COMPFEST 9

Pembahasan Coder Class

SCPC - Minggu 2

COMPFEST 9

Daftar Soal

A. Euis
B. XOR A B
C. Random Generator
D. Jualan Balon
E. Amar dan Kina
F. Jajar Genjang Pascal
G. Relokasi Warga KaliJodie
H. Menyelamatkan Mbak Miku

A. Euis

Tag

ad hoc

Pembahasan

Perhatikan bahwa cara baca tiap *eu* maupun *e* independen satu sama lain. Oleh karena itu, kita dapat menghitung banyak cara membaca menggunakan aturan perkalian. Lakukan iterasi untuk menghitung kemunculan *eu* dan *e*. Misal kemunculan *eu* adalah a, dan *e* adalah b. Maka, hasil akhirnya adalah $3^a \cdot 2^b$.

Perhatikan bahwa *e* yang merupakan bagian dari *eu* tidak dimasukkan dalam penghitungan a. Selain itu, kesalahan yang sering terjadi adalah menggunakan tipe data 32 bit bertanda dalam menampung jawaban, karena dapat terjadi *overflow* ketika perkalian dengan 3.

Kompleksitas: $O(|S|)$

B. XOR A B

Tag

ad hoc

Pembahasan

Pertama, kita buat dulu $f(x) = 0 \oplus 1 \oplus ... \oplus x$. Ini akan mempermudah pencarian solusi nantinya. Namun, bagaimana cara menghitung $f(x)$ dengan cepat?

Untuk mempermudah pekerjaan, kita dapat melakukan perhitungan untuk masing-masing bit. Apabila diamati, nilai *xor* dari bit ke-i, pasti bernilai 0 pada kelipatan 2^{i+1} kecuali pada bit ke-0 (menjadi kasus khusus). Karena itu kita dapat melihat hasil dari bit ke-i dengan menghitung hasil dari x $(mod\ 2^{i+1})$, sebut angka tersebut y. apabila $y < 2^i$, maka hasil *xor*-nya pasti 0. Apabila $y - 2^i$ adalah genap, maka nilai *xor* untuk bit tersebut adalah 1, dan 0 apabila ganjil.

Nah, bagaimana cara memanfaatkan $f(x)$ untuk menghitung pada suatu rentang $[A, B]$? Kita bisa memanfaatkan salah satu properti operasi *xor*, yaitu $x \oplus x = 0$. Untuk pertanyaan dengan rentang $[A, B]$, dapat dihitung sebagai $f(B) \oplus f(A-1)$. Ini karena:

$$f(A-1) \oplus f(B)$$

$$= f(A-1) \oplus f(A-1) \oplus A \oplus (A+1) \ldots \oplus (B)$$

$$= 0 \oplus A \oplus (A+1) \ldots \oplus (B)$$

$$= A \oplus (A+1) \ldots \oplus (B)$$

Maka, kita bisa menjawab tiap pertanyaan dengan cepat, mengingat kompleksitas solusi di atas adalah $O(logA + logB)$. Sisanya, yang perlu diperhatikan adalah mengonversinya ke biner.

Dalam pengerjaannya, hati-hati pada bit ke-63, perhatikan setiap operasi agar tidak *overflow*. Selain itu, hati-hati pada kasus dimana $A = 0$.

Selain solusi di atas, ada solusi lain untuk mencari $f(x)$. Hint: perhatikan nilainya untuk setiap modulo 4.

Kompleksitas: $O(logA + logB)$

C. Random Generator

Tag

Math, kombinatorik

Pembahasan

Solusi 1: Observasi

Seperti definisi dari *expected value*, kontribusi dari sebuah susunan bit adalah hasil kali dari nilai susunan bit tersebut dan peluang munculnya susunan bit tersebut.

Contoh: susunan "10001" memiliki nilai (16 + 1) dan peluang $(\frac{P}{100})^2 \cdot (\frac{100-P}{100})^3$. Maka kontribusi dari "10001" adalah $(16+1) \cdot (\frac{P}{100})^2 \cdot (\frac{100-P}{100})^3$.

