Training Contest 1 Editorial, April 30, 2021

April 30, 2021

Mikhail Tikhomirov & Filipp Rukhovich

Bytedance-Moscow Workshops Training Camp, 2021

A. Algorithmia's Sky

Given n points $P = \{p_1, p_2, ..., p_n\}$ on the plane; each point p_i moves with speed v_i. The problem is to find a day between 0 and given D such that diameter of set of points at this day is minimal.

A. Algorithmia's Sky

Let $dst2_{ij}(t)$ be a square of distance between p_i and p_j at day t, and $diam2(t)$ be a square of diameter at that day. Obviously, $dst2_{ij}(t)$ is convex downward function; $diam2(t)$ is convex downward too as maximum of $n(n-1)/2$ convex downward functions. Then, we can use ternary search to find a minimal possible diameter.

For any fixed day t, the square of diameter $diam2(t)$ can be found in $O(nlogn)$ time (full description of the algorithm can be found in `http://euro.ecom.cmu.edu/people/faculty/mshamos/1978ShamosThesis.pdf`, pp.76-82). Then, the complexity of our solution is $O(n\ log\ n\ log\ D)$.

B. Build The Trees

Given undirected graph G with n enumerated vertices and m edges, find number of different trees on n vertices containing G as subgraph.

B. Build The Trees

If there are some cycles in G, the answer is 0.

Otherwise, suppose that G contains k components of connection, and sizes of these components are $s_1, s_2, \cdots, s_k$. Then, the answer is

$answer = s_1 s_2 \ldots s_k n^{k-2}$.

This formula is a simple generalization of Cayley's formula about a number of trees and can be proved using Prufer's code. Full proof can be found, for example, here: https://acmcairoscience.wordpress.com/2015/04/07/prufer-code-cayley-formula-the-number-of-ways-to-make-a-graph-connected/

C. Contest on TV

There is a two-stage competition, with total rankings determined by sum of scores. We know a — the results of the first stage, and c — shuffled results of the second stage. How many assignments between second stage results and participants are there so that top-k stays the same as in the ranking only by the first stage?

C. Contest on TV

Suppose that a and c are sorted by decreasing. Consider a candidate permutation π: a participant with score a_i in the first stage scored $c_{\pi(i)}$ in the second stage.

C. Contest on TV

Suppose that a and c are sorted by decreasing. Consider a candidate permutation π: a participant with score a_i in the first stage scored $c_{\pi(i)}$ in the second stage.

If π is good, then there exists such x that:

- $a_i + c_{\pi(i)} \geqslant x$ for $i \leqslant k$;

C. Contest on TV

Suppose that a and c are sorted by decreasing. Consider a candidate permutation π: a participant with score a_i in the first stage scored $c_{\pi(i)}$ in the second stage.

If π is good, then there exists such x that:

- $a_i + c_{\pi(i)} \geqslant x$ for $i \leqslant k$;
- $a_i + c_{\pi(i)} \leqslant x$ for $i > k$.

C. Contest on TV

Graphically, we mark cells (i, j) with green if $\pi(j) = i$ is possible according to criterion above. Our task is to count the number of assignments in which all corresponding cells $(\pi(j), j)$ are green; one such assignment is denoted by circled cells.

$a \backslash c$	1	2	4	6	8	9
1	2	3	5	7	9	10
3	4	5	7	9	11	12
4	5	6	8	10	12	13
5	6	7	9	11	13	14
6	7	8	10	12	14	15
7	8	9	11	13	15	16

C. Contest on TV

Graphically, we mark cells (i, j) with green if $\pi(j) = i$ is possible according to criterion above. Our task is to count the number of assignments in which all corresponding cells $(\pi(j), j)$ are green; one such assignment is denoted by circled cells.

$a \backslash c$	1	2	4	6	8	9
1	2	3	5	7	9	10
3	4	5	7	9	11	12
4	5	6	8	10	12	13
5	6	7	9	11	13	14
6	7	8	10	12	14	15
7	8	9	11	13	15	16

This is the same as the *permanent* of the matrix with 1's in green

C. Contest on TV

From the constraints on the green cells we notice that it is possible to draw a vertical line so that the top-left and bottom-right parts of the table are all green (let m denote the number of columns to the left of the line).

a \ c	1	2	4	6	8	9
1	2	3	5	7	9	10
3	4	5	7	9	11	12
4	5	6	8	10	12	13
5	6	7	9	11	13	14
6	7	8	10	12	14	15
7	8	9	11	13	15	16

C. Contest on TV

The strategy is now as follows: for all x count the number of ways to circle x elements in top-right and bottom-left parts independently, then try to place the rest the all-green parts (they are easier).

$a \backslash c$	1	2	4	6	8	9
1	2	3	5	7	9	10
3	4	5	7	9	11	12
4	5	6	8	10	12	13
5	6	7	9	11	13	14
6	7	8	10	12	14	15
7	8	9	11	13	15	16

C. Contest on TV

Consider, say, bottom-left part. Its green cells form a ladder-like configuration.

