T;scanf("%d\n",&T);
  for(int d=1;d<=T;d++){printf("Dialogue #%d:\n",d);processDialogue();}
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

Bagian yang hilang melakukan dua hal: (1) mencatat pernyataan baru ke posMap/negMap tanpa duplikat, (2) memeriksa kontradiksi — pernyataan positif bertabrakan dengan negatifnya, atau dengan 'nobody ...'; kebalikannya untuk pernyataan negatif melawan 'everybody ...'.

Kunci peta berbentuk 'subjek|verba|objek'. Untuk everybody/nobody, cocokkan dengan akhiran '|verba|objek' pada semua kunci.

Untuk pertanyaan do/does: jawaban 'yes, ...' bila fakta positif ada (langsung atau lewat everybody), 'no, ...' bila fakta negatif ada (langsung atau lewat nobody), selain itu 'maybe.'. Kini juga disembunyikan: seluruh cabang penjawab pertanyaan — pembangun jawaban "who" (mengumpulkan subjek, menyambungnya dengan "and"/koma, mencocokkan bentuk kata kerja tunggal/jamak), pembangun jawaban "what" (mendaftar setiap tindakan yang diketahui dilakukan/tidak dilakukan subjek), dan fallback "maybe.". Bangun ulang ini dari deskripsi Fase 3 di atas: cocokkan lewat kunci

verb+object, utamakan fakta everybody/nobody, dan selalu akhiri dengan titik.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error) dan diuji jalan (smoke test). Status arsip: ACCEPTED di SPOJ; contoh uji resmi soal ini tidak dipublikasikan utuh, jadi verifikasi akhir adalah submit ulang ke SPOJ — jadikan itu bagian dari latihan Anda.

1.6 Latihan Sesi 1

1. Tulis template pribadi Anda (fast I/O + 6 struktur inti) dari nol, ukur waktunya. Target: < 25 menit total.
2. Ambil paket 8 soal OSN tahun lalu: HANYA lakukan protokol menit 0-10 (tanpa coding), lalu bandingkan peta Anda dengan pembahasan resmi.
3. Selesaikan HOTLINE di bawah: soal yang menang bukan karena algoritma dalam, tapi karena membaca teliti — persis keterampilan yang diuji baris pertama OSN.

1.R Ringkasan Sesi dan Poin Kemenangan

- Peta dulu, kode kemudian: 10 menit pemetaan menghemat 60 menit kesasar.
- Poin per menit adalah kompas; subtask murah diamankan lebih dulu.
- Aturan 45 menit dan protokol debug menjaga Anda dari dua pembunuh medali: keras kepala dan panik.
- Semua senjata (template, stress test, konvensi indeks) disiapkan SEBELUM hari-H.

Referensi

[1] Panduan resmi OSN bidang Informatika, Puspresnas/BPTI.

[2] IOI Syllabus (International Olympiad in Informatics), ioinformatics.org.

[3] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[4] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[5] SPOJ — Sphere Online Judge, spoj.com (problem HOTLINE).

[6] SESI 2 · SESI MATERI (2 JAM)

Segment Tree: Struktur Data Penjawab Rentang

Hampir setiap OSN memuat setidaknya satu soal yang berteriak 'rentang': jumlah pada [l,r], maksimum pada [l,r], atau — seperti studi kasus kita — subarray terbaik di dalam [l,r]. Segment tree adalah jawabannya, dan menguasainya sampai level merancang node sendiri adalah pembeda medali perak dan emas.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Membangun segment tree O(N) dan menjawab query O(log N)

(2) Merancang node multi-field dan operasi gabung (merge) yang asosiatif

(3) Menurunkan node 4-field GSS1 (sum, pre, suf, ans) dari nol

(4) Mengenali kapan segment tree kalah oleh prefix sum / sparse table (dan sebaliknya)

2.1 Ide Inti: Jawaban Rentang dari Jawaban Setengah-Rentang

Segment tree membelah [1..N] menjadi dua, rekursif, membentuk pohon biner berkedalaman log N. Setiap node menyimpan JAWABAN untuk rentangnya. Kuncinya satu: jawaban node harus bisa dihitung HANYA dari jawaban dua anaknya — operasi gabung yang asosiatif. Jika Anda bisa menulis fungsi combine(kiri, kanan), Anda bisa menjawab query rentang apa pun dalam O(log N).

- Build: O(N) — setiap node dihitung sekali dari anak-anaknya.
- Query [l,r]: O(log N) — rentang terpecah menjadi ≤ 2 log N node kanonik.
- Memori: array `tree[4N]` — hafalkan faktor 4 ini, penyebab RE paling umum.
- Point update: O(log N) — hitung ulang node di sepanjang satu jalur akar-daun.

Mengapa bukan prefix sum saja?

Prefix sum menjawab JUMLAH rentang statis dalam O(1) — tak terkalahkan. Segment tree menang saat ada UPDATE, atau saat operasinya bukan penjumlahan sederhana (max, gcd, node komposit GSS1).

2.2 Merancang Node: Seni yang Sesungguhnya

Soal HARD tidak pernah meminta 'jumlah rentang'. Ia meminta sesuatu yang aneh — dan tugas Anda menemukan node yang membuat keanehan itu bisa digabung. Resep: (1) tulis apa yang ditanya; (2) coba gabungkan dua jawaban tetangga — informasi apa yang KURANG?; (3) tambahkan informasi itu ke node; (4) ulangi sampai tertutup. Untuk subarray maksimum: ans

kiri dan ans kanan saja tidak cukup, karena jawaban bisa MELINTASI batas — Anda butuh suffix terbaik kiri dan prefix terbaik kanan; dan agar pre/suf bisa digabung, Anda butuh sum. Empat field: tertutup sempurna.

```
// Node GSS1 — hasil dari resep penutupan / GSS1 node — the closure recipe's result
struct Node { long long sum, pre, suf, ans; };
// sum: jumlah seluruh rentang          / total of the range
// pre: subarray terbaik menempel kiri  / best subarray glued to the left edge
// suf: subarray terbaik menempel kanan / best subarray glued to the right edge
// ans: subarray terbaik bebas          / best subarray anywhere
```

Pola pikir juara

Di OSN, 'apakah ini segment tree?' adalah pertanyaan yang salah. Pertanyaan juara: 'informasi apa yang membuat jawaban rentang ini BISA digabung?' — struktur datanya mengikuti.

2.3 Studi Kasus HARD: GSS1 (SPOJ #1043)

Kartu soal

SPOJ GSS1 | Struktur Datas | Hard | Max-Subarray Segment Tree

Sumber: https://www.spoj.com/problems/GSS1/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Diberikan barisan A[1..N] (N ≤ 50.000, |Ai| ≤ 15.007) dan M pertanyaan (x, y). Untuk tiap pertanyaan, cetak nilai maksimum A[i] + A[i+1] + ... + A[j] dengan x ≤ i ≤ j ≤ y — subarray (tak boleh kosong) dengan jumlah terbesar yang seluruhnya berada di dalam rentang [x, y].

- Baris 1: N. Baris 2: N bilangan. Baris 3: M, lalu M baris pasangan x y.
- Semua elemen bisa negatif — subarray kosong tidak diizinkan.

Contoh masukan/keluaran resmi

```
Masukan:
3
-1 2 3
1
1 2
Keluaran:
2
```

Fase 1-2 — Pahami dan Modelkan

Node segment tree menyimpan empat nilai rentangnya: jumlah total (sum), awalan terbaik (pre), akhiran terbaik (suf), dan subarray terbaik (ans). Keempatnya cukup — dan perlu — agar dua rentang bertetangga bisa digabung tanpa kehilangan jawaban yang melintasi batas.

Fase 3 — Rancang Algoritmanya

1. Bangun pohon dari daun make_node(a[l]); node dalam = combine(kiri, kanan).
2. combine: sum dijumlah; pre = max(pre kiri, sum kiri + pre kanan); suf simetris; ans = max(ans kiri, ans kanan, suf kiri + pre kanan).
3. Query mengembalikan NODE utuh (bukan angka), sehingga potongan-potongan rentang bisa digabung dengan combine yang sama.
4. Semua field long long dan boleh negatif — tidak ada 'max dengan 0'.

Fase 4 — Analisis Kompleksitas

Build O(N), tiap query O(log N): untuk N = 50.000 dan M query, total jauh di bawah satu detik. Memori 4N node × 32 byte ≈ 6 MB — aman.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <cstdio>
#include <algorithm>
using namespace std;
const int MX = 50001;
struct Node {
  long long sum, pre, suf, ans;
};
Node tree[4*MX];
int a[MX];
Node make_node(long long v) {
  Node n;
  n.sum = n.pre = n.suf = n.ans = v;
  return n;
}
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
int main() {
  int n;
  scanf("%d", &n);
  for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
  build(1, 1, n);
  int q;
  scanf("%d", &q);
  while (q--) {
    int l, r;
    scanf("%d %d", &l, &r);
    printf("%lld\n", query(1, 1, n, l, r).ans);
  }
  return 0;
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

Empat rumus combine: sum = jumlah kedua sum; pre = max(pre kiri, sum kiri + pre kanan); suf = max(suf kanan, sum kanan + suf kiri); ans = max(ans kiri, ans kanan, suf kiri + pre kanan).

build() standar: daun = make_node(a[l]); node dalam = combine kedua anak setelah rekursi.

Uji mental tercepat: array {-1, 2} query penuh harus menghasilkan 2, bukan 1. Kini juga disembunyikan: combine() (gabungan empat rumus yang sudah dijelaskan di atas), build() (build rekursif standar: leaf = make_node(a[l]), internal = combine dari kedua anak), dan query() (pembagian tiga arah: tercakup penuh / seluruhnya kiri / seluruhnya kanan / menyilang, menggabungkan dua hasil rekursif pada kasus menyilang).

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error), diuji terhadap contoh resmi SPOJ, dan di-stress-test dengan brute force pembanding pada ratusan kasus acak — semua identik. Status arsip: ACCEPTED di SPOJ.

2.4 Latihan Sesi 2

1. Turunkan ulang keempat rumus combine GSS1 di kertas TANPA melihat buku, lalu lengkapi separuh kode yang disembunyikan.
2. Modifikasi node untuk menjawab: panjang subarray naik terpanjang pada [l,r]. Field apa yang dibutuhkan?
3. Stress-test solusi GSS1 Anda melawan brute force $O(N^2)$ dengan 1.000 kasus acak.

2.R Ringkasan Sesi dan Poin Kemenangan

- Segment tree = jawaban rentang yang bisa digabung; combine asosiatif adalah kontraknya.
- Node dirancang lewat resep penutupan: tambah field sampai merge tidak kekurangan informasi.
- GSS1: empat field (sum, pre, suf, ans) menutup masalah subarray maksimum.
- tree[4N], query O(log N), build O(N) — angka-angka yang wajib melekat.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem GSS1).

[4] SESI 3 · SESI MATERI (2 JAM)

Segment Tree Persisten & Statistik Urutan

Bagaimana menjawab 'berapa elemen terkecil ke-k pada rentang [l,r]?' untuk ratusan ribu query — tanpa mengurutkan ulang setiap kali? Jawabannya salah satu ide terindah competitive programming: menyimpan SEMUA versi sejarah sebuah segment tree, hanya dengan biaya O(log N) per versi.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Menjelaskan persistence: path copying dan mengapa biayanya hanya O(log N) per update

(2) Membangun frequency tree atas nilai terkompresi

(3) Menjawab k-th smallest pada rentang dengan selisih dua akar (roots[r] - roots[l-1])

(4) Menerapkan kompresi koordinat dengan sort + unique + lower_bound

3.1 Persistence: Menyimpan Sejarah Tanpa Menyalin Dunia

Update pada segment tree hanya menyentuh satu jalur akar-daun: log N node. Path copying memanfaatkan ini: alih-alih menimpa, buat SALINAN node di jalur itu saja, dan biarkan penunjuk lainnya menunjuk ke node lama yang dibagi bersama versi sebelumnya. Hasilnya: versi ke-i dari pohon hidup berdampingan dengan semua versi sebelumnya, masing-masing hanya menambah O(log N) node baru.

- Node baru per update: O(log N); total memori: O(N log N) — itulah arti MAXN*18 di kode.
- Versi = akar. Menyimpan `roots[i]` berarti menyimpan seluruh keadaan setelah elemen ke-i dimasukkan.
- Dua versi bisa DIKURANGKAN field-per-field: cnt di versi r minus cnt di versi l-1 = frekuensi milik rentang [l,r] saja.

Mengapa pengurangan dua versi sah?

Karena penyisipan bersifat aditif: setiap node menyimpan cnt = banyaknya elemen di sub-rentang nilainya. Prefix [1..r] minus prefix [1..l-1] menyisakan tepat kontribusi elemen l..r — prinsip prefix sum, diangkat ke bentuk pohon.

3.2 Berjalan Menuruni Dua Pohon Sekaligus

Query k-th smallest berjalan menuruni KEDUA akar serempak. Di setiap node hitung left_cnt = cnt(anak kiri versi r) − cnt(anak kiri versi l−1): banyaknya elemen rentang yang jatuh di separuh

nilai bawah. Jika k ≤ left_cnt, jawaban ada di kiri; jika tidak, belok kanan dengan k dikurangi left_cnt. Sampai daun: itulah nilai ke-k. Persis binary search, tapi atas struktur frekuensi.

- Kompresi koordinat wajib: nilai sampai 10^9, tetapi hanya N nilai berbeda — sort, unique, lower_bound.
- Kompleksitas total: O((N+M) log N) — jauh di bawah batas untuk N, M ≤ 10^5.
- Pola 'selisih dua versi' juga menjawab: berapa elemen ≤ x di [l,r], rank sebuah nilai, dan banyak lagi.

