Olimpiade Sains Nasional 2015

Bidang Informatika

Pembahasan

Scientific Committee OSN 2015

Catatan Scientific Committee

Scientific Committee OSN 2015 kali ini terdiri atas:

- Ahmad Zaky, IMO 2010-2012
- Christian Anthony Setiawan, TOKI 2014
- Nathan James, TOKI 2014
- Rakina Zata Amni, TOKI 2013
- Zamil Majdy, IOI 2014

Kami menerima berbagai soal dari alumni-alumni TOKI dan kontributor lainnya, hingga terpilih 10 soal sebagai *shortlist*, dan akhirnya berdasarkan persetujuan dari pembina dipilih 6 soal yang dikeluarkan sebagai soal OSN 2015. Dengan berbagai pertimbangan, problemset yang dikeluarkan *memang* lebih mudah relatif terhadap OSN tiga tahun terakhir. Meskipun demikian, terdapat tren bahwa soal termudah pada OSN ini *tidak lebih mudah* daripada soal termudah OSN sebelum-sebelumnya (mungkin kecuali OSN 2012). Hal ini mengikuti tren IOI, yaitu tingkat kesulitan tersebar pada subsoal, bukan pada soal itu sendiri.

Kami berterima kasih kepada pihak-pihak berikut, yang tanpa mereka OSN 2015 ini tidak mungkin berjalan dengan baik.

- Ashar Fuadi (IOI 2010) sebagai *advisor* dan *proofreader*
- Christianto Handojo (IOI 2010), Irvan Jahja (IOI 2008), dan Anthony (TOKI 2013) sebagai *tester*
- Technical Committee OSN 2015
- Para pembina dan alumni TOKI
- Serta pihak-pihak lain yang tidak mungkin disebutkan satu persatu yang turut berkontribusi

1A - Pertahanan Yogya

Pembuat Soal	: Alham Fikri Aji (IOI 2010)
Tingkat Kesulitan	: Mudah
Tag	: Greedy, Binary Search

Menentukan Hasil Pertarungan

Hal pertama yang harus dilakukan adalah menentukan hasil pertarungan antara dua petarung. Misalkan A akan melawan B dengan A menyerang duluan dan masing-masing mempunyai kemampuan berturut-turut (S_a, T_a) dan (S_b, T_b). Maka, setelah k kali diserang, stamina A akan menjadi $S'_a = S_a - kT_b$. Artinya, untuk membuat stamina A menjadi 0 atau kurang, harus berlaku $S'_a = S_a - kT_b \leq 0 \Leftrightarrow k \geq \frac{S_a}{T_b}$. Jadi, dibutuhkan $\left\lceil \frac{S_a}{T_b} \right\rceil$ kali penyerangan oleh B untuk membuat A kalah, dengan $\lceil x \rceil$ adalah bilangan bulat terkecil yang tidak kurang dari x. Hal yang sama berlaku untuk B.

Agar A menang, maka stamina B harus menjadi 0 atau kurang sebelum A. Jadi, $\left\lceil \frac{S_b}{T_a} \right\rceil \leq \left\lceil \frac{S_a}{T_b} \right\rceil$ adalah kondisi sedemikian sehingga A memenangkan pertandingan.

Subsoal 3: Complete Search

Untuk menyelesaikan subsoal ini, hal yang perlu dilakukan adalah mencoba semua urutan kemungkinan musuh. Kompleksitasnya adalah $\mathcal{O}(N!)$.

Subsoal 4: Brute Force

Selama masih ada musuh yang tersisa dan makhluk jahat (makhluk nomor 1) belum dapat dikalahkan, cari satu musuh yang bisa dikalahkan. Hal ini dapat dilakukan dengan menyimpan daftar musuh yang sudah dikalahkan dengan, misalnya, sebuah *array boolean*. Tiap kali kita ingin mengalahkan makhluk baru, lakukan iterasi pada seluruh makhluk sisanya, coba apakah dengan kemampuan sekarang kita dapat mengalahkan salah satunya, yang mana saja.

Kompleksitasnya adalah $\mathcal{O}(N^2)$.

