Problem A. Alice and Money Tree

Attempts: 53.

Accepted: 4.

First accepted: 01:18.

The path of the token can be viewed as a simple path with several ramifications via vertices containing 2 coins. Let's try all vertices v for the role of the start of this path.

For each v define $L(v)$ to be the subtree of v (that is, all vertices reachable from v by downward edges), and $H(v) = (V \setminus L(v)) \cup \{v\}$. Let $dp_1(v)$ denote the maximum answer if the path starts at v and lies entirely within $L(v)$. Let $dp_2(v)$ denote the maximum ramification length of a branch starting from v and lying entirely within $L(v)$. For example, if $a_v = 1$ (v contains 1 coin), then $dp_2(v) = 0$.

These dp values can be easily computed by a `dfs`. For example, if $a_v = 2$, then the optimal path ($dp_1(v)$) may first descend to one subtree to get a ramification, then return back to v, and then descend into another subtree taking dp_1. That is, we can obtain $2dp_2(u_i) + 1 + dp_1(u_j)$ for any two distinct children u_i, u_j of v. We can store two maximum values of dp_1 and dp_2 over children or, alternatively, assume that $i < j$, store a single maximum, and then reverse the order of children and do the same thing.

Now we need to consider paths that leave the subtree of v at some moment of time. Introduce $dp_3(v)$ to be the length of the optimal path starting at v and lying in $H(v)$ (except for a single possible ramification in $L(v)$ in case $a_v = 2$). We will also need $dp'_3(v)$ — the length of the optimal path starting at v which lies within $H(v)$ and doesn't have a ramification at v. In its turn $dp_4(v)$ will be equal to the maximum ramification length from v in $H(v)$.

Assume that p has children $v_1, \ldots, v_k$, and $a_{v_i} = 1$. Then $dp_4(v_i) = 0$. If $a_p = 1$, then $dp_3(v_i) = 1 + dp_3(p)$ or $dp_3(v_i) = \max\limits_{i \neq j}\{2 + dp_1(v_j)\}$. If $a_p = 2$, we need to consider a possible ramification at p. It either descends to a different child and then ascends upwards ($\max\limits_{i \neq j}\{2dp_2(v_j) + 2\} + dp_3(p)$), or descends to a different child and then descends to another one ($\max\limits_{i \neq j \neq k, k \neq i}\{2dp_2(v_j) + 3 + dp_1(v_k)\}$), or ascends upwards to get a ramification and then descends to a different child ($2dp_4(p) + \max\limits_{j \neq i}\{dp_1(v_j)\}$).

In case $a_{v_i} = 2$ we have to consider a possible ramification starting at v_i.

Long story short, all the necessary values can be computed one via the others in total running time $O(N)$ for each testcase.

Problem B. Warehouses for SberMarket

Attempts: 221.

Accepted: 24.

First accepted: 00:46.

Consider the case when one of the sides is exactly 1. Then, there is no solution, since there is no option to fill the cell with exactly one neighbor.

Consider the case where the size of the warehouse is 3×3. Let's show that there is no solution either. Corner cells shall be filled with 1, 2, 3, and 4. respectively. The edge sectors can be filled only by 5 or 6's, depending on the orientation. Then we can check that any of the 6 existing options do not fit the center cell.

Let's build a constructive solution for all other cases.

If we have at least one side of even length:

Without loss of generality, we will assume that we have an even width.

Consider our warehouse as a union of warehouses of dimensions $w \times 2$ (or as a union of warehouses of dimensions $2 \times w$).

To solve the $w \times 2$ warehouse, set each of its non-corner elements to 5, and put the elements 1, 2, 3, and 4 at the corners.

To solve $2 \times w$ warehouse, set each of its non-corner elements to 6, and put the elements 1, 2, 3, and 4 at the corners.

For example, you can see the solution for the warehouse 7×2:

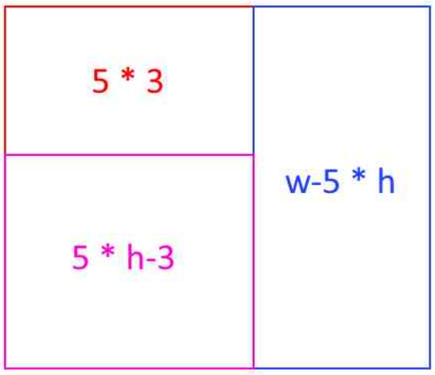
```
3 5 5 5 5 5 4
2 5 5 5 5 5 1
```

In the case when the sides are of odd length, one side is is strictly greater than 3 and other is 3 or equal, we will consider the solution for the matrix 5×3:

```
3 4 3 5 4
6 2 5 4 6
2 5 1 2 1
```

and the respective solution for the matrix 3×5.

