Nizhny Novgorod SU Contest brief editorial

Maxim Akhmedov
Moscow State University, Yandex

January 24th, 2017

1 Problem A. Prevent a Galactic War!

This is a task for implementing any of the different iterative approaches.

First of all, using the original values of c_{ij} and y_i, find out the amount of product of all remaining kinds that is needed for creating a unit of product i.

For example, start by letting $\breve{x}_i = \breve{y}_i$. This is not a correct solution yet, because creating $\breve{x}_i$ product of the i-th type would require to also provide some amount of i-th product for creating the remaining products. By $\breve{x}_i$ calculate the amount of the rest products $\breve{c_i j}$ that is needed, and replace all $\breve{x}_i$ with $\breve{x'_i} = \breve{y}_i + \sum_{j \neq i} \breve{c_i j}$.

Continue doing that until the vector $\breve{x}_i$ will stop changing much. One may prove that this solution works in $O(n^2 \log PREC^{-1})$, where $PREC$ is the precision one wants to achieve.

2 Problem B. Forcefield

Simulate the process using an `std::set` or `TreeSet`.

3 Problem C. Missing Part

For each letter x among ``abcde'' and letter $y$ among ``ABCDE'' calculate the vector of numbers $M_{xy}[0], M_{xy}[1], \ldots, M_{xy}[n-1]$ that are defined as: $M_{xy}[i]$ is the number of positions such that the first string contains an x character and the second string contains an y character if we shift the second string circularly by i positions.

For a fixed x and y such a vector may be found via Fast Fourier Transform (this is called a circular convolution problem). Make such convolution for all 25 pairs of x and y, then iterate over all $5! = 120$ matchings $p = (p_{\mathtt{a}}, p_{\mathtt{b}}, p_{\mathtt{c}}, p_{\mathtt{d}}, p_{\mathtt{e}})$ (x matches to p_x, $p_x \in$ "ABCDE") and find such p and i that the sum $M_{\mathtt{a}p_{\mathtt{a}}}[i] + \ldots + M_{\mathtt{e}p_{\mathtt{e}}}[i]$ is the largest possible.

4 Problem D. Handling a Spaceship

First, ask the speed for all g_i set to 1, let the resulting vector be $\vec{v_0}$. Then, perform n queries, on j-th query set all g_i to 1 except the g_j that is set to 2. Let the result of such query be $\vec{v_j}$.

After that, the $\vec{v_j} - \vec{v_0}$ is equal to $(K_{j2} - K_{j1})\vec{X_j}$. Consider the $\vec{b_j} = \vec{v_j} - \vec{v_0}$, the set of vectors $\vec{b_1}, \vec{b_2}, \ldots, \vec{b_n}$ form the basis of a set $\vec{X_1}, \vec{X_2}, \ldots, \vec{X_n}$.

Suppose we have an absolutely precise real numbers. We will spend the remaining 19 queries while making the *parallel binary search*. Suppose that before i-th step we know that the zero

selector for j-th direction is between l_j and r_j. Consider the $m_j = \left\lfloor \frac{l_j + r_j}{2} \right\rfloor$ state of the j-th selector, let $g_j = m_j$ for all j. Make a query for such selectors, express the resulting speed vector $\vec{v}$ as a linear combination of basis vectors b_j using the Gauss algorithm. For those j that we have positive coefficient we perform $r_j = m_j$, for those that we have negative coefficient we perform $l_j = m_j$, for those we have the zero coefficient we already found the answer.

This will require about $\log n$ more operations. Now in order to deal with losing precision during the Gauss algorithm, reduce each of the basis vectors by GCD of their components, this will make all the coefficients in an expression of $\vec{v}$ as a linear combination of basis vectors be integral. After that perform all the computations modulo $10^9 + 7$. Now the "positive" numbers will be those that are closer to 0 rather than to $10^9 + 7$.

5 Problem E. Cryptographic Argument

This problem requires a careful look at how the resulting sequence looks like. Suppose that the sequence is zero-based.