C. Contest on TV

Consider, say, bottom-left part. Its green cells form a ladder-like configuration.

The number of ways to place several elements there can be counted with a simple DP $ways_{i,j} = \#$ of ways to circle j elements in i first columns of the ladder.

C. Contest on TV

Consider, say, bottom-left part. Its green cells form a ladder-like configuration.

The number of ways to place several elements there can be counted with a simple DP $ways_{i,j} = \#$ of ways to circle j elements in i first columns of the ladder.

The formula is $ways_{i,j} = ways_{i-1,j} + ways_{i-1,j-1} \cdot (h_i - j + 1)$, where h_i is the number of green cells in the i-th column.

C. Contest on TV

Consider, say, bottom-left part. Its green cells form a ladder-like configuration.

The number of ways to place several elements there can be counted with a simple DP $ways_{i,j} = \#$ of ways to circle j elements in i first columns of the ladder.

The formula is $ways_{i,j} = ways_{i-1,j} + ways_{i-1,j-1} \cdot (h_i - j + 1)$, where h_i is the number of green cells in the i-th column.

Indeed, if $j-1$ elements were placed before, they forbid exactly $j-1$ positions in the current column (here we use the ladderness).

C. Contest on TV

Consider, say, bottom-left part. Its green cells form a ladder-like configuration.

The number of ways to place several elements there can be counted with a simple DP $ways_{i,j} = \#$ of ways to circle j elements in i first columns of the ladder.

The formula is $ways_{i,j} = ways_{i-1,j} + ways_{i-1,j-1} \cdot (h_i - j + 1)$, where h_i is the number of green cells in the i-th column.

Indeed, if $j - 1$ elements were placed before, they forbid exactly $j - 1$ positions in the current column (here we use the ladderness).

We do the similar DP to count the same quantities in the top-right part.

C. Contest on TV

Suppose now that we have placed x elements in the bottom-left part, and y elements in the top-right part.

C. Contest on TV

Suppose now that we have placed x elements in the bottom-left part, and y elements in the top-right part.

The top-left green part has $n - k - y$ free rows and $m - x$ free columns. To fill these rows and columns completely we must have $n - k - y = m - x = t$, and the number of ways to do it in this case is $t!$.

C. Contest on TV

Suppose now that we have placed x elements in the bottom-left part, and y elements in the top-right part.

The top-left green part has $n-k-y$ free rows and $m-x$ free columns. To fill these rows and columns completely we must have $n-k-y=m-x=t$, and the number of ways to do it in this case is $t!$.

A similar multiplier corresponds to filling the bottom-right part.

C. Contest on TV

Suppose now that we have placed x elements in the bottom-left part, and y elements in the top-right part.

The top-left green part has $n-k-y$ free rows and $m-x$ free columns. To fill these rows and columns completely we must have $n-k-y=m-x=t$, and the number of ways to do it in this case is $t!$.

A similar multiplier corresponds to filling the bottom-right part.

Altogether, we are able to obtain the answer in $O(n^2)$ time for a single x.

C. Contest on TV

However, there may be multiple x's corresponding to a valid solution.

C. Contest on TV

However, there may be multiple x's corresponding to a valid solution.

Instead we will count the number of valid assignments so that minimal $a_i + c_{\pi(i)}$ among top-k contestants is exactly x.

C. Contest on TV

However, there may be multiple x's corresponding to a valid solution.

Instead we will count the number of valid assignments so that minimal $a_i + c_{\pi(i)}$ among top-k contestants is exactly x.

To do this, we perform the previous computation, and subtract the number of assignments that satisfy:

- $a_i + c_{\pi(i)} > x$ for $i \leqslant k$;
- $a_i + c_{\pi(i)} \leqslant x$ for $i > k$.

C. Contest on TV

However, there may be multiple x's corresponding to a valid solution.

Instead we will count the number of valid assignments so that minimal $a_i + c_{\pi(i)}$ among top-k contestants is exactly x.

To do this, we perform the previous computation, and subtract the number of assignments that satisfy:

- $a_i + c_{\pi(i)} > x$ for $i \leqslant k$;
- $a_i + c_{\pi(i)} \leqslant x$ for $i > k$.

...which can be computed roughly in the same way.

C. Contest on TV

However, there may be multiple x's corresponding to a valid solution.

Instead we will count the number of valid assignments so that minimal $a_i + c_{\pi(i)}$ among top-k contestants is exactly x.

To do this, we perform the previous computation, and subtract the number of assignments that satisfy:

- $a_i + c_{\pi(i)} > x$ for $i \leqslant k$;
- $a_i + c_{\pi(i)} \leqslant x$ for $i > k$.

...which can be computed roughly in the same way.

Finally, sum the answers over all x. The resulting complexity is $O(an^2)$, where a is the maximal score.

D. Damage

Given n skills, each of them helps hero to do d_i damage to the monster with probability p_i. Each round, hero chooses skills equiprobably one-by-one and try to use it; round is over when skills are over or some skills does damage successfully; each skill can't be chosen more than once. The question is: what it the mathematical expectation E of damage after one round?

