

~~Write Up~~
Editorial
Seleksi IEEEXtreme 17.0 ITB



Gak nemu foto yang lucu jadi pake ini aja

Disusun oleh:
神様

Kode Soal

- A : Abi Eshuh
- B : Caffeine Fighter
- C : Constant Moderato
- D : Formasi Tentara
- E : Mantra Hujan
- F : Peko Peko
- G : Pixel Time
- H : Plum Blossom Garden
- I : Rabbit House

Author

- Dewana : A, B, C, E, F, I
- Arya : D, G, H

Expected Difficulty Jury

- Easy : D, E
- Easy-Medium : B
- Medium : A, C, F
- Medium-Hard : G, H, I

Expected Difficulty Berdasarkan Scoreboard

- Easy : D, E
- Easy-Medium : A, B
- Medium : F, I
- Medium-Hard : C, G, H

Buat setiap soal bakal dikasih materi pre requisite yang rekomen buat dipelajarin sebelum baca editorial

Buat setiap soal juga bakal dikasih reference ceritanya 😊😊😊

A. Abi Eshuh



Sistem alat tempur mutakhir millenium, Abi-Eshuh
kamus definisi:

- subproblem(Z, n) : apakah kita bisa membasmikan Z buah robot dalam tepat n detik dimana kekuatan kita bermulai dari 0
- $S(n) = 1 + 2 + 3 + \dots + n = \text{sigma}[i=1][n] = n * (n + 1) / 2$
- $F(k, n)$: cara pemilihan penekanan tombol yang akan menghasilkan jumlah tembakan laser terbanyak apabila kita hanya ingin memakai laser dengan kekuatan kurang dari k selama n detik
- $SF(k, n)$: jumlah laser yang ditembakkan dengan cara pemilihan $F(k,n)$
- $T1(n, k) = n * k$
- $T2(n, k) = S(n-1) + n * k$

karena kekuatan kita bermulai dari K , dan kekuatan tidak mungkin berkurang. disini kita bisa memodelkan soal kita menjadi lebih mudah yaitu dengan kekuatan yang bermulai dari 0 dengan cara mengurangi kekuatan laser sekarang dengan K . perhatikan juga bahwa karena

kekuatan kita selalu $\geq K$, maka ketika kita sudah melalui n detik, Toki telah menembakkan $n * K$ buah laser. disini kita bisa memodelkan persoalan baru, misal subproblem(Z, n) berarti apakah kita bisa membasmi Z buah robot dalam tepat n detik dimana kekuatan kita bermula dari 0. untuk mencari solusi kita perhatikan bahwa karena setelah n detik kita pasti menembak $n * K$ laser, untuk sebuah n yang pasti persoalan kita dapat terselesaikan apabila subproblem($X - n*K, n$) dapat terselesaikan.

TLDR : buat kebawah ini isinya solusi tentang kondisi apa yang buat subproblem kita bisa diselesaikan dan proofnya, mungkin ada juga cara proof yang lebih singkat
buat yang gak mau baca proof intinya subproblem(Z, n) dapat selesai apabila $0 \leq Z \leq S(n-1)$

Untuk dapat menyelesaikan subproblem(Z, n), perhatikan bahwa terdapat 4 kasus yang trivial:

1. apabila $Z = 0$, karena kita mempunyai laser berkekuatan 0 dan sudah tidak ada robot, Toki dapat terus menekan tombol A sehingga Abi-Eshuh tidak menembak laser, dan keuatannya pun tidak bertambah
2. apabila $Z < 0$, ini merupakan kasus dimana kita telah menembak laser dengan jumlah lebih dari robot yang tersisa, maka kasus ini tidak dapat terselesaikan
3. apabila $Z = S(n-1)$, perhatikan apabila ingin menembak laser dengan jumlah maksimal, kita dapat memutuskan secara greedy untuk selalu menekan tombol B, hal ini dikarenakan maupun tombol A atau tombol B akan menembakkan jumlah laser yang sama, namun karena tombol B menambah kekuatan laser sedangkan tombol A tidak, jumlah maksimum laser yang dapat dikeluarkan selama n detik apabila kita selalu menekan tombol B adalah $S(n-1)$.
4. apabila $Z > S(n-1)$, pada kasus 3 dijelaskan bahwa jumlah maksimum laser yang dapat dikeluarkan selama n detik adalah $S(n-1)$, apabila jumlah robot melebihi $S(n-1)$, maka jumlah tersebut tidak dapat dibasmi hanya dalam n detik.

