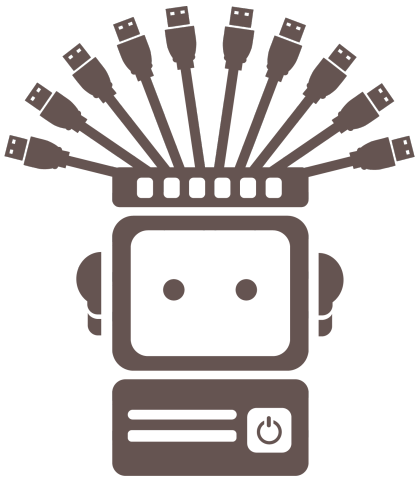
BUKLET PEMBAHASAN SOAL



GEMASTIK 10
2017

Final PEMROGRAMAN GEMASTIK 10
5 November 2017

Soal-Soal

Kode	Judul	Penulis
A	Organisasi Kemahasiswaan	Alham Fikri Aji
B	Operasi Bitwise	Alham Fikri Aji
C	Sirkus Keliling	Pusaka Kaleb Setyabudi
D	Kuis Kata	Pusaka Kaleb Setyabudi
E	Pak Grandi	Jonathan Irvin Gunawan
F	Balon Poligon	William Gozali
G	Juara Umum	Alham Fikri Aji
H	Harta Kekayaan	Ashar Fuadi
I	Berbagi Lawakan	William Gozali

Penguji

Ammar Fathin Sabili

Organisasi Kemahasiswaan

Penulis soal : Alham Fikri Aji
Penulis editorial : Alham Fikri Aji
Tema : math, greedy

Catatan/Komentar

Batasan kedua versi:

- $1 \leq K \leq N$
- $1 \leq M[i] \leq 100.000$

Versi Mudah

Batasan: $1 \leq N \leq 2$

Karena hanya ada paling banyak 2 organisasi, maka kita bisa selesaikan persoalan ini dengan melihat setiap kasus. Jika hanya ada tepat 1 organisasi, maka dipastikan mahasiswa minimum yang mungkin adalah $M[1]$. Jika ada 2 organisasi, maka kita harus melihat nilai K. Jika $K = 1$, maka setiap mahasiswa hanya boleh mengikuti paling banyak 1 organisasi, sehingga mahasiswa minimum yang mungkin adalah $M[1] + M[2]$. Namun jika $K = 2$, maka mahasiswa minimum yang mungkin adalah $\max(M[1], M[2])$

Versi Sulit

Batasan: $1 \leq N \leq 100.000$

Cukup mudah disimpulkan bahwa mahasiswa paling sedikit berjumlah

$$\max_{1 \leq i \leq N} M[i]$$

atau sama dengan anggota dari organisasi dengan jumlah anggota terbanyak.

Jika asumsikan Fasilkom memiliki total T mahasiswa, dan setiap mahasiswa boleh mengikuti paling banyak K organisasi, dapat disimpulkan bahwa $(\sum_{i=1}^{N} M[i]) \leq T \times K$ harus terpenuhi. Jika tidak, menurut teorema pigeon-hole principle paling tidak ada 1 mahasiswa yang mengikuti lebih dari K organisasi. Dengan demikian, kita dapat menghitung mahasiswa Fasilkom dengan rumus

$$T = \left\lceil \frac{\sum_{i=1}^{N} M[i]}{K} \right\rceil$$

dan jawaban akhir berupa

$$\max \left(\max_{1 \leq i \leq N} M[i], \left\lceil \frac{\sum_{i=1}^{N} M[i]}{K} \right\rceil \right)$$

Operasi Bitwise

Penulis soal : Alham Fikri Aji
Penulis editorial : Alham Fikri Aji
Tema : *ad-hoc*, *constructive*

Catatan/Komentar

Untuk pembahasan soal ini, kita definisikan solusi sebagai $S_1, S_2, ..., S_N$.

Versi Mudah

Batasan: $N = 2; 0 \leq A, B, C \leq 2^{20}$

Ada beberapa solusi yang dapat digunakan untuk menyelesaikan versi mudah. Salah satunya adalah dengan melakukan brute-force. Perhatikan bahwa N selalu 2. Lakukan perulangan untuk mencoba seluruh nilai S_1 dari 0 sampai B. Perulangan cukup dilakukan hingga B, karena nilai hasil operasi bitwise OR dua bilangan selalu lebih besar atau sama dengan hasil operasi AND ataupun XOR-nya. Untuk setiap nilai S_1, kita akan mencoba menghitung S_2. Perhatikan bahwa S_1 XOR $S_2 = C$, dan berdasarkan sifat XOR maka kita dapat menghitung $S_2 = C$ XOR S_1. Setelah didapatkan S_1 dan S_2, kita cukup melakukan validasi terhadap A dan B. Kompleksitas solusi ini adalah $O(B)$.

Versi Sulit

Batasan: $1 \leq N \leq 50.000; 0 \leq A, B, C \leq 2^{31}$

Mari kita bagi persoalan ini menjadi 3 kasus.

Kasus N = 1

$S_1 = A$ jika dan hanya jika $A = B = C$. Selain itu tidak ada solusi.

Kasus N = 2

Bisa diselesaikan sesuai pembahasan di versi mudah.