D. Damage

Consider a scenario in which damage was done by skill i, and before it, k different skills $j_1, j_2, ..., j_k$ was chosen in some order and was unsuccessful. The probability of this scenario is $p_i * k! * \frac{(1-p_{j_1})*(1-p_{j_2})*\cdots*(1-p_{j_k})}{n*(n-1)*...*(n-k+1)}$. So, the answer is

$$E = \sum_{i=1}^{n} d_i p_i \sum_{k=0}^{n-1} (k! * f(i,k)), \text{ where}$$
$$f(i,k) = \sum\{(1-p_{j_1})...(1-p_{j_k}) | 1 \leqslant j_1 < j_2 < ... < j_k \leqslant n,\ each\ j_l - s\ is\ not\ equal\ to\ i\}.$$

So, if we know $f(i,k)$ for all i and k, then we can solve our problem in $O(n^2)$. But how to find $f(i,k)$?

D. Damage

Let $d[i][k]$ be $\sum\{(1-p_{j_1})\ldots(1-p_{j_k})|1 \leqslant j_1 < j_2 < \ldots < j_k \leqslant i$.

Then, $d[i][k] = d[i-1][k] + (1-p_i) * d[i-1][k-1]$ for $k, i \geqslant 1$; also, $d[i][0] = 1$ for any i. It means that:

1. all $d[i][k]$ can be found in $O(n^2)$;
2. $f(i,k) = d[n][k] - f(i,k-1) * (1-p_i)$ ("sum of products without i-th is sum of products minus sum of products with i-th"); so, all $f(i,k)$ can be calculated in $O(n^2)$ too.

So, complexity of calculating $f(i,k)$-s and the whole solution is $O(n^2)$.

E. Engines and Pumps

There are n red points and m blue points on the real line ($n \leqslant m$). We have to assign a blue point to each read point so that:

- no blue point is assigned to more than one red point;
- sum of the distances between assigned pairs is smallest possible.

E. Engines and Pumps

Consider a particular assignment f of a blue point $f(x)$ to each red point x. Let us say that a red point x is of *type* L if $f(x) \leqslant x$, and otherwise x is of *type* R.

E. Engines and Pumps

Consider a particular assignment f of a blue point $f(x)$ to each red point x. Let us say that a red point x is of *type L* if $f(x) \leqslant x$, and otherwise x is of *type R*.

If f is an optimal assignment, then:

- If x is a red point of type R, y is a red point of type L, and $x < y$, then $f(x) < f(y)$ (otherwise we can swap $f(x)$ and $f(y)$ to obtain a better assignment);

E. Engines and Pumps

Consider a particular assignment f of a blue point $f(x)$ to each red point x. Let us say that a red point x is of *type L* if $f(x) \leqslant x$, and otherwise x is of *type R*.

If f is an optimal assignment, then:

- If x is a red point of type R, y is a red point of type L, and $x < y$, then $f(x) < f(y)$ (otherwise we can swap $f(x)$ and $f(y)$ to obtain a better assignment);
- Therefore, if x is, say, an L-type red point, and another red point y satisfies $f(x) \leqslant y < x$, then y is an L-type point as well.

E. Engines and Pumps

Consider a particular assignment f of a blue point $f(x)$ to each red point x. Let us say that a red point x is of *type L* if $f(x) \leqslant x$, and otherwise x is of *type R*.

If f is an optimal assignment, then:

- If x is a red point of type R, y is a red point of type L, and $x < y$, then $f(x) < f(y)$ (otherwise we can swap $f(x)$ and $f(y)$ to obtain a better assignment);
- Therefore, if x is, say, an L-type red point, and another red point y satisfies $f(x) \leqslant y < x$, then y is an L-type point as well.
- If $f(x) < x$ for a red point x, then all blue points in the range $(f(x), x]$ are assigned to type-L red points.

E. Engines and Pumps

It follows from these observations that all assigned points can be decomposed into contiguous segments such that:

- the number of red and blue points in the segment are the same;

E. Engines and Pumps

It follows from these observations that all assigned points can be decomposed into contiguous segments such that:

- the number of red and blue points in the segment are the same;
- all red points are assigned to the blue points of the segment, and vice versa;

E. Engines and Pumps

It follows from these observations that all assigned points can be decomposed into contiguous segments such that:

- the number of red and blue points in the segment are the same;
- all red points are assigned to the blue points of the segment, and vice versa;
- all red points are of the same type (L or R).

E. Engines and Pumps

It follows from these observations that all assigned points can be decomposed into contiguous segments such that:

- the number of red and blue points in the segment are the same;
- all red points are assigned to the blue points of the segment, and vice versa;
- all red points are of the same type (L or R).

It follows that the balance of each prefix of such subsegment (where red point counts as $+1$, and a blue point as -1) is either always non-negative or non-positive (depending of L/R type of red points). If the balance reaches zero at a position other than endpoints, we can decompose the segment into smaller ones.

