# Editorial Final Schematics NPC Senior

**17 Oktober 2021**

| Problem | Expected Difficulty | Author |
|---|---|---|
| Angka Sial | Easy | Naufal Faadhilah |
| Berhitung | Hard | Muhammad Amin |
| Crash Loyale | Medium-Hard | Muhamad Affan |
| Dilanda Pandemi | Easy-Mediuum | Muhamad Affan |
| Eksplorasi Kota | Easy-Medium | Muhamad Affan |
| Fun Tour | Medium-Hard | Muhamad Affan |
| Good Game | Medium-Hard | Muhamad Affan |
| Hakuna Matata | Medium | Muhamad Affan |
| Indahnya Bintang di Langit | Medium | Muhamad Affan |
| Jangan-jangan Muat | Medium-Hard | Vania Rizky J W |
| Kelas Pemrograman | Hard | Muhamad Affan |
| Lingkaran Gelang | Easy | Aldo Yaputra Hartono |
| Minimumnya Berapa? | Easy | Zydhan Linnar Putra |

**Tester**

- **Andreas Cendranata**
- **Fadhil Musaad**
- **Haniif Ahmad Jauhari**
- **Nurlita Dhuha Fatmawati**

## Angka Sial

**Pembuat Soal : Naufal Faadhilah**

**Tag : Math, Brute Force**

**Expected Difficulty : Easy**

**Batasan**

Karena jumlah dari faktor bilangan $N$ yang dicari bernilai 13, maka cukup dilakukan pencarian faktor bilangan $N$ hingga faktor bernilai 13 saja. Kemudian, ketika telah didapat deretan faktor bilangan $N$, digunakan algoritma *brute-force* pada deret tersebut untuk mencari setiap kemungkinan sub-deret yang bernilai 13.

**Kompleksitas:** O(T * $2^{13}$)

## B - Berhitung

**Pembuat Soal : Muhammad Amin**

**Tag : Math, DP, Matrix Exponentiation**

**Expected Difficulty : Hard**

**Perhatikan disini nilai a,b,c,d dapat dipermutasikan, sehingga dapat diasumsikan bahwa a,b,c,d merupakan akar dari suatu polynomial sehingga:**

$$f(x) = (x-a)(x-b)(x-c)(x-d)$$

$$f(x) = x^4 - \sum_{p=a}^{d} p\,x^3 + \sum_{p,q=a}^{d} pq\,x^2 - \sum_{p,q,r=a}^{d} pqr\,x + abcd$$

$$\sum_{p=a}^{d} p = a+b+c+d = i$$

$$\left(\sum_{p=a}^{d} p\right)^2 = \sum_{p=a}^{d} p^2 + 2\sum_{p,q=a}^{d} pq$$

$$\sum_{p,q=a}^{d} pq = \frac{i^2-j}{2}$$

$$\sum_{p=a}^{d} p^3 - 3\left(\sum_{p,q,r=a}^{d} pqr\right) = \sum_{p=a}^{d} p\left(\sum_{p=a}^{d} p^2 - \sum_{p,q=a}^{d} pq\right)$$

$$\sum_{p,q,r=a}^{d} pqr = \frac{\sum_{p=a}^{d} p^3 - \sum_{p=a}^{d} p\left(\sum_{p=a}^{d} p^2 - \sum_{p,q=a}^{d} pq\right)}{3} = \frac{k - i(j - \frac{i^2-j}{2})}{3}$$

$$f(x) = x^4 - \sum_{p=a}^{d} p\,x^3 + \sum_{p,q=a}^{d} pq\,x^2 - \sum_{p,q,r=a}^{d} pqr\,x + abcd$$

$$f(x) = x^4 - i\,x^3 + \frac{i^2-j}{2}\,x^2 - \frac{k - i\left(j - \frac{i^2-j}{2}\right)}{3}\,x + abcd$$

$$f(a) + f(b) + f(c) + f(d) = 0$$

$$\sum_{p=a}^{d} p^4 - i \sum_{p=a}^{d} p^3 + \frac{i^2 - j}{2} \sum_{p=a}^{d} p^2 - \frac{k - i\left(j - \frac{i^2 - j}{2}\right)}{3} \sum_{p=a}^{d} p + 4abcd = 0$$