Subsoal 5: Brute Force

Sebenarnya subsoal ini dapat diselesaikan dengan cara yang sama pada subsoal 4 Namun, batasan $L_t = 0$ memudahkan kita (dalam konteks kompleksitas waktu) untuk langsung menentukan berapa banyak makhluk yang perlu Pak Dengklek kalahkan (dengan kata lain, berapa banyak kemampuan tambahan yang diperlukan) sebelum dapat mengalahkan makhluk tertentu.

Setelah mengalahkan k makhluk, maka kemampuan Pak Dengklek akan menjadi $(S_d + kL_s, T_d)$ karena $L_t = 0$. Maka, syarat agar Pak Dengklek dapat mengalahkan makhluk dengan kemampuan (S_a, T_a) adalah $\left\lceil \frac{S_a}{T_d} \right\rceil \leq \left\lceil \frac{S_d + kL_s}{T_a} \right\rceil$. Karena $\left\lceil \frac{x}{y} \right\rceil = \left\lfloor \frac{x+y-1}{y} \right\rfloor$, untuk bilangan bulat positif x dan y, maka

$$\left\lceil \frac{S_a}{T_d} \right\rceil \leq \left\lceil \frac{S_d + kL_s}{T_a} \right\rceil = \left\lfloor \frac{S_d + kL_s + T_a - 1}{T_a} \right\rfloor \leq \frac{S_d + kL_s + T_a - 1}{T_a} \Rightarrow k \geq \frac{T_a \left\lceil \frac{S_a}{T_d} \right\rceil - S_d - T_a + 1}{L_s}$$

Jadi, nilai k terkecil adalah $\left\lceil \frac{T_a \left\lceil \frac{S_a}{T_d} \right\rceil - S_d - T_a + 1}{L_s} \right\rceil$ Nilai ini tentu saja dapat dihitung dengan kompleksitas $\mathcal{O}(1)$.

Subsoal 6: Greedy

Dengan rumus yang telah diperoleh sebelumnya, kita dapat menentukan banyak musuh minimal yang perlu dikalahkan untuk mengalahkan musuh tertentu. Kita hitung nilai ini untuk seluruh musuh yang ada, misalkan kita butuh z_i kemampuan tambahan untuk mengalahkan musuh ke-i. Urutkan semua musuh sesuai nilai z_i. Maka, musuh pertama yang dapat dikalahkan harus mempunyai nilai $z_i = 0$, musuh kedua yang dapat dikalahkan harus mempunyai nilai $z_i \leq 1$, dan seterusnya. Jika kita membutuhkan k kemampuan tambahan untuk mengalahkan makhluk jahat, maka periksa apakah k elemen pertama dari musuh yang telah diurutkan sesuai dengan nilai z memenuhi persyaratan tersebut. Jika iya, maka k elemen pertama tersebut adalah jawaban yang diinginkan. Jika tidak, maka tidak ada cara yang mungkin.

Kompleksitas: $\mathcal{O}(N \log N)$, atau $\mathcal{O}(N)$ jika menggunakan *counting sort*.

Subsoal 7: Greedy + Binary Search

Nilai z_i sebenarnya dapat ditentukan dengan *binary search* pada banyaknya musuh yang perlu dikalahkan. Kompleksitas yang diperlukan adalah $\mathcal{O}(N \log N)$ karena kita melakukan *binary search* sebanyak N kali untuk tiap musuh yang ada.

1B - Menyiram Sawah

Pembuat Soal	: Alham Fikri Aji (IOI 2010)
Tingkat Kesulitan	: Sulit
Tag	: Dynamic Programming, Flood Fill

Subsoal 3: Simulasi

Simulasikan saat sawah disiram dari kiri atas dan kanan bawah. Karena $N = 1$, maka aliran air pasti hanya satu arah saja. Kompleksitasnya adalah $\mathcal{O}(MQ)$.

Subsoal 4: Simulasi, Flood Fill

Simulasikan juga aliran air, namun harus menggunakan teknik *flood fill*. Kompleksitas (terburuk) adalah $\mathcal{O}(MNQ)$.