Kasus N > 2

Perhatikan bahwa setiap bit dapat kita selesaikan secara independen. Artinya, untuk setiap $0 \leq i \leq 31$, kita dapat mengerjakan soal ini dengan nilai baru A_i, B_i, C_i yang mana A_i bernilai 1 jika dan hanya jika $(A \text{AND} 2^i) > 0$, B_i bernilai 1 jika dan hanya jika $(B \text{AND} 2^i) > 0$, dan C_i bernilai 1 jika dan hanya jika $(C \text{AND} 2^i) > 0$. Untuk setiap i, kita akan mencari solusi $s[i]_1, s[i]_2, ..., s[i]_N$. Pada akhirnya, kita dapat menggabungkan solusi-solusi sebagai

$$S_j \sum_{i=0}^{31} 2^i \times s[i]_j$$

Untuk setiap nilai i, kita tahu hanya ada 8 kemungkinan A_i, B_i, C_i berbeda. Selain itu, kita bisa simpulkan bahwa nilai A_i akan 1 jika seluruh $s[i]_j = 1$, nilai $B[i]_1$ akan 1 jika paling tidak ada 1 dari nilai j yang memenuhi $s[i]_j = 1$, dan nilai C_i akan 1 jika j yang memenuhi $s[i]_j = 1$ berjumlah ganjil. Maka, kita dapat mencoba seluruh kemungkinan dan mendapatkan solusi sebagai berikut:

A_i	B_i	C_i	$s[i]_1$	$s[i]_2$	$s[i]_3 \dots s[i]_N$
0	0	0	0	0	0
0	0	1	tidak ada solusi		
0	1	0	1	1	0
0	1	1	1	0	0
1	0	0	tidak ada solusi		
1	0	1	tidak ada solusi		
1	1	0	1	1	1, hanya ada solusi jika N genap
1	1	1	1	1	1, hanya ada solusi jika N ganjil

Sirkus Keliling

Penulis soal : Pusaka Kaleb Setyabudi
Penulis editorial : Pusaka Kaleb Setyabudi
Tema : *min-cost max flow*

Catatan/Komentar

Batasan yang berlaku pada versi mudah dan sulit:

- $1 \leq$ T ≤ 5
- $1 \leq$ M $\leq$ N $\times$ (N - 1)
- $1 \leq$ S[i], W[i] $\leq$ 8.000.000
- $1 \leq$ U[i], V[i] $\leq$ N
- U[i] $\neq$ V[i]
- (U[i], V[i]) $\neq$ (U[j], V[j]) untuk i $\neq$ j

Trivia: Soal ini pada awalnya berjudul "Trayek Bis" dengan tema penentuan trayek-trayek transportasi. Namun, karena hal "trayek yang menghubungkan satu kota dengan dirinya sendiri" dianggap terlalu *absurd*, maka diganti dengan tema yang dipakai saat ini.

Versi Mudah

Batasan: $1 \leq N \leq 12$

Karena batasan nilai N yang cukup kecil, penyelesaian versi mudah dapat dilakukan dengan menggunakan *Dynamic Programming* dengan teknik *bitmask* sebagai berikut.

Didefinisikan dua buah fungsi berikut:

1. $f(u, C, s)$ dengan $1 \leq u, s \leq N$, $C \subseteq \{1, 2, \ldots, N\} \backslash \{u, s\}$, yang menyatakan biaya untuk memanfaatkan sirkus keliling yang dimulai dari kota s untuk mengunjungi kota-kota C dengan keadaan telah mencapai kota u.
2. $g(K)$ dengan $K \subseteq \{1, 2, \ldots, N\}$, yang menyatakan biaya untuk menentukan pemanfaatan sirkus keliling dan sirkus lokal untuk kota-kota K.

Untuk fungsi DP pertama, rekurensi dapat dilakukan serupa dengan rekurensi pada penyelesaian DP untuk permasalahan *traveling salesman problem*[1] yang memiliki kompleksitas $\mathcal{O}\left(N^2 \cdot 2^N\right)$.

Untuk fungsi DP kedua, rekurensinya dapat didefinisikan sebagai berikut.

$$g(K) = \begin{cases} 0, & K = \emptyset \\ \min\limits_{\substack{K' \subseteq K \\ K' \neq \emptyset \\ u \in K'}} \left(f\left(u, K' - \{u\}, u\right) + g\left(K - K'\right)\right), & K \neq \emptyset \end{cases}$$

Perhatikan bahwa pada rekurensi di atas, pencarian $K' \subseteq K$ dapat dilakukan dalam kompleksitas $\mathcal{O}\left(3^N\right)$.

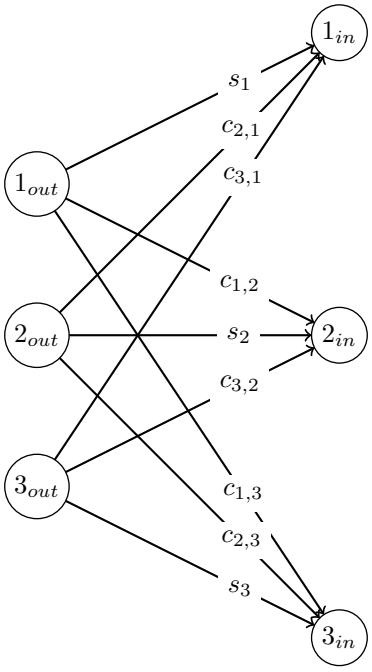
Kompleksitas akhir dari solusi ini adalah $\mathcal{O}\left(3^N + N^2 \cdot 2^N\right)$, cukup untuk mendapatkan *accepted* pada versi mudah.

[1] dapat mengacu pada halaman ini: http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/

Versi Sulit

Batasan: $1 \leq N \leq 250$

Solusi untuk versi sulit dapat dicari melalui nilai *min-cost max flow* dari bentuk graf awal yang telah dimodelkan sebagai berikut (sebagai contoh, diambil $N = 3$).



Dalam bentuk tersebut, setiap *node* pada soal dipecah menjadi dua bagian, yaitu *node* yang harus tepat memiliki *outdegree* tepat 1 dan *node* yang harus tepat memiliki *indegree* tepat 1. Dengan demikian, setiap *node* harus dipasangkan antara dengan dirinya sendiri (*outdegree* dan *indegree* dari *node* tersebut dipakai untuk membuat suatu *loop*) atau dengan satu *node* lain yang nantinya akan membentuk suatu *cycle* (bayangkan bahwa pada *cycle* terdapat n *node* dengan *node* pertama "memakai" *outdegree*-nya untuk *indegree* dari *node* kedua, *node* kedua "memakai" *outdegree*-nya untuk *indegree* dari *node* ketiga, dan seterusnya). Perhatikan bahwa pada solusi di atas, definisi "memanfaatkan sirkus lokal" pada suatu kota u diubah menjadi "membuat suatu *self-loop*" pada kota u dengan biaya sebesar s_u.