$$-4abcd = \sum_{p=a}^{d} p^4 - i \sum_{p=a}^{d} p^3 + \frac{i^2 - j}{2} \sum_{p=a}^{d} p^2 - \frac{k - i\left(j - \frac{i^2 - j}{2}\right)}{3} \sum_{p=a}^{d} p$$

$$abcd = \frac{l - i\,k + \frac{i^2 - j}{2}\,j - \frac{k - i\left(j - \frac{i^2 - j}{2}\right)}{3}\,i}{-4}$$

$$af(a) = a^5 - i\,a^4 + \frac{i^2 - j}{2}\,a^3 - \frac{k - i\left(j - \frac{i^2 - j}{2}\right)}{3}\,a^2 + \frac{l - i\,k + \frac{i^2 - j}{2}\,j - \frac{k - i\left(j - \frac{i^2 - j}{2}\right)}{3}\,i}{-4}\,a = 0$$

$$af(a) + bf(b) + cf(c) + df(d) = 0$$

**n starting from 3 or 4**

$$\sum_{p=a}^{d} p^{n+1} = i \sum_{p=a}^{d} p^n - \frac{i^2 - j}{2} \sum_{p=a}^{d} p^{n-1} + \frac{k - i\left(j - \frac{i^2 - j}{2}\right)}{3} \sum_{p=a}^{d} p^{n-2} - \frac{l - i\,k + \frac{i^2 - j}{2}\,j - \frac{k - i\left(j - \frac{i^2 - j}{2}\right)}{3}\,i}{-4} \sum_{p=a}^{d} p^{n-3}$$

$$Let \sum_{p=a}^{d} p^n = p_n$$

$$p_{n+1} = k_0 * p_n - k_1 * p_{n-1} + k_2 * p_{n-2} - k_3 * p_{n-3}$$

Kita dapat menghitung nilai $p_n$ menggunakan rekurens ini.

Tetapi, karena constrain N bisa mencapai 10^18, kita tidak bisa melakukan penghitungan secara naif.

Karena terjadi linear occurrence, maka dapat digunakan Matrix Exponentiation yang membuat time complexity perhitungan untuk setiap test case menjadi O(log(N)) dengan matrixnya adalah

$$\begin{matrix} k_0 & k_1 & k_2 & k_3 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$$

**Kompleksitas**: O(T * log(N))

## Crash Loyale

**Pembuat Soal : Muhamad Affan**

**Tag : Dynamic Programming, Probability**

**Expected Difficulty : Medium-Hard**

Pertama-tama, kita dapat merepresentasikan sebuah dek tangan sebagai sebuah *bitmask* yang terdiri dari 8 bit dengan bit ke-*i* bernilai 1 jika kartu ke-*i* sedang berada di dek tangan, dan 0 jika sebaliknya.

Kita dapat mendefinisikan $DP[i][j]$ = sebagai nilai harapan pemakaian *elixir* sampai ronde ke-*i,* serta dek tangan setelah ronde ke-*i* bernilai *j.*

Serta $prob[i][j]$ adalah probabilitas kita memiliki dek tangan bernilai *j* setelah ronde ke-i.

Didefinisikan pula sebuah *bitmask* $x$ sebagai *substitute* dari sebuah *bitmask* $y$ jika bit yang menyala pada hasil $x\ xor\ y$ berjumlah **tepat** 2 bit. Sebagai contoh, jika $x = (10110010)_2$ dan $y = (10011010)_2$ maka $x$ merupakan *substitute* dari $y$.

Nilai $prob[i][j]$ dapat dihitung menggunakan rumus :

$$prob[i][j] = \sum_{j'} \frac{1}{16} \times prob[i-1][j']$$

Dimana $j'$ merupakan *substitute* dari *j*.

Ini bisa didapatkan dari probabilitas nilai $j'$ ber-transisi menjadi $j$ dalam 1 ronde adalah $\frac{1}{16}$.

