Day 1.

2019-02-16

Gleb Evstropov, Artsem Zhuk, Mikhail Tikhomirov

ByteDance - Moscow Workshops ICPC Programming Camp 2019

A. Annual Beauty Contest

You are one of n judges of a competition with k participants. Each judge had his own list of perfect participants' arrangement. Judges move in turns. At each turn a judge considers all participants that have not been listed yet and names one them that is at the worst place in his list. You promote the i-th participant and you haven't made your list yet. What is the best place the i-th participant can get?

A. Annual Beauty Contest

First we consider the whole process assuming we skip all the moves. Some contestants will be named before the i-th (let's say there are a of them) and some after the i-th (there will be $k - 1 - a$ of them). Denote as x the number of turns that we have skipped before the i-th contestant was named.

You can use each of your x turns to force one of $k - 1 - a$ contestants that are better than the i-th to be kicked earlier.

A. Annual Beauty Contest

How can one show that it is not possible to do better because of some weird dependencies?

Let $p(j)$ be the index in the list of $j \ mod \ n$ judge he named at the j-th move. We can use induction to prove that any action we insert may only increase $p(j)$.

B. Chess

You are given a three-dimensional toroidal chess field of size not exceeding $4 \times 4 \times 4$. This field contains one white king and no more than 4 black kings that try to take white one.

B. Chess

The total number of possible states can be estimated as $O(S^5)$, where S is the number of field cells. That is $O(64^5)$ or $O(2^{30})$. For each state there might be 4×26 possible moves. Obviously, that is too much.

B. Chess

The total number of possible states can be estimated as $O(S^5)$, where S is the number of field cells. That is $O(64^5)$ or $O(2^{30})$. For each state there might be 4×26 possible moves. Obviously, that is too much.

The first idea is to use symmetry of toroidal field and always center it on the white king. Thus, there will be only $O(S^4)$ possible states. The more precise estimation gives us $\binom{63}{4} = 595665$, approximately $600k$. Multiply this by 2 to account who's turn it is, resulting in $1.2kk$ states.

B. Chess

The total number of possible states can be estimated as $O(S^5)$, where S is the number of field cells. That is $O(64^5)$ or $O(2^{30})$. For each state there might be 4×26 possible moves. Obviously, that is too much.

The first idea is to use symmetry of toroidal field and always center it on the white king. Thus, there will be only $O(S^4)$ possible states. The more precise estimation gives us $\binom{63}{4} = 595665$, approximately $600k$. Multiply this by 2 to account who's turn it is, resulting in $1.2kk$ states.

For $600k$ we should consider 26 moves and for other $600k$ there will be 104 moves, providing $78kk$ moves in total. Apply retrograde analysis to find the game outcome.

C. Concatenation

Concatenate all substrings of string s in lexicographical order.

Find a_i-th character for multiple a_i-s.

C. Concatenation

Imangine a trie of all suffixes of s.

C. Concatenation

Imangine a trie of all suffixes of s.

We can calculate dynamic `d[v]` — number of strings starting in node v, `h[v]` — their total length.

C. Concatenation

Imangine a trie of all suffixes of s.

We can calculate dynamic `d[v]` — number of strings starting in node v, `h[v]` — their total length.

Now we can traverse downward trie and find required charater.

Trie doesn't fit into memory limit, so we can use compressed trie.

D. Dictionary

Count number of ways to split string into multiple pieces that are sorted lexicographically.

D. Dictionary

Dynamic programming `d[l][r]` – number of ways to split prefix `s[:r]`, such that last string is `s[l:r]`.

D. Dictionary

Dynamic programming `d[l][r]` – number of ways to split prefix `s[:r]`, such that last string is `s[l:r]`.

$$\texttt{d[l][r]} = \sum_{i \text{ such that } \texttt{s[i:l]} < \texttt{s[l:r]}} \texttt{d[i][l]}.$$

D. Dictionary

Dynamic programming `d[l][r]` – number of ways to split prefix `s[:r]`, such that last string is `s[l:r]`.

$$\texttt{d[l][r]} = \sum_{i \text{ such that } \texttt{s[i:l]} < \texttt{s[l:r]}} \texttt{d[i][l]}.$$

Current complexity is $O(n^3)$. How to improve it?

D. Dictionary

Fix l.

Consider all strings `s[i:l]` and `s[l:r]` for all i and r in sorted order.

