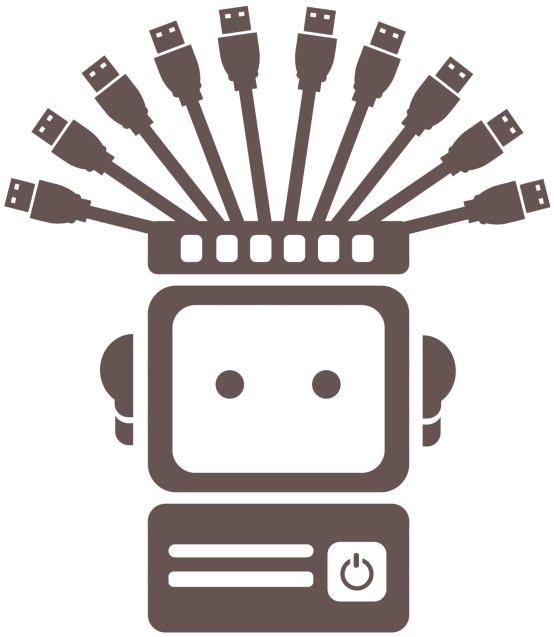
BUKLET PEMBAHASAN SOAL



GEMASTIK 10
2017

PENYISIHAN PEMROGRAMAN GEMASTIK 10

30 September 2017

Soal dan Penulis Soal

Kode	Judul	Penulis
A	Berbalas Pantun	Alham Fikri Aji
B	Fotografer Wisuda	Ashar Fuadi
C	Penulis Soal	Ashar Fuadi
D	Saklar Lhompat II	Anthony
E	Pasangan Terbaik	Alham Fikri Aji
F	Rubrik Petakata	Ashar Fuadi

Penguji

Ammar Fathin Sabili

Statistik Penyisihan

Catatan: Statistik dibuat berdasarkan kondisi sebelum tabel peringkat dibekukan.

- Penyisihan Pemrograman GEMASTIK 10 diikuti oleh 316 tim, dengan 233 tim setidaknya mengirimkan satu pengumpulan jawaban.
- Statistik banyaknya soal yang diselesaikan oleh seluruh tim:

# *Accepted*	0	1	2	3	4	5	6	7	8	9	10	11	12
# Tim	31	67	39	60	11	9	2	3	4	3	2	0	2

- Statistik per soal:

	A		**B**		**C**		**D**		**E**		**F**	
	1	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**
# *Accepted*	199	121	113	28	25	14	15	7	18	5	3	2
# Pengiriman Solusi	384	653	459	209	262	56	53	24	335	41	9	4

  Pertama untuk mendapatkan *Accepted*:
  - Soal A1:
    - TMCLL (Institut Teknologi Sepuluh Nopember)
    - pada menit ke-2
  - Soal A2:
    - gak ada pencerahan (Universitas Indonesia)
    - pada menit ke-3
  - Soal B1:
    - Fata Ganteng (Universitas Indonesia)
    - pada menit ke-10
  - Soal B2:
    - Fata Ganteng (Universitas Indonesia)
    - pada menit ke-22
  - Soal C1:
    - Ainge ST (Institut Teknologi Bandung)
    - pada menit ke-16
  - Soal C2:
    - Ainge ST (Institut Teknologi Bandung)
    - pada menit ke-16
  - Soal D1:
    - collateral_damage (Universitas Bina Nusantara)
    - pada menit ke-33
  - Soal D2:
    - gak ada pencerahan (Universitas Indonesia)
    - pada menit ke-61
  - Soal E1:
    - gak ada pencerahan (Universitas Indonesia)
    - pada menit ke-15
  - Soal E2:
    - gak ada pencerahan (Universitas Indonesia)
    - pada menit ke-26
  - Soal F1:
    - gak ada pencerahan (Universitas Indonesia)
    - pada menit ke-83
  - Soal F2:
    - gak ada pencerahan (Universitas Indonesia)
    - pada menit ke-83

Berbalas Pantun

Penulis soal : Alham Fikri Aji
Penulis editorial : Anthony
Tema : *ad hoc*

Catatan/Komentar

Soal ini merupakan soal penyisihan termudah karena tidak dibutuhkan pengetahuan khusus untuk menyelesaikannya.

Batasan kedua versi: $1 \leq A[i], B[i] \leq 100$

Versi Mudah

Batasan: $\mathbf{N = 1}$

Untuk versi mudah dari soal ini, karena $N = 1$, maka jawabannya adalah $A[1] + B[1]$.