Dari sini, kita juga dapat melihat bahwa saat bertransisi dari *mask* $j'$ menjadi $j$, terdapat 1 buah bit yang menyala pada $j'$ dan mati pada $j$. Misalkan bit tersebut merupakan bit ke-$k$, maka artinya pada saat ronde ke-$i$, kita menggunakan kartu dengan indeks $k$.

Dengan begitu, kita dapat merumuskan $DP[i][j]$ sebagai :

$$DP[i][j] = \sum_{j'} \frac{1}{16} \times (A_k \times prob[i-1][j'] + DP[i-1][j']).$$

Dimana $j'$ merupakan *substitute* dari *j*.

Jawaban bisa didapatkan dari $\sum_{mask} DP[N][mask]$.

Perhatikan bahwa jumlah *bitmask* yang valid hanya ada sebanyak $\binom{8}{4} = 70$ sehingga total transisi untuk tiap ronde ada sebanyak $70 \times 4^2$

Total kompleksitas : $O(N \times 70 \times 4^2)$

## Dilanda Pandemi

**Pembuat Soal : Muhamad Affan**

**Tag : Data Structure,Graph**

**Expected Difficulty : Easy-Medium**

Kita dapat 'meratakan' *tree* menjadi sebuah segmen array dengan menggunakan *euler tour*. Dengan ini, suatu subtree $U$ dapat dinyatakan dalam sebuah *range* $(T_{in}[u], T_{out}[u])$ pada array.

Observasi : kita tidak butuh nilai asli dari nilai keuntungan tiap verteks, kita hanya butuh tanda apabila nilai keuntungan verteks tersebut positif atau negatif.

Dengan begitu kita dapat mengubah $A_i = 1$ jika $A_i > 0$ dan $A_i = -1$ jika sebaliknya. Selanjutnya, kita perlu melakukan perkalian pada *range* serta melakukan *update* dengan cepat. Ini dapat dilakukan menggunakan *segment tree* dengan kompleksitas waktu $O(\log(N))$.

Kompleksitas akhir : $O(Qlog(N))$.

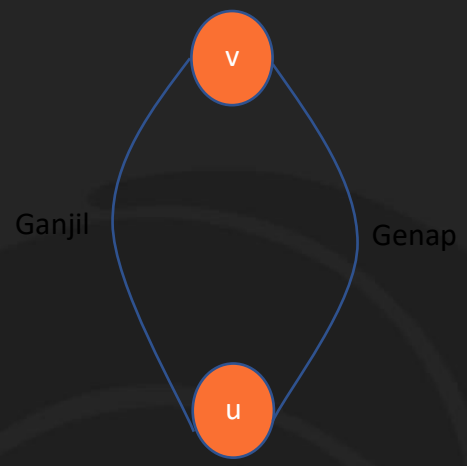
## Eksplorasi Kota

**Pembuat Soal : Muhamad Affan**

**Tag : Graph**

**Expected Difficulty : Easy-medium**

Observasi : jika terdapat 2 buah *path* yang tidak saling berpotongan dari $U$ ke $V$ dengan paritas panjang yang berbeda, maka *path* tersebut akan membentuk sebuah *cycle* dengan banyak verteks yang berjumlah ganjil.




Maka dari itu, soal ini dapat direduksi menjadi mengecek apakah terdapat sebuah *cycle* ganjil atau tidak.

Observasi 2 : sebuah graf memiliki *cycle* ganjil jika dan hanya jika graf tersebut bukan merupakan *bipartite*.

Dengan begitu, kita hanya perlu mengecek jika terdapat *connected component* yang bukan merupakan *bipartite*.

Kompleksitas akhir : $O(N + M)$

## Fun Tour

**Pembuat Soal : Muhamad Affan**

**Tag : Graph,Adhoc**

**Expected Difficulty : Medium-hard**

WLOG kita asumsikan $N \leq M$, serta $c_1 \leq c_2$.

Mari kita selesaikan untuk nilai $N = 1$ serta $N = 2$ terlebih dahulu.

