Training Contest 3 Editorial

May 2, 2021

Mikhail Tikhomirov, Gleb Evstropov & Filipp Rukhovich

Bytedance-Moscow Workshops Training Camp, 2021

A. Astronomer's Nightmare

Given convex polyhedrons SA and SB set as a convex hulls of sets of 3D points $PA = (pa_1, pa_2, ..., pa_n)$ and $PB = (pb_1, pb_2, ..., pb_m)$ and their 3D speeds vA and vB, the task is to understand is there any moment of time in which polyhedrons collided or will collide or not. It's guaranteed that polyhedrons are not intersecting or touching at the initial time.

A. Astronomer's Nightmare

Pass into reference system in which PB doesn't move. To do that, we assign $vA := vA - vB$, $vB := 0$.

Then, if $vA = (0, 0, 0)$ then the answer is NO.

Otherwise, suppose that $vA.z \neq 0$ (if $vA.z = 0$, but, for example, $vA.y \neq 0$, then swap y-s and z-s in all points and speeds of input data; so do if $vA.z = vA.y = 0$, but $vA.x \neq 0$).

Obviously, SA and SB collided or will collide if and only if there are points $qA \in SA$ and $qb \in SB$ such that vector $qb - qa$ is collinear with vA. So, if SA' and SB' are projections SA and SB on the plane Oxy by the line parallel to vA then answer is YES iff SA' and SB' are intersecting.

A. Astronomer's Nightmare

It can be easily proved that to find SA', we should project all points of PA on the plane Oxy and find 2D-convex hull of projections by, for example, Graham's algorithm; so do SB'. So, SA' and SB' are two convex polygons, and we are just to check of they are intersecting or not. It can be done in $O(n+m)$ by, for example, calculating a Minkowsky sum of SA' and $-SB'$ ($-SB'$ is a reflection of SB across point $(0,0)$) and check if $(0,0)$ lies in this sum or not.

The complexity of described solution is $O(n\ log\ n)$ (because of calculating a plane convex hull).

B. Build Them All!

Given numbers T, L and L numbers, we are to build B-tree of degree T with L number as keys in such a way that each node should contain exactly $T - 1$ keys.

B. Build Them All!

If you know the height h of the tree to build then the leaves will be at the level h (assuming that the root is located at level 1). Then it is possible to construct the tree in in-order placing $T - 1$ keys in each node and assigning to each node T children. The list must be previously ordered and the values taken from left to right.

The height of the tree must be calculated. As all nodes have exactly $T - 1$ keys and T children, then the number of keys to store in the BTree (obtained by adding the keys for levels) is $n = (t-1)(t^0 + t^1 + \cdots + t^{h-1})$ which corresponds to a geometric series, then is possible to calculate the height of the tree from the number n of elements in the list of keys. If the value obtained for the height is not an integer then it is impossible to build the B-Tree with the number n of keys offered.

C. Count The Rebuildings

We are choosing n points in a 1×1 square at random.

Then we add these points one by one to form the cartesian tree.

Find the expected number of times, that the added point became a non-leaf vertex just after addition.

C. Count The Rebuildings

Solution

- Consider we added $n-1$ points and adding n-th one

C. Count The Rebuildings

Solution

- Consider we added $n-1$ points and adding n-th one
- So there's a $\frac{1}{n}$ probability for n-th added point to end up being k-th if ordered by x-coordinate, for each k

C. Count The Rebuildings

Solution

- Consider we added $n - 1$ points and adding n-th one
- So there's a $\frac{1}{n}$ probability for n-th added point to end up being k-th if ordered by x-coordinate, for each k
- If k-th point has greater y-coordinate than $(k - 1)$-th or $(k + 1)$-th point, then it contains it in it's subtree

C. Count The Rebuildings

Solution

- Consider we added $n-1$ points and adding n-th one
- So there's a $\frac{1}{n}$ probability for n-th added point to end up being k-th if ordered by x-coordinate, for each k
- If k-th point has greater y-coordinate than $(k-1)$-th or $(k+1)$-th point, then it contains it in it's subtree
- Contrary is also true, if it's contained in a subtree, then obviously it has less y-coordinate

C. Count The Rebuildings

Solution

- Consider we added $n-1$ points and adding n-th one
- So there's a $\frac{1}{n}$ probability for n-th added point to end up being k-th if ordered by x-coordinate, for each k
- If k-th point has greater y-coordinate than $(k-1)$-th or $(k+1)$-th point, then it contains it in it's subtree
- Contrary is also true, if it's contained in a subtree, then obviously it has less y-coordinate
- So k-th point is leaf if and only if its y-coordinate is both less than y-coordinates of $(k-1)$-th and $(k+1)$-th points
    - This probability is $\frac{1}{3}$, if $1 < k < n$, and is $\frac{1}{2}$, if $k = 1$ or $k = n$