Versi Sulit

Batasan: $\mathbf{1 \leq N \leq 100.000}$

Berikut adalah jumlahan yang diminta pada soal:

$$\begin{gathered}(A[1] + B[1]) + (A[1] + B[2]) + (A[1] + B[3]) + \cdots + (A[1] + B[N]) \\ +(A[2] + B[1]) + (A[2] + B[2]) + (A[2] + B[3]) + \cdots + (A[2] + B[N]) \\ + \ldots \\ +(A[N] + B[1]) + (A[N] + B[2]) + (A[N] + B[3]) + \cdots + (A[N] + B[N])\end{gathered}$$

Setiap elemen dari A akan dipasangkan dengan setiap elemen dari B, sehingga jumlahan untuk sebuah $A[i]$ dapat dituliskan sebagai berikut:

$$\begin{aligned}(A[i] + B[1]) + (A[i] + B[2]) + \cdots + (A[i] + B[N]) &= (A[i] \times N) + (B[1] + B[2] + \cdots + B[N]) \\ &= (A[i] \times N) + \sum_{j=1}^{N} B[j]\end{aligned}$$

Jawaban akhir diperoleh dengan menjumlahkan persamaan di atas untuk setiap i dari 1 hingga N:

$$\begin{aligned}\sum_{i=1}^{N} \left((A[i] \times N) + \sum_{j=1}^{N} B[j] \right) &= N \times \left(\sum_{j=1}^{N} B[j] \right) + \sum_{i=1}^{N} (A[i] \times N) \\ &= N \times \left(\sum_{j=1}^{N} B[j] \right) + N \times \left(\sum_{i=1}^{N} A[i] \right) \\ &= N \times \left(\sum_{i=1}^{N} A[i] + \sum_{j=1}^{N} B[j] \right)\end{aligned}$$

Rumus jawaban akhir menjadi sangat sederhana, yakni: N dikalikan dengan jumlahan semua elemen-elemen A dan B.

Fotografer Wisuda

Penulis soal : Ashar Fuadi
Penulis editorial : Anthony
Tema : algoritma konstruktif

Catatan/Komentar

Ide untuk menyelesaikan soal ini tidak sulit tetapi cukup rawan terjadi kesalahan pada implementasinya.

Batasan kedua versi: $1 \leq K \leq B \leq 1.000$

Versi Mudah

Batasan: $\mathbf{0 \leq A \leq 1}$

Untuk menyelesaikan versi mudah dari soal ini, penyelesaian untuk seluruh kemungkinan masukan dapat dikategorikan sebagai berikut:

- Jika tidak ada teman laki-laki ($A = 0$):
  - jika banyaknya teman perempuan melebihi K ($B > K$), jawabannya adalah `mustahil`,
  - jika tidak, cetak `P` sebanyak B kali.
- Jika terdapat tepat 1 teman laki-laki ($A = 1$):
  - jika banyaknya teman perempuan melebihi $2K$ ($B > 2K$), jawabannya adalah `mustahil`,
  - selain itu, cetak `P` sebanyak $\min(B, K)$, kemudian `L`, kemudian `P` sebanyak $B - \min(B, K)$.

Versi Sulit

Batasan: $\mathbf{0 \leq A \leq 1.000}$

Hal pertama yang dapat dilakukan adalah pemeriksaan kasus `mustahil`, yaitu jika tidak memungkinkan untuk menempatkan teman laki-laki sehingga paling banyak dua teman laki-laki bersebelahan untuk setiap deretan teman laki-laki. Dengan kata lain, kasus `mustahil` hanya terjadi pada masukan yang memenuhi:

- $A > 2(B + 1)$ atau
- $A < \lceil \frac{B}{K} \rceil - 1$.

Perhatikan bahwa $\lceil \frac{B}{K} \rceil - 1$ dan $2(B+1)$ berturut-turut merupakan batas bawah dan batas atas banyaknya deretan teman perempuan (`P`) yang dipisahkan oleh deretan teman laki-laki (`L`).

Untuk masukan yang tidak `mustahil`, solusi dapat dibangun dengan cara menemukan pola "penyelipan" A buah `L` ke dalam deretan yang terdiri dari B buah `P` sedemikian rupa sehingga memenuhi persyaratan yang diberikan pada soal. Sebagai contoh, pola yang dapat dibentuk adalah `P...PLLP...PLLP...PLP...PLP...P`.

Penulis Soal

Penulis soal : Ashar Fuadi
Penulis editorial : Alham Fikri Aji
Tema : *bipartite matching, maximum flow*

Versi Mudah

Batasan: $\mathbf{1 \leq N \leq 10; 1 \leq S[i] \leq 2; 0 \leq P[i] \leq 2}$

Kita asumsikan kondisi terburuk dengan setiap penulis memiliki afiliasi terhadap 2 perusahaan. Untuk setiap penulis, terdapat 2 kasus yaitu:

- $S[i] = 1$. Untuk kasus ini, penulis ke-i dapat menggunakan soal tersebut untuk:
  - Dirinya sendiri
  - Perusahaan pertama si penulis
  - Perusahaan kedua si penulis
- $S[i] = 2$. Untuk kasus ini, penulis ke-i dapat menggunakan soal tersebut untuk:
  - Dirinya sendiri + Perusahaan pertama si penulis
  - Dirinya sendiri + Perusahaan kedua si penulis
  - Perusahaan pertama dan kedua si penulis

Dapat dilihat bahwa penulis memiliki paling banyak 3 pilihan. Karena jumlah penulis paling banyak adalah 10, maka semua pilihan dapat disimulasikan, dengan kompleksitas akhir $O(3^N)$.