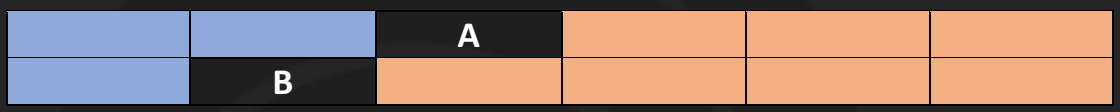
**Kasus N = 1**

Karena grid hanya terdiri dari 1 baris, maka hanya terdapat tepat 1 jalan yang menghubungkan tiap pasang kota dengan panjang $c_2 - c_1 + 1$.

**Kasus N = 2**

Terdapat 2 kondisi berbeda untuk kasus ini

- $c_1 = c_2$ dan $c_1 \neq 1, c_1 \neq M$ atau $c_1 = c_2 - 1$.




  Pada kondisi ini, jika kita menghapus kedua titik, maka grid akan terbagi menjadi 2 bagian dimana sebuah *path* yang menghubungkan titik $A$ dan $B$ pasti melewati salah satu dari 2 bagian tersebut. Sehingga panjang *path* terpanjang adalah ukuran maksimal dari kedua bagian yang terbentuk. Atau dengan kata lain $\max(c_1 + c_2, 2M - c_1 - c_2 + 2)$.

- Selain kasus di atas.
  Misalkan kita mewarnai sebuah titik $X(r, c)$ dengan warna putih jika $r + c$ bernilai genap dan hitam jika sebaliknya. Dapat dilihat bahwa jika titik $A$ dan $B$ memiliki warna yang berbeda, maka kita dapat membuat sebuah *path* dengan panjang $N * M$. Jika $A$ dan $B$ memiliki warna yang sama, maka kita dapat membuat sebuah *path* dengan panjang $N * M - 1$.

**Kasus N > 2**

Seperti pada kasus $N = 2$, kita dapat mewarnai sebuah titik $X(r, c)$ dengan warna putih jika $r + c$ bernilai genap dan hitam jika sebaliknya. Dapat diperhatikan bahwa dengan ini, grid dapat dimodelkan sebagai sebuah *bipartite graph*. Perhatikan bahwa dalam *bipartite graph*, setiap *path* akan terdiri dari verteks hitam dan putih yang saling bergantian.

Dengan begitu, kita dapat membagi permasalahan menjadi beberapa kasus :

1. Grid berukuran genap, serta $A$ dan $B$ memiliki warna yang berbeda.
   Perhatikan bahwa pada *bipartite graph*, sebuah *path* yang memiliki panjang genap pasti berawal dan berakhir di dua warna yang berbeda. Karena jumlah verteks berjumlah genap, sebuah *path* yang mengunjungi semua verteks pada grid pasti memiliki panjang genap serta memiliki warna awal dan akhir yang berbeda. Dengan begitu, jawaban untuk kasus ini adalah $N * M$.

2. Grid berukuran ganjil, serta $A$ dan $B$ memiliki warna putih.
   Seperti observasi pada kasus sebelumnya, sebuah *path* yang memiliki panjang ganjil pasti berawal dan berakhir di dua warna yang sama. Karena pada grid berukuran ganjil banyaknya verteks putih berjumlah lebih besar 1 dari banyaknya verteks hitam, maka sebuah *path* yang mengunjungi semua verteks pada grid pasti berawal dan berakhir pada verteks berwarna putih. Dengan begitu, jawaban untuk kasus ini adalah $N * M$.

3. Grid berukuran genap, serta $A$ dan $B$ memiliki warna yang sama.
   Sama seperti observasi pada kasus 1, karena $A$ dan $B$ memiliki warna yang sama, maka *path* yang menghubungkan harus memiliki panjang ganjil. Dengan begitu, panjang *path* maksimal yang bisa dibentuk adalah $N * M - 1$.

4. Grid berukuran ganjil, serta $A$ dan $B$ memiliki warna yang berbeda.
   Kasus ini dapat diselesaikan dengan analisis yang sama dengan kasus 3.

5. Grid berukuran ganjil, serta $A$ dan $B$ memiliki warna hitam.
   Kasus ini dapat diselesaikan seperti kasus 2, tetapi pada kasus ini jika panjang *path* sebesar $N * M$, maka *path* tersebut pasti hanya berawal serta berakhir pada verteks putih. Maka dari itu, jika kedua verteks berwarna hitam, maka panjang *path* terpanjang yang mungkin adalah $N * M - 2$.

