Problem Tutorial: "Zero Sum"

Let's maintain $dp_{i,j}$: the smallest sum of elements of the first i rows with the sum of indices equal to j.

It works in $O(n^2k^2)$, which is too slow.

Let's random shuffle all rows, and bound j to some magic value C, so it will work in $O(nCk)$.

Let's take the array of -1 and 1 with sum 0, and randomly shuffle it. What is the expected value of prefix with the largest sum?

If the length of the array is n the expected value is $O(\sqrt{n})$.

Well, you just can run this idea on your computer and look at the values of the largest prefix sum that you are getting.

So if numbers are $-k \dots k$, we can set C around $O(k\sqrt{n})$.

So we can solve the problem in $O(nk^2\sqrt{n})$.

Constraints were very friendly, so probably even not optimal choose of C will give you OK.

Problem Tutorial: "MST"

The simplest and the most obvious way to solve this problem is to use Boruvka algorithm.

In this algorithm, $O(\log n)$ times, for each vertex, you should find the smallest weight of edge, which is outgoing from it to another connected component.

The smallest weight of edge outgoing to another component is either edge to the maximum on prefix from another component or to the minimum on the suffix.

For that purpose, you can maintain two maximums on the prefix (i.e, maximum value in the array and the maximum value among these that are not in the same connected component as the first maximum). And similarly two minimums on the suffix. And look only at these edges, to relax minimum outgoing edge from the connected component.

If you don't know about Boruvka algorithm, you can read it at Wikipedia.

Also, it is possible to solve it by simulating other MST algorithms with data structures and there exists a linear solution which doesn't simulate faster MST algorithms.

Problem Tutorial: "Tree Circles"

For each query, for each vertex, you should find another vertex, such that maximum on the path to it is minimal, and if d_v is this maximum on the path. Then, the answer is $d_{v_1} \cdot d_{v_2} \cdot \ldots \cdot d_{v_k}$.

Some solutions use the virtual tree and some ugly data structures, but I will tell the intended solution, which is very easy to write.

The key idea is that we can build the line tree which has the same maximums on all paths. I.e we can reduce maximum on path problem to the maximum on the segment.

How to do it? Well, just when you are adding an edge (in the order of increasing weights) you can take answers (line trees) for two connected components and connect them to one big line tree by adding edge with the current weight between them.

You can prove that the maximums on paths in this line tree are equal to the maximums on paths on the initial tree.

Now, if you want to find the vertex with the smallest maximum on the path, obviously you need to check only two adjacent vertices in the line graph.

So all that you need is dsu to build that line tree and sparse table for maximum on path queries.

Problem Tutorial: "Angle Beats 2.0"

Let's look at the fixed '*'.

For two adjacent dots to the left and the right, let's add an edge between them. Similarly to the up and the right.

If you don't have adjacent dots on horizontal or vertical obviously the number of ways is zero. Otherwise, if you have only one dot at some side you can add self-loop.

And let's say that orientation of the edge corresponds to the end of this edge that you are picking for the fixed star.

So we need to find the number of orientations of the edges, such that each vertex has in-degree at most 1.

I claim that if some connected component with n vertices has more than n edges, then the answer is 0. Otherwise, if it has exactly n edges, the answer is 2 (because you have two orientations of the cycle) or 1 if this component has self-loop. And if it has $n-1$ edge, answer for it is n (because you can pick any vertex as root and orient edges towards it).

So we just need to multiply all answers for connected components.

Problem Tutorial: "LIS"

For each i let's maintain the longest increasing subsequence with the end in i. After that, for each j you need to find i with the largest dp such that constraint in the statement is true.

Let's maintain the segment tree of convex hull tricks. Segment tree will be on segments of dp answers. And for each segment of the segment tree $[l, r]$ we will maintain convex hull of all a_i, b_i with $l \le dp_i \le r$.

If you have such a segment tree for $1, 2, \ldots, i-1$ you can easily get it for i. Just go down on this segment tree, and if the best value of the linear function for x_i in the right subtree is good enough, go there. Else, go to the left subtree.

After that, you need to add your linear function to $O(\log n)$ convex hulls (on the path in the segment tree).

So the complexity is $O(n \log^2 n)$ because dynamic convex hull works in $O(\log n)$ and also you need to do queries or adding edges to it $O(n \log n)$ times because of the segment tree.