Solusi di atas sudah cukup untuk mendapatkan nilai penuh di versi mudah. Namun, jika dianalisis lebih lanjut, solusi optimal selalu dicapai jika setiap penulis menggunakan salah satu soalnya atas dirinya sendiri. Langkah ini selalu optimal karena hanya penulis ke-i yang dapat berafiliasi dengan dirinya sendiri, sehingga tidak mungkin terjadi bentrok. Dengan memanfaatkan strategi tersebut, kompleksitas dapat ditekan menjadi $O(2^N)$.

Versi Sulit

Batasan: $\mathbf{1 \leq N \leq 50; 1 \leq S[i] \leq 50; 0 \leq P[i] \leq 50}$

Solusi optimal selalu dicapai jika setiap penulis menggunakan salah satu soalnya atas dirinya sendiri. Dengan demikian, kita hanya perlu memasangkan paling banyak 49 soal dari setiap penulis ke 50 perusahaan yang ada. Relasi ini dapat direpresentasikan dalam bentuk graf, yang mana terdapat edge dari node i dan $(j + 50)$ jika penulis ke i bekerja di perusahaan ke-j. Ternyata, graph yang dihasilkan akan selalu bipartite, sehingga kita dapat melakukan algoritma bipartite matching. Namun, setiap penulis bisa menuliskan lebih dari 1 soal, sehingga kita harus menduplikasi node i sebanyak $S[i] - 1$ kali.

Alternatif lain, kita juga dapat membuat sebuah node baru *source* dengan terdapat edge dari node *source* ke i dengan kapasitas $S[i] - 1$. Setelah itu, kita buat juga node *sink* dengan terdapat edge dari $(j + 50)$ ke *sink* dengan kapasitas 1. Untuk mendapatkan solusi, kita cukup menjalankan algoritma max-flow dari *source* ke *sink*.

Saklar Lhompat II

Penulis soal : Anthony
Penulis editorial : Ashar Fuadi
Tema : *shortest path*, *dynamic programming*

Catatan/Komentar

Latar belakang cerita dari soal ini merupakan sekuel dari soal Saklar Lhompat pada Gemastik 9 tahun 2016.

Batasan kedua versi:

- $2 \leq R \leq 50$
- $1 \leq C \leq 50$
- $-N \leq G[i][j] \leq 100.000$
- $G[i][j] = 0$
- G berisi semua bilangan bulat antara $-N$ hingga -1, masing-masing tepat sekali

Versi Mudah

Batasan: $\mathbf{N = 1}$

Karena hanya terdapat tepat satu orang mahasiswa, maka:

- Terdapat tepat C konfigurasi (yang semuanya pasti teratur): sambungkan laptop tersebut ke meja ke-i di baris pertama, untuk setiap $1 \leq i \leq C$.
- Untuk setiap konfigurasi, tingkat kesemrawutan = tingkat efisiensi penyambungan = panjang terpendek kabel untuk penyambungan tersebut.
- Dengan kata lain, jawabannya adalah jumlah dari jarak terpendek dari laptop ke setiap meja pada baris pertama.

Untuk menghitung jarak terpendek laptop ke semua meja pada baris pertama, dapat digunakan algoritma Dijkstra (satu kali saja, dengan meja satu-satunya mahasiswa sebagai *source*). Kompleksitasnya adalah $\mathcal{O}(RC \log RC)$.

Versi Sulit

Batasan: $\mathbf{1 \leq N \leq C}$

Pertama-tama, mari kita hitung, ada berapa banyak konfigurasi teratur yang mungkin?

Misalkan kita sudah menetapkan N meja pada baris pertama untuk disambungkan ke laptop-laptop mahasiswa. Dalam konfigurasi yang teratur, pastilah meja terkiri dari N meja tersebut adalah untuk mahasiswa 1, meja berikutnya untuk mahasiswa 2, dan seterusnya sampai meja terkanan dari N meja tersebut untuk mahasiswa N. Yakni, untuk setiap pemilihan N meja, terdapat tepat 1 cara untuk memasang-masangkannya dengan N laptop mahasiswa. Oleh karena itu, banyaknya konfigurasi teratur sama dengan banyaknya cara memilih N meja dari C meja pada baris pertama, atau dinotasikan dengan $\binom{C}{N}$.