Accumulate sum of `d[i][l]`-s and add to appropriate `d[l][r]`.

E. Problem E

You are given graph of friends, check if there is two vertices without common friends.

E. Problem E

You are given graph of friends, check if there is two vertices without common friends.

What does it mean to have common friend for vertices i and j?

E. Problem E

You are given graph of friends, check if there is two vertices without common friends.

What does it mean to have common friend for vertices i and j?

that means there is k, such that:

```
a[i][k] = a[j][k] = 1
```

E. Problem E

You are given graph of friends, check if there is two vertices without common friends.

What does it mean to have common friend for vertices i and j?

that means there is k, such that:

```
a[i][k] = a[j][k] = 1
```

or, using bitwise-AND:

```
a[i][k] & a[j][k] = 1.
```

E. Problem E

Naive solution: for each triple i, j, k check that `a[i][k] & a[j][k] = 0`.

Too long — $O(n^3)$.

E. Problem E

Naive solution: for each triple i, j, k check that `a[i][k] & a[j][k] = 0`.

Too long — $O(n^3)$.

Let's use bitwise optimization!

Split `a[i][0..n-1]` into chunks of size 32, calculate and in $O(1)$ time.

E. Problem E

Naive solution: for each triple i, j, k check that `a[i][k] & a[j][k] = 0`.

Too long — $O(n^3)$.

Let's use bitwise optimization!

Split `a[i][0..n-1]` into chunks of size 32, calculate and in $O(1)$ time.

Overall $O(n^3/32)$, fits the timelimit.

F. Problem F

You are given a tree, in how many ways you can put tokens in some vertices such that distance between any of them at least c?

F. Problem F

You are given a tree, in how many ways you can put tokens in some vertices such that distance between any of them at least c?

$O(n \cdot c)$ dynamic programming!

F. Problem F

`d[v][k]` — number of ways to put tokens in subtree, rooted in v, such that minimal distance from token to v is exactly k.

F. Problem F

`d[v][k]` — number of ways to put tokens in subtree, rooted in v, such that minimal distance from token to v is exactly k.

Arrangements without tokens are put in `d[v][c]`.

F. Problem F

`d[v][k]` — number of ways to put tokens in subtree, rooted in v, such that minimal distance from token to v is exactly k.

Arrangements without tokens are put in `d[v][c]`.

Initially `d[v][0]=1`, `d[v][k]=0` for $k > 0$.

Answer:

$$\sum_{k=0}^{k=c} \texttt{d[v][k]}$$

F. Problem F

Will add subtress one by one. Let u will be child of v.

`d_new[v]` – new value of dynamic.

F. Problem F

Will add subtress one by one. Let u will be child of v.

`d_new[v]` – new value of dynamic.

$$\texttt{d\_new[v][k]} = \texttt{d[v][k]} + \sum_{\max(x,y)=k, x+y+1 \geqslant c} \texttt{d[v][x]} \cdot \texttt{d[u][y]}$$

F. Problem F

Will add subtress one by one. Let u will be child of v.

`d_new[v]` – new value of dynamic.

$$\texttt{d\_new[v][k]} = \texttt{d[v][k]} + \sum_{\max(x,y)=k, x+y+1 \geqslant c} \texttt{d[v][x]} \cdot \texttt{d[u][y]}$$

$$\begin{aligned}\texttt{d\_new[v][k]} &= \texttt{d[v][k]} \cdot d[u][c-1] \\ &+ \texttt{d[v][k]} \cdot \textstyle\sum_{t \geqslant k, t+k+1 \geqslant c} \texttt{d[u][t]} \\ &+ \texttt{d[u][k-1]} \cdot \textstyle\sum_{t \geqslant k, t+k \geqslant c} \texttt{d[v][t]}\end{aligned}$$

G. Problem G

Perform two queries on array:

- For $i \in [l, r]$ set $a_i = \max(a_i, x)$,
- Find sum $\sum_{i \in [l,r]} a_i$.

G. Problem G

Split array in chunks of size k. For each chunk, store:

- Sorted list of pairs $(\text{value}, \text{position}) = (a_i, i)$,
- sum of number on this chunk.

G. Problem G

Split array in chunks of size k. For each chunk, store:

- Sorted list of pairs $(\text{value}, \text{position}) = (a_i, i)$,
- sum of number on this chunk.