6. Grid memenuhi syarat berikut:
   - Banyaknya baris berjumlah 3.
   - Banyaknya kolom berjumlah genap.
   - $A$ berwarna hitam serta $B$ berwarna putih.
   - $r_1 = 2$ dan $c_1 < c_2$ atau $r_1 \neq 2$ dan $c_1 < c_2 - 1$
     Contoh:

| | | | |
|---|---|---|---|
| | | | |
| A | | | |
| | B | | |

Meskipun kasus ini termasuk pada kasus 1, dapat dibuktikan bahwa kita tidak dapat membuat $path$ sepanjang $N * M$ dari $A$ ke $B$. Pada kasus ini, panjang *path* terpanjang yang mungkin adalah $N * M - 2$.

**Good Game**

**Pembuat Soal : Muhamad Affan**

**Tag : Adhoc**

**Expected Difficulty : Hard**

## Subsoal 2 (15 poin)

Observasi : Perhatikan bahwa jika $freq[i]$ adalah jumlah kemunculan nilai $i$ pada array $A$, maka kita dapat mereduksi permainan menjadi sebuah *nim game* klasik pada array $freq$. Dimana Elsi akan menang jika hasil $xor$ dari array $freq$ bernilai 0.

Dengan begitu, kita dapat mereduksi permasalahan menjadi menghitung jumlah *subset* yang memiliki hasil *xor* nilai-nilai frekuensinya bernilai 0.

Untuk menghitung jumlah *subset* yang memiliki hasil $xor$ bernilai 0, kita dapat menggunakan *dynamic programming*. Misalkan kita definisikan $DP[i][j]$ adalah banyak cara membuat *subset* dari $i$ bilangan pertama dengan hasil $xor$ bernilai $j$.

Kita dapat menghitung nilai $DP[i][j]$ dengan cara :

$$DP[i][j] = DP[i-1][j] + DP[i-1][j \; xor \; freq[i]]$$

Untuk mendapatkan hasil $xor$ bernilai $j$, kita dapat mengambil $subset$ dari $i-1$ yang memiliki hasil $xor$ bernilai $j$, atau kita dapat mengambil *subset* dari $i-1$ yang memiliki hasil $xor$ bernilai $j \; xor \; freq[i]$ lalu memasukan $i$ ke *subset* tersebut.

Jawaban bisa didapatkan dari $DP[N][0]$.

Tetapi, karena batasan yang besar, kita tidak dapat menggunakan solusi *dynamic programming* tersebut. Tetapi, kita dapat menggunakan ide dari solusi tersebut.

**Lemma :** $DP[i][j]$ bernilai 0 jika $j$ tidak dapat dibentuk dari hasil $xor$ *subset-subset* yang diambil dari $i$ bilangan pertama. Serta $DP[i][j]$ bernilai sama untuk semua $j$ dimana $j$ bisa dibentuk dari hasil $xor \; subset$ dari $i$ bilangan pertama.

Bukti untuk bagian pertama lemma cukup *trivial*.

Kita dapat membuktikan bagian kedua dari *lemma* menggunakan induksi.

Misalkan $S_i$ adalah nilai-nilai hasil $xor$ yang mungkin terbentuk dari $subset \; i$ bilangan pertama. Dengan $S_i = \{0\}$.

Dapat dilihat bahwa pada saat $i = 0$, $DP[0][0]$ bernilai 1 dan $DP[0][j]$ bernilai 0 untuk $1 \leq j \leq N$.

Pada saat melakukan transisi di indeks $i$, hanya terdapat 2 kemungkinan, yaitu $Freq[i]$ bukan merupakan elemen dari $S_{i-1}$, atau sebaliknya. Kita dapat meninjau masing-masing dari kedua kemungkinan ini.

