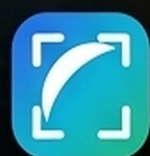


OSN Bidang Informatika

Menuju Juara Nasional — 15 Studi Kasus SPOJ



Irfan Subakti



Profitina

profitina.com

OSN Bidang Informatika: Menuju Juara Nasional

15 Studi Kasus SPOJ

Irfan Subakti



Hak cipta © 2026 Irfan Subakti. Hak cipta dilindungi undang-undang.

Diterbitkan oleh Profitina · profitina.com

Edisi pertama, September 2026.

Setiap program dalam buku ini sudah dikompilasi dan dijalankan sebelum keluarannya dicetak. Kode sumber lengkapnya ada di folder Code yang menyertai buku ini. Program-program tersebut disediakan untuk keperluan belajar-mengajar; Irfan Subakti dan penerbit tidak bertanggung jawab atas penggunaannya di luar keperluan itu.

Daftar Isi

Kata Pengantar	xvi
Tentang Edisi Publik ini	xvi
Untuk siapa buku ini ditulis	xvii
Satu bahasa, program yang teruji	xvii
Isi buku ini	xvii
Cara membacanya	xviii
Tentang penulis	xix
Ucapan terima kasih	xix
 BAB 1 Buku Pedoman Sang Juara: Strategi Lomba dan Membaca dengan Cermat —	
HOTLINE	1
1.1 Surat untuk Calon Finalis	1
1.2 OSN Informatika pada 2026	2
1.2.1 Tahap-tahapnya	2
1.2.2 Cara babak nasional dinilai	2
1.2.3 Sesudah babak nasional: TOKI dan IOI	3
1.2.4 Lingkungan lomba	3
1.3 Meja Kerja dalam Sepuluh Menit	3
1.4 Membaca Batasan seperti Pembuat Soal	4
1.5 Hari Lomba, Menit demi Menit	5
1.5.1 Menit 0 sampai 10: petakan semuanya	5
1.5.2 Poin per menit	5
1.5.3 Bentuk hari yang baik	5
1.5.4 Aturan 45 menit	6
1.5.5 Sebelum hari lomba	6
1.6 Subsoal: Matematika Medali	6
1.7 Pemanasan: Memotong Baris Menjadi Kata	7
1.8 Protokol Debug dan <i>Stress Test</i>	8
1.9 Studi Kasus SULIT: HOTLINE	9
1.9.1 Soal dengan kata-kata kita sendiri	9
1.9.2 Contoh dialog	9
1.9.3 Pemodelan: fakta sebagai kunci	10
1.9.4 Solusi lengkap	10
1.9.5 Baris demi baris	14
1.9.6 Jebakan	15
1.9.7 <i>Stress test</i> melawan <i>brute force</i>	15
1.10 Taktik Hari Lomba	17
1.11 Ringkasan	17
1.12 Soal Latihan	17
1.13 Penyelesaian Soal Latihan	18
Daftar Pustaka	18
 BAB 2 Pohon Segmen dan Jawaban yang Dapat Digabung — GSS1	20
2.1 Buku Kas Koperasi	20
2.2 Gagasannya dengan Kata-Kata Sederhana	20
2.3 Soal Pemanasan	21
2.3.1 Jumlah rentang tanpa pembaruan	21
2.3.2 Blok terbaik seluruh larik	22
2.4 Pohon Segmen	22
2.4.1 Bentuk dan penomoran	22

2.4.2 Membangun dalam waktu linear	23
2.4.3 Kueri dalam waktu logaritmik	23
2.5 Merancang Simpul	24
2.5.1 Resep ketertutupan	24
2.5.2 Empat medan untuk blok terbaik.....	24
2.5.3 Mengapa rumus-rumusnya benar.....	25
2.5.4 Mengerjakan contoh dengan tangan.....	25
2.5.5 Resep yang sama, pertanyaan lain	25
2.6 Studi Kasus SULIT: GSS1	26
2.6.1 Soal dengan kata-kata kita sendiri	26
2.6.2 Contoh masukan dan keluaran	26
2.6.3 Rancangan.....	27
2.6.4 Solusi lengkap.....	27
2.6.5 Baris demi baris.....	28
2.6.6 Jebakan.....	29
2.6.7 <i>Stress test</i>	29
2.7 Ketika Alat Lain Lebih Baik	30
2.8 Taktik Hari Lomba	31
2.9 Ringkasan	31
2.10 Soal Latihan.....	31
2.11 Penyelesaian Soal Latihan.....	32
Daftar Pustaka.....	32
BAB 3 Pohon Segmen Persisten dan Statistik Urutan — MKTHNUM	33
3.1 Median Petugas Pasar.....	33
3.2 Gagassannya dengan Kata-Kata Sederhana	33
3.3 Soal Pemanasan.....	34
3.3.1 Kompresi koordinat.....	34
3.3.2 Mengurangkan dua cacah prefiks.....	34
3.4 Pohon Segmen atas Nilai.....	35
3.5 Persistensi dengan Penyalinan Lintasan.....	36
3.5.1 Salin lintasannya, bagi sisanya	36
3.5.2 Berapa banyak memori.....	37
3.5.3 Mengapa berbagi itu aman	37
3.6 Mengurangkan Dua Versi.....	37
3.7 Studi Kasus SULIT: MKTHNUM	38
3.7.1 Soal dengan kata-kata kita sendiri	38
3.7.2 Contoh masukan dan keluaran	38
3.7.3 Rancangan.....	38
3.7.4 Solusi lengkap.....	39
3.7.5 Baris demi baris.....	40
3.7.6 Jebakan.....	41
3.7.7 <i>Stress test</i>	41
3.7.8 Biaya dalam angka	42
3.8 Kerabat dan Alternatif.....	42
3.9 Taktik Hari Lomba	43
3.10 Ringkasan.....	43
3.11 Soal Latihan.....	43
3.12 Penyelesaian Soal Latihan.....	44
Daftar Pustaka.....	44
BAB 4 Tumpukan Monoton dan Teknik Larik Linear — HISTOGRA	45
4.1 Spanduk di Jalan Utama.....	45
4.2 Gagassannya dengan Kata-Kata Sederhana	45
4.2.1 Dua jalan yang lebih lambat menuju jawaban yang sama	46

4.3 Soal Pemanasan.....	46
4.3.1 Elemen lebih kecil terdekat.....	46
4.3.2 Rentang saham.....	47
4.4 Tumpukan Monoton.....	48
4.4.1 Invariannya	48
4.4.2 Mengapa perulangan di dalam perulangan tetap linear	48
4.4.3 Empat varian, satu pola	49
4.5 Satu Lintasan: Jawaban pada Saat Pengeluaran.....	49
4.5.1 Mengapa satu lintasan menemukan maksimum	51
4.6 Studi Kasus SULIT: HISTOGRA	51
4.6.1 Soal dengan kata-kata kita sendiri.....	51
4.6.2 Contoh masukan dan keluaran	51
4.6.3 Rancangan.....	51
4.6.4 Solusi lengkap.....	52
4.6.5 Baris demi baris.....	52
4.6.6 Jebakan.....	52
4.6.7 <i>Stress test</i>	53
4.7 Dari Histogram ke Matriks	54
4.8 Taktik Hari Lomba	54
4.9 Ringkasan	55
4.10 Soal Latihan.....	55
4.11 Penyelesaian Soal Latihan.....	55
Daftar Pustaka.....	56
BAB 5 Struktur Data yang Hidup: Pembaruan Titik pada Pohon Segmen — GSS3	57
5.1 Koreksi di Buku Kas	57
5.2 Gagasannya dengan Kata-Kata Sederhana	57
5.2.1 Betapa sedikit yang berubah	58
5.3 Pemanasan: Jumlah Rentang dengan Pembaruan.....	58
5.4 Pembaruan Titik.....	60
5.4.1 Algoritmanya	60
5.4.2 Mengapa benar	60
5.4.3 Biaya	60
5.4.4 Alternatif dari bawah ke atas	61
5.5 Siklus Latihan yang Meniru Lomba	61
5.5.1 Apa yang dicatat selama berlatih.....	61
5.6 Studi Kasus SULIT: GSS3	62
5.6.1 Soal dengan kata-kata kita sendiri.....	62
5.6.2 Contoh masukan dan keluaran	62
5.6.3 Rancangan.....	62
5.6.4 Solusi lengkap.....	63
5.6.5 Baris demi baris.....	64
5.6.6 Jebakan.....	64
5.6.7 <i>Stress test</i>	65
5.6.8 <i>Bug</i> yang tertangkap basah	66
5.7 Melampaui Pembaruan Titik.....	66
5.8 Taktik Hari Lomba	67
5.9 Ringkasan	67
5.10 Soal Latihan.....	67
5.11 Penyelesaian Soal Latihan.....	68
Daftar Pustaka.....	68
BAB 6 Graf Ruang Keadaan dan Algoritma Dial — CHIPSLL.....	69
6.1 Seorang Pahlawan, Sebuah Peta, dan Stopwatch	69
6.2 Melihat Graf yang Tersembunyi.....	70

6.2.1 Keadaan, aksi, dan biaya.....	70
6.2.2 Tangga senjata	70
6.2.3 Mengapa Dijkstra benar, dalam satu paragraf.....	71
6.2.4 Algoritma Dial: cincin ember.....	71
6.2.5 Ruang keadaan tanpa peta	73
6.3 Soal Pemanasan.....	73
6.3.1 Kuda catur pada papan dengan petak terlarang.....	73
6.3.2 Menjebol dinding sesedikit mungkin: BFS 0-1	74
6.4 Studi Kasus: CHIPSELL.....	75
6.4.1 Soal dengan kata-kata kita sendiri.....	75
6.4.2 Contoh yang dikerjakan dengan tangan	76
6.4.3 Merancang penyelesaian.....	76
6.4.4 Penyelesaian lengkap.....	77
6.4.5 Baris demi baris.....	79
6.4.6 Jebakan.....	80
6.5 Uji Stres terhadap <i>Brute Force</i>	80
6.6 Taktik pada Hari Lomba	82
6.7 Ringkasan Bab	82
6.8 Soal Latihan.....	83
6.9 Penyelesaian Soal Latihan	83
Daftar Pustaka.....	84
BAB 7 Himpunan Saling Lepas, Algoritma Kruskal, dan Sapuan — VILUNIT	85
7.1 Desa-Desa di Atas Peta.....	85
7.2 Alat-Alatnya dari Nol	86
7.2.1 Gabungan himpunan saling lepas dalam lima baris.....	86
7.2.2 Algoritma Kruskal dan sifat potongan.....	86
7.2.3 Menyapu selang.....	87
7.2.4 Mengapa DSU cukup cepat	88
7.3 Pemanasan: Kruskal pada Graf Kecil	88
7.4 Studi Kasus: VILUNIT.....	89
7.4.1 Soal dengan kata-kata kita sendiri.....	89
7.4.2 Contoh masukan dan keluaran	90
7.4.3 Merancang penyelesaian.....	90
7.4.4 Menelusuri contoh kedua dengan tangan.....	91
7.4.5 Penyelesaian lengkap.....	92
7.4.6 Baris demi baris.....	93
7.4.7 Kompleksitas, secara jujur.....	94
7.4.8 Jebakan.....	94
7.5 Uji Stres terhadap <i>Brute Force</i>	95
7.6 Memakai Gagasan Ini di Luar VILUNIT	96
7.7 Taktik pada Hari Lomba	96
7.8 Ringkasan Bab	97
7.9 Soal Latihan.....	97
7.10 Penyelesaian Soal Latihan.....	97
Daftar Pustaka.....	98
BAB 8 Tur Euler, Fenwick Tree, dan Binary Lifting — HBD	99
8.1 Mengumpulkan Sumbangan ke Atas Pohon Keluarga.....	99
8.2 Tiga Roda Gigi dari Nol	100
8.2.1 Roda gigi 1: tur Euler mengubah subpohon menjadi selang.....	100
8.2.2 Roda gigi 2: <i>Fenwick tree</i> untuk jumlah prefiks dengan pembaruan	101
8.2.3 Jumlah lintasan sebagai selisih.....	102
8.2.4 <i>Fenwick tree</i> dengan tangan.....	102
8.2.5 Roda gigi 3: <i>binary lifting</i>	102

8.3 Pemanasan: Tiga Roda Gigi pada Pohon Kecil	103
8.4 Studi Kasus: HBD	105
8.4.1 Soal dengan kata-kata kita sendiri	105
8.4.2 Contoh masukan dan keluaran	105
8.4.3 Merancang penyelesaian	106
8.4.4 Menelusuri satu pertanyaan	106
8.4.5 Penyelesaian lengkap	107
8.4.6 Baris demi baris	109
8.4.7 Kompleksitas dan jebakan	110
8.5 Uji Stres terhadap <i>Brute Force</i>	110
8.6 Roda Gigi yang Sama, Soal yang Lain	112
8.7 Taktik pada Hari Lomba	113
8.8 Ringkasan Bab	113
8.9 Soal Latihan	113
8.10 Penyelesaian Soal Latihan	114
Daftar Pustaka	114
BAB 9 Menghitung Objek Berbeda dengan Pemrograman Dinamis — DSUBSEQ	115
9.1 Kata-Kata Berbeda dari Satu Kata	115
9.2 Menghitung Tanpa Menghitung Ganda	116
9.2.1 Subbarisan	116
9.2.2 Keadaan, sebagai sebuah kalimat	116
9.2.3 Menurunkan rekurens	116
9.2.4 Dua kekeliruan yang menggoda	117
9.2.5 Penurunan kedua: menghitung menurut huruf terakhir	118
9.2.6 Aritmetika modulo bilangan prima	118
9.3 Soal Pemanasan	118
9.3.1 Menghitung cara: kemunculan ABC	118
9.3.2 Mengamati rekurens bekerja	119
9.3.3 BANANA, kata di papan tulis	120
9.3.4 Dari cara ke untai	120
9.4 Studi Kasus: DSUBSEQ	120
9.4.1 Soal dengan kata-kata kita sendiri	120
9.4.2 Contoh masukan dan keluaran	121
9.4.3 Merancang penyelesaian	121
9.4.4 Penyelesaian lengkap	121
9.4.5 Baris demi baris	122
9.4.6 Mengapa modulus tidak berbahaya	122
9.4.7 Jebakan	123
9.5 Uji Stres terhadap <i>Brute Force</i>	123
9.6 Pola Ini dalam Soal Lain	124
9.7 Taktik pada Hari Lomba	125
9.8 Ringkasan Bab	125
9.9 Soal Latihan	125
9.10 Penyelesaian Soal Latihan	126
Daftar Pustaka	126
BAB 10 Pemrograman Dinamis <i>Bitmask</i> dan Profil — CFONISMG	128
10.1 Menata Tempat Duduk Kelas	128
10.2 Himpunan sebagai Bilangan	129
10.2.1 Operasi bit	129
10.2.2 Teladannya: penjual keliling	130
10.3 Profil: Penampang Kisi sebagai Keadaan	131
10.3.1 Mengisi sel demi sel	131
10.3.2 Pemanasan: pemasangan domino	131

10.3.3 Menyandikan profil sebagai satu bilangan	132
10.3.4 Perkiraan sebelum menulis kode	133
10.3.5 Larik atau <i>hash map</i> ?	133
10.4 Studi Kasus: CFONISMG	134
10.4.1 Soal dengan kata-kata kita sendiri	134
10.4.2 Contoh masukan dan keluaran	134
10.4.3 Merancang penyelesaian	134
10.4.4 Satu jalannya program dengan tangan	135
10.4.5 Penyelesaian lengkap	135
10.4.6 Baris demi baris	137
10.4.7 Jebakan	138
10.5 Uji Stres terhadap <i>Brute Force</i>	138
10.6 Taktik pada Hari Lomba	140
10.7 Ringkasan Bab	140
10.8 Soal Latihan	141
10.9 Penyelesaian Soal Latihan	141
Daftar Pustaka	142
BAB 11 Divide and Conquer CDQ dan LIS Dua Dimensi — LIS2	143
11.1 Di Mana Ini Berguna: Catatan Latihan Dewi	143
11.2 Tekniknya dari Nol	144
11.2.1 Dari satu dimensi ke dua dimensi	144
11.2.2 Gagasannya: belah menurut posisi	144
11.2.3 Mengapa urutan kiri, silang, kanan itu wajib	145
11.2.4 Biayanya	146
11.3 Soal Pemanasan	146
11.3.1 Inversi: paruh kiri memengaruhi paruh kanan	146
11.3.2 LIS satu dimensi dengan pohon Fenwick maksimum	147
11.4 Studi Kasus: LIS2	148
11.4.1 Soalnya dengan kata-kata kita sendiri	148
11.4.2 Contoh yang dikerjakan dengan tangan	149
11.4.3 Rancangan	149
11.4.4 Solusi lengkap	149
11.4.5 Baris demi baris	151
11.4.6 Jebakan	152
11.5 Mengujinya: CDQ Melawan DP Kuadratik	152
11.6 Taktik pada Hari Lomba	153
11.6.1 CDQ atas waktu: pembaruan dan kueri	154
11.7 Ringkasan Bab	154
11.8 Soal Latihan	154
11.9 Penyelesaian Soal Latihan	155
Daftar Pustaka	155
BAB 12 Meet in the Middle: Membelah Ledakan Eksponensial — ABCDEF	157
12.1 Di Mana Ini Berguna: Paket Bu Sari	157
12.2 Tekniknya dari Nol	158
12.2.1 Bentuk yang perlu dicari	158
12.2.2 Resepnya	158
12.2.3 Waktu dan memori, diperkirakan sebelum menulis kode	159
12.2.4 Jalannya dengan tangan	159
12.3 Soal Pemanasan	160
12.3.1 Menghitung pasangan dengan jumlah tertentu	160
12.3.2 Jumlah himpunan bagian dalam jendela, $n \leq 40$	161
12.4 Studi Kasus: ABCDEF	162
12.4.1 Soalnya dengan kata-kata kita sendiri	162

12.4.2 Dari satu persamaan menjadi dua sisi.....	162
12.4.3 Di mana $d \neq 0$ ditegakkan	162
12.4.4 Contoh-contohnya	162
12.4.5 Solusi lengkap.....	163
12.4.6 Baris demi baris.....	163
12.4.7 Bilangan yang menyelamatkan atau menguburkan	164
12.5 Mengujinya: Meet in the Middle Melawan n^6	164
12.6 Taktik pada Hari Lomba	165
12.6.1 Kerabat meet in the middle	166
12.6.2 Kesalahan yang sering terjadi	166
12.7 Ringkasan Bab.....	167
12.8 Soal Latihan.....	167
12.9 Penyelesaian Soal Latihan.....	167
Daftar Pustaka.....	168
BAB 13 Larik Sufiks dan LCP: Menghitung Substring Berbeda — DISUBSTR	169
13.1 Di Mana Ini Berguna: Kata-kata dalam Motif Batik.....	169
13.2 Tekniknya dari Nol.....	170
13.2.1 Larik sufiks dan larik LCP	170
13.2.2 Membangun larik sufiks dengan penggandaan prefiks	170
13.2.3 Larik LCP dalam waktu linear: algoritma Kasai	172
13.2.4 Dari larik LCP ke banyaknya substring berbeda.....	172
13.2.5 Kasai dengan tangan pada ABABA	173
13.3 Soal Pemanasan.....	173
13.3.1 Larik sufiks yang lambat, sebagai tabel	173
13.3.2 Pencarian pola dengan larik sufiks	174
13.4 Studi Kasus: DISUBSTR	175
13.4.1 Soalnya dengan kata-kata kita sendiri.....	175
13.4.2 Contoh	175
13.4.3 Rancangan	176
13.4.4 Solusi lengkap.....	176
13.4.5 Baris demi baris.....	178
13.5 Mengujinya: Larik Sufiks Melawan Himpunan String.....	178
13.6 Taktik pada Hari Lomba	179
13.6.1 Dua string sekaligus.....	179
13.6.2 Memilih alat yang tepat.....	180
13.6.3 Kesalahan yang sering terjadi	180
13.7 Ringkasan Bab.....	180
13.8 Soal Latihan.....	181
13.9 Penyelesaian Soal Latihan.....	181
Daftar Pustaka.....	182
BAB 14 Beda Hingga, Interpolasi, dan Aritmetika Modular — BATTLECRY	183
14.1 Di Mana Ini Berguna: Menghitung Tanpa Mendaftar	183
14.2 Tekniknya dari Nol.....	184
14.2.1 Beda hingga: stetoskop polinomial.....	184
14.2.2 Interpolasi Lagrange: dari titik ke polinomial	184
14.2.3 Aturan Horner dan aritmetika modular	185
14.2.4 Mengapa penempatan menghasilkan polinomial.....	185
14.2.5 Perpangkatan cepat dengan tangan	186
14.2.6 Berapa banyak nilai yang kita perlukan?	186
14.3 Soal Pemanasan.....	186
14.3.1 Tabel beda	186
14.3.2 Evaluasi Lagrange modulo bilangan prima	187
14.3.3 Seluruh alur pada soal yang lebih kecil	188

14.4 Studi Kasus: BATTLECRY	189
14.4.1 Soalnya dengan kata-kata kita sendiri.....	189
14.4.2 Memahami skakmat secara tepat	189
14.4.3 Alurnya, langkah demi langkah	189
14.4.4 Contoh	190
14.4.5 Solusi lengkap.....	190
14.4.6 Baris demi baris.....	191
14.5 Mengujinya: Rumus Melawan Papan Catur.....	192
14.6 Taktik pada Hari Lomba	194
14.6.1 Bila jawabannya bukan polinomial.....	195
14.6.2 Kesalahan yang sering terjadi	195
14.7 Ringkasan Bab.....	196
14.8 Soal Latihan.....	196
14.9 Penyelesaian Soal Latihan.....	196
Daftar Pustaka.....	197
BAB 15 Aliran Maksimum dengan Algoritma Dinic — FASTFLOW.....	198
15.1 Di Mana Ini Berguna: Air untuk Kebun	198
15.2 Tekniknya dari Nol.....	198
15.2.1 Aliran, kapasitas, dan potongan	198
15.2.2 Graf residu dan lintasan penambah.....	199
15.2.3 Edmonds–Karp: lintasan penambah terpendek.....	199
15.2.4 Dinic: banyak lintasan per BFS	199
15.2.5 Mengapa setiap fase membuat muara lebih jauh	201
15.3 Pemanasan: Edmonds–Karp pada Matriks.....	201
15.4 Studi Kasus: FASTFLOW	203
15.4.1 Soalnya dengan kata-kata kita sendiri.....	203
15.4.2 Contoh	203
15.4.3 Keputusan rancangan	203
15.4.4 Solusi lengkap.....	203
15.4.5 Baris demi baris.....	205
15.5 Mengujinya: Dinic Melawan Potongan Minimum.....	206
15.6 Taktik pada Hari Lomba	207
15.6.1 Contoh pemodelan yang dikerjakan.....	207
15.7 Jalan Anda Menuju Podium Nasional	208
15.7.1 Apa yang diberikan kelima belas bab	208
15.7.2 Rencana enam belas minggu.....	208
15.7.3 Kebiasaan harian dan mingguan	208
15.7.4 Pada hari lomba	209
15.7.5 Daftar Periksa kesiapan	209
15.8 Ringkasan Bab.....	210
15.9 Soal Latihan.....	210
15.10 Penyelesaian Soal Latihan.....	211
Daftar Pustaka.....	211

Daftar Gambar

Gambar 1.1 Hari lomba lima jam: petakan, amankan subsoal mudah, serang, selesaikan soal lain, dan kunci setengah jam terakhir; garis putus-putus adalah titik periksa aturan 45 menit.	6
Gambar 2.1 Pohon segmen dari 2, -5, 4, -1, 3, -6, 5, 1; kueri [2,5] mengunjungi simpul merah dan menggabungkan tiga simpul kuning [2], [3,4], dan [5].	23
Gambar 3.1 Penyalinan lintasan: menyisipkan nilai 3 ke versi 1 hanya membuat tiga simpul kuning di lintasan menuju daun [3]; sisi putus-putus menunjuk ke simpul yang dipakai bersama dengan versi 1.	36
Gambar 4.1 Histogram 2, 1, 4, 5, 1, 3, 3 dan persegi panjang terbesarnya; persegi panjang itu ditemukan ketika batang 4 mengeluarkan batang 2 dari tumpukan.	50
Gambar 5.1 Pohon untuk 1, 2, -3, 4 sesudah pembaruan “0 3 -3”: daun kuning diganti, dua simpul merah di lintasannya digabung ulang, dan simpul lain mempertahankan nilainya.	58
Gambar 6.1 Peta 5 × 6 dengan biaya masuk (angka besar) dan jarak d yang dihitung algoritma Dial; petak yang diarsir membentuk rute termurah berbiaya 17.	72
Gambar 7.1 Lima desa dan proyeksinya (batang berwarna pada sumbu); biaya sambungan 0, 1, dan 2 ditentukan oleh proyeksi mana yang bertumpuk.	90
Gambar 8.1 Pohon dengan waktu masuk dan keluar $[tin, tout]$; dalam urutan masuk, subpohon simpul 5 mengisi satu selang, yaitu posisi 4 sampai 7.	101
Gambar 9.1 Dari AB ke ABA: menambahkan A pada setiap subbarisan lama menciptakan satu duplikat, yaitu untai A, yang sudah dihasilkan oleh A sebelumnya.	117
Gambar 10.1 Mengisi kisi 3 × 4 dalam urutan membaca: ketika sel 6 diisi, hanya empat sel terakhir (profil) yang masih dapat memengaruhi masa depan.	131
Gambar 11.1 CDQ pada delapan titik. (a) Rentang dipotong menurut indeks; paruh kiri dituntaskan sebelum memengaruhi paruh kanan. (b) Untuk titik kanan $p = (7,6)$ setiap titik kiri dengan x lebih kecil telah disisipkan ke pohon Fenwick, yang kemudian menjawab “ dp terbesar dengan $y < 6$ ”	145
Gambar 12.1 Dua daftar terurut dan jabat tangannya: setiap garis adalah satu pasangan yang terhitung, dan nilai yang sama membentuk blok yang ukurannya dikalikan.	159
Gambar 13.1 Sufiks ABABA yang telah diurutkan. $lcp[k]$ karakter pertama setiap sufiks (abu-abu) mengulang tetangga di atasnya; hanya prefiks sisanya (hijau toska) yang merupakan substring baru, sebanyak $15 - 6 = 9$	170
Gambar 14.1 Tabel beda hingga hitungan BATTLECRY untuk $N = 3$ sampai 10. Setiap baris memuat beda dari baris di atasnya; baris 5 konstan, sehingga hitungannya mengikuti polinomial berderajat 5.	184
Gambar 15.1 Contoh FASTFLOW sebagai graf tak berarah dengan kapasitas dan level BFS. Gelang putus-putus dibuang; sisi 2-3 terletak di dalam level 1 dan tidak dipakai pada fase pertama.	200

Daftar Tabel

Tabel 1.1 Jalan dari usulan sekolah menuju tim internasional, sebagaimana diuraikan TOKI dan panduan OSN 2026.	2
Tabel 1.2 Membaca batasan: masukan terbesar memberi tahu kompleksitas mana yang lolos batas waktu satu detik.	4
Tabel 4.1 Empat pertanyaan tetangga terdekat dan syarat pengeluaran yang menjawab masing-masing dalam satu lintasan kiri-ke-kanan.	49
Tabel 4.2 Algoritma satu lintasan pada 2, 1, 4, 5, 1, 3, 3; setiap luas calon dihitung pada saat batangnya dikeluarkan.	50
Tabel 5.1 Rutinitas latihan dua jam dengan irama hari lomba.	61

Tabel 6.1 Tangga lintasan terpendek: bentuk bobot menentukan algoritma; D adalah jarak terbesar yang ditemukan.....	70
Tabel 10.1 Operasi himpunan pada <i>bitmask</i> ; perulangan terakhir mengunjungi setiap subhimpunan tak kosong dari m	129
Tabel 11.1 Waktu jalan LIS2 . cpp pada masukan terbesar.....	153
Tabel 12.1 Rentang nilai pada ABCDEF: nilainya muat dalam 32 bit, hitungannya tidak.....	164
Tabel 12.2 Waktu jalan ABCDEF . cpp pada ukuran maksimum.....	165
Tabel 13.1 Waktu jalan DISUBSTR . cpp pada ukuran maksimum.	179
Tabel 15.1 Waktu jalan FASTFLOW . cpp pada masukan berukuran maksimum.	207
Tabel 15.2 Rencana latihan enam belas minggu menjelang babak nasional.	208
Tabel 15.3 Daftar periksa kesiapan untuk minggu-minggu menjelang babak nasional.	210

Daftar Kode

Kode 1.1 tokenize . cpp: memotong setiap baris menjadi kata dan melaporkan tanda akhir pertama.....	7
Kode 1.2 stress . sh: kompilasi kedua program, lalu bandingkan pada masukan acak sampai keduanya berbeda.....	8
Kode 1.3 HOTLINE . cpp: fakta sebagai kunci, pemeriksaan kontradiksi bertahap, dan satu fungsi untuk setiap jenis pertanyaan.	10
Kode 1.4 gen . py: dialog HOTLINE acak dari kosakata empat orang dan tiga kata kerja.	15
Kode 2.1 prefix_sum . cpp: jumlah rentang statis dalam waktu konstan per pertanyaan.	21
Kode 2.2 kadane . cpp: blok tak kosong terbaik dari seluruh larik dalam satu lintasan.	22
Kode 2.3 GSS1 . cpp: pohon segmen empat-medan; pembangunan dalam $O(N)$, setiap pertanyaan dalam $O(\log N)$	27
Kode 2.4 brute . cpp: setiap blok dari setiap rentang, $O(N^2)$ per pertanyaan.	29
Kode 2.5 gen . py: masukan GSS1 acak pada tiga skala nilai.....	30
Kode 3.1 compress . cpp: urutkan, buang duplikat, dan ganti setiap nilai dengan peringkatnya.....	34
Kode 3.2 count_prefix . cpp: mencacah nilai yang paling besar x di sebuah rentang dengan mengurangi dua tabel cacah prefiks.....	35
Kode 3.3 MKTHNUM . cpp: pohon segmen persisten atas nilai terkompresi; setiap pertanyaan menuruni dua versi.	39
Kode 3.4 brute . cpp: urutkan salinan setiap rentang, $O(n \log n)$ per pertanyaan.	41
Kode 3.5 gen . py: masukan MKTHNUM acak, dengan dan tanpa duplikat.....	41
Kode 4.1 nearest_smaller . cpp: elemen tegas lebih kecil terdekat di setiap sisi, dua lintasan dengan tumpukan.	46
Kode 4.2 stock_span . cpp: rentang saham dengan tumpukan menurun berisi pasangan (harga, rentang).....	47
Kode 4.3 HISTOGRA . cpp: satu lintasan dengan tumpukan monoton dan sentinel setinggi 0.....	52
Kode 4.4 brute . cpp: setiap pasangan batang pertama dan terakhir, dengan tinggi minimum berjalan.	53
Kode 4.5 gen . py: masukan HISTOGRA acak, beberapa baris uji diakhiri 0.....	53
Kode 5.1 sum_update . cpp: pohon segmen untuk jumlah rentang dengan penetapan titik.	58
Kode 5.2 GSS3 . cpp: pohon GSS1 dengan pembaruan titik; setiap operasi memakan $O(\log N)$	63
Kode 5.3 brute . cpp: pembaruan pada larik biasa dan setiap blok dari setiap rentang.	65
Kode 5.4 gen . py: masukan GSS3 acak yang mencampur pembaruan dan pertanyaan.	65
Kode 6.1 knight . cpp: <i>breadth-first search</i> atas 64 keadaan kuda, dengan petak terlarang.....	73
Kode 6.2 zero_one_bfs . cpp: BFS 0-1 dengan antrean dua ujung pada peta berisi petak bebas dan dinding.	74

Kode 6.3 CHIPSLL .cpp: algoritma Dial dengan 64 ember melingkar pada kisi datar.....	77
Kode 6.4 brute .cpp: penyelesaian acuan dengan <i>binary heap</i>	80
Kode 6.5 gen .py: masukan acak kecil untuk CHIPSLL.....	81
Kode 6.6 stress .sh: kompilasi, bangkitkan, bandingkan, berhenti pada ketidakcocokan pertama.....	82
Kode 7.1 interval_groups .cpp: mengelompokkan selang dengan sapuan ujung kanan terjauh.....	87
Kode 7.2 kruskal .cpp: algoritma Kruskal dengan DSU lima baris.....	88
Kode 7.3 VILUNIT .cpp: Kruskal per kelas bobot, dengan lintasan 0 dan dua sapuan.....	92
Kode 7.4 brute .cpp: semua pasangan, bobot 0, 1, 2 yang eksplisit, dan Kruskal.....	95
Kode 7.5 gen .py: desa-desa acak kecil untuk VILUNIT.....	95
Kode 8.1 tree_gears .cpp: tur Euler, jumlah lintasan dengan Fenwick, dan <i>binary lifting</i> pada sembilan simpul.....	103
Kode 8.2 HBD .cpp: tur Euler, <i>Fenwick tree</i> , dan <i>binary lifting</i> yang dirangkai.....	107
Kode 8.3 brute .cpp: acuan jalan-naik untuk HBD.....	110
Kode 8.4 gen .py: pohon dan operasi acak kecil untuk HBD.....	111
Kode 9.1 abc_count .cpp: menghitung kemunculan pola sebagai subbarisan.....	118
Kode 9.2 dsub_table .cpp: tabel DSUBSEQ untuk ABAB, langkah demi langkah.....	119
Kode 9.3 DSUBSEQ .cpp: subbarisan berbeda dengan koreksi kemunculan terakhir.....	121
Kode 9.4 brute .cpp: mendaftar setiap subbarisan ke dalam sebuah himpunan.....	123
Kode 9.5 gen .py: untai pendek acak atas alfabet kecil.....	123
Kode 10.1 tsp_bitmask .cpp: pemrograman dinamis Held–Karp untuk lima kota.....	130
Kode 10.2 domino_profile .cpp: menghitung pemasangan domino dengan profil satu bit per kolom.....	132
Kode 10.3 CFONISMG .cpp: <i>bitmask</i> siswa yang sudah duduk ditambah profil basis 17, disimpan dalam <i>hash map</i>	135
Kode 10.4 brute .cpp: runut-balik atas setiap denah tempat duduk.....	138
Kode 10.5 gen .py: kelas-kelas acak kecil untuk CFONISMG.....	140
Kode 11.1 inversions .cpp: menghitung inversi dengan <i>merge sort</i> , leluhur CDQ.....	147
Kode 11.2 lis_fenwick .cpp: LIS klasik dalam $O(n \log n)$ dengan pohon Fenwick maksimum prefiks.....	147
Kode 11.3 LIS2 .cpp: LIS dua dimensi dalam $O(n \log^2 n)$ dengan divide and conquer CDQ.....	149
Kode 11.4 brute .cpp: program dinamis $O(n^2)$ yang dipakai sebagai acuan.....	152
Kode 11.5 gen .py: masukan acak kecil dengan banyak nilai kembar.....	152
Kode 11.6 stress .sh: penggerak uji stres.....	153
Kode 12.1 two_sum_count .cpp: menghitung pasangan dengan jumlah tertentu melalui pengurutan dan dua pencarian biner.....	160
Kode 12.2 subset_mitm .cpp: menghitung himpunan bagian yang jumlahnya dalam $[A, B]$ dengan meet in the middle.....	161
Kode 12.3 ABCDEF .cpp: menghitung tupel Persamaan (12.4) dengan meet in the middle.....	163
Kode 12.4 brute .cpp: acuan n^6	164
Kode 12.5 gen .py: himpunan acak kecil untuk ABCDEF.....	165
Kode 13.1 naive_sa .cpp: larik sufiks dan larik LCP yang dihitung dengan cara lambat, dicetak sebagai tabel.....	173
Kode 13.2 pattern_count .cpp: menghitung kemunculan pola dengan pencarian biner pada larik sufiks.....	174
Kode 13.3 DISUBSTR .cpp: substring berbeda dengan larik sufiks (penggandaan prefiks) dan LCP Kasai.....	176
Kode 13.4 brute .cpp: acuan berupa himpunan substring.....	178
Kode 13.5 gen .py: string acak atas alfabet kecil dan alfabet campuran.....	179
Kode 14.1 diff_table .cpp: tabel beda hingga dan derajat yang terdeteksi.....	186
Kode 14.2 lagrange .cpp: mengevaluasi polinomial interpolasi di titik baru tanpa menghitung koefisien.....	187

Kode 14.3 BATTLECRY .cpp: rumus derajat 5 yang dievaluasi dengan aturan Horner dan invers modular.	190
Kode 14.4 brute .cpp: menghitung skakmat dengan mencoba setiap posisi.	192
Kode 14.5 gen .py: ukuran papan kecil secara acak.	194
Kode 15.1 edmonds_karp .cpp: aliran maksimum dengan lintasan penambah terpendek pada matriks kapasitas.	201
Kode 15.2 FASTFLOW .cpp: algoritma Dinic dengan graf level dan busur kini, $O(V^2E)$ pada kasus terburuk.	203
Kode 15.3 brute .cpp: potongan minimum dengan mencoba setiap himpunan S	206
Kode 15.4 gen .py: multigraf tak berarah acak yang kecil.	206

Daftar Pseudokode

Pseudokode 6.1 Algoritma Dial dengan $C + 1$ ember melingkar; baris 6 melewati salinan yang tertinggal ketika jarak suatu simpul membaik.	71
Pseudokode 11.1 Divide and conquer CDQ untuk LIS dua dimensi.	145
Pseudokode 12.1 Menghitung pasangan yang cocok dengan meet in the middle.	158
Pseudokode 13.1 Membangun larik sufiks dengan penggandaan prefiks.	171
Pseudokode 15.1 Algoritma Dinic dengan graf level dan busur kini.	200

Daftar Persamaan

(1.1)	4
(1.2)	5
(1.3)	10
(2.1)	21
(2.2)	22
(2.3)	23
(2.4)	24
(2.5)	24
(2.6)	24
(2.7)	24
(2.8)	24
(3.1)	35
(3.2)	36
(3.3)	37
(3.4)	37
(4.1)	46
(4.2)	49
(4.3)	50
(4.4)	54
(5.1)	60
(5.2)	61
(5.3)	62
(5.4)	66
(6.1)	70

(6.2)	71
(6.3)	71
(6.4)	76
(7.1)	87
(7.2)	87
(8.1)	100
(8.2)	101
(8.3)	102
(8.4)	102
(8.5)	102
(8.6)	106
(9.1)	116
(9.2)	117
(9.3)	118
(9.4)	118
(10.1).....	129
(10.2).....	130
(10.3).....	133
(10.4).....	135
(11.1).....	144
(11.2).....	144
(11.3).....	146
(12.1).....	158
(12.2).....	159
(12.3).....	162
(12.4).....	162
(13.1).....	170
(13.2).....	170
(13.3).....	171
(13.4).....	172
(14.1).....	184
(14.2).....	184
(14.3).....	185
(14.4).....	185
(14.5).....	185
(14.6).....	185
(15.1).....	198
(15.2).....	199
(15.3).....	200

Kata Pengantar

Pukul sembilan lewat dua puluh menit pada hari pertama babak nasional. Dimas sudah membaca ketiga soal. Soal pertama ia kenali: sebuah larik, serangkaian pertanyaan tentang rentang, masing-masing seratus ribu banyaknya. Ia tahu bahwa perulangan atas setiap rentang akan terlalu lambat, dan ia pernah mendengar struktur bernama pohon segmen, tetapi ia belum pernah menuliskannya tanpa melihat rujukan, dan rujukan tidak boleh dibawa ke ruang lomba. Ia menulis perulangan itu, mengumpulkan poin subsoal terkecil, lalu menghabiskan sisa pagi dengan pohon yang tidak pernah benar-benar jalan. Malamnya, ketika hasil diumumkan, ia tahu bahwa medali perunggu hanya membutuhkan tiga puluh poin lagi — tepat poin subsoal yang gagal ia selesaikan.

Buku ini ditulis agar pagi itu berjalan lain. Olimpiade Sains Nasional bidang informatika, OSN, meminta siswa menyelesaikan soal algoritmik dalam C++ selama lima jam, dengan nilai parsial untuk setiap subsoal yang berhasil diselesaikan. Yang membedakan peserta yang pulang membawa medali dari yang tidak jarang sekali hanya bakat. Pembedanya adalah sejumlah kecil teknik yang dikuasai begitu baik sehingga dapat ditulis dari ingatan, kebiasaan membaca batasan sebelum memilih algoritma, rutinitas menguji program sebelum juri mengujinya, dan rencana hari lomba yang mengamankan poin mudah sebelum mengejar poin sulit.

Judul buku ini menjanjikan jalan *menuju juara nasional*, dan kata *juara* di sini dibaca seperti sekolah-sekolah kita memakainya: siswa yang meraih medali emas, perak, atau perunggu di babak nasional adalah juara nasional, dan dalam arti yang sungguh-sungguh, begitu pula siswa yang meraih penghargaan (*honourable mention*) atau sekadar lolos ke final nasional setelah mengungguli ribuan peserta lain di babak kabupaten/kota dan provinsi. Tidak ada buku yang dapat menjanjikan satu tempat teratas, yang bergantung pada hari itu dan pada finalis lain sama besarnya dengan pada diri Anda. Buku ini bertujuan menjadikan Anda finalis yang meraih nilai baik, dan yang tahu mengapa setiap poin itu didapat.

Latihannya diambil dari SPOJ, Sphere Online Judge, salah satu arsip soal lomba yang tertua dan terluas. Lima belas soal sulit klasiknya — HOTLINE, GSS1, MKTHNUM, HISTOGRAM, GSS3, CHPSLL, VILUNIT, HBD, DSUBSEQ, CFONISM, LIS2, ABCDEF, DISUBSTR, BATTLECRY, dan FASTFLOW — menjadi studi kasus untuk mengajarkan lima belas teknik olimpiade, satu teknik per bab.

Tentang Edisi Publik ini

Ini adalah Edisi Publik buku ini, *15 Studi Kasus SPOJ*. Teksnya adalah teks edisi lengkap: setiap gagasan, setiap gambar, setiap sketsa bukti, setiap program pemanasan, dan setiap uji stres ada di sini, utuh. Bedanya terletak pada kelima belas program studi kasus. Masing-masing hanya menampilkan sekitar separuh kodenya — pembacaan masukan, struktur data, paruh pertama algoritma inti, dan keluaran — sedangkan separuh lainnya, bagian yang mengubah gagasan menjadi program yang diterima, diserahkan kepada Anda. Di dalam listing, bagian itu tampil sebagai blok yang dibuka dengan baris `// <P50>` dan ditutup dengan `// </P50>`. Tepat sesudah baris pembuka, satu sampai tiga baris komentar menyebutkan dengan tepat apa yang harus dikerjakan bagian yang hilang itu; sesudahnya muncul baris `// >>> YOUR TURN (Public Edition): write this part yourself, following the hint above. <<<`.

Ini bukan lubang dalam buku; inilah latihan yang paling berharga di dalamnya. Setiap bagian yang hilang dijelaskan sepenuhnya dalam uraian bab, biasanya dengan contoh yang dikerjakan dan sebuah persamaan, dan program *brute force*, pembangkit, serta skrip uji stres setiap bab tetap lengkap, sehingga Anda dapat memeriksa tulisan Anda dengan cara yang sama seperti program-program buku ini diperiksa. Tulislah bagian itu, jalankan uji stres sampai tidak lagi menemukan selisih, lalu kirimkan ke SPOJ. Ketika juri menjawab *Accepted*, program itu menjadi milik Anda dengan cara yang tidak pernah dimiliki program salinan — dan di ruang lomba, tempat buku tidak boleh dibawa, hanya jenis itulah yang berarti. Edisi lengkap, *Menaklukkan SPOJ*, mencetak kelima belas solusi secara utuh.

Untuk siapa buku ini ditulis

Buku ini ditulis untuk siswa sekolah menengah yang bersiap menghadapi OSN bidang informatika, untuk guru dan pembina mereka, dan untuk mahasiswa yang menginginkan teknik yang sama untuk ICPC atau wawancara teknis. Buku ini mengandaikan Anda sudah dapat menulis program C++ yang berjalan dengan perulangan, larik, fungsi, dan kontainer standar, serta sudah mengenal rekursi dan gagasan menghitung langkah dengan notasi O . Jika belum, buku pendampingnya, *Pemrograman Kompetitif dengan SPOJ*, dimulai dari nol: buku itu memasang meja kerja, mengajarkan cara membaca soal, anggaran waktu dan memori, serta uji stres, dan membangun perkakas dasar pemrograman dinamis, algoritma serakah, pencarian biner, dan graf. Buku ini dimulai di tempat buku itu berakhir.

Setiap teknik tetap dibangun dari nol. Setiap bab menjelaskan gagasannya dengan kata-kata sederhana, mengerjakannya dengan tangan pada contoh kecil disertai gambar, menunjukkan mengapa ia benar, menghitung biayanya, dan baru sesudah itu menulis kode. Soal-soal pemanasan dengan program pendek datang sebelum studi kasus yang sulit, sehingga studi kasus itu menjadi anak tangga terakhir, bukan tembok.

Satu bahasa, program yang teruji

Babak nasional adalah lomba C++, dan begitu pula buku ini: setiap program ditulis dalam **C++17** dan dikompilasi dengan `g++ -std=c++17 -O2 -Wall -Wextra` tanpa satu peringatan pun, dan juga dijalankan dengan `-fsanitize=address,undefined`, yang menghentikan program begitu ia membaca memori yang bukan miliknya. Pembangkit masukan kecil ditulis dalam Python 3, karena lebih ringkas.

Tidak ada program dalam buku ini yang dipercaya hanya karena lolos contoh. Setiap solusi studi kasus **diuji stres terhadap brute force**: program kedua yang sengaja dibuat sederhana dan lambat, yang kebenarannya jelas, dijalankan berdampingan dengannya pada ribuan masukan acak kecil, dan pengujian berhenti pada ketidakcocokan pertama. Setiap soal diceritakan ulang dengan kata-kata kami sendiri, beserta kode SPOJ-nya, agar Anda dapat membaca aslinya dan mengirimkan solusi Anda sendiri.

Isi buku ini

Bab 1 adalah buku pedoman sang juara. Bab ini menggambarkan OSN sebagaimana berjalan pada 2026 — babak kabupaten/kota, provinsi, dan nasional, penilaian per subsoal, lingkungan lomba, dan jalan melalui pelatihan nasional TOKI menuju IOI — menyiapkan meja kerja dalam sepuluh menit, mengubah batasan menjadi anggaran langkah, merencanakan lima jam hari lomba, serta

memperkenalkan protokol *debug* dan uji stres pada HOTLINE, soal yang sebagian besar menuntut ketelitian membaca.

Bab 2 sampai 5 membahas larik dan rentang. Bab 2 membangun pohon segmen dan gagasan jawaban yang dapat digabung, simpul dengan empat medan, pada GSS1. Bab 3 membuat pohon itu persisten dengan penyalinan jalur dan menjawab “bilangan terkecil ke-k dalam sebuah rentang” dengan mengurangkan dua versi, pada MKTHNUM. Bab 4 mengajarkan tumpukan monoton dan persegi panjang terbesar dalam histogram, pada HISTOGRAM. Bab 5 menghidupkan pohon segmen dengan pembaruan titik, pada GSS3.

Bab 6 sampai 8 membahas graf dan pohon. Bab 6 menemukan graf tersembunyi di dalam ruang keadaan dan menjalankan algoritma Dial dengan ember melingkar, pada CHIPSLL. Bab 7 memadukan himpunan saling lepas, algoritma Kruskal, dan sapuan menjadi pohon rentang minimum per kelas bobot, pada VILUNIT. Bab 8 merangkai tur Euler, *Fenwick tree*, dan *binary lifting* untuk menjawab pertanyaan tentang lintasan di pohon, pada HBD.

Bab 9 dan 10 membahas pemrograman dinamis. Bab 9 menghitung subbarisan berbeda tanpa menghitung dua kali, pada DSUBSEQ. Bab 10 mengubah himpunan menjadi bilangan dan penampang sebuah kisi menjadi keadaan, dengan pemrograman dinamis *bitmask* dan profil, pada CFONISMG.

Bab 11 sampai 15 adalah perkakas lanjut. Bab 11 menyelesaikan subbarisan naik terpanjang dua dimensi dengan *divide and conquer* CDQ, pada LIS2. Bab 12 membelah pencarian eksponensial dengan *meet in the middle*, pada ABCDEF. Bab 13 membangun larik sufiks dan larik LCP lalu menghitung substring berbeda, pada DISUBSTR. Bab 14 menemukan rumus dari tabel jawaban kecil dengan beda hingga, interpolasi, dan aritmetika modular, pada BATTLECRY. Bab 15 menghitung aliran maksimum dengan algoritma Dinic, pada FASTFLOW, dan ditutup dengan peta jalan dari sini menuju podium nasional.

Setiap bab menempuh jalan yang sama: sebuah kisah dan masalah yang dimunculkannya, teknik dari nol dengan gambar dan contoh kecil yang dikerjakan dengan tangan, soal pemanasan dengan program pendek, studi kasus yang sulit — soal yang diceritakan ulang, rancangannya, program lengkap dan penjelasannya baris demi baris, jebakan-jebakannya, dan uji stres terhadap *brute force* — lalu taktik hari lomba, ringkasan, latihan dengan penyelesaian lengkap, dan daftar pustaka.

Program-program buku ini dikumpulkan bab demi bab dalam folder kode pendamping, dari [Chapter01](#) sampai [Chapter15](#). Setiap folder berisi solusi studi kasus, program pemanasan, *brute force* [brute.cpp](#), pembangkit [gen.py](#), skrip uji stres [stress.sh](#), contoh masukan, dan README yang menjelaskan cara membangun dan menjalankannya di terminal dan di VS Code pada Windows, macOS, dan Linux.

Cara membacanya

Dengan kompilator terbuka dan akun SPOJ siap. Bacalah Bab 1 lebih dahulu, karena bab itu menanamkan kebiasaan kerja yang diandalkan semua bab berikutnya; sesudah itu, Bab 2, 3, dan 5 membentuk satu alur tentang pohon segmen dan sebaiknya dibaca berurutan, sedangkan bab-bab lain dapat dibaca dalam urutan apa pun, dan setiap bab menyebutkan bab sebelumnya yang menjadi landasannya. Sebelum membaca solusi sebuah studi kasus, bacalah soal yang diceritakan ulang dan cobalah sendiri setidaknya setengah jam: pergulatan itulah yang membuat gagasannya melekat. Kerjakan contoh-contoh kecil dengan tangan, ketik ulang programnya alih-alih

menyalinnya, dan ketika sebuah program sudah jalan, rusaklah dengan sengaja — buat larik kurang satu elemen, lupakan sebuah modulo, balik sebuah perbandingan — lalu lihat uji stres menangkapnya. Kemudian tutup buku dan tulis ulang tekniknya dari ingatan; di ruang lomba, ingatan adalah satu-satunya yang Anda punya.

Gambar, tabel, listing, pseudokode, dan persamaan yang bernomor semuanya berupa tautan hidup: dalam edisi elektronik, klik “Listing 5.3” atau “[4]” di mana pun dalam teks dan Anda langsung dibawa ke sana. Daftar isi dan daftar-daftar di bagian depan buku bekerja dengan cara yang sama.

Tentang penulis

Irfan Subakti menggambarkan dirinya dalam satu baris — *Father / Educator / Entrepreneur / Bridging Tech & Nature: CS Lecturer × Crypto/Digital Business × AI/ML Specialist × Fruit Farmer / Storyteller* — ayah, pendidik, wirausahawan, pen jembatan teknologi dan alam: dosen ilmu komputer, pelaku bisnis kripto dan digital, spesialis AI/ML, petani buah, dan pencerita. Setiap bagian dari baris itu ikut masuk ke dalam buku ini. Ia mengajar di Departemen Teknik Informatika ITS sejak 1998, dengan perancangan dan analisis algoritma sebagai salah satu mata ajarnya, dan telah bertahun-tahun membina mahasiswa untuk lomba pemrograman. Ia membimbing dan menguji tugas akhir mahasiswa sarjana, mengajar MLOps dalam program IBM SkillsBuild, dan membangun produk: ia mendirikan Profitina, platform visibilitas AI, yang para insinyurnya menghadapi anggaran waktu dan memori yang sama dengan yang diajarkan buku ini. Jauh dari papan ketik, ia merawat kebun buah dan durian keluarga — di sanalah ia belajar bahwa perangkat lunak yang baik, seperti kebun yang baik, ditumbuhkan dengan sabar, satu bagian yang terawat demi satu bagian. Jiwa penceritanya adalah alasan setiap bab dalam buku ini dibuka dengan seseorang dan masalahnya, bukan dengan sebuah definisi.

Ucapan terima kasih

Kepada para mahasiswa yang berlatih dengan versi-versi awal materi ini, dan yang pengiriman-pengirimannya yang ditolak, yang ditanyakan larut malam, menyingkapkan di mana sebuah penjelasan masih tipis, serta kepada rekan-rekan di ITS yang bersama-sama mengajarkan dan menyempurnakan materi algoritma dan pemrograman dari tahun ke tahun. Kepada Tim Olimpiade Komputer Indonesia dan para guru serta pembina di seluruh tanah air yang setiap tahun menyiapkan siswa untuk olimpiade. Kepada para pembuat soal dan pengelola SPOJ, yang soal-soalnya telah melatih satu generasi pemrogram, dan kepada komunitas pemrograman kompetitif yang lebih luas, yang pembahasan dan diskusinya menjaga keterampilan ini tetap hidup. Standar buku ini juga berutang pada dokumentasi resmi C++ dan GCC, serta pada buku teks klasik dan literatur riset algoritma, yang dikutip di setiap tempat penggunaannya.

Surabaya, September 2026



Irfan Subakti · yifana@gmail.com



profitina.com

Bab 1

Buku Pedoman Sang Juara: Strategi Lomba dan Membaca dengan Cermat — HOTLINE

Olimpiade informatika tingkat nasional dimenangkan jauh sebelum ruang lomba dibuka, dan cukup sering dikalahkan justru pada sepuluh menit pertama hari lomba. Bab ini membahas kedua sisi itu. Anda akan mengenal wujud Olimpiade Sains Nasional (OSN) bidang informatika pada 2026, bagaimana tahap-tahapnya berujung pada pelatihan nasional dan International Olympiad in Informatics (IOI), serta bagaimana sistem penilaiannya menghargai kemajuan parsial yang teliti. Anda akan menyiapkan meja kerja yang kecil tetapi andal, belajar membaca batasan soal sebagaimana pembuat soal menuliskannya, merencanakan hari lomba lima jam menit demi menit, dan membangun satu kebiasaan yang membedakan peserta yang tenang dari peserta yang panik: *stress test*, yaitu program lambat yang jelas benar diadu dengan program cepat Anda pada ratusan masukan acak. Bab ini ditutup dengan soal SPOJ yang sulit, HOTLINE, yang kesulitannya sama sekali bukan pada algoritma, melainkan pada membaca: setiap kata dalam pernyataan soal menjadi aturan dalam program, dan satu karakter yang meleset berarti jawaban salah.

1.1 Surat untuk Calon Finalis

Bayangkan Dimas, siswa kelas XI sebuah SMA negeri di kota kecil di Jawa Timur. Ia menyukai teka-teki, sudah menulis beberapa program C++, dan bulan lalu gurunya berkata bahwa sekolah akan mengirimnya ke tahap pertama olimpiade informatika. Ia bersemangat, sekaligus sedikit takut. Ia pernah membaca tentang siswa yang “juara OSN”, pernah melihat foto mereka berkalung medali, dan ia bertanya-tanya apakah anak dari sekolahnya, tanpa pelatih privat, bisa berdiri di tempat yang sama.

Buku ini ditulis untuk Dimas, dan untuk Anda. Judulnya menjanjikan jalan menuju *juara nasional*, dan maknanya perlu dikatakan dengan jelas dan ramah. Dalam budaya sekolah kita, kata *juara* dipakai dengan murah hati: siswa yang pulang membawa medali emas, perak, atau perunggu dari babak nasional adalah juara nasional, dan dalam arti yang sungguh-sungguh begitu pula siswa yang meraih penghargaan harapan atau sekadar lolos ke babak final nasional setelah menyisihkan ribuan peserta di tingkat kabupaten/kota dan provinsi. Tujuan buku ini bukan satu peringkat pertama, yang bergantung pada hari itu, pada keberuntungan, dan pada finalis lain sebesar bergantung pada Anda. Tujuannya adalah menjadikan Anda peserta yang mencapai babak nasional dan meraih nilai yang baik di sana, serta tahu persis mengapa setiap poin itu diperoleh.

Tujuan itu realistis. Soal babak nasional memang sulit, tetapi dibangun dari himpunan gagasan yang terbatas: struktur data yang menjawab pertanyaan tentang rentang, pemrograman dinamis, algoritma graf, pencacahan yang teliti, dan disiplin membaca pernyataan soal secara tepat. Setiap bab buku ini mengambil satu gagasan, membangunnya dari nol, lalu memakainya pada soal sulit dari Sphere Online Judge (SPOJ), *online judge* gratis dengan ribuan soal yang telah melatih pemrogram kompetitif selama dua dasawarsa. Buku pendamping *Pemrograman Kompetitif*

dengan SPOJ [1] membahas dasar-dasarnya (*judge*, masukan dan keluaran, kompleksitas, pengurutan, struktur data pertama); buku ini menganggap Anda sudah mengenal gagasan-gagasan itu dan mendaki dari sana.

1.2 OSN Informatika pada 2026

1.2.1 Tahap-tahapnya

Olimpiade Sains Nasional diselenggarakan oleh Pusat Prestasi Nasional (Puspresnas) Kementerian Pendidikan Dasar dan Menengah, dan untuk bidang informatika, soal, silabus, serta sistem lombanya disiapkan oleh Tim Olimpiade Komputer Indonesia (TOKI). Situs resmi OSN informatika menyebut tiga tingkat [2]: tingkat kabupaten/kota (OSN-K), tingkat provinsi (OSN-P), dan tingkat nasional (OSN). **Tabel 1.1** merangkumnya. Untuk 2026, pemberitaan tentang panduan kementerian melaporkan bahwa OSN-P jenjang SMA berlangsung daring pada 6–9 Juli 2026 [3]. Babak nasional bidang informatika diselenggarakan di Malang pada 14–20 September 2026, dengan hari lomba lima jam (hari kedua berlangsung pukul 07.30–12.30 WIB) [4].

Tabel 1.1 Jalan dari usulan sekolah menuju tim internasional, sebagaimana diuraikan TOKI dan panduan OSN 2026.

Tahap	Siapa yang ikut	Apa yang dituntut dari Anda
OSN-K (kabupaten/kota)	Siswa yang diusulkan sekolahnya	Sebagian besar soal isian singkat: logika, pencacahan, membaca dan menelusuri kode
OSN-P (provinsi)	Yang terbaik dari tiap kabupaten/kota	Soal penalaran yang lebih sulit dan, dalam beberapa tahun terakhir, soal pemrograman
OSN (nasional)	Yang terbaik dari tiap provinsi	Dua hari lomba soal pemrograman bersubsoal, masing-masing lima jam
Pelatnas TOKI	Peraih medali nasional terpilih	Beberapa tahap pelatihan dan seleksi menuju tim IOI

Silabusnya mengikuti silabus International Olympiad in Informatics dengan penyesuaian lokal. Silabus OSN 2026 memuat sepuluh bidang [2]: dasar pemrograman; logika dan operasi bit; aritmetika dan teori bilangan; pencacahan dan peluang; rekursi, termasuk *divide and conquer* (bagi dan kuasai) dan *backtracking* (runut balik); pencarian dan pengurutan; strategi pemecahan masalah seperti pencarian lengkap, algoritma *greedy* (rakus), dan pemrograman dinamis; struktur data dari larik dan tumpukan hingga *heap* dan pohon; graf dan pohon, termasuk penelusuran, lintasan terpendek, dan pohon perentang; serta geometri dasar. Setiap bab buku ini berada di dalam daftar itu.

1.2.2 Cara babak nasional dinilai

Babak nasional mengikuti format IOI. Peraturan yang diterbitkan untuk babak nasional beberapa tahun terakhir menyebut dua hari lomba masing-masing lima jam, dengan tiga soal per hari [5]. Setiap soal bernilai 100 poin dan dibagi menjadi *subsoal* (*subtask*): kelompok kasus uji dengan batasan tambahan, masing-masing bernilai sebagian dari 100 poin itu. Sebuah subsoal bisa berbunyi “ $N \leq 10$ ”, “semua nilai sama”, atau “pohonnya berupa lintasan”. Poin subsoal hanya diperoleh bila semua kasus ujinya lolos, dan nilai yang dihitung untuk sebuah subsoal adalah nilai maksimum di antara semua pengumpulan Anda, sehingga pengumpulan yang lebih buruk di kemudian waktu tidak pernah mengurangi poin. Nilai total adalah jumlah atas semua soal kedua hari; peserta dengan nilai sama berbagi peringkat, dan waktu pengumpulan tidak berpengaruh [5].

Halaman hasil OSN 2026 dari TOKI memperlihatkan artinya dalam praktik [6]. Dengan enam soal, nilai maksimum adalah 600 poin. Nilai tertinggi pada 2026 adalah 552; halaman itu mencantumkan 5 medali emas, 10 perak, 16 perunggu, dan 10 juara harapan, dan nilai peraih medali terendah adalah 197 poin, kira-kira sepertiga dari maksimum. Baca angka itu sekali lagi: medali perunggu di olimpiade nasional membutuhkan kira-kira nilai dua soal yang tersebar di enam soal. Sebagian besar poin itu berasal dari subsoal.

1.2.3 Sesudah babak nasional: TOKI dan IOI

TOKI mempersiapkan siswa sekolah menengah terbaik Indonesia untuk mewakili negara dalam kompetisi internasional [7]. Peraih medali babak nasional menjadi himpunan asal pembentukan pelatnas, dan pelatnas, melalui beberapa tahap, memilih empat siswa yang berangkat ke IOI. IOI 2026 diselenggarakan di Tashkent, Uzbekistan, pada 9–16 Agustus 2026, diikuti 375 peserta dari 92 negara dan delegasi; batas emas, perak, dan perunggu sekitar 361, 303, dan 229 poin dari 600 [8]. Keempat peserta Indonesia pulang membawa medali perunggu [9]. Lomba di sana tampak persis seperti babak nasional: dua hari, tiga soal sehari, subsoal, dan *judge* yang menampilkan nilai Anda begitu Anda mengumpulkan.

1.2.4 Lingkungan lomba

Peraturan nasional yang diterbitkan TOKI menggambarkan lomba dengan bahasa C++, dikompilasi dengan GCC dalam mode C++17 atau C++20, dan peserta boleh memakai kertas kosong, penyunting teks dan kompilator luring, serta dokumentasi bahasa, tidak lebih [5]. Jumlah pengumpulan per soal dibatasi, dan soal dapat berjenis *batch* (baca masukan, tulis keluaran), *interaktif* (berdialog dengan program *grader*), atau *output-only*. Rinciannya berubah dari tahun ke tahun, jadi bacalah halaman peraturan tahun Anda di situs TOKI pada minggu sebelum lomba, dan berlatihlah dalam kondisi yang sama: penyunting sederhana, kompilator di terminal, tanpa internet.

WAWASAN KUNCI: Aturan “nilai maksimum di antara semua pengumpulan, per subsoal” mengubah cara Anda seharusnya bekerja. Sebuah pengumpulan tidak pernah bisa merugikan Anda. Kumpulkan lebih awal, kumpulkan sering, dan perlakukan setiap pengumpulan sebagai cara mengamankan subsoal, bukan sebagai jawaban akhir.

1.3 Meja Kerja dalam Sepuluh Menit

Semua kode dalam buku ini ditulis dalam C++17 dan dikompilasi dengan `g++ -std=c++17 -O2 -Wall -Wextra`, yang diterima baik oleh babak nasional maupun SPOJ. Pemasangan lengkap, dengan tangkapan layar dan penanganan masalah untuk setiap sistem, ada di Bab 1 buku *Pemrograman Kompetitif dengan SPOJ* [1]. Berikut versi singkatnya, dengan versi yang berlaku pada September 2026:

- **Windows.** Pasang MSYS2 dan, di terminal UCRT64-nya, paket `mingw-w64-ucrt-x86_64-gcc` (GCC 16.2.0 pada September 2026) [10]. Tambahkan folder `bin`-nya ke `PATH`.
- **macOS.** Jalankan `xcode-select --install`; Command Line Tools menyediakan Clang milik Apple, yang juga menjawab panggilan `g++`. Bila Anda menginginkan GCC yang sebenarnya, pasang lewat Homebrew.
- **Linux.** Pasang `build-essential` (Ubuntu dan Debian) atau paket `gcc-c++` (Fedora). GCC 16.2, dirilis 7 Agustus 2026, adalah rilis terbaru [11].

- **Penyunting.** Visual Studio Code (versi 1.139 pada September 2026) [12] dengan ekstensi C/C++ dari Microsoft; Python 3 hanya diperlukan untuk pembangkit uji kecil dalam buku ini.

Kode bab ini berada di folder `Code_new/Chapter01`, dan folder setiap bab berbentuk sama: solusi yang dinamai menurut soal SPOJ-nya, `brute.cpp` yang lambat, pembangkit masukan acak `gen.py`, skrip `stress.sh` yang membandingkan kedua program, program pemanasan, contoh masukan, dan `README.md` berisi perintah untuk Windows, macOS, dan Linux.

Berlatihlah dalam kondisi lomba. Di babak nasional Anda tidak akan memiliki *plug-in* kesayangan, dan mesinnya mungkin bukan milik Anda. Maka, paling tidak seminggu sekali, tulislah satu solusi utuh di penyunting sederhana dan kompilasi di terminal. Hafalkan cara mengompilasi, cara mengalihkan masukan dari berkas (`./prog < in.txt`), dan cara membandingkan dua berkas (`cmp` atau `diff` di Linux dan macOS, `fc` di Windows).

1.4 Membaca Batasan seperti Pembuat Soal

Setiap pernyataan soal membawa pesan tersembunyi dalam batasannya. Pembuat soal memilih “ $N \leq 100\,000$ ” dan bukan “ $N \leq 1\,000$ ” karena ia ingin menggugurkan solusi lambat tertentu. Membaca batasan berarti mengetahui jenis algoritma yang diharapkan, bahkan sebelum Anda punya gambaran cara menyelesaikannya.

Aturan praktisnya sederhana. Mesin *judge* modern menjalankan sekitar 10^8 operasi C++ sederhana per detik. Bila algoritma Anda melakukan $f(N)$ langkah dasar pada masukan terbesar dan batas waktunya t detik, Anda menginginkan

$$f(N_{\max}) \leq 10^8 \cdot t \quad (1.1)$$

dengan cadangan yang lega, karena faktor konstanta, *cache miss*, dan masukan yang lambat dengan mudah menghabiskan faktor tiga. [Persamaan \(1.1\)](#) mengubah sebuah batasan menjadi daftar kompleksitas yang layak dicoba, yang dijabarkan **Tabel 1.2**.

Tabel 1.2 Membaca batasan: masukan terbesar memberi tahu kompleksitas mana yang lolos batas waktu satu detik.

N Terbesar	Kompleksitas aman	Teknik yang lazim
$N \leq 20$	$O(2^N \cdot N)$	<i>bitmask</i> , runut balik, <i>meet in the middle</i>
$N \leq 500$	$O(N^3)$	DP interval, Floyd–Warshall
$N \leq 5\,000$	$O(N^2)$	DP kuadratik, semua pasangan
$N \leq 200\,000$	$O(N \log N)$	pengurutan, pohon segmen, pohon Fenwick, pencarian biner atas jawaban
$N \leq 10^7$	$O(N)$	saringan, jumlah prefiks, dua penunjuk, tumpukan monoton
$N \leq 10^{18}$	$O(\log N)$ atau $O(\sqrt{N})$	pemangkatan cepat, pangkat matriks, DP digit

Mari lihat tabel itu bekerja. Misalkan sebuah soal memberi larik $N \leq 50\,000$ bilangan dan $M \leq 50\,000$ pertanyaan, masing-masing tentang satu rentang larik, dengan batas waktu satu detik. Menjawab setiap pertanyaan dengan menyisir rentangnya memakan hingga $N \cdot M = 2.5 \times 10^9$ langkah: jauh di atas anggaran. Jawaban $O(\log N)$ per pertanyaan memakan sekitar $50\,000 \times 16 = 8 \times 10^5$ langkah: dua ribu kali di bawah anggaran. Batasan itu baru saja menyuruh Anda mencari struktur data yang menjawab pertanyaan rentang dalam waktu logaritmik. Itulah tepatnya Bab 2.

Subsoal membawa pesan yang sama, satu tingkat lebih rendah. Subsoal “ $N \leq 1\,000$ ” berkata: pembuat soal menerima $O(N^2)$ di sini. Subsoal “semua pertanyaan memiliki $l = 1$ ” berkata:

jawaban prefiks sudah cukup di sini. Setiap subsoal adalah petunjuk, dan setiap petunjuk adalah poin.

1.5 Hari Lomba, Menit demi Menit

1.5.1 Menit 0 sampai 10: petakan semuanya

Jangan sentuh papan ketik. Bacalah setiap soal hari itu, ketiganya, termasuk subsoalnya. Untuk tiap soal, tulis empat baris di kertas: soal itu sebagai satu kalimat matematika (“diberikan larik dan rentang, laporkan subarray terbaik di dalam setiap rentang”); batasannya dan apa kata **Tabel 1.2** tentangnya; tebakan keluarga solusinya; dan daftar subsoal beserta poinnya, dengan tanda pada subsoal yang sudah bisa Anda selesaikan. Sepuluh menit ini terasa mahal. Justru inilah menit termurah sepanjang hari, karena mencegah Anda menuangkan dua jam ke soal tersulit sementara 40 poin yang mudah menunggu tanpa terbaca di halaman berikutnya.

1.5.2 Poin per menit

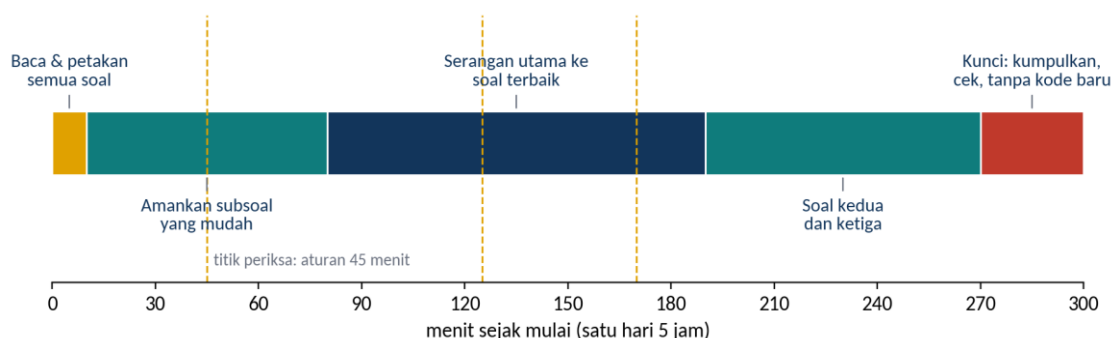
Dengan peta di tangan, urutkan pekerjaan menurut harapan poin per menit. Untuk setiap subsoal s yang Anda yakini bisa diselesaikan, perkirakan nilainya v_s dalam poin, menit c_s yang dibutuhkan, dan peluang p_s bahwa usaha Anda benar. Prioritasnya adalah

$$priority(s) = \frac{p_s \cdot v_s}{c_s} \quad (1.2)$$

dan Anda bekerja menurut prioritas yang menurun. Tidak ada yang menghitung **Persamaan** (1.2) sampai dua angka desimal di ruang lomba; nilainya terletak pada kebiasaan mengajukan pertanyaan itu. *Brute force* untuk subsoal 12 poin yang memakan sepuluh menit berprioritas sekitar 1,2 poin per menit; solusi penuh yang cerdas bernilai 60 poin tambahan yang mungkin memakan sembilan puluh menit, dengan peluang benar lima puluh persen, berprioritas sekitar 0,3. Kerjakan *brute force* lebih dulu. Program itu kelak juga menjadi pemeriksa solusi penuh, jadi nilainya lebih besar daripada poinnya.

1.5.3 Bentuk hari yang baik

Gambar 1.1 memperlihatkan rencana yang, dalam berbagai bentuk, diikuti banyak peserta kuat. Sepuluh menit pertama memetakan hari itu. Satu jam berikutnya mengamankan setiap subsoal mudah di ketiga soal. Lalu datang serangan utama ke soal yang menawarkan poin per menit terbesar, disusul soal kedua dan ketiga. Setengah jam terakhir dikunci: tidak ada gagasan baru, hanya mengumpulkan yang sudah siap, memastikan versi terbaik setiap program sudah terkumpul, dan membaca ulang format keluaran.



Gambar 1.1 Hari lomba lima jam: petakan, amankan subsoal mudah, serang, selesaikan soal lain, dan kunci setengah jam terakhir; garis putus-putus adalah titik periksa aturan 45 menit.

1.5.4 Aturan 45 menit

Cara paling umum kehilangan medali adalah jatuh cinta pada satu soal. Aturan yang melindungi Anda terdengar keras: bila Anda sudah 45 menit mengerjakan soal yang sama tanpa kemajuan yang terukur (subsoal baru diterima, pembuktian selesai, *bug* ditemukan), berhentilah. Tulis di kertas sampai di mana Anda, kumpulkan apa pun yang menghasilkan poin, lalu pindah. Anda boleh kembali nanti, dan sering kali memang kembali, dengan kepala yang segar. Garis putus-putus pada **Gambar 1.1** adalah saat untuk bertanya jujur kepada diri sendiri apakah 45 menit terakhir menghasilkan sesuatu.

1.5.5 Sebelum hari lomba

Seorang juara tidak menulis pohon segmen dari nol di ruang lomba; pohon itu mengalir dari jari karena sudah ditulis lima puluh kali saat berlatih. Simpan perangkat pribadi berisi struktur yang paling sering Anda pakai (masukan cepat, *disjoint-set union*, penelusuran melebar, Dijkstra, pohon segmen, pohon Fenwick, aritmetika modular) dan latih menulis ulang masing-masing dari ingatan dalam beberapa menit; Halim, Halim, dan Effendy [13] memberi daftar periksa rutinitas semacam itu yang lebih panjang. Simpan pula *buku catatan kesalahan*: setiap selesai berlatih, tulis satu kalimat tentang *bug* yang paling banyak menghabiskan waktu. Bacalah buku itu malam sebelum lomba. Isinya adalah peta jebakan pribadi Anda.

Rawatlah mesin yang berpikir. Tidurlah cukup pada dua malam terakhir, sarapan dengan benar, bawa air minum, dan datang cukup awal agar bisa duduk dengan tenang. Tidak ada yang misterius di sini, dan semuanya tampak pada jam keempat lomba lima jam.

1.6 Subsoal: Matematika Medali

Mari buat aritmetika poin parsial menjadi konkret. Misalkan satu hari berisi tiga soal, dan perhatikan dua peserta.

Ayu menyelesaikan soal A secara penuh (100 poin) dan menghabiskan sisa hari bergulat dengan soal B tanpa hasil. Nilainya 100.

Bima menulis subsoal *brute force* dari ketiga soal dalam sembilan puluh menit pertama: 20 poin di A, 15 di B, dan 12 di C. Kemudian ia menyelesaikan subsoal kedua soal A dengan gagasan rakus (30 poin lagi), dan dalam dua jam terakhir menemukan gagasan yang dimaksud untuk subsoal tengah soal C (35 poin lagi). Nilainya $20 + 15 + 12 + 30 + 35 = 112$.

Bima tidak pernah menyelesaikan satu soal pun secara penuh, dan ia mengalahkan Ayu. Dalam dua hari lomba dan dengan batas medali nasional sekitar sepertiga dari maksimum, seperti pada 2026 [6], pola inilah yang menentukan medali. Tiga kebiasaan mengikutinya:

- Subsoal N kecil hampir selalu berupa *brute force* yang bisa ditulis dalam sepuluh sampai lima belas menit. Tulis, kumpulkan, amankan.
- Simpan setiap versi yang diterima. Jangan pernah menimpa kode yang sudah benar dengan eksperimen; salin berkasnya dulu (`A_brute.cpp`, `A_v2.cpp`).
- *Brute force* yang sama menjadi acuan solusi cepat Anda dalam *stress test* (Subbab 1.8). Satu kode, dua kegunaan.

1.7 Pemanasan: Memotong Baris Menjadi Kata

Studi kasus bab ini membaca kalimat bahasa Inggris, jadi keterampilan pertama yang perlu dilatih adalah mengubah sebaris teks menjadi kata-kata dan satu tanda akhir. Kedengarannya sepele, padahal inilah sumber banyak jawaban salah: spasi ganda, kata yang menempel pada titiknya, baris tanpa tanda akhir, *carriage return* Windows di ujung baris. **Kode 1.1** mengerjakannya dengan satu kali lewat atas karakter-karakternya.

Kode 1.1 `tokenize.cpp`: memotong setiap baris menjadi kata dan melaporkan tanda akhir pertamanya.

```
// Warm-up: split each line into words and report its end mark.
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main() {
    string line;
    while (getline(cin, line)) {
        vector<string> words;
        string word;
        char end = '-'; // '-' means: no end mark found
        for (char c : line) {
            if (c == '.' || c == '?' || c == '!' || c == ' ') {
                if (!word.empty()) words.push_back(word);
                word.clear();
                if (c != ' ') { end = c; break; }
            } else {
                word += c;
            }
        }
        cout << "end=" << end << " words=" << words.size() << ":";
        for (const string& w : words) cout << " [" << w << " ]";
        cout << "\n";
    }
    return 0;
}
```

Perulangan menyimpan kata yang sedang dibangun di `word`. Sebuah spasi, atau salah satu dari tiga tanda akhir, *menutup* kata itu: bila tidak kosong, kata masuk ke `words`. Tanda akhir juga menghentikan pemindaian, sehingga apa pun sesudah titik diabaikan. Pada masukan ini:

```
tom likes big cats.
who eats fish?
bye!
no mark here
```

program mencetak:

Perhatikan baris pertama: dua spasi di antara `tom` dan `likes` tidak menghasilkan kata kosong, karena `word` yang kosong tidak pernah dimasukkan. Perhatikan baris terakhir: kata `here` hilang. Kata itu tidak pernah ditutup, karena tidak diikuti spasi maupun tanda akhir. Untuk keperluan kita itu perilaku yang benar (baris tanpa tanda akhir bukan kalimat), tetapi justru rincian seperti inilah yang harus Anda putuskan dengan sengaja, bukan Anda temukan di *judge*.

1.8 Protokol Debug dan *Stress Test*

Ketika solusi gagal, rasa panik menyarankan perubahan acak. Protokol menggantikan panik dengan prosedur:

1. **Perkecil masukan yang gagal.** Susutkan sampai *bug*-nya telanjang, idealnya segelintir bilangan yang bisa Anda telusuri dengan tangan.
2. **Coba tepi-tepinya dengan sengaja.** $N = 1$; semua nilai sama; semua nilai negatif; nilai terbesar; rentang kosong bila diizinkan. Beberapa kasus ini menangkap sebagian besar jawaban salah.
3. **Periksa tipe data.** Setiap hasil kali atau jumlah yang bisa melewati sekitar 2×10^9 memerlukan `long long`. Tulis ini di kertas Anda pada awal setiap lomba.
4. **Pilih satu konvensi indeks seumur hidup.** Larik berbasis 0 atau berbasis 1, tidak pernah campuran dalam satu program.
5. **Lakukan *stress test*.** Bandingkan program cepat Anda dengan program lambat yang jelas benar pada banyak masukan acak kecil.

Langkah kelima paling ampuh, dan setiap studi kasus dalam buku ini memakainya. Anda memerlukan tiga program: solusi, *brute force* yang menyelesaikan soal yang sama dengan cara paling langsung yang terpikirkan (lambat tidak masalah pada masukan kecil), dan *pembangkit* yang mencetak masukan acak kecil dari sebuah *seed*. Sebuah skrip pendek menjalankannya dalam perulangan dan berhenti pada masukan pertama yang jawabannya berbeda. **Kode 1.2** adalah skrip yang dipakai di setiap bab.

Kode 1.2 `stress.sh`: kompilasi kedua program, lalu bandingkan pada masukan acak sampai keduanya berbeda.

```
#!/usr/bin/env bash
# Stress test: compare the solution with the brute force on random inputs.
# Usage: ./stress.sh PROBLEM [ROUNDS]      (e.g. ./stress.sh HOTLINE 500)
set -e
P=${1:?problem name}; N=${2:-500}
g++ -std=c++17 -O2 -Wall -Wextra -o sol "$P.cpp"
g++ -std=c++17 -O2 -Wall -Wextra -o brute brute.cpp
for ((i = 1; i <= N; i++)); do
    python3 gen.py "$i" > in.txt
    ./sol < in.txt > out_sol.txt
    ./brute < in.txt > out_brute.txt
    if ! cmp -s out_sol.txt out_brute.txt; then
        echo "DIFFERENT on seed $i"; cat in.txt; exit 1
    fi
done
echo "all $N tests identical"
```

Dua rincian penting. Pembangkit memakai penghitung perulangan sebagai *seed*, sehingga masukan yang gagal dapat diulang: `python3 gen.py 137` mencetak masukan yang sama besok. Dan masukannya *kecil*, sehingga ketika kedua program berbeda, masukannya cukup pendek

untuk ditelusuri dengan tangan. *Stress test* tidak membuktikan program benar; ia menemukan, dalam semenit, jenis kesalahan yang kalau tidak akan menghabiskan satu jam untuk dipelototi.

JEBAKAN: *Brute force* yang berbagi kode dengan solusi juga berbagi *bug*-nya. Tulis *brute force* dari pernyataan soal, dengan cara paling naif, tanpa melihat program cepatnya.

1.9 Studi Kasus SULIT: HOTLINE

1.9.1 Soal dengan kata-kata kita sendiri

Soal SPOJ HOTLINE [14] membayangkan sebuah layanan telepon (*hotline*) yang bercakap dengan seorang penelepon. Penelepon mengucapkan kalimat bahasa Inggris sederhana, satu per baris. Sebagian berupa *pernyataan*, diakhiri titik, seperti `tom likes fish.`, `I don't eat fish.`, atau `everybody sees big cats.` Sebagian berupa *pertanyaan*, diakhiri tanda tanya, seperti `does tom like fish?`, `who likes fish?`, atau `what does tom do?`. *Hotline* harus mengingat setiap pernyataan dan menjawab setiap pertanyaan hanya dengan apa yang sudah didengarnya, dalam tata bahasa yang ditetapkan persis oleh soal. Bila penelepon pernah mengucapkan dua hal yang tidak mungkin sama-sama benar, *hotline* menyerah: sejak saat itu setiap pertanyaan dijawab `I am abroad.` Baris yang diakhiri tanda seru mengakhiri dialog. Masukan berisi beberapa dialog; untuk setiap dialog program mencetak `Dialogue #k:`, lalu setiap pertanyaan diikuti jawabannya dan satu baris kosong, dan terakhir baris penutupnya diikuti satu baris kosong.

Kesulitannya bukan kecepatan: satu dialog berisi paling banyak sekitar seratus kalimat. Kesulitannya adalah kesetiaan pada aturan. Subjek `I` dan `you` memakai kata kerja tanpa `s`; subjek lain memakai `s`. `I` milik penelepon adalah `you` bagi *hotline*, sehingga jawabannya menukar keduanya. `everybody` dan `nobody` adalah subjek yang berbicara atas nama semua orang. Beberapa subjek digabung sebagai `A`, `B` and `C`. Satu koma yang hilang berarti jawaban salah, dan SPOJ tidak akan memberi tahu koma yang mana.

1.9.2 Contoh dialog

Berikut masukan dua dialog buatan kita untuk menguji sebagian besar aturan:

```
2
I like fish.
tom doesn't eat fish.
ann likes fish.
everybody sees big cats.
do you like fish?
does tom eat fish?
does ann eat fish?
who likes fish?
who sees big cats?
what does tom do?
see you!
nobody eats fish.
tom eats fish.
do I eat fish?
bye!
```

dan keluaran solusinya:

Beberapa aturan sudah terlihat. Penelepon berkata `I like fish.`, tetapi pertanyaan `do you like fish?` menanyakan *hotline* itu sendiri, yang tentangnya belum diketahui apa-apa: `maybe`. Jawaban atas `who likes fish?` menukar `I` penelepon menjadi `you` dan menggabungkan dua subjek dengan

and, dengan kata kerja tanpa *s* karena subjeknya jamak. *everybody sees big cats.* menjawab *who sees big cats?* sendirian, dan juga muncul dalam jawaban atas *what does tom do?*. Pada dialog kedua, *nobody eats fish.* dan *tom eats fish.* tidak mungkin sama-sama benar, sehingga pertanyaan berikutnya dijawab *I am abroad.*

1.9.3 Pemodelan: fakta sebagai kunci

Keputusan rancangan pertama adalah cara menyimpan ucapan penelepon. Setiap pernyataan, setelah dinormalkan, adalah sebuah *fakta* dengan empat bagian: subjek, tanda (positif atau negatif), kata kerja dalam bentuk dasar, dan objek (bisa kosong, bisa beberapa kata). *tom doesn't eat fish.* menjadi (*tom*, negatif, *eat*, *fish*); *nobody eats fish.* menjadi (*nobody*, negatif, *eat*, *fish*), karena “tak seorang pun makan” adalah pernyataan negatif tentang semua orang. Menormalkan kata kerja berarti membuang *s* orang ketiga: *likes* menjadi *like*. Untuk *I* dan *you* kata kerjanya sudah dalam bentuk dasar.

Agar pencarian cepat, setiap fakta juga ditulis sebagai kunci string *subject|verb|object* dan disimpan di salah satu dari dua kamus, *posMap* untuk fakta positif dan *negMap* untuk fakta negatif. Garis tegak adalah pemisah yang tidak mungkin muncul dalam kata, sehingga fakta yang berbeda tidak pernah menghasilkan kunci yang sama.

Kapan sebuah dialog kontradiktif? Tulis fakta positif sebagai *P* dan fakta negatif sebagai *Q*, keduanya tentang kata kerja dan objek yang sama. Keduanya bertentangan tepat ketika

$$P.\text{subj} = Q.\text{subj} \vee P.\text{subj} = \text{everybody} \vee Q.\text{subj} = \text{nobody} \quad (1.3)$$

Dengan kata lain: orang yang sama sekaligus melakukan dan tidak melakukan; atau semua orang melakukan sementara seseorang tidak; atau tak seorang pun melakukan sementara seseorang melakukan. *Persamaan* (1.3) simetris, sehingga dapat diperiksa *secara bertahap*: ketika fakta baru datang, cukup membandingkannya dengan fakta tersimpan yang bertanda berlawanan. Fakta positif baru dengan kunci *k* bertentangan bila *negMap* memuat *k*, atau memuat kunci dengan subjek *nobody*, atau, bila subjek barunya *everybody*, memuat kunci *apa saja* yang berakhiran *|verb|object*. Kasus negatifnya adalah bayangan cerminnya.

Menjawab pertanyaan ya/tidak *do X verb object?* cukup dengan dua pencarian. Bila fakta positif tentang *X* ada, atau fakta positif tentang *everybody*, jawabannya *yes, ...* Bila tidak, dan fakta negatif tentang *X* ada, atau fakta negatif tentang *nobody*, jawabannya *no, ...* Selain itu *maybe..* Pertanyaan *who* dan *what* memerlukan fakta menurut urutan pengucapannya, sehingga program juga menyimpan daftar *stmts* berisi fakta-fakta berbeda menurut urutan kemunculan.

1.9.4 Solusi lengkap

Kode 1.3 adalah solusi lengkapnya. Logikanya adalah logika solusi yang tercatat ACCEPTED di SPOJ, disusun ulang menjadi fungsi-fungsi kecil agar setiap aturan soal punya satu rumah.