Pada kasus terburuk, dengan $N = 50, C = 25$, banyaknya konfigurasi teratur adalah $\binom{50}{25}$, yang merupakan bilangan yang sangat besar sehingga tidak bisa kita iterasi satu-persatu secara *brute-force*. Mari kita selesaikan dengan cara lain. Soal-soal kombinatorika semacam ini cocok diselesaikan dengan metode *dynamic programming*.

Kita definisikan $dp(c, n, e)$ sebagai total tingkat kesemrawutan, apabila terdapat c meja terkiri baris pertama, n mahasiswa pertama, dan tingkat efisiensi penyambungan terbesar saat ini adalah e. Nilainya adalah salah satu dari:

- e, jika $n = 0$ (semua laptop sudah disambungkan)

- ∞, jika $c = 0$ dan $n > 0$ (terdapat laptop yang tidak disambungkan, namun meja habis)
- $dp(c-1, n, e) + dp(c-1, n-1, max(e, dist(c, n)))$, selainnya (terdapat dua pilihan: sambungkan meja ke-c baris pertama dengan laptop mahasiswa ke-n, atau tidak. $dist(c, n)$ menyatakan jarak terpendek dari meja ke-c baris pertama ke laptop mahasiswa ke-n)

Jawaban akhirnya adalah $dp(C, N, 0)$.

Solusi di atas tidak dapat diimplementasikan secara naif karena dimensi ketiga bisa sangat besar (tinggi suatu meja bisa mencapai 100.000). Namun, perhatikan bahwa sebenarnya hanya terdapat paling banyak $\mathcal{O}(NC)$ nilai berbeda untuk dimensi ketiga ini, karena dimensi ketiga pasti merupakan tingkat efisiensi penyambungan salah satu laptop ke salah satu meja baris pertama. Dengan implementasi yang tepat, kompleksitas akhir solusi dapat ditekan menjadi (kompleksitas penghitungan $dist$ + kompleksitas penghitungan dp) = $\mathcal{O}(NRC \log RC + C^2N^2)$.

Pasangan Terbaik

Penulis soal : Alham Fikri Aji
Penulis editorial : William Gozali
Tema : matematika, heuristik

Catatan/Komentar

Berikut adalah rumus yang diberikan pada soal.

$$f(i,j) = A[i] \times B[j] + C[(A[i] \times B[j]) \bmod M]$$

Bagian penjumlahan dengan suatu elemen pada array C merupakan bagian yang paling sulit dikendalikan. Sebab elemen C yang dipilih memiliki pola yang sulit ditebak. Solusi yang digunakan akan fokus pada menganalisis bagian tersebut.

Untuk kemudahan, definisikan nilai terbesar yang mungkin untuk elemen array adalah $\mathcal{Z}$, dengan $\mathcal{Z} = 1.000.000$.

Batasan kedua versi: $1 \leq N \leq 100.000; 0 \leq A[i], B[i], C[i] \leq 1.000.000$

Versi Mudah

Batasan: $\mathbf{1 \leq M \leq 1.000}$

Perhatikan rumus indeks array C yang akan dipilih. Berdasarkan sifat modulo, diketahui:

$$(A[i] \times B[j]) \bmod M = ((A[i] \bmod M) \times (B[j] \bmod M)) \bmod M$$

Berapapun nilai $A[i]$, yang menentukan indeks array C yang digunakan adalah nilai modulonya. Artinya ketika terdapat beberapa elemen $A[i]$ yang memiliki nilai $(A[i] \bmod M)$ sama, sudah pasti lebih menguntungkan untuk hanya menggunakan nilai $A[i]$ yang terkecil. Berdasarkan observasi ini, kita dapat mengurangi banyaknya elemen pada array A menjadi paling banyak M elemen. Terapkan pula hal serupa untuk array B.

Kini array A dan array B memiliki paling banyak M elemen. Coba semua kemungkinan pasangan dan cari yang menghasilkan nilai f terkecil. Kompleksitas solusi $\mathcal{O}(M^2)$.

Versi Sulit

Batasan: $\mathbf{1 \leq M \leq 100.000}$

Untuk nilai elemen A dan B yang cukup besar, perkalian antar suatu elemen array A dengan array B memiliki kontribusi kepada nilai f yang jauh lebih besar daripada bagian penjumlahan dengan suatu elemen array C. Artinya nilai elemen C manapun yang dipilih kurang signifikan pada nilai akhir f.

Dengan sifat tersebut dan perilaku pemilihan elemen C yang polanya sulit ditebak, kita dapat menganggap penjumlahan dengan suatu elemen C sebagai suatu *error* atau *noise*. Tuliskan rumus f menjadi:

$$f(i,j) = A[i] \times B[j] + err$$

Dengan err adalah suatu nilai yang berkisar antara 0 sampai dengan $\mathcal{Z}$.

