TOKI Regular Open Contest #10 Editorial

(English version is available on page 5.)

Penulis soal

Judul Soal		**Author**	**Editorialis**
Div 2A	Visi Tahun Baru	**prabowo**	**patrick**
Div 2B	Bolongan Tahun Baru	**hocky**	**patrick**
Div 1A	Sarang Lebah Tahun Baru	**patrick**	**patrick**
Div 1B	Matriks Tahun Baru	**phillohambali**	**prabowo**
Div 1C	Benteng Tahun Baru	**phillohambali**	**prabowo**
Div 1D	Katak Tahun Baru	**phillohambali**	**patrick**
Div 1E	Pohon Tahun Baru	**hocky**	**prabowo**
Div 1F	Bilangan Tahun Baru	**patrick**	**patrick**

Div 2A. Visi Tahun Baru

Setiap huruf dalam string S dan T bisa diubah menjadi huruf besar (atau huruf kecil), lalu dibandingkan secara langsung.
Mengubah sebuah huruf menjadi huruf kecil bisa dengan menggunakan `tolower` pada C, atau `.lower()` pada Python, ataupun dengan bitwise | `32` pada kode ASCII-nya.
Kode: https://ideone.com/zMAPs0
Kompleksitas waktu: O(|S|)

Div 2B. Bolongan Tahun Baru

Untuk mengerjakan soal ini, kita dapat membagi menjadi beberapa kasus.

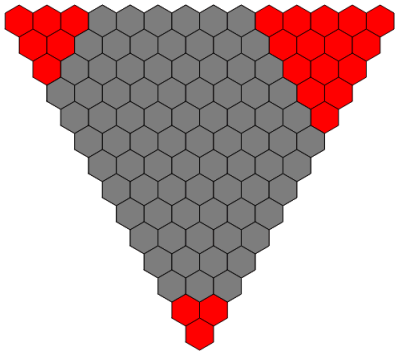
- Untuk N = 0, jawaban bernilai 1.
- Untuk N = 1, jawaban bernilai 0.
- Untuk nilai N > 1, terdapat observasi bahwa untuk memperkecil angka, kita harus meminimalkan jumlah digit. Sehingga, kita harus menggunakan digit sesedikit mungkin,

agar bilangan tersebut terkecil. Saat N genap, kita bisa mengulang digit 8 sebanyak N/2 kali. Jika nilai N ganjil, digit 4 dapat diletakan sebelum (N-1)/2 banyak digit 8. Ini dikarenakan tidak ada digit yang memiliki > 2 bolongan, dan 4 adalah tak nol terkecil dengan 1 bolongan. Maka dari itu ini adalah solusi optimal.

Kode: https://ideone.com/uv4YhP

Kompleksitas waktu: O(N)

Div 1A. Sarang Lebah Tahun Baru



Dari gambar, bisa dilihat bahwa banyak sel dalam sarang lebah (abu-abu) = (banyak sel dalam segitiga besar) - (banyak sel dalam tiga segitiga merah). Panjang sisi segitiga besar adalah f+a+b, sedangkan panjang sisi segitiga merah adalah b-1, d-1, dan f-1. Formula dari banyaknya sel dalam sebuah segitiga didapatkan dari formula bilangan segitiga (n(n+1)/2, untuk n panjang sisi).

Kode: https://ideone.com/FMmkX8

Kompleksitas waktu: O(1)

Div 1B. Matriks Tahun Baru

Operasi persegi dapat digunakan secara greedy, dari posisi (1, 1), (1, 2), …, (1, M-P+1), (2, 1), (2, 2), …, (N-P+1, M-P+1). Untuk setiap posisi (i, j), kita tambahkan nilai pada posisi itu hingga sebesar $A_{i,j}$. Apabila untuk setiap waktu, Anda menemukan posisi yang mempunyai nilai > $A_{i,j}$, maka jawabannya adalah `TIDAK`. Terlebih, apabila setelah melakukan operasi di atas, masih terdapat petak yang tidak sama dengan $A_{i,j}$, maka jawabannya juga `TIDAK`. Selain dari itu, jawabannya adalah `YA`.

Operasi persegi dapat dihitung dengan bantuan *prefix sum*. Prefix sum ini dapat dibangun di saat yang bersamaan dengan iterasi posisi-posisi yang disebutkan di atas.