Salah satu strategi yang bisa dipakai adalah mencari kontribusi tiap bit pada *expected value*. Misal, kita ingin mencari kontribusi bit ke-$(N-1)$. Tentunya, bit ke-$(N-1)$ akan memberi kontribusi apabila ia bernilai 1. Oleh karena itu, kontribusinya:

$$2^{N-1} \cdot \frac{P}{100} \cdot \sum_{0}^{2^{N-1}-1} (\frac{P}{100})^{active\ bit\ in\ i} \cdot (\frac{100-P}{100})^{inactive\ bit\ in\ i}$$

Apabila dihitung, ternyata

$$\sum_{0}^{2^{N-1}-1} \left(\frac{P}{100}\right)^{active\ bit\ in\ i} \cdot \left(\frac{100-P}{100}\right)^{inactive\ bit\ in\ i}$$

pasti bernilai 1! Penjelasan intuitifnya adalah, apapun nilai bit ke-$(N-1)$, konfigurasi nilai bit sisanya pasti berupa 0/1, dan pasti valid. Hal ini terjadi akibat kita menginginkan *expected value* dari 0 sampai $2^N - 1$. Karena pasti valid, nilainya pasti 1. Sehingga, total kontribusi bit ke-$(N-1)$ adalah $2^{N-1} \cdot \frac{P}{100}$.

Bila diamati lebih lanjut, kontribusi bit ke-$(N-1)$ bisa digeneralisasi untuk semua bit ke-i. Akhirnya, setiap bit ke-i memiliki kontribusi sebesar $2^i \cdot \frac{P}{100}$ untuk setiap $0 \leq i < N$. Jadi, expected valuenya adalah:

$$\sum_{0}^{N-1} 2^i \cdot \frac{P}{100}$$

$$= \frac{P}{100} \cdot \sum_{0}^{N-1} 2^i$$

$$= \frac{P}{100} \cdot (2^N - 1)$$

Solusi 2: Rekursif

Solusi ini menggunakan strategi rekursif dalam penyelesaiannya. Bentuk umum dari rekursif untuk *expected value* adalah F(x) = transisi_i * kemungkinan_i + transisi_{i+1} * kemungkinan_{i+1} + …

Pada soal ini yang dapat diiterasi adalah bitnya. Dan terdapat 2 kemungkinan transisi untuk setiap bit, yaitu 0 dan 1.

Contoh: sekarang di state 10, bila ditambah 1 bit di belakangnya terdapat 2 kemungkinan, yaitu 100 ($(100 - P)$ %) atau 101(P %).

Maka, kita bisa mendefinisikan $f(x)$ sebagai *expected value* apabila hanya tersisa x bit lagi, dengan pada setiap transisi, kita meletakkan *most significant bit*-nya. Penyelesaiannya adalah sebagai berikut:

$$f(0) = 0$$

$$f(N) = (transisi\ bit\ 1) \cdot \frac{P}{100} + (transisi\ bit\ 0) \cdot \frac{100 - P}{100}$$

$$= (f(N-1) + 2^{N-1}) \cdot \frac{P}{100} + f(N-1) \cdot \frac{100 - P}{100}$$

$$= f(N-1) \cdot \frac{P}{100} + 2^{N-1} \cdot \frac{P}{100} + f(N-1) \cdot \frac{100 - P}{100}$$

$$= f(N-1) + 2^{N-1} \cdot \frac{P}{100}$$

Apabila disimplifikasi, dapat didapatkan lagi rumus berikut:

$$f(N) = \sum_{0}^{N-1} 2^i \cdot \frac{P}{100}$$

$$= \frac{P}{100} \cdot \sum_{0}^{N-1} 2^i$$

$$= \frac{P}{100} \cdot (2^N - 1)$$

Kompleksitas: $O(1)$ $atau$ $O(N)$

D. Jualan Balon

Tag

ad hoc, simulasi

Pembahasan

Untuk menyelesaikan soal ini, kita cukup menyimulasikan pembelian yang terjadi. Untuk menyimulasikannya, kita dapat menggunakan bantuan struktur data *queue*. Perhatikan bahwa dalam simulasinya, terdapat 2 jenis antrean yaitu (1) antrean pembeli di setiap penjual, dan (2) antrean balon yang ingin dibeli setiap pembeli.

Untuk (1), kita cukup membuat 3 buah antrean, sedangkan untuk (2), kita bisa membuat N buah antrean. Selanjutnya, kita tinggal menyimulasikan pembelian sesuai perintah soal. Tentunya, seorang pembeli akan membeli sebanyak-banyaknya balon di antreannya sesuai penjual yang sedang ia kunjungi. Jika sudah, periksa apakah ia masih ingin membeli balon, dan jika iya, masukkan ke antrean penjual yang bersesuaian.