Kode 1.3 *HOTLINE.cpp*: fakta sebagai kunci, pemeriksaan kontradiksi bertahap, dan satu fungsi untuk setiap jenis pertanyaan.

```

// HOTLINE (SPOJ): remember simple English statements, detect contradictions
// and answer do/does, who and what questions. C++17, ACCEPTED Logic.
#include <cstdio>
#include <cstring>
#include <map>
#include <set>
#include <string>
#include <vector>
using namespace std;

static char buf[128];

bool isIorYou(const string& s) { return s == "I" || s == "you"; }
// "like" for I/you, "likes" for everyone else
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"; }
// the speaker's "I" is the hotline's "you" and vice versa
string swapIY(const string& s) {
    if (s == "I") return "you";
    if (s == "you") return "I";
    return s;
}
string baseVerb(const string& v) { // "likes" -> "Like"
    if (!v.empty() && v.back() == 's') return v.substr(0, v.size() - 1);
    return v;
}
string joinFrom(const vector<string>& w, size_t from) { // words from..end
    string r;
    for (size_t i = from; i < w.size(); i++) {
        if (!r.empty()) r += " ";
        r += w[i];
    }
    return r;
}
string finish(string s, const string& obj) { // add " obj" (if any) and "."
    if (!obj.empty()) s += " " + obj;
    return s + ".";
}

struct Stmt { string subj; bool neg; string verb, obj; };

struct Dialogue {
    vector<Stmt> stmts; // distinct facts, in order of appearance
    map<string, bool> posMap, negMap; // keys "subject/verb/object"
    bool contradiction = false;
};

```

Kode 1.3 (Lanjutan) *HOTLINE.cpp: fakta sebagai kunci, pemeriksaan kontradiksi bertahap, dan satu fungsi untuk setiap jenis pertanyaan.*

```
bool anyKeyEndsWith(const map<string, bool>& m, const string& suffix) {
    for (const auto& kv : m) {
        const string& k = kv.first;
        if (k.size() >= suffix.size() &&
            k.compare(k.size() - suffix.size(), suffix.size(), suffix) == 0)
            return true;
    }
    return false;
}

void addStatement(const vector<string>& words) {
    string subj = words[0], verb, obj;
    bool neg = false;
    if (words[1] == "don't" || words[1] == "doesn't") {
        if (words.size() < 3) return;
        neg = true;
        verb = words[2];
        obj = joinFrom(words, 3);
    } else {
        verb = isIorYou(subj) ? words[1] : baseVerb(words[1]);
        obj = joinFrom(words, 2);
        if (subj == "nobody") neg = true;
    }
    string tail = "|" + verb + "|" + obj, key = subj + tail;
    map<string, bool>& mine = neg ? negMap : posMap;
    if (!mine.count(key)) {
        mine[key] = true;
        stmts.push_back({subj, neg, verb, obj});
    }
    if (contradiction) return;
    if (!neg) { // a "yes" clashes with the same "no" or with "nobody"
        if (negMap.count(key) || negMap.count("nobody" + tail))
            contradiction = true;
        if (subj == "everybody" && anyKeyEndsWith(negMap, tail)) contradiction = true;
    } else { // a "no" clashes with the same "yes" or with "everybody"
        if (posMap.count(key) || posMap.count("everybody" + tail))
            contradiction = true;
        if (subj == "nobody" && anyKeyEndsWith(posMap, tail)) contradiction = true;
    }
}
```

Kode 1.3 (Lanjutan) *HOTLINE.cpp: fakta sebagai kunci, pemeriksaan kontradiksi bertahap, dan satu fungsi untuk setiap jenis pertanyaan.*

```
string answerDo(const vector<string>& words) { // do/does X verb obj?
    string qsubj = words[1], qverb = words[2], qobj = joinFrom(words, 3);
    string tail = "|" + qverb + "|" + qobj, who = swapIY(qsubj);
    if (posMap.count(qsubj + tail) || posMap.count("everybody" + tail))
        return finish("yes, " + who + " " + verbS(qverb, who), qobj);
    if (negMap.count(qsubj + tail) || negMap.count("nobody" + tail))
        return finish("no, " + who + " " + dontS(who) + " " + qverb, qobj);
    return "maybe.";
}

// <P50>
// answerWho: List the subjects of positive facts with this verb and object
// (everybody wins; "A, B and C"; else nobody; else "I don't know.").
// answerWhat: List X's distinct facts, own plus everybody/nobody, in order.
// >>> YOUR TURN (Public Edition): write this part yourself, following the hint above.
<<<
// </P50>

string answer(const vector<string>& words) {
    if (contradiction) return "I am abroad.";
    if (words.empty()) return "maybe.";
    if ((words[0] == "do" || words[0] == "does") && words.size() >= 3)
        return answerDo(words);
    if (words[0] == "who" && words.size() >= 2) return answerWho(words);
    if (words[0] == "what" && words.size() >= 4) return answerWhat(words);
    return "maybe.";
}

void run() {
    vector<pair<string, string>> out; // (question, answer)
    while (fgets(buf, sizeof(buf), stdin)) {
        int len = (int)strlen(buf);
        while (len > 0 && (buf[len - 1] == '\n' || buf[len - 1] == '\r'))
            buf[--len] = 0;
        if (len == 0) continue;
        char end = 0;
        vector<string> words;
        string word;
        for (int i = 0; i < len && !end; i++) {
            char c = buf[i];
            if (c == '.' || c == '?' || c == '!' || c == ' ') {
                if (!word.empty()) words.push_back(word);
                word.clear();
                if (c != ' ') end = c;
            } else {
                word += c;
            }
        }
    }
}
```

Kode 1.3 (Lanjutan) *HOTLINE.cpp: fakta sebagai kunci, pemeriksaan kontradiksi bertahap, dan satu fungsi untuk setiap jenis pertanyaan.*

```
    }  
    if (end == '!') { // the dialogue is over: print everything  
        for (auto& qa : out)  
            printf("%s\n%s\n\n", qa.first.c_str(), qa.second.c_str());  
        printf("%s\n\n", buf);  
        return;  
    }  
    if (end == '.' && words.size() >= 2) addStatement(words);  
    else if (end == '?') out.push_back({buf, answer(words)});  
}  
};  
  
int main() {  
    int T;  
    if (scanf("%d", &T) != 1) return 0;  
    for (int d = 1; d <= T; d++) {  
        printf("Dialogue #d:\n", d);  
        Dialogue dlg;  
        dlg.run();  
    }  
    return 0;  
}
```

1.9.5 Baris demi baris

Fungsi bantu tata bahasa. `isIorYou`, `verbS`, dan `dontS` menyandikan satu-satunya tata bahasa Inggris yang dibutuhkan soal: **I** dan **you** memakai **like** dan **don't**, subjek lain memakai **likes** dan **doesn't**. `swapIY` mengubah sudut pandang penelepon menjadi sudut pandang *hotline*. `baseVerb` membuang satu **s** di akhir. `joinFrom` merangkai kata-kata yang tersisa menjadi objek, dan `finish` menambahkan objek (bila ada) dan titik, sehingga tidak ada jawaban yang lupa salah satunya.

Menyimpan pernyataan. `addStatement` menerima kata-kata dari kalimat yang diakhiri titik dan terdiri atas paling sedikit dua kata. Bila kata kedua **don't** atau **doesn't**, faktanya negatif, kata kerjanya kata ketiga, dan objeknya mulai dari kata keempat. Selain itu kata kerjanya kata kedua, dinormalkan kecuali subjeknya **I** atau **you**, dan subjek **nobody** membuat faktanya negatif. Fakta disimpan hanya bila kuncinya baru, sehingga kalimat yang diulang tidak terulang dalam jawaban berikutnya. Lalu, kecuali dialognya sudah kontradiktif, pemeriksaan bertahap **Persamaan** (1.3) dijalankan. `anyKeyEndsWith` mengerjakan bagian “subjek apa saja” dengan membandingkan ujung setiap kunci dengan `|verb|object`.

Pertanyaan ya/tidak. `answerDo` adalah aturan dua pencarian di atas. Perhatikan bahwa pencariannya memakai subjek pertanyaan apa adanya (**do you ...** mencari fakta tentang **you**), sedangkan jawabannya memakai subjek yang sudah ditukar.

Who dan what. Di antara penanda `// <P50>` dan `// </P50>` terletak dua jawaban yang menyusun daftar. `answerWho` memindai fakta menurut urutannya. Fakta positif tentang **everybody** langsung menuntaskan pertanyaan. Selain itu, subjek fakta-fakta positif dikumpulkan masing-masing sekali, ditukar ke sudut pandang *hotline*, dan digabung dengan koma dan **and** di akhir; satu subjek menentukan bentuk kata kerjanya, beberapa subjek memakai bentuk jamak. Bila tidak ada yang diketahui melakukannya tetapi **nobody ...** pernah diucapkan, jawabannya **nobody ...**; selain itu **I don't know**. `answerWhat` mengumpulkan, menurut urutan, setiap fakta berbeda yang subjeknya orang yang ditanyakan, **everybody**, atau **nobody**, lalu menuliskannya sebagai satu

kalimat: tindakan pertama mengikuti subjek, yang lain dipisahkan koma, dan yang terakhir diawali , and.

Perulangan dialog. `run` membaca baris dengan `fgets`, membuang akhir baris (termasuk `\r` Windows), dan melewati baris kosong. Baris dipotong menjadi kata persis seperti **Kode 1.1**. Tanda `!` mengakhiri dialog: program mencetak semua pertanyaan dan jawaban yang tersimpan, lalu baris penutup. Jawaban disimpan alih-alih langsung dicetak karena keluaran setiap dialog dicetak sebagai satu blok sesudah baris terakhirnya dibaca. Akhirnya, `main` membaca banyaknya dialog dan menjalankan objek `Dialogue` baru untuk masing-masing, sehingga tidak ada fakta yang bocor dari satu dialog ke dialog berikutnya.

1.9.6 Jebakan

- **Mencampur sudut pandang.** `I` milik penelepon disimpan sebagai `I`, dan pertanyaan `do I ...` mencari `I`; hanya *jawabannya* yang berkata `you`. Menukar pada saat menyimpan justru membuat separuh pertanyaan terjawab salah.
- **Lupa bahwa `nobody` bermakna negatif.** `nobody eats fish.` tidak mengandung `doesn't`, tetapi faktanya negatif; melewatkannya merusak pemeriksaan kontradiksi sekaligus jawaban `who`.
- **Menjawab sesudah kontradiksi.** Begitu penandanya menyala, *setiap* pertanyaan berikutnya dalam dialog itu dijawab `I am abroad.`, termasuk pertanyaan yang jawabannya diketahui.
- **Duplikat.** Pernyataan yang diucapkan dua kali harus muncul sekali dalam jawaban `who` dan `what`.
- **Tata letak keluaran.** Setiap pertanyaan dan jawabannya diikuti satu baris kosong, begitu pula baris penutup. *Judge* membandingkan keluaran secara persis.
- **Keadaan basi.** Fakta dari dialog 1 tidak boleh menjawab pertanyaan di dialog 2. Membuat objek `Dialogue` baru untuk setiap dialog menjamin hal ini secara otomatis.

1.9.7 Stress test melawan brute force

Apa itu *brute force* untuk soal membaca? Bukan algoritma yang lebih lambat, melainkan implementasi *mandiri* dari aturan yang sama, ditulis dengan cara paling langsung. Berkas `brute.cpp` di folder bab ini menyimpan daftar fakta biasa, memeriksa ulang setiap pasangan fakta dengan *Persamaan* (1.3) sesudah setiap pernyataan, dan menjawab setiap pertanyaan dengan memindai daftar itu. Berkas itu tidak berbagi kode dengan **Kode 1.3**. Pembangkit **Kode 1.4** menyusun dialog acak dari kosakata mungil (empat orang, tiga kata kerja, tiga objek), sehingga fakta berulang, `everybody`, `nobody`, dan kontradiksi sering muncul.

Kode 1.4 `gen.py`: dialog HOTLINE acak dari kosakata empat orang dan tiga kata kerja.

```
#!/usr/bin/env python3
"""Random HOTLINE dialogues from a tiny vocabulary: python3 gen.py SEED"""
import random
import sys

rng = random.Random(int(sys.argv[1]) if len(sys.argv) > 1 else 1)
PEOPLE = ["I", "you", "tom", "ann"]
VERBS = ["like", "eat", "see"]
OBS = ["", "fish", "big cats"]

def verb_for(subj, verb):
    return verb if subj in ("I", "you") else verb + "s"

def statement():
    v, o = rng.choice(VERBS), rng.choice(OBS)
    r = rng.random()
    if r < 0.08:
        words = ["everybody", v + "s"]
    elif r < 0.14:
        words = ["nobody", v + "s"]
    else:
        s = rng.choice(PEOPLE)
        if rng.random() < 0.3:
            words = [s, "don't" if s in ("I", "you") else "doesn't", v]
        else:
            words = [s, verb_for(s, v)]
    return " ".join(words + ([o] if o else [])) + "."

def question():
    v, o = rng.choice(VERBS), rng.choice(OBS)
    tail = [o] if o else []
    kind = rng.randrange(3)
    s = rng.choice(PEOPLE)
    if kind == 0:
        aux = "do" if s in ("I", "you") else "does"
        return " ".join([aux, s, v] + tail) + "?"
    if kind == 1:
        return " ".join(["who", v + "s"] + tail) + "?"
    aux = "do" if s in ("I", "you") else "does"
    return " ".join(["what", aux, s, "do"]) + "?"
```

Kode 1.4 (Lanjutan) *gen.py*: dialog HOTLINE acak dari kosakata empat orang dan tiga kata kerja.

```
T = rng.randint(1, 3)
print(T)
for _ in range(T):
    for _ in range(rng.randint(1, 12)):
        print(statement() if rng.random() < 0.6 else question())
    print("bye!")
```

Menjalankan skrip *stress test* sebanyak 300 putaran mencetak:

```
all 300 tests identical
```

Kedua program juga sepakat dengan kode ACCEPTED aslinya pada masukan yang sama, yang menegaskan bahwa penyusunan ulang itu tidak mengubah apa pun. Pemeriksaan pamungkasnya, tentu saja, adalah pengumpulan ke SPOJ sendiri.

1.10 Taktik Hari Lomba

- **Petakan selama sepuluh menit.** Baca semua soal dan semua subsoal sebelum menulis kode; tulis satu baris matematika untuk setiap soal.
- **Amankan subsoal murah lebih awal.** Subsoal *brute force* setiap soal biasanya bernilai sepuluh sampai dua puluh poin dan memakan sepuluh sampai lima belas menit.
- **Patuhi aturan 45 menit.** Tanpa kemajuan terukur dalam 45 menit berarti Anda pindah.
- **Simpan setiap versi yang diterima.** Salin sebelum bereksperimen.
- **Lakukan stress test sebelum percaya.** *Brute force* ditambah pembangkit ditambah **Kode 1.2** menangkap kesalahan yang tidak pernah diperlihatkan contoh soal.
- **Baca format keluaran dua kali.** Sekali di awal, sekali di setengah jam terakhir yang dikunci. HOTLINE memperlihatkan bagaimana satu koma bisa menghabiskan satu soal utuh.
- **Kunci tiga puluh menit terakhir.** Kumpulkan, periksa, dan tahan godaan gagasan baru yang cemerlang.

Ketika panik datang, dan dalam lomba lima jam ia pasti datang, tulislah dua daftar di kertas: apa yang pasti benar (sudah diuji, sudah diterima) dan apa yang belum. Panik lahir dari mencampur keduanya. Memisahkannya dengan tinta mengubah perasaan itu kembali menjadi rencana.

1.11 Ringkasan

- OSN informatika memiliki tiga tahap (kabupaten/kota, provinsi, nasional); babak nasional terdiri atas dua hari lima jam berisi tiga soal gaya IOI dengan subsoal, dan pada 2026 medali membutuhkan kira-kira sepertiga dari 600 poin yang tersedia.
- Pengumpulan terbaik yang dihitung per subsoal, sehingga mengumpulkan tidak pernah merugikan dan solusi parsial sangat berharga.
- Batasan soal, dibaca melalui [Persamaan \(1.1\)](#) dan **Tabel 1.2**, memberi tahu kompleksitas mana yang dapat lolos sebelum Anda tahu solusinya.
- Urutkan pekerjaan menurut harapan poin per menit, [Persamaan \(1.2\)](#), dan lindungi diri dengan aturan 45 menit serta setengah jam terakhir yang dikunci.
- *Stress test* membandingkan solusi cepat dengan *brute force* mandiri pada banyak masukan acak kecil dan menemukan sebagian besar *bug* dalam hitungan menit.
- HOTLINE diselesaikan dengan memodelkan setiap kalimat sebagai fakta bertanda, menyimpan fakta sebagai kunci, memeriksa kontradiksi secara bertahap dengan [Persamaan \(1.3\)](#), dan memberi setiap jenis pertanyaan fungsi kecilnya sendiri.

1.12 Soal Latihan

1. Sebuah soal memiliki $N \leq 3\,000$ dan batas waktu dua detik. Dengan [Persamaan \(1.1\)](#), manakah di antara $O(N^3)$, $O(N^2 \log N)$, dan $O(N^2)$ yang aman?
2. Satu hari menawarkan subsoal berikut: A1 bernilai 15 poin (10 menit, Anda yakin), A2 bernilai 40 (80 menit, peluang separuh), B1 bernilai 20 (25 menit, cukup yakin, katakanlah 0,8), C1 bernilai 10 (20 menit, yakin). Urutkan menurut [Persamaan \(1.2\)](#).
3. Jalankan **Kode 1.1** dengan tangan pada baris `who sees?big cats.` Berapa kata yang dilaporkan, dan tanda akhir apa?
4. Pada HOTLINE diberikan dialog `everybody likes fish. / ann doesn't like fish. / does tom like fish? / bye!`. Apa yang dicetak program, dan mengapa?

5. Apa jawaban solusi atas `who eats fish?` sesudah `tom eats fish.`, `I eat fish.`, `ann eats fish.`, dan `tom eats fish.`? Tulis jawabannya secara persis.
6. Ubah pembangkit agar tidak pernah menghasilkan kontradiksi. Mengapa hal itu membuat *stress test* lebih lemah?
7. Jelaskan mengapa pengumpulan yang nilainya lebih rendah daripada pengumpulan sebelumnya tidak pernah menurunkan nilai total Anda di babak nasional.

1.13 Penyelesaian Soal Latihan

1. Anggarannya $10^8 \cdot 2 = 2 \times 10^8$. $N^3 = 2.7 \times 10^{10}$: terlalu lambat. $N^2 \log N \approx 9 \times 10^6 \times 12 \approx 1.1 \times 10^8$: masih dalam anggaran, tetapi dengan cadangan tipis bila konstantanya besar. $N^2 = 9 \times 10^6$: sangat aman.
2. Prioritasnya: A1 $1 \cdot 15/10 = 1.5$; C1 $1 \cdot 10/20 = 0.5$; B1 $0.8 \cdot 20/25 = 0.64$; A2 $0.5 \cdot 40/80 = 0.25$. Urutannya: A1, B1, C1, A2.
3. Karakternya `who`, dua spasi, `sees`, lalu `?`. Spasi menutup `who` (spasi kedua mendapati `word` kosong dan tidak menambahkan apa pun); tanda tanya menutup `sees` dan menghentikan pemindaian. Keluaran: `end=? words=2: [who] [sees]`. Teks sesudah `?` diabaikan.
4. `ann doesn't like fish.` bertentangan dengan `everybody likes fish.` menurut kasus kedua *Persamaan* (1.3), sehingga penandanya menyala sebelum pertanyaan datang. Program mencetak `Dialogue #1:`, lalu `does tom like fish?`, lalu `I am abroad.`, satu baris kosong, `bye!`, dan satu baris kosong.
5. Subjek positif yang berbeda menurut urutan kemunculan: `tom`, `I`, `ann` (`tom` yang berulang diabaikan). Setelah ditukar: `tom`, `you`, `ann`. Tiga subjek memakai kata kerja jamak: `tom`, `you` and `ann eat fish.`
6. Kontradiksi memicu aturan `I am abroad.` dan seluruh kode pemeriksaan kontradiksi, termasuk `anyKeyEndsWith`. Pembangkit tanpa kontradiksi tidak akan pernah menguji baris-baris itu, sehingga *bug* di sana lolos di setiap putaran *stress test*.
7. Karena nilai setiap subsoal adalah maksimum di antara semua pengumpulan [5]. Pengumpulan yang lebih buruk hanya menambahkan bilangan yang lebih kecil ke sebuah maksimum yang sudah memuat bilangan yang lebih besar.

Daftar Pustaka

- [1] *Pemrograman Kompetitif dengan SPOJ: Dari Membaca Soal hingga Accepted, dengan C++ dan Python*. Profitina, 2026.
- [2] Tim Olimpiade Komputer Indonesia, "Silabus OSN 2026 — OSN Bidang Informatika." [Daring]. Tersedia: <https://osn.toki.id/silabus> (diakses Sep. 2026).
- [3] Suara Pecari, "Panduan Lengkap OSN 2026: Jadwal, Aturan, dan Komponen," Jun. 2026. [Daring]. Tersedia: <https://suarapecari.com/pendidikan/sp-2026060535876/panduan-lengkap-osn-2026-jadwal-aturan-dan-komponen/> (diakses Sep. 2026).
- [4] Tim Olimpiade Komputer Indonesia, "OSN 2026 Bidang Informatika, Malang, 14–20 September 2026 — Peringkat." [Daring]. Tersedia: <https://peringkat.toki.id/osn-2026/> (diakses Sep. 2026).
- [5] Tim Olimpiade Komputer Indonesia, "Peraturan — OSN Bidang Informatika." [Daring]. Tersedia: <https://osn.toki.id/OSN2022/peraturan> (diakses Sep. 2026).
- [6] Tim Olimpiade Komputer Indonesia, "Olimpiade Sains Nasional 2026 Bidang Informatika — Hasil." [Daring]. Tersedia: <https://osn.toki.id/OSN2026/hasil> (diakses Sep. 2026).
- [7] Tim Olimpiade Komputer Indonesia, "About." [Daring]. Tersedia: <https://toki.id/about/> (diakses Sep. 2026).

- [8] International Olympiad in Informatics, “IOI 2026 — Tashkent, Uzbekistan,” IOI Statistics. [Daring]. Tersedia: <https://stats.ioinformatics.org/olympiads/2026> (diakses Sep. 2026).
- [9] Kaldera News, “Membanggakan! 4 Pelajar Indonesia Bawa Pulang Medali dari IOI 2026,” Agu. 2026. [Daring]. Tersedia: <https://www.kalderanews.com/2026/08/membanggakan-4-pelajar-indonesia-bawa-pulang-medali-dari-ioi-2026/> (diakses Sep. 2026).
- [10] MSYS2 Developers, “Package: mingw-w64-ucrt-x86_64-gcc,” versi 16.2.0. [Daring]. Tersedia: https://packages.msys2.org/packages/mingw-w64-ucrt-x86_64-gcc (diakses Sep. 2026).
- [11] GNU Project, “GCC, the GNU Compiler Collection,” daftar rilis dengan GCC 16.2 (7 Agu. 2026). [Daring]. Tersedia: <https://gcc.gnu.org/> (diakses Sep. 2026).
- [12] Microsoft, “Visual Studio Code Updates, version 1.139,” Sep. 2026. [Daring]. Tersedia: <https://code.visualstudio.com/updates> (diakses Sep. 2026).
- [13] S. Halim, F. Halim, dan S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.
- [14] Sphere Online Judge, “HOTLINE — Hotline.” [Daring]. Tersedia: <https://www.spoj.com/problems/HOTLINE/> (diakses Sep. 2026).

Bab 2

Pohon Segmen dan Jawaban yang Dapat Digabung — GSS1

Hampir setiap olimpiade nasional memuat soal yang mengajukan jenis pertanyaan yang sama berulang-ulang tentang sebuah *rentang* larik: jumlah posisi l sampai r , nilai terkecil di antaranya, atau sesuatu yang lebih aneh. *Segment tree* (pohon segmen) adalah alat baku untuk semuanya, dan keterampilan sesungguhnya yang dituntut bukan pohon itu sendiri, yang pendek, melainkan rancangan isi setiap simpulnya. Di akhir bab ini Anda akan mampu membangun pohon segmen dalam waktu linear, menjawab pertanyaan rentang dalam waktu logaritmik beserta alasannya, dan, yang terpenting, merancang simpul untuk pertanyaan yang tidak lazim dengan bertanya informasi apa yang hilang ketika dua jawaban bertetangga direkatkan. Studi kasusnya adalah soal klasik SPOJ GSS1, subarray berjumlah terbesar di dalam sebuah rentang, yang simpul empat-medannya termasuk rancangan kecil paling berguna dalam pemrograman kompetitif.

2.1 Buku Kas Koperasi

Sebuah koperasi petani buah di perbukitan mencatat pendapatan bersih hariannya dalam buku kas: uang hasil penjualan hari itu dikurangi biaya hari itu. Ada hari yang baik (satu truk manggis terjual di kota), ada hari yang buruk (hujan, pompa rusak, buah busuk di perjalanan). Bendahara sering ditanya pertanyaan yang sama dalam berbagai bentuk. “Antara 3 dan 20 Maret, rangkaian hari berturut-turut mana yang paling baik?” “Pada musim durian, dari hari ke-150 sampai ke-210, deretan hari mana yang paling menguntungkan?” Jawabannya selalu jumlah terbesar dari satu blok hari berturut-turut di dalam periode itu.

Untuk satu periode, bendahara cukup mencoba setiap blok: setiap hari awal dan setiap hari akhir. Untuk buku kas beberapa ribu hari dan segelintir pertanyaan, itu tidak masalah. Namun koperasi menginginkan aplikasi kecil tempat anggota dapat menanyakan periode apa saja, sementara buku kas bertambah setiap hari. Dengan N hari dan M pertanyaan, mencoba setiap blok memakan hingga N^2 langkah per pertanyaan, dan bahkan metode satu lintasan yang lebih cerdas pada Subbab 2.3 memakan N langkah per pertanyaan, $N \cdot M$ secara keseluruhan. Dengan lima puluh ribu hari dan lima puluh ribu pertanyaan, itu 2.5×10^9 langkah, jauh melampaui satu detik. Kita memerlukan struktur yang sudah mengerjakan sebagian besar pekerjaan di muka dan dapat menjawab setiap pertanyaan dalam sekitar $\log_2 N$ langkah.

Itulah tepatnya soal SPOJ GSS1 [1], dan alatnya adalah pohon segmen.

2.2 Gagasannya dengan Kata-Kata Sederhana

Belah larik menjadi dua bagian. Belah lagi setiap bagian menjadi dua, dan seterusnya, sampai setiap potongan berisi satu elemen. Potongan-potongan itu membentuk pohon biner: akarnya seluruh larik, dua anaknya adalah kedua belahan, cucunya adalah perempatan, dan daunnya elemen tunggal. Sekarang simpan, di setiap potongan, *jawaban* pertanyaan kita untuk potongan itu.

Dua fakta membuat hal ini berguna. Pertama, jawaban untuk sebuah potongan dapat dihitung dari jawaban kedua belahannya, asalkan jawaban itu membawa informasi yang tepat; perhitungan itu kita sebut **combine**. Kedua, rentang $[l, r]$ apa pun dapat dipotong menjadi sedikit potongan pohon, tidak pernah lebih dari sekitar $2\log_2 N$, sehingga jawaban rentang diperoleh dengan menggabungkan beberapa jawaban tersimpan. Membangun pohon memakan satu **combine** per potongan, dan banyaknya potongan kurang dari $2N$. Menjawab satu pertanyaan memakan beberapa puluh **combine**. Seluruh kesulitan soal yang berat berpindah ke satu pertanyaan: apa yang harus disimpan sebuah potongan agar **combine** mungkin dilakukan?

2.3 Soal Pemanasan

2.3.1 Jumlah rentang tanpa pembaruan

Sebelum membangun pohon, ingat alat rentang yang paling sederhana. Bila pertanyaannya hanya “jumlah $a_l, \dots, a_r$ ” dan lariknya tidak pernah berubah, jumlah prefiks menjawab setiap pertanyaan dalam waktu konstan. Definisikan

$$p_0 = 0, \quad p_i = p_{i-1} + a_i \quad (1 \leq i \leq N) \quad (2.1)$$

sehingga p_i adalah jumlah i elemen pertama. Maka jumlah $a_l, \dots, a_r$ adalah $p_r - p_{l-1}$: r elemen pertama dikurangi $l - 1$ elemen pertama. **Kode 2.1** mengimplementasikan **Persamaan (2.1)**.

Kode 2.1 `prefix_sum.cpp`: jumlah rentang statis dalam waktu konstan per pertanyaan.

```
// Warm-up: static range sums with a prefix array, O(1) per question.
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n, m;
    if (scanf("%d", &n) != 1) return 0;
    vector<long long> p(n + 1, 0); // p[i] = a[1] + ... + a[i], p[0] = 0
    for (int i = 1; i <= n; i++) {
        long long v;
        if (scanf("%lld", &v) != 1) return 0;
        p[i] = p[i - 1] + v;
    }
    if (scanf("%d", &m) != 1) return 0;
    while (m--) {
        int l, r;
        if (scanf("%d %d", &l, &r) != 2) return 0;
        printf("%lld\n", p[r] - p[l - 1]);
    }
    return 0;
}
```

Pada larik 3, -4, 2, 5, -1, 2 dengan pertanyaan [1,6], [2,3], dan [1,2]:

```
6
3 -4 2 5 -1 2
3
1 6
2 3
1 2
```

Jumlah prefiks tak tertandingi bila dapat dipakai, dan pohon segmen yang menjawab jumlah statis adalah usaha yang sia-sia. Ingat ini ketika sebuah soal tampak seperti soal pohon segmen:

tanyakan dulu apakah trik prefiks sudah cukup. Trik itu tidak lagi cukup dalam dua keadaan: ketika larik berubah (Bab 5), dan ketika pertanyaannya bukan jumlah yang dapat “dikurangkan”. Blok hari berturut-turut terbaik adalah pertanyaan semacam itu: tidak ada cara mengurangi blok terbaik dari $l - 1$ hari pertama dari blok terbaik dari r hari pertama.

2.3.2 Blok terbaik seluruh larik

Untuk seluruh larik, blok terbaik memiliki solusi satu lintasan yang terkenal, biasanya dikaitkan dengan Kadane dan dipopulerkan oleh Bentley [2]. Berjalanlah dari kiri ke kanan sambil menyimpan e_i , jumlah terbaik blok yang *berakhir tepat di* posisi i . Blok semacam itu bisa berupa a_i saja, atau memperpanjang blok terbaik yang berakhir di $i - 1$:

$$e_1 = a_1, \quad e_i = \max(a_i, e_{i-1} + a_i) \quad (i \geq 2) \quad (2.2)$$

Jawabannya adalah e_i terbesar. **Kode 2.2** mengimplementasikan [Persamaan \(2.2\)](#) dengan satu variabel untuk e dan satu untuk yang terbaik.

Kode 2.2 `kadane.cpp`: blok tak kosong terbaik dari seluruh larik dalam satu lintasan.

```
// Warm-up: Kadane's algorithm, best non-empty subarray of the whole array.
#include <algorithm>
#include <cstdio>
using namespace std;

int main() {
    int n;
    if (scanf("%d", &n) != 1) return 0;
    long long endHere = 0, best = 0;
    for (int i = 0; i < n; i++) {
        long long v;
        if (scanf("%lld", &v) != 1) return 0;
        // best piece ending at i: extend the previous one, or start again at i
        endHere = (i == 0) ? v : max(v, endHere + v);
        best = (i == 0) ? endHere : max(best, endHere);
    }
    printf("%lld\n", best);
    return 0;
}
```

Pada 3, -4, 2, 5, -1, 2 program menemukan 8 (blok 2, 5, -1, 2), dan pada larik bilangan negatif -3, -1, -7, -2 menemukan -1, nilai tunggal yang paling tidak negatif:

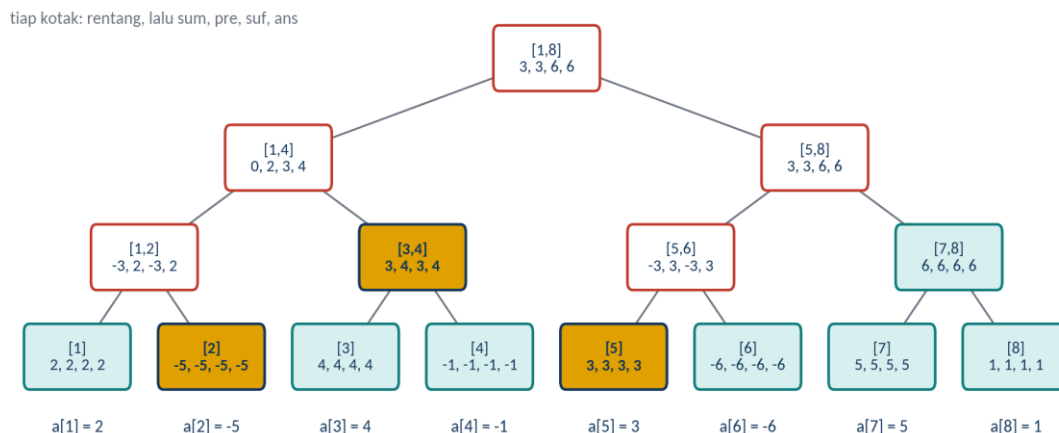
Keluaran kedua itu penting. Soal meminta blok yang *tidak kosong*, jadi ketika semua bilangan negatif, jawabannya negatif. “Penyederhanaan” yang sering dilakukan, yaitu memulai nilai terbaik dari 0, akan mencetak 0 di sana, yaitu jumlah blok kosong, dan itu jawaban salah. Metode Kadane menjawab satu rentang dalam $O(N)$; sisa bab ini menjawab rentang apa pun dalam $O(\log N)$.

2.4 Pohon Segmen

2.4.1 Bentuk dan penomoran

Nomori simpul seperti pada *binary heap*: akarnya simpul 1 dan meliputi $[1, N]$; simpul v yang meliputi $[lo, hi]$ dengan $lo < hi$ memiliki titik tengah $mid = \lfloor (lo + hi)/2 \rfloor$, anak kiri $2v$ yang meliputi $[lo, mid]$, dan anak kanan $2v + 1$ yang meliputi $[mid + 1, hi]$. Simpul dengan $lo = hi$ adalah daun. Tidak ada penunjuk yang disimpan; anak-anak v ditemukan dengan aritmetika.

Gambar 2.1 memperlihatkan pohon untuk larik delapan elemen 2, -5, 4, -1, 3, -6, 5, 1. Setiap kotak menampilkan rentangnya dan empat bilangan yang akan dibawa simpul kita (Subbab 2.5); abaikan dulu bilangan itu dan perhatikan bentuknya.



Gambar 2.1 Pohon segmen dari 2, -5, 4, -1, 3, -6, 5, 1; kueri [2,5] mengunjungi simpul merah dan menggabungkan tiga simpul kuning [2], [3,4], dan [5].

Seberapa besar larik simpul harus disiapkan? Kedalaman pohon adalah $\lceil \log_2 N \rceil$, dan dengan penomoran *heap* indeks terbesar yang dipakai berada di bawah $2^{\lceil \log_2 N \rceil + 1}$. Karena $2^{\lceil \log_2 N \rceil} < 2N$,

$$\text{indeks terbesar} < 2 \cdot 2^{\lceil \log_2 N \rceil} < 4N \quad (2.3)$$

itulah sebabnya kode pohon segmen mendeklarasikan `tree[4 * N]`. Batas pada [Persamaan \(2.3\)](#) bukan takhayul; menyiapkan hanya $2N$ simpul berhasil bila N perpangkatan dua dan gagal untuk nilai lain, dan inilah salah satu *runtime error* paling umum dalam lomba.

2.4.2 Membangun dalam waktu linear

`build(v, lo, hi)` mengisi simpul v : daun mendapat nilai elemennya; simpul lain membangun anak-anaknya dulu lalu menggabungkannya. Setiap simpul dikunjungi sekali, dan banyaknya simpul nyata kurang dari $2N$ (pohon dengan N daun yang setiap simpul dalamnya beranak dua memiliki tepat $N - 1$ simpul dalam), sehingga pembangunan memakan $O(N)$ panggilan `combine`.

2.4.3 Kueri dalam waktu logaritmik

`query(v, lo, hi, ql, qr)` mengembalikan jawaban untuk bagian $[ql, qr]$ yang berada di dalam rentang simpul v , yang dijamin pemanggilnya tidak kosong. Ada tiga kasus:

- Bila $[lo, hi]$ sepenuhnya berada di dalam $[ql, qr]$, simpul tersimpan itulah jawabannya.
- Bila kueri sepenuhnya berada di satu anak ($qr \leq mid$ atau $ql > mid$), tanyakan hanya kepada anak itu.
- Selain itu kueri melintasi titik tengah: tanyakan kepada kedua anak dan gabungkan dua jawaban parsialnya dengan `combine`, kiri lebih dulu.

Mengapa ini cepat? Lihat satu tingkat pohon mana pun. Simpul-simpul pada tingkat itu yang dikunjungi kueri *tanpa terliput penuh* pasti memuat salah satu ujung rentang kueri, ql atau qr , karena simpul yang tidak memuat kedua ujung itu pasti sepenuhnya di dalam atau sepenuhnya di luar rentang. Jadi paling banyak ada dua simpul semacam itu per tingkat, dan masing-masing membuat paling banyak dua panggilan ke tingkat di bawahnya. Akibatnya paling banyak empat simpul dikunjungi per tingkat, dan banyaknya simpul yang dikunjungi memenuhi

$$\text{dikunjungi}(N) \leq 4\lceil \log_2 N \rceil + 1 = O(\log N) \quad (2.4)$$

Simpul kuning pada **Gambar 2.1** adalah potongan *kanonik* dari $[2,5]$: $[2]$, $[3,4]$, dan $[5]$. Simpul merah adalah yang dikunjungi di sepanjang jalan. Bahkan untuk $N = 50\,000$, **Persamaan** (2.4) menjanjikan kurang dari tujuh puluh simpul yang dikunjungi.

WAWASAN KUNCI: Tidak ada bagian dari pembangunan maupun kueri yang tahu pertanyaannya apa. Semua yang khas untuk soal tinggal di dua fungsi: `makeNode` (jawaban untuk satu elemen) dan `combine` (jawaban untuk dua tetangga). Pohon segmen adalah mesin; simpul adalah program yang Anda umpankan kepadanya.

2.5 Merancang Simpul

2.5.1 Resep ketertutupan

Soal sulit tidak pernah menanyakan jumlah rentang biasa. Soal itu menanyakan sesuatu yang tidak lazim, dan tugas kita adalah menemukan simpul yang membuat jawaban tak lazim itu *dapat digabung*. Resepnya:

1. Tuliskan apa yang ditanyakan untuk sebuah rentang.
2. Cobalah menghitungnya untuk gabungan dua rentang bertetangga L dan R hanya dari jawaban keduanya. Tanyakan: informasi apa yang *hilang*?
3. Tambahkan informasi yang hilang itu ke simpul.
4. Periksa apakah medan-medan baru itu sendiri dapat digabung. Bila tidak, kembali ke langkah 2 untuk medan-medan itu.

Berhentilah ketika tidak ada lagi yang hilang. Simpul itu kini *tertutup* terhadap penggabungan.

2.5.2 Empat medan untuk blok terbaik

Terapkan resep itu pada blok berjumlah terbesar. Langkah 1: simpan *ans*, jumlah terbaik blok tak kosong di dalam rentang. Langkah 2: blok terbaik dari $L \cup R$ berada di dalam L , di dalam R , atau *melintasi* batasnya. Dua yang pertama adalah $L.\text{ans}$ dan $R.\text{ans}$. Blok yang melintas adalah blok yang berakhir di tepi kanan L disambung blok yang dimulai di tepi kiri R , dan nilai terbaiknya adalah *sufiks* terbaik L ditambah *prefiks* terbaik R . Keduanya belum tersimpan. Langkah 3: tambahkan *suf* (blok tak kosong terbaik yang menyentuh tepi kanan) dan *pre* (blok tak kosong terbaik yang menyentuh tepi kiri). Langkah 4: dapatkah *pre* digabung? Prefiks terbaik $L \cup R$ tetap di dalam L ($L.\text{pre}$) atau mengambil *seluruh* L lalu berlanjut ke R ($L.\text{sum} + R.\text{pre}$). Itu memerlukan total *sum*. Jumlah digabung dengan menjumlahkan. Tidak ada lagi yang hilang, dan keempat medannya adalah:

$$\text{sum} = L.\text{sum} + R.\text{sum} \quad (2.5)$$

$$\text{pre} = \max(L.\text{pre}, L.\text{sum} + R.\text{pre}) \quad (2.6)$$

$$\text{suf} = \max(R.\text{suf}, R.\text{sum} + L.\text{suf}) \quad (2.7)$$

$$\text{ans} = \max(L.\text{ans}, R.\text{ans}, L.\text{suf} + R.\text{pre}) \quad (2.8)$$

Daun yang memuat nilai x memiliki keempat medan sama dengan x : satu-satunya blok tak kosong dari satu elemen adalah elemen itu sendiri.

2.5.3 Mengapa rumus-rumusnya benar

Ambil **Persamaan** (2.6). Prefiks tak kosong dari $L \cup R$ berakhir di dalam L , sehingga merupakan prefiks tak kosong dari L dan nilainya paling besar $L.pre$ (dan $L.pre$ sendiri adalah prefiks semacam itu), atau berakhir di dalam R , sehingga berupa seluruh L disusul prefiks tak kosong dari R , bernilai paling besar $L.sum + R.pre$ (dan nilai itu tercapai). Maksimum dari kedua kasus karenanya tepat merupakan prefiks terbaik. **Persamaan** (2.7) adalah bayangan cerminnya. Untuk **Persamaan** (2.8), setiap blok tak kosong dari $L \cup R$ terletak di L , di R , atau melintas; blok yang melintas adalah sufiks tak kosong L yang direkatkan pada prefiks tak kosong R , dan pasangan terbaiknya adalah $L.suf + R.pre$ karena kedua bagian dapat dipilih secara bebas satu sama lain.

Masih diperlukan satu sifat lagi: kueri pada Subbab 2.4.3 menggabungkan jawaban parsial dengan pengelompokan tertentu, misalnya $(A \oplus B) \oplus C$ dengan $\oplus$ sebagai **combine**, sedangkan pembangunan bisa mengelompokkan potongan yang sama sebagai $A \oplus (B \oplus C)$. Keduanya harus memberi hasil yang sama. Memang demikian, karena keduanya adalah deskripsi empat-medan yang benar dari rentang gabungan, dan bukti di atas menunjukkan bahwa **combine** selalu menghasilkan deskripsi yang benar dari deskripsi yang benar. Dalam bahasa aljabar, **combine** bersifat *asosiatif*, tetapi tidak komutatif: L harus didahulukan.

2.5.4 Mengerjakan contoh dengan tangan

Mari jawab kueri $[2,5]$ pada larik **Gambar 2.1**, dengan setiap simpul ditulis sebagai (sum, pre, suf, ans) . Ketiga potongan kanoniknya adalah $[2] = (-5, -5, -5, -5)$, $[3,4] = (3, 4, 3, 4)$, dan $[5] = (3, 3, 3, 3)$. Pertama gabungkan $[2]$ dengan $[3,4]$:

- $sum = -5 + 3 = -2$;
- $pre = \max(-5, -5 + 4) = -1$;
- $suf = \max(3, 3 + (-5)) = 3$;
- $ans = \max(-5, 4, -5 + 4) = 4$.

Jadi $[2,4] = (-2, -1, 3, 4)$. Lalu gabungkan dengan $[5]$:

- $sum = -2 + 3 = 1$;
- $pre = \max(-1, -2 + 3) = 1$;
- $suf = \max(3, 3 + 3) = 6$;
- $ans = \max(4, 3, 3 + 3) = 6$.

Blok terbaik di dalam $[2,5]$ bernilai 6: blok 4, $-1, 3$, yang tepat merupakan kasus melintas, sufiks terbaik $[2,4]$ direkatkan pada prefiks terbaik $[5]$.

2.5.5 Resep yang sama, pertanyaan lain

Resep ini layak dilatih pada beberapa pertanyaan lain, karena babak nasional jarang menanyakan hal yang sama dua kali. Gagasan menyimpan informasi tambahan di simpul pohon agar pertanyaan baru dapat dijawab sudah lama dan umum; Cormen dkk. menyebutnya *augmenting a data structure* (memperkaya struktur data) dan mengembangkannya untuk pohon pencarian seimbang [3]. Berikut tiga latihan singkat dengan semangat yang sama, masing-masing dengan simpul yang dihasilkannya.

Minimum dan berapa kali ia muncul. Langkah 1 menyimpan mn . Langkah 2: minimum $L \cup R$ adalah $\min(L.mn, R.mn)$, tidak ada yang hilang. Untuk cacahnya, cacah baru adalah $L.cnt$ bila minimum L lebih kecil, $R.cnt$ bila minimum R lebih kecil, dan $L.cnt + R.cnt$ bila sama. Dua medan, tertutup.

Apakah rentangnya terurut? Langkah 1 menyimpan penanda ok . Langkah 2: $L \cup R$ terurut bila kedua belahan terurut dan nilai terakhir L paling besar sama dengan nilai pertama R . Nilai tepi hilang, sehingga langkah 3 menambahkan $first$ dan $last$, yang digabung dengan mudah ($L.first$ dan $R.last$). Tiga medan, tertutup.

Banyaknya pasangan bertetangga yang “naik”. Pasangan (a_i, a_{i+1}) dengan $a_i < a_{i+1}$ di dalam $L \cup R$ berada di dalam L , di dalam R , atau berupa satu-satunya pasangan yang melintasi batas. Maka simpul menyimpan cacahnya ditambah, lagi-lagi, $first$ dan $last$, dan penggabungan menambahkan satu bila $L.last < R.first$.

Perhatikan pola yang muncul di keempat rancangan, termasuk GSS1: jawaban untuk sebuah rentang membutuhkan informasi tentang *tepi* rentang, karena di situlah dua tetangga bersentuhan. Ketika bertemu pertanyaan rentang baru dalam lomba, hal pertama yang perlu ditanyakan adalah: “apa yang terjadi tepat di batas antara dua belahan?” Medan-medan yang menggambarkan tepi biasanya adalah yang hilang.

Satu peringatan terakhir tentang resep ini: ketertutupan bisa gagal. Beberapa pertanyaan, seperti “banyaknya nilai berbeda di $[l, r]$ ”, tidak dapat ditutup dengan simpul kecil, karena kedua belahan bisa berbagi nilai dengan cara yang hanya dapat diungkap oleh daftar nilai lengkap. Ketika resep terus menambah medan tanpa akhir, berhentilah dan carilah teknik lain; untuk nilai berbeda, pemrosesan *offline* dengan pohon Fenwick atau pohon persisten pada Bab 3 adalah jawaban yang lazim.

2.6 Studi Kasus SULIT: GSS1

2.6.1 Soal dengan kata-kata kita sendiri

Diberikan barisan N bilangan bulat $a_1, \dots, a_N$, dengan $N \leq 50\,000$ dan setiap nilai antara $-15\,007$ dan $15\,007$. Lalu menyusul M pertanyaan, masing-masing sepasang $x \leq y$. Untuk setiap pertanyaan, cetak jumlah terbesar $a_i + a_{i+1} + \dots + a_j$ atas semua i, j dengan $x \leq i \leq j \leq y$. Bloknya tidak boleh kosong, sehingga jawaban negatif dimungkinkan. Masukan memberikan N , lalu N nilai, lalu M , lalu M pasangan.

2.6.2 Contoh masukan dan keluaran

Contoh dari pernyataan soal berisi larik $-1, 2, 3$ dan satu pertanyaan, $[1, 2]$, yang jawabannya 2 (blok yang hanya berisi 2):

```
3
-1 2 3
1
1 2
```

Contoh kedua buatan kita memakai larik **Gambar 2.1**:

```
8
2 -5 4 -1 3 -6 5 1
4
1 8
2 5
5 7
6 6
```

Jawaban kedua adalah angka 6 yang kita hitung dengan tangan di Subbab 2.5.4; jawaban terakhir, -6, adalah rentang berisi satu elemen negatif.

2.6.3 Rancangan

Semuanya sudah siap. Batasan ($N, M \leq 50\,000$) meminta $O(\log N)$ per pertanyaan (Tabel 1.2 di Bab 1). Simpulnya adalah simpul empat-medan dari Subbab 2.5. Pohon dibangun sekali dalam $O(N)$; setiap pertanyaan memanggil `query` dan mencetak medan `ans` dari simpul yang dikembalikannya. Satu-satunya pilihan rancangan yang halus adalah bahwa `query` mengembalikan *simpul* utuh, bukan bilangan: jawaban parsial dari kedua anak digabung dengan `combine` yang sama yang membangun pohon, dan itulah tepatnya yang dibutuhkan kasus melintas.

Seberapa besar nilai-nilainya? Jumlah hingga 50 000 nilai dengan nilai mutlak paling besar 15 007 bernilai mutlak paling besar sekitar 7.5×10^8 , yang muat dalam `int` 32 bit, tetapi hanya dengan selisih faktor tiga. Solusinya memakai `long long` di simpul, yang di sini tidak memakan biaya apa pun dan menghapus pertanyaan itu sama sekali.

2.6.4 Solusi lengkap

Kode 2.3 `GSS1.cpp`: pohon segmen empat-medan; pembangunan dalam $O(N)$, setiap pertanyaan dalam $O(\log N)$.

```
// GSS1 (SPOJ): maximum-sum non-empty subarray inside [x, y], static array.
// Segment tree whose nodes carry (sum, pre, suf, ans). C++17.
#include <algorithm>
#include <cstdio>
using namespace std;

const int MX = 50001;
struct Node {
    long long sum; // total of the range
    long long pre; // best sum of a non-empty piece touching the left edge
    long long suf; // best sum of a non-empty piece touching the right edge
    long long ans; // best sum of a non-empty piece anywhere in the range
};
Node tree[4 * MX];
int a[MX];

Node makeNode(long long v) { // a range of one element: every field is v
    Node n;
    n.sum = n.pre = n.suf = n.ans = v;
    return n;
}

Node combine(const Node& l, const Node& r); // defined below

void build(int v, int lo, int hi) {
    if (lo == hi) {
        tree[v] = makeNode(a[lo]);
        return;
    }
    int mid = (lo + hi) / 2;
    build(2 * v, lo, mid);
    build(2 * v + 1, mid + 1, hi);
    tree[v] = combine(tree[2 * v], tree[2 * v + 1]);
}

// <P50>
// combine: glue two neighboring ranges (left l, right r) into one node.
// query: return the node of [ql, qr] inside [lo, hi]; ask only the children
// that overlap, and combine the two partial nodes when both do.
// >>> YOUR TURN (Public Edition): write this part yourself, following the hint above. <<<
// </P50>
```

Kode 2.3 (Lanjutan) *GSS1.cpp: pohon segmen empat-medan; pembangunan dalam $O(N)$, setiap pertanyaan dalam $O(\log N)$.*

```
int main() {
    int n, m;
    if (scanf("%d", &n) != 1) return 0;
    for (int i = 1; i <= n; i++)
        if (scanf("%d", &a[i]) != 1) return 0;
    build(1, 1, n);
    if (scanf("%d", &m) != 1) return 0;
    while (m--) {
        int x, y;
        if (scanf("%d %d", &x, &y) != 2) return 0;
        printf("%lld\n", query(1, 1, n, x, y).ans);
    }
    return 0;
}
```

2.6.5 Baris demi baris

Simpul. `struct Node` menyimpan keempat medan dari Persamaan (2.5) sampai Persamaan (2.8), dengan komentar untuk masing-masing. `tree` memiliki `4 * MX` entri, mengikuti Persamaan (2.3).

Daun. `makeNode(v)` mengembalikan simpul yang keempat medannya sama dengan `v`. Parameternya `long long` agar medan-medan dihitung dalam 64 bit sejak awal.

Pembangunan. `build` dideklarasikan sesudah deklarasi awal (*forward declaration*) `combine`, karena di dalam berkas `combine` didefinisikan lebih ke bawah. Daun menyalin `a[10]` melalui `makeNode`; simpul lain membangun kedua anaknya dan menyimpan gabungannya. Panggilan `build(1, 1, n)` di `main` mengisi seluruh pohon.

Combine. Di antara penanda `// <P50>` dan `// </P50>` terletak jantung bab ini. `combine` adalah salinan harfiah [Persamaan \(2.5\)](#), [Persamaan \(2.6\)](#), [Persamaan \(2.7\)](#), dan [Persamaan \(2.8\)](#). Parameternya `const Node&`, yang menghindari penyalinan 32 bita per panggilan; urutannya penting, `l` adalah tetangga kiri.

Kueri. `query` mengimplementasikan tiga kasus Subbab 2.4.3. Simpul yang terliput penuh mengembalikan nilai tersimpannya. Kueri yang seluruhnya di belahan kiri ($qr \leq mid$) atau seluruhnya di belahan kanan ($ql > mid$) turun ke satu anak saja. Kasus melintas bertanya kepada kedua anak dan menggabungkan dua simpul parsial, kiri sebelum kanan. Karena pemanggil hanya turun ke anak yang beririsan dengan kueri, `query` tidak pernah dipanggil pada irisan kosong, dan tidak diperlukan “simpul netral”. Hal ini penting: tidak ada simpul netral alami untuk maksimum tak kosong, dan mengarah satu (misalnya dengan nilai yang sangat negatif) adalah sumber *bug* luapan.

Main. `main` membaca masukan dengan `scanf`, memeriksa setiap nilai kembaliannya agar program berhenti dengan rapi pada masukan yang terpotong, membangun pohon, dan mencetak `query(1, 1, n, x, y).ans` untuk setiap pertanyaan.

2.6.6 Jebakan

- **Memulai dari nol.** Menulis `pre = max(0, ...)` atau menginisialisasi nilai terbaik dengan 0 menghitung blok terbaik yang *boleh kosong*. Pada rentang yang seluruhnya negatif, program mencetak 0 alih-alih elemen terbesar. Uji tercepatnya adalah larik `-1, 2` dengan kueri `[1,2]`: jawabannya harus 2, dan untuk larik `-3` saja harus `-3`.
- **Pohon yang terlalu kecil.** `tree[2 * N]` gagal untuk kebanyakan N ; pakailah $4N$ ([Persamaan \(2.3\)](#)).
- **Menggabungkan dengan urutan terbalik.** `combine(right, left)` menghasilkan *pre* dan *suf* yang salah sehingga jawaban melintas juga salah, tetapi sering tetap lolos contoh soal.
- **Mengembalikan bilangan dari query.** Kueri yang hanya mengembalikan `ans` tidak dapat menangani kasus melintas; ia harus mengembalikan simpul utuh.
- **Memperlakukan `x` dan `y` sebagai berbasis 0.** Soalnya berbasis 1, begitu pula pohonnya.

2.6.7 Stress test

Brute force pada **Kode 2.4** mencoba setiap awal i dan memperpanjang jumlahnya ke kanan sambil menyimpan yang terbaik; kompleksitasnya $O(N^2)$ per pertanyaan dan jelas benar.

Kode 2.4 `brute.cpp`: setiap blok dari setiap rentang, $O(N^2)$ per pertanyaan.

```
// Brute force for GSS1: try every start i and extend the sum to the right.
#include <algorithm>
#include <climits>
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, m;
    cin >> n;
    vector<long long> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];
    cin >> m;
    while (m--) {
        int x, y;
        cin >> x >> y;
        long long best = LLONG_MIN;
        for (int i = x; i <= y; i++) {
            long long s = 0;
            for (int j = i; j <= y; j++) best = max(best, s += a[j]);
        }
        cout << best << "\n";
    }
    return 0;
}
```

Pembangkit **Kode 2.5** membuat larik hingga 12 elemen yang nilainya diambil dari tiga skala: mungil (± 3 , banyak nilai kembar dan nol), kecil (± 10), dan penuh $\pm 15\,007$.

Kode 2.5 `gen.py`: masukan GSS1 acak pada tiga skala nilai.

```
#!/usr/bin/env python3
"""Random GSS1 input: python3 gen.py SEED"""
import random
import sys

rng = random.Random(int(sys.argv[1]) if len(sys.argv) > 1 else 1)
n = rng.randint(1, 12)
lim = rng.choice([3, 10, 15007])
print(n)
print(*[rng.randint(-lim, lim) for _ in range(n)])
m = rng.randint(1, 10)
print(m)
for _ in range(m):
    x = rng.randint(1, n)
    print(x, rng.randint(x, n))
```

Skrip `stress.sh` dari Bab 1 (Kode 1.2; berkas yang sama ada di setiap folder bab) menjalankan kedua program:

```
all 500 tests identical
```

Pada masukan terbesar yang diizinkan soal, $N = M = 50\,000$ dengan nilai dan rentang acak, solusi ini berjalan sekitar 0,03 detik di mesin kami, sesuai dengan [Persamaan \(2.4\)](#): kira-kira $50\,000 \times 70$ kunjungan simpul.

2.7 Ketika Alat Lain Lebih Baik

Pohon segmen bersifat umum, tetapi tidak selalu pilihan terbaik. Bila lariknya statis dan pertanyaannya jumlah atau cacah, jumlah prefiks ([Persamaan \(2.1\)](#)) menjawab dalam $O(1)$ dengan beberapa baris kode. Bila lariknya statis dan pertanyaannya minimum atau maksimum, *sparse table* menjawab dalam $O(1)$ sesudah praproses $O(N \log N)$, karena tumpang-tindih dua

potongan tidak mengubah minimum. Bila pertanyaannya “jumlah dengan pembaruan titik”, pohon Fenwick lebih pendek dan lebih cepat. Pohon segmen unggul ketika operasinya tidak lazim (seperti empat medan GSS1), ketika ada pembaruan (Bab 5), atau keduanya. Memilih alat terkecil yang berhasil adalah bagian dari keterampilan; pembahasan alternatif-alternatif ini dalam Laaksonen [4] dan Halim dkk. [5] layak dibaca berdampingan.

2.8 Taktik Hari Lomba

- **Simpan kerangka yang sudah teruji.** Fungsi `build` dan `query` pada **Kode 2.3** sama untuk hampir setiap soal pohon segmen statis. Latih menulisnya dari ingatan sampai cukup lima menit, dan dalam lomba ubah hanya `Node`, `makeNode`, dan `combine`.
- **Rancang di kertas dulu.** Jalankan resep ketertutupan Subbab 2.5.1 di kertas, dan periksa `combine` pada dua rentang mungil dengan tangan sebelum mengetik.
- **Amankan subsoal $O(N^2)$.** Subsoal dengan N kecil diselesaikan oleh **Kode 2.4**. Kumpulkan lebih dulu; program itu kemudian menjadi acuan *stress test* Anda.
- **Uji kasus serba negatif.** Inilah kegagalan senyap klasik soal jumlah maksimum.

2.9 Ringkasan

- Pohon segmen menyimpan jawaban di setiap simpul pohon yang membelah dua; pembangunan memakan $O(N)$ dan setiap pertanyaan rentang memakan $O(\log N)$ menurut **Persamaan (2.4)**, dengan larik $4N$ simpul menurut **Persamaan (2.3)**.
- Semua logika khas soal tinggal di `makeNode` dan `combine`; `combine` harus asosiatif, dan sering kali tidak komutatif.
- Resep ketertutupan merancang simpul: tambahkan informasi apa pun yang hilang ketika dua tetangga direkatkan, sampai tidak ada yang hilang.
- Blok berjumlah terbesar memerlukan empat medan, *sum*, *pre*, *suf*, dan *ans*, yang digabung dengan **Persamaan (2.5)** sampai **Persamaan (2.8)**.
- GSS1 adalah simpul itu di dalam pohon segmen statis yang `query`-nya mengembalikan simpul utuh, sehingga jawaban parsial tergabung dengan benar.
- Jumlah prefiks, *sparse table*, dan pohon Fenwick mengalahkan pohon segmen bila syarat-syarat sempitnya terpenuhi.

2.10 Soal Latihan

1. Bangun simpul $[5,8]$ pada **Gambar 2.1** dari anak-anaknya $[5,6] = (-3,3,-3,3)$ dan $[7,8] = (6,6,6,6)$, lalu cocokkan dengan gambar.
2. Untuk $N = 5$, daftarkan rentang semua simpul pohon dan indeks simpul terbesar yang dipakai. Apakah di bawah $4N$?
3. Simpul mana yang dikembalikan `query(1, 1, 8, 3, 8)` tanpa turun lebih jauh, pada pohon **Gambar 2.1**?
4. Rancang simpul untuk “panjang deret terpanjang nilai bertetangga yang sama di dalam $[l,r]$ ”. Medan apa saja yang diperlukan?
5. Seorang teman menulis `res.ans = max(l.ans, r.ans)` dan lupa suku melintas. Berikan masukan terkecil yang membuat programnya salah.
6. Jelaskan mengapa `combine(query(left), query(right))` akan salah bila `query` boleh dipanggil pada simpul yang tidak beririsan dengan kueri.

2.11 Penyelesaian Soal Latihan

1. $sum = -3 + 6 = 3$; $pre = \max(3, -3 + 6) = 3$; $suf = \max(6, 6 + (-3)) = 6$; $ans = \max(3, 6, -3 + 6) = 6$. Gambar menampilkan [5,8] sebagai 3, 3, 6, 6.
2. Simpul 1 [1,5]; 2 [1,3]; 3 [4,5]; 4 [1,2]; 5 [3]; 6 [4]; 7 [5]; 8 [1]; 9 [2]. Indeks terbesarnya 9, di bawah $4N = 20$. Di sini $2N = 10$ pun sudah cukup, tetapi untuk $N = 6$ indeks terbesarnya 13, di atas $2N = 12$, dan itulah sebabnya $2N$ bukan ukuran yang aman.
3. Rentang [3,8] terbelah menjadi [3,4] (simpul 5) dan [5,8] (simpul 3), keduanya terliput penuh, sehingga **query** langsung mengembalikan keduanya dan menggabungkannya sekali.
4. Simpan panjang rentang, deret terpanjang di mana saja (*best*), deret yang menyentuh tepi kiri (*pre*), deret yang menyentuh tepi kanan (*suf*), serta nilai di kedua tepi. Saat menggabung, deret melintas hanya ada bila nilai tepi kanan L sama dengan nilai tepi kiri R ; panjangnya $L.suf + R.pre$, dan *pre* hanya berlanjut ke R bila $L.pre$ sama dengan panjang L .
5. Larik 1, 1 dengan kueri [1,2]: setiap belahan memiliki $ans = 1$, sehingga program yang salah mencetak 1, padahal jawaban sebenarnya 2.
6. Simpul yang tidak beririsan harus mengembalikan jawaban “kosong”, padahal simpul menggambarkan blok tak kosong, dan tidak ada nilai keempat medan yang berperilaku sebagai unsur netral; menggabung dengan simpul karangan akan merusak *pre*, *suf*, dan *ans*. Kueri tiga kasus menghindari pertanyaan kepada simpul semacam itu.

Daftar Pustaka

- [1] Sphere Online Judge, “GSS1 — Can you answer these queries I.” [Daring]. Tersedia: <https://www.spoj.com/problems/GSS1/> (diakses Sep. 2026).
- [2] J. Bentley, “Programming pearls: Algorithm design techniques,” *Communications of the ACM*, vol. 27, no. 9, pp. 865–873, 1984.
- [3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, dan C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
- [4] A. Laaksonen, *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, 3rd ed. Cham, Switzerland: Springer, 2024.
- [5] S. Halim, F. Halim, dan S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.

Bab 3

Pohon Segmen Persisten dan Statistik Urutan — MKTHNUM

Bagaimana menjawab “berapa nilai terkecil ke- k di antara posisi l sampai r ?” untuk ribuan rentang tanpa mengurutkan setiap rentang dari awal? Bab ini menjawabnya dengan salah satu gagasan paling anggun dalam pemrograman kompetitif: pohon segmen yang menyimpan *setiap* versi lamanya, dengan biaya hanya sekitar $\log_2 N$ simpul baru per versi. Anda akan mempelajari kompresi koordinat, pohon frekuensi atas nilai, penurunan yang menemukan nilai terkecil ke- k dalam waktu logaritmik, persistensi dengan *path copying* (penyalinan lintasan), dan trik yang membuat seluruh bab ini berjalan: dua versi pohon dapat dikurangkan, persis seperti dua jumlah prefiks. Studi kasusnya, SPOJ MKTHNUM, merangkum semua itu dalam kurang dari enam puluh baris.

3.1 Median Petugas Pasar

Setiap pagi di sepanjang jalan menuju sebuah kota di perbukitan, para penjual durian membuka lapak berjajar, satu deret panjang yang dinomori dari kaki bukit sampai ke puncak. Seorang petugas pasar mencatat harga sekilo durian di setiap lapak. Pembeli, penjual, dan koperasi setempat terus mengajukan pertanyaan yang sebetulnya: “di antara lapak 20 sampai 80, berapa harga tengahnya?” atau “di antara tiga puluh lapak pertama, berapa harga termurah ketiga?” Harga tengah, *median*, adalah ringkasan yang lebih adil daripada rata-rata, karena satu lapak dengan harga sangat tinggi untuk varietas terkenal akan menarik rata-rata ke atas tetapi hampir tidak menggeser median.

Untuk satu pertanyaan, ia dapat menyalin harga lapak-lapak dalam rentang itu, mengurutkannya, dan membaca posisi ke- k . Biayanya sekitar $n \log n$ langkah untuk rentang berisi n lapak. Untuk deret seratus ribu lapak dan lima ribu pertanyaan, mengurutkan setiap rentang memakan hingga $5\,000 \times 100\,000 \times 17 \approx 8.5 \times 10^9$ langkah, jauh terlalu lambat. Soal SPOJ MKTHNUM [1] menanyakan persis pertanyaan ini dengan ukuran persis ini.

3.2 Gagasannya dengan Kata-Kata Sederhana

Andaikan sejenak pertanyaannya selalu tentang sebuah *prefiks*, lapak 1 sampai r . Maka kita dapat menyimpan *cacah harga*: tabel yang menyebutkan, untuk setiap harga yang mungkin, berapa banyak dari r lapak pertama yang memilikinya. Untuk menemukan harga terkecil ke- k , telusuri harga dari rendah ke tinggi sambil menjumlahkan cacahnya, sampai jumlah berjalan mencapai k . Pohon segmen atas *harga* (bukan atas posisi) membuat penelusuran itu hanya memakan $\log_2$ langkah alih-alih satu langkah per harga.

Sekarang tambahkan lapak satu demi satu. Tabel cacah sesudah lapak r sama dengan tabel sesudah lapak $r - 1$ ditambah satu pada harga lapak r . Bila semua tabel itu dapat disimpan, versi 0 (kosong) sampai versi N (semuanya), maka pertanyaan tentang lapak l sampai r dijawab dengan *mengurangkan* versi $l - 1$ dari versi r : sisanya mencacah tepat lapak l sampai r .

Menyimpan $N + 1$ pohon utuh akan menghabiskan memori terlalu banyak, tetapi versi-versi berurutan hanya berbeda di sepanjang satu lintasan dari akar ke daun. *Path copying* menyimpan hanya lintasan itu untuk setiap versi baru dan berbagi sisanya. Itulah seluruh isi bab ini.

3.3 Soal Pemanasan

3.3.1 Kompresi koordinat

Harga dalam MKTHNUM berkisar dari -10^9 sampai 10^9 , jadi pohon dengan satu daun untuk setiap nilai yang mungkin tidak dapat dipertimbangkan. Namun hanya N nilai berbeda yang benar-benar muncul, dan untuk pertanyaan terkecil ke- k yang penting hanya *urutannya*. *Kompresi koordinat* mengganti setiap nilai dengan peringkatnya di antara nilai-nilai yang berbeda: urutkan salinannya, buang duplikat, lalu cari setiap nilai dengan pencarian biner. **Kode 3.1** memperlihatkan tiga panggilan pustaka baku itu.

Kode 3.1 `compress.cpp`: urutkan, buang duplikat, dan ganti setiap nilai dengan peringkatnya.

```
// Warm-up: coordinate compression with sort + unique + lower_bound.
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n;
    if (scanf("%d", &n) != 1) return 0;
    vector<int> a(n);
    for (int& x : a)
        if (scanf("%d", &x) != 1) return 0;
    vector<int> vals = a;
    sort(vals.begin(), vals.end());
    vals.erase(unique(vals.begin(), vals.end()), vals.end());
    for (int x : a) { // rank of x among the distinct values, from 1
        int rank = (int)(lower_bound(vals.begin(), vals.end(), x) - vals.begin()) + 1;
        printf("%d ", rank);
    }
    printf("\n");
    for (int i = 0; i < (int)vals.size(); i++) printf("%d->%d ", i + 1, vals[i]);
    printf("\n");
    return 0;
}
```

Pada nilai $10^9, -7, 42, -7, 5, 42$:

Baris pertama memberi peringkat setiap nilai masukan; baris kedua adalah tabel untuk menerjemahkan peringkat kembali menjadi nilai. Sesudah kompresi, nilainya adalah bilangan bulat $1, \dots, U$, dengan $U \leq N$ banyaknya nilai berbeda, dan pohon atas $[1, U]$ memiliki paling banyak $4N$ simpul.

3.3.2 Mengurangkan dua cacah prefiks

Pemanasan kedua memperlihatkan gagasan “selisih versi” tanpa pohon sama sekali. Bila nilainya mungil (1 sampai 9), kita mampu menyimpan, untuk setiap prefiks i , satu tabel cacah utuh $c_i[v]$, yaitu banyaknya posisi di antara i posisi pertama yang bernilai v . Maka banyaknya posisi di $[l, r]$ yang bernilai v adalah

$$c_r[v] - c_{l-1}[v] \quad (3.1)$$

dan banyaknya nilai yang paling besar x di $[l, r]$ adalah jumlah [Persamaan \(3.1\)](#) atas $v \leq x$. **Kode 3.2** mengerjakannya dengan tabel biasa.

Kode 3.2 `count_prefix.cpp`: mencacah nilai yang paling besar x di sebuah rentang dengan mengurangkan dua tabel cacah prefiks.

```
// Warm-up: "how many of a[l..r] are <= x?" by subtracting two prefix counts.
// Values are small (1..9), so cntUpTo[i][v] can be a plain table.
#include <cstdio>
using namespace std;

int cntUpTo[1001][10]; // cntUpTo[i][v] = how many of a[1..i] equal v

int main() {
    int n, m;
    if (scanf("%d %d", &n, &m) != 2) return 0;
    for (int i = 1; i <= n; i++) {
        int x;
        if (scanf("%d", &x) != 1) return 0;
        for (int v = 1; v <= 9; v++) cntUpTo[i][v] = cntUpTo[i - 1][v];
        cntUpTo[i][x]++; // version i = version i-1 plus one value
    }
    while (m--) {
        int l, r, x, total = 0;
        if (scanf("%d %d %d", &l, &r, &x) != 3) return 0;
        for (int v = 1; v <= x; v++) total += cntUpTo[r][v] - cntUpTo[l - 1][v];
        printf("%d\n", total);
    }
    return 0;
}
```

Pada larik 3, 1, 4, 1, 5, 9, 2 dan pertanyaan $(l, r, x) = (2, 5, 3)$, $(1, 7, 4)$, dan $(3, 6, 1)$:

Di $[2, 5]$ nilainya 1, 4, 1, 5, dan dua di antaranya paling besar 3. Perhatikan apa yang dilakukan program: versi i tabel adalah versi $i - 1$ ditambah satu nilai, dan pertanyaan rentang mengurangkan dua versi. Pohon segmen persisten mengerjakan hal yang persis sama, dengan dua perbedaan: pohon alih-alih tabel datar, agar bekerja untuk 10^5 nilai berbeda, dan simpul yang dipakai bersama alih-alih salinan, agar $N + 1$ versi muat dalam memori.

3.4 Pohon Segmen atas Nilai

Bangun pohon segmen atas rentang nilai $[1, U]$. Simpul v yang meliputi nilai $[lo, hi]$ menyimpan $cnt(v)$, banyaknya elemen tersisip yang nilai terkompresinya berada di $[lo, hi]$. Cacah akar adalah banyaknya elemen yang sudah disisipkan; cacah daun adalah berapa kali nilainya disisipkan. Menyisipkan nilai p menambah satu pada setiap simpul di lintasan dari akar ke daun p , dan tidak ada yang lain yang berubah.

Untuk menemukan nilai tersisip terkecil ke- k , mulailah dari akar dengan k . Pada simpul yang meliputi $[lo, hi]$ dengan titik tengah mid , misalkan $c = cnt(\text{anak kiri})$ adalah banyaknya nilai tersisip di $[lo, mid]$. Bila $k \leq c$, nilai terkecil ke- k ada di antara mereka: turun ke kiri dengan k yang sama. Bila tidak, c nilai pertama semuanya lebih kecil daripada jawaban: turun ke kanan dan cari nilai terkecil ke- $(k - c)$ di sana:

$$kth(v, k) = \begin{cases} kth(left(v), k) & \text{jika } k \leq cnt(left(v)) \\ kth(right(v), k - cnt(left(v))) & \text{selainnya} \end{cases} \quad (3.2)$$

dan di daun jawabannya adalah nilai daun itu. **Persamaan (3.2)** adalah pencarian biner, tetapi atas cacah alih-alih atas larik terurut: setiap langkah membelah dua rentang nilai calon, sehingga penurunan memakan $\lceil \log_2 U \rceil$ langkah. Kebenarannya berupa invarian: *jawaban terletak di rentang nilai simpul saat ini, dan jawaban itu adalah nilai terkecil ke- k di antara nilai tersisip dalam rentang tersebut*. Invarian itu berlaku di akar, dan setiap kasus **Persamaan (3.2)** mempertahankannya.

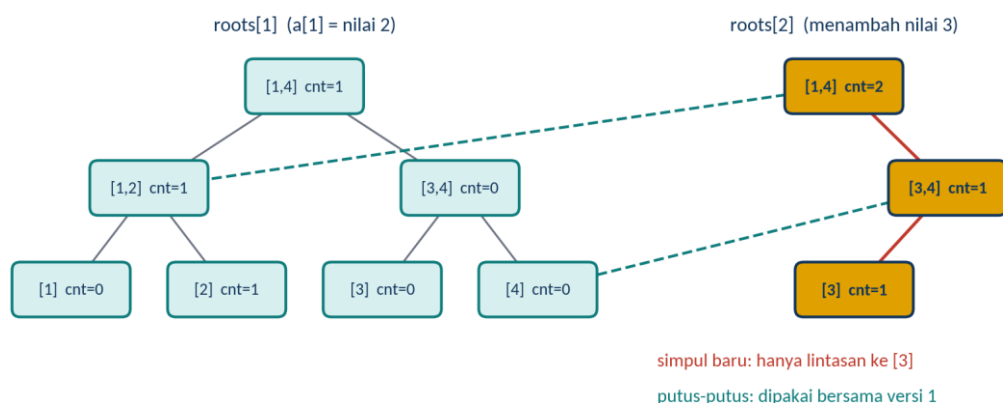
Contoh kecil membuat penurunan ini konkret. Sisipkan peringkat 3, 1, 4, 1 ke pohon atas [1,4]. Akar mencacah 4 nilai; anak kirinya [1,2] memuat dua nilai 1, 1, sehingga cacahnya 2, dan anak kanan [3,4] mencacah 3 dan 4, juga 2. Untuk menemukan nilai terkecil ketiga, mulai di akar dengan $k = 3$. Anak kiri hanya memuat 2 nilai, kurang dari 3, jadi jawabannya ada di anak kanan, tempat kita kini mencari nilai terkecil ke- $(3 - 2) = 1$. Anak kiri dari [3,4] adalah daun [3] dengan cacah 1, dan $1 \leq 1$, sehingga kita turun ke kiri dan tiba di daun 3. Bila diurutkan, nilainya 1, 1, 3, 4, dan yang ketiga memang 3. Setiap langkah membuang separuh rentang nilai yang tersisa, dan tidak satu langkah pun melihat elemen satu per satu.

3.5 Persistensi dengan Penyalinan Lintasan

3.5.1 Salin lintasannya, bagi sisanya

Pembaruan biasa menimpa simpul, sehingga versi lama hilang. Pembaruan *persisten* tidak pernah menimpa. Ia membuat salinan baru setiap simpul di lintasan dari akar ke daun yang berubah, dengan cacah bertambah satu, dan membuat setiap salinan menunjuk ke salinan baru di bawahnya pada lintasan itu serta ke anak *lama* di sisi lain. Akar baru menggambarkan versi baru; akar lama tetap menggambarkan versi lama, tanpa tersentuh. Driscoll, Sarnak, Sleator, dan Tarjan mengkaji cara ini dan cara-cara lain untuk membuat struktur data persisten dalam sebuah makalah klasik [2].

Gambar 3.1 memperlihatkan gagasan itu pada empat nilai terkompresi. Versi 1 memuat nilai 2. Menyisipkan nilai 3 hanya membuat tiga simpul baru, yaitu akar [1,4], simpul [3,4], dan daun [3], dengan cacah 2, 1, dan 1. Akar baru menunjuk ke kiri pada simpul *lama* [1,2], dan [3,4] yang baru menunjuk ke kanan pada daun *lama* [4]: keduanya dipakai bersama, digambar putus-putus.



Gambar 3.1 Penyalinan lintasan: menyisipkan nilai 3 ke versi 1 hanya membuat tiga simpul kuning di lintasan menuju daun [3]; sisi putus-putus menunjuk ke simpul yang dipakai bersama dengan versi 1.

3.5.2 Berapa banyak memori

Setiap penyisipan membuat satu simpul per tingkat, jadi $\lceil \log_2 U \rceil + 1$ simpul. Dengan satu simpul kosong untuk versi 0, totalnya sesudah N penyisipan adalah

$$\text{simpul} = 1 + N(\lceil \log_2 U \rceil + 1) \quad (3.3)$$

Untuk $N = U = 100\,000$ itu berarti $1 + 100\,000 \times 18 = 1\,800\,001$ simpul. Dengan tiga larik `int` (anak kiri, anak kanan, cacah) masing-masing 4 bita, pohon menempati sekitar 21,6 MB, dan itulah sebabnya solusi mendeklarasikan lariknya berukuran `MAXN * 18`. Persamaan (3.3) adalah angka yang harus dihitung *sebelum* menulis kode: bila melampaui batas memori, persistensi adalah alat yang salah.

Satu rincian yang berguna: pohon kosong tidak memerlukan U simpul. Satu simpul 0 dengan cacah 0 yang anak-anaknya adalah simpul 0 itu sendiri mewakili “subpohon kosong berukuran berapa pun”. Versi baru yang turun ke kiri dari simpul kosong menyalin simpul 0 dan melanjutkan lagi dari simpul 0, sehingga bagian kosong setiap versi tidak memakan apa-apa.

3.5.3 Mengapa berbagi itu aman

Berbagi simpul antarversi terdengar berbahaya: bila dua versi menunjuk simpul yang sama, apakah perubahan untuk satu versi tidak ikut muncul di versi lain? Jawabannya: *tidak ada simpul yang pernah diubah sesudah dibuat*. `insert` hanya membaca simpul lama dan menulis ke simpul yang benar-benar baru. Struktur yang simpulnya tidak pernah berubah sesudah dibuat disebut *immutable* (tak berubah), dan bagian yang tak berubah dapat dipakai bersama dengan bebas, persis seperti dua kalimat dapat mengutip kutipan yang sama tanpa satu pun mampu mengubahnya.

Dengan itu, kebenaran versi i mengikuti dengan induksi. Versi 0 adalah pohon kosong, dan semua cacahnya nol, yang benar untuk prefiks kosong. Andaikan versi $i - 1$ adalah pohon cacah yang benar untuk $a_1, \dots, a_{i-1}$. Penyisipan peringkat p dari a_i membuat salinan baru tepat dari simpul-simpul yang rentang nilainya memuat p , masing-masing dengan cacah bertambah satu, dan itulah persis perubahan yang ditimbulkan a_i pada cacah. Setiap posisi simpul lain mempertahankan simpul versi $i - 1$, yang cacahnya tetap benar karena a_i tidak jatuh di rentangnya. Jadi versi i adalah pohon cacah yang benar untuk $a_1, \dots, a_i$, dan induksinya lengkap.

Argumen yang sama menjelaskan mengapa versi-versi hanya boleh dibangun berurutan dan tidak pernah diubah sesudahnya. MKTHNUM tidak memiliki pembaruan, jadi di sini hal itu bukan pembatasan. Ketika sebuah soal mencampur pertanyaan rentang dengan pembaruan di tengah larik, pohon persisten biasa tidak lagi cukup, dan alat yang lebih berat (pohon Fenwick yang selnya berupa pohon, atau pemrosesan *offline*) mengambil alih.

3.6 Mengurangkan Dua Versi

Bangun versi i dengan menyisipkan a_i ke versi $i - 1$, untuk $i = 1, \dots, N$, dan simpan akar-akarnya $root_0, \dots, root_N$. Karena setiap penyisipan hanya *menambah* satu pada cacah, cacah posisi simpul mana pun di versi r dikurangi cacahnya di versi $l - 1$ adalah banyaknya elemen di antara $a_l, \dots, a_r$ yang nilainya jatuh di rentang simpul itu:

$$cnt_{[l,r]}(v) = cnt(v \text{ pada versi } r) - cnt(v \text{ pada versi } l - 1) \quad (3.4)$$

Persamaan (3.4) adalah **Persamaan** (3.1) yang diangkat ke dalam pohon. Kedua versi memiliki *bentuk* yang sama (rentang nilai yang sama di posisi yang sama), meskipun berbagi simpul yang berbeda, sehingga kita dapat menuruni keduanya sekaligus: pada setiap langkah, hitung cacah kiri rentang itu dengan **Persamaan** (3.4), lalu terapkan **Persamaan** (3.2) padanya, dengan memindahkan kedua penunjuk ke anak kiri masing-masing atau keduanya ke anak kanan.

Mari telusuri pertanyaan “terkecil ketiga di posisi 2 sampai 5” pada larik 1, 5, 2, 6, 3, 7, 4, yang nilainya sudah berupa peringkat 1 sampai 7. Rentang itu memuat 5, 2, 6, 3.

- Akar, nilai [1,7], titik tengah 4. Anak kiri mencacah nilai 1 sampai 4: 3 buah di versi 5 (1, 2, 3), 1 buah di versi 1 (1). Selisihnya 2 (nilai 2 dan 3). Karena $k = 3 > 2$, turun ke kanan dengan $k = 1$.
- Simpul [5,7], titik tengah 6. Anak kiri [5,6]: versi 5 memuat 2 (5, 6), versi 1 memuat 0. Karena $k = 1 \leq 2$, turun ke kiri.
- Simpul [5,6], titik tengah 5. Anak kiri [5]: selisih 1. Karena $k = 1 \leq 1$, turun ke kiri.
- Daun [5]: jawabannya nilai 5.

Bila diurutkan, rentang itu menjadi 2, 3, 5, 6, dan elemen ketiganya memang 5.

WAWASAN KUNCI: Persistensi mengubah soal tentang *rentang posisi* menjadi soal tentang *dua prefiks*, dan sifat aditif memungkinkan keduanya dikurangkan. Pola yang sama menjawab “berapa banyak nilai di $[l, r]$ yang paling besar x ”, “peringkat x di $[l, r]$ ”, dan banyak lagi.

3.7 Studi Kasus SULIT: MKTHNUM

3.7.1 Soal dengan kata-kata kita sendiri

Diberikan larik $a_1, \dots, a_N$ berisi bilangan bulat berbeda dengan nilai mutlak hingga 10^9 , dengan $N \leq 100\,000$, dan $M \leq 5\,000$ pertanyaan (i, j, k) . Untuk setiap pertanyaan, bayangkan bagian $a_i, \dots, a_j$ diurutkan naik, lalu cetak elemen ke- k -nya, dengan $1 \leq k \leq j - i + 1$. Masukan memberikan N dan M , lalu lariknya, lalu satu pertanyaan per baris.

Pernyataan soal menjanjikan nilai yang berbeda, tetapi solusi kita tidak bergantung padanya: duplikat hanya membuat cacah sebuah daun lebih dari satu, dan **Persamaan** (3.2) tetap bekerja. Ketangguhan semacam ini tidak memakan biaya dan melindungi Anda ketika sebuah pernyataan soal kurang tepat daripada yang Anda kira.

3.7.2 Contoh masukan dan keluaran

```
7 3
1 5 2 6 3 7 4
2 5 3
4 4 1
1 7 3
```

Pertanyaan pertama adalah yang ditelusuri di Subbab 3.6. Yang kedua menanyakan satu-satunya elemen [4,4], yaitu 6. Yang ketiga menanyakan nilai terkecil ketiga dari seluruh larik: 1, 2, 3, jadi 3.

3.7.3 Rancangan

Rencananya mengikuti bab ini. Kompresi nilainya (Subbab 3.3.1). Buat versi 0 sebagai satu simpul kosong. Untuk $i = 1$ sampai N , sisipkan peringkat a_i ke versi $i - 1$ dengan penyalinan lintasan untuk memperoleh versi i . Untuk setiap pertanyaan, turuni versi $i - 1$ dan j bersama-sama

dengan [Persamaan \(3.2\)](#) dan [Persamaan \(3.4\)](#), lalu terjemahkan peringkat daunnya kembali menjadi nilai. Pembangunan memakan waktu $O(N \log N)$ dan memori sebesar [Persamaan \(3.3\)](#); setiap pertanyaan memakan $O(\log N)$, sehingga totalnya $O((N + M) \log N)$, beberapa juta langkah.

3.7.4 Solusi lengkap

Kode 3.3 adalah program lengkapnya. Program itu memiliki tiga fungsi pendek, dan hanya satu di antaranya, `kth`, yang tahu sesuatu tentang pertanyaannya.

Kode 3.3 `MKTHNUM.cpp`: pohon segmen persisten atas nilai terkompresi; setiap pertanyaan menuruni dua versi.

```
// MKTHNUM (SPOJ): k-th smallest value of a[i..j], many queries.
// Persistent segment tree over compressed values: version r holds a[1..r].
#include <algorithm>
#include <cstdio>
using namespace std;

const int MAXN = 100001;
const int MAXNODES = MAXN * 18 + 5; // one empty root + about log2(n)+1 per insert
int lc[MAXNODES], rc[MAXNODES], cnt[MAXNODES];
int nodeCount = 0; // node 0 is the shared "empty" node: lc = rc = cnt = 0
int roots[MAXN], a[MAXN], vals[MAXN];

int newNode() {
    ++nodeCount;
    lc[nodeCount] = rc[nodeCount] = cnt[nodeCount] = 0;
    return nodeCount;
}

// Returns the root of a new version: old version prev plus one value at pos.
// Only the nodes on the path to pos are copied; all others are shared.
int insert(int prev, int lo, int hi, int pos) {
    int cur = newNode();
    lc[cur] = lc[prev];
    rc[cur] = rc[prev];
    cnt[cur] = cnt[prev] + 1;
    if (lo == hi) return cur;
    int mid = (lo + hi) / 2;
    if (pos <= mid) lc[cur] = insert(lc[prev], lo, mid, pos);
    else rc[cur] = insert(rc[prev], mid + 1, hi, pos);
    return cur;
}

// <P50>
// kth: walk down version vr and version vl together; the counts of their left
// children differ by how many of a[l..r] are in the lower half of the values.
// Go left if k fits there, otherwise go right with k reduced; return the leaf.
// >>> YOUR TURN (Public Edition): write this part yourself, following the hint above. <<<
// </P50>
```

Kode 3.3 (Lanjutan) MKTHNUM.cpp: pohon segmen persisten atas nilai terkompresi; setiap pertanyaan menuruni dua versi.

```
int main() {
    int n, m;
    if (scanf("%d %d", &n, &m) != 2) return 0;
    for (int i = 1; i <= n; i++) {
        if (scanf("%d", &a[i]) != 1) return 0;
        vals[i] = a[i];
    }
    sort(vals + 1, vals + n + 1); // coordinate compression: 1..un
    int un = (int)(unique(vals + 1, vals + n + 1) - (vals + 1));
    roots[0] = newNode(); // version 0: no values yet
    for (int i = 1; i <= n; i++) {
        int pos = (int)(lower_bound(vals + 1, vals + un + 1, a[i]) - vals);
        roots[i] = insert(roots[i - 1], 1, un, pos);
    }
    while (m--) {
        int l, r, k;
        if (scanf("%d %d %d", &l, &r, &k) != 3) return 0;
        printf("%d\n", vals[kth(roots[l - 1], roots[r], 1, un, k)]);
    }
    return 0;
}
```

3.7.5 Baris demi baris

Penyimpanan. Alih-alih `struct` dengan penunjuk, pohon tinggal di tiga larik sejajar, `lc`, `rc`, dan `cnt`, yang diindeks dengan nomor simpul. Cara ini ringkas (12 bita per simpul) dan cepat. `MAXNODES` mengikuti [Persamaan \(3.3\)](#), dengan lima entri cadangan. Simpul 0 tidak pernah dibuat oleh `newNode`; sebagai variabel global isinya nol semua, sehingga ia menjadi simpul kosong bersama yang anak-anaknya dirinya sendiri.

newNode. Fungsi ini memberikan indeks bebas berikutnya dan mengosongkan medan-medannya. Karena lariknya global dan program menangani satu kasus uji, pengosongan itu sebenarnya tidak wajib, tetapi menjaga fungsi tetap benar bila Anda memakainya ulang untuk beberapa kasus uji.

insert. `insert(prev, lo, hi, pos)` mengembalikan akar versi baru yang sama dengan `prev` ditambah satu nilai di `pos`. Ia menyalin `prev` ke simpul baru `cur` dengan cacah bertambah, dan bila `cur` bukan daun, ia mengganti tepat satu anak: anak di sisi `pos`, dengan panggilan rekursif pada anak `prev` yang bersesuaian. Penunjuk anak yang lain tetap menunjuk ke versi lama: itulah pemakaian bersama pada [Gambar 3.1](#).

kth. Di antara penanda `// <P50>` dan `// </P50>`, `kth(vl, vr, lo, hi, k)` menuruni dua versi sekaligus, `vl` untuk versi $l - 1$ dan `vr` untuk versi r . `leftCount` adalah [Persamaan \(3.4\)](#) yang diterapkan pada anak-anak kiri, dan kedua cabangnya adalah [Persamaan \(3.2\)](#). Rekursi berhenti di daun dan mengembalikan peringkatnya.

main. Fungsi ini membaca larik sambil menyimpan salinannya di `vals`, lalu mengompresi: `sort` dan `unique` menyisakan U nilai berbeda di `vals[1..un]`, dan `lower_bound` mengembalikan posisi setiap a_i di antaranya, yaitu peringkatnya. Versi 0 adalah satu simpul kosong baru; lalu setiap versi dibangun dari versi sebelumnya. Setiap pertanyaan mencetak `vals[kth(roots[l - 1], roots[r], 1, un, k)]`, mengubah peringkat kembali menjadi nilai aslinya.