3.3 Studi Kasus HARD: MKTHNUM (SPOJ #3947)

Kartu soal

SPOJ MKTHNUM | Struktur Datas | Hard | Persistent Segment Tree

Sumber: https://www.spoj.com/problems/MKTHNUM/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Diberikan array A[1..N] (N ≤ 100.000) dan M ≤ 5.000 pertanyaan (i, j, k): jika segmen A[i..j] diurutkan, cetak elemen pada posisi ke-k. Nilai elemen bisa sampai ±10^9, sehingga kompresi koordinat diperlukan.

- Baris 1: N M. Baris 2: N bilangan. Lalu M baris i j k (1 ≤ k ≤ j−i+1).

Contoh masukan/keluaran resmi

```
Masukan:
7 3
1 5 2 6 3 7 4
2 5 3
4 4 1
1 7 3
Keluaran:
5
6
3
```

Fase 1-2 — Pahami dan Modelkan

Pohon frekuensi atas nilai terkompresi, dibangun bertahap: versi ke-i memuat elemen 1..i. Karena penyisipan aditif, versi r dikurangi versi l−1 memberikan frekuensi murni milik rentang [l,r]; k-th smallest tinggal berjalan menuruni dua versi itu serempak.

Fase 3 — Rancang Algoritmanya

1. Kompresi nilai: sort + unique + lower_bound memetakan ±10^9 ke 1..un.
2. update path-copying: salin node berjalur (cnt+1), turunkan hanya sisi yang memuat posisi nilai; sisi lain berbagi dengan versi lama.

3. kth: left_cnt = cnt[lc[vr]] − cnt[lc[vl]]; k ≤ left_cnt → kiri, selain itu kanan dengan k − left_cnt; daun = jawaban.
4. Pool node MAXN·18 menampung N penyisipan × (log N + 1) node baru.

Fase 4 — Analisis Kompleksitas

O((N+M) log N) waktu dan O(N log N) memori: N = 10^5 berarti ±1,8 juta node — beberapa MB, jauh di dalam batas. Setiap query hanya log N langkah.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <cstdio>
#include <algorithm>
using namespace std;
const int MAXN = 100001;
const int MAXNODES = MAXN * 18; // each insert touches O(log n) nodes
int lc[MAXNODES], rc[MAXNODES], cnt[MAXNODES];
int node_cnt = 0;
int new_node() {
  ++node_cnt;
  lc[node_cnt] = rc[node_cnt] = cnt[node_cnt] = 0;
  return node_cnt;
}
/* ================================================================
   >>> TUGAS ANDA — 17 baris inti DISEMBUNYIKAN (~25%). <<<
   Lengkapi bagian ini sendiri. Petunjuk ada di bawah kode.
   ================================================================ */
}
int roots[MAXN];
int a[MAXN], vals[MAXN];
int main() {
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
  while (m--) {
    int l, r, k;
    scanf("%d %d %d", &l, &r, &k);
    int pos = kth(roots[l-1], roots[r], 1, un, k);
    printf("%d\n", vals[pos]);
  }
  return 0;
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

update(prev, …) menyalin node lama (lc, rc, cnt+1) lalu hanya turun ke SATU sisi sesuai posisi nilai; sisi lain tetap menunjuk versi lama — itulah path copying.

kth berjalan atas dua versi serempak: left_cnt = cnt[lc[vr]] − cnt[lc[vl]]; jika k ≤ left_cnt belok kiri, selain itu belok kanan dengan k − left_cnt.

Basis: lo == hi berarti daun — kembalikan lo (indeks nilai terkompresi). Kini juga disembunyikan: loop pembangunan persistent tree di main() — mengompresi nilai dengan sort+unique, lalu memanggil

update(roots[i-1], 1, un, pos) sekali per elemen untuk menumbuhkan rantai versi. update() sendiri (path copying: kloning node, rekursi hanya ke sisi yang memuat posisi target) dan kth() (menelusuri dua versi tree berbarengan dengan membandingkan jumlah subtree kiri) sudah disembunyikan sejak edisi asli dan tetap disembunyikan di sini.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error), diuji terhadap contoh resmi SPOJ, dan di-stress-test dengan brute force pembanding pada ratusan kasus acak — semua identik. Status arsip: ACCEPTED di SPOJ.

3.4 Latihan Sesi 3

1. Gambar tangan: pohon frekuensi atas [1..4] setelah menyisipkan 3, 1, 4, 1 — semua empat versinya, dengan node yang dibagi bersama diberi warna sama.
2. Lengkapi separuh kode MKTHNUM yang disembunyikan (update path-copying dan penurunan kth).
3. Perluas: jawab 'banyaknya elemen bernilai ≤ x pada [l,r]' dengan pohon yang sama, tanpa struktur baru.

3.R Ringkasan Sesi dan Poin Kemenangan

- Persistence = path copying: O(log N) node baru per versi, semua versi hidup selamanya.
- roots[r] − roots[l−1] mengisolasi frekuensi rentang — prefix sum dalam bentuk pohon.
- K-th smallest = binary search menuruni dua versi serempak.
- Kompresi koordinat adalah refleks, bukan pilihan.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem MKTHNUM).

[4] SESI 4 · SESI MATERI (2 JAM)

Stack Monoton & Teknik Array Linear

Beberapa soal N ≤ 100.000 tampak menuntut struktur berat, padahal jawabannya satu lintasan O(N) dengan stack yang dijaga terurut. Persegi panjang terbesar dalam histogram — soal klasik dunia yang juga favorit juri OSN — adalah contoh sempurnanya: setiap batang masuk stack sekali, keluar sekali, dan jawaban lahir tepat saat sebuah batang 'tertutup'.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Menjaga invarian stack monoton dan membaca maknanya secara geometris

(2) Menghitung 'nearest smaller element' kiri-kanan dalam satu lintasan

(3) Menyelesaikan HISTOGRA O(N) dan membuktikan amortisasinya

(4) Mengenali kerabatnya: trapping rain water, span saham, subarray minimum-sum

4.1 Invarian yang Bekerja untuk Anda

Stack monoton menyimpan indeks dengan nilai menaik (atau menurun). Saat elemen baru melanggar keterurutan, elemen puncak di-pop — dan justru pada momen pop itulah kita tahu SEGALANYA tentang elemen tersebut: tetangga lebih kecil di kanannya adalah elemen baru, tetangga lebih kecil di kirinya adalah elemen di bawahnya dalam stack. Dua informasi yang tampak global, diperoleh secara lokal.

- Amortisasi: setiap indeks push sekali dan pop maksimal sekali → total O(N), meski ada loop dalam loop.
- Sentinel menyederhanakan: batang tinggi 0 di ujung memaksa semua isi stack diproses tanpa kasus khusus.
- Lebar saat pop: dari (elemen di bawah puncak) + 1 sampai (posisi sekarang) − 1.

Cara mengenalinya di lomba

Kata kunci: 'terdekat yang lebih kecil/besar', 'selama masih ≥', 'persegi panjang maksimal', 'berapa lama bertahan'. Jika jawaban tiap elemen ditentukan tetangga terdekat yang melanggar syarat — itu stack monoton.

4.2 Dari Histogram ke Matriks: Naik Kelas

Soal 'submatriks 1 terbesar pada matriks biner' — favorit lain juri — adalah HISTOGRA yang ditumpuk: baris demi baris, tinggi[i] = berapa banyak 1 berturut-turut di atas sel ini, lalu jalankan

HISTOGRA per baris. O(R·C). Sekali Anda menguasai bentuk dasarnya, varian dua dimensinya gratis.

```
// Pola nearest-smaller kiri dalam satu lintasan / left nearest-smaller in one pass
stack<int> st;                // indeks, nilai menaik / indices, increasing values
for (int i = 0; i < n; i++) {
  while (!st.empty() && h[st.top()] >= h[i]) st.pop();
  L[i] = st.empty() ? -1 : st.top();   // tetangga kiri < h[i] / left neighbor < h[i]
  st.push(i);
}
```

4.3 Studi Kasus HARD: HISTOGRA (SPOJ #1805)

Kartu soal

SPOJ HISTOGRA | Searching & Sorting | Hard | Monotonic Stack

Sumber: https://www.spoj.com/problems/HISTOGRA/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Sebuah histogram terdiri atas n ≤ 100.000 batang selebar 1 dengan tinggi hi ≤ 10^9. Cari luas persegi panjang terbesar yang muat sepenuhnya di dalam histogram (sisi-sisinya sejajar sumbu, alasnya di garis dasar). Input berisi banyak baris uji; baris berawalan 0 mengakhiri input.

- Tiap baris: n lalu n tinggi. Jawaban bisa melebihi 32-bit — pakai 64-bit.

Contoh masukan/keluaran resmi

```
Masukan:
7 2 1 4 5 1 3 3
4 1000 1000 1000 1000
0
Keluaran:
8
4000
```

Fase 1-2 — Pahami dan Modelkan

Persegi panjang optimal selalu 'mentok atas' pada salah satu batang: untuk tiap batang, jawabannya tinggi batang itu kali lebar maksimal ia bisa membentang tanpa terpotong batang yang lebih pendek. Stack monoton menemukan kedua batas itu untuk semua batang dalam satu lintasan.

Fase 3 — Rancang Algoritmanya

1. Jaga stack indeks bertinggi menaik; batang baru yang lebih pendek 'menutup' batang-batang puncak.
2. Saat pop: lebar = i − s.top() − 1 (atau i bila stack kosong); kandidat = tinggi × lebar.

3. Sentinel tinggi 0 di posisi n memaksa semua batang tersisa diproses tanpa kasus khusus.
4. Jumlahkan dalam long long: $10^5 \times 10^9$ melampaui int.

Fase 4 — Analisis Kompleksitas

Setiap indeks push sekali dan pop maksimal sekali — O(N) amortized per baris uji, praktis instan pada batas soal.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <cstdio>
#include <stack>
#include <algorithm>
using namespace std;
long long h[100001];
int main(){
  int n;
  while(scanf("%d",&n),n){
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
    printf("%lld\n",ans);
  }
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

Invarian stack: indeks-indeks dengan tinggi menaik. Saat batang sekarang lebih pendek dari puncak stack, batang puncak 'tertutup': tingginya kali lebarnya adalah kandidat jawaban.

Lebar batang yang di-pop: bila stack kosong, seluruh [0..i); bila tidak, i − s.top() − 1.

Batang penjaga tinggi 0 di posisi n memaksa semua batang tersisa ter-pop — itulah alasan loop berjalan sampai i == n. Kini juga disembunyikan: pembacaan n tinggi batang ke h[], serta penyiapan sentinel (stack<int> s; long long ans=0). Loop stack-dan-sentinel yang sudah disembunyikan sebelumnya adalah keseluruhan algoritmanya — bangun ulang dari petunjuk di atas: push tinggi yang menaik, pop-dan-hitung harga saat batang lebih pendek datang, dan biarkan sentinel tinggi-0 di i==n mengosongkan stack di akhir.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error), diuji terhadap contoh resmi SPOJ, dan di-stress-test dengan brute force pembanding pada ratusan kasus acak — semua identik. Status arsip: ACCEPTED di SPOJ.

4.4 Latihan Sesi 4

1. Lengkapi separuh HISTOGRA (loop pop) lalu stress-test melawan brute $O(N^2)$.
2. Selesaikan varian matriks biner R×C ≤ 2.000 dengan menumpuk histogram per baris.
3. Buktikan di kertas: total operasi pop di seluruh eksekusi ≤ N.

4.R Ringkasan Sesi dan Poin Kemenangan

- Pop adalah momen ilmu: saat elemen keluar, kedua batas terdekatnya diketahui.
- Push sekali + pop sekali = O(N) amortized, apa pun bentuk loop-nya.
- Sentinel 0 menghapus semua kasus tepi.
- HISTOGRA per baris menyelesaikan submatriks maksimal 2-D.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem HISTOGRA).

[4] SESI 5 · SESI PRAKTIKUM (2 JAM)

Praktikum 1 — Struktur Data Hidup: Update Dinamis

Dua jam penuh praktik. Bekal teori Sesi 2-4 diuji pada GSS3 — GSS1 yang arraynya berubah-ubah. Format praktikum meniru ruang lomba: waktu dibatasi, submit dinilai, dan setiap regu wajib menjalankan protokol stress-test sebelum menyatakan solusinya benar (Tabel 5.1).

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Menambahkan point update pada segment tree GSS1 tanpa merusak invarian combine

(2) Menjalankan siklus penuh: implementasi → sample → stress-test → submit

(3) Mengukur dan mencatat waktu Anda per tahap sebagai data latihan

5.1 Rencana 120 Menit

Menit	Aktivitas	Target
0-10	Baca GSS3, tulis model & rencana node	Peta soal di kertas
10-45	Implementasi penuh (lengkapi separuh yang disembunyikan)	Kompilasi bersih + contoh resmi lolos
45-70	Stress-test vs brute $O(N^2)$ + perbaikan	1.000 kasus acak identik
70-85	Submit ke SPOJ; kejar ACCEPTED	Verdict AC tercatat
85-120	Varian: range update ganti-nilai? diskusi lazy propagation	Peta menuju GSS-series lanjutan

Tabel 5.1 — Setiap praktikum meniru ritme lomba sesungguhnya.

Aturan praktikum

Dilarang membuka solusi penuh sebelum menit 70. Petunjuk hanya dari asisten, dan hanya tiga kali per regu — persis penjatahan 'clarification' di lomba.

5.2 Titik Rawan GSS3

- update wajib meng-combine ulang SEMUA node di jalur pulang — satu yang terlewat = WA sunyi yang baru muncul di kasus besar.