C. Count The Rebuildings

Formula

- $f(n) = f(n-1) + (\frac{2}{n} \cdot \frac{1}{2} + \frac{n-2}{n} \cdot \frac{2}{3})$ and $f(1) = 0$
  - $f(n) = \sum\limits_{i=2}^{n} \frac{1}{i} + \frac{2(n-1)}{3} - \frac{4}{3} \sum\limits_{i=2}^{n} \frac{1}{i} = \frac{2n-1-\sum\limits_{i=1}^{n} \frac{1}{i}}{3}$
- To calculate $\sum\limits_{i=1}^{n} \frac{1}{i}$, precalculate every 10^6-th value of this one, then in 10^6 operations calculate the required one

D. Diophantine

Solve $n^n + n^m \equiv a \pmod{p}$.

D. Diophantine

Remember the fact that

$$a^b \ mod \ p = a^{b \ mod \ \varphi(b)} = a^{b \ mod \ (p-1)}$$

for any prime p and $0 \leqslant a, b < p$.

D. Diophantine

Remember the fact that

$$a^b \ mod \ p = a^{b \ mod \ \varphi(b)} = a^{b \ mod \ (p-1)}$$

for any prime p and $0 \leqslant a, b < p$.

As $p-1$ and p are coprime and p is up to 10^9 while we can select n up to 10^{18}, any power of n can be obtained. Thus, we have to find any solution for

$$n^x + n^m = a \ (mod \ p),$$

where we are free to pick any n and x.

D. Diophantine

Recall that a primitive root $b \not\equiv 0$ that has order $p-1$, that is, $b^x \not\equiv 1$ for $x < p-1$.

D. Diophantine

Recall that a primitive root $b \not\equiv 0$ that has order $p-1$, that is, $b^x \not\equiv 1$ for $x < p-1$.

Pick any primitive root modulo p. The equation now is:

$$b^x = (a - b^m) \ (mod \ p)$$

D. Diophantine

If $a - b^m \not\equiv 0$, we can solve for x by taking discrete logarithm. One approach is "baby-step-giant-step" (we'll place a link or an explanation here later).

D. Diophantine

If $a - b^m \not\equiv 0$, we can solve for x by taking discrete logarithm. One approach is "baby-step-giant-step" (we'll place a link or an explanation here later).

If $a - b^m \equiv 0$, there is no solution for x and we have to try another root.

D. Diophantine

If $a - b^m \not\equiv 0$, we can solve for x by taking discrete logarithm. One approach is "baby-step-giant-step" (we'll place a link or an explanation here later).

If $a - b^m \equiv 0$, there is no solution for x and we have to try another root.

How many times can this happen? There are at most m solutions to $x^m \equiv a$, therefore we will have to skip at most m roots.

D. Diophantine

If $a - b^m \not\equiv 0$, we can solve for x by taking discrete logarithm. One approach is "baby-step-giant-step" (we'll place a link or an explanation here later).

If $a - b^m \equiv 0$, there is no solution for x and we have to try another root.

How many times can this happen? There are at most m solutions to $x^m \equiv a$, therefore we will have to skip at most m roots.

To try all primitive roots, we can just try all b starting from 2 and check if b is primitive.

E. Experimental Coverings

Given n, k and $n \times n$ field, we are to find an expected proportion of vertical dominoes in some random covering of gield by fractal dominoes of order k.

E. Experimental Coverings

Let p be an expected proportion of vertical dominoes in random covering of given field by "usual" dominoes. It can be done by using a simple modification of method of dynamic programming with broken profile and calculating an array $dp[mask][x][y][vert]$; here $dp[mask][x][y][vert]$ is a number of full coverings of part of the field left to (x, y)-broken profile, $mask$ is bit mask of covered cells of the profile, and $vert$ is number of used vertical dominoes. All dp-s can be calculated in $O(2^n \cdot n^4)$, and then

$$p = \frac{\sum\limits_{vert=0}^{\lfloor n^2/2 \rfloor} (dp[2^{n+1}-1][n][n][vert] \cdot vert)}{\sum\limits_{vert=0}^{\lfloor n^2/2 \rfloor} (dp[2^{n+1}-1][n][n][vert])},$$ and p can be calculated in $O(n^2)$ with dp-s.

E. Experimental Coverings

On next step, let pv_k be an expected proportion of vertical dominoes in random vertical fractal domino of order k, and ph_k be the same for horizontal fractal domino of order k. Then, $pv_0 = 1$, $ph_0 = 0$, and pv_1 and ph_1 can be calculated in the same way as p (and with the same complexity).