Complexity of operations:

- Change query in $O(n/k + k \log k)$ amortized,
- sum query in $O(n/k + k)$.

G. Problem G

Split array in chunks of size k. For each chunk, store:

- Sorted list of pairs $(\text{value}, \text{position}) = (a_i, i)$,
- sum of number on this chunk.

Complexity of operations:

- Change query in $O(n/k + k \log k)$ amortized,
- sum query in $O(n/k + k)$.

Set $k = \sqrt{\frac{n}{\log n}}$, complexity becames $O(\sqrt{n \log n})$ per query.

G. Problem G

Alternative: segment tree of treaps.

In every treap store:

- pairs (`number`, `number of copies`),
- sum of numbers,
- use lazy propagation.

G. Problem G

Alternative: segment tree of treaps.

In every treap store:

- pairs (`number`, `number of copies`),
- sum of numbers,
- use lazy propagation.

Complexity:

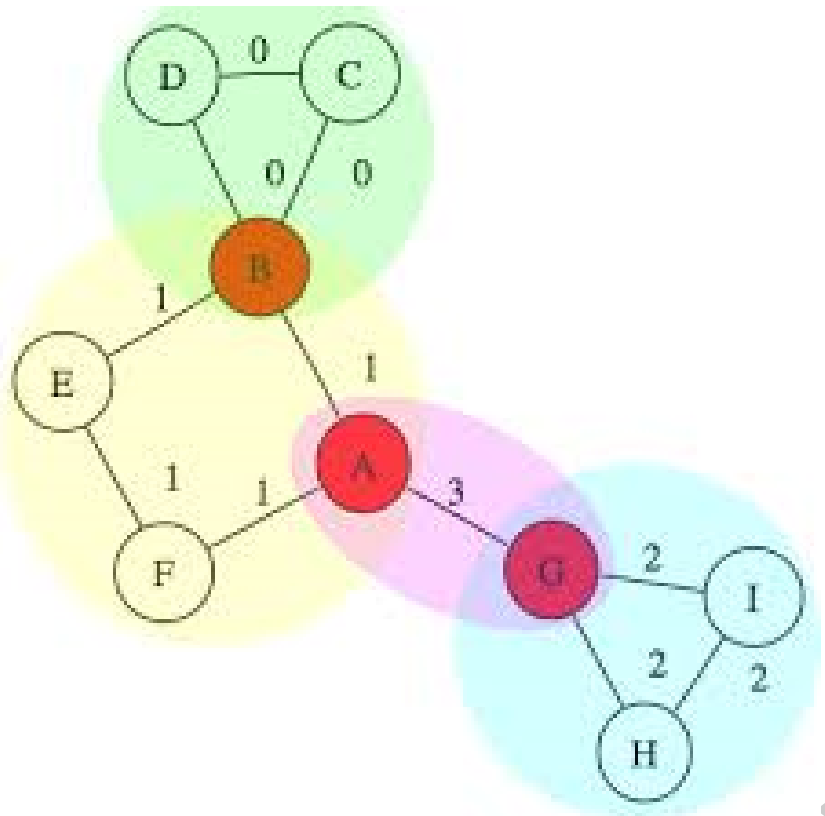
- Change query — $O(\log^2 n)$ amortized,
- sum query — $O(\log^2 n)$.

H. Oriental Puzzle

You are given an undirected graph G with two vertices s and t selected. The goal is to assign each vertex some unique height $h(v)$ so that for each edge there exists a path from s to t that passes through this edge and the values of h are strictly decreasing along the path.

H. Oriental Puzzle

As the first step we find all articulation points of this graph and all its bi-connected components. Now if we look at the structure of this components we can observe that if there exists at least one component that any path from s to t enters and leaves at the same vertex, the answer doesn't exists.



H. Oriental Puzzle

Build any simple path from s to t, check that it passes through all biconnected components. For each component identify its entry and leaving point.

H. Oriental Puzzle

Build any simple path from s to t, check that it passes through all biconnected components. For each component identify its entry and leaving point.

We now need to solve the original problem for a bi-connected graph.