**Kemungkinan 1 : $freq[i]$ bukan merupakan elemen dari $S_{i-1}$**

Saat melakukan transisi pada indeks $i$, untuk setiap nilai $j$, hanya paling banyak 1 diantara nilai $DP[i-1][j]$ dan $DP[i-1][j\ xor\ freq[i]]$ yang bernilai tidak sama dengan 0. Karena hanya paling banyak 1 diantara $j$ dan $j\ xor\ freq[i]$ yang merupakan elemen dari $S_{i-1}$. Ini dikarenakan jika $j$ dan $j\ xor\ freq[i]$ keduanya merupakan elemen dari $S_{i-1}$, maka $freq[i]$ pasti juga merupakan elemen dari $S_{i-1}$ yang menimbulkan kontradiksi dengan kondisi awal.

Misalkan $DP[i-1][j] = K$ untuk setiap $j$ merupakan elemen dari $S_{i-1}$. Maka, $DP[i][j] = DP[i-1][j] + DP[i-1][j\ xor\ freq[i]] = K$ untuk setiap $j$ merupakan elemen dari $S_i$ dan $DP[i][j] = 0$ jika sebaliknya.

**Kemungkinan 2 : $freq[i]$ merupakan elemen dari $S_{i-1}$**

Mirip seperti observasi pada kemungkinan pertama,pada kasus ini, saat melakukan transisi pada indeks $i$, kedua nilai $DP[i-1][j]$ dan $DP[i-1][j\ xor\ freq[i]]$ bernilai tak 0 jika $j$ merupakan elemen dari $S_{i-1}$.

Misalkan $DP[i-1][j] = K$ untuk setiap $j$ merupakan elemen dari $S_{i-1}$. Maka, $DP[i][j] = DP[i-1][j] + DP[i-1][j\ xor\ freq[i]] = 2K$ untuk setiap $j$ merupakan elemen dari $S_i$ dan $DP[i][j] = 0$ jika sebaliknya.

Dengan begitu, kita dapat melihat bahwa pada saat melakukan transisi pada indeks $i$, jika $freq[i]$ tidak bisa dibentuk dari hasil $xor\ subset$ dari *i* indeks pertama, maka nilai-nilai DP tidak akan berubah. Dan jika sebaliknya, maka nilai DP akan bertambah menjadi 2 kali lipat.

Dengan begitu, untuk menghitung jawaban, kita hanya perlu meng-*keeptrack* hasil hasil *xor subset* pada tiap indeks di dalam sebuah $set$. Jika nilai frekuensi indeks tersebut ada di dalam $set$, maka jawaban kita kalikan dengan 2. Jika tidak, maka kita lakukan *update* pada isi $set$ tersebut dengan memasukkan hasil $xor$ elemen-elemen pada $set$ dengan $freq[i]$.

Perhatikan bahwa jumlah elemen pada set tidak akan melebihi $2^k$ dimana $k$ merupakan bilangan terkecil yang memenuhi $2^k > N$.

Kompleksitas akhir : $O(\mathrm{N}\log(Max(A_i)))$

## Hakuna Matata

**Pembuat Soal : Muhamad Affan**

**Tag :Dynamic Programming**

**Expected Difficulty : Medium**

Observasi : bilangan-bilangan yang ada tidak penting, yang penting hanya urutan bilangan tersebut dari paling rendah ke paling tinggi.

Kita dapat memandang permasalahan seperti memasukkan bilangan-bilangan tersebut satu persatu ke sebuah *array* dari bilangan terkecil hingga bilangan terbesar.

Misalkan kita sudah memasukkan $i$ bilangan pertama serta terdapat $j$ buah indeks dimana $A_j > A_{j+1}$. Misalkan indeks-indeks ini disebut sebagai indeks turun.

Saat memasukkan bilangan ke-$i+1$, jika kita memasukkan bilangan tersebut di indeks sebelum indeks-indeks turun, maka jumlah indeks turun akan bertambah sebanyak 1. Sebaliknya, jika kita memasukkan bilangan tersebut di indeks setelah indeks-indeks turun, maka jumlah indeks turun akan tetap.

Dengan begitu, kita dapat mendefinisikan $DP[i][j]$ sebagai jumlah konfigurasi menggunakan $i$ bilangan terkecil dengan indeks turun sebanyak $j$.