Kesalahan yang cukup sering terjadi untuk soal ini adalah salah memahami kapan seseorang bisa membeli tak hingga balon. Seringkali, kita menganggap bahwa seseorang bisa membeli tak hingga balon ketika antrean di penjual yang ia kunjungi hanya berisi dirinya, padahal bukan. Seseorang dapat membeli tak hingga balon sekaligus pada satu penjual hanya jika tinggal ia yang masih ingin membeli balon (antrean di penjual lain harus kosong, dan hanya dia yang mengantre

di penjual sekarang).

Kompleksitas: $O(N * jumlah\_balon)$

E. Amar dan Kina

Tag

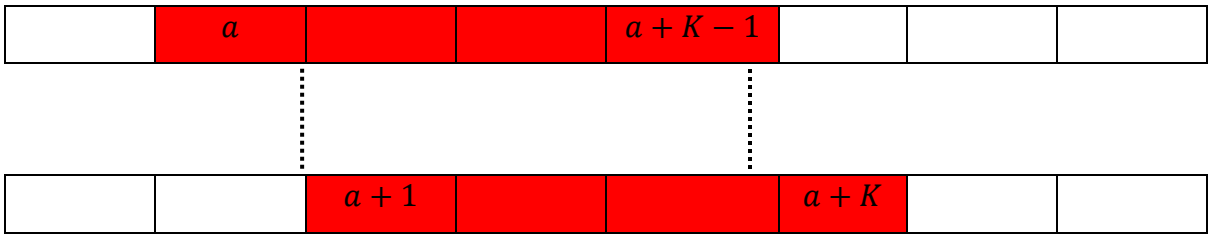
ad hoc, sliding window

Pembahasan

Kita mulai dari solusi *brute force* dari soal ini. Kita coba hitung untuk setiap subarray berukuran K, ada berapa nilai x, sehingga kemunculan x sama dengan x. Memeriksanya bisa menggunakan bantuan array, sehingga untuk satu subarray, pemeriksaan berjalan selama $O(K)$. Namun, bagaimana kita bisa menggunakan array untuk menghitung kemunculan, jika A_i bisa bernilai 10^9? Observasi penting: kita bisa mengabaikan perhitungan kemunculan semua $A_i$yang bernilai lebih besar dari K, karena kemunculannya tidak mungkin mencapai A_i. Sehingga, ukuran array untuk menghitung cukup sebesar K.

Masalahnya, terdapat $N - K + 1$ subarray yang berukuran K. Jika dilakukan *brute force*, kompleksitasnya akan mencapai $O(NK)$, yang akan terlalu lambat. Bagaimana cara mengoptimasinya?

Misal, kita memiliki array bantu hitung bernama cnt dan variabel cnt_valid untuk mencatat banyaknya x, sehingga $cnt[x] = x$. Misal, kita memiliki data cnt dan cnt_valid untuk suatu subarray$[a, a + K - 1]$. Bagaimana cara meng-update data tersebut untuk menjadi data pada subarray$[a + 1, a + K]$? Perhatikan gambar berikut:



Ya, ketika menggeser "jendela" (subarray), kita mendapati bahwa hanya ada 2 perbedaan, yaitu membuang elemen ke-a, dan menambahkan elemen ke-$(a+K)$. Nah, bagaimana cara meng-update cnt_valid? Setiap kali kita mengubah kemunculan x dimana $cnt[x] = x$, baik menambah maupun mengurangi kemunculan, maka cnt_valid pasti berkurang sebanyak 1. Sebaliknya, ketika akibat pengubahan, $cnt[x]$ menjadi bernilai x, maka cnt_valid pasti bertambah sebanyak 1.

Maka, algoritmanya adalah seperti berikut: Buat data cnt dan cnt_valid untuk subarray $[1, K]$ terlebih dahulu. Lalu, setiap menggeser "jendela", update data cnt dan cnt_valid. Akhirnya, jawaban merupakan jumlahan dari cnt_valid untuk setiap subarray berukuran K.

Kompleksitas: $O(N)$

F. Jajar Genjang Pascal

Tag

Math, kombinatorik

Pembahasan

Pembahasan ini akan menggunakan *0-based indexing*, sehingga N dan M pada pembahasan merupakan $N-1$ dan $M-1$ dari soal. Selain itu, pembahasan menggunakan banyak gambar.