Eksekusi algoritma *min-cost max flow* pada graf tersebut akan memastikan bahwa setiap *node* dalam soal akan terpasangkan pada (1) dirinya sendiri (melalui *edge* dengan *cost* sebesar s_i) atau (2) *node* lain dengan *cost* sebesar $c_{i,j}$. Beberapa algoritma yang dapat digunakan adalah modifikasi algoritma Edmonds-Karp atau dapat juga menggunakan algoritma Kuhn-Munkres (Hungarian Algorithm). Modifikasi dari algoritma Edmonds-Karp memiliki kompleksitas $\mathcal{O}\left(N^3 \cdot \log N\right)$ dan algoritma Kuhn-Munkres memiliki kompleksitas $\mathcal{O}\left(N^3\right)$. Keduanya mampu mendapatkan *accepted* pada versi sulit dari soal ini.

Kuis Kata

Penulis soal : Pusaka Kaleb Setyabudi
Penulis editorial : Pusaka Kaleb Setyabudi
Tema : *Sprague-Grundy theorem, dynamic programming*

Catatan/Komentar

Batasan yang berlaku pada versi mudah dan sulit:

- $1 \leq \mathrm{T} \leq 100$
- S[1] terdiri atas 1 hingga L karakter abjad `a - z`
- Karakter-karakter pada S[1] terurut tidak menurun berdasarkan abjad

Pembahasan di bawah ditulis dengan asumsi bahwa pembaca telah memahami mengenai *Grundy number* dan *Sprague-Grundy theorem*.[2]

Versi Mudah

Batasan: $N = 2; 1 \leq L \leq 5$

Definisikan suatu fungsi $g(S)$ yang menyatakan *Grundy number* dari suatu *string* S. Jawaban dari versi mudah dapat didapat dengan mencari banyaknya *string* S' yang memenuhi $g(S')$ xor $g(S[0]) = 0$. *Bruteforce* dapat dilakukan untuk mendapatkan *string-string* S'.

Perhatikan bahwa prekomputasi terhadap banyaknya *string* S' yang memenuhi $g(S') = x$ untuk suatu *Grundy number* x diperlukan untuk mendapatkan *accepted* pada versi ini.

Sehingga, kompleksitas solusi adalah $\mathcal{O}\left((\text{banyak kemungkinan } \textit{string} \text{ valid}) + T\right)$.

Fun fact: Grundy number terbesar untuk suatu *string* S' adalah 25. Pembuktian diserahkan kepada pembaca.

Versi Sulit

Batasan: $1 \leq N \leq 1.000.000.000; 1 \leq L \leq 10.000$

Solusi juri dan pembahasan untuk versi sulit dibuat berdasarkan bentuk ekivalen/transformasi soal asli yaitu sebagai berikut.

Pendahuluan: Transformasi Soal

Suatu *string* $A = a_1 a_2 a_3 \ldots a_{|A|}$ dapat diubah bentuknya menjadi barisan $A' = (a'_1, a'_2, a'_3, \ldots, a'_{|A|}, a'_{|A|+1})$ dengan $a'_1 = \text{jarak}(a_1, \texttt{a}), a'_{|A|+1} = \text{jarak}(a_{|A|}, \texttt{z})$, dan $a'_i = \text{jarak}(a_i, a_{i-1})$ untuk $2 \leq i \leq |A|$. Sebagai contoh, *string*

$$A = \texttt{aku}$$

ekivalen dengan barisan

$$A' = (0, 10, 10, 5).$$

Untuk selanjutnya, istilah "*state*" akan dipakai secara ekivalen dengan istilah "barisan".

[2]untuk referensi, dapat mengacu pada `https://www.hackerrank.com/topics/game-theory-and-grundy-numbers/` dan `http://www.gabrielnivasch.org/fun/combinatorial-games/sprague-grundy`.

Pada bentuk yang diubah ini, perhatikan bahwa suatu langkah mengganti karakter a_i dengan $a_i + x$ pada *string* A ekivalen dengan menambah nilai a'_i sebanyak x dan mengurangi nilai a'_{i+1} sebanyak x. Sebagai contoh, jika *string* `aku` di atas diubah menjadi `anu`, maka A' berubah menjadi

$$A'' = (0, 13, 7, 5).$$

Dengan bentuk demikian, kondisi selesai dari kuis kata adalah jika barisan A' berbentuk

$$A' = (25, 0, 0, \ldots).$$

atau ekivalen dengan *string*

`zzz...`

Pendahuluan: *Grundy Number* untuk A

Observasi 1. $A = (*, 0, 0, 0, \ldots)$ dengan $*$ sembarang bilangan bulat nonnegatif, memiliki *Grundy number* 0 (*losing state*).

Lemma 1. $A = (*, 0, *, 0, 0, 0, \ldots)$ memiliki *Grundy number* 0 (*losing state*).

Bukti Lemma 1: Perhatikan bahwa pemain pertama hanya dapat melakukan langkah sehingga A berubah menjadi $(*, a, b, 0, 0, 0, \ldots)$ (memindahkan sebanyak a dari posisi ketiga). Pada langkah selanjutnya, pemain kedua dapat selalu memastikan kemenangan dengan cara mengikuti langkah pemain pertama yaitu dengan memindahkan a tersebut ke posisi selanjutnya/di depannya (yaitu menjadi $(*, 0, b, 0, 0, 0, \ldots)$). Pola tersebut terus dilakukan oleh pemain kedua hingga keadaan permainan menjadi $(*, 0, 0, 0, \ldots)$ (keadaan dalam Observasi 1).

Corollary 1. $A = (*, 0, *, 0, *, 0, *, \ldots)$ memiliki *Grundy number* 0 (*losing state*).