3.7.6 Jebakan

- **Lupa kompresi.** Pohon atas $[-10^9, 10^9]$ memerlukan kedalaman 31 dan jauh lebih banyak simpul; kompresi lebih dulu adalah keharusan.
- **Memakai versi l alih-alih $l - 1$.** Mengurangkan versi l membuang a_l dari rentang. Contoh soal menangkapnya hanya bila beruntung; *stress test* menangkapnya seketika.
- **Kumpulan simpul yang terlalu kecil.** Hitunglah [Persamaan](#) (3.3). Dengan kumpulan berukuran `MAXN * 17`, program berjalan pada masukan kecil dan menulis melewati ujung larik pada masukan terbesar.
- **Mencetak peringkat.** Penurunan mengembalikan *peringkat*; jawabannya adalah `vals[rank]`.
- **Menuruni dua versi yang bentuknya berbeda.** Kedua versi harus dibangun atas rentang nilai yang sama $[1, U]$. Membangun setiap versi atas “nilai-nilai yang sudah terlihat” akan merusak [Persamaan](#) (3.4).

3.7.7 Stress test

Kode 3.4 menjawab setiap pertanyaan dengan cara yang jelas: salin rentangnya, urutkan, baca posisi ke- k .

Kode 3.4 `brute.cpp`: urutkan salinan setiap rentang, $O(n \log n)$ per pertanyaan.

```
// Brute force for MKTHNUM: copy a[l..r], sort it, read position k.
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];
    while (m--) {
        int l, r, k;
        cin >> l >> r >> k;
        vector<int> part(a.begin() + l, a.begin() + r + 1);
        sort(part.begin(), part.end());
        cout << part[k - 1] << "\n";
    }
    return 0;
}
```

Pembangkit **Kode 3.5** menghasilkan larik pada tiga skala nilai; skala mungil (± 3) menimbulkan banyak duplikat, yang menguji ketangguhan yang dijanjikan di Subbab 3.7.1.

Kode 3.5 `gen.py`: masukan MKTHNUM acak, dengan dan tanpa duplikat.

```
#!/usr/bin/env python3
"""Random MKTHNUM input: python3 gen.py SEED"""
import random
import sys

rng = random.Random(int(sys.argv[1]) if len(sys.argv) > 1 else 1)
n, m = rng.randint(1, 12), rng.randint(1, 10)
lim = rng.choice([3, 20, 10**9])
print(n, m)
print(*[rng.randint(-lim, lim) for _ in range(n)])
for _ in range(m):
    i = rng.randint(1, n)
    j = rng.randint(i, n)
    print(i, j, rng.randint(1, j - i + 1))
```

all 500 tests identical

Pada masukan terbesar, $N = 100\,000$ nilai acak dengan nilai mutlak hingga 10^9 dan $M = 5\,000$ pertanyaan acak, solusi berjalan sekitar 0,09 detik di mesin kami; sebagian besar waktu itu dipakai untuk membaca masukan dan mengurutkan demi kompresi.

3.7.8 Biaya dalam angka

Analisis perlu dicocokkan dengan batasan nyata, karena kebiasaan inilah yang mencegah kejutan batas waktu. Membaca 10^5 bilangan dengan `scanf` memakan beberapa milidetik. Mengurutkannya untuk kompresi memakan sekitar $N \log_2 N \approx 1.7 \times 10^6$ perbandingan. Membangun N versi membuat 1.8×10^6 simpul, masing-masing dengan segelintir operasi tulis ke larik. Setiap dari 5 000 pertanyaan menuruni 18 tingkat, menyentuh empat sel larik per tingkat: kurang dari 4×10^5 akses memori seluruhnya. Semua angka ini jauh di bawah 10^8 operasi per detik pada Tabel 1.2 di Bab 1, dan konsisten dengan 0,09 detik yang terukur.

Memori adalah sumber daya yang lebih ketat. Tiga larik berisi `MAXNODES` bilangan bulat menempati sekitar 21,6 MB, dan larik lainnya 1,2 MB lagi. Itu lega di SPOJ dan di babak nasional, tetapi inilah jenis angka yang sebaiknya Anda tulis di pinggir kertas sebelum menulis kode, karena menggandakannya (misalnya dengan menyimpan jumlah 64 bit di simpul) atau mengalikannya dengan banyaknya kasus uji bisa tiba-tiba menjadi masalah.

3.8 Kerabat dan Alternatif

Pohon persisten yang sama menjawab pertanyaan lain dengan penurunan yang sama: banyaknya nilai yang paling besar x di $[l, r]$ (turun menuju peringkat x sambil menjumlahkan cacah kiri), peringkat x , atau jumlah k nilai terkecil (simpan jumlah di samping setiap cacah). Alat lain juga menyelesaikan soal terkecil ke- k , dan seorang peserta sebaiknya mengetahui paling tidak nama dan untung-ruginya:

- *Merge sort tree* menyimpan, di setiap simpul pohon segmen biasa atas posisi, daftar terurut elemen-elemennya. Kueri cacah memakan $O(\log^2 N)$, dan kueri ke- k memerlukan pencarian biner tambahan atas nilai, $O(\log^3 N)$. Lebih mudah dinalar tetapi lebih lambat.
- *Wavelet tree* menjawab terkecil ke- k dalam $O(\log U)$ dengan memori lebih sedikit daripada persistensi; survei Navarro [3] adalah pengantar bakunya.
- Metode *offline*, seperti mengurutkan pertanyaan dan memprosesnya dengan pohon Fenwick serta pencarian biner paralel, menghindari persistensi sama sekali bila semua pertanyaan diketahui di muka.

Untuk soal statistik urutan klasik pada satu larik (terkecil ke- k dari semuanya), ada algoritma seleksi berwaktu linear; Cormen dkk. [4] menyajikan metode acak maupun metode linear kasus terburuk. Pohon persistenlah yang membuat versi *rentang*-nya cepat. Buku-buku untuk peserta lomba, seperti Laaksonen [5] dan Halim dkk. [6], memperlakukan persistensi sebagai teknik lanjut tetapi baku, dan teknik ini muncul secara teratur dalam soal seleksi nasional dan IOI.

3.9 Taktik Hari Lomba

- **Kenali polanya.** “Rentang”, “terkecil ke- k ”, “median”, “berapa nilai di bawah x dalam rentang”, dengan nilai yang terlalu besar untuk tabel: pikirkan kompresi ditambah pohon persisten.
- **Hitung memori lebih dulu.** Terapkan [Persamaan \(3.3\)](#) sebelum menulis apa pun. Bila hasil kali N dan kedalamannya terlalu besar, lihat alternatif *offline* di Subbab 3.8.
- **Amankan subsoal pengurutan.** [Kode 3.4](#) menyelesaikan subsoal kecil dan menjadi acuan *stress test*.
- **Pakai larik, bukan penunjuk.** Larik sejajar dengan indeks bilangan bulat lebih cepat ditulis, lebih mudah di-*debug*, dan menghindari *new* di dalam perulangan.
- **Uji dengan duplikat** bahkan ketika pernyataan soal menjanjikan nilai yang berbeda.

3.10 Ringkasan

- Kompresi koordinat memetakan hingga N nilai berbeda ke peringkat $1, \dots, U$ dengan [sort](#), [unique](#), dan [lower_bound](#).
- Pohon segmen atas nilai dengan cacah menemukan nilai tersisip terkecil ke- k melalui penurunan [Persamaan \(3.2\)](#) dalam $O(\log U)$ langkah.
- Penyalinan lintasan menyimpan setiap versi pohon dengan hanya menyalin $\lceil \log_2 U \rceil + 1$ simpul di lintasan yang diperbarui; memorinya diberikan oleh [Persamaan \(3.3\)](#).
- Karena penyisipan bersifat aditif, mengurangi versi $l - 1$ dari versi r ([Persamaan \(3.4\)](#)) memberi cacah rentang $[l, r]$, dan penurunan menuruni kedua versi bersama-sama.
- MKTHNUM diselesaikan dalam waktu $O((N + M)\log N)$ dengan sekitar 1.8×10^6 simpul.

3.11 Soal Latihan

1. Kompresi nilai 40, -3, 40, 17, 0 dengan tangan. Peringkat apa yang Anda peroleh, dan berapa U ?
2. Berapa simpul yang diberikan [Persamaan \(3.3\)](#) untuk $N = 50\,000$ nilai berbeda? Berapa bita dengan tiga larik [int](#)?
3. Telusuri penurunan untuk pertanyaan “terkecil pertama di posisi 1 sampai 7” pada larik contoh.
4. Ubah [kth](#) menjadi fungsi [countAtMost\(vl, vr, lo, hi, x\)](#) yang mengembalikan banyaknya nilai dalam rentang yang peringkatnya paling besar x . Tuliskan rekursinya dengan kata-kata.
5. Mengapa versi 0 harus ada, alih-alih mulai dari versi 1?
6. Lariknya 2, 2, 2 dan pertanyaannya (1,3,2). Apakah solusinya bekerja, dan apa yang dicetaknya?
7. Perluas simpul agar sebuah pertanyaan juga dapat menanyakan *jumlah* k nilai terkecil dari $a_i, \dots, a_j$. Apa yang harus disimpan setiap simpul, dan bagaimana penurunannya berubah?

3.12 Penyelesaian Soal Latihan

1. Nilai berbeda yang terurut adalah $-3, 0, 17, 40$, sehingga $U = 4$ dan peringkatnya $4, 1, 4, 3, 2$.
2. $\lceil \log_2 50\,000 \rceil = 16$, jadi $1 + 50\,000 \times 17 = 850\,001$ simpul, dan $850\,001 \times 12 \approx 10.2$ MB.
3. Versi 0 dan 7. Di $[1,7]$ anak kiri $[1,4]$ bercacah $4 - 0 = 4 \geq 1$: turun ke kiri. Di $[1,4]$ anak kiri $[1,2]$ bercacah 2: turun ke kiri. Di $[1,2]$ anak kiri $[1]$ bercacah 1: turun ke kiri. Daunnya peringkat 1, jadi jawabannya nilai 1.
4. Di daun, kembalikan selisih cacah daun itu bila peringkatnya paling besar x , selainnya 0. Di simpul dalam dengan titik tengah mid : bila $x \leq mid$, kembalikan hasil untuk anak-anak kiri saja; selainnya kembalikan seluruh selisih cacah kiri (setiap nilai di sana paling besar x) ditambah hasil untuk anak-anak kanan. Fungsi ini mengunjungi $O(\log U)$ simpul.
5. Pertanyaan dengan $l = 1$ mengurangi versi $l - 1 = 0$. Versi 0 adalah pohon kosong yang semua cacahnya nol; tanpanya, pertanyaan yang dimulai di posisi 1 memerlukan kasus khusus.
6. Bekerja: hanya ada satu nilai berbeda, $U = 1$, dan setiap versi berupa satu daun dengan cacah 1, 2, atau 3. `kth` langsung mengembalikan peringkat 1, dan program mencetak 2.
7. Setiap simpul juga menyimpan sum , jumlah nilai asli yang disisipkan ke rentang nilainya, bertipe `long long` (hingga 10^5 nilai sebesar 10^9 melampaui 32 bit). `insert` menambahkan nilai yang disisipkan ke setiap simpul yang disalin. Selama penurunan, setiap kali turun ke kanan, seluruh anak kiri rentang itu termasuk k nilai terkecil, jadi selisih jumlahnya (versi r dikurangi versi $l - 1$) ditambahkan ke akumulator. Di daun, k salinan sisa dari nilai daun ditambahkan. Penurunan tetap $O(\log U)$, dan memori bertambah 8 bita per simpul, menjadi sekitar 36 MB untuk $N = 10^5$.

Daftar Pustaka

- [1] Sphere Online Judge, “MKTHNUM — K-th Number.” [Daring]. Tersedia: <https://www.spoj.com/problems/MKTHNUM/> (diakses Sep. 2026).
- [2] J. R. Driscoll, N. Sarnak, D. D. Sleator, dan R. E. Tarjan, “Making data structures persistent,” *Journal of Computer and System Sciences*, vol. 38, no. 1, pp. 86–124, 1989.
- [3] G. Navarro, “Wavelet trees for all,” *Journal of Discrete Algorithms*, vol. 25, pp. 2–20, 2014.
- [4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, dan C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
- [5] A. Laaksonen, *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, 3rd ed. Cham, Switzerland: Springer, 2024.
- [6] S. Halim, F. Halim, dan S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.

Bab 4

Tumpukan Monoton dan Teknik Larik Linear — HISTOGRA

Sebagian soal dengan N hingga seratus ribu tampak seperti menuntut peralatan berat, padahal jawabannya satu lintasan kiri-ke-kanan dengan sebuah tumpukan yang dijaga tetap terurut. Bab ini membangun alat itu, *monotonic stack* (tumpukan monoton), dari dasar. Anda akan belajar menghitung, untuk setiap elemen larik, tetangga lebih kecil terdekatnya di kedua sisi dalam satu lintasan; membuktikan, dengan argumen pencacahan sederhana, bahwa perulangan di dalam perulangan tetap dapat berjalan dalam waktu linear; dan mengenali keluarga soal yang jawaban setiap elemennya ditentukan oleh tetangga pertama yang melanggar suatu syarat. Studi kasusnya adalah soal klasik dunia HISTOGRA, persegi panjang terbesar dalam histogram, kegemaran juri olimpiade karena singkat dinyatakan, mudah diselesaikan secara lambat, dan indah diselesaikan dalam $O(N)$.

4.1 Spanduk di Jalan Utama

Untuk perayaan 17 Agustus, sebuah desa ingin memasang spanduk persegi panjang sebesar mungkin di muka bangunan-bangunan di jalan utamanya. Bangunan-bangunan itu berdiri berdampingan, masing-masing tepat selebar satu satuan, dengan tinggi yang berbeda-beda: toko dua lantai, warung satu lantai, menara masjid yang tinggi, rumah kecil. Spanduk harus berupa persegi panjang yang bertumpu pada garis tanah, dan tidak boleh lebih tinggi daripada bangunan mana pun yang ditutupinya. Persegi panjang mana yang luasnya terbesar?

Cobalah pada tujuh bangunan dengan tinggi 2, 1, 4, 5, 1, 3, 3. Spanduk di atas satu bangunan tertinggi berluas $5 \times 1 = 5$. Spanduk di atas dua bangunan tinggi di tengah bertinggi 4 (tidak boleh melebihi yang lebih pendek) dan berlebar 2: luasnya 8. Spanduk di sepanjang jalan bertinggi 1 dan berlebar 7: luasnya 7. Yang terbaik 8, tetapi menemukannya untuk jalan berisi seratus ribu bangunan adalah perkara lain. Ada sekitar $N^2/2$ cara memilih bangunan pertama dan terakhir spanduk, lima miliar untuk $N = 10^5$. Itulah soal SPOJ HISTOGRA [1].

4.2 Gagasannya dengan Kata-Kata Sederhana

Perhatikan spanduk terbaik. Tepi atasnya menyentuh suatu bangunan, yaitu bangunan terendah yang ditutupinya; kalau tidak, spanduk itu bisa dinaikkan sedikit dan menjadi lebih besar. Jadi spanduk terbaik adalah, untuk suatu bangunan i , spanduk terlebar bertinggi tepat h_i yang memuat bangunan i . Seberapa lebar? Ia membentang ke kiri sampai bangunan pertama yang lebih pendek daripada h_i , dan ke kanan sampai bangunan pertama yang lebih pendek daripada h_i . Bila L_i adalah indeks bangunan terdekat di kiri yang lebih pendek secara tegas (atau -1 bila tidak ada) dan R_i yang terdekat di kanan (atau N bila tidak ada), dengan bangunan dinomori 0 sampai $N - 1$, maka spanduk untuk bangunan i menutupi bangunan-bangunan yang terletak tegas di antara keduanya, dan

$$area_i = h_i \cdot (R_i - L_i - 1) \quad (4.1)$$

Jawabannya adalah $area_i$ terbesar. Jadi seluruh soal menyusut menjadi menghitung *elemen lebih kecil terdekat* di kedua sisi setiap posisi, dan itulah yang dikerjakan tumpukan monoton dalam waktu linear.

4.2.1 Dua jalan yang lebih lambat menuju jawaban yang sama

Sebelum metode cepat, ada gunanya melihat metode yang lebih lambat, karena dalam lomba keduanya adalah solusi subsoal dan acuan *stress test*. Metode paling langsung menetapkan bangunan pertama i dan menggeser bangunan terakhir j ke kanan, sambil menyimpan tinggi minimum berjalan m dari bangunan i sampai j ; persegi panjang terbaik yang membentang tepat di atas bangunan-bangunan itu berluas $m \cdot (j - i + 1)$. Waktunya $O(N^2)$: cukup untuk $N \leq 5\,000$, tanpa harapan untuk 10^5 .

Jalan yang lebih cerdas adalah *divide and conquer* (bagi dan kuasai). Batang terendah seluruh histogram, di posisi p , membelah soalnya: persegi panjang terbaik membentang di seluruh histogram setinggi h_p (luas $h_p \cdot N$), atau seluruhnya di kiri p , atau seluruhnya di kanan p , karena tidak ada persegi panjang yang dapat melintasi batang p lebih tinggi daripada h_p . Selesaikan kedua sisi secara rekursif dan ambil yang terbaik dari ketiganya. Dengan *sparse table* atau pohon segmen untuk mencari minimum sebuah rentang, setiap panggilan memakan $O(\log N)$ atau $O(1)$, dan ada N panggilan, sehingga metode ini berjalan dalam $O(N \log N)$ bila belahannya seimbang. Namun pada histogram yang terurut, kedalaman rekursinya menjadi N , yang dapat meluapkan *call stack* untuk $N = 10^5$. Tumpukan monoton menghindari logaritma sekaligus rekursi yang dalam.

4.3 Soal Pemanasan

4.3.1 Elemen lebih kecil terdekat

Berjalanlah dari kiri ke kanan dan simpan tumpukan berisi indeks yang tingginya menaik dari dasar ke puncak. Ketika bangunan i tiba, setiap indeks di puncak tumpukan yang tingginya paling sedikit h_i tidak akan pernah lagi menjadi elemen lebih kecil terdekat di kiri bagi siapa pun: bangunan i paling tidak sama rendahnya dan lebih dekat ke semua yang datang kemudian. Maka keluarkan (*pop*) mereka. Apa pun yang tersisa di puncak adalah bangunan terdekat yang tegas lebih rendah daripada h_i , yaitu L_i . Lalu masukkan (*push*) i . Sisi kanan adalah bayangan cerminnya, satu lintasan dari kanan ke kiri. **Kode 4.1** menghitung keduanya dan mencetak *Persamaan* (4.1) untuk setiap bangunan.

Kode 4.1 `nearest_smaller.cpp`: elemen tegas lebih kecil terdekat di setiap sisi, dua lintasan dengan tumpukan.

```
// Warm-up: for every bar, the nearest strictly smaller bar on each side.
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n;
    if (scanf("%d", &n) != 1) return 0;
    vector<long long> h(n);
    for (auto& x : h)
        if (scanf("%lld", &x) != 1) return 0;
    vector<int> L(n), R(n), st;
    for (int i = 0; i < n; i++) { // Left to right
        while (!st.empty() && h[st.back()] >= h[i]) st.pop_back();
        L[i] = st.empty() ? -1 : st.back(); // -1: nothing smaller on the left
        st.push_back(i);
    }
    st.clear();
    for (int i = n - 1; i >= 0; i--) { // right to left, the mirror image
        while (!st.empty() && h[st.back()] >= h[i]) st.pop_back();
        R[i] = st.empty() ? n : st.back(); // n: nothing smaller on the right
        st.push_back(i);
    }
    for (int i = 0; i < n; i++)
        printf("i=%d h=%lld L=%d R=%d area=%lld\n", i, h[i], L[i], R[i],
            h[i] * (R[i] - L[i] - 1));
    return 0;
}
```

Pada jalan di Subbab 4.1:

Bangunan 2 (tinggi 4) memiliki tetangga lebih pendek terdekat di posisi 1 dan 4, sehingga spanduknya menutupi posisi 2 dan 3: luas 8, jawabannya. Bangunan 1 dan bangunan 4 (keduanya bertinggi 1) tidak memiliki bangunan yang tegas lebih pendek di mana pun, sehingga masing-masing dapat membentang di sepanjang jalan: luas 7.

4.3.2 Rentang saham

Soal klasik kedua memakai tumpukan yang *menurun*. *Span* (rentang) harga saham suatu hari adalah banyaknya hari berturut-turut, berakhir hari ini, yang harganya paling tinggi sama dengan harga hari ini. Tumpukan pasangan (harga, rentang) dengan harga yang tegas menurun menjawabnya: ketika harga hari ini tiba, setiap hari di tumpukan yang harganya paling tinggi harga hari ini ditelan, dan rentangnya ditambahkan ke rentang hari ini. **Kode 4.2** memperlihatkan seluruh programnya.

Kode 4.2 `stock_span.cpp`: rentang saham dengan tumpukan menurun berisi pasangan (harga, rentang).

```
// Warm-up: stock span, how many consecutive days up to today had a price
// no higher than today's. A decreasing stack of (price, span) pairs.
#include <cstdio>
#include <utility>
#include <vector>
using namespace std;

int main() {
    int n;
    if (scanf("%d", &n) != 1) return 0;
    vector<pair<int, int>> st; // prices strictly decrease from bottom to top
    for (int i = 0; i < n; i++) {
        int price, span = 1;
        if (scanf("%d", &price) != 1) return 0;
        while (!st.empty() && st.back().first <= price) {
            span += st.back().second; // swallow the days that bar already covered
            st.pop_back();
        }
        st.push_back({price, span});
        printf("%d ", span);
    }
    printf("\n");
    return 0;
}
```

Untuk harga 100, 80, 60, 70, 60, 75, 85:

Pada hari keenam, harga 75 menelan pasangan (60, 1) dan (70, 2), yang sebelumnya sudah menelan sebuah 60 yang lebih awal; rentangnya $1 + 1 + 2 = 4$. Tumpukan itu mengingat, dalam bentuk yang dipadatkan, segala sesuatu yang masih mungkin dibutuhkan masa depan.

4.4 Tumpukan Monoton

4.4.1 Invariannya

Pada kedua pemanasan, tumpukan memenuhi sebuah *invariant*, pernyataan yang benar sebelum dan sesudah setiap langkah perulangan. Untuk elemen lebih kecil terdekat, invariannya: *tumpukan memuat indeks-indeks menurut urutan posisi yang menaik, dengan tinggi yang tegas menaik, dan indeks-indeks itu tepat merupakan posisi-posisi yang sudah terlihat yang tidak memiliki posisi tegas lebih rendah di kanannya di antara posisi-posisi yang sudah terlihat*. Hanya posisi-posisi itulah yang masih mungkin menjadi elemen lebih kecil terdekat bagi bangunan yang akan datang. Ketika bangunan baru i tiba, posisi yang kehilangan sifat ini tepat yang tingginya paling sedikit h_i , dan mereka berada di puncak tumpukan, karena tingginya menaik ke arah puncak. Mengeluarkan mereka lalu memasukkan i memulihkan invariannya.

Secara geometris, tumpukan itu adalah “tangga yang terlihat” ketika Anda memandang ke kiri dari bangunan saat ini: bangunan terendah, lalu yang terendah berikutnya di kanannya yang lebih tinggi, dan seterusnya sampai bangunan saat ini.

4.4.2 Mengapa perulangan di dalam perulangan tetap linear

Kodenya memiliki perulangan `while` di dalam perulangan `for`, yang tampak kuadratik. Padahal tidak, dan argumennya layak dipelajari karena berlaku pada banyak algoritma. Cacahlah operasi pada tumpukan, bukan iterasi perulangannya. Setiap indeks dimasukkan tepat sekali. Setiap

indeks dikeluarkan paling banyak sekali, karena sekali keluar ia tidak pernah kembali. Setiap iterasi **while** di dalam mengeluarkan satu indeks. Karena itu, sepanjang seluruh eksekusi,

$$\text{push} + \text{pop} \leq N + N = 2N \quad (4.2)$$

dan total kerjanya $O(N)$, meskipun satu langkah bisa mengeluarkan banyak indeks. Cara membebaskan biaya langkah yang mahal kepada langkah-langkah murah sebelumnya ini disebut *amortized analysis* (analisis amortisasi); Cormen dkk. [2] menyajikannya secara umum, dan makalah Tarjan [3] adalah sumber klasiknya. Kebiasaan yang penting adalah bertanya “berapa kali setiap elemen dapat disentuh secara keseluruhan?” alih-alih “berapa lama satu langkah dapat berlangsung?”

4.4.3 Empat varian, satu pola

Perulangan yang sama menjawab empat pertanyaan yang berkerabat, bergantung pada urutan yang dijaga di tumpukan dan perbandingan yang memicu pengeluaran. **Tabel 4.1** mendaftarkannya untuk lintasan kiri-ke-kanan; lintasan kanan-ke-kiri memberi tetangga di kanan. Tabel ini layak dihafal sebagai satu kesatuan alih-alih empat program terpisah: tumpukan menyimpan elemen-elemen yang merupakan *calon lebih baik* daripada semua yang di atasnya, dan syarat pengeluaran membuang tepat elemen-elemen yang dijadikan tak berguna oleh pendatang baru.

Tabel 4.1 Empat pertanyaan tetangga terdekat dan syarat pengeluaran yang menjawab masing-masing dalam satu lintasan kiri-ke-kanan.

Pertanyaan untuk posisi i (memandang ke kiri)	Urutan tumpukan, dasar ke puncak	Keluarkan selama puncaknya
terdekat yang tegas lebih kecil	tegas menaik	$\geq h_i$
terdekat yang lebih kecil atau sama	tidak menurun	$> h_i$
terdekat yang tegas lebih besar	tegas menurun	$\leq h_i$
terdekat yang lebih besar atau sama	tidak menaik	$< h_i$

Baris pertama adalah **Kode 4.1**; rentang saham pada **Kode 4.2** adalah baris ketiga, karena hari yang menghentikan sebuah rentang adalah hari terdekat dengan harga yang tegas lebih besar. Ketika sebuah soal memerlukan “terdekat yang lebih kecil” di satu sisi dan “terdekat yang lebih kecil atau sama” di sisi lain, seperti soal jumlah minimum subarray agar tidak mencacah subarray dua kali ketika ada nilai kembar, tabel ini langsung memberi tahu syarat pengeluaran mana yang ditulis di setiap lintasan.

4.5 Satu Lintasan: Jawaban pada Saat Pengeluaran

Metode dua lintasan pada **Kode 4.1** sudah menyelesaikan soal dalam waktu linear. Ada satu lintasan yang lebih rapi, dan itulah yang dipakai dalam studi kasus. Perhatikan saat bangunan j dikeluarkan karena bangunan i dengan $h_i < h_j$ tiba:

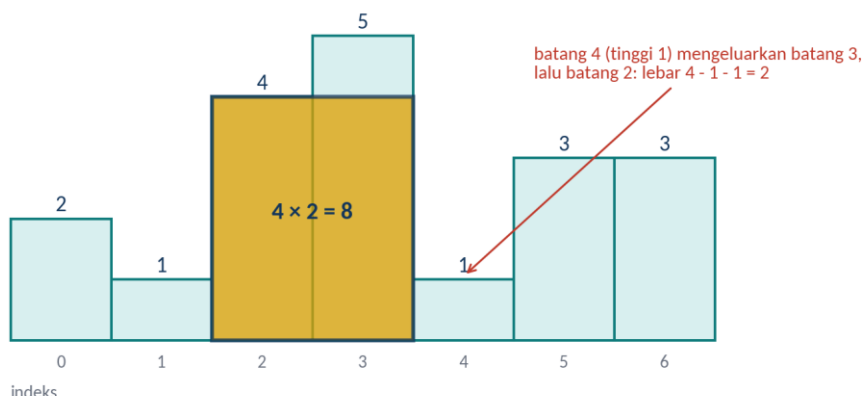
- bangunan i adalah bangunan pertama di kanan j yang tegas lebih rendah, jadi $R_j = i$;
- indeks tepat di bawah j dalam tumpukan adalah bangunan terdekat di kiri j yang lebih rendah atau sama dengan h_j , sehingga ia berperan sebagai L_j (bila tumpukan menjadi kosong, tidak ada, dan ujung kirinya -1).

Jadi pada saat itu semua yang diperlukan **Persamaan** (4.1) sudah diketahui, dan luas calon untuk j dihitung di situ juga:

$$width_j = \begin{cases} i & \text{jika tumpukan kosong sesudah } j \text{ dikeluarkan} \\ i - t - 1 & \text{dengan } t \text{ puncak yang baru} \end{cases} \quad (4.3)$$

Agar setiap bangunan pada akhirnya dikeluarkan, sebuah *sentinel* setinggi 0 diproses sesudah bangunan terakhir, di posisi N . Sentinel itu lebih rendah daripada semuanya, sehingga mengosongkan tumpukan, dan tidak diperlukan kasus khusus sesudah perulangan.

Gambar 4.1 memperlihatkan saat kuncinya di jalan kita: bangunan 4, setinggi 1, mengeluarkan bangunan 3 (tinggi 5, lebar 1, luas 5) lalu bangunan 2 (tinggi 4). Sesudah bangunan 2 dikeluarkan, puncak barunya bangunan 1, sehingga lebarnya $4 - 1 - 1 = 2$ dan luasnya 8.



Gambar 4.1 Histogram 2, 1, 4, 5, 1, 3, 3 dan persegi panjang terbesarnya; persegi panjang itu ditemukan ketika batang 4 mengeluarkan batang 2 dari tumpukan.

Tabel 4.2 menelusuri seluruh lintasan. Setiap baris adalah satu nilai i ; tumpukan ditulis dari dasar ke puncak sebagai indeks.

Tabel 4.2 Algoritma satu lintasan pada 2, 1, 4, 5, 1, 3, 3; setiap luas calon dihitung pada saat batangnya dikeluarkan.

i	Tinggi	Dikeluarkan: (indeks, tinggi, lebar, luas)	Tumpukan sesudahnya	Terbaik
0	2	tidak ada	0	0
1	1	(0, 2, 1, 2)	1	2
2	4	tidak ada	1, 2	2
3	5	tidak ada	1, 2, 3	2
4	1	(3, 5, 1, 5), (2, 4, 2, 8)	1, 4	8
5	3	tidak ada	1, 4, 5	8
6	3	tidak ada (tinggi sama tetap tinggal)	1, 4, 5, 6	8
7	0 (sentinel)	(6, 3, 1, 3), (5, 3, 2, 6), (4, 1, 5, 5), (1, 1, 7, 7)	7	8

Perhatikan baris terakhir. Bangunan 6 dikeluarkan lebih dulu dengan lebar 1, padahal spanduk setinggi 3 dapat menutupi bangunan 5 dan 6. Itu tidak merugikan: bangunan 5 dikeluarkan berikutnya dengan lebar 2 dan luas 6, yang meliputi spanduk yang sama. Pada tinggi yang sama, batang yang *lebih awal* mendapat lebar penuh. Begitu pula bangunan 4 mendapat lebar 5 sementara bangunan 1, dengan tinggi yang sama, mendapat lebar penuh 7. Lintasan tunggal ini tidak menghitung setiap $area_i$ pada [Persamaan \(4.1\)](#) secara tepat, tetapi selalu menghitung nilai yang benar untuk paling sedikit satu batang dari setiap spanduk maksimal, dan hanya itu yang kita perlukan untuk maksimum.

4.5.1 Mengapa satu lintasan menemukan maksimum

Misalkan persegi panjang terbaik bertinggi H dan menutupi batang a sampai b . Tingginya sama dengan batang terendah di $[a, b]$, dan karena tidak dapat diperlebar, batang $a - 1$ (bila ada) dan batang $b + 1$ (bila ada) tegas lebih rendah daripada H . Di antara batang di $[a, b]$ yang tingginya tepat H , ambil yang *paling kiri*, sebut j . Setiap batang antara a dan $j - 1$ tegas lebih tinggi daripada H .

Sekarang ikuti batang j sepanjang lintasan. Ia dimasukkan pada saat j . Ia tidak mungkin dikeluarkan sebelum saat $b + 1$, karena setiap batang di $(j, b]$ bertinggi paling sedikit H , dan lintasan hanya mengeluarkan untuk batang yang tegas lebih rendah. Ia dikeluarkan pada saat $b + 1$ (batang $b + 1$ lebih rendah, atau $b + 1 = N$ adalah sentinel). Pada saat itu, apa yang berada di bawah j dalam tumpukan? Setiap batang antara a dan $j - 1$ lebih tinggi daripada h_j , sehingga masing-masing sudah dikeluarkan oleh j pada saat j , atau lebih awal oleh batang lain. Batang $a - 1$, yang lebih rendah daripada H , masih ada, karena batang-batang di $[a, b]$ semuanya paling sedikit H dan tidak dapat mengeluarkannya. Jadi puncak baru sesudah j dikeluarkan tepat $a - 1$. Lebar yang dihitung [Persamaan \(4.3\)](#) karenanya $(b + 1) - (a - 1) - 1 = b - a + 1$, dan luas yang dihitung untuk j tepat $H \cdot (b - a + 1)$, yaitu optimumnya. Setiap luas lain yang dihitung adalah luas suatu persegi panjang nyata, sehingga tidak pernah melampaui optimum. Maka maksimum dari luas-luas yang dihitung adalah jawabannya.

Gaya pembuktian ini, mengikuti satu elemen yang dipilih dengan cermat sepanjang algoritma, adalah cara baku untuk bernalar tentang algoritma tumpukan, dan layak ditulis sekali dengan tangan.

4.6 Studi Kasus SULIT: HISTOGRA

4.6.1 Soal dengan kata-kata kita sendiri

Masukan berisi beberapa kasus uji, satu per baris. Setiap baris diawali n ($1 \leq n \leq 100\,000$) disusul n tinggi $h_1, \dots, h_n$ dengan $0 \leq h_i \leq 10^9$; baris yang hanya berisi 0 mengakhiri masukan. Histogram terdiri atas n batang selebar 1 yang berdiri berdampingan pada garis dasar yang sama. Untuk setiap kasus uji, cetak luas persegi panjang sejajar sumbu terbesar yang seluruhnya berada di dalam histogram.

4.6.2 Contoh masukan dan keluaran

```
7 2 1 4 5 1 3 3
4 1000 1000 1000 1000
0
```

Baris pertama adalah jalan kita. Baris kedua adalah empat batang setinggi 1000: seluruh histogram merupakan persegi panjang berluas 4000.

4.6.3 Rancangan

Batasannya (Tabel 1.2 di Bab 1) mengizinkan $O(n \log n)$ atau $O(n)$ per kasus uji. Lintasan tunggal Subbab 4.5 bernilai $O(n)$ menurut [Persamaan \(4.2\)](#). Dua ukuran perlu diperhatikan. Tinggi mencapai 10^9 dan lebar 10^5 , sehingga luas mencapai 10^{14} , jauh melampaui 32 bit: tinggi, lebar, dan luas bertipe **long long**. Dan karena banyaknya kasus uji tidak diberikan, program harus terus membaca sampai menemukan 0.

4.6.4 Solusi lengkap

Kode 4.3 adalah program lengkapnya; intinya delapan baris.

Kode 4.3 `HISTOGRAM.cpp`: satu lintasan dengan tumpukan monoton dan sentinel setinggi 0.

```
// HISTOGRAM (SPOJ): Largest rectangle in a histogram, O(n) per test.
// Monotonic stack of indices whose heights increase from bottom to top.
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

long long h[100001];

int main() {
    int n;
    while (scanf("%d", &n) == 1 && n != 0) {
        for (int i = 0; i < n; i++)
            if (scanf("%lld", &h[i]) != 1) return 0;
        vector<int> st; // indices; h[st[0]] <= h[st[1]] <= ... (bottom to top)
        st.reserve(n);
        long long best = 0;
        for (int i = 0; i <= n; i++) {
            long long cur = (i < n) ? h[i] : 0; // bar n is a height-0 sentinel
            // <P50>
            // Pop every bar taller than cur: cur is its first shorter bar on the
            // right, the new top is its first shorter-or-equal bar on the left, so
            // its widest rectangle spans the gap in between. Then push i.
            // >>> YOUR TURN (Public Edition): write this part yourself, following the hint
            above. <<<
            // </P50>
        }
        printf("%lld\n", best);
    }
    return 0;
}
```

4.6.5 Baris demi baris

Membaca. `while (scanf("%d", &n) == 1 && n != 0)` menangani 0 penutup sekaligus akhir masukan yang tak terduga. Tinggi dibaca ke larik global `long long` bernama `h`, sehingga hasil kali tidak pernah meluap.

Tumpukan. `vector<int>` yang dipakai sebagai tumpukan (`push_back`, `back`, `pop_back`) secepat `std::stack` dan memungkinkan kita memesan memori sekali dengan `reserve`. Isinya indeks, bukan tinggi; tinggi dilihat di `h`.

Perulangan. `i` berjalan dari 0 sampai `n` inklusif. Untuk `i = n`, `cur` adalah tinggi sentinel 0.

Mengeluarkan. Di antara penanda `// <P50>` dan `// </P50>`, perulangan `while` mengeluarkan setiap batang yang tegas lebih tinggi daripada `cur`. Untuk setiap batang yang dikeluarkan, ia membaca tinggi batang itu, mengeluarkannya, menghitung lebarnya dengan [Persamaan \(4.3\)](#), dan memperbarui luas terbaik. Lalu `i` dimasukkan. Perbandingannya tegas (`>`), sehingga batang yang tingginya sama tetap tinggal di tumpukan; Subbab 4.5 memperlihatkan mengapa itu benar.

Keluaran. Sesudah lintasan, `best` dicetak dengan `%lld`.

4.6.6 Jebakan

- **Luas 32 bit.** $10^9 \times 10^5$ tidak muat dalam `int`; bahkan dengan tinggi `long long`, menulis `int width` lalu mengalikan dua `int` akan meluap sebelum konversi.

- **Lupa sentinel.** Tanpa batang penutup setinggi 0, batang-batang yang masih di tumpukan pada akhir tidak pernah diproses; pada histogram menaik seperti 1, 2, 3 program akan mencetak 0.
- **Lebar yang salah ketika tumpukan kosong.** Pada saat itu batang yang dikeluarkan adalah yang terendah sejauh ini dan membentang sampai tepi kiri: lebarnya i , bukan $i - 1$.
- **Membaca seluruh masukan sebagai satu kasus uji.** Setiap baris adalah kasus uji tersendiri; perulangan harus mengulang tumpukan dan luas terbaik untuk masing-masing.
- **Tinggi nol.** Batang setinggi 0 sah. Ia mengeluarkan semuanya, dan luas calonnya sendiri 0; tidak diperlukan perlakuan khusus, tetapi sertakan dalam pengujian Anda.

4.6.7 Stress test

Kode 4.4 memeriksa setiap pasangan (i, j) dengan minimum berjalan: $O(n^2)$ dan jelas benar.

Kode 4.4 `brute.cpp`: setiap pasangan batang pertama dan terakhir, dengan tinggi minimum berjalan.

```
// Brute force for HISTOGRA: every pair (i, j) with the running minimum.
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n;
    while (cin >> n && n != 0) {
        vector<long long> h(n);
        for (auto& x : h) cin >> x;
        long long best = 0;
        for (int i = 0; i < n; i++) {
            long long low = h[i];
            for (int j = i; j < n; j++) {
                low = min(low, h[j]);
                best = max(best, low * (j - i + 1));
            }
        }
        cout << best << "\n";
    }
    return 0;
}
```

Pembangkit **Kode 4.5** menulis satu sampai empat baris uji disusul 0; tingginya berasal dari empat skala, termasuk histogram serba nol dan skala penuh 10^9 .

Kode 4.5 `gen.py`: masukan HISTOGRA acak, beberapa baris uji diakhiri 0.

```
#!/usr/bin/env python3
"""Random HISTOGRA input (several tests, then 0): python3 gen.py SEED"""
import random
import sys

rng = random.Random(int(sys.argv[1]) if len(sys.argv) > 1 else 1)
for _ in range(rng.randint(1, 4)):
    n = rng.randint(1, 12)
    lim = rng.choice([0, 3, 10, 10**9])
    print(n, *[rng.randint(0, lim) for _ in range(n)])
print(0)
```

all 500 tests identical

Pada empat baris uji masing-masing 10^5 batang (tiga acak dan satu tegas menaik, yang jawabannya 2 500 050 000), program berjalan sekitar 0,04 detik di mesin kami, sebagian besar untuk membaca bilangan.

4.7 Dari Histogram ke Matriks

Variasi kegemaran lainnya menanyakan persegi panjang terbesar yang seluruhnya berisi 1 dalam matriks biner berukuran R baris dan C kolom. Itu HISTOGRA yang menyamar. Untuk setiap baris r dan kolom c , misalkan $up_r[c]$ adalah banyaknya 1 berturut-turut yang berakhir di baris r pada kolom c :

$$up_r[c] = \begin{cases} up_{r-1}[c] + 1 & \text{jika sel } (r, c) \text{ bernilai 1} \\ 0 & \text{selainnya} \end{cases} \quad (4.4)$$

Setiap persegi panjang serba-1 memiliki baris dasar; untuk baris itu, persegi panjang tersebut adalah persegi panjang di bawah histogram up_r . Jadi menjalankan lintasan tunggal Subbab 4.5 pada histogram setiap baris dan mengambil jawaban terbaiknya menyelesaikan soal matriks dalam waktu $O(R \cdot C)$. **Persamaan** (4.4) memperbarui satu baris dari baris sebelumnya dalam $O(C)$, sehingga seluruh algoritma hanya beberapa baris di sekitar **Kode 4.3**.

Anggota keluarga lain layak dikenal namanya, karena babak nasional menyukainya: soal *trapping rain water* (berapa banyak air yang tertahan di antara batang sesudah hujan), *jumlah minimum subarray* (setiap elemen menyumbang h_i kali banyaknya subarray yang minimumnya elemen itu, yaitu lagi-lagi $(i - L_i)(R_i - i)$ dari elemen lebih kecil terdekat), dan soal *next greater element* pada larik melingkar. Laaksonen [4] dan Halim dkk. [5] mengumpulkan banyak di antaranya.

Soal air hujan adalah latihan kedua yang baik dengan semangat yang sama. Sesudah hujan, air di atas batang i naik sampai yang lebih rendah di antara batang tertinggi di kirinya dan batang tertinggi di kanannya, sehingga air di atas batang i adalah $\min(\max L_i, \max R_i) - h_i$, dengan $\max L_i$ dan $\max R_i$ mencakup batang i sendiri. Dua maksimum prefiks, satu dari setiap sisi, memberi jawaban dalam $O(N)$ tanpa tumpukan apa pun; tumpukan monoton memberi solusi $O(N)$ lain yang mengisi air lapis demi lapis setiap kali sebuah “lembah” tertutup. Melihat kedua solusi berdampingan adalah cara terbaik untuk belajar kapan tumpukan diperlukan dan kapan perhitungan prefiks sederhana sudah cukup.

WAWASAN KUNCI: Ketika jawaban setiap elemen ditentukan oleh elemen *pertama* di kiri atau kanannya yang melanggar suatu syarat (“batang pertama yang lebih pendek”, “harga pertama yang lebih tinggi”), pikirkan tumpukan monoton. Elemen yang tidak mungkin lagi menjadi jawaban siapa pun dikeluarkan, dan masing-masing dikeluarkan hanya sekali.

4.8 Taktik Hari Lomba

- **Kenali kata kuncinya.** “Terdekat yang lebih kecil atau lebih besar”, “berapa lama bertahan”, “persegi panjang maksimal”, “rentang”, “terlihat”. Masing-masing menunjuk tumpukan monoton.
- **Putuskan tegas atau tidak tegas dengan sengaja.** Tuliskan apa yang terjadi pada nilai yang sama sebelum menulis kode, dan uji kasus dengan nilai kembar, seperti 3, 3.
- **Pakai sentinel.** Batang setinggi 0 di akhir (atau batang yang sangat tinggi untuk tumpukan menurun) menghapus perulangan pembersihan beserta *bug*-nya.

- **Amankan subsoal $O(n^2)$. Kode 4.4** meraih subsoal kecil dan menjadi acuan *stress test*.
- **Cacah operasi tumpukan, bukan perulangan**, ketika memperkirakan waktu jalan.

4.9 Ringkasan

- Persegi panjang terbaik menyentuh batang terendahnya, sehingga luasnya adalah **Persamaan** (4.1) untuk suatu batang, dengan L_i dan R_i batang tegas lebih kecil terdekat.
- Tumpukan monoton menghitung elemen lebih kecil terdekat dalam satu lintasan; invariannya hanya menyimpan posisi yang masih mungkin menjadi jawaban seseorang.
- Setiap indeks dimasukkan sekali dan dikeluarkan paling banyak sekali, sehingga lintasannya $O(N)$ menurut **Persamaan** (4.2), meskipun perulangannya bersarang.
- HISTOGRAM menghitung setiap luas calon pada saat pengeluaran dengan lebar **Persamaan** (4.3), dan sentinel setinggi 0 mengosongkan tumpukan di akhir.
- Luas mencapai 10^{14} dan harus bertipe **long long**.
- Persegi panjang serba-1 terbesar dalam matriks adalah HISTOGRAM pada histogram baris **Persamaan** (4.4).

4.10 Soal Latihan

1. Telusuri **Kode 4.1** pada tinggi 3, 1, 3, 2, 2 dan berikan L_i serta R_i untuk setiap posisi.
2. Apa yang dicetak **Kode 4.3** untuk masukan **5 1 2 3 4 5** disusul **0**? Pengeluaran mana yang menghasilkan jawabannya?
3. Seorang teman mengubah perbandingan di HISTOGRAM dari **>** menjadi **>=**. Apakah programnya masih benar? Jelaskan.
4. Berikan histogram lima batang yang jawabannya dicapai oleh *dua* persegi panjang berbeda.
5. Dengan **Persamaan** (4.4), tuliskan histogram baris dari matriks berbaris **101**, **111**, **011**, lalu temukan persegi panjang serba-1 terbesar.
6. Jelaskan dalam dua kalimat mengapa tumpukan rentang saham pada **Kode 4.2** melakukan paling banyak $2N$ operasi *push* dan *pop*.

4.11 Penyelesaian Soal Latihan

1. $L = (-1, -1, 1, 1, 1)$ dan $R = (1, 5, 3, 5, 5)$. Misalnya posisi 2 (tinggi 3) memiliki posisi 1 (tinggi 1) di kiri dan posisi 3 (tinggi 2) di kanan; posisi 4 (tinggi 2) tidak memiliki batang tegas lebih kecil di kanannya, sehingga $R_4 = 5$.
2. Program mencetak 9. Batang-batang dimasukkan dalam urutan menaik; sentinel pada $i = 5$ mengeluarkannya dari puncak: (4, 5, lebar 1, luas 5), (3, 4, lebar 2, luas 8), (2, 3, lebar 3, luas 9), (1, 2, lebar 4, luas 8), (0, 1, lebar 5, luas 5). Jawaban 9 berasal dari pengeluaran batang 2, persegi panjang setinggi 3 di atas batang 2 sampai 4.
3. Ya. Dengan **>=**, sebuah batang dikeluarkan oleh batang yang sama tingginya, dan lebarnya lalu berhenti terlalu awal, tetapi batang *terakhir* dari deretan batang sama tinggi mendapat lebar penuh, karena tetangga kirinya di tumpukan tegas lebih rendah. Paling sedikit satu batang dari setiap persegi panjang maksimal tetap diukur dengan benar, sehingga maksimumnya tidak berubah.
4. Misalnya 2, 2, 0, 4, 1: persegi panjang setinggi 2 di atas batang 0 dan 1 serta batang tunggal setinggi 4 sama-sama berluas 4, dan tidak ada persegi panjang yang lebih besar. Program mencetak 4 untuk histogram itu.

5. Histogramnya $(1,0,1)$, $(2,1,2)$, dan $(0,2,3)$. Persegi panjang terbaiknya berluas 1, 3, dan 4. Jawabannya 4: tinggi 2 di atas dua kolom terakhir pada baris 2 dan 3.
6. Pasangan setiap hari dimasukkan tepat sekali, dan pasangan yang sudah dikeluarkan tidak pernah kembali, sehingga dikeluarkan paling banyak sekali. Karena itu banyaknya *pop* paling banyak sama dengan banyaknya *push*, yaitu N , dan totalnya paling banyak $2N$.

Daftar Pustaka

- [1] Sphere Online Judge, “HISTOGRA — Largest Rectangle in a Histogram.” [Daring]. Tersedia: <https://www.spoj.com/problems/HISTOGRA/> (diakses Sep. 2026).
- [2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, dan C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
- [3] R. E. Tarjan, “Amortized computational complexity,” *SIAM Journal on Algebraic and Discrete Methods*, vol. 6, no. 2, pp. 306–318, 1985.
- [4] A. Laaksonen, *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, 3rd ed. Cham, Switzerland: Springer, 2024.
- [5] S. Halim, F. Halim, dan S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.

Bab 5

Struktur Data yang Hidup: Pembaruan Titik pada Pohon Segment — GSS3

Pohon segment pada Bab 2 menjawab pertanyaan tentang larik yang tidak pernah berubah. Data nyata berubah: catatan buku kas dikoreksi, harga direvisi, sensor melaporkan bacaan baru. Bab ini mengajari pohon segment untuk hidup bersama perubahan. Anda akan mempelajari *point update* (pembaruan titik), yang memperbaiki pohon di sepanjang satu lintasan akar-ke-daun dalam waktu logaritmik, membuktikan mengapa lintasan itu cukup, dan melihat bagaimana unsur netral sebuah operasi menentukan cara kueri menangani rentang yang tidak beririsan. Yang sama pentingnya, bab ini menjalani satu siklus latihan lengkap sebagaimana berlangsung pada hari lomba: membaca, merancang, mengimplementasikan, memeriksa contoh, melakukan *stress test*, mengumpulkan. Studi kasusnya adalah GSS3, soal Bab 2 dengan pembaruan yang diselipkan di antara pertanyaan.

5.1 Koreksi di Buku Kas

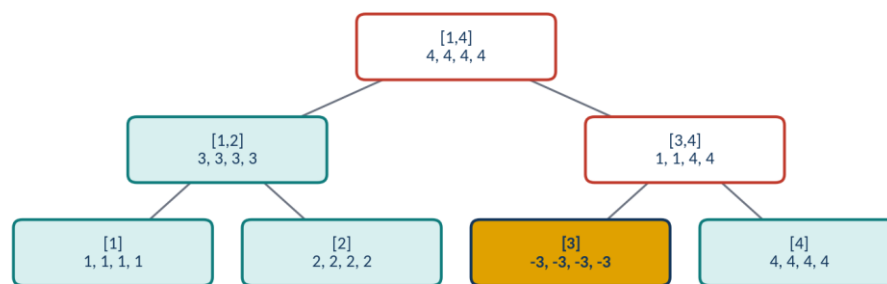
Koperasi buah dari Bab 2 kini memiliki aplikasinya, dan para anggota memakainya setiap hari untuk menanyakan rangkaian hari terbaik dalam periode apa pun. Lalu bendahara menemukan kekeliruan: pendapatan hari ke-1 812 tercatat 3 500 alih-alih -3 500, karena sebuah pengembalian uang tercatat sebagai penjualan. Ia mengoreksi catatan itu. Sejak saat itu, setiap jawaban yang melibatkan hari ke-1 812 harus memakai nilai yang sudah dikoreksi.

Pohon yang dibangun di Bab 2 tidak tahu-menahu tentang koreksi itu. Membangunnya ulang dari awal memakan $O(N)$, dan bila koreksi sesering pertanyaan, N koreksi dan N pertanyaan kembali memakan $O(N^2)$, solusi lambat yang ingin kita hindari. Kita memerlukan pembaruan yang biayanya kira-kira sama dengan kueri: $O(\log N)$. Itulah tepatnya soal SPOJ GSS3 [1]: blok berjumlah terbesar di dalam sebuah rentang, dengan operasi yang mengubah satu elemen di sela-sela pertanyaan.

5.2 Gagasannya dengan Kata-Kata Sederhana

Ketika satu elemen berubah, simpul mana yang menjadi salah? Sebuah simpul menyimpan jawaban untuk rentangnya, sehingga ia menjadi salah hanya bila rentangnya memuat posisi yang berubah. Rentang-rentang yang memuat suatu posisi tepat merupakan simpul-simpul di lintasan dari akar ke daun posisi itu: satu simpul per tingkat. Setiap simpul lain menggambarkan rentang yang tidak disentuh perubahan, sehingga jawaban tersimpannya tetap benar.

Jadi pembaruannya: turun ke daun, ganti daun itu, dan dalam perjalanan kembali ke atas hitung ulang setiap simpul di lintasan dari kedua anaknya, dengan *combine* yang sama yang dipakai saat membangun. Anak-anaknya benar (satu baru saja diperbaiki, yang lain tidak pernah salah), sehingga setiap simpul yang dihitung ulang juga benar. Itulah seluruh gagasannya, dan **Gambar 5.1** memperlihatkan pada empat elemen: sesudah nilai di posisi 3 berubah, hanya daun [3] dan dua simpul merah di atasnya yang dihitung ulang.



sesudah "0 3 -3": daun [3] diganti (kuning); kotak merah digabung ulang saat naik; yang lain tidak disentuh

Gambar 5.1 Pohon untuk 1, 2, -3, 4 sesudah pembaruan "0 3 -3": daun kuning diganti, dua simpul merah di lintasannya digabung ulang, dan simpul lain mempertahankan nilainya.

5.2.1 Betapa sedikit yang berubah

Ada baiknya merasakan betapa kecil perbaikan itu. Untuk $N = 50\,000$ pohon memiliki sekitar 10^5 simpul, dan sebuah pembaruan menghitung ulang 16 di antaranya, kurang dari satu per enam ribu. Simpul-simpul lain menyimpan jawaban yang tetap benar, dan terus dipakai ulang oleh setiap pertanyaan berikutnya. Inilah prinsip umum di balik struktur data dinamis yang efisien: sebuah operasi seharusnya hanya menyentuh bagian informasi tersimpan yang benar-benar dibatalkan oleh perubahan. Struktur yang harus dibangun ulang sesudah setiap perubahan hanyalah struktur statis yang dipakai berulang-ulang.

Prinsip yang sama memberi tahu kapan pohon segmen *tidak* akan berhasil untuk pembaruan. Bila mengubah satu elemen dapat mengubah jawaban tersimpan simpul-simpul yang rentangnya tidak memuat elemen itu, perbaikan lintasan akan salah. Hal itu tidak pernah terjadi selama setiap simpul menyimpan fungsi dari rentangnya sendiri saja, dan itulah persis yang dijamin resep ketertutupan di Bab 2. Jadi simpul yang dirancang dengan resep itu mendukung pembaruan titik secara cuma-cuma: pembaruan di Subbab 5.4 tidak memerlukan gagasan baru, hanya disiplin untuk menggabung ulang dalam perjalanan naik.

5.3 Pemanasan: Jumlah Rentang dengan Pembaruan

Mulailah dengan pohon hidup yang paling sederhana: jumlah rentang dengan penetapan titik. **Kode 5.1** mendukung dua operasi, $S \ x \ y$ (cetak jumlah a_x sampai a_y) dan $U \ x \ v$ (tetapkan a_x menjadi v).

Kode 5.1 `sum_update.cpp`: pohon segmen untuk jumlah rentang dengan penetapan titik.

```
// Warm-up: segment tree for range sums with point assignment.
// "S x y": print a[x] + ... + a[y]; "U x v": set a[x] = v.
#include <cstdio>
#include <vector>
using namespace std;

int n;
vector<long long> t; // t[v] = sum of the range that node v covers

void update(int v, int lo, int hi, int pos, long long val) {
    if (lo == hi) { t[v] = val; return; }
    int mid = (lo + hi) / 2;
    if (pos <= mid) update(2 * v, lo, mid, pos, val);
    else update(2 * v + 1, mid + 1, hi, pos, val);
    t[v] = t[2 * v] + t[2 * v + 1]; // the only line that knows about "sum"
}

long long query(int v, int lo, int hi, int ql, int qr) {
    if (qr < lo || hi < ql) return 0; // no overlap: neutral value
    if (ql <= lo && hi <= qr) return t[v];
    int mid = (lo + hi) / 2;
    return query(2 * v, lo, mid, ql, qr) + query(2 * v + 1, mid + 1, hi, ql, qr);
}

int main() {
    int m;
    if (scanf("%d %d", &n, &m) != 2) return 0;
    t.assign(4 * n, 0);
    for (int i = 1; i <= n; i++) {
        long long v;
        if (scanf("%lld", &v) != 1) return 0;
        update(1, 1, n, i, v); // n updates also build the tree, in O(n log n)
    }
    while (m--) {
        char op;
        int x;
        long long y;
        if (scanf("%c %d %lld", &op, &x, &y) != 3) return 0;
        if (op == 'U') update(1, 1, n, x, y);
        else printf("%lld\n", query(1, 1, n, x, (int)y));
    }
    return 0;
}
```

Pada larik 4, -1, 3, 0, 2, jumlah seluruh larik adalah 8; sesudah a_2 ditetapkan menjadi 10, jumlah tiga elemen pertama 17; elemen keempat saja bernilai 0:

```
5 4
4 -1 3 0 2
S 1 5
U 2 10
S 1 3
S 4 4
```

Mari telusuri kueri **S 1 3** sesudah pembaruan, pada pohon atas posisi 1 sampai 5 (akar meliputi [1,5], anak-anaknya [1,3] dan [4,5]). Akar hanya sebagian beririsan dengan kueri, sehingga bertanya kepada kedua anak. Anak kiri [1,3] sepenuhnya berada di dalam kueri dan mengembalikan jumlah tersimpannya, $4 + 10 + 3 = 17$. Anak kanan [4,5] sama sekali tidak beririsan dengan [1,3] dan mengembalikan 0 yang netral. Akar menjumlahkannya: 17. Dua kunjungan di bawah akar sudah cukup, dan salah satunya tidak memakan biaya apa pun.

Tiga rincian pemanasan ini menyiapkan studi kasus.

Baris terakhir pembaruan adalah satu-satunya baris yang tahu operasinya. $t[v] = t[2 * v] + t[2 * v + 1]$ adalah `combine` untuk jumlah. Ganti dengan `max`, dan pohon yang sama menjawab maksimum rentang.

Membangun dengan pembaruan. Alih-alih `build` tersendiri, program menyisipkan N nilai awal dengan N pembaruan. Biayanya $O(N \log N)$ alih-alih $O(N)$; untuk $N = 10^5$ selisihnya beberapa milidetik, dan satu fungsi yang bisa mengandung *bug* hilang. Dalam lomba, lebih sedikit fungsi sering kali merupakan tukar-guling yang lebih baik.

Unsur netral dalam kueri. Kueri ini memiliki kasus keempat yang tidak dibutuhkan kueri Bab 2: bila rentang simpul sama sekali tidak beririsan dengan kueri, ia mengembalikan 0. Cara itu berhasil karena 0 adalah *unsur netral* penjumlahan: menambahkannya tidak mengubah apa pun. Secara formal, unsur e netral untuk operasi $\oplus$ bila

$$e \oplus x = x \oplus e = x \quad \text{untuk setiap } x \quad (5.1)$$

Untuk jumlah $e = 0$, untuk minimum $e = +\infty$, untuk maksimum $e = -\infty$, untuk hasil kali $e = 1$. Dengan unsur netral, kueri cukup bertanya kepada kedua anak setiap saat dan membiarkan sisi yang tidak beririsan mengembalikan e . Tanpanya, seperti pada simpul empat-medan GSS1 dan GSS3 (Subbab 2.6.5), kueri tidak boleh mengunjungi simpul yang tidak beririsan, dan itulah sebabnya ia memakai bentuk tiga kasus.

5.4 Pembaruan Titik

5.4.1 Algoritmanya

`update(v, lo, hi, pos, val)` menetapkan posisi `pos` menjadi `val` di dalam rentang simpul v :

- Bila v daun ($lo = hi$), ganti simpulnya dengan `makeNode(val)`.
- Selainnya turun ke anak yang rentangnya memuat `pos` (kiri bila $pos \leq mid$, kanan bila tidak), dan sesudah panggilan rekursif kembali, hitung ulang v sebagai `combine` dari kedua anaknya.

5.4.2 Mengapa benar

Pohon benar bila setiap daun menggambarkan elemennya dan setiap simpul dalam sama dengan `combine` dari anak-anaknya; sebut ini *invarian pohon*. Sebelum pembaruan invarian itu berlaku. Pembaruan mengubah daun `pos`, yang kemudian benar untuk nilai barunya. Dalam perjalanan naik, setiap simpul di lintasan dihitung ulang dari anak-anaknya *sesudah* anaknya di lintasan diperbaiki, sehingga memenuhi invarian lagi. Simpul yang tidak berada di lintasan meliputi rentang tanpa `pos`, sehingga baik anak-anaknya maupun nilainya sendiri tidak berubah, dan invarian tetap berlaku baginya. Jadi invarian berlaku di mana-mana sesudah pembaruan, dan setiap kueri berikutnya benar menurut argumen Bab 2.

Urutannya penting: penghitungan ulang harus terjadi *sesudah* panggilan rekursif. Menghitung ulang dalam perjalanan turun, sebelum anaknya diperbaiki, menyalin nilai lama ke atas dan membiarkan pohon salah secara senyap.

5.4.3 Biaya

Lintasan dari akar ke daun memiliki satu simpul per tingkat, dan kedalaman pohon adalah $\lceil \log_2 N \rceil$, sehingga sebuah pembaruan melakukan

$$\text{combine per pembaruan} = \lceil \log_2 N \rceil \quad (5.2)$$

Untuk $N = 50\,000$ itu berarti 16 panggilan `combine`. Barisan M operasi, pembaruan dan kueri bercampur, memakan $O(N + M \log N)$ termasuk pembangunan awal.

5.4.4 Alternatif dari bawah ke atas

Pembaruan rekursif adalah bentuk yang paling jelas, tetapi banyak peserta berpengalaman juga mengenal pohon segmen *bottom-up* (dari bawah ke atas), yang sama sekali tanpa rekursi. Simpan N daun di indeks N sampai $2N - 1$ dari larik berukuran $2N$, dan biarkan simpul v (untuk $1 \leq v < N$) menjadi gabungan simpul $2v$ dan $2v + 1$. Pembaruan menulis daun di indeks $N + pos$ lalu naik dengan $v \leftarrow \lfloor v/2 \rfloor$, menggabung ulang setiap simpul, sampai mencapai akar. Kueri menjalankan dua indeks, satu dari setiap ujung rentang, saling mendekat, sambil mengumpulkan simpul-simpul yang sepenuhnya berada di dalam. Pohon *bottom-up* hanya memakai $2N$ simpul dan kira-kira dua kali lebih cepat, tetapi kuerinya mengumpulkan bagian kiri dan kanan dalam dua akumulator terpisah, dan untuk `combine` yang tidak komutatif seperti milik GSS, akumulator kanan harus digabung dengan urutan terbalik. Rincian itu mudah keliru di bawah tekanan, dan itulah sebabnya buku ini memakai bentuk rekursif untuk studi kasus. Pelajari bentuk *bottom-up* setelah Anda nyaman dengan bentuk rekursif, dan ujilah dengan skrip *stress test* yang sama.

5.5 Siklus Latihan yang Meniru Lomba

Soal struktur data adalah tempat rutinitas kerja yang rapi paling berbuah, karena *bug*-nya senyap: program berjalan, mencetak angka yang masuk akal, dan salah pada satu dari lima puluh kasus uji. **Tabel 5.1** adalah rutinitas latihan dua jam untuk soal seperti GSS3. Jalankan dengan pengatur waktu, sendiri atau bersama teman, dan catat berapa lama setiap tahap sesungguhnya berlangsung; sesudah beberapa minggu catatan itu memperlihatkan ke mana waktu Anda pergi.

Tabel 5.1 Rutinitas latihan dua jam dengan irama hari lomba.

Menit	Kegiatan	Sasaran
0–10	Baca soal; tulis model dan simpulnya di kertas	Model satu kalimat dan rumus <code>combine</code>
10–45	Implementasikan solusi penuh	Kompilasi bersih dan contoh soal lolos
45–70	<i>Stress test</i> melawan <i>brute force</i> $O(N^2)$, perbaiki <i>bug</i>	Ratusan kasus acak identik
70–85	Kumpulkan ke <i>judge</i>	Vonis <i>Accepted</i>
85–120	Pelajari varian (penetapan rentang, <i>lazy propagation</i>)	Rencana untuk soal berikutnya yang lebih sulit

Dua aturan membuat rutinitas ini jujur. Jangan membuka solusi jadi sebelum menit ke-70: pergulatan menit 10 sampai 70 itulah tempat belajar terjadi. Dan izinkan diri Anda paling banyak tiga petunjuk, dari teman atau pembina, kira-kira sebanyak bantuan yang didapat peserta dari permintaan klarifikasi dalam lomba sungguhan.

5.5.1 Apa yang dicatat selama berlatih

Catatan yang Anda buat selama rutinitas ini lebih berharga daripada vonis *Accepted*. Untuk setiap tahap, catat tiga hal: berapa menit yang dihabiskan, satu kesalahan yang paling banyak memakan waktu, dan apa yang akan menangkap kesalahan itu lebih awal. Sesudah beberapa soal, pola-pola muncul. Sebagian peserta kehilangan sebagian besar waktunya pada menit 10 sampai 45 karena mulai mengetik sebelum simpulnya dirancang; obatnya adalah bersikeras menulis rumus `combine` di kertas lebih dulu. Yang lain cepat meloloskan contoh soal lalu kehilangan setengah jam karena

bug yang akan ditemukan *stress test* dalam semenit; obatnya adalah menulis *brute force* dan pembangkit *sebelum* solusi cepat, selagi soalnya masih segar. Yang lain lagi terhambat oleh kompilator: mereka tidak tahu cara membaca pesan galat *template*, atau lupa sebuah *flag*. Masing-masing adalah kelemahan berbeda dengan obat berbeda, dan hanya catatan yang jujur yang mengungkap kelemahan Anda.

Simpan catatan itu di buku catatan kesalahan dari Bab 1. Bacalah sebelum setiap sesi latihan, dan pilih satu kebiasaan untuk diperbaiki hari itu. Kemajuan dalam pemrograman kompetitif datang bukan terutama dari menyelesaikan lebih banyak soal, melainkan dari semakin jarang mengulang kesalahan yang sama.

5.6 Studi Kasus SULIT: GSS3

5.6.1 Soal dengan kata-kata kita sendiri

Diberikan $N \leq 50\,000$ bilangan bulat $a_1, \dots, a_N$ dengan nilai mutlak paling besar 10 000, lalu $M \leq 50\,000$ operasi, masing-masing sebaris tiga bilangan bulat. Operasi $0 \ x \ y$ menetapkan a_x menjadi y . Operasi $1 \ x \ y$ menanyakan jumlah terbesar blok tak kosong $a_i + \dots + a_j$ dengan $x \leq i \leq j \leq y$, yang harus dicetak pada barisnya sendiri. Masukan memberikan N , lariknya, M , dan operasi-operasinya.

5.6.2 Contoh masukan dan keluaran

```
4
1 2 3 4
4
1 1 3
0 3 -3
1 2 4
1 3 3
```

Pada 1, 2, 3, 4, blok terbaik di dalam [1,3] adalah seluruh rentang: 6. Lalu a_3 menjadi -3 . Di dalam [2,4] nilainya 2, -3 , 4, dan blok terbaiknya 4 saja ($2 - 3 + 4$ hanya 3). Akhirnya [3,3] hanya memuat -3 , dan karena bloknya tidak boleh kosong, jawabannya -3 .

5.6.3 Rancangan

Simpul, `makeNode`, `combine`, `build`, dan `query` persis sama dengan milik GSS1 yang diturunkan di Subbab 2.5. Satu-satunya fungsi baru adalah `update` dari Subbab 5.4. Operasi 0 memanggil `update`; operasi 1 memanggil `query` dan mencetak medan `ans`.

Tipe apa yang sebaiknya dipakai medan-medannya? Nilai mutlak terbesar yang mungkin dari sebuah jumlah adalah

$$N \cdot \max|a_i| \leq 50\,000 \times 10\,000 = 5 \times 10^8 < 2^{31} - 1 \approx 2.15 \times 10^9 \quad (5.3)$$

dan `suf + pre` di dalam `combine` menjumlahkan dua nilai semacam itu, paling besar 10^9 , masih di bawah batas `int`. Jadi `int` aman di sini, menurut [Persamaan \(5.3\)](#), dan solusinya memakainya. Melakukan pemeriksaan ini secara eksplisit, bukan menebak, adalah kebiasaan yang penting: nilai GSS1 lebih besar, dan soal dengan nilai hingga 10^9 akan memerlukan `long long`.

5.6.4 Solusi lengkap

Kode 5.2 adalah program lengkapnya. Dibandingkan dengan solusi GSS1 di Bab 2 (Kode 2.3), program ini menambahkan **update** dan kode operasi, serta memindahkan **query** ke depan penanda, sehingga bagian yang diserahkan kepada pembaca dalam edisi publik adalah **combine** bersama **update**.

Kode 5.2 GSS3.cpp: pohon GSS1 dengan pembaruan titik; setiap operasi memakan $O(\log N)$.

```
// GSS3 (SPOJ): GSS1 plus point updates. "0 x y": a[x] = y;
// "1 x y": best non-empty subarray sum inside [x, y]. C++17.
#include <algorithm>
#include <cstdio>
using namespace std;

const int MAXN = 50001;
struct Node {
    int sum, pre, suf, ans; // |sum| <= 50000 * 10000, so int is enough
};
Node tree[4 * MAXN];
int a[MAXN], N;

Node makeNode(int v) {
    Node n;
    n.sum = n.pre = n.suf = n.ans = v;
    return n;
}

Node combine(const Node& l, const Node& r); // defined below

void build(int v, int lo, int hi) {
    if (lo == hi) {
        tree[v] = makeNode(a[lo]);
        return;
    }
    int mid = (lo + hi) / 2;
    build(2 * v, lo, mid);
    build(2 * v + 1, mid + 1, hi);
    tree[v] = combine(tree[2 * v], tree[2 * v + 1]);
}

Node query(int v, int lo, int hi, int ql, int qr) {
    if (ql <= lo && hi <= qr) return tree[v];
    int mid = (lo + hi) / 2;
    if (qr <= mid) return query(2 * v, lo, mid, ql, qr);
    if (ql > mid) return query(2 * v + 1, mid + 1, hi, ql, qr);
    return combine(query(2 * v, lo, mid, ql, qr),
                  query(2 * v + 1, mid + 1, hi, ql, qr));
}
```

Kode 5.2 (Lanjutan) *GSS3.cpp: pohon GSS1 dengan pembaruan titik; setiap operasi memakan $O(\log N)$.*

```
// <P50>
// combine: the four GSS1 formulas (sum, pre, suf, ans) for two neighbors.
// update: walk down to the leaf of pos, replace it with makeNode(val), and
// recombine every node on the way back up so all ancestors are fresh again.
// >>> YOUR TURN (Public Edition): write this part yourself, following the hint above. <<<
// </P50>

int main() {
    if (scanf("%d", &N) != 1) return 0;
    for (int i = 1; i <= N; i++)
        if (scanf("%d", &a[i]) != 1) return 0;
    build(1, 1, N);
    int M;
    if (scanf("%d", &M) != 1) return 0;
    while (M--) {
        int op, x, y;
        if (scanf("%d %d %d", &op, &x, &y) != 3) return 0;
        if (op == 0) update(1, 1, N, x, y);
        else printf("%d\n", query(1, 1, N, x, y).ans);
    }
    return 0;
}
```

5.6.5 Baris demi baris

Simpul dan daun. `Node` memiliki empat medan `int` yang dibenarkan oleh [Persamaan \(5.3\)](#). `makeNode(v)` menetapkan keempatnya menjadi `v`, karena satu-satunya blok tak kosong dari satu elemen adalah elemen itu sendiri.

build dan query. Keduanya tidak berubah dari Bab 2. `build` mengisi pohon sekali dalam $O(N)$; `query` memakai bentuk tiga kasus, tidak pernah mengunjungi simpul yang tidak beririsan dengan kueri, karena simpul empat-medan tidak memiliki unsur netral.

combine. Di antara penanda `// <P50>` dan `// </P50>`, `combine` menyatakan ulang keempat rumus (2.5) sampai (2.8) dari Bab 2: jumlah dijumlahkan, prefiks terbaik tetap di bagian kiri atau mengambil seluruhnya lalu berlanjut, sufiks terbaik adalah bayangan cerminnya, dan blok terbaik berada di dalam kiri, di dalam kanan, atau melintas.

update. Juga di dalam penanda, `update` adalah Subbab 5.4.1 dalam bentuk kode. Di daun ia menyimpan `makeNode(val)`. Selainnya ia berekursi ke satu anak lalu, *sesudah* panggilan itu, menghitung ulang `tree[v]` dari kedua anak. Kedalaman rekursinya paling banyak 17 untuk $N = 50\,000$.

main. Fungsi ini membaca N dan lariknya, membangun pohon, membaca M , lalu memproses setiap operasi: `op == 0` memperbarui, selainnya bertanya dan mencetak. Setiap hasil `scanf` diperiksa.

5.6.6 Jebakan

- **Lupa menggabung ulang.** Pembaruan yang mengganti daun tetapi tidak menghitung ulang leluhurnya lolos contoh soal hanya bila tidak ada pertanyaan sesudah pembaruan yang menyentuh posisi yang berubah. *Stress test* menemukannya dalam beberapa putaran pertama.
- **Menggabung ulang sebelum rekursi.** Seperti dijelaskan di Subbab 5.4.2, cara ini menyalin nilai basi ke atas.
- **Memperbarui `a[x]` tetapi tidak memperbarui pohon.** Larik `a` hanya dipakai oleh `build`; sesudah itu, pohonlah yang menjadi kebenaran. Mengubah `a[x]` saja tidak berbuat apa-apa.

- **Tertukar kode operasi.** Pada GSS3, 0 berarti pembaruan dan 1 berarti kueri. Bacalah pernyataan soal dua kali.
- **Memotong di nol.** Seperti pada GSS1, `makeNode` harus menerima nilai negatif, dan tidak satu medan pun boleh dipotong dengan `max(0, ...)`.

5.6.7 Stress test

Kode 5.3 menyimpan lariknya sendiri, menerapkan pembaruan secara langsung, dan menjawab pertanyaan dengan mencoba setiap blok.

Kode 5.3 `brute.cpp`: pembaruan pada larik biasa dan setiap blok dari setiap rentang.

```
// Brute force for GSS3: change the array directly, scan every subarray.
#include <algorithm>
#include <climits>
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, m;
    cin >> n;
    vector<long long> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];
    cin >> m;
    while (m--) {
        int op, x, y;
        cin >> op >> x >> y;
        if (op == 0) { a[x] = y; continue; }
        long long best = LLONG_MIN;
        for (int i = x; i <= y; i++) {
            long long s = 0;
            for (int j = i; j <= y; j++) best = max(best, s += a[j]);
        }
        cout << best << "\n";
    }
    return 0;
}
```

Pembangkit **Kode 5.4** mencampur pembaruan dan pertanyaan, sekitar 40 persen pembaruan, sehingga sebagian besar pertanyaan datang sesudah paling sedikit satu perubahan.

Kode 5.4 `gen.py`: masukan GSS3 acak yang mencampur pembaruan dan pertanyaan.

```
#!/usr/bin/env python3
"""Random GSS3 input: python3 gen.py SEED"""
import random
import sys

rng = random.Random(int(sys.argv[1]) if len(sys.argv) > 1 else 1)
n = rng.randint(1, 12)
lim = rng.choice([3, 10, 10000])
print(n)
print(*[rng.randint(-lim, lim) for _ in range(n)])
m = rng.randint(1, 15)
print(m)
for _ in range(m):
    if rng.random() < 0.4:
        print(0, rng.randint(1, n), rng.randint(-lim, lim))
    else:
        x = rng.randint(1, n)
        print(1, x, rng.randint(x, n))
```

```
all 500 tests identical
```

Pada masukan terbesar, $N = M = 50\,000$ dengan separuh pembaruan dan separuh pertanyaan, program berjalan sekitar 0,04 detik di mesin kami, sebagaimana diramalkan [Persamaan \(5.2\)](#).

5.6.8 Bug yang tertangkap basah

Untuk melihat apa yang dibeli *stress test*, bayangkan solusi yang *update*-nya mengganti daun dengan benar tetapi melupakan baris `tree[v] = combine(tree[2 * v], tree[2 * v + 1]);`. Contoh resmi menangkap yang satu itu: sesudah `0 3 -3`, akar masih menggambarkan 1, 2, 3, 4, dan pertanyaan [2,4] dijawab dari simpul [3,4] yang basi, sehingga tercetak 7 alih-alih 4. Kesalahan yang lebih halus adalah menggabung ulang hanya ketika pembaruan turun ke *kiri*. Apakah contoh soal menangkapnya bergantung pada letak satu-satunya pembaruan di contoh itu, dan menalar contoh mana yang diloloskan sebuah *bug* adalah pemborosan waktu lomba. Jalankan saja *stress test*. Kami menanam *bug* persis seperti ini di salinan **Kode 5.2** lalu menjalankan skripnya: skrip berhenti pada *seed* pertama dan mencetak masukan dengan larik tiga bilangan dan delapan operasi. Menyusutkan masukan itu dengan tangan menghasilkan sesuatu sekecil larik 1, 2 dengan pembaruan `0 2 5` dan pertanyaan `1 1 2`, yang jawaban benarnya 6 sementara program yang salah mencetak 3.

Pelajarannya berlaku umum. Contoh soal dipilih pembuat soal untuk menjelaskan pernyataannya, bukan untuk menangkap *bug* khusus Anda. Pembangkit menjelajahi masukan yang tidak terpikirkan oleh Anda, dan *brute force* menilainya, sehingga yang tersisa bagi Anda hanyalah membaca satu kasus gagal yang pendek.

5.7 Melampaui Pembaruan Titik

GSS3 mengubah satu elemen setiap kali. Soal yang lebih sulit mengubah seluruh rentang sekaligus: “tambahkan 5 ke setiap elemen $[l, r]$ ” atau “tetapkan setiap elemen $[l, r]$ menjadi c ”. Mengerjakannya dengan pembaruan titik memakan $O(N \log N)$ per operasi, terlalu lambat. Obat bakunya adalah *lazy propagation* (propagasi malas): simpul yang terliput penuh oleh pembaruan mencatat perubahan yang tertunda (sebuah *tag*) dan memperbarui jawabannya sendiri, tanpa mengunjungi anak-anaknya; *tag* itu baru didorong ke anak-anak ketika operasi berikutnya perlu turun ke bawah simpul itu. Untuk penetapan rentang bernilai c pada simpul yang meliputi ℓ elemen, keempat medan GSS menjadi

$$\text{sum} = c \ell, \quad \text{pre} = \text{suf} = \text{ans} = \max(c, c \ell) \quad (5.4)$$

karena blok tak kosong terbaik dari ℓ nilai sama c adalah seluruh rentang bila $c > 0$ dan satu elemen bila tidak. [Persamaan \(5.4\)](#) adalah rumus kunci solusi semacam itu, dan dengannya setiap operasi tetap $O(\log N)$.

Contoh kecil memperlihatkan penghematannya. Pada delapan elemen, penetapan “tetapkan [1,8] menjadi 2” hanya menyentuh akar: akar menyimpan *tag* “semua 2” dan, menurut [Persamaan \(5.4\)](#) dengan $\ell = 8$, simpul (16,16,16,16). Pertanyaan berikutnya tentang [3,4] harus turun ke bawah akar, sehingga akar lebih dulu *mendorong tag*-nya ke kedua anak (masing-masing menjadi (8,8,8,8) dengan *tag* “semua 2”) lalu melupakannya; dorongan berlanjut hanya di sepanjang lintasan yang benar-benar diambil pertanyaan itu. Simpul yang tidak pernah dikunjungi pertanyaan mana pun menyimpan *tag*-nya selamanya, dan tidak ada yang membayar untuknya.

Lazy propagation adalah langkah alami sesudah bab ini; Laaksonen [2] dan Halim dkk. [3] mengembangkannya dengan cermat. Untuk kasus khusus jumlah prefiks dengan pembaruan titik, pohon Fenwick [4] lebih pendek daripada pohon segment mana pun dan juga layak dikenal.

Soal-soal GSS di SPOJ membentuk tangga yang terkenal: GSS1 (statis, Bab 2), GSS3 (pembaruan titik, bab ini), dan soal-soal sesudahnya yang menambahkan pembaruan rentang, akar kuadrat elemen, atau syarat bahwa setiap nilai hanya boleh dihitung sekali. Menaiki tangga itu adalah latihan yang sangat baik untuk babak nasional, dan simpul empat-medan yang kini Anda kenal muncul kembali di hampir setiap anak tangganya.

5.8 Taktik Hari Lomba

- **Pakai ulang pohon statis.** Bila sebuah soal menambahkan pembaruan pada soal rentang statis, pertahankan `makeNode`, `combine`, dan `query`, lalu tambahkan `update` enam baris.
- **Periksa batas numerik secara eksplisit** dengan perhitungan seperti [Persamaan \(5.3\)](#) sebelum memilih `int` atau `long long`.
- **Bangkitkan pembaruan dalam *stress test* Anda.** Pembangkit tanpa pembaruan hanya menguji bagian statis.
- **Tentukan unsur netralnya.** Bila operasi Anda memilikinya ([Persamaan \(5.1\)](#)), kueri empat kasus paling sederhana; bila tidak, pakai kueri tiga kasus.
- **Ikuti siklus latihan** pada [Tabel 5.1](#) saat berlatih, agar iramanya otomatis pada hari lomba.

5.9 Ringkasan

- Mengubah satu elemen hanya membatalkan simpul-simpul di lintasan akar-ke-daunnya, sehingga pembaruan titik menghitung ulang $\lceil \log_2 N \rceil$ simpul ([Persamaan \(5.2\)](#)) dengan `combine` yang sama yang dipakai untuk membangun.
- Kebenarannya bertumpu pada invarian pohon (setiap simpul dalam sama dengan `combine` dari anak-anaknya), yang dipulihkan pembaruan dari bawah ke atas.
- Unsur netral ([Persamaan \(5.1\)](#)) memungkinkan kueri mengunjungi simpul yang tidak beririsan; simpul GSS tidak memilikinya, sehingga kuerinya memakai tiga kasus.
- GSS3 adalah GSS1 ditambah pembaruan titik; nilainya muat dalam `int` menurut [Persamaan \(5.3\)](#).
- Pembaruan rentang memerlukan *lazy propagation*, dengan rumus seperti [Persamaan \(5.4\)](#) untuk setiap jenis *tag*.

5.10 Soal Latihan

1. Pada pohon [Gambar 5.1](#), simpul mana yang dihitung ulang oleh pembaruan “0 1 7”? Berikan nilai empat-medan barunya.
2. Apa yang salah bila `update` menghitung ulang `tree[v]` *sebelum* panggilan rekursif? Berikan contoh dua elemen.
3. Tuliskan `combine` dan unsur netral untuk pohon yang menjawab minimum rentang beserta banyaknya posisi yang mencapai minimum itu.
4. Nilai-nilai sebuah soal mirip GSS3 dapat mencapai 10^6 secara mutlak dengan $N \leq 10^5$. Apakah `int` masih aman? Pakai perhitungan seperti [Persamaan \(5.3\)](#).

5. Solusi GSS3 seorang teman membangun ulang seluruh pohon sesudah setiap pembaruan. Untuk $N = M = 50\,000$, perkirakan banyaknya panggilan `combine` dan putuskan apakah lolos batas waktu satu detik.
6. Dengan [Persamaan \(5.4\)](#), berikan simpul untuk rentang tiga elemen yang semuanya ditetapkan -2 , dan untuk rentang tiga elemen yang semuanya ditetapkan 5 .
7. Pada pohon segmen *bottom-up* (Subbab 5.4.4), kueri untuk GSS3 mengumpulkan simpul dari ujung kiri dan dari ujung kanan rentang. Dengan urutan apa simpul-simpul yang terkumpul harus digabung, dan mengapa?

5.11 Penyelesaian Soal Latihan

1. Lintasannya daun $[1]$, simpul $[1,2]$, dan akar. Daun menjadi $(7,7,7,7)$. Simpul $[1,2]$ menggabungkan $(7,7,7,7)$ dengan $[2] = (2,2,2,2)$: $(9,9,9,9)$. Akar menggabungkan $(9,9,9,9)$ dengan $[3,4] = (1,1,4,4)$: $sum = 10$, $pre = \max(9, 9 + 1) = 10$, $suf = \max(4, 1 + 9) = 10$, $ans = \max(9, 4, 9 + 1) = 10$.
2. Simpul dihitung ulang dari anak-anak lamanya, sehingga mempertahankan jawaban lama, dan perbaikan pada anaknya tidak pernah dirambatkan. Untuk larik $1, 2$ dan pembaruan $"0\ 1\ 5"$, akar tetap $(3,3,3,3)$ padahal jawaban sebenarnya untuk $[1,2]$ kini 7 .
3. Simpulnya (minimum, cacah). `combine`: bila minimum kiri lebih kecil, ambil simpul kiri; bila minimum kanan lebih kecil, ambil simpul kanan; bila sama, pertahankan minimumnya dan jumlahkan cacahnya. Unsur netralnya $(+\infty, 0)$.
4. $10^5 \times 10^6 = 10^{11}$, jauh di atas 2.15×10^9 : `int` meluap, dan semua medan harus `long long`.
5. Pembangunan ulang memakan sekitar N `combine`; dengan sekitar $25\,000$ pembaruan itu berarti 1.25×10^9 `combine` empat-medan, beberapa detik paling cepat. Tidak lolos; pembaruan lintasan hanya memakan $25\,000 \times 16 = 4 \times 10^5$ `combine`.
6. Untuk $c = -2$ dan $\ell = 3$: $sum = -6$ dan $pre = suf = ans = \max(-2, -6) = -2$. Untuk $c = 5$: $sum = 15$ dan ketiga medan lainnya $\max(5, 15) = 15$.
7. Simpul-simpul yang dikumpulkan dari ujung kiri ditemui dalam urutan kiri-ke-kanan, sehingga digabung sebagai `leftAcc = combine(leftAcc, node)`. Simpul-simpul dari ujung kanan ditemui dalam urutan kanan-ke-kiri, sehingga harus digabung sebagai `rightAcc = combine(node, rightAcc)`, dan jawabannya `combine(leftAcc, rightAcc)`. Urutan itu penting karena `combine` tidak komutatif: pre dan suf menggambarkan tepi kiri dan kanan, dan menukar argumen berarti menukar kedua tepi itu.