Subsoal 5

Cara 1: Greedy

Saat sub-sawah bisa dibasahi dari kiri (atas/bawah dihilangkan karena N = 1), maka pasti ketinggiannya menurun dari kiri ke kanan. Sebaliknya, ketinggian pasti menaik saat sawah dapat dibasahi dari kanan. Jadi, kita cukup menentukan apakah suatu sub-array $T_b \dots T_d$ menaik atau menurun, jika T_i adalah ketinggian sawah pada baris 1 dan kolom i.

Salah satu caranya adalah dengan membuat suatu array L_i, yang menyatakan indeks terkiri sehingga nilai $T_{L_i} ... T_i$ membentuk suatu barisan menurun. Sebagai contoh, jika $T = [2, 7, 6, 3, 3, 2, 8, 5]$, maka $L = [1, 2, 2, 2, 5, 5, 7, 7]$ (indeks dimulai dari 1). Definisikan R_i dengan cara yang sama, namun ke arah kanan. Seluruh nilai L_i dan R_i dapat dihitung dengan kompleksitas $\mathcal{O}(M)$.

Sekarang, sub-array $T_b \dots T_d$ menurun apabila $L_d \leq b$. Dengan cara yang sama, sub-array tersebut menaik jika $R_b \geq d$. Jadi, kompleksitasnya adalah $\mathcal{O}(M)$ untuk prekomputasi nilai L_i dan R_i, dan $\mathcal{O}(1)$ per *query*.

Cara 2: DP

Misalkan kita mempunyai suatu fungsi f sedemikian sehingga untuk $a < b$:

$$f(a,b) = \begin{cases} 1 & \text{jika sub-sawah } [1,\, a] \dots [1,\, b] \text{ dapat dibasahi dari kiri} \\ 2 & \text{jika sub-sawah } [1,\, a] \dots [1,\, b] \text{ dapat dibasahi dari kanan} \\ 0 & \text{jika bukan keduanya} \end{cases}$$

Untuk $a = b - 1$, nilai $f(a, b)$ ini dapat dihitung dengan hanya membandingkan tinggi petak di $[1, a]$ dan $[1, b]$. Selain itu,

$$f(a,b) = \begin{cases} 1 & \text{jika } f(a+1,b) = 1 \text{ dan } T_a > T_{a+1} \\ 2 & \text{jika } f(a,b-1) = 2 \text{ dan } T_b > T_{b-1} \\ 0 & \text{jika bukan keduanya} \end{cases}$$

Maka, seluruh nilai f dapat dihitung dalam $\mathcal{O}\left(M^2\right)$ dengan *dynamic programming* dan tiap *query* dapat dijawab dalam $\mathcal{O}\left(1\right)$.

Subsoal 6: DP Partial Sum

Definisikan sebuah petak *dominan* bila petak tersebut adalah petak yang tingginya tidak kurang dari tetangga-tetangganya.

Lemma 1. *Untuk membasahi semua petak, semua petak dominan perlu disiram.*

Bukti. Petak dominan tidak dapat dialiri oleh air dari keempat sisinya, karena tinggi petak tetangganya lebih dari atau sama dengan tinggi petak itu sendiri. Jadi, lemma terbukti. □

Lemma 2. *Untuk membasahi semua petak, kita cukup membasahi semua petak dominan saja.*

Bukti. Pilih suatu petak yang tidak dominan. Pasti ada salah satu petak tetangga yang tingginya lebih dari petak ini. Bila petak tetangga ini masih tidak dominan, kita pilih tetangga dari petak ini yang lebih tinggi darinya. Begitu seterusnya, hingga kita sampai pada petak yang dominan. Kita pasti bisa sampai pada petak dominan karena saat kita berpindah petak, ketinggiannya naik sehingga kita tidak mungkin "terjebak" dalam suatu *cycle*. Saat petak dominan ini disiram, maka melalui jalur penelusuran tadi, suatu saat petak yang paling pertama dipilih akan terbasahi. Dengan itu, maka seluruh petak tidak dominan akan terkena air dari suatu petak dominan. Terbukti. □

Maka, soal dapat berubah menjadi *apakah petak kiri atas atau kanan bawah adalah petak dominan dari suatu sub-sawah, dan apakah petak tersebut hanyalah satu-satunya petak yang dominan?*

