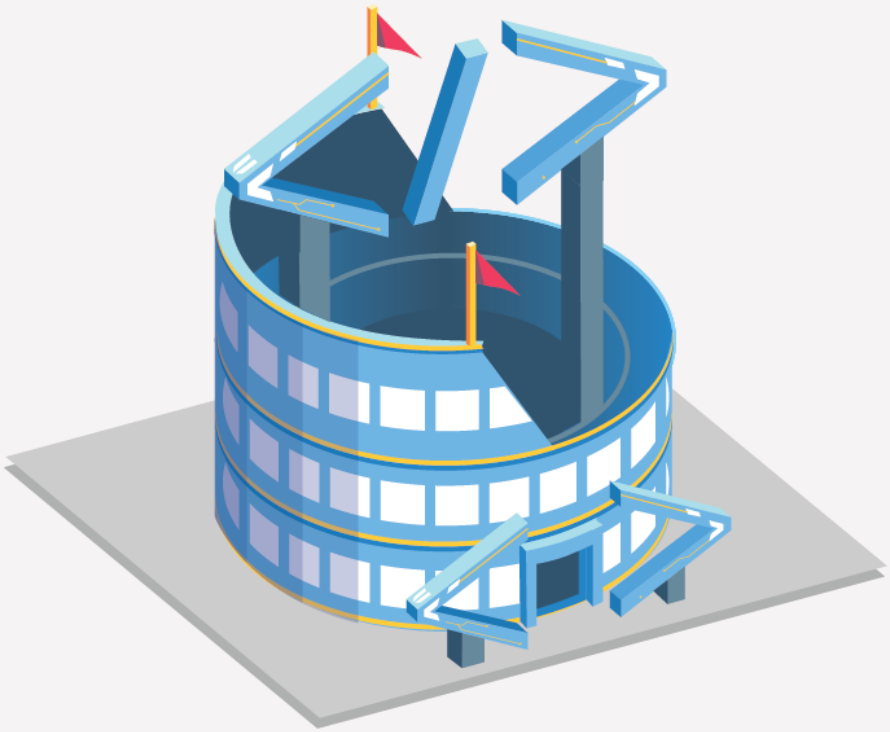
COMPFEST 8

COMPETITION

COMPETITIVE PROGRAMMING CONTEST



COMPFEST 8

Pembahasan Coderclass

SCPC – Minggu 2

Daftar Soal

A. Dinas Perhubungan
B. Tebak Basis
C. Jualan Balon
D. Semut dan Balok
E. Main Batu Lagi
F. Amar dan Kina
G. Jajar Genjang Pascal
H. Jalan-Jalan

A. Dinas Perhubungan

Tag

graf

Pembahasan

Pertama, kita cari tiap *connected component*. Hal ini bisa dilakukan menggunakan traversal graf biasa seperti DFS dan BFS, atau menggunakan struktur data *disjoint set*. Misal, graf tersebut terdiri dari K buah *connected component* C_1, C_2, …, C_K. Kita tinjau untuk salah satu *connected component*, sebut saja C_i.

Observasi yang bisa kita ambil adalah, semua anggota C_i dapat menjangkau satu sama lain. Observasi penting selanjutnya adalah, himpunan kota-kota yang tidak dapat dijangkau masing-masing anggota C_i adalah sama, yaitu semua kota di luar C_i! Dengan kata lain, semua kota di C_i tidak dapat menjangkau $N - |C_i|$ kota. Sehingga, kontribusinya terhadap jawaban adalah $|C_i| \cdot (N - |C_i|)$.

Lebih lanjut lagi, banyaknya pasangan terurut kota-kota yang tidak dapat saling jangkau adalah $\sum_{i=1}^{K} |C_i| \cdot (N - |C_i|)$. Namun, yang diminta di soal merupakan pasangan tak terurut, sehingga kita harus menghilangkan *double counting*. Cukup dengan membagi hasilnya dengan 2. Sehingga, jawabannya adalah $\frac{\sum_{i=1}^{K} |C_i| \cdot (N - |C_i|)}{2}$.