Karena input/output bisa sangat besar, direkomendasikan untuk menggunakan metode masukan/keluaran yang cepat: sebagai contoh, menggunakan `scanf/printf` daripada `cin/cout` di C++.

Kode: https://ideone.com/XvTj8r
Kompleksitas waktu: O(NM)

Div 1C. Benteng Tahun Baru

Kita akan menyelesaikan permasalahan ini secara *offline* menggunakan struktur data *union-find*.

Pertama, kita inisialisasikan tiap posisi dengan "jawaban" yang bernilai sebesar Manhattan distance dari (1, 1). Selanjutnya, kita akan menjawab pertanyaan dari level prajurit yang paling kecil. Untuk setiap pasang posisi (i, j) dan tetangganya yang masing-masing mempunyai level tidak lebih dari prajurit sekarang, kita gabungkan kedua posisi tersebut dengan struktur data union-find kita, dan jawaban pada himpunan ini menjadi minimum dari kedua himpunan yang baru saja kita gabungkan.

Mengurutkan posisi berdasarkan level membutuhkan kompleksitas O(NM log(NM)), sedangkan mengurutkan Q prajurit berdasarkan level membutuhkan O(Q log (Q)).
Kode: https://ideone.com/eyLnsK
Kompleksitas waktu: O(NM log(NM) + Q log(Q))

Div 1D. Katak Tahun Baru

Perhatikan bahwa ketika K ≥ L atau K ≥ N-1, strategi katak A adalah melompat sejauh mungkin, dan katak B tidak akan bisa membalap katak A, sehingga katak A dijamin bisa menang.

Ketika K < L dan K < N-1, katak B bisa hanya bergerak 1 langkah. Dengan strategi ini, katak A selalu berada di depan katak B pada akhir langkahnya. Ini memungkinkan katak B untuk langsung lompat ke titik ke-N sebelum katak A memiliki kesempatan karena adanya batasan K. Maka dari itu, katak B dijamin bisa menang pada kasus ini.
Kode: https://ideone.com/B6JfGD
Kompleksitas waktu: O(1)

Div 1E. Pohon Tahun Baru

Kita akan melakukan DFS dari *root*, kemudian mencoba menyelesaikan problem ini dari *leaf*, kemudian ke *parent-parent*-nya.

Pada akhir DFS dari sebuah node u, kita akan mengembalikan nilai berikut pada akhir DFS-nya: Operasi-operasi yang dimulai dari leaf pada subtree u, dan path-nya melewati ancestor dari u. Kita sebut path demikian sebagai "pending path".

Apabila kita berada pada leaf node, kita cukup mengembalikan w_u buah pending path ke parent, karena kita membutuhkan setidaknya w_u operasi yang dimulai dari node ini.
Selain dari itu, misalkan banyaknya path yang dikembalikan children dari u adalah a_1, a_2, …, a_c, dengan c banyaknya children dari u. Berhubung hanya w_u path yang boleh melewati node u ini, kita akan mempertemukan ($a_1 + a_2 + \ldots + a_c - w_u$) pasang pending path pada node ini (dengan kata lain, node u ini akan menjadi LCA dari path yang kita gabungkan). Pending path yang belum dihubungkan akan diteruskan lagi ke parent, sehingga banyaknya pending path yang dikembalikan oleh DFS ini adalah ($a_1 + a_2 + \ldots + a_c - 2w_u$).

Berhubung pasangan path yang dihubungkan pada sebuah node tidak boleh berasal dari child yang sama, salah satu cara memasangnya adalah dengan aturan berikut: kita urutkan semua operasi berdasarkan children asal path ini, kemudian pasangkan path ke-i dengan path ke-(A-k+i), dengan $A = a_1 + a_2 + \ldots + a_c$, dan k adalah banyaknya pasangan yang akan dibentuk.

Mengembalikan path dari tiap node untuk diteruskan ke parent dapat digunakan teknik *small-to-large* dalam O(N log N) dan ini sudah cukup untuk mendapatkan Accepted. Dapat pula operasi-operasi di atas dioptimasi dengan linked list untuk mencapai kompleksitas O(N).
Kompleksitas waktu: O(N)

Div 1F. Bilangan Tahun Baru

Perhatikan bahwa semua operasi penjumlahan, perkalian, dan perpangkatan dapat mengubah sisa hasil bagi dari A dengan 1.000.000.007, sehingga operasi modulo dapat dilakukan di akhir saja (apabila diperlukan), dan kita cukup menyimpan sisa hasil bagi nilai A di sepanjang komputasi. Untuk seterusnya, kita anggap A berada pada sistem modulo $10^9 + 7$.