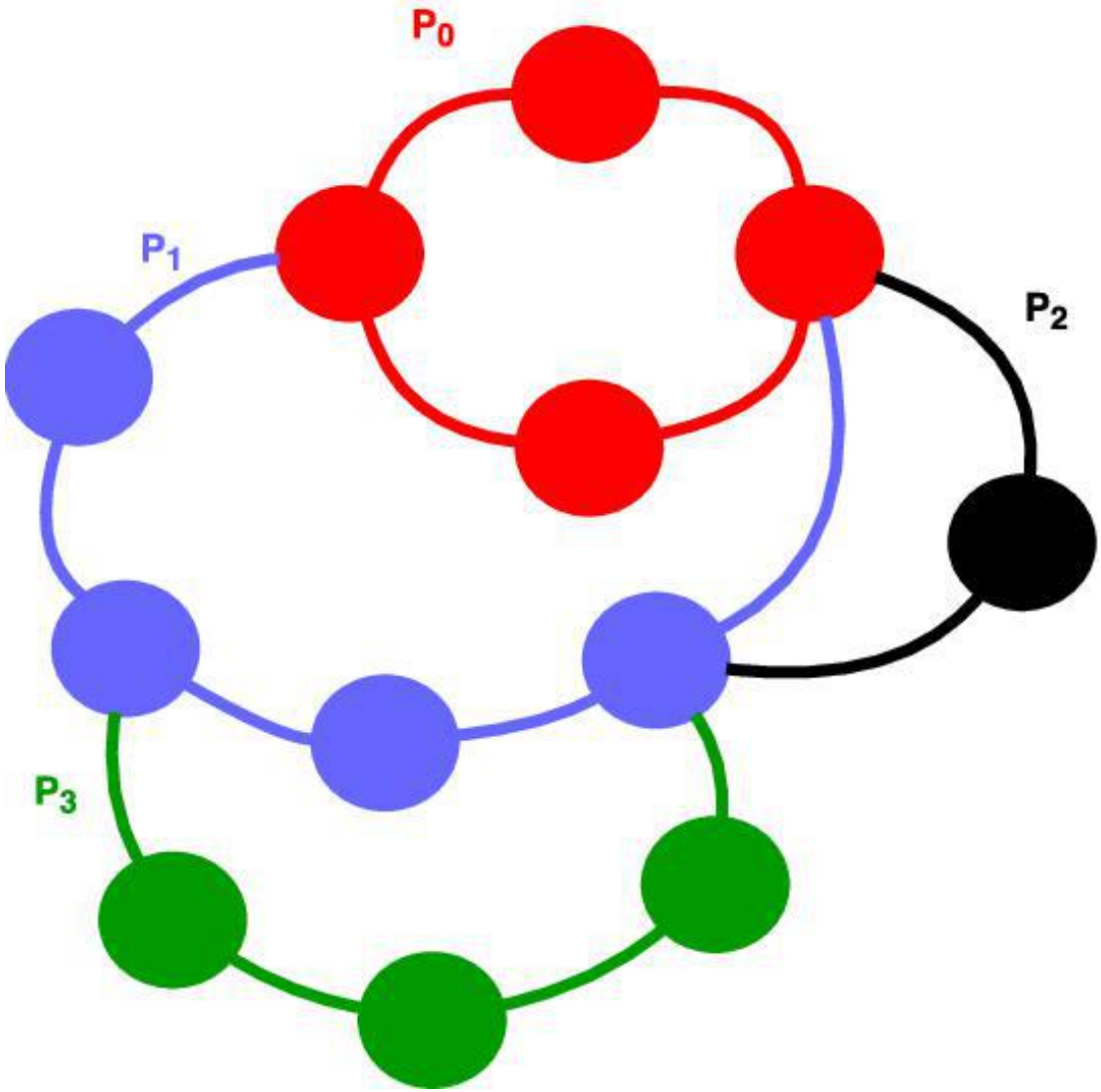
H. Oriental Puzzle

Build any simple path from s to t, check that it passes through all biconnected components. For each component identify its entry and leaving point.

We now need to solve the original problem for a bi-connected graph.

From this point it will be useful to consider an object called ear decomposition. Ear is a simple path $v_0, v_1, \ldots, v_k$ in a graph such that its endpoints have degrees $\geqslant 2$ and all intermediate vertices are of degree exactly 2.

H. Oriental Puzzle



H. Oriental Puzzle

Imagine we have an ear decomposition of the given graph, i.e. the sequence of ears removals that leaves the graph as the path from s to t.

