Day 3 Contest: Worldwide Selection 1

Moscow International Workshop 2020

November 30, 2020

A. Acetone, Water and ICPC

Keywords: angle sweep-line.

We are given points of two types (call them red and blue) in the plane.
We are asked to draw a straight line dividing the plane into regions A and B, so that (number of red points in A) + (number of blue points in B) is maximum.
We'll assume that the line is directed, so that A and B are on the left and right of the line respectively.

Cases where all points are in the same half are trivial.
In all other cases, *by moving the line around*, we can always obtain a solution so that there are points immediately to the left and right of the line. Effectively, we only need to consider lines nearly passing through a pair of points. We could try all such pairs, and explicitly count the results in $O(n^3)$ time. To speed up, fix a point p that's immediately to the left of the line, and imagine rotating the line around the point.
For any other point q, there will be two events "q enters A" and "q enters B". We can sort these events by slopes of lines pq, and then process them while bookkeeping the number of points in each half. This results in an $O(n^2 \log n)$ solution.
It is not guaranteed that no three points line on a common line, so extra care is needed for points lying on the same line that contains p. Since the line is very close to p, we can see that events for points in one of the half-lines will precede the other half, and the order depends on the distance to p.

B. Big Boss

Keywords: Euler tour, sweep-line.

Let $tin_i[v], tout_i[v]$ be the in- and out-times of a vertex v in the DFS of the tree $i = 1, 2$ starting from the root.
u is a descendant of v iff $tin[v] \leqslant tin[u] \leqslant tout[v]$.
For each vertex, in the 2D plane create a point $p_v = (tin_1[v], tin_2[v])$ and a rectangle $R_v = [tin_1[v], tout_1[v]] \times [tin_2[v], tout_2[v]]$. The condition above implies that u is a descendant of v in both trees iff $p_u \in R_v$.
To count the number of points in each rectangle, a standard sweep-line works. Move a vertical line left to right, and process events for points, left and right sides of rectangles.
Maintain an RSQ structure $a_1, \ldots, a_n$, and increment a_y whenever a point (x, y) in encountered.
The number of points inside the rectangle = (# points to the left of the right side) - (# points to the left of the left side). Each of the two can be found with a range sum query at an appropriate time in the process.

Complexity: $O(n \log n)$.

C. Circular Task

Keywords: circle tangents.

If a solution exists, *by moving the segment around* we can find a position where the segment is a common tangent for two of the given circles (either obstacles or holes).
Try all pairs of circles, and all (up to four) common tangent lines. For each line find intersections with the holes (if they don't exist, the line is invalid).
Check that there are no collisions with obstacles between the two intersections.

Complexity: $O(n^3)$. *Could be optimized to $O(n^2 \log n)$ with angle sweep-line?*

D. Diary

Keywords: DP.

For a given compact form $b_1, \ldots, b_n$, we can go from left to right and for each index i figure out the smallest possible value v_i that agrees with provided digits of b_i, and $v_i > v_{i-1}$. If for any i we have $v_i < a_i$, then the form is invalid. Note that $v_i > a_i$ is impossible.
Let $dp_{i,j}$ be the total length of the shortest common form $b_1, \ldots, b_i$ such that $v_j = a_j$ for all $j \leqslant i$, and b_i has j digits. The transitions $dp_{i,j} \to dp_{i+1,k}$ are straightforward.

Complexity: $O(nd^2)$, where d is the largest number of digits in any a_i.

E. Example

Keywords: parsing, elbow grease.

Split the input into tokens, and parse it (e.g. with recursive descent). Carefully implement all the rules.

F. Find The Integer

Keywords: binary search.

Suppose we subtract $x \leqslant 9$. The last digit of v was at least x before the subtraction iff the digit sum s decreases by x.
We can pin down the last digit by keeping track of the range of its possible values $[l, r]$, and subtracting $\lfloor (l+r)/2 \rfloor$ if the range contains more than one value.
Additionally, we can always infer $r \leqslant s$, which makes sure we never underflow the number.
Once we know the last digit, subtract it, and proceed to figure out the 10's digit, and so on.
In the end $v = 0$, thus we know the number as the sum of all subtractions.
We will need at most $4 + 1$ queries per digit, which is small enough.

G. Game with Blocks

Keywords: implementation.

There will be at most nm block removals. We want to process each of them in $O(nm)$, or close to that.

First, let's figure out which (if any) blocks are removed on the current step. Extend the grid boundary in each direction to accomodate possible segments.
Start by finding the closest directly visible block from each cell in all four directions. If removed blocks are connected with a single segment, they are directly visible from each other.
For two segments, the removed blocks are both visible from the turning point, thus we can try all possible turning points.
For three segments, consider a maximal unblocked strip that contains the middle segment. Both of the removed blocks must be visible from cells in the strip, and any matching pair works. Thus, for each strip we are only interested in two lexicographically earliest visible blocks with each letter.
Only the last part is non-trivial, and it can be done in $O(nm \cdot 26)$, $O(nm \log 26)$, or even $O(nm)$ (probably not much difference in practice).