Daftar Pustaka

- [1] Sphere Online Judge, "GSS3 — Can you answer these queries III." [Daring]. Tersedia: <https://www.spoj.com/problems/GSS3/> (diakses Sep. 2026).
- [2] A. Laaksonen, *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, 3rd ed. Cham, Switzerland: Springer, 2024.
- [3] S. Halim, F. Halim, dan S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.
- [4] P. M. Fenwick, "A new data structure for cumulative frequency tables," *Software: Practice and Experience*, vol. 24, no. 3, pp. 327–336, 1994.

Bab 6

Graf Ruang Keadaan dan Algoritma Dial — CHIPSLL

Banyak soal lintasan terpendek di olimpiade tingkat nasional tidak pernah menyebut kata “graf”. Soalnya berbicara tentang labirin, permainan papan, robot dengan baterai, dua ember air, atau seorang pahlawan yang menerobos peta penuh monster. Keterampilan pertama yang dibangun bab ini adalah *melihat* graf yang tersembunyi: memutuskan apa yang menjadi simpul, apa yang menjadi sisi, dan berapa biaya setiap sisi. Keterampilan kedua adalah memilih alat yang tepat setelah graf itu terlihat. *Breadth-first search* (penelusuran melebar) biasa menangani sisi berbobot 1, antrean dua ujung menangani bobot 0 dan 1, sebuah cincin ember (algoritma Dial) menangani bobot bilangan bulat kecil, dan barulah sesudah itu algoritma Dijkstra dengan *heap* layak dipakai. Di akhir bab ini Anda akan mampu memodelkan teka-teki sebagai ruang keadaan, memilih algoritma benar yang tercepat dari bentuk bobot sisinya, dan mengimplementasikan algoritma Dial pada kisi berisi empat juta sel, persis seperti yang dituntut soal SPOJ CHIPSLL [1].

6.1 Seorang Pahlawan, Sebuah Peta, dan Stopwatch

Bayangkan sebuah gim video sederhana. Petanya berupa persegi panjang yang terbagi atas petak-petak. Pahlawan kita, Chip, berdiri di satu petak; temannya, Melinda, menunggu di petak lain. Sebagian petak kosong. Petak lain berisi genangan air, monster yang sedang tidur, atau pintu terkunci, dan setiap rintangan itu memiliki *kekuatan*, yaitu bilangan bulat yang menyatakan berapa detik tambahan diperlukan untuk menyingkirkannya. Masuk ke petak kosong memakan waktu satu detik. Masuk ke petak berintangian memakan waktu satu detik ditambah kekuatan rintangan itu. Chip boleh bergerak ke atas, ke bawah, ke kiri, atau ke kanan. Seberapa cepat ia bisa sampai kepada Melinda?

Seorang anak yang memainkan gim itu akan mencari rute yang menghindari rintangan terkuat, dan dengan senang hati berjalan beberapa petak lebih jauh untuk memutar monster besar. Naluri itu benar, dan itulah yang diformalkan oleh algoritma lintasan terpendek. Kesulitannya ada pada ukuran. Pada CHIPSLL peta bisa berukuran 2000 kali 2000 petak, sehingga ada empat juta petak dan sekitar enam belas juta langkah tunggal yang mungkin, dan satu berkas uji bisa memuat sepuluh peta seperti itu. Batas waktunya tidak memberi ruang bagi algoritma yang ceroboh, dan batas memorinya tidak memberi ruang bagi struktur data yang ceroboh.

Bentuk pertanyaan yang sama muncul di mana-mana di luar gim. Kurir bersepeda motor yang menyeberangi kota ketika setiap ruas jalan memiliki lama kemacetan yang berbeda; petani yang memindahkan pipa air melintasi ladang yang sebagian petaknya lebih sulit digali; robot penyedot debu yang membayar sedikit energi setiap kali memanjat tepi karpet. Dalam setiap kasus, “biaya” satu langkah adalah bilangan bulat kecil, dan detail kecil itulah yang dimanfaatkan bab ini.

6.2 Melihat Graf yang Tersembunyi

6.2.1 Keadaan, aksi, dan biaya

Graf adalah himpunan simpul yang dihubungkan oleh sisi. Graf *berbobot* juga memberi setiap sisi sebuah bilangan, yaitu *bobot* atau *biaya*-nya. *Lintasan terpendek* dari simpul s ke simpul t adalah barisan sisi dari s menuju t yang total bobotnya sekecil mungkin; total itu disebut *jarak* $d(s, t)$.

DEFINISI. *Ruang keadaan (state space)* adalah graf yang simpul-simpulnya adalah situasi (keadaan) yang mungkin dari suatu teka-teki atau proses, dan sisi-sisinya adalah aksi sah yang mengubah satu keadaan menjadi keadaan lain. Bobot sebuah sisi adalah harga aksi itu: waktu, jumlah langkah, energi, atau uang.

Dalam labirin kisi, keadaan hanyalah petak tempat kita berdiri. Dalam teka-teki dengan kunci dan pintu, keadaan juga harus mengingat kunci mana yang sedang dipegang, sehingga satu petak menjadi beberapa keadaan. Dalam teka-teki klasik dua ember, keadaan adalah pasangan (air di ember 1, air di ember 2), dan sama sekali tidak ada “peta”. Disiplin yang memenangkan lomba adalah menuliskan empat jawaban di kertas sebelum menulis kode apa pun:

1. Apa itu keadaan? (Informasi apa yang harus saya ingat untuk mengetahui apa yang bisa terjadi berikutnya?)
2. Ada berapa keadaan? (Kalikan rentang dari tiap bagiannya.)
3. Aksi apa saja dari suatu keadaan? (Inilah sisi-sisinya.)
4. Berapa biaya setiap aksi? (Inilah bobotnya, dan *bentuk*-nya menentukan algoritma.)

Untuk CHIPSLL jawabannya singkat. Keadaan adalah petak (r, c) , sehingga ada paling banyak $2000 \cdot 2000 = 4\,000\,000$ keadaan. Dari setiap petak ada paling banyak empat aksi. Biaya melangkah masuk ke petak v adalah

$$w(v) = 1 + \text{power}(v), \quad 0 \leq \text{power}(v) \leq 62 \quad (6.1)$$

sehingga setiap bobot bernilai antara 1 dan 63. Perhatikan dari [Persamaan \(6.1\)](#) bahwa bobot hanya bergantung pada petak yang *dimasuki*, bukan petak yang ditinggalkan. Hal ini khas pada soal kisi, dan memungkinkan kita menyimpan satu bilangan kecil per petak alih-alih satu per sisi.

6.2.2 Tangga senjata

Setelah ruang keadaan jelas, bobotlah yang menentukan algoritma. **Tabel 6.1** mendaftarkan pilihan-pilihannya dari yang paling sederhana hingga yang paling umum. Setiap baris dapat menyelesaikan semua soal pada baris di atasnya, tetapi lebih lambat; peserta yang selalu meraih baris terakhir membuang waktu yang sengaja disisakan juri untuk gagasan yang lebih baik.

Tabel 6.1 Tangga lintasan terpendek: bentuk bobot menentukan algoritma; D adalah jarak terbesar yang ditemukan.

Bobot pada sisi	Algoritma	Waktu jalan
semuanya 1	<i>breadth-first search</i> (BFS) dengan antrean	$O(V + E)$
hanya 0 atau 1	BFS 0-1 dengan antrean dua ujung	$O(V + E)$
bilangan bulat kecil $1..C$	algoritma Dial, $C + 1$ ember melingkar	$O(V + E + D)$
sembarang tak negatif	algoritma Dijkstra dengan <i>binary heap</i>	$O(E \log V)$
ada yang negatif	Bellman–Ford (atau SPFA dalam praktik)	$O(VE)$

Di sini V adalah banyaknya simpul dan E banyaknya sisi. Untuk CHIPSLL, $V = 4 \cdot 10^6$ dan $E \approx 1.6 \cdot 10^7$ per peta. Sebuah *heap* melakukan sekitar $\log_2 V \approx 22$ langkah dasar per operasi, dengan lokalitas memori yang buruk; ember hanya melakukan sejumlah konstan langkah. Selisih itu menentukan antara *Accepted* dan *Time Limit Exceeded*. Buku pegangan *competitive programming* yang baku mengajarkan tangga yang sama dan layak dibaca bersama bab ini [2], [3].

6.2.3 Mengapa Dijkstra benar, dalam satu paragraf

Algoritma Dijkstra [4], [5] menyimpan jarak sementara $d[v]$ untuk setiap simpul, awalnya ∞ kecuali $d[s] = 0$. Algoritma ini berulang kali mengambil simpul belum selesai u yang jarak semmentaranya terkecil, menyatakannya selesai, lalu melakukan *relaksasi* pada sisi-sisinya:

$$d[v] \leftarrow \min(d[v], d[u] + w(u, v)) \quad (6.2)$$

Mengapa jarak sementara terkecil sudah pasti final? Misalkan ada rute lain yang mencapai u lebih murah. Rute itu meninggalkan daerah yang sudah selesai di suatu tempat, pada simpul belum selesai pertama x ; bagian rute hingga x sudah berbiaya paling sedikit $d[x] \geq d[u]$, dan sisa sisinya berbobot tak negatif, sehingga rute itu berbiaya paling sedikit $d[u]$. Kontradiksi. Seluruh algoritma benar selama kita bisa berulang kali menemukan “simpul belum selesai dengan jarak sementara terkecil”. *Heap* adalah salah satu cara menemukannya. Gagasan Dial adalah cara yang lebih murah ketika bobotnya bilangan bulat kecil.

6.2.4 Algoritma Dial: cincin ember

Pada tahun 1969 Robert Dial mengamati bahwa jika semua bobot adalah bilangan bulat antara 1 dan C , jarak-jarak sementara yang masih menunggu diproses tidak pernah tersebar pada lebih dari $C + 1$ nilai berurutan [6]. Misalkan algoritma sedang menyelesaikan simpul-simpul berjarak cur . Setiap nilai yang menunggu dihasilkan oleh relaksasi sisi dari simpul selesai u dengan $d[u] \leq cur$, sehingga

$$cur \leq d[v] \leq cur + C \quad \text{untuk setiap simpul menunggu } v \quad (6.3)$$

Jendela pada [Persamaan](#) (6.3) memuat $C + 1$ nilai. Jika kita menyimpan larik berisi $C + 1$ daftar, yaitu *ember*, dan menaruh simpul berjarak sementara d ke dalam ember $d \bmod (C + 1)$, maka dua jarak menunggu yang berbeda tidak pernah jatuh ke ember yang sama. Untuk menemukan simpul berikutnya yang harus diselesaikan, kita cukup melihat ember $cur \bmod (C + 1)$; jika kosong, cur dinaikkan satu dan kita melihat ember berikutnya, memutar di sepanjang cincin.

Pseudokode 6.1 menuliskan prosedurnya.

Pseudokode 6.1 Algoritma Dial dengan $C + 1$ ember melingkar; baris 6 melewati salinan yang tertinggal ketika jarak suatu simpul membaik.

```
DIAL(G, s, C)
1  for each node v: d[v] = INF
2  d[s] = 0; put s into bucket[0]; cur = 0
3  while some bucket is non-empty
4      while bucket[cur mod (C+1)] is empty: cur = cur + 1
5      take any node u out of bucket[cur mod (C+1)]
6      if d[u] != cur: continue // stale copy, ignore it
7      for each edge (u, v) with weight w
8          if cur + w < d[v]
9              d[v] = cur + w
10             put v into bucket[(cur + w) mod (C+1)]
```

Baris 6 perlu dijelaskan. Ketika jarak sementara suatu simpul membaik dari 40 menjadi 35, kita tidak mencari salinan lamanya di ember 40 untuk dihapus; itu lambat. Kita cukup menambahkan salinan baru ke ember 35 dan membiarkan salinan lama di tempatnya. Ketika salinan lama akhirnya diambil, jarak yang tercatat padanya tidak lagi sama dengan $d[v]$ dan kita membuangnya. *Penghapusan malas (lazy deletion)* ini adalah trik yang sama yang sudah biasa dipakai sebagian besar peserta pada Dijkstra berbasis *heap*.

Waktu jalannya diperoleh dengan menghitung. Setiap relaksasi yang berhasil menambahkan satu salinan ke suatu ember, sehingga totalnya paling banyak E salinan, masing-masing diambil sekali. Penunjuk *cur* hanya bergerak maju dan berhenti pada jarak terbesar D , sehingga pemeriksaan ember kosong berbiaya total $O(D)$. Bersama-sama menjadi $O(V + E + D)$, dan karena $D \leq C(V - 1)$ batas klasiknya adalah $O(E + VC)$. Pada kisi, jarak lintasan jauh di bawah kasus terburuk itu, dan perilaku praktisnya linear.

Gambar 6.1 memperlihatkan jarak yang dihitung algoritma Dial pada peta kecil berukuran 5×6 . Bilangan besar di tiap petak adalah biaya masuknya w , dan label kecil adalah jaraknya dari titik awal. Petak yang diarsir membentuk satu rute termurah dengan total biaya 17; perhatikan bahwa rute itu rela memutar lewat tepi kiri untuk menghindari angka 9 dan 8 di dekat titik awal.

awal (biaya diabaikan)

1 d=0	9 d=9	3 d=12	1 d=13	7 d=20	2 d=22
2 d=2	8 d=10	1 d=10	6 d=16	1 d=17	5 d=22
4 d=6	1 d=7	2 d=9	9 d=18	3 d=20	1 d=20
3 d=9	7 d=14	1 d=10	1 d=11	8 d=19	2 d=19
5 d=14	2 d=16	6 d=16	4 d=15	1 d=16	1 d=17

tujuan

Gambar 6.1 Peta 5×6 dengan biaya masuk (angka besar) dan jarak d yang dihitung algoritma Dial; petak yang diarsir membentuk rute termurah berbiaya 17.

WAWASAN KUNCI: Algoritma Dial *adalah* algoritma Dijkstra. Yang berubah hanyalah struktur data untuk menemukan “jarak sementara terkecil”: sebuah cincin berisi $C + 1$ daftar sebagai ganti *heap*. Semua yang Anda ketahui tentang kebenaran, salinan basi, dan penghentian dini tetap berlaku tanpa perubahan.

6.2.5 Ruang keadaan tanpa peta

Untuk melihat bahwa metode ini tidak bergantung pada gambar, perhatikan teka-teki lama dua ember. Ember 5 liter dan ember 3 liter berdiri di samping keran; kita ingin tepat 4 liter di ember besar. Keadaannya adalah pasangan (a, b) , yaitu liter air di ember besar dan ember kecil, dengan $0 \leq a \leq 5$ dan $0 \leq b \leq 3$, sehingga ada $6 \cdot 4 = 24$ keadaan. Aksinya: mengisi satu ember dari keran, mengosongkan satu ember ke tanah, atau menuang satu ember ke ember lain sampai yang pertama kosong atau yang kedua penuh. Jika setiap aksi berbiaya satu langkah, BFS dari $(0,0)$ menemukan jumlah langkah tersedikit; jawabannya enam, melalui $(5,0)$, $(2,3)$, $(2,0)$, $(0,2)$, $(5,2)$, dan $(4,3)$.

Ubahlah satu kalimat dalam ceritanya, maka bobotnya ikut berubah. Jika mengisi dari keran berbiaya sebanyak liter air yang dipakai, sedangkan menuang dan mengosongkan gratis, bobotnya menjadi bilangan bulat kecil 0 sampai 5, dan alat yang tepat berpindah dari BFS ke algoritma Dial dengan enam ember. Tidak ada bagian lain dari program yang berubah: keadaan dan aksinya sama, hanya biayanya yang berbeda. Itulah inti pemisahan empat pertanyaan pada Subbab 6.2.1. Dalam lomba, soal yang tampak asing sangat sering hanyalah soal lintasan terpendek biasa atas keadaan-keadaan yang belum Anda tuliskan, dan sepuluh menit untuk menuliskannya adalah sepuluh menit paling berharga dalam mengerjakan soal itu.

6.3 Soal Pemanasan

Sebelum studi kasus, dua program kecil melatih anak tangga bagian bawah. Masing-masing lengkap dan dapat langsung dijalankan.

6.3.1 Kuda catur pada papan dengan petak terlarang

Sebuah kuda catur berangkat dari $a1$ dan ingin mencapai $h8$; sebagian petak terlarang. Setiap langkah berbiaya 1, sehingga BFS biasa adalah alat yang tepat. Keadaannya adalah petak tempat kuda berada (64 keadaan) dan aksinya adalah delapan lompatan kuda. **Kode 6.1** menjalankan tiga skenario.

Kode 6.1 `knight.cpp`: *breadth-first search* atas 64 keadaan kuda, dengan petak terlarang.

```
// State-space BFS: fewest knight moves from a1 to h8 avoiding forbidden squares.
#include <iostream>
#include <queue>
#include <string>
#include <vector>
using namespace std;
int knightDist(const vector<string>& banned) {
    bool bad[8][8] = {};
    for (auto& s : banned) bad[s[0] - 'a'][s[1] - '1'] = true;
    int dist[8][8];
    for (auto& row : dist)
        for (int& x : row) x = -1;
    int dx[8] = {1, 2, 2, 1, -1, -2, -2, -1}, dy[8] = {2, 1, -1, -2, -2, -1, 1, 2};
    queue<pair<int, int>> q;
    dist[0][0] = 0;
    q.push({0, 0});
    while (!q.empty()) {
        auto [x, y] = q.front();
        q.pop();
        for (int k = 0; k < 8; k++) {
            int nx = x + dx[k], ny = y + dy[k];
            if (nx < 0 || nx > 7 || ny < 0 || ny > 7 || bad[nx][ny]) continue;
            if (dist[nx][ny] != -1) continue; // first visit = shortest in BFS
            dist[nx][ny] = dist[x][y] + 1;
            q.push({nx, ny});
        }
    }
    return dist[7][7]; // -1 if h8 was never reached
}
int main() {
    cout << "no square banned: " << knightDist({}) << " moves\n";
    cout << "b3 and f7 banned: " << knightDist({"b3", "f7"}) << " moves\n";
    cout << "f7 and g6 banned: " << knightDist({"f7", "g6"}) << "\n";
}
```

```
no square banned: 6 moves
b3 and f7 banned: 6 moves
f7 and g6 banned: -1
```

Dua jawaban pertama menunjukkan bahwa kuda memerlukan enam langkah dari sudut ke sudut, dan bahwa melarang b3 dan f7 masih menyisakan rute enam langkah lainnya. Jawaban ketiga adalah pelajaran kecil tentang pemodelan: f7 dan g6 adalah satu-satunya petak tempat kuda dapat melompat ke h8, sehingga melarang keduanya membuat h8 tak terjangkau. BFS melaporkannya dengan benar karena -1 adalah nilai awal dan h8 tidak pernah ditemukan. Pada BFS, saat pertama kali suatu simpul ditemukan juga merupakan saat jaraknya final, sehingga program ini sama sekali tidak memerlukan pemeriksaan “salinan basi”.

6.3.2 Menjebol dinding sesedikit mungkin: BFS 0-1

Kini setiap petak pada sebuah peta kecil adalah petak bebas (.) atau dinding (#). Melangkah ke petak bebas berbiaya 0 dan melangkah ke dinding (menjebolnya) berbiaya 1. Dengan bobot 0 dan 1, simpul-simpul dapat disimpan dalam antrean dua ujung (*deque*): simpul yang dicapai lewat sisi berbobot 0 masuk ke *depan*, karena jaraknya sama dengan simpul yang sedang diproses, dan simpul yang dicapai lewat sisi berbobot 1 masuk ke *belakang*. *Deque* itu selalu memuat paling banyak dua jarak berbeda dalam urutan terurut; inilah algoritma Dial dengan hanya dua ember. **Kode 6.2** mencetak jawaban dan seluruh tabel jarak.

Kode 6.2 `zero_one_bfs.cpp`: BFS 0-1 dengan antrean dua ujung pada peta berisi petak bebas dan dinding.

```
// 0-1 BFS: fewest walls to break from the top-left to the bottom-right cell.
// Stepping into '.' costs 0, stepping into '#' (breaking it) costs 1.
#include <deque>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main() {
    vector<string> g = {"..#....",
                      "###.##.",
                      "...#..#",
                      ".#.#.#.",
                      ".#...#."};

    int R = g.size(), C = g[0].size();
    vector<vector<int>> d(R, vector<int>(C, 1 << 30));
    deque<pair<int, int>> dq;
    d[0][0] = 0;
    dq.push_back({0, 0});
    int dr[4] = {-1, 1, 0, 0}, dc[4] = {0, 0, -1, 1};
    while (!dq.empty()) {
        auto [r, c] = dq.front();
        dq.pop_front();
        for (int k = 0; k < 4; k++) {
            int nr = r + dr[k], nc = c + dc[k];
            if (nr < 0 || nr >= R || nc < 0 || nc >= C) continue;
            int w = (g[nr][nc] == '#'); // edge weight 0 or 1
            if (d[r][c] + w < d[nr][nc]) {
                d[nr][nc] = d[r][c] + w;
                if (w == 0) dq.push_front({nr, nc}); // weight 0: same layer
                else dq.push_back({nr, nc}); // weight 1: next layer
            }
        }
    }
    cout << "walls to break: " << d[R - 1][C - 1] << "\n";
    for (int r = 0; r < R; r++) {
        for (int c = 0; c < C; c++) cout << d[r][c] << (c + 1 < C ? ' ' : '\n');
    }
}
```

```
walls to break: 2
0 0 1 1 1 1 1
1 1 2 1 2 2 1
1 1 1 2 1 1 2
1 2 1 2 1 2 2
1 2 1 1 1 2 2
```

Dua dinding sudah cukup. Membaca tabel baris demi baris adalah kebiasaan yang baik: setiap petak bebas yang bertetangga dengan petak berjarak k berjarak paling banyak k , dan setiap dinding di sebelahnya berjarak paling banyak $k + 1$. Jika pemeriksaan manual atas aturan itu gagal di mana pun, implementasinya salah.

6.4 Studi Kasus: CHIPSLL

6.4.1 Soal dengan kata-kata kita sendiri

Soalnya adalah SPOJ CHIPSLL [1]. Masukan diawali banyaknya kasus uji T (paling banyak 10). Setiap kasus uji memberikan enam bilangan bulat N, M, C_x, C_y, M_x, M_y lalu sebuah *seed*. Peta terdiri atas N baris dan M kolom, $N, M \leq 2000$. Chip mulai di petak (C_x, C_y) dan Melinda menunggu di petak (M_x, M_y) ; baris dan kolom dinomori mulai 1.

Peta *tidak* diberikan dalam masukan. Peta dihasilkan oleh pembangkit bilangan acak semu yang dimulai dari *seed*. Petak-petak dikunjungi baris demi baris, dari kiri ke kanan, dengan melewati petak Chip dan petak Melinda. Untuk setiap petak lainnya, *seed* diperbarui sebagai

$$seed \leftarrow (seed \cdot 1\,000\,003 + 10\,007) \bmod 1\,000\,000\,009 \quad (6.4)$$

dan kekuatan petak itu menjadi $p = seed \bmod 63$. Kekuatan 0 berarti petak kosong; 1 sampai 26 berarti air, 27 sampai 52 monster, 53 sampai 62 pintu. Jenis rintangan tidak memengaruhi waktu: memasuki petak selalu memakan $1 + p$ detik, seperti pada [Persamaan \(6.1\)](#). Petak Chip dan petak Melinda berbiaya 1. Keluaran untuk kasus uji t adalah baris **Case t : X** dengan X total waktu terkecil.

Dua peringatan muncul dari uraian ini. Pertama, pembangkitnya harus direproduksi *persis*: konstanta yang sama, urutan petak yang sama, petak yang dilewati sama. Satu digit yang salah pada [Persamaan \(6.4\)](#) menghasilkan peta yang sama sekali berbeda dan jawaban salah pada setiap uji. Kedua, hasil kali $seed \cdot 1\,000\,003$ mencapai sekitar 10^{15} , sehingga harus dihitung dengan aritmetika 64 bit (**long long**).

6.4.2 Contoh yang dikerjakan dengan tangan

Ambil masukan mungil

```
1
2 3 1 1 2 3
7
```

Petanya dua baris dan tiga kolom; Chip di (1,1) dan Melinda di (2,3). Pembangkit mengunjungi (1,2), (1,3), (2,1), dan (2,2), melewati dua petak khusus. Dimulai dari *seed* 7, [Persamaan \(6.4\)](#) menghasilkan 7010028, 48977001, 147500217, dan 658183158, yang sisa baginya terhadap 63 adalah 18, 45, 18, dan 45. Dengan demikian biaya masuknya adalah

```
1 19 46
19 46 1
```

Setiap rute tiga langkah dari pojok kiri atas ke pojok kanan bawah melewati satu 19, satu 46, dan angka 1 milik Melinda, sehingga jawabannya $19 + 46 + 1 = 66$. Program memberikan hasil yang sama:

```
Case 1: 66
```

Berkas **sample.txt** di folder bab ini memuat dua peta yang lebih besar (3×4 dan 5×5), yang untuknya program mencetak **Case 1: 125** dan **Case 2: 160**; uji stres pada Subbab 6.5 memeriksa jawaban semacam itu secara otomatis.

6.4.3 Merancang penyelesaian

Dengan model dari Subbab 6.2, rancangannya nyaris mekanis.

1. **Bangkitkan biayanya.** Simpan $w(v) = 1 + p$ untuk setiap petak dalam larik **unsigned char**: 63 muat dalam satu *byte*, dan empat juta *byte* itu murah.
2. **Ratakan kisinya.** Beri petak (i, j) indeks $i \cdot 2021 + j$. Larik satu dimensi lebih cepat daripada *vector* berisi *vector*: satu alokasi, tanpa mengejar penunjuk, dan tetangga hanyalah selisih indeks (± 1 untuk kiri dan kanan, ± 2021 untuk atas dan bawah).
3. **Jalankan algoritma Dial** dengan $W = 64$ ember. Karena $C = 63$, jendela [Persamaan \(6.3\)](#) memuat tepat 64 nilai. Berhenti segera setelah petak Melinda diambil dari ember dengan jarak terkini: sejak saat itu jaraknya final.

4. Cetak **Case t: X**.

Menjumlahkan kebutuhan memori sebelum menulis kode selalu menguntungkan. Larik biaya memerlukan $2021 \cdot 2021 \approx 4.1$ juta *byte*, sekitar 4 MB. Larik jarak menampung nilai *int* sebanyak itu pula, sekitar 16 MB. Ember adalah satu-satunya bagian yang ukurannya bergantung pada peta: setiap relaksasi yang berhasil menambahkan satu entri berukuran empat *byte*, dan walaupun sebuah petak dapat diperbaiki beberapa kali sebelum selesai, pada peta-peta ini ember paling banyak berisi beberapa juta entri, dan memori ember yang sudah kosong dipakai ulang ketika cincin berputar. Secara keseluruhan program berada di kisaran 30 sampai 50 MB, dengan nyaman di dalam batas yang lazim. Perkiraan yang sama untuk versi berbasis *heap* memberikan larik jarak serupa, tetapi ditambah *heap* berisi pasangan (jarak, petak), delapan *byte* per entri, dan *heap* tidak pernah mengecilkan penyimpanannya selama program berjalan; itu satu alasan lagi mengapa versi ember adalah pilihan yang lebih aman di sini.

Untuk memahami seberapa besar perbedaan kecepatan yang dapat diharapkan, bayangkan pekerjaan setiap versi. Versi ember mengerjakan beberapa operasi sederhana untuk setiap relaksasi: menambahkan ke sebuah *vector* dan kelak mengambilnya kembali. Versi *heap* harus menjaga urutan seluruh isinya pada setiap penyisipan dan pengambilan, dengan menukar elemen di sepanjang jalur setinggi pohon *heap*. Pada jutaan relaksasi, selisih itu terasa.

Bilangan 2021 sebagai langkah baris layak diberi komentar. Memakai langkah sedikit lebih besar daripada M terbesar membuat baris 0 dan $N + 1$ serta kolom 0 dan $M + 1$ ada sebagai bantalan yang tidak dipakai, dan menjaga rumus indeks tetap sama untuk setiap kasus uji, berapa pun M -nya.

6.4.4 Penyelesaian lengkap

Kode 6.3 adalah program lengkap yang tercatat *Accepted* di SPOJ. Baris-baris di antara `// <P50>` dan `// </P50>` adalah paruh kedua intinya: relaksasi keempat tetangga.

Kode 6.3 `CHIPSLL.cpp`: algoritma Dial dengan 64 ember melingkar pada kisi datar.

```
#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){
    int start=sr*MAXN+sc, target=tr*MAXN+tc;
    for(int i=1;i<=N;i++) memset(dist_flat+i*MAXN+1, 0x3f, M*sizeof(int));
    for(int i=0;i<W;i++) bkt[i].clear();
    dist_flat[start]=0;
    bkt[0].push_back(start);
    int cur_d=0;
    while(true){
        bool found=false;
        for(int s=0;s<W;s++){
            if(!bkt[(cur_d+s)%W].empty()){cur_d+=s;found=true;break;}
        }
        if(!found) break;
        int b=cur_d%W;
        while(!bkt[b].empty()){
            int id=bkt[b].back(); bkt[b].pop_back();
            if(dist_flat[id]!=cur_d) continue;
            if(id==target) return cur_d;
            // <P50>
            // Relax the four neighbors of id: nd = cur_d + entry cost of the
```

Kode 6.3 (Lanjutan) CHIPSSL.cpp: algoritma Dial dengan 64 ember melingkar pada kisi datar.

```
// neighbor; if nd improves it, store it and push into bucket nd%W.
// >>> YOUR TURN (Public Edition): write this part yourself, following the hint
// above. <<<
// </P50>
}
cur_d++;
}
return dist_flat[target];
}
int main(){
    int T; if(scanf("%d",&T)!=1) return 0;
    for(int t=1;t<=T;t++){
        int N,M,Cx,Cy,Mx,My;
        if(scanf("%d%d%d%d%d",&N,&M,&Cx,&Cy,&Mx,&My)!=6) return 0;
        if(scanf("%lld",&seed_g)!=1) return 0;
        gen_pwv(N,M,Cx,Cy,Mx,My);
        printf("Case %d: %d\n",t,solve(N,M,Cx,Cy,Mx,My));
    }
}
```

6.4.5 Baris demi baris

Konstanta dan larik. `MAXN = 2021` adalah langkah baris dan `MAXCELL` ukuran larik datar. `W = 64` adalah banyaknya ember. `pwv` menyimpan biaya masuk setiap petak dan `dist_flat` jarak sementara; keduanya variabel global `static`, sehingga tinggal di segmen data program, bukan di *stack* yang akan meluap pada 16 MB.

update_seed dan gen_pwv. Keduanya mengimplementasikan [Persamaan \(6.4\)](#) secara harfiah. Perulangannya mengunjungi baris $1..N$ dan kolom $1..M$ secara berurutan, menetapkan biaya dua petak khusus menjadi 1 *tanpa* memajukan *seed*, dan selain itu memajukan *seed* sekali lalu menyimpan $1 + (\text{seed} \bmod 63)$.

Inisialisasi di solve. `memset` dengan *byte* `0x3f` mengisi setiap `int` dengan `0x3f3f3f3f`, sekitar $1.06 \cdot 10^9$, “tak hingga” yang praktis dan masih menyisakan ruang untuk ditambah satu bobot tanpa luapan. Hanya N baris yang dipakai yang diatur ulang, dan hanya M kolomnya; mengatur ulang keempat juta sel untuk uji 10×10 akan membuang waktu pada sepuluh kasus uji. Ember dikosongkan, titik awal mendapat jarak 0 dan masuk ke ember 0.

Mencari ember tak kosong berikutnya. Perulangan `for` bagian dalam atas `s` melihat paling jauh W ember ke depan, mulai dari `cur_d`. Jika ke-64 ember kosong, tidak ada simpul yang menunggu dan perulangan berakhir (pada kisi CHIPSSL yang terhubung, ini hanya terjadi setelah target ditemukan). Jika tidak, `cur_d` langsung melompat ke jarak tak kosong yang pertama.

Memproses satu ember. Simpul diambil dari *belakang vector*: di dalam satu ember semua simpul berjarak sama, jadi urutannya tidak penting, dan `pop_back` adalah operasi termurah pada *vector*. Pemeriksaan `dist_flat[id] != cur_d` membuang salinan basi (baris 6 pada [Pseudokode 6.1](#)). Pemeriksaan `id == target` menghentikan pencarian lebih awal.

Relaksasi (blok P50). Baris dan kolom diperoleh kembali dari indeks datar dengan pembagian dan sisa bagi. Lambda `relax` menerapkan [Persamaan \(6.2\)](#) pada satu tetangga, dan empat pemanggilan berpenjaga mencakup atas, bawah, kiri, dan kanan. Penjaga `r>1`, `r<N`, `c>1`, `c<M` menjaga Chip tetap di dalam peta.

main. Fungsi ini membaca kepala setiap uji, membaca *seed* ke variabel global `seed_g`, membangkitkan peta, dan mencetak jawaban dalam format yang diminta. Hasil setiap `scanf`

diperiksa, sehingga kompilator tetap tenang di bawah `-Wall -Wextra` dan program berhenti dengan rapi di akhir masukan.

6.4.6 Jebakan

JEBAKAN: Memakai `W = 63`. Bobot mencapai 63, sehingga jendela *Persamaan* (6.3) memuat 64 nilai berbeda. Dengan 63 ember, jarak d dan $d + 63$ berbagi tempat, sebuah simpul dapat diproses pada waktu yang salah, dan jawaban menjadi salah hanya pada sebagian peta, jenis *bug* yang paling buruk.

- **Memajukan *seed* pada petak khusus.** Pembangkit melewatinya. Memajukan *seed* di sana menggeser semua petak sesudahnya.
- **Aritmetika *seed* 32 bit.** `seed_g * 1000003` meluap pada `int`; program memakai `long long` dan akhiran `LL`.
- **Melupakan salinan basi.** Tanpa pemeriksaan `dist_flat[id] != cur_d` program tetap benar, tetapi dapat merelaksasi simpul yang sama berkali-kali dan kehabisan waktu.
- ***Vector* berisi *vector* untuk kisi.** Empat juta *vector* kecil memakan memori untuk kepalanya dan waktu untuk alokasinya; larik datar menghindari keduanya.
- **Mengatur ulang seluruh larik pada setiap uji.** Benar tetapi lambat bila banyak uji kecil menyusul satu uji besar; atur ulang hanya bagian yang dipakai uji itu.

6.5 Uji Stres terhadap *Brute Force*

Program untuk soal besar baru dapat dipercaya setelah dibandingkan dengan program lambat yang jelas benar pada banyak masukan acak kecil. Di sini program lambatnya, **Kode 6.4**, memakai pembangkit yang sama tetapi Dijkstra berbasis *heap* ala buku teks pada *vector* berisi *vector*. Program ini tidak berbagi struktur data apa pun dengan penyelesaian cepat, sehingga *bug* pada yang satu kecil kemungkinannya terulang pada yang lain.

Kode 6.4 `brute.cpp`: penyelesaian acuan dengan *binary heap*.

```
// Reference for CHIPSLL: same grid generator, textbook heap Dijkstra.
#include <stdio>
#include <queue>
#include <vector>
using namespace std;
long long seed_g;
int main() {
    int T;
    if (scanf("%d", &T) != 1) return 0;
    for (int t = 1; t <= T; t++) {
        int N, M, Cx, Cy, Mx, My;
        if (scanf("%d%d%d%d%d%d%lld", &N, &M, &Cx, &Cy, &Mx, &My, &seed_g) != 7)
            return 0;
        vector<vector<int>> cost(N + 1, vector<int>(M + 1, 1));
        for (int i = 1; i <= N; i++)
            for (int j = 1; j <= M; j++) {
                if ((i == Cx && j == Cy) || (i == Mx && j == My)) continue;
                seed_g = (seed_g * 1000003LL + 10007LL) % 1000000009LL;
                cost[i][j] = 1 + (int)(seed_g % 63);
            }
        const int INF = 1 << 30;
        vector<vector<int>> d(N + 1, vector<int>(M + 1, INF));
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
        d[Cx][Cy] = 0;
        pq.push({0, Cx * (M + 1) + Cy});
        int dr[4] = {-1, 1, 0, 0}, dc[4] = {0, 0, -1, 1};
        while (!pq.empty()) {
            auto [du, id] = pq.top();
            pq.pop();
            int r = id / (M + 1), c = id % (M + 1);
            if (du != d[r][c]) continue;
            for (int k = 0; k < 4; k++) {
                int nr = r + dr[k], nc = c + dc[k];
                if (nr < 1 || nr > N || nc < 1 || nc > M) continue;
                if (du + cost[nr][nc] < d[nr][nc]) {
                    d[nr][nc] = du + cost[nr][nc];
                    pq.push({d[nr][nc], nr * (M + 1) + nc});
                }
            }
        }
        printf("Case %d: %d\n", t, d[Mx][My]);
    }
}
```

Pembangkit **Kode 6.5** menuliskan hingga empat peta acak berukuran paling besar 9×9 petak, dengan titik awal, target, dan *seed* acak. Karena bibit acaknya diambil dari baris perintah, setiap kasus yang gagal dapat direproduksi dengan menjalankan ulang nomor yang sama.

Kode 6.5 *gen.py*: masukan acak kecil untuk CHIPSLL.

```
# Random small CHIPSLL inputs: python3 gen.py SEED
import random, sys
random.seed(int(sys.argv[1]))
T = random.randint(1, 4)
print(T)
for _ in range(T):
    n, m = random.randint(1, 9), random.randint(1, 9)
    print(n, m, random.randint(1, n), random.randint(1, m),
          random.randint(1, n), random.randint(1, m))
    print(random.randint(0, 10**9))
```

Akhirnya, **Kode 6.6** mengompilasi kedua program dengan peringatan dinyalakan, menjalankannya pada 300 masukan bangkitan, dan berhenti pada perbedaan pertama sambil mencetak masukan penyebabnya. Skrip yang sama, dengan hanya nama soal yang diganti, dipakai di setiap bab buku ini.

Kode 6.6 `stress.sh`: kompilasi, bangkitkan, bandingkan, berhenti pada ketidakcocokan pertama.

```
#!/usr/bin/env bash
# Stress test: CHIPSLL against brute.cpp on random small inputs.
# usage: bash stress.sh [ROUNDS] (default 300)
set -e
cd "$(dirname "$0")"
FLAGS="-std=c++17 -O2 -Wall -Wextra"
g++ $FLAGS -o sol CHIPSLL.cpp
g++ $FLAGS -o brute brute.cpp
for i in $(seq 1 "${1:-300}"); do
    python3 gen.py "$i" > in.txt
    ./sol < in.txt > out_sol.txt
    ./brute < in.txt > out_brute.txt
    if ! cmp -s out_sol.txt out_brute.txt; then
        echo "MISMATCH on seed $i"; cat in.txt; exit 1
    fi
done
echo "all ${1:-300} random tests passed"
```

```
all 300 random tests passed
```

Uji kecil memeriksa kebenaran; uji kecil tidak mengatakan apa pun tentang kecepatan. Karena itu kami juga mengukur waktu masukan sah terbesar: sepuluh peta berukuran 2000×2000 . Seluruh berkas selesai dalam sekitar 1,1 detik di mesin kami, kira-kira 0,11 detik per peta termasuk pembangkitan. Pada satu peta seperti itu, versi *heap* memerlukan sekitar 0,66 detik dibandingkan 0,19 detik untuk algoritma Dial, selisih tiga setengah kali yang seluruhnya berasal dari penggantian *heap* dengan ember.

6.6 Taktik pada Hari Lomba

- **Klasifikasikan bobot lebih dulu.** Sebelum memilih algoritma, tulis bobot sisi terkecil dan terbesar di pinggir kertas buram. Tangga pada **Tabel 6.1** lalu pilihlah untuk Anda.
- **Salin pembangkit karakter demi karakter.** Bila soal membangkitkan masukannya sendiri, ketik pembangkit dari soal secara perlahan, lalu cetak beberapa nilai pertama untuk kasus mungil dan bandingkan dengan tangan, seperti pada Subbab 6.4.2.
- **Perkirakan memori sebelum menulis kode.** Empat juta nilai `int` adalah 16 MB; empat juta nilai `unsigned char` adalah 4 MB. *Map* berisi `pair<int,int>` per sel akan memerlukan beberapa kali lipat. Ketahui angka-angka ini sebelum memilih tipe.
- **Siapkan templat Dijkstra dengan *heap*.** Templat itu adalah *brute force* bagi setiap metode lintasan terpendek yang lebih cepat, dan menuliskannya dari ingatan dalam lima menit adalah keterampilan yang layak dilatih.
- **Berhenti lebih awal.** Bila hanya satu target yang penting, kembalikan jawaban begitu target itu selesai. Pada peta acak, hal ini sering menghemat separuh pekerjaan.

6.7 Ringkasan Bab

- Ruang keadaan mengubah teka-teki menjadi graf; tuliskan keadaan, banyaknya keadaan, aksi, dan biaya sebelum menulis kode.
- Bentuk bobot menentukan algoritma: BFS untuk bobot 1, BFS 0-1 untuk bobot 0 dan 1, algoritma Dial untuk bilangan bulat kecil hingga C , Dijkstra dengan *heap* untuk bobot tak negatif umum.
- Algoritma Dial adalah Dijkstra dengan cincin berisi $C + 1$ ember; jendela **Persamaan** (6.3) menjamin tidak ada dua jarak menunggu yang berbeda ember.

- Penghapusan malas (lewati salinan yang jaraknya tidak lagi cocok) menjaga Dial maupun Dijkstra dengan *heap* tetap sederhana dan cepat.
- Pada kisi raksasa, larik datar dan tipe kecil menentukan apakah program muat dalam batas waktu dan memori.

6.8 Soal Latihan

1. Sebuah labirin memuat kunci `a-d` dan pintu `A-D`; sebuah pintu hanya dapat dilewati selama kita memegang kuncinya. Setiap langkah berbiaya 1. Uraikan keadaannya, hitung banyaknya keadaan untuk labirin 50×50 , dan sebutkan algoritmanya.
2. Ubah **Kode 6.2** sehingga melangkah ke petak bebas berbiaya 1 dan menjebol dinding berbiaya 3. Algoritma mana dari **Tabel 6.1** yang kini menjadi pilihan alami, dan berapa ember yang diperlukan?
3. Jelaskan mengapa algoritma Dial bisa salah jika ada sisi berbobot 0 dan program memakai aturan “berhenti memindai ember saat ember menjadi kosong” tetapi memasukkan tetangga berbobot 0 ke ember *yang sama* yang sedang dipindai. Apakah program pada **Kode 6.3** aman dari masalah ini?
4. Buktikan bahwa pada **Kode 6.2** *deque* tidak pernah memuat lebih dari dua jarak berbeda.
5. Paling besar berapa jawaban CHIPSLL untuk sebuah peta? Gunakan **Persamaan (6.1)** dan ukuran kisi untuk memberikan batas atas, lalu putuskan apakah `int` sudah cukup.

6.9 Penyelesaian Soal Latihan

1. Keadaan harus mengingat petak dan kunci yang dipegang: $(r, c, mask)$ dengan *mask* himpunan 4 bit. Banyaknya $50 \cdot 50 \cdot 2^4 = 40\,000$ keadaan. Memungut kunci mengubah *mask*; memasuki petak pintu hanya diizinkan jika bit yang bersesuaian menyala. Semua langkah berbiaya 1, sehingga BFS biasa atas keadaan-keadaan ini benar dan cepat.
2. Bobotnya kini 1 dan 3, bilangan bulat kecil dengan $C = 3$, sehingga algoritma Dial dengan $C + 1 = 4$ ember adalah pilihan alami. (BFS 0-1 tidak lagi berlaku, dan *heap* bisa dipakai tetapi lebih lambat.)
3. Dengan sisi berbobot 0, sebuah simpul dapat dimasukkan ke ember yang sedang dipindai. Program yang mengiterasi *salinan* ember, atau berhenti setelah isi awalnya habis, akan melewatkan simpul-simpul itu. Program CHIPSLL mengambil dari ember sampai ember benar-benar kosong (`while(!bkt[b].empty())`), sehingga simpul yang baru ditambahkan ke ember yang sama juga akan diproses; lagi pula bobot CHIPSLL paling kecil 1, sehingga situasi itu tidak pernah muncul.
4. Induksi pada banyaknya operasi. Mula-mula *deque* hanya berisi titik awal, satu jarak. Misalkan *deque* berisi jarak k (bagian depan) dan mungkin $k + 1$ (bagian belakang), dengan urutan itu. Kita mengambil simpul berjarak k dari depan; tetangga lewat bobot 0 mendapat jarak k dan masuk ke depan, tetangga lewat bobot 1 mendapat $k + 1$ dan masuk ke belakang. Urutan dan sifat dua nilai tetap terjaga. Ketika bagian k habis, semua simpul berjarak $k + 1$, dan argumen berulang dengan $k + 1$.
5. Sebuah rute tidak pernah perlu mengunjungi petak dua kali, dan biaya lintasan terpendek tidak melebihi biaya rute monoton mana pun yang terdiri atas $(N - 1) + (M - 1)$ langkah, masing-masing berbiaya paling banyak 63. Jadi jawabannya paling banyak $63 \cdot (1999 + 1999) = 251\,874$. Sebuah `int` menampung nilai hingga sekitar $2.1 \cdot 10^9$, jadi jauh lebih dari cukup, dan menambahkan satu bobot pada “tak hingga” `0x3f3f3f3f` juga tidak dapat meluap.

Daftar Pustaka

- [1] Sphere Online Judge, “Problem CHIPSSL.” [Online]. Available: <https://www.spoj.com/problems/CHIPSSL/>
- [2] S. Halim, F. Halim, and S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.
- [3] A. Laaksonen, *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, 3rd ed. Cham, Switzerland: Springer, 2024.
- [4] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.
- [5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022, ch. 22.
- [6] R. B. Dial, “Algorithm 360: Shortest-path forest with topological ordering,” *Communications of the ACM*, vol. 12, no. 11, pp. 632–633, 1969.

Bab 7

Himpunan Saling Lepas, Algoritma Kruskal, dan Sapuan — VILUNIT

Algoritma Kruskal untuk pohon perentang minimum biasanya diajarkan sebagai “urutkan semua sisi, lalu ambil satu per satu”. Resep itu runtuh ketika graf memiliki terlalu banyak sisi untuk dituliskan: lima ribu objek saja sudah memiliki lebih dari dua belas juta pasangan. Soal olimpiade gemar sekali pada situasi ini, karena situasi itu memisahkan peserta yang sekadar menghafal resep dari peserta yang memahami mengapa resep itu bekerja. Bab ini membangun pemahaman tersebut. Anda akan mempelajari struktur *disjoint-set union* (DSU, gabungan himpunan saling lepas) yang menjadi sandaran algoritma Kruskal, melihat mengapa algoritma Kruskal hanya memerlukan sisi yang *dikelompokkan menurut bobot* alih-alih diurutkan penuh, dan memakai *sapuan* (*sweep*) terurut untuk menemukan sedikit sisi yang penting di antara jutaan kandidat. Studi kasusnya, SPOJ VILUNIT [1], menanyakan cara termurah menghubungkan hingga 5000 desa berbentuk persegi panjang ketika setiap sambungan berbiaya 0, 1, atau 2, dan soal itu tunduk tepat pada gabungan ketiga gagasan tadi.

7.1 Desa-Desa di Atas Peta

Sebuah kantor kecamatan ingin menghubungkan semua desa di wilayahnya menjadi satu jaringan, misalnya dengan jalan bersama atau sistem air bersih bersama. Pada peta, setiap desa berbentuk persegi panjang. Menghubungkan dua desa tidak berbiaya apa pun jika wilayah keduanya sudah bertumpuk, karena pada hakikatnya keduanya adalah satu permukiman. Biayanya satu satuan jika kedua persegi panjang terletak berdampingan di sepanjang satu jalur bersama: bayangan keduanya pada sumbu timur–barat bertumpuk, atau bayangan keduanya pada sumbu utara–selatan bertumpuk, sehingga sebuah jalan lurus di sepanjang jalur itu dapat menyatukan keduanya. Pasangan lainnya berbiaya dua satuan. Kantor itu ingin setiap desa dapat dicapai dari setiap desa lain, langsung atau melalui desa lain, dengan total biaya sekecil mungkin.

Ini adalah soal *pohon perentang minimum* (*minimum spanning tree*). Desa-desanya adalah simpul; setiap pasangan desa adalah sisi berbobot 0, 1, atau 2; kita menginginkan himpunan sisi yang menghubungkan semuanya dengan total bobot terkecil. Rintangannya murni soal ukuran. Dengan $n = 5000$ desa terdapat $n(n - 1)/2 \approx 1.25 \cdot 10^7$ pasangan per kasus uji, dan satu berkas uji memuat beberapa kasus uji. Menghitung setiap bobot, menyimpan setiap pasangan, lalu mengurutkannya saja sudah terlalu lambat sebelum pekerjaan sesungguhnya dimulai.

Pola yang sama bersembunyi dalam banyak tugas praktis: menggabungkan pemesanan yang jadwalnya bertumpuk menjadi blok-blok, mengelompokkan petak sawah yang saluran irigasinya bersilangan, atau mengelompokkan lokasi toko yang berbagi satu ruas jalan. Setiap kali “jarak” antara dua objek hanya mengambil sedikit nilai, dan nilai itu berasal dari geometri, biasanya ada cara untuk tidak memeriksa semua pasangan.

Rencana bab ini mengikuti rencana penyelesaiannya. Subbab 7.2 membangun tiga alat dari nol: gabungan himpunan saling lepas, algoritma Kruskal yang dipandang sebagai barisan kelas bobot, dan sapuan selang. Subbab 7.3 menjalankan algoritma Kruskal pada graf kecil agar setiap

keputusannya dapat diperiksa dengan tangan. Subbab 7.4 memadukan alat-alat itu pada VILUNIT, lengkap dengan penelusuran sebuah contoh, program utuh, dan tinjauan jujur atas waktu jalannya. Subbab 7.5 mengujinya terhadap definisi yang dituliskan sebagai *brute force*, dan Subbab 7.6 memperlihatkan di mana lagi penalaran yang sama berlaku. Setiap alat itu pendek; pelajarannya terletak pada cara alat-alat itu saling mengunci menjadi satu penyelesaian yang utuh dan dapat dipercaya.

7.2 Alat-Alatnya dari Nol

7.2.1 Gabungan himpunan saling lepas dalam lima baris

Disjoint-set union (DSU, disebut juga *union-find*) memelihara partisi objek $0, 1, \dots, n - 1$ ke dalam kelompok-kelompok. Struktur ini mendukung dua operasi: `find(x)` mengembalikan wakil dari kelompok yang memuat x , dan `unite(a, b)` menggabungkan kelompok a dan kelompok b . Representasinya berupa hutan: setiap objek menyimpan induk `par[x]`, dan akar (objek dengan `par[x] == x`) mewakili pohonnya.

DEFINISI. *Path compression* (pemampatan lintasan) membuat setiap simpul yang dilewati `find` menunjuk langsung ke akar. *Union by rank* (atau *by size*) selalu menempelkan pohon yang lebih kecil di bawah pohon yang lebih besar. Dengan keduanya, barisan m operasi atas n objek memerlukan waktu $O(m \alpha(n))$, dengan α fungsi invers Ackermann, yang nilainya paling besar 4 untuk setiap masukan yang muat di alam semesta [2], [3].

Dalam kode lomba, *path compression* saja sudah lazim: kasus terburuknya $O(\log n)$ teramortisasi per operasi, dan dalam praktik tidak dapat dibedakan dari versi lengkapnya. Lima baris yang dipakai di sepanjang bab ini adalah

```
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; }
```

Hasil Boolean dari `unite` adalah kunci algoritma Kruskal: `true` berarti dua kelompok berbeda telah digabung (sisinya berguna), `false` berarti kedua objek sudah terhubung (sisi itu akan menutup siklus). Menghitung kelompok di akhir sama mudahnya: hitung objek i dengan `par[i] == i`.

7.2.2 Algoritma Kruskal dan sifat potongan

Algoritma Kruskal [4] memeriksa sisi-sisi menurut bobot yang menaik dan mempertahankan sebuah sisi tepat ketika sisi itu menghubungkan dua kelompok berbeda. Mengapa pilihan rakus (*greedy*) ini menghasilkan pohon perentang minimum? Argumen bakunya memakai *sifat potongan* (*cut property*): untuk setiap pembagian simpul menjadi dua sisi, sisi termurah yang menyeberang di antara kedua bagian termasuk dalam suatu pohon perentang minimum [3]. Ketika algoritma Kruskal menerima sisi (u, v) , ambillah satu bagian sebagai kelompok u saat itu. Tidak ada sisi yang lebih murah yang menyeberangi potongan ini, sebab sisi menyeberang yang lebih murah pasti sudah diperiksa lebih dulu dan diterima, sehingga kedua kelompok sudah bergabung. Jadi sisi itu aman, dan dengan induksi setiap sisi yang diterima menjaga himpunan terpilih tetap berada di dalam suatu pohon perentang minimum.

Sekarang perhatikan apa yang sebenarnya diperlukan argumen itu. Argumen itu memerlukan sisi yang lebih murah datang sebelum sisi yang lebih mahal. Argumen itu *tidak* peduli urutan pemeriksaan sisi-sisi yang *sama* bobotnya. Lebih jauh, banyaknya sisi yang diterima dalam satu kelas bobot sudah tertentu: jika ada k_{w-1} kelompok setelah semua sisi yang lebih ringan dari w

dan k_w kelompok setelah sisi berbobot w juga diproses, maka tepat $k_{w-1} - k_w$ sisi berbobot w diterima, sisi mana pun itu. Karena itu total bobotnya adalah

$$\text{MST} = \sum_w w \cdot (k_{w-1} - k_w) \quad (7.1)$$

dengan k_w banyaknya komponen terhubung dari graf yang hanya mempertahankan sisi berbobot paling besar w . [Persamaan \(7.1\)](#) mengubah pohon perentang minimum menjadi pertanyaan tentang *keterhubungan*, satu kelas bobot demi satu kelas bobot. Untuk bobot 0, 1, dan 2 pada graf lengkap (setiap pasangan memiliki suatu sisi) persamaan itu menjadi

$$\text{MST} = 1 \cdot (k_0 - k_1) + 2 \cdot (k_1 - 1) \quad (7.2)$$

karena pada akhirnya semuanya terhubung, sehingga $k_2 = 1$. Kelas bobot 0 tidak menyumbang biaya, tetapi tetap penting: kelas itulah yang menentukan k_0 .

7.2.3 Menyapu selang

Jadi yang kita perlukan adalah keterhubungan, bukan daftar sisi. Untuk selang-selang pada garis, hal ini mudah. Sebut dua selang *bertumpuk* bila keduanya berbagi potongan yang panjangnya positif. Urutkan selang menurut ujung kirinya dan telusuri satu per satu sambil mengingat `cur_max`, ujung kanan terjauh yang sejauh ini terlihat. Selang berikutnya $[l, r]$ bertumpuk dengan suatu selang sebelumnya tepat ketika $l < \text{cur_max}$; dan jika demikian, selang itu bertumpuk dengan selang pemilik `cur_max`, karena selang tersebut dimulai tidak lebih lambat dari l dan berakhir sesudah l . Jika $l \geq \text{cur_max}$, setiap selang sebelumnya berakhir pada atau sebelum l , dan sebuah kelompok baru dimulai.

WAWASAN KUNCI: Tumpukan menjalar melalui rantai. Jika A bertumpuk dengan B dan B bertumpuk dengan C , ketiganya berada dalam satu kelompok walaupun A dan C berjauhan. Menggabungkan setiap selang dengan *wakil* yang memegang `cur_max` membangun kelompok-kelompok ini dengan paling banyak $n - 1$ penggabungan, alih-alih memeriksa n^2 pasangan.

Kode 7.1 memperlihatkan sapuan pada tujuh selang.

Kode 7.1 `interval_groups.cpp`: mengelompokkan selang dengan sapuan ujung kanan terjauh.

```
// Sweep: group intervals that overlap directly or through a chain.
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<pair<int, int>> iv = {{1, 4}, {9, 12}, {3, 6}, {14, 15},
                                {5, 8}, {11, 13}, {16, 18}};
    sort(iv.begin(), iv.end()); // by Left end
    int groups = 0, cur_max = -1000000000;
    for (auto [l, r] : iv) {
        if (l >= cur_max) groups++; // starts after everything seen: new group
        cur_max = max(cur_max, r); // farthest right end so far
        cout << "[" << l << ", " << r << "]" -> group " << groups << "\n";
    }
    cout << groups << " groups\n";
}
```

```
[1,4] -> group 1  
[3,6] -> group 1  
[5,8] -> group 1  
[9,12] -> group 2  
[11,13] -> group 2  
[14,15] -> group 3  
[16,18] -> group 4  
4 groups
```

[1,4] dan [5,8] tidak bertumpuk, tetapi keduanya berada di kelompok 1 melalui [3,6]. Patut ditanyakan mengapa program mengingat selang dengan *ujung kanan terjauh*, bukan sekadar selang sebelumnya. Perhatikan [0,10], [1,2], dan [5,6] dengan urutan itu. Selang ketiga tidak bertumpuk dengan selang kedua, yang berakhir di 2, tetapi bertumpuk dengan selang pertama yang menjangkau 10. Sapuan yang hanya membandingkan setiap selang dengan pendahulunya akan memulai kelompok baru di [5,6] dan keliru. Ujung kanan terjauh meringkas seluruh masa lalu dalam satu bilangan: jika selang baru dimulai sebelum titik itu, ada tumpukan dengan selang pemiliknya; jika tidak, tidak ada tumpukan dengan apa pun yang sudah terlihat. Satu bilangan itulah yang membuat sapuan berjalan linear setelah pengurutan. Syaratnya ketat: [14,15] dan [16,18] tetap terpisah walaupun selang kedua dimulai di 15, karena bersentuhan pada satu titik bukanlah tumpukan dengan panjang positif.

7.2.4 Mengapa DSU cukup cepat

Ada gunanya melihat mengapa *path compression* saja sudah cepat, karena argumennya juga memberi tahu kapan ia *tidak* cepat. Tanpa pemampatan, *find* berjalan dari simpul ke akarnya, dan barisan penggabungan yang ceroboh dapat membangun satu rantai sepanjang n ; maka setiap *find* berbiaya n langkah dan n operasi berbiaya n^2 . Dengan pemampatan, *find* pertama pada rantai itu memang berbiaya n langkah, tetapi setelahnya setiap simpul yang dilewati menunjuk langsung ke akar, sehingga *find* berikutnya pada simpul mana pun di antaranya hanya berbiaya satu langkah. Operasi mahal membayar agar operasi sesudahnya murah. Perhitungan cermat atas gagasan ini memberikan batas teramortisasi $O(\log n)$ per operasi untuk pemampatan saja, dan batas $O(\alpha(n))$ yang terkenal bila *union by rank* ditambahkan [2].

Untuk ukuran lomba, perbedaannya tidak terlihat: dengan $n = 5000$, bahkan perkiraan pesimistis $\log_2 n \approx 13$ langkah per operasi tidak berarti apa-apa. Satu-satunya tempat pemampatan saja dapat merugikan adalah *rekursi* yang dalam: *find* rekursif pada Subbab 7.2.1 memanggil dirinya sekali per simpul di lintasan, dan rantai berisi 10^6 simpul yang terbentuk dari penggabungan yang sial dapat meluapkan *stack* pada pemampatan pertamanya. Bila n sebesar itu, tambahkan *union by size* (beberapa baris lagi) atau tulis *find* dengan perulangan. Untuk VILUNIT, dengan $n \leq 5000$, kelima baris itu sempurna.

Sifat kedua dipakai diam-diam di sepanjang bab ini: DSU tidak pernah *memecah* kelompok. Sekali dua objek terhubung, keduanya tetap terhubung. Itulah perilaku yang dibutuhkan algoritma Kruskal, dan itu pula sebabnya DSU tidak dapat menjawab pertanyaan tentang penghapusan sisi; soal yang menghapus sisi biasanya diselesaikan dengan memproses operasi secara mundur, sehingga penghapusan berubah menjadi penyisipan.

7.3 Pemanasan: Kruskal pada Graf Kecil

Sebelum menggabungkan potongan-potongannya, jalankan algoritma klasiknya sekali. **Kode 7.2** mengurutkan sembilan sisi sebuah graf berisi enam simpul dan melaporkan setiap keputusannya.

Kode 7.2 *kruskal1.cpp*: algoritma Kruskal dengan DSU lima baris.

```
// Kruskal with a five-line DSU on a small weighted graph.
#include <algorithm>
#include <iostream>
#include <tuple>
#include <vector>
using namespace std;
int par[6];
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) return false; // already connected: edge would close a cycle
    par[a] = b;
    return true;
}
int main() {
    vector<tuple<int, int, int>> e = {{4, 0, 1}, {1, 0, 2}, {3, 1, 2}, {2, 1, 3},
                                     {5, 2, 3}, {7, 3, 4}, {6, 2, 4}, {2, 4, 5},
                                     {8, 3, 5}};

    sort(e.begin(), e.end());
    for (int i = 0; i < 6; i++) par[i] = i;
    int total = 0;
    for (auto [w, u, v] : e) {
        bool used = unite(u, v);
        cout << "edge " << u << "-" << v << " w=" << w
              << (used ? " take" : " skip (cycle)") << "\n";
        if (used) total += w;
    }
    cout << "MST weight = " << total << "\n";
}
```

```
edge 0-2 w=1 take
edge 1-3 w=2 take
edge 4-5 w=2 take
edge 1-2 w=3 take
edge 0-1 w=4 skip (cycle)
edge 2-3 w=5 skip (cycle)
edge 2-4 w=6 take
edge 3-4 w=7 skip (cycle)
edge 3-5 w=8 skip (cycle)
MST weight = 14
```

Lima sisi diambil untuk enam simpul, seperti yang dituntut sebuah pohon perentang. Periksa [Persamaan \(7.1\)](#) terhadap keluarannya: setelah sisi berbobot 1 ada 5 kelompok, setelah sisi-sisi berbobot 2 ada 3, setelah bobot 3 ada dua, dan setelah bobot 6 tinggal satu. Jumlah $1 \cdot (6 - 5) + 2 \cdot (5 - 3) + 3 \cdot (3 - 2) + 6 \cdot (2 - 1) = 1 + 4 + 3 + 6 = 14$ cocok.

7.4 Studi Kasus: VILUNIT

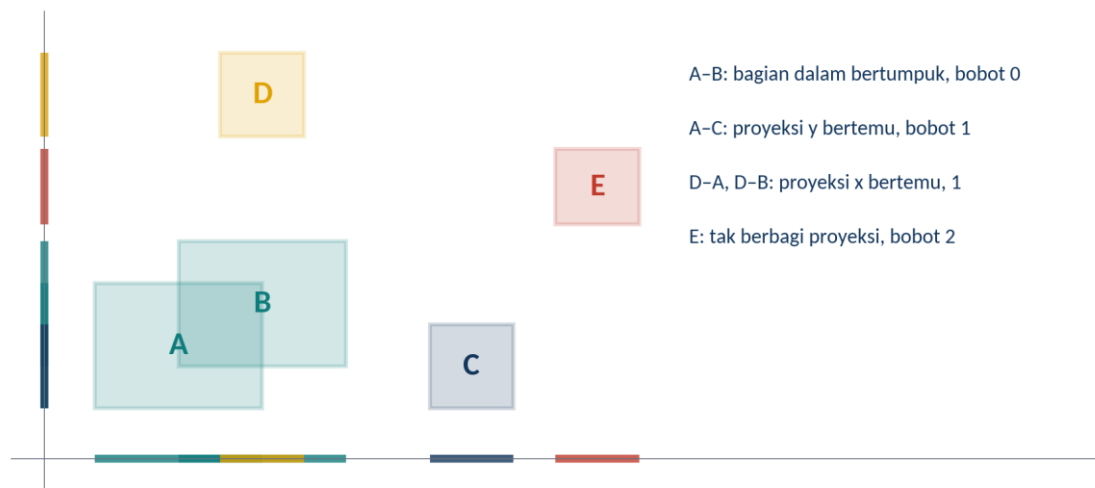
7.4.1 Soal dengan kata-kata kita sendiri

Soalnya adalah SPOJ VILUNIT [1]. Masukan diawali banyaknya kasus uji. Setiap kasus uji memberikan n ($n \leq 5000$) lalu n desa, masing-masing berupa empat bilangan bulat x_1, y_1, x_2, y_2 : dua sudut berseberangan dari sebuah persegi panjang yang sejajar sumbu, dalam urutan mana pun. Untuk dua desa, biaya menghubungkannya adalah

- 0 jika proyeksi-x keduanya bertumpuk *dan* proyeksi-y keduanya bertumpuk (bagian dalamnya bertumpuk);
- 1 jika tepat satu dari kedua proyeksi bertumpuk;
- 2 untuk selainnya.

Di sini “bertumpuk” berarti berbagi segmen dengan panjang positif, seperti pada Subbab 7.2.3. Jawaban untuk satu kasus uji adalah total biaya minimum dari himpunan sambungan yang menghubungkan semua desa, yaitu bobot pohon perentang minimum. Untuk $n \leq 1$ jawabannya 0.

Gambar 7.1 memperlihatkan lima desa beserta bayangannya pada kedua sumbu. A dan B bertumpuk, sehingga menghubungkannya gratis. C berbagi jalur-y dengan A , D berbagi jalur-x dengan A dan B , sehingga masing-masing sambungan itu berbiaya 1. E tidak berbagi jalur dengan siapa pun dan memerlukan sambungan berbiaya 2. Menurut [Persamaan \(7.2\)](#), $k_0 = 4$ ($\{A, B\}, C, D, E$), $k_1 = 2$, dan jawabannya $1 \cdot 2 + 2 \cdot 1 = 4$.



Gambar 7.1 Lima desa dan proyeksinya (batang berwarna pada sumbu); biaya sambungan 0, 1, dan 2 ditentukan oleh proyeksi mana yang bertumpuk.

7.4.2 Contoh masukan dan keluaran

```
2
3
0 0 4 4
2 2 6 6
10 0 12 2
4
0 0 2 2
5 5 7 7
1 5 3 9
9 1 11 3
```

```
1
3
```

Pada uji pertama, dua persegi pertama bertumpuk (biaya 0) dan persegi ketiga berbagi jalur-y dari 0 sampai 2 dengan persegi pertama (biaya 1): total 1. Pada uji kedua tidak ada bagian dalam yang bertumpuk. Desa ketiga berbagi jalur-x dengan desa pertama ($[1,3]$ terhadap $[0,2]$) dan jalur-y dengan desa kedua ($[5,9]$ terhadap $[5,7]$); desa keempat berbagi jalur-y dengan desa pertama ($[1,3]$ terhadap $[0,2]$). Tiga sambungan berbiaya 1 menghubungkan keempatnya: total 3.

7.4.3 Merancang penyelesaian

[Persamaan \(7.2\)](#) menyatakan bahwa kita hanya memerlukan dua bilangan: k_0 , banyaknya kelompok yang terbentuk oleh sambungan gratis, dan k_1 , banyaknya kelompok yang terbentuk oleh sambungan berbiaya paling banyak 1.

1. **Normalkan** setiap persegi panjang sehingga $x_1 \leq x_2$ dan $y_1 \leq y_2$.

2. **Lintasan 0 (bobot 0).** Urutkan menurut x_1 . Untuk setiap desa u , pindai desa-desa v berikutnya dalam urutan ini selama $x_1(v) < x_2(u)$; itulah tepat desa-desa yang proyeksi-x-nya bertumpuk dengan milik u dari kanan. Jika proyeksi-y keduanya juga bertumpuk, gabungkan.
3. **Bobot 1, sapuan-x.** Telusuri desa-desa menurut urutan x_1 dengan `cur_max` dan pemiliknya `rep` seperti pada Subbab 7.2.3; bila $x_1(u) < \text{cur_max}$, gabungkan u dengan `rep`, dan jika penggabungan berhasil tambahkan 1 ke biaya.
4. **Bobot 1, sapuan-y.** Sama, dengan y_1, y_2 , dan urutan terurut yang kedua.
5. **Bobot 2.** Hitung kelompok yang tersisa, k_1 , dan tambahkan $2(k_1 - 1)$.

Setiap penggabungan yang berhasil pada langkah 3 atau 4 menyatukan dua kelompok berbeda melalui sambungan nyata berbiaya 1: sambungan itu ada karena proyeksinya bertumpuk, dan biayanya bukan 0, karena dua desa yang bagian dalamnya bertumpuk sudah digabung pada langkah 2. Jadi banyaknya penggabungan yang berhasil adalah $k_0 - k_1$, seperti yang dituntut [Persamaan \(7.2\)](#), *asalkan* kedua sapuan bersama-sama menemukan keterhubungan penuh dari graf berbobot ≤ 1 . Dan memang demikian: sapuan-x menggabungkan setiap kelompok dari graf tumpukan-x, sapuan-y setiap kelompok dari graf tumpukan-y, dan DSU memadukan keduanya, yang tidak lain adalah keterhubungan pada gabungan kedua graf.

Pertanyaan yang wajar adalah apakah semua kecerdikan ini memang diperlukan. Untuk graf *padat*, ketika setiap pasangan simpul memiliki sisi, alternatif buku teksnya adalah algoritma Prim dalam bentuk larik: simpan untuk setiap simpul di luar pohon sambungan termurah ke pohon, tambahkan simpul dengan sambungan termurah, lalu perbarui yang lain dalam satu lintasan. Biayanya $O(n^2)$ waktu dan $O(n)$ memori, dan cara ini tidak pernah menyimpan daftar sisi, karena setiap bobot dihitung dari koordinat saat diperlukan [3]. Untuk $n = 5000$ ini berarti $2.5 \cdot 10^7$ evaluasi bobot per kasus uji, layak untuk beberapa kasus uji tetapi menjadi berisiko bila satu berkas memuat banyak kasus besar. Metode kelas bobot lebih cepat pada kasus umum dan, yang lebih penting, gagasannya dapat diperbesar: dengan $n = 10^5$ desa, bentuk larik algoritma Prim akan memerlukan 10^{10} langkah, sedangkan sapuan masih berjalan dalam sepersekian detik bila lintasan 0 diganti dengan struktur garis sapu. Dalam lomba, menguasai keduanya adalah kekuatan yang sesungguhnya: bentuk larik algoritma Prim adalah *brute force* lima menit yang bahkan dapat menjadi pasangan uji stres, dan sapuan adalah penyelesaian yang bertahan pada uji terbesar.

7.4.4 Menelusuri contoh kedua dengan tangan

Sebelum membaca kodenya, telusuri rancangan ini pada uji contoh kedua, yang keempat desanya setelah dinormalkan adalah $V_0 = [0,2] \times [0,2]$, $V_1 = [5,7] \times [5,7]$, $V_2 = [1,3] \times [5,9]$, dan $V_3 = [9,11] \times [1,3]$.

Lintasan 0. Menurut urutan x_1 desanya adalah V_0 (0), V_2 (1), V_1 (5), V_3 (9). Untuk V_0 pemindaian melihat V_2 , karena $1 < 2$; proyeksi-y keduanya $[0,2]$ dan $[5,9]$ tidak bertumpuk, jadi tidak terjadi apa-apa, dan pemindaian berhenti di V_1 karena $5 \geq 2$. Desa-desa lain tidak menemukan kandidat. Tidak ada bagian dalam yang bertumpuk, jadi $k_0 = 4$.

Sapuan-x. V_0 menetapkan `cur_max` menjadi 2 dengan wakil V_0 . V_2 dimulai di $1 < 2$: gabungkan V_2 dengan V_0 , biaya 1; `cur_max` menjadi 3 dengan wakil V_2 . V_1 dimulai di $5 \geq 3$: kelompok baru; `cur_max` menjadi 7. V_3 dimulai di $9 \geq 7$: kelompok baru lagi.

Sapuan-y. Menurut urutan y_1 : V_0 (0), V_3 (1), lalu V_1 dan V_2 (keduanya 5). V_0 menetapkan `cur_max` menjadi 2. V_3 dimulai di $1 < 2$: gabungkan dengan V_0 , biaya 2; `cur_max` menjadi 3. Desa berikutnya dimulai di $5 \geq 3$ dan hanya menaikkan `cur_max`; desa terakhir dimulai di 5, di bawah

`cur_max` yang baru, dan bergabung: biaya 3. Desa mana pun di antara V_1 dan V_2 yang datang lebih dulu setelah pengurutan, hasilnya sama.

Putaran kedua dari perulangan `changed` tidak menemukan hal baru, tersisa satu kelompok, dan jawabannya $3 + 2 \cdot (1 - 1) = 3$, sesuai harapan. Menelusuri contoh seperti ini memakan lima menit dan merupakan perlindungan terbaik terhadap sapuan yang berjalan pada larik indeks yang salah.

7.4.5 Penyelesaian lengkap

Kode 7.3 adalah program lengkap yang tercatat *Accepted* di SPOJ. Blok P50 memuat kedua sapuan bobot 1.

Kode 7.3 `VILUNIT.cpp`: Kruskal per kelas bobot, dengan lintasan 0 dan dua sapuan.

```
#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];
static int 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();
```

Kode 7.3 (Lanjutan) *VILUNIT.cpp: Kruskal per kelas bobot, dengan lintasan 0 dan dua sapuan.*

```
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];
});
// Pass 0: Weight 0 (Overlapping Interiors)
for (int i = 0; i < n; i++) {
    int u = idxX[i];
    for (int j = i + 1; j < n; j++) {
        int v = idxX[j];
        if (X1[v] >= X2[u]) break;
        if (Y1[v] < Y2[u] && Y2[v] > Y1[u]) unite(u, v);
    }
}
long long cost = 0;
bool changed = true;
while (changed) {
    changed = false;
    // <P50>
    // Weight-1 sweeps: walk idxX (then idxY) keeping the farthest right
    // end cur_max and its owner rep; a box starting before cur_max joins
    // rep, and each successful unite adds 1 to cost and sets changed.
    // X-Sweep (Weight 1)
    // >>> YOUR TURN (Public Edition): write this part yourself, following the hint
    // above. <<<
```

Kode 7.3 (Lanjutan) *VILUNIT.cpp: Kruskal per kelas bobot, dengan lintasan 0 dan dua sapuan.*

```
// </P50>
}
int comps = 0;
for (int i = 0; i < n; i++) if (par[i] == i) comps++;
printf("%lld\n", cost + 2LL * (comps - 1));
}
return 0;
}
```

7.4.6 Baris demi baris

Masukan cepat. Masukannya bisa besar: banyak kasus uji dengan hingga 5000 baris masing-masing. `readInt` membaca seluruh masukan dalam blok satu megabita dengan `fread` dan

mengurai digit secara manual, beberapa kali lebih cepat daripada `scanf`. Fungsi ini melewati karakter bukan digit, mengingat tanda minus, dan mengisi ulang `buffer` setiap kali habis.

DSU. `find` dan `unite` adalah lima baris dari Subbab 7.2.1, bekerja pada larik global `par`.

Kasus kecil. Bila $n \leq 1$, program membaca satu desa itu (jika ada) dan langsung mencetak 0. Tanpa ini, $n = 0$ akan sampai pada rumus penutup dengan nol kelompok dan mencetak -2 .

Membaca dan menormalkan. Setiap persegi panjang dibaca, sudut-sudutnya ditukar bila perlu, `par[i] = i` menjadikan setiap desa kelompoknya sendiri, dan dua larik indeks disiapkan untuk diurutkan. Mengurutkan `indeks`, bukan persegi panjangnya sendiri, membuat keempat larik koordinat tidak tersentuh, sehingga urutan- x dan urutan- y dapat hidup berdampingan.

Lintasan 0. Perulangan ganda mengimplementasikan langkah 2 rancangan. `break` ketika $x_1(v) \geq x_2(u)$ benar karena urutannya terurut menurut x_1 : setiap desa berikutnya dimulai lebih ke kanan lagi. Uji- y `Y1[v] < Y2[u] && Y2[v] > Y1[u]` adalah uji tumpukan baku untuk dua selang.

Sapuan (blok P50). Perulangan `while (changed)` mengulang sapuan- x dan sapuan- y sampai keduanya tidak lagi melakukan penggabungan yang berhasil. Di dalamnya, setiap sapuan adalah sapuan selang dari **Kode 7.1**, dengan `unite(u, rep)` menggantikan pencacah kelompok dan `cost++` untuk setiap penggabungan yang berhasil. Nilai awal $-2 \cdot 10^9$ berada di bawah setiap koordinat.

Jawaban. Baris-baris terakhir menghitung akar, yaitu k_1 , dan mencetak `cost + 2LL * (comps - 1)`, yaitu **Persamaan (7.2)**.

7.4.7 Kompleksitas, secara jujur

Kedua pengurutan berbiaya $O(n \log n)$ dan setiap sapuan $O(n \alpha(n))$. Lintasan 0 berbeda: perulangan dalamnya berjalan atas semua desa yang bertumpuk dengan u sepanjang sumbu- x , dan bila semua desa panjang dan ramping ke arah x , itu berarti semua desa, sehingga kasus terburuknya $O(n^2)$, sekitar $1.25 \cdot 10^7$ perbandingan murah untuk $n = 5000$. Kami mengukur jenis masukan terburuk, dua puluh kasus uji berisi 5000 persegi panjang acak raksasa yang semuanya bertumpuk, dan program selesai dalam sekitar 1,2 detik, jadi lintasan kuadratik itu masih terjangkau pada ukuran ini. Perulangan `changed` menjalankan badannya paling banyak dua kali dalam praktik; Soal Latihan 2 menunjukkan bahwa satu putaran selalu cukup.

7.4.8 Jebakan

JEBAKAN: Menghitung setiap penggabungan sebagai biaya 1 *sebelum* lintasan 0 dijalankan. Jika sapuan bobot 1 berjalan lebih dulu, sapuan itu dapat menggabungkan dua desa yang bagian dalamnya bertumpuk dan menagih 1 untuk sambungan yang seharusnya gratis. Urutan Kruskal, kelas termurah lebih dulu, bukanlah pilihan.

- **Bersentuhan bukan bertumpuk.** Setiap perbandingan ketat (`<`, `>`). Memakai `<=` mengubah maknanya dan jawabannya.
- **Sudut dalam urutan sembarang.** Tanpa penukaran, persegi panjang yang diberikan dari kanan atas ke kiri bawah menjadi “selang” berpanjang negatif dan diam-diam merusak kedua lintasan.
- **Melupakan $n = 0$.** Rumus $2(k_1 - 1)$ mengandaikan paling sedikit satu kelompok.
- **Luapan.** Biaya dapat mencapai $2(n - 1)$, kecil di sini, tetapi program tetap memakai `long long` untuk jumlahnya; kebiasaan yang layak dipertahankan.

7.5 Uji Stres terhadap *Brute Force*

Brute force pada **Kode 7.4** adalah definisinya yang dituliskan apa adanya: hitung semua $n(n-1)/2$ bobot langsung dari proyeksi, urutkan, lalu jalankan Kruskal. Program ini tidak berbagi apa pun dengan sapuan, sehingga menjadi hakim yang dapat dipercaya.

Kode 7.4 `brute.cpp`: semua pasangan, bobot 0, 1, 2 yang eksplisit, dan Kruskal.

```
// Reference for VILUNIT: all  $n(n-1)/2$  pairs with weights 0/1/2, then Kruskal.
#include <algorithm>
#include <cstdio>
#include <numeric>
#include <tuple>
#include <vector>
using namespace std;
vector<int> par;
int find(int x) { return par[x] == x ? x : par[x] = find(par[x]); }
int main() {
    int T;
    if (scanf("%d", &T) != 1) return 0;
    while (T--) {
        int n;
        if (scanf("%d", &n) != 1) return 0;
        vector<int> a(n), b(n), c(n), d(n);
        for (int i = 0; i < n; i++) {
            if (scanf("%d%d%d%d", &a[i], &b[i], &c[i], &d[i]) != 4) return 0;
            if (a[i] > c[i]) swap(a[i], c[i]);
            if (b[i] > d[i]) swap(b[i], d[i]);
        }
        vector<tuple<int, int, int>> e;
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {
                bool ox = a[i] < c[j] && a[j] < c[i]; // x-projections share length
                bool oy = b[i] < d[j] && b[j] < d[i]; // y-projections share length
                e.emplace_back(ox && oy ? 0 : (ox || oy ? 1 : 2), i, j);
            }
        sort(e.begin(), e.end());
        par.assign(n, 0);
        iota(par.begin(), par.end(), 0);
        long long cost = 0;
        for (auto [w, i, j] : e)
            if (find(i) != find(j)) par[find(i)] = find(j), cost += w;
        printf("%lld\n", cost);
    }
}
```

Pembangkit **Kode 7.5** menarik hingga sembilan persegi panjang dengan lebar dan tinggi positif di dalam persegi kecil, sehingga tumpukan, jalur bersama, dan tepi yang bersentuhan sering terjadi. Rentang koordinatnya dipilih acak dari 6, 12, dan 30 untuk memvariasikan kepadatannya.

Kode 7.5 `gen.py`: desa-desa acak kecil untuk VILUNIT.

```
# Random small VILUNIT inputs (boxes with positive area): python3 gen.py SEED
import random, sys
random.seed(int(sys.argv[1]))
T = random.randint(1, 5)
print(T)
for _ in range(T):
    n = random.randint(1, 9)
    L = random.choice([6, 12, 30])
    print(n)
    for _ in range(n):
        x1, x2 = random.sample(range(L), 2)
        y1, y2 = random.sample(range(L), 2)
        print(x1, y1, x2, y2)
```

Menjalankan `stress.sh` yang sama seperti pada Bab 6 dengan nama soal VILUNIT memberikan

```
all 300 random tests passed
```

7.6 Memakai Gagasan Ini di Luar VILUNIT

Gabungan “sedikit kelas bobot, keterhubungan per kelas” muncul dalam banyak samaran. Tiga di antaranya layak diketahui.

Ambang, bukan kelas. “Berapa L terkecil sehingga semua objek terhubung hanya dengan sambungan berbiaya paling banyak L ?” juga dijawab oleh algoritma Kruskal: L adalah bobot sisi terakhir yang diterima, yaitu *bottleneck* (leher botol) pohon perentang minimum. Bila ambang kandidatnya sedikit, jalankan uji keterhubungan sekali per ambang, seperti yang dilakukan sapuan untuk ambang 1.

Tetangga terdekat pada garis. Bila objek berupa titik pada garis dan biaya sambungan adalah jaraknya, hanya titik yang bertetangga dalam urutan terurut yang dapat berguna: sambungan yang melompati sebuah titik dapat diganti dengan dua sambungan yang lebih pendek melalui titik itu. Jadi garis berisi n titik hanya memiliki $n - 1$ sisi kandidat, dan pohon perentang minimumnya sekadar jumlah celah-celahnya. Penalaran “urutkan dulu, lalu hanya tetangga yang penting” yang sama itulah yang membuat sapuan bekerja.

Jarak Manhattan pada bidang. Untuk titik-titik dengan biaya $|x_1 - x_2| + |y_1 - y_2|$, sebuah teorema klasik menunjukkan bahwa setiap titik hanya perlu disambungkan ke tetangga terdekatnya di masing-masing dari delapan daerah bersudut, sehingga diperoleh $O(n)$ sisi kandidat yang dapat ditemukan dengan sapuan berbantuan *Fenwick tree* dalam $O(n \log n)$ [5]. Rinciannya lebih panjang, tetapi falsafahnya sama dengan bab ini: jangan pernah mendaftar semua pasangan; biarkan geometri memilih sisi yang penting, dan biarkan DSU mengerjakan sisanya.

Dalam setiap kasus, titik tolaknya adalah satu kalimat dari Subbab 7.2.2: algoritma Kruskal hanya memerlukan *urutan* bobot dan *keterhubungan* di dalam setiap kelas. Apa pun yang menyediakan dua hal ini, tanpa daftar semua pasangan, adalah implementasi yang sah.

7.7 Taktik pada Hari Lomba

- **Sedikit bobot berbeda? Berpikirlah per kelas.** Bila soal MST memiliki bobot dari himpunan yang sangat kecil, tuliskan *Persamaan* (7.1) di kertas dan tanyakan cara menghitung banyaknya komponen per kelas, bukan cara mendaftar sisi.

- **Cari geometri yang dapat diurutkan.** Tumpukan selang, keterjangkauan pada garis, dan “elemen lebih besar berikutnya” semuanya diselesaikan oleh satu lintasan terurut. Dua sumbu biasanya berarti dua lintasan.
- **Hafalkan DSU di ujung jari.** DSU adalah struktur yang paling sering dipakai ulang dalam soal graf olimpiade [5]; mengetiknya harus kurang dari satu menit.
- **Uji masukan yang merosot.** $n = 0$, $n = 1$, semua desa identik, semua desa dalam satu baris. Di sinilah rumus yang mengandung “ -1 ” patah.
- **Ukur kasus terburuk yang Anda curigai.** Jika sebuah perulangan kuadratik secara teori, bangun masukan itu dan ukur waktunya, seperti pada Subbab 7.4.7. Mengetahui biaya sebenarnya lebih baik daripada berharap.
- **Siapkan *brute force* untuk graf padat.** Algoritma Prim dalam bentuk larik, dengan bobot yang dihitung saat diperlukan, adalah pendapat kedua yang dapat dipercaya untuk pohon perentang minimum atas beberapa ribu objek, dan sama sekali tidak memerlukan daftar sisi.

7.8 Ringkasan Bab

- DSU dengan *path compression* menjawab “apakah keduanya terhubung?” dan “gabungkan keduanya” dalam waktu nyaris konstan; **unite** yang mengembalikan Boolean adalah keputusan Kruskal.
- Algoritma Kruskal hanya memerlukan kelas bobot dalam urutan menaik; di dalam satu kelas, urutan mana pun memberikan banyak sisi diterima yang sama, yang menghasilkan **Persamaan (7.1)**.
- Sapuan terurut dengan wakil pemilik ujung kanan terjauh menemukan keterhubungan selang dengan paling banyak $n - 1$ penggabungan.
- VILUNIT memadukan ketiganya: sambungan gratis pada lintasan 0, sambungan berbiaya 1 pada dua sapuan, dan sisanya dengan aritmetika, seperti pada **Persamaan (7.2)**.

7.9 Soal Latihan

1. Jalankan sapuan selang **Kode 7.1** dengan tangan pada [2,5], [1,3], [6,7], [4,6]. Ada berapa kelompok?
2. Buktikan atau bantah: pada **Kode 7.3** satu putaran sapuan-x yang diikuti sapuan-y selalu cukup, sehingga perulangan **changed** dapat dihapus.
3. Misalkan biaya sambungan adalah 0, 1, dan 5, bukan 0, 1, dan 2. Satu baris program mana yang berubah?
4. Hapus kedua baris **swap** dari **Kode 7.3**. Tentukan jawaban yang benar untuk desa-desa **4 2 2 1, 7 7 3 6**, dan **0 3 7 6**, lalu jelaskan mengapa program yang telah diubah itu salah.
5. Tambahkan *union by size* pada DSU di **Kode 7.2**. Apa yang berubah pada keluarannya?
6. Sebuah kasus uji berisi $n = 5000$ desa yang tidak ada dua di antaranya berbagi proyeksi apa pun. Apa yang dicetak program, dan seberapa banyak pekerjaan lintasan 0?

7.10 Penyelesaian Soal Latihan

1. Setelah diurutkan: [1,3], [2,5], [4,6], [6,7]. [1,3] memulai kelompok 1, **cur_max** menjadi 3. [2,5]: $2 < 3$, kelompok yang sama, **cur_max** 5. [4,6]: $4 < 5$, kelompok yang sama, **cur_max** 6. [6,7]: $6 < 6$ salah, kelompok baru. Dua kelompok.