E. Engines and Pumps

Consequently, for each possible right endpoint r of a segment in the aggregate sorted list of points there is exactly one candidate position for the left endpoint, namely, $prev(r)$ — the previous position with the same prefix balance.

E. Engines and Pumps

Consequently, for each possible right endpoint r of a segment in the aggregate sorted list of points there is exactly one candidate position for the left endpoint, namely, $prev(r)$ — the previous position with the same prefix balance.

We can now implement DP: dp_k — minimal cost of assigning k leftmost red points to blue points.

E. Engines and Pumps

Consequently, for each possible right endpoint r of a segment in the aggregate sorted list of points there is exactly one candidate position for the left endpoint, namely, $prev(r)$ — the previous position with the same prefix balance.

We can now implement DP: dp_k — minimal cost of assigning k leftmost red points to blue points.

To make a transition, we consider all sensible segments of points (see above). Suppose that a segment $[prev(p), p]$ contains red points with numbers in $[l, r]$. We then have to try to improve dp_r with $dp_{l-1}+$the cost of assigning points inside $[prev(p), p]$.

E. Engines and Pumps

Consequently, for each possible right endpoint r of a segment in the aggregate sorted list of points there is exactly one candidate position for the left endpoint, namely, $prev(r)$ — the previous position with the same prefix balance.

We can now implement DP: dp_k — minimal cost of assigning k leftmost red points to blue points.

To make a transition, we consider all sensible segments of points (see above). Suppose that a segment $[prev(p), p]$ contains red points with numbers in $[l, r]$. We then have to try to improve dp_r with $dp_{l-1}+$the cost of assigning points inside $[prev(p), p]$.

Since all red points always have the same type in this assignment, the cost is equal to |(sum of red points coordinates) - (sum of blue points coordinates)| inside the segment.

E. Engines and Pumps

Consequently, for each possible right endpoint r of a segment in the aggregate sorted list of points there is exactly one candidate position for the left endpoint, namely, $prev(r)$ — the previous position with the same prefix balance.

We can now implement DP: dp_k — minimal cost of assigning k leftmost red points to blue points.

To make a transition, we consider all sensible segments of points (see above). Suppose that a segment $[prev(p), p]$ contains red points with numbers in $[l, r]$. We then have to try to improve dp_r with dp_{l-1}+the cost of assigning points inside $[prev(p), p]$.

Since all red points always have the same type in this assignment, the cost is equal to |(sum of red points coordinates) - (sum of blue points coordinates)| inside the segment.

F. Funny Lottery

There are R red, G green and B blue marble in the lottery wheel; on each step, some random marble is spat from the wheel; if it's red marble, then we do nothing more on this step; otherwise, we return this marble to the wheel. The problem is to find a mathematical expectation of number of step in which blue marble will be spat in K-th time by modulo $MOD = 10^9 + 7$.

F. Funny Lottery

Let $d[r][k]$ be an answer to the problem for $R = r, K = k$, G and B are the same as in input data. Then, it follows from the statement that for all natural r, k:

$$d[0][0] = d[r][0] = 0,\ d[0][k] = k,$$

$d[r][k] =$
$1 + \frac{r}{r+G+B} * d[r-1][k] + \frac{B}{r+G+B} * d[r][k-1] + \frac{G}{r+G+B} * d[r][k]$,
i.e.

$$d[r][k] = \frac{r+G+B+R*d[r-1][k]+B*d[r][k-1]}{r+G+B}.$$

Then, it can be proven by induction and calculated in $O(logk + logMOD)$ that

$$answer = d[R][K] = R + K + \frac{KG}{B} - R(\frac{B}{B+1})^K.$$

The details of the proof are left to the reader.

G. Giant Aquarium

An aquarium is filled with water. The bottom of the aquarium is an axes-aligned polyline with non-decreasing x coordinate. We can punch k holes in the bottom, the water will then flow out where it can. Find the maximal amount of water we can get to flow out.

G. Giant Aquarium

Consider the (arbitrary) highest horizontal segment of the bottom. All the water located to the top of it (not necessarily directly above) will flow out regardless of where we punch the holes (if we punch at least one).

G. Giant Aquarium

Consider the (arbitrary) highest horizontal segment of the bottom. All the water located to the top of it (not necessarily directly above) will flow out regardless of where we punch the holes (if we punch at least one).

Now consider the parts of the aquarium to the left and to the right of the segment. If the top body of water flows out, we are basically left with two independent instances of the problem.

G. Giant Aquarium

Consider the (arbitrary) highest horizontal segment of the bottom. All the water located to the top of it (not necessarily directly above) will flow out regardless of where we punch the holes (if we punch at least one).

Now consider the parts of the aquarium to the left and to the right of the segment. If the top body of water flows out, we are basically left with two independent instances of the problem.

When we build the whole decomposition, the problem is reduced to the following:

There is a weighted rooted tree. Choose k disjoint vertical paths so that the total weight of covered vertices is largest possible.