Lemma 2. $A = (*, a, 0, 0, 0, \ldots)$ untuk $a \geq 1$ adalah *winning state* dengan *Grundy number* a.

Bukti Lemma 2: Untuk sembarang a, perhatikan bahwa *state* A dapat mengunjungi tepat $a-1$ *state* lainnya yaitu $(*, 0, 0, 0, \ldots), (*, 1, 0, 0, 0, \ldots), \ldots, (*, a-1, 0, 0, 0, \ldots)$ dengan masing-masing memiliki *Grundy number* sesuai dengan bilangan pada posisi kedua. Oleh karena itu, *Grundy number* dari $(*, a, 0, 0, 0, \ldots)$ adalah $\text{mex}(\{0, 1, 2, \ldots, a-1\}) = a$. Secara intuisi, *state* A adalah *winning state* karena pemain pertama dapat mengubah A menjadi $(*, 0, 0, 0, \ldots)$ yang adalah *losing state* (Observasi 1).

Corollary 2. $A = (*, a, *, 0, *, 0, *, 0, \ldots)$ untuk $a \geq 1$ adalah *winning state* dengan *Grundy number* a.

Corollary 3. *State* $A = (*, a, *, 0, *, 0, *, 0, \ldots)$ ekivalen dengan *state* $A' = (0, a, 0, 0, 0, \ldots)$.

Lemma 3. *State* $A = (0, a, 0, b, 0, 0, 0, \ldots)$ ekivalen dengan gabungan dari dua *state* yang saling independen $B = (0, a, 0, 0, 0, \ldots)$ dan $C = (0, b, 0, 0, 0, \ldots)$. Dengan kata lain, *Grundy number* keduanya bernilai sama ($g(A) = g(B)$ xor $g(C)$).

Bukti Lemma 3: Salah satu pembuktian dapat dilakukan melalui *bruteforce* terhadap ke 26×26 kemungkinan nilai a dan b yang ada. *State* A dapat didekomposisi menjadi dua *state* lain tersebut karena selama permainan, pengurangan terhadap suatu bilangan pada posisi genap tertentu tidak akan mempengaruhi bilangan pada posisi genap lainnya (Corollary 3).

Sebagai contoh, perubahan terhadap *state*

$$A = (0, a, 0, b, 0, 0, 0, \ldots)$$

menjadi

$$A' = (0, a, b', b - b', 0, 0, 0, \ldots)$$

adalah ekivalen dengan perubahan *state* A menjadi

$$A'' = (0, a, 0, b - b', 0, 0, 0, \ldots).$$

Perhatikan bahwa terdapat dua kesimpulan: (1) *state* A' dan *state* A'' ekivalen, dan (2) nilai b' yang merupakan pengurang dari b (suatu bilangan pada posisi genap) tidak akan mempengaruhi bilangan posisi genap lainnya.

Corollary 4. *State* $A = (*, x_2, *, x_4, *, x_6, \ldots)$ memiliki *Grundy number* yaitu $g(A) = x_2$ xor x_4 xor x_6 xor $\ldots$.

Solusi

Berdasarkan *Sprague-Grundy theorem*, solusi untuk versi sulit dapat dirumuskan menjadi mencari banyaknya kombinasi *string* $S[2], S[3], \ldots, S[N]$ sehingga $g(S[1])$ xor $g(S[2])$ xor $g(S[3])$ xor $\ldots$ xor $g(S[N]) = 0$. Tahap-tahap pencarian nilai tersebut dapat dicapai melalui perumusan DP berikut.

$$f(n, r) = \begin{cases} 0, & n = 0 \wedge r \neq 0 \\ 1, & n = 0 \wedge r = 0 \\ \sum_{i=0}^{25} f(n-1, r \text{ xor } i) * h(L, i), & n > 0 \end{cases}$$

dengan $f(n, r)$ menyatakan banyaknya kemunkinan n *string* yang memiliki *Grundy number* r dan $h(n, r)$ menyatakan banyaknya *string* dengan panjang maksimal n yang memiliki *Grundy number* r. Perhatikan bahwa $h(n, r)$ juga dapat dikomputasi menggunakan *dynamic programming*. Jawaban terdapat pada $f(N - 1, g(S[1]))$.

Namun, implementasi DP di atas secara mentah tidak akan mendapatkan *accepted* pada versi ini karena batasan N yang sangat besar. Untuk mendapatkan *accepted*, penghitungan fungsi f di atas perlu dilakukan melalui perkalian matriks.[3]

[3]untuk referensi, silakan kunjungi `http://fusharblog.com/solving-linear-recurrence-for-programming-contest/`.

Pak Grandi

Penulis soal : Jonathan Irvin Gunawan (dengan modifikasi)
Penulis editorial : Anthony
Tema : *range query*

Catatan/Komentar

Batasan kedua versi:

- $1 \leq P[i] \leq 2$
- $1 \leq L[i], R[i] \leq 10^9$
- $1 \leq K[i] \leq 10^9$
- $1 \leq N \leq 50.000$

Versi Mudah

Batasan: $\mathbf{L[i]} = \mathbf{R[i]}$

Perhatikan bahwa untuk versi mudah pada soal ini, operasi-operasi dengan $L[i] > N$ dapat diabaikan.

Penyelesaian versi ini cukup sederhana. Salah satu cara yang dapat digunakan adalah sebagai berikut:

1. Inisialisasi sebuah *set* dengan bilangan-bilangan dari 1 hingga N + 1 dan sebuah *array* untuk menyimpan nilai banyaknya kemunculan masing-masing bilangan dari 1 sampai N.
2. Untuk setiap operasi:
   - Tambah: tambahkan nilai kemunculan L[i] sebanyak K[i]. Jika L[i] terdapat dalam *set*, hapus L[i] dari *set*.
   - Hapus: kurangkan nilai kemunculan L[i] sebanyak K[i] (jika hasilnya bernilai kurang dari 0, nilai kemunculan dianggap 0). Jika nilai kemunculan L[i] bernilai 0, tambahkan L[i] ke dalam *set*.