B. Tebak Basis

Tag

ad hoc

Pembahasan

Terdapat dua permasalahan, yaitu mencari basis terkecil yang valid, serta basis terbesar yang valid. Pertama, kita mulai dari basis terbesar yang valid. Kita bisa berpikir secara rakus untuk menyelesaikannya. Berapa basis terbesar yang ada pada kalkulator? 62. Tentunya, semua bilangan yang muncul pada kalkulator akan valid pada basis 62. Oleh karena itu, pada semua kasus uji, basis terbesar yang valid pasti adalah 62.

Untuk mencari basis terkecil, kita harus mencari digit terbesar yang terdapat dalam bilangan tersebut. Lalu, kita cari basis terkecil yang memiliki digit tersebut. Misal digit tebesar adalah x. Berapa basis terkecil yang memiliki digit tersebut? Ya, $x + 1$! Maka, untuk mencari basis terkecil yang valid, kita cukup mencari digit terbesar pada bilangan, dan basis terkecil yang valid adalah digit tersebut ditambah 1.

Salah satu permasalahan kecil di soal ini adalah mendapatkan nilai untuk setiap digit. Kita dapat memetakannya dengan percabangan biasa, atau memetakannya dulu di awal menggunakan array atau sejenisnya.

C. Jualan Balon

Tag

ad hoc

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).

D. Semut dan Balok

Tag

matematika

Pembahasan

Terdapat tiga permasalahan: (1) Mencari jarak terdekat dari dua titik sudut yang berseberangan dengan melewati sisi-sisi; (2) Mencari jarak terjauh dari dua titik sudut yang berseberangan dengan melewati sisi-sisi; (3) Mencari jarak euclidean antara dua titik sudut yang berseberangan.

Untuk permasalahan (1), tentunya semut akan berjalan "miring" di sisi balok. Oleh karena itu, jarak terdekatnya pasti nilai minimum dari tiga kemungkinan berikut:

- $\sqrt[2]{(P+L)^2+T^2}$
- $\sqrt[2]{(P+T)^2+L^2}$
- $\sqrt[2]{(L+T)^2+P^2}$

Untuk permasalahan (2), jalur terjauh yang tidak pernah menjauhi tujuan tentunya jalur yang menelusuri rusuk-rusuk balok. Oleh karena itu, jaraknya pasti $P+L+T$.

Untuk permasalahan (3), jarak yang diminta adalah jarak euclidean antara dua titik sudut yang bersebrangan. Jarak ini bisa didapat dari $\sqrt[2]{P^2+L^2+T^2}$.

E. Main Batu Lagi

Tag

sprague-grundy theorem, sliding window

Pembahasan

Sebelum membaca pembahasan ini, ada baiknya Anda memahami dulu tentang *sprague-grundy theorem.* Anda dapat membaca teorema tersebut pada tautan berikut:

- http://www.gabrielnivasch.org/fun/combinatorial-games/sprague-grundy

Observasi pertama adalah, permainan Batuman merupakan *impartial game*, karena seluruh pemain memiliki informasi permainan, serta himpunan gerakan yang valid untuk tiap *state* permainan sama bagi kedua pemain. Observasi kedua, *state-state* pada permainan ini dapat digambarkan sebagai suatu *directed acyclic graph* (DAG).

Mengapa permainan ini dapat digambarkan sebagai DAG? Hal ini karena terdapat batasan:

- Untuk setiap $i, a_i \leq b_i < i$
- Untuk setiap i dan j dimana $i < j$, $a_i \leq a_j$ dan $b_i \leq b_j$

Dengan batasan tersebut, tentunya Batuman yang berada pada suatu petak *x*, tidak akan dapat kembali lagi ke petak *x*. Sehingga, dari suatu *state*, kita tidak dapat kembali lagi ke *state* tersebut.

Untuk soal-soal seperti ini, seperti biasa kita harus mencari nilai grundy (*grundy value*) untuk setiap posisi, sebut saja $g(pos)$ sebagai nilai grundy pada petak pos. Awalnya, kita hanya tahu nilai grundy untuk *terminal position* (posisi yang tidak bisa bergerak lagi), yaitu 0. Pada permainan Batuman, *terminal position* berada pada posisi 1, sehingga awalnya, kita punya $g(1) = 0$.

