Day 6 Contest

Moscow Pre-finals Workshop 2020

April 24, 2020

A. Fire On Field

Keywords: easy.

Construct elements of A_i one by one. Increase A_i while the condition is violated. This works fast enough.

B. Low Diameter Separator

Keywords: subtree DP, data structures.

Let $dp_{v,h}$ be the number of subsets S in the subtree of v such that:
- after removing S, each connected component has diameter $< K$;
- the longest path from v not containing any vertices of S has h vertices.

Use subtree DP to compute $dp_{v,h}$. For the case when $v \in S$, simply take $dp_{v,0}$ as the product of sums of $dp_{u,h}$ over all children u.
For the case $v \notin S$, consider children of v one by one.
Let $dp_{v,h}$ be the values before considering a child u. Then the new values $dp'_{v,h}$ can be obtained as follows: for all h_1, h_2 such that $h_1 + h_2 < K$ do $dp'_{v,\max(h_1,h_2+1)}$ += $dp_{v,h_1} \cdot dp_{u,h_2}$.

This is too slow. Instead let's split the cases $h_1 \geqslant h_2 + 1$ and $h_1 < h_2 + 1$. Then, for example, in the first case the update can be done with dp'_{v,h_1} += $dp_{v,h_1} \cdot \sum_{h_2 \leqslant \min(K-h_1-1,h_1-1)} dp_{u,h_2}$; second case is symmetrical. Use prefix sums of dp_u and dp_v to make each of these updates in $O(1)$.
Now each merge of dp_v and dp_u works in $O(d_v + d_u)$, where d_v is the depth of the subtree of v.

This is still too slow. Suppose that $d_v \geqslant d_u$, otherwise swap dp_v and dp_u. For $h_1, h_2 \leqslant d_u$ use the previous method to make all updates in $O(d_u)$.
For $h_1 > d_u$ we have $dp'_{v,h_1} = dp_{v,h_1} \cdot \sum_{h_2 \leqslant \min(d_u, K-h_1-1)} dp_{u,h_2}$.
There are $\leqslant d_u$ values of $h_1 > K - d_u - 1$. Use the same method with prefix sums as with $h_1, h_2, \leqslant d_u$.
For $d_u \leqslant h_1 \leqslant K - d_u - 1$ we effectively do dp_{v,h_1} *= $\sum_{h_2 \leqslant d_u} dp_{u,h_2}$. If $dp_{v,h}$ are stored in a segment tree, we can make all such updates with a single range multiplication.

Now each merge is done in $O(\min(d_v, d_u))$, times $O(\log n)$ for segment tree.
Statement. $\sum \min(d_v, d_u)$ over all merges is $O(n)$.
Proof. If in a merge $d_u \leqslant d_v$, draw the longest path down from u. These do not intersect.
It follows that the **total complexity** is $O(n \log n)$. *Exercise*: optimize to $O(n)$.

C. Islands

Keywords: DP.

Suppose that cyclic segments $[l_1, r_1]$ and $[l_2, r_2]$ contain the same set of indices, and we can cover both segments with paths visiting the indices in the same order.
If $a_{l_1-1} = b_{l_2-1}$, we can extend the segments. Similar for any of the other three possible equalities.
If with this process we can extend both segments to cover entire circles, we win. Let's do DP to find all reachable pairs of segments.
Statement. If the answer is reachable from a starting pair of indices $[l_1, r_1 = l_1]$ and $[l_2, r_2 = l_2]$, then it is always reachable as follows:
- While any of the current equalities holds, expand on them greedily.

Note that the maximal extensions of any pair of indices i and j in any directions can be precomputed in $O(n)$. Even with this optimization, not clear how many reachable intermediate states there are between greedy extensions.

For each pair of consecutive numbers a_i and a_{i+1}, draw a chord connecting these numbers in the second circle. Whenever we switch between equalities (say, from $a_{l_1} = b_{l_2}$ to $a_{l_1-1} = b_{r_2+1}$), the segments a_{l_1}, a_{l_1-1} and a_{r_1}, a_{r_1+1} in the second circles have to intersect.
Further, any intersection corresponds to at most two intermediate states.