Also, it's obvious from general considerations of theory of probability that:

1. $ans = p \cdot pv_k + (1 - p) \cdot ph_l$, where ans is the answer to the problem;
2. for each l, $\begin{pmatrix} pv_{l+1} \\ ph_{l+1} \end{pmatrix} = A \begin{pmatrix} pv_l \\ ph_k \end{pmatrix}$, $A = \begin{pmatrix} pv_1 & 1 - pv_1 \\ ph_1 & 1 - ph_1 \end{pmatrix}$

It means that $\begin{pmatrix} pv_k \\ ph_k \end{pmatrix} = A^k \begin{pmatrix} pv_0 \\ ph_0 \end{pmatrix}$; so, p, pv_k and ph_k can be calculated in $O(\log k)$ with binary exponentation of A and having

E. Experimental Coverings

The complexity of our solution is $O(2^n \cdot n^4 + \log k)$.

F. Fantastic Mountains

We are given several special cells in the rectangular grid. The number of neighbours of each cell is the number of directions (up, down, left, and right) such that there is a special cell in that direction. Count the number of cells with 0, 1, 2, 3, and 4 neighbours.

F. Fantastic Mountains

If c and c' are special cells sharing a row/column, the number of neighbours for all cells strictly between c and c' increases by 2.

F. Fantastic Mountains

If c and c' are special cells sharing a row/column, the number of neighbours for all cells strictly between c and c' increases by 2.

If c is, say, the leftmost special cell in its row, the number of neighbours for all cells to the left of c increases by 1.

F. Fantastic Mountains

If c and c' are special cells sharing a row/column, the number of neighbours for all cells strictly between c and c' increases by 2.

If c is, say, the leftmost special cell in its row, the number of neighbours for all cells to the left of c increases by 1.

These can be viewed as 2D range updates that we are required to process offline.

F. Fantastic Mountains

Let us do a sweepline over the grid. Store an RSQ structure that for current x and each y remembers the value in the cell (x, y), and also the number of cells with 0, 1, 2, 3, and 4 respectively.

F. Fantastic Mountains

Let us do a sweepline over the grid. Store an RSQ structure that for current x and each y remembers the value in the cell (x, y), and also the number of cells with 0, 1, 2, 3, and 4 respectively.

Each update is then reduced to 2 range updates to the one-dimensional structure, that are easy to process with lazy propagation.

F. Fantastic Mountains

Let us do a sweepline over the grid. Store an RSQ structure that for current x and each y remembers the value in the cell (x, y), and also the number of cells with 0, 1, 2, 3, and 4 respectively.

Each update is then reduced to 2 range updates to the one-dimensional structure, that are easy to process with lazy propagation.

We have to compress the coordinates since the grid is large.

F. Fantastic Mountains

Let us do a sweepline over the grid. Store an RSQ structure that for current x and each y remembers the value in the cell (x, y), and also the number of cells with 0, 1, 2, 3, and 4 respectively.

Each update is then reduced to 2 range updates to the one-dimensional structure, that are easy to process with lazy propagation.

We have to compress the coordinates since the grid is large.

The complexity is $O(n \log n)$.

G. Guard's Practice

Several targets (points) are given in the plane. Several soldiers shoot bullets at targets. All soldiers are standing on a common line so that all targets are to the same side of the line, and shoot towards targets. For each soldier, determine if he hit a target or space between two targets.

G. Guard's Practice

It follows from the restrictions that we are free to extend the bullet trace backwards to form a full line l.

G. Guard's Practice

It follows from the restrictions that we are free to extend the bullet trace backwards to form a full line l.

Also, the answer is "YES" iff there is a target on the line l, or there are two targets on opposite sides of the line.

G. Guard's Practice

It follows from the restrictions that we are free to extend the bullet trace backwards to form a full line l.

Also, the answer is "YES" iff there is a target on the line l, or there are two targets on opposite sides of the line.

Recall that any line's equation is $f(x, y) = ax + by - c = 0$. One approach to answering queries above would be finding two targets p and q that maximize and minimize f respectively. The answer is then "NO" iff $f(p)$ and $f(q)$ have the same sign (and none of them is zero).

G. Guard's Practice

How to find $\max_p f(p)$ quickly? Obviously, we are only interested in p that are convex hull vertices.

G. Guard's Practice

How to find $\max_p f(p)$ quickly? Obviously, we are only interested in p that are convex hull vertices.

The problem is then reduced to performing binary (or ternary) search on the convex hull border carefully.

G. Guard's Practice

How to find $\max_p f(p)$ quickly? Obviously, we are only interested in p that are convex hull vertices.

The problem is then reduced to performing binary (or ternary) search on the convex hull border carefully.

Here is one approach: select any vertex v on the convex hull, draw a line parallel to l through v. With binary search find the other point u of intersection between l and the convex hull border (it may belong to a side).

G. Guard's Practice

How to find $\max_p f(p)$ quickly? Obviously, we are only interested in p that are convex hull vertices.

The problem is then reduced to performing binary (or ternary) search on the convex hull border carefully.