Soal di atas dapat diselesaikan dengan DP *partial sum* 2D. Jadi, buat sebuah matriks 2D $D_{i,j}$ yang memenuhi

$$D_{i,j} = \begin{cases} 1 & \text{jika petak pada baris } i \text{ dan kolom } j \text{ dominan} \\ 0 & \text{jika tidak} \end{cases}$$

Maka untuk tiap *query*, kita cukup memeriksa apakah *partial sum* D dari $[a, b] \ldots [c, d]$ bernilai 1 atau tidak. Bila iya, periksa apakah $D_{a,b}$ atau $D_{c,d}$ bernilai 1 atau tidak. Bila $D_{a,b} = 1$, maka seluruh petak dapat dibasahi dengan menyiram petak kiri atas, dan jika $D_{c,d} = 1$, maka jawabannya adalah kanan bawah.

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

Seluruh petak dominan ditandai dengan warna biru

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

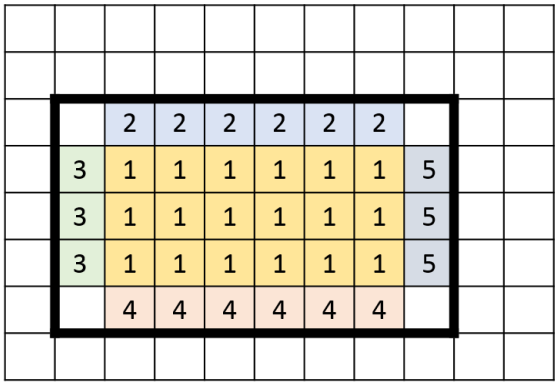
Perhatikan bahwa suatu petak dapat menjadi dominan di suatu sub-sawah tertentu

Masalahnya sekarang adalah petak yang tidak dominan di sawah secara keseluruhan bisa menjadi petak dominan di suatu sub-sawah $[a, b] \ldots [c, d]$. Gambar di atas menjelaskan hal tersebut. Oleh karena itu, perhitungan petak dominan di tengah sub-sawah dan di pinggir sub-sawah harus dibedakan. Salah satu caranya, kita dapat membuat enam *partial sum*:

1. Petak yang dominan terhadap keempat tetangganya.
2. Petak yang dominan terhadap seluruh tetangga kecuali petak di atasnya.
3. Petak yang dominan terhadap seluruh tetangga kecuali petak di kirinya.
4. Petak yang dominan terhadap seluruh tetangga kecuali petak di bawahnya.
5. Petak yang dominan terhadap seluruh tetangga kecuali petak di kanannya.
6. Petak yang dominan terhadap tetangga di kiri dan kanannya.

Petak di keempat sudut sub-sawah dapat ditentukan secara langsung apakah itu dominan atau tidak. Misalkan $D_i([a,b]\ldots[c,d])$ adalah *partial sum* ke-i dari $[a,b]$ sampai $[c,d]$. Maka, untuk menjawab sebuah *query* $[a,b]\ldots[c,d]$:

- Jika $a \neq c$, maka hitung $D_1([a+1,b+1]\ldots[c-1,d-1])+D_2([a,b+1]\ldots[a,d-1])+D_3([a+1,b]\ldots[c-1,b])+D_4([c,b+1]\ldots[c,d-1])+D_5([a+1,d]\ldots[c-1,d])$. Bila nilai ini tidak nol, maka jawabannya adalah `TIDAK MUNGKIN`. Bila nol, maka periksa keempat titik sudut sub-sawah: $[a,b],[a,d],[c,b],[c,d]$. Bila petak $[a,b]$ adalah satu-satunya petak dominan dari keempat petak tersebut, maka jawabannya adalah `KIRI ATAS`. Jika petak $[c,d]$ adalah satu-satunya petak dominan, maka jawabannya adalah `KANAN BAWAH`. Selain itu, jawabannya pasti `TIDAK MUNGKIN`.
- Jika $a = c$, maka hitung $D_6([a,b+1]\ldots[a,d-1])$. Nilai ini harus sama dengan nol. Kemudian, sama seperti kasus sebelumnya, periksa kedua petak $[a,b]$ dan $[c,d]$.