sejauh ini kita mendapat kesimpulan bahwa:

subproblem(Z, n) dapat terselesaikan apabila $Z = 0$ atau $Z = S(n-1)$, dan
subproblem(Z, n) tidak dapat terselesaikan apabila $Z < 0$ atau $Z > S(n-1)$
maka kita perlu mengetahui bagaimana solusi subproblem(Z, n) apabila nilai Z berada dalam rentang $0 < Z < S(n-1)$.

misal kita definisikan berapa kali kita menekan tombol ketika berada di kekuatan k adalah x_k . perhatikan bahwa jumlah laser yang kita tembakkan (misal Y) dapat dimodelkan dengan persamaan

$$Y = x_0 * 0 + x_1 * 1 + x_2 * 2 + \dots + x_{(n-1)} * (n-1) = \sum_{i=0}^{n-1} (x_i * i),$$
dengan syarat

syarat 1. apabila $x_i > 0$, maka $x_{i-1} \geq 1$ dengan $i > 0$ (apabila ingin menekan tombol ketika berkekuatan i , kita harus pernah menekan tombol ketika berkekuatan $i-1$ atau kekautan sebelumnya)

syarat 2. dan juga syarat $x_0 + x_1 + x_2 + \dots + x_{(n-1)} = \sum_{i=0}^{n-1} (x_i) = n$, (jumlah kita menekan tombol adalah adalah n)

misal definisikan $F(k, n)$ adalah cara pemilihan penekanan tombol yang akan menghasilkan jumlah tembakan laser terbanyak apabila kita hanya ingin memakai laser dengan kekuatan kurang dari k selama n detik, cara pemilihan yang memenuhi syarat tersebut adalah dengan terus menerus menekan tombol B sampai kita mencapai $k-1$ lalu terus menekan tombol A pada saat kekuatan terbesar ($k-1$) sampai waktu yang tersisa habis. secara lebih formal untuk sebuah $F(k, n)$, $x_i = 1$ apabila $0 \leq i < (k-1)$, dan $x_{(k-1)} = n-k+1$.

misal kita definisikan $SF(k, n)$ adalah jumlah tembakan laser dari $F(k, n)$. nilai $SF(k, n)$ dapat dihitung dengan cara sebagai berikut

$$SF(k, n) = 0 + 1 + 2 + \dots + (k-1) * (n-k+1)$$

$$SF(k, n) = S(k-2) + (k-1) * (n-k+1)$$

$$SF(k, n) = (k-2) * (k-1) / 2 + (k-1) * (n-k+1)$$

$$SF(k, n) = (k-1) * ((k-2)/2 + n-k+1)$$

$$SF(k, n) = (k-1) * (n - k/2)$$

karena $F(k, n)$ merupakan cara pemilihan penekanan tombol yang valid, dan $SF(k, n)$ adalah banyaknya laser yang ditekan dengan cara pemilihan $F(k, n)$, maka kasus subproblem(Z, n) dengan $Z = SF(k, n)$ dan $k \leq n$, sudah pasti dapat diselesaikan.

perhatikan ketika kita memakai cara pemilihan penekanan tombol $F(k, n)$ dengan $k < n$, kita dapat memindahkan sebuah nilai dari $x_{(k-1)}$ ke x_k dan jumlah tembakan akan bertambah 1 dengan tetap memenuhi syarat, pembuktian:

syarat 1 : tetap terpenuhi, apabila nilai $x_{(k-1)} > 1$, maka ketika kita memindahkan sebuah nilai $x_{(k-1)}$ ke x_k , nilai $x_{(k-1)}$ akan menjadi $x_{(k-1)} \geq 1$, yang mana merupakan syarat x_k untuk dapat menjadi valid

syarat 2 : pada pemilihan $F(k, n)$, syarat $\sum_{i=0}^{n-1} (x_i) = n$ telah terpenuhi, apabila kita memindahkan sebuah nilai $x_{(k-1)}$ ke x_k , maka jumlah menekan tombol yang dilakukan akan berkurang 1 dan bertambah 1, $\sum_{i=0}^{n-1} (x_i) = n - 1 + 1 = n$, syarat tetap terpenuhi karena batasan syarat 1, untuk sebuah $F(k, n)$ dengan $k < n$, karena terdapat sebanyak $(n-k+1)$ $x_{(k-1)}$, kita dapat memindahkan nilai $x_{(k-1)}$ menjadi x_k sebanyak $n-k$ kali.