   Setiap selesai operasi, cukup ambil elemen terkecil dalam *set*. Perhatikan bahwa *set* tersebut digunakan untuk menyimpan bilangan-bilangan yang **tidak** sedang berada dalam *multiset*.

Versi Sulit

Batasan: $\mathbf{L[i]} \leq \mathbf{R[i]}$

Untuk pengerjaan versi sulit pada soal ini, salah satu pendekatan yang dapat digunakan yaitu dengan memproses secara *offline*.

Mengerjakan secara *offline* maksudnya adalah dengan membaca semua perintah atau operasi yang perlu dilakukan terlebih dahulu. Misalnya untuk soal ini berarti semua nilai (P, L, R, K) akan disimpan terlebih dahulu tanpa menghitung jawaban secara langsung.

Setelah semua interval [L, R] dibaca, maka interval-interval tersebut dapat dikompres. Sebagai kasus umum (tidak terbatas pada soal ini), jika terdapat N buah bilangan berbeda, bilangan-bilangan tersebut akan diubah menjadi bilangan dari 1 hingga N (atau lainnya sesuai kebutuhan). Contohnya yaitu jika terdapat [1, 6, 8, 10, 100], dapat dibuat pemetaan satu-satu berikut:

- $1 \leftrightarrow 1$
- $6 \leftrightarrow 2$
- $8 \leftrightarrow 3$
- $10 \leftrightarrow 4$
- $100 \leftrightarrow 5$

Karena paling banyak terdapat 2N batas-batas interval, pada hasil kompresi akan terdapat paling banyak 2N bilangan pula, sehingga telah dapat dilakukan operasi-operasi menggunakan struktur data seperti *segment tree*.

Node pada *segment tree* perlu mengandung informasi mengenai apakah interval yang dicakup lengkap atau tidak sehingga pada saat *query* hanya membutuhkan waktu $\mathcal{O}(\log N)$. Langkah *query* pada suatu *node* adalah:

- jika *node* tersebut mempunyai anak:
  - jika interval yang dicakup anak kirinya lengkap, telusuri anak kanan
  - jika tidak lengkap, telusuri anak kiri
- jika tidak mempunyai anak (dengan kata lain hanya mencakup 1 posisi), maka bagian ini merupakan jawaban.

Dengan cara ini maka secara keseluruhan untuk menyelesaikan N buah operasi membutuhkan waktu $\mathcal{O}(N \log N)$

Balon Poligon

Penulis soal :	William Gozali
Penulis editorial :	William Gozali
Penulis Solusi :	Pusaka Kaleb Setyabudi
Tema :	*ternary search*, geometri

Catatan/Komentar

Batasan kedua versi: $3 \leq M \leq 10; -10^6 \leq X[i], Y[i] \leq 10^6$

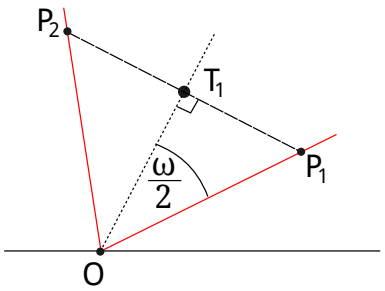
Untuk memudahkan penjelasan, digunakan definisi sebagai berikut:

1. O menyatakan titik $(0, 0)$.
2. $T_1, T_2, ..., T_N$ sebagai titik-titik pada bidang Kartesius.
3. $P_1, P_2, ..., P_{M-1}$ sebagai titik-titik sudut dari poligon.
4. ω menyatakan besarnya sudut $\angle P_1OP_2$

Versi Mudah

Batasan: $\mathbf{N = 1}$

Solusi optimal dicapai apabila segmen garis OT_1 dijadikan garis yang tegak lurus terhadap P_1P_2.



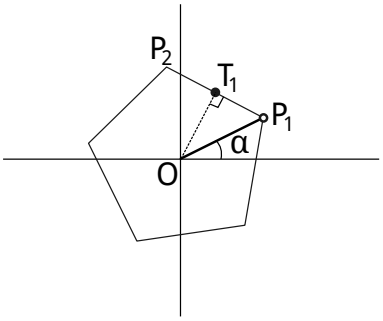
Ukuran dari balon poligon adalah $||OP_1||$, yang dapat dicari dengan cara berikut:

$$\begin{aligned} ||OT_1|| &= ||OP_1|| \cos \frac{\omega}{2} \\ ||OP_1|| &= \frac{||OT_1||}{\cos \frac{\omega}{2}} \\ ||OP_1|| &= \frac{1}{\cos \frac{\omega}{2}} ||OT_1|| \end{aligned}$$

Versi Sulit

Batasan: $\mathbf{1 \leq N \leq 100}$

Untuk versi sulit, definisikan sebuah variabel tambahan, yaitu α yang menyatakan sudut yang dibentuk oleh sumbu-x positif terhadap P_1.



Karena poligon memiliki simetri putar, nilai α yang perlu dipedulikan adalah $0 \leq \alpha < \omega$.

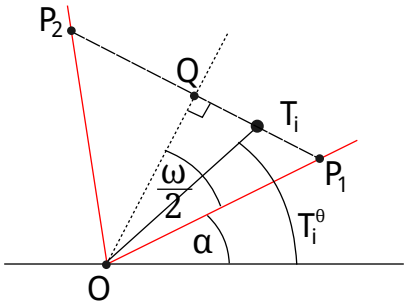
Versi ini juga lebih mudah apabila kita bekerja dengan sistem koordinat kutub. Artinya, setiap titik T_i memiliki dua informasi

1. Jarak T_i ke titik O. Nyatakan nilai ini dengan T_i^r.
2. Sudut yang dibentuk dari T_i, ke titik O, lalu menuju ke suatu titik $(\infty, 0)$. Nyatakan nilai ini dengan T_i^θ. Nilai T_i^θ adalah anggota dari $[0, 360)$.

Misalkan diberikan sebuah nilai α, dan kita hendak menghitung berapa ukuran poligon apabila hanya terdapat titik T_i. Diketahui pula T_i berada di dalam daerah yang diapit oleh P_1 dan P_2.