Sekarang misalkan kita memilih suatu $i = x$. Kita dapat mencoba seluruh kemungkinan nilai elemen B, dari yang paling kecil hingga suatu elemen yang cukup besar sehingga err menjadi tidak lagi signifikan.

Misalkan B_{min} adalah elemen terkecil pada array B. Misalkan juga C_{min} dan C_{max} menyatakan elemen

terkecil dan terbesar pada array C. Kita harus mencoba seluruh elemen $B[j]$ yang memenuhi:

$$\begin{aligned} A[x] \times B[j] + C_{min} &< A[x] \times B_{min} + C_{max} \\ A[x] \times B[j] - A[x] \times B_{min} &< C_{max} - C_{min} \\ A[x](B[j] - B_{min}) &< 1.000.000 \\ B[j] - B_{min} &< \frac{1.000.000}{A[x]} \end{aligned} \tag{1}$$

Pada awalnya, $A[x] \times B_{min} + err$ merupakan kandidat solusi. Seluruh $B[j]$ yang tidak memenuhi pertidaksamaan 1 tidak mungkin menjadi kandidat solusi, karena sudah pasti lebih besar daripada $A[x] \times B_{min} + err$, tanpa peduli berapapun nilai err.

Dengan asumsi array A dan array B memiliki nilai yang unik, banyaknya $B[j]$ unik yang perlu diperiksa untuk suatu $A[x]$ dibatasi oleh $O\left(\frac{1.000.000}{A[x]}\right)$. Banyaknya pasangan yang perlu diperiksa dapat diperkirakan dengan cara berikut:

$$\begin{aligned} \sum_{i=0}^{|A|-1} \frac{1.000.000}{A[i]} &= 1.000.000 \sum_{i=0}^{|A|-1} \frac{1}{A[i]} \\ &\leq 1.000.000 \sum_{i=0}^{\mathcal{Z}} \frac{1}{i} && (2) \\ &\leq 1.000.000(\ln \mathcal{Z} + 1) && (3) \end{aligned}$$

Pada kenyataannya, $|A|$ terbesar adalah 100.000, yaitu $\frac{1}{10}$ dari $\mathcal{Z}$. Jadi pertidaksamaan yang ditunjukkan pada 2 merupakan estimasi yang berlebihan. Sebagai catatan, perpindahan dari 2 ke 3 memanfaatkan sifat deret harmonik.

Sebagai kesimpulan, solusi yang digunakan adalah:

1. Urutkan array B dari kecil ke besar, lalu buang elemen-elemen duplikat sehingga didapatkan array yang unik.
2. Coba seluruh kemungkinan $A[i]$.
3. Untuk suatu $A[i]$ yang dipilih, coba seluruh elemen B mulai dari yang terkecil sampai ke elemen terbesar yang masih memenuhi pertidaksamaan 1.
4. Cetak yang menghasilkan nilai f terkecil.

Estimasi kasar berlebihan untuk kompleksitas solusi ini adalah $\mathcal{O}(\mathcal{Z} \ln \mathcal{Z})$.

Rubrik Petakata

Penulis soal : Ashar Fuadi
Penulis editorial : Pusaka Kaleb Setyabudi
Tema : *dynamic programming*, *string matching*

Catatan/Komentar

Untuk dapat mendapatkan *Accepted* pada versi manapun pada soal ini, dibutuhkan pengetahuan mengenai *dynamic programming*. Lebih lanjut, untuk mendapatkan *Accepted* pada versi sulit, dibutuhkan pengetahuan mengenai algoritma Knuth-Morris-Pratt (KMP) dan representasi algoritma KMP dalam *deterministic finite state machine* (DFSM) (atau bisa juga tidak; lihat bagian terakhir pada pembahasan soal ini).

Beberapa notasi dan kesepakatan yang digunakan:

- A_K, menyatakan himpunan K karakter pertama dalam alfabet. Contoh: $A_6 =$ {'a', 'b', 'c', 'd', 'e', 'f'}
- S_i untuk $1 \leq i \leq |S|$, menyatakan karakter ke-i dari S
- petak-petak rubrik petakata memiliki ukuran 2 baris ke bawah dan N baris ke kanan. Dari kiri ke kanan, setiap kolom dinomori dari 1 hingga N.

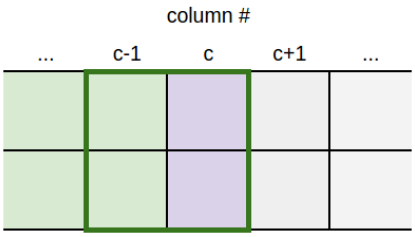
Pra-Versi Mudah

Bagaimana solusi naif yang dapat menyelesaikan soal ini?