bisa dilihat bahwa selain $SF(k, n) = SF(k, n)$ dapat terselesaikan, karena kita dapat melakukan perpindahan nilai seperti yang dijelaskan sebelumnya, subproblem(Z, n) juga akan terpenuhi apabila nilai Z berada dalam rentang $[SF(k) \dots SF(k) + n-k]$ atau $SF(k, n) \leq Z \leq SF(k, n) + n-k$, dengan $k < n$.

sekarang perhatikan apa yang terjadi pada $SF(k+1, n)$,

$$SF(k, n) = (k-1) * (n - k/2)$$

$$SF(k+1, n) = k * (n - (k+1)/2)$$

$$SF(k+1, n) = k*n - k*(k+1)/2$$

$$SF(k+1, n) = k*n - k*k/2 - k/2$$

$$SF(k+1, n) = k*n - k*k/2 - k/2 - n + n + k - k$$

$$\begin{aligned}
 SF(k+1, n) &= k*n - n - k*k/2 - k/2 + k + n - k \\
 SF(k+1, n) &= k*n - n - k*k/2 + k/2 + n - k \\
 SF(k+1, n) &= n*(k-1) - k*(k-1)/2 + n - k \\
 SF(k+1, n) &= (k-1)*(n - k/2) + n - k \\
 SF(k+1, n) &= SF(k, n) + n - k
 \end{aligned}$$

atau bisa juga dibuktikan dengan visual proof, misal cara pemilihan sekarang adalah $F(k, n)$, ketika kita memindahkan $(n-k)$ buah $x(k-1)$ menjadi x_k , cara pemilihan kita sekarang adalah $F(k+1, n)$ dan jumlah laser yang ditembak bertambah sebanyak $n-k$

sebelumnya kita telah mengetahui bahwa untuk $k < n$, subproblem(Z, n) dapat terselesaikan apabila nilai Z berada dalam rentang $[SF(k, n) \dots SF(k, n) + n-k]$, dan karena $SF(k, n) + n-k = SF(k+1, n)$, maka kita mendapat fakta baru bahwa subproblem(Z, n) dapat terpenuhi apabila nilai Z berada dalam rentang $[SF(k, n) \dots SF(k+1, n)]$ dengan $k < n$.

apabila kita perhatikan rentang nilai Z yang membuat subproblem(Z, n) terselesaikan, rentang tersebut adalah $[SF(1, n) \dots SF(2, n)] \cup [SF(2, n) \dots SF(3, n)] \cup \dots \cup [SF(n-1, n) \dots SF(n, n)]$ nilai yang memenuhi akan berbentuk teleskopik dan dapat disederhanakan menjadi subproblem(Z, n) dapat terpenuhi apabila nilai Z berada dalam rentang $[SF(1, n) \dots SF(n, n)]$

nilai dari $SF(1, n)$ dan $SF(n, n)$ dapat dengan mudah kita hitung,

$$\begin{aligned}
 SF(k, n) &= (k-1) * (n - k/2) \\
 SF(1, n) &= (1-1) * (n - 1/2) = 0 * (n - 1/2) = 0 \\
 SF(n, n) &= (n-1) * (n - n/2) = (n-1) * (n/2) = S(n-1)
 \end{aligned}$$

apabila kita mengumpulkan seluruh fakta yang kita ketahui, kita mendapat kesimpulan bahwa:

subproblem(Z, n) dapat terselesaikan apabila $0 \leq Z \leq S(n-1)$, dan
 subproblem(Z, n) tidak dapat terselesaikan apabila $Z < 0$ atau $Z > S(n-1)$
 apabila rentang nilai Z yang berada dalam kesimpulan kita gabungkan,
 rentang keseluruhan = $[0 \dots S(n-1)] \cup (-\infty, 0) \cup (S(n-1), \infty) = (-\infty, \infty)$. rentang nilai Z pada kesimpulan telah mencakup seluruh nilai bilangan bulat yang mungkin, maka dari itu kesimpulan yang kita dapat sudah mencakup seluruh kasus yang mungkin.