Now we can represent any of the considered matrices as a union of three matrices of sizes 5×3, $w - 5 \times h$ and $5 \times h - 3$ (or 3×5, $w - 3 \times h$ and $3 \times h - 5$), where one or both even-sided matrices may be empty.



The resulting time complexity is $O(w \cdot h)$.

Problem C. Revaccination

Attempts: 31.

Accepted: 8.

First accepted: 00:45.

Suppose $f_i(x) = a_i x + b_i$. Then we need to maximize $f_i(f_j(x))$ over all $i \neq j$. When i is fixed we are to minizime or maximize $f_j(x)$, since f_i is linear.

Consider a random partition of all the vaccines into two sets I and J. Then the optimal indices i_{opt} and j_{opt} (for a given x) will be in different sets with probability $\frac{1}{2}$. Let's build the convex hulls over (a_i, b_i) in both sets I and J separately. When answering a query x, find the maximum M and minimum m value of $f_j(x)$ over all $j \in J$. After this, find the maximum value of $f_i(M)$ and $f_i(m)$ over all $i \in I$. Do the same when I and J are swapped.

Repeating this process 30 times suffices to get the optimal (i, j) in different sets, that is, to find the maximum.

Alternatively, we can split all vaccines into sets I_k and J_k based on the k-th bit in the binary representation of i and j.

Problem D. Yandexatellite

Attempts: 36.

Accepted: 23.

First accepted: 00:56.

Since the orbit is a paraboloid, there is symmetry, that is, that if we know at least one point with integer coordinates, then there are some good vectors (dx, dy, dz) such as adding then we will find more shooting points (i.e. we can find the basis).

This process can be repeated infinitely as long as the values of x, y, and z are not negative. Note that $(-200 \leq dx, dy, dz \leq 200)$, therefore, it is not necessary to use the Lenstra-Lenstra-Lovas algorithm, it is enough to enumerate the possible vectors.

Let us find non-collinear good vectors of the **minimum** length by brute force. We will assume that dx is not negative (otherwise the direction of the vector can be reversed).

To do that, we will use the method of undefined coefficients. The coefficients for x, y, z and the constant should not change after substitution in the equation $(x = x + dx)$, $(y = y + dy)$ and $(z = z + dz)$. The non-linear parts: $(x \cdot y)$, $(y \cdot z)$ and $(y \cdot z)$ are eliminated using the undefined coefficient method.

We will assume that the dx and dz in the vector are non-negative. In this case, you can always make a transformation by replacing variables with each other, or with those with opposite signs; then the value of dy will not be positive.

As a result, it turns out that we have a solution (X, Y, Z), where either $(0 \leq X \leq dx - 1)$, or $(0 \leq Z \leq dz - 1)$ is held. Let us fix the value of X by iteration. It follows that $(Y^2 \leq n)$. We will iterate over the value of Y from 0 to $sqrt(n)$. Since $(dy \leq 0)$, then by enumerating from the value of 0 we will maximize the possible value of Z. The value of Z can be found using a binary search.

The first point found by this algorithm becomes the reference point. Under the indicated restrictions on the coefficients and n, one can make sure that it is enough to operate with only one displacement vector. With respect to the pivot point, in a loop, we go through all the points, iterating by shifting by the vector (dx, dy, dz).

If the pivot point was not found in the previous step, then we need to fix the Z value at the beginning of the algorithm, iterate over the Y value from 0 to $sqrt(n)$ and search for the X value with a binary search.

The resulting asymptotics in time is O $(d * sqrt(n) * log(n) + d^3)$, where d is the upper bound on the coefficients in the bias vector. In our problem, $d = 200$.

Problem E. A Folding Task

Attempts: 1.

Accepted: 1.

First accepted: 03:48.