- query tidak boleh membaca node yang hanya separuh tercakup tanpa menggabungkan dua sisi.
- Nilai pengganti bisa negatif; make_node harus membiarkan pre/suf/ans negatif — jangan 'max dengan 0'.

5.3 Studi Kasus HARD: GSS3 (SPOJ #1716)

Kartu soal

SPOJ GSS3 | Struktur Datas | Hard | Max-Subarray Segment Tree + Point Update

Sumber: https://www.spoj.com/problems/GSS3/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Seperti GSS1 — subarray berjumlah maksimum pada rentang — tetapi kini barisan bisa BERUBAH: operasi '0 x y' mengganti A[x] menjadi y, dan operasi '1 x y' menanyakan subarray maksimum di dalam [x, y]. N, M ≤ 50.000.

- Node 4-field GSS1 tetap berlaku; yang baru hanyalah jalur update akar-daun.

Contoh masukan/keluaran resmi

```
Masukan:
4
1 2 3 4
4
1 1 3
0 3 -3
1 2 4
1 3 3
Keluaran:
6
4
-3
```

Fase 1-2 — Pahami dan Modelkan

Node empat-field GSS1 tak berubah; operasi baru hanya penggantian nilai satu posisi. Karena satu daun berubah, cukup hitung ulang node di sepanjang jalur akar-daun itu — combine yang sama menjahit kembali seluruh pohon.

Fase 3 — Rancang Algoritmanya

1. update(pos, val): turun ke sisi yang memuat pos, ganti daun dengan make_node(val), combine ulang saat pulang.
2. query tiga-kasus persis GSS1; hasil parsial digabung dengan combine, tidak pernah dibaca mentah.
3. Operasi 0 = update, operasi 1 = query — baca op code dengan teliti.

Fase 4 — Analisis Kompleksitas

O(log N) per operasi; 50.000 operasi × log 50.000 ≈ 10^6 langkah — beberapa milidetik. Kebenaran, bukan kecepatan, yang diuji soal ini.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <cstdio>
#include <algorithm>
using namespace std;
const int MAXN = 50001;
struct Node {
  int sum, pre, suf, ans;
};
Node tree[4*MAXN];
int a[MAXN], N;
Node make_node(int v) {
  Node n;
  n.sum = n.pre = n.suf = n.ans = v;
  return n;
}
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
int main() {
  scanf("%d", &N);
  for (int i = 1; i <= N; i++) scanf("%d", &a[i]);
  build(1, 1, N);
  int M;
  scanf("%d", &M);
  while (M--) {
    int op, x, y;
    scanf("%d %d %d", &op, &x, &y);
    if (op == 0) update(1, 1, N, x, y);
    else printf("%d\n", query(1, 1, N, x, y).ans);
  }
  return 0;
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

update: turun ke sisi yang memuat pos (bandingkan dengan m), ganti daun dengan make_node(val), lalu combine ulang di jalan pulang.

query: tiga kasus — rentang penuh (kembalikan node), seluruhnya kiri, seluruhnya kanan; kasus melintasi menggabungkan dua hasil rekursif.

Jangan pernah menjawab dari node parsial tanpa combine — di situlah WA GSS3 paling sering lahir. Kini juga disembunyikan: combine() itu sendiri (gabungan empat rumus — tuliskan ulang mengikuti logika gaya GSS1: sum, pre, suf, ans) dan build() (leaf = make_node(a[l]), internal = combine dari dua anak). update() dan query() sudah disembunyikan sejak edisi asli dan tetap disembunyikan di sini — bangun ulang keduanya memakai petunjuk di atas.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error), diuji terhadap contoh resmi SPOJ, dan di-stress-test dengan brute force pembanding pada ratusan kasus acak — semua identik. Status arsip: ACCEPTED di SPOJ.

5.4 Checklist Kelulusan Praktikum

- [ ] separuh kode dilengkapi tanpa melihat solusi penuh
- [ ] Contoh resmi + 1.000 stress-test lolos
- [ ] ACCEPTED di SPOJ (lampirkan tangkapan verdict)
- [ ] Log waktu per tahap terisi di jurnal latihan

5.R Ringkasan Sesi dan Poin Kemenangan

- Update = jalur akar-daun + combine ulang di setiap langkah pulang.
- Stress-test bukan opsional: ia bagian dari definisi 'selesai'.
- Data waktu latihan hari ini = strategi alokasi waktu Anda di hari-H.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem GSS3).

[4] SESI 6 · SESI MATERI (2 JAM)

Graf I: BFS Ruang-Keadaan & Jalur Terpendek Berbobot Kecil

Banyak soal jalur terpendek OSN tidak menyebut kata 'graf' sama sekali — grid, teka-teki, konfigurasi mesin. Keterampilan pertamanya adalah MEMBACA graf tersembunyi; keterampilan keduanya memilih senjata yang pas: BFS untuk bobot 1, 0-1 BFS untuk bobot {0,1}, Dial untuk bobot kecil terbatas, dan barulah Dijkstra ber-heap untuk kasus umum (Tabel 6.1).

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Memodelkan grid/teka-teki sebagai graf ruang-keadaan

(2) Memilih antara BFS, 0-1 BFS, Dial, dan Dijkstra dari struktur bobot

(3) Mengimplementasikan Dial dengan ember melingkar sebesar bobot maksimum + 1

(4) Menghindari jebakan memori & entri basi pada graf jutaan simpul

6.1 Tangga Senjata Jalur Terpendek

Struktur bobot	Senjata	Kompleksitas
Semua 1	BFS	O(V+E)
Hanya {0,1}	0-1 BFS (deque: 0 ke depan, 1 ke belakang)	O(V+E)
Bulat kecil ≤ C	Dial (ember melingkar C+1)	O(E + V·C)
Umum non-negatif	Dijkstra + heap	O(E log V)
Ada negatif	Bellman-Ford / SPFA	O(V·E)

Tabel 6.1 — Bobot menentukan senjata; jangan bawa heap ke perang ember.

Mengapa Dial menang di CHIPSLL

Grid 2000×2000 = 4 juta simpul, 16 juta sisi, bobot 1..63. Heap: 16 juta operasi log — berat dan boros memori. Dial: jarak berurutan monoton, ember hanya 64 slot yang dipakai melingkar — praktis linear, cache-friendly, dan muat memori.

6.2 Ruang Keadaan: Graf yang Tidak Kelihatan

Simpul bukan selalu 'tempat'. Simpul adalah KEADAAN: (posisi, kunci yang dipegang), (posisi kuda, giliran), (sisa air di dua ember). Sisi adalah aksi yang sah. Begitu keadaan dan aksi terdefinisi, semua teori jalur terpendek langsung berlaku. Latihan terbaik: sebelum koding, tulis eksplisit — apa keadaannya? berapa banyak? apa transisinya? berapa biayanya?

- CHIPSLL: keadaan = sel; biaya masuk sel = 1 + kekuatan isinya — bobot sisi 1..63.
- Grid dibangkitkan dari seed: baca spesifikasi generator SETELITI membaca soal — salah satu digit konstanta = WA total.
- Simpan grid sebagai array datar 1-D (id = baris·LEBAR + kolom): lebih cepat dan hemat dari struktur 2-D.

6.3 Studi Kasus HARD: CHIPSLL (SPOJ #40642)

Kartu soal

SPOJ CHIPSLL | Classical | Hard | Shortest Path (Dial's Algorithm)

Sumber: https://www.spoj.com/problems/CHIPSLL/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Chip harus mencapai Melinda pada grid N×M (sampai 2000×2000, T ≤ 10 kasus). Melangkah ke sel kosong berbiaya 1 detik; sel berisi (air a-z, monster A-Z, pintu 0-9) harus dihancurkan dengan biaya tambahan sesuai kekuatannya (1-62). Isi grid TIDAK diberikan: ia dibangkitkan dari sebuah seed dengan pembangkit pseudorandom yang didefinisikan soal. Cari total waktu minimum.

- Bobot sisi kecil dan terbatas (1..63) pada graf 4 juta simpul — sinyal untuk Dial, bukan Dijkstra ber-heap.

Fase 1-2 — Pahami dan Modelkan

Grid adalah graf berbobot kecil: masuk sel berbiaya 1 + kekuatan isinya (1..63). Grid dibangkitkan dari seed persis mengikuti resep soal, lalu jarak terpendek dihitung dengan Dijkstra ember (Dial) — ember melingkar sebanyak bobot maksimum + 1.

Fase 3 — Rancang Algoritmanya

1. Bangkitkan pwv[sel] = 1 + kekuatan dari generator seed; sel awal/tujuan berbiaya 1.
2. Ember: bkt[d mod 64]; maju melingkar ke ember tak kosong berikutnya; entri basi (dist ≠ cur_d) dibuang saat pop.
3. Relaksasi empat arah: nd = cur_d + pwv[tetangga]; bila lebih kecil dari dist lama, catat dan dorong ke bkt[nd mod 64].
4. Array datar 1-D + unsigned char untuk biaya: 4 juta sel muat cache dan memori.

Fase 4 — Analisis Kompleksitas

O(E + V·W) dengan W = 64: praktis linear pada 4 juta simpul dan 16 juta relaksasi — jauh lebih ringan daripada heap O(E log V), dan itulah selisih antara TLE dan AC di soal ini.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;
static const int MAXN = 2021;
static const int MAXCELL = MAXN * MAXN;
static const int W = 64;
static const int INF = 0x3f3f3f3f;
static unsigned char pwv[MAXCELL]; // edge cost to enter cell (1 + power)
static int dist_flat[MAXCELL];
static long long seed_g;
static void update_seed(){
  seed_g = (seed_g * 1000003LL + 10007LL) % 1000000009LL;
}
static void gen_pwv(int N, int M, int Cx, int Cy, int Mx, int My){
  for(int i=1;i<=N;++i)
  for(int j=1;j<=M;++j){
    int id=i*MAXN+j;
    if((i==Cx&&j==Cy)||(i==Mx&&j==My)){pwv[id]=1;continue;}
    update_seed();
    int p=seed_g%63;
    // power(cell) == p exactly (water p<=26, monster p<=52, door p<=62)
    pwv[id]=(unsigned char)(1+p);
  }
}
static vector<int> bkt[W];
static int solve(int N, int M, int sr, int sc, int tr, int tc){
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
int main(){
  int T; scanf("%d",&T);
  for(int t=1;t<=T;t++){
    int N,M,Cx,Cy,Mx,My;
    scanf("%d%d%d%d%d%d",&N,&M,&Cx,&Cy,&Mx,&My);
    scanf("%lld",&seed_g);
    gen_pwv(N,M,Cx,Cy,Mx,My);
    printf("Case %d: %d\n",t,solve(N,M,Cx,Cy,Mx,My));
  }
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

Dial = Dijkstra dengan ember: bucket[d mod W] menampung simpul berjarak d; W cukup sebesar bobot maksimum + 1 (di sini 64).

Loop utama: cari ember tak kosong berikutnya (maju melingkar), pop simpulnya; buang entri basi (dist ≠ cur_d), berhenti saat target keluar.

Relaksasi 4 arah memakai lambda relax(ni): nd = cur_d + biaya masuk sel; bila lebih kecil, catat dan dorong ke bucket[nd mod W]. Kini seluruh isi solve() disembunyikan, termasuk penyiapan yang sebelumnya tampak (reset array jarak, pembersihan bucket, menaruh node awal pada jarak 0). Bangun ulang loop algoritma Dial dari petunjuk di atas: maju melalui bucket yang tidak kosong secara berurutan, buang pop yang basi, relaksasi ke 4 tetangga ke dalam bucket[jarak_baru mod W], dan berhenti begitu target ter-pop.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error) dan diuji jalan (smoke test). Status arsip: ACCEPTED di SPOJ; contoh uji resmi soal ini tidak dipublikasikan utuh, jadi verifikasi akhir adalah submit ulang ke SPOJ — jadikan itu bagian dari latihan Anda.

6.4 Latihan Sesi 6

1. Lengkapi loop ember Dial pada CHIPSLL (separuh tersembunyi), lalu uji dengan grid kecil yang digenerate manual.
2. Tulis 0-1 BFS untuk varian: menghancurkan sel apa pun berbiaya 1, melangkah 0.
3. Modelkan sebagai ruang-keadaan lalu selesaikan: kuda catur dari a1 ke h8 dengan k kotak terlarang.

6.R Ringkasan Sesi dan Poin Kemenangan

- Baca struktur bobot sebelum memilih algoritma — tabel senjata di kepala.
- Dial: ember melingkar sebesar bobot maks + 1; buang entri basi saat pop.
- Ruang keadaan mengubah teka-teki menjadi graf; definisikan (keadaan, aksi, biaya) di kertas dulu.
- Grid raksasa: array datar + tipe kecil (unsigned char) = selamat dari MLE.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem CHIPSLL).

[4] SESI 7 · SESI MATERI (2 JAM)

Graf II: DSU, MST & Teknik Sapuan

Kruskal mengajarkan MST sebagai 'urutkan semua sisi'. Tetapi soal HARD sering punya sisi terlalu banyak untuk dibuat — n^2 pasangan desa VILUNIT, misalnya. Juara membalik urutan berpikir: karena bobot hanya {0,1,2}, proses bobot termurah dulu memakai geometri (sapuan) untuk menemukan sisi yang perlu saja, dan biarkan DSU merangkum sisanya.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Menggunakan DSU dengan path compression sebagai mesin komponen

(2) Menjalankan Kruskal per-kelas-bobot tanpa membuat semua sisi

(3) Menemukan sisi relevan lewat sapuan terurut (sweep) pada proyeksi X/Y

(4) Menutup sisa komponen dengan rumus biaya 2·(komponen − 1)

7.1 DSU: Struktur 5 Baris yang Memenangkan Medali

```
int find(int x){ return par[x]==x ? x : par[x]=find(par[x]); }
bool unite(int a,int b){ a=find(a); b=find(b);
  if(a!=b){ par[a]=b; return true; } return false; }
```

- Path compression saja sudah nyaris O(α(n)) amortized dalam praktik lomba — union by rank opsional.
- unite yang mengembalikan bool adalah kunci Kruskal: true = sisi dipakai (pohon bertambah), false = sisi mubazir.
- Hitung komponen akhir: banyaknya i dengan par[i] == i setelah semua find dipanggil.