Kita dapat menghitung $DP[i][j]$ dengan cara

$$DP[i][j] = (j+1) * DP[i-1][j] + (i-j) * DP[i-1][j-1]$$

Jawaban bisa didapatkan dari nilai $DP[N][K]$.

Tetapi, menyimpan nilai $DP$ secara naif akan melebihi *memory limit*. Untuk itu, kita dapat menggunakan teknik *flying table* dalam penghitungan nilai $DP$.

Kompleksitas akhir : $O(N^2)$

Kompleksitas memori : $O(N)$

## Indahnya Bintang di Langit

**Pembuat Soal : Muhamad Affan**

**Tag : Geometry,Two Pointer**

**Expected Difficulty : Medium**

Untuk suatu titik $A$ yang menjadi titik pusat sudut, kita dapat melakukan *sorting* pada titik-titik lain berdasarkan *polar angle* terhadap titik $A$. Untuk setiap indeks $L$, kita dapat mencari indeks terbesar $R$ dimana sudut yang terbentuk dari triplet $(A, L, R)$ memiliki ukuran lebih kceil dari $90°$. Ini dapat dilakukan menggunakan teknik *two pointer* dimana untuk indeks pertama, kita dapat melakukan pencarian secara naif hingga menemukan $R$ terbesar dimana triplet $(A, L, R)$ membentuk sudut yang lebih kecil dari $90°$.

Setelah itu, kita dapat menggeser pointer $L$ ke titik selanjutnya dan kembali menggeser pointer $R$ hingga sudut yang terbentuk sudah lebih dari $90$ °.

Dengan begitu, kita dapat menghitung jumlah triplet dengan titik pusat $A$ dalam waktu $O(N)$.

Kita dapat menghitung jawaban akhir dengan mengulang langkah diatas untuk semua $N$ titik yang ada.

Kompleksitas akhir : $O(N^2)$

**Jangan-Jangan Muat**

**Pembuat Soal : Vania Rizky J W**

**Tag : Math**

**Expected Difficulty : Medium-Hard**

**Batasan**

$\mathbf{2 \leq P \leq Q \leq R \leq 10^6}$

$\mathbf{1 \leq A, B \leq 10^6}$

$\boldsymbol{2 \mid A \times B}$

$\boldsymbol{P, Q, R = 2 \times bilangan\ ganjil}$

$\mathbf{1 \leq N \leq 10^{18}}$

Sebenarnya di batasan sudah tertera cluenya, yaitu $\frac{P \times Q \times R}{8}$ bilangan ganjil

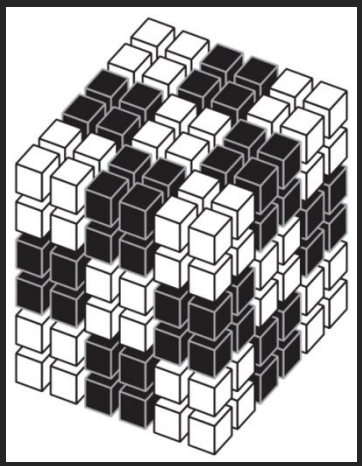
Pakai contoh studi kasus saja

Apakah bisa menaruh 53 balok berukuran $1 \times 1 \times 4$ ke dalam kotak berukuran $6 \times 6 \times 6$

Jika ditinjau dari volumenya

$$53 \times 1 \times 1 \times 4 < 6 \times 6 \times 6$$

memang memenuhi, namun pada kenyataannya tidak akan bisa menaruh ke 53 balok tersebut



Potong kotak $6 \times 6 \times 6$ menjadi $216$ kubus satuan. Warnai setiap kubus $2 \times 2 \times 2$ hitam putih berselang seling seperti pada gambar di samping.