untuk mencari solusi persoalan asli, kita akan mencoba memakai fakta solusi subproblem kedalam persoalan asli. ketika kita menyuruh Toki untuk membasmi seluruh robot dalam waktu n detik, persoalan kita dapat dimodelkan sebagai subproblem($X - n*k, n$). kita telah mengetahui bahwa subproblem tersebut dapat terselesaikan apabila pertidaksamaan berikut terpenuhi

$$0 \leq X - n*k \leq S(n-1)$$

kita dapat memecah pertidaksamaan tersebut menjadi 2 pertidaksamaan berbeda yaitu

$$X - n*k \geq 0$$

$$X \geq n*k,$$

dan

$$X - n*k \leq S(n-1)$$

$$X \leq S(n-1) + n*k$$

untuk kemudahan penulisan misal definisikan $T1(n, k) = n*k$, dan $T2(n, k) = S(n-1) + n*k$ sehingga pertidaksamaan sebelumnya dapat disederhanakan menjadi $X \geq T1(n, k)$, dan $X \leq T2(n, k)$

perhatikan bahwa karena pada soal nilai k adalah tetap, sehingga nilai fungsi $T1(n, k)$ dan $T2(n, k)$ hanya terpengaruh oleh nilai n . perhatikan juga bahwa fungsi $T1(n, k)$ dan $T2(n, k)$ bersifat monotonic increasing.

ingat kembali bahwa pada soal tujuan kita adalah mencari waktu minimum untuk membasi seluruh robot, yang mana waktu disini kita definisikan dengan variabel n . tujuan kita selanjutnya adalah mencari nilai n yang paling minimum dimana nilai n akan memenuhi pertidaksamaan sebelumnya. dalam pencarian nilai n , kita dapat bayangkan fungsi $T2(n, k)$ berperan sebagai fungsi minimal nilai n dapat memenuhi pertidaksamaan, semakin besar nilai n semakin mungkin pertidaksamaan terpenuhi. sedangkan fungsi $T1(n, k)$ dapat dibayangkan sebagai fungsi pembatas, karena semakin besar nilai n maka pertidaksamaan semakin tidak mungkin dipenuhi.

dalam melakukan pencarian, kita dapat memanfaatkan fungsi $T2(n, k)$ untuk mencari nilai n pertama yang memenuhi, lalu kenapa hanya diperlukan nilai n pertama yang memenuhi?. misal kita mendapat nilai n pertama sehingga pertidaksamaan $T2(n, k)$ terpenuhi, karena sifat monotonic increasing maka $(n+1)$ pasti terpenuhi juga, namun apabila nilai n pertama yang didapat sebelumnya tidak memenuhi pertidaksamaan $T1(n, k)$, maka nilai $(n+1)$ juga tidak akan terpenuhi karena sifat monotonic increasing pada $T1(n, k)$. untuk nilai $(n-1)$ masih mungkin memenuhi pertidaksamaan $T1(n, k)$ namun dapat dipastikan bahwa tidak dapat memenuhi pertidaksamaan $T2(n, k)$. dengan alasan tersebut, mencari nilai n terkecil yang memenuhi pertidaksamaan $T2(n, k)$ sudah cukup, dan selanjutnya dapat kita lanjutkan dengan mengecek apakah nilai tersebut memenuhi pertidaksamaan $T1(n, k)$. untuk melakukan pencarian nilai n yang efisien dapat memanfaatkan algoritma binary search dengan memanfaatkan fakta bahwa $T2(n, k)$ bersifat monotonic.

