TOKI Regular Open Contest #14 Editorial

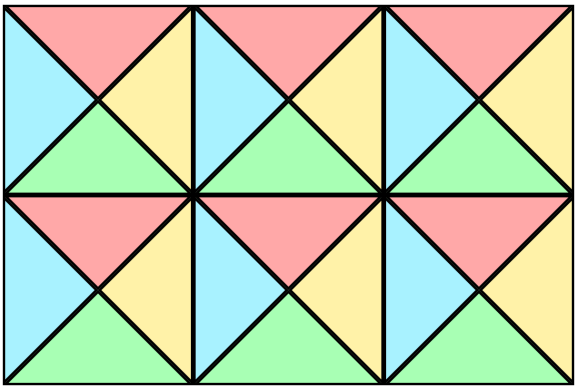
(English version is available on page 8.)

Penulis soal

Judul Soal		**Author**	**Editorialis**
Div 2A	Permainan Ubin	**steven_n**	**steven_n**
Div 2B	Permainan Menara	**steven_n**	**prabowo**
Div 1A	Permainan FPB	**steven_n**	**prabowo**
Div 1B	Permainan Xor	**steven_n**	**steven_n**
Div 1C	Permainan Rotasi	**steven_n**	**steven_n**
Div 1D	Permainan Platform	**steven_n**	**steven_n**
Div 1E	Permainan Bertahan Hidup	**steven_n**	**steven_n**
Div 1F	Permainan Pohon	**steven_n**	**steven_n**

Div 2A. Permainan Ubin

Perhatikan bahwa kita dapat memecah persegi panjang $N \times M$ menjadi sekumpulan persegi 2×2, yang masing-masing dapat diisi dengan 4 buah segitiga. Banyaknya segitiga yang dibutuhkan adalah $(N / 2) \times (M / 2) \times 4 = N \times M$.



Kompleksitas waktu: O(1)

Div 2B. Permainan Menara

Salah satu pembangunan menara yang memaksimalkan jawaban adalah dengan membangun menara hitam pada baris genap, dan menara putih pada baris ganjil.
Keluaran akan berbentuk seperti:

```
WWWW
BBBB
WWWW
BBBB
WWWW
```

Kompleksitas waktu: O(NM)

Div 1A. Permainan FPB

Perhatikan bahwa di antara semua bilangan 1 sampai N, hanya terdapat $floor(N / 2)$ buah bilangan yang memiliki faktor 2. Oleh karena itu, untuk setiap bilangan A_i kita perlu melakukan operasi GCD dengan $floor(N / 2) + 1$ buah angka berbeda di array B. Sehingga kita cukup melakukan rangkaian operasi “G” yang disusul dengan operasi “LG” sebanyak $floor(N / 2)$ kali atau operasi “RG” sebanyak $floor(N / 2)$ kali.

Kompleksitas waktu: O(N)

Div 1B. Permainan Xor

Misalkan sum adalah *xor value* dari array A dan x sebagai bit terbesar yang muncul setidaknya sekali dan sebanyak genap kali pada *array* A. Jika terdapat x yang memenuhi, maka jawabannya adalah $sum \mid (2^{x+1} - 1)$, dengan $\mid$ merupakan operasi bitwise or. Namun jika tidak ada x yang memenuhi, maka jawabannya adalah sum.
Hal ini dikarenakan terdapat sebuah tumpukan yang bit x-nya nyala, kemudian kita dapat mengurangi tumpukan ini sehingga bit x menjadi padam (sehingga *xor value* pada bit x menjadi nyala), bit yang lebih besar x tetap konstan, dan bit yang lebih kecil x dapat diubah menjadi apapun.
Hal ini dapat dilakukan dalam O(N log A) atau O(N) dengan operasi bitwise.

Kompleksitas waktu: O(N)

Div 1C. Permainan Rotasi

Selalu terdapat rangkaian operasi yang memindahkan N buah elemen terbesar ke array A, sehingga kita cukup mengeluarkan penjumlahan dari N elemen terbesar.

Bukti:

Misalkan (k, l) sebagai indeks dari pasangan A_k dan B_l yang ingin ditukar.

Kita ingin setelah penukaran, A_k berada pada array B, dan B_l berada pada array A, dan untuk semua elemen lainnya, apabila ia berada pada A sebelum penukaran, akan tetap berada pada A, dan apabila ia berada pada B sebelum penukaran, akan tetap berada pada B.

Perhatikan bahwa operasi *rotation* di posisi i sama saja dengan menukarkan A_i dengan B_{i+1}.