Secara umum, ada dua rumus kombinasi yang dikenal, yaitu:

1. $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$
2. $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$

Dalam menyelesaikan soal ini, kita akan mengeksploitasi kedua rumus tersebut. Mula-mula, ketimbang segitiga sama kaki di soal, akan lebih mudah jika melihatnya sebagai segitiga siku-siku. Berikut beberapa contoh, dengan jawaban adalah jumlahan isi sel-sel yang diarsir merah:

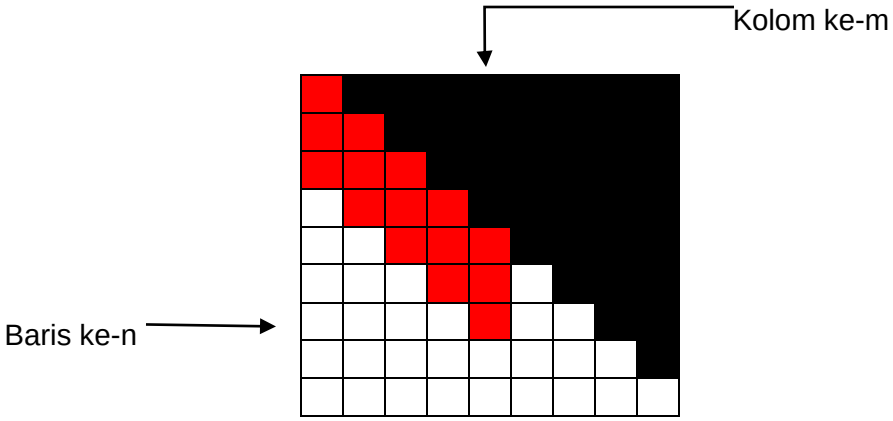
	0	1	2	3	4	5	6	7	8
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	

$N = 3, M = 2, L = 2, R = 3$

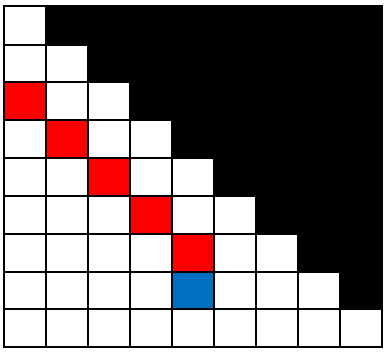
	0	1	2	3	4	5	6	7	8
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	

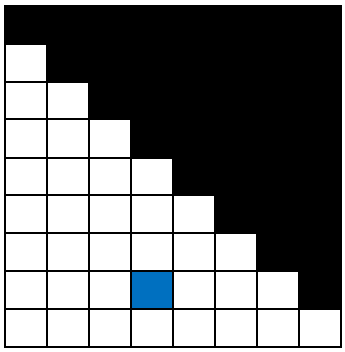
$N = 2\ M = 1, L = 4, R = 3$

Biasanya, soal-soal seperti ini menimbulkan aroma penggunaan teknik seperti*partial sum 2D.* Akan tetapi, berhubung nilai N dan M yang sangat besar, ini tidak dapat dilakukan secara naif. Untungnya, kita bisa memanfaatkan properti rumus kombinasi. Akan dibuktikan bahwa jumlah isi sel-sel yang diarsir berikut ini

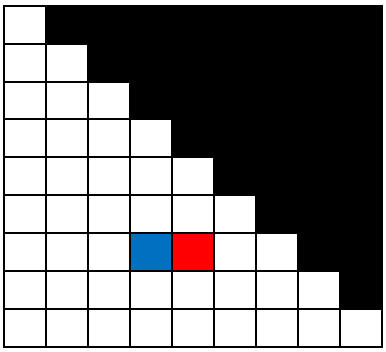


adalah $\binom{n+2}{m+1} - 1$.Pertama-tama, kita akan mendapatkan bahwa jumlahan sel-sel berwarna merah dan biru adalah sama, dengan bukti di bawahnya:

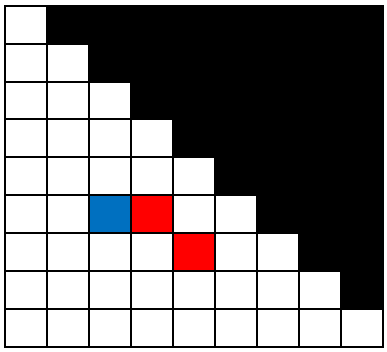




Awalnya, sebuah sel biru

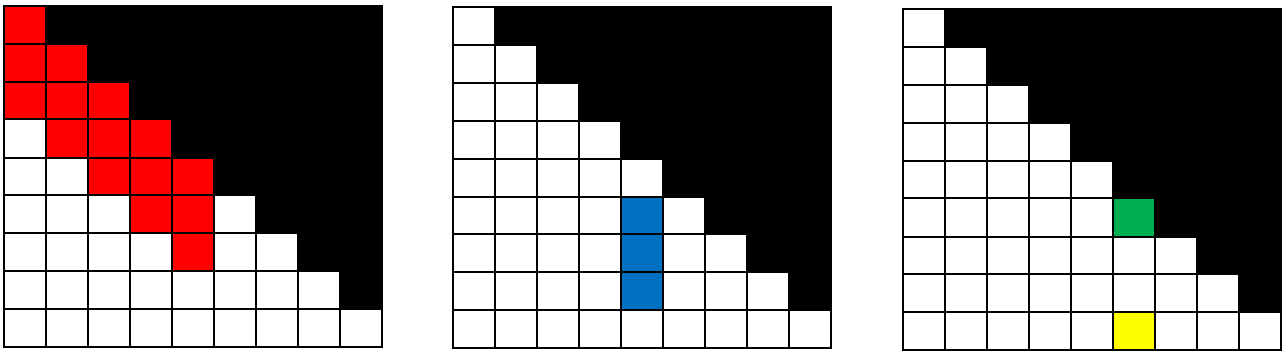


Menggunakan rumus 1...

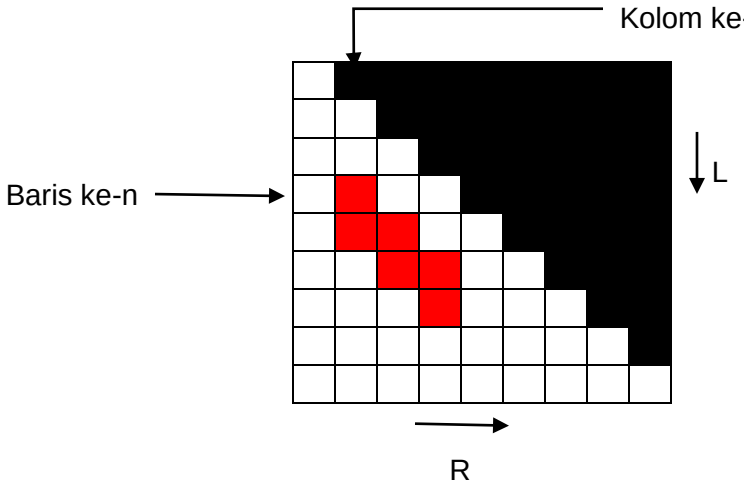


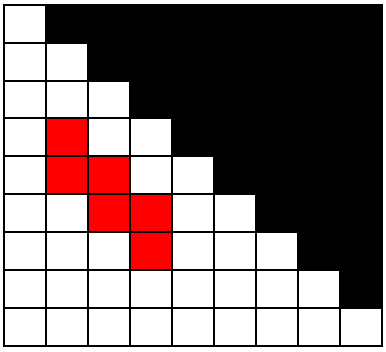
Akan didapatkan sesuai gambar yang ingin dibuktikan apabila diteruskan sampai habis

Maka, jumlahan sel-sel berwarna merah dan biru di bawah ini berjumlah sama. Selain itu, menggunakan manipulasi rumus yang mirip dengan sebelumnya, akan didapat bahwa jumlahan sel-sel berwarna merah, biru, dan (kuning-hijau) adalah sama, yaitu $\binom{n+2}{m+1} - 1$.

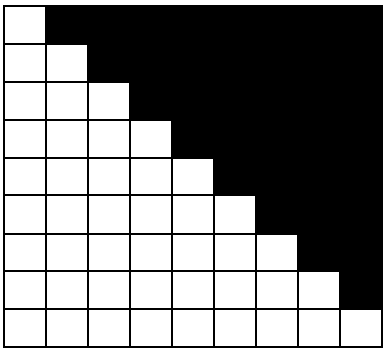


Sekarang, kita dapat mencari jawaban untuk setiap pertanyaan menggunakan inklusi-eksklusi. Penyelesaiannya adalah sebagai berikut (catatan: putih berarti terhitung 0, merah terhitung 1, biru terhitung -1).