7.2 Kruskal Tanpa Daftar Sisi: Kelas Bobot + Sapuan

Bila bobot hanya sedikit nilai (di VILUNIT: 0, 1, 2), Kruskal menyederhana menjadi tiga gelombang: satukan semua pasangan berbobot 0, lalu berbobot 1, lalu tutup dengan 2. Gelombang 0 dan 1 tidak butuh daftar sisi n^2: cukup URUTKAN objek dan sapu — pasangan yang relevan selalu bertetangga dalam urutan. Interval yang saling menjangkau membentuk rantai; menyatukan tiap objek dengan wakil rantainya menghasilkan komponen yang sama dengan menyatukan semua pasangan, dengan biaya O(n log n).

Mengapa cukup satukan dengan 'wakil'?

Relasi 'proyeksinya beririsan' pada interval bersifat menular lewat rantai: bila A menjangkau B dan B menjangkau C dalam urutan sapuan, komponen {A,B,C} tetap terbentuk walau A dan C tidak beririsan langsung — DSU mengakumulasi rantai itu dengan n−1 unite, bukan n^2 pemeriksaan.

- Sapuan bobot-1: pelihara cur_max (ujung kanan terjauh) dan wakilnya; objek yang mulai sebelum cur_max → unite (+1 biaya).
- Ulangi X-sweep dan Y-sweep sampai tidak ada perubahan: satu unite bisa membuka peluang unite lain.
- Penutup: setiap pasangan bebas berbobot 2 → total + 2·(komponen − 1).

7.3 Studi Kasus HARD: VILUNIT (SPOJ #42526)

Kartu soal

SPOJ VILUNIT | Classical | Hard | MST with Fixed Edge Weights via Sweep

Sumber: https://www.spoj.com/problems/VILUNIT/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Ada n ≤ 5.000 desa berbentuk persegi panjang. Dua desa berbiaya 0 untuk disatukan bila interiornya tumpang tindih, 1 bila proyeksinya beririsan pada salah satu sumbu, dan 2 untuk pasangan lainnya. Cari biaya minimum menyatukan semua desa — MST dengan bobot hanya {0,1,2}, yang bisa diselesaikan tanpa membangun semua sisi.

- Banyak kasus uji; input besar → pembaca cepat (fread) sudah termasuk di kode.

Fase 1-2 — Pahami dan Modelkan

MST dengan bobot hanya {0,1,2}: proses per kelas bobot. Tumpang tindih interior disatukan gratis; keterjangkauan proyeksi X/Y disatukan berbiaya 1 lewat sapuan terurut; sisa komponen ditutup berbiaya 2 per sambungan.

Fase 3 — Rancang Algoritmanya

1. Pass 0: urut menurut X1; untuk tiap u telusuri v selama X1[v] < X2[u]; irisan proyeksi Y → unite gratis.
2. Sapuan bobot-1 (X lalu Y): pelihara ujung kanan terjauh cur_max dan wakilnya; objek mulai sebelum cur_max → unite (+1).
3. Ulangi kedua sapuan sampai tak ada perubahan — satu unite bisa membuka unite berikutnya.
4. Penutup: jawaban = biaya + 2·(komponen tersisa − 1).

Fase 4 — Analisis Kompleksitas

Sort mendominasi: O(n log n) per kasus dengan konstanta kecil; putaran 'changed' dalam praktik hanya beberapa kali. Pembaca fread menjaga input raksasa tetap murah.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;
#define MAXN 5005
#define IN_BUF (1 << 20)
static int X1[MAXN], Y1[MAXN], X2[MAXN], Y2[MAXN], par[MAXN], idxX[MAXN],
idxY[MAXN];
static char ibuf[IN_BUF];
static int ipos, ilen;
static inline int readInt() {
  int x = 0, f = 1;
  while (ipos < ilen && (ibuf[ipos] < '0' || ibuf[ipos] > '9')) {
    if (ibuf[ipos] == '-') f = -1;
    if (++ipos == ilen) {
      ilen = fread(ibuf, 1, IN_BUF, stdin);
      ipos = 0;
      if (ilen <= 0) return 0;
    }
  }
  while (ipos < ilen && ibuf[ipos] >= '0' && ibuf[ipos] <= '9') {
    x = x * 10 + (ibuf[ipos++] - '0');
    if (ipos == ilen) {
      ilen = fread(ibuf, 1, IN_BUF, stdin);
      ipos = 0;
    }
  }
  return x * f;
}
int find(int x) {
  return par[x] == x ? x : par[x] = find(par[x]);
}
bool unite(int a, int b) {
  a = find(a);
  b = find(b);
  if (a != b) {
    par[a] = b;
    return true;
  }
  return false;
}
int main() {
  ilen = fread(ibuf, 1, IN_BUF, stdin);
  int T = readInt();
  while (T--) {
    int n = readInt();
    if (n <= 1) {
      if (n == 1) {
        readInt();
        readInt();
        readInt();
        readInt();
      }
      printf("0\n");
      continue;
    }
    for (int i = 0; i < n; i++) {
      X1[i] = readInt();
      Y1[i] = readInt();
      X2[i] = readInt();
```

```
      Y2[i] = readInt();
      if (X1[i] > X2[i]) swap(X1[i], X2[i]);
      if (Y1[i] > Y2[i]) swap(Y1[i], Y2[i]);
      par[i] = i;
      idxX[i] = i;
      idxY[i] = i;
    }
    sort(idxX, idxX + n, [](int a, int b) {
      return X1[a] < X1[b];
    });
    sort(idxY, idxY + n, [](int a, int b) {
      return Y1[a] < Y1[b];
    });
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
    int comps = 0;
    for (int i = 0; i < n; i++) if (par[i] == i) comps++;
    printf("%lld\n", cost + 2LL * (comps - 1));
  }
  return 0;
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

Pass 0: urutkan menurut X1; untuk tiap u, telusuri v selama X1[v] < X2[u]; bila proyeksi Y juga beririsan, unite(u, v) — sisi berbobot 0.

X-Sweep bobot 1 PERSIS mencerminkan Y-Sweep yang dibiarkan terbuka di bawahnya: jaga cur_max dan wakilnya; bila X1[u] < cur_max, unite dan tambah biaya 1.

Komponen yang tersisa disatukan dengan sisi bobot 2: jawaban akhir cost + 2·(komponen − 1). Kini juga disembunyikan: seluruh pass sapuan-X (sudah disembunyikan sejak edisi asli — urutkan berdasar X1, telusuri selama X1[v] < X2[u], unite saat Y overlap untuk edge berbobot-0) DAN pass sapuan-Y yang tampak di bawahnya (mencerminkan sapuan-X persis, tapi pada proyeksi Y, menambah edge berbobot-1). Kedua pass memakai pola sapuan cur_max/rep yang sama seperti dijelaskan di petunjuk atas.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error) dan diuji jalan (smoke test). Status arsip: ACCEPTED di SPOJ; contoh uji resmi soal ini tidak dipublikasikan utuh, jadi verifikasi akhir adalah submit ulang ke SPOJ — jadikan itu bagian dari latihan Anda.

7.4 Latihan Sesi 7

1. Lengkapi separuh VILUNIT: Pass-0 tumpang tindih dan X-sweep — cermin Y-sweep yang dibiarkan terbuka.
2. Buktikan atau bantah: satu putaran X-sweep + Y-sweep selalu cukup (tanpa loop 'changed'). Cari kontra-contoh.

3. Kerjakan Kruskal klasik pada 10^5 sisi sebagai pemanasan kecepatan: target < 15 menit dari nol sampai AC lokal.

7.R Ringkasan Sesi dan Poin Kemenangan

- DSU 5 baris + unite ber-bool = mesin komponen serba guna.
- Bobot sedikit nilai → Kruskal per gelombang, termurah dulu.
- Sapuan terurut menemukan sisi relevan tanpa n^2 kandidat.
- Sisa komponen ditutup dengan aritmetika, bukan algoritma.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem VILUNIT).

[4] SESI 8 · SESI PRAKTIKUM (2 JAM)

Praktikum 2 — Pohon Hidup: BIT Euler + Binary Lifting

Praktikum graf tingkat lanjut: HBD menggabungkan tiga teknik yang masing-masing sudah Anda kenal — Euler tour, BIT, binary lifting — menjadi satu solusi query jalur dinamis. Kemampuan MERANGKAI teknik adalah tanda kesiapan nasional; dua jam ini melatih persis itu.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Memelihara jumlah nilai jalur akar→simpul dengan BIT atas waktu Euler

(2) Menjawab 'berapa langkah sampai akumulasi ≥ x' dengan binary lifting O(log² n)

(3) Merangkai tiga teknik menjadi satu solusi utuh di bawah tekanan waktu

8.1 Tiga Roda Gigi HBD

- Roda 1 — Euler tour: in[u]/out[u] menjadikan subtree sebuah interval; 'tambah y ke simpul u' = add(in[u], +y) dan add(out[u]+1, −y); maka query(in[w]) = jumlah nilai di jalur akar→w. Trik klasik yang wajib hafal.
- Roda 2 — S(w): jumlah jalur v→u = S(v) − S(parent(u)); semua pertanyaan jalur berubah menjadi selisih dua prefiks.
- Roda 3 — binary lifting: up[u][j] = leluhur 2^j; melompat dari besar ke kecil menemukan simpul batas dalam log n lompatan, tiap lompatan dicek dengan satu query BIT — total log² n.

Pola pikir merangkai

Soal rangkaian tidak dipecahkan sekaligus. Pecah jadi kontrak antar-bagian: 'kalau saya punya S(w) murah, sisanya jadi apa?' — lalu penuhi kontrak itu dengan teknik yang Anda hafal.

8.2 Rencana 120 Menit

Menit	Aktivitas
0-15	Baca HBD; tulis kontrak tiga roda gigi di kertas
15-55	Implementasi + lengkapi separuh (blok jawaban query tipe-1)

Menit	Aktivitas
55-80	Uji pohon kecil buatan tangan (rantai, bintang, seimbang)
80-95	Submit; kejar AC
95-120	Diskusi varian: nilai di SISI, bukan simpul? jalur v→u non-leluhur (via LCA)?

8.3 Studi Kasus HARD: HBD (SPOJ #37921)

Kartu soal

SPOJ HBD | Classical | Hard | Tree + BIT + Binary Lifting

Sumber: https://www.spoj.com/problems/HBD/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Sebuah pohon berakar dengan n ≤ 10^5 simpul; tiap simpul menyimpan nilai. Dua jenis operasi: (2) tambahkan y ke nilai simpul u; (1) diberikan leluhur u dari v dan ambang x — berjalan dari v naik menuju u, berapa banyak simpul yang harus dikunjungi hingga jumlah nilainya mencapai x (atau -1 bila total jalur pun kurang)? q ≤ 10^5 operasi.

- Jumlah nilai di jalur akar-simpul dipelihara dengan BIT atas waktu masuk Euler (add pada in, kurang setelah out).

Fase 1-2 — Pahami dan Modelkan

Jumlah nilai jalur akar→simpul dipelihara BIT atas waktu Euler (tambah di in, kurang setelah out); jumlah jalur v→u adalah selisih dua prefiks. Pertanyaan 'berapa langkah sampai akumulasi ≥ x' dijawab binary lifting: lompat dari pangkat dua besar ke kecil mencari simpul batas.

Fase 3 — Rancang Algoritmanya

1. DFS sekali: in/out, depth, dan tabel leluhur up[u][j] (j < 20).
2. Update nilai simpul u: add(in[u], +y), add(out[u]+1, −y) — S(w) = query(in[w]).
3. Query: bila S(v) − S(parent(u)) < x → -1; selain itu turunkan curr dari v: lompat ke p selama depth(p) ≥ depth(u) dan S(p) > S(v) − x.
4. Jawaban = depth(v) − depth(curr) + 1.

Fase 4 — Analisis Kompleksitas

Tiap query O(log n) lompatan × O(log n) BIT = O($\log^2$ n); 10^5 query ≈ 3×10^7 operasi ringan. I/O manual getchar_unlocked menuntaskan sisa anggaran waktu.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
// #pragma GCC optimize("O3,unroll-loops")
// #pragma GCC target("tune=native")
// #pragma GCC target("avx,popcnt,bmi,bmi2") // As suggested by SPOJ, avoiding
AVX2
// Removed all #pragma GCC target(...) to prevent SIGILL on SPOJ's CPU
#include <iostream>
#include <vector>
using namespace std;
const int MAXN = 100005;
vector<int> adj[MAXN];
int up[MAXN][20];
int depth[MAXN];
int in[MAXN], out[MAXN];
int timer_dfs = 0;
long long bit[MAXN];
int n, q;
// Ultra-fast manual I/O for SPOJ supremacy
inline long long read_ll() {
  long long x = 0; int f = 1; char ch = getchar_unlocked();
  while (ch < '0' || ch > '9') { if (ch == '-') f = -1; ch =
getchar_unlocked(); }
  while (ch >= '0' && ch <= '9') { x = x * 10 + ch - '0'; ch =
getchar_unlocked(); }
  return x * f;
}
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  n = read_ll();
  q = read_ll();
  for (int i = 0; i < n - 1; i++) {
    int u = read_ll(), v = read_ll();
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  // Depth 0 is our dummy out-of-bounds node
  depth[0] = 0;
  dfs(1, 0, 1);
  for (int i = 1; i <= n; i++) {
    long long c = read_ll();
    add(in[i], c);
    add(out[i] + 1, -c);
  }
  for (int i = 0; i < q; i++) {
    int type = read_ll();
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
    } else {
      int u = read_ll();
      long long y = read_ll();
      add(in[u], y);
      add(out[u] + 1, -y);
```

```
    }
  }
  return 0;
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

S(w) = jumlah nilai pada jalur akar→w (query BIT di in[w]). Jumlah jalur v→u = S(v) − S(parent(u)); bila < x, jawab -1.