(Vertex weights correspond to the top water body volume in the corresponding subproblem.)

G. Giant Aquarium

How to build the decomposition quickly?

G. Giant Aquarium

How to build the decomposition quickly?

The problem is highly reminiscent of building a treap by (x, y) pairs. The standard approaches are:

G. Giant Aquarium

How to build the decomposition quickly?

The problem is highly reminiscent of building a treap by (x, y) pairs. The standard approaches are:

- Construct an RMQ structure on y's of segments. Find the highest segment, descend recursively. $O(n \log n)$.

G. Giant Aquarium

How to build the decomposition quickly?

The problem is highly reminiscent of building a treap by (x, y) pairs. The standard approaches are:

- Construct an RMQ structure on y's of segments. Find the highest segment, descend recursively. $O(n \log n)$.
- Process segments from left to right with stack. $O(n)$.

G. Giant Aquarium

How to solve the vertical path decomposition problem?

G. Giant Aquarium

How to solve the vertical path decomposition problem?

A greedy approach works. First, find the longest path in the whole tree. Erasing the path results in several disjoint subtrees; decompose them recursively.

G. Giant Aquarium

How to solve the vertical path decomposition problem?

A greedy approach works. First, find the longest path in the whole tree. Erasing the path results in several disjoint subtrees; decompose them recursively.

This can be implemented in $O(n)$ time.

G. Giant Aquarium

How to solve the vertical path decomposition problem?

A greedy approach works. First, find the longest path in the whole tree. Erasing the path results in several disjoint subtrees; decompose them recursively.

This can be implemented in $O(n)$ time.

The answer to the problem is simply taking k best paths.

H. Hospital

We are given a weighted tree, with two special vertices containing hospitals. There are a_i people living at i-th vertex. Each person takes the shortest path to one of the hospitals when needed. We are allowed to shorten some of the edges by 1 at most B times, but can not make any edge shorter than L this way. Spend the money so that:

- sum of the distances to the closest hospital for all people is minimal;
- the largest distance is minimal.

H. Hospital

How to solve the problem when there is only one hospital? Root the tree at the hospital vertex.

H. Hospital

How to solve the problem when there is only one hospital? Root the tree at the hospital vertex.

Minimizing total distance: when we shorten an edge, the total distance decreases by s_i — total number of people in the subtree. Thus each edge allows us to perform $\max(0, w_i - L)$ operations that bring profit s_i. Now we sort the operations by their profit and perform them while we can.

H. Hospital

How to solve the problem when there is only one hospital? Root the tree at the hospital vertex.

Minimizing total distance: when we shorten an edge, the total distance decreases by s_i — total number of people in the subtree. Thus each edge allows us to perform $\max(0, w_i - L)$ operations that bring profit s_i. Now we sort the operations by their profit and perform them while we can.

Minimizing maximal distance: binary search on the answer. For each vertex of the tree store the longest path down. Naturally, to get rid of all paths greater than x it makes sense to shorten higher edges first. The total cost of satisfying the restriction can be found with a single DFS.

H. Hospital

How to solve the problem when there is only one hospital? Root the tree at the hospital vertex.

Minimizing total distance: when we shorten an edge, the total distance decreases by s_i — total number of people in the subtree. Thus each edge allows us to perform $\max(0, w_i - L)$ operations that bring profit s_i. Now we sort the operations by their profit and perform them while we can.

Minimizing maximal distance: binary search on the answer. For each vertex of the tree store the longest path down. Naturally, to get rid of all paths greater than x it makes sense to shorten higher edges first. The total cost of satisfying the restriction can be found with a single DFS.

The total complexity is $O(n \log A)$, where A is the total length of all edges.

H. Hospital

Now we have two hospitals. Consider the unique path between them.

H. Hospital

Now we have two hospitals. Consider the unique path between them.

Each path from a vertex to a hospital starts by reaching a vertex on this path, and then deciding which hospital to visit.

H. Hospital

Now we have two hospitals. Consider the unique path between them.

Each path from a vertex to a hospital starts by reaching a vertex on this path, and then deciding which hospital to visit.

If we travel along this path from the hospital A to the hospital B, then at some point it becomes faster to go to B instead of A. It follows that one of the edges in the A–B path will be useless.

H. Hospital

Now we have two hospitals. Consider the unique path between them.

Each path from a vertex to a hospital starts by reaching a vertex on this path, and then deciding which hospital to visit.

If we travel along this path from the hospital A to the hospital B, then at some point it becomes faster to go to B instead of A. It follows that one of the edges in the A–B path will be useless.

Try all possible options to erase one of the edges in the path. We are now faced with two independent instances of the problem (with the shared budgets and aggregated metrics). We can now use virtually the same approaches as in the case of a single hospital.

H. Hospital

Now we have two hospitals. Consider the unique path between them.

Each path from a vertex to a hospital starts by reaching a vertex on this path, and then deciding which hospital to visit.