H. Oriental Puzzle

Imagine we have an ear decomposition of the given graph, i.e. the sequence of ears removals that leaves the graph as the path from s to t.

We can set the right height for one path. Then, for any ear addition we can take the heights of its endpoints and properly set the heights on intermediate vertices.

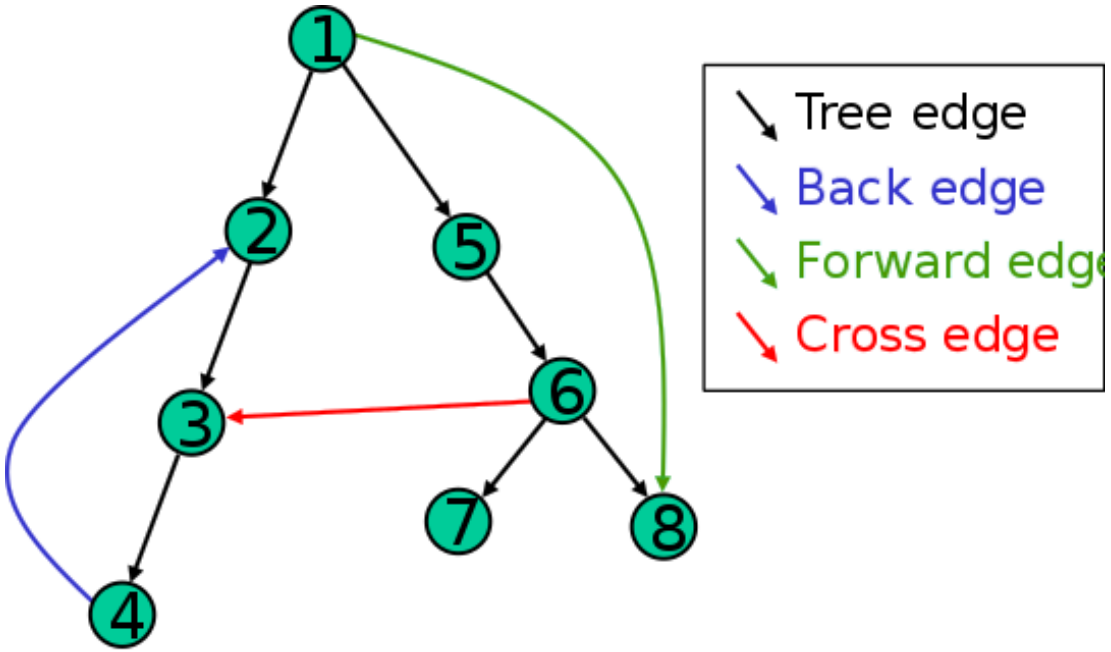
H. Oriental Puzzle

Imagine we have an ear decomposition of the given graph, i.e. the sequence of ears removals that leaves the graph as the path from s to t.

We can set the right height for one path. Then, for any ear addition we can take the heights of its endpoints and properly set the heights on intermediate vertices.

Thus, if we find a way to construct the proper decomposition, the problem will be solved. We claim such decomposition always exists for any bi-connected graph. Consider DFS-tree.

H. Oriental Puzzle



H. Oriental Puzzle

For any leaf we can drop all external edges, except for the topmost one. Any leaf now defines some ear. We can cut them one by one to build desired decomposition.

H. Oriental Puzzle

For any leaf we can drop all external edges, except for the topmost one. Any leaf now defines some ear. We can cut them one by one to build desired decomposition.

One should carefully select the leaf with external edge of the maximum depth and remove it. Do not remove node t.

I. Data Packetd

Split a string of digits 1 through 9 into two subsequences so that their difference (as decimal numbers) is smallest possible. Several testcases, total length does not exceed 100.

(Note that the string cannot contain any zeros, hence we don't have to care about leading zeros.)

I. Data Packetd

Author's solution contains a lot of tricky backtracking optimization.

J. Quartet Distance

A quartet $AB|CD$ in a tree is a quadruple of leaves such that A, B and C, D are in different components with respect to some edge. Denote $Q(T)$ the set of all quartets of a tree T. Find the size of $Q(T_1) \triangle Q(T_2)$ for two given unrooted binary trees T_1, T_2 both with n leaves (and therefore $2n - 2$ vertices in total).

An unrooted binary tree has degrees of all vertices equal to either 1 or 3.