Cari simpul teratas 'curr' dengan binary lifting dari v: lompat ke leluhur p (tanpa melewati u) selama S(p) > S(v) − x — sisa di bawah p masih kurang dari x.

Jawaban = depth(v) − depth(curr) + 1: banyak simpul dari v sampai curr inklusif. Kini juga disembunyikan: helper BIT add()/query()/get_S() (update-titik dan prefix-sum Fenwick-tree standar, memakai get_S(u) = query(in[u]) untuk membaca jumlah jalur-akar node u), dan dfs() (pembangunan Euler-tour + binary-lifting: timestamp in[u]/out[u], serta up[u][i] = up[up[u][i-1]][i-1] untuk tabel leluhur). Cabang query type==1 sudah disembunyikan sejak edisi asli dan tetap disembunyikan di sini — bangun ulang memakai petunjuk di atas.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error) dan diuji jalan (smoke test). Status arsip: ACCEPTED di SPOJ; contoh uji resmi soal ini tidak dipublikasikan utuh, jadi verifikasi akhir adalah submit ulang ke SPOJ — jadikan itu bagian dari latihan Anda.

8.4 Checklist Kelulusan Praktikum

- [ ] Tiga uji pohon tangan lolos (rantai 5 simpul, bintang, biner seimbang)
- [ ] Kasus -1 (jalur kurang dari x) tertangani
- [ ] Lompatan tidak pernah melewati u (cek depth)
- [ ] ACCEPTED di SPOJ tercatat

8.R Ringkasan Sesi dan Poin Kemenangan

- Subtree = interval Euler; jalur akar→w = prefix BIT — dua identitas yang membuka lusinan soal pohon.
- Binary lifting menjawab pencarian di sepanjang jalur dalam $\log^2 n$.
- Teknik dihafal satu-satu, dimenangkan saat dirangkai.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem HBD).

[4] SESI 9 · SESI MATERI (2 JAM)

DP I: Menghitung Objek Berbeda

DP penghitung adalah senjata wajib OSN: berapa banyak cara, berapa banyak objek berbeda, modulo bilangan besar. Bahayanya satu: MENGHITUNG GANDA. DSUBSEQ mengajarkan pola pikir anti-duplikat paling elegan — kejadian terakhir sebuah karakter menanggung semua duplikasinya — pola yang sama muncul di lusinan soal penghitungan level nasional.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Merumuskan DP penghitung dengan definisi keadaan yang TEPAT satu-makna

(2) Menurunkan rekurensi DSUBSEQ: gandakan lalu kurangi kejadian sebelumnya

(3) Berhitung aman dalam aritmetika modular (negatif, perkalian 64-bit)

(4) Memverifikasi DP penghitung dengan enumerasi brute force kecil

9.1 Definisi Keadaan: Kalimat yang Menyelamatkan

Setiap WA pada DP penghitung berakar pada definisi keadaan yang kabur. Disiplin juara: tulis definisi sebagai KALIMAT LENGKAP sebelum menulis rekurensi. Untuk DSUBSEQ: 'dp[i] = banyaknya subsequence BERBEDA dari prefiks s[1..i], termasuk yang kosong.' Dari kalimat itu, rekurensi bisa dibuktikan — bukan ditebak. Menambah karakter c: setiap subsequence lama boleh diperpanjang dengan c atau tidak → 2·dp[i−1]. Tapi bila c pernah muncul di posisi p, subsequence yang berakhir dengan c dari masa itu terhitung dua kali — tepatnya sebanyak dp[p−1] — kurangi.

```
// Rekurensi DSUBSEQ / the DSUBSEQ recurrence
// dp[0] = 1                          (subsequence kosong / the empty
subsequence)
// dp[i] = 2*dp[i-1] - dp[last[c]-1]  (kurangi bila c pernah muncul / subtract if
c appeared)
// jawab dp[n] mod 1e9+7              (kosong ikut dihitung / empty included)
```

Dua jebakan modular klasik

(a − b) % MOD bisa negatif di C++ — selalu tulis (a − b + MOD) % MOD.

Baca soal CERMAT: 'termasuk kosong' vs 'tidak termasuk' mengubah jawaban tepat satu — dan mengubah verdict dari AC ke WA. Teks resmi DSUBSEQ menghitung yang kosong.

9.2 Pola 'Kejadian Terakhir Menanggung Duplikat'

- Pola yang sama menghitung: string berbeda hasil penghapusan, jalur berbeda dengan label, himpunan bagian berbeda multiset.
- Bentuk umumnya: total transisi − koreksi dari kemunculan terakhir penyebab tabrakan.
- Verifikasi wajib: enumerasi set semua objek untuk n ≤ 12 dan cocokkan — lima menit yang membeli kepastian.

9.3 Studi Kasus HARD: DSUBSEQ (SPOJ #2416)

Kartu soal

SPOJ DSUBSEQ | Dynamic Programming | Hard | Distinct Subsequence Counting

Sumber: https://www.spoj.com/problems/DSUBSEQ/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Untuk T ≤ 100 string huruf kapital (panjang ≤ 100.000), hitung banyaknya subsequence BERBEDA — termasuk subsequence kosong — modulo 10^9+7. 'AGH' adalah subsequence dari 'ABCDEFGH'; 'AHG' bukan.

- dp harus linear; jawaban di-mod dan pengurangan dijaga tetap non-negatif.

Contoh masukan/keluaran resmi

```
Masukan:
2
AAA
ABCDEFG
Keluaran:
4
128
```

Fase 1-2 — Pahami dan Modelkan

dp[i] = banyaknya subsequence berbeda dari prefiks i karakter, termasuk yang kosong. Karakter baru menggandakan; kemunculan sebelumnya dari karakter yang sama menyumbang duplikat sebanyak dp[posisi_itu − 1], yang harus dikurangi.

Fase 3 — Rancang Algoritmanya

1. dp[0] = 1 (subsequence kosong); pindai karakter kiri ke kanan.
2. dp[i] = 2·dp[i−1] (ambil atau tidak karakter baru), lalu − dp[last[c]−1] bila c pernah muncul.
3. last[26] diperbarui setelah dipakai; pengurangan dijaga (x + MOD) % MOD.
4. Jawaban dp[n] — teks resmi menghitung subsequence kosong.

Fase 4 — Analisis Kompleksitas

O(n) per string, n ≤ 10^5, T ≤ 100 — total ≤ 10^7 langkah. Satu-satunya cara gagal adalah salah baca 'termasuk kosong' atau salah higiene modular.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```cpp
#include <cstdio>
#include <cstring>
using namespace std;
const int MOD=1000000007;
long long dp[100002];
int last[26];
int main() {
  int t; scanf("%d",&t);
  while(t--) {
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
    }
    printf("%lld\n",dp[n]%MOD);
  }
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

dp[i] = banyak subsequence berbeda dari prefiks i karakter (termasuk kosong). Menambah karakter c menggandakan: dp[i] = 2·dp[i−1].

Duplikat muncul bila c pernah tampil di posisi p: semua subsequence yang dulu diakhiri c terhitung dua kali — kurangi dp[p−1].

Simpan posisi terakhir tiap huruf di last[26]; jaga hasil pengurangan dengan +MOD sebelum %MOD. Kini juga disembunyikan: pembacaan string dan reset last[]/dp[0]. Rekurensi dp yang sudah disembunyikan sebelumnya adalah keseluruhan algoritmanya — bangun ulang dari petunjuk di atas: dp[i] = 2*dp[i-1], lalu kurangi dp[p-1] saat karakter saat ini terakhir muncul di posisi p, jaga dengan +MOD sebelum %MOD.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Semantik dicek ke teks resmi SPOJ (menghitung subsequence kosong). Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error), diuji terhadap contoh resmi SPOJ, dan di-stress-test dengan brute force pembanding pada ratusan kasus acak — semua identik. Status arsip: ACCEPTED di SPOJ.

9.4 Latihan Sesi 9

1. Lengkapi separuh DSUBSEQ (loop rekurensi), verifikasi 'AAA' → 4 dan 'ABCDEFG' → 128, lalu brute-force n ≤ 12.
2. Turunkan varian: banyaknya subsequence berbeda BERPANJANG k (dp dua dimensi).
3. Buktikan formal dengan induksi bahwa dp[i] bebas duplikat — tulis buktinya lima kalimat.

9.R Ringkasan Sesi dan Poin Kemenangan

- Definisi keadaan = kalimat lengkap; rekurensi dibuktikan darinya.
- Gandakan lalu kurangi dp[last−1]: kejadian terakhir menanggung duplikat.
- (a − b + MOD) % MOD — refleks, bukan pilihan.
- Setiap DP penghitung diverifikasi enumerasi kecil sebelum submit.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem DSUBSEQ).

[4] SESI 10 · SESI MATERI (2 JAM)

DP II: Bitmask & Profile DP

N ≤ 17 adalah kode rahasia juri: 'silakan eksponensial, asal terkendali'. Bitmask DP menjadikan himpunan sebagai indeks array; profile DP menjadikan PENAMPANG grid sebagai keadaan. CFONISMG menggabungkan keduanya — bitmask siswa terpakai × profil C kursi terakhir — contoh sempurna soal penempatan yang menakutkan di mata, mekanis di tangan yang terlatih.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Membaca N ≤ 17-20 sebagai undangan bitmask/eksponensial terkendali

(2) Merancang keadaan gabungan (bitmask, profil) dan transisinya sel-demi-sel

(3) Mengendalikan ruang keadaan dengan map + pemangkasan keadaan tak tercapai

(4) Menaksir jumlah keadaan SEBELUM koding untuk memastikan muat waktu

10.1 Bitmask: Himpunan sebagai Bilangan

- mask & (1<<i): apakah i di himpunan; mask | (1<<i): tambahkan i; mask ^ (1<<i): keluarkan.
- Iterasi subset dari subset: for (int s = m; s; s = (s-1) & m) — total 3^N di seluruh mask, bukan 4^N.
- __builtin_popcount, __builtin_ctz: teman lama yang menghemat baris dan waktu.

Contoh mental wajib: TSP bitmask dp[mask][kota] = rute terpendek mengunjungi himpunan mask dan berakhir di kota — O(2^N · N^2). Setiap soal 'kunjungi semua, sekali saja, minimalkan/hitung' dengan N ≤ 20 adalah kerabatnya.

10.2 Profile DP: Penampang sebagai Keadaan

Mengisi grid sel demi sel (kiri→kanan, atas→bawah), keputusan di sel sekarang hanya berinteraksi dengan tetangga ATAS dan KIRI. Maka cukup mengingat 'penampang': isi C sel terakhir. CFONISMG menyandikannya dalam basis 17 (0 = kosong, 1..16 = siswa) — profil ≤ 17^4 kemungkinan, digabung bitmask 2^{17} siswa terpakai. Kunci efisiensi: simpan HANYA keadaan yang benar-benar tercapai dalam unordered_map, karena mayoritas kombinasi (bitmask, profil) tidak konsisten satu sama lain.

Taksir dulu, koding kemudian

Sebelum menulis satu baris: perkirakan keadaan tercapai × biaya transisi. Di CFONISMG: keadaan praktis jauh di bawah $2^{17}\cdot 17^4$ teoretis karena popcount(bitmask) harus cocok dengan jumlah sel terisi. Jika taksiran > 10^8 operasi — desain ulang, jangan berharap.

10.3 Studi Kasus HARD: CFONISMG (SPOJ #42573)

Kartu soal

SPOJ CFONISMG | Classical | Hard | Profile Bitmask DP

Sumber: https://www.spoj.com/problems/CFONISMG/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Sebuah kelas berukuran R×C (C ≤ 4) akan diisi siswa dari daftar N ≤ 17 kandidat, masing-masing boleh dipakai sekali; sel boleh dikosongkan. Sepasang teman yang duduk bersebelahan (atas/kiri) menambah skor 1; sepasang rival — atau dua siswa 'pengganggu' — tidak boleh bersebelahan. Maksimalkan skor total.

- Keadaan DP: (bitmask siswa terpakai, profil C sel terakhir dalam basis 17) — dp berjalan sel demi sel.

Fase 1-2 — Pahami dan Modelkan

Isi kelas sel demi sel. Keadaan = (bitmask siswa terpakai, profil isi C sel terakhir dalam basis 17). Transisi mencoba menempatkan tiap siswa yang tersisa — atau membiarkan kosong — sambil menegakkan larangan rival/pengganggu terhadap tetangga atas dan kiri, dan menambah skor per pertemanan yang terbentuk.

Fase 3 — Rancang Algoritmanya

1. Profil basis 17: profile[0] = tetangga atas sel sekarang; profile[C−1] = tetangga kiri (bila kolom > 0).
2. try_place(s): tolak jika s terpakai / bertetangga rival / dua pengganggu; skor += pertemanan dengan atas & kiri.
3. Kunci baru = (bitmask | bit s) << 17 | (profil digeser + s·17^{C−1}); simpan skor maksimum per kunci di ndp.
4. unordered_map menampung hanya keadaan tercapai; dp = move(ndp) tiap sel.