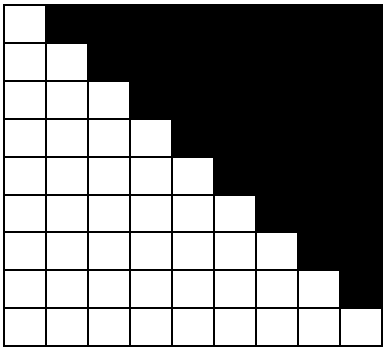




Target

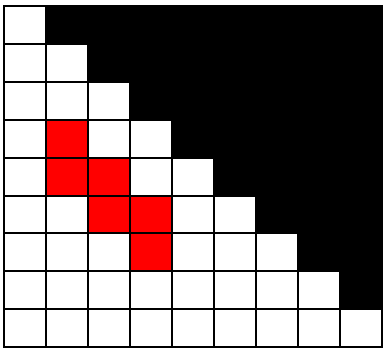


Jumlahan

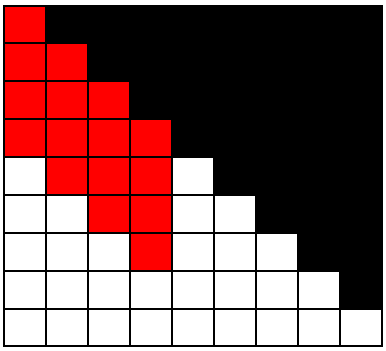


Sel yang dipakai

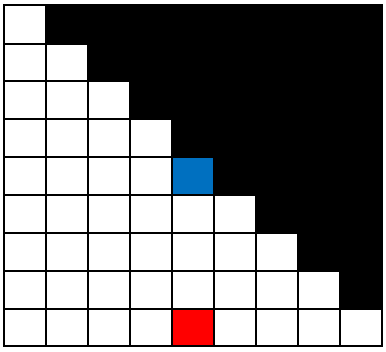
$$Jawaban = 0$$



Target

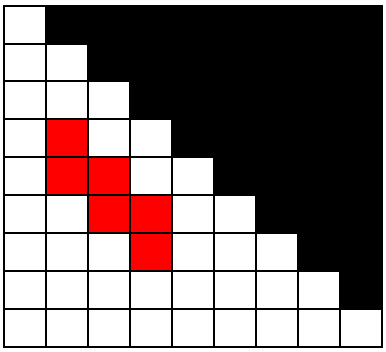


Jumlahan

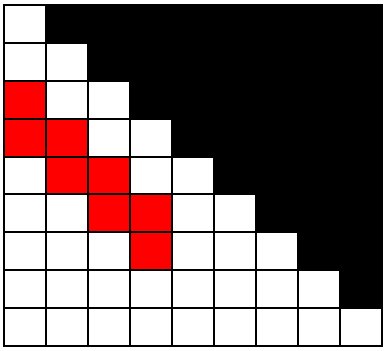


Sel yang dipakai

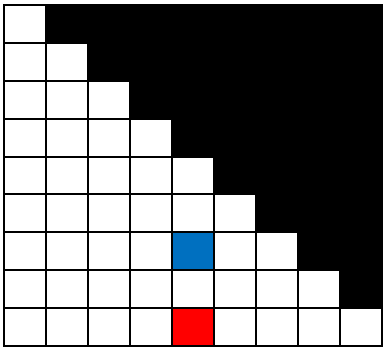
$$Jawaban = \binom{n+l+r}{m+r} - 1$$



Target

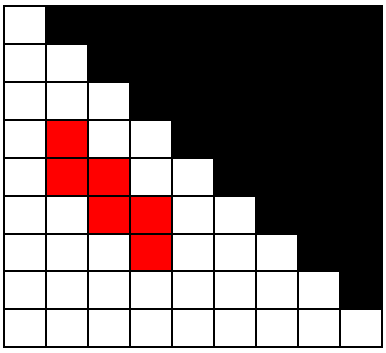


Jumlahan

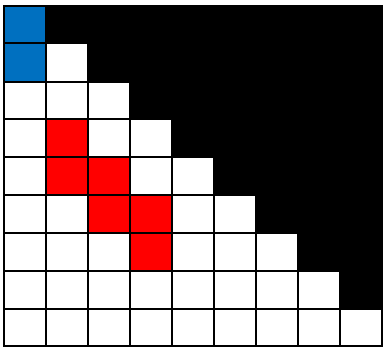


Sel yang dipakai

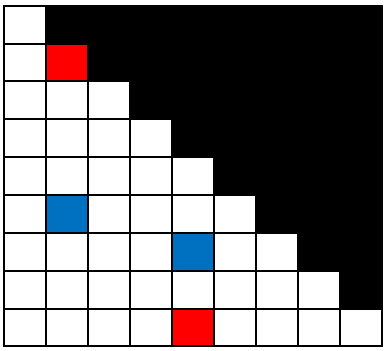
$$Jawaban = \binom{n+l+r}{m+r} - \binom{n+r}{m+r}$$