If we travel along this path from the hospital A to the hospital B, then at some point it becomes faster to go to B instead of A. It follows that one of the edges in the A–B path will be useless.

Try all possible options to erase one of the edges in the path. We are now faced with two independent instances of the problem (with the shared budgets and aggregated metrics). We can now use virtually the same approaches as in the case of a single hospital.

The complexity is $O(n^2 \log A)$.

I. ICPC and Bulbs

There is a rotateable tile in each of the puzzle board cells. Each tile contains a lightbulb, a wire connecting two sides of the tile, or nothing. Rotate all tiles so that the two lightbulbs are connected, and all wires are used.

I. ICPC and Bulbs

First, try all possible cases of rotating the lightbulbs. Process all the remaining cells in row-major order, that is, from top row to bottom, and from left to right in each row.

I. ICPC and Bulbs

First, try all possible cases of rotating the lightbulbs. Process all the remaining cells in row-major order, that is, from top row to bottom, and from left to right in each row.

We claim that there is at most one possible rotation of each tile that agrees with everything we already determined.

I. ICPC and Bulbs

First, try all possible cases of rotating the lightbulbs. Process all the remaining cells in row-major order, that is, from top row to bottom, and from left to right in each row.

We claim that there is at most one possible rotation of each tile that agrees with everything we already determined.

Indeed, when we reach a cell, we must know (by induction hypothesis) if a wire enters the cell from the left side and from the top side.

I. ICPC and Bulbs

First, try all possible cases of rotating the lightbulbs. Process all the remaining cells in row-major order, that is, from top row to bottom, and from left to right in each row.

We claim that there is at most one possible rotation of each tile that agrees with everything we already determined.

Indeed, when we reach a cell, we must know (by induction hypothesis) if a wire enters the cell from the left side and from the top side.

If we have a q cell, this information determines the rotation uniquely.

I. ICPC and Bulbs

First, try all possible cases of rotating the lightbulbs. Process all the remaining cells in row-major order, that is, from top row to bottom, and from left to right in each row.

We claim that there is at most one possible rotation of each tile that agrees with everything we already determined.

Indeed, when we reach a cell, we must know (by induction hypothesis) if a wire enters the cell from the left side and from the top side.

If we have a `q` cell, this information determines the rotation uniquely.

If we have a `l` cell, either the rotation is unique, or we have a conflict in any case.

I. ICPC and Bulbs

If we've processed all the tiles without fail, we must also check that the bulbs are reachable from each other, and all wires take part in the path.

I. ICPC and Bulbs

If we've processed all the tiles without fail, we must also check that the bulbs are reachable from each other, and all wires take part in the path.

Complexity is $O(nm)$.

J. Jokes of Loki

We are trying to guess a number x by asking queries "is $x \leqslant q$?". We are allowed to make at most a_x queries. Can we win, and what are the possible options for the first question?

J. Jokes of Loki

Suppose that $x \in [l, r]$. If $l = r$, no more questions are needed. Otherwise, we choose m and ask if $x \leqslant m$; the possible scenarios are to proceed to segments $[l, m]$ and $[m + 1, r]$.

J. Jokes of Loki

Suppose that $x \in [l, r]$. If $l = r$, no more questions are needed. Otherwise, we choose m and ask if $x \leqslant m$; the possible scenarios are to proceed to segments $[l, m]$ and $[m + 1, r]$.

For a certain strategy of choosing m we can build a binary rooted tree of all scenarios. The number of questions to guess the number x is exactly the *depth* of the leaf $[x, x]$, that is, the distance to the root.

J. Jokes of Loki

Suppose that $x \in [l, r]$. If $l = r$, no more questions are needed. Otherwise, we choose m and ask if $x \leqslant m$; the possible scenarios are to proceed to segments $[l, m]$ and $[m + 1, r]$.

For a certain strategy of choosing m we can build a binary rooted tree of all scenarios. The number of questions to guess the number x is exactly the *depth* of the leaf $[x, x]$, that is, the distance to the root.

Thus, we can win the game iff we can construct a tree with n leaves with constraints on their respective depths.

J. Jokes of Loki

Proposition

If d_x is the depth of a leaf x in a binary rooted tree, then $\sum_x 2^{-d_x} = 1$.

Proof: induction.

J. Jokes of Loki

Proposition

If d_x is the depth of a leaf x in a binary rooted tree, then $\sum_x 2^{-d_x} = 1$.

Proof: induction.

It follows that if $\sum_x 2^{-a_x} > 1$, then winning is impossible.

J. Jokes of Loki

Proposition

If d_x is the depth of a leaf x in a binary rooted tree, then $\sum_x 2^{-d_x} = 1$.

Proof: induction.

It follows that if $\sum_x 2^{-a_x} > 1$, then winning is impossible.

Turns out that when a_x are monotonous, then the converse holds: if $\sum_x 2^{-a_x} \leqslant 1$, then we can construct an appropriate tree.