2. Benar. Setelah sapuan- x , setiap pasangan desa yang proyeksi- x -nya bertumpuk berada dalam satu kelompok, karena sapuan membangun kelompok-kelompok terhubung dari graf tumpukan- x (Subbab 7.2.3). Setelah sapuan- y hal yang sama berlaku untuk graf tumpukan- y . Kelompok hanya bergabung, tidak pernah pecah, sehingga setelah kedua sapuan setiap sisi dari gabungan kedua graf menghubungkan dua objek dalam kelompok yang sama; artinya kelompok-kelompok DSU tepat sama dengan komponen graf berbobot ≤ 1 . Putaran kedua tidak dapat menemukan apa pun yang baru; putaran itu hanya berfungsi sebagai jaring pengaman.
3. Hanya pernyataan cetak terakhir: `cost + 5LL * (comps - 1)`. Kelas-kelas tetap diproses dalam urutan menaik, dan **Persamaan** (7.1) dengan $w = 5$ memberikan suku itu.
4. Setelah dinormalkan, desanya adalah $A = [2,4] \times [1,2]$, $B = [3,7] \times [6,7]$, dan $C = [0,7] \times [3,6]$. Tidak ada bagian dalam yang bertumpuk. A dan C berbagi jalur- x , B dan C berbagi jalur- x , dan B serta C hanya *bersentuhan* di sepanjang $y = 6$. Dua sambungan berbiaya 1 menghubungkan semuanya, jadi jawabannya 2, dan program lengkap mencetak 2. Tanpa penukaran, B tersimpan dengan $x_1 = 7$ dan $x_2 = 3$, sehingga sapuan- x mengira B dimulai di 7. Pada saat itu `cur_max` bernilai 7 (ujung kanan C), uji ketat $7 < 7$ gagal, dan B tidak digabung dengan C . Sapuan- y tidak dapat menolong, karena B dan C hanya bersentuhan. B tetap sendirian, program membayar 2 untuknya alih-alih 1, dan mencetak 3.
5. Keluarannya tidak berubah sama sekali: *union by size* mengubah bentuk pohon-pohon DSU, bukan sisi mana yang menghubungkan kelompok berbeda, sehingga sisi yang sama diambil dan bobotnya tetap 14. Hanya waktu jalan pada masukan besar yang membaik.
6. Setiap pasangan berbiaya 2, jadi jawabannya $2(n - 1) = 9998$. Lintasan 0 murah di sini: untuk setiap desa pemindaian berhenti pada desa pertama berikutnya dalam urutan- x , karena tidak ada proyeksi- x yang bertumpuk, sehingga perulangan ganda hanya melakukan sekitar n perbandingan. Kasus terburuk kuadratik memerlukan banyak proyeksi- x yang bertumpuk, situasi yang sebaliknya.

Daftar Pustaka

- [1] Sphere Online Judge, "Problem VILUNIT." [Online]. Available: <https://www.spoj.com/problems/VILUNIT/>
- [2] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *Journal of the ACM*, vol. 22, no. 2, pp. 215–225, 1975.
- [3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022, ch. 19 and 21.
- [4] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.
- [5] S. Halim, F. Halim, and S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.

Bab 8

Tur Euler, *Fenwick Tree*, dan *Binary Lifting* — HBD

Sebagian soal olimpiade tidak diselesaikan dengan satu gagasan cerdas, melainkan dengan tiga gagasan biasa yang saling mengait seperti roda gigi. Bab ini mengajarkan cara bekerja seperti itu. Anda akan belajar meratakan pohon berakar menjadi sebuah larik dengan *tur Euler*, sehingga setiap subpohon menjadi selang yang bersambung; menyimpan jumlah atas selang-selang itu dengan *Fenwick tree* (*binary indexed tree*), sehingga jumlah nilai pada lintasan dari akar ke simpul mana pun cukup dengan satu kueri prefiks; dan memanjat pohon dengan langkah 2^j memakai *binary lifting*, sehingga pencarian sepanjang lintasan memerlukan waktu logaritmik. Setiap roda gigi pendek dan mudah bila berdiri sendiri. Keterampilan yang memenangkan medali adalah menyadari bahwa sebuah soal meminta ketiganya sekaligus, lalu menuliskannya sebagai kontrak-kontrak yang rapi. Studi kasusnya, SPOJ HBD [1], meminta persis hal ini: pada pohon berisi hingga 10^5 simpul yang nilainya berubah-ubah, berapa langkah ke atas dari sebuah simpul yang diperlukan sebelum nilai yang terkumpul mencapai suatu target?

8.1 Mengumpulkan Sumbangan ke Atas Pohon Keluarga

Sebuah keluarga besar menyiapkan hajatan. Pohon keluarganya digambar dengan leluhur tertua di puncak. Setiap orang telah menjanjikan sejumlah uang, dan janji itu bertambah ketika hari-H makin dekat: seorang paman menambah lima puluh ribu rupiah, seorang sepupu menambah dua puluh ribu. Panitia sering mengajukan pertanyaan dengan bentuk yang sama: “Mulai dari cucu ini, naik melalui orang tuanya, kakek-neneknya, dan seterusnya, tetapi tidak lebih tinggi dari leluhur yang ini, berapa orang yang harus menyumbang sebelum kita mengumpulkan paling sedikit sejumlah ini?”

Dengan segelintir orang, panitia cukup berjalan naik di pohon sambil menjumlahkan. Dengan seratus ribu orang dan seratus ribu pembaruan serta pertanyaan yang bercampur, berjalan terlalu lambat: pohon keluarga bisa berupa rantai panjang, dan setiap pertanyaan mungkin berjalan puluhan ribu langkah. Kita memerlukan cara untuk mengetahui total sepanjang lintasan ke atas mana pun seketika, bahkan ketika nilainya berubah, dan cara untuk menemukan titik berhenti tanpa berjalan. Kebutuhan yang sama muncul dalam hierarki perusahaan (berapa tingkat manajer yang harus menyetujui anggaran sebelum total yang disetujui mencapai ambang), dalam sistem berkas (ukuran folder sepanjang sebuah jalur), dan dalam pohon perutean jaringan.

Bab ini disusun mengikuti ketiga roda gigi itu. Subbab 8.2 membangun masing-masing dari nol dan membuktikan kesamaan yang menghubungkannya dengan roda gigi berikutnya: subpohon adalah selang, jumlah dari akar ke simpul adalah jumlah prefiks, jumlah lintasan vertikal adalah selisih, dan pencarian monoton sepanjang lintasan adalah barisan lompatan berukuran pangkat dua. Subbab 8.3 menjalankan ketiganya pada pohon berisi sembilan simpul yang angka-angkanya dapat diperiksa dengan tangan. Subbab 8.4 merangkainya menjadi penyelesaian HBD, menelusuri satu pertanyaan langkah demi langkah, dan menjelaskan setiap baris program. Subbab 8.5

mengujinya terhadap cara lambat panitia berjalan naik, dan Subbab 8.6 mendaftar soal-soal lain yang diselesaikan oleh roda gigi yang sama.

8.2 Tiga Roda Gigi dari Nol

Di sepanjang bab ini pohon *berakar* di simpul 1. *Induk* sebuah simpul adalah tetangganya yang satu langkah lebih dekat ke akar, *leluhur* sebuah simpul adalah induknya, induk dari induknya, dan seterusnya hingga akar (bila memudahkan, simpul itu sendiri juga dihitung sebagai leluhurnya), dan *subpohon* dari u adalah u beserta semua simpul di bawahnya. *Kedalaman* akar adalah 1, dan setiap langkah ke bawah menambah 1.

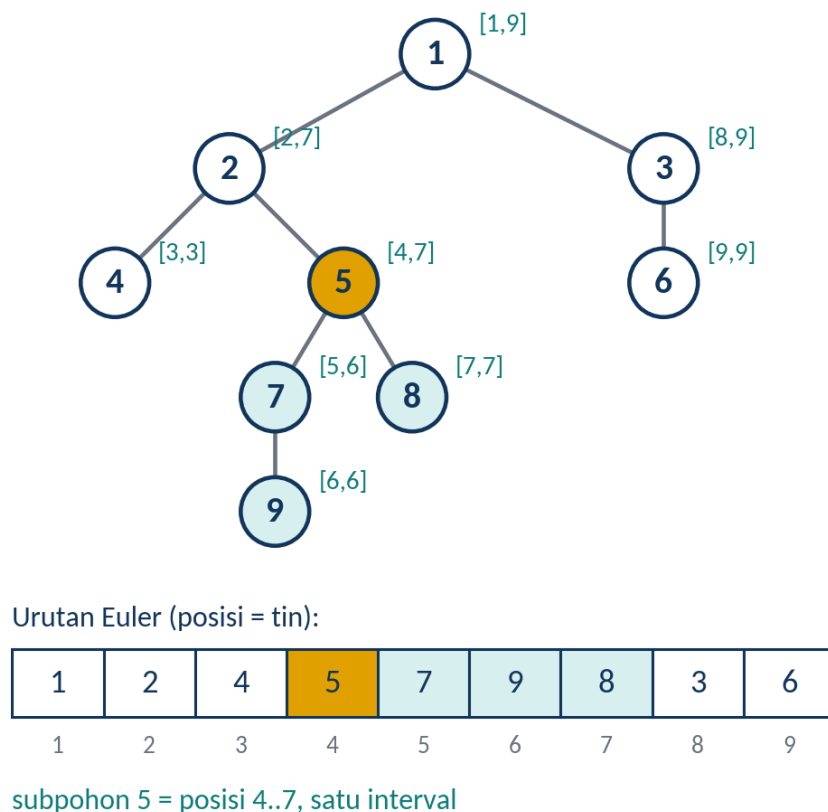
8.2.1 Roda gigi 1: tur Euler mengubah subpohon menjadi selang

Jalankan *depth-first search* (penelusuran mendalam) dari akar sambil memegang sebuah jam. Ketika penelusuran pertama kali memasuki simpul u , majukan jam dan catat waktunya sebagai $tin[u]$. Ketika penelusuran meninggalkan u untuk terakhir kalinya, catat nilai jam saat itu sebagai $tout[u]$, tanpa memajukannya. Setiap simpul memperoleh waktu masuk yang berbeda antara 1 dan n . Ikuti jamnya pada pohon di **Gambar 8.1**, dengan mengunjungi anak-anak dalam urutan menaik. Penelusuran memasuki 1 (waktu 1), lalu 2 (waktu 2), lalu 4 (waktu 3); simpul 4 tidak memiliki anak, sehingga ditinggalkan dengan $tout[4] = 3$. Kembali di 2, penelusuran memasuki 5 (waktu 4), lalu 7 (waktu 5), lalu 9 (waktu 6), sebuah daun: $tout[9] = 6$ dan, karena 7 tidak punya anak lain, $tout[7] = 6$ pula. Kembali di 5 penelusuran memasuki 8 (waktu 7), sebuah daun, lalu meninggalkan 8, 5, dan 2 dengan waktu keluar 7. Akhirnya penelusuran memasuki 3 (waktu 8) dan 6 (waktu 9) lalu meninggalkan 6, 3, dan 1 dengan waktu keluar 9. Waktu keluar sebuah simpul selalu sama dengan waktu masuk simpul terakhir di subpohonnya.

Karena *depth-first search* menuntaskan seluruh subpohon u sebelum meninggalkan u , simpul-simpul subpohon itu tepat simpul-simpul yang dimasuki di antara saat masuk dan saat keluar u :

$$w \text{ berada di subpohon } u \Leftrightarrow tin[u] \leq tin[w] \leq tout[u] \quad (8.1)$$

Gambar 8.1 memperlihatkan pohon sembilan simpul yang setiap simpulnya diberi label $[tin, tout]$, dan di bawahnya simpul-simpul ditulis menurut urutan waktu masuk. Subpohon simpul 5, yaitu $\{5, 7, 9, 8\}$, menempati posisi 4 sampai 7: satu selang.



Gambar 8.1 Pohon dengan waktu masuk dan keluar $[tin, tout]$; dalam urutan masuk, subpohon simpul 5 mengisi satu selang, yaitu posisi 4 sampai 7.

8.2.2 Roda gigi 2: Fenwick tree untuk jumlah prefiks dengan pembaruan

Fenwick tree [2] menyimpan larik $a[1..n]$ sedemikian rupa sehingga “tambahkan v ke $a[i]$ ” maupun “hitung $a[1] + \dots + a[i]$ ” memerlukan waktu $O(\log n)$. Penyimpanannya adalah larik **bit**, dengan **bit** $[i]$ memuat jumlah a atas blok sepanjang $\text{low}(i)$ yang berakhir di i , dan $\text{low}(i)$ adalah nilai bit menyala terendah dari i , yang dalam C++ dihitung sebagai $i \& -i$. Jumlah prefiks berjalan turun dengan membuang bit terendah setiap kali; pembaruan berjalan naik dengan menambahkannya:

```
add(i, v): while i <= n: bit[i] += v; i += i & -i
query(i): s = 0; while i > 0: s += bit[i]; i -= i & -i; return s
```

Setiap perulangan berjalan paling banyak $\lceil \log_2 n \rceil + 1$ kali, karena setiap langkah mengubah bit menyala terendah.

Trik yang penting di sini adalah memakai *Fenwick tree* secara terbalik: *pembaruan selang*, *kueri titik*. Beri setiap simpul u nilai $val[u]$ dan lakukan

$$\text{add}(tin[u], +val[u]), \quad \text{add}(tout[u] + 1, -val[u]) \quad (8.2)$$

Jumlah prefiks hingga posisi $tin[w]$ lalu mengumpulkan $+val[u]$ tepat ketika $tin[u] \leq tin[w]$, dan tidak mengumpulkan pasangannya $-val[u]$ tepat ketika $tin[w] \leq tout[u]$. Menurut [Persamaan \(8.1\)](#) hal itu terjadi persis ketika w berada di subpohon u , yaitu ketika u adalah leluhur w . Jadi

$$S(w) = \text{query}(\text{tin}[w]) = \sum_{u \text{ leluhur } w} \text{val}[u] \quad (8.3)$$

adalah jumlah nilai pada lintasan dari akar turun ke w . Ketika sebuah nilai berubah sebesar y , menerapkan [Persamaan \(8.2\)](#) dengan y memperbarui setiap $S(w)$ yang terdampak sekaligus, dalam $O(\log n)$.

8.2.3 Jumlah lintasan sebagai selisih

Misalkan a leluhur dari v . Simpul-simpul pada lintasan naik dari v ke a , keduanya termasuk, adalah leluhur-leluhur v dikurangi leluhur-leluhur $\text{parent}(a)$. Karena itu

$$\text{sum}(v \rightarrow a) = S(v) - S(\text{parent}(a)) \quad (8.4)$$

dengan kesepakatan $S(0) = 0$ untuk induk khayalan dari akar. Setiap pertanyaan tentang lintasan vertikal kini menjadi pengurangan dua jumlah prefiks. Sebagai contoh pada pohon [Gambar 8.1](#) dengan nilai-nilai pada Subbab 8.3, jumlah lintasan dari simpul 9 naik ke simpul 5 adalah $S(9) - S(2) = 23 - 8 = 15$, yaitu $8 + 6 + 1$, tanpa perlu berjalan satu langkah pun. Inilah kontrak dari dua roda gigi pertama: *berikan simpul mana pun, dan saya beri tahu jumlah dari akar ke simpul itu dalam $O(\log n)$, bahkan ketika nilainya berubah.*

8.2.4 Fenwick tree dengan tangan

Fenwick tree tampak seperti sulap sampai Anda menggambar blok-bloknya sekali. Untuk $n = 9$, nilai $i \& -i$ adalah 1 untuk i ganjil, 2 untuk $i = 2, 6$, 4 untuk $i = 4$, dan 8 untuk $i = 8$. Jadi [bit\[1\]](#) meliputi $a[1]$, [bit\[2\]](#) meliputi $a[1..2]$, [bit\[3\]](#) meliputi $a[3]$, [bit\[4\]](#) meliputi $a[1..4]$, [bit\[5\]](#) meliputi $a[5]$, [bit\[6\]](#) meliputi $a[5..6]$, [bit\[7\]](#) meliputi $a[7]$, [bit\[8\]](#) meliputi $a[1..8]$, dan [bit\[9\]](#) meliputi $a[9]$.

Kueri prefiks untuk posisi 7 berjalan $7 \rightarrow 6 \rightarrow 4 \rightarrow 0$: menjumlahkan [bit\[7\]](#), [bit\[6\]](#), dan [bit\[4\]](#), yaitu $a[7] + a[5..6] + a[1..4]$, tepat $a[1..7]$, dengan tiga penjumlahan alih-alih tujuh. Pembaruan pada posisi 3 berjalan $3 \rightarrow 4 \rightarrow 8 \rightarrow 16$: mengubah [bit\[3\]](#), [bit\[4\]](#), dan [bit\[8\]](#), persis blok-blok yang memuat posisi 3, lalu berhenti karena 16 melebihi n . Dalam biner polanya jelas: kueri menghapus bit 1 terendah pada setiap langkah, pembaruan menambahkannya, dan keduanya menyentuh paling banyak satu blok per bit dari n .

Sekarang padukan ini dengan [Persamaan \(8.2\)](#). Ketika simpul 5 pada [Gambar 8.1](#) bertambah 10, program menambahkan $+10$ pada posisi $\text{tin}[5] = 4$ dan -10 pada posisi $\text{tout}[5] + 1 = 8$. Kueri pada posisi 6 (simpul 9) mengumpulkan $+10$ dari [bit\[4\]](#) dan tidak mencapai posisi 8, sehingga $S(9)$ bertambah 10. Kueri pada posisi 9 (simpul 6) mengumpulkan $+10$ sekaligus -10 melalui [bit\[8\]](#), sehingga $S(6)$ tidak berubah, sebagaimana mestinya, karena simpul 5 bukan leluhur simpul 6.

8.2.5 Roda gigi 3: *binary lifting*

Misalkan $\text{up}[u][j]$ adalah leluhur u yang berada 2^j langkah lebih tinggi, atau 0 jika tidak ada. Tabelnya diisi dari hubungan

$$\text{up}[u][j] = \text{up}[\text{up}[u][j-1]][j-1] \quad (8.5)$$

karena melompat 2^j langkah sama dengan melompat 2^{j-1} langkah dua kali. Jika *depth-first search* mengisi baris milik u ketika memasuki u , semua leluhur u sudah memiliki baris lengkap, sehingga

Persamaan (8.5) selalu tersedia. Dengan 20 kolom kita dapat melompat hingga $2^{19} > 10^5$ langkah sekaligus.

Binary lifting menjawab *pencarian* sepanjang lintasan. Misalkan sebuah sifat P berlaku untuk leluhur-leluhur rendah dari v dan gagal untuk yang tinggi, dengan satu titik peralihan di antaranya: sifat yang *monoton*. Maka kita dapat menemukan leluhur tertinggi yang memenuhi P dengan mencoba lompatan $2^{19}, 2^{18}, \dots, 2^0$ secara berurutan dan mengambil setiap lompatan yang mendarat pada simpul tempat P masih berlaku. Inilah pencarian biner, dengan pangkat dua sebagai ganti titik tengah, dan memerlukan $O(\log n)$ lompatan. Penunjuk lompatan yang sama adalah jalan klasik menuju leluhur bersama terendah [3], dan pustaka *competitive programming* memakainya untuk segala macam pencarian ke atas [4].

WAWASAN KUNCI: Rangkailah melalui kontrak. Tanyakan: “Seandainya saya bisa mendapatkan $S(w)$ untuk sembarang w dalam $O(\log n)$, soal selebihnya menjadi apa?” Di sini soal itu menjadi pencarian monoton sepanjang lintasan ke atas, yang diselesaikan *binary lifting* dengan $O(\log n)$ penjajakan. Setiap penjajakan memanggil kontrak pertama sekali, sehingga satu pertanyaan berbiaya $O(\log^2 n)$.

8.3 Pemanasan: Tiga Roda Gigi pada Pohon Kecil

Kode 8.1 membangun pohon pada **Gambar 8.1**, memberi simpul-simpulnya nilai 5, 3, 2, 7, 1, 4, 6, 2, 8, mencetak waktu masuk, waktu keluar, dan jumlah dari akar ke simpul, melompat dengan *binary lifting*, dan akhirnya menambahkan 10 pada simpul 5.

Kode 8.1 `tree_gears.cpp`: tur Euler, jumlah lintasan dengan Fenwick, dan *binary lifting* pada sembilan simpul.

```
// Euler tour + BIT (root-to-node path sums) and binary lifting on a small tree.
#include <iostream>
#include <vector>
using namespace std;
const int N = 9, LOG = 4;
vector<int> adj[N + 1];
int tin[N + 1], tout[N + 1], up[N + 1][LOG], dep[N + 1], timer_ = 0;
long long bit[N + 2];
void add(int i, long long v) {
    for (; i <= N; i += i & -i) bit[i] += v;
}
long long pref(int i) {
    long long s = 0;
    for (; i > 0; i -= i & -i) s += bit[i];
    return s;
}
void dfs(int u, int p) {
    tin[u] = ++timer_;
    up[u][0] = p;
    for (int j = 1; j < LOG; j++) up[u][j] = up[up[u][j - 1]][j - 1];
    for (int v : adj[u])
        if (v != p) dep[v] = dep[u] + 1, dfs(v, u);
    tout[u] = timer_;
}
void addNode(int u, long long y) { // node value += y
    add(tin[u], y);
    add(tout[u] + 1, -y);
}
int kth(int u, int k) { // k-th ancestor, 0 if it does not exist
    for (int j = 0; j < LOG; j++)
        if (k >> j & 1) u = up[u][j];
    return u;
}
int main() {
    int e[8][2] = {{1, 2}, {1, 3}, {2, 4}, {2, 5}, {3, 6}, {5, 7}, {5, 8}, {7, 9}};
    for (auto& x : e) adj[x[0]].push_back(x[1]), adj[x[1]].push_back(x[0]);
    dfs(1, 0);
    long long val[N + 1] = {0, 5, 3, 2, 7, 1, 4, 6, 2, 8};
    for (int u = 1; u <= N; u++) addNode(u, val[u]);
    cout << "node tin tout S(root->node)\n";
    for (int u = 1; u <= N; u++)
        cout << u << " " << tin[u] << " " << tout[u] << " " << pref(tin[u])
            << "\n";
    cout << "2nd ancestor of 9 = " << kth(9, 2) << ", 3rd = " << kth(9, 3) << "\n";
    addNode(5, 10); // node 5 gains 10: every path through 5 grows
    cout << "after +10 at node 5: S(9) = " << pref(tin[9])

```

Kode 8.1 (Lanjutan) *tree_gears.cpp*: tur Euler, jumlah lintasan dengan Fenwick, dan binary lifting pada sembilan simpul.

```
<< ", S(4) = " << pref(tin[4]) << "\n";
}
```

```
node tin tout S(root->node)
1 1 9 5
2 2 7 8
3 8 9 7
4 3 3 15
5 4 7 9
6 9 9 11
7 5 6 15
8 7 7 11
9 6 6 23
2nd ancestor of 9 = 5, 3rd = 2
after +10 at node 5: S(9) = 33, S(4) = 15
```

Periksa beberapa angka dengan tangan. Lintasan dari akar ke simpul 9 adalah 1,2,5,7,9 dengan nilai $5 + 3 + 1 + 6 + 8 = 23$. Leluhur 9 dua dan tiga langkah ke atas adalah 5 dan 2. Setelah 10 ditambahkan pada simpul 5, $S(9)$ bertambah 10 karena 5 terletak pada lintasannya, sedangkan $S(4)$ tetap 15, karena simpul 4 tidak berada di subpohon 5: waktu masuknya, 3, berada di luar selang $[4,7]$.

8.4 Studi Kasus: HBD

8.4.1 Soal dengan kata-kata kita sendiri

Soalnya adalah SPOJ HBD [1]. Baris pertama memuat n dan q , keduanya hingga 10^5 . $n - 1$ baris berikutnya adalah sisi-sisi pohon yang akarnya simpul 1. Lalu datang n bilangan bulat, nilai awal simpul $1..n$. Terakhir ada q operasi dari dua jenis:

- **2 u y**: nilai simpul u bertambah y .
- **1 u v x**: simpul u adalah leluhur v (mungkin v sendiri). Berjalanlah naik dari v menuju u sambil menjumlahkan nilai simpul-simpul yang dikunjungi, termasuk v . Cetak berapa simpul yang sudah dikunjungi pada saat total pertama kali mencapai paling sedikit x . Jika bahkan seluruh lintasan dari v ke u berjumlah kurang dari x , cetak -1 .

Nilai dan penambahannya tak negatif. Hal ini penting: total berjalan bertambah ketika kita naik, sehingga “sudahkah kita mencapai x ?” adalah sifat monoton, persis yang diperlukan *binary lifting*.

8.4.2 Contoh masukan dan keluaran

Kita memakai pohon pada **Gambar 8.1** dengan nilai yang sama seperti pemanasan:

```

9 6
1 2
1 3
2 4
2 5
3 6
5 7
5 8
7 9
5 3 2 7 1 4 6 2 8
1 1 9 10
1 2 9 16
1 5 9 100
2 7 5
1 5 9 16
1 1 4 20
```

```

2
4
-1
2
-1
```

Berjalan naik dari simpul 9 mengunjungi 9,7,5,2,1 dengan nilai 8,6,1,3,5 dan total berjalan 8,14,15,18,23. Pertanyaan pertama memerlukan 10: tercapai pada simpul kedua. Pertanyaan kedua memerlukan 16: tercapai pada simpul keempat (18), dan simpul 2 masih di dalam lintasan yang diizinkan. Pertanyaan ketiga memerlukan 100, tetapi lintasan dari 9 ke 5 hanya berjumlah 15. Lalu simpul 7 bertambah menjadi 11, totalnya menjadi 8,19, ..., dan 16 tercapai pada simpul kedua. Pertanyaan terakhir berjalan 4,2,1 dengan total $7 + 3 + 5 = 15 < 20$.

8.4.3 Merancang penyelesaian

1. **Satu *depth-first search*** dari akar menghitung *tin*, *tout*, kedalaman, dan tabel *lifting* dengan **Persamaan** (8.5). Simpul 0 adalah penjaga (*sentinel*): kedalamannya 0, dan semua lompatannya kembali ke 0.
2. **Nilai awal** dimuat dengan **Persamaan** (8.2); operasi **2 u y** melakukan hal yang sama dengan *y*.
3. **Pertanyaan 1 u v x**. Pertama hitung jumlah seluruh lintasan $S(v) - S(\text{parent}(u))$ dengan **Persamaan** (8.4). Jika kurang dari *x*, cetak -1 .
4. **Selain itu, cari**. Misalkan $\text{curr} = v$. Untuk $j = 19$ turun hingga 0, lihat $p = \text{up}[\text{curr}][j]$. Lompatan diizinkan jika p tidak berada di atas u , yaitu $\text{depth}(p) \geq \text{depth}(u)$. Lompatan diambil jika simpul-simpul yang tepat berada di bawah p pada lintasan masih berjumlah kurang dari x :

$$S(v) - S(p) < x \Leftrightarrow S(p) > S(v) - x \quad (8.6)$$

5. **Jawabannya** adalah $\text{depth}(v) - \text{depth}(\text{curr}) + 1$, banyaknya simpul dari v naik hingga curr .
Sebuah alternatif yang wajar layak dikomentari. Kita bisa melakukan pencarian biner atas *banyaknya langkah k*, menghitung leluhur ke- k dengan tabel *lifting* untuk setiap kandidat k lalu menguji jumlah lintasannya. Cara itu benar tetapi berbiaya $O(\log n)$ kandidat, masing-masing dengan perhitungan leluhur $O(\log n)$ dan kueri Fenwick $O(\log n)$: $O(\log^3 n)$ per pertanyaan. Menuruni pangkat dua secara langsung, seperti pada langkah 4, meleburkan pencarian biner dan perhitungan leluhur dalam satu perulangan dan menghemat satu faktor logaritma penuh.

Simpul penjaga 0 layak mendapat kalimatnya sendiri. Setiap lompatan yang akan keluar dari pohon melalui akar mendarat di simpul 0, yang semua lompatannya kembali ke 0 dan kedalamannya 0. Karena setiap simpul nyata berkedalaman paling sedikit 1, uji kedalaman pada langkah 4 otomatis menolak setiap lompatan ke penjaga, dan **get_S(0)** mengembalikan 0, tepat nilai yang diperlukan **Persamaan** (8.4) ketika u adalah akar dan “induk”-nya adalah simpul khayalan di atasnya. Satu simpul tambahan menghapus tiga kasus khusus dari kode: induk akar, lompatan melewati akar, dan jumlah lintasan yang berakhir di akar.

Mengapa curr adalah simpul tempat total pertama kali mencapai x ? Pencarian menjaga invarian “simpul-simpul yang tepat berada di bawah curr berjumlah kurang dari x ”, yang benar pada awalnya (tidak ada apa pun di bawah v pada lintasan). Pencarian berakhir pada simpul *tertinggi* seperti itu yang tidak berada di atas u . Jika $\text{curr} = u$, seluruh lintasan berjumlah paling sedikit x (diperiksa pada langkah 3) sedangkan semua yang tepat di bawah u berjumlah kurang, sehingga u adalah simpul berhenti. Jika tidak, induk curr gagal memenuhi **Persamaan** (8.6): simpul-simpul dari v hingga curr berjumlah paling sedikit x , sedangkan yang tepat di bawah curr berjumlah kurang. Dalam kedua kasus, curr adalah simpul pertama tempat total mencapai x .

8.4.4 Menelusuri satu pertanyaan

Ambil pertanyaan kedua pada contoh, **1 2 9 16**, sebelum ada pembaruan. Kedalaman sepanjang lintasan adalah 5 untuk simpul 9, 4 untuk simpul 7, 3 untuk simpul 5, 2 untuk simpul 2, dan 1 untuk akar. Jumlah dari akar ke simpul adalah $S(9) = 23$, $S(7) = 15$, $S(5) = 9$, $S(2) = 8$, dan $S(1) = 5$; induk dari $u = 2$ adalah akar, sehingga seluruh lintasan dari 9 naik ke 2 berjumlah $23 - 5 = 18 \geq 16$, dan pencarian dimulai dengan $\text{curr} = 9$ dan ambang $S(v) - x = 7$.

- Untuk $j = 19$ turun hingga $j = 3$, lompatan menuju ke luar akar, ke penjaga, yang kedalamannya 0 berada di bawah $\text{depth}(u) = 2$: ditolak.
- $j = 2$: empat langkah ke atas dari 9 adalah simpul 1, kedalaman 1, di atas u : ditolak.

- $j = 1$: dua langkah ke atas adalah simpul 5, kedalaman 3, diizinkan. $S(5) = 9 > 7$, sehingga simpul-simpul yang tepat di bawah simpul 5 (yaitu 9 dan 7) berjumlah $14 < 16$: lompat, $curr = 5$.
- $j = 0$: satu langkah ke atas dari 5 adalah simpul 2, kedalaman 2, diizinkan. $S(2) = 8 > 7$, sehingga simpul-simpul yang tepat di bawah simpul 2 (9, 7, dan 5) berjumlah $15 < 16$: lompat, $curr = 2$.

Jawabannya $depth(9) - depth(2) + 1 = 5 - 2 + 1 = 4$, cocok dengan baris keluaran kedua. Perhatikan bagaimana lompatan 2 lalu 1 bersama-sama memanjat tiga tingkat: setiap banyaknya langkah dari 0 hingga $2^{20} - 1$ adalah jumlah pangkat dua yang berbeda, dan pencarian memilih tepat digit-digit biner dari jarak ke simpul berhenti.

8.4.5 Penyelesaian lengkap

Kode 8.2 adalah program lengkap yang tercatat *Accepted* di SPOJ. Blok P50 adalah pencarian *binary lifting* dari langkah 4.

Kode 8.2 `HBD.cpp`: tur Euler, *Fenwick tree*, dan *binary lifting* yang dirangkai.

```
#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;
// Fast manual input: one character at a time, no locking
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;
}
inline void add(int idx, long long val) {
    for (; idx <= n; idx += idx & -idx) bit[idx] += val;
}
inline long long query(int idx) {
    long long sum = 0;
    for (; idx > 0; idx -= idx & -idx) sum += bit[idx];
    return sum;
}
inline long long get_S(int u) {
    if (u == 0) return 0;
    return query(in[u]);
}
void dfs(int u, int p, int d) {
    in[u] = ++timer_dfs;
    depth[u] = d;
    up[u][0] = p;
    for (int i = 1; i < 20; i++) {
        up[u][i] = up[up[u][i - 1]][i - 1];
    }
    for (int v : adj[u]) {
        if (v != p) {
            dfs(v, u, d + 1);
        }
    }
    out[u] = timer_dfs;
}
int main() {
    ios_base::sync_with_stdio(false);
```

Kode 8.2 (Lanjutan) *HBD.cpp: tur Euler, Fenwick tree, dan binary lifting yang dirangkai.*

```

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();
    if (type == 1) {
        int u = read_ll(), v = read_ll();
        long long x = read_ll();
        long long S_v = get_S(v);
        long long S_parent_u = get_S(up[u][0]);
        // If the total sum on the path is not enough
        if (S_v - S_parent_u < x) {
            cout << "-1\n";
            continue;
        }
        // <P50>
        // Climb from v with jumps 2^19..2^0, never above u: jump to p while the
        // sum strictly below p, S_v - S_p, is still smaller than x.
        // >>> YOUR TURN (Public Edition): write this part yourself, following the hint
        // above. <<<
        // </P50>
        cout << depth[v] - depth[curr] + 1 << "\n";
    } else {
        int u = read_ll();
        long long y = read_ll();
        add(in[u], y);
        add(out[u] + 1, -y);
    }
}
return 0;
}

```

8.4.6 Baris demi baris

Variabel global. `adj` menyimpan pohon, `up` tabel *lifting* dengan 20 kolom, `depth`, `in`, dan `out` data Euler, dan `bit` larik Fenwick bertipe `long long`: 10^5 nilai, masing-masing ditambah hingga 10^5 kali, dapat melampaui jangkauan `int`.

read_ll. Pembaca buatan sendiri dengan `getchar_unlocked`, yang melewati karakter bukan digit dan mengingat tanda minus. Dengan sekitar setengah juta bilangan, pembaca ini menghemat waktu yang terasa. Program tidak pernah mencampurnya dengan `cin` untuk masukan; `cout` dipakai hanya untuk keluaran.

add, query, get_S. Operasi Fenwick dari Subbab 8.2.2, dan `get_S(u)`, yaitu $S(u)$ dari Persamaan (8.3), yang mengembalikan 0 untuk simpul penjaga 0.

dfs. Fungsi ini mencatat *tin*, menetapkan kedalaman dan induk langsung, mengisi sisa baris dengan **Persamaan** (8.5), mengunjungi anak-anak, lalu mencatat *tout*. Untuk akar, $p = 0$, sehingga semua lompatannya menuju penjaga.

main, persiapan. Membaca sisi-sisi ke dalam daftar ketetanggaan, menjalankan `dfs(1, 0, 1)`, dan memuat nilai awal dengan **Persamaan** (8.2).

Sebuah pertanyaan. `S_v` dan `S_parent_u` memberikan jumlah seluruh lintasan; jika terlalu kecil, cetak -1 dan lanjutkan. Blok P50 menjalankan pencarian langkah 4: uji `depth[p] >= depth[u]` menolak lompatan ke atas u (dan ke penjaga, yang kedalamannya 0 berada di bawah setiap kedalaman nyata), dan uji `S_p > S_v - x` adalah **Persamaan** (8.6). Baris terakhir mencetak banyaknya simpul.

Sebuah pembaruan. Dua penambahan Fenwick, seperti pada **Persamaan** (8.2).

8.4.7 Kompleksitas dan jebakan

Pencarian melakukan 20 penjajakan, masing-masing sebuah kueri Fenwick sekitar 17 langkah, sehingga satu pertanyaan berbiaya kira-kira 350 langkah dasar dan 10^5 pertanyaan sekitar $3.5 \cdot 10^7$: cepat dengan leluasa. Kami mengukur masukan terbesar yang dapat kami bangun, rantai berisi 10^5 simpul (pohon terdalam yang mungkin) dengan 10^5 operasi campuran: 0,08 detik.

JEBAKAN: Rekursi yang dalam. Pada rantai, `dfs` berekursi sedalam 10^5 tingkat. Itu aman di Linux dan di *judge*, yang *stack*-nya besar, tetapi *stack* bawaan 1 MB milik program Windows dapat meluap dan membuat program berhenti tanpa pesan. Di Windows kompilasilah dengan `-Wl,--stack=268435456`, atau tulis penelusurannya secara iteratif (Soal Latihan 5).

- **Meleset satu pada jumlah lintasan.** Lintasan dari v ke u memuat u , jadi kurangkan $S(\text{parent}(u))$, bukan $S(u)$.
- **Melompat melewati u .** Tanpa uji kedalaman, pencarian dapat menghitung simpul yang tidak diizinkan.
- **Nilai negatif.** Jika nilai boleh negatif, total berjalan tidak monoton dan *binary lifting* diam-diam memberikan jawaban salah. Baca batasan soal sebelum memilih pencarian biner.
- **Jumlah bertipe `int`.** Total melampaui 2^{31} setelah cukup banyak pembaruan; semua yang menyimpan jumlah bertipe `long long`.

8.5 Uji Stres terhadap *Brute Force*

Kode 8.3 melakukan apa yang dilakukan panitia pada Subbab 8.1 dengan tangan: menyimpan penunjuk induk, menerapkan pembaruan langsung pada sebuah larik, dan menjawab pertanyaan dengan berjalan naik satu simpul demi satu simpul.

Kode 8.3 `brute.cpp`: acuan jalan-naik untuk HBD.

```
// Reference for HBD: parent array, walk up node by node.
#include <cstdio>
#include <vector>
using namespace std;
int main() {
    int n, q;
    if (scanf("%d", &n, &q) != 2) return 0;
    vector<vector<int>> adj(n + 1);
    for (int i = 0; i < n - 1; i++) {
        int u, v;
        if (scanf("%d", &u, &v) != 2) return 0;
        adj[u].push_back(v), adj[v].push_back(u);
    }
    vector<int> par(n + 1, 0), st = {1}, seen(n + 1, 0);
    seen[1] = 1;
    while (!st.empty()) {
        int u = st.back();
        st.pop_back();
        for (int v : adj[u])
            if (!seen[v]) seen[v] = 1, par[v] = u, st.push_back(v);
    }
    vector<long long> val(n + 1);
    for (int i = 1; i <= n; i++)
        if (scanf("%lld", &val[i]) != 1) return 0;
    while (q--) {
        int t;
        if (scanf("%d", &t) != 1) return 0;
        if (t == 1) {
            int u, v;
            long long x, s = 0;
            if (scanf("%d%d%lld", &u, &v, &x) != 3) return 0;
            int cnt = 0, w = v, ans = -1;
            while (true) {
                s += val[w], cnt++;
                if (s >= x) { ans = cnt; break; }
                if (w == u) break;
                w = par[w];
            }
            printf("%d\n", ans);
        } else {
            int u;
            long long y;
            if (scanf("%d%lld", &u, &y) != 2) return 0;
            val[u] += y;
        }
    }
}
```

Kode 8.3 (Lanjutan) *brute.cpp: acuan jalan-naik untuk HBD.*

```
}
```

Pembangkit **Kode 8.4** membangun pohon acak berisi hingga 12 simpul dengan memberi setiap simpul $v \geq 2$ induk acak yang nomornya lebih kecil, mencetak setiap sisi dengan arah acak (agar program tidak dapat bergantung pada urutannya), dan menghasilkan pertanyaan yang u -nya benar-benar leluhur v .

Kode 8.4 `gen.py`: pohon dan operasi acak kecil untuk HBD.

```
# Random small HBD inputs (values >= 0, x >= 1): python3 gen.py SEED
import random, sys
random.seed(int(sys.argv[1]))
n, q = random.randint(1, 12), random.randint(1, 15)
par = [0, 0] + [random.randint(1, i - 1) for i in range(2, n + 1)]
print(n, q)
for v in range(2, n + 1):
    print(par[v], v) if random.random() < .5 else print(v, par[v])
print(*[random.randint(0, 9) for _ in range(n)])
for _ in range(q):
    if random.random() < .7:
        v = random.randint(1, n)
        path = [v]
        while path[-1] != 1:
            path.append(par[path[-1]])
        print(1, random.choice(path), v, random.randint(1, 40))
    else:
        print(2, random.randint(1, n), random.randint(0, 9))
```

all 300 random tests passed

Ada gunanya mengetahui kesalahan apa saja yang ditangkap uji ini. Mengurangkan $S(u)$ alih-alih $S(\text{parent}(u))$ gagal setiap kali jawaban sebuah pertanyaan tepat seluruh lintasan, yang sering dihasilkan pembangkit karena ambangnya kecil. Uji kedalaman yang hilang membiarkan pencarian memanjat melewati u dan gagal pada pertanyaan yang u -nya bukan akar. Melupakan penambahan Fenwick kedua pada [Persamaan \(8.2\)](#) membuat sebuah pembaruan bocor ke simpul-simpul di luar subpohon, dan arah sisi yang acak menangkap program yang mengira titik ujung pertama setiap sisi selalu induknya. Yang tidak dapat diungkap uji kecil adalah program yang berhenti karena rekursi terlalu dalam atau pembaca masukan yang lambat; untuk keduanya, rantai berisi 10^5 simpul dari Subbab 8.4.7 adalah uji pendamping yang tepat.

8.6 Roda Gigi yang Sama, Soal yang Lain

Ketiga roda gigi muncul kembali dalam banyak soal, dan mengenal gabungan-gabungan bakunya menghemat banyak pemikiran selama lomba.

Jumlah subpohon. Simpan $val[u]$ pada posisi $tin[u]$ dari *Fenwick tree* biasa (pembaruan titik, kueri prefiks). Maka jumlah atas subpohon u adalah selang $[tin[u], tout[u]]$, dua kueri prefiks menurut [Persamaan \(8.1\)](#). Inilah bayangan cermin dari trik yang dipakai pada HBD: di sana nilai sebuah simpul disebar ke selang subpohonnya, di sini selang itu dijumlahkan.

Menambah seluruh subpohon. “Tambahkan y pada setiap nilai di subpohon u ” yang dipadukan dengan “beri tahu nilai simpul w ” sekali lagi adalah [Persamaan \(8.2\)](#): pembaruan selang kini menggambarkan operasinya sendiri, dan kueri titik membaca hasilnya.

Lintasan antara dua simpul sembarang. *Binary lifting* juga menemukan leluhur bersama terendah dari dua simpul a dan b : angkat simpul yang lebih dalam ke kedalaman yang sama, lalu angkat keduanya bersama dengan lompatan terbesar yang menjaga keduanya tetap terpisah. Dengan leluhur bersama terendah, setiap lintasan terbelah menjadi dua lintasan vertikal, dan [Persamaan \(8.4\)](#) menangani masing-masing (Soal Latihan 4).

Simpul ke- k pada lintasan. Mengangkat menurut digit-digit biner k , seperti yang dilakukan program pemanasan dengan `kth`, memberikan leluhur ke- k dalam $O(\log n)$; bersama leluhur bersama terendah, cara ini memberikan simpul ke- k pada lintasan mana pun.

Bila pernyataan soal menyebut pohon berakar, nilai yang berubah, dan pertanyaan tentang “ke atas” atau “di bawah”, telusurilah daftar ini sebelum merancang sesuatu yang baru. Sangat sering soal itu adalah salah satu gabungan ini dengan sedikit pelintiran, dan pelintiran itu satu-satunya bagian yang memerlukan pemikiran orisinal.

8.7 Taktik pada Hari Lomba

- **Tulis kontraknya lebih dulu.** Di kertas: “ $S(w)$ dalam $O(\log n)$ dengan pembaruan”, “leluruh ke-k dalam $O(\log n)$ ”, “pencarian monoton ke atas”. Lalu implementasikan setiap roda gigi secara terpisah dan uji secara terpisah pula.
- **Uji setiap roda gigi sendiri-sendiri.** Cetak *tin*, *tout*, dan $S(w)$ untuk pohon kecil, seperti yang dilakukan program pemanasan, lalu bandingkan dengan gambar tangan sebelum menulis pencariannya. *Bug* yang ditemukan pada satu roda gigi berbiaya beberapa menit; *bug* yang sama yang ditemukan melalui keseluruhan penyelesaian berbiaya satu jam.
- **Kenali kesamaan-kesamaannya.** Subpohon sama dengan selang Euler; lintasan akar-ke-simpul sama dengan prefiks Fenwick setelah [Persamaan](#) (8.2); lintasan vertikal sama dengan selisih menurut [Persamaan](#) (8.4). Tiga kalimat ini membuka keluarga besar soal pohon.
- **Cari kemonotonan sebelum melakukan pencarian biner.** Nilai tak negatif, kedalaman yang tak menurun, total yang bertambah: sebutkan alasan mengapa pencarian Anda sah.
- **Bangun uji rantai dan bintang.** Rantai menyingkap kedalaman rekursi dan lompatan panjang; bintang menyingkap simpul dengan banyak anak. Keduanya dibangkitkan dalam semenit, dan bersama pohon acak keduanya meliputi tiga bentuk tempat program pohon biasanya gagal.
- **Anggarkan faktor log-nya.** $O(\log^2 n)$ per kueri aman untuk 10^5 kueri; untuk 10^6 mungkin tidak, dan saat itu diperlukan rangkaian lain (misalnya menelusuri *Fenwick tree* secara langsung) [5].

8.8 Ringkasan Bab

- Tur Euler menomori simpul-simpul sehingga setiap subpohon menjadi sebuah selang, [Persamaan](#) (8.1).
- *Fenwick tree* dengan pembaruan selang dan kueri titik menjaga jumlah akar-ke-simpul di bawah nilai yang berubah, [Persamaan](#) (8.2) dan [Persamaan](#) (8.3); jumlah lintasan vertikal adalah selisih, [Persamaan](#) (8.4).
- *Binary lifting*, [Persamaan](#) (8.5), mengubah pencarian monoton sepanjang lintasan ke atas menjadi $O(\log n)$ lompatan.
- HBD merangkai ketiga roda gigi: setiap pertanyaan berbiaya $O(\log^2 n)$ dan setiap pembaruan $O(\log n)$.

8.9 Soal Latihan

1. Hitung *tin* dan *tout* dengan tangan untuk pohon dengan sisi 1–2, 1–3, 3–4, 3–5, 5–6, dengan mengunjungi anak-anak dalam urutan menaik. Selang manakah yang merupakan subpohon?
2. Misalkan nilai disimpan pada *sisi*, bukan pada simpul (setiap simpul bukan akar w membawa nilai sisi menuju induknya). Bagaimana [Persamaan](#) (8.4) dan jawaban sebuah pertanyaan berubah?
3. Berikan contoh mungil dengan satu nilai negatif ketika pencarian *binary lifting* mengembalikan jawaban salah, sedangkan *brute force* benar.

4. Bagaimana Anda menjawab “jumlah nilai pada lintasan antara dua simpul sembarang a dan b ”, ketika tidak satu pun leluhur yang lain?
5. Tulis ulang `dfs` pada **Kode 8.2** tanpa rekursi.
6. Berapa memori yang dipakai tabel *lifting* pada **Kode 8.2**, dan bagaimana memori itu dapat dihemat?

8.10 Penyelesaian Soal Latihan

1. Urutan masuknya 1, 2, 3, 4, 5, 6, sehingga *tin* bernilai 1 sampai 6 dengan urutan itu. Waktu keluar: simpul 2 keluar pada 2, simpul 4 pada 4, simpul 6 pada 6, simpul 5 pada 6, simpul 3 pada 6, simpul 1 pada 6. Subpohon 3 adalah selang [3,6], yang memuat simpul 3, 4, 5, dan 6.
2. Lintasan dari v naik ke u kini memakai sisi-sisi milik simpul $v, \dots, \text{child}(u)$, sehingga jumlahnya $S(v) - S(u)$, bukan $S(v) - S(\text{parent}(u))$. Sebuah pertanyaan menghitung sisi, bukan simpul, sehingga pencarian harus tetap tepat di bawah u (uji kedalaman dengan $>$), dan jawabannya menjadi $\text{depth}(v) - \text{depth}(\text{curr}) + 1$ sisi, dengan *curr* ujung bawah dari sisi terakhir yang dipakai.
3. Rantai 1 – 2 – 3 berakar di 1 dengan nilai $\text{val}[3] = 5, \text{val}[2] = -5, \text{val}[1] = 5$, dan pertanyaan 1 1 3 5. Berjalan naik dari 3, total pada simpul pertama sudah 5, sehingga jawabannya 1. Namun pencarian menguji lompatan dua langkah lebih dulu: $S(1) = 5 > S(3) - 5 = 0$, sehingga pencarian melompat ke simpul 1 dan mencetak 3. Kemonotonan dirusak oleh nilai negatif.
4. Pakai leluhur bersama terendah c dari a dan b (ditemukan dengan tabel *lifting* yang sama): jumlahnya $S(a) + S(b) - 2S(c) + \text{val}[c]$, karena leluhur-leluhur c terhitung dua kali dan c sendiri harus dihitung sekali.
5. Simpan *stack* eksplisit berisi pasangan (simpul, indeks anak berikutnya yang akan dikunjungi). Ketika sebuah simpul didorong, catat *tin*, kedalaman, dan baris *lifting*-nya; ketika indeks anaknya melewati ujung daftar, catat *tout* dan keluarkan simpul itu. Lewati induk ketika memindai anak-anak. Urutan kunjungan, dan karena itu semua angkanya, sama dengan versi rekursif.
6. Tabelnya berisi $100\,005 \times 20$ nilai bertipe `int`, masing-masing empat *byte*: sekitar 8 MB. Karena $2^{17} = 131\,072 > 10^5$, 17 kolom sudah cukup, yang menghemat 15 persen; penghematan lebih besar diperoleh dengan hanya menyimpan lompatan 2^j untuk j yang ditentukan oleh bit menyala terendah dari kedalaman (tata letak “*jump pointer*”), yang hanya memerlukan satu penunjuk tambahan per simpul dengan harga pencarian yang sedikit lebih panjang. Untuk HBD, 8 MB jauh di dalam batas dan tabel sederhana adalah pilihan yang tepat.

Daftar Pustaka

- [1] Sphere Online Judge, “Problem HBD.” [Online]. Available: <https://www.spoj.com/problems/HBD/>
- [2] P. M. Fenwick, “A new data structure for cumulative frequency tables,” *Software: Practice and Experience*, vol. 24, no. 3, pp. 327–336, 1994.
- [3] M. A. Bender and M. Farach-Colton, “The LCA problem revisited,” in *LATIN 2000: Theoretical Informatics*, LNCS vol. 1776. Berlin, Germany: Springer, 2000, pp. 88–94.
- [4] S. Halim, F. Halim, and S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.
- [5] A. Laaksonen, *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, 3rd ed. Cham, Switzerland: Springer, 2024.

Bab 9

Menghitung Objek Berbeda dengan Pemrograman Dinamis — DSUBSEQ

“Ada berapa cara?” dan “ada berapa yang berbeda?” adalah dua pertanyaan yang paling sering muncul di olimpiade, dan keduanya bukan pertanyaan yang sama. Menghitung cara biasanya berupa pemrograman dinamis yang lugas. Menghitung objek yang *berbeda* menyimpan jebakan: objek yang sama dapat dihasilkan dengan beberapa cara, dan rekurens yang ceroboh menghitungnya beberapa kali. Bab ini membahas cara menghindari jebakan itu dengan pembuktian, bukan dengan untung-untungan. Anda akan belajar menuliskan definisi keadaan sebagai kalimat utuh, menurunkan rekurens darinya, membuktikan dengan argumen himpunan bahwa tidak ada yang terhitung dua kali, dan menjaga seluruh aritmetika tetap benar modulo bilangan prima besar. Studi kasusnya, SPOJ DSUBSEQ [1], menghitung subbarisan berbeda dari untai sepanjang hingga 10^5 huruf, dan jawabannya adalah salah satu rekurens paling anggun dalam khazanah olimpiade: *gandakan hitungannya, lalu kurangi apa yang sudah dihasilkan kemunculan sebelumnya dari huruf yang sama*.

9.1 Kata-Kata Berbeda dari Satu Kata

Seorang guru menulis sebuah kata di papan, misalnya BANANA, lalu meminta kelas membuat “kata” baru dengan mencoret huruf mana saja yang mereka suka sambil mempertahankan urutan huruf yang tersisa. Tak lama kemudian kelas menyadari bahwa banyak cara mencoret yang berbeda menghasilkan kata yang sama: mencoret huruf ke-1, ke-3, dan ke-5 dari BANANA menghasilkan ANA, demikian pula mencoret huruf ke-1, ke-5, dan ke-6 lalu mempertahankan huruf ke-2, ke-3, dan ke-4. Ada berapa kata *berbeda* yang mungkin dibuat kelas itu?

Ada $2^6 = 64$ cara memilih huruf mana yang dipertahankan, tetapi hasil yang berbeda jauh lebih sedikit, karena huruf A dan N yang berulang membuat banyak pilihan menghasilkan kata yang sama. Untuk kata sepanjang 10^5 huruf, banyaknya pilihan adalah 2^{100000} , bilangan dengan sekitar tiga puluh ribu digit, sehingga mendaftar semuanya mustahil. Kita memerlukan cara untuk menghitung hasil yang berbeda secara langsung, sekaligus cara untuk yakin bahwa hitungannya benar, karena tidak seorang pun sanggup memeriksa jawaban berdigit tiga puluh ribu dengan tangan. Keyakinan itu harus datang dari pembuktian, dan pembuktian itulah yang dibangun bab ini langkah demi langkah.

Gagasan pertama adalah membangkitkan hasilnya satu per satu dan menyimpannya dalam sebuah himpunan sambil membuang yang berulang. Untuk BANANA cara ini selesai seketika. Untuk untai sepanjang empat puluh huruf, cara ini sudah memerlukan memori lebih besar daripada yang dimiliki komputer mana pun, karena banyaknya hasil yang *berbeda* sendiri dapat bersifat eksponensial: untai berisi empat puluh huruf yang semuanya berbeda memiliki 2^{40} , lebih dari satu triliun, subbarisan berbeda. Jadi kita harus menghitung tanpa mendaftar, dan menghitung tanpa mendaftar persis tempat penghitungan ganda menyelip masuk: kita tidak lagi dapat *melihat* duplikatnya, kita harus *membuktikan* di mana duplikat itu berada.

Pertanyaan ini bukan sekadar permainan kelas. Dalam genetika, untai DNA adalah untai atas empat huruf dan subbarisan memodelkan apa yang tersisa setelah sebagian posisi hilang; dalam pemampatan data dan pengindeksan teks, menghitung subbarisan atau substring yang berbeda mengukur seberapa beragam isi sebuah teks. Dan dalam soal olimpiade, gagasan “kemunculan terakhir” yang akan kita turunkan juga menghitung subhimpunan berbeda dari multihimpunan, jalan berlabel yang berbeda, dan untai berbeda yang dapat diperoleh dengan penghapusan.

9.2 Menghitung Tanpa Menghitung Ganda

9.2.1 Subbarisan

Subbarisan (subsequence) dari untai $s = s_1s_2 \dots s_n$ adalah apa yang tersisa setelah menghapus sembarang himpunan posisi, dengan urutan sisanya dipertahankan. AGH adalah subbarisan dari ABCDEFGH; AHG bukan, karena H muncul sesudah G pada untai aslinya. Untai kosong, ditulis ε , adalah subbarisan dari setiap untai (hapus semuanya). Kita selalu menghitung *untai*, bukan himpunan posisi: dua himpunan posisi berbeda yang menghasilkan untai yang sama dihitung satu kali.

9.2.2 Keadaan, sebagai sebuah kalimat

Setiap program penghitungan yang salah berawal dari keadaan yang kabur; metode pemrograman dinamis [2] hanya sebaik keadaan yang diberikan kepadanya. Karena itu, sebelum rumus apa pun, kita menuliskan satu kalimat:

DEFINISI. $dp[i]$ adalah banyaknya subbarisan berbeda dari prefiks $s_1s_2 \dots s_i$, termasuk subbarisan kosong. Kita menulis D_i untuk *himpunan* subbarisan-subbarisan itu, sehingga $|D_i| = dp[i]$, dan $D_0 = \{\varepsilon\}$.

Buatlah kalimat itu konkret seketika. Untuk untai AB, prefiks sepanjang 0 kosong dan $D_0 = \{\varepsilon\}$, sehingga $dp[0] = 1$. Prefiks A memiliki $D_1 = \{\varepsilon, A\}$ dan $dp[1] = 2$. Prefiks AB memiliki $D_2 = \{\varepsilon, A, B, AB\}$ dan $dp[2] = 4$. Menuliskan dua atau tiga himpunan pertama dengan tangan, sebelum rekurens apa pun, adalah cara termurah untuk menemukan bahwa sebuah definisi masih ambigu: jika Anda ragu apakah ε termasuk dalam D_0 , kalimatnya belum cukup berbicara. Semua yang mengikuti adalah akibat dari kalimat ini. Jika sebuah langkah kelak tampak memerlukan sesuatu yang tidak dinyatakan kalimat itu, kalimatnya yang salah dan harus diperbaiki lebih dulu.

9.2.3 Menurunkan rekurens

Misalkan $c = s_i$ adalah huruf baru. Sebuah subbarisan dari prefiks sepanjang i tidak memakai posisi i , dan dengan demikian merupakan subbarisan prefiks yang lebih pendek, atau memakainya, dan dengan demikian merupakan subbarisan prefiks yang lebih pendek yang diikuti c . Jadi

$$D_i = D_{i-1} \cup \{tc : t \in D_{i-1}\} \quad (9.1)$$

Himpunan kedua memiliki tepat $|D_{i-1}|$ anggota, karena menambahkan huruf yang sama pada untai-untai yang berbeda menghasilkan untai-untai yang berbeda. Jika kedua himpunan pada [Persamaan \(9.1\)](#) saling lepas, $dp[i]$ akan bernilai $2 dp[i-1]$. Keduanya memang saling lepas ketika c belum pernah muncul sebelumnya: tidak ada untai di D_{i-1} yang memuat c , sedangkan setiap untai di himpunan kedua berakhir dengan c .

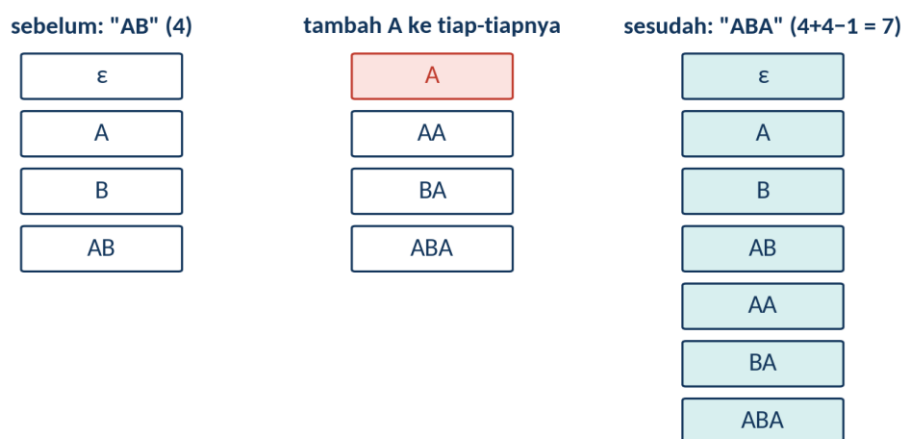
Ketika c sudah pernah muncul, misalkan p posisi kemunculan terakhirnya sebelum ini ($p < i$, $s_p = c$). Irisan kedua himpunan terdiri atas untai-untai di D_{i-1} yang berakhir dengan c . Masing-masing dapat dibentuk dengan mengambil c terakhirnya dari posisi p , c terakhir yang tersedia, dan huruf-huruf sisanya dari posisi sebelum p ; jadi irisannya adalah $\{t c : t \in D_{p-1}\}$, yang memiliki tepat $|D_{p-1}|$ anggota. Menurut prinsip inklusi-eksklusi [3],

$$\begin{aligned} dp[i] &= 2 dp[i-1] - dp[p-1] \quad (\text{jika } c \text{ pernah muncul di } p), & dp[i] \\ &= 2 dp[i-1] \quad (\text{jika tidak}) \end{aligned} \quad (9.2)$$

dengan $dp[0] = 1$. Jawaban untuk seluruh untai adalah $dp[n]$.

WAWASAN KUNCI: Kemunculan terakhir memiliki duplikatnya. Setiap duplikat yang diciptakan oleh c yang baru sudah pernah diciptakan ketika c muncul di posisi p , dan saat itu tepat $dp[p-1]$ untai yang diciptakan. Kemunculan c yang lebih awal tidak memerlukan koreksi sendiri: duplikatnya sudah dibuang ketika posisi p diproses.

Gambar 9.1 memperlihatkan langkah dari AB ke ABA. Keempat subbarisan lama dipertahankan, empat yang baru dibuat dengan menambahkan A, dan tepat satu dari yang baru, yaitu untai A yang dibuat dari ϵ , sudah ada sebelumnya. Di sini $p = 1$, sehingga koreksinya $dp[0] = 1$ dan hitungannya $4 + 4 - 1 = 7$.



"A" terbentuk dua kali: dulu ($\epsilon + A$ pertama) dan sekarang ($\epsilon + A$ baru); kurangi $dp[\text{last}-1] = dp[0] = 1$.

Gambar 9.1 Dari AB ke ABA: menambahkan A pada setiap subbarisan lama menciptakan satu duplikat, yaitu untai A, yang sudah dihasilkan oleh A sebelumnya.

9.2.4 Dua kekeliruan yang menggoda

Ada pelajaran berharga dalam melihat apa yang salah tanpa pembuktian. Rekurens menggoda yang pertama adalah “gandakan, lalu kurangi satu jika hurufnya pernah muncul”, dengan anggapan bahwa hanya untai satu huruf yang terulang. Pada ABA cara ini kebetulan benar, memberikan $4 \cdot 2 - 1 = 7$. Pada ABAB cara ini memberikan $7 \cdot 2 - 1 = 13$, bukan 12, karena B yang kedua menciptakan ulang *dua* untai lama, B dan AB, seperti yang diperlihatkan tabel pada Subbab 9.3.2. Koreksinya bukan konstanta: koreksinya adalah banyaknya untai yang dapat mendahului huruf itu pada kemunculan sebelumnya, $dp[p-1]$.

Rekurens menggoda yang kedua mengurangkan $dp[q-1]$ untuk *setiap* kemunculan q sebelumnya dari huruf itu, bukan hanya yang terakhir. Pada AAA cara ini memberikan $dp[3] = 2 \cdot 3 - dp[1] - dp[0] = 6 - 2 - 1 = 3$, padahal jawaban yang benar 4. Untai-untai yang dibuang oleh

kemunculan yang lebih lama sudah dibuang sekali ketika kemunculan yang lebih baru diproses; membuangnya lagi berarti menghitungnya sebagai kekurangan. Argumen himpunan pada Subbab 9.2.3 mencegah kedua kekeliruan itu: argumen itu menamai irisannya dengan tepat, dan irisan itu digambarkan oleh kemunculan terakhir saja.

9.2.5 Penurunan kedua: menghitung menurut huruf terakhir

Keadaan yang berbeda memberikan angka yang sama dan kadang lebih mudah diperluas. Misalkan $e[c]$ adalah banyaknya subbarisan *tak kosong* berbeda, sejauh yang telah dibaca, yang berakhir dengan huruf c . Ketika huruf berikutnya pada untai adalah c , subbarisan-subbarisan berbeda yang berakhir dengan c kini tepat untai-untai tc dengan t sembarang subbarisan berbeda yang telah terlihat, termasuk yang kosong. Jadi

$$e[c] \leftarrow 1 + \sum_d e[d], \quad \text{jawaban} = 1 + \sum_d e[d] \quad (9.3)$$

dengan penjumlahan atas ke-26 huruf dan angka 1 mewakili untai kosong. Pembaruan itu *mengganti* $e[c]$, bukan menambahinya: untai-untai lama yang berakhir dengan c semuanya termasuk di antara yang baru, dan dengan cara itulah duplikatnya lenyap. Pada ABAB, nilai $(e[A], e[B])$ berturut-turut $(1,0)$, $(1,2)$, $(4,2)$, dan $(4,7)$, dan jawabannya $1 + 4 + 7 = 12$, sama seperti sebelumnya. **Persamaan** (9.3) berbiaya 26 langkah per huruf alih-alih satu, yang masih cepat, dan secara alami dapat diperluas ke pertanyaan seperti “ada berapa subbarisan berbeda yang berakhir dengan huruf vokal?”.

9.2.6 Aritmetika modulo bilangan prima

Hitungannya tumbuh seperti 2^n , sehingga soal memintanya modulo $M = 10^9 + 7$. Penjumlahan dan perkalian dapat dipertukarkan dengan pengambilan sisa bagi, sehingga kita boleh mereduksi setelah setiap langkah [3]. Pengurangan memerlukan kehati-hatian: di C++, operator `%` mempertahankan tanda operan kirinya, sehingga $(a - b) \bmod M$ yang dihitung sebagai $(a - b) \% M$ bernilai negatif setiap kali $a < b$, dan hal itu terjadi terus-menerus pada nilai yang sudah direduksi. Bentuk yang aman adalah

$$(a - b) \bmod M = ((a - b) + M) \bmod M \quad \text{untuk } 0 \leq a, b < M \quad (9.4)$$

Ukuran nilai antara juga penting: $2 \cdot dp[i - 1] < 2 \cdot 10^9 + 14$ muat dalam bilangan bulat bertanda 64 bit dengan ruang yang lapang (dan *tidak* muat dalam `int` 32 bit).

9.3 Soal Pemanasan

9.3.1 Menghitung cara: kemunculan ABC

Pertama, kerabatnya yang lebih mudah: hitung *cara* sebuah pola muncul sebagai subbarisan, dengan himpunan posisi yang berbeda dihitung terpisah. Misalkan $f[j]$ banyaknya cara mencocokkan j huruf pertama pola dengan teks yang sudah dibaca. Ketika huruf teks berikutnya sama dengan huruf pola ke- j , setiap cara mencocokkan $j - 1$ huruf pertama dapat diperpanjang, sehingga $f[j]$ bertambah $f[j - 1]$. Memperbarui j dari besar ke kecil menjamin satu huruf teks dipakai paling banyak sekali per cara, alasan yang sama mengapa perulangan *knapsack* 0/1 berjalan menurun. **Kode 9.1** menghitung ABC pada empat untai.

Kode 9.1 `abc_count.cpp`: menghitung kemunculan pola sebagai subbarisan.

```
// How many times does "ABC" occur as a subsequence? (positions count, mod p)
#include <iostream>
#include <string>
using namespace std;
int main() {
    const long long MOD = 1000000007;
    string pat = "ABC";
    for (string s : {"ABC", "AABBCC", "ABCABC", "CBA"}) {
        // f[j] = number of ways to match the first j letters of pat so far
        long long f[4] = {1, 0, 0, 0};
        for (char ch : s)
            for (int j = 3; j >= 1; j--) // downward: each letter used once per way
                if (ch == pat[j - 1]) f[j] = (f[j] + f[j - 1]) % MOD;
        cout << s << ": " << f[3] << "\n";
    }
}
```

```
ABC: 1
AABBCC: 8
ABCABC: 4
CBA: 0
```

Pada AABBCC ada $2 \cdot 2 \cdot 2 = 8$ pilihan satu A, satu B, dan satu C, semuanya dalam urutan yang benar. Pada ABCABC keempat caranya adalah triple posisi (1,2,3), (1,2,6), (1,5,6), dan (4,5,6). Di sini tidak ada yang dibuang sebagai duplikat, karena yang kita hitung adalah posisi; studi kasus menghitung untai, dan itulah seluruh perbedaannya.

9.3.2 Mengamati rekurens bekerja

Kode 9.2 mencetak setiap langkah [Persamaan \(9.2\)](#) untuk ABAB, tanpa modulus agar angkanya dapat diperiksa dengan tangan.

Kode 9.2 `dsub_table.cpp`: tabel DSUBSEQ untuk ABAB, langkah demi langkah.

```
// Prints the DSUBSEQ table step by step for one string.
#include <iomanip>
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s = "ABAB";
    long long dp[5];
    int last[26] = {};
    dp[0] = 1;
    cout << " i char  2*dp[i-1]  minus  dp[i]\n";
    for (int i = 1; i <= 4; i++) {
        int c = s[i - 1] - 'A';
        long long twice = 2 * dp[i - 1], minus = last[c] ? dp[last[c] - 1] : 0;
        dp[i] = twice - minus;
        last[c] = i;
        cout << setw(2) << i << setw(5) << s[i - 1] << setw(11) << twice << setw(7)
            << minus << setw(7) << dp[i] << "\n";
    }
}
```

i	char	2*dp[i-1]	minus	dp[i]
1	A	2	0	2
2	B	4	0	4
3	A	8	1	7
4	B	14	2	12

Pada $i = 3$ huruf A sebelumnya berada di posisi 1, sehingga koreksinya $dp[0] = 1$. Pada $i = 4$ huruf B sebelumnya berada di posisi 2, sehingga koreksinya $dp[1] = 2$: untai B dan AB sudah

dihasilkan oleh B pertama dan akan dihasilkan lagi. Kedua belas subbarisan ABAB adalah ε , A, B, AA, AB, BA, BB, AAB, ABA, ABB, BAB, dan ABAB; mendaftarnya adalah pemeriksaan lima menit yang sangat berharga, dan sekaligus cara terbaik untuk benar-benar merasakan mengapa koreksinya bernilai dua, bukan satu.

9.3.3 BANANA, kata di papan tulis

Kembali ke cerita pada Subbab 9.1. Untuk BANANA rekurens memberikan, huruf demi huruf: B baru, $dp[1] = 2$; A baru, $dp[2] = 4$; N baru, $dp[3] = 8$. A yang kedua memiliki kemunculan sebelumnya di posisi 2, sehingga $dp[4] = 16 - dp[1] = 14$. N yang kedua memiliki kemunculan sebelumnya di posisi 3, sehingga $dp[5] = 28 - dp[2] = 24$. A yang ketiga memiliki kemunculan sebelumnya di posisi 4, sehingga $dp[6] = 48 - dp[3] = 40$. Dari 64 cara mencoret huruf, hanya 40 kata berbeda yang dihasilkan, termasuk kata kosong; program penyelesaian pada Subbab 9.4 mencetak tepat 40 untuk masukan ini. Kelas itu akan memerlukan satu sore yang panjang untuk mendaftar semuanya, sedangkan rekurensnya hanya memerlukan enam baris aritmetika.

9.3.4 Dari cara ke untai

Letakkan kedua program pemanasan berdampingan, karena perbedaannya adalah pelajaran utama bab ini. Pada **Kode 9.1** sebuah cara adalah himpunan posisi, dan dua himpunan yang berbeda adalah dua cara yang berbeda walaupun keduanya mengeja huruf yang sama; tidak ada yang perlu dibuang sebagai duplikat, dan rekurensnya cukup menjumlahkan. Pada **Kode 9.2** sebuah objek adalah *untai*, dan seluruh kesulitannya terletak pada kenyataan bahwa banyak himpunan posisi mengeja untai yang sama. Karena itu rekurens harus memilih satu himpunan posisi *baku* (kanonis) untuk setiap untai dan hanya menghitung yang satu itu.

Aturan kemunculan terakhir adalah pilihan semacam itu, walaupun tersembunyi di dalam pembuktiannya. Di antara semua cara mengeja sebuah subbarisan, pilih cara yang mengambil setiap huruf *selambat* mungkin: huruf terakhir pada posisi terakhir yang mungkin, huruf sebelumnya pada posisi terakhir yang mungkin sebelum itu, dan seterusnya. Setiap untai berbeda memiliki tepat satu ejaan rakus seperti itu, sehingga menghitung ejaan rakus sama dengan menghitung untai. Ketika huruf c tiba di posisi i , untai-untai yang ejaan rakusnya berakhir di i adalah semua untai tc ; untai yang tadinya berakhir di c sebelumnya kini berpindah ke i , dan perpindahan itulah tepatnya pengurangan $dp[p - 1]$. Memandang sebuah rekurens penghitungan sebagai “hitung satu wakil baku per objek” adalah cara paling andal untuk merancang rekurens-rekurens baru.

9.4 Studi Kasus: DSUBSEQ

9.4.1 Soal dengan kata-kata kita sendiri

Soalnya adalah SPOJ DSUBSEQ [1]. Baris pertama memuat banyaknya kasus uji T , paling banyak 100. Setiap kasus uji adalah satu untai huruf kapital A–Z, sepanjang paling banyak 10^5 . Untuk setiap untai, cetak banyaknya subbarisan berbeda, *termasuk yang kosong*, modulo 1 000 000 007.

Frasa “termasuk yang kosong” layak mendapat barisnya sendiri. Frasa itu menggeser setiap jawaban tepat satu, sehingga program yang tidak menghitung subbarisan kosong gagal pada setiap uji. Contoh resmi di bawah ini menuntaskan pertanyaannya: AAA memiliki subbarisan berbeda ε , A, AA, dan AAA, empat seluruhnya. Kalimat keadaan pada Subbab 9.2.2 sejak awal

sudah memasukkan untai kosong, sehingga rekurensnya langsung cocok dengan soal tanpa penyesuaian apa pun.

9.4.2 Contoh masukan dan keluaran

```
2
AAA
ABCDEFGG
```

```
4
128
```

ABCDEFGG memiliki tujuh huruf yang berbeda, sehingga tidak ada dua pilihan posisi yang menghasilkan untai sama dan semua $2^7 = 128$ pilihan berbeda: [Persamaan \(9.2\)](#) tidak pernah mengurangkan. Untuk AAA tabelnya $dp = 1, 2, 3, 4$: setiap A baru menggandakan lalu mengurangkan hitungan sebelum A sebelumnya.

9.4.3 Merancang penyelesaian

1. Simpan `last[c]`, posisi terbaru huruf c , atau 0 jika c belum muncul. Atur ulang untuk setiap kasus uji.
2. Tetapkan $dp[0] = 1$ lalu pindai untainya. Untuk posisi i dengan huruf c : hitung $2 dp[i - 1] \bmod M$; jika `last[c]` bukan 0, kurangkan $dp[last[c] - 1]$ dengan [Persamaan \(9.4\)](#); lalu tetapkan `last[c] = i`.
3. Cetak $dp[n]$.

Urutan dua tindakan terakhir pada langkah 2 penting: `last[c]` harus dibaca *sebelum* ditimpa dengan posisi saat ini.

Larik `dp` disimpan utuh hanya karena koreksinya memerlukan $dp[p - 1]$ untuk posisi lama p . Satu pengamatan kecil menghilangkan kebutuhan itu pun: nilai lama yang pernah diperlukan hanyalah $dp[last[c] - 1]$, satu per huruf. Jadi kita boleh menyimpan, untuk setiap huruf, `before[c]`, nilai dp tepat sebelum kemunculan terbarunya, dan memperbaruinya menjadi $dp[i - 1]$ saat ini setiap kali huruf itu muncul. Seluruh algoritma lalu hanya memerlukan 26 bilangan dan satu nilai berjalan, tidak bergantung pada n . Untuk DSUBSEQ lariknya cukup kecil sehingga penghalusan ini tidak perlu, tetapi gagasan yang sama penting ketika untainya dialirkan dan terlalu panjang untuk disimpan, dan itulah yang dilakukan [Persamaan \(9.3\)](#) dalam bentuk lain.

Biayanya $O(n)$ per untai dan paling banyak 10^7 langkah untuk seluruh masukan, ditambah membaca 10^7 karakter. Memorinya satu larik berisi $10^5 + 2$ nilai 64 bit, di bawah satu megabita.

9.4.4 Penyelesaian lengkap

Kode 9.3 adalah program lengkap yang tercatat *Accepted* di SPOJ. Blok P50 memuat koreksi dan pembaruan `last`.

Kode 9.3 `DSUBSEQ.cpp`: subbarisan berbeda dengan koreksi kemunculan terakhir.

```
#include <stdio>
#include <cstring>
using namespace std;
const int MOD=1000000007;
long long dp[100002];
int last[26];
int main() {
    int t; if(scanf("%d",&t)!=1) return 0;
    while(t--){
        char s[100001]; if(scanf("%100000s",s+1)!=1) return 0;
        int n=strlen(s+1);
        memset(last,0,sizeof last);
        dp[0]=1;
        for(int i=1;i<=n;i++){
            int c=s[i]-'A';
            dp[i]=2*dp[i-1]%MOD;
            // <P50>
            // If c was seen before at position last[c], remove the dp[last[c]-1]
            // subsequences counted twice (keep the value non-negative), then record i.
            // >>> YOUR TURN (Public Edition): write this part yourself, following the hint
            // above. <<<
            // </P50>
        }
        printf("%lld\n",dp[n]%MOD);
    }
}
```

9.4.5 Baris demi baris

Variabel global. `MOD` adalah bilangan prima $10^9 + 7$. `dp` adalah larik global bertipe `long long`, cukup besar untuk untai terpanjang ditambah posisi 0. `last` menyimpan satu posisi per huruf.

Membaca. `scanf("%d", &t)` membaca banyaknya uji; setiap untai dibaca ke `s + 1` agar posisinya berjalan dari 1 sampai `n`, sesuai dengan matematikanya. Batas lebar `%100000s` menjamin baris yang lebih panjang tidak pernah meluapkan *buffer*, dan pemeriksaan nilai kembalian menghentikan program dengan rapi di akhir masukan. *Buffer* itu tinggal di dalam perulangan, di *stack*; dengan ukuran sekitar 100 KB ia jauh di dalam batas *stack* bawaan mana pun.

Per uji. `memset` mengosongkan `last`, `dp[0] = 1` adalah subbarisan kosong, dan perulangannya menerapkan *Persamaan* (9.2): pertama penggandaan, direduksi modulo M , lalu (di blok P50) koreksi ketika `last[c]` terisi, ditulis sebagai *Persamaan* (9.4), dan akhirnya pembaruan `last[c]`.

Membaca masukan adalah bagian yang terasa dari waktu jalan: hingga 10^7 karakter. `scanf("%s")` cukup cepat di sini, hanya beberapa perseratus detik, karena menyalin satu kata utuh sekaligus. Membaca karakter demi karakter dengan `cin >> ch` akan beberapa kali lebih lambat, sedangkan `getline` ke `std::string` lalu disalin juga memadai. Bila ukuran masukan tumbuh sepuluh kali lagi, pembaca blok dari Bab 7 adalah pilihan yang aman.

Keluaran. `dp[n]` sudah tereduksi; `% MOD` tambahan tidak berbahaya.

9.4.6 Mengapa modulus tidak berbahaya

Modulus mungkin tampak membahayakan pengurangan pada *Persamaan* (9.2): nilai sebenarnya $2dp[i-1] - dp[p-1]$ selalu positif, karena himpunan yang digandakan memuat irisannya, tetapi setelah direduksi bilangan yang tersimpan hanyalah sisa bagi, dan $2dp[i-1]$ yang tersimpan bisa lebih kecil daripada $dp[p-1]$ yang tersimpan. Itu bukan kesalahan matematika. Sisa bagi menghormati penjumlahan, pengurangan, dan perkalian, sehingga sisa bagi jawaban sebenarnya sama dengan sisa bagi hasil perhitungan atas sisa-sisa bagi, asalkan hasilnya

dikembalikan ke rentang $0..M - 1$. **Persamaan** (9.4) melakukan tepat hal itu. Pilihan $M = 10^9 + 7$ bersifat konvensional: bilangan itu prima, berada tepat di bawah 2^{30} , sehingga jumlah dua sisa bagi muat dalam bilangan bulat bertanda 32 bit dan hasil kali dua sisa bagi muat dalam 64 bit. Keprimaan tidak berperan dalam soal ini, yang tidak pernah membagi, tetapi berperan dalam banyak soal penghitungan yang memerlukan invers modular, sehingga modulus yang sama dipakai di mana-mana.

9.4.7 Jebakan

JEBAKAN: Menulis $(dp[i] - dp[last[c]-1]) \% MOD$. Ketika nilai $dp[i]$ yang tereduksi kebetulan lebih kecil daripada yang dikurangkan, hasilnya negatif, dan setiap nilai berikutnya mewarisi kesalahan itu. Perbaikannya satu $+ MOD$, seperti pada **Persamaan** (9.4).

- **Kosong atau tidak.** Carilah kata “kosong” atau “tak kosong” pada pernyataan soal. Di sini subbarisan kosong ikut dihitung, sehingga jawabannya $dp[n]$, bukan $dp[n] - 1$.
- **Lupa mengatur ulang `last`.** Posisi dari untai sebelumnya akan membuat huruf-huruf pertama untai berikutnya mengurangkan koreksi yang tidak ada.
- **Mengurangkan untuk setiap kemunculan sebelumnya.** Hanya yang terbaru yang berlaku; wawasan kunci di atas menjelaskan alasannya.
- **`dp` 32 bit.** $2 dp[i - 1]$ dapat mencapai sekitar $2 \cdot 10^9$, di luar jangkauan `int`.

9.5 Uji Stres terhadap *Brute Force*

Untuk untai pendek kita benar-benar dapat mendaftar setiap subbarisan. **Kode 9.4** mencoba semua 2^n pilihan posisi, membangun setiap untai, dan memasukkannya ke `std::set`, yang membuang duplikat untuk kita; jawabannya adalah ukuran himpunan itu.

Kode 9.4 `brute.cpp`: mendaftar setiap subbarisan ke dalam sebuah himpunan.

```
// Reference for DSUBSEQ: put every subsequence (as a string) into a set.
#include <iostream>
#include <set>
#include <string>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        string s;
        cin >> s;
        int n = s.size();
        set<string> all;
        for (int mask = 0; mask < (1 << n); mask++) {
            string sub;
            for (int i = 0; i < n; i++)
                if (mask >> i & 1) sub += s[i];
            all.insert(sub); // the empty string is included (mask 0)
        }
        cout << all.size() << "\n";
    }
}
```

Pembangkit pada **Kode 9.5** menuliskan hingga enam untai sepanjang paling banyak 14 atas alfabet berisi satu sampai empat huruf. Alfabet kecil itu penting: dengan 26 huruf, untai pendek acak jarang mengulang huruf dan koreksi pada **Persamaan** (9.2) hampir tidak pernah teruji.

Kode 9.5 `gen.py`: untai pendek acak atas alfabet kecil.

```
# Random small DSUBSEQ inputs: python3 gen.py SEED
import random, sys
random.seed(int(sys.argv[1]))
T = random.randint(1, 6)
print(T)
for _ in range(T):
    k = random.randint(1, 4)
    print(''.join(random.choice('ABCD'[:k]) for _ in range(random.randint(1, 14))))
```

all 300 random tests passed

Ada gunanya memikirkan *bug mana* yang akan ditangkap uji ini. Program yang lupa untai kosong mencetak setiap jawaban kurang satu dan gagal pada perbandingan pertama. Program yang mengurangi konstanta, atau mengurangi untuk setiap kemunculan sebelumnya, gagal segera setelah sebuah huruf muncul tiga kali atau dua huruf berselang-seling, yang terjadi pada hampir setiap untai bangkitan atas dua atau tiga huruf. Program yang lupa mengatur ulang **last** di antara kasus uji gagal pada untai kedua dari berkas pertama yang memuat lebih dari satu untai. **+ MOD** yang hilang adalah satu-satunya *bug* yang *tidak dapat* ditangkap uji ini, karena angka-angka untuk untai sepanjang 14 huruf tidak pernah mencapai modulus. Untuk *bug* itu uji yang tepat berbeda: untai panjang, misalnya 5000 huruf, dibandingkan dengan implementasi kedua yang mandiri seperti **Persamaan** (9.3), ketika kedua program mereduksi modulo M . Mengetahui batas sebuah uji sama pentingnya dengan menjalankannya.

Masukan sah terbesar, 100 untai berisi 10^5 huruf acak, selesai dalam sekitar 0,11 detik.

9.6 Pola Ini dalam Soal Lain

Gagasan “kemunculan terakhir memiliki duplikatnya” adalah sebuah pola, bukan trik untuk satu soal. Tiga kerabatnya muncul secara teratur.

Subhimpunan berbeda dari multihimpunan menurut jumlahnya. Ketika butir-butir bernilai sama tidak dapat dibedakan, hitungan jumlah-subhimpunan per nilai harus memperlakukan sekelompok k butir yang sama sebagai “ambil 0, 1, ..., k di antaranya”, bukan sebagai k pilihan ya-atau-tidak yang terpisah; kalau tidak, multihimpunan yang sama terhitung beberapa kali. Pengelompokan itu memainkan peran yang dimainkan kemunculan terakhir di sini: setiap objek ditugaskan pada tepat satu cara menghasilkannya.

Untai berbeda setelah penghapusan dengan suatu aturan. Soal seperti “ada berapa untai berbeda yang dapat diperoleh dengan menghapus tepat k karakter” atau “dengan menghapus karakter sehingga tidak ada dua huruf sama yang bersebelahan” diselesaikan dengan menambahkan informasi tambahan pada keadaan, persis seperti Soal Latihan 4 menambahkan panjang. Suku koreksinya mempertahankan bentuknya: kurangkan apa yang sudah dihasilkan kemunculan sebelumnya *dengan informasi tambahan yang sama*.

Menghitung substring berbeda. Potongan yang bersambung adalah cerita lain, karena substring ditentukan oleh awal dan akhirnya, bukan oleh pilihan posisi yang bebas. DISUBSTR dari SPOJ, pokok bahasan Bab 13, menghitungnya dengan struktur sufiks; bahaya penghitungan gandanya sama, dan obatnya lagi-lagi menugaskan setiap objek berbeda pada satu cara baku untuk menghasilkannya.

Maka resep umumnya: (1) gambarkan objek-objeknya sebagai himpunan; (2) gambarkan bagaimana himpunan itu tumbuh ketika masukan bertambah satu elemen, seperti pada **Persamaan** (9.1); (3) namai irisannya dengan tepat dan hitunglah; (4) baru kemudian tulis kodenya. Langkah (1) sampai (3) muat dalam setengah halaman dan mengubah tebakan menjadi

pembuktian. Bila langkah (3) sulit, carilah wakil baku, seperti pada Subbab 9.3.4: jika setiap objek dapat dihasilkan dengan tepat satu cara yang disepakati, menghitung cara-cara itu sama dengan menghitung objeknya, dan irisannya tidak perlu dihitung sama sekali.

9.7 Taktik pada Hari Lomba

- **Tulis kalimat keadaan lebih dulu** [4]. “dp ke- i adalah banyaknya ... berbeda dari i ... pertama, termasuk yang kosong.” Jika Anda tidak dapat menuntaskan kalimat itu, Anda belum siap menulis rekurensya.
- **Buktikan keberbedaan dengan himpunan.** Tuliskan persamaan himpunan seperti [Persamaan](#) (9.1), kenali irisannya, dan hitunglah. Lima baris pembuktian menghemat satu jam mencari *bug*.
- **Jadikan + MOD refleks.** Setiap pengurangan di bawah modulus mendapatkannya, bahkan ketika Anda yakin nilainya sudah terurut.
- **Enumerasikan untuk $n \leq 12$.** Enumerasi *brute force* ke dalam himpunan ditulis dalam tiga menit dan menangkap kesalahan meleset-satu pada untai kosong, kemunculan pertama, dan modulus.
- **Pakai alfabet mungil dalam pengujian.** Pengulangan adalah tempat tinggal *bug* penghitungan; bangkitkan pengulangan dengan sengaja.
- **Periksa silang dua penurunan.** Ketika dua keadaan yang berbeda, seperti [Persamaan](#) (9.2) dan [Persamaan](#) (9.3), memberikan angka yang sama pada masukan acak, keduanya sangat mungkin benar; cara ini menangkap *bug* yang tidak terjangkau *brute force* kecil.

9.8 Ringkasan Bab

- Menghitung objek berbeda berbeda dari menghitung cara: ketiadaan duplikat harus dibuktikan, bukan diharapkan.
- Definisi keadaan adalah sebuah kalimat utuh; persamaan himpunan [Persamaan](#) (9.1) mengikutinya.
- Untuk subbarisan berbeda, irisan yang diciptakan huruf c tepat sama dengan apa yang diciptakan kemunculan sebelumnya, sehingga diperoleh [Persamaan](#) (9.2): gandakan, lalu kurangi $dp[p - 1]$.
- Menghitung objek berbeda berarti menghitung satu wakil baku per objek; untuk subbarisan, ejaan bakunya mengambil setiap huruf selambat mungkin.
- Pengurangan modular memerlukan $+M$, [Persamaan](#) (9.4); nilai antara memerlukan 64 bit.
- Untai pendek atas alfabet kecil, dienumerasikan ke dalam himpunan, adalah *brute force* yang tepat; penurunan kedua seperti [Persamaan](#) (9.3) meliputi untai-untai panjang.

9.9 Soal Latihan

1. Hitung dp untuk ZZZZ dengan tangan. Berapa jawabannya untuk untai berisi n huruf yang sama, dan mengapa?
2. Bagaimana jawabannya berubah jika subbarisan kosong *tidak* boleh dihitung? Ubah satu baris pada **Kode 9.3**.
3. Buktikan dengan induksi pada i bahwa [Persamaan](#) (9.2) memberikan $\$D_i\$$ untuk setiap i .
4. Perluas metodenya untuk menghitung subbarisan berbeda yang panjangnya tepat k . Berikan rekurensya.

