Problem A. Archaeology

Author and problem developer: Mikhail Ivanov

Each answer to the query gives us a random half-plane where the artifact lies, with radar on the border of this half-plane. The intersection of these half-planes is a convex polygon. So the idea is to make this polygon smaller and smaller until it is so small that we by accident put the radar in the same point where the artifact is.

All jury solutions did it by somehow maintaining the convex hull and putting the radar in its "center". However, they differed in what was considered the center: for example, the center of the bounding box can be used. For that you need four points: top, bottom, leftmost and rightmost. Each of them can be found with a linear algorithm which, instead of maintaining all vertices of the convex hull, only finds the extreme point in some direction. You can also choose the center of mass of the vertices of the polygon, it also works. There is a small probability that the polygon becomes of very strange shape, and you become making a lot of queries without actually making the polygon smaller. But if you implement everything carefully and add some tweaks for preventing this issue (e.g. if you notice that the solution constantly puts the radar in the same point, then you can randomly put it in adjacent points instead), you should be good to go.

Problem B. Glass Stepping Stones

Note first that Alice only needs to delete the leftmost R and the rightmost L, since deleting another R or L will leave her with fewer possible LR, and Bob with more possible RL. Similarly, Bob removes only the rightmost R and the leftmost L.

Let's learn how to solve the following problem: "Alice wants to win, Bob wants to draw the game", the initial solution follows from it. Let a contain the letters L and b contain the letters R. Let's draw a rectangle $(a+1)$ by $(b+1)$, and the path from cell $(a, 0)$ to $(0, b)$ in it: its initial cell of path is $(a, 0)$, then we go along the string. If the current symbol is L, the next cell will be the one next to it horizontally, otherwise vertically. Let's call the cells of this path shaded.

You can see that the game is equivalent to such a rectangle game: "Alice stands at point $(0, 0)$, Bob stands at point (a, b). They take turns moving horizontally or vertically to the square in the direction of the other player. 1 loses as soon as one of the players' coordinates matches. 1 wins if it falls into a shaded cell."

Let's call the horizontal segment $[u, v]$ **beautiful** if the following is true:"Let's consider such deterministic strategy: " Alice walks horizontally until his coordinate is less than u, then begins to walk vertically. Bob goes horizontally until his coordinate is less than u, then begins to go vertically. Then Alice wins in this game" Now let's define the **successful** segments as follows:

1) If the segment is beautiful, it cannot be successful.

2)Otherwise, the segment is successful if $v \leq u + 1$, or the segment $[u + 1, v - 1]$ is successful, or the segment $[u + 1, v]$ is successful.

Note that this definition follows by induction that if Alice chooses the horizontal coordinate u in the game, and Bob chooses v, and the segment $[u, v]$ is successful, then Bob wins: he will convert successful segments into successful ones on his turn. That is, if the segment $[0, a]$ is successful (horizontally), then Bob wins. Similarly, we define vertically successful segments, and if $[0, b]$ is vertically successful, then Bob also wins. It can be proved by induction that if none of these conditions are met, then Alice wins.

A segment $[0, a]$ is successful if and only if it can lead to a segment of the form $[t, t]$ or $[t, t + 1]$, moving from $[x, y]$ to $[x + 1, y]$ or to $[x + 1, y]$ so that we don't meet a beautiful segment on the way. We will store the segments achievable in this way from $[0, a]$. Note that if $[x, y]$ is achievable, then $[x, y - 1]$ is achievable, that is, it is enough for us to store the maximum possible y for each x, and this can be recalculated in $O(1)$, that is, to check the success of $[0, a]$ in linear time.

In terms of the game, this corresponds to the following statement: "If Bob wins, then he wins by one of these strategies: he fixes the direction and coordinate along it (say, horizontal and the coordinate x). Now, if Alice's horizontal coordinate is no more than x, then he always goes down. Otherwise Bob will repeat Alice's previous move." This statement can also be verified in linear time.

Total operating time is $O(n)$.

Problem C. Elegia's Mind

The problem is very difficult, but becomes much easier if you have some experience working with tensor decompositions (not in the "machine learning" sense, but in the "fast matrx multiplication" sense). However, very little background knowledge and experience is needed to understand the editorial.