Misalkan segmen garis OQ merupakan segmen garis yang tegak lurus dengan P_1 dan P_2. Perhitungan nilai $||OP_1||$ dapat dicari dengan cara berikut:

$$\begin{aligned} ||OP_1|| &= \frac{1}{\cos\frac{\omega}{2}}||OQ|| \\ &= \frac{1}{\cos\frac{\omega}{2}}||OT_i||\cos\left|\alpha + \frac{\omega}{2} - T_i^\theta\right| \end{aligned}$$

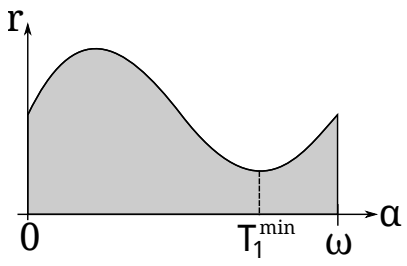


Untuk kasus ketika titik T_i tidak berada di daerah yang diapit oleh P_1 dan P_2, normalisasi nilai T_i^θ dengan mengurangkannya dengan ω sampai berada di daerah yang diapit oleh P_1 dan P_2. Hasil perhitungan akan tetap benar, dikarenakan adanya simetri putar.

Apabila rumus ukuran poligon tersebut dinyatakan sebagai fungsi terhadap α, yang mana terdefinisi untuk $0 \leq \alpha < \omega$ maka fungsi tersebut akan membentuk suatu kurva yang memiliki tepat sebuah puncak dan sebuah lembah. Titik puncak dicapai ketika nilai α menyebabkan T_i berada di posisi Q, dan titik terendah dicapai saat α menyebabkan T_i berada di posisi P_1.

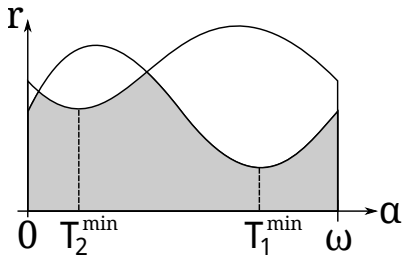
Misalkan T_i^{min} menyatakan nilai α yang menyebabkan ukuran poligon paling kecil ketika hanya terdapat titik T_i. Nilai α yang memenuhi adalah nilai sebesar T_i^θ yang telah dinormalisasi.

Untuk kasus ketika hanya terdapat T_1, didapatkan kurva berikut, dengan r menyatakan ukuran poligon:



Terdapat dua daerah yang mana masing-masing merupakan fungsi yang memiliki tepat sebuah nilai puncak, yaitu daerah $\alpha \in [0, T_1^{min}]$ dan $\alpha \in [T_1^{min}, \omega]$.

Untuk kasus ketika terdapat T_1 dan T_2, kita dapat mencari irisan kedua kurva tersebut:



Kini terdapat tiga daerah yang mana masing-masing merupakan fungsi yang memiliki tepat sebuah nilai puncak, yaitu daerah $\alpha \in [0, T_2^{min}]$, $\alpha \in [T_2^{min}, T_1^{min}]$, $\alpha \in [T_1^{min}, \omega]$.

Ketika seluruh kurva untuk T_i digabungkan, didapatkan $\mathcal{O}(N)$ daerah yang masing-masing memiliki tepat sebuah nilai puncak. Lakukan *ternary search* untuk masing-masing daerah tersebut, dan ambil nilai terbesar. Nilai ini adalah ukuran poligon terbesar yang mungkin.

Untuk akurasi, *ternary search* dapat dilakukan sebanyak k kali, misalnya antara 50 sampai dengan 100. Pada setiap tahap, diperlukan pemeriksaan terhadap nilai terkecil r yang mungkin dari N titik yang ada. Kompleksitas waktu untuk setiap *ternary search* ini adalah $\mathcal{O}(kN)$. Karena dilakukan sebanyak $\mathcal{O}(N)$ kali, kompleksitas akhirnya adalah $\mathcal{O}\left(kN^2\right)$.

Catatan Tambahan

Terdapat solusi untuk versi sulit yang memiliki kompleksitas waktu $\mathcal{O}(kN)$ dengan k adalah konstanta untuk *binary search*, yaitu dengan melakukan *binary search* sejauh k kali pada ukuran poligon. Implementasi secara lebih rincinya diserahkan kepada pembaca sebagai latihan.

Juara Umum

Penulis soal : Alham Fikri Aji
Penulis editorial : William Gozali
Tema : Ad-hoc, Constructive

Catatan/Komentar

Terdapat beberapa solusi untuk menyelesaikan soal ini. Setidaknya, tiga orang yang terlibat dalam pembuatan soal ini memiliki solusi yang berbeda. Solusi yang dibahas pada tulisan ini adalah dari Alham Fikri Aji.

Misalkan universitas ke-i dinyatakan sebagai u_i, dan cabang lomba ke-i dinyatakan sebagai c_i.

Batasan kedua versi: $1 \leq N \leq 50000; 3 \leq M \leq 50000; K \leq N$

Versi Mudah

Batasan: $\mathbf{K = 2}$

Untuk mudahnya, u_1 dan u_2 diberikan $\lfloor N/2 \rfloor$ buah medali emas dan perak. Medali perunggu sebanyak $\lfloor N/2 \rfloor$ buah tidak dapat diberikan kepada u_1 dan u_2, sebab mereka menjadi perlu memenangkan $3 \times \lfloor N/2 \rfloor$, yang mana akan melebihi N (banyaknya lomba). Untuk mengatasi masalah ini, kita dapat memberikan seluruh medali perunggu kepada universitas yang bukan juara umum, yaitu u_3. Hal ini selalu dapat dilakukan, sebab dijamin paling sedikit terdapat 3 universitas.