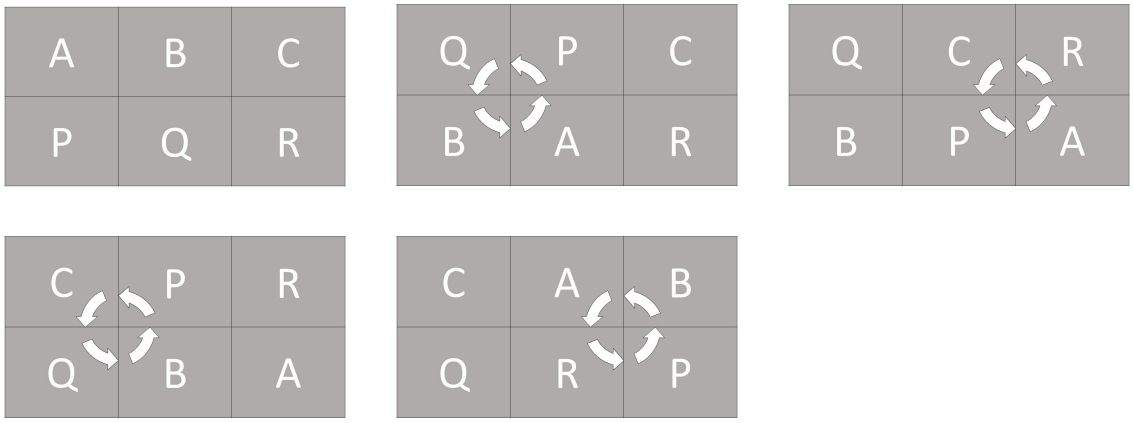
Misalkan $k < l$, maka terdapat dua buah kasus:

Jika $l - k = 1$, maka kita bisa langsung menukarkan A_k dengan B_l.

Jika $l - k > 1$, kita bisa melakukan operasi *rotation* pada $i = k$, yaitu menukarkan A_k dengan B_{k+1}. Namun bilangan yang kita inginkan bukanlah B_{k+1}, sehingga kita perlu menukarkan B_{k+1}, yang sekarang berada di posisi A_{k+1}, dengan B_l. Perhatikan bahwa operasi ini sama saja dengan mengurangi jarak pasangan yang ingin ditukar sebesar 1, dan jika melakukan operasi ini terus menerus maka jarak mereka akan menjadi 1.

Perhatikan bahwa dengan melakukan operasi *rotation* di index yang sama 3 kali kita sama saja dengan menukarkan A_{i+1} dengan B_i. Sehingga untuk kasus $l > k$, kita bisa melakukan rangkaian operasi yang sama seperti $k < l$.

Untuk kasus $k = l$, kita dapat melakukan rangkaian operasi dibawah:



Perhatikan bahwa untuk semua pasangan (A_k, B_l) dengan $k = l$, setelah rangkaian operasi tersebut tidak berlaku lagi kondisi $k = l$ untuk semua pasangan (A_k, B_l).

Kompleksitas waktu: O(N + A) atau O(N log N)

Div 1D. Permainan Platform

Misalkan i sebagai posisi robot.

Dalam rangkaian operasi optimal, terdapat x sehingga apabila $i < x$ robot hanya melakukan operasi 1, yaitu operasi menambah tinggi H_i, dan apabila $i \geq x$ robot hanya melakukan operasi 2, yaitu operasi mengurangi tinggi H_{i+1}, untuk semua ronde.
Hal ini dikarenakan, apabila Anda melakukan operasi 2 diikuti oleh operasi 1, maka operasi 2 itu dapat diganti dengan operasi 1 dan banyaknya operasi tidaklah bertambah.

Setelah melakukan operasi pertama sampai posisi x dalam 1 ronde, maka ketinggian H_i berubah menjadi H_{i-1} untuk $i < x$.

Definisikan $cost[i]$ sebagai banyak operasi yang diperlukan agar robot bisa sampai ke posisi i hanya dengan melakukan operasi 1, untuk semua k ronde, maka:

- $cost[i] = 0$, untuk $i = 1$.
- $cost[i] = cost[i - 1] + (H_i - H_{i-1}) * \min(i - 1, k)$

dengan $(H_i - H_{i-1}) * \min(i - 1, k)$ merupakan banyak operasi yang diperlukan agar k buah robot dapat berpindah dari ketinggian H_{i-1} ke H_i.