- $a_{2j} + a_{2j+1} = a_{2j} \oplus a_{2j+1} = 2^k - 1$ since after the first step the numbers having complementary binary representation are paired together;
- Since for the operation priority depends on the pairity of l, it results in that the operations having the largest priority are always performed on numbers a_{2j} and a_{2j+1} for some j. So, it's easy to account their effect to the whole sum/xor by calculating the number p of such pairs inside range $[l, r]$ and adding $p \cdot (2^k - 1)$ / xoring with $(p \bmod 2) \cdot (2^k - 1)$;
- So, the answer to the query consists of at most two unpaired numbers near l and r and all the pairs in the middle that can be dealt in $O(1)$. So all we have to do is to be able to find the numbers given their indices in the final sequence;
- In order to do that use the first observation and the recursive nature of the sequence and find a way to reduce the $findNumber(i, k)$ to $findNumber(i', k - 1)$ for some i'.

6 Problem F. The Jedi Killer

- If all three points belong to the same line, they should all appear either on main ray or on the guard;
- The other possibilities are: either one point will belong to the main ray and two to the guard or vice versa;
- If one of the points is on the ray and two on the guard: try each of the three points to be the ray one. Let's say C is on the ray and AB should be on guard. Let X be a projection of C onto AB. X has to be the base point of a lightsaber, so just check that $|XA|$ and $|XB|$ are no larger than L_g and XC is no larger than L_m;
- If one of the points is on the guard and two are on the ray: do almost the same thing. Let C be on guard and A and B be on main ray. Let X be a projection of C onto AB. It should be true that $|XC| \le L_g$ and $|XA|, |XB| \le L_m$, but there is also an important extra condition: A and B should not lie by the different sides from X since that restricts us from choosing the single direction for a main ray.

7 Problem G. Youngling Tournament

Let's try to find the number of winners for a current situation fast enough. Suppose younglings follow in order of increasing f_i. We are willing to find all such i that $f_i \geq f_1 + f_2 + \ldots + f_{i-1}$.

It is true that f_1 and f_2 are always winners. Create a variable $S = f_1 + f_2$, it will denote the sum on the current prefix we are considering. The next winner should have sum at least S: find the first $f_i \geq S$, and check if it is the sum by calculating the $f_1 + f_2 + \ldots + f_{i-1}$ and comparing with f_i. If he is a winner, increase the number of winners. In both cases, replace S with the sum $f_1 + f_2 + \ldots + f_i$, this is the minimum sum on the prefix for the next winner. Note that the new value for S is at least twice as large as the previous one, because both $f_1 + f_2 + \ldots + f_{i-1}$ and f_i are no smaller than S. Hence, we will have at most $\log_2 10^{12}$ steps. In order to compute the sums fast enough, compress all numbers (already existing and the new one that appear) to the range of $n + m$, build a Fenwick tree and using it you will be able to calculate the sums in $O(\log(n + m))$.

So, the overall complexity of one query is $O(\log MAX \cdot \log(n + m))$ where MAX is 10^{12} in our case.

8 Problem H. Garland Checking

Everybody who submitted this problem used a Linking-Cutting trees data structure. Although it may be solved without any heavy data structures.

Keep a color of each vertex that defines the connected component it belongs to. When linking two trees together, recolor the smaller of them into the color of larger of them. If you only had links, it would result in $O(n \log n)$ complexity because it is similar to the rank heuristic in DSU.

Let's perform cuts in a similar manner: from two formed pieces, color the smaller one into a new color. In order for this to work in $O(\min(s_1, s_2))$ where s_1 and s_2 are the sizes of two components, run a parallel DFS from each of the endpoints of the edge being cut, visiting one vertex from each side of the cut at a single step.

This works still in $O(n \log n)$ because the worst case for this solution would be when all joins precede all the cuts, and in this case all joins will work in $O(n \log n)$ due to the rank heuristics argument, and all cuts will also work in $O(n \log n)$ due to the same argument if we reverse the time.

9 Problem I. Equipment Assembling

To be added later.