Sebagai contoh, berikut pengaturan pemenang untuk $N = 4$:

	Emas	Perak	Perunggu
c_1	u_1	u_2	u_3
c_2	u_2	u_1	u_3
c_3	u_1	u_2	u_3
c_4	u_2	u_1	u_3

Cara ini dapat digunakan untuk N bernilai genap. Untuk N bernilai ganjil, terdapat sisa sebuah medali emas, perak, dan perunggu yang harus dibagikan. Medali emas sisa ini harus diberikan kepada universitas selain u_1 dan u_2. Kita dapat memberikannya kepada u_3.

Untuk membagikan medali perak dan perunggu yang tersisa, kita dapat sedikit mengubah konfigurasi pemenang lombanya dengan cara berikut:

1. Berikan medali perak kepada u_2.
2. Berikan medali perunggu kepada u_1.
3. Tukarkan pemenang medali perak dan perunggu pada c_1, sehingga kini u_3 memenangkan perak dan u_2 memenangkan perunggu.

	Emas	Perak	Perunggu
c_1	u_1	$\mathbf{u_3}$	$\mathbf{u_2}$
c_2	u_2	u_1	u_3
c_3	u_1	u_2	u_3
c_4	u_2	u_1	u_3
c_5	u_3	$\mathbf{u_2}$	$\mathbf{u_1}$

Khusus untuk kasus $N = 1$ dan $N = 3$, tidak mungkin ada konfigurasi pemenang yang memungkinkan terdapat 2 juara umum.

Versi Sulit

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

Kasus $K = 1$

Dapat diselesaikan dengan memberikan seluruh medali emas kepada u_1, seluruh medali perak kepada u_2, dan seluruh medali perunggu kepada u_3.

Kasus $K = 2$

Dapat diselesaikan dengan solusi versi mudah.

Kasus $K \geq 3$

Kita dapat membagikan $\lfloor N/K \rfloor$ medali emas, perak, dan perunggu kepada $u_1, u_2, ..., u_K$. Hal ini kini dapat dilakukan sebab dipastikan $(3 \times \lfloor N/K \rfloor) \leq N$. Misalkan $N = 21$ dan $K = 4$, penentuan pemenangnya dapat diatur dengan pola berotasi dan berulang berikut untuk $(K \times \lfloor N/K \rfloor)$ lomba pertama:

	Emas	Perak	Perunggu
c_1	u_1	u_2	u_3
c_2	u_2	u_3	u_4
c_3	u_3	u_4	u_1
c_4	u_4	u_1	u_2
c_5	u_1	u_2	u_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Medali emas, perak, dan perunggu yang tersisa adalah masing-masing sebanyak sisa bagi N terhadap K, yang akan dituliskan sebagai N mod K.

Medali pada cabang lomba yang tersisa perlu diberikan kepada $M - K$ universitas di peringkat selanjutnya. Untuk menjamin tidak ada satupun dari universitas ini yang memiliki medali sebanyak juara umum, perlu diatur sehingga setiap universitas memenuhi salah satu syarat berikut:

1. Memiliki medali emas sebanyak $\lfloor N/K \rfloor$, medali perak paling banyak $(\lfloor N/K \rfloor - 1)$, dan medali perunggu sebanyak apapun selama masih valid.
2. Memiliki medali emas paling banyak $(\lfloor N/K \rfloor - 1)$, dan boleh memiliki medali perak atau perunggu sebanyak apapun selama masih valid.

Kita dapat memilih salah satu universitas, misalnya u_{K+1}, untuk mendapatkan medali emas sebanyak $\min(\lfloor N/K \rfloor - 1, N \bmod K)$ dan mendapatkan medali perak/perunggu sebanyak mungkin. Medali emas sisanya dibagi rata kepada universitas lainnya, masing-masing paling banyak $\lfloor N/K \rfloor$.

Strategi berikut dapat digunakan untuk penentuan pemenang pada masing-masing N mod K cabang lomba yang tersisa. Misalnya untuk penentuan pemenang c_i:

1. Jika medali emas masih dapat diberikan kepada u_{K+1}, maka pemenang medali emas adalah u_{K+1}. Selain daripada itu, berikan kepada universitas lainnya yang masih belum menerima emas sebanyak $\lfloor N/K \rfloor$.
2. Jika u_{K+1} menerima medali emas, untuk sementara berikan medali perak kepada u_{K+1}. Kemudian tukarkan pemenang medali perak c_i dengan suatu cabang lomba c_x, yang mana pemenang medali perak pada c_x bukan u_{K+1}. Sementara ketika u_{K+1} tidak menerima medali emas, berikan medali perak kepadanya.
3. Untuk sementara berikan medali perunggu kepada u_{K+1}. Kemudian tukarkan pemenang medali perunggu c_i dengan suatu cabang lomba c_x, yang mana pemenang medali perunggu pada c_x bukan u_{K+1}.

Apabila seluruh universitas telah menerima medali emas sebanyak yang mereka bisa, tetapi medali emas masih tersisa, berarti jawabannya adalah mustahil.

Harta Kekayaan

Penulis soal : Ashar Fuadi
Penulis editorial : Anthony
Tema : *lowest common ancestor*, *range query*

Catatan/Komentar

Untuk dapat menyelesaikan soal ini diperlukan pengetahuan mengenai *lowest common ancestor* (LCA) dan struktur data untuk melakukan *range query* seperti *Segment Tree* dan *Fenwick Tree*.

Secara singkat, inti permasalahan yang perlu diselesaikan pada soal ini yaitu jika LCA dari karyawan p dan q adalah LCA(p, q), hitung banyaknya atasan dari LCA(p, q) maupun LCA(p, q) sendiri yang mempunyai kekayaan lebih dari max(R[p], R[q]). Jumlahkan jawaban untuk setiap pasang karyawan p dan q.

Batasan kedua versi:

- $1 \leq P[i] < i$ untuk $i > 1$
- $1 \leq R[i] \leq 100.000$

Versi Mudah

Batasan: $\mathbf{2 \leq N \leq 2.000}$

Terdapat beberapa pendekatan yang dapat digunakan untuk menyelesaikan versi mudah dari soal ini. Salah satunya adalah menggunakan cara berikut.