Ilustrasi di atas menunjukkan bagaimana menghitung banyak petak dominan di suatu sub-sawah. Angka menunjukkan *partial sum* mana yang harus dihitung untuk petak yang mana

Kompleksitasnya adalah $\mathcal{O}(MN)$ untuk prekomputasi seluruh *partial sum* dan $\mathcal{O}(1)$ untuk menjawab tiap *query*.

Catatan

- Syarat petak dominan adalah petak tersebut harus memiliki tinggi *lebih dari atau sama dengan* petak tetangganya, bukan *lebih dari*. Hal ini karena petak yang bersebelahan yang memiliki tinggi yang sama tidak bisa mengaliri air satu sama lain.
- Ada versi lain soal ini, yaitu *tentukan banyaknya petak minimal yang harus disiram agar semua sub-sawah terbasahi*. Namun, dengan begitu maka petunjuk ke solusi yang melibatkan petak dominan akan lebih terlihat.

1C - Bisa Jadi Tebak Angka

Pembuat Soal	: Ivan Adrian Koswara (IMO 2012)
Tingkat Kesulitan	: Sedang
Tag	: Ad Hoc, Interactive

Subsoal 3 dan 4: By Hand

Solusi subsoal ini bisa beragam, dan kami menyarankan pembaca untuk mencoba-coba sendiri. Meskipun demikian, solusi untuk subsoal 5 dan 7 tentu saja dapat menyelesaikan subsoal ini.

Subsoal 6: Binary Search

Bila soal disederhanakan dengan menganggap kembalian dari program juri hanya berupa `YA` atau `TIDAK` (balasan `BISAJADI` dianggap `TIDAK`) maka soal ini dapat diselesaikan dengan binary search. Tebak angka pada setengah bagian pertama barisan yang dimiliki sekarang, bila balasan berupa `YA` maka barisan yang dimiliki sekarang adalah bagian pertama barisan awal, sedangkan bila balasan berupa `BISAJADI` atau `TIDAK` maka barisan sekarang adalah setengah bagian kedua. Ulangi ini sampai tersisa satu angka.

Subsoal 5 dan 7: Ternary Search

Perhatikan bahwa karena tiap pertanyaan bisa mendapatkan tiga respon berbeda, maka dengan k pertanyaan, kita hanya bisa mendapatkan 3^k kemungkinan serangkaian respon berbeda. Oleh karena itu, hal terpenting adalah kita harus menjaga kondisi bahwa kandidat jawaban harus tidak lebih dari 3^k saat pertanyaan tinggal tersisa k. Berikut adalah *salah satu* cara yang mudah diimplementasikan.

Pertama-tama, kandidat solusi adalah $1 \ldots N$. Kandidat ini diperlukan untuk dibagi menjadi 3 setiap bertanya agar dapat mendapat jawaban dalam jumlah pertanyaan yang dapat ditanyakan. Setelah tiap pertanyaan, kandidat solusi ini mungkin berubah-ubah. Namun, kandidat solusi pasti merupakan salah satu dari kedua pilihan di bawah:

- Kondisi 1: Kandidat solusi yang dimiliki berisi semua bilangan bulat dalam suatu rentang tertentu. Lakukan pertanyaan berisi bilangan ganjil sampai $\frac{2}{3}$ banyak bilangan dalam deret tersebut.
  - Bila jawaban berupa `YA` maka bilangan yang dicari terdapat dalam bilangan-bilangan ganjil di 2/3 bagian pertama.
  - Bila jawaban berupa `BISAJADI` maka bilangan yang dicari terdapat di antara bilangan-bilangan genap di 2/3 bagian pertama.
  - Bila jawaban berupa `TIDAK` maka bilangan yang dicari terdapat pada 1/3 deret di luar.

  Contohnya, bila kandidat solusi adalah 1 2 3 4 5 6 7 8 9, maka tebak bilangan 1 3 5. Bila jawaban `YA` maka bilangan terdapat di antara 1, 3, atau 5. Bila jawaban `BISAJADI` maka bilangan terdapat di antara 2, 4, atau 6. Bila jawaban `TIDAK` maka bilangan terdapat di antara 7, 8, atau 9.
- Kondisi 2: Deret bilangan yang dimiliki berupa bilangan genap saja atau bilangan ganjil saja dalam suatu rentang tertentu.