Statement. If the answer exists, then the number of intersections between segments a_i, a_{i+1} and a_j, a_{j+1} in the second circle is $O(n)$.
Proof. Each stage of the above process leading to the answer creates at most one new intersection.

It follows that adding memoization to the above process (do not process any pair of segments $[l_1, r_1]$, $[l_2, r_2]$ more than once) leads to a fast solution: depending on if map or hashmap used for memoization, the **total complexity** is $O(n \log n)$ or $O(n)$.

D. Ladder Game

Keywords: greedy.

Consider all swaps corresponding to the bars. Suppose that numbers x and y were swapped more than once. Note that the swap $x, y \to y, x$ at consecutive positions should be followed by $y, x \to x, y$ (possibly at different positions), since there is no way to place x before y again without making the latter swap.
If these two swaps happen at bars i and j, then removing both bars i and j does not change the resulting permutation.
Indeed, numbers other than x and y do not care, and x and y will follow the same paths, except for the part between i and j.
Simulate the process, and remove bars with the above rule while possible.

Suppose that now each pair x, y is swapped at most once. Then in each swap $x, y \to y, x$ we have $x < y$.
Now removing any subset of bars will reduce the number of inversions in the resulting permutation, hence the ladder is minimal.
Total complexity: roughly $O(h^2)$, where h is the number of bars. Probably faster than that.

E. Network Vulnerability

Keywords: dynamic programming, segment tree.

Let $dp_{i,j}$ be the maximum number of intervals that can be contained in i connected components of intervals such that segment j has the rightmost end among all intervals involved in these components. It's easy to calculate $c_k(G)$ afterwards.

Let's represent any connected component in the interval graph by a segment $[L; R]$ on the line. Consider all intervals contained in $[L; R]$. If their union covers $[L; R]$ completely, these intervals can form a connected component. Since we want to maximize the number of used intervals, it always makes sense to keep all intervals contained in $[L; R]$ if we have such a connected component.

Thus, $dp_{i,j}$ stands for the case when $R = r_j$ for the rightmost component.

Suppose we have found $dp_{i-1,j}$ for all j and want to find $dp_{i,j}$.

Initially, assume that $L = l_j$ for the rightmost component. Then $dp'_{i,j} = \max\limits_{r_k < l_j} dp_{i-1,k} + 1 + nested_j$, where $nested_j$ is the number of intervals contained in $[l_j; r_j]$. All these values can be calculated in $O(n)$ for fixed i.

Now let's find $dp_{i,j}$ values to account for the case when $L < l_j$ as well. Assume that the intervals are sorted in increasing order of their right ends. Then $dp_{i,j} = \max(dp'_{i,j}, \max\limits_{k<j, r_k \geq l_j} dp_{i,k} + 1 + cnt_{k,j})$, where $cnt_{k,j}$ is the number of intervals with indices between k and j contained in interval j.

A straightforward implementation will have $O(n^3)$ time complexity.

To optimize this, we can use a segment tree over the number line supporting the following operations:
- increase values of a_i on segment by 1;
- set values of a_i on segment to $\max(a_i, v)$;
- get the value in a point.

Then $dp_{i,j}$ can be calculated as follows. For $i = 1 \dots n$:
- set $dp_{i,j} = \max(dp'_{i,j}, query(l_j) + 1)$;
- increase a_i on segment $[-\infty; l_j]$ by 1;
- set values of a_i on segment $[l_j, r_j]$ to $\max(a_i, dp_{i,j})$.

Total complexity: $O(n^2 \log n)$, where n is the number of intervals.

F. Quadrilaterals

Keywords: angle scanline.

Consider a vector v rotating counter-clockwise in the upper halfplane. Let us maintain points $p_1, \ldots, p_n$ sorted by increasing of $p_i \cdot v$.
Points p_i, p_j are swapped in this order when v is orthogonal to the line p_ip_j. Create $O(n^2)$ swap events and sort them by angle (use cross-product instead of actual angles).