Every folding operation can be viewed as doubling the polygon: we need to cut the polygon with a line, the right part remains as it is, the left part should be reflected. So, after L operations we can view the paper as a union of at most 2^L (convex) polygons.

For each $i \in \{1, \ldots, 2^L\}$ we are to establish the total area covered by exactly i of these polygons. To do so, intersect all pairs of polygons' edges. Now we may split the plane into several vertical strips, in every strip no two segments intersect. We can sort these segments from top to bottom and store the type of each segment: the corresponding polygons either opens or closes. This can be easily processed by maintaining the current number of open polygons.

Problem F. Bonsai Tree

Attempts: 123.

Accepted: 33.

First accepted: 00:24.

Let's precalculate for each integer from 1 to 10^6 some prime number by which it is divisible. After that for each vertex we will maintain a multiset of primes S_v, and a_v — the GCD of all numbers in the subtree

v. We will call vertices brothers if they have a common parent.

Let's build the function push(v, p), which tries to propagate a prime p from vertex v to its parent. First, we assign $a_v := a_v \cdot p \mod 10^9 + 7$. Let u be the brother of the vertex v, and w is their parent. If $p \in S_u$, then we remove p from S_u and call push(w, p) recursively. Otherwise, we will simply add p to the set S_v.

Using push and the precalculated sieve of Eratosthenes, we can process the requests in an obvious way. Now let's estimate the time complexity of our solution. Let c_v denote the size of the subtree of v. Consider the potential $\Phi = \sum\limits_v c_v \cdot |S_v|$. Each call of push increases it by 1, and for obvious reasons it cannot exceed $2Q \cdot \log_2 C$. So, the algorithm works in time $O(R \log R)$, where $R = 2Q \cdot \log_2 C$.

Problem G. Pac-Man Speedrun

Attempts: 7.

Accepted: 4.

First accepted: 01:27.

Consider a checkerboard-like coloring of a $r \times c$ board. Any two adjacent cells are colored in two different colors, so it is easy to understand which cell we will get into after $r \cdot c - 1$ moves. If the cell (a, b) is painted in a different color, then let's choose any cell adjacent to it, but not $(1, 1)$; first we get to the chosen cell, and then in one move we will go to the cell (a, b).

So, we assume that the parity of the path length is consistent with the colors of the initial and final cells, then we need to constructively build a Hamiltonian path that starts at $(1, 1)$ and ends into (a, b). The case where r or c is equal to two is easily processed by hand. Otherwise, we can either walk along the entire first row from left to right and take one step down or walk along the entire first column and take a step to the right so as not to go through the cell (a, b). Note that after that we reduced the problem to the one with a smaller board size. We will repeat this algorithm until we come to the cell (a, b).

An accurate implementation consumes $O(1)$ memory and takes $O(r \cdot c)$ time.

Problem H. Most Skilled

Attempts: 19.

Accepted: 1.

First accepted: 03:02.

Let us introduce a graph, the vertices of which will be students, and the edges (u, v) will be drawn if the student u considers the student v to be the most skilled.

Let's look at our graph at some point in time. If we ignore the orientation of edges, it will consist of several connected components, while the number of edges in a given component will coincide with the number of vertices. If this graph contains a cycle from v to v, then if the teacher starts the process with any student from this component, then the student v will be assigned with the project, but if there is no such vertices, then the whole group gets the project.

We will solve the problem offline using the "divide and conquer" method applied to the queries.

To do this, we will assume that edges are not added and removed from the graph, but exist during some continuous period of time. We will also use the data structure "disjoint-sets with rollbacks" (DSR).

Let us consider the time interval $[T_l; T_r)$, then add to the DSR the edges that exist throughout this period of time, but were not added to the DSR earlier. Next, if $T_l + 1 = T_r$, we will answer the query with the number T_l. Otherwise, we will make two recursive calls over time intervals. After that, it will be necessary to remove the edges added in the current recursive call from the DSR (and just undo all the changes).

Let us figure out what should happen when edges are added to the DSR. For each component, we will store the number of vertex from which there is a self-loop (we will call such a vertex special), or the fact of its absence, as well as the number of steps to this loop h_v for each vertex v. This information is easily maintained when merging components.

Indeed, if an edge is added from v to u and at the same time v and u lie in different connected components, then the number h_u must be added to all numbers h_t, where t are the vertices from the connected component of vertex v. Rollback, that is, deleting the last added edge, is very easy to do by remembering what changes we made when adding a new edge.