5. Untai kini boleh memuat huruf kecil, huruf kapital, dan angka. Apa yang harus berubah pada program?
6. Gunakan **Persamaan (9.3)** untuk menghitung jawaban bagi BANANA dan bandingkan dengan Subbab 9.3.3.
7. Untuk untaian sepanjang n atas alfabet dua huruf, berapa jawaban terbesar yang mungkin? Cobalah n kecil dengan *brute force* dan tebak polanya.

9.10 Penyelesaian Soal Latihan

1. $dp[0..4] = 1, 2, 3, 4, 5$. Pada setiap langkah penggandaan menambahkan semua untaian lama dengan satu Z lagi, dan koreksinya membuang semuanya kecuali yang terpanjang, karena setiap untaian Z yang lebih pendek sudah ada. Subbarisan berbeda dari n huruf yang sama adalah untaian berisi $0, 1, \dots, n$ huruf, sehingga jawabannya $n + 1$.
2. Cetak $(dp[n] - 1 + \text{MOD}) \% \text{MOD}$. Rekurensnya sendiri harus tetap menghitung untaian kosong, karena untaian kosonglah yang membuat huruf tunggal muncul sebagai subbarisan baru.
3. Basis: $D_0 = \{\varepsilon\}$ dan $dp[0] = 1$. Langkah: andaikan $D_j = D_{j-1} \cup \{c\}$ untuk semua $j < i$. **Persamaan (9.1)** menuliskan D_i sebagai gabungan D_{i-1} dan sebuah himpunan berukuran sama. Jika $c = s_i$ baru, kedua himpunan saling lepas (hanya yang kedua memuat c), sehingga $|D_i| = 2|D_{i-1}| = 2dp[i-1]$. Jika tidak, irisannya adalah $\{t \mid t \in D_{i-1}, t \text{ berakhir dengan } s_i\}$ seperti ditunjukkan pada Subbab 9.2.3, berukuran $|D_{i-1}| = dp[i-1]$ menurut hipotesis, dan inklusi-eksklusi memberikan **Persamaan (9.2)**.
4. Misalkan $dp[i][k]$ menghitung subbarisan berbeda sepanjang k dari prefiks sepanjang i . Argumen himpunan yang sama, dibatasi pada panjang k , memberikan $dp[i][k] = dp[i-1][k] + dp[i-1][k-1] - dp[p-1][k-1]$ (suku terakhir hanya jika c pernah muncul di p), dengan $dp[i][0] = 1$. Biayanya $O(nk)$.
5. Hanya alfabetnya: buat `last` cukup besar untuk setiap kode karakter, misalnya `int last[128]`, dan indekslah dengan `(unsigned char)s[i]` alih-alih `s[i] - 'A'`. Rekurensnya tidak peduli huruf apa yang dipakai.
6. Dengan $(e[B], e[A], e[N])$: setelah B, (1,0,0); setelah A, (1,2,0); setelah N, (1,2,4); setelah A, $e[A] = 1 + 7 = 8$, memberikan (1,8,4); setelah N, $e[N] = 1 + 13 = 14$, memberikan (1,8,14); setelah A, $e[A] = 1 + 23 = 24$, memberikan (1,24,14). Jawabannya $1 + 1 + 24 + 14 = 40$, sama dengan hasil rekurens.
7. Percobaan dengan *brute force* untuk $n \leq 12$ menunjukkan bahwa untaian berselang-seling ABAB... dan BABA... memberikan jawaban terbesar. Pada untaian itu kemunculan sebelumnya dari setiap huruf baru berada dua posisi di belakang, sehingga $dp[i] = 2dp[i-1] - dp[i-3]$ dengan $dp[0..2] = 1, 2, 4$; ini menghasilkan 1, 2, 4, 7, 12, 20, 33, ..., yaitu bilangan Fibonacci dikurangi satu: $dp[n] = F_{n+3} - 1$ dengan $F_1 = F_2 = 1$. Secara intuitif, huruf yang sama dengan pendahulunya membuat koreksinya sebesar mungkin ($dp[i-2]$), dan selang-seling menghindarinya pada setiap langkah.

Daftar Pustaka

- [1] Sphere Online Judge, "Problem DSUBSEQ." [Online]. Available: <https://www.spoj.com/problems/DSUBSEQ/>
- [2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022, ch. 14.
- [3] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1994.

- [4] S. Halim, F. Halim, and S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.

Bab 10

Pemrograman Dinamis *Bitmask* dan Profil — CFONISMG

Ketika sebuah soal menyatakan “paling banyak 16 orang” atau “paling banyak 4 kolom”, juri sedang mengirim pesan: algoritma eksponensial dipersilakan, asalkan eksponensialnya *terkendali*. Bab ini mengajarkan dua cara baku untuk mengendalikannya. *Pemrograman dinamis bitmask* menyimpan himpunan berisi hingga dua puluh objek sebagai bit-bit dari satu bilangan bulat, sehingga sebuah himpunan menjadi indeks larik. *Pemrograman dinamis profil* mengisi kisi sel demi sel dan hanya mengingat batas tipis antara bagian yang sudah selesai dan yang belum, yaitu *profil*, alih-alih seluruh kisi. Anda akan belajar menyandikan keduanya, menghitung berapa banyak keadaan yang tercipta sebelum menulis kode apa pun, dan hanya menyimpan keadaan yang benar-benar dapat terjadi. Studi kasusnya, SPOJ CFONISMG [1], menata tempat duduk siswa di kelas agar sahabat duduk berdekatan dan pembuat onar duduk berjauhan; keadaannya memadukan *bitmask* siswa yang sudah duduk dengan profil deretan bangku terakhir, dan soal itu diselesaikan dengan *map* yang hanya menyimpan keadaan yang terjangkau.

10.1 Menata Tempat Duduk Kelas

Seorang guru menyiapkan denah tempat duduk untuk kelas kecil berisi R baris dengan C meja. Ia memiliki daftar siswa yang boleh didudukkan; tidak setiap meja harus terisi, dan tidak ada siswa yang duduk di dua tempat. Ia tahu pasangan siswa mana yang bersahabat: sepasang sahabat yang duduk bersebelahan atau yang satu tepat di belakang yang lain dapat bekerja sama dengan baik, dan ia memberi satu poin untuk setiap pasangan seperti itu. Ia juga tahu pasangan mana yang bermusuhan, yang tidak boleh duduk bersebelahan, dan siswa mana yang suka mengganggu; dua siswa pengganggu juga tidak boleh duduk bersebelahan. Denah mana yang memberikan poin terbanyak?

Cobalah menyelesaikannya dengan tangan untuk kelas 2×3 dan empat siswa, dan Anda segera memahami mengapa komputer diperlukan. Setiap meja dapat kosong atau ditempati siswa mana pun yang belum duduk, sehingga sudah ada ratusan denah; untuk kelas 4×4 dengan enam belas siswa, banyaknya denah memiliki lebih dari dua puluh digit. Namun penalaran sang guru memiliki bentuk yang berguna. Ketika ia memutuskan siapa yang duduk di sebuah meja, hanya dua tetangga yang penting saat itu: meja di depan dan meja di kiri, jika ia mengisi kelas dari baris depan ke belakang dan dari kiri ke kanan. Dan satu-satunya hal tentang masa lalu yang harus ia ingat, selain kedua tetangga itu, adalah *siapa yang sudah duduk*. Kedua ingatan itu kecil, dan itulah seluruh gagasan bab ini. Urutan siswa yang duduk lebih dulu, siapa yang duduk jauh di baris depan, dan berapa lama ia berpikir sebelum memutuskan, semuanya tidak lagi penting; yang penting hanyalah daftar nama yang sudah dicoret dan deretan meja terakhir di belakangnya.

Bentuk soal yang sama muncul dalam memasang ubin persegi panjang di lantai, menanam ladang ketika sebagian tanaman tidak boleh tumbuh berdampingan, memasang antenna di kisi agar yang bertetangga tidak saling mengganggu, dan dalam soal penjual keliling (*travelling salesman*) untuk rute pengiriman melewati belasan pelanggan. Dalam setiap kasus, sebuah himpunan kecil atau sebuah jalur sempit sudah merupakan seluruh masa lalu yang Anda perlukan.

Bab ini mengikuti kedua gagasan itu secara bergantian. Subbab 10.2 memperkenalkan *bitmask* dari nol, menghitung biaya mengenumerasi subhimpunan, dan mengerjakan soal penjual keliling sebagai program *bitmask* teladan. Subbab 10.3 memperkenalkan profil, menghitung pemasangan domino dengannya, menunjukkan cara mengemas profil menjadi satu bilangan, dan yang terpenting, cara memperkirakan dan mengukur banyaknya keadaan sebelum memercayai program eksponensial. Subbab 10.4 memadukan kedua gagasan pada CFONISMG, lengkap dengan penelusuran tangan, program utuh, dan jebakan-jebakannya, dan Subbab 10.5 mengujinya terhadap pencarian runut-balik (*backtracking*) biasa.

10.2 Himpunan sebagai Bilangan

10.2.1 Operasi bit

Nomori objek-objeknya $0, 1, \dots, n-1$. Subhimpunan S disimpan sebagai bilangan bulat yang bit ke- i -nya bernilai 1 tepat ketika objek i termasuk dalam S . Subhimpunan $\{0, 2, 3\}$ menjadi $2^0 + 2^2 + 2^3 = 13$, biner 1101. Dengan demikian setiap subhimpunan dari n objek adalah bilangan bulat antara 0 dan $2^n - 1$, dan operasi himpunan menjadi instruksi mesin tunggal, seperti diperlihatkan **Tabel 10.1**.

Tabel 10.1 Operasi himpunan pada *bitmask*; perulangan terakhir mengunjungi setiap subhimpunan tak kosong dari m .

Operasi himpunan	Ekspresi C++
apakah i di S ?	<code>(mask >> i) & 1</code>
tambahkan i	<code>mask = (1 << i)</code>
buang i	<code>mask & ~(1 << i)</code>
gabungan, irisan	<code>a = b, a & b</code>
ukuran S	<code>__builtin_popcount(mask)</code>
anggota terkecil	<code>__builtin_ctz(mask)</code>
semua subhimpunan m	<code>for (s = m; s; s = (s - 1) & m)</code>

Kerjakan satu contoh secara utuh. Dengan $mask = 13$ (biner 1101, himpunan $\{0, 2, 3\}$): menguji objek 2 menghitung $13 \gg 2 = 3$, biner 11, yang bit terakhirnya 1, jadi 2 adalah anggota; menambahkan objek 1 memberikan $13 | 2 = 15$, biner 1111; membuang objek 3 memberikan 13 dengan bit 3 dimatikan, 5, biner 0101; ukurannya $popcount(13) = 3$ dan anggota terkecilnya $ctz(13) = 0$, banyaknya bit nol di ujung kanan. Setiap operasi itu hanya satu atau dua instruksi mesin, dan itulah sebabnya program *bitmask* sangat cepat dalam praktik.

Baris terakhir tabel perlu dijelaskan. Mengurangkan 1 dari s mengubah bit 1 terendahnya menjadi 0 dan semua bit di bawahnya menjadi 1; $\& m$ kemudian hanya mempertahankan bit-bit yang diizinkan m . Hasilnya adalah subhimpunan m berikutnya yang lebih kecil dalam urutan numerik, sehingga perulangan itu mengunjungi semua subhimpunan tak kosong dari m , masing-masing sekali. Menjalankannya untuk setiap m berbiaya total

$$\sum_m 2^{\text{popcount}(m)} = \sum_{k=0}^n \binom{n}{k} 2^k = 3^n \quad (10.1)$$

langkah, menurut teorema binomial. Untuk $n = 16$ itu sekitar $4.3 \cdot 10^7$, cepat; cara naif “setiap m , setiap s , uji apakah $s \subseteq m$ ” berbiaya $4^n \approx 4.3 \cdot 10^9$, jauh terlalu lambat.

10.2.2 Teladannya: penjual keliling

Pemrograman dinamis *bitmask* klasik adalah penyelesaian Held dan Karp untuk soal penjual keliling [2]. Seorang penjual berangkat dari kota 0, harus mengunjungi setiap dari n kota tepat sekali lalu kembali. Misalkan $dp[S][j]$ panjang rute terpendek yang berangkat dari kota 0, mengunjungi tepat kota-kota di S , dan berakhir di kota $j \in S$. Langkah terakhir rute seperti itu datang dari suatu kota $i \in S, i \neq j$, sehingga

$$dp[S][j] = \min_{i \in S, i \neq j} (dp[S \setminus \{j\}][i] + d(i, j)), \quad dp[\{0\}][0] = 0 \quad (10.2)$$

dan jawabannya $\min_j (dp[\text{semua}][j] + d(j, 0))$. Ada $2^n n$ keadaan dan masing-masing memerlukan $O(n)$ kerja: $O(2^n n^2)$ seluruhnya, sekitar $1.7 \cdot 10^7$ untuk $n = 16$, dibandingkan dengan $15! \approx 1.3 \cdot 10^{12}$ rute. Pengamatan kuncinya adalah bahwa masa depan sebuah rute parsial hanya bergantung pada kota *mana* yang sudah dikunjungi dan di *mana* kita berdiri, bukan pada urutan kota-kota itu dikunjungi.

WAWASAN KUNCI: Keadaan *bitmask* melupakan urutan masa lalu dan hanya mengingat isinya. Itulah takaran ingatan yang tepat setiap kali masa depan hanya bergantung pada “yang mana saja yang sudah dipakai”, dan itulah sebabnya faktorial berubah menjadi pangkat dua.

Kode 10.1 mengimplementasikan [Persamaan](#) (10.2) dalam bentuk “dorong”: dari setiap keadaan yang tercapai, rute diperpanjang ke setiap kota yang belum dikunjungi.

Kode 10.1 `tsp_bitmask.cpp`: pemrograman dinamis Held–Karp untuk lima kota.

```
// Bitmask DP for the shortest round trip through all cities (Held-Karp).
#include <algorithm>
#include <iostream>
using namespace std;
int main() {
    const int n = 5, INF = 1e9;
    int d[n][n] = {{0, 3, 4, 2, 7},
                  {3, 0, 4, 6, 3},
                  {4, 4, 0, 5, 8},
                  {2, 6, 5, 0, 6},
                  {7, 3, 8, 6, 0}};

    static int dp[1 << n][n];
    for (auto& row : dp) fill(row, row + n, INF);
    dp[1][0] = 0; // visited {0}, standing at city 0
    for (int mask = 1; mask < (1 << n); mask++)
        for (int last = 0; last < n; last++) {
            if (dp[mask][last] == INF) continue;
            for (int nxt = 0; nxt < n; nxt++) {
                if (mask >> nxt & 1) continue; // already visited
                int nm = mask | 1 << nxt;
                dp[nm][nxt] = min(dp[nm][nxt], dp[mask][last] + d[last][nxt]);
            }
        }
    int best = INF;
    for (int last = 1; last < n; last++)
        best = min(best, dp[(1 << n) - 1][last] + d[last][0]);
    cout << "shortest tour = " << best << "\n";
}
```

shortest tour = 19

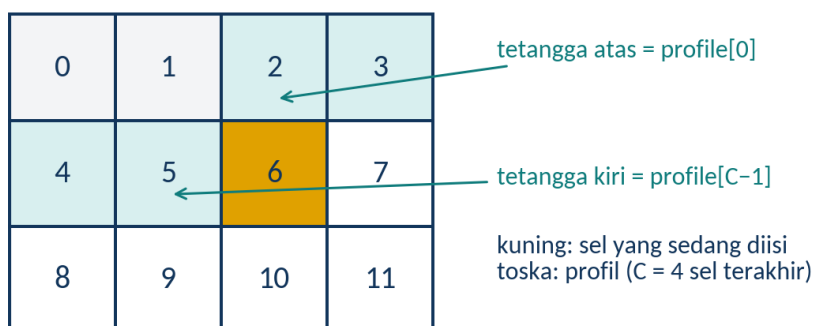
Tur $0 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 0$ panjangnya $2 + 6 + 3 + 4 + 4 = 19$. *Mask* diproses dalam urutan numerik menaik, dan itu sah karena menambahkan kota selalu membuat *mask* lebih besar: setiap keadaan sudah final sebelum dipakai. Tabel yang sama dapat juga diisi dalam bentuk “tarik”, dengan menghitung setiap $dp[S][j]$ langsung dari [Persamaan \(10.2\)](#) sambil melihat semua pendahulunya; kedua bentuk mengerjakan pekerjaan yang sama. Bentuk dorong sering lebih nyaman ketika banyak keadaan tak terjangkau, karena bentuk itu tidak pernah menyentuhnya, dan pengamatan ini membawa kita langsung ke *hash map* pada studi kasus.

10.3 Profil: Penampang Kisi sebagai Keadaan

10.3.1 Mengisi sel demi sel

Nomori sel-sel kisi $R \times C$ dalam urutan membaca: sel k berada di baris $\lfloor k/C \rfloor$ dan kolom $k \bmod C$. Misalkan setiap kendala soal hanya melibatkan tetangga yang berbagi sisi. Ketika kita memutuskan isi sel k , tetangganya di atas adalah sel $k - C$ dan tetangganya di kiri adalah sel $k - 1$; tetangga di bawah dan di kanan masih kosong dan akan memeriksa sel k sendiri ketika giliran mereka tiba.

Jadi, seberapa banyak bagian yang terisi harus diingat? Tepat sel-sel yang kelak akan dilihat oleh suatu sel di *masa depan*. Sel k dan setiap sel sesudahnya melihat ke belakang paling jauh C posisi, sehingga masa depan hanya bergantung pada C sel terakhir yang terisi, $k - C, \dots, k - 1$. Jendela inilah *profil*. **Gambar 10.1** memperlihatkankannya untuk $C = 4$ ketika sel 6 sedang diisi: sel 2, 3, 4, 5 membentuk profil, sel 2 adalah tetangga atas, sel 5 tetangga kiri, dan sel 0 dan 1 tidak akan pernah dilihat lagi.



sel abu-abu sudah selesai dan dilupakan; sel putih masih kosong.

Gambar 10.1 Mengisi kisi 3×4 dalam urutan membaca: ketika sel 6 diisi, hanya empat sel terakhir (profil) yang masih dapat memengaruhi masa depan.

DEFINISI. Dalam *pemrograman dinamis profil* atas kisi yang diisi dalam urutan membaca, keadaan setelah k sel terdiri atas isi sel $k - C, \dots, k - 1$ (profil), ditambah informasi global apa pun yang diperlukan soal, misalnya himpunan objek yang sudah dipakai.

10.3.2 Pemanasan: pemasangan domino

Contoh klasiknya menghitung banyaknya cara menutup papan $R \times C$ dengan domino 1×2 . Di sini setiap posisi profil hanya memerlukan satu bit: “sel ini sudah tertutup oleh domino yang menjulur dari sel sebelumnya”. Ketika kita sampai di sel k pada kolom c :

- jika bit c menyala, sel itu sudah tertutup dari atas; matikan bitnya dan lanjutkan;

- jika tidak, pasang domino tegak yang menutup sel ini dan sel di bawahnya, yang menyalakan bit c untuk baris berikutnya;
- atau, jika sel di kanannya ada dan masih bebas, pasang domino mendatar, yang menyalakan bit $c + 1$ untuk baris saat ini.

Keadaannya bernilai 2^C dan setiap langkah berbiaya $O(1)$ per keadaan, sehingga seluruh hitungan berbiaya $O(RC 2^C)$. **Kode 10.2** menjalankannya untuk empat papan.

Kode 10.2 `domino_profile.cpp`: menghitung pemasangan domino dengan profil satu bit per kolom.

```
// Broken-profile DP: number of ways to tile an R x C board with 1x2 dominoes.
#include <iostream>
#include <vector>
using namespace std;
long long tilings(int R, int C) { // C <= 10: profile = C bits
    vector<long long> dp(1 << C, 0), nd;
    dp[0] = 1;
    for (int cell = 0; cell < R * C; cell++) {
        int c = cell % C;
        nd.assign(1 << C, 0);
        for (int m = 0; m < (1 << C); m++) {
            if (!dp[m]) continue;
            if (m >> c & 1) { // already covered from above: just move on
                nd[m & ~(1 << c)] += dp[m];
                continue;
            }
            nd[m | 1 << c] += dp[m]; // vertical domino sticks into next row
            if (c + 1 < C && !(m >> (c + 1) & 1))
                nd[m | 1 << (c + 1)] += dp[m]; // horizontal domino covers c+1
        }
        dp = nd;
    }
    return dp[0];
}
int main() {
    cout << "2x3: " << tilings(2, 3) << "\n";
    cout << "3x4: " << tilings(3, 4) << "\n";
    cout << "4x4: " << tilings(4, 4) << "\n";
    cout << "8x8: " << tilings(8, 8) << "\n";
}
```

```
2x3: 3
3x4: 11
4x4: 36
8x8: 12988816
```

Papan 2×3 memiliki tiga pemasangan (tiga domino tegak, atau satu domino tegak dan dua domino mendatar bertumpuk di salah satu sisinya), dan hitungan papan catur 12 988 816 adalah bilangan yang terkenal, sehingga keluaran ini sekaligus berfungsi sebagai uji. Teknik *broken profile* ini dibahas panjang lebar dalam buku pegangan baku [3], [4]. Perhatikan bahwa di sini satu bit per sel sudah cukup, karena satu-satunya fakta yang diperlukan masa depan adalah “tertutup atau belum”. Pada CFONISMG masa depan perlu tahu *siapa* yang duduk di sebuah sel, sehingga setiap posisi profil menyimpan nomor siswa yang utuh.

10.3.3 Menyandikan profil sebagai satu bilangan

Jika satu posisi profil dapat menyimpan B nilai berbeda, profil berisi C posisi adalah bilangan dalam basis B :

$$prof = \sum_{k=0}^{C-1} profile[k] \cdot B^k \quad (10.3)$$

Dengan *profile*[0] sebagai sel tertua (sel di atas sel saat ini) pada digit terendah, maju ke sel berikutnya hanyalah aritmetika sederhana: membuang sel tertua berarti “kurangkan digit 0, bagi dengan B ”, dan menambahkan nilai sel baru s sebagai digit terbaru berarti “tambahkan $s \cdot B^{C-1}$ ”. Untuk CFONISMG, $B = 17$ (0 untuk meja kosong, 1 sampai 16 untuk siswa) dan $C \leq 4$, sehingga profil bernilai di bawah $17^4 = 83\,521 < 2^{17}$.

10.3.4 Perkirakan sebelum menulis kode

Sebelum menulis DP profil, hitunglah keadaannya. Batas jujur untuk CFONISMG mengalikan *mask* siswa yang sudah duduk, 2^{16} , dengan profilnya, 17^4 : sekitar $5.5 \cdot 10^9$, mustahil sebagai larik. Namun sebagian besar kombinasi itu tidak pernah dapat terjadi: siswa di dalam profil juga harus ada di dalam *mask*, tidak ada siswa yang muncul dua kali dalam profil, dan setelah k sel paling banyak k siswa yang sudah duduk. Itulah sebabnya penyelesaiannya menyimpan keadaan dalam *hash map*, dengan kunci pasangan (*mask*, profil) yang dikemas, dan hanya membuat entri ketika sebuah keadaan benar-benar tercapai.

Berapa banyak yang tercapai bergantung pada masukan, jadi kami mengukurnya. Pada kelas acak dengan sekitar sepertiga dari semua pasangan bersahabat dan seperempat bermusuhan, banyaknya keadaan terbesar yang hidup bersamaan adalah 33 350 untuk kelas 3×4 dengan 8 siswa (0,03 detik), 39 296 untuk 4×4 dengan 8 siswa (0,05 detik), tetapi 9,5 juta untuk kelas 2×4 dengan 16 siswa (hampir 10 detik). Profil tidak pernah menjadi masalah; *mask*-lah masalahnya. Ketika banyak siswa memperebutkan banyak meja, banyaknya himpunan “siapa yang sudah duduk” meledak. Program ini tepat untuk ukuran berapa pun, dan diterima oleh *judge*, sehingga uji-uji *judge* menjaga hasil kali ini tetap terkendali, tetapi pengukuran itu memberi tahu dengan tepat di mana bahayanya. Lakukan pengukuran yang sama untuk setiap penyelesaian eksponensial yang Anda tulis; cukup tambahkan satu baris yang mencetak ukuran *map* setelah setiap sel, jalankan pada masukan terbesar, lalu hapus baris itu sebelum mengirim.

10.3.5 Larik atau *hash map*?

DP profil dapat menyimpan keadaannya dengan dua cara. Bila setiap kombinasi *mask* dan profil mungkin terjadi dan hasil kalinya kecil, seperti pada pemasangan domino dengan 2^C keadaannya, larik biasa yang diindeks oleh keadaan adalah yang tercepat: tanpa *hashing*, lokalitas memori sempurna. Bila hasil kali teoretisnya sangat besar tetapi bagian yang terjangkau kecil, seperti pada CFONISMG, *hash map* seperti `std::unordered_map` hanya menyimpan yang ada. Harganya faktor konstan: setiap pencarian meng-*hash* kuncinya dan mungkin mengikuti penunjuk, biasanya sepuluh sampai lima puluh kali lebih lambat daripada akses larik. Jalan tengah, yang berguna ketika *map* menjadi hambatan, adalah mengumpulkan semua keadaan baru dari satu sel dalam *vector* berisi pasangan (kunci, skor), mengurutkannya menurut kunci, lalu memadukan kunci yang sama dalam satu lintasan; pengurutan memiliki lokalitas yang sangat baik dan sering mengalahkan *hashing* untuk jutaan entri.

Hal praktis kedua adalah struktur *bergulir*. Program tidak pernah memerlukan keadaan dari dua sel sebelumnya, sehingga hanya menyimpan dua *map*, yang sekarang dan yang berikutnya, lalu menukarnya dengan `std::move`, yang memindahkan penyimpanan internalnya tanpa menyalin

satu entri pun. Gagasan bergulir yang sama muncul di hampir setiap DP kisi: memori sebanding dengan satu lapis, bukan dengan seluruh kisi.

10.4 Studi Kasus: CFONISMG

10.4.1 Soal dengan kata-kata kita sendiri

Soalnya adalah SPOJ CFONISMG [1]. Baris pertama memuat R , C , dan N : kelas memiliki R baris berisi C meja, $C \leq 4$, dan ada N calon siswa, paling banyak 16. N baris berikutnya masing-masing memberikan nama siswa dan sebuah kata status; kata **DISRUPTIVE** menandai siswa pengganggu, kata lain apa pun menandai siswa yang tenang. Lalu datang bilangan F yang diikuti F pasangan nama yang bersahabat, dan bilangan V yang diikuti V pasangan nama yang bermusuhan.

Sebuah denah menempatkan setiap siswa di paling banyak satu meja dan membiarkan meja lainnya kosong. Dua meja *bertetangga* bila keduanya berbagi sisi. Denah sah bila tidak ada dua meja bertetangga yang ditempati pasangan bermusuhan dan tidak ada dua meja bertetangga yang ditempati dua siswa pengganggu. Skornya adalah banyaknya pasangan meja bertetangga yang ditempati sahabat. Cetak skor terbesar dari denah yang sah (denah kosong sah, sehingga jawabannya paling kecil 0).

10.4.2 Contoh masukan dan keluaran

```
2 3 4
Ani CALM
Budi DISRUPTIVE
Citra CALM
Dedi DISRUPTIVE
3
Ani Budi
Ani Citra
Citra Dedi
1
Budi Citra
```

```
3
```

Ada tiga pasangan sahabat, jadi 3 adalah nilai terbesar yang dapat diharapkan. Nilai itu tercapai: dudukkan Budi, Ani, dan Citra di baris depan dengan urutan itu, dan Dedi di belakang Citra. Ani duduk di sebelah kedua sahabatnya, Citra memiliki Dedi di belakangnya, Budi dan Citra yang bermusuhan berjarak dua meja, dan dua siswa pengganggu, Budi dan Dedi, tidak bertetangga.

10.4.3 Merancang penyelesaian

1. **Baca dan beri indeks.** Beri siswa nomor 1 sampai N melalui *map* dari nama; catat `disruptive[i]`, serta tabel simetris `is_friend_pair` dan `is_rival`.
2. **Keadaan.** Setelah k sel, keadaan adalah pasangan (*mask* siswa yang sudah duduk, profil C sel terakhir seperti pada [Persamaan \(10.3\)](#)), dan nilainya adalah skor terbaik yang dapat dicapai dengan keadaan itu. Kemas pasangan itu menjadi satu kunci 64 bit: *mask* digeser ke kiri 17 bit, ditambah profil.
3. **Transisi.** Untuk sel berikutnya, baca tetangga atas (`profile[0]`, jika barisnya bukan yang pertama) dan tetangga kiri (`profile[C - 1]`, jika kolomnya bukan yang pertama). Coba $s = 0$ (biarkan meja kosong) dan setiap siswa $s = 1..N$. Tolak s jika sudah ada di *mask*, atau jika bermusuhan dengan salah satu tetangga, atau jika s dan seorang tetangga sama-sama pengganggu. Selain itu, tambahannya adalah banyaknya tetangga yang bersahabat dengan s ,

dan keadaan barunya adalah (*mask* dengan *s* ditambahkan, profil yang digeser dengan *s* ditambahkan). Simpan skor maksimum per kunci baru:

$$ndp[key'] = \max(ndp[key'], dp[key] + gain(s)) \quad (10.4)$$

4. **Jawaban.** Setelah semua *RC* sel, nilai terbesar di dalam *map*.

Mengapa cukup menyimpan maksimum per kunci? Dua denah parsial dengan kunci yang sama telah mendudukkan siswa-siswa yang sama dan memiliki *C* sel terakhir yang sama. Setiap keputusan masa depan, keabsahannya, dan tambahan skornya hanya bergantung pada kedua hal itu. Jadi setiap kelanjutan dari denah yang satu juga merupakan kelanjutan dari denah yang lain, dengan tambahan skor yang sama; denah dengan skor lebih tinggi sejauh ini tidak mungkin lebih buruk. Inilah argumen substruktur optimal di balik setiap pemrograman dinamis [5], dituliskan untuk keadaan ini.

10.4.4 Satu jalannya program dengan tangan

Ikuti program pada kelas terkecil yang menarik: satu baris berisi tiga meja ($R = 1, C = 3$), seorang siswa tenang bernama Ani (nomor 1) dan seorang siswa pengganggu bernama Budi (nomor 2), yang bersahabat. Tuliskan keadaan sebagai (himpunan yang sudah duduk, profil), dengan profil dicantumkan dari meja tertua ke terbaru.

Sebelum meja mana pun: satu keadaan, ($\emptyset, [0,0,0]$) dengan skor 0.

Meja 0. Dari satu-satunya keadaan, program mencoba meja kosong, Ani, dan Budi. Tidak ada tetangga (baris 0, kolom 0), sehingga ketiganya sah dan tidak menambah apa-apa: ($\emptyset, [0,0,0]$), ($\{1\}, [0,0,1]$), dan ($\{2\}, [0,0,2]$), semuanya berskor 0.

Meja 1. Tetangga kiri adalah digit profil terbaru. Dari ($\emptyset, \dots$) program kembali dapat membiarkan meja kosong atau mendudukkan salah satu siswa: tiga keadaan berskor 0. Dari ($\{1\}, [0,0,1]$) program dapat membiarkan meja kosong atau mendudukkan Budi di samping Ani, sahabatnya: skor 1, keadaan ($\{1,2\}, [0,1,2]$). Secara simetris dari ($\{2\}, [0,0,2]$) program dapat mendudukkan Ani di samping Budi: skor 1, keadaan ($\{1,2\}, [0,2,1]$). Seluruhnya tujuh keadaan; perhatikan bahwa ($\{1\}, [0,1,0]$) (Ani di meja 0, meja 1 kosong) dan ($\{1\}, [0,0,1]$) (Ani di meja 1) adalah kunci yang *berbeda*, karena posisi Ani masih penting bagi meja 2.

Meja 2. Penalaran yang sama berlanjut; misalnya ($\{1\}, [0,0,1]$) dapat mendudukkan Budi di meja 2 dengan skor 1. Setelah meja terakhir, skor terbesar di dalam *map* adalah 1, dan itu benar: hanya ada satu pasangan sahabat. Menelusuri satu jalannya program seperti ini, sekali saja, dengan pensil dan kertas, adalah cara tercepat untuk yakin digit mana yang “atas” dan mana yang “kiri”.

10.4.5 Penyelesaian lengkap

Kode 10.3 adalah program lengkap yang tercatat *Accepted* di SPOJ. Blok P50 memuat penskoran dan pembentukan kunci baru di dalam `try_place`.

Kode 10.3 `CFONISMG.cpp`: *bitmask* siswa yang sudah duduk ditambah profil basis 17, disimpan dalam *hash map*.

```
#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) {
            int bm=(int)(key>>PB),prof=(int)(key&((1<<PB)-1));
            int profile[4];
            {
                int v=prof;
```

Kode 10.3 (Lanjutan) *CFONISMG.cpp*: bitmask siswa yang sudah duduk ditambah profil basis 17, disimpan dalam hash map.

```
for(int i=0; i<C; i++) {
    profile[i]=v%17;
    v/=17;
}
}
int top_stu=(row>0)?profile[0]:0, left_stu=(col>0)?profile[C-1]:0;
int base_prof=(prof-profile[0])/17;
auto try_place=[&](int s) {
    if(s>0&&(bm&(1<<(s-1))))return;
    if(s>0) {
        if(top_stu>0&&(is_rival[s][top_stu]||
            (disruptive[s]&&disruptive[top_stu])))return;
        if(left_stu>0&&(is_rival[s][left_stu]||
            (disruptive[s]&&disruptive[left_stu])))return;
    }
    // <P50>
    // Score +1 per friend among top/left, add s to the bitmask, append s
    // to base_prof (oldest digit already dropped), keep the best score.
    // >>> YOUR TURN (Public Edition): write this part yourself, following the hint
above. <<<
    // </P50>
};
try_place(0);
for(int s=1; s<=N; s++)try_place(s);
}
dp=move(ndp);
}
int ans=0;
for(auto&[k,v]:dp)ans=max(ans,v);
cout<<ans<<"\n";
}
```

10.4.6 Baris demi baris

Tabel. `disruptive`, `is_rival`, dan `is_friend_pair` diindeks dengan nomor siswa 1 sampai 16, sehingga larik berukuran 17 sudah cukup. Tabel sahabat dan musuh diisi ke dua arah, karena ketetanggaan tidak memiliki arah.

Nama. `map<string,int> id` mengubah nama menjadi nomor sekali saja, saat membaca; sesudahnya program hanya bekerja dengan nomor.

Pangkat dan pengemasan. `pw` menyimpan $17^0, \dots, 17^4$; `PB = 17` adalah banyaknya bit yang disediakan bagi profil di dalam kunci. *Map* dimulai dengan satu keadaan “belum ada yang duduk, semua meja kosong”, kunci 0, skor 0. `reserve` menghindari *rehashing* berulang ketika *map* tumbuh.

Perulangan sel. Untuk setiap sel, program menghitung baris dan kolomnya lalu membangun *map* baru `ndp`. Untuk setiap keadaan tersimpan, program membongkar *mask* `bm` dan profil `prof`, menguraikan digit-digit profil dengan *Persamaan* (10.3), membaca `top_stu` dan `left_stu`, dan menghitung lebih dulu `base_prof`, yaitu profil yang digit tertuanya telah dibuang.

`try_place(s)`, paruh pertama. Menolak siswa yang sudah ada di *mask* dan siswa yang berbenturan dengan tetangga atas atau kiri, persis seperti langkah 3 rancangan. Meja kosong, $s = 0$, melewati semua pemeriksaan.

`try_place(s)`, paruh kedua (blok P50). Menghitung sahabat di antara kedua tetangga, membangun *mask* baru `nb` dan kunci baru `nk` dengan menambahkan s sebagai digit profil terbaru, lalu menerapkan *Persamaan* (10.4) dengan satu pencarian di `ndp`.

Akhir sel. `dp = move(ndp)` menyerahkan *map* baru tanpa menyalinnya. Setelah sel terakhir, program memindai *map* untuk mencari skor terbesar lalu mencetaknya.

10.4.7 Jebakan

JEBAKAN: Menganggap tetangga kiri sebuah meja di kolom 0 sebagai `profile[C - 1]`. Digit itu adalah *meja terakhir dari baris sebelumnya*, yang tidak bertetangga. Uji `col > 0` itulah yang mencegah benturan palsu atau persahabatan palsu melintasi ujung baris.

- **Terlalu banyak siswa untuk digitnya.** Basis 17 mengizinkan 16 siswa. Dengan 17 siswa, digit 17 akan terbawa ke posisi berikutnya dan merusak profil, dan larik berukuran 17 akan diindeks di luar batasnya.
- **Kunci bertipe `int`.** Kunci memerlukan 17 bit *mask* ditambah 17 bit profil, 34 bit: harus `long long`.
- **Melupakan meja kosong.** Tanpa `try_place(0)` setiap meja harus terisi, yang mustahil bila meja lebih banyak daripada siswa, dan *map*-nya menjadi kosong.
- **Menskor sepasang dua kali.** Setiap pasangan bertetangga diskor tepat sekali, oleh meja yang belakangan dari keduanya, dengan melihat ke atas atau ke kiri. Melihat ke empat arah akan menggandakan setiap skor.
- **Memercayai banyaknya keadaan secara buta.** Ukurlah, seperti pada Subbab 10.3.4, pada masukan terbesar yang Anda kira diizinkan soal.

10.5 Uji Stres terhadap *Brute Force*

Brute force pada **Kode 10.4** mencoba setiap denah: untuk setiap meja dalam urutan membaca, program membiarkannya kosong atau mendudukkan siswa mana pun yang belum duduk dan tidak berbenturan dengan tetangga di atas dan di kiri. Program ini menjelajahi keputusan yang sama dengan pemrograman dinamis, tetapi tanpa memadukan keadaan apa pun, sehingga lambat secara eksponensial dan sepenuhnya bebas dari pengemasan kunci dan aritmetika profil.

Kode 10.4 `brute.cpp`: runut-balik atas setiap denah tempat duduk.

```
// Reference for CFONISMG: try every seating (empty or unused student per cell).
#include <iostream>
#include <map>
#include <string>
#include <vector>
using namespace std;
int R, C, N, best = 0;
bool dis[18], riv[18][18], fr[18][18], used[18];
vector<int> seat;
void go(int cell, int score) {
    if (cell == R * C) {
        best = max(best, score);
        return;
    }
    int r = cell / C, c = cell % C;
    int top = r > 0 ? seat[cell - C] : 0, left = c > 0 ? seat[cell - 1] : 0;
    seat[cell] = 0;
    go(cell + 1, score);
    for (int s = 1; s <= N; s++) {
        if (used[s]) continue;
        bool ok = true;
        int g = 0;
        for (int nb : {top, left}) {
            if (!nb) continue;
            if (riv[s][nb] || (dis[s] && dis[nb])) ok = false;
            if (fr[s][nb]) g++;
        }
        if (!ok) continue;
        used[s] = true, seat[cell] = s;
        go(cell + 1, score + g);
        used[s] = false, seat[cell] = 0;
    }
}
int main() {
    cin >> R >> C >> N;
    map<string, int> id;
    for (int i = 1; i <= N; i++) {
        string nm, st;
        cin >> nm >> st;
        id[nm] = i, dis[i] = (st == "DISRUPTIVE");
    }
    int F, V;
    cin >> F;
    for (int i = 0; i < F; i++) {
        string a, b;
        cin >> a >> b;

```

Kode 10.4 (Lanjutan) *brute.cpp*: runut-balik atas setiap denah tempat duduk.

```
    fr[id[a]][id[b]] = fr[id[b]][id[a]] = true;
}
cin >> V;
for (int i = 0; i < V; i++) {
    string a, b;
    cin >> a >> b;
    riv[id[a]][id[b]] = riv[id[b]][id[a]] = true;
}
seat.assign(R * C, 0);
go(0, 0);
cout << best << "\n";
}
```

Pembangkit pada **Kode 10.5** menghasilkan kelas berisi paling banyak tujuh meja, hingga lima siswa, status acak, dan pembagian acak pasangan-pasangan menjadi sahabat, musuh, dan netral, sehingga tidak ada pasangan yang sekaligus bersahabat dan bermusuhan.

Kode 10.5 `gen.py`: kelas-kelas acak kecil untuk CFONISMG.

```
# Random small CFONISMG inputs: python3 gen.py SEED
import random, sys, itertools
random.seed(int(sys.argv[1]))
while True:
    R, C = random.randint(1, 3), random.randint(1, 4)
    if R * C <= 7:
        break
N = random.randint(1, 5)
names = ['S%d' % i for i in range(1, N + 1)]
print(R, C, N)
for s in names:
    print(s, random.choice(['DISRUPTIVE', 'CALM', 'CALM']))
pairs = list(itertools.combinations(names, 2))
random.shuffle(pairs)
f = random.randint(0, len(pairs))
v = random.randint(0, len(pairs) - f)
print(f)
for a, b in pairs[:f]:
    print(a, b)
print(v)
for a, b in pairs[f:f + v]:
    print(a, b)
```

all 300 random tests passed

10.6 Taktik pada Hari Lomba

- **Bacalah batasan kecil sebagai petunjuk.** $N \leq 20$ menyarankan *bitmask* atas objek; sisi kisi ≤ 5 atau sekitar itu menyarankan profil atas sisi tersebut. Keduanya sekaligus menyarankan keduanya, seperti di sini.
- **Tulis banyaknya keadaan di kertas.** Kalikan rentang-rentangnya, lalu kurangkan yang tidak mungkin terjadi. Jika hasil kalinya masih terlalu besar, pakai *map* keadaan yang tercapai dan *ukur*lah.
- **Isi dalam urutan membaca dan hanya melihat ke belakang.** Atas dan kiri adalah satu-satunya tetangga yang sudah ada ketika sebuah sel diputuskan; setiap kendala dan setiap skor lalu ditangani tepat sekali.
- **Kemas kunci dengan cermat.** Tentukan banyaknya bit untuk setiap bagian, periksa bahwa jumlahnya muat dalam 64 bit, dan tuliskan kode pengemasan dan pembongkarannya berdampingan.
- **Pakai *mask* 64 bit ketika n mencapai 31.** Menggeser `int` 32 bit sejauh 31 posisi menghasilkan bilangan negatif dan sejauh 32 atau lebih adalah perilaku tak terdefinisi di C++; `1LL << i` menghindari keduanya.
- **Siapkan kembaran yang lambat.** *Brute force* runut-balik adalah pasangan uji stres yang paling alami bagi pemrograman dinamis yang memadukan keadaan: keputusannya sama, tanpa pepaduan.

10.7 Ringkasan Bab

- *Bitmask* menyimpan himpunan berisi hingga sekitar 20 objek dalam satu bilangan bulat; operasi himpunan adalah instruksi tunggal, dan mengenumerasi semua subhimpunan dari semua *mask* berbiaya 3^n , *Persamaan* (10.1).

- Held–Karp, [Persamaan](#) (10.2), memperlihatkan kekuatan melupakan urutan: $O(2^n n^2)$ alih-alih $n!$.
- DP profil mengisi kisi dalam urutan membaca dan hanya mengingat C sel terakhir, disandikan sebagai bilangan basis B , [Persamaan](#) (10.3).
- CFONISMG memadukan keduanya: (*mask* siswa yang sudah duduk, profil C meja terakhir), transisi dengan [Persamaan](#) (10.4), disimpan dalam *hash map* berisi keadaan yang tercapai.
- Penyelesaian eksponensial harus *diukur*: banyaknya keadaan yang tercapai, bukan batas teoretisnya, yang menentukan waktu jalan.

10.8 Soal Latihan

1. Daftarkan nilai-nilai s yang dihasilkan `for (s = m; s; s = (s - 1) & m)` untuk $m = 13$ (biner 1101).
2. Periksa dengan tangan bahwa papan 2×3 memiliki tepat tiga pemasangan domino, dan gambarkan ketiganya.
3. Pada **Kode 10.1**, apa yang berubah jika penjual tidak perlu kembali ke kota 0?
4. Ubah penskoran CFONISMG sehingga sahabat yang duduk *diagonal* di belakang satu sama lain (atas-kiri) juga memperoleh satu poin. Informasi tambahan apa yang harus dimuat profil?
5. Jelaskan mengapa *brute force* pada **Kode 10.4** dan pemrograman dinamisnya sepakat, padahal pemrograman dinamis membuang semua denah kecuali satu per kunci.
6. Pada jalannya program mungil di Subbab 10.4.4, paling banyak berapa keadaan yang ada setelah meja 2, dan mengapa kenyataannya bisa lebih sedikit?

10.9 Penyelesaian Soal Latihan

1. 13 (1101), 12 (1100), 9 (1001), 8 (1000), 5 (0101), 4 (0100), 1 (0001), lalu 0 mengakhiri perulangan: ketujuh subhimpunan tak kosong dari $\{0,2,3\}$, dalam urutan menurun.
2. Nomori kolomnya 1 sampai 3. Ketiga domino tegak; atau kolom 1 memuat domino tegak dan kolom 2–3 memuat dua domino mendatar yang bertumpuk; atau kolom 1–2 memuat dua domino mendatar bertumpuk dan kolom 3 sebuah domino tegak. Tidak ada susunan lain yang menutup papan, jadi hitungannya 3, cocok dengan program.
3. Jawabannya menjadi $\min_j dp[\text{semua}][j]$ tanpa suku $d(j, 0)$; tabelnya sendiri tidak berubah.
4. Tetangga atas-kiri dari sel k adalah sel $k - C - 1$, satu posisi lebih tua daripada jangkauan profil saat ini. Profil karena itu harus memuat $C + 1$ sel, dan tetangga atas-kiri hanya ada bila baris dan kolomnya sama-sama positif. Penyandian basis 17 tetap muat: $17^5 \approx 1.4 \cdot 10^6 < 2^{21}$, sehingga profil memerlukan 21 bit dari kuncinya.
5. Subbab 10.4.3 menunjukkan bahwa dua denah parsial dengan kunci yang sama memiliki himpunan kelanjutan sah yang persis sama dengan tambahan skor yang sama. Membuang yang lebih buruk karena itu tidak pernah menyingkirkan denah lengkap terbaik: prefikisnya adalah denah yang disimpan atau memiliki skor yang tidak lebih tinggi daripada denah yang disimpan dengan kunci yang sama, dan denah yang disimpan menerima kelanjutan yang sama. *Brute force* memeriksa setiap denah secara eksplisit, sehingga keduanya menemukan maksimum yang sama.
6. Setiap keadaan setelah meja 2 ditentukan oleh isi ketiga meja (profil saat itu meliputi seluruh kelas), dan setiap meja berisi tak seorang pun, Ani, atau Budi, tanpa siswa yang muncul dua kali: $1 + 2 \cdot 3 + 2 \cdot 3 = 13$ susunan (tidak ada yang duduk; satu siswa di salah satu dari tiga meja; kedua siswa di dua dari tiga meja dalam urutan mana pun). Budi dan Ani yang

bersebelahan sah karena mereka bersahabat, bukan bermusuhan, dan hanya satu di antara mereka yang pengganggu, sehingga ke-13 susunan dapat terjadi; *mask* tidak menambah keadaan di sini karena sudah ditentukan oleh profil. Pada kelas yang lebih besar, *mask* mengingat siswa yang sudah keluar dari profil, dan dari situlah pertumbuhan yang diukur pada Subbab 10.3.4 berasal.

Daftar Pustaka

- [1] Sphere Online Judge, “Problem CFONISMG.” [Online]. Available: <https://www.spoj.com/problems/CFONISMG/>
- [2] M. Held and R. M. Karp, “A dynamic programming approach to sequencing problems,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, no. 1, pp. 196–210, 1962.
- [3] S. Halim, F. Halim, and S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.
- [4] A. Laaksonen, *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, 3rd ed. Cham, Switzerland: Springer, 2024.
- [5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022, ch. 14.

Bab 11

Divide and Conquer CDQ dan LIS Dua Dimensi — LIS2

Subbarisan naik terpanjang (*longest increasing subsequence*, LIS) adalah salah satu program dinamis pertama yang dijumpai setiap peserta lomba. Berikan dua bilangan, bukan satu, pada setiap elemen, tuntutan agar *keduanya* naik, dan pertahankan urutan masukan: trik $O(n \log n)$ yang ramah itu pun berhenti bekerja. Program dinamis polosnya memerlukan waktu $O(n^2)$, terlalu lambat untuk $n = 10^5$. Bab ini mengajarkan metode yang menyelamatkannya, yaitu *divide and conquer CDQ*: belah barisan menurut posisi, tuntaskan paruh kiri, biarkan paruh kiri yang sudah tuntas menyalurkan pengaruhnya ke paruh kanan dengan sebuah sapuan dan pohon Fenwick, lalu barulah tuntaskan paruh kanan. Di akhir bab Anda akan mampu menjelaskan mengapa urutan itu wajib, membuktikan batas $O(n \log^2 n)$, menulis seluruh solusi soal SPOJ LIS2 dari ingatan, dan mengujinya terhadap *brute force* pada ratusan masukan acak. Pola yang sama menyelesaikan sekeluarga soal di mana suatu keadaan bergantung pada keadaan sebelumnya melalui dua pertidaksamaan, sehingga teknik ini termasuk bekal paling berharga yang dapat dibawa peserta ke babak nasional.

11.1 Di Mana Ini Berguna: Catatan Latihan Dewi

Dewi berlari tiga atau empat kali seminggu dan mencatat setiap larinya di sebuah buku: jaraknya dalam kilometer dan kecepatan rata-ratanya. Setelah setahun ia ingin menceritakan kisah kemajuan yang sederhana. Ia akan memilih beberapa lari dari buku itu, sesuai urutan terjadinya, sedemikian rupa sehingga setiap lari yang dipilih *lebih jauh* sekaligus *lebih cepat* daripada lari yang dipilih sebelumnya. Berapa panjang kisah semacam itu paling banyak?

Ia tidak boleh mengubah urutan, karena kisah kemajuan harus bergerak maju dalam waktu. Ia juga tidak boleh melewati kata “sekaligus”: lari yang lebih jauh tetapi lebih lambat tidak menunjukkan kemajuan, demikian pula lari yang lebih cepat tetapi lebih pendek. Yang ia tanyakan adalah *rantai terpanjang* dari lari-lari yang naik dalam dua ukuran sekaligus, dengan tetap menghormati urutan tanggal.

Pertanyaan yang sama bersembunyi di banyak tempat. Pemilik toko mencari rentang bulan terpanjang, tidak harus berturut-turut, yang pendapatan dan jumlah pelanggannya sama-sama naik. Petani durian yang mengukur bibit sesuai urutan tanam mencari barisan pohon terpanjang yang masing-masing lebih tinggi dan berbatang lebih tebal daripada pohon sebelumnya, untuk melihat apakah pupuk baru terus membantu. Penyusun kotak, amplop, atau tugas yang bersarang menanyakan hal yang sama dengan ukuran. Dalam semua contoh ini data datang dalam urutan tetap dan setiap butir membawa pasangan (x, y) .

Dengan beberapa puluh catatan, Dewi dapat membandingkan setiap pasangan dengan tangan. Dengan seratus ribu catatan, seperti pada soal lomba bab ini, cara yang jelas membandingkan sekitar $5 \cdot 10^9$ pasangan, jauh melampaui satu atau dua detik yang diizinkan juri. Kita memerlukan metode yang hampir linear, dan metode itu datang dari gagasan *divide and conquer* yang dipopulerkan oleh pelatihan olimpiade Tiongkok dengan nama *CDQ* [1].

11.2 Tekniknya dari Nol

11.2.1 Dari satu dimensi ke dua dimensi

Mulailah dengan satu bilangan per elemen. Diberikan $a_1, a_2, \dots, a_n$, *subbarisan naik terpanjang* (LIS) adalah daftar posisi terpanjang $i_1 < i_2 < \dots < i_k$ dengan $a_{i_1} < a_{i_2} < \dots < a_{i_k}$. Misalkan dp_i adalah panjang daftar terpanjang semacam itu yang *berakhir* di posisi i . Setiap daftar yang berakhir di i adalah i sendirian, atau daftar yang berakhir di suatu j sebelumnya dengan $a_j < a_i$, lalu disusul i . Jadi

$$dp_i = 1 + \max(\{0\} \cup \{dp_j : j < i, a_j < a_i\}) \quad (11.1)$$

dan jawabannya adalah dp_i terbesar. **Persamaan** (11.1) memuat dua syarat pada j : harus datang *lebih awal* ($j < i$) dan harus *lebih kecil* ($a_j < a_i$). Syarat pertama terpenuhi dengan sendirinya bila posisi diproses dari kiri ke kanan; ketika kita tiba di i , hanya posisi sebelumnya yang telah diproses. Syarat kedua adalah *maksimum prefiks* atas nilai: di antara semua elemen yang telah diproses dengan nilai di bawah a_i , berapa dp terbesar? Pohon Fenwick (*binary indexed tree*) yang menyimpan maksimum, bukan jumlah, menjawabnya dalam $O(\log n)$ [2], [3]. Itulah LIS $O(n \log n)$ yang sudah kita kenal.

Sekarang berikan setiap elemen pasangan (x_i, y_i) dan carilah rantai yang kedua koordinatnya naik tegas. Bentuk rekurensinya tetap:

$$dp_i = 1 + \max(\{0\} \cup \{dp_j : j < i, x_j < x_i, y_j < y_i\}) \quad (11.2)$$

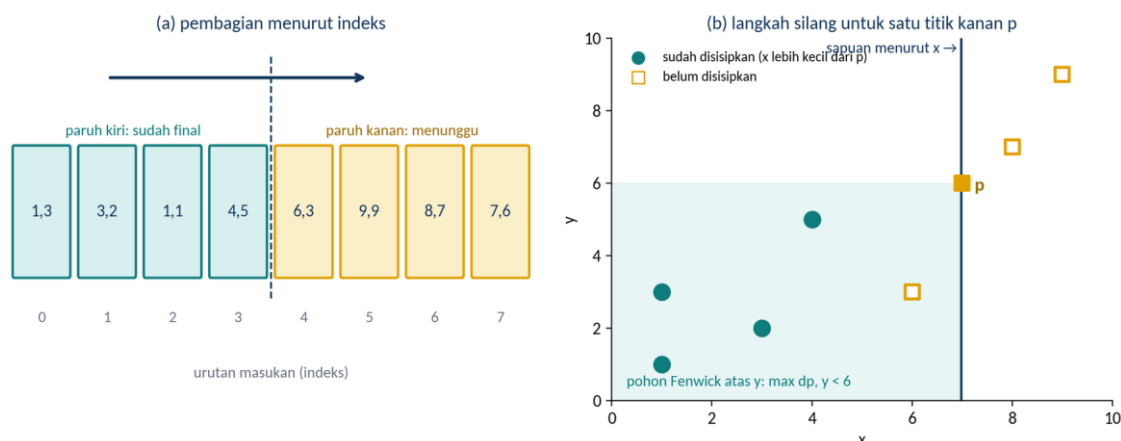
tetapi **Persamaan** (11.2) memuat *tiga* syarat pada j : lebih awal dalam masukan, lebih kecil dalam x , dan lebih kecil dalam y . Pemrosesan dari kiri ke kanan menghapus satu syarat, pohon Fenwick atas y menghapus satu lagi, dan syarat pada x tersisa tanpa ada yang menanganinya. Mengurutkan menurut x alih-alih posisi tidak menolong, karena syarat posisilah yang kemudian tersisa. Tiga urutan yang saling bebas tidak dapat ditangani oleh satu sapuan ditambah satu pohon. Kita perlu menghapus satu urutan dengan cara lain.

11.2.2 Gagasannya: belah menurut posisi

CDQ menghapus urutan posisi dengan *divide and conquer* (bagi dan kuasai). Potong rentang indeks $[l, r]$ di tengahnya m . Setiap pasangan (j, i) dengan $j < i$ di dalam rentang termasuk salah satu dari tiga jenis:

- keduanya di paruh kiri $[l, m]$,
- keduanya di paruh kanan $[m + 1, r]$,
- j di paruh kiri dan i di paruh kanan.

Dua jenis pertama adalah soal yang sama pada rentang yang lebih kecil, jadi rekursi menanganinya. Jenis ketiga adalah bagian baru, dan ia memiliki sifat yang indah: *setiap* elemen kiri datang lebih awal daripada *setiap* elemen kanan, sehingga syarat posisi $j < i$ otomatis benar untuk semua pasangan ini. Yang tersisa hanya $x_j < x_i$ dan $y_j < y_i$, dan dua syarat adalah persis yang dapat ditangani satu sapuan ditambah satu pohon Fenwick. Urutkan paruh kiri dan paruh kanan menurut x . Telusuri elemen kanan dengan x menaik; sebelum menangani elemen kanan i , sisipkan ke pohon Fenwick setiap elemen kiri j dengan $x_j < x_i$, di posisi y_j dengan nilai dp_j . Maka nilai terbesar di antara posisi Fenwick di bawah y_i adalah tepat dp_j terbaik atas semua elemen kiri yang boleh mendahului i . **Gambar 11.1** menggambar kedua langkah itu pada contoh kecil.



Gambar 11.1 CDQ pada delapan titik. (a) Rentang dipotong menurut indeks; paruh kiri dituntaskan sebelum memengaruhi paruh kanan. (b) Untuk titik kanan $p = (7,6)$ setiap titik kiri dengan x lebih kecil telah disisipkan ke pohon Fenwick, yang kemudian menjawab “ dp terbesar dengan $y < 6$ ”.

Langkah tengah ini kita sebut *cross* (langkah silang). Seluruh metode muat dalam beberapa baris pseudokode pada **Pseudokode 11.1**.

Pseudokode 11.1 Divide and conquer CDQ untuk LIS dua dimensi.

```
SOLVE(l, r)
1  if l >= r
2      return                                // satu elemen: dp sudah 1
3  m = floor((l + r) / 2)
4  SOLVE(l, m)                                // paruh kiri menjadi final
5  CROSS(l, m, r)                            // nilai dp kiri memperbaiki dp kanan
6  SOLVE(m + 1, r)                          // paruh kanan dituntaskan

CROSS(l, m, r)
1  A = indices l..m sorted by x; B = indices m+1..r sorted by x
2  j = 1
3  for each i in B, in increasing x
4      while j <= |A| and x[A[j]] < x[i]
5          FENWICK-MAX-UPDATE(y[A[j]], dp[A[j]]); j = j + 1
6      dp[i] = max(dp[i], FENWICK-PREFIX-MAX(y[i] - 1) + 1)
7  batalkan posisi Fenwick yang ditulis di baris 5
```

DEFINISI. *Divide and conquer CDQ* adalah teknik *offline* untuk rekurensi yang nilai suatu elemennya bergantung pada elemen-elemen sebelumnya melalui syarat urutan tambahan. Teknik ini membelah elemen menurut waktu (posisi), menuntaskan paruh yang lebih awal sepenuhnya, menyalurkan kontribusi paruh awal ke paruh akhir dalam satu kelompok (langkah *cross*), lalu menuntaskan paruh akhir. Langkah *cross* tidak lagi perlu memeriksa waktu, sehingga satu dimensi terhapus dari soal [1], [4].

11.2.3 Mengapa urutan kiri, silang, kanan itu wajib

Perhatikan lagi baris 4–6 dari SOLVE. Dua urutan lain tampak sama wajarnya, dan keduanya salah.

Jika CROSS dipanggil sebelum SOLVE(l, m), pohon Fenwick menerima nilai kiri yang belum lengkap: elemen kiri yang semestinya mencapai 3 melalui elemen kiri lain mungkin masih bernilai 1. Elemen kanan lalu mewarisi nilai yang terlalu kecil.

Jika SOLVE($m + 1, r$) dipanggil sebelum CROSS, paruh kanan meneruskan nilai di antara elemennya sendiri sebelum nilai-nilai itu menerima pengaruh kiri. Misalkan rantai terbaik adalah $j \rightarrow i \rightarrow k$ dengan j di paruh kiri dan i, k di paruh kanan. Mata rantai $j \rightarrow i$ ditemukan di CROSS,

dan mata rantai $i \rightarrow k$ ditemukan di dalam $\text{SOLVE}(m+1, r)$. Jika paruh kanan dituntaskan lebih dahulu, k membaca dp_i ketika nilainya masih 1, dan rantai itu hilang.

Berikut argumen bahwa urutan yang benar menghitung setiap dp_i dengan tepat, ditulis sebagai invarian.

WAWASAN KUNCI: *Klaim:* ketika $\text{SOLVE}(l, r)$ dipanggil, setiap dp_i dengan $i \in [l, r]$ sudah memuat kontribusi terbaik dari setiap $j < l$; ketika panggilan itu selesai, setiap dp_i di $[l, r]$ sudah final. *Bukti dengan induksi pada panjang rentang.* Rentang sepanjang 1 tidak memiliki pasangan di dalamnya, jadi nilainya sudah final sejak awal. Untuk rentang yang lebih panjang, $\text{SOLVE}(l, m)$ melihat prasyarat yang sama dan, menurut hipotesis induksi, kembali dengan paruh kiri final. **CROSS** lalu menambahkan, untuk setiap i di kanan, kontribusi terbaik setiap $j \in [l, m]$, dan boleh melakukannya dalam satu kelompok justru karena nilai-nilai itu sudah final. Kini setiap elemen kanan membawa kontribusi semua $j \leq m$, yang merupakan prasyarat $\text{SOLVE}(m+1, r)$, sehingga menurut induksi paruh kanan kembali dalam keadaan final.

Invarian ini juga menjelaskan mengapa kesalahan tersebut begitu berbahaya dalam lomba: pada masukan mungil rantainya pendek, dan banyak uji acak tidak memiliki rantai sepanjang 3 yang melintasi titik tengah, sehingga program dengan urutan panggilan yang salah lolos contoh tetapi gagal di juri.

11.2.4 Biayanya

Misalkan $T(n)$ adalah waktu untuk rentang berisi n elemen. Langkah *cross* mengurutkan kedua paruh menurut x , berbiaya $O(n \log n)$, dan melakukan paling banyak n pembaruan Fenwick, n kueri, dan n pembersihan, masing-masing $O(\log n)$. Jadi

$$T(n) = 2T(n/2) + O(n \log n) \quad (11.3)$$

Pohon rekursi **Persamaan** (11.3) memiliki $\log_2 n$ tingkat, dan setiap tingkat mengerjakan total $O(n \log n)$, karena rentang-rentang pada satu tingkat mempartisi n elemen. Karena itu $T(n) = O(n \log^2 n)$, dengan argumen pohon rekursi yang sama seperti untuk *merge sort* [5]. Untuk $n = 10^5$ itu sekitar $10^5 \cdot 17 \cdot 17 \approx 3 \cdot 10^7$ langkah dasar: sepersekian detik. Sebaliknya, program dinamis kuadratik **Persamaan** (11.2) memerlukan sekitar $5 \cdot 10^9$ perbandingan.

Satu detail menjaga batas itu tetap jujur. Setelah langkah *cross*, pohon Fenwick harus kosong kembali, atau langkah *cross* berikutnya melihat nilai basi. Mengosongkan seluruh larik berbiaya $O(m)$ setiap kali, dengan m banyaknya nilai y yang berbeda, dan ada sekitar n langkah *cross*, sehingga pembersihan saja dapat menjadi $O(n \cdot m) = O(n^2)$. Sebagai gantinya kita menelusuri lagi lintasan pembaruan yang sama dan menulis 0, yang biayanya persis sama dengan pembaruannya. Semboyan yang perlu diingat: *batalkan hanya yang Anda tulis*.

11.3 Soal Pemanasan

11.3.1 Inversi: paruh kiri memengaruhi paruh kanan

Soal paling sederhana yang berbentuk CDQ menghitung *inversi*: pasangan $i < j$ dengan $a_i > a_j$. Belah di tengah; inversi di dalam setiap paruh dihitung secara rekursif; inversi dengan i di kiri dan j di kanan dihitung sambil menggabungkan kedua paruh yang telah terurut. Setiap kali penggabungan mengambil elemen dari paruh kanan, semua elemen kiri yang masih menunggu lebih besar, dan masing-masing membentuk inversi dengannya. **Kode 11.1** adalah programnya secara lengkap.

Kode 11.1 `inversions.cpp`: menghitung inversi dengan *merge sort*, leluhur CDQ.

```
// Warm-up: count pairs  $i < j$  with  $a[i] > a[j]$  by divide and conquer.
// The left half is finished before it "influences" the right half.
#include <cstdio>
#include <vector>
using namespace std;

long long sortCount(vector<int>& a, int l, int r) { // sorts  $a[l..r]$ 
    if (l >= r) return 0;
    int mid = (l + r) / 2;
    long long cnt = sortCount(a, l, mid) + sortCount(a, mid + 1, r);
    vector<int> merged;
    int i = l, j = mid + 1;
    while (i <= mid || j <= r) {
        if (j > r || (i <= mid && a[i] <= a[j])) {
            merged.push_back(a[i++]);
        } else {
            cnt += mid - i + 1; //  $a[j]$  is smaller than all of  $a[i..mid]$ 
            merged.push_back(a[j++]);
        }
    }
    for (int k = l; k <= r; k++) a[k] = merged[k - l];
    return cnt;
}

int main() {
    int n;
    if (scanf("%d", &n) != 1) return 0;
    vector<int> a(n);
    for (int& v : a)
        if (scanf("%d", &v) != 1) return 0;
    printf("%lld\n", n ? sortCount(a, 0, n - 1) : 0LL);
    return 0;
}
```

Masukan 6 diikuti 5 3 8 1 9 2 memiliki delapan inversi (5 sebelum 3, 1, dan 2; 3 sebelum 1 dan 2; 8 sebelum 1 dan 2; 9 sebelum 2), dan program setuju:

8

Bandingkan strukturnya dengan **Pseudokode 11.1**: belah menurut posisi, tangani kedua paruh, dan hitung pasangan silang dalam satu lintasan yang bersandar pada kedua paruh yang terurut. Bedanya, inversi tidak memerlukan urutan di antara ketiga langkah, karena tidak ada apa pun di dalam satu paruh yang bergantung pada paruh lainnya. Pada LIS, nilai-nilainya *memang* saling bergantung, dan itulah yang memaksa urutan pada Subbab 11.2.3.

11.3.2 LIS satu dimensi dengan pohon Fenwick maksimum

Pemanasan kedua adalah alat yang dipakai CDQ di dalam langkah *cross*-nya. **Kode 11.2** menghitung *Persamaan* (11.1) dari kiri ke kanan. Mula-mula program *memampatkan* nilai: nilai yang berbeda diurutkan dan setiap nilai diganti peringkatnya $1, 2, \dots, m$, sehingga pohon Fenwick hanya memerlukan $m \leq n$ sel meskipun nilai mencapai 10^9 . Kueri $p - 1$ menanyakan nilai yang tegas lebih kecil daripada nilai saat ini; memakai p justru mengizinkan nilai sama dan menghitung subbarisan *tidak turun* terpanjang.

Kode 11.2 `lis_fenwick.cpp`: LIS klasik dalam $O(n \log n)$ dengan pohon Fenwick maksimum prefiks.

```
// Warm-up: the classic strict LIS of one sequence in  $O(n \log n)$  with a
// Fenwick tree of maxima over compressed values.
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n;
    if (scanf("%d", &n) != 1) return 0;
    vector<int> a(n);
    for (int& v : a)
        if (scanf("%d", &v) != 1) return 0;
    vector<int> vals(a);
    sort(vals.begin(), vals.end());
    vals.erase(unique(vals.begin(), vals.end()), vals.end());
    int m = (int)vals.size(), best = 0;
    vector<int> fen(m + 1, 0);
    for (int v : a) {
        int p = int(lower_bound(vals.begin(), vals.end(), v) - vals.begin()) + 1;
        int got = 0; // best chain with value < v
        for (int i = p - 1; i > 0; i -= i & -i) got = max(got, fen[i]);
        int here = got + 1;
        for (int i = p; i <= m; i += i & -i) fen[i] = max(fen[i], here);
        best = max(best, here);
        printf("value %3d -> chain %d\n", v, here);
    }
    printf("LIS length: %d\n", best);
    return 0;
}
```

Pada barisan 3 1 4 1 5 9 2 6, program mencetak panjang rantai yang berakhir di setiap elemen:

```
value  3 -> chain 1
value  1 -> chain 1
value  4 -> chain 2
value  1 -> chain 1
value  5 -> chain 3
value  9 -> chain 4
value  2 -> chain 2
value  6 -> chain 4
LIS length: 4
```

Rantai terpanjang berpanjang 4, misalnya 3, 4, 5, 9 atau 1, 4, 5, 6. Perhatikan bahwa angka 1 yang kedua mendapat panjang 1, bukan 2: ketegasan dijaga oleh $p - 1$.

Pohon Fenwick *maksimum* memiliki satu keterbatasan yang perlu diketahui: nilainya hanya boleh naik. Anda tidak dapat menurunkan sel ke nilai yang lebih kecil, karena maksimum yang tersimpan di tingkat atas tidak akan mengetahuinya. Dalam CDQ hal ini bukan masalah, karena kita tidak pernah menurunkan nilai; kita hanya mengembalikan sel ke 0, melalui lintasan yang sama, setelah seluruh langkah *cross* selesai.

11.4 Studi Kasus: LIS2

11.4.1 Soalnya dengan kata-kata kita sendiri

Soal SPOJ LIS2 [6] memberikan N titik (x_i, y_i) dalam urutan tetap, dengan N sampai 10^5 . Carilah k terbesar sehingga terdapat posisi $i_1 < i_2 < \dots < i_k$ dengan $x_{i_1} < x_{i_2} < \dots < x_{i_k}$ sekaligus $y_{i_1} < y_{i_2} < \dots < y_{i_k}$. Cetak k itu. Masukannya adalah bilangan N diikuti N baris berisi dua bilangan bulat. Inilah buku catatan Dewi, dengan jarak sebagai x dan kecepatan sebagai y .

11.4.2 Contoh yang dikerjakan dengan tangan

Berikut masukan kecil buatan kita sendiri: delapan titik pada **Gambar 11.1**.

```
8
1 3
3 2
1 1
4 5
6 3
9 9
8 7
7 6
```

Kerjakan **Persamaan** (11.2) dengan tangan. Titik 0 adalah (1,3) dan mendapat 1. Titik 1, (3,2), tidak dapat mengikuti (1,3) karena $2 < 3$ pada y ; ia mendapat 1. Titik 2, (1,1), mendapat 1. Titik 3, (4,5), dapat mengikuti (1,3), (3,2), atau (1,1): 2. Titik 4, (6,3), dapat mengikuti (3,2) atau (1,1): 2. Titik 5, (9,9), dapat mengikuti (4,5): 3. Titik 6, (8,7), dapat mengikuti (4,5) atau (6,3): 3. Titik 7, (7,6), dapat mengikuti (4,5) atau (6,3): 3; ia tidak dapat mengikuti (8,7) atau (9,9) yang lebih besar. Jawabannya 3, misalnya $(1,1) \rightarrow (4,5) \rightarrow (9,9)$, dan program mencetak:

```
3
```

Dalam jalannya CDQ pada masukan ini, langkah *cross* pertama di tingkat teratas memakai tepat situasi **Gambar 11.1(b)**: ketika sapuan tiba di $p = (7,6)$, keempat titik kiri memiliki x lebih kecil, yang ber- $y < 6$ adalah (1,3), (3,2), (1,1), dan (4,5), dan nilai final terbaik di antara mereka adalah 2, sehingga p menerima 3.

11.4.3 Rancangan

Rancangannya mengikuti **Pseudokode 11.1** dengan tiga keputusan praktis.

1. **Mampatkan** y . Koordinat dapat besar atau negatif, tetapi hanya urutannya yang penting. Kita mengganti y dengan peringkatnya di antara nilai y yang berbeda, $1..m$, sehingga pohon Fenwick memiliki $m + 1$ sel.
2. **Urutkan daftar indeks, bukan titik**. Langkah *cross* mengurutkan *salinan indeks* setiap paruh menurut x ; larik px , py , dan dp tetap dalam urutan masukan. Dengan begitu rekursi ke paruh kanan tetap melihat posisi dalam urutan masukan, yang memang ia perlukan.
3. **Pertidaksamaan tegas**. Titik kiri disisipkan selama $px[\text{left}[j]] < px[i]$, sehingga titik kiri dengan x yang *sama* dengan i tidak pernah disisipkan sebelum i dijawab; dan kueri membaca posisi sampai $py[i] - 1$, sehingga nilai y yang sama juga tersingkir.

11.4.4 Solusi lengkap

Kode 11.3 adalah program utuhnya. Program ini terkompilasi tanpa peringatan dengan `std=c++17 -O2 -Wall -Wextra` dan lolos uji stres pada Subbab 11.5. Bagian di antara kedua penanda `<P50>` adalah paruh kedua algoritma inti, yaitu sapuan pada langkah *cross*; baris komentar tepat setelah `// <P50>` menyebutkan dengan tepat apa yang harus dikerjakan bagian itu.

Kode 11.3 `LIS2.cpp`: LIS dua dimensi dalam $O(n \log^2 n)$ dengan divide and conquer CDQ.

```
// LIS2: the longest chain of points, taken in input order, whose x AND y
// both increase strictly. CDQ divide and conquer + a Fenwick tree of maxima.
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

int n, m; // number of points; number of distinct y values
vector<int> px, py, dp; // x, compressed y (1..m), best chain ending here
vector<int> fen; // Fenwick tree: prefix maxima of dp over y

void fenUpdate(int i, int v) {
    for (; i <= m; i += i & -i) fen[i] = max(fen[i], v);
}
int fenQuery(int i) { // the largest value stored at positions 1..i
    int r = 0;
    for (; i > 0; i -= i & -i) r = max(r, fen[i]);
    return r;
}
void fenClear(int i) { // undo every write that touched position i
    for (; i <= m; i += i & -i) fen[i] = 0;
}

// Every point in [l, mid] is final; let it improve the points in [mid+1, r].
void cross(int l, int mid, int r) {
    vector<int> left, right;
    for (int i = l; i <= mid; i++) left.push_back(i);
    for (int i = mid + 1; i <= r; i++) right.push_back(i);
    auto byX = [](int a, int b) { return px[a] < px[b]; };
    sort(left.begin(), left.end(), byX);
    sort(right.begin(), right.end(), byX);
    // <P50>
    // Sweep the right points by x; first insert every left point with a smaller
    // x (value dp at its y), then extend the best chain whose y is below ours.
    // Afterwards clear only the Fenwick positions that were written.
    // >>> YOUR TURN (Public Edition): write this part yourself, following the hint above.
    <<<
    // </P50>
}
```

Kode 11.3 (Lanjutan) LIS2.cpp: LIS dua dimensi dalam $O(n \log^2 n)$ dengan divide and conquer CDQ.

```
void solve(int l, int r) {
    if (l >= r) return;
    int mid = (l + r) / 2;
    solve(l, mid);           // 1. the left half becomes final
    cross(l, mid, r);        // 2. it passes its influence to the right half
    solve(mid + 1, r);       // 3. only now may the right half finish
}

int main() {
    if (scanf("%d", &n) != 1) return 0;
    px.resize(n), py.resize(n), dp.assign(n, 1);
    for (int i = 0; i < n; i++)
        if (scanf("%d %d", &px[i], &py[i]) != 2) return 0;
    vector<int> ys(py);       // compress y to 1..m
    sort(ys.begin(), ys.end());
    ys.erase(unique(ys.begin(), ys.end()), ys.end());
    m = (int)ys.size();
    for (int i = 0; i < n; i++)
        py[i] = int(lower_bound(ys.begin(), ys.end(), py[i]) - ys.begin()) + 1;
    fen.assign(m + 1, 0);
    solve(0, n - 1);
    printf("%d\n", n ? *max_element(dp.begin(), dp.end()) : 0);
    return 0;
}
```

11.4.5 Baris demi baris

Keadaan global. `px` menyimpan x dalam urutan masukan, `py` menyimpan y yang telah dimampatkan, dan `dp[i]` menyimpan panjang rantai terbaik yang sejauh ini diketahui untuk rantai yang berakhir di titik i . Semua `dp` dimulai dari 1, karena satu titik sudah merupakan rantai. Pohon Fenwick `fen` memiliki $m + 1$ sel, dan sel 0 tidak dipakai.

Tiga pembantu Fenwick. `fenUpdate(i, v)` menaikkan setiap sel yang bertanggung jawab atas posisi i menjadi sekurang-kurangnya v ; `fenQuery(i)` mengembalikan maksimum yang tersimpan atas posisi $1..i$ dengan berjalan turun melalui `i -= i & -i`; `fenClear(i)` menelusuri tepat lintasan `fenUpdate` dan menulis 0. Ekspresi `i & -i` adalah bit terendah yang menyala pada i , yaitu ukuran blok yang dicakup sebuah sel [2].

`cross(l, mid, r)`. Enam baris pertama membangun `left` dan `right`, daftar indeks kedua paruh, lalu mengurutkan keduanya menurut x . Kemudian datang sapuan. Penunjuk `j` hanya bergerak maju melalui `left`: untuk setiap titik kanan i , dengan x menaik, gelung `while` menyisipkan setiap titik kiri yang x -nya tegas lebih kecil. Karena titik kanan dikunjungi dengan x menaik, titik kiri yang disisipkan untuk satu titik kanan juga sah untuk setiap titik kanan berikutnya, sehingga tidak ada titik yang disisipkan dua kali, dan seluruh sapuan linear kecuali biaya Fenwick. Kueri `fenQuery(py[i] - 1)` memberikan `dp` terbaik di antara titik yang telah disisipkan dengan y lebih kecil; bila tidak ada, hasilnya 0 dan `dp[i]` tetap sekurang-kurangnya 1. Baris terakhir membatalkan tepat `j` penyisipan tadi.

`solve(l, r)`. Tiga panggilan, dalam satu-satunya urutan yang benar, sebagaimana dibahas pada Subbab 11.2.3.

`main`. Fungsi ini membaca titik, membangun daftar terurut nilai y yang berbeda dengan `sort` dan `unique`, lalu mengganti setiap y dengan `lower_bound + 1`. Setelah `solve(0, n - 1)` setiap `dp` sudah final, dan jawabannya adalah yang terbesar. Penjaga `n ? ... : 0` mencegah pemanggilan `max_element` pada vektor kosong.

11.4.6 Jebakan

JEBAKAN. Empat kesalahan membuat peserta kehilangan soal ini. (1) *Urutan panggilan salah:* memanggil CROSS setelah kedua panggilan rekursif. Program lolos contoh dan gagal pada rantai panjang. (2) *Perbandingan tidak tegas:* `<=` pada gelung `while` atau `py[i]` alih-alih `py[i] - 1` pada kueri menganggap koordinat yang sama sebagai naik. (3) *Mengosongkan dengan fill:* mengatur ulang seluruh pohon Fenwick pada setiap langkah *cross* mengubah $O(n \log^2 n)$ menjadi $O(n^2)$ dan melampaui batas waktu. (4) *Mengurutkan larik aslinya:* menyusun ulang `px` dan `py` menurut `x` di langkah *cross* merusak urutan masukan yang diandalkan rekursi paruh kanan; urutkan daftar indeks saja.

11.5 Mengujinya: CDQ Melawan DP Kuadratik

Program cepat yang salah tidak ada harganya, maka setiap studi kasus dalam buku ini diakhiri dengan *uji stres*: program lambat yang jelas benar, pembangkit masukan acak kecil, dan skrip yang membandingkan keduanya pada ratusan masukan. Untuk LIS2, program yang jelas benar adalah *Persamaan* (11.2) yang ditulis secara langsung, pada **Kode 11.4**.

Kode 11.4 `brute.cpp`: program dinamis $O(n^2)$ yang dipakai sebagai acuan.

```
// LIS2 reference: the plain  $O(n^2)$  dynamic program, for stress testing only.
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n;
    if (scanf("%d", &n) != 1) return 0;
    vector<int> x(n), y(n), dp(n, 1);
    for (int i = 0; i < n; i++)
        if (scanf("%d %d", &x[i], &y[i]) != 2) return 0;
    int best = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++)
            if (x[j] < x[i] && y[j] < y[i]) dp[i] = max(dp[i], dp[j] + 1);
        best = max(best, dp[i]);
    }
    printf("%d\n", best);
    return 0;
}
```

Pembangkit pada **Kode 11.5** menghasilkan sampai 14 titik dengan koordinat dalam rentang mungil seperti $[-3, 3]$. Rentang mungil itu disengaja: nilai `x` dan `y` yang sama menjadi sering muncul, dan di situlah kesalahan ketegasan bersarang. Benih acak diambil dari baris perintah, sehingga setiap kasus gagal dapat direproduksi dengan tepat.

Kode 11.5 `gen.py`: masukan acak kecil dengan banyak nilai kembar.

```
# Random LIS2 input: few points and a tiny coordinate range, so ties are common.
import random
import sys

rnd = random.Random(int(sys.argv[1]))
n = rnd.randint(1, 14)
k = rnd.randint(1, 6)
print(n)
for _ in range(n):
    print(rnd.randint(-k, k), rnd.randint(-k, k))
```

Penggerak pada **Kode 11.6** mengompilasi kedua program dengan semua peringatan aktif dan membandingkan keluarannya byte demi byte. Penggerak yang sama, dengan nama soal yang berbeda, dipakai pada Bab 12 sampai 15.

Kode 11.6 `stress.sh`: penggerak uji stres.

```
#!/usr/bin/env bash
# Stress test: LIS2.cpp against brute.cpp on random inputs made by gen.py.
# Usage: ./stress.sh [number_of_tests] (default 500)
set -e
N=${1:-500}
g++ -std=c++17 -O2 -Wall -Wextra -o sol LIS2.cpp
g++ -std=c++17 -O2 -Wall -Wextra -o brute brute.cpp
for i in $(seq 1 "$N"); do
    python3 gen.py "$i" > in.txt
    ./sol < in.txt > out_sol.txt
    ./brute < in.txt > out_brute.txt
    if ! cmp -s out_sol.txt out_brute.txt; then
        echo "Mismatch on seed $i:"; cat in.txt; exit 1
    fi
done
echo "All $N tests passed."
```

Menjalankan `./stress.sh` mencetak:

```
All 500 tests passed.
```

Sangat mendidik untuk merusak program dengan sengaja dan melihat uji stres menangkapnya. Tukarkan baris `cross(l, mid, r)`; dan `solve(mid + 1, r)`; lalu jalankan lagi: dalam beberapa benih saja skrip berhenti pada masukan yang jawaban sebenarnya 3 atau 4 dan mencetak bilangan yang lebih kecil. Ubah `px[left[j]] < px[i]` menjadi `<=` dan ketidakcocokan pertama muncul pada masukan dengan dua titik yang berbagi nilai x . Kebiasaan ini, merusak solusi untuk melihat apakah uji menyadarinya, memberi tahu Anda apakah uji Anda cukup kuat.

Kami juga mengompilasi program dengan `-fsanitize=address,undefined` dan menjalankan contoh serta semua masukan acak; tidak ada akses di luar batas atau luapan yang dilaporkan.

11.6 Taktik pada Hari Lomba

Kecepatan hanya penting bila kurang, jadi kami mengukurnya. **Tabel 11.1** mencantumkan masukan terbesar yang kami coba dan waktunya pada mesin uji kami (g++ 13, -O2, Linux x86-64).

Tabel 11.1 Waktu jalan `LIS2.cpp` pada masukan terbesar.

Masukan	N	Jawaban	Waktu
koordinat acak dalam $[-10^9, 10^9]$	100 000	107	0,15 s
diagonal (i, i)	100 000	100 000	0,06 s

Kedua kasus selesai jauh di bawah satu detik, jadi tidak perlu optimasi mikro seperti mengganti alokasi `vector` di dalam `cross` dengan larik statis. Gunakan waktu yang terhemat untuk kebenaran. Beberapa kebiasaan membantu pada hari lomba.

- **Kenali bentuknya.** Rekurensi “nilai i bergantung pada j sebelumnya dengan $a_j < a_i$ dan $b_j < b_i$ ” adalah tanda untuk CDQ. Demikian pula soal *offline* dengan *pembaruan dan kueri dalam urutan waktu*, di mana “pembaruan yang lebih awal memengaruhi kueri yang lebih akhir” berperan sebagai $j < i$; di sana CDQ membelah daftar operasi menurut waktu dan langkah *cross* menerapkan pembaruan kiri pada kueri kanan [4].

- **Tulis brute force lebih dahulu.** DP kuadratik ditulis dalam tiga menit, sekaligus menegaskan pemahaman Anda atas contoh dan menjadi acuan uji stres.
- **Letakkan tiga panggilan pada tiga baris berkomentar.** Itulah titik yang paling rawan salah, dan komentar seperti `// 1. left final` membuat urutan yang keliru menonjol ketika kode dibaca ulang.
- **Ketahui alternatifnya.** LIS 2D juga dapat diselesaikan dengan pohon segmen atas x yang berisi pohon Fenwick atas y (*online*, memori lebih besar), atau dengan sapuan atas x dan pohon Fenwick atas *posisi* bila posisi berperan sebagai koordinat ketiga. CDQ biasanya memberikan kode terpendek dengan memori terkecil.

11.6.1 CDQ atas waktu: pembaruan dan kueri

Kerangka yang sama menyelesaikan soal-soal yang sama sekali tidak berhubungan dengan subbarisan. Bayangkan daftar operasi dalam urutan waktu: “tambahkan titik (x, y) ” dan “berapa titik sejauh ini yang terletak di persegi panjang $[0, x] \times [0, y]$?”. Solusi *online* memerlukan struktur dua dimensi. CDQ menjawab semua kueri secara *offline* hanya dengan pohon Fenwick satu dimensi. Belah daftar operasi menurut waktu. Pembaruan di paruh kiri memengaruhi kueri di paruh kanan, dan tidak ada hal lain yang melintasi potongan. Pada langkah *cross*, urutkan pembaruan kiri dan kueri kanan menurut x , lakukan sapuan, tambahkan setiap pembaruan ke pohon Fenwick penjumlahan di posisi y , dan biarkan setiap kueri membaca jumlah prefiks sampai y -nya. Kueri di dalam setiap paruh ditangani rekursi. Di sini ketiga panggilan boleh datang dalam urutan apa pun, karena kueri tidak pernah mengembalikan jawabannya ke dalam pembaruan; LIS lebih sulit justru karena “jawabannya” (nilai dp) sekaligus “pembaruannya”. Melihat perbedaan itu dengan cepat adalah bagian dari mengenali versi CDQ mana yang diperlukan suatu soal.

11.7 Ringkasan Bab

- LIS dua dimensi memiliki tiga syarat urutan: posisi, x , dan y . Satu sapuan ditambah satu pohon Fenwick hanya mampu menangani dua.
- Divide and conquer CDQ membelah menurut posisi. Untuk pasangan yang melintasi potongan, syarat posisi otomatis benar, sehingga sapuan menurut x dan pohon Fenwick maksimum atas y yang dimampatkan menuntaskan pekerjaan.
- Urutan *tuntaskan kiri, silang, tuntaskan kanan* adalah kontrak kebenaran, dibuktikan oleh invarian pada Subbab 11.2.3.
- Biayanya memenuhi **Persamaan** (11.3) dan bernilai $O(n \log^2 n)$, asalkan pohon Fenwick dibersihkan hanya di posisi yang pernah ditulis.
- Ketegasan tinggal di dua tempat: $<$ pada sapuan dan $- 1$ pada kueri.
- Uji stres melawan DP $O(n^2)$, dengan rentang koordinat mungil, adalah cara tercepat menangkap kesalahan yang khas.

11.8 Soal Latihan

1. Jalankan **Pseudokode 11.1** dengan tangan pada titik $(2, 2)$, $(1, 1)$, $(3, 3)$, $(2, 4)$ dan tuliskan setiap nilai dp setelah setiap panggilan CROSS.
2. Ubah **LIS2.cpp** sehingga menghitung rantai terpanjang yang x -nya tidak turun dan y -nya naik tegas. Karakter mana yang berubah?

3. Jelaskan dalam dua atau tiga kalimat mengapa waktu jalan menjadi kuadratik bila `cross` mengatur ulang pohon Fenwick dengan `fill(fen.begin(), fen.end(), 0)`.
4. Ubah program agar juga mencetak satu rantai terpanjang, sebagai daftar posisi masukan.
5. *Penghitungan dominasi tiga dimensi*. Diberikan n titik (a_i, b_i, c_i) , hitung untuk setiap titik berapa titik lain yang ketiga koordinatnya tegas lebih kecil. Tunjukkan bagaimana CDQ mereduksinya menjadi langkah *cross* dengan sapuan dan pohon Fenwick *penjumlahan*, dan sebutkan kompleksitasnya.
6. Pada **Kode 11.1**, perbandingan `a[i] <= a[j]` mengambil elemen kiri bila keduanya sama. Apa yang dihitung program bila perbandingannya `<`?