Here is one approach: select any vertex v on the convex hull, draw a line parallel to l through v. With binary search find the other point u of intersection between l and the convex hull border (it may belong to a side).

v and u split the convex hull into two parts where f is unimodal (has one extreme point), hence we can perform a search on each of these parts.

G. Guard's Practice

How to find $\max_p f(p)$ quickly? Obviously, we are only interested in p that are convex hull vertices.

The problem is then reduced to performing binary (or ternary) search on the convex hull border carefully.

Here is one approach: select any vertex v on the convex hull, draw a line parallel to l through v. With binary search find the other point u of intersection between l and the convex hull border (it may belong to a side).

v and u split the convex hull into two parts where f is unimodal (has one extreme point), hence we can perform a search on each of these parts.

The complexity is $O((n+m)\log n)$.

H. Hardware For Cookies

We have n identical cookie-producing machines, each of them is set on a certain number of cookies to produce. We do m steps: choose a subset of machines that will produce the number of cookies as close to G as possible (out of them equiprobably), produce the cookies, and reset these machines randomly. Find the average number of cookies produced.

H. Hardware For Cookies

The current state is described by values of m — the number of remaining steps, and S — aggregated current states of all machines. Let $f(S, m)$ be the answer to the problem.

H. Hardware For Cookies

The current state is described by values of m — the number of remaining steps, and S — aggregated current states of all machines. Let $f(S, m)$ be the answer to the problem.

Let $T_1, \ldots, T_s$ be all subsets of machines in state S that produce cookies as close to G as possible. We can now compute

$$f(S, m) = \frac{1}{s}\sum_{i=1}^{s}(prod(S, T_i) + avgf(S, T_i, m-1)),$$

where $prod(S, T_i)$ is how many cookies are produced in state S by machines in T_i, and $avgf(S, T_i, m-1)$ is the average of $f(S', m)$, where S' is obtained from S be randomly resetting machines in T_i.

H. Hardware For Cookies

Since n and k are very small, we can explicitly generate all S, and for each S generate all T_i, and then apply DP to count the answer. Still, we have few optimizations to do.

H. Hardware For Cookies

Since n and k are very small, we can explicitly generate all S, and for each S generate all T_i, and then apply DP to count the answer. Still, we have few optimizations to do.

First, m is too large to count $f(S, m)$ with simple DP. We instead note that vector $v_m = (f(S_1, m), \ldots, f(S_t, m), 1)^T$ is obtained from v_{m-1} by a matrix multiplication: $v_m = AV_{m-1}$, therefore $v_m = A^m v_0$ and can be found in $O(t^3 \log m)$ time.

H. Hardware For Cookies

Since n and k are very small, we can explicitly generate all S, and for each S generate all T_i, and then apply DP to count the answer. Still, we have few optimizations to do.

First, m is too large to count $f(S, m)$ with simple DP. We instead note that vector $v_m = (f(S_1, m), \ldots, f(S_t, m), 1)^T$ is obtained from v_{m-1} by a matrix multiplication: $v_m = AV_{m-1}$, therefore $v_m = A^m v_0$ and can be found in $O(t^3 \log m)$ time.

Second, this still takes too long since $t = k^n > 1000$. However, we don't care about the order of the machines, hence we can group together all states that are permutations of each other. The number of states become $t = \binom{4+6-1}{4} = 126$, which is small enough.

H. Hardware For Cookies

Since n and k are very small, we can explicitly generate all S, and for each S generate all T_i, and then apply DP to count the answer. Still, we have few optimizations to do.

First, m is too large to count $f(S,m)$ with simple DP. We instead note that vector $v_m = (f(S_1,m),\dots,f(S_t,m),1)^T$ is obtained from v_{m-1} by a matrix multiplication: $v_m = AV_{m-1}$, therefore $v_m = A^m v_0$ and can be found in $O(t^3 \log m)$ time.

Second, this still takes too long since $t = k^n > 1000$. However, we don't care about the order of the machines, hence we can group together all states that are permutations of each other. The number of states become $t = \binom{4+6-1}{4} = 126$, which is small enough.

The complexity is $O(n^3)$.

I. Interesting Matching

There are two sets of integers A and B. Find a collection of pairs (a, b) with $a \in A$, $b \in B$ such that each number in each set is present in at least one of the pairs (on its respective position), and the sum $\sum |a - b|$ over all pairs is smallest possible.

I. Interesting Matching

Let us view the set of pairs as a bipartite graphs with vertices being the integers on respective sides.

I. Interesting Matching

Let us view the set of pairs as a bipartite graphs with vertices being the integers on respective sides.

Obviously, there is an optimal graph that doesn't contain any cycles, since any edge of the cycle can be erased.

I. Interesting Matching

Let us view the set of pairs as a bipartite graphs with vertices being the integers on respective sides.