J. Jokes of Loki

Proposition

If d_x is the depth of a leaf x in a binary rooted tree, then $\sum_x 2^{-d_x} = 1$.

Proof: induction.

It follows that if $\sum_x 2^{-a_x} > 1$, then winning is impossible.

Turns out that when a_x are monotonous, then the converse holds: if $\sum_x 2^{-a_x} \leqslant 1$, then we can construct an appropriate tree.

Sketch of a proof: just place each leaf as deep as possible, observe that everything is fine.

J. Jokes of Loki

How can we check the condition $\sum_x 2^{-a_x} \leqslant 1$ precisely? Store the number of repetitions for each x, then perform the carries in a binary representation of the sum.

J. Jokes of Loki

How can we check the condition $\sum_x 2^{-a_x} \leqslant 1$ precisely? Store the number of repetitions for each x, then perform the carries in a binary representation of the sum.

How can we determine if x is a winning move? Applying our criterion to both halves, we must have $\sum_{z \leqslant x} 2^{-a_z} \leqslant 1/2$ and $\sum_{z > x} 2^{-a_z} \leqslant 1/2$.

J. Jokes of Loki

How can we check the condition $\sum_x 2^{-a_x} \leqslant 1$ precisely? Store the number of repetitions for each x, then perform the carries in a binary representation of the sum.

How can we determine if x is a winning move? Applying our criterion to both halves, we must have $\sum_{z \leqslant x} 2^{-a_z} \leqslant 1/2$ and $\sum_{z > x} 2^{-a_z} \leqslant 1/2$.

It follows that winning moves form a subsegment of $[1, n]$. The endpoints can be determined by binary search, or just careful sequential processing of the entries from both sides of the sequence.

J. Jokes of Loki

How can we check the condition $\sum_x 2^{-a_x} \leqslant 1$ precisely? Store the number of repetitions for each x, then perform the carries in a binary representation of the sum.

How can we determine if x is a winning move? Applying our criterion to both halves, we must have $\sum_{z \leqslant x} 2^{-a_z} \leqslant 1/2$ and $\sum_{z > x} 2^{-a_z} \leqslant 1/2$.

It follows that winning moves form a subsegment of $[1, n]$. The endpoints can be determined by binary search, or just careful sequential processing of the entries from both sides of the sequence.

When then answer is inf? In this case at least one our winning option is to waste a question, so $\sum_x 2^{-a_x} \leqslant 1/2$.

K. Kitchen-based Economy

Given cactus - undirected connected graph with N vertices and M edges with no edge lie in more than one simple cycle. For each vertex v, there exist numbers a_v and b_v of people and restaurants in the vertex, respectively. Each man choose a restaurant with equal probability and go there using one of the shortests paths (with equal probability). The problem is to find all edges with maximum expected value of people going through it.

K. Kitchen-based Economy

Run DFS on our graph and find all cycles; it will be simple because any cycle will consist on some path in DFS-tree from ancestor to descendant and one back edge. Write these cycles in vector *cycles* (in fact, vector of vectors), for each vertex write all numbers cycles vertex lies in and a position of it's number, and for each edge write a number of cycle it lies in. Also, we will consider each bridge as cycle of length 2.

Consider some cycle $v_0, v_1, \ldots, v_{k-1}$, $k \geqslant 2$. If we delete all edges of this cycle, graph will decay to components of connection $C_0, C_1, \ldots, C_{k-1}$ containing vertices $v_0, v_1, \ldots, v_{k-1}$ respectively. Let $csuma_i$ and $csumb_i$ be summary numbers of people and restaurants in component C_i.

K. Kitchen-based Economy

ByteDance Moscow Workshops

Find an expected number $E_{v_{k-1}v_0}$ of people going through edge $e = (v_{k-1}, v_0)$ strictly in this direction. For each concrete man m, probability p_{em} of visiting e depends only on number of C_i of start and finish of his journey; let it be C_s and C_f, respectively. Any path of our man goes through vertices v_s and v_f and choose shortest of two paths. if $s \leqslant f$, then $p_{em} = 0$; otherwise $p_{em} = 1$ if $2 * (f + k - 1 - s) < k$, $p_{em} = 0.5$ if $2 * (f + k - 1 - s) = k$ and $p_{em} = 0$ if $2 * (f + k - 1 - s) > k$.

It means that $E_{v_{k-1}v_0} B * F_{v_{k-1}v_0}$,
$F_{v_{k-1}v_0} = (2 * \sum\limits_{0 \leqslant s,f < k; f+k-1-s<k} (suma_s * sumb_f) + \sum\limits_{0 \leqslant s,f < k; f+k-1-s=k}) * (csuma_s * csumb_f))$; here B is a positive constant (equal to $0.5/(b_1 + b_2 + \cdots + b_n)$) which doesn't depend on concrete edge).