11.9 Penyelesaian Soal Latihan

1. Mula-mula semua `dp` bernilai 1. `SOLVE(0,3)` membelah menjadi `[0,1]` dan `[2,3]`. `SOLVE(0,1)` memanggil `CROSS(0,0,1)`: titik 1, (1,1), tidak dapat mengikuti (2,2), jadi tidak ada yang berubah. `CROSS(0,1,3)`: terurut menurut x , titik kanan adalah (2,4) lalu (3,3). Untuk (2,4) kita hanya menyisipkan (1,1) (titik (2,2) ber- x sama); kueri di bawah $y = 4$ memberikan 1, jadi $dp_3 = 2$. Untuk (3,3) kita juga menyisipkan (2,2); kueri di bawah $y = 3$ memberikan 1, jadi $dp_2 = 2$. Akhirnya `SOLVE(2,3)` memanggil `CROSS(2,2,3)`: (2,4) tidak dapat mengikuti (3,3). Nilai akhirnya 1,1,2,2 dan jawabannya 2.
2. Sapuan harus menyisipkan juga titik kiri dengan x yang sama, jadi `px[left[j]] < px[i]` menjadi `px[left[j]] <= px[i]`. Itu hanya satu karakter. Kueri `py[i] - 1` tetap, karena y tetap tegas. (Titik ber- x sama di dalam satu paruh tetap ditangani dengan benar oleh rekursi, karena urutan posisi tidak disentuh.)
3. Ada satu langkah *cross* per simpul dalam pohon rekursi, yaitu $n - 1$ buah, dan setiap pengaturan ulang penuh berbiaya $\Theta(m)$ dengan m banyaknya nilai y yang berbeda, sampai n . Totalnya $\Theta(n \cdot m)$, yang pada kasus terburuk $\Theta(n^2)$: 10^{10} operasi tulis untuk $n = 10^5$.
4. Simpan bersama setiap nilai Fenwick indeks yang menghasilkannya, misalnya sebagai `pair<int,int>` (`dp`, indeks) yang dibandingkan dengan `max`. Ketika `dp[i]` membaik di langkah *cross*, catat `parent[i]` sebagai indeks tersebut. Di akhir, mulailah dari titik dengan `dp` terbesar dan ikuti `parent` ke belakang, lalu balik daftarnya. Kompleksitasnya tidak berubah.
5. Urutkan titik menurut a (pecahkan nilai kembar sedemikian rupa sehingga nilai a yang sama tidak saling menghitung, misalnya dengan mengelompokkan titik yang sama lebih dahulu). Posisi terurut kini berperan sebagai waktu. CDQ membelah daftar itu; pada langkah *cross*, urutkan kedua paruh menurut b , sapu paruh kanan dengan b menaik, tambahkan $+1$ di posisi c_j pada pohon Fenwick penjumlahan untuk setiap titik kiri dengan b lebih kecil, lalu tambahkan jumlah prefiks di bawah c_i ke hitungan i . Di sini urutan ketiga panggilan tidak penting, karena hitungan tidak mengalir kembali ke hitungan lain. Biayanya lagi-lagi $O(n \log^2 n)$ [4].
6. Dengan `<`, elemen yang sama diambil dari kanan lebih dahulu dan masing-masing terhitung sebagai inversi dengan setiap elemen sama yang masih menunggu di kiri. Program akan menghitung pasangan $i < j$ dengan $a_i \geq a_j$, bukan $a_i > a_j$.

Daftar Pustaka

- [1] D. Chen, "From 'Cash' to a class of divide-and-conquer algorithms" (in Chinese), China National Olympiad in Informatics Team Selection Papers, 2008.
- [2] P. M. Fenwick, "A new data structure for cumulative frequency tables," *Software: Practice and Experience*, vol. 24, no. 3, pp. 327–336, 1994.

- [3] A. Laaksonen, *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, 3rd ed. Cham, Switzerland: Springer, 2024.
- [4] S. Halim, F. Halim, and S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 2. Lulu Press, 2020.
- [5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
- [6] Sphere Online Judge, “LIS2 - Another Longest Increasing Subsequence Problem.” [Online]. Available: <https://www.spoj.com/problems/LIS2/>

Bab 12

Meet in the Middle: Membelah Ledakan Eksponensial — ABCDEF

Sebagian soal meminta kita mencoba “semua kombinasi”, dan banyaknya kombinasi itu astronomis: 2^{40} himpunan bagian, atau 100^6 cara mengisi enam tempat. Tidak ada trik di dalam gelung yang dapat menyelamatkan program yang harus mengunjungi 10^{12} kasus. *Meet in the middle* (bertemu di tengah) menghindari kunjungan itu sama sekali. Teknik ini membelah pilihan menjadi dua paruh yang saling bebas, mendaftarkan setiap hasil dari masing-masing paruh secara terpisah, lalu membiarkan kedua daftar “berjabat tangan” melalui satu nilai penghubung, biasanya dengan pengurutan dan pencarian biner. Pekerjaannya turun dari hasil kali kedua paruh menjadi kira-kira jumlahnya. Di akhir bab ini Anda akan mampu mengenali soal yang dapat dibelah seperti ini, menghitung pasangan yang cocok dengan `lower_bound` dan `upper_bound`, memperkirakan waktu dan memori sebelum menulis satu baris pun, dan menyelesaikan soal SPOJ ABCDEF, tempat 10^{12} tupel runtuh menjadi dua daftar berisi 10^6 bilangan.

12.1 Di Mana Ini Berguna: Paket Bu Sari

Bu Sari mengelola loket pengiriman kecil di desanya. Para petani membawa paket buah, dan truk kurir yang lewat pada sore hari hanya menerima muatan yang berat totalnya berada dalam jendela sempit, misalnya antara 95 dan 100 kilogram. Pada hari yang ramai ada empat puluh paket di lantai. Berapa banyak pilihan paket berbeda yang akan diterima truk?

Setiap paket ada di dalam pilihan atau tidak, jadi terdapat 2^{40} pilihan, sekitar $1.1 \cdot 10^{12}$. Program yang memeriksa satu miliar pilihan per detik, yang sudah sangat optimistis, tetap memerlukan hampir dua puluh menit. Padahal pertanyaan ini memiliki struktur yang diabaikan oleh *brute force*. Letakkan dua puluh paket pertama di sisi kiri ruangan dan dua puluh lainnya di sisi kanan. Setiap pilihan adalah pilihan kiri ditambah pilihan kanan, dan satu-satunya yang menghubungkan keduanya adalah berat total: bila bagian kiri seberat x , bagian kanan harus seberat antara $95 - x$ dan $100 - x$. Hanya ada $2^{20} \approx 10^6$ pilihan kiri dan 10^6 pilihan kanan. Daftarkan berat kanan sekali, urutkan, dan untuk setiap berat kiri hitung berat kanan yang berada di dalam jendela dengan dua pencarian biner. Sejuta pencarian biner hanya memakan beberapa milidetik.

Itulah seluruh gagasan bab ini: bila ruang pencarian yang besar merupakan *hasil kali* dua bagian bebas yang hanya terhubung oleh satu nilai, daftarkan setiap bagian secara terpisah dan cocokkan keduanya melalui nilai itu. Teknik ini sudah tua; Horowitz dan Sahni memakainya pada 1974 untuk menyelesaikan soal *knapsack* secara eksak dalam sekitar $2^{n/2}$ langkah [1], dan teknik ini masih muncul di olimpiade nasional setiap tahun dengan penyamaran baru [2], [3].

Bentuk yang sama bersembunyi di kisah lain. Seorang guru yang menyiapkan lomba matematika ingin memilih enam soal, satu dari setiap enam kelompok topik, sehingga skor kesulitan totalnya tepat 60. Bila setiap kelompok berisi 100 soal, ada 10^{12} pilihan. Tetapi tiga kelompok pertama dan tiga kelompok terakhir hanya bertemu melalui total: bila tiga soal pertama berjumlah x , tiga soal terakhir harus berjumlah $60 - x$. Dua daftar berisi 10^6 jumlah parsial, satu pengurutan, dan sejuta pencarian biner menjawabnya dalam jauh kurang dari satu detik. Seorang ahli keamanan

yang menguji sandi dengan kunci 40 bit yang diterapkan dalam dua tahap bebas masing-masing 20 bit memakai trik yang sama untuk menemukan kuncinya: enkripsi maju dari teks asli dengan semua 2^{20} kunci tahap pertama, dekripsi mundur dari teks sandi dengan semua 2^{20} kunci tahap kedua, lalu cari nilai yang muncul di kedua daftar. Serangan itulah alasan mengapa “enkripsi ganda” tidak dua kali lebih kuat daripada enkripsi tunggal, dan serangan itu memang disebut *meet-in-the-middle attack*.

12.2 Tekniknya dari Nol

12.2.1 Bentuk yang perlu dicari

Misalkan calon solusi adalah pasangan (p, q) , dengan p berasal dari himpunan P berisi “pilihan kiri” dan q dari himpunan Q berisi “pilihan kanan”. *Brute force* mencoba semua $|P| \cdot |Q|$ pasangan. Meet in the middle bekerja bila syarat pada pasangan dapat ditulis sebagai

$$g(p) = h(q) \quad \text{atau, lebih umum,} \quad g(p) + h(q) \in [A, B] \quad (12.1)$$

untuk fungsi g dan h yang masing-masing hanya melihat satu sisi. Nilai $g(p)$ adalah *nilai penghubung* pilihan kiri: begitu kita mengetahuinya, tidak ada hal lain tentang p yang penting bagi pencocokan. **Persamaan** (12.1) adalah hal yang perlu dicari ketika membaca soal. Jumlah, selisih, hasil kali yang dapat dipindahkan ke satu sisi, XOR, dan sisa bagi semuanya cocok.

12.2.2 Resepnya

1. **Belah** variabel atau objek menjadi dua kelompok yang hanya berinteraksi melalui nilai penghubung.
2. **Daftarkan** setiap kelompok: hitung daftar $L = [g(p) : p \in P]$ dan daftar $R = [h(q) : q \in Q]$.
3. **Urutkan** salah satu daftar, misalnya R .
4. **Cocokkan**: untuk setiap nilai v di L , hitung elemen R yang cocok dengan v . Untuk kesamaan, elemen itu adalah yang sama dengan v ; dalam larik terurut mereka membentuk satu *blok* bersambung, dan panjangnya adalah `upper_bound(v) - lower_bound(v)`.
5. **Jumlahkan** hitungannya, dalam bilangan bulat 64 bit.

Pseudokode 12.1 menuliskan resep ini untuk kasus kesamaan.

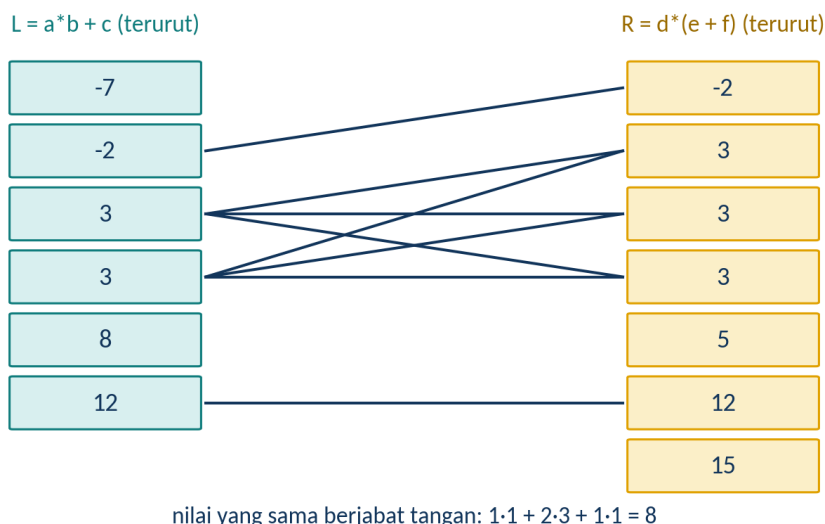
Pseudokode 12.1 Menghitung pasangan yang cocok dengan meet in the middle.

```
MITM-COUNT(P, Q)
1  L = [ g(p) for every p in P ]
2  R = [ h(q) for every q in Q ]
3  sort R
4  total = 0
5  for each v in L
6      total = total + (UPPER-BOUND(R, v) - LOWER-BOUND(R, v))
7  return total
```

DEFINISI. Untuk larik terurut R dan nilai v , `lower_bound(v)` adalah posisi pertama yang elemennya $\geq v$ dan `upper_bound(v)` adalah posisi pertama yang elemennya $> v$. Elemen yang sama dengan v menempati tepat posisi dari `lower_bound(v)` sampai sebelum `upper_bound(v)`, jadi banyaknya adalah selisih kedua posisi itu. Dalam C++, `equal_range(v)` mengembalikan kedua posisi sekaligus [4].

Mengapa hitungannya benar. Setiap pasangan (p, q) dihitung paling banyak sekali: ketika gelang tiba di $g(p)$, ia menghitung q jika dan hanya jika $h(q) = g(p)$. Dan setiap pasangan yang

cocok dihitung sekurang-kurangnya sekali, dengan alasan yang sama. Bila beberapa pilihan kiri berbagi nilai penghubung yang sama, masing-masing menjadi elemen tersendiri di L dan masing-masing menghitung pasangannya, dan itu benar karena mereka adalah pasangan yang berbeda. **Gambar 12.1** memperlihatkan jabat tangan pada dua daftar mungil: nilai kiri 3 muncul dua kali dan nilai kanan 3 tiga kali, sehingga keduanya menyumbang $2 \cdot 3 = 6$ pasangan.



Gambar 12.1 Dua daftar terurut dan jabat tangannya: setiap garis adalah satu pasangan yang dihitung, dan nilai yang sama membentuk blok yang ukurannya dikalikan.

12.2.3 Waktu dan memori, diperkirakan sebelum menulis kode

Misalkan $|L| = |R| = K$. Membangun daftar berbiaya $O(K)$, mengurutkan R berbiaya $O(K \log K)$, dan K pasang pencarian biner berbiaya $O(K \log K)$. Jadi

$$T = O(K \log K) \quad \text{alih-alih} \quad O(K^2) \quad (12.2)$$

Untuk jumlah himpunan bagian dari n butir, $K = 2^{n/2}$, dan **Persamaan** (12.2) mengubah 2^{40} menjadi sekitar $2^{20} \cdot 20 \approx 2 \cdot 10^7$ langkah. Harganya adalah memori: kedua daftar tinggal di RAM pada saat yang sama. Selalu perkiraan: 10^6 nilai bertipe `long long` memakan 8 MB, jadi dua daftar memakan 16 MB, jauh di bawah batas lazim 256 MB; 2^{25} nilai per daftar akan memerlukan 512 MB dan tidak muat. Bila memori menjadi hambatan, Anda dapat mengurutkan *kedua* daftar dan menelusurinya dengan dua penunjuk (Soal Latihan 3), atau hanya menyimpan satu daftar dan membangkitkan daftar lain sambil jalan; Schroeppe dan Shamir bahkan menunjukkan cara menurunkan memori sampai sekitar $2^{n/4}$ untuk jumlah himpunan bagian sambil menjaga waktu tetap dekat $2^{n/2}$ [5].

WAWASAN KUNCI: Meet in the middle tidak membuat soal eksponensial menjadi polinomial; teknik ini mengambil akar kuadrat dari ruang pencarian. Inilah alat yang tepat justru ketika $\sqrt{\text{ruang pencarian}}$ sekitar 10^6 sampai 10^7 : $n \approx 40$ pilihan biner, atau tiga variabel bebas dari enam dengan masing-masing 100 nilai.

12.2.4 Jalannya dengan tangan

Ambil empat paket seberat 3, 5, 6, dan 8 kilogram serta jendela [9,11] kilogram. Paruh kiri {3,5} memiliki jumlah himpunan bagian 0,3,5,8; paruh kanan {6,8} memiliki jumlah himpunan bagian 0,6,8,14, yang sudah terurut. Sekarang cocokkan:

- jumlah kiri 0 memerlukan jumlah kanan dalam [9,11]: tidak ada;

- jumlah kiri 3 memerlukan [6,8]: jumlah kanan 6 dan 8, dua himpunan bagian;
- jumlah kiri 5 memerlukan [4,6]: jumlah kanan 6, satu himpunan bagian;
- jumlah kiri 8 memerlukan [1,3]: tidak ada.

Totalnya 3, dan memang pilihan yang diterima adalah {3,6}, {3,8}, dan {5,6}. Kita hanya melihat empat jumlah kiri dan melakukan dua pencarian biner untuk masing-masing, alih-alih mendaftar keenam belas himpunan bagian. Dengan empat puluh paket, pekerjaan yang sama adalah sejuta jumlah kiri alih-alih satu triliun himpunan bagian, dan perbedaan itulah inti teknik ini.

Satu detail kecil dalam jalannya ini layak diperhatikan: jumlah kiri 3 menemukan *dua* pasangan. Meet in the middle tidak pernah bertanya “adakah pasangan?” ketika soalnya menghitung; ia bertanya “berapa banyak pasangan?”, dan jawabannya datang dari lebar suatu selang di dalam daftar terurut, bukan dari pencarian ya-atau-tidak.

12.3 Soal Pemanasan

12.3.1 Menghitung pasangan dengan jumlah tertentu

Meet in the middle yang paling kecil memiliki satu variabel di setiap sisi: diberikan dua daftar a dan b serta target T , hitung pasangan (i, j) dengan $a_i + b_j = T$. Nilai penghubung a_i adalah $T - a_i$, pasangan yang ia perlukan. **Kode 12.1** mengurutkan b dan menghitung blok $T - a_i$ untuk setiap i .

Kode 12.1 `two_sum_count.cpp`: menghitung pasangan dengan jumlah tertentu melalui pengurutan dan dua pencarian biner.

```
// Warm-up: how many pairs (i, j) satisfy a[i] + b[j] = T?
// Sort b once; each a[i] then asks for the run of values equal to T - a[i].
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n, m;
    long long T;
    if (scanf("%d %d %lld", &n, &m, &T) != 3) return 0;
    vector<long long> a(n), b(m);
    for (auto& v : a)
        if (scanf("%lld", &v) != 1) return 0;
    for (auto& v : b)
        if (scanf("%lld", &v) != 1) return 0;
    sort(b.begin(), b.end());
    long long pairs = 0;
    for (long long v : a) {
        auto lo = lower_bound(b.begin(), b.end(), T - v);
        auto hi = upper_bound(b.begin(), b.end(), T - v);
        pairs += hi - lo;
    }
    printf("%lld\n", pairs);
    return 0;
}
```

Dengan $a = [1,4,6,9]$, $b = [4,6,6,9,1]$, dan $T = 10$, pasangannya adalah $1 + 9$, $4 + 6$ (dua kali, karena b memuat 6 dua kali), $6 + 4$, dan $9 + 1$:

```
4 5 10
1 4 6 9
4 6 6 9 1
```

12.3.2 Jumlah himpunan bagian dalam jendela, $n \leq 40$

Pertanyaan Bu Sari dari Subbab 12.1 adalah soal SPOJ klasik SUBSUMS [6]: hitung himpunan bagian dari $n \leq 34$ bilangan yang jumlahnya berada dalam $[A, B]$. **Kode 12.2** menyelesaikannya untuk sampai 40 bilangan. Fungsi pembantu `sums` mendaftarkan semua jumlah himpunan bagian dari satu paruh dengan menggandakan daftar: mulai dari himpunan kosong, dan untuk setiap bilangan baru tambahkan, untuk setiap jumlah yang sudah ada, jumlah itu ditambah bilangan tersebut. Untuk syarat jendela, pasangan y dari jumlah kiri x harus memenuhi $A - x \leq y \leq B - x$, dan banyaknya y semacam itu di daftar kanan yang terurut adalah `upper_bound(B - x) - lower_bound(A - x)`, bentuk kedua dari [Persamaan \(12.1\)](#).

Kode 12.2 `subset_mitm.cpp`: menghitung himpunan bagian yang jumlahnya dalam $[A, B]$ dengan meet in the middle.

```
// Warm-up: count the subsets of n <= 40 numbers whose sum lies in [A, B].
// Split into two halves of at most 20, list both halves' subset sums, sort one.
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

vector<long long> sums(const vector<long long>& part) {
    vector<long long> out(1, 0);           // the empty subset
    for (long long v : part) {
        size_t k = out.size();
        for (size_t i = 0; i < k; i++) out.push_back(out[i] + v);
    }
    return out;
}

int main() {
    int n;
    long long A, B;
    if (scanf("%d %lld %lld", &n, &A, &B) != 3) return 0;
    vector<long long> a(n);
    for (auto& v : a)
        if (scanf("%lld", &v) != 1) return 0;
    vector<long long> left(a.begin(), a.begin() + n / 2);
    vector<long long> right(a.begin() + n / 2, a.end());
    vector<long long> X = sums(left), Y = sums(right);
    sort(Y.begin(), Y.end());
    long long cnt = 0;
    for (long long x : X) // need A - x <= y <= B - x
        cnt += upper_bound(Y.begin(), Y.end(), B - x)
            - lower_bound(Y.begin(), Y.end(), A - x);
    printf("%lld\n", cnt);
    return 0;
}
```

Untuk lima bilangan 2, -1, 4, 3, 5 dan jendela [3, 7] program mencetak:

```
5 3 7
2 -1 4 3 5
```

14

Anda dapat memeriksa angka 14 dengan tangan dalam beberapa menit, atau lebih baik, dengan *brute force* tiga baris atas semua $2^5 = 32$ himpunan bagian. Himpunan kosong juga ikut dihitung; jumlahnya 0 berada di luar jendela ini, jadi ia tidak termasuk dalam 14 itu.

12.4 Studi Kasus: ABCDEF

12.4.1 Soalnya dengan kata-kata kita sendiri

Soal SPOJ ABCDEF [7] memberikan himpunan S berisi n bilangan bulat berbeda, $1 \leq n \leq 100$, masing-masing antara $-30\,000$ dan $30\,000$. Hitung tupel enam (a, b, c, d, e, f) dari elemen S (boleh berulang) yang memenuhi

$$\frac{a \cdot b + c}{d} - e = f, \quad d \neq 0 \quad (12.3)$$

Masukannya adalah n diikuti elemen-elemennya; keluarannya adalah hitungan tersebut.

12.4.2 Dari satu persamaan menjadi dua sisi

Brute force atas keenam variabel berbiaya $n^6 = 10^{12}$ langkah. Langkah pertama adalah aljabar. Tambahkan e ke kedua ruas [Persamaan](#) (12.3) dan kalikan dengan d , yang diperbolehkan karena $d \neq 0$:

$$a \cdot b + c = d \cdot (e + f), \quad d \neq 0 \quad (12.4)$$

Kini [Persamaan](#) (12.4) memiliki tepat bentuk [Persamaan](#) (12.1): ruas kiri hanya bergantung pada (a, b, c) dan ruas kanan hanya pada (d, e, f) . Setiap ruas memiliki $n^3 = 10^6$ kombinasi. Daftar $L =$ setiap $a \cdot b + c$, daftar $R =$ setiap $d \cdot (e + f)$ dengan $d \neq 0$, urutkan R , dan hitung jabat tangan dengan [Pseudokode 12.1](#). Pekerjaannya turun dari 10^{12} menjadi sekitar $10^6 \cdot 20 = 2 \cdot 10^7$ perbandingan.

Mengapa pembelahan ini dan bukan yang lain? Cobalah memindahkan e ke kiri: $a \cdot b + c - d \cdot e = d \cdot f$. Kini d muncul di kedua ruas, sehingga kedua sisi tidak lagi bebas. Pembelahan harus menempatkan setiap variabel tepat di satu sisi, dan [Persamaan](#) (12.4) adalah yang seimbang, dengan masing-masing tiga variabel.

12.4.3 Di mana $d \neq 0$ ditegakkan

Syarat $d \neq 0$ milik ruas kanan. Kita menegakkannya *di tempat nilai itu lahir*: gelung atas d cukup melewati nol, sehingga tidak ada nilai terlarang yang pernah masuk ke R . Alternatifnya, memasukkan setiap hasil kali ke R lalu mengurangkan kecocokan terlarang di akhir, lebih panjang dan mudah keliru. Perhatikan bahwa $d = 0$ menghasilkan nilai 0 untuk setiap (e, f) ; jika 0 muncul di L , yang terjadi bila $0 \in S$ atau suatu $a \cdot b = -c$, kecocokan palsu itu akan terhitung dalam jumlah besar.

12.4.4 Contoh-contohnya

Pernyataan soal disertai empat contoh, dan contoh-contoh itu layak dikerjakan dengan tangan. Untuk $S = \{1\}$ satu-satunya tupel adalah semua bernilai satu: $1 \cdot 1 + 1 = 2 = 1 \cdot (1 + 1)$, jadi hitungannya 1. Untuk $S = \{2, 3\}$ jawabannya 4, untuk $S = \{-1, 1\}$ jawabannya 24, dan untuk $S = \{5, 7, 10\}$ jawabannya 10. Setiap masukan terdiri atas dua baris, n lalu himpunannya, misalnya [3](#) diikuti [5](#) [7](#) [10](#). Program kita, dijalankan pada keempat masukan satu per satu, mencetak:

```
1
4
24
10
```

Salah satu contoh layak diperiksa dengan tangan, karena memperlihatkan mengapa nilai berulang harus tetap berada di dalam daftar. Ambil $S = \{-1, 1\}$. Hasil kali $a \cdot b$ adalah $1, -1, -1, 1$, dan menambahkan $c \in \{-1, 1\}$ menghasilkan delapan nilai kiri $0, 2, -2, 0, -2, 0, 0, 2$: nilai 0 empat kali, 2 dua kali, dan -2 dua kali. Di kanan, $e + f$ bernilai $-2, 0, 0, 2$; dikalikan $d = -1$ menjadi $2, 0, 0, -2$ dan dikalikan $d = 1$ menjadi $-2, 0, 0, 2$, sehingga daftar kanan juga memuat 0 empat kali, 2 dua kali, dan -2 dua kali. Jabat tangannya $4 \cdot 4 + 2 \cdot 2 + 2 \cdot 2 = 24$, jawaban yang diharapkan. Seandainya duplikat dibuang dari salah satu daftar, kita hanya akan menghitung 3.

12.4.5 Solusi lengkap

Kode 12.3 adalah program utuhnya. Paruh pertama inti, membangun kedua daftar, berada di luar penanda; jabat tangannya, pengurutan dan penghitungan, adalah bagian di antara `// <P50>` dan `// </P50>`.

Kode 12.3 `ABCDEF.cpp`: menghitung tupel **Persamaan** (12.4) dengan meet in the middle.

```
// ABCDEF: count the six-tuples from the set S with (a*b + c) / d - e = f and
// d != 0, i.e. a*b + c = d*(e + f). Meet in the middle: n^3 values per side.
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n;
    if (scanf("%d", &n) != 1) return 0;
    vector<long long> s(n);
    for (auto& v : s)
        if (scanf("%lld", &v) != 1) return 0;
    vector<long long> L, R;
    L.reserve((size_t)n * n * n);
    R.reserve((size_t)n * n * n);
    for (long long a : s) // Left side: a*b + c
        for (long long b : s)
            for (long long c : s) L.push_back(a * b + c);
    for (long long d : s) { // right side: d*(e + f)
        if (d == 0) continue; // d = 0 is illegal: never born
        for (long long e : s)
            for (long long f : s) R.push_back(d * (e + f));
    }
    // <P50>
    // Sort R; each Left value v matches as many right values as are equal to
    // v, which is the width of v's run in R: upper_bound - lower_bound.
    // >>> YOUR TURN (Public Edition): write this part yourself, following the hint above.
    <<<
    // </P50>
    printf("%lld\n", answer);
    return 0;
}
```

12.4.6 Baris demi baris

Pembacaan. Elemen dibaca ke dalam `vector<long long> s`. Memakai bilangan 64 bit sejak awal berarti setiap hasil kali di bawahnya dihitung dalam 64 bit tanpa konversi tipe.

Dua panggilan `reserve` menyiapkan ruang untuk n^3 elemen sekali saja, sehingga `push_back` tidak pernah melakukan alokasi ulang. Untuk $n = 100$ itu 10^6 elemen, 8 MB per daftar.

Gelung kiri menghasilkan setiap $a \cdot b + c$. Ada tepat n^3 nilai, termasuk yang berulang; nilai berulang harus tetap ada, karena tupel yang berbeda adalah solusi yang berbeda.

Gelung kanan menghasilkan setiap $d \cdot (e + f)$ dengan $d \neq 0$: paling banyak n^3 nilai, lebih sedikit bila $0 \in S$.

Jabat tangan. `sort(R.begin(), R.end())` mengumpulkan nilai yang sama. Untuk setiap nilai kiri v , `equal_range` mengembalikan pasangan `(lower_bound, upper_bound)` dari blok v di R , dan selisih kedua iterator adalah ukuran blok itu. Itulah banyaknya tripel kanan (d, e, f) yang melengkapi tripel kiri menjadi solusi. Akumulator `answer` bertipe `long long`.

Mengapa bukan tiga gelung dan satu pencarian? Jalan pintas yang menggoda adalah menggelung (a, b, c) dan, untuk setiap nilai kiri, menggelung d dan memeriksa apakah $v/d - e$ ada di himpunan untuk suatu e . Itu $n^4 = 10^8$ langkah dengan pembagian dan pencarian di himpunan: mungkin terlalu lambat, dan penuh kasus keterbagian. Pembelahan yang bersih pada **Persamaan** (12.4) menjaga kedua sisi simetris dan membuat setiap langkah menjadi operasi bilangan bulat biasa.

Anggaran yang ditulis sebelum menulis kode. Untuk $n = 100$: membangun L dan R memerlukan $2 \cdot 10^6$ perkalian; mengurutkan R memerlukan sekitar $10^6 \cdot 20 = 2 \cdot 10^7$ perbandingan; 10^6 panggilan `equal_range` memerlukan sekitar $2 \cdot 10^6 \cdot 20 = 4 \cdot 10^7$ perbandingan. Totalnya kira-kira $6 \cdot 10^7$ operasi sederhana, jauh di bawah satu detik. Waktu terukur pada Subbab 12.6 adalah 0,14 detik, sesuai dengan perkiraan. Menjadikan perkiraan semacam ini kebiasaan berarti Anda jarang mengirim solusi yang terlalu lambat, dan tidak pernah membuang waktu mengoptimalkan solusi yang sudah cukup cepat.

12.4.7 Bilangan yang menyelamatkan atau menguburkan

Periksa rentang nilai sebelum memilih tipe; **Tabel 12.1** melakukannya untuk soal ini.

Tabel 12.1 Rentang nilai pada ABCDEF: nilainya muat dalam 32 bit, hitungannya tidak.

Besaran	Nilai mutlak terbesar	Muat dalam <code>int</code> ?
$a \cdot b + c$	$30\,000^2 + 30\,000 = 900\,030\,000$	ya
$d \cdot (e + f)$	$30\,000 \cdot 60\,000 = 1.8 \cdot 10^9$	ya, nyaris
jawaban	sampai $n^6 = 10^{12}$	tidak
memori L dan R	$2 \cdot 10^6 \cdot 8 \text{ bytes} = 16 \text{ MB}$	—

Nilainya memang muat dalam `int`, tetapi di ruas kanan hanya dengan sisa sekitar 16%, dan satu perubahan ceroboh (misalnya varian soal dengan $e + f + g$) akan meluap tanpa suara. Hitungannya jelas memerlukan 64 bit: dengan $S = \{-50, \dots, 49\}$ program kita melaporkan 349 653 095 tupel, dan himpunan yang lebih besar dapat menghasilkan lebih dari $2^{31} \approx 2.1 \cdot 10^9$.

JEBAKAN. (1) Akumulator `int` meluap pada jawaban besar. (2) Memasukkan hasil kali dengan $d = 0$ ke R menghitung tupel yang mustahil. (3) Membuang nilai berulang dari L atau R dengan `unique` menghilangkan solusi: setiap tripel adalah cara tersendiri. (4) Membaca persamaan sebagai pembagian bilangan bulat dan mencoba memeriksa bahwa d membagi habis $a \cdot b + c$ tidak perlu: **Persamaan** (12.4) setara dengan **Persamaan** (12.3) untuk $d \neq 0$, dan itulah yang dihitung juri.

12.5 Mengujinya: Meet in the Middle Melawan n^6

Acuan untuk ABCDEF adalah persamaan itu sendiri, diperiksa untuk setiap tupel, pada **Kode 12.4**. Dengan $n \leq 6$ program ini melakukan paling banyak $6^6 = 46\,656$ pemeriksaan.

Kode 12.4 `brute.cpp`: acuan n^6 .

```
// ABCDEF reference: try all  $n^6$  tuples (only for tiny  $n$ ).
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n;
    if (scanf("%d", &n) != 1) return 0;
    vector<long long> s(n);
    for (auto& v : s)
        if (scanf("%lld", &v) != 1) return 0;
    long long cnt = 0;
    for (long long a : s) for (long long b : s) for (long long c : s)
        for (long long d : s) for (long long e : s) for (long long f : s)
            if (d != 0 && a * b + c == d * (e + f)) cnt++;
    printf("%lld\n", cnt);
    return 0;
}
```

Pembangkit pada **Kode 12.5** mengambil sampai enam nilai berbeda dari rentang kecil di sekitar nol, sehingga nol sering berada di dalam himpunan (menguji aturan $d \neq 0$) dan banyak tupel cocok (menguji penghitungan blok). Sesekali ia memakai rentang penuh $\pm 30\,000$ untuk menguji hasil kali yang besar.

Kode 12.5 `gen.py`: himpunan acak kecil untuk ABCDEF.

```
# Random ABCDEF input: up to 6 distinct values from a small range (0 included
# often), so that many tuples match.
import random
import sys

rnd = random.Random(int(sys.argv[1]))
k = rnd.choice([3, 5, 10, 30000])
n = rnd.randint(1, min(6, 2 * k + 1))
vals = rnd.sample(range(-k, k + 1), n)
print(n)
print(*vals)
```

Penggeraknya adalah `stress.sh` yang sama seperti pada Bab 11 dengan `ABCDEF.cpp` menggantikan `LIS2.cpp`. Hasilnya:

```
All 500 tests passed.
```

12.6 Taktik pada Hari Lomba

Tabel 12.2 memperlihatkan program pada masukan terbesar, diukur pada mesin uji kami (g++ 13, -O2).

Tabel 12.2 Waktu jalan `ABCDEF.cpp` pada ukuran maksimum.

Masukan ($n=100$)	Jawaban	Waktu
100 nilai acak berbeda dalam $[-30\,000, 30\,000]$	2 700	0,14 s
nilai $-50, -49, \dots, 49$	349 653 095	0,12 s

Beberapa pelajaran untuk ruang lomba:

- **Baca batasan sebagai petunjuk.** $n \leq 40$ untuk himpunan bagian, atau enam variabel atas 100 nilai, praktis mengumumkan meet in the middle: akar kuadrat dari *brute force* sekitar 10^6 .

- **Kerjakan aljabarnya di kertas lebih dahulu.** Pembelahan pada **Persamaan** (12.4) hanya memerlukan dua baris aljabar dan menghemat 10^{12} langkah. Cobalah memindahkan setiap variabel ke sisi tempat ia hidup sendirian.
- **Utamakan pengurutan daripada `map`.** `map<long long, int>` berisi hitungan juga bekerja, tetapi beberapa kali lebih lambat dan memakai jauh lebih banyak memori untuk 10^6 kunci. `unordered_map` dapat diserang masukan khusus pada sebagian juri. `vector` terurut dengan pencarian biner cepat, kecil, dan dapat diramalkan.
- **Dua penunjuk lebih cepat lagi.** Urutkan L dan R , lalu telusuri keduanya bersama-sama: ketika nilai saat ini sama, hitung panjang blok di masing-masing dan tambahkan hasil kali panjangnya. Ini mengganti 10^6 pencarian biner dengan satu lintasan linear (Soal Latihan 3).
- **Ucapkan perkiraan memori.** “Dua daftar berisi sejuta bilangan 8 byte, 16 MB” hanya perlu lima detik untuk diucapkan dan mencegah vonis batas memori.

12.6.1 Kerabat meet in the middle

Jabat tangan yang sama muncul dalam beberapa bentuk terkenal, dan mengenalinya menghemat waktu dalam lomba.

- **XOR alih-alih jumlah.** Hitung himpunan bagian yang XOR-nya sama dengan target K : pasangan dari XOR kiri x adalah XOR kanan $x \oplus K$. Pengurutan dan pencarian biner bekerja tanpa perubahan, karena yang diperlukan hanya kesamaan.
- **Pencarian dua arah.** Pada teka-teki yang keadaannya membentuk graf raksasa (ubin geser, cincin berputar, kubus teracak), pencarian melebar dari awal sampai kedalaman k dan pencarian lain dari tujuan sampai kedalaman k bertemu di tengah; keadaan yang ditemukan kedua pencarian menghubungkan kedua paruh. Bila setiap keadaan memiliki b penerus, pekerjaannya turun dari sekitar b^{2k} menjadi sekitar $2b^k$.
- **Baby-step giant-step.** Untuk menyelesaikan $a^x \equiv b \pmod{p}$, tulis $x = i \cdot m - j$ dengan $m \approx \sqrt{p}$. Simpan semua $b \cdot a^j$ untuk $0 \leq j < m$ (langkah bayi), lalu coba a^{im} untuk $i = 1, 2, \dots$ (langkah raksasa) sampai ada yang cocok. Waktunya sekitar $\sqrt{p}$ alih-alih p [3].
- **Empat daftar.** “Pilih satu elemen dari setiap empat daftar sehingga jumlahnya nol” terbelah menjadi dua pasang daftar, persis seperti pada Soal Latihan 5.

Dalam setiap kasus, ajukan tiga pertanyaan yang sama: apa kedua paruhnya, apa nilai penghubungnya, dan seberapa besar setiap daftar? Jawabannya langsung memberi tahu apakah metode ini muat dalam batas waktu dan memori.

12.6.2 Kesalahan yang sering terjadi

Di luar jebakan ABCDEF, tiga kesalahan umum layak diberi nama. Yang pertama adalah membelah *tidak seimbang*: paruh berisi 15 dan 25 butir menghasilkan daftar berisi $3 \cdot 10^4$ dan $3 \cdot 10^7$ nilai, dan daftar yang lebih besar mendominasi waktu dan memori, jadi selalu belah seimbang sejauh strukturnya mengizinkan. Yang kedua adalah *mengurutkan daftar yang salah*: urutkan daftar tempat Anda mencari, dan iterasikan daftar yang lain; bila satu daftar jauh lebih kecil, biasanya lebih baik mengurutkan yang besar sekali dan mengiterasikan yang kecil. Yang ketiga adalah memakai *peta hash dengan kunci yang sial*: pada sebagian juri, masukan buatan membuat `unordered_map` merosot menjadi waktu linear per pencarian, sedangkan vektor terurut tidak pernah mengalami masalah itu.

12.7 Ringkasan Bab

- Meet in the middle membelah ruang pencarian berbentuk hasil kali menjadi dua paruh yang terhubung oleh satu nilai penghubung, seperti pada [Persamaan \(12.1\)](#).
- Setiap paruh didaftarkan menjadi sebuah daftar; satu daftar diurutkan; nilai yang cocok dihitung dengan `upper_bound - lower_bound` atau `equal_range`.
- Biayanya turun dari K^2 menjadi $O(K \log K)$ ([Persamaan \(12.2\)](#)), dengan harga menyimpan kedua daftar.
- ABCDEF menjadi [Persamaan \(12.4\)](#): $a \cdot b + c = d \cdot (e + f)$, dua daftar berisi n^3 nilai, dengan $d \neq 0$ ditegakkan di tempat nilai kanan diciptakan.
- Nilai berulang harus tetap berada di daftar, dan hitungannya harus 64 bit.

Meet in the middle mengingatkan bahwa bilangan besar pada batasan soal tidak selalu berarti tembok. Bila ruang pencarian berbentuk hasil kali, carilah jahitan tempat ia dapat dipotong, dan biarkan kedua paruh bertemu di sana.

12.8 Soal Latihan

1. Untuk $S = \{2,3\}$, daftarkan nilai kiri $a \cdot b + c$ dan nilai kanan $d \cdot (e + f)$, lalu periksa dengan tangan bahwa jawabannya 4.
2. Jelaskan mengapa memindahkan e ke ruas kiri [Persamaan \(12.4\)](#), seperti pada $a \cdot b + c - d \cdot e = d \cdot f$, merusak meet in the middle.
3. Tulis ulang bagian penghitungan `ABCDEF.cpp` sehingga mengurutkan kedua daftar dan menghitung kecocokan dengan dua penunjuk. Berapa kompleksitas baru tahap pencocokan?
4. Diberikan $n \leq 40$ bilangan dan target S , tentukan apakah ada himpunan bagian yang berjumlah *tepat* S dan cetak salah satunya.
5. *Menghitung 4SUM*. Diberikan empat daftar A, B, C, D sepanjang $n \leq 4000$, hitung kuadrupel dengan $a + b + c + d = 0$. Pembelahan mana yang Anda pakai, dan berapa memori yang diperlukan?
6. Soal ini meminta $n \leq 100$. Perkirakan waktu dan memori bila batasnya $n \leq 300$. Apakah pendekatan ini masih bekerja?
7. Diberikan $n \leq 36$ barang, masing-masing dengan berat dan nilai, carilah total nilai terbesar dengan total berat paling banyak W . Bagaimana meet in the middle diterapkan bila syarat penghubungnya berupa pertidaksamaan dan yang dicari adalah maksimum, bukan hitungan?
8. Jelaskan mengapa jawaban ABCDEF untuk $S = \{0\}$ adalah 0, lalu hitung jawaban untuk $S = \{0,1\}$ dengan bernalar tentang kedua daftar.

12.9 Penyelesaian Soal Latihan

1. Nilai kiri, untuk (a, b, c) berurutan 222,223,232,233,322,323,332,333: 6,7,8,9,8,9,11,12. Nilai kanan untuk (d, e, f) : dengan $d = 2$ jumlah $e + f$ adalah 4,5,5,6, menghasilkan 8,10,10,12; dengan $d = 3$ menghasilkan 12,15,15,18. Kecocokan: nilai kiri 8 muncul dua kali dan nilai kanan 8 sekali (2 pasangan); nilai kiri 12 muncul sekali dan nilai kanan 12 dua kali (2 pasangan); tidak ada nilai lain yang sama. Totalnya $2 + 2 = 4$.
2. Setelah dipindahkan, d muncul di kedua ruas. Nilai kiri tidak lagi dapat dihitung tanpa mengetahui triple kanan, sehingga kedua daftar tidak bebas dan tidak dapat didaftarkan terpisah. Setiap variabel harus hidup tepat di satu sisi.

3. Urutkan L dan R . Simpan indeks i dan j . Bila $L[i] < R[j]$ majukan i ; bila lebih besar, majukan j ; bila sama dengan nilai v , hitung panjang blok p dari v di L dan q di R , tambahkan $p * q$ (sebagai `long long`), lalu lewati kedua blok. Pencocokan kini $O(n^3)$ setelah kedua pengurutan, jadi totalnya didominasi pengurutan, $O(n^3 \log n)$.
4. Belah menjadi paruh berukuran $\lfloor n/2 \rfloor$ dan $\lfloor n/2 \rfloor$. Simpan jumlah kanan bersama `mask` himpunan bagiannya sebagai pasangan, lalu urutkan menurut jumlah. Untuk setiap jumlah kiri x dengan `mask` m_1 , cari biner $S - x$; bila ditemukan dengan `mask` m_2 , cetak elemen yang dipilih oleh m_1 dan m_2 . Waktunya $O(2^{n/2} \cdot n)$.
5. Belah menjadi (a, b) dan (c, d) . Bangun $L = [a + b]$ dan $R = [-(c + d)]$, masing-masing berukuran $n^2 = 1.6 \cdot 10^7$, urutkan R , dan hitung jabat tangan. Memori: dua daftar berisi $1.6 \cdot 10^7$ nilai; dengan nilai `int` 32 bit, masing-masing 64 MB, total 128 MB, masih di dalam 256 MB. Dengan `long long` ukurannya berlipat dua, jadi periksa rentang nilai lebih dahulu (jumlah dua nilai 32 bit mungkin memerlukan 64 bit).
6. $n^3 = 2.7 \cdot 10^7$ nilai per daftar: masing-masing 216 MB sebagai `long long`, 432 MB bersama-sama, terlalu banyak untuk 256 MB. Nilainya muat dalam 32 bit, karena $|a \cdot b + c| \leq 9 \cdot 10^8$ dan $|d \cdot (e + f)| \leq 1.8 \cdot 10^9 < 2^{31}$, sehingga daftar `int` memerlukan masing-masing 108 MB, total 216 MB, yang berisiko. Mengurutkan $2.7 \cdot 10^7$ nilai memakan sekitar dua detik. Pendekatan ini berada di ambang batas dan memerlukan daftar 32 bit serta dua penunjuk agar aman.
7. Daftarkan semua himpunan bagian setiap paruh sebagai pasangan (berat, nilai). Urutkan pasangan kanan menurut berat dan, sambil berjalan dengan berat menaik, ganti setiap nilai dengan nilai terbaik yang sejauh ini terlihat, sehingga daftar itu menjawab “nilai terbaik dengan berat paling banyak w ” melalui pencarian biner. Buang pasangan yang terdominasi bila mau. Untuk setiap pasangan kiri (w_1, v_1) dengan $w_1 \leq W$, cari biner berat kanan terbesar yang $\leq W - w_1$ dan ambil v_1 ditambah nilai terbaik prefiks di sana. Simpan maksimumnya. Waktunya $O(2^{n/2} \cdot n)$.
8. Dengan $S = \{0\}$ satu-satunya d yang mungkin adalah 0, yang terlarang, sehingga daftar kanan kosong dan tidak ada solusi. Dengan $S = \{0, 1\}$: nilai kiri $a \cdot b + c$ atas 8 tripel adalah 0 (dari $a \cdot b = 0$ dan $c = 0$: 3 tripel), 1 (dari $a \cdot b = 0$, $c = 1$: 3 tripel, dan dari $a \cdot b = 1$, $c = 0$: 1 tripel, total 4), dan 2 (1 tripel). Di kanan $d = 1$, jadi nilainya adalah $e + f$: 0 sekali, 1 dua kali, 2 sekali. Jabat tangannya $3 \cdot 1 + 4 \cdot 2 + 1 \cdot 1 = 12$.

Daftar Pustaka

- [1] E. Horowitz and S. Sahni, “Computing partitions with applications to the knapsack problem,” *Journal of the ACM*, vol. 21, no. 2, pp. 277–292, 1974.
- [2] S. Halim, F. Halim, and S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 1. Lulu Press, 2020.
- [3] A. Laaksonen, *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, 3rd ed. Cham, Switzerland: Springer, 2024.
- [4] cppreference.com, “std::equal_range.” [Online]. Available: https://en.cppreference.com/w/cpp/algorithm/equal_range
- [5] R. Schroepel and A. Shamir, “A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems,” *SIAM Journal on Computing*, vol. 10, no. 3, pp. 456–464, 1981.
- [6] Sphere Online Judge, “SUBSUMS - Subset Sums.” [Online]. Available: <https://www.spoj.com/problems/SUBSUMS/>
- [7] Sphere Online Judge, “ABCDEF - ABCDEF.” [Online]. Available: <https://www.spoj.com/problems/ABCDEF/>

Bab 13

Larik Sufiks dan LCP: Menghitung Substring Berbeda — DISUBSTR

Pertanyaan tentang substring sebuah teks muncul di setiap olimpiade: berapa banyak potongan berbeda yang dikandung sebuah kata, di mana sebuah pola muncul, berapa potongan terpanjang yang muncul dua kali? Masing-masing dapat dijawab secara lambat dengan memeriksa semua $O(n^2)$ substring. Bab ini membangun satu struktur yang menjawab semuanya dengan cepat: *larik sufiks* (*suffix array*), yaitu daftar semua sufiks teks dalam urutan terurut, bersama *larik LCP*, yang mencatat seberapa banyak setiap sufiks berbagi awalan dengan tetangganya dalam urutan itu. Anda akan belajar membangun larik sufiks dengan *penggandaan prefiks* (*prefix doubling*) dalam $O(n \log^2 n)$, menghitung larik LCP dalam $O(n)$ dengan algoritma Kasai, dan membuktikan rumus satu baris yang menyelesaikan soal SPOJ DISUBSTR: banyaknya substring berbeda sama dengan $n(n + 1)/2$ dikurangi jumlah nilai LCP.

13.1 Di Mana Ini Berguna: Kata-kata dalam Motif Batik

Sebuah bengkel batik menggambarkan setiap motifnya sebagai untaian panjang kode ragam hias: A untuk lingkaran kawung, B untuk garis parang, C untuk bunga truntum, dan seterusnya. Para perancang ingin mengetahui seberapa *kaya* suatu motif. Salah satu ukuran sederhana adalah banyaknya rentang ragam hias berbeda yang dikandungnya. Motif ABABA tampak panjang, tetapi hanya memuat sembilan rentang berbeda: A, B, AB, BA, ABA, BAB, ABAB, BABA, dan ABABA. Motif CCCCC hanya memuat lima.

Hitungan yang sama mengukur kerumitan potongan DNA dalam biologi, membantu program kompresi memutuskan apakah suatu teks berulang-ulang, dan muncul dalam deteksi plagiarisme, tempat substring panjang yang sama di dua esai patut dicurigai. Dalam semua kasus ini teksnya panjang, dan mendaftar semua substring tidak ada harapan: teks sepanjang n memiliki $n(n + 1)/2$ substring (dihitung dengan pengulangan), dan membandingkannya bahkan lebih mahal. Kita menginginkan metode yang hanya melihat teks beberapa kali.

Kuncinya adalah pengamatan sederhana yang akan kita pakai di sepanjang bab.

WAWASAN KUNCI: Setiap substring adalah *prefiks* dari suatu *sufiks*. Substring yang dimulai di posisi i dan berpanjang ℓ adalah ℓ karakter pertama dari sufiks yang dimulai di i . Jadi, untuk memahami semua substring, cukup memahami semua n sufiks beserta prefiksnya.

Mengapa tidak membangkitkan saja semua substring dan memasukkannya ke sebuah himpunan, seperti yang dilakukan *brute force* pada Subbab 13.5? Karena biayanya tumbuh sangat cepat. String sepanjang 1000 karakter memiliki 500 500 substring dengan panjang rata-rata sekitar 333 karakter, sehingga himpunan itu menampung sekitar 10^8 karakter dan setiap penyisipan membandingkan string panjang. Untuk 20 string semacam itu hasilnya sudah lambat, dan untuk 10^5 karakter mustahil: $5 \cdot 10^9$ substring. Larik sufiks hanya menyimpan n bilangan bulat, dan larik LCP n lagi. Seluruh informasi tentang $n(n + 1)/2$ substring dipadatkan ke dalam $2n$ bilangan itu, dan [Persamaan \(13.4\)](#) membaca jawabannya dengan satu lintasan. Pemampatan ini, struktur

berukuran kuadratik yang digambarkan oleh data berukuran linear, adalah yang membuat larik sufiks begitu ampuh.

13.2 Tekniknya dari Nol

13.2.1 Larik sufiks dan larik LCP

Nomori karakter string s dari 0 sampai $n - 1$, dan tulis $s[i..]$ untuk sufiks yang dimulai di posisi i . *Larik sufiks* sa adalah daftar posisi awal yang diurutkan sehingga sufiks-sufiksnya berada dalam urutan leksikografis (urutan kamus):

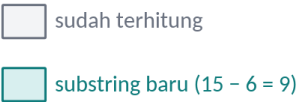
$$s[sa[0]..] < s[sa[1]..] < \dots < s[sa[n-1]..] \quad (13.1)$$

Larik LCP menyimpan, untuk setiap $k \geq 1$, panjang prefiks bersama terpanjang (*longest common prefix*) dari dua tetangga dalam urutan itu:

$$lcp[k] = \text{LCP}(s[sa[k-1]..], s[sa[k]..]), \quad lcp[0] = 0 \quad (13.2)$$

Untuk $s = \text{ABABA}$ kelima sufiks dalam urutan terurut adalah A, ABA, ABABA, BA, dan BABA, sehingga $sa = [4, 2, 0, 3, 1]$ dan $lcp = [0, 1, 3, 0, 2]$. **Gambar 13.1** menggambar tabel ini, dan arsirannya sudah mengisyaratkan rumus penghitungan pada Subbab 13.2.4. Larik sufiks diperkenalkan oleh Manber dan Myers [1] sebagai alternatif ringkas bagi pohon sufiks; ia hanya memakai beberapa larik bilangan bulat sepanjang n .

k	sa[k]	lcp	sufiks
0	4	0	A
1	2	1	A B A
2	0	3	A B A B A
3	3	0	B A
4	1	2	B A B A



Gambar 13.1 Sufiks ABABA yang telah diurutkan. $lcp[k]$ karakter pertama setiap sufiks (abu-abu) mengulang tetangga di atasnya; hanya prefiks sisanya (hijau toska) yang merupakan substring baru, sebanyak $15 - 6 = 9$.

13.2.2 Membangun larik sufiks dengan penggandaan prefiks

Mengurutkan sufiks dengan perbandingan string biasa memang bekerja, tetapi setiap perbandingan dapat memakan $O(n)$ karakter, sehingga pengurutannya berbiaya $O(n^2 \log n)$. Penggandaan prefiks menghindari perbandingan panjang dengan mengurutkan sufiks menurut 1, 2, 4, 8, ... karakter pertamanya, dan setiap putaran memakai peringkat dari putaran sebelumnya.

Misalkan $rk_g[i]$ adalah peringkat g karakter pertama sufiks i di antara g karakter pertama semua sufiks: prefiks yang sama berbagi peringkat, dan prefiks yang lebih kecil mendapat peringkat lebih kecil. $2g$ karakter pertama sufiks i adalah g karakter pertamanya disusul g karakter pertama sufiks $i + g$. Karena itu membandingkan $2g$ karakter pertama dua sufiks sama dengan membandingkan *pasangan*

$$(rk_g[i], rk_g[i + g]) \quad (13.3)$$

dengan $rk_g[i + g] = -1$ bila $i + g \geq n$ (sufiks yang berakhir lebih cepat lebih kecil, sebagaimana “AB” mendahului “ABA” dalam kamus). Urutkan posisi menurut pasangan ini, beri pasangan yang sama peringkat yang sama, dan kita memperoleh rk_{2g} . Mulailah dengan $rk_1[i] = s[i]$ (kode karakter) dan gandakan g sampai semua peringkat berbeda, yang terjadi setelah paling banyak $\lceil \log_2 n \rceil$ putaran. **Pseudokode 13.1** menuliskannya.

Pseudokode 13.1 Membangun larik sufiks dengan penggandaan prefiks.

```
SUFFIX-ARRAY(s, n)
1  for i = 0 to n - 1
2      sa[i] = i; rk[i] = code of s[i]
3  g = 1
4  while g < n
5      sort sa by the pair (rk[i], rk[i+g] or -1 if i+g >= n)
6      tmp[sa[0]] = 0
7      for k = 1 to n - 1
8          tmp[sa[k]] = tmp[sa[k-1]] + (pair of sa[k-1] < pair of sa[k] ? 1 : 0)
9      rk = tmp
10     if rk[sa[n-1]] == n - 1: stop      // semua peringkat berbeda
11     g = 2g
12 return sa
```

Jalannya dengan tangan pada ABABA. Dengan $g = 1$ pasangan posisi 0 sampai 4 adalah (A,B), (B,A), (A,B), (B,A), (A,−1). Terurut: posisi 4, lalu 0 dan 2 (pasangan sama), lalu 1 dan 3 (pasangan sama). Peringkat barunya $rk_2 = [1,2,1,2,0]$ untuk posisi 0 sampai 4: belum semuanya berbeda. Dengan $g = 2$ pasangannya adalah (1,1), (2,2), (1,0), (2,−1), (0,−1). Terurut: posisi 4 (0,−1), posisi 2 (1,0), posisi 0 (1,1), posisi 3 (2,−1), posisi 1 (2,2). Semua pasangan berbeda, peringkatnya menjadi 0..4, dan $sa = [4,2,0,3,1]$, seperti pada **Gambar 13.1**.

Biaya. Setiap putaran adalah satu pengurutan berbasis perbandingan, $O(n \log n)$, dengan perbandingan pasangan bilangan bulat yang $O(1)$; putarannya paling banyak $\lceil \log_2 n \rceil$. Totalnya $O(n \log^2 n)$. Mengganti pengurutan perbandingan dengan dua lintasan *counting sort* menghasilkan $O(n \log n)$, dan konstruksi waktu linear pun ada [2], tetapi untuk DISUBSTR, dengan $n \leq 1000$, versi sederhana berjalan dalam hitungan mikrodetik.

Mengapa penggandaan benar. Argumennya adalah induksi pada putaran. Setelah putaran pertama, peringkat mengurutkan sufiks dengan benar menurut karakter pertamanya, dan karakter yang sama berbagi peringkat. Andaikan setelah suatu putaran peringkat mengurutkan sufiks dengan benar menurut g karakter pertamanya. Dua sufiks i dan j dibandingkan pada $2g$ karakter pertamanya persis seperti paruh pertamanya dibandingkan, dan, bila paruh pertama sama, seperti paruh keduanya dibandingkan. Paruh pertama diperingkat oleh $rk_g[i]$ dan $rk_g[j]$; paruh kedua adalah g karakter pertama sufiks $i + g$ dan $j + g$, diperingkat oleh $rk_g[i + g]$ dan $rk_g[j + g]$, atau tidak ada (peringkat −1) bila sufiksnya terlalu pendek. Jadi mengurutkan menurut pasangan **Persamaan** (13.3) mengurutkan sufiks dengan benar menurut $2g$ karakter pertamanya, dan memberi pasangan yang sama peringkat yang sama menjaga invariannya. Begitu $2g \geq n$, $2g$ karakter pertama setiap sufiks adalah seluruh sufiks itu, dan semua peringkat berbeda karena sufiks yang berbeda memiliki panjang yang berbeda.

Satu jalannya lagi, pada CCCCC. Pada putaran pertama setiap pasangan adalah (C, C) kecuali sufiks terakhir, yang pasangannya (C, −1); jadi sufiks 4 mendapat peringkat 0 dan semua lainnya peringkat 1. Pada putaran kedua pasangannya (1,1) untuk sufiks 0 dan 1, (1,0) untuk sufiks 2,

$(1, -1)$ untuk sufiks 3, dan $(0, -1)$ untuk sufiks 4; terurut, itu memberikan sufiks 4, lalu 3, lalu 2, lalu 0 dan 1 bersama. Pada putaran ketiga, dengan $g = 4$, sufiks 1 memiliki pasangan $(3, -1)$ dan sufiks 0 pasangan $(3, 0)$, sehingga sufiks 1 datang lebih dahulu dan kini setiap peringkat berbeda: $sa = [4, 3, 2, 1, 0]$, yaitu sufiks C, CC, CCC, CCCC, CCCCC. Larik LCP-nya $[0, 1, 2, 3, 4]$, dan **Persamaan** (13.4) memberikan $15 - 10 = 5$. String berkarakter sama adalah kasus terburuk bagi penggandaan, karena peringkatnya baru berbeda semua pada putaran terakhir.

13.2.3 Larik LCP dalam waktu linear: algoritma Kasai

Menghitung setiap $lcp[k]$ dengan membandingkan karakter dari awal berbiaya $O(n)$ per pasangan tetangga, $O(n^2)$ totalnya. Kasai, Lee, Arimura, Arikawa, dan Park menemukan cara memakai ulang pekerjaan [3]. Kunjungi sufiks dalam *urutan teks* $i = 0, 1, \dots, n - 1$, bukan dalam urutan terurut. Fakta kuncinya adalah:

LEMA (Kasai). Misalkan h adalah LCP sufiks i dengan pendahulunya dalam urutan terurut. Maka LCP sufiks $i + 1$ dengan pendahulunya sekurang-kurangnya $h - 1$.

Sketsa bukti. Misalkan sufiks j adalah pendahulu sufiks i , berbagi $h \geq 1$ karakter. Buang karakter pertama keduanya: sufiks $j + 1$ dan $i + 1$ berbagi $h - 1$ karakter, dan sufiks $j + 1$ tetap lebih kecil daripada sufiks $i + 1$, karena karakter pertamanya sama. Jadi ada sufiks yang lebih kecil daripada sufiks $i + 1$ yang berbagi $h - 1$ karakter dengannya. Pendahulu langsung sufiks $i + 1$ terletak di antara sufiks $j + 1$ dan sufiks $i + 1$ dalam urutan terurut, dan setiap sufiks di antara dua string yang berbagi $h - 1$ karakter juga berbagi sekurang-kurangnya $h - 1$ karakter itu dengan keduanya. ▀

Jadi ketika berpindah dari i ke $i + 1$ kita dapat mulai membandingkan dari $h - 1$, bukan dari 0. Nilai h bertambah satu untuk setiap perbandingan karakter yang berhasil dan berkurang paling banyak satu per langkah, dan tidak pernah melampaui n ; karena itu banyaknya perbandingan paling banyak $2n$, dan seluruh larik LCP berbiaya $O(n)$.

13.2.4 Dari larik LCP ke banyaknya substring berbeda

Telusuri sufiks-sufiks terurut ke bawah. Sufiks $sa[k]$ memiliki $n - sa[k]$ prefiks, satu untuk setiap panjang. $lcp[k]$ prefiks pertamanya juga merupakan prefiks sufiks sebelumnya, jadi sudah terhitung. Prefiks yang lebih panjang adalah baru: seandainya salah satunya pernah muncul lebih awal dalam urutan terurut, sebagai prefiks suatu sufiks $s[sa[k']..]$ dengan $k' < k$, maka setiap sufiks di antara keduanya dalam urutan terurut, termasuk $sa[k - 1]$, akan berbagi prefiks itu, bertentangan dengan definisi $lcp[k]$. Jadi sufiks $sa[k]$ menyumbang tepat $n - sa[k] - lcp[k]$ substring baru, dan menjumlahkannya atas k memberikan

$$\begin{aligned} \# \text{ substring berbeda} &= \sum_{k=0}^{n-1} (n - sa[k] - lcp[k]) \\ &= \frac{n(n+1)}{2} - \sum_{k=1}^{n-1} lcp[k] \end{aligned} \quad (13.4)$$

karena nilai-nilai $n - sa[k]$ adalah bilangan $1, 2, \dots, n$ dalam suatu urutan. Untuk ABABA, **Persamaan** (13.4) memberikan $15 - (1 + 3 + 0 + 2) = 9$, sesuai dengan Subbab 13.1; untuk CCCCC memberikan $15 - (1 + 2 + 3 + 4) = 5$.

13.2.5 Kasai dengan tangan pada ABABA

Ada baiknya menyaksikan variabel h bergerak sekali. Larik sufiksnya $sa = [4,2,0,3,1]$, sehingga larik kebalikannya $pos[0] = 2, pos[1] = 4, pos[2] = 1, pos[3] = 3, pos[4] = 0$.

- $i = 0$: sufiks ABABA berada di tempat 2; pendahulunya sufiks 2, ABA. Mulai dari $h = 0$ kita mencocokkan A, B, A dan berhenti di ujung ABA: $h = 3$, jadi $lcp[2] = 3$. Lalu h turun menjadi 2.
- $i = 1$: sufiks BABA di tempat 4; pendahulunya sufiks 3, BA. Kita *mulai* dari $h = 2$, yang sudah merupakan ujung BA, jadi tidak ada perbandingan yang berhasil: $lcp[4] = 2$. h turun menjadi 1.
- $i = 2$: sufiks ABA di tempat 1; pendahulunya sufiks 4, A. Kita mulai dari $h = 1$, ujung A: $lcp[1] = 1$. h turun menjadi 0.
- $i = 3$: sufiks BA di tempat 3; pendahulunya sufiks 0, ABABA. Karakter pertamanya berbeda: $lcp[3] = 0$.
- $i = 4$: sufiks A adalah yang terkecil; ia tidak memiliki pendahulu dan h diatur ulang ke 0.

Hanya empat perbandingan karakter yang berhasil secara keseluruhan, dan pada dua langkah lema itu memungkinkan kita melewati perbandingan sama sekali. Larik LCP-nya $[0,1,3,0,2]$, seperti pada **Gambar 13.1**.

13.3 Soal Pemanasan

13.3.1 Larik sufiks yang lambat, sebagai tabel

Sebelum menulis versi cepat, tulislah versi yang jelas dan perhatikan keluarannya. **Kode 13.1** mengurutkan sufiks dengan `string::compare`, menghitung setiap LCP dengan perbandingan langsung, dan mencetak tabel **Gambar 13.1** beserta banyaknya prefiks baru per baris.

Kode 13.1 `naive_sa.cpp`: larik sufiks dan larik LCP yang dihitung dengan cara lambat, dicetak sebagai tabel.

```
// Warm-up: build the suffix array and the LCP column the slow, obvious way
// (sort whole suffixes as strings) and print them as a table.
#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>
using namespace std;

int main() {
    char buf[1001];
    if (scanf("%1000s", buf) != 1) return 0;
    string s = buf;
    int n = (int)s.size();
    vector<int> sa(n);
    for (int i = 0; i < n; i++) sa[i] = i;
    sort(sa.begin(), sa.end(), [&](int a, int b) {
        return s.compare(a, string::npos, s, b, string::npos) < 0;
    });
    printf("k sa lcp new suffix\n");
    long long distinct = 0;
    for (int k = 0; k < n; k++) {
        int h = 0;
        if (k > 0)
            while (sa[k] + h < n && sa[k - 1] + h < n &&
                    s[sa[k] + h] == s[sa[k - 1] + h]) h++;
        int fresh = n - sa[k] - h; // prefixes not seen before
        distinct += fresh;
        printf("%2d %2d %3d %3d %s\n", k, sa[k], h, fresh, s.c_str() + sa[k]);
    }
    printf("distinct substrings: %lld\n", distinct);
    return 0;
}
```

Untuk masukan ABABA:

```
k sa lcp new suffix
0 4 0 1 A
1 2 1 2 ABA
2 0 3 2 ABABA
3 3 0 2 BA
4 1 2 2 BABA
distinct substrings: 9
```

Program ini $O(n^2 \log n)$, yang cukup untuk beberapa ribu karakter dan menjadi acuan sempurna untuk pengujian.

13.3.2 Pencarian pola dengan larik sufiks

Karena sufiks-sufiksnya teratur, semua sufiks yang diawali pola p duduk bersebelahan dalam larik sufiks; mereka membentuk satu blok, sebagaimana semua kata yang diawali “ban” berkumpul dalam kamus. Dua pencarian biner menemukan ujung-ujung blok itu, dan ukurannya adalah banyaknya kemunculan. **Kode 13.2** melakukan persis hal itu, dengan hanya membandingkan $|p|$ karakter pertama setiap sufiks.

Kode 13.2 `pattern_count.cpp`: menghitung kemunculan pola dengan pencarian biner pada larik sufiks.

```
// Warm-up: count the occurrences of a pattern p in a text s by binary search
// on the suffix array (here built naively): the suffixes that start with p
// form one contiguous block of the sorted order.
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main() {
    string s, p;
    if (!(cin >> s >> p)) return 0;
    int n = (int)s.size();
    vector<int> sa(n);
    for (int i = 0; i < n; i++) sa[i] = i;
    sort(sa.begin(), sa.end(), [&](int a, int b) {
        return s.compare(a, string::npos, s, b, string::npos) < 0;
    });
    // compare only the first |p| characters of each suffix with p
    auto below = [&](int start, const string& q) {
        return s.compare(start, q.size(), q) < 0;
    };
    auto above = [&](const string& q, int start) {
        return s.compare(start, q.size(), q) > 0;
    };
    auto lo = lower_bound(sa.begin(), sa.end(), p, below);
    auto hi = upper_bound(sa.begin(), sa.end(), p, above);
    cout << p << " occurs " << (hi - lo) << " time(s) at:";
    vector<int> at(lo, hi);
    sort(at.begin(), at.end());
    for (int i : at) cout << ' ' << i;
    cout << '\n';
    return 0;
}
```

Untuk teks ABRACADABRA dan pola ABRA:

```
ABRA occurs 2 time(s) at: 0 7
```

Setiap pencarian berbiaya $O(|p|\log n)$ setelah larik sufiks dibangun, berapa pun banyaknya kemunculan pola. Begitu teks terindeks, Anda dapat menjawab ribuan kueri pola dengan cepat.

13.4 Studi Kasus: DISUBSTR

13.4.1 Soalnya dengan kata-kata kita sendiri

Soal SPOJ DISUBSTR [4] memberikan $T \leq 20$ kasus uji. Masing-masing berupa string berisi paling banyak 1000 karakter tanpa spasi. Untuk setiap string, cetak banyaknya substring tak kosong yang berbeda.

13.4.2 Contoh

```
2
CCCCC
ABABA
```

```
5
9
```

Ini adalah dua string yang sudah kita analisis: CCCCC memiliki lima substring berbeda (C, CC, CCC, CCCC, CCCCC) dan ABABA memiliki sembilan.

Sebelum merancang, ada baiknya melihat betapa kecilnya bilangan-bilangan ini. Dengan $T = 20$ string sepanjang 1000 karakter, bahkan metode $O(n^2)$ hanya melakukan sekitar $2 \cdot 10^7$ langkah, jadi larik sufiks tidak *diwajibkan* demi kecepatan di sini. Larik sufiks tetap pilihan yang tepat, karena kode yang sama menyelesaikan versi soal yang lebih sulit, dan karena DISUBSTR adalah tempat yang wajar untuk mempelajarinya pada masukan yang cukup kecil untuk diperiksa dengan tangan dan dengan *brute force*.

Uji tambahan buatan kita sendiri sama mendidiknya. String A memiliki satu substring berbeda. String AB memiliki tiga: A, B, dan AB. String AAB memiliki lima: A, B, AA, AB, dan AAB; sufiks terurutnya adalah AAB, AB, B dengan nilai LCP 1 dan 0, dan **Persamaan** (13.4) memberikan $6 - 1 = 5$. Mengerjakan beberapa string semacam itu dengan tangan sebelum menulis kode adalah cara tercepat memastikan bahwa rumusnya, bukan hanya programnya, benar-benar dipahami.

13.4.3 Rancangan

Rancangannya adalah **Persamaan** (13.4): bangun larik sufiks dengan penggandaan (**Pseudokode 13.1**), hitung larik LCP dengan Kasai, lalu kurangkan jumlah LCP dari $n(n + 1)/2$. Dua keputusan kecil penting.

- **Kode karakter.** Peringkat awalnya `(unsigned char)s[i]`, sehingga setiap nilai byte menjadi peringkat yang sah dan tidak negatif, dan penanda di luar jangkauan -1 lebih kecil daripada semuanya.
- **Total 64 bit.** Untuk $n = 1000$ total $n(n + 1)/2 = 500\,500$ muat dalam `int`, tetapi kode yang sama yang diterapkan pada $n = 10^5$ (Soal Latihan 5) akan mencapai $5 \cdot 10^9$. Kita memakai `long long` sejak awal.

13.4.4 Solusi lengkap

Kode 13.3 adalah program utuhnya. Membangun larik sufiks adalah paruh pertama inti; gelung Kasai, di antara kedua penanda, adalah paruh kedua.

Kode 13.3 DISUBSTR.cpp: substring berbeda dengan larik sufiks (penggandaan prefiks) dan LCP Kasai.

```
// DISUBSTR: count the distinct substrings of a string.
// Suffix array by prefix doubling, LCP array by Kasai, then
// answer = n(n+1)/2 - (sum of the LCPs of neighboring suffixes).
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// Suffix array: sa[k] is the start of the k-th smallest suffix of s.
vector<int> suffixArray(const string& s) {
    int n = (int)s.size();
    vector<int> sa(n), rk(n), tmp(n);
    for (int i = 0; i < n; i++) sa[i] = i, rk[i] = (unsigned char)s[i];
    for (int gap = 1; gap < n; gap *= 2) {
        // sort by the pair (rank of first gap chars, rank of the next gap)
        auto key2 = [&](int i) { return i + gap < n ? rk[i + gap] : -1; };
        auto less = [&](int a, int b) {
            if (rk[a] != rk[b]) return rk[a] < rk[b];
            return key2(a) < key2(b);
        };
        sort(sa.begin(), sa.end(), less);
        tmp[sa[0]] = 0; // equal pairs share a rank
        for (int k = 1; k < n; k++)
            tmp[sa[k]] = tmp[sa[k - 1]] + (less(sa[k - 1], sa[k]) ? 1 : 0);
        rk = tmp;
        if (rk[sa[n - 1]] == n - 1) break; // all ranks distinct: done
    }
    return sa;
}

// lcp[k] = length of the longest common prefix of suffixes sa[k-1] and sa[k].
vector<int> lcpArray(const string& s, const vector<int>& sa) {
    int n = (int)s.size();
    vector<int> pos(n), lcp(n, 0);
    for (int k = 0; k < n; k++) pos[sa[k]] = k;
    // <P50>
    // Kasai: visit suffixes in text order i = 0, 1, ...; compare suffix i with
    // its left neighbor in sa, reusing h, which drops by at most 1 per step.
    // >>> YOUR TURN (Public Edition): write this part yourself, following the hint above.
    <<<
    // </P50>
    return lcp;
}
```

Kode 13.3 (Lanjutan) DISUBSTR.cpp: substring berbeda dengan larik sufiks (penggandaan prefiks) dan LCP Kasai.

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int t;
    if (!(cin >> t)) return 0;
    while (t--) {
        string s;
        cin >> s;
        long long n = (long long)s.size();
        vector<int> sa = suffixArray(s);
        vector<int> lcp = lcpArray(s, sa);
        long long total = n * (n + 1) / 2;
        for (int v : lcp) total -= v;
        cout << total << '\n';
    }
    return 0;
}
```

13.4.5 Baris demi baris

suffixArray. Vektor `sa`, `rk`, dan `tmp` menyimpan urutan saat ini, peringkat saat ini, dan peringkat berikutnya. Lambda `key2` mengembalikan komponen kedua pasangan **Persamaan** (13.3), dengan -1 di luar ujung. Pembandingan `less` membandingkan pasangan secara leksikografis. Setelah `sort`, gelung atas `k` memberikan peringkat baru: sufiks pertama mendapat 0, dan setiap sufiks berikutnya mendapat peringkat sebelumnya ditambah satu tepat ketika pasangannya tegas lebih besar, yang diuji oleh `less(sa[k - 1], sa[k])`. Jalan keluar dini memeriksa apakah sufiks terakhir berperingkat $n - 1$; karena peringkat naik paling banyak satu per langkah, hal itu hanya terjadi bila semua n peringkat berbeda, dan putaran berikutnya tidak dapat mengubah urutan.

lcpArray. `pos` adalah kebalikan `sa`: `pos[i]` adalah tempat sufiks i dalam urutan terurut. Gelungnya mengunjungi $i = 0, 1, \dots$ dalam urutan teks. Sufiks terkecil tidak memiliki pendahulu, jadi `h` diatur ulang dan gelung berlanjut. Selain itu `j` adalah pendahulunya, gelung `while` memperpanjang prefiks bersama mulai dari `h` saat ini, hasilnya disimpan di `lcp[pos[i]]`, dan `h` turun satu untuk langkah berikutnya, sebagaimana diizinkan lema Kasai.

main. Untuk setiap string, fungsi ini menghitung larik-lariknya dan mencetak $n(n + 1)/2 - \sum lcp$. Perhatikan bahwa `lcp[0]` bernilai 0, jadi menjumlahkan seluruh vektor sudah benar.

JEBAKAN. (1) Melupakan -1 untuk posisi di luar ujung dan membaca `rk[i + gap]` di luar batas. (2) Memberikan peringkat baru dari posisi terurut `k` alih-alih membandingkan pasangan tetangga: pasangan yang sama harus berbagi peringkat, atau putaran berikutnya mengurutkan menurut urutan yang tidak ada. (3) Mengatur ulang `h` ke 0 pada setiap i dalam Kasai: tetap benar, tetapi $O(n^2)$. (4) Melewati sufiks terkecil tanpa mengatur `h = 0`: lema Kasai tidak memberikan batas bawah setelah sufiks yang tidak memiliki pendahulu, dan sisa `h` akan melewati perbandingan dan menyimpan LCP yang terlalu besar. (5) Membaca string dengan `scanf("%s")` ke dalam penyangga yang kurang satu karakter untuk nol penutupnya.

13.5 Mengujinya: Larik Sufiks Melawan Himpunan String

Acuan untuk DISUBSTR hampir merupakan definisinya: masukkan setiap substring ke `set<string>` dan cetak ukurannya (**Kode 13.4**). Program ini $O(n^3 \log n)$ dan dengan senang hati menangani string sepanjang 40 karakter.

Kode 13.4 `brute.cpp`: acuan berupa himpunan substring.

```
// DISUBSTR reference: put every substring into a set and count the set.
#include <iostream>
#include <set>
#include <string>
using namespace std;

int main() {
    int t;
    if (!(cin >> t)) return 0;
    while (t--) {
        string s;
        cin >> s;
        set<string> seen;
        for (size_t i = 0; i < s.size(); i++)
            for (size_t len = 1; i + len <= s.size(); len++)
                seen.insert(s.substr(i, len));
        cout << seen.size() << '\n';
    }
    return 0;
}
```

Pembangkitnya (**Kode 13.5**) mencampur dua alfabet. String yang hanya berisi A dan B sangat berulang, sehingga menguji LCP panjang dan banyak putaran penggandaan; string atas alfabet campuran yang memuat huruf kecil dan angka memeriksa bahwa kode karakter ditangani dengan benar.

Kode 13.5 `gen.py`: string acak atas alfabet kecil dan alfabet campuran.