Implementation note :

- hati-hati overflow pas ngitung $S(n)$ atau $k*n$, bisa pasang bates di binser buat ngecegah
- selain binser yang dimulai dari $low=0$, $high=INT_MAX$, bisa juga nilai $high$ dibatesin biar selalu memenuhin $T1(n, k)$, tapi ini gak perlu, karena penjelasan sebelumnya kalo cukup nyari n terkecil yang memenuhi $T2(n, k)$ dan tinggal perlu cek 1 nilai buat cek pertidaksamaan $T1(n, k)$

B. Caffeine Fighter



Caffeine Fighter

Prerequisite : Binary Search, Dijkstra

Perhatikan bahwa apabila kita memiliki kecepatan X, maka bobot edge pada graf akan berubah menjadi $\text{ceil}(\text{dist} / X)$. dari sini kita dapat mencoba seluruh kemungkinan kecepatan dan dapat mencari jarak minimum ke pos tujuan dengan menggunakan algoritma dijkstra. Namun nilai maksimum kecepatan yang mungkin adalah sebesar nilai maksimum jarak edge yang mana nilai maksimumnya adalah 10^9 sehingga kita tidak bisa mencoba seluruh kemungkinan.

Perhatikan observasi simpel dimana semisal $\text{Time}(X)$ adalah waktu minimum dari start ke finish apabila kita memiliki kecepatan X, perhatikan terdapat sebuah sifat yaitu $\text{Time}(X + 1) \leq \text{Time}(X)$. Untuk pembuktian terdapat pada bagian bawah. Karena nilai Time selalu berkurang seiring bertambahnya nilai X, maka fungsi Time memiliki sifat monotonic dan dapat dilakukan binary search untuk mencari nilai X terkecil sehingga $\text{Time}(X)$ memenuhi waktu yang diinginkan Syaro.

Pembuktian $\text{Time}(X + 1) \leq \text{Time}(X)$. Semisal kita definisikan $\text{Path}(X)$ adalah cara berjalan sehingga waktu untuk sampai ke tujuan dengan kecepatan X adalah minimum. Ada 2 kemungkinan kasus nilai $\text{Path}(X + 1)$ dilihat dari $\text{Path}(X)$

1. $\text{Path}(X + 1)$ berbeda dengan $\text{Path}(X)$, sudah pasti $\text{Time}(X + 1) \leq \text{Time}(X)$ karena menemukan jalan lebih baik
2. $\text{Path}(X + 1) = \text{Path}(X)$, disini waktu yang ditempuh pasti akan kurang dari atau sama dengan waktu dengan kecepatan sebelumnya, dikarenakan nilai path adalah penjumlahan dari nilai edge, sedangkan untuk setiap nilai pada edge pasti berlaku $\text{ceil}(\text{dist} / (X+1)) \leq \text{ceil}(\text{dist}/X)$ sehingga sudah pasti $\sigma(\text{ceil}(\text{dist} / (X+1))) \leq \sigma(\text{ceil}(\text{dist}/X))$

C. Constant Moderato



Murid SMA yang hobinya agak laen 

Prerequisite : Combinatoric Modulo

Perhatikan observasi simpel bahwa nilai hasil perampukan untuk sebuah bank independen dengan bank lain, sehingga kita bisa mengubah seluruh nilai bank menjadi nilai hasil perampukan. Info yang diberikan oleh Ayane merupakan sebuah subset dari seluruh bank, dan nilai dari sebuah subset adalah nilai terbesar pada subset tersebut. Pada soal kita diminta untuk mencari nilai harapan yaitu semisal nilai harapannya P/Q maka $P = \text{total seluruh nilai subset dengan ukuran } M$, dan $Q = \text{jumlah kemungkinan subset dengan ukuran } M$. Nilai Q dapat dihitung dengan mudah yaitu $C(n, m)$.

Untuk nilai P kita bisa mengurutkan terlebih dahulu nilai bank terurut menurun. Perhatikan semisal kita memilih sebuah indeks i , nilai bank pada indeks i akan menjadi nilai maksimum subset apabila untuk isi subset selanjutnya kita hanya memilih bank dengan indeks lebih dari i . Kita akan mencoba seluruh kasus dimana sebuah bank menjadi nilai terbesar subset. Untuk menghitung nilai dengan efisien perhatikan apabila kita telah memisalkan sebuah indeks i menjadi nilai maksimum, kita perlu mengambil $M-1$ bank lagi untuk menjadi subset, dan tersedia $n-i-1$ kemungkinan bank yang berada setelah indeks i . Sehingga banyaknya kemungkinan bank i menjadi bank dengan nilai maksimum adalah sebanyak $C(n-i-1, m-1)$.