  Misalkan bilangan yang terdapat di deret sekarang adalah X. Untuk 1/3 bagian pertama dari deret yang dimiliki tebak X-1, untuk 1/3 bagian kedua tebak X, dan abaikan 1/3 bagian terakhir.

- Bila jawaban berupa `YA` maka bilangan yang dicari terdapat pada 1/3 bagian kedua.
- Bila jawaban berupa `BISAJADI` maka bilangan yang dicari terdapat pada 1/3 bagian pertama.
- Bila jawaban `TIDAK` maka bilangan yang dicari terdapat pada 1/3 bagian terakhir.

Contohnya, bila kandidat solusi adalah 2 4 6, maka tebak bilangan 1 dan 4. Bila jawaban berupa `YA` maka bilangan yang dicari adalah 4. Bila jawaban berupa `BISAJADI` maka bilangan yang dicari adalah 2. Bila jawaban `TIDAK` maka bilangan yang dicari adalah 6.

Perhatikan bahwa bagaimanapun responnya, kandidat solusi pasti merupakan barisan aritmetika yang berselisih 1 atau 2, jadi kita pasti bisa kembali ke salah satu kondisi di atas setelah kandidat solusi berubah.

2A - Belanja di Malioboro

Pembuat Soal	: William Gozali (IOI 2011)
Tingkat Kesulitan	: Sulit
Tag	: BFS, Math

Subsoal 3: Brute Force

Untuk setiap *query*, hitung posisi bebek setelah t detik. Bebek ke-i akan berada pada posisi $P_i + tU_i \mod M$ setelah t detik. Untuk setiap pasang bebek, hitung jarak di antara mereka berdua.

Kompleksitasnya adalah $\mathcal{O}(KN^2)$.

Subsoal 4: Simulasi

Sama seperti sebelumnya hitung posisi bebek setelah t detik. Namun, daripada harus membandingkan seluruh pasangan bebek, kita cukup mengurutkan posisi ke-N bebek tersebut dan kemudian membandingkan posisi bebek-bebek yang bersebelahan. Hal ini bisa juga dilakukan dengan cara menandai seluruh toko, apakah ada bebek pada toko ke-x pada waktu t, dan kemudian untuk setiap toko yang ada bebeknya, kita cari bebek yang terdekat. Proses ini sebenarnya sama saja dengan *counting sort*.

Kompleksitasnya adalah $\mathcal{O}(KN \log N)$ atau $\mathcal{O}(K(M+N))$ dengan *counting sort*.

Subsoal 5: Simulasi + Observasi

Observasi tambahan yang perlu dilakukan hanyalah posisi bebek-bebek pada waktu t sama saja dengan pada waktu $t + M$. Artinya, hanya ada M kemungkinan posisi yang berbeda. Kita hitung seluruh kemungkinan tersebut untuk $t = 0 \ldots M-1$ dan hitung jarak terpendek antara dua bebek seperti pada subsoal sebelumnya. Lalu, untuk setiap *query*, lakukan modulo M pada t yang diberikan, dan keluarkan jawaban yang telah diprekomputasi sebelumnya.

Kompleksitasnya adalah $\mathcal{O}(K + MN \log N)$ atau $\mathcal{O}(K + M^2 + MN)$.

Subsoal 6: BFS

Perhatikan suatu pasangan bebek ke-a dan b. Jika kita plot jarak antara keduanya terhadap waktu, maka ada dua kemungkinan:

- Jika $U_a = U_b$, maka jaraknya konstan untuk setiap waktu. Ambil nilai ini sebagai *upper bound* dari jawaban.
- Jika $U_a \neq U_b$, maka grafiknya akan berbentuk zig-zag. Artinya, mereka akan menjauh sampai jarak maksimal yaitu $\frac{M}{2}$, kemudian akan mendekat sampai bertemu (jarak = 0), kemudian menjauh lagi, dan seterusnya. Tentu saja grafik ini bisa dimulai dengan mereka berdua mendekat terlebih dahulu.