Obviously, there is an optimal graph that doesn't contain any cycles, since any edge of the cycle can be erased.

Further, observe that there is an optimal answer such that the graph doesn't have a path of length 3. Indeed, suppose there is a (vertex-disjoint) path (u, v), (v, w), (w, x). But v and w are covered by the first and the last edge, hence the second edge is not needed.

I. Interesting Matching

Let us view the set of pairs as a bipartite graphs with vertices being the integers on respective sides.

Obviously, there is an optimal graph that doesn't contain any cycles, since any edge of the cycle can be erased.

Further, observe that there is an optimal answer such that the graph doesn't have a path of length 3. Indeed, suppose there is a (vertex-disjoint) path (u, v), (v, w), (w, x). But v and w are covered by the first and the last edge, hence the second edge is not needed.

Any connected component of a graph with properties above is a tree of diameter 2, that is, a *star graph*, i.e. a single vertex with several attached leaves.

I. Interesting Matching

We now claim that the solution is roughly the same as the solution to the problem F (*Fire Engines*) from Day 3, with the following addition:

For each vertex consider two more transitions: try to match the vertex with closest points to the left and to the right from the other set.

I. Interesting Matching

The reason is roughly as follows: Consider an optimal solution. Take any star subgraph that is not a single edge, that is, a vertex v is connected to $u_1, \ldots, u_k$ from the other half.

I. Interesting Matching

The reason is roughly as follows: Consider an optimal solution. Take any star subgraph that is not a single edge, that is, a vertex v is connected to $u_1, \ldots, u_k$ from the other half.

We can find an extreme point u_i such that v is closest to u_i in some direction (otherwise we are free to rematch u_i). This particular matching of u_i will be considered by our algorithm, hence we erase u_i from the graph.

I. Interesting Matching

The reason is roughly as follows: Consider an optimal solution. Take any star subgraph that is not a single edge, that is, a vertex v is connected to $u_1, \ldots, u_k$ from the other half.

We can find an extreme point u_i such that v is closest to u_i in some direction (otherwise we are free to rematch u_i). This particular matching of u_i will be considered by our algorithm, hence we erase u_i from the graph.

Repeat until all star graphs are single edges. The rest of the answer will be obtained by "segments" transitions, since the problem is now a partial case of the problem F.

J. Junctions, Roads and Racing

In an undirected graph there are highways and regular roads. All vertices are connected via highways only. Find the maximal set of edges that includes all highways and allows for an Eulerian tour.

J. Junctions, Roads and Racing

We want to find a subset of edges, such that:

1. all highways are included;
2. There exists an Eulerian cycle visiting all its edges.

J. Junctions, Roads and Racing

We want to find a subset of edges, such that:

1. all highways are included;
2. There exists an Eulerian cycle visiting all its edges.

Reformulate the problem using criteria of existence of Eulerian cycle. We would like to find the subset of edges A, such that:

1. None of them is a highway;
2. Set $E \backslash A$ forms one connected component;
3. $degree(v)$ in graph $E \backslash A$ is even for every v.

As it's possible to get from each node to any other node using only highways, condition 2 will follow from condition 1.

J. Junctions, Roads and Racing

Now, consider the graph of *all* roads. Some nodes have odd degree and are marked as interesting. We should find a set of paths, such that:

1. there is exactly one path endpoint at every interesting node;
2. no edge is used by two paths;
3. the total length of all paths is minimum possible.

J. Junctions, Roads and Racing

Now, consider the graph of *all* roads. Some nodes have odd degree and are marked as interesting. We should find a set of paths, such that:

1. there is exactly one path endpoint at every interesting node;
2. no edge is used by two paths;
3. the total length of all paths is minimum possible.

It turns out that condition 2 follows from condition 1. If some edge is used by two paths, lets say one connects u_1 with v_1, while the other one connects u_2 and v_2, we can swap the beginning and the tailing parts connecting u_1 with u_2 and v_1 with v_2.

J. Junctions, Roads and Racing

Now, consider the graph of *all* roads. Some nodes have odd degree and are marked as interesting. We should find a set of paths, such that:

1. there is exactly one path endpoint at every interesting node;
2. no edge is used by two paths;
3. the total length of all paths is minimum possible.

It turns out that condition 2 follows from condition 1. If some edge is used by two paths, lets say one connects u_1 with v_1, while the other one connects u_2 and v_2, we can swap the beginning and the tailing parts connecting u_1 with u_2 and v_1 with v_2.

Now, we only need to distribute interesting nodes in pairs in order to minimize the total length of all paths used. This is similar to finding minimum weight matching in general graph.

J. Junctions, Roads and Racing

Solution 1. Implement weighted primal-dual Edmonds algorithm to find minimum weight perfect matching in polynomial time. Definitely not the algorithm you want to implement during the contest (actually, not only during the contest).