D. Formasi Tentara



Sang pemimpin pasukan tentara Red Winter

Perhatikan matrik dengan bentuk mirip papan catur yaitu dengan nilai 0 dan 1 selang seling

0 1 0 1 0 1 0 1

1 0 1 0 1 0 1 0

0 1 0 1 0 1 0 1

1 0 1 0 1 0 1 0

0 1 0 1 0 1 0 1

1 0 1 0 1 0 1 0

0 1 0 1 0 1 0 1

1 0 1 0 1 0 1 0

0 1 0 1 0 1 0 1

1 0 1 0 1 0 1 0

Dapat dilihat bahwa untuk seluruh kemungkinan submatrix selisih jumlah angka 1 dan

jumlah angka 0 pasti tidak akan lebih dari 1.

E. Mantra Hujan



Sang pawang hujan dari Holoid yang akan menyinari harimu

Prerequisite : Greedy

Untuk memperbesar nilai kuadrat, kita akan mengambil secara greedy nilai dengan perbedaan nilai terbesar. Untuk mengambil nilai dengan perbedaan terbesar kita dapat melakukan sort pada array, dan mengambil secara selang seling nilai terbesar dan nilai terkecil array.

F. Peko Peko



Definitely just a normal bunny girl, nothing wrong here

Prerequisite : Point Update Range Query Data Structure (Segment Tree / Fenwick Tree)
biasanya kalo pengen ngubah string s jadi string t, kita bakal ngeloop semua hurufnya terus cek apakah dia sama atau gak, terus nanti jumlah ngubah = jumlah huruf gak sama. tapi kita perhatiin kalimat kita itu repeating, bisa coba nyari observasi dari situ. observasi yang bisa didapat itu misal dia definisiin $d[i]$ = berapa banyak harus ngubah biar $s[i]...s[i+3]$ berubah jadi "peko". misal perintah Pekora kita simbolin jadi $\text{Query}(i, K)$ dan pas kita ngubah huruf kita simbolin jadi $\text{Update}(j, c)$. bisa diliat dari observasi tadi, kalo kita mau $\text{Query}(i, K)$ jawaban kita itu $d[i] + d[i+4] + d[i+8] + \dots + d[i+K*4]$ atau kalo pake definisi matematis $\sum_{j=0}^{K-1} d[i + j * 4]$. tapi langkah query tadi tetep perlu nilai i dari 0 sampai K yang mana artinya untuk setiap query kompleksitasnya $O(K)$ dan untuk full solusi bakal $O(N * K)$ jadi kita perlu observasi tambahan. observasi lanjutannya adalah karena iterasinya bakal lompat 4-4, misal kita definisiin nilai parity sebuah query itu adalah $(i \bmod 4)$, bisa diliat

bahwa untuk 2 query dengan parity yang beda gak bakal nge akses nilai d yang sama. buat sedikit ilustrasi misal kita punya array d dengan panjang 20, buat setiap parity nilai array yang mungkin diakses adalah

parity 0 : d[4] d[8] d[12] d[16] d[20]

parity 1 : d[1] d[5] d[9] d[13] d[17]

parity 2 : d[2] d[6] d[10] d[14] d[18]