Perhatikan bahwa banyak operasi agar robot-robot bisa sampai posisi N dari x hanya dengan melakukan operasi 2 adalah $\sum_{j=i}^{N} (H_j - H_x)$

Semua x dalam $1 \le x \le N$ dapat dicoba dalam kompleksitas O(N).

Kompleksitas waktu: O(N)

Div 1E. Permainan Bertahan Hidup

Kita dapat mengiterasikan i dari N sampai 1.

Misalkan $goal$ adalah batas hari minimal kita harus bertahan hidup tanpa kehabisan bahan makanan supaya bisa menyelesaikan game. Nilai awal dari $goal$ adalah D.
Dalam suatu iterasi kita akan melakukan:

- $goal := T_i - 1$, jika $T_i + C_i > goal$. Hal ini dikarenakan bantuan yang diberikan cukup untuk bertahan sampai $goal$, sehingga kita cukup menyediakan makanan sampai hari ke-T_i.
- $goal := goal - C_i$, jika $T_i + C_i + R_i > goal$. Hal ini dikarenakan untuk bisa mencapai $goal$, apabila kita bisa mencapai hari T_i kita memerlukan makanan tambahan sebanyak $goal - T_i - C_i$ (dan nilai ini belum melebihi R_i). Sehingga, kita sama saja perlu bertahan hidup sampai hari ke $T_i + goal - T_i - C_i$ atau $goal - C_i$.

- Jika tidak ada kondisi di atas yang memenuhi, maka kita bisa bertahan hidup sampai hari ke-D tanpa mengandalkan bantuan hari ke-T_i, sehingga bantuan ini bisa diabaikan.

Jawaban dari permasalahan ini adalah nilai $goal$ setelah iterasi selesai.

Kompleksitas waktu: O(N)

Div 1F. Permainan Pohon

Kita akan mengelompokkan operasi-operasi menjadi kelompok operasi berukuran K untuk diselesaikan secara *offline*.

Untuk memproses K buah operasi, kita akan melakukan DFS. Selama melakukan DFS, kita akan menjawab operasi 2 dan meng-*update* tree sesuai dengan tree setelah K buah operasi.
Definisikan ukuran pada sebuah subtree sebagai banyaknya node putih pada subtree tersebut. Untuk menjawab operasi 2 di verteks u, kita perlu mengetahui ukuran subtree dari anak-anak dari verteks u setelah operasi tertentu. Untuk itu, kita dapat menyimpan nilai $(order, diff)$ untuk semua operasi yang berada di semua subtree v, dengan $order$ merupakan urutan operasi, $diff$ merupakan perubahan ukuran subtree v, dan v merupakan anak dari verteks u. Simpan semua nilai tersebut dalam sebuah *vector*, yang diurutkan berdasarkan $order$.
Misalkan con_u adalah *vector* yang berisi pasangan-pasangan nilai dari semua operasi pada *subtree* u, dan $lazy_u$ adalah perubahan nilai $diff$ pada con_u yang tidak diubah secara langsung. $lazy_u$ hanya menyimpan perubahan $diff$ yang indeksnya ganjil. Untuk indeks genap, perubahannya adalah $-lazy_u$.

Dalam DFS, kita tidak akan mengakses con_u secara langsung untuk setiap verteks u. Saat mengkonstruksikan con_u, apabila hanya terdapat sebuah v yang mempunyai nilai con_v yang tidak kosong, maka kita cukup memindahkan con_v ke con_u dengan melakukan swap vector, kemudian menambah lazy ke con_u. Isi dari con_u hanya diakses setiap kali kita mau menggabungkan lebih dari satu con_v. Saat penggabungan, kita perlu meng-*update* isi con_u sesuai $lazy_u$.

Karena con_u hanya diakses setiap kita ingin menggabungkannya dan pada saat ingin menjawab operasi 2, maka con_u tidak akan diakses lebih dari $K - 1$ kali, sehingga DFS ini berjalan dalam $O(N + K^2)$. Jika $K = sqrt(Q)$, maka kompleksitasnya menjadi semua DFS menjadi $O((N + Q)\, sqrt(Q))$.