J. Quartet Distance

A quartet $AB|CD$ in a tree is a quadruple of leaves such that A, B and C, D are in different components with respect to some edge. Denote $Q(T)$ the set of all quartets of a tree T. Find the size of $Q(T_1) \triangle Q(T_2)$ for two given unrooted binary trees T_1, T_2 both with n leaves (and therefore $2n - 2$ vertices in total).

An unrooted binary tree has degrees of all vertices equal to either 1 or 3.

Outline: assign closest separating edges to each common quartet of T_1 and T_2, count the number of common quartets for each pair of assigned edges with simple combinatorics. Count common leaves in subtrees of T_1 and T_2 with 2D prefix sums.

J. Quartet Distance

Equivalently, $AB|CD$ is a quartet if paths AB and CD do not have common vertices. One can easily see that there is exactly one quartet formed by any quadruple of leaves A, B, C, D, hence there are $\binom{n}{4}$ quartets in any unrooted binary tree with n leaves. Since $|Q(T_1) \triangle Q(T_2)| = 2\binom{n}{4} - 2|Q(T_1) \cap Q(T_2)|$, we will be computing $|Q(T_1) \cap Q(T_2)|$.

J. Quartet Distance

Equivalently, $AB|CD$ is a quartet if paths AB and CD do not have common vertices. One can easily see that there is exactly one quartet formed by any quadruple of leaves A, B, C, D, hence there are $\binom{n}{4}$ quartets in any unrooted binary tree with n leaves. Since $|Q(T_1) \triangle Q(T_2)| = 2\binom{n}{4} - 2|Q(T_1) \cap Q(T_2)|$, we will be computing $|Q(T_1) \cap Q(T_2)|$.

Assuming $AB|CD$ is a quartet, let $v_i(AB, CD)$ be the vertex on the path AB closest to the path CD in the tree T_i, and $u_i(AB, CD)$ be the unique neighbour of $v_i(AB, CD)$ such that the edge $v_i(\ldots)u_i(\ldots)$ separates AB and CD in T_i. One can see that $v_i(\ldots)$ and $u_i(\ldots)$ must have degree 3 in T_i.

J. Quartet Distance

For a pair of edges $v_1u_1 \in T_1$, $v_2u_2 \in T_2$, let us count the number of quartets $AB|CD$ with $v_i(AB|CD) = v_i$, $u_i(AB|CD) = u_i$ for $i = 1, 2$. Denote $C_i(v, u)$ the component of T_i containing v after erasing the edge vu, and $|C_i(\ldots)|$ be the number of *leaves* in the component.

J. Quartet Distance

For a pair of edges $v_1u_1 \in T_1$, $v_2u_2 \in T_2$, let us count the number of quartets $AB|CD$ with $v_i(AB|CD) = v_i$, $u_i(AB|CD) = u_i$ for $i = 1, 2$. Denote $C_i(v, u)$ the component of T_i containing v after erasing the edge vu, and $|C_i(\ldots)|$ be the number of *leaves* in the component.

We must have the following conditions:

- $A, B \in C_1(v_1, u_1) \cap C_2(v_2, u_2)$

J. Quartet Distance

For a pair of edges $v_1u_1 \in T_1$, $v_2u_2 \in T_2$, let us count the number of quartets $AB|CD$ with $v_i(AB|CD) = v_i$, $u_i(AB|CD) = u_i$ for $i = 1, 2$. Denote $C_i(v, u)$ the component of T_i containing v after erasing the edge vu, and $|C_i(\ldots)|$ be the number of *leaves* in the component.

We must have the following conditions:

- $A, B \in C_1(v_1, u_1) \cap C_2(v_2, u_2)$
- $C, D \in C_1(u_1, v_1) \cap C_2(u_2, v_2)$

J. Quartet Distance

For a pair of edges $v_1u_1 \in T_1$, $v_2u_2 \in T_2$, let us count the number of quartets $AB|CD$ with $v_i(AB|CD) = v_i$, $u_i(AB|CD) = u_i$ for $i = 1, 2$. Denote $C_i(v, u)$ the component of T_i containing v after erasing the edge vu, and $|C_i(\ldots)|$ be the number of *leaves* in the component.

We must have the following conditions:

- $A, B \in C_1(v_1, u_1) \cap C_2(v_2, u_2)$
- $C, D \in C_1(u_1, v_1) \cap C_2(u_2, v_2)$
- A, B are not in the same subtree of $C_i(v_i, u_i)$ rooted at v_i.