The problem is solved by a Karatsuba-like algorithm. If the color of the face is c, then the color of the opposite face is $5 - c$. So, each rotation of the cube can be written down as the colors of, let's say, front, top and left faces. The 24 possible rotations are the following: $0 \times 1 \times 2$, $0 \times 4 \times 3$, $0 \times 2 \times 4$, $0 \times 3 \times 1$, $5 \times 1 \times 3$, $5 \times 4 \times 2$, $5 \times 2 \times 1$, $5 \times 3 \times 4$, $1 \times 0 \times 3$, $1 \times 5 \times 2$, $1 \times 2 \times 0$, $1 \times 3 \times 5$, $4 \times 0 \times 2$, $4 \times 5 \times 3$, $4 \times 2 \times 5$, $4 \times 3 \times 0$, $2 \times 0 \times 1$, $2 \times 5 \times 4$, $2 \times 1 \times 5$, $2 \times 4 \times 0$, $3 \times 0 \times 4$, $3 \times 5 \times 1$, $3 \times 1 \times 0$, $3 \times 4 \times 5$.

If the string in the input is $a[0] \ldots \ldots a[6^n - 1]$, then the answer for $n = 1$ is $a[0] \times a[1] \times a[2] + a[0] \times a[4] \times a[3] + a[0] \times a[2] \times a[4] + \ldots + a[4] \times a[3] \times a[0] + \ldots$. The answer for $n = 2$ is $a[00] \times a[11] \times a[22] + a[00] \times a[14] \times a[23] + \ldots$. Let us ignore the fact the fact that the left, the middle and the last arrays are the same and solve the more general problem: $a[00] \times b[11] \times c[22] + a[00] \times b[14] \times c[23] + \ldots$.

In Karatsuba's algorithm we say that $a[0] \times b[0] \times c[0] + a[0] \times b[1] \times c[1] + a[1] \times b[0] \times c[1] + a[1] \times b[1] \times c[2] = (a[0]+a[1]) \times ($
Here, we can do something similar.

If we somehow manage to represent $a[0] \times b[1] \times c[2] + a[0] \times b[4] \times c[3] + a[0] \times b[2] \times c[4] + \ldots + a[4] \times b[3] \times c[0] + \ldots$ as a sum of k such products, where each multiplier is a linear function of a/b/c.

Problem D. Preparation for the Exam

Author and problem developer: Mikhail Ivanov

It can be noticed that if Misha (me!) has two parts of exam for which he prepared p_i and p_j questions, and $p_i < q_i$ and $p_i < p_j - 1$, then he can increase the probability of passing the exam by taking $p'_i = p_i + 1$ and $p'_j = p_j - 1$. In other words, all p_i's should be almost equal, except for the cases when q_i is smaller than the number of questions that would be equal to others.

Therefore, the following algorithm works in $\mathcal{O}(n \log n)$. Firstly, Misha should sort all q_i's. He maintains the number of remaining hours in variable t and go through the array q from the beginning to the end and find p's. If $q_i \le \left\lceil \frac{t}{n-i+1} \right\rceil$, Misha takes $p_i = q_i$ and learns all questions, otherwise he takes $p_i = \left\lceil \frac{t}{n-i+1} \right\rceil$ and learns almost equal number of questions in each of the remaining parts of the exam. Then $t := t - p_i$, and Misha switches to the next part.

Problem E. Fractal Maze

Let us first consider a recursive solution in $\mathcal{O}(answer)$. On a large $2^n \times 2^n$ square board, how to get from A to B?

- If $A = B$, we are done.
- Otherwise, consider the four smaller $2^{n-1} \times 2^{n-1}$ squares the board is made of.
- Let us call them S_0, S_1, S_2, and S_3, in clockwise direction.
- Consider S_A and S_B: the smaller squares where A and B are.
- If $S_A = S_B$, solve the subproblem inside that square, and we are done.

- Otherwise, consider the big picture: looking at the four smaller squares, how do we get from S_A to S_B? The route is either clockwise or counter-clockwise, and goes through 2, 3, or 4 smaller squares.
- We know the adjacent cells of the smaller squares. Let us break the route into subproblems and single steps.
- For example, let A be in square S_2, B be in S_0, and the path be $S_2 \to S_3 \to S_0$.
- Let the cell of square S_i adjacent to square S_j be $C_{i,j}^{(n)}$.
- Then we go like this: $A \Rightarrow C_{2,3}^{(n)} \to C_{3,2}^{(n)} \Rightarrow C_{3,0}^{(n)} \to C_{0,3}^{(n)} \Rightarrow B$.
- Now, $C_{2,3}^{(n)} \to C_{3,2}^{(n)}$ and $C_{3,0}^{(n)} \to C_{0,3}^{(n)}$ are single steps.
- Whereas $A \Rightarrow C_{2,3}^{(n)}, \to C_{3,2}^{(n)} \Rightarrow C_{3,0}^{(n)}$, and $C_{0,3}^{(n)} \Rightarrow B$ are subproblems. Solve them recursively.