Perhatikan bahwa untuk mencari tahu nilai $g(x)$, maka kita perlu tahu nilai $g(y)$ untuk $a_x \leq y \leq b_x$. Nilai $g(x)$ dapat kita cari menggunakan *dynamic programming*, dengan $g(x) = mex(\{g(y) | a_x \leq y \leq b_x\})$. Namun, banyaknya kemungkinan nilai y bisa mencapai N, dalam hal ini 10^5. Terdapat N nilai grundy yang kita butuhkan, sehingga untuk

mencari semua nilai grundy, akan dibutuhkan waktu $O(N^2)$, yang pasti akan mendapat verdict TLE.

Salah satu teknik yang bisa dipakai untuk mengoptimasinya adalah menggunakan teknik *sliding window*, ditambah struktur data seperti *set* dan *priority_queue*. Gambaran kasar untuk teknik *sliding window* adalah sebagai berikut. Anggap, kita memiliki informasi $g(pos)$ untuk $a_x \leq pos \leq b_x$. Maka, informasi tersebut akan dimanfaatkan untuk mencari nilai $g(pos')$ untuk $a_{x+1} \leq pos' \leq b_{x+1}$.

Saat kita memiliki nilai $g(pos)$ untuk $a_x \leq pos \leq b_x$, kita bisa meng-update-nya untuk medapatkan semua nilai $g(pos')$ untuk $a_{x+1} \leq pos' \leq b_{x+1}$. Caranya, cukup lakukan dua langkah berikut: (1) buang semua $g(y)$, untuk $a_x \leq y < a_{x+1}$, dan (2) masukkan semua $g(y)$, untuk $b_x < y \leq b_{x+1}$. Dengan begitu, kita bisa memiliki informasi $g(pos')$ untuk $a_{x+1} \leq pos' \leq b_{x+1}$. Perhatikan bahwa total pembuangan, serta memasukkan nilai grundy tentunya tidak akan melebihi $O(N)$. Selanjutnya, dari informasi $g(pos')$, kita bisa mencari nilai $g(x+1)$.

Setelah kita memiliki nilai grundy semua petak, yang tersisa adalah mencari nim sum dari petak-petak yang berisi bidak Batuman.

Pencarian semua nilai grundy yang diharapkan adalah $O(N \log^2 N)$. Namun, terdapat juga solusi $O(N \log N)$.

F. Amar dan Kina

Tag

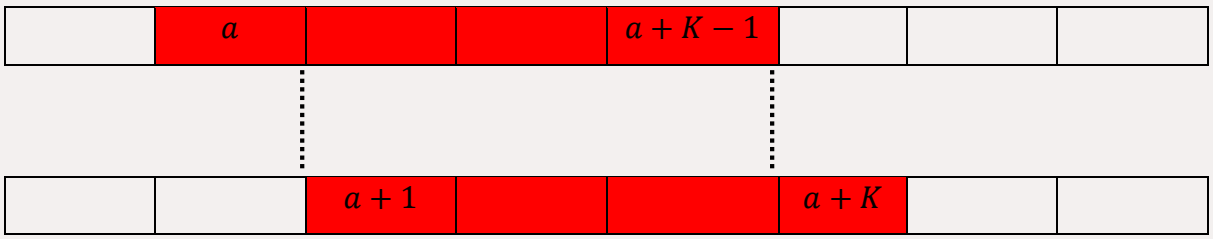
sliding window

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.