J. Quartet Distance

For a pair of edges $v_1u_1 \in T_1$, $v_2u_2 \in T_2$, let us count the number of quartets $AB|CD$ with $v_i(AB|CD) = v_i$, $u_i(AB|CD) = u_i$ for $i = 1, 2$. Denote $C_i(v, u)$ the component of T_i containing v after erasing the edge vu, and $|C_i(\ldots)|$ be the number of *leaves* in the component.

We must have the following conditions:

- $A, B \in C_1(v_1, u_1) \cap C_2(v_2, u_2)$
- $C, D \in C_1(u_1, v_1) \cap C_2(u_2, v_2)$
- A, B are not in the same subtree of $C_i(v_i, u_i)$ rooted at v_i.

If not for the last condition, the number of suitable quartets could be found as $\binom{|C_1(u_1,v_1)\cap C_2(u_2,v_2)|}{2} \times \binom{|C_1(v_1,u_1)\cap C_2(v_2,u_2)|}{2}$. We now have to apply inclusion-exclusion for cases when AB are in the same subtree of $C_1(v_1, u_1)/C_2(v_2, u_2)$. Note that it suffices to process queries for finding $|C_1(a, b) \cap C_2(c, d)|$ for given edges ab, cd.

J. Quartet Distance

How to compute $|C_1(a, b) \cap C_2(c, d)|$ fast? Let us construct a DFS preorder of each tree (starting from any vertex), and $x(A)$ and $y(A)$ be the numbers of the leaf A in preorders of T_1 and T_2 respectively.

J. Quartet Distance

How to compute $|C_1(a,b) \cap C_2(c,d)|$ fast? Let us construct a DFS preorder of each tree (starting from any vertex), and $x(A)$ and $y(A)$ be the numbers of the leaf A in preorders of T_1 and T_2 respectively.

Note that preorder numbers of all leaves in any $C_i(\ldots)$ form either a segment, or a complement of a segment. For simplicity assume that numbers of leaves in $C_1(a,b)$ and $C_2(c,d)$ form segments $[l_1, r_1]$ and $[l_2, r_2]$; then $|C_1(a,b) \cap C_2(c,d)|$ is the number of two-dimensional points $(x(A), y(A))$ in the rectangle $[l_1, r_1] \times [l_2, r_2]$. (A segment complement case can be reduced to two segments).

J. Quartet Distance

Since preorder numbers do not exceed n, we can answer $|C_1(a, b) \cap C_2(c, d)|$ queries in $O(1)$ by constructing an $n \times n$ matrix, and precomputing 2D prefix sums to answer rectangle sum queries.

J. Quartet Distance

Since preorder numbers do not exceed n, we can answer $|C_1(a, b) \cap C_2(c, d)|$ queries in $O(1)$ by constructing an $n \times n$ matrix, and precomputing 2D prefix sums to answer rectangle sum queries.

In total, we have to consider $O(n^2)$ edge pairs, with each pair giving rise to $O(1)$ rectangle sum queries. The above solution thus has complexity $O(n^2)$.

K. Rectangles

Given a matrix of numbers, find a subrectangle that maximizes the ratio "the sum of numbers / the number of external cell borders".

K. Rectangles

We'll do binary search on the answer. Let S denote the sum inside a rectangle, and P its perimeter. To check if the answer is at least λ, we have to check if there is a rectangle such that $S - \lambda P \geqslant 0$.

K. Rectangles

We'll do binary search on the answer. Let S denote the sum inside a rectangle, and P its perimeter. To check if the answer is at least λ, we have to check if there is a rectangle such that $S - \lambda P \geqslant 0$.

We can do that similarly to the largest sum submatrix problem. Try all options of the top and bottom rows of the rectangle. Suppose that the current height is h, then there are $2h$ horizontal cell borders.

K. Rectangles

Let s_j be the sum of all numbers in the j-th column lying in the selected range of rows, and $S_i = \sum_{j=1}^{i-1} s_j$.

K. Rectangles

Let s_j be the sum of all numbers in the j-th column lying in the selected range of rows, and $S_i = \sum_{j=1}^{i-1} s_j$.