Target

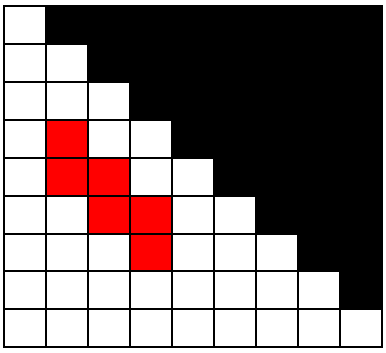


Jumlahan

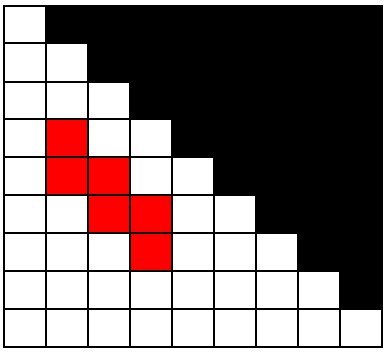


Sel yang dipakai

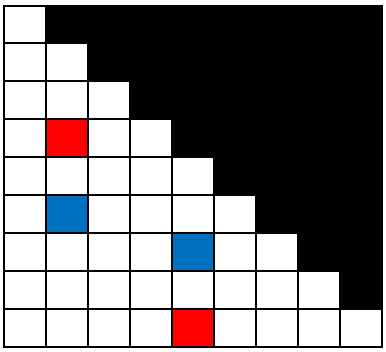
$$Jawaban = \binom{n+l+r}{m+r} - \binom{n+r}{m+r} - \binom{n+l}{m} + 1$$



Target



Jumlahan



Sel yang dipakai

$$Jawaban = \binom{n+l+r}{m+r} - \binom{n+r}{m+r} - \binom{n+l}{m} + \binom{n}{m}$$

Maka, didapatkan bahwa rumus akhir untuk menjawab tiap pertanyaan adalah $\binom{n+l+r}{m+r} - \binom{n+r}{m+r} - \binom{n+l}{m} + \binom{n}{m}$. Namun, didapatkan masalah baru, yaitu cara menghitung kombinasinya. Kita dapat memanfaatkan rumus 2, yaitu $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$.

Namun, dalam rumus tersebut terdapat pembagian. Untungnya, pembagian tersebut dapat dilakukan menggunakan *modular multiplicative inverse*, yang juga dapat dicari menggunakan *Fermat's little theorem* karena nilai modulo yang berupa bilangan prima.Menggunakan teorema tersebut, kita punya:

$$k^{-1}(mod\ 10^9 + 9) \equiv\ k^{10^9+7}(mod\ 10^9 + 9)$$

Jadi, kita dapat melakukan prekomputasi seluruh nilai $x!$ dan mencatatnya dalam sebuah array. Setelahnya, tiap nilai kombinasi dapat dicari dalam $O(log\ MOD)$, karena pencarian *modular multiplicative inverse* daat dilakukan menggunakan *fast modular exponentiation*.