To answer a query in which the process begins with a student p, it is enough to establish if the component of vertex p has a special vertex, and if there is, then the length of the path to it will be written in h_p.

Problem I. Nice Colorings — 3

Attempts: 79.

Accepted: 24.

First accepted: 00:43.

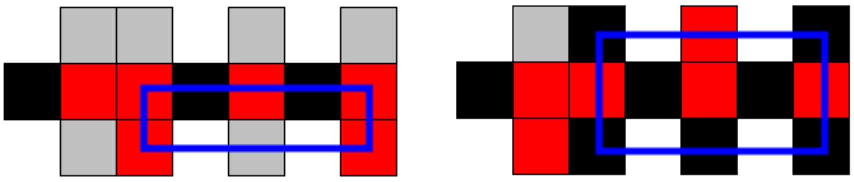
First, establish the colors of cells $(2, 1), (2, 2), \dots, (2, 7)$. From these 7 cells at least 4 will have the same color, say $(2, y_1), \dots, (2, y_4)$. Assume that this color is red.

Second, ask for the colors of $(1, y_1), (3, y_1), \dots, (1, y_4), (3, y_4)$.

We claim that one can always choose 4 needed cells out of these 15. Indeed, if at least 3 of 8 cells from the second step are colored red, then at least two of them lie in the same row, so we can find a red rectangle (refer to the left picture below).

Otherwise, both 1-st and 3-rd row have at least 3 black cells each, so we can find a black rectangle (refer to the right picture).

The authors argue that it suffices to use 14 cells.



(unimportant cells are colored grey)

Problem J. Vasily's Ladder

Attempts: 165.

Accepted: 39.

First accepted: 00:29.

For each i introduce $left\_greater(i)$ to be the maximum $j < i$ such that $a_j > a_i$. The values $right\_greater$ can be defined analogously. You can find them with a stack or with a segment tree.

For each i introduce $left\_repeat(i)$ to be the minimum $j \le i$ such that the values on $[j, i]$ do not repeat. The values $right\_repeat$ can be defined analogously. These values can be found by a two-pointers technique.

Now, for every i we want to understand whether there exists a suitable segment $[l, r]$, in which a_i is the maximum element. If $L = \max\{left\_repeat(i), left\_greater(i) + 1\}$ and $R = \min\{right\_greater(i) - 1, right\_repeat(i)\}$, then these boundaries have to satisfy the inequalities $l \ge L, r \le R$. It suffices to choose such l, r that $r - l + 1 = a_i$ and all values on $[l, r]$ are distinct, i.e. $left\_repeat(r) \le l$.

So, we want to establish wheter there exists an $r \in [L + a_i - 1, R]$, s.t. $left\_repeat(r) \le r - a_i + 1$. This can be done by supporting a sparse table for values $left\_repeat(r) - r$ and quering a segment for the minimum. The total running time is $O(n \log n)$.

Problem K. Sandwiches

Attempts: 216.

Accepted: 57.

First accepted: 00:23.

Introducing $\Phi = \sum\limits_{i,j} |c_i - c_j|$, where c_k equals the number of slices of cheese on top of the k-th sandwich, one can establish that the naive implementation is good enough: the total number of cheese moves is at most $O(KN)$.

Thus, we may manually perform operations of type 3 step by step. If we store a trie with K marked vertices, corresponding to the current state of sandwiches, we will only need to create new edges or ascend to a vertex's parent. The final answer is equal to the number of vertices in the trie excluding the root.

Problem L. Tea With Milk

Attempts: 100.

Accepted: 36.

First accepted: 00:15.

One can easily notice that all possible mixtures are defined by the Minkowski sum of segments $(0, 0) \div (t_i, m_i)$.

To find this sum we can treat these segments as polygons and perform the standard algorithm to find the Minkowski sum of two such polygons. Adding up these polygons by pairs, then by tuples of 4, 8, etc., we obtain the result in $O(N \log N)$ time. Mind the case of collinear segments.

The last thing to do is to find the number of lattice points inside the polygon. To do so, utilize Pick's theorem. The coordinates of the vertices may be up to $10^9 \cdot 10^5$, so be careful when multiplying these numbers. The point $(0, 0)$ should be subtracted.