```
# Random DISUBSTR input: short strings over a small alphabet (many repeats).
import random
import sys

rnd = random.Random(int(sys.argv[1]))
t = rnd.randint(1, 5)
print(t)
for _ in range(t):
    alpha = "AB" if rnd.random() < 0.5 else "ABCZaz09"
    n = rnd.randint(1, 40)
    print("".join(rnd.choice(alpha) for _ in range(n)))
```

Dengan penggerak `stress.sh` yang biasa:

```
All 500 tests passed.
```

`Sanitizer (-fsanitize=address,undefined)` juga tidak melaporkan apa pun pada masukan ini, yang menegaskan bahwa penjaga -1 di `key2` menjaga setiap indeks dalam jangkauan.

13.6 Taktik pada Hari Lomba

Tabel 13.1 memperlihatkan bahwa DISUBSTR pada batasnya bukan tantangan waktu; tantangannya adalah kebenaran.

Tabel 13.1 Waktu jalan `DISUBSTR.cpp` pada ukuran maksimum.

Masukan	Waktu
20 string acak berisi 1000 huruf dari A–J	0,01 s
20 string berisi 1000 salinan A	0,01 s

- **Simpan larik sufiks yang teruji di kepala, bukan hanya di catatan.** Di OSN Anda mengetik semuanya sendiri. Berlatihlah mengetik **Pseudokode 13.1** dan Kasai sampai selesai dalam kurang dari sepuluh menit, lalu periksa terhadap `naive_sa.cpp`.
- **Kenali kamus kegunaannya.** Dengan (sa, lcp) di tangan: substring berulang terpanjang adalah $maxlcp$; banyaknya substring berbeda adalah **Persamaan (13.4)**; pencarian pola adalah pencarian biner; substring berbeda terkecil ke- k ditemukan dengan menelusuri sufiks terurut dan membelanjakan prefiks baru $n - sa[k] - lcp[k]$ dari masing-masing; substring bersama terpanjang dari dua string diperoleh dari larik sufiks $s \# t$ [5], [6].
- **Pilih alat paling sederhana yang cocok.** Untuk $n \leq 1000$ bahkan `naive_sa.cpp` akan lolos DISUBSTR: $O(n^2 \log n)$ dengan perbandingan cepat. *Hash* semua substring ke dalam himpunan juga akan lolos. Larik sufiks baru terbayar ketika n mencapai 10^5 , seperti pada soal saudaranya SUBST1 (Soal Latihan 5).

13.6.1 Dua string sekaligus

Banyak soal string melibatkan dua teks: temukan potongan terpanjang yang dimuat kedua esai, atau hitung potongan yang muncul di satu teks tetapi tidak di teks lain. Trik bakunya adalah merekatkan keduanya menjadi satu string $u = s \# t$, dengan $\#$ pemisah yang tidak muncul di kedua teks dan terurut sebelum setiap huruf. Bangun larik sufiks dan larik LCP dari u . Setiap

sufiks kini milik s (dimulai sebelum $\#$) atau milik t (dimulai sesudahnya). Substring bersama terpanjang dari s dan t adalah $lcp[k]$ terbesar di antara tetangga $sa[k-1]$ dan $sa[k]$ yang berasal dari teks *berbeda*. Pemisah menjamin tidak ada prefiks bersama yang melintasi batas kedua teks, sehingga setiap LCP antara sufiks- s dan sufiks- t adalah substring bersama yang sesungguhnya.

Sebagai contoh, dengan $s = ABAB$ dan $t = BABA$ string rekatannya adalah $ABAB\#BABA$. Di antara sufiks terurutnya, ABA (dari t) dan $ABAB\#BABA$ (dari s) bertetangga dengan LCP 3, dan $BAB\#BABA$ (dari s) serta $BABA$ (dari t) juga bertetangga dengan LCP 3, sehingga substring bersama terpanjang berpanjang 3: ABA atau BAB . Seluruh perhitungannya hanya satu larik sufiks sepanjang $|s| + |t| + 1$.

13.6.2 Memilih alat yang tepat

Peserta yang mengenal beberapa alat string perlu tahu kapan masing-masing terbayar. *Hashing polinomial* membandingkan dua substring dalam $O(1)$ setelah persiapan $O(n)$ dan cepat ditulis; alat ini ideal untuk “apakah substring ini sama dengan substring itu?” dan untuk pencarian biner atas panjang, tetapi jawabannya hanya benar dengan peluang tinggi, dan menghitung substring berbeda dengan *hash* memerlukan himpunan berisi $O(n^2)$ nilai *hash*, cukup untuk $n = 1000$, mustahil untuk $n = 10^5$. *Larik sufiks dengan LCP* deterministik, hemat memori, dan menjawab segala hal yang bergantung pada urutan terurut sufiks: substring berbeda, substring ke- k , pengulangan terpanjang, substring bersama dua teks. *Automaton sufiks* dan *pohon sufiks* dapat berbuat lebih banyak lagi secara *online*, tetapi lebih panjang dan lebih sulit ditulis di bawah tekanan [6]. Untuk olimpiade, larik sufiks yang terlatih ditambah *hashing* mencakup hampir setiap tugas string yang akan Anda temui.

13.6.3 Kesalahan yang sering terjadi

Tiga kesalahan khas muncul pada implementasi larik sufiks pertama. Mencampuradukkan sa dengan kebalikannya: $sa[k]$ menjawab “sufiks mana yang ke- k ?”, sedangkan $pos[i]$ menjawab “di mana sufiks i ?”, dan Kasai memerlukan keduanya. Membandingkan *karakter* alih-alih *peringkat* pada putaran penggantian berikutnya, yang diam-diam hanya mengurutkan menurut karakter pertama. Dan melupakan bahwa masukan dapat memuat karakter selain huruf kapital: memakai $s[i] - 'A'$ sebagai peringkat rusak pada huruf kecil atau angka, sedangkan kode karakter tak bertanda selalu bekerja.

13.7 Ringkasan Bab

- Setiap substring adalah prefiks dari suatu sufiks; mengurutkan sufiks ([Persamaan \(13.1\)](#)) menata semua substring.
- Penggantian prefiks mengurutkan sufiks menurut pasangan peringkat ([Persamaan \(13.3\)](#)), menggandakan panjang yang dibandingkan setiap putaran: $O(n \log^2 n)$ dengan pengurutan perbandingan.
- Lema Kasai memungkinkan LCP antartetangga ([Persamaan \(13.2\)](#)) dihitung dalam $O(n)$ dengan mengunjungi sufiks dalam urutan teks.
- Substring berbeda $= n(n+1)/2 - \sum lcp$, [Persamaan \(13.4\)](#), karena setiap sufiks hanya menyumbang prefiks yang lebih panjang daripada LCP-nya dengan pendahulunya.
- Himpunan semua substring adalah acuan sempurna untuk uji stres pada string pendek.

13.8 Soal Latihan

1. Bangun *sa* dan *lcp* untuk BANANA dengan tangan dan tentukan banyaknya substring berbeda.
2. Dengan larik dari Soal Latihan 1, tentukan substring terpanjang BANANA yang muncul sekurang-kurangnya dua kali.
3. Berapa banyak substring berbeda pada string berisi n karakter yang sama? Dan pada string berisi n karakter yang semuanya berbeda? Periksa keduanya dengan [Persamaan \(13.4\)](#).
4. Jelaskan cara menemukan substring berbeda terkecil ke- k dalam urutan leksikografis dengan *sa* dan *lcp*.
5. SPOJ SUBST1 [7] adalah DISUBSTR dengan string sampai 50 000 karakter. Bagian [DISUBSTR.cpp](#) mana yang harus berubah, bila ada?
6. Mengapa **Pseudokode 13.1** memperlakukan posisi di luar ujung string sebagai peringkat -1 , bukan peringkat 0?
7. Dengan larik sufiks *s*, bagaimana Anda menemukan panjang substring terpanjang yang muncul sekurang-kurangnya k kali?
8. Sebuah string *s* berisi 1000 karakter dan semua nilai LCP-nya 0. Apa yang dapat Anda katakan tentang *s*?

13.9 Penyelesaian Soal Latihan

1. Sufiks BANANA (posisi 0–5) dalam urutan terurut adalah A (5), ANA (3), ANANA (1), BANANA (0), NA (4), NANA (2). Jadi *sa* = [5,3,1,0,4,2] dan *lcp* = [0,1,3,0,0,2]. Hitungannya $21 - 6 = 15$.
2. Substring berulang terpanjang adalah yang ber-LCP terbesar: *lcp*[2] = 3, antara ANA dan ANANA, jadi jawabannya ANA, yang muncul di posisi 1 dan 3 (kemunculannya bertumpang tindih).
3. Untuk n karakter yang sama, sufiks terurutnya berpanjang $1, 2, \dots, n$ dan *lcp*[k] = k , jadi hitungannya $n(n+1)/2 - n(n-1)/2 = n$. Untuk n karakter yang semuanya berbeda, semua LCP bernilai 0 dan hitungannya $n(n+1)/2$.
4. Telusuri $k' = 0, 1, \dots$ pada sufiks terurut. Sufiks *sa*[k'] menyumbang substring baru berpanjang *lcp*[k'] + 1 sampai $n - sa[k']$, dalam urutan menaik, dan semuanya datang setelah setiap substring yang disumbangkan sebelumnya. Kurangkan $n - sa[k'] - lcp[k']$ dari k sampai k muat di dalam sumbangan sufiks saat ini; jawabannya adalah prefiks sepanjang *lcp*[k'] + k dari sufiks *sa*[k'].
5. Algoritmanya tetap. Dengan $n = 50\,000$ penggandaan berbiaya sekitar $50\,000 \cdot 16 \cdot 16 \approx 1.3 \cdot 10^7$ perbandingan per string, cukup untuk beberapa string. Jawabannya dapat mencapai $1.25 \cdot 10^9$, yang tetap muat dalam [long long](#) (dan kodenya sudah memakainya). Bila banyak string panjang muncul, ganti [sort](#) dengan *radix sort* untuk mendapatkan $O(n \log n)$.
6. Sufiks yang berakhir sebelum panjang yang dibandingkan adalah prefiks sejati dari $2g$ karakter pertama sufiks lain dan harus terurut *sebelumnya*. Peringkat 0 sudah dipakai oleh karakter sungguhan dan akan membuat “A” dan “A diikuti karakter terkecil” tampak sama; -1 lebih kecil daripada setiap peringkat sungguhan.
7. Suatu substring muncul sekurang-kurangnya k kali tepat ketika ia merupakan prefiks bersama dari k sufiks yang berurutan dalam larik sufiks. Prefiks bersama terpanjang dari k sufiks berurutan *sa*[i], ..., *sa*[$i + k - 1$] adalah minimum dari *lcp*[$i + 1$], ..., *lcp*[$i + k - 1$]. Jadi jawabannya adalah maksimum, atas semua jendela berisi $k - 1$ nilai LCP berurutan, dari minimum jendela itu. Minimum jendela geser dengan *deque* menghitungnya dalam $O(n)$ setelah

larik dibangun; untuk $k = 2$ ini menjadi maxlcp , substring berulang terpanjang pada Subbab 13.6.

8. Sufiks bertetangga dalam urutan terurut tidak berbagi karakter pertama, jadi tidak ada karakter yang berulang: seandainya suatu karakter muncul dua kali, kedua sufiks yang diawali karakter itu akan bertetangga atau hanya dipisahkan oleh sufiks yang diawali karakter yang sama, sehingga LCP-nya positif. Jadi kesemua 1000 karakter berbeda, yang hanya mungkin dengan alfabet berisi sekurang-kurangnya 1000 simbol, dan jawaban DISUBSTR-nya $n(n + 1)/2 = 500\,500$.

Daftar Pustaka

- [1] U. Manber and G. Myers, "Suffix arrays: A new method for on-line string searches," *SIAM Journal on Computing*, vol. 22, no. 5, pp. 935–948, 1993.
- [2] J. Kärkkäinen, P. Sanders, and S. Burkhardt, "Linear work suffix array construction," *Journal of the ACM*, vol. 53, no. 6, pp. 918–936, 2006.
- [3] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park, "Linear-time longest-common-prefix computation in suffix arrays and its applications," in *Proc. 12th Annual Symposium on Combinatorial Pattern Matching (CPM)*, LNCS 2089, Springer, 2001, pp. 181–192.
- [4] Sphere Online Judge, "DISUBSTR - Distinct Substrings." [Online]. Available: <https://www.spoj.com/problems/DISUBSTR/>
- [5] S. Halim, F. Halim, and S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 2. Lulu Press, 2020.
- [6] cp-algorithms.com, "Suffix Array." [Online]. Available: <https://cp-algorithms.com/string/suffix-array.html>
- [7] Sphere Online Judge, "SUBST1 - New Distinct Substrings." [Online]. Available: <https://www.spoj.com/problems/SUBST1/>

Bab 14

Beda Hingga, Interpolasi, dan Aritmetika Modular — BATTLECRY

Sebagian soal penghitungan tampak menakutkan: hitung posisi empat bidak catur pada papan $N \times N$ yang memenuhi aturan halus, untuk N sampai seratus ribu. Tidak ada pencarian yang dapat mengunjungi posisi-posisi itu, dan rumus yang diturunkan dengan tangan akan memakan waktu berjam-jam dan mengundang kesalahan. Namun banyak soal semacam itu menyimpan kesederhanaan tersembunyi: begitu N cukup besar, jawabannya adalah *polinomial* dalam N . Bab ini mengajarkan alur kerja mekanis untuk situasi itu. Tulis *brute force* yang jujur untuk N kecil; deteksi derajatnya dengan *tabel beda hingga*; pulihkan polinomialnya dengan *interpolasi Lagrange*; lalu evaluasi untuk N raksasa dengan *aturan Horner* dan *invers modular*. Anda akan menerapkan alur ini pada soal SPOJ BATTLECRY, tempat dua benteng dan seorang raja men-*skakmat* raja yang sendirian, dan berakhir dengan solusi $O(1)$ yang kebenarannya telah Anda periksa sendiri terhadap *brute force*.

14.1 Di Mana Ini Berguna: Menghitung Tanpa Mendaftar

Pak Hadi memasang lantai ubin persegi untuk masjid dan rumah. Seorang pelanggan bertanya berapa banyak cara menempatkan empat ubin hias pada lantai $N \times N$ sehingga suatu aturan pola terpenuhi. Untuk $N = 4$ ia dapat menggambar semuanya di kertas berpetak; untuk $N = 40$ tidak mungkin. Tetapi ia memperhatikan bahwa ketika ia menghitung untuk $N = 4, 5, 6, 7$, bilangan-bilangannya tumbuh dengan mulus, seperti luas atau volume sesuatu. Jika hitungannya memang polinomial, segelintir kasus kecil sudah menentukannya sepenuhnya, dan setiap N tinggal satu rumus lagi.

Gejala yang sama muncul setiap kali benda ditempatkan pada kisi dengan *banyak bidak tetap* dan aturan *lokal*: menghitung ratu atau raja yang tidak saling menyerang dalam jumlah tetap, menghitung persegi panjang atau segitiga jenis tertentu dalam kisi $N \times N$, menghitung lintasan berpanjang tetap dalam struktur yang tumbuh. Dalam soal olimpiade hal ini biasanya tampil sebagai “untuk banyak nilai N , cetak jawabannya modulo suatu bilangan prima”. Alur kerja bab ini mengubah soal semacam itu menjadi satu sore *brute force* yang cermat disusul beberapa baris aritmetika.

Ada baiknya jujur tentang apa sebenarnya alur kerja ini. Ia bukan bukti bahwa rumusnya benar; ia adalah percobaan yang disusul pemeriksaan cermat. Dalam lomba itulah persis yang kita perlukan: juri hanya menguji apakah jawabannya benar, dan gabungan *brute force* yang tepercaya, baris beda yang jelas konstan, dan titik validasi tambahan membuat rumus yang salah menjadi sangat tidak mungkin. Setelah lomba, bila penasaran, Anda dapat mencoba menurunkan rumusnya dengan tangan dari bentuk-bentuk pada Subbab 14.2.4, dan itu latihan menghitung cermat yang memuaskan.

WAWASAN KUNCI: *Brute force* kecil bukan pekerjaan sia-sia. Ia adalah *data*. Bila jawabannya polinomial dalam N , *brute force* untuk N kecil memuat seluruh solusinya, dan beda hingga serta interpolasi mengekstraknya.

14.2 Tekniknya dari Nol

14.2.1 Beda hingga: stetoskop polinomial

Diberikan barisan nilai $f(3), f(4), f(5), \dots$, beda pertama-nya adalah barisan

$$\Delta f(N) = f(N+1) - f(N) \quad (14.1)$$

dan beda ke- d adalah $\Delta^d f = \Delta(\Delta^{d-1} f)$. Sifat kuncinya adalah:

TEOREMA. Jika f polinomial berderajat d dengan koefisien utama a , maka Δf polinomial berderajat $d-1$, beda ke- d adalah konstanta $a \cdot d!$, dan semua beda yang lebih tinggi bernilai 0. Sebaliknya, jika beda ke- d dari nilai $f(N_0), f(N_0+1), \dots$ konstan, maka nilai-nilai itu sesuai dengan suatu polinomial berderajat d [1].

Mengapa. Untuk $f(N) = aN^d + \dots$, beda $f(N+1) - f(N)$ memuat $a((N+1)^d - N^d) = adN^{d-1} + \dots$: derajatnya turun satu dan koefisien utamanya dikalikan derajat lama. Menerapkannya d kali menyisakan konstanta $a \cdot d \cdot (d-1) \cdots 1 = a \cdot d!$.

Jadi prosedur untuk barisan misterius adalah: tulis nilainya dalam satu baris, tulis bedanya di baris bawahnya, dan ulangi. Jika suatu baris menjadi konstan, indeksnya adalah derajatnya.

Gambar 14.1 memperlihatkan tabel untuk jawaban soal bab ini, dihitung oleh *brute force* pada Subbab 14.5: baris kelima adalah konstanta 480.

	N=3	N=4	N=5	N=6	N=7	N=8	N=9	N=10
baris 0	232	1432	5188	14536	34464	72392	138652	246968
baris 1		1200	3756	9348	19928	37928	66260	108316
baris 2			2556	5592	10580	18000	28332	42056
baris 3				3036	4988	7420	10332	13724
baris 4					1952	2432	2912	3392
baris 5						480	480	480

← baris konstan: derajat 5

Gambar 14.1 Tabel beda hingga hitungan BATTLECRY untuk $N = 3$ sampai 10. Setiap baris memuat beda dari baris di atasnya; baris 5 konstan, sehingga hitungannya mengikuti polinomial berderajat 5.

Dari teorema itu, $480 = a \cdot 5!$, sehingga koefisien utamanya $a = 480/120 = 4$: untuk papan besar hitungannya tumbuh seperti $4N^5$.

14.2.2 Interpolasi Lagrange: dari titik ke polinomial

Polinomial berderajat d ditentukan oleh nilainya pada $d+1$ titik berbeda. Rumus Lagrange menuliskannya secara langsung. Diberikan titik $(x_0, y_0), \dots, (x_d, y_d)$ dengan x_i yang berbeda,

$$P(x) = \sum_{i=0}^d y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \quad (14.2)$$

Setiap hasil kali pada **Persamaan** (14.2) adalah polinomial berderajat d yang bernilai 1 di x_i dan 0 di setiap x_j lainnya, sehingga $P(x_i) = y_i$ untuk setiap i . Ketunggalannya menyusul karena selisih

dua polinomial semacam itu berderajat paling tinggi d dan memiliki $d + 1$ akar, yang memaksanya bernilai nol [2].

Kita dapat memakai [Persamaan \(14.2\)](#) dengan dua cara. Kita dapat mengevaluasinya langsung di x baru, tanpa memerlukan koefisien sama sekali; atau kita dapat menjabarkannya sekali, di kertas atau dengan skrip kecil yang memakai pecahan eksak, untuk memperoleh koefisiennya. Untuk BATTLECRY, interpolasi melalui enam titik $N = 3, \dots, 8$ yang dijabarkan memberikan

$$f(N) = 4N^5 - \frac{56}{3}N^4 + 22N^3 + \frac{554}{3}N^2 - 764N + 808, \quad N \geq 3 \quad (14.3)$$

Koefisien utama 4 sesuai dengan tabel beda. Dua koefisien berupa pecahan, jadi kita kalikan dengan 3 agar bekerja dengan bilangan bulat:

$$3f(N) = 12N^5 - 56N^4 + 66N^3 + 554N^2 - 2292N + 2424 \quad (14.4)$$

lalu periksa [Persamaan \(14.4\)](#) terhadap nilai *brute force* untuk $N = 9$ dan $N = 10$, yang *tidak* dipakai dalam interpolasi: keduanya cocok (138 652 dan 246 968). Pemeriksaan itu penting. Tabel beda adalah bukti yang kuat tetapi bukan pembuktian, dan titik tambahan menjaga dari pola yang berubah belakangan.

14.2.3 Aturan Horner dan aritmetika modular

Jawabannya harus dicetak modulo bilangan prima $M = 7\,340\,033$. Dua alat diperlukan.

Aturan Horner mengevaluasi polinomial dengan satu perkalian dan satu penjumlahan per koefisien, dengan cara bersarang:

$$\begin{aligned} &12N^5 - 56N^4 + 66N^3 + 554N^2 - 2292N + 2424 \\ &= (((((12N - 56)N + 66)N + 554)N - 2292)N + 2424 \end{aligned} \quad (14.5)$$

Mengambil sisa bagi setelah setiap langkah menjaga semua nilai antara di bawah $M^2 \approx 5.4 \cdot 10^{13}$, jauh di dalam jangkauan 64 bit.

Pembagian modulo bilangan prima adalah perkalian dengan *invers*. Kita tidak dapat membagi sisa bagi $3f(N)$ dengan 3 memakai `/`, karena sisa bagi itu belum tentu kelipatan 3. Sebagai gantinya kita mengalikan dengan bilangan 3^{-1} yang memenuhi $3 \cdot 3^{-1} \equiv 1 \pmod{M}$. Teorema kecil Fermat menyatakan $a^{M-1} \equiv 1 \pmod{M}$ untuk M prima dan a yang tidak habis dibagi M , sehingga

$$a^{-1} \equiv a^{M-2} \pmod{M} \quad (14.6)$$

dan perpangkatan cepat menghitung [Persamaan \(14.6\)](#) dalam $O(\log M)$ perkalian [3]. Untuk $a = 3$ hasilnya $3^{-1} = 2\,446\,678$; memang $3 \cdot 2\,446\,678 = 7\,340\,034 = M + 1$.

14.2.4 Mengapa penempatan menghasilkan polinomial

Mengapa hitungan catur harus berupa polinomial? Golongkan setiap posisi menurut *bentuk*-nya: bidak mana yang berbagi baris atau kolom, selisih kecil yang tepat antara bidak yang berdekatan, dan bidak mana yang menyentuh tepi papan. Untuk bentuk tetap, banyaknya posisi adalah hasil kali faktor seperti $(N - c)$, satu untuk setiap koordinat bebas, sehingga berupa polinomial dalam N . Aturan “raja hitam di-*skakmat*” hanya bergantung pada bentuk, karena hanya melibatkan garis, ketetanggaan, dan tepi. Begitu N cukup besar sehingga setiap bentuk muat, totalnya adalah

jumlah dari berhingga banyak polinomial, jadi polinomial. Untuk N kecil sebagian bentuk tidak muat, itulah sebabnya rumusnya baru berlaku mulai suatu N_0 ; di sini $N_0 = 3$, dan $f(1) = f(2) = 0$, sedangkan [Persamaan \(14.3\)](#) akan memberikan nilai lain. Argumen bentuk juga membatasi derajatnya: ada 8 koordinat, dan setiap kendala yang mengikat suatu koordinat pada bidak lain atau pada tepi menghapus satu derajat kebebasan. Tabel lalu memberi tahu derajat tepatnya, 5.

14.2.5 Perpangkatan cepat dengan tangan

Rumus Fermat [Persamaan \(14.6\)](#) memerlukan pangkat dengan eksponen raksasa, dan perpangkatan cepat menghitungnya dengan pengkuadratan berulang. Ambil kasus mungil $3^{-1} \bmod 7 = 3^5 \bmod 7$. Dalam biner, $5 = 101_2$. Mulai dengan hasil $r = 1$ dan basis $b = 3$.

- Bit terendah 5 adalah 1: $r = 1 \cdot 3 = 3$. Kuadratkan basis: $b = 9 \bmod 7 = 2$. Geser: eksponen menjadi 2.
- Bit terendah 2 adalah 0: r tetap 3. Kuadratkan: $b = 4$. Eksponen menjadi 1.
- Bit terendah 1 adalah 1: $r = 3 \cdot 4 = 12 \bmod 7 = 5$. Eksponen menjadi 0 dan kita berhenti.

Jadi $3^{-1} \equiv 5 \pmod{7}$, dan memang $3 \cdot 5 = 15 \equiv 1$. Untuk $M = 7\,340\,033$ eksponen $M - 2$ memiliki 23 bit, sehingga [power](#) melakukan paling banyak 23 pengkuadratan dan 23 perkalian. Setiap hasil kali adalah perkalian dua bilangan di bawah M , sehingga tetap di bawah $5.4 \cdot 10^{13}$, aman di dalam [long long](#).

14.2.6 Berapa banyak nilai yang kita perlukan?

Untuk polinomial berderajat d kita memerlukan $d + 1$ nilai untuk menentukannya, dan sekurang-kurangnya dua atau tiga lagi untuk *melihat* bahwa beda ke- d konstan dan untuk memvalidasi hasilnya. Karena biasanya kita tidak tahu d sebelumnya, *brute force* sebaiknya menghasilkan nilai sebanyak yang mampu ia hasilkan dalam beberapa menit waktu jalan. Untuk BATTLECRY *brute force* berbiaya sekitar $N^8/2$ pemeriksaan, jadi $N = 10$ (sekitar $5 \cdot 10^7$ posisi) nyaman dan $N = 14$ (sekitar $7 \cdot 10^8$) masih mungkin bila kita sabar. Dengan nilai untuk $N = 3, \dots, 10$ kita memiliki delapan titik: enam untuk menginterpolasi polinomial derajat lima dan dua untuk memeriksanya.

Berpikir tentang derajat *sebelum* menghitung juga bermanfaat. Setiap koordinat bebas suatu bidak menyumbang paling banyak satu faktor N . Di sini salah satu benteng harus berbagi baris atau kolom dengan raja hitam untuk memberi skak, dan benteng lainnya serta raja putih harus berdiri dalam hubungan yang ketat dengan raja itu untuk menutup petak pelariannya. Kendala semacam itu menghapus derajat kebebasan, sehingga derajatnya semestinya jauh di bawah 8 koordinat, dan 5 masuk akal. Ramalan yang bertentangan dengan tabel beda adalah peringatan bahwa *brute force* atau penalarannya keliru.

14.3 Soal Pemanasan

14.3.1 Tabel beda

Kode 14.1 mencetak tabel pada [Gambar 14.1](#) dan berhenti pada baris konstan pertama. Nilainya adalah hitungan *brute force* untuk $N = 3$ sampai 10.

Kode 14.1 [diff_table.cpp](#): tabel beda hingga dan derajat yang terdeteksi.

```
// Warm-up: print the finite-difference table of f(3), f(4), ..., f(10) and
// report the first row that is constant: that row's index is the degree.
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    vector<long long> row = {232, 1432, 5188, 14536,
                           34464, 72392, 138652, 246968};
    for (int d = 0; !row.empty(); d++) {
        printf("row %d:", d);
        bool constant = true;
        for (size_t i = 0; i < row.size(); i++) {
            printf(" %lld", row[i]);
            if (row[i] != row[0]) constant = false;
        }
        printf("\n");
        if (constant && row.size() > 1) {
            printf("row %d is constant -> degree %d\n", d, d);
            break;
        }
        vector<long long> next;
        for (size_t i = 0; i + 1 < row.size(); i++)
            next.push_back(row[i + 1] - row[i]);
        row = next;
    }
    return 0;
}
```

```
row 0: 232 1432 5188 14536 34464 72392 138652 246968
row 1: 1200 3756 9348 19928 37928 66260 108316
row 2: 2556 5592 10580 18000 28332 42056
row 3: 3036 4988 7420 10332 13724
row 4: 1952 2432 2912 3392
row 5: 480 480 480
row 5 is constant -> degree 5
```

Dengan delapan nilai kita dapat melihat baris konstan sepanjang 3 pada kedalaman 5. Dalam praktik, hitunglah beberapa nilai lebih banyak daripada derajat yang Anda perkirakan, sehingga baris konstannya memiliki sekurang-kurangnya dua atau tiga entri; baris “konstan” sepanjang 1 tidak membuktikan apa pun.

14.3.2 Evaluasi Lagrange modulo bilangan prima

Kode 14.2 mengevaluasi Persamaan (14.2) secara langsung, modulo $10^9 + 7$, untuk titik di $x = 1, \dots, k$. Barisan contoh 1, 5, 14, 30 adalah jumlah kuadrat $1^2 + \dots + n^2$, polinomial kubik, jadi empat titik sudah cukup.

Kode 14.2 `lagrange.cpp`: mengevaluasi polinomial interpolasi di titik baru tanpa menghitung koefisien.

```
// Warm-up: given  $f(1), \dots, f(k)$  of a polynomial of degree  $< k$ , evaluate  $f(x)$ 
// modulo a prime with the Lagrange formula (no coefficients needed).
#include <cstdio>
#include <vector>
using namespace std;

const long long MOD = 1000000007;

long long power(long long b, long long e) {
    long long r = 1;
    b %= MOD;
    for (; e > 0; e >>= 1, b = b * b % MOD)
        if (e & 1) r = r * b % MOD;
    return r;
}

long long interpolate(const vector<long long>& y, long long x) {
    int k = (int)y.size(); // points are (1, y[0]) .. (k, y[k-1])
    long long total = 0;
    for (int i = 1; i <= k; i++) {
        long long num = 1, den = 1; // basis polynomial  $L_i(x)$ 
        for (int j = 1; j <= k; j++) {
            if (j == i) continue;
            num = num * (((x - j) % MOD + MOD) % MOD) % MOD;
            den = den * (((i - j) % MOD + MOD) % MOD) % MOD;
        }
        long long term = y[i - 1] % MOD * num % MOD * power(den, MOD - 2) % MOD;
        total = (total + term) % MOD;
    }
    return total;
}

int main() {
    vector<long long> y = {1, 5, 14, 30}; //  $f(1..4)$ : sums of squares
    for (long long x : {5LL, 10LL, 100LL})
        printf("f(%lld) = %lld\n", x, interpolate(y, x));
    return 0;
}
```

```
f(5) = 55
f(10) = 385
f(100) = 338350
```

Bentuk tertutup yang terkenal $n(n+1)(2n+1)/6$ juga memberikan 55, 385, dan 338 350. Setiap evaluasi berbiaya $O(k^2 + k \log M)$; dengan hasil kali prefiks dan sufiks yang klasik biayanya dapat dijadikan $O(k)$ bila titik-titiknya bilangan bulat berurutan.

14.3.3 Seluruh alur pada soal yang lebih kecil

Sebelum soal catur, jalankan alur ini pada soal yang jawabannya juga dapat kita turunkan dengan tangan: *berapa banyak cara menempatkan dua raja identik pada papan $N \times N$ sehingga keduanya tidak saling menyerang?* Brute force (atau penghitungan cermat di kertas) memberikan, untuk $N = 1, \dots, 6$:

0, 0, 16, 78, 228, 520

Bedanya 0,16,62,150,292; lalu 16,46,88,142; lalu 30,42,54; lalu 12,12. Baris keempat konstan, jadi derajatnya 4 dan koefisien utamanya $12/4! = 1/2$. Lima titik menentukan polinomial derajat empat, dan interpolasi memberikan

$$g(N) = \frac{N^2(N^2 - 1)}{2} - 2N(N - 1) - 2(N - 1)^2$$

yang persis sama dengan hasil penghitungan langsung: semua $\binom{N^2}{2}$ pasangan petak, dikurangi $N(N - 1)$ pasangan tetangga mendatar, $N(N - 1)$ tegak, dan $2(N - 1)^2$ diagonal. Berbeda dengan BATTLECRY, polinomial ini berlaku sejak $N = 1$: argumen bentuk pada Subbab 14.2.4 meramalkannya, karena setiap bentuk (dua petak berdampingan, bertumpuk, atau diagonal) sudah muat begitu papannya 2×2 , dan untuk $N = 1$ setiap sukunya memang 0. Alur kerja itu tidak memerlukan wawasan tersebut, tetapi wawasan itu menegaskan alurnya, dan dalam lomba pemeriksaan ganda semacam itu sepadan dengan satu menit.

14.4 Studi Kasus: BATTLECRY

14.4.1 Soalnya dengan kata-kata kita sendiri

Soal SPOJ BATTLECRY [4] menempatkan empat bidak pada papan catur $N \times N$: raja putih, raja hitam, dan dua benteng putih yang identik, semuanya di petak berbeda. Hitung posisi yang kedua rajanya tidak berada di petak bertetangga dan raja hitam, yang mendapat giliran jalan, di-*skakmat*: ia diserang benteng dan tidak memiliki langkah sah. Ada banyak kasus uji, masing-masing memberikan satu N (sampai sekitar 10^5); untuk kasus uji ke- k cetak **Case k**: diikuti hitungannya modulo 7 340 033.

14.4.2 Memahami skakmat secara tepat

Brute force harus menerapkan aturan secara tepat, jadi aturan itu layak dinyatakan.

- Benteng menyerang sepanjang baris dan kolomnya sampai petak terisi pertama. Raja putih dan benteng lainnya menghalangi garisnya; raja hitam tidak menghalangi petak *di belakang dirinya* ketika ia bergerak, karena dialah bidak yang pergi.
- Raja hitam *terkena skak* bila suatu benteng menyerang petaknya.
- Langkah raja ke petak tetangga t sah bila t berada di papan, t tidak bertetangga dengan raja putih, dan tidak ada benteng tersisa yang menyerang t . Bila t ditempati benteng, langkah itu memakannya, dan hanya benteng lainnya yang dapat menyerang t .
- *Skakmat* berarti terkena skak dan tidak memiliki langkah sah.

Skakmat pada papan terkecil. Nomori baris dan kolom dari 0 pada papan 3×3 . Letakkan raja hitam di sudut (0,0), satu benteng di (0,2) dan benteng lainnya di (1,2), serta raja putih di (2,0). Kedua raja terpisah dua baris, jadi tidak bertetangga. Benteng di (0,2) menyerang sepanjang baris 0 melalui petak kosong (0,1) sampai ke raja: skak. Ketiga tetangga raja tertutup semua: (0,1) oleh benteng pertama, (1,0) dan (1,1) oleh benteng kedua sepanjang baris 1, dan keduanya juga oleh raja putih. Skakmat, dan salah satu dari 232 posisi yang terhitung untuk $N = 3$. Pindahkan benteng pertama ke (0,1) dan raja cukup memakannya, karena tidak ada bidak putih yang melindungi petak itu: inilah sebabnya *brute force* harus menangani pemakanan dengan cermat. Posisi kecil yang diperiksa dengan tangan semacam ini adalah uji terbaik bahwa kode aturan dalam *brute force* bermakna seperti yang kita kira.

14.4.3 Alurnya, langkah demi langkah

1. **Brute force.** Daftarkan setiap posisi untuk $N \leq 10$ dan hitung skakmat (Subbab 14.5). Hitungan untuk $N = 1, 2, \dots, 10$ adalah 0, 0, 232, 1432, 5188, 14536, 34464, 72392, 138652, dan 246968.

2. **Deteksi derajat.** Tabel beda nilai-nilai mulai $N = 3$ (**Gambar 14.1**) memiliki beda kelima yang konstan: derajat 5.
3. **Interpolasi.** Enam titik, $N = 3..8$, memberikan **Persamaan (14.3)**; mengalikannya dengan 3 memberikan koefisien bulat **Persamaan (14.4)**.
4. **Validasi.** $N = 9$ dan $N = 10$ sesuai dengan rumus.
5. **Evaluasi cepat.** Untuk setiap kueri: 0 bila $N < 3$; selain itu Horner (**Persamaan (14.5)**) modulo M , lalu kalikan dengan 3^{-1} .

14.4.4 Contoh

Masukan uji buatan kita sendiri dengan lima kueri, dan keluaran solusinya:

```
5
1
3
4
8
100000
```

```
Case 1: 0
Case 2: 232
Case 3: 1432
Case 4: 72392
Case 5: 4690063
```

Empat nilai pertama adalah hitungan *brute force*; yang terakhir adalah $f(100\,000) \bmod 7\,340\,033$, yang juga kami periksa dengan aritmetika bilangan bulat besar yang eksak di Python.

14.4.5 Solusi lengkap

Kode 14.3 adalah seluruh programnya. Semua pemikiran terjadi di kertas dan di *brute force*; kodenya pendek. Paruh kedua evaluasi, empat langkah Horner terakhir dan pembagian dengan 3, berada di antara kedua penanda.

Kode 14.3 **BATTLECRY.cpp**: rumus derajat 5 yang dievaluasi dengan aturan Horner dan invers modular.

```
// BATTLECRY: checkmates with a white king and two white rooks against a lone
// black king on an N x N board, modulo 7340033. For N >= 3 the count is
// f(N) = (12N^5 - 56N^4 + 66N^3 + 554N^2 - 2292N + 2424) / 3; below 3 it is 0.
#include <cstdio>
using namespace std;

const long long MOD = 7340033; // a prime: 7 * 2^20 + 1

long long power(long long b, long long e) { // b^e mod MOD, fast exponentiation
    long long r = 1;
    b %= MOD;
    while (e > 0) {
        if (e & 1) r = r * b % MOD;
        b = b * b % MOD;
        e >>= 1;
    }
    return r;
}

const long long INV3 = power(3, MOD - 2); // Fermat: 3 * INV3 = 1 (mod MOD)

long long mates(long long n) {
    if (n < 3) return 0; // the polynomial starts at N = 3
    long long v = n % MOD;
    long long h = 12; // Horner, highest coefficient first
    h = (h * v % MOD - 56 + MOD) % MOD;
    // <P50>
    // Continue Horner with the coefficients 66, 554, -2292, 2424 (keep every
    // step in [0, MOD) by adding MOD before %), then divide by 3 via INV3.
    // >>> YOUR TURN (Public Edition): write this part yourself, following the hint above.
    <<<
    // </P50>
}

int main() {
    int t;
    if (scanf("%d", &t) != 1) return 0;
    for (int tc = 1; tc <= t; tc++) {
        long long n;
        if (scanf("%lld", &n) != 1) return 0;
        printf("Case %d: %lld\n", tc, mates(n));
    }
    return 0;
}
```

14.4.6 Baris demi baris

MOD. Bilangan prima $7\,340\,033 = 7 \cdot 2^{20} + 1$. Keprimaannya penting untuk [Persamaan \(14.6\)](#).

power. Perpangkatan cepat: kuadratkan basis pada setiap langkah dan kalikan ke hasil bila bit eksponen saat ini bernilai 1. Diperlukan sekitar $\log_2 M \approx 23$ langkah.

INV3. Dihitung sekali, ketika program dimulai, dari [Persamaan \(14.6\)](#). Menghitungnya alih-alih mengetik konstantanya menghapus satu tempat persembunyian salah ketik.

mates. Mengembalikan 0 untuk $N < 3$, di luar daerah berlakunya polinomial. Selain itu fungsi ini mereduksi N modulo M dan menjalankan Horner. Setiap langkah mengalikan dengan v , mereduksi, menambahkan koefisien berikutnya, dan mereduksi lagi. Untuk koefisien negatif -56 dan -2292 kita menambahkan MOD sebelum $\%$ penutup, sehingga nilai antara tetap dalam $[0, M)$; dalam C++ sisa bagi bilangan negatif bernilai negatif, dan nilai negatif akan diam-diam merusak perkalian berikutnya. Baris terakhir membagi dengan 3 dengan cara mengalikan **INV3**.

main. Membaca banyaknya kueri dan mencetak **Case k:** beserta nilainya untuk masing-masing.

JEBAKAN. Tiga dosa modular: (1) nilai antara yang negatif, karena $(h * v - 2292) \% MOD$ dapat bernilai negatif dalam C++; (2) mengalikan dua nilai yang belum direduksi, yang dapat meluap 64 bit untuk modulus yang lebih besar; (3) membagi dengan $/$ alih-alih mengalikan dengan invers. Jebakan keempat khas interpolasi: lupa bahwa polinomialnya baru berlaku mulai $N = 3$. Mengevaluasi [Persamaan](#) (14.3) pada $N = 1$ memberikan $f(1) = 236$ alih-alih 0, dan itu salah.

14.5 Mengujinya: Rumus Melawan Papan Catur

Brute force pada **Kode 14.4** adalah program terpenting dalam bab ini: setiap koefisien [Persamaan](#) (14.4) berasal dari keluarannya, jadi program ini harus benar tanpa keraguan. Ia menomori petak $r \cdot N + c$, mencoba setiap raja putih, raja hitam, dan pasangan tak berurut petak benteng, serta menerapkan aturan Subbab 14.4.2 secara harfiah. Fungsi `rookHits` berjalan dari benteng menuju sasaran dan berhenti pada petak terisi pertama. Di `mated`, petak raja hitam sendiri tidak pernah ditandai terisi, yang mewujudkan “raja tidak menghalangi di belakang dirinya”. Ketika raja memakan benteng di petak `s`, hanya benteng lainnya yang diuji, dan petak benteng yang dimakan untuk sementara ditandai kosong.

Kode 14.4 `brute.cpp`: menghitung skakmat dengan mencoba setiap posisi.

```
// BATTLECRY brute force: place a white king, a black king and two identical
// white rooks on an N x N board in every possible way and count the positions
// where the kings are not adjacent and the black king is checkmated.
#include <cstdio>
#include <cstdlib>
using namespace std;

int N;

bool adjacent(int a, int b) { // squares are numbered r * N + c
    int dr = abs(a / N - b / N), dc = abs(a % N - b % N);
    return dr <= 1 && dc <= 1;
}

// Does a rook on square r attack square t? occ[] marks the blocking squares.
bool rookHits(int r, int t, const bool occ[]) {
    int rr = r / N, rc = r % N, tr = t / N, tc = t % N;
    if (r == t || (rr != tr && rc != tc)) return false;
    int sr = (tr > rr) - (tr < rr), sc = (tc > rc) - (tc < rc);
    for (int x = rr + sr, y = rc + sc; x != tr || y != tc; x += sr, y += sc)
        if (occ[x * N + y]) return false;
    return true;
}

bool mated(int wk, int bk, int r1, int r2) {
    static bool occ[400];
    for (int i = 0; i < N * N; i++) occ[i] = false;
    occ[wk] = occ[r1] = occ[r2] = true; // the black king never blocks
    if (!rookHits(r1, bk, occ) && !rookHits(r2, bk, occ)) return false;
    for (int dr = -1; dr <= 1; dr++)
        for (int dc = -1; dc <= 1; dc++) {
            int x = bk / N + dr, y = bk % N + dc;
            if ((dr == 0 && dc == 0) || x < 0 || y < 0 || x >= N || y >= N)
                continue;
            int s = x * N + y;
            if (adjacent(s, wk)) continue; // walking next to the white king
            bool safe = true;
            if (s != r1) { // r1 survives; is s attacked?
                occ[r2] = (s != r2);
                if (rookHits(r1, s, occ)) safe = false;
                occ[r2] = true;
            }
            if (s != r2) {
                occ[r1] = (s != r1);
                if (rookHits(r2, s, occ)) safe = false;
                occ[r1] = true;
            }
        }
}
```

Kode 14.4 (Lanjutan) *brute.cpp*: menghitung skakmat dengan mencoba setiap posisi.

```
        }
        if (safe) return false;           // the king escapes to s
    }
    return true;
}

int main() {
    int t;
    if (scanf("%d", &t) != 1) return 0;
    for (int tc = 1; tc <= t; tc++) {
        if (scanf("%d", &N) != 1) return 0;
        long long cnt = 0;
        int S = N * N;
        for (int wk = 0; wk < S; wk++)
            for (int bk = 0; bk < S; bk++) {
                if (bk == wk || adjacent(wk, bk)) continue;
                for (int r1 = 0; r1 < S; r1++)
                    for (int r2 = r1 + 1; r2 < S; r2++) {
                        if (r1 == wk || r1 == bk || r2 == wk || r2 == bk) continue;
                        if (mated(wk, bk, r1, r2)) cnt++;
                    }
            }
        printf("Case %d: %lld\n", tc, cnt % 7340033);
    }
    return 0;
}
```

Brute force ini berbiaya sekitar $N^8/2$ pemeriksaan posisi: untuk $N = 10$ itu $5 \cdot 10^7$ posisi, beberapa detik, dan itu cukup untuk program yang hanya dijalankan sekali di rumah. Kode aturannya cukup pendek untuk diperiksa baris demi baris terhadap Subbab 14.4.2, dan ada tanda kedua yang bebas bahwa ia menghitung dengan benar: nilainya untuk $N = 9$ dan $N = 10$ terletak tepat pada polinomial derajat 5 yang dicocokkan pada $N = 3.8$, sesuatu yang sangat tidak mungkin dicapai program yang salah hitung.

Pembangkitnya (**Kode 14.5**) meminta beberapa ukuran papan acak antara 1 dan 9, dan *stress.sh* yang biasa membandingkan rumus dengan *brute force*:

Kode 14.5 *gen.py*: ukuran papan kecil secara acak.

```
# Random BATTLECRY input: a few board sizes small enough for the brute force.
import random
import sys

rnd = random.Random(int(sys.argv[1]))
t = rnd.randint(1, 4)
print(t)
for _ in range(t):
    print(rnd.randint(1, 9))
```

All 500 tests passed.

Karena hanya sembilan ukuran yang mungkin, 500 uji acak memeriksa setiap ukuran berkali-kali. Itu tidak sia-sia: uji itu juga memeriksa penomoran **Case k**: dengan beberapa kueri per masukan.

14.6 Taktik pada Hari Lomba

- **Coba stetoskopnya lebih awal.** Setiap kali masukannya satu bilangan N dan jawabannya berupa hitungan, hitung 8–12 nilai pertama dengan *brute force* dan perhatikan tabel bedanya. Jika tabelnya menjadi konstan, Anda selesai dalam hitungan menit. Jika tidak, cobalah beda dari

$f(N)/N$, atau pisahkan N genap dan ganjil (sebagian jawaban berupa polinomial pada setiap kelas paritas, yang disebut *kuasi-polinomial*).

- **Pakai aritmetika eksak untuk interpolasi.** Jabarkan [Persamaan](#) (14.2) dengan pecahan (`Fractions.Fraction` di Python, atau dengan tangan memakai penyebut bersama). Bilangan titik mengambang kehilangan koefisien eksaknya.
- **Jangan mempercayai nilai-nilai pertama begitu saja.** Polinomialnya bisa baru dimulai pada suatu $N_0 > 1$. Lihat di mana tabel beda menjadi teratur, dan simpan titik tambahan untuk validasi, seperti yang kita lakukan dengan $N = 9$ dan 10 .
- **Bila modulusnya bukan prima,** invers dari 3 hanya ada jika 3 tidak membagi modulus. Kalau tidak, hindari pembagian dengan mengevaluasi $f(N)$ langsung dengan pecahan yang direduksi sebelum modulus, atau gunakan rumus Lagrange atas bilangan bulat.
- **Ketahui biayanya.** Seperti setiap rumus $O(1)$ dalam kotak perkakas buku ini [5], solusi yang dihasilkan menjawab setiap kueri dalam $O(1)$: lima perkalian. Tidak ada larik, tidak ada prakomputasi, dan tidak ada peluang melampaui batas waktu.

14.6.1 Bila jawabannya bukan polinomial

Tabel beda tidak selalu menjadi konstan, dan itu pun informasi.

- Bila bedanya tampak seperti *kelipatan tetap dari baris sebelumnya*, barisannya tumbuh eksponensial, dan jawabannya kemungkinan *rekurensi linear* seperti $f(N) = 3f(N - 1) - f(N - 2)$. Nilai *brute force* kecil sekali lagi menentukan rekurensinya (algoritma Berlekamp–Massey menemukan yang terpendek secara otomatis), lalu perpangkatan matriks mengevaluasinya untuk N raksasa dalam $O(\log N)$ langkah.
- Bila tabelnya hanya konstan pada N genap atau hanya pada N ganjil, pisahkan barisan menurut paritas dan tangani setiap paruh secara terpisah.
- Bila tidak ada yang menjadi teratur, jawabannya mungkin bergantung pada N dengan cara yang lebih halus (pembagi, bilangan prima, digit), dan nilai *brute force* tetap berguna: bandingkan dalam kepala dengan barisan yang Anda kenal, atau cari pola dalam faktorisasi primanya.

Dalam setiap kasus langkah pertamanya sama seperti dalam bab ini: *brute force* jujur yang Anda percayai, dan tabel nilai yang Anda perhatikan sebelum memikirkan rumus.

14.6.2 Kesalahan yang sering terjadi

Beberapa kesalahan muncul berulang kali ketika peserta memakai alur ini untuk pertama kalinya.

- **Brute force yang diam-diam salah.** Setiap koefisien mewarisi kesalahannya. Untuk soal catur biang keladinya biasanya penghalangan (apakah suatu bidak menghalangi garis atau tidak?), pemakanan, dan petak raja yang bergerak itu sendiri. Periksa dua atau tiga posisi dengan tangan, seperti pada Subbab 14.4.2, sebelum mempercayai tabel apa pun.
- **Terlalu sedikit nilai.** Interpolasi melalui tepat $d + 1$ nilai selalu menghasilkan polinomial, entah barisannya polinomial atau bukan. Simpan sekurang-kurangnya dua nilai untuk validasi.
- **Memulai tabel terlalu awal.** Bila nilai pertama berasal dari papan yang tidak memuat sebagian bentuk, tabelnya tampak tidak teratur. Buang nilai pertama satu per satu sampai bedanya tenang; di sini artinya mulai dari $N = 3$.
- **Pecahan dan bilangan titik mengambang.** Koefisien seperti $56/3$ tidak dapat disimpan secara eksak dalam `double`. Hilangkan penyebutnya dan gunakan invers modular, seperti yang dilakukan [Kode 14.3](#).
- **Lupa mereduksi N .** Bila N dapat lebih besar daripada modulus, hitung `v = n % MOD` lebih dahulu; kalau tidak, perkalian pertama dapat meluap.

14.7 Ringkasan Bab

- Hitungan penempatan dengan banyak bidak tetap dan aturan lokal sering berupa polinomial dalam N mulai suatu N_0 , karena setiap bentuk posisi menyumbang hasil kali faktor-faktor linear.
- Beda hingga ([Persamaan \(14.1\)](#)) mendeteksi derajatnya: baris d konstan berarti derajat d , dengan konstanta $a \cdot d!$.
- Interpolasi Lagrange ([Persamaan \(14.2\)](#)) memulihkan polinomial dari $d + 1$ nilai; nilai tambahan memvalidasinya.
- Aturan Horner ([Persamaan \(14.5\)](#)) mengevaluasinya dengan satu perkalian per koefisien, dan teorema Fermat ([Persamaan \(14.6\)](#)) memberikan invers yang diperlukan untuk koefisien pecahan.
- Untuk BATTLECRY, $3f(N) = 12N^5 - 56N^4 + 66N^3 + 554N^2 - 2292N + 2424$ untuk $N \geq 3$, dan $f(N) = 0$ untuk $N < 3$.

14.8 Soal Latihan

1. Nilai suatu barisan untuk $n = 1..5$ adalah 2,9,28,65,126. Bangun tabel bedanya, tentukan derajat dan koefisien utamanya, lalu tebak rumusnya.
2. Gunakan [Persamaan \(14.2\)](#) untuk menentukan $f(10)$ dari polinomial kubik dengan $f(1..4) = 1,5,14,30$ dengan tangan.
3. Hitung 3^{-1} modulo 7 dengan [Persamaan \(14.6\)](#), lalu periksa.
4. Evaluasi [Persamaan \(14.4\)](#) pada $N = 3$ dengan aturan Horner, tunjukkan setiap nilai antara, lalu bagi dengan 3.
5. Hitung dengan *brute force* penempatan dua benteng identik yang tidak saling menyerang pada papan $N \times N$ untuk $N = 1..6$, deteksi derajatnya, dan temukan rumusnya.
6. Jelaskan mengapa `(h * v - 2292) % MOD` dapat menghasilkan jawaban akhir yang salah meskipun C++ menghitung sisa baginya “dengan benar”.
7. Mengapa tidak cukup menginterpolasi melalui *tepat* $d + 1$ nilai bila derajat d ditebak dari tabel?
8. Banyaknya persegi segala ukuran pada papan $N \times N$ adalah 1,5,14,30, ... untuk $N = 1,2,3,4$. Polinomial apakah ini, dan berapa nilainya untuk $N = 100$ modulo $10^9 + 7$?

14.9 Penyelesaian Soal Latihan

1. Bedanya: 7,19,37,61; lalu 12,18,24; lalu 6,6. Derajat 3, koefisien utama $6/3! = 1$. Rumusnya $n^3 + 1$: 2,9,28,65,126 cocok.
2. Dengan $x_i = 1,2,3,4$ dan $x = 10$ keempat nilai basisnya adalah $\frac{(8)(7)(6)}{(-1)(-2)(-3)} = -56$, $\frac{(9)(7)(6)}{(1)(-1)(-2)} = 189$, $\frac{(9)(8)(6)}{(2)(1)(-1)} = -216$, dan $\frac{(9)(8)(7)}{(3)(2)(1)} = 84$. Maka $f(10) = 1 \cdot (-56) + 5 \cdot 189 + 14 \cdot (-216) + 30 \cdot 84 = -56 + 945 - 3024 + 2520 = 385$.
3. $3^{7-2} = 3^5 = 243 = 34 \cdot 7 + 5$, jadi $3^{-1} \equiv 5 \pmod{7}$. Periksa: $3 \cdot 5 = 15 = 2 \cdot 7 + 1$.
4. Mulai dengan 12. Lalu $12 \cdot 3 - 56 = -20$; $-20 \cdot 3 + 66 = 6$; $6 \cdot 3 + 554 = 572$; $572 \cdot 3 - 2292 = -576$; $-576 \cdot 3 + 2424 = 696$. Dan $696/3 = 232$, sesuai dengan *brute force*. (Modulo M , nilai negatifnya menjadi $M - 20$ dan $M - 576$, dan hasilnya sama.)
5. Dua benteng identik tidak saling menyerang tepat ketika keduanya berada di baris berbeda dan kolom berbeda: pilih dua baris, dua kolom, dan satu dari dua pemasangan, $\binom{N}{2}^2 \cdot 2$. Nilainya

- untuk $N = 1..6$ adalah 0,2,18,72,200,450; beda keempatnya konstanta 12, jadi derajatnya 4 dan koefisien utamanya $12/4! = 1/2$, sesuai dengan $\frac{N^2(N-1)^2}{2}$.
6. Dalam C++, `%` mempertahankan tanda operand kiri, jadi bila `h * v < 2292` hasilnya negatif. Langkah berikutnya mengalikan bilangan negatif dengan `v`, dan `return` terakhir dapat mencetak bilangan negatif atau, setelah penjumlahan berikutnya, nilai yang meleset satu kelipatan `MOD` tetapi di luar $[0, M)$. Menambahkan `MOD` sebelum `%` menjaga setiap nilai dalam jangkauan.
7. Melalui sembarang $d + 1$ titik selalu ada *suatu* polinomial berderajat paling tinggi d , sehingga interpolasi selalu “berhasil”, bahkan ketika barisan sebenarnya bukan polinomial atau berderajat lebih tinggi. Hanya nilai yang tidak dipakai dalam interpolasi yang dapat menegaskan hasilnya. Itulah sebabnya kita mencocokkan $N = 3..8$ dan memeriksa $N = 9$ dan 10.
8. Itu adalah jumlah kuadrat $1^2 + 2^2 + \dots + N^2 = N(N + 1)(2N + 1)/6$, barisan yang sama seperti pada **Kode 14.2**: ada $(N - k + 1)^2$ persegi bersisi k . Untuk $N = 100$ nilainya 338 350, lebih kecil daripada modulus, sehingga sekaligus merupakan sisa baginya, sebagaimana dicetak program.

Daftar Pustaka

- [1] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1994.
- [2] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.
- [3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
- [4] Sphere Online Judge, “BATTLECRY - Battle Of The Bastards - Kings Vs Rooks.” [Online]. Available: <https://www.spoj.com/problems/BATTLECRY/>
- [5] A. Laaksonen, *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, 3rd ed. Cham, Switzerland: Springer, 2024.

Bab 15

Aliran Maksimum dengan Algoritma Dinic — FASTFLOW

Bab terakhir ini merangkum banyak hal yang telah dibangun buku ini: graf, pencarian melebar, pencarian mendalam, aritmetika 64 bit yang cermat, dan disiplin menguji terhadap *brute force*. Topiknya adalah *aliran maksimum* (*maximum flow*): berapa banyak yang dapat dikirim dari sumber ke muara melalui jaringan yang sisinya berkapasitas terbatas? Anda akan mempelajari gagasan graf residu dan lintasan penambah dari nol, melihat mengapa teorema aliran maksimum–potongan minimum membuat jawabannya dapat diperiksa, berkenalan dengan algoritma Edmonds–Karp yang sederhana sebagai pemanasan, lalu menulis algoritma Dinic dengan *graf level* dan *busur kini* (*current arc*), cukup cepat untuk soal SPOJ FASTFLOW dengan 5000 simpul dan 30 000 sisi tak berarah. Bab ini, dan buku ini, ditutup dengan rencana latihan praktis: jalan Anda menuju podium nasional.

15.1 Di Mana Ini Berguna: Air untuk Kebun

Sebuah koperasi desa mengalirkan air dari mata air di bukit ke tandon di samping kebun buah dan durian. Di antara keduanya terbentang jaringan pipa dan saluran, masing-masing dengan laju maksimum dalam liter per detik, yang bertemu di bak-bak kecil tempat air dapat berpisah dan bergabung. Berapa liter per detik paling banyak yang dapat mencapai tandon? Dan bila koperasi punya dana untuk melebarkan beberapa pipa, pipa mana yang sebenarnya membatasi pasokan?

Ganti air dengan data dan pipa dengan jalur jaringan, atau dengan jalan dan truk, atau dengan guru dan kelas, dan pertanyaan yang sama muncul: jumlah terbesar yang dapat berpindah dari *sumber* ke *muara* ketika setiap sambungan memiliki kapasitas. Banyak soal penugasan juga tereduksi ke sini: memasang siswa dengan proyek, pekerja dengan giliran, atau peserta dengan komputer, dengan setiap “satuan aliran” berarti satu penugasan. Aliran maksimum adalah salah satu model yang paling sering dipakai ulang dalam algoritmika [1], dan olimpiade nasional beserta pelatnasnya memperlakukannya sebagai materi lanjut yang wajar.

15.2 Tekniknya dari Nol

15.2.1 Aliran, kapasitas, dan potongan

Jaringan aliran adalah graf berarah dengan sumber s , muara t , dan kapasitas $c(u, v) \geq 0$ pada setiap sisi. *Aliran* memberikan nilai $f(u, v)$ pada setiap sisi dengan dua aturan:

$$0 \leq f(u, v) \leq c(u, v), \quad \sum_u f(u, v) = \sum_w f(v, w) \text{ untuk setiap } v \neq s, t \quad (15.1)$$

Aturan pertama pada [Persamaan \(15.1\)](#) menyatakan tidak ada pipa yang membawa lebih dari kapasitasnya; aturan kedua menyatakan air tidak diciptakan dan tidak hilang di simpul dalam. *Nilai aliran* adalah jumlah bersih yang meninggalkan s . Sisi *tak berarah* berkapasitas c , seperti pada FASTFLOW, dapat membawa sampai c ke salah satu arah.

Potongan s - t membelah simpul menjadi himpunan S yang memuat s dan himpunan T yang memuat t . Kapasitasnya adalah jumlah kapasitas sisi dari S ke T . Setiap satuan aliran harus menyeberang dari S ke T di suatu tempat, jadi nilai aliran apa pun paling banyak sebesar kapasitas potongan apa pun. Fakta yang menakjubkan adalah bahwa batas itu selalu tercapai:

TEOREMA (aliran maksimum–potongan minimum). Dalam setiap jaringan aliran, nilai aliran maksimum sama dengan kapasitas minimum potongan s - t [2], [1].

Teorema ini bukan hanya indah; ia juga praktis. Untuk graf mungil kita dapat menghitung potongan minimum dengan mencoba setiap himpunan S , dan itu memberikan *brute force* yang bebas untuk pengujian, yang kita pakai pada Subbab 15.5.

15.2.2 Graf residu dan lintasan penambah

Bagaimana menemukan aliran maksimum? Mulailah dengan aliran nol dan berulang kali dorong lebih banyak aliran melalui suatu lintasan dari s ke t . Masalahnya, pilihan awal dapat menghalangi pilihan yang lebih baik belakangan, jadi kita harus mengizinkan aliran *dibatalkan*. Graf residu melakukan persis hal itu. Untuk setiap sisi (u, v) yang membawa aliran f , graf ini menyimpan dua sisi residu:

$$r(u, v) = c(u, v) - f(u, v), \quad r(v, u) = r(v, u) + f(u, v) \quad (15.2)$$

Kapasitas residu maju adalah apa yang masih dapat ditampung pipa; kapasitas residu mundur adalah berapa banyak aliran saat ini yang dapat dikirim balik, yaitu *dibatalkan*. *Lintasan penambah (augmenting path)* adalah lintasan dari s ke t yang setiap kapasitas residunya positif. Mendorong kapasitas residu terkecil sepanjang lintasan itu menaikkan nilai aliran dan menjaga [Persamaan \(15.1\)](#) tetap benar. Ford dan Fulkerson membuktikan bahwa suatu aliran maksimum tepat ketika tidak ada lintasan penambah [2].

Dalam kode, setiap sisi disimpan bersama sisi *pasangan* dengan arah berlawanan, dan mendorong d satuan mengurangi d dari sisi itu dan menambahkan d ke pasangannya. Untuk sisi berarah, pasangannya dimulai dengan kapasitas 0. Untuk sisi tak berarah berkapasitas c , kedua arah dimulai dengan c , dan keduanya saling menjadi pasangan: mendorong d dari u ke v menyisakan $c - d$ dari u ke v dan $c + d$ dari v ke u , dan itu tepat, karena aliran dari u ke v dapat *dibatalkan* lalu c lagi dapat dikirim dari v ke u .

15.2.3 Edmonds–Karp: lintasan penambah terpendek

Memilih lintasan penambah secara sembarangan dapat memakan waktu sangat lama; dengan kapasitas irasional prosesnya bahkan dapat tidak berhenti. Edmonds dan Karp menunjukkan bahwa memilih lintasan penambah *terpendek* (paling sedikit sisi), yang ditemukan dengan BFS, selalu memerlukan paling banyak $O(VE)$ penambahan, sehingga totalnya $O(VE^2)$ [3]. Inilah pemanasan pada Subbab 15.3.

15.2.4 Dinic: banyak lintasan per BFS

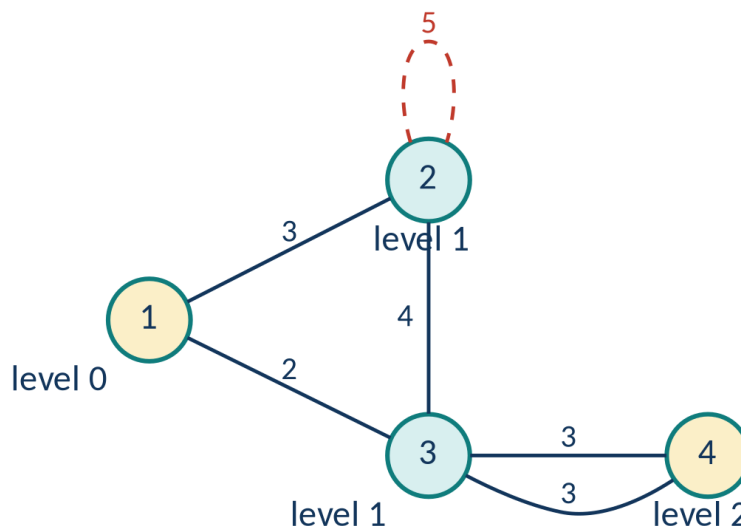
Algoritma Dinic [4] membuat gagasan lintasan terpendek jauh lebih cepat dengan mengerjakan satu *fase* penuh penambahan per BFS.

1. **Graf level.** BFS dari s dalam graf residu memberikan setiap simpul jaraknya $level[v]$. Pertahankan hanya sisi residu yang berjalan dari level k ke level $k + 1$. Setiap lintasan penambah terpendek hanya memakai sisi-sisi ini. Jika t tidak terjangkau, berhenti: alirannya sudah maksimum.

2. **Aliran pemblokir.** Dengan DFS, dorong aliran sepanjang lintasan graf level dari s ke t sampai tidak ada lagi lintasan semacam itu. Hasilnya disebut *aliran pemblokir* (*blocking flow*).

3. **Ulangi** dari langkah 1.

Gambar 15.1 memperlihatkan graf level contoh FASTFLOW: simpul 1 di level 0, simpul 2 dan 3 di level 1, simpul 4 di level 2. Sisi antara 2 dan 3 menghubungkan dua simpul pada level yang sama, jadi fase pertama mengabaikannya; gelang di simpul 2 tidak pernah membawa apa pun.



Gambar 15.1 Contoh FASTFLOW sebagai graf tak berarah dengan kapasitas dan level BFS. Gelang putus-putus dibuang; sisi 2–3 terletak di dalam level 1 dan tidak dipakai pada fase pertama.

Mesin kecepatannya adalah *busur kini*. Setiap simpul v menyimpan indeks $\text{arc}[v]$ ke dalam daftar sisinya. Selama satu fase, bila suatu sisi yang keluar dari v ternyata tidak berguna (penuh, atau menuju jalan buntu), sisi itu tetap tidak berguna selama sisa fase, sehingga DFS tidak pernah melihatnya lagi: $\text{arc}[v]$ hanya bergerak maju. Karena itu setiap fase memeriksa setiap sisi dalam jumlah terbatas ditambah sekali per lintasan penambah, yang memberikan $O(VE)$ per fase. Setiap fase menaikkan jarak dari s ke t sekurang-kurangnya satu, sehingga paling banyak ada $V - 1$ fase, dan totalnya

$$O(V^2E) \quad (15.3)$$

Pada jaringan berkapasitas satuan, seperti pemasangan bipartit, batasnya membaik menjadi $O(E\sqrt{V})$ [5], dan pada graf lomba yang nyata Dinic biasanya jauh lebih cepat daripada yang disiratkan [Persamaan](#) (15.3). **Pseudokode 15.1** merangkumnya.

Pseudokode 15.1 Algoritma Dinic dengan graf level dan busur kini.

```
MAX-FLOW(s, t)
1  flow = 0
2  while BFS(s, t) reaches t           // hitung level[]
3      arc[v] = 0 for every v
4      repeat
5          d = DFS(s, t, infinity)
6          flow = flow + d
7      until d == 0
8  return flow

DFS(v, t, f)
1  if v == t: return f
2  while arc[v] < number of edges of v
3      e = edge arc[v] of v
4      if e.cap > 0 and level[e.to] == level[v] + 1
5          d = DFS(e.to, t, min(f, e.cap))
6          if d > 0
7              e.cap -= d; partner(e).cap += d
8          return d
9      arc[v] = arc[v] + 1             // e tak berguna di fase ini
10 return 0
```

15.2.5 Mengapa setiap fase membuat muara lebih jauh

Misalkan δ adalah jarak dari s ke t pada awal suatu fase. Setelah aliran pemblokir, setiap lintasan sepanjang δ dalam graf residu telah hancur: masing-masing memuat sisi yang jenuh. Dapatkah lintasan pendek baru muncul? Mendorong aliran sepanjang sisi dari level k ke level $k + 1$ dapat menciptakan sisi kebalikannya, yang berjalan dari level $k + 1$ kembali ke level k , yaitu *turun* satu level. Lintasan dari s ke t harus naik δ level secara total; setiap sisi naik paling banyak satu level, dan setiap sisi kebalikan turun. Jadi setiap lintasan yang memakai sisi kebalikan baru memerlukan sekurang-kurangnya $\delta + 2$ sisi, dan setiap lintasan yang tepat δ sisi harus hanya memakai sisi graf level lama, yang sudah diblokir oleh aliran pemblokir. Karena itu BFS berikutnya menemukan t pada jarak sekurang-kurangnya $\delta + 1$. Jarak paling besar $V - 1$, sehingga paling banyak ada $V - 1$ fase, dan dari situlah faktor V pada [Persamaan \(15.3\)](#) berasal.

15.3 Pemanasan: Edmonds–Karp pada Matriks

Untuk graf kecil, matriks ketetanggaan berisi kapasitas residu membuat gagasan residu sangat konkret. **Kode 15.1** membaca graf berarah, menggabungkan sisi sejajar dengan menjumlahkan kapasitasnya, lalu mengulang: BFS untuk lintasan terpendek, temukan kemacetannya, dorong, dan perbarui kedua arah setiap sisi pada lintasan seperti pada [Persamaan \(15.2\)](#).

Kode 15.1 `edmonds_karp.cpp`: aliran maksimum dengan lintasan penambah terpendek pada matriks kapasitas.

```
// Warm-up: maximum flow in a small directed graph with Edmonds-Karp
// (always augment along a shortest path found by BFS), on a matrix.
#include <algorithm>
#include <cstdio>
#include <queue>
#include <vector>
using namespace std;

int main() {
    int n, m, s, t;
    if (scanf("%d %d %d %d", &n, &m, &s, &t) != 4) return 0;
    vector<vector<long long>> cap(n + 1, vector<long long>(n + 1, 0));
    for (int i = 0; i < m; i++) {
        int u, v;
        long long c;
        if (scanf("%d %d %lld", &u, &v, &c) != 3) return 0;
        cap[u][v] += c;           // parallel edges add up
    }
    long long flow = 0;
    while (true) {
        vector<int> parent(n + 1, -1);
        parent[s] = s;
        queue<int> q;
        q.push(s);
        while (!q.empty() && parent[t] < 0) {
            int x = q.front();
            q.pop();
            for (int y = 1; y <= n; y++)
                if (parent[y] < 0 && cap[x][y] > 0) parent[y] = x, q.push(y);
        }
        if (parent[t] < 0) break;           // no augmenting path is left
        long long push = -1;
        for (int y = t; y != s; y = parent[y])
            if (push < 0 || cap[parent[y]][y] < push) push = cap[parent[y]][y];
        for (int y = t; y != s; y = parent[y]) {
            cap[parent[y]][y] -= push;     // use forward capacity
            cap[y][parent[y]] += push;     // allow undoing it later
        }
        printf("augment by %lld\n", push);
        flow += push;
    }
    printf("max flow = %lld\n", flow);
    return 0;
}
```

Pada jaringan kecil dengan sumber 1 dan muara 4, sisi $1 \rightarrow 2$ (3), $1 \rightarrow 3$ (2), $2 \rightarrow 3$ (1), $2 \rightarrow 4$ (2), dan $3 \rightarrow 4$ (3):

```
4 5 1 4
1 2 3
1 3 2
2 3 1
2 4 2
3 4 3
```

```
augment by 2
augment by 2
augment by 1
max flow = 5
```

Dua penambahan pertama memakai dua lintasan sepanjang 2, $1 \rightarrow 2 \rightarrow 4$ dan $1 \rightarrow 3 \rightarrow 4$, masing-masing dengan kemacetan 2. Yang ketiga memakai $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ dengan kemacetan 1. Nilai 5 sama dengan kapasitas potongan {1} melawan sisanya, $3 + 2$, sehingga menurut teorema aliran maksimum–potongan minimum nilai itu maksimum. Dengan memori V^2 dan $O(V^2)$ per BFS,

program ini hanya cocok untuk beberapa ratus simpul, dan justru itulah sebabnya FASTFLOW memerlukan sesuatu yang lebih baik.

15.4 Studi Kasus: FASTFLOW

15.4.1 Soalnya dengan kata-kata kita sendiri

Soal SPOJ FASTFLOW [6] memberikan graf tak berarah dengan $N \leq 5000$ simpul dan $M \leq 30\,000$ sisi. Setiap sisi menghubungkan dua simpul dan berkapasitas sampai 10^9 ; pasangan yang sama dapat dihubungkan beberapa sisi, dan suatu sisi dapat menghubungkan simpul dengan dirinya sendiri. Carilah aliran maksimum dari simpul 1 ke simpul N . Seperti namanya, solusinya harus cepat.

15.4.2 Contoh

```
4 6
1 2 3
2 3 4
3 1 2
2 2 5
3 4 3
4 3 3
```

5

Dengan tangan: potongan $\{1\}$ melawan sisanya berkapasitas $3 + 2 = 5$ (sisi $1 \rightarrow 2$ dan $3 \rightarrow 1$). Potongan $\{1, 2, 3\}$ melawan $\{4\}$ berkapasitas $3 + 3 = 6$. Tidak ada potongan yang lebih kecil dari 5, dan Dinic menemukan alirannya: pada fase pertama ia mendorong 2 sepanjang $1 \rightarrow 3 \rightarrow 4$ (lintasan melalui simpul 2 buntu, karena 2 tidak memiliki sisi ke level 2); BFS kedua memberikan level 1: 0, 2: 1, 3: 2, 4: 3, dan fase itu mendorong 3 sepanjang $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$, terbagi pada dua sisi sejajar antara 3 dan 4. Setelah itu simpul 1 tidak memiliki sisi residu lagi, dan totalnya 5.

15.4.3 Keputusan rancangan

- **Daftar ketetanggaan dengan indeks pasangan.** Setiap `Edge` menyimpan kepala `to`, kapasitas residu `cap`, dan `rev`, indeks pasangannya di dalam `adj[to]`. Ini memberikan akses $O(1)$ ke pasangan dan memakai memori $O(V + E)$.
- **Sisi tak berarah sebagai satu pasangan.** Satu panggilan `addEdge(u, v, c)` menciptakan kedua arah dengan kapasitas c , yang saling menjadi pasangan, seperti dijelaskan pada Subbab 15.2.2.
- **Gelang dibuang,** karena gelang tidak pernah dapat menjadi bagian lintasan dari s ke t . Sisi sejajar dipertahankan sebagai pasangan terpisah; tidak perlu digabung.
- **Aliran 64 bit.** Satu sisi membawa sampai 10^9 , dan totalnya dapat mencapai $30\,000 \cdot 10^9 = 3 \cdot 10^{13}$. Setiap kapasitas, setiap jumlah yang didorong, dan totalnya bertipe `long long`.

15.4.4 Solusi lengkap

Kode 15.2 adalah program utuhnya. Paruh pertama inti, BFS yang membangun level, berada di luar penanda. Paruh kedua, DFS aliran pemblokir dengan busur kini, berada di antara `// <P50>` dan `// </P50>`.

Kode 15.2 `FASTFLOW.cpp`: algoritma Dinic dengan graf level dan busur kini, $O(V^2E)$ pada kasus terburuk.

```
// FASTFLOW: maximum flow from node 1 to node N in an undirected graph with
// up to 5000 nodes and 30000 edges. Dinic's algorithm with current arcs.
#include <algorithm>
#include <cstdio>
#include <queue>
#include <vector>
using namespace std;

struct Edge {
    int to, rev;          // head of the edge; index of its partner in adj[to]
    long long cap;        // residual capacity
};

int n, m;
vector<vector<Edge>> adj;
vector<int> level, arc; // BFS level of each node; current arc of each node

void addEdge(int u, int v, long long c) {
    // undirected: each direction starts with the full capacity c, and the
    // two edges are each other's reverse in the residual graph
    adj[u].push_back({v, (int)adj[v].size(), c});
    adj[v].push_back({u, (int)adj[u].size() - 1, c});
}

bool bfs(int s, int t) { // levels in the residual graph; can we reach t?
    fill(level.begin(), level.end(), -1);
    queue<int> q;
    level[s] = 0;
    q.push(s);
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (const Edge& e : adj[v])
            if (e.cap > 0 && level[e.to] < 0) {
                level[e.to] = level[v] + 1;
                q.push(e.to);
            }
    }
    return level[t] >= 0;
}
```

Kode 15.2 (Lanjutan) FASTFLOW.cpp: algoritma Dinic dengan graf level dan busur kini, $O(V^2 E)$ pada kasus terburuk.

```
long long dfs(int v, int t, long long f) {
    if (v == t) return f;
    // <P50>
    // Try the edges of v from its current arc on; follow only residual edges
    // that go exactly one level deeper. Push along the first path that
    // reaches t, update both residual edges, and keep arc[v] where it stopped.
    // >>> YOUR TURN (Public Edition): write this part yourself, following the hint above.
    <<<
    // </P50>
}

long long maxFlow(int s, int t) {
    long long flow = 0;
    while (bfs(s, t)) { // one phase per BFS
        fill(arc.begin(), arc.end(), 0);
        while (long long d = dfs(s, t, (long long)4e18)) flow += d;
    }
    return flow;
}

int main() {
    if (scanf("%d %d", &n, &m) != 2) return 0;
    adj.assign(n + 1, {});
    level.assign(n + 1, -1);
    arc.assign(n + 1, 0);
    for (int i = 0; i < m; i++) {
        int u, v;
        long long c;
        if (scanf("%d %d %lld", &u, &v, &c) != 3) return 0;
        if (u != v) addEdge(u, v, c); // a self-loop carries nothing
    }
    printf("%lld\n", maxFlow(1, n));
    return 0;
}
```

15.4.5 Baris demi baris

addEdge. `push_back` pertama menambahkan sisi $u \rightarrow v$; pasangannya akan menjadi elemen berikutnya di `adj[v]`, yang indeksnya adalah ukuran `adj[v]` saat ini. `push_back` kedua menambahkan $v \rightarrow u$, yang pasangannya adalah sisi yang baru saja ditambahkan, elemen terakhir `adj[u]`. Urutan kedua baris itu penting: indeks `adj[u].size() - 1` harus dibaca setelah `push_back` pertama.

bfs. BFS buku teks atas sisi yang kapasitas residunya positif. Fungsi ini mengembalikan apakah muara mendapat level.

dfs. Referensi `int& i = arc[v]` membuat variabel gelung *menjadi* busur kini: menaikkan `i` menggerakkan `arc[v]` maju secara permanen untuk fase ini. Suatu sisi hanya diikuti bila masih berkapasitas dan turun tepat satu level lebih dalam. Bila panggilan rekursif menghantarkan `d > 0` satuan, sisi itu kehilangan `d`, pasangannya memperoleh `d`, dan `d` langsung dikembalikan. Perhatikan bahwa ketika berhasil, gelung kembali *sebelum* `i++`, sehingga sisi yang sama akan dicoba lagi oleh DFS berikutnya dalam fase itu; itu benar, karena sisi itu mungkin masih berkapasitas. Ketika gagal, gelung melewati sisi itu untuk selamanya.

maxFlow. Untuk setiap BFS yang berhasil, fungsi ini mengatur ulang semua busur kini dan memanggil `dfs` sampai hasilnya 0, menambahkan setiap hasil ke `flow`. Bentuk `while (long long d = ...)` mendeklarasikan `d` di dalam kondisi dan berhenti bila nilainya 0.

main. Membaca graf, melewati gelang, dan mencetak aliran maksimum dari 1 ke N .

JEBAKAN. (1) *Mengatur ulang busur kini di tengah fase, atau tidak memakainya sama sekali: jawabannya tetap benar, tetapi waktu jalan dapat meledak dan FASTFLOW melampaui batas waktu.* (2) *Memodelkan sisi tak berarah sebagai dua sisi berarah, masing-masing dengan pasangan berkapasitas nol: benar tetapi sisinya dua kali lipat; lebih buruk lagi, melupakan salah satunya membagi dua kapasitas ke satu arah.* (3) *Kapasitas atau total bertipe `int`: jawabannya dapat melampaui 2^{31} .* (4) *Memakai `Level[e.to] > Level[v]` alih-alih `== Level[v] + 1`: tetap berhenti, tetapi keluar dari graf level dan kehilangan jaminan Subbab 15.2.4.* (5) *Melupakan bahwa simpul dinomori dari 1: larik berukuran N alih-alih $N + 1$.*

15.5 Mengujinya: Dinic Melawan Potongan Minimum

Teorema aliran maksimum–potongan minimum memberikan *brute force* yang sama sekali tidak berbagi kode dengan Dinic: coba setiap himpunan simpul S dengan $1 \in S$ dan $N \notin S$, jumlahkan kapasitas sisi yang menyeberang, dan ambil minimumnya (**Kode 15.3**). Untuk $N \leq 9$ ada paling banyak $2^7 = 128$ himpunan semacam itu.

Kode 15.3 `brute.cpp`: potongan minimum dengan mencoba setiap himpunan S .

```
// FASTFLOW reference for tiny graphs: by the max-flow min-cut theorem the
// answer is the cheapest cut, so try every set S with 1 in S and N not in S.
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n, m;
    if (scanf("%d %d", &n, &m) != 2) return 0;
    vector<int> u(m), v(m);
    vector<long long> c(m);
    for (int i = 0; i < m; i++)
        if (scanf("%d %d %lld", &u[i], &v[i], &c[i]) != 3) return 0;
    long long best = -1;
    for (int mask = 0; mask < (1 << n); mask++) {
        bool in1 = mask & 1, inN = (mask >> (n - 1)) & 1;
        if (!in1 || inN) continue;
        long long cut = 0;
        for (int i = 0; i < m; i++) {
            bool a = (mask >> (u[i] - 1)) & 1, b = (mask >> (v[i] - 1)) & 1;
            if (a != b) cut += c[i]; // undirected edge crosses the cut
        }
        if (best < 0 || cut < best) best = cut;
    }
    printf("%lld\n", best < 0 ? 0 : best);
    return 0;
}
```

Pembangkitnya (**Kode 15.4**) membuat multigraf mungil dengan gelang, sisi sejajar, kapasitas nol, dan pada sekitar sepertiga kasus kapasitas sampai 10^9 untuk menguji aritmetika 64 bit.

Kode 15.4 `gen.py`: multigraf tak berarah acak yang kecil.

```
# Random FASTFLOW input: tiny undirected multigraphs with loops, parallel
# edges and occasionally huge capacities.
import random
import sys

rnd = random.Random(int(sys.argv[1]))
n = rnd.randint(2, 9)
m = rnd.randint(0, 20)
big = rnd.random() < 0.3
print(n, m)
for _ in range(m):
    c = rnd.randint(0, 10**9) if big else rnd.randint(0, 9)
    print(rnd.randint(1, n), rnd.randint(1, n), c)
```

All 500 tests passed.

Untuk kecepatan kami membangkitkan dua graf berukuran maksimum: satu dengan 30 000 sisi acak, dan satu graf berlapis dengan 49 lapis berisi 100 simpul dan enam sisi acak dari setiap simpul ke lapis berikutnya, bentuk yang memaksa banyak fase. **Tabel 15.1** memperlihatkan keduanya selesai seketika.

Tabel 15.1 Waktu jalan **FASTFLOW.cpp** pada masukan berukuran maksimum.

Masukan ($N=5000, M=30\,000$)	Aliran maksimum	Waktu
sisi acak, kapasitas sampai 10^9	3 904 368 904	0,01 s
49 lapis berisi 100 simpul	49 995 537 200	0,01 s

Kedua jawaban melampaui 2^{31} , yang kedua lebih dari 20 kali lipat: program dengan aliran **int** akan mencetak sampah pada jenis uji yang pasti disertakan juri.

15.6 Taktik pada Hari Lomba

- **Simpan satu Dinic di kepala.** Kode **Kode 15.2** sekitar 60 baris. Berlatihlah mengetiknya dari ingatan sampai selesai dalam kurang dari lima belas menit dan lolos *brute force* pada percobaan pertama.
- **Modelkan dulu, baru tulis kode.** Kebanyakan soal aliran adalah soal *pemodelan*: bagian sulitnya adalah membangun jaringan [7]. Pecah simpul menjadi “masuk” dan “keluar” yang dihubungkan sisi berkapasitas k untuk membatasi berapa banyak yang melewatinya; tambahkan super-sumber dan super-muara untuk banyak sumber dan muara; pakai kapasitas 1 untuk “setiap butir paling banyak sekali”.
- **Baca potongan minimum dari BFS terakhir.** Setelah algoritma selesai, simpul yang dijangkau BFS terakhir membentuk sisi S dari potongan minimum. Soal yang menanyakan sisi *mana* yang dipotong atau pipa *mana* yang dilebarkan menginginkan tepat himpunan ini.
- **Periksa banyaknya sisi setelah pemodelan.** Model dengan 10^6 sisi memerlukan memori untuk $2 \cdot 10^6$ rekaman **Edge** berukuran 16 byte: 32 MB. Perkirakan sebelum menulis kode.

15.6.1 Contoh pemodelan yang dikerjakan

Tiga siswa, Adi, Bayu, dan Citra, harus ditugaskan ke tiga proyek P1, P2, dan P3, dan setiap proyek menerima paling banyak satu siswa. Adi dapat mengerjakan P1 atau P2, Bayu hanya P1, Citra P2 atau P3. Bangun jaringannya: sumber s dengan sisi berkapasitas 1 ke setiap siswa, sisi berkapasitas 1 dari setiap siswa ke setiap proyek yang dapat ia kerjakan, dan sisi berkapasitas 1 dari setiap proyek ke muara t . Aliran bernilai k berpadanan dengan k penugasan, karena setiap siswa mengirim paling banyak satu satuan dan setiap proyek menerima paling banyak satu.

Lintasan penambah pertama mungkin menugaskan Adi ke P1. Lalu Bayu tidak punya tujuan: satu-satunya proyeknya sudah terisi. Graf residu menyelamatkan kita. Lintasan $s \rightarrow \text{Bayu} \rightarrow \text{P1} \rightarrow \text{Adi} \rightarrow \text{P2} \rightarrow t$ memakai sisi residu mundur dari P1 ke Adi, yang *membatalkan* penugasan Adi ke P1 dan memindahkan Adi ke P2. Akhirnya Citra mengambil P3. Aliran maksimumnya 3: Adi–P2, Bayu–P1, Citra–P3. Inilah persis pembatalan yang digambarkan [Persamaan \(15.2\)](#), dan itulah sebabnya penugasan rakus gagal sementara aliran berhasil.

15.7 Jalan Anda Menuju Podium Nasional

Lima belas bab yang lalu buku ini menjanjikan sebuah jalan, bukan keajaiban. Medali nasional, entah emas, perak, atau perunggu, penghargaan *honorable mention*, atau tempat di antara para finalis, adalah buah latihan yang tekun dan terarah selama berbulan-bulan. Subbab ini mengubah semua yang telah Anda pelajari menjadi rencana yang dapat diikuti.

15.7.1 Apa yang diberikan kelima belas bab

Pandanglah kembali teknik-teknik itu sebagai kotak perkakas: pedoman hari lomba; pohon segmen untuk kueri rentang, dengan dan tanpa pembaruan; pohon segmen persisten untuk statistik urutan; tumpukan monoton; BFS ruang keadaan dan lintasan terpendek berbobot kecil; himpunan saling lepas, pohon perentang minimum, dan teknik sapuan; tur Euler dengan pohon Fenwick dan *binary lifting* pada pohon yang hidup; pemrograman dinamis untuk menghitung objek berbeda serta atas *bitmask* dan profil; divide and conquer CDQ; meet in the middle; larik sufiks; interpolasi dan aritmetika modular; dan aliran maksimum. Di atas semua itu, Anda memiliki sebuah *metode*: pahami soal dengan kata-kata sendiri, temukan modelnya, rancang, perkirakan biaya sebelum menulis kode, implementasikan dengan rapi, dan buktikan program itu benar dengan uji stres.

15.7.2 Rencana enam belas minggu

Tabel 15.2 adalah rencana untuk empat bulan terakhir menjelang babak nasional. Sesuaikan tanggalnya dengan kalender Anda; pertahankan proporsinya.

Tabel 15.2 Rencana latihan enam belas minggu menjelang babak nasional.

Minggu	Fokus	Rutinitas mingguan
1–4	Fondasi	10–15 soal mudah dan sedang; ketik ulang satu solusi buku per hari dari ingatan; uji stres setiap solusi
5–8	Keluasan	satu topik bab per minggu (pohon, DP, string, graf); 3 soal per topik dengan kesulitan menaik; satu soal nasional tahun lalu
9–12	Kedalaman	2 soal sulit per minggu, masing-masing diberi sekurang-kurangnya 90 menit sebelum membaca petunjuk apa pun; satu lomba virtual berwaktu per minggu
13–15	Simulasi	dua lomba virtual penuh per minggu dalam kondisi OSN: durasi sama, tanpa internet, hanya dokumentasi yang diizinkan
16	Pengenduran	latihan ringan, meninjau jurnal kesalahan dan templat Anda; tidur dan istirahat

15.7.3 Kebiasaan harian dan mingguan

- **Satu soal sulit sehari.** Pilih yang sedikit di atas kemampuan Anda. Berikan upaya sungguh-sungguh sebelum membaca solusi apa pun; setelah membacanya, tutup dan tulis kodenya sendiri.

- **Jurnal kesalahan.** Setiap *wrong answer*, batas waktu, atau galat saat jalan masuk ke buku catatan beserta penyebabnya: luapan, meleset satu, urutan panggilan salah, lupa mengatur ulang antarkasus uji, salah membaca soal. Baca ulang sebelum setiap lomba. Kotak-kotak jebakan dalam buku ini adalah awal jurnal itu.
- **Lomba virtual.** Juri daring seperti TOKI Learning Center (TLX) [8], Codeforces, dan AtCoder menyediakan lomba lampau yang dapat diikuti dengan pengatur waktu sungguhan. TLX juga memuat soal-soal olimpiade Indonesia terdahulu, latihan terbaik untuk gaya dan tingkat kesulitan OSN.
- **Templat di ujung jari.** Masukan cepat, pohon Fenwick, pohon segmen, DSU, Dijkstra, larik sufiks, perpangkatan modular, dan Dinic masing-masing semestinya dapat diketik dalam hitungan menit dan sudah teruji.
- **Upsolving.** Setelah setiap lomba, selesaikan soal-soal yang tidak berhasil Anda selesaikan selama lomba. Sebagian besar kemajuan Anda terjadi di sini.

15.7.4 Pada hari lomba

- **Baca semua soal lebih dahulu.** Gunakan menit-menit pertama untuk membaca semua tugas dan mencatat subtugasnya. Nilai dari solusi parsial terus bertambah: *brute force* yang meraih subtugas kecil sering bernilai lebih daripada solusi penuh yang tidak selesai.
- **Ikuti aturan 45 menit.** Bila suatu soal tidak mengalami kemajuan selama 45 menit, amankan nilai parsial yang bisa diraih dan beralihlah; kembali lagi nanti dengan pikiran segar.
- **Uji sebelum mengirim.** Contoh buatan sendiri, kasus tepi (N terkecil, semua nilai sama, nilai terbesar), dan, bila waktu mengizinkan, uji stres cepat seperti yang ada di buku ini.
- **Tetap tenang ketika orang lain tampak lebih cepat.** Anda bertanding melawan soal, bukan melawan orang-orang di sekitar Anda.

WAWASAN KUNCI: Gelar nasional adalah jumlah dari disiplin-disiplin kecil yang tidak pernah libur: satu soal diselesaikan dengan jujur, satu kesalahan dicatat, satu templat diuji. Podium dibangun dari hari-hari biasa.

Ingat pula bahwa jalannya panjang dan kemajuan jarang berupa garis lurus. Akan ada minggu-minggu ketika tidak ada yang tampak membaik, lalu satu minggu ketika beberapa gagasan tiba-tiba tersambung. Pertahankan rutinitas melewati minggu-minggu datar itu; saat itulah fondasinya mengendap. Berlatihlah bersama teman bila bisa: menjelaskan solusi kepada orang lain adalah salah satu cara tercepat menemukan celah dalam pemahaman sendiri, dan kelompok belajar kecil menjaga semua orang tetap bergerak pada hari-hari ketika semangat sedang rendah.

15.7.5 Daftar Periksa kesiapan

Sebelum babak nasional, periksa diri Anda dengan jujur terhadap **Tabel 15.3**. Setiap baris menggambarkan sesuatu yang dapat Anda *tunjukkan*, bukan sesuatu yang Anda *rasakan*.

Tabel 15.3 Daftar periksa kesiapan untuk minggu-minggu menjelang babak nasional.

Bidang	Bukti bahwa Anda siap
Implementasi	Anda dapat menulis Dinic, pohon segmen, dan larik sufiks dari ingatan, dan masing-masing terkompilasi bersih pada percobaan pertama
Verifikasi	Anda menulis <i>brute force</i> dan pembangkit untuk soal baru dalam kurang dari 15 menit
Pengelolaan waktu	Dalam lomba virtual Anda menyelesaikan soal-soal yang lebih mudah dengan cepat dan memakai aturan 45 menit
Pengetahuan	Anda mengenali pola-pola buku ini dalam pernyataan soal yang asing
Kestabilan	Empat lomba virtual terakhir Anda semuanya meraih nilai yang dekat dengan nilai terbaik Anda

15.8 Ringkasan Bab

- Aliran mematuhi kapasitas dan kekekalan ([Persamaan \(15.1\)](#)); maksimumnya sama dengan kapasitas potongan minimum.
- Kapasitas residu ([Persamaan \(15.2\)](#)) memungkinkan penambahan berikutnya membatalkan pilihan sebelumnya; sisi tak berarah adalah satu pasangan sisi, masing-masing dimulai dengan kapasitas penuh.
- Edmonds–Karp menambah sepanjang lintasan terpendek; Dinic menambahkan satu aliran pemblokir penuh per BFS dalam graf level, dan busur kini membuat setiap fase $O(VE)$, sehingga totalnya $O(V^2E)$ ([Persamaan \(15.3\)](#)).
- Untuk FASTFLOW: daftar ketetanggaan dengan indeks pasangan, gelang dibuang, sisi sejajar dipertahankan, dan **long long** di mana pun aliran berjalan.
- Potongan minimum atas semua himpunan bagian adalah *brute force* yang bebas untuk uji stres.
- Jalan menuju podium adalah sebuah rencana: enam belas minggu, satu soal sulit sehari, jurnal kesalahan, lomba virtual yang teratur, dan pemeriksaan diri yang jujur.

15.9 Soal Latihan

1. Jalankan Edmonds–Karp dengan tangan pada contoh FASTFLOW (perlakukan setiap sisi tak berarah sebagai dua sisi berarah berkapasitas penuh) dan tuliskan lintasan-lintasan penambahannya.
2. Pada contoh FASTFLOW, temukan potongan minimum dan periksa bahwa kapasitasnya 5.
3. Mengapa busur kini harus tetap berada di suatu sisi setelah dorongan yang berhasil, alih-alih bergerak melewatinya?
4. Tunjukkan cara menemukan pemasangan maksimum dalam graf bipartit (siswa dan proyek, setiap siswa paling banyak satu proyek, setiap proyek paling banyak satu siswa) dengan aliran maksimum.
5. Setelah **maxFlow** selesai, bagaimana Anda mencetak simpul di sisi sumber dari suatu potongan minimum?
6. Suatu soal membatasi berapa banyak aliran yang boleh melewati setiap simpul v sebesar k_v . Bagaimana Anda memodelkannya?
7. Mengapa FASTFLOW tidak memerlukan sisi pasangan terpisah untuk setiap arah sisi tak berarah, sedangkan sisi berarah selalu mendapat pasangan berkapasitas 0?

15.10 Penyelesaian Soal Latihan

1. Lintasan terpendek lebih dahulu: $1 \rightarrow 3 \rightarrow 4$ dengan kemacetan $\min(2,3) = 2$. Lalu $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ dengan kemacetan $\min(3,4,1+3)$; setelah kedua sisi 3-4 digabung menjadi satu kapasitas 6, dengan 2 sudah terpakai, kemacetannya $\min(3,4,4) = 3$. Totalnya 5, dan setelah itu simpul 1 tidak memiliki kapasitas residu lagi.
2. $S = \{1\}$: sisi 1-2 (3) dan 1-3 (2) menyeberang, totalnya 5. Aliran sebesar 5 menunjukkan tidak ada potongan yang lebih kecil.
3. Dorongan yang berhasil dapat menyisakan kapasitas pada sisi itu (kemacetannya mungkin berada lebih jauh di lintasan). Seandainya busur bergerak melewatinya, panggilan DFS berikutnya dalam fase yang sama akan melewatkan kapasitas yang masih dapat dipakai, dan fase itu berakhir sebelum alirannya memblokir. Hasil keseluruhannya tetap benar (BFS berikutnya akan menemukannya), tetapi dapat memakan fase tambahan.
4. Tambahkan sumber yang dihubungkan ke setiap siswa dengan kapasitas 1, sisi berkapasitas 1 dari setiap siswa ke setiap proyek yang dapat diterimanya, dan sisi berkapasitas 1 dari setiap proyek ke muara. Aliran maksimumnya sama dengan ukuran pemasangan maksimum, dan sisi dari siswa ke proyek yang membawa aliran membentuk pemasangan itu.
5. Jalankan BFS pada `bfs` sekali lagi pada graf residu terakhir (atau pakai ulang larik `level` terakhir, yang dihitung oleh BFS yang gagal). Simpul dengan `level[v] >= 0` terjangkau dari sumber dalam graf residu dan membentuk himpunan S dari potongan minimum.
6. Pecah setiap simpul v menjadi v_{in} dan v_{out} dengan sisi $v_{\text{in}} \rightarrow v_{\text{out}}$ berkapasitas k_v . Setiap sisi yang dulu masuk ke v kini masuk ke v_{in} , dan setiap sisi yang dulu keluar dari v kini keluar dari v_{out} .
7. Sisi berarah $u \rightarrow v$ memerlukan sisi residu mundur agar aliran dapat dibatalkan; sisi mundur itu dimulai dari 0 karena belum ada yang dapat dibatalkan. Untuk sisi tak berarah, arah mundurnya sendiri merupakan sisi sungguhan berkapasitas c , sehingga kedua arah dapat saling menjadi pasangan: mendorong d dari u ke v menurunkan $u \rightarrow v$ menjadi $c - d$ dan menaikkan $v \rightarrow u$ menjadi $c + d$, yang tepat merupakan kapasitas sisi sungguhan ditambah aliran yang dapat dibatalkan. Satu pasangan sisi alih-alih dua membagi dua memori dan pekerjaan DFS.

Daftar Pustaka

- [1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
- [2] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.
- [3] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM*, vol. 19, no. 2, pp. 248–264, 1972.
- [4] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," *Soviet Mathematics Doklady*, vol. 11, pp. 1277–1280, 1970.
- [5] S. Even and R. E. Tarjan, "Network flow and testing graph connectivity," *SIAM Journal on Computing*, vol. 4, no. 4, pp. 507–518, 1975.
- [6] Sphere Online Judge, "FASTFLOW - Fast Maximum Flow." [Online]. Available: <https://www.spoj.com/problems/FASTFLOW/>
- [7] S. Halim, F. Halim, and S. Effendy, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, Book 2. Lulu Press, 2020.

- [8] Tim Olimpiade Komputer Indonesia, “TLX — TOKI Learning Center.” [Online]. Available: <https://tlx.toki.id/>