Kompleksitas waktu: O((N+Q) sqrt(Q))

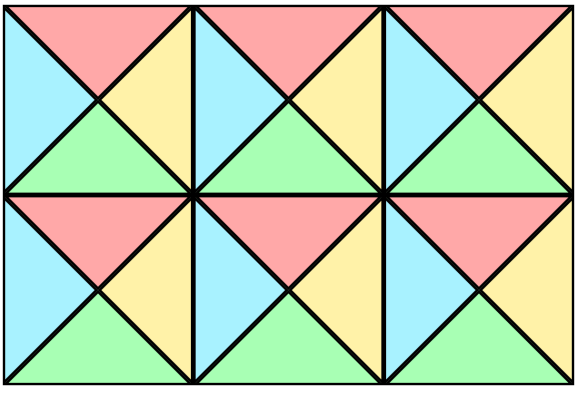
TOKI Regular Open Contest #14 Editorial

Problem Authors

Problem Title		Author	Editorialist
Div 2A	Tiling Game	**steven_n**	**steven_n**
Div 2B	Tower Game	**steven_n**	**prabowo**
Div 1A	GCD Game	**steven_n**	**prabowo**
Div 1B	Xor Game	**steven_n**	**steven_n**
Div 1C	Rotation Game	**steven_n**	**steven_n**
Div 1D	Platforming Game	**steven_n**	**steven_n**
Div 1E	Survival Game	**steven_n**	**steven_n**
Div 1F	Tree Game	**steven_n**	**steven_n**

Div 2A. Tiling Game

Notice that we can break the $N \times M$ rectangles into a bunch of 2×2 squares, where each can be tiled using 4 triangles. The number of triangles needed is $(N / 2) \times (M / 2) \times 4 = N \times M$ .4



Time Complexity: O(1)

Div 2B. Tower Game

One way to build towers that maximize the answer is to build black towers on even rows and white towers on odd rows.
The output will look as:

```
WWWW
BBBB
WWWW
BBBB
WWWW
```

Time Complexity: O(NM)

Div 1A. GCD Game

Notice that from the numbers 1 to N, only $floor(N / 2)$ of them has factor 2. Therefore, for each A_i we need to do the GCD operations on $floor(N / 2) + 1$ different numbers in array B. Therefore, we can simply perform the sequence of operation “G” followed by operations “LG” $floor(N / 2)$ times or operations “RG” $floor(N / 2)$ times.
Time Complexity: O(N)

Div 1B. Xor Game

Let sum be the *xor value* of array A and x be the largest bit that appears at least once and appears even times on *array* A. If there exists such x, then the answer is $sum \mid (2^{x+1} - 1)$, where $\mid$ is the bitwise or operation. If there is no such x, then the answer is sum.
This is because there exists a pile where the x-th bit is on, then we can subtract that pile such that bit x is off (therefore the *xor value* of bit x became on), bits larger than x stay constant, and bits smaller than x can be changed into anything.
These can be done in O(N log A) or O(N) using bitwise operations.
Time Complexity: O(N)

Div 1C. Rotation Game

There always exists a sequence of operations to move the largest N elements to the array A, so we can simply output the sum of the largest N elements.

Proof:

Let (k, l) be the indices of the pair A_k and B_l that need to be swapped.

We want that after the swap, A_k is on the array B, and B_l is on the array A, and for every other element, if it was on A before the swap, it stays on A, and if it was on B before the swap, it stays on B.

Notice that the rotation operation at position i is the same as swapping A_i with B_{i+1}.

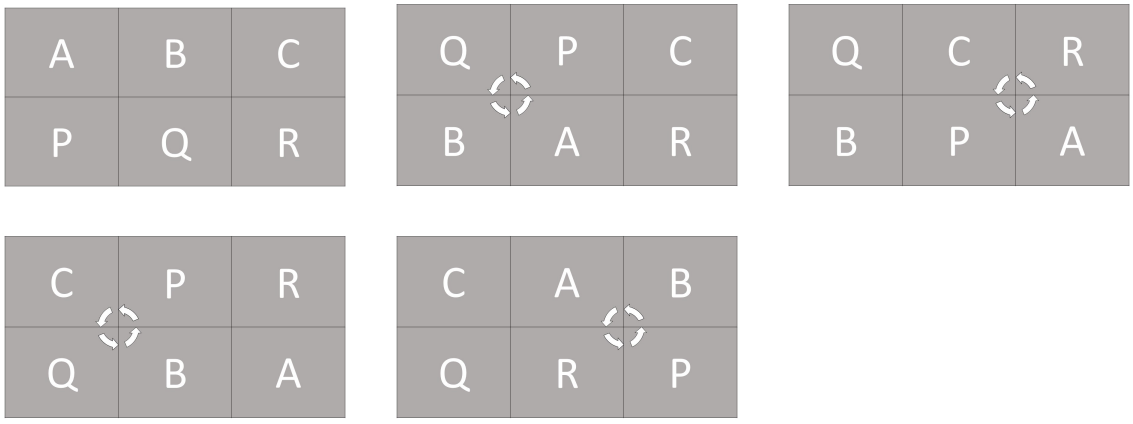
Let $k < l$, then there are two cases:

If $l - k = 1$, then we can immediately swap A_k with B_l.