Maka banyak kotak hitam adalah $13 \times 8 = 104$

dan banyak kotak putih adalah $14 \times 8 = 112$

Observasi bahwa dimanapun balok $1 \times 1 \times 4$ ditempatkan, akan selalu memiliki 2 kotak hitam dan 2 kotak putih

Maka 53 balok berukuran $1 \times 1 \times 4$ membutuhkan 106 kotak hitam dan 106 kotak putih. Padahal banyaknya kotak hitam hanya ada 104. Sehingga tidak memungkinan menaruh 53 balok berukuran $1 \times 1 \times 4$ ke kotak $6 \times 6 \times 6$

Untuk contoh kasus dimana banyak balok hitam dan putih memenuhi namun tetap tidak muat seperti pada kasus meletakkan 37 balok $1 \times 1 \times 100$ ke kotak $6 \times 6 \times 198$ diserahkan kepada pembaca.

**Kelas Pemrograman**

**Pembuat Soal : Muhamad Affan**

**Tag : Data Structure,Geometry**

**Expected Difficulty : Hard**

Untuk *query* 1 sampai 3, kita bisa menggunakan *implicit treap* untuk melakukannya dalam kompleksitas $O(\log(N))$.

Untuk *query* ke-4, kita dapat mengimplementasikan algoritma welzl untuk mencari lingkaran terkecil dalam waktu linear terhadap jumlah titik.

Misalkan kumpulan titik yang diberikan adalah $S$, pencarian dilakukan secara rekursif dimana pada setiap langkah, akan dipilih sebuah titik $P$ secara acak dan *uniform* lalu mencari lingkaran terkecil yang mengandung semua titik di $S$ kecuali titik $P$. Jika $P$ termasuk di dalam lingkaran tersebut, maka pencarian dapat dihentikan.

Jika tidak, pencarian akan dilakukan sambil meng-*keeptrack* titik-titik yang berada di sisi lingkaran saat ini. *Base case* dari algoritma ini adalah saat ukuran $S \leq 3$, dimana jika ukuran $S \leq 3$, kita dapat menghitung besar lingkaran terkecil secara manual.

Kompleksitas akhir : $O(Q \log N + M)$ dimana $M$ merupakan total titik yang diproses pada *query* 4.

## Lingkaran Gelang

**Pembuat Soal : Aldo Yaputra Hartono**

**Editorialist : Aldo Yaputra Hartono**

**Tag : Ad Hoc**

**Expected Difficulty : Easy**

Observasi :

Tanpa mempedulikan banyaknya manik-manik, untuk semua gelang yang tersusun dari manik-manik dengan jari-jari yang sama besarnya akan memiliki selisih luas daerah hitam dan putih yang sama besarnya.

Total sudut semua manik = $360N$

Total sudut daerah hitam = Total sudut poligon $N$ sisi

= $180(N-2)$

Total sudut daerah putih = Total sudut semua manik – Total sudut daerah hitam

= $360N - 180(N-2)$

= $180(N+2)$

Luas daerah juring hitam = $\frac{Total\ sudut\ daerah\ hitam}{360} \times Luas\ 1\ manik$

= $\frac{180(N-2)}{360} \times Luas\ 1\ manik$

= $\frac{N-2}{2} \times Luas\ 1\ manik$

Luas daerah juring putih = $\frac{Total\ sudut\ daerah\ putih}{360} \times Luas\ 1\ manik$

= $\frac{180(N+2)}{360} \times Luas\ 1\ manik$

= $\frac{N+2}{2} \times Luas\ 1\ manik$

Selisih luas = Luas daerah juring putih – Luas daerah juring hitam

= $\left(\frac{N+2}{2} - \frac{N-2}{2}\right) \times Luas\ 1\ manik$

= $2 \times Luas\ 1\ manik$

= $2\pi R^2$

Dalam perhitungan poin terbesar dapat difokuskan membandingkan nilai $R$ tiap gelang karena $2\pi$ bernilai konstan. Pemain satu-satunya yang memiliki gelang dengan nilai $R$ terbesar adalah pemenangnya.

Kompleksitas $O(T)$

**Jangan-Jangan Muat**

**Pembuat Soal : Zydhan Linnar Putra**

**Tag : Greedy**

**Expected Difficulty : Easy**

Karena *array* sudah terurut non-descending, maka cek saja jumlah elemen-elemen

$$a_N, a_{N-1}, \dots, a_p$$

Cari $p$ terkecil sehingga

$$a_N + a_{N-1} + \cdots + a_p \geq K$$