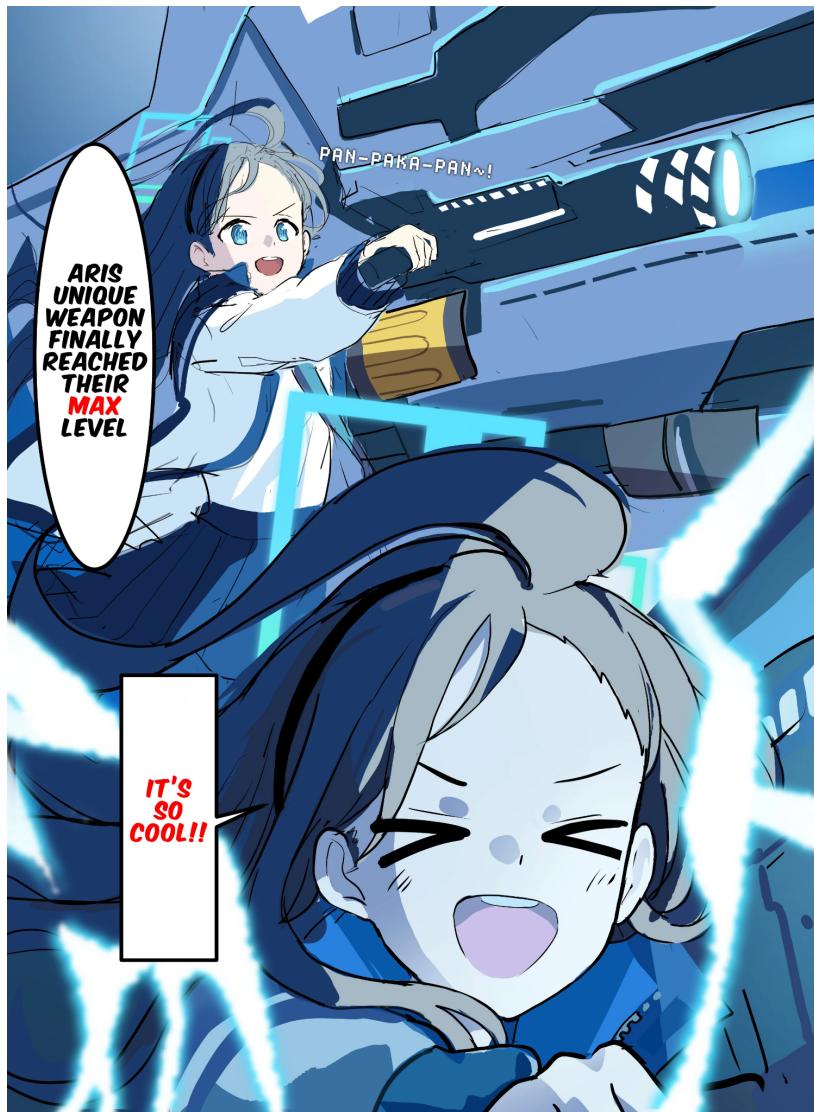
parity 3 : d[3] d[7] d[11] d[15] d[19]

disini kita bisa maintain 4 array baru untuk masing-masing parity misal p[i] itu array parity dengan parity 4

setelah kita punya 4 array baru untuk masing-masing parity ketika kita ngelakuin query(i, K) querynya bisa dijawab dengan cara : misal kita akses array dengan parity $(i \bmod 4)$, misal definisiindeks d[i] di array parity itu j, jawaban query kita sekarang adalah $p[i][j] + p[i][j + 1] + \dots + p[i][j + k - 1]$ atau kalo pake definisi matematis $\sigma[j=i][i+k-1](p[j])$. disini kita udah berhasil ngubah query dari yang iterasinya loncat jadi iterasi berurutan. untuk nylesaiin masalah query di array yang querynya range sum query atau matematisnya $\sigma[i=l][r](a[i])$ kita bisa pake struktur data yang udah terkenal, bisa pilih fenwick tree / segment tree.

sekarang buat nylesaiin masalah ada update(j,c) buat ngubah string, kita bisa dapet observasi bahwa untuk sebuah update cuma bakal ngubah maksimum 4 nilai array d. buat ngehandle perubahan nilai array parity bisa langsung pake operasi point update di segtree/fenwick dan karena nilai yang berubah cuma 4 perubahannya bisa di bruteforce.

G. Pixel Time



Sword of Light : Supernova

Prerequisite : Radial Sweep Line, Trigonometry

Untuk sebuah titik pada papan, misal kita ambil sudut yang dibentuk dengan garis sumbu x. Semisal kita urutkan seluruh titik berdasarkan sudut yang dibentuk, karena terdapat 2 titik pada setiap papan, misal kita sebut sudut yang dibentuk masing-masing titik adalah SudutIn dan SudutOut dengan ketentuan SudutIn < SudutOut. Perhatikan bahwa apabila kita menembakkan laser dengan sudut X, maka seluruh papan yang terkena adalah papan yang memenuhi kondisi SudutIn <= X <= SudutOut. Untuk melakukan pencarian nilai sudut X sehingga mengenai papan yang paling banyak adalah dengan menggunakan metode Radial Sweep Line.

