Contest 4

Mikhail Tikhomirov, Artsem Zhuk

19th February 2019

A. Another LCA Problem

Given a rooted tree, process queries "find LCA of two vertices" and "change parent of a vertex"

A. Another LCA Problem

Solution 1: process queries in blocks of K.

A. Another LCA Problem

Solution 1: process queries in blocks of K.

Cut parent links for vertices whose parents are going to change in the upcoming block. Construct "compressed" trees of persisting links, build LCA structure for each compressed tree.

A. Another LCA Problem

Solution 1: process queries in blocks of K.

Cut parent links for vertices whose parents are going to change in the upcoming block. Construct "compressed" trees of persisting links, build LCA structure for each compressed tree.

Change parents explicitly.

A. Another LCA Problem

Solution 1: process queries in blocks of K.

Cut parent links for vertices whose parents are going to change in the upcoming block. Construct "compressed" trees of persisting links, build LCA structure for each compressed tree.

Change parents explicitly.

To answer LCA query, find the compressed tree containing the LCA explicitly in $O(K)$, then query the LCA structure.

A. Another LCA Problem

Solution 1: process queries in blocks of K.

Cut parent links for vertices whose parents are going to change in the upcoming block. Construct "compressed" trees of persisting links, build LCA structure for each compressed tree.

Change parents explicitly.

To answer LCA query, find the compressed tree containing the LCA explicitly in $O(K)$, then query the LCA structure.

Assuming optimal LCA, complexity is $O(Q/K \times (N + K^2))$, with optimum $K \sim \sqrt{N}$ yielding complexity $O(Q\sqrt{N})$. Simpler LCA is $O(\log N)$ slower, but should still get OK.

A. Another LCA Problem

Solution 2: use link-cut trees.

A. Another LCA Problem

Solution 2: use link-cut trees.

Relink query: cut above v, link v below u.

A. Another LCA Problem

Solution 2: use link-cut trees.

Relink query: cut above v, link v below u.

LCA query: `expose`(v), `expose`(u), then the parent of the top vertex of v's path is LCA.

A. Another LCA Problem

Solution 2: use link-cut trees.

Relink query: cut above v, link v below u.

LCA query: `expose`(v), `expose`(u), then the parent of the top vertex of v's path is LCA.

Complexity $O((q+n)\log n)$.

B. Bar

Find infinite sum

$$\sum_{n=1}^{\infty} \frac{\ln n}{n^s}.$$

B. Bar

We can approximate the sum by integral:

$$\sum_{n=1}^{\infty} \frac{\ln n}{n^s} = \sum_{n=1}^{n=m} \frac{\ln n}{n^s} + \int_m^{\infty} \frac{\ln x}{x^s} dx.$$

B. Bar

We can approximate the sum by integral:

$$\sum_{n=1}^{\infty} \frac{\ln n}{n^s} = \sum_{n=1}^{n=m} \frac{\ln n}{n^s} + \int_m^{\infty} \frac{\ln x}{x^s} dx.$$

$$\int_a^{\infty} \frac{lnx}{x^s} dx = \frac{a^{1-s}(1 + (s-1)\ln a)}{(s-1)^2}.$$

B. Bar

We can approximate the sum by integral:

$$\sum_{n=1}^{\infty} \frac{\ln n}{n^s} = \sum_{n=1}^{n=m} \frac{\ln n}{n^s} + \int_m^{\infty} \frac{\ln x}{x^s} dx.$$

$$\int_a^{\infty} \frac{lnx}{x^s} dx = \frac{a^{1-s}(1 + (s-1)\ln a)}{(s-1)^2}.$$

Empirically 30 summands are enough for the most unstable case $s = 1.0001$.

C. Counting Palindromes

For a given string s, process queries "count the number of palindromic substrings of $s_l \ldots s_r$".

C. Counting Palindromes

Find the (half-)sizes p_i of maximal palindromes with all possible centers i with Manacher's algorithm.

C. Counting Palindromes

Find the (half-)sizes p_i of maximal palindromes with all possible centers i with Manacher's algorithm.

Let's count odd-sized subpalindomes of $s_l \dots s_r$ (even-sized ones are counted similarly).

C. Counting Palindromes

Find the (half-)sizes p_i of maximal palindromes with all possible centers i with Manacher's algorithm.

Let's count odd-sized subpalindomes of $s_l \dots s_r$ (even-sized ones are counted similarly).

The answer is $\sum_{i=l}^{r} \min(p_i, i - l + 1, r - i + 1)$.

C. Counting Palindromes

Find the (half-)sizes p_i of maximal palindromes with all possible centers i with Manacher's algorithm.

Let's count odd-sized subpalindomes of $s_l \dots s_r$ (even-sized ones are counted similarly).

The answer is $\sum_{i=l}^{r} \min(p_i, i - l + 1, r - i + 1)$.

Observe that this is equal to
$\sum_{i=l}^{\lfloor (l+r)/2 \rfloor} \min(p_i, i - l + 1) + \sum_{i=\lfloor (l+r)/2 \rfloor + 1}^{r} \min(p_i, r - i + 1)$.

C. Counting Palindromes

Find the (half-)sizes p_i of maximal palindromes with all possible centers i with Manacher's algorithm.

Let's count odd-sized subpalindomes of $s_l \dots s_r$ (even-sized ones are counted similarly).

The answer is $\sum_{i=l}^{r} \min(p_i, i - l + 1, r - i + 1)$.

Observe that this is equal to
$\sum_{i=l}^{\lfloor (l+r)/2 \rfloor} \min(p_i, i - l + 1) + \sum_{i=\lfloor (l+r)/2 \rfloor + 1}^{r} \min(p_i, r - i + 1)$.

We process the two halves of each query offline independently.

C. Counting Palindromes

Let us answer several queries $\sum_{i=l}^{m} \min(p_i, i - l + 1)$. We process queries by decreasing of l, and maintain RSQ structure on the changing array $q_