Fase 4 — Analisis Kompleksitas

Batas teoretis $2^{17}\cdot 17^4$ keadaan tidak pernah tercapai: popcount bitmask terikat jumlah sel terisi. Praktiknya ratusan ribu keadaan × ≤ 18 transisi — di bawah 10^8 operasi ringan.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```cpp
#include <bits/stdc++.h>
using namespace std;
int R,C,N,cells;
bool disruptive[17],is_rival[17][17],is_friend_pair[17][17];
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin>>R>>C>>N;
  cells=R*C;
  map<string,int> id;
  for(int i=1; i<=N; i++) {
    string nm,st;
    cin>>nm>>st;
    id[nm]=i;
    disruptive[i]=(st=="DISRUPTIVE");
  }
  int F;
  cin>>F;
  for(int i=0; i<F; i++) {
    string a,b;
    cin>>a>>b;
    int ia=id[a],ib=id[b];
    is_friend_pair[ia][ib]=is_friend_pair[ib][ia]=true;
  }
  int V;
  cin>>V;
  for(int i=0; i<V; i++) {
    string a,b;
    cin>>a>>b;
    int ia=id[a],ib=id[b];
    is_rival[ia][ib]=is_rival[ib][ia]=true;
  }
  int pw[5]= {1,17,289,4913,83521};
  const int PB=17;
  unordered_map<long long,int> dp;
  dp.reserve(1<<18);
  dp[0]=0;
  for(int cell=0; cell<cells; cell++) {
    int row=cell/C,col=cell%C;
    unordered_map<long long,int> ndp;
    ndp.reserve(dp.size()*4);
    for(auto&[key,score]:dp) {
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
    }
    dp=move(ndp);
  }
  int ans=0;
  for(auto&[k,v]:dp)ans=max(ans,v);
  cout<<ans<<"\n";
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

try_place(s): tolak bila s sudah terpakai di bitmask, atau bila bertetangga (atas = profile[0], kiri = profile[C−1] saat col>0) dengan rival / sesama pengganggu.

Skor bertambah 1 per tetangga (atas, kiri) yang berteman dengan s. Sel kosong = s = 0, selalu sah.

Kunci baru: geser profil basis-17 (buang sel tertua, tambah s), gabungkan dengan bitmask baru; simpan skor maksimum per kunci di ndp. Kini seluruh isi transisi per-state disembunyikan, termasuk pembongkaran yang sebelumnya tampak (memecah key menjadi bitmask + profil basis-17, membaca

murid tetangga atas/kiri). Bangun ulang try_place(s) dari petunjuk di atas: tolak jika s sudah dipakai atau berbenturan dengan tetangga rival/sesama disruptive, score += 1 per tetangga yang berteman, lalu dorong profil+bitmask yang sudah digeser ke ndp sambil menjaga skor maksimum per key.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error) dan diuji jalan (smoke test). Status arsip: ACCEPTED di SPOJ; contoh uji resmi soal ini tidak dipublikasikan utuh, jadi verifikasi akhir adalah submit ulang ke SPOJ — jadikan itu bagian dari latihan Anda.

10.4 Latihan Sesi 10

1. Lengkapi separuh CFONISMG (lambda try_place): validasi rival/pengganggu, skor pertemanan, pembaruan kunci.
2. Tulis TSP bitmask murni untuk N = 16 kota acak; target $O(2^N \cdot N^2)$ berjalan < 1 detik.
3. Hitung eksperimental: berapa keadaan tercapai CFONISMG pada grid 4×4 dengan 16 siswa? Bandingkan dengan batas teoretis.

10.R Ringkasan Sesi dan Poin Kemenangan

- N ≤ 17-20 = izin resmi bermain eksponensial terkendali.
- Profil = penampang C sel terakhir; hanya itu yang dilihat keputusan berikutnya.
- Map keadaan-tercapai memangkas ruang teoretis yang bengkak.
- Taksir operasi sebelum koding — angka menolak lebih murah daripada TLE.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem CFONISMG).

[4] SESI 11 · SESI PRAKTIKUM (2 JAM)

Praktikum 3 — DP Skala Besar: CDQ Divide & Conquer

LIS dua dimensi dengan n = 10^5 membunuh DP kuadratik. Praktikum ini membedah CDQ: pecah barisan di tengah, selesaikan kiri, SALURKAN pengaruh kiri ke kanan dengan sapuan + BIT, lalu selesaikan kanan. Teknik yang sama menjawab banyak 'DP dengan syarat dua dimensi' — salah satu senjata paling kuat menjelang level nasional.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Menjelaskan mengapa urutan solve(kiri) → cross → solve(kanan) wajib

(2) Mengimplementasikan cross(): dua penunjuk atas x + BIT-maksimum atas y

(3) Membersihkan BIT secara selektif agar $\log^2$ tidak membengkak

11.1 Mengapa CDQ Bekerja

dp[i] = LIS-2D terbaik berakhir di i, bergantung pada semua j < i dengan xj < xi dan yj < yi. Tiga dimensi keterurutan (indeks, x, y) sulit ditangani serentak — CDQ menghapus SATU dimensi: di dalam cross(l, mid, r), semua kiri otomatis berindeks lebih kecil dari semua kanan. Tinggal dua dimensi: urutkan kedua sisi menurut x (dua penunjuk), dan atasi y dengan BIT-maksimum atas y terkompresi. Rekursinya T(n) = 2T(n/2) + O(n log n) = O(n $\log^2$ n).

Urutan adalah segalanya

solve(kiri) HARUS selesai sebelum cross, karena cross membaca dp kiri sebagai nilai final; dan solve(kanan) HARUS sesudahnya, karena dp kanan baru lengkap setelah menerima pengaruh kiri. Membalik urutan = jawaban salah yang lolos kasus kecil — jenis bug paling berbahaya.

11.2 Rencana 120 Menit

Menit	Aktivitas
0-15	Baca LIS2; gambar diagram cross di kertas
15-55	Lengkapi separuh (fungsi cross) + kompilasi bersih
55-85	Stress-test vs DP O(n^2) — 40+ kasus acak
85-100	Submit; kejar AC

Menit	Aktivitas
100-120	Diskusi: CDQ untuk 'hitung pasangan dominasi 3-D' & varian bobot

11.3 Studi Kasus HARD: LIS2 (SPOJ #2371)

Kartu soal

SPOJ LIS2 | Dynamic Programming | Hard | CDQ Divide & Conquer

Sumber: https://www.spoj.com/problems/LIS2/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Diberikan n ≤ 100.000 pasangan (x, y) dalam urutan tetap. Cari panjang subsequence terpanjang yang KEDUA koordinatnya naik ketat: xi < xj dan yi < yj untuk setiap pasangan berurutan dalam subsequence — LIS dua dimensi.

- $O(n^2)$ DP mati di $n = 10^5$; CDQ divide & conquer + BIT menurunkannya ke $O(n \log^2 n)$.

Fase 1-2 — Pahami dan Modelkan

dp[i] = LIS-2D terbaik berakhir di titik i. CDQ membelah menurut indeks: selesaikan kiri, salurkan pengaruhnya ke kanan lewat sapuan x dua-penunjuk dengan BIT-maksimum atas y terkompresi, lalu selesaikan kanan.

Fase 3 — Rancang Algoritmanya

1. solve(l, r): solve(l, mid) → cross(l, mid, r) → solve(mid+1, r) — urutan mutlak.
2. cross: urutkan kiri & kanan menurut x; untuk tiap kanan, masukkan dulu semua kiri ber-x lebih kecil ke BIT (posisi y, nilai dp).
3. dp[kanan] = max(dp[kanan], bq(y−1) + 1) — y−1 menegakkan kenaikan ketat.
4. Bersihkan BIT hanya pada posisi yang ditulis (bc) sebelum kembali.

Fase 4 — Analisis Kompleksitas

$T(n) = 2T(n/2) + O(n \log n) = O(n \log^2 n)$: $n = 10^5$ berarti $\pm 3 \times 10^7$ operasi BIT — kira-kira sepersepuluh detik. DP kuadratik pembanding hanya untuk stress-test.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <cstdio>
#include <algorithm>
```

```cpp
using namespace std;
const int MAXN=100001;
int px[MAXN],py[MAXN],dp[MAXN],yc[MAXN],n,yN;
int bit[MAXN];
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
void solve(int l,int r){
  if(l>=r)return;
  int mid=(l+r)/2;
  solve(l,mid);
  cross(l,mid,r);
  solve(mid+1,r);
}
int main(){
  scanf("%d",&n);
  for(int i=0;i<n;i++)scanf("%d%d",&px[i],&py[i]);
  int ys[MAXN];
  for(int i=0;i<n;i++)ys[i]=py[i];
  sort(ys,ys+n);
  yN=unique(ys,ys+n)-ys;
  for(int i=0;i<n;i++)yc[i]=(int)(lower_bound(ys,ys+yN,py[i])-ys)+1;
  for(int i=0;i<n;i++)dp[i]=1;
  solve(0,n-1);
  int ans=0;
  for(int i=0;i<n;i++)ans=max(ans,dp[i]);
  printf("%d\n",ans);
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

cross(l, mid, r) menyalurkan pengaruh separuh kiri ke separuh kanan: urutkan kedua sisi menurut x, lalu jalankan dua penunjuk.

Untuk tiap kanan (urut x naik), masukkan dulu semua kiri dengan x lebih kecil ke BIT (posisi = y terkompresi, nilai = dp); lalu dp[kanan] = max(dp[kanan], bq(y−1) + 1).

Bersihkan BIT hanya pada posisi yang tadi ditulis (bc), bukan memset penuh — di situlah $\log^2$ tetap $\log^2$.

Urutan CDQ wajib: solve(kiri) dulu, cross, baru solve(kanan) — dp kiri harus final sebelum disalurkan. Kini juga disembunyikan: helper BIT bu()/bq()/bc() (update-maks, query-maks, dan clear-titik Fenwick-tree standar) serta deklarasi lb[]/rb[]. cross() sendiri sudah disembunyikan sejak edisi asli dan tetap disembunyikan di sini — bangun ulang memakai petunjuk di atas: two pointer berdasar x, insert-lalu-query BIT berdasar y terkompresi, dan bersihkan hanya posisi yang disentuh.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error), diuji terhadap contoh resmi SPOJ, dan di-stress-test dengan brute force pembanding pada ratusan kasus acak — semua identik. Status arsip: ACCEPTED di SPOJ.

11.4 Checklist Kelulusan Praktikum

- [ ] Diagram cross tergambar dan disetujui asisten
- [ ] Stress-test 40 kasus vs $O(n^2)$ identik
- [ ] Pembersihan BIT selektif (bc), bukan memset penuh
- [ ] ACCEPTED di SPOJ tercatat

11.R Ringkasan Sesi dan Poin Kemenangan

- CDQ menghapus dimensi indeks; sisa dua dimensi ditangani sort + BIT.
- Urutan solve-cross-solve adalah kontrak kebenaran, bukan gaya.
- Pembersihan selektif menjaga $O(n \log^2 n)$ tetap nyata.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem LIS2).

[4] SESI 12 · SESI MATERI (2 JAM)

Meet in the Middle: Membelah Eksponensial

Enam variabel bebas dari himpunan 100 bilangan: 10^{12} kombinasi — mustahil. Tetapi persamaan $a \cdot b + c = d \cdot (e + f)$ punya dua sisi alami, masing-masing hanya $n^3 = 10^6$ nilai. Bangkitkan kiri, bangkitkan kanan, urutkan salah satunya, dan cocokkan: 10^{12} runtuh menjadi 10^6 log 10^6. Itulah meet in the middle — seni membelah ledakan menjadi dua ledakan kecil yang bisa berjabat tangan.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Mengenali struktur 'dua sisi yang bisa dipisah' dalam persamaan/himpunan

(2) Menghitung kemunculan dengan sort + upper_bound − lower_bound

(3) Menaksir memori dan waktu kedua belahan sebelum koding

(4) Menggeneralisasi ke subset-sum n ≤ 40 dan kerabatnya

12.1 Resep Umum MITM

- Pisahkan variabel/objek menjadi dua paruh yang tidak berinteraksi KECUALI lewat satu nilai penghubung.
- Enumerasi tiap paruh; simpan nilai penghubungnya (array untuk dihitung, map bila butuh data ekstra).
- Sortir satu sisi; untuk tiap nilai sisi lain, hitung pasangan dengan biner: upper_bound − lower_bound.
- Subset-sum n = 40: dua paruh 2^{20} = sejuta — dari mustahil menjadi sedetik.

Mengapa d ≠ 0 ditangani saat GENERATE

Menyaring di akhir berarti mengotori array R dengan nilai haram lalu berharap ingat mengecualikannya. Prinsip juara: batasan ditegakkan di tempat ia lahir — loop d melewatkan nol, dan seluruh sisa program tak perlu tahu batasan itu pernah ada.

12.2 Aritmetika yang Menyelamatkan atau Mengubur

- $|a \cdot b + c| \le 30.000^2 + 30.000 \approx 9 \times 10^8$ — muat int, TAPI jumlah pasangannya bisa jauh melampaui 32-bit: jawaban wajib long long.
- 10^6 elemen long long = 8 MB per sisi — cek batas memori soal SEBELUM memilih tipe.
- sort + dua binary search per nilai kiri ≈ $10^6 \times 2 \log(10^6) \approx 4 \times 10^7$ — nyaman di bawah satu detik.