G. Jajar Genjang Pascal

Tag

Kombinatorika, inklusi-eksklusi, modular multiplicative inverse

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:

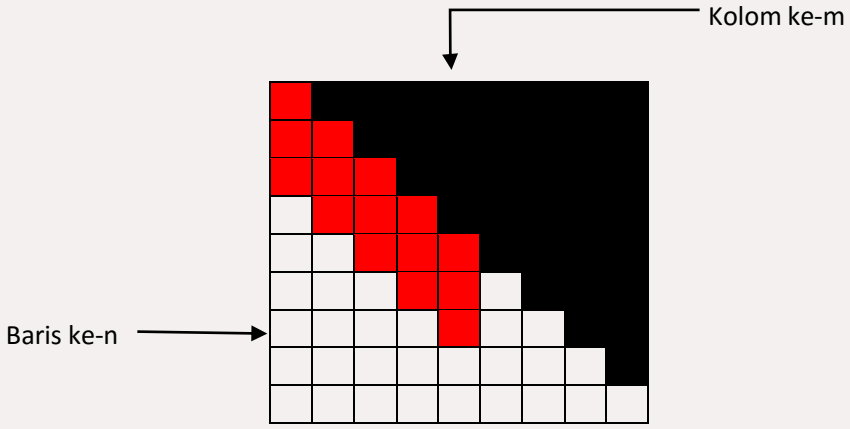
	0	1	2	3	4	5	6	7	8
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	

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

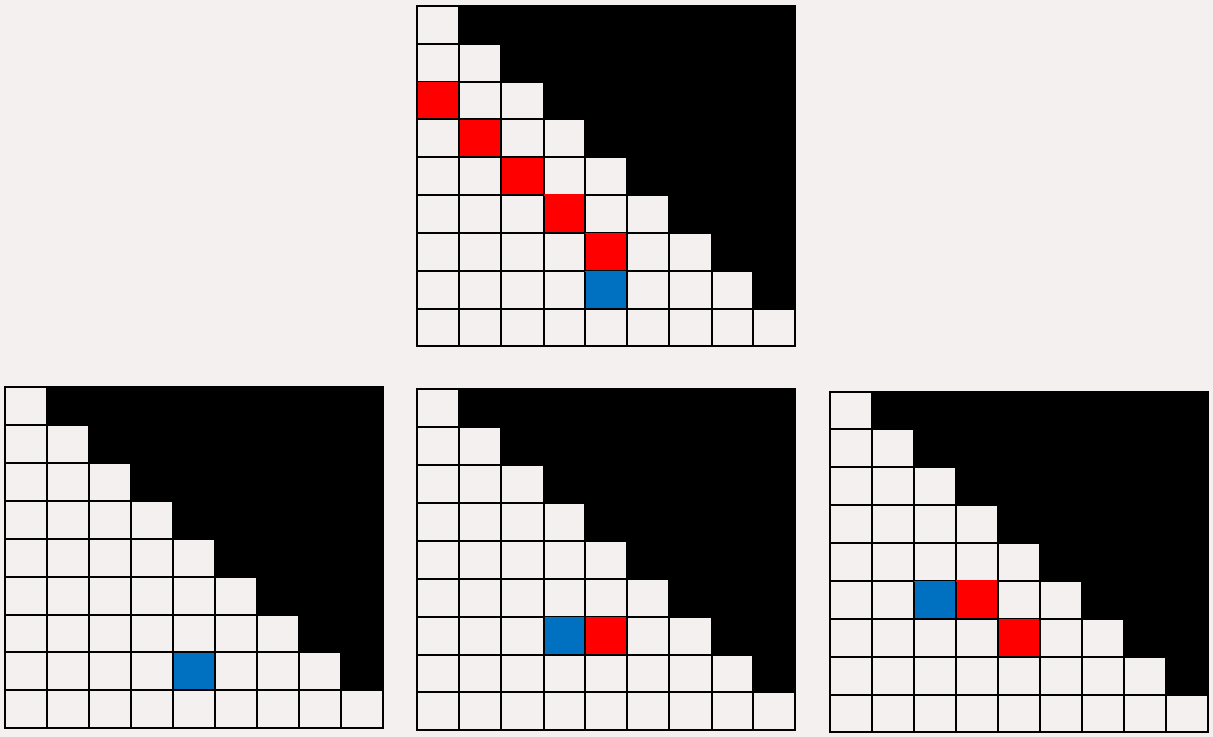
	0	1	2	3	4	5	6	7	8
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	

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

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:

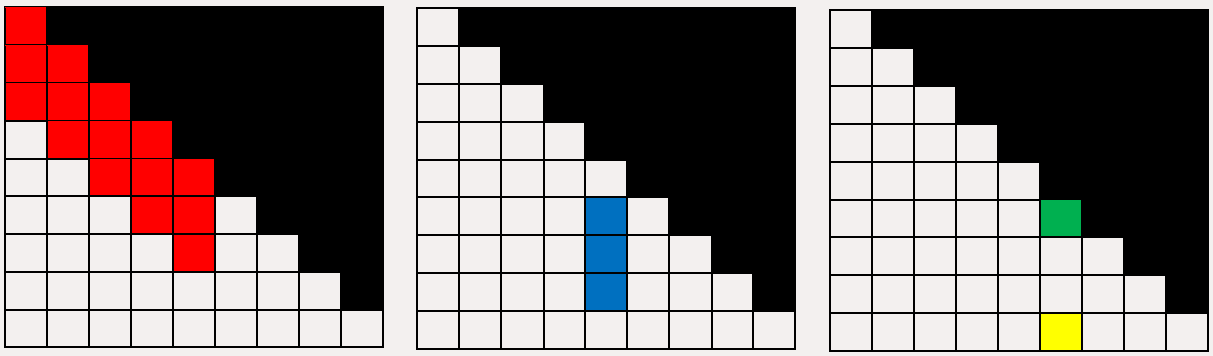


Awalnya, sebuah sel biru

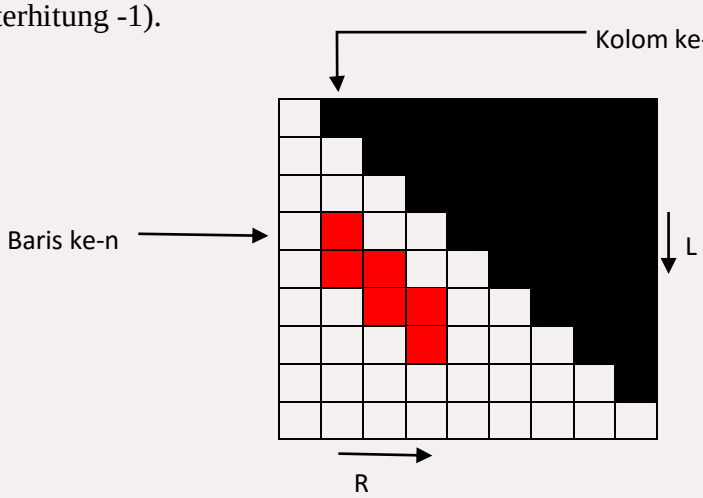
Menggunakan rumus 1...

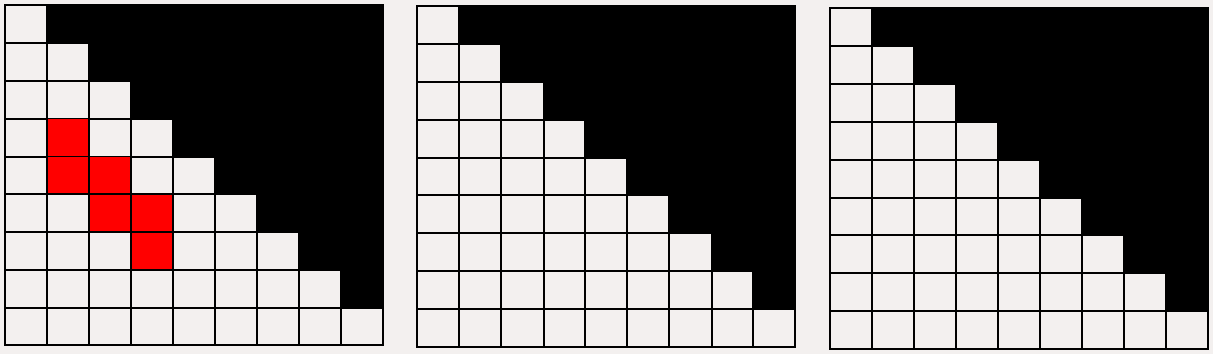
Akan didapatkan sesuai gambar yang ingin dibuktikan apabila diteruskan sampai habis

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 $\binom{n+2}{m+1} - 1$.