Salah satu solusi naif untuk soal ini adalah dengan menghasilkan setiap K^{2N} konfigurasi petak yang mungkin, lalu melakukan verifikasi mengenai ada-tidaknya S dalam masing-masing konfigurasi petak. Solusi ini memiliki kompleksitas $\mathcal{O}(K^{2N} \cdot N)$ yang tentu tidak dapat lolos versi mudah soal ini.

Bagaimana cara memperkecil kompleksitas solusi di atas?

Solusi tersebut dapat dioptimisasi dengan cara melakukan verifikasi sambil menghasilkan ke-K^{2N} konfigurasi petak yang ada. Untuk dapat melakukan hal tersebut, pengisian konfigurasi petak dilakukan secara kolom-per-kolom dari kolom pertama hingga terakhir: Untuk setiap pengisian suatu kolom $c \in [1, N]$ dengan dua karakter $a, b \in A_K$, cek apakah a, b serta dua karakter pada kolom sebelumnya (kolom $c-1$, jika ada) dapat membentuk S. Jika ya, maka sisa $(N - c)$ kolom di kanan kolom ke-c dapat diisi dengan karakter-karakter sembarang. Perhatikan bahwa dalam hal ini, kita dapat berhenti mengisi kolom-kolom selanjutnya dan menganggap telah menemukan $K^{2(N-c)}$ konfigurasi. Jika tidak, maka pengisian kolom dilanjutkan untuk kolom ke-$(c + 1)$. Optimisasi ini memiliki kompleksitas $\mathcal{O}(K^{2N})$.



Gambar 1: Jika S dapat ditemukan pada petak-petak dalam kotak hijau, maka pengisian $N - c$ kolom sisanya dapat diabaikan (daerah abu-abu).

Perhatikan bahwa pada optimisasi ini, untuk mengisi petak-petak pada suatu kolom c dibutuhkan pula informasi mengenai karakter-karakter pada kolom sebelumnya (kolom $c - 1$) untuk dapat membantu menentukan dibentuk-tidaknya S.

Solusi ini akan dioptimisasi lebih lanjut dengan *dynamic programming* pada subbagian editorial berikutnya.

Batasan kedua versi: $1 \leq N \leq 50; 1 \leq K \leq 26$

Versi Mudah

Batasan: $|\mathbf{S}| = \mathbf{2}$

Dari penjelasan pada subbagian sebelumnya, perhatikan bahwa suatu *state*/keadaan pengisian petak-petak rubrik petakata dapat direpresentasikan dengan komponen-komponen berikut.

- c, yang menyatakan posisi kolom yang sedang/akan diisi, dan
- (k'_1, k'_2), yaitu karakter-karakter pengisi kolom ke-$(c-1)$.

Dengan menerapkan *dynamic programming* (memoisasi terhadap *state* dan hasil) pada *state* di atas dan menggunakan transisi yang serupa pada solusi pada subbagian Pra-Versi Mudah, maka kompleksitas solusi yang didapat adalah $\mathcal{O}(NK^2 \cdot K^2) = \mathcal{O}(NK^4)$. Untuk lebih jelasnya, berikut relasi rekurensi dari *dynamic programming* yang diterapkan:

$$\mathrm{dp}(c, k'_1, k'_2) = \begin{cases} 0, & c > N \\ \displaystyle\sum_{\substack{a,b \in A_K \\ \mathrm{canMakeS}(k'_1,k'_2,a,b)}} K^{2(N-c)} + \sum_{\substack{a,b \in A_K \\ \neg\mathrm{canMakeS}(k'_1,k'_2,a,b)}} \mathrm{dp}(c+1, a, b), & c \leq N \end{cases}$$

dengan $\mathrm{canMakeS}(p, q, r, s)$ bernilai true jika salah satu dari pr, ps, qr, qs, rs, atau sr membentuk S.

Jawaban terdapat pada $\mathrm{dp}(1, -, -)$ dengan $-$ adalah karakter yang tidak terdapat dalam A_K. Karakter $-$ dipilih dikarenakan pada *state* pada kolom pertama, tidak terdapat karakter yang sebelumnya telah terpasang (tidak terdapat karakter pada kolom ke-0).