Untuk setiap karyawan, buat daftar maksimum kekayaan setiap dua orang bawahannya yang mempunyai LCA sama dengan dirinya. Dengan kata lain, untuk setiap pasang karyawan p dan q, simpan informasi max(R[p], R[q]) pada karyawan LCA(p, q) (misalnya disimpan pada suatu *list*).

Setelah itu, lakukan *traversal* dari *root* (karyawan yang tidak mempunyai atasan). Dengan bantuan struktur data yang mendukung *range sum query*, setiap memasuki sebuah *node* u, lakukan hal berikut:

1. Tambahkan nilai 1 pada struktur data *range sum query* di indeks R[u] yang berarti terdapat tambahan 1 orang karyawan dengan kekayaan R[u].
2. Untuk setiap informasi max(R[p], R[q]) yang telah disimpan pada karyawan u, hitung banyaknya bilangan yang lebih besar dari max(R[p], R[q]) (*query* jumlahan bilangan pada posisi max(R[p], R[q]) + 1 sampai nilai maksimum kekayaan) dan tambahkan ke jawaban.
3. Kunjungi setiap anak dari *node* u (rekursif).
4. Kurangkan nilai 1 pada struktur data *range sum query* di indeks R[u] karena karyawan ke-u telah selesai diproses.

Versi Sulit

Batasan: $\mathbf{2 \leq N \leq 100.000}$

Untuk menyelesaikan versi sulit dari soal ini, diperlukan pemahaman terhadap konsep *lowest common ancestor* dari dua buah *nodes* untuk menghindari cara perhitungan yang dapat dilakukan pada versi mudah.

Pada versi mudah, perhitungan jawaban dapat dilakukan dengan mencoba mencari LCA dari setiap pasang karyawan. Untuk versi sulit, perhitungan yang dilakukan merupakan kebalikan dari versi mudah, yaitu untuk setiap karyawan, berapa banyak pasangan karyawan bawahannya yang nilai maksimum kekayaan keduanya kurang dari kekayaan dirinya.

Salah satu pendekatan yang dapat digunakan adalah dengan memproses karyawan-karyawan dengan nilai kekayaan dari yang tertinggi ke terendah. Saat sedang memproses karyawan-karyawan dengan kekayaan K, tandai semua karyawan tersebut sebagai sedang diproses. Untuk setiap karyawan tersebut, cari

berapa banyak karyawan bawahannya yang belum diproses, anggap ada sebanyak X, maka tambahkan $\frac{X(X-1)}{2}$ ke jawaban. Rumus $\frac{X(X-1)}{2}$ merupakan banyaknya cara pemilihan 2 orang karyawan dari X karyawan.

Agar dapat menghitung banyaknya bawahan yang belum diproses dengan mudah, representasi hubungan atasan-bawahan dapat diubah menjadi suatu daftar yang linier menggunakan teknik yang disebut *Euler Tour Tree*. *Euler Tour Tree* dilakukan menggunakan *pre-order tree traversal* dan menyimpan posisi atau indeks saat memasuki dan meninggalkan sebuah *node*.

Jika indeks ketika memasuki sebuah *node* u adalah B[u] dan ketika meninggalkannya adalah E[u], maka setiap *node* pada *sub-tree* dengan *root* u akan mempunyai nilai B[] dan E[] di dalam rentang [B[u], E[u]], sehingga untuk menghitung banyaknya bawahan u yang belum diproses, cukup lakukan *range sum query* pada rentang [B[u], E[u]].

Berbagi Lawakan

Penulis soal : William Gozali
Penulis editorial : William Gozali
Tema : *graph*, *memoization*

Catatan/Komentar

Batasan kedua versi: $1 \leq N, M, Q \leq 50000$

Versi Mudah

Batasan: $\mathbf{A[i] = 0; B[i] = 100}$

Tugas kita adalah menentukan apakah suatu *node* akan menuju ke suatu *cycle*.

Cara sederhananya adalah melakukan *Depth First Search* (DFS) mulai dari *node* yang ditanyakan, yaitu *node* x. Misalkan sejauh ini DFS telah membentuk *path* dari x sampai *node* p. Untuk seluruh *node* q yang dapat dikunjungi langsung dari p, periksa apakah q merupakan anggota dari *path* yang dibentuk dari x sampai p. Apabila ya, berarti lawakan dimulai dari *node* x menjadi abadi, dan tidak untuk kasus sebaliknya.

Hasil penelusuran dari DFS dapat digunakan untuk menjawab pertanyaan berikutnya. Untuk setiap *node*, catat informasinya sebagai salah satu dari nilai berikut:

- 0, bila *node* ini belum diketahui apakah ke depannya akan berakhir pada *cycle*.
- 1, bila *node* ini sudah diketahui akan berakhir pada *cycle*.
- 2, bila *node* ini sudah diketahui tidak akan berakhir pada *cycle*.

Awalnya seluruh *node* meiliki informasi bernilai 0.

Saat melakukan DFS untuk menjawab pertanyaan berikutnya, jika ditemui *node* yang memiliki informasi bernilai 1 atau 2, maka DFS dapat dihentikan dan jawaban langsung ditemukan.

Kompleksitas waktu solusi ini adalah $\mathcal{O}(M + N + Q)$.

Versi Sulit

Batasan: $\mathbf{0 \leq A[i] \leq B[i] \leq 100}$

Versi ini dapat diselesaikan dengan solusi pada versi mudah, tetapi dengan mempertimbangkan bahwa kali ini terdapat paling banyak 100 *graph* yang berbeda.

Anda dapat menggunakan *array* sebesar 100×50000 untuk menampung informasi dari tiap *node*, atau memproses seluruh pertanyaan dari yang nilai humornya paling rendah ke paling besar. Kompleksitas waktu solusi ini adalah $\mathcal{O}(Q + 100(N + M))$, atau tetap dapat dituliskan dengan $\mathcal{O}(M + N + Q)$.