Kompleksitas: $O(N + L + R)$

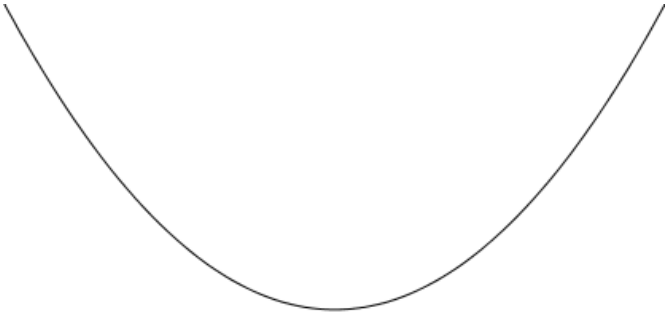
G. Relokasi Warga KaliJodie

Tag

Ternary search

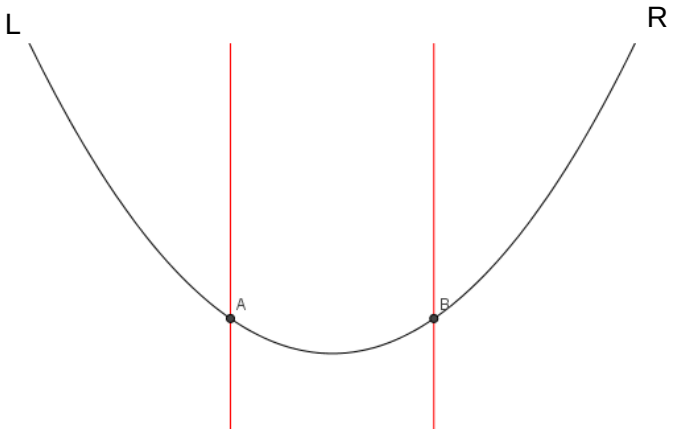
Pembahasan

Pertama, semakin panjang daerah relokasi tentu semakin baik jadi kita pasti membuat daerah relokasi sepanjang K meter. Selanjutnya, perhatikan bahwa apabila biaya minimum relokasi adalah dengan membuat daerah relokasi mulai dari koordinat X (hingga koordinat $X + K$) maka daerah relokasi yang dibuat mulai dari $X - e$ dan $X + e$ pasti memiliki biaya yang lebih mahal dari biaya di koordinat X (dengan e adalah nilai epsilon yang sangat kecil). Dengan kata lain apabila kita mendefinisikan fungsi $F(X)$ sebagai biaya yang diperlukan untuk relokasi apabila daerah relokasi terletak pada koordinat X hingga $X + K$ maka grafik $F(X)$ akan berbentuk kurva seperti berikut.



Untuk mencari nilai $F(X)$ kita dapat melakukan iterasi terhadap semua warga, namun yang harus kita lakukan adalah menemukan X sehingga nilai $F(X)$ minimum, untuk itu kita bisa menggunakan *ternary search* atau *binary search* yang dimodifikasi. Pada pembahasan ini hanya akan dibahas cara *ternary search*.

Mula-mula kita perlu menentukan batas atas dan batas bawah untuk *ternary search* kita, kita dapat menggunakan batasan koordinat X warga, karena daerah relokasi yang optimal pasti di antara warga. Pada setiap iterasi, misal kita memproses daerah *search* dari interval L hingga R, kita akan membagi daerah search kita menjadi 3 bagian.



Hanya terdapat 2 kasus untuk setiap iterasi, yaitu:

- $F(A) < F(B)$ maka area interval B hingga R tidak mungkin merupakan solusi sehingga kita dapat memperkecil daerah *search* menjadi interval L hingga B.
- selain itu maka kita dapat eliminasi interval L hingga A dan memperkecil daerah *search* menjadi interval A hingga R.

Kompleksitas: $O(N \log MAX\_VALUE)$

H. Menyelamatkan Mbak Miku

Tag

Graph traversal, flood fill

Pembahasan

Pertama, misalkan kita memiliki himpunan S yang elemennya merupakan ID kota yang ada dalam jalur yang tekecil secara leksikografis. Mula-mula terdapat ID kota di koordinat (1,1) dan koordinat (R,C) dalam S. Selanjutnya kita akan memasukkan ID kota mulai dari yang terkecil.

Langkah pertama adalah menentukan koordinat setiap ID dari ID 1 hingga ID $R * C$. Setelah itu kita dapat iterasi dari ID terkecil dan memasukkannya ke dalam S jika memungkinkan. Perhatikan bahwa jika kita memasukkan kota pada koordinat (a,b) maka kita tidak mungkin memasukkan kota pada daerah berwarna merah pada tabel di bawah karena akan menjadi bukan jalur terpendek apabila kita melakukannya.

1,1									
				a,b					
									R,C

Maka kita cukup menandai daerah berwarna merah ini, selama suatu kota tidak berada dalam daerah yang sudah ditandai berwarna merah maka dapat kita masukkan dalam S. Dalam menandai daerah berwarna merah, kita dapat menggunakan sejenis *flood fill*, atau dengan loop sederhana. Selain itu, misal setelah mengambil petak (a, b) kita hendak mengambil suat petak (x, y) seperti berikut:

1,1									
				a,b					
					x,y				
									R,C

Petak berwarna biru adalah petak yang baru saja kita tandai sebagai tidak boleh diambil. Perhatikan bahwa petak berwarna ungu tidak perlu kita tandai lagi, karena sudah ditandai sebelumya.

Kompleksitas: $O(R * C)$