K. Kitchen-based Economy

Then, the $O(n^3)$ solution is:

1) find all cycles in the graph;

2) run second $DFS(v, parentCycleNumber)$ on vertices and cycles; for each vertex v, find $suma_v$ and $sumb_v$ - sum of a-s and b-s for all vertices connected with v by path which doesn't contain edges of cycle $parentCycleNumber$. To launch this DFS, just call $DFS(0, -1)$; then $DFS(v, parentCycleNumber)$ will go through all cycles contains v, except $parentCycleNumber$. For each such cycle C and vertices u_i of it, run $DFS(u_i, C)$ and sum up calculated $suma_{u_i}$ and $sumb_{u_i}$ to $suma_v$ and $sumb_v$. At the end, add a_v to $suma_v$ and b_v to $sumb_v$ and finish DFS;

K. Kitchen-based Economy

3) Having *suma*-s and *sumb*-s, try to find an answer for each edge. Let $C = (v_0, v_1, v_2, \ldots, v_{k-1})$ be a cycle, and v_0 is a vertex which was first visited by *DFS* on second step. Then, $csuma_i = suma_{v_i}, csumb_i = sumb_{v_i}$ for $i = 1, 2, \ldots, k-1$; then, $suma_0 = suma - \sum_{i=1}^{k-1} csuma_i$, $sumb_0 = sumb - \sum_{i=1}^{k-1} csumb_i$; here, *suma* and *sumb* are full number of people and restaurants in the whole graph.

4) Having *csuma*-s and *csumb*-s, find each F in $O(k^2)$. So, our algorithm works in $O(m^3)$ time.

But $O(m^3)$ is too slow for us ($m \leqslant 200000$). How to improve it?

K. Kitchen-based Economy

The only thing we should optimize is calculating F-s on cycle C - other parts of algorithm works in linear time.

First of all, calculate prefix sums on arrays *suma* and *sumb*; then, for any path on cycle (containing edge (v_{k-1}, v_0) or not) we can find sum of *suma*-s and *sumb*-s of it's vertices in linear time.

Then, suppose that $k = 2 * l$ (case $k = 2 * l + 1$ is very similar and simpler). Then,

$F_{v_{k-1},v_0} = 2 * (suma_{k-l+1} * sumb_0 + suma_{k-l+2} * (sumb_0 + sumb_1) + \cdots + suma_{k-1} * (sumb_0 + sumb_1 + \cdots + sumb_{k-2})) + (suma_{k-l} * sumb_0 + suma_{k-l+1} * sumb_1 + \cdots + suma_{k-1} * sumb_{l-1})$.

K. Kitchen-based Economy

Hence, F_{v_{k-1},v_0} can be calculated in $O(k)$ using partial sums, and now we can solve the problem in $O(m^2)$. The idea which gives us a linear solution is that difference between neighbouring F-s can be calculated in $O(1)$, because, for example, $F_{v_0,v_1} - F_{v_{k-1},v_0} = 2 * suma_0 * (sumb_1 + sumb_2 + \cdots + sumb_{l-1}) + suma_0 * sumb_l - 2 * sumb_0 * (suma_{k-1} + suma_{k-2} + \cdots + suma_{k-l+1}) - sumb_0 * suma_{k-l}$ (to move to F_{v_0,v_1} from F_{v_{k-1},v_0}, we just added short paths starting in v_0 and subtract short paths finishing in v_0). So, we can calculate F_{v_{k-1},v_0} in linear time and then find each new $F_{v_i,v_{i+1}}$ from previous one in $O(1)$ with partial sums. The complexity of our last version of algorithm is $O(m)$.

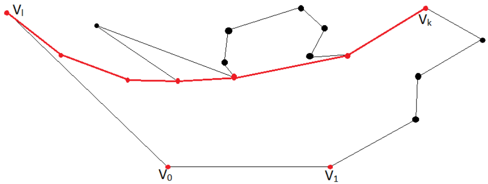
L. Light Sources

Given a polygon with N vertices $v_0, v_1, \ldots, v_{n-1}$; whole polygon is illuminated by at least one of light sources in vertices v_0 and v_1, and also two number a and b. The problem is to find lexicographically minimal path from v_a to v_b.

L. Light Sources

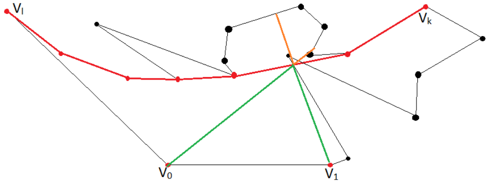
Replace number of v_0 from 0 to n; so, we have vertices $v_1, v_2, \cdots, v_n$.

Then, suppose that $a < b$. Then, go through vertices from v_a to v_b using stack and build something similar to building lower part of convex hull in Andrew's algorithm:



L. Light Sources

It seems intuitively obvious and can be proven that polyline we've built is an optimal path among paths lying non-higher than points $v_a, v_{a+1}, \ldots, v_b$; this polyline must lie in the polygon, because otherwise, we go to contradiction with constraint of visibility:



L. Light Sources

So, our algorithm works in a linear time. A last small detail is that three points on the same line is bad for optimal way, and this case should be considered in implementation.