Perhatikan bahwa semua operasi (kecuali modulo) mengubah nilai A menjadi suatu nilai yang terbilang acak (bisa dianggap Pseudo Random Number Generator). Perhatikan pula tiap operasi bisa dibuat inversnya:

- Untuk operasi penjumlahan, A bisa dikurangi dengan 2019.
- Untuk operasi perkalian, A bisa dikalikan dengan $2020^{1000000005} \equiv 147029704$ (konsekuensi Fermat Little Theorem)
- Untuk operasi ketiga (perpangkatan), A bisa dipangkatkan dengan 143493321 karena $2021 * 143493321 \equiv 1 \, mod \, 1000000006$. 143493321 bisa ditemukan dengan cara extended euler, bruteforce, ataupun menggunakan teorema Euler, berhubung 2021 dan $10^9 + 6$ saling prima.

Dengan adanya operasi invers, kita bisa membuat solusi dengan teknik meet in the middle. Peluang dua himpunan mempunyai perpotongan sudah sangat besar apabila kedua himpunan

mempunyai anggota sebanyak O(sqrt(M)). Kompleksitas untuk menggunakan operasi perpangkatan atau inversnya adalah O(log M). Karena akan terdapat sekitar sqrt(M ln(M)) banyak pengoperasian, kompleksitasnya adalah O(sqrt(M log (M)) log(M)).
Kompleksitas waktu: O(sqrt(M log (M)) log(M))

TOKI Regular Open Contest #10 Editorial

Problem Authors

Problem Title		Author	Editorialist
Div 2A	New Year Vision	**prabowo**	**patrick**
Div 2B	New Year Holes	**hocky**	**patrick**
Div 1A	New Year Honeycomb	**patrick**	**patrick**
Div 1B	New Year Matrix	**phillohambali**	**prabowo**
Div 1C	New Year Castle	**phillohambali**	**prabowo**
Div 1D	New Year Frogs	**phillohambali**	**patrick**
Div 1E	New Year Number	**hocky**	**prabowo**
Div 1F	New Year Tree	**patrick**	**patrick**

Div 2A. New Year Vision

Every character in S and T can be converted into lowercase (or uppercase) letters, then we can simply compare both strings.
Converting a character to lowercase can be done using `tolower` on C, or `.lower()` on Python, or using the bitwise | `32` on its ASCII code.
Code: https://ideone.com/zMAPs0
Time Complexity: O(|S|)

Div 2B. New Year Holes

We can solve the problem by dividing it into several cases

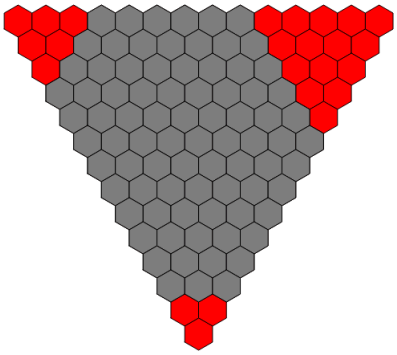
- For N = 0, the answer is 1
- For N = 1, the answer is 0
- For every N > 1, observe that, to minimize a number, we should minimize the length of the number itself. if N is even, we can repeat the digit 8 for N/2 times. If N is odd, we can

use digit 4 before (N-1)/2 digit 8. This is because no other digit has > 2 holes, and digit 4 is the smallest non-zero with 1 hole. Therefore this is the optimal solution.

Code: https://ideone.com/uv4YhP

Time Complexity: O(N)

Div 1A. New Year Honeycomb



From the figure, it can be seen that the number of cells in the honeycomb (gray) = (the number of cells in the big triangle) - (the number of cells in the three red triangles). The length of the big triangle is f+a+b, and consecutively b-1, d-1, f-1 for the red-sided triangles. (Mind that the all of the triangles are equilateral in shape). The formula of the number of cells in a triangle can be obtained from the triangular number.