Now, we only need to simulate the blocks' movement. Maintain a set S of blocks that currently can move forward in their respective direction, updating it carefully after each removal and each block movement.
To start a turn of checking, move the lexicographically earliest block in S. Later in the same turn, if p was the last moved block, take the upper bound of p in S.
Make sure not to move the same block twice in the same turn, e.g. by storing the moved blocks in another temporary set until the turn ends.
Repeat turns until S is empty. Note that the total number of steps for all blocks doesn't exceed $nm \max(n, m)$, thus overall complexity of all movements if $O(nm \max(n, m) \log(nm))$.

H. Hieroglyphs

Keywords: binary search, inclusion-exclusion.

For each query q, binary search on the answer x. We have to check if there are at least q numbers not exceeding x that are divisible by at least one of $a_1, \ldots, a_n$.
Apply inclusion-exclusion: add the numbers divisible by one of $a_1, \ldots, a_n$, subtract the number divisible by each pair among $a_1, \ldots, a_n$, and so on.

There are $\lfloor x/LCM(b_1, \ldots, b_k) \rfloor$ numbers up to x that are divisible by all $b_1, \ldots, b_k$.
Constraints ensure that no overflows ever happen.
Complexity is $O(\log \max ans \cdot 2^n)$ per query. Additional factors like n should also be fine.

I. Incident in Bytesburg

Keywords: subtree DP.
This solution was made famous in the "Looking For A Challenge?" book by Polish community.

Number the turns $0, \ldots, t-1$ **from last to first**. Each strategy can be viewed as a decision tree, and each vertex is queried in exactly one node of the tree.
Let a_v be the turn number when vertex v is queried. If two vertices v, u are queried in the same turn $a_v = a_u$, it must happen in different branches of the decision tree, thus there must be a vertex w on the vu-path that was asked prior to that, i.e. $a_w > a_v$.
One can see that any assignment of numbers a_v subject to the above constraint leads to a valid guessing strategy. Now we want to choose a_v so that to minimize $\max a_v$.

Root the tree at any vertex. Let s_v be the set of all a_u in the subtree of v such that there is no $a_w > a_u$ on the uv-path.
If values of a have been determined in subtrees of all children $u_1, \ldots, u_k$ of v, let $S = s_{u_1} \cup \ldots \cup s_{u_k}$. a_v can only take values not present in S. Also, if x is present in at least two of s_{u_i}, then $a_v > x$.
Greedy considerations show that a_v should take the lowest value x possible. Then s_v is obtained by adding x to S, and removing $0, \ldots, x-1$.
If sets are treated as binary masks, we have $s_v \leqslant S + 1 \leqslant \sum s_{u_i} + 1$. In particular, we always have $s_v \leqslant n$.
The answer is $\max a_v$ assigned this way. Representing s_v as integers, this solution can be implemented in $O(n)$.

J. James Bond

Keywords: basic (but nasty) geometry, binary search.
We are looking for a point p such that the angle between the tangents from p to the polygon is at least A, and the distance to the polygon is smallest possible. There is also a lower bound on the distance, which is insubstantial since we can increase the answer if needed by moving p away.
Try all options of where the tangents touch the polygon. The touching points are always vertices v, u. Since the polygon is regular, we only care about the number of sides between v and u. p is limited to an angle D enclosed by two lines containing sides of the polygon adjacent to touching points.
$\angle vpu = \frac{1}{2}\angle vOu$, where O is the center of the circle passing through p, u, v. We then have $\angle vOu \geqslant 2A$. There is a unique suitable circle center O on the midperpendicular of vu such that $\angle vOu = 2A$, and p has to outside or on the boundary of this circle C.

We now have to find p inside of $D \setminus C$ so that the distance to the polygon is smallest. One can see that p must belong to the arc δ of C inside D.
Let us binary search on the smallest distance d. Consider the region $R(d)$ of points at distance $\leqslant d$ from the polygon. The boundary $B(d)$ of $R(d)$ consists of segments and circle arcs.
We now need to check if δ is completely inside of $R(d)$. To do this, check if any point of δ is inside $R(d)$, and look for intersections between δ and parts of $B(d)$.

The total complexity is something like $O(n^2 \log \varepsilon^{-1})$, where ε is desired accuracy.

K. King and Highways

Keywords: matroid intersection.
We are looking for a largest subgraph without cycles and repeating edge colors.
Sets of edges without cycles form a *graph matroid*, and sets of edges of disjoint colors form a *colorful (rainbow) matroid.* The problem of finding a largest common set between two matroids is known as *matroid intersection.*
A great introduction to matroid intersection is given in in this Codeforces post.