H. Plum Blossom Garden



Para pengajar di Plum Blossom Garden

Prerequisite : DP on Tree, Tree Re-rooting, Modular Arithmetic, Number Theory

Apabila untuk setiap node kita mencatat nilai array berukuran 11 dimana $\text{array}[i]$ adalah berapa banyak node pada subtree dimana apabila dilakukan traversal dari node ke node subtree memiliki nilai $\% 11 = i$. Disini terdapat transisi simpel yaitu untuk setiap child pada node

$\text{dp}[\text{node}][(10*i + \text{value}[\text{node}]) \% 11] += \text{dp}[\text{child}][i]$.

Setelah mengisi seluruh nilai dp, apabila kita lihat nilai pada root kita bisa mengetahui berapa banyak pasangan (root, j) yang habis dibagi 11 untuk seluruh j . Karena kita ingin menghitung seluruh pasangan (i, j) kita perlu melakukan DP rooting untuk setiap root yang mana cara tersebut memiliki kompleksitas $O(n^2)$.

Disini kita bisa menggunakan teknik re-rooting dimana ketika melakukan evaluasi pada node selanjutnya kita akan mencatat nilai dp node sekarang untuk nantinya menjadi nilai child pada node selanjutnya yang nantinya kita akan melakukan update ulang untuk nilai dp node selanjutnya.

Note : mungkin terdapat solusi menggunakan divisibility rule pada mod 11, namun karena juri menemukan solusi yang bisa berlaku pada seluruh mod kecil, maka solusi ini yang dipilih

I. Rabbit House



Sang pemilik cafe dan pekerja paruh baya

Prerequisite : DP Bitmask, Combinatoric Modulo, DP Sum Over Subsets

Fun fact : solusi juri kompleksitasnya $O(n^2 * 2^n)$ tapi kebanyakan solusinya pake $O(n * 2^n)$ bahkan ada yang $O(n)$ 😱😱😱

Misal kita punya $dp[mask][cnt]$ dimana mask adalah representasi biner berukuran n dimana menandakan apakah pelanggan ini sudah memberi rating dan cnt adalah jumlah kopi yang sudah dirating sejauh ini. Untuk nilai dp tersebut kita akan menyimpan 2 nilai yaitu {total nilai, jumlah kombinasi}. Terdapat transisi simpel yaitu misal kita punya 2 buah mask dan kita ingin memasukkan nilai mask2 ke mask1 dimana pelanggan dengan mask2 ingin merating sebuah kopi yang sama, dan juga mask tersebut tidak saling tumpang tindih ($mask1 \& mask2 = 0$), terdapat transisi

$dp[mask1 | mask2][cnt + 1].total\ nilai += dp[m2][cnt].total\ nilai + dp[m2][cnt].jumlah\ kombinasi * nilai\ rating(m2)$

Dan juga terdapat transisi

$dp[mask1 | mask2][cnt + 1].jumlah\ kombinasi += dp[m2][cnt].jumlah\ kombinasi$

Nantinya untuk seluruh kemungkinan cnt kopi disini kita punya K buah kopi tetapi hanya sebanyak cnt kopi yang ter rating, maka akan terdapat $C(K, cnt)$ kemungkinan penyusunan cara me rating kopi yang equivalent dengan nilai yang telah kita hitung. Disini kita sudah mendapatkan solusi dp straightforward tetapi kompleksitasnya masih $O(n * 4^n)$

Terdapat beberapa improvisasi yang bisa dilakukan pada DP dan cara menghitung nilai kombinatorik, untuk menghitung nilai kombinatorik karena nilai yang perlu dihitung hanya sebanyak n, maka perhitungan kombinatorik bisa dihitung dengan bruteforce O(N) dibanding perhitungan kombinatorik O(log(n)) yang memerlukan pre compute. Untuk improvisasi DP perhatikan karena mask1 dan mask2 tidak boleh tumpang tindih, maka apabila kita memiliki mask1 dengan banyak bit aktif k, maka jumlah mask2 yang mungkin tidak tumpang tindih adalah sebanyak 2^{n-k} . Perhitungan seluruh kemungkinan mask1 dan mask2 yang perlu

dihitung adalah sebanyak $\sum_{i=0}^n 2^i * 2^{n-i} = n * 2^n$. karena kita perlu melakukan transisi untuk

seluruh cnt kopi maka kompleksitas total adalah $O(n^2 * 2^n)$.