Suppose that points p_i and p_j are now begin swapped. Points preceding/following p_i, p_j in the current order are on the opposite sides of the line p_ip_j.
The number of quadrilaterals with a diagonal p_ip_j is (the number of points before p_i in the order) $\times$ (the number of points after p_j in the order), add this number to the counter C.
The smallest quadrilateral $p_kp_ip_lp_j$ with diagonal p_ip_j has p_k and p_l on opposite sides of the line p_ip_j, and point-line distances $\text{dist}(p_k, p_ip_j) = \frac{area(\triangle p_ip_jp_k)}{|p_ip_j|} = |p_k \cdot v - p_i \cdot v|$ and $\text{dist}(p_l, p_ip_j) = \frac{area(\triangle p_lp_jp_k)}{|p_ip_j|} = |p_l \cdot v - p_i \cdot v|$ are smallest possible. Thus p_k and p_l are closest to p_i and p_j in the current order.
If the next point p'_k has the same distance as p_k, then $p'_kp_ip_lp_j$ is also smallest. In this case $p_kp'_k$ is parallel to p_ip_j, thus there are at most two candidates on either side of p_i, p_j in the current order (thus at most four candidates for a smallest quadrilateral).
For each smallest quadrilateral check if its convex or not.

In the end of this process we know the number of smallest quadrilaterals of each kind.
If there are C_c convex and C_n non-convex quadrilaterals in total, then $C = 2C_c + C_n$ and $\binom{n}{4} = C_c + C_n$. From this find C_c and C_n.

Total complexity: $O(n^2 \log n)$ for sorting the events.

G. Same Color

Keywords: DP optimization.

Sort all points by increasing of x.
Consider a set of chosen points S. To check if it's valid it suffices to check for each pair of consecutive points p, q in S if all points between them have correct closest color among p, q (also check points before the first and after the last).
Let dp_i be the smallest size of a valid subset of S with the last point being p_i. p_i can be the first point in S (to put $dp_i = 1$) if all points before it has the same color.
In the other case, there are two cases for the previous point p_j:
• all points between p_i and p_j (inclusive) have the same color;
• there is one color change between p_i and p_j.
In all of these cases $dp_i = \min(dp_i, dp_j + 1)$. The first case for p_j is easy to consider.

For the second case, let p_k be the last point before p_i with $c_k \neq c_i$.
Then all points between p_j and p_k (inclusive) must have the same color, and also $p_i - p_k \geqslant p_k - p_j$, $p_i - p_{k+1} \leqslant p_{k+1} - p_j$.
This gives us a range for possible p_j, update dp_i with the minimum dp_j in the range $(+1)$.
We can further notice that whlie the color of p_i does not change, the borders of the range for p_j can not increase, thus we can use minimum queue to maintain $\min dp_j$.

The answer is $\min dp_i$ over each i such that $c_i = \ldots = c_n$.

Total complexity: $O(n)$ ($O(n \log n)$ should be fine too).

H. Strike Zone

Keywords: scanline, RSQ.

Compress coordinates. Let the points be (i, y_i)
Fix the top border y_l of the rectangle, and move the bottom border y_r down.
Keep $a[i] = 0$ if the point with $x = i$ is $y_i \notin [y_l, y_r]$, otherwise $a[i] =$ the score of (i, y_i) being inside the rectangle (either c_1 or $-c_2$).
For the current values of y_l, y_r update the answer with the largest subsegment sum of $a[i]$. This can be found with RSQ.
Each step of y_r introduces one new point into the range $[y_l, y_r]$, thus one RSQ update.

Total complexity: $O(n^2 \log n)$.

I. Thread Knots

Keywords: binary search.

Since the segments are not nested, they can be ordered such that for $i < j$ we have $l_i < l_j$ and $r_i < r_j$.
Binary search on the smallest distance d. For each $i = 1, \ldots, n$ the leftmost possible knot is $p_i = \max(l_i, p_{i-1} + d)$.
The answer is at least d if for all i we have $p_i \leqslant r_i$.