Versi Sulit

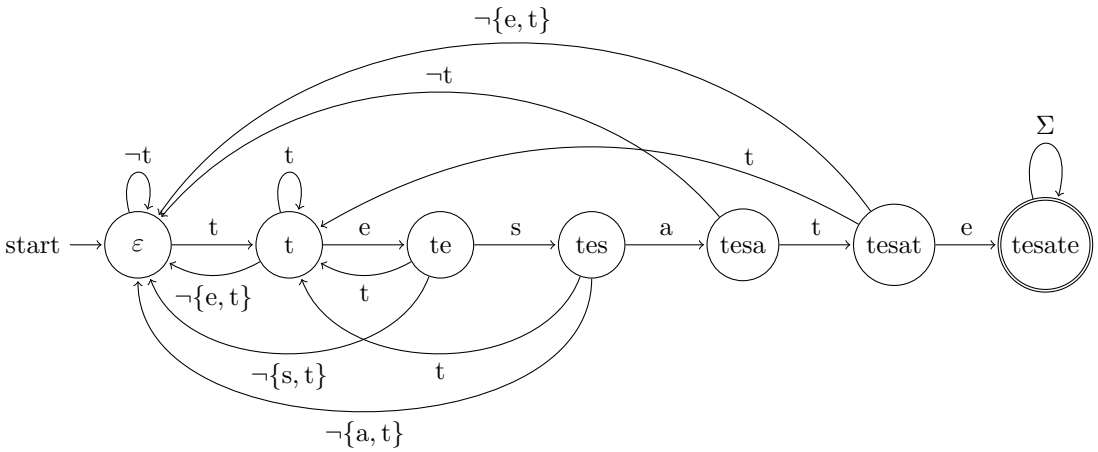
Batasan: $\mathbf{1} \leq |\mathbf{S}| \leq \mathbf{10}$

Solusi untuk versi mudah tidak dapat secara langsung diperumum/diterapkan untuk menyelesaikan versi sulit karena terdapat batasan bahwa untuk suatu masukan S, dalam pengisian petak-petak pada kolom c dibutuhkan informasi mengenai karakter-karakter yang mengisi kolom-kolom $c - |S| + 1$ hingga $c - 1$, yang mengakibatkan kompleksitas solusi menjadi $\mathcal{O}(NK^{2|S|} \cdot K^2) = \mathcal{O}(NK^{2(|S|+1)})$. Untuk versi ini, $|S|$ dapat mencapai 10; penerapan solusi versi mudah secara langsung memiliki kompleksitas $\mathcal{O}(NK^{22})$ yang akan mendapatkan penilaian *Time Limit Exceeded* pada versi sulit.

Algoritma KMP

Algoritma Knuth-Morris-Pratt (KMP) adalah salah satu algoritma pencarian *string*. Silakan merujuk pada literatur terkait untuk mengetahui lebih lanjut mengenai algoritma tersebut.

Himpunan *state* dalam suatu eksekusi algoritma KMP untuk mencari suatu string S dalam suatu teks T dapat direpresentasikan dalam suatu *deterministic finite state machine* (DFSM). Sebagai contoh, DFSM dalam eksekusi algoritma KMP untuk mencari $S =$ `tesate` adalah sebagai berikut.



Perhatikan bahwa untuk suatu string S, maka DFSM yang terbentuk akan memiliki tepat $|S| + 1$ *state*. Untuk DFSM di atas, string S dikatakan ditemukan dalam teks T apabila *state* pada akhir eksekusi algoritma KMP adalah pada *state* akhir (*state* dengan dua garis pinggir).

Sebagai kesepakatan, berikut beberapa notasi terkait DFSM.

- Q, menyatakan himpunan *state* dalam DFSM.
- Σ, menyatakan himpunan karakter/masukan dalam DFSM.
- $\delta : Q \times \Sigma \to Q$, menyatakan transisi dalam DFSM.
- q_0, menyatakan *state* awal/mulai.
- A, menyatakan himpunan *state* akhir/selesai.

Dalam contoh DFSM untuk $S =$ `tesate` di atas, maka

- $Q = \{q_\epsilon, q_t, q_{te}, q_{tes}, q_{tesa}, q_{tesat}, q_{tesate}\}$,
- $\Sigma = \{a, b, c, \ldots\}$
- $\delta = \{((q_\epsilon, \neg t), q_\epsilon), ((q_\epsilon, t), q_t), ((q_t, t), q_t), ((q_t, e), q_{te}), \ldots\}$
- $q_0 = q_\epsilon$
- $A = \{q_{tesate}\}$.

Solusi untuk Petak Berukuran $1 \times N$

Jika soal yang diberikan tetap sama namun petak yang diberikan berukuran $1 \times N$, maka salah satu solusi yang dapat menyelesaikan varian ini adalah dengan terlebih dahulu membangun DFSM untuk string S dan menggunakan *dynamic programming* dengan komponen-komponen *state*

- c, yang menyatakan posisi kolom yang sedang/akan diisi, dan
- q, *state* DFSM yang dicapai pada pengisian kolom ke-$(c-1)$ setelah mengisi ke-$(c-1)$ kolom sebelumnya.

serta dengan relasi rekurensi

$$\mathrm{dp}(c, q) = \begin{cases} 0, & c > N + 1 \\ 0, & c = N + 1 \wedge q \notin A \\ 1, & c = N + 1 \wedge q \in A \\ \displaystyle\sum_{k \in A_K} \mathrm{dp}(c + 1, \delta(q, k)), & c \leq N. \end{cases}$$

Perhatikan bahwa *base case* yang memiliki nilai kembalian 1 terjadi pada kasus dengan $c = N+1 \wedge q \in A$, yaitu saat seluruh petak telah terisi karakter dan terdapat string S pada petak-petak yang telah terisi.