Implementation is a bit tedious for placing $C_{i,j}^{(n)}$.

How do we speed up this solution? By using memoization! Why would it help?

- Let us memoize only when both endpoints are not A or B but some intermediate cells $C_{i,j}^{(n)}$.
- Rough estimate: there are 30 sizes, 4 smaller squares for each size, and 2 neighbors for each smaller square.
- So, there are at most $30 \times 4 \times 2 = 240$ such cells.
- Thus there are no more than $240 \cdot 240 = 57\,600$ pairs of cells.
- In fact, only a few thousand pairs will be ever needed.

If we have already memoized all pairs, all we have to do is, for each size, to compute the distances from A and B to some intermediate cell in the path.

The solution takes $\mathcal{O}(\text{Poly}(n))$ per question, which is fine since $n \leq 30$.

Problem F. Interactive Primality

Author and problem developer: Mikhail Ivanov

Let us call the primes from 2 to 53 *small primes*. The product of all small primes is at least 10^{18}. If you find the residue of x modulo all small primes, you can apply the Chinese Remainder Theorem and find x.

To do that, we will take kinda random y's and ask about them. (We will always take $y \geqslant 53$, so if $x + y$ is divisible by one of the small primes, then it is definitely composite.) All `Composite` answers will be ignored; we will only look at `Prime` results. That is because they give the most information. Namely, if $x + y$ is prime, then x cannot be congruent to y modulo any small prime.

Let us for each small prime store the set of all residues that x could have modulo this prime; initially, all residues are possible. Each time a prime y is noticed, $-y$ is eliminated from all these sets (because if x had had the residue $-y$, then we couldn't have received the `Prime` outcome this turn).

How to choose y then? After the first `Prime` result, we will not choose y completely at random. Instead, for each small prime, if in its set there is only one residue x left, we take any y such that $-y$ *is not* congruent to x modulo this prime. For each small prime, if in its set S there are at least two residues left, we take any y such that $-y$ *is* congruent to one of the residues in this set. All these congruences are combined via CRT, and if it results in a too big number, we try again. That way, each time we receive an outcome of `Prime`, each set of size at least two loses one element.

We need to get 52 `Prime` results to make all sets of size 1 and find the value of x. The probability that a random number is prime is something like $\frac{1}{\log 10^{18}} \approx \frac{1}{41}$, so the number of queries can be assessed as $52 \log 10^{18} \approx 2155$. However, the actual number of queries is much smaller: for example, after the first `Prime` outcome, we will only try odd numbers, effectively doubling the probability of prime number; after the second `Prime` outcome, we will try only numbers not divisible by three, effectively multiplying the probability by 1.5, etc. The actual expected number of the queries is hard to estimate analytically, but in the author's solution, the probability that ten secret x's take less than 8750 queries to guess is greater than 99.9%.

Problem G. Jump the Frog

Author and problem developer: Mikhail Ivanov

For each snippet s, compose a $k \times k$ matrix m. m_{ij} should equal the number of ways to get from s_i to $s_{|s|+j}$ (all indices are 1-based), assuming that you put an additional lily pad in the point $|s| + j$. (Note that i can also be greater than $|s|$, in which case $m_{ij} = 0$ unless $i = |s| + j$.)

Then a single character "`~`" corresponds to a matrix with $k - 1$ ones below the main diagonal; whereas a single character "`0`" corresponds to a matrix with both k ones in the top row and $k-1$ ones below the main diagonal. Concatenation means multiplying these matrices. In total this solution works in $\mathcal{O}((n + a)k^{\alpha})$, and $\mathcal{O}((n + a)k^3)$ is enough to pass all tests.

Problem H. Slot Machine

At any given time, the set of possible secret numbers forms a segment. Moreover, the current number on the machine can be considered one of the ends of the segment. Thus, there is a simple solution for a cube of the input size (i.e. 10^{3k}), namely the dynamic programming, where the state is a segment and which of its ends is on the machine screen ($dp[u][v][l/r]$). Transitions: we go through the number in the segment that we will be asking about, and move on to the sub-sections.

Let's apply 2 optimizations to this solution.