Problem Tutorial: "Good Coloring"

Let's orient edges from the smaller colors to larger. And let's set dp_v: the longest path starting from v in this oriented graph. Now, of course, we have a path of size $\max dp_1, dp_2, \ldots, dp_n$ which contain all colors, and this coloring is proper.

Problem Tutorial: "Circle Convertation"

If n is odd, I claim that we can transform the string to one-colored by the following operations:

Take any one-colored maximal segment of equal colors of even length, and flip it to another color. That will decrease the number of maximal segments of equal colors on the circle.

If you don't have segments of even length, I claim that now the circle is one-colored. Because if it is not, you have an odd number of maximal segments so you should have two adjacent segments with the same color.

Of course, this operation don't change parity. So you just need to check that for s and t you will get the same string in the end. After that, you need to take operations for s and append to them operations of t in reverse order.

If n is even, let's invert all characters on even positions. And now your operations is a swap of two elements with a different color. So you just can use bubble sort to sort both s and t, check that they are equal in the end, and similarly as for odd n you can take operations for s and append to them operations of t in reverse order.

Problem Tutorial: "Equal MEX"

If the array is divided into segments with equal MEX, MEX on each segment is equal to the MEX on the array. It is easy to see because you have all elements $0, 1, \ldots, x-1$ and you don't have element x.

So the problem is: divide the array into segments with MEX equal to x. Where x is the MEX of elements of the array.

Let's use dp. For fixed r you have a prefix of l on which MEX is equal to x. You can find this prefix with a set, maintaining the last positions of each element among $0, 1, \ldots, x-1$ or with two-pointers. And with prefix sums, you can calculate dp for known l.

Problem Tutorial: "Cactus is Money"

For the fixed set of points of non-negative coordinates $(x_1, y_1), \ldots, (x_n, y_n)$ points with the smallest $x_i \cdot y_i$ lies on convex hull. That is because for fixed t hyperbola $xy = t$ when you will decrease t at first intersect a vertex on a convex hull. That is because of this hyperbola convex downward.

So you need to find the convex hull of all weights of the spanning trees. For that, for the fixed cycle, you can take the convex hull of all possible weights of this cycle after deletion one edge. And take the Minkowski sum of these hulls. Also, shift it by the weights of all bridges.

After that, you can naively find the smallest $x_i \cdot y_i$ of points on the derived convex polygon.

Problem Tutorial: "Good Permutations"

At first, let's solve the problem in $O(n^5)$.

Let's maintain $dp_{n,j_1,j_2,j_3,x}$: the number of permutations of n elements with three smallest j's in pairs with $i < j$ and $p_i < p_j$. (So, for example if you have these pairs $1, 3, 2, 3, 3, 4$, we will maintain $3, 3, 4$) and x triples $i < j < k$ and $p_i < p_j < p_k$, ofc we only interested in $0 \le x \le 3$.

To go $n \to (n+1)$ you can fix the number of $(n+1)$-th element in permutation, after that you can shift all j_1, j_2, j_3, check that you produced some new good triples or good pairs, and maybe add some new j_i, equal to $(n+1)$ (because, of course you may have some -1 if you have less than three such pairs).

Ok, so you can precalculate answers for $n \le 35$ for any $m \le 3$.

After that, I claim that you can rewrite the answer as

$a_n = \sum a_{n-k} \cdot P(n)$. Where k is not so big, and degree of $P(x)$ (yes, it is a polynomial) is not so big too. So you can do Gaussian elimination, to find proper coefficients of the small polynomial for that small recurrence to cover first 35 values.

Problem Tutorial: "Number Theory"

By brute force, you can found out that the answer is at most 31.

So at first in $O(p)$ for each value you can find it's square root modulo if it exists.

And after that for each x you can naively Bruteforce a from small to large, and if $x - a^2$ has a square root, relax the answer by a and break for current x.

As the answer is at most 31, it works fast enough to pass all tests.

Problem Tutorial: "Modulo Magic"

Let's assume that $1 \le i \le n$, not $< n$, because $n \mod 1$ is actually equal $n \mod n$.

Note that $n \mod i$, if $i > \frac{n}{2}$ is equal to $n - i$. So that's how we can get all remainders $0 \le i \le \frac{(n+1)}{2}$, and we can't get smaller remainder.

So if n is even, the answer is $\frac{n}{2}$. Otherwise, the answer is $\frac{(n+1)}{2}$.