If left and right borders are at columns l and r respectively, then

$$S - \lambda P = S_{r+1} - S_l - 2\lambda(h + r + 1 - l) = (S_{r+1} + 2\lambda(r+1)) - (S_l + 2\lambda l) - 2\lambda h.$$

K. Rectangles

Let s_j be the sum of all numbers in the j-th column lying in the selected range of rows, and $S_i = \sum_{j=1}^{i-1} s_j$.

If left and right borders are at columns l and r respectively, then

$$S - \lambda P = S_{r+1} - S_l - 2\lambda(h + r + 1 - l) = (S_{r+1} + 2\lambda(r+1)) - (S_l + 2\lambda l) - 2\lambda h.$$

If we denote $A_j = S_j + 2\lambda j$, if suffices to check if there is a pair of indices $i < j$ such that $A_j - A_i > 2\lambda h$. This can be done in linear time by maintaining prefix minimums of A_i.

K. Rectangles

Let s_j be the sum of all numbers in the j-th column lying in the selected range of rows, and $S_i = \sum_{j=1}^{i-1} s_j$.

If left and right borders are at columns l and r respectively, then

$$S - \lambda P = S_{r+1} - S_l - 2\lambda(h + r + 1 - l) = (S_{r+1} + 2\lambda(r+1)) - (S_l + 2\lambda l) - 2\lambda h.$$

If we denote $A_j = S_j + 2\lambda j$, if suffices to check if there is a pair of indices $i < j$ such that $A_j - A_i > 2\lambda h$. This can be done in linear time by maintaining prefix minimums of A_i.

Use 2D prefix sums to compute s_i, S_i in constant time.

K. Rectangles

Let s_j be the sum of all numbers in the j-th column lying in the selected range of rows, and $S_i = \sum_{j=1}^{i-1} s_j$.

If left and right borders are at columns l and r respectively, then

$$S - \lambda P = S_{r+1} - S_l - 2\lambda(h + r + 1 - l) = (S_{r+1} + 2\lambda(r+1)) - (S_l + 2\lambda l) - 2\lambda h.$$

If we denote $A_j = S_j + 2\lambda j$, if suffices to check if there is a pair of indices $i < j$ such that $A_j - A_i > 2\lambda h$. This can be done in linear time by maintaining prefix minimums of A_i.

Use 2D prefix sums to compute s_i, S_i in constant time.

Total complexity is $O(n^3 C)$, where C is the number of binary search iterations (~ 60 should be enough).

L. Christmas Tree

A person is searching for a high tree in a forest. Starting from any tree, they repeat the following: if the current tree height in h, if the heighest tree within the $2h \times 2h$ centered at the current tree is at most h, terminate at the current tree, otherwise proceed to the highest tree in the square. Determine the lowest tree that the process can terminate at (we are able to choose the starting tree arbitrarily).

L. Christmas Tree

The process can terminate at a tree iff there is no higher tree that can be seen from it. Indeed, if we start at such a tree, the process terminates immediately. On the other hand, the process can never terminate at a tree that violates this condition.

L. Christmas Tree

The process can terminate at a tree iff there is no higher tree that can be seen from it. Indeed, if we start at such a tree, the process terminates immediately. On the other hand, the process can never terminate at a tree that violates this condition.

We should now find the lowest tree such that no higher tree can be seen in $2h \times 2h$ range around it. We can do it by building a 2D segment tree with max operation on the n given points and making n rectangle maximum queries to it.

L. Christmas Tree

Brief desctiption of 2D segment tree:

- Sort all x-coordinates of all points, construct a segment tree on them. Each vertex of this segment tree is responsible for a horizontal range, and contains another segment tree with all points in this range sorted by y-coordinate.

L. Christmas Tree

Brief desctiption of 2D segment tree:

- Sort all x-coordinates of all points, construct a segment tree on them. Each vertex of this segment tree is responsible for a horizontal range, and contains another segment tree with all points in this range sorted by y-coordinate.
- To make a rectangle query, split its horizontal span into $O(\log n)$ vertex ranges, and make a query to each of the vertices' internal segment trees.

L. Christmas Tree

Brief desctiption of 2D segment tree:

- Sort all x-coordinates of all points, construct a segment tree on them. Each vertex of this segment tree is responsible for a horizontal range, and contains another segment tree with all points in this range sorted by y-coordinate.
- To make a rectangle query, split its horizontal span into $O(\log n)$ vertex ranges, and make a query to each of the vertices' internal segment trees.

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