1) We fix which end of the segment is written on the monitor (say, the left one), and what kind of end it is (say, the number u). Then $dp[u][v][l]$ is monotonic over v, which means that only those places where its value changes can be maintained. On the other hand, it changes a logarithmic number of times (because, for example, you can do a regular binary search), but in fact all dp does not exceed 23. Thus, by storing the change locations, we get a solution for $10^{2k}a$, where a is the answer to the problem, $a \leq 23$.

2) In the transition, instead of the usual iteration of all points in the segment, we will iterate over the position mask that we are changing. Then the set of positions of this mask in the segment also forms a segment, and we just need a minimum of the maximum of 2 speakers on the left and right in it, which can be found, say, by a binary search. This allows you to solve the problem in $k10^{2k}2^k$.

By carefully combining the ideas, we will get a solution for $20^k a$ (without unnecessary logarithms). Well-written solutions with an extra logarithm can also pass.

Problem I. Product

The naive solution is as follows: write the dynamics of $dp[t][x]$, the sum of the weights of the beginnings of the substructures with the tth element equal to x. It works for $O(nmk)$.

Consider the polynomial $P_i(x) = \sum dp[t][i]x^i$. Then the transition from t to $t + 1$ either multiplies it by $\frac{1}{1-x}$, or takes the derivative and multiplies it by $\frac{x}{1-x}$. That is, a block of k operations translates P into $P' \frac{x}{(1-x)^k}$. We want to do this operation quickly.

Let's take $x = Q^{-1}(y)$ for a certain Q (Q^{-1} means the function, such that $Q(Q^{-1}(x)) = x$). Then the formula on the right is $P'(Q(y))\frac{Q(y)}{(1-Q(y))^k}$. But $P(Q(y))' = P'(Q(y))Q'(y)$. Taking Q such that $yQ'(y) = \frac{Q(y)}{(1-Q(y))^k}$, the operation will turn into a simple multiplication of coefficient i by i. Such a Q can be found by solving the differential equation using the standard method. In total, we will find Q^{-1}, the composition, in a simple way we will apply the operation m times and return back. People learned how to make a composition in $O(n \log^2 n)$ https://codeforces.com/blog/entry/127674 https://codeforces.com/blog/entry/128204. Total asimptotics is $O(n \log^2 n + n \log m)$.

Problem K. Vortex

Let's start with the problem statement for the rooted tree. We can choose the centroid of the tree as the root. In the case where there are two centroids, we can split the edge between them into two, solve the problem for the resulting tree, and then remove the new vertex from the answer.

So, we can solve the problem for the rooted tree. Suppose we have a deterministic perfect hash for the subtrees. Then we can hash all the subtrees and take a depth-first or breadth-first traversal of the tree as the answer. It is important to note that simply sorting the vertices by hash is not sufficient because we will lose information about which vertices are connected to which.

Thus, it only remains to construct a deterministic perfect hash for the subtrees. This can be done inductively in increasing order of subtree depth. Let's assume we have constructed hashes for all subtrees of height less than h; then we can determine the hashes for the subtrees of height exactly h by sorting sets of hashes of the vertices of depth 1 (i.e. children of the root) and mapping them deterministically to natural numbers (for example, the smallest unused ones).

Problem L. Random Sum

We are asked to calculate the product of m polynomials of the form $(1 + ax^k)$ modulo $x^p - 1$. We want to group us multiples somehow in such a way, that in any group after some substitution of type $y = x^t$ total degree of polynomimal will be small. Then we will multiply things in one group using divide-and conquer, and group results in usual way with FFT. To show simplest usage of this idea, we may multiply for any t all monomials of type $(1 + ax^t)$ using divide-and-conquer, and then multiply results in usual way and obtain running time $m \log^2 m + p^2 \log p$.

Let's take a certain number B. For any residue k, there exists a $t < \frac{p}{B}$ such that either $kt \mod p < B$ or $-kt \mod p < B$. That is, if we make a substitution $x = y^t$, the degree will be small. Let us split all multiples by $\frac{p}{B}$ groups by this t. Within one group, we make a substitution $x = y^k$, then all degrees became less then B. So running time of multiplications inside groups is $O(Bm \log^2 m)$. Then we make a reverse substitution and multiply products of groups in $O(\frac{p^2}{B} \log p)$ time. Taking appropriate B, we obtain $O(p\sqrt{m}(\log(p + m))^{\frac{3}{2}})$ running time. B is about 50 in my code.