12.3 Studi Kasus HARD: ABCDEF (SPOJ #4580)

Kartu soal

SPOJ ABCDEF | Searching & Sorting | Hard | Meet in the Middle

Sumber: https://www.spoj.com/problems/ABCDEF/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Diberikan himpunan S berisi n ≤ 100 bilangan bulat berbeda (|nilai| ≤ 30.000). Hitung banyaknya tuple (a,b,c,d,e,f) — semua anggota S, boleh berulang — yang memenuhi (a·b + c) / d − e = f dengan d ≠ 0; setara: a·b + c = d·(e + f).

- Enam variabel → 10^{12} kombinasi naif. Belah dua: kiri a·b+c (n^3), kanan d·(e+f) (n^3) — cocokkan.

Contoh masukan/keluaran resmi

```
Masukan:
1
1
Keluaran:
1
```

Fase 1-2 — Pahami dan Modelkan

Tulis ulang persamaan menjadi a·b + c = d·(e + f). Sisi kiri hanya menyentuh (a,b,c), sisi kanan (d,e,f) — dua belahan berukuran n^3 yang bertemu lewat satu nilai penghubung.

Fase 3 — Rancang Algoritmanya

1. Bangkitkan L = semua a·b + c (n^3 nilai, 64-bit).
2. Bangkitkan R = semua d·(e + f) dengan d ≠ 0 dilewati SAAT generate; urutkan R.
3. Jawaban = Σ atas L dari (upper_bound − lower_bound) di R.
4. Akumulator long long: banyaknya pasangan bisa melampaui 32-bit.

Fase 4 — Analisis Kompleksitas

$2 \cdot 10^6$ pembangkitan + sort 10^6 + 10^6 × 2 pencarian biner ≈ 5×10^7 operasi — di bawah sedetik dengan -O2. Memori dua array 8 MB.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#pragma GCC optimize("O3,unroll-loops")
#include <cstdio>
#include <algorithm>
```

```
using namespace std;
int s[101]; long long L[1000001],R[1000001];
int main() {
  int n;scanf("%d",&n);
  for(int i=0;i<n;i++) scanf("%d",&s[i]);
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
  printf("%lld\n",ans);
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

Bangkitkan sisi kanan: untuk tiap d ≠ 0, semua e, f → simpan d·(e+f) di R (n^3 nilai), lalu sort R.

Untuk tiap nilai kiri L[i], banyaknya pasangan = upper_bound − lower_bound pada R — hitungan kemunculan dalam array terurut.

64-bit wajib: |a·b+c| bisa mencapai ~9×10^8, dan jawabannya sendiri bisa melampaui 32-bit. Kini loop pembangkitan array L (sebelumnya tampak) ikut disembunyikan, bersama pembangkitan array R, pengurutan, dan penghitungan yang sudah disembunyikan sebelumnya. Bangun ulang semuanya dari petunjuk di atas: L[i,j,k] = s[i]*s[j]+s[k] untuk tanda targetnya, R dibangkitkan secara cerminnya untuk setiap d != 0, urutkan R, lalu hitung kecocokan per nilai L dengan upper_bound - lower_bound dalam 64-bit.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Rumus dicek ke teks resmi SPOJ; keempat contoh resmi ({1}→1, {2,3}→4, {−1,1}→24, {5,7,10}→10) lolos. Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error), diuji terhadap contoh resmi SPOJ, dan di-stress-test dengan brute force pembanding pada ratusan kasus acak — semua identik. Status arsip: ACCEPTED di SPOJ.

12.4 Latihan Sesi 12

1. Lengkapi separuh ABCDEF (generasi sisi kanan + pencocokan biner); verifikasi keempat contoh resmi.
2. Kerjakan subset-sum n = 38: adakah subset berjumlah tepat S? — MITM murni.
3. Analisis: mengapa memindah e ke sisi kiri (a·b+c−?) merusak simetri n^3/n^3? Tulis dua kalimat.

12.R Ringkasan Sesi dan Poin Kemenangan

- MITM = belah, enumerasi dua kali kecil, jabat tangan lewat nilai penghubung.
- upper_bound − lower_bound menghitung kemunculan pada array terurut.
- Batasan ditegakkan saat generate; jawaban selalu 64-bit.
- n = 40 eksponensial → dua kali 2^{20}: pola yang sama di banyak soal nasional.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem ABCDEF).

[4] SESI 13 · SESI MATERI (2 JAM)

String: Suffix Array & LCP

Satu struktur menjawab hampir semua pertanyaan substring: urutkan semua suffix. Substring = prefiks dari suffix, sehingga suffix array + tabel LCP mengubah pertanyaan 'berapa substring berbeda' menjadi aritmetika satu baris: n(n+1)/2 dikurangi jumlah tumpang tindih antar-tetangga. DISUBSTR adalah pintu masuknya; pintu-pintu berikutnya (pencarian pola, substring terpanjang berulang) memakai kunci yang sama.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Membangun suffix array dengan doubling O(n log² n)

(2) Membangun LCP antar-tetangga dengan algoritme Kasai O(n)

(3) Menurunkan rumus substring berbeda: n(n+1)/2 − Σ lcp

(4) Memetakan soal substring lain ke pasangan (SA, LCP)

13.1 Doubling: Mengurutkan Suffix dengan Rank Berpasangan

Mengurutkan suffix dengan strcmp berbiaya O(n² log n) — mati di n besar. Doubling mengurutkan berdasarkan 2·gap karakter pertama menggunakan PASANGAN rank: (rank[i], rank[i+gap]). Mulai dari rank = karakter, tiap iterasi menggandakan panjang yang sudah terurut, dan rank baru dihitung dengan membandingkan pasangan tetangga. log n iterasi × sort O(n log n) = O(n log² n) — lebih dari cukup untuk n ≤ 10^3 DISUBSTR, dan tetap layak sampai n = 10^5.

- rank[i+gap] di luar string = −1: suffix lebih pendek selalu lebih kecil.
- Berhenti dini saat semua rank unik — penghematan besar di string acak.
- Kasai memakai fakta cantik: lcp suffix i (menurut posisi teks) turun paling banyak 1 dari suffix i−1 — maka h berjalan maju total O(n).

13.2 Dari LCP ke Jawaban

Setiap suffix sa[i] menyumbang semua prefiksnya sebagai substring: n − sa[i] buah. Namun lcp[i] karakter pertamanya sudah disumbangkan suffix tetangga sebelumnya — duplikat. Maka substring berbeda = Σ (n − sa[i]) − Σ lcp[i] = n(n+1)/2 − Σ lcp[i]. Contoh resmi: 'CCCCC' → 15 − (4+3+2+1) = 5; 'ABABA' → 15 − (3+2+1+0) = 9.

Kamus soal (SA, LCP)

Substring berulang terpanjang = max(lcp). Pencarian pola P = binary search di SA. Substring berbeda ke-k = jalan menyusuri SA dengan sisa kuota. K-th smallest suffix = sa[k]. Satu investasi, empat soal.

13.3 Studi Kasus HARD: DISUBSTR (SPOJ #694)

Kartu soal

SPOJ DISUBSTR | Strings | Hard | Suffix Array + LCP

Sumber: https://www.spoj.com/problems/DISUBSTR/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Untuk T ≤ 20 string (panjang ≤ 1.000), hitung banyaknya SUBSTRING berbeda. Contoh resmi: 'CCCCC' → 5, 'ABABA' → 9.

- Total substring = n(n+1)/2; duplikat dihilangkan lewat LCP antar-suffix bertetangga pada suffix array.

Contoh masukan/keluaran resmi

```
Masukan:
2
CCCCC
ABABA
Keluaran:
5
9
```

Fase 1-2 — Pahami dan Modelkan

Setiap substring adalah prefiks sebuah suffix. Urutkan semua suffix (suffix array), hitung tumpang tindih antar-tetangga (LCP), dan substring berbeda = n(n+1)/2 − Σ lcp.

Fase 3 — Rancang Algoritmanya

1. Doubling: urutkan menurut pasangan (rank[i], rank[i+gap]); di luar string bernilai −1; gandakan gap.
2. Rank baru: +1 dari tetangga bila pasangan kuncinya berbeda; berhenti dini saat semua unik.
3. Kasai: proses posisi teks berurutan, h turun maksimal 1 — lcp lengkap dalam O(n).
4. Jawaban long long: n(n+1)/2 bisa melampaui int untuk n besar.

Fase 4 — Analisis Kompleksitas

O(n $\log^2$ n) membangun SA untuk n ≤ 1.000 adalah mikro-detik; T ≤ 20 kasus tidak mengubah apa pun. Rumusnya yang membawa seluruh nilai soal.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int MAXN = 1001;
int sa[MAXN], rk[MAXN], tmp[MAXN], lcp[MAXN];
int n;
char s[MAXN];
void build_sa() {
  for (int i = 0; i < n; i++) {
    sa[i] = i;
    rk[i] = s[i];
  }
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
  }
}
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
int main() {
  int t;
  scanf("%d", &t);
  while (t--) {
    scanf("%s", s);
    n = strlen(s);
    build_sa();
    build_lcp();
    long long total = (long long)n*(n+1)/2;
    for (int i = 1; i < n; i++) total -= lcp[i];
    printf("%lld\n", total);
  }
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

Doubling: pada iterasi gap, urutkan suffix menurut pasangan (rank[i], rank[i+gap]); rank i+gap di luar string = −1.

Rank baru: tmp[sa[0]] = 0; tmp[sa[i]] = tmp[sa[i−1]] + 1 bila pasangan kunci berbeda dari tetangga, selain itu sama.

Berhenti dini saat semua rank unik (rk[sa[n−1]] == n−1). Jawaban akhir: n(n+1)/2 − Σ lcp[i]. Kini juga disembunyikan: build_lcp() (algoritma Kasai) secara utuh — bangun ulang dari petunjuk di atas: pertahankan h dari suffix sebelumnya (jangan pernah direset ke 0), telusuri rank[i] untuk menemukan suffix tepat sebelumnya dalam urutan terurut, perpanjang prefix bersama selama karakter cocok, catat, lalu kurangi h paling banyak satu untuk i berikutnya. Inti doubling di dalam build_sa() sudah disembunyikan sejak edisi asli dan tetap disembunyikan di sini.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error), diuji terhadap contoh resmi SPOJ, dan di-stress-test dengan brute force pembanding pada ratusan kasus acak — semua identik. Status arsip: ACCEPTED di SPOJ.

13.4 Latihan Sesi 13

1. Lengkapi separuh DISUBSTR (loop doubling); verifikasi CCCCC → 5, ABABA → 9, lalu brute-force semua string panjang ≤ 8.
2. Pakai SA+LCP yang sama untuk mencari substring berulang terpanjang string 'BANANA'. Jawab di kertas dulu.
3. Tantangan: jawab 'substring berbeda ke-k menurut leksikografis' dengan berjalan di SA.

13.R Ringkasan Sesi dan Poin Kemenangan

- Substring = prefiks suffix; urutkan suffix dan dunia substring terbuka.
- Doubling: pasangan rank menggandakan panjang terurut per iterasi.
- Kasai O(n): lcp turun maksimal 1 antar posisi teks berurutan.
- Berbeda = n(n+1)/2 − Σ lcp — satu baris yang memenangkan DISUBSTR.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem DISUBSTR).

[4] SESI 14 · SESI MATERI (2 JAM)

Matematika Kompetitif: Selisih Hingga, Interpolasi & Aritmetika Modular

Beberapa soal 'menghitung penempatan' paling menakutkan runtuh oleh satu pengamatan: jawabannya polinomial dalam N. Deteksinya mekanis — selisih hingga yang menjadi konstan — dan begitu derajatnya diketahui, brute force pada beberapa titik kecil + interpolasi Lagrange menghasilkan rumus tertutup. BATTLECRY (skakmat dua benteng di papan N×N) memamerkan pipeline penuh: brute kecil → deteksi derajat 5 → interpolasi → Horner modular O(1) per query.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Mendeteksi jawaban polinomial lewat tabel selisih hingga

(2) Membangun rumus tertutup dengan interpolasi Lagrange dari titik brute force

(3) Mengevaluasi polinomial modular dengan Horner + invers modular

(4) Mewaspadai jebakan: kasus kecil di luar pola (di sini N < 3)

14.1 Selisih Hingga: Stetoskop Polinomial

Ambil f(3), f(4), ..., f(10) dari brute force. Hitung baris selisih: Δf = f(k+1) − f(k); ulangi. Jika baris ke-d konstan (dan ke-(d+1) nol), f adalah polinomial derajat d pada domain itu. Untuk BATTLECRY, selisih kelima konstan 480 → derajat 5 → cukup 6 titik untuk menentukan f sepenuhnya. Ini alat diagnosis pertama juara pada soal hitung-penempatan dengan satu parameter.

- Brute force kecil BUKAN buangan: ia data. Tulis brute jujur $O(N^6)$ untuk N ≤ 8, tabelkan, diagnosis.
- Interpolasi menghasilkan koefisien pecahan? Kalikan penyebut bersama (di sini 3), evaluasi, lalu bagi kembali dengan invers modular.
- Waspadai domain: pola polinomial sering baru berlaku mulai N tertentu — N < 3 bernilai 0 di BATTLECRY.

14.2 Horner & Invers Modular: Evaluasi Kelas Juara

```
// Evaluasi p(v) = 12v^5 - 56v^4 + 66v^3 + 554v^2 - 2292v + 2424 (mod M)
// Horner: (((((12v - 56)v + 66)v + 554)v - 2292)v + 2424)
// 5 perkalian, 5 penjumlahan — dan SETIAP langkah di-mod / 5 mults, 5 adds —
EVERY step reduced
// Pembagian oleh 3 = perkalian INV3 = pow(3, M-2, M) (Fermat, M prima)
```

Tiga dosa aritmetika modular

Lupa +MOD sebelum % pada koefisien negatif; mengalikan dua long long yang sudah dekat 2^{63} tanpa reduksi antara; membagi langsung dengan '/' alih-alih invers modular. Ketiganya lolos kasus kecil dan meledak di kasus juri.