J. Junctions, Roads and Racing

Solution 1. Implement weighted primal-dual Edmonds algorithm to find minimum weight perfect matching in polynomial time. Definitely not the algorithm you want to implement during the contest (actually, not only during the contest).

Solution 2. Randomly split the set of nodes in two and solve bipartite graph minimum weight perfect matching in $O(n^3)$ time. The probability to find optimum answer is at least $\frac{1}{2^{n/2}}$, that is the total running time is $O(n^3 2^{n/2} \log \frac{1}{eps})$, where *eps* is the desired probability error.

J. Junctions, Roads and Racing

Solution 1. Implement weighted primal-dual Edmonds algorithm to find minimum weight perfect matching in polynomial time. Definitely not the algorithm you want to implement during the contest (actually, not only during the contest).

Solution 2. Randomly split the set of nodes in two and solve bipartite graph minimum weight perfect matching in $O(n^3)$ time. The probability to find optimum answer is at least $\frac{1}{2^{n/2}}$, that is the total running time is $O(n^3 2^{n/2} \log \frac{1}{eps})$, where *eps* is the desired probability error.

Solution 3. Optimize backtracking or bitmask dynamic programming.

K. Keep The Angle Maximal

Given points on the plane p_1, p_2, ..., p_n, no three points are collinear.

For each i and j, find k, so that angle $p_i p_k p_j$ is maximized.

K. Keep The Angle Maximal

Observation

- Let's make inversion of point p_i

K. Keep The Angle Maximal

Observation

- Let's make inversion of point p_i
- Angle $p_i p_k p_j$ is maximized if after inversion angle $p_i p_j p_k$ is minimized
    - It's because this two angles are equal, but why?

K. Keep The Angle Maximal

Observation

- Let's make inversion of point p_i
- Angle $p_i p_k p_j$ is maximized if after inversion angle $p_i p_j p_k$ is minimized
  - It's because this two angles are equal, but why?
  - Consider triangle ABC, let's make inversion of point A

K. Keep The Angle Maximal

Observation

- Let's make inversion of point p_i
- Angle $p_i p_k p_j$ is maximized if after inversion angle $p_i p_j p_k$ is minimized
  - It's because this two angles are equal, but why?
  - Consider triangle ABC, let's make inversion of point A
  - B' and C' are where B and C were mapped

K. Keep The Angle Maximal

Observation

- Let's make inversion of point p_i
- Angle $p_i p_k p_j$ is maximized if after inversion angle $p_i p_j p_k$ is minimized
  - It's because this two angles are equal, but why?
  - Consider triangle ABC, let's make inversion of point A
  - B' and C' are where B and C were mapped
  - $\mathtt{AB} \cdot \mathtt{AB}' = \mathtt{AC} \cdot \mathtt{AC}' \Rightarrow \frac{\mathtt{AB}}{\mathtt{AC}} = \frac{\mathtt{AC}'}{\mathtt{AB}'}$

K. Keep The Angle Maximal

Observation

- Let's make inversion of point p_i
- Angle $p_i p_k p_j$ is maximized if after inversion angle $p_i p_j p_k$ is minimized
  - It's because this two angles are equal, but why?
  - Consider triangle ABC, let's make inversion of point A
  - B' and C' are where B and C were mapped
  - $\mathtt{AB} \cdot \mathtt{AB}' = \mathtt{AC} \cdot \mathtt{AC}' \Rightarrow \frac{\mathtt{AB}}{\mathtt{AC}} = \frac{\mathtt{AC}'}{\mathtt{AB}'}$
  - And angle BAC equals to angle B′AC′

K. Keep The Angle Maximal

Observation

- Let's make inversion of point p_i
- Angle $p_i p_k p_j$ is maximized if after inversion angle $p_i p_j p_k$ is minimized
  - It's because this two angles are equal, but why?
  - Consider triangle ABC, let's make inversion of point A
  - B' and C' are where B and C were mapped
  - $\mathtt{AB} \cdot \mathtt{AB}' = \mathtt{AC} \cdot \mathtt{AC}' \Rightarrow \frac{\mathtt{AB}}{\mathtt{AC}} = \frac{\mathtt{AC}'}{\mathtt{AB}'}$
  - And angle BAC equals to angle B′AC′
  - Triangles ABC and AC′B′ are similar, so angles ABC and AC′B′ are equal

K. Keep The Angle Maximal

Solution

- For fixed i let's find answers for all (i, j)

K. Keep The Angle Maximal

Solution

- For fixed i let's find answers for all (i, j)
- Make an inversion and sort all the points by angle from p_i

K. Keep The Angle Maximal

Solution

- For fixed i let's find answers for all (i, j)
- Make an inversion and sort all the points by angle from p_i
- If we want to find answer for (i, j), $p_i p_j$ divides plane into two parts
  - Minimizing the angle means finding tangent from p_j to convex hull of points in one of the parts