Total complexity: $O(n \log \max x_i)$.

J. Triangulation

Keywords: math.

There is a bijection between triangualtions of the n-gon and binary trees with n leaves and $n-2$ internal vertices with degrees 3.
Diameter of the triangulation is smaller than diameter of the corresponding tree by 2.
Let us find the largest number of leaves in a valid tree with diameter $d \geqslant 2$. Note that any smaller number of leaves can be achieved by removing leaves in pairs.
If d is even, then by considering the center of a diameter we can the answer $3 \cdot 2^{d/2-1}$.
If d is odd, then the center of a diameter is an edge, and the answer is $2^{(d+1)/2}$.
Find the smallest d such that the bound is at least n, and print $d - 2$.

K. Washer

Keywords: stereometry.

$k = 1$ is trivial.
Let $\overline{x}, \overline{y}, \overline{z}$ be averages of x_i, y_i, z_i respectively.
Observe that $\sum_i (x_i - \overline{x})^2 + (y_i - \overline{y})^2 + (z_i - \overline{z})^2 = \min_{x,y,z} \sum_i (x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2$, where $\min_{x,y,z}$ is taken over all points.
Suppose we now instead choose two points P and Q, and match each point $p_i = (x_i, y_i, z_i)$ to the closest of P and Q. The minimum answer over all P and Q is equal to the original answer (since we can take P and Q equal to $(\overline{x}, \overline{y}, \overline{z})$ for parts of the answer).
For each P and Q, the division into two parts is created by the midperpendicular plane of P and Q. Thus, we can try all partitions of the given points with a plane.
Each plane can be moved to contain three of the given points without affecting the partition, thus there are only $O(n^3)$ possible partitions. For each of them, construct the plane, divide the points into two groups and compute the answer.
Take care to consider all cases of assigning groups to the three points deciding the plane.

Total complexity: $O(n^4)$.

L. What's Mine is Mine

Keywords: DP.

Let dp_x be the largest answer is we need to finish not later than x.
Consider segments $[l, r]$ by increasing of l (or r), and use dp_{l-1} to update dp_r.
Total complexity: $O(\max r_i + n)$.
Constraints are really small, so even slower solutions are accepted.

M. Alternative Facts

Keywords: bipartite matching, data structures.

Hall's theorem. A bipartite graph has a matching covering all vertices of its left half iff for any subset A of the left half the number of distinct neighbours of vertices of A is at least $|A|$.
Statement. Suppose that in a bipartite graph each vertex of the left half i is connected to a range $[l_i, r_i]$ of right half vertices, and further l_i and r_i are non-decreasing. Then it suffices to verify Hall's theorem condition only for subsets A that are ranges $[i, j]$.

Construct a bipartite graph:
• the left half contains sorted numbers $a_1, \ldots, a_l$; • the right half contains sorted numbers $b_1, \ldots, b_n$; • numbers are connected when their absolute difference is at most k.

The graph satisfy the criterion above. Introduce the following numbers:
• p_i is the number of unmatched vertices $j \leqslant i$ of the left half;
• r_i is the number of unmatched vertices b_x of the right half with $b_x \leqslant a_i + k$;
• l_i is the number of unmatched vertices b_x of the right half with $b_x \leqslant a_i - k$.

Then Hall's theorem implies that the answer can be completed if for any $i < j$ we have $p_j - p_i \leqslant r_j - l_i$. Transform this to $p_j - r_j \leqslant p_i - l_i$.
Matching vertex i of the left half with a number x of the right half results in:
• forgetting about p_i, l_i, r_i; • decreasing p_i by 1 for all i such that $a_i \geqslant b_x$;
• decreasing r_i by 1 for all i such that $a_i \geqslant b_x - k$;
• decreasing l_i by 1 for all i such that $a_i \geqslant b_x + k$.

We can use this with segment tree to find the largest number $b_x \leqslant a_i + k$ to match with current a_i.