14.3 Studi Kasus HARD: BATTLECRY (SPOJ #40192)

Kartu soal

SPOJ BATTLECRY | Classical | Hard | Finite Differences / Lagrange Interpolation

Sumber: https://www.spoj.com/problems/BATTLECRY/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Hitung banyak penempatan Raja Putih, Raja Hitam, dan dua Benteng Putih identik pada papan N×N sehingga kedua raja tidak bersebelahan dan Raja Hitam ter-skakmat — modulo 7.340.033, untuk banyak nilai N ($N < 10^5$). Format keluaran: 'Case k: jawaban'.

- Untuk $N \geq 3$ jawabannya polinomial derajat 5 dalam N — selisih hingga tingkat 5 konstan; $N < 3$ bernilai 0.

Fase 1-2 — Pahami dan Modelkan

Brute force kecil menunjukkan f(N) polinomial derajat 5 untuk $N \geq 3$ (selisih hingga kelima konstan 480) dan 0 untuk $N < 3$. Interpolasi Lagrange atas enam titik memberi $3 \cdot f(N) = 12N^5 - 56N^4 + 66N^3 + 554N^2 - 2292N + 2424$; jawab per query dengan Horner modular lalu kalikan invers 3.

Fase 3 — Rancang Algoritmanya

1. $N < 3 \rightarrow 0$ langsung; pola polinomial baru berlaku dari $N = 3$.
2. Horner lima langkah: mulai $12v - 56$, tiap langkah kali v tambah koefisien, semua di-mod 7.340.033.
3. Koefisien negatif diamankan +MOD sebelum %.
4. Tutup dengan × INV3 (= pow(3, MOD−2)) — pembagian modular yang sah.

Fase 4 — Analisis Kompleksitas

O(1) per query — lima perkalian modular. Tidak ada larik, tidak ada prakomputasi; seluruh kecerdasan soal sudah dipindahkan ke tahap analisis di atas kertas.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <stdio.h>
typedef long long ll;
#define MOD 7340033LL
#define INV3 2446678LL /* pow(3, MOD-2, MOD) = 2446678 */
static ll solve(ll n) {
  ll v, h;
  if (n < 3) return 0LL;
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
}
int main(void) {
  int T, t;
  ll N;
  scanf("%d", &T);
  for (t = 1; t <= T; t++) {
    scanf("%lld", &N);
    printf("Case %d: %lld\n", t, solve(N));
  }
  return 0;
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

$3 \cdot f(N) = 12N^5 - 56N^4 + 66N^3 + 554N^2 - 2292N + 2424$ — evaluasi dengan Horner: mulai dari 12v−56, kalikan v dan tambahkan koefisien berikutnya bergiliran, semua bermodulo.

Koefisien negatif dijaga positif dengan +MOD sebelum %MOD.

Akhiri dengan mengalikan invers modular 3 (INV3 = 2446678) untuk membagi 3 secara sah di aritmetika modular. Kini seluruh isi solve() setelah base case disembunyikan: baris reduksi v=n%MOD, komentar rumus, dan evaluasi Horner itu sendiri. Bangun ulang dari petunjuk di atas: mulai dari 12v-56, kalikan berulang dengan v dan tambahkan koefisien berikutnya (66, 554, -2292, 2424 berurutan, masing-masing dijaga dengan +MOD sebelum %MOD), lalu kalikan dengan INV3 untuk membagi 3 di bawah modulus.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error) dan diuji jalan (smoke test). Status arsip: ACCEPTED di SPOJ; contoh uji resmi soal ini tidak dipublikasikan utuh, jadi verifikasi akhir adalah submit ulang ke SPOJ — jadikan itu bagian dari latihan Anda.

14.4 Latihan Sesi 14

1. Lengkapi separuh BATTLECRY (Horner + INV3); uji f(3) = 232 sesuai tabel brute force di soal.
2. Tulis brute force N ≤ 8 versi Anda sendiri dan reproduksi tabel selisih sampai baris konstan 480.
3. Latihan interpolasi: f polinomial derajat 3 dengan f(1..4) = 1, 5, 14, 30 — temukan f(10) tanpa menemukan koefisien eksplisit (Lagrange langsung).

14.R Ringkasan Sesi dan Poin Kemenangan

- Selisih hingga konstan di baris d = polinomial derajat d — diagnosis mekanis.
- Brute kecil adalah data; interpolasi mengubahnya jadi rumus O(1).
- Pembagian modular = perkalian invers (Fermat pada modulus prima).
- Selalu cek domain pola: kasus kecil di luar pola dijawab terpisah.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem BATTLECRY).

[4] SESI 15 · SESI PRAKTIKUM (2 JAM)

Praktikum 4 — Simulasi Kontes Penuh + Soal Pamungkas Max Flow

Sesi penutup: gladi bersih lomba sesungguhnya. Dua jam, aturan OSN penuh, dan satu soal pamungkas — FASTFLOW, aliran maksimum yang menuntut Dinic yang benar DAN cepat. Semua playbook Sesi 1 dieksekusi: pemetaan, alokasi waktu, subtask, stress-test, dan keputusan dingin di menit-menit akhir. Inilah cermin kesiapan nasional Anda.

Sasaran sesi — setelah 2 jam ini Anda mampu:

(1) Mengeksekusi playbook Sesi 1 utuh dalam simulasi berdurasi penuh

(2) Mengimplementasikan Dinic: BFS level + DFS blocking flow + current-arc

(3) Menangani graf tak berarah, sisi ganda, dan kapasitas 64-bit dengan benar

(4) Menilai diri dengan rubrik juara dan menyusun rencana latihan pasca-TC

15.1 Dinic dalam Empat Kalimat

(1) BFS dari sumber memberi level[v] = jarak sisi-positif; berhenti bila sink tak tercapai. (2) DFS hanya boleh maju ke level+1 — inilah 'level graph' yang mencegah putaran. (3) current-arc iter_[v] mengingat sisi mana yang sudah gagal, sehingga tiap sisi diperiksa O(1) kali per fase — tanpa ini Dinic berubah TLE. (4) Ulangi BFS+blocking-flow sampai sink tak terjangkau; jumlah fase O(V), praktik jauh lebih kecil.

- Tak berarah: kedua arah berkapasitas penuh c — pasangan residual saling menjadi sisi balik.
- Sisi ganda dibiarkan apa adanya (jangan digabung — tidak perlu); gelung (u = v) dibuang.
- Semua aliran long long: kapasitas 10^9 × 30.000 sisi menghancurkan int (Tabel 15.1).

15.2 Rubrik Penilaian Simulasi

Aspek	Bukti kelulusan
Pemetaan (menit 0-10)	Peta tertulis: model, batasan, rencana
Implementasi	Dinic penuh, kompilasi bersih, contoh resmi lolos (output 5)
Verifikasi	Cross-check vs max-flow referensi pada graf acak kecil
Manajemen waktu	Log fase terisi; aturan 45 menit dihormati

Aspek	Bukti kelulusan
Verdict	ACCEPTED di SPOJ

Tabel 15.1 — Lima aspek, lima bukti; juara memenuhi semuanya.

Pesan penutup pelatih

Setelah TC ini: satu kontes virtual per minggu, satu soal HARD per hari, dan jurnal kesalahan yang terus hidup. Juara 1 nasional adalah akumulasi disiplin kecil yang tidak pernah libur. Sampai jumpa di podium.

15.3 Studi Kasus HARD: FASTFLOW (SPOJ #4110)

Kartu soal

SPOJ FASTFLOW | Graph Algorithms | Hard | Dinic's Algorithm

Sumber: https://www.spoj.com/problems/FASTFLOW/ — kerjakan dan submit langsung ke SPOJ.

Pernyataan soal (ringkas, setia pada teks asli SPOJ)

Diberikan graf TAK berarah dengan N ≤ 5.000 simpul dan M ≤ 30.000 sisi berkapasitas (hingga 10^9, bisa ada sisi ganda dan gelung). Hitung aliran maksimum dari simpul 1 ke simpul N. Judul soal adalah tantangannya: harus CEPAT — Dinic dengan level graph dan current-arc.

- Sisi tak berarah = kapasitas penuh dua arah pada pasangan sisi residual; gelung dibuang.
- Jawaban bisa melampaui 32-bit — long long di semua tempat aliran.

Contoh masukan/keluaran resmi

```
Masukan:
4 6
1 2 3
2 3 4
3 1 2
2 2 5
3 4 3
4 3 3
Keluaran:
5
```

Fase 1-2 — Pahami dan Modelkan

Aliran maksimum 1 → N pada graf tak berarah: setiap sisi diberi kapasitas penuh di kedua arah dan pasangannya saling menjadi sisi balik residual. Dinic — BFS level, DFS blocking flow, current-arc — memenuhi tuntutan kecepatan judul soal.

Fase 3 — Rancang Algoritmanya

1. add_edge dua arah berkapasitas c; gelung u = v dibuang; sisi ganda dibiarkan.
2. BFS memberi level; DFS hanya maju ke level+1 dan memangkas lewat iterator current-arc per simpul.
3. Saat menemukan aliran d: kurangi cap sisi, tambah cap sisi baliknya, kembalikan d ke atas.
4. Ulangi fase sampai BFS gagal mencapai sink; jumlahkan dalam long long.

Fase 4 — Analisis Kompleksitas

Dinic berjalan $O(V^2E)$ teoretis tetapi jauh lebih cepat di graf nyata; dengan current-arc, FASTFLOW selesai dalam hitungan ratusan milidetik. Tanpa current-arc, kompleksitas yang sama di atas kertas berubah TLE di mesin.

Fase 5 — Implementasi (Edisi Publik — ~50% dibuka, ~50% tugas Anda)

Ini adalah Edisi Publik dari solusi ini. Kira-kira separuh kode ditunjukkan di bawah; sisanya — bagian yang benar-benar menyelesaikan soal — adalah tugas Anda, dipandu oleh petunjuk di bawah kode dan penjabaran di Fase 1-4 di atas.

```
#include <cstdio>
#include <vector>
#include <queue>
#include <cstring>
#include <algorithm>
using namespace std;
const int MX = 5001;
const long long INF = 1e18;
struct Edge {
  int to, rev;
  long long cap;
};
vector<Edge> graph[MX];
int level[MX], iter_[MX], n, m;
void add_edge(int u, int v, long long c) {
  // Undirected: full capacity c in both directions
  graph[u].push_back({v, (int)graph[v].size(), c});
  graph[v].push_back({u, (int)graph[u].size()-1, c});
}
/* ================================================================
   >>> EDISI PUBLIK — kira-kira separuh solusi ini disembunyikan untuk Anda
lengkapi. <<<
Bangun ulang memakai petunjuk di bawah kode dan penalaran di Fase 1-4 di atas.
   ================================================================ */
int main() {
  scanf("%d %d", &n, &m);
  for (int i = 0; i < m; i++) {
    int u, v;
    long long c;
    scanf("%d %d %lld", &u, &v, &c);
    if (u != v) add_edge(u, v, c); // skip self-loops
  }
  printf("%lld\n", max_flow(1, n));
}
```

Petunjuk untuk separuh yang hilang — cukup untuk juara, tidak lebih

dfs blocking-flow: hanya lewat sisi dengan cap > 0 dan level[v] < level[e.to]; kembalikan aliran yang benar-benar sampai ke t.

Current-arc: iterator per simpul (int& i = iter_[v]) TIDAK di-reset dalam satu fase — inilah yang menjadikan Dinic cepat; reset hanya setelah BFS baru.

Saat menemukan d > 0: kurangi e.cap, tambah kapasitas sisi balik graph[e.to][e.rev], kembalikan d; fase luar mengulang dfs sampai 0, lalu BFS lagi. Kini juga disembunyikan: bfs() itu sendiri (BFS level-graph standar — level[s]=0, relaksasi setiap edge dengan cap>0 dan level[e.to]<0). dfs() (pencarian blocking-flow) dan max_flow() (loop penggerak luar BFS-lalu-dfs berulang) sudah disembunyikan sejak edisi asli dan tetap disembunyikan di sini — bangun ulang keduanya memakai petunjuk di atas.

Log verifikasi — kode ini diperiksa ulang sebelum dicetak

Kode lengkap dikompilasi ulang dengan g++ -O2 -std=c++17 (0 error), diuji terhadap contoh resmi SPOJ, dan di-stress-test dengan brute force pembanding pada ratusan kasus acak — semua identik. Status arsip: ACCEPTED di SPOJ.

15.4 Checklist Kelulusan Simulasi

- [ ] Peta menit-0-10 dikumpulkan ke pelatih
- [ ] Dinic lengkap (separuh dilengkapi mandiri) + contoh resmi lolos
- [ ] Cross-check aliran pada ≥ 20 graf acak kecil
- [ ] ACCEPTED FASTFLOW tercatat
- [ ] Rencana latihan pasca-TC tertulis dan disepakati

15.R Ringkasan Sesi dan Poin Kemenangan

- Dinic = BFS level + DFS blocking + current-arc; hilangkan satu, hilang pula cepatnya.
- Tak berarah berarti kapasitas penuh dua arah; long long di semua jalur aliran.
- Simulasi dinilai dengan rubrik — kesiapan nasional itu terukur, bukan perasaan.
- Latihan tidak berhenti di sesi 15; ia baru berpindah rumah ke jadwal harian Anda.

Referensi

[1] Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] Halim, S., Halim, F. — Competitive Programming 4, 2020.

[3] SPOJ — Sphere Online Judge, spoj.com (problem FASTFLOW).