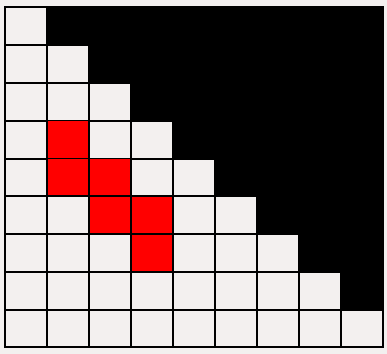
Sekarang, kita dapat mencari jawaban untuk setiap pertanyaan menggunakan inklusi-eksklusi. Penyelesaiannya adalah sebagai berikut (catatan: putih berarti terhitung 0, merah terhitung 1, biru terhitung -1).



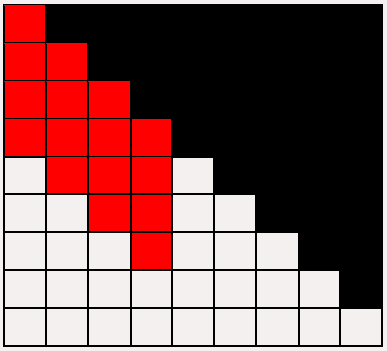


Target *Jumlahan* *Sel yang dipakai*

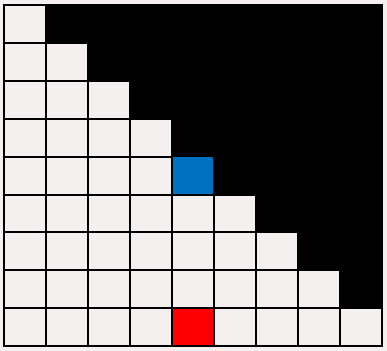
$$Jawaban = 0$$



Target

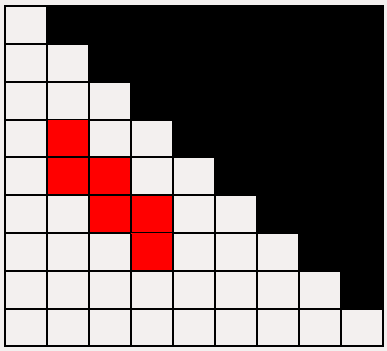


Jumlahan

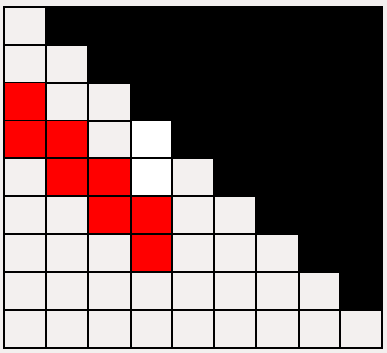


Sel yang dipakai

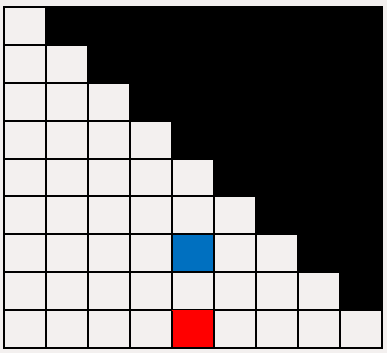
$$Jawaban = \binom{n+l+r}{m+r} - 1$$



Target

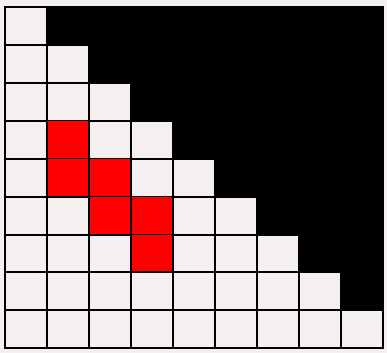


Jumlahan

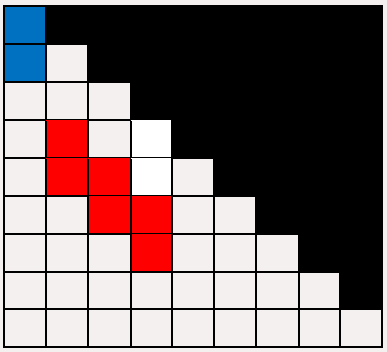


Sel yang dipakai

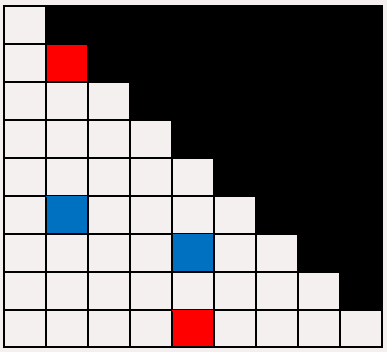
$$Jawaban = \binom{n+l+r}{m+r} - \binom{n+r}{m+r}$$



Target

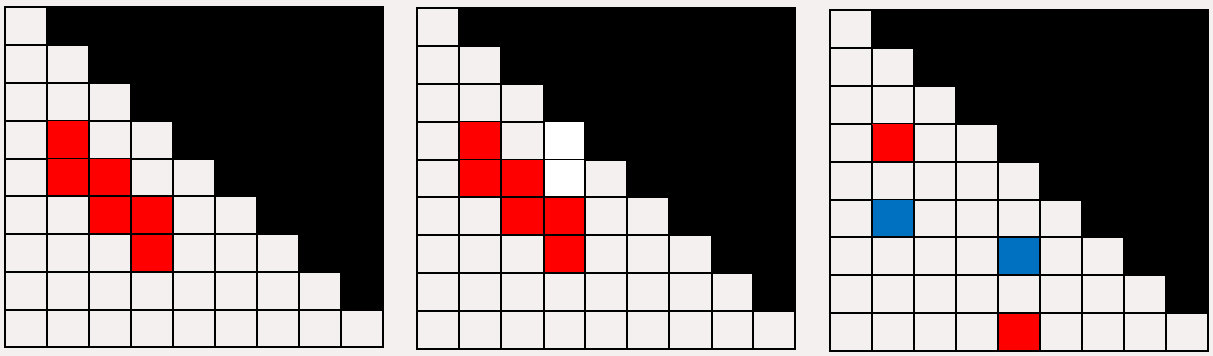


Jumlahan



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*.

H. Jalan-Jalan

Tag

dynamic programming

Pembahasan

Misal S merupakan $\sum_{i=1}^{N} D_i$. Maka, $y = S - x$. Sehingga, kita ingin memaksimalkan nilai $x \cdot y = x \cdot (S - x)$. Dengan sedikit observasi tambahan, kita tahu bahwa $x \cdot (S - x)$. semakin besar apabila selisih x dengan $S - x$ semakin kecil. Oleh karena itu, sekarang kita ingin mencari tahu apakah untuk suatu x pada $[0, S]$, kita dapat membentuknya sebagai jumlahan beberapa D_i. Setelah itu, untuk setiap x yang dapat dibentuk, kita cari yang memaksimalkan $x \cdot (S - x)$.

Mencari tahu apakah untuk suatu x pada $[0, S]$, kita dapat membentuknya sebagai jumlahan beberapa D_i merupakan salah satu permasalahan klasik, yaitu *subset sum*. Kita dapat menyelesaikannya dengan menggunakan *dynamic programming*, dengan *state* DP-nya berupa (*posisi*, *jumlah_yang_ingin_dibentuk*). Kurang lebih, formulasi DP-nya sebagai berikut:

$$f(pos, target) = \begin{cases} true & , pos = N \ \wedge \ target = 0 \\ false & , pos = N \ \wedge \ target \neq 0 \\ f(pos + 1, target) & , pos < N \ \wedge target < D_{pos} \\ f(pos + 1, target) \vee f(pos + 1, target - D_{pos}) & , pos < N \ \wedge target \geq D_{pos} \end{cases}$$

Mencari tahu apakah suatu x dapat dibentuk dapat dilakukan dengan memanggil $f(0, x)$. Untuk mencari tahu pembagian kanan dan bawah, kita bisa melakukan *backtracking* dengan memanfaatkan tabel DP-nya.