Code: https://ideone.com/FMmkX8

Time Complexity: O(1)

Div 1B. New Year Matrix

The square-adding operation can be used greedily, from position (1, 1), (1, 2), …, (1, M-P+1), (2, 1), (2, 2), …, (N-P+1, M-P+1). For every position (i, j), we add value to that position until $A_{i,j}$. If for every moment, you found a position with value > $A_{i,j}$, then the answer is `TIDAK` (no). Moreover, after you have done the operations above, you can still find a position with value not equal to $A_{i,j}$, then the answer is `TIDAK` (no) too. Other than that, the answer is `YA` (yes).

The square-adding operation can be done with the help of prefix sum. This prefix sum can be built at the same time when iterating the positions mentioned above.

As input/output can reach huge size it is recommended to use fast input/output methods: for example, prefer to use `scanf/printf` instead of `cin/cout` in C++.

Code: https://ideone.com/XvTj8r

Time Complexity: O(NM)

Div 1C. New Year Castle

We will solve this problem offline using the union-find data structure.

First, we initialize every position with “answer” equals to Manhattan distance from (1, 1). Next , we will answer the questions starting from soldier with the lowest level. For every position pair (i, j) and its neighbors with level not more than current soldier, we will merge those two positions with the union-find data structure, and the answer for this set became the minimum of the two sets we just joined.

Sorting positions according to level needs O(NM log(NM)) complexity, while sorting Q soldiers according to their level needs O(Q log (Q)).
Code: https://ideone.com/eyLnsK
Time Complexity: O(NM log(NM) + Q log(Q))

Div 1D. New Year Frogs

Notice that if K ≥ L or K ≥ N-1, the strategy of frog A is to jump as far as possible, and frog B will not be able to race frog A, so frog A can be guaranteed to win.

When K < L and K < N-1, frog B can move 1 unit. Using this strategy, frog A will always be in from of frog B at the end of their turn. This made it possible for frog B to immediately jump to point N before frog A had the chance because of the K constraint. Therefore, frog B can be guaranteed to win in this case.
Code: https://ideone.com/B6JfGD
Time Complexity: O(1)

Div 1E. New Year Tree

We will run DFS from root, then try to solve this problem from leaves, dan proceed to their parents.

At the end of the DFS from a node u, we will return this value: The operations that start from the leaves of the subtree u, and the paths will pass through the ancestor of u. We call this path the “pending path”.

If we are already in the leaf node, we simply returns w_u pending paths to its parent, since this node needs at least w_u operation that started from this node.
Other than that, let the number of paths returned by the children of u be a_1, a_2, …, a_c, with c the number of children of u. Since only w_u paths may pass through this node u, we will join (a_1 + a_2

+ … + a_c - w_u) pair of pending paths on this node (in other words, node u will be the LCA of the path that we joined). Pending paths that are not joined will be carried on to its parent, hence the number of pending path returned by this DFS is (a_1 + a_2 + … + a_c - 2w_u).

Since the joined path pair must not come from the same child, one way to join them is using the following rule: sort the operations according to their children they came from, then we join the i-th path with the (A-k+i)-th path, where A = a_1 + a_2 + … + a_c, and k is the number of pair that will be formed.

Returning paths from every node to its parent can be done using the small-to-large technique in O(N log N) and this is enough for Accepted. The operations above can be optimized using to achieve the complexity O(N).
Time Complexity: O(N)

Div 1F. New Year Number

Notice that all the addition, multiplication, and exponentiation operation can change the remainder of A with 1,000,000,007, therefore the modulo operation can be done only at the end (if needed), and we simply store the remainder of A along the computation. For the rest, we will assume A is under the system modulo 10^9 + 7.

Notice that all operations (except modulo) can change the value A to a value that can be regarded as random (also can be regarded as a Pseudo Random Number Generator). Notice that all operations have their inverses too:

- For addition operation, A can be subtracted by 2019.
- For multiplication operation, A can be multiplied with $2020^{1000000005} \equiv 147029704$ (consequence of Fermat Little Theorem)
- For exponentiation operation, A can be raised to the power of 143493321 because $2021 * 143493321 \equiv 1 \, mod \, 1000000006$. 143493321 can be obtained using extended euler, bruteforce, or Euler's theorem, since 2021 and 10^9 + 6 are coprime.

With the existence of inverse operations, we can solve this using meet in the middle. The probability that two sets have intersection is already huge when each set has O(sqrt(M)) elements. The complexity to use the exponentiation operation or its inverse is O(log M). Since there will be around sqrt(M ln(M)) operation, the complexity is O(sqrt(M log (M)) log(M)).
Time Complexity: O(sqrt(M log M) log(M))