If $l - k > 1$, then we can perform the rotation operation on $i = k$, which is swapping A_k with B_{k+1}. But the element we wanted to swap with is not B_{k+1}, so we need to swap B_{k+1}, which is now on position A_{k+1}, with B_l. Notice that this operation decreases the distance of the pair by one, and if we repeatedly perform this operation, the distance will become 1.

Notice that by performing the rotation operation at the same index three time is equivalent to swapping A_{i+1} with B_i. Therefore for the case $l > k$, we can do similar operations as $k < l$.

For the case $k = l$, we can perform the sequence of operation as follow:



Notice that for every pair (A_k, B_l), where $k = l$, after that sequence of operations, the condition $k = l$ is no longer satisfied for very pair (A_k, B_l).
Time Complexity: O(N)

Div 1D. Platforming Game

Let i be the position of the robot.

In an optimal sequence of operations, there exists x such that if $i < x$ then the robot only perform operation 1, which is the operation that increases the height H_i, and if $i \geq x$ then the robot only perform operation 2, which is decreasing the height H_{i+1}, for all rounds.
This is because, if you perform operation 2 followed by operation 1, then operation 2 can be replaced by operation 1, and the number of operations does not change.

After performing the first operation until position x in one round, the height H_i will become H_{i-1} for $i < x$.

Define $cost[i]$ be the number of operation required so that the robot can reach position i using only operation 1, for all k rounds, then:

- $cost[i] = 0$, for $i = 1$.
- $cost[i] = cost[i - 1] + (H_i - H_{i-1}) * \min(i - 1, k)$

where $(H_i - H_{i-1}) * \min(i - 1, k)$ is the number of operation required for k robots to move from height H_{i-1} to H_i.

Notice that the number of operation required for the robots to position N fromi x using only operation 2 is $\sum_{j=i}^{N} (H_j - H_x)$

All x in $1 \leq x \leq N$ can be tried in complexity O(N).
Time Complexity: O(N)

Div 1E. Survival Game

Let's iterate i from N to 1.

Define $goal$ to be the lower bound number of days we can survive without running out of food to finish the game. Initially, the value of $goal$ is D.
In each iteration, we can do:

- $goal := T_i - 1$, if $T_i + C_i > goal$. The aid given is enough to survive until $goal$, hence we only need food supply until T_i-th day.
- $goal := goal - C_i$, if $T_i + C_i + R_i \geq goal$. To reach $goal$, If we can survive until day T_i, we need and addition of $goal - T_i - C_i$ food (this value doesn't exceed R_i). In other words, we need to survive until day $T_i + goal - T_i - C_i$ or $goal - C_i$.
- If none of the condition above is true, then we can survive without relying on the aid from day T_i, so we can ignore this aid.

The answer to the problem is the value of $goal$ after the iteration is finished.
Time Complexity: O(N)

Div 1F. Tree Game

We will bucket the operations into bucket of size K to be solved offline.

To process K operations, we will perform DFS. While performing DFS, we will answer operation 2 and update the tree according to the K operations.
Define the size of a subtree as the number of white nodes in that subtree. To answer operation 2 in vertex u, we need to know the size of the subtrees of the children of vertex u after some operations. For that, we can store the value $(order, diff)$ for every operation in the subtree of v, where $order$ is the order of operation, $diff$ is the changes of the subtree size v, and v is the child of vertex u. Store all these values in a vector, and sorted according to $order$.

Let con_u be a vector containing the values of all operations in subtree u, and $lazy_u$ be the changes of the value $diff$ on con_u which is not changed directly. $lazy_u$ will only store the changes of $diff$ with odd indices. For even indices, the changes is $-lazy_u$.

During DFS, we do not access con_u directly for every vertices u. When constructing con_u, if there is only one v that has non-empty value of con_v, then we simply move con_v to con_u using vector swap, then add lazy to con_u. The content of con_u is only accessed every time we need to merge more than one con_v. During merging, we need to update the content of con_u with $lazy_u$.

Because con_u will only be accessed every time we wanted to merge and every time we wanted to answer operation 2, then con_u will be accessed not more than $K - 1$ times, so the DFS will run in $O(N + K^2)$. If $K = sqrt(Q)$, then complexity of the whole DFS is $O((N + Q)\ sqrt(Q))$.

Time Complexity: O((N+Q) sqrt(Q))