Jawaban terdapat pada $\mathrm{dp}(1, q_0)$. Solusi ini memiliki kompleksitas $\mathcal{O}$(pembangunan DFSM untuk S) + $\mathcal{O}$(eksekusi solusi dengan DP) = $\mathcal{O}(|S|) + \mathcal{O}(NK^2) = \mathcal{O}(|S| + NK^2)$.

Solusi untuk Petak Berukuran $2 \times N$ (Soal Versi Sulit)

Solusi untuk varian petak berukuran $1 \times N$ di atas dapat dikembangkan menjadi solusi untuk soal versi sulit dengan melakukan modifikasi terhadap komponen-komponen *state* dari *dynamic programming* yaitu

- c, menyatakan posisi kolom yang sedang/akan diisi, dan
- V, menyatakan himpunan *state* DFSM yang mungkin dicapai jika telah melakukan pengisian pada ke-$(c-1)$ kolom sebelumnya dan berakhir pada posisi baris manapun pada kolom ke-$(c-1)$

dan dengan relasi rekurensi sebagai berikut.

$$\mathrm{dp}(c,V) = \begin{cases} 0, & c > N+1 \\ 0, & c = N+1 \wedge A \cap V = \varnothing \\ 1, & c = N+1 \wedge A \cap V \neq \varnothing \\ \displaystyle\sum_{a,b \in A_K} \mathrm{dp}\left(c+1, \bigcup_{q \in V} \left(\left\{\delta\left(q,a\right), \delta\left(q,b\right), \delta\left(\delta\left(q,a\right),b\right), \delta\left(\delta\left(q,b\right),a\right)\right\}\right)\right), & c \leq N. \end{cases}$$

Karena banyak maksimal elemen berbeda dari V adalah $|S|+1$, maka secara implementasi V dapat dinyatakan dalam *bitmask*.

Dalam pengerjaannya, implementasi naif untuk menghitung $\bigcup_{q \in V} \delta(q,a)$ dalam transisi pada kasus keempat dalam relasi rekurensi di atas memiliki kompleksitas sebesar $\mathcal{O}(|S|)$. Hal tersebut dapat dioptimisasi dengan melakukan prekomputasi terhadap pemetaan $\delta^* : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$ sehingga untuk suatu himpunan *state* V dan karakter a, penghitungan $V' = \bigcup_{q \in V} \delta(q,a)$ dapat dilakukan dalam $O(1)$, yaitu $V' = \delta^*(V,a)$.

Bentuk relasi rekurensi yang telah dioptimisasi:

$$\mathrm{dp}(c,V) = \begin{cases} 0, & c > N+1 \\ 0, & c = N+1 \wedge A \cap V = \varnothing \\ 1, & c = N+1 \wedge A \cap V \neq \varnothing \\ \displaystyle\sum_{a,b \in A_K} \mathrm{dp}\left(c+1, \delta^*\left(V,a\right) \cup \delta^*\left(V,b\right) \cup \delta^*\left(\left(V,a\right),b\right) \cup \delta^*\left(\left(V,b\right),a\right)\right), & c < N+1. \end{cases}$$

Jawaban terdapat pada $\mathrm{dp}\left(1, \{q_\epsilon\}\right)$ Solusi ini memiliki kompleksitas sebesar $\mathcal{O}$(pembangunan DFSM untuk S)+ $\mathcal{O}(\textit{state}\ \mathrm{DP}) \cdot \mathcal{O}(\text{transisi DP}) = \mathcal{O}(|S|) + \mathcal{O}(N2^{|S|+1}) \cdot \mathcal{O}(K^2) = \mathcal{O}(|S| + NK^2 2^{|S|+1})$.

Solusi Tanpa Pembuatan DFSM KMP

Terdapat pula solusi yang lebih mudah, yakni tanpa membuat DFSM KMP. Observasinya adalah sebagai berikut: untuk setiap *bitmask* V yang *valid* pada DP, jika sebuah *state* q adalah elemen dari V, maka q' juga pasti adalah elemen dari V, untuk setiap q' yang merepresentasikan sufiks dari string yang direpresentasikan oleh q, yang mana juga merupakan prefiks dari S. Sebagai contoh, jika V mengandung `tesat`, maka V juga pasti mengandung `t`. Sisanya diserahkan kepada pembaca sebagai latihan.