K. Keep The Angle Maximal

Solution

- For fixed i let's find answers for all (i, j)
- Make an inversion and sort all the points by angle from p_i
- If we want to find answer for (i, j), $p_i p_j$ divides plane into two parts
  - Minimizing the angle means finding tangent from p_j to convex hull of points in one of the parts
  - Only one side of convex hull needed (visible from p_i)

K. Keep The Angle Maximal

Solution

- For fixed i let's find answers for all (i, j)
- Make an inversion and sort all the points by angle from p_i
- If we want to find answer for (i, j), $p_i p_j$ divides plane into two parts
  - Minimizing the angle means finding tangent from p_j to convex hull of points in one of the parts
  - Only one side of convex hull needed (visible from p_i)
  - It can be maintained while adding the points in "by-angle" order as in Graham algorithms

K. Keep The Angle Maximal

Solution

- For fixed i let's find answers for all (i, j)
- Make an inversion and sort all the points by angle from p_i
- If we want to find answer for (i, j), $p_i p_j$ divides plane into two parts
  - Minimizing the angle means finding tangent from p_j to convex hull of points in one of the parts
  - Only one side of convex hull needed (visible from p_i)
  - It can be maintained while adding the points in "by-angle" order as in Graham algorithms
  - It's done in linear time

K. Keep The Angle Maximal

Solution

- For fixed i let's find answers for all (i, j)
- Make an inversion and sort all the points by angle from p_i
- If we want to find answer for (i, j), $p_i p_j$ divides plane into two parts
  - Minimizing the angle means finding tangent from p_j to convex hull of points in one of the parts
  - Only one side of convex hull needed (visible from p_i)
  - It can be maintained while adding the points in "by-angle" order as in Graham algorithms
  - It's done in linear time
- Do it twice clockwise and counter-clockwise

K. Keep The Angle Maximal

Solution

- For fixed i let's find answers for all (i, j)
- Make an inversion and sort all the points by angle from p_i
- If we want to find answer for (i, j), $p_i p_j$ divides plane into two parts
  - Minimizing the angle means finding tangent from p_j to convex hull of points in one of the parts
  - Only one side of convex hull needed (visible from p_i)
  - It can be maintained while adding the points in "by-angle" order as in Graham algorithms
  - It's done in linear time
- Do it twice clockwise and counter-clockwise
- $O(n^2 \log n)$ is the time complexity

L. Look At The Sign

There are n subrectangles in an $n \times n$ matrix. We try all options to place 1 in each rectangle, and sum the determinants of resulting matrices. Determine the sign of the answer.

L. Look At The Sign

Let (x_i, y_i) be the chosen element in i-th subrectangle. Note that if elements repeat, or two distinct elements share a row or a column, the determinant is 0.

L. Look At The Sign

Let (x_i, y_i) be the chosen element in i-th subrectangle. Note that if elements repeat, or two distinct elements share a row or a column, the determinant is 0.

If follows that x's should form a permutation, as do y's. Moreover, each possible permutation of x's is compatible with each permutation of y's.

L. Look At The Sign

Let (x_i, y_i) be the chosen element in i-th subrectangle. Note that if elements repeat, or two distinct elements share a row or a column, the determinant is 0.

If follows that x's should form a permutation, as do y's. Moreover, each possible permutation of x's is compatible with each permutation of y's.

If π_x is the permutation of x's, and π_y is the permutation of y's, determinant of the matrix obtained from the pairs $(\pi_x(i), \pi_y(i))$ is equal to $\mathrm{sgn}(\pi_x \cdot \pi_y)$.

L. Look At The Sign

Let (x_i, y_i) be the chosen element in i-th subrectangle. Note that if elements repeat, or two distinct elements share a row or a column, the determinant is 0.

If follows that x's should form a permutation, as do y's. Moreover, each possible permutation of x's is compatible with each permutation of y's.

If π_x is the permutation of x's, and π_y is the permutation of y's, determinant of the matrix obtained from the pairs $(\pi_x(i), \pi_y(i))$ is equal to $\operatorname{sgn}(\pi_x \cdot \pi_y)$.

Let S_x and S_y be the sets of suitable permutations of x's and y's respectively. The answer is

$$\sum_{\pi_x \in S_x, \pi_y \in S_y} \operatorname{sgn}(\pi_x \cdot \pi_y) = \left(\sum_{\pi_x \in S_x} \operatorname{sgn}(\pi_x) \right) \cdot \left(\sum_{\pi_y \in S_y} \operatorname{sgn}(\pi_y) \right)$$

L. Look At The Sign

The problem is then reduced to two instances of a one-dimensional problem, namely: find the sum of $\operatorname{sgn}(\pi)$ over all permutations π satisfying constraints $\pi(i) \in [l_i, r_i]$.