Misalkan jarak bebek ke-a dan b pada waktu t adalah $f(a, b, t)$. Maka, hal yang perlu dilakukan adalah menggabungkan seluruh grafik $f(a, b, t)$ untuk semua kemungkinan pasangan bebek. Jarak yang diminta adalah nilai f terkecil pada nilai t tertentu. Untuk memudahkan perhitungan, kita akan menghitung seluruh

nilai minimal f (sebut nilai minimal ini $g(t)$) pada waktu kelipatan $\frac{1}{2}$, untuk menghindari pembagian kasus antara bebek yang bertemu pada waktu bilangan bulat dan tidak.

Maka, untuk t kelipatan $\frac{1}{2}$, nilai $g(t) = 0$ saat ada dua bebek yang bertemu di waktu t, dan

$$g(t) = \min\left(g\left(t - \frac{1}{2}\right), g\left(t + \frac{1}{2}\right)\right) + 1$$

karena pada suatu waktu t kelipatan $\frac{1}{2}$ di mana tidak ada bebek yang bertemu, pasti ada pasangan bebek yang mendekat atau menjauh.

Dengan fakta tersebut, kita dapat menghitung seluruh nilai g dengan BFS. Pertama, cari seluruh t di mana $g(t) = 0$. Masukkan seluruh nilai t ini ke sebuah antrian q. Seperti BFS pada umumnya, tiap saat ambil elemen t terdepan q, dan kemudian simpan nilai $g\left(t + \frac{1}{2}\right) = g(t) + 1$ jika nilai $t + \frac{1}{2}$ belum ada. Masukkan $t + \frac{1}{2}$ ke q. Lakukan hal yang sama dengan $t - \frac{1}{2}$. Tentunya, nilai g ini harus tidak lebih dari jarak minimal yang diperoleh dari pasangan bebek dengan arah gerak yang sama.

Kompleksitasnya adalah $\mathcal{O}\left(N^2 + M + K\right)$, yaitu $\mathcal{O}\left(N^2\right)$ untuk menghitung semua t untuk pasangan bebek yang berbeda arah, $\mathcal{O}\left(M\right)$ untuk melakukan BFS, dan $\mathcal{O}\left(K\right)$ untuk menjawab seluruh *query*.

Subsoal 7: BFS

Cara menyelesaikan subsoal ini sama saja dengan subsoal 6 ditambah observasi yang sama pada subsoal 5.

Catatan

- Terdapat solusi $\mathcal{O}\left(N^2 \log N + K \log N\right)$ yang tidak bergantung pada nilai M. Caranya adalah untuk setiap *query*, kita mencari titik (waktu) terdekat di mana terdapat dua bebek yang bertemu pada titik tersebut. Detailnya diserahkan kepada pembaca sebagai latihan.
- Bahkan terdapat solusi dengan kompleksitas $\mathcal{O}\left(N + (M + K) \log M\right)$ yang dapat mengatasi nilai N yang besar. Namun, solusi ini sangat sulit untuk ditemukan dan diimplementasikan. *Hint*: algoritma *fast Fourier transform* untuk mengalikan dua buah polinomial.

2B - Motif Batik

Pembuat Soal	: William Gozali (IOI 2011)
Tingkat Kesulitan	: Mudah
Tag	: Counting, Sorting

Subsoal 3: Brute Force

Cukup hitung selisih untuk setiap pasang batik yang ada, bila mereka berbeda warna. Kompleksitasnya adalah $\mathcal{O}\left(N^2\right)$.

Subsoal 4, 5, 6: Partial Sum

Hal yang membedakan ketiga subsoal ini adalah nilai M dan W_i yang cukup kecil. Intinya adalah kita dapat menghitung frekuensi batik yang memiliki nilai C_i dan W_i tertentu. Partial sum tersebut diserahkan kepada pembaca sebagai latihan (pendekatannya berbeda-beda untuk setiap subsoal).

Subsoal 7: Sorting, Partial/Prefix Sum

Karena semua batik berbeda motif, maka sama saja dengan menghitung total dari selisih semua pasang batik. Hal ini dapat dilakukan dengan mudah saat tingkat kecerahan batik telah terurut menaik. Maka, untuk $i < j$, $|W_i - W_j| = W_j - W_i$. Secara matematis,

$$\sum_{i<j} |W_i - W_j| = \sum_{i=1}^{n} \sum_{j=1}^{i} |W_i - W_j| = \sum_{i=1}^{n} \sum_{j=1}^{i} W_i - W_j = \sum_{i=1}^{n} \left(i \times W_i - \sum_{j=1}^{i} W_j \right)$$

Ekspresi di atas dapat dihitung dalam $\mathcal{O}(N)$. Jadi, total kompleksitas adalah $\mathcal{O}(N \log N)$.

Subsoal 8: Sorting, Partial/Prefix Sum

Jika semua motif batik berbeda, maka jawabannya adalah seperti pada subsoal 7. Setelah menghitung nilai ini, maka kita perlu mengurangkan selisih yang "tidak diperlukan", yaitu selisih antara batik-batik yang warnanya sama. Hal ini dapat dilakukan dengan menghitung hal yang sama, tapi pada tiap kelompok batik yang warnanya sama.

Kompleksitasnya tetap $\mathcal{O}(N \log N)$.

Catatan

- Dari seluruh submisi pada saat kontes (baik resmi maupun Open Contest), tidak ada yang mengerjakan sampai subsoal 4, 5, atau 6. Mungkin karena solusi untuk subsoal 7 atau solusi penuh lebih mudah diimplementasi daripada subsoal-subsoal ini.

2C - Ayam Aneh

Pembuat Soal	: Tracy Filbert (TOKI 2010), dengan perubahan
Tingkat Kesulitan	: Mudah
Tag	: Ad Hoc, String, Binary Search

Subsoal 3

Kita cukup mencari urutan permutasi A-Z yang memenuhi. Pertama-tama, mulai dengan mencari huruf di sebelah kanan A. Lakukan `TANYA A?` untuk ? dari B-Z, sampai mendapatkan respon `YA`. Jika seluruh pertanyaan mendapatkan respon `TIDAK`, maka huruf A ada di paling kanan string DNA. Jika tidak, misalkan huruf kedua ini adalah B. Lakukan hal yang sama, `TANYA B?` untuk seluruh huruf sisanya. Lakukan terus sampai huruf paling kanan ditemukan. Terakhir, ulangi untuk menentukan urutan huruf di depan A.

Diperlukan paling banyak $25+25+24+23+\cdots+1 = 350$ pertanyaan jika mengikuti cara di atas, dengan kasus terburuk pada string `BCDEFGHIJKLMNOPQRSTUVWXYZA`. Banyak interaksi dapat berkurang 1 karena pertanyaan terakhir pasti mendapatkan jawaban `YA`.

Subsoal 4: Linear Search

Untuk memudahkan, kita perlu mengidentifikasi huruf mana saja yang muncul dan berapa kali huruf tersebut muncul, dengan cara melakukan `TANYA A`, `TANYA AA`, `TANYA AAA`, ... sampai mendapatkan jawaban `TIDAK`. Banyak pertanyaan yang dibutuhkan paling banyak adalah $N + 26$.

Terakhir, cukup melakukan hal yang sama seperti subsoal 3 untuk menentukan urutan huruf. Pada kasus terburuknya, dibutuhkan $N + 26 + 350$ interaksi, atau saat $N = 100$, nilai ini sama dengan 476 (atau 475 jika perbandingan terakhir tidak dilakukan).

Subsoal 5: Binary Search

Caranya sama seperti subsoal 4, hanya saja perlu dilakukan *binary search* menggantikan *linear search* untuk menentukan banyak kemunculan tiap huruf.

Catatan

- Batasan K sengaja dibuat agak tinggi (kecuali untuk subsoal 3) karena soal ini memang ditujukan sebagai soal bonus. Kami ingin mengetahui pendekatan-pendekatan berbeda yang mungkin muncul dari tiap peserta.
- Solusi *binary search* pada solusi resmi tidak dapat memenuhi subsoal 4 sekaligus pada kasus terburuknya. Meskipun demikian, beberapa solusi peserta tidak membutuhkan pembagian kasus antara subsoal 4 dan 5.