L. Look At The Sign

The problem is then reduced to two instances of a one-dimensional problem, namely: find the sum of $\operatorname{sgn}(\pi)$ over all permutations π satisfying constraints $\pi(i) \in [l_i, r_i]$.

By definition, this is equal to determinant of a matrix C defined as

$$C_{i,j} = \begin{cases} 1 & \text{if } l_i \leqslant j \leqslant r_i, \\ 0 & \text{otherwise.} \end{cases}$$

L. Look At The Sign

The problem is then reduced to two instances of a one-dimensional problem, namely: find the sum of $\operatorname{sgn}(\pi)$ over all permutations π satisfying constraints $\pi(i) \in [l_i, r_i]$.

By definition, this is equal to determinant of a matrix C defined as

$$C_{i,j} = \begin{cases} 1 & \text{if } l_i \leqslant j \leqslant r_i, \\ 0 & \text{otherwise.} \end{cases}$$

For example, if $n = 4$, $l = (1, 3, 2, 3)$, $r = (3, 3, 4, 4)$, the matrix C looks as follows:

$$C = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

L. Look At The Sign

This determinant can be found by a simple algorithm:

L. Look At The Sign

This determinant can be found by a simple algorithm:

- Consider all rows with $l_i = 1$, if there are none, the answer is zero.

L. Look At The Sign

This determinant can be found by a simple algorithm:

- Consider all rows with $l_i = 1$, if there are none, the answer is zero.
- Out of these choose i such that r_i is minimal, and subtract row i from all rows with $l_j = 1$ (note that this is equivalent to setting $l_j = r_i + 1$).

L. Look At The Sign

This determinant can be found by a simple algorithm:

- Consider all rows with $l_i = 1$, if there are none, the answer is zero.
- Out of these choose i such that r_i is minimal, and subtract row i from all rows with $l_j = 1$ (note that this is equivalent to setting $l_j = r_i + 1$).
- There is now a unique row with $l_i = 1$, switch it with the first row. The problem is reduced to a matrix with smaller dimension.

L. Look At The Sign

This determinant can be found by a simple algorithm:

- Consider all rows with $l_i = 1$, if there are none, the answer is zero.
- Out of these choose i such that r_i is minimal, and subtract row i from all rows with $l_j = 1$ (note that this is equivalent to setting $l_j = r_i + 1$).
- There is now a unique row with $l_i = 1$, switch it with the first row. The problem is reduced to a matrix with smaller dimension.

The resulting matrix is upper-triangular, which determinant can be found as the product of diagonal entries.

L. Look At The Sign

This determinant can be found by a simple algorithm:

- Consider all rows with $l_i = 1$, if there are none, the answer is zero.
- Out of these choose i such that r_i is minimal, and subtract row i from all rows with $l_j = 1$ (note that this is equivalent to setting $l_j = r_i + 1$).
- There is now a unique row with $l_i = 1$, switch it with the first row. The problem is reduced to a matrix with smaller dimension.

The resulting matrix is upper-triangular, which determinant can be found as the product of diagonal entries.

Of course, this may take too long if implemented trivially.

L. Look At The Sign

Note that subtracting operation of our algorithm does not change connectivity of the graph with edges $(l_i, r_i + 1)$. Also, the graph must be connected in the end, it must also be connected as the start.

L. Look At The Sign

Note that subtracting operation of our algorithm does not change connectivity of the graph with edges $(l_i, r_i + 1)$. Also, the graph must be connected in the end, it must also be connected as the start.

The graph has $n + 1$ vertices and n edges, hence it is a tree.

L. Look At The Sign

Note that subtracting operation of our algorithm does not change connectivity of the graph with edges $(l_i, r_i + 1)$. Also, the graph must be connected in the end, it must also be connected as the start.

The graph has $n + 1$ vertices and n edges, hence it is a tree.

We may now perform our algorithm in a DFS-like (or BFS-like) fashion, reducing the matrix to a normal form. Each DFS run will normalize the the set of rows with l_i and $r_i + 1$ in the subtree of the current vertex x, that is, each row must contain several consequent 1's, and no column must contain more than 1.

L. Look At The Sign

Note that subtracting operation of our algorithm does not change connectivity of the graph with edges $(l_i, r_i + 1)$. Also, the graph must be connected in the end, it must also be connected as the start.

The graph has $n + 1$ vertices and n edges, hence it is a tree.

We may now perform our algorithm in a DFS-like (or BFS-like) fashion, reducing the matrix to a normal form. Each DFS run will normalize the the set of rows with l_i and $r_i + 1$ in the subtree of the current vertex x, that is, each row must contain several consequent 1's, and no column must contain more than 1.

This can be done in $O(n \log n)$ with data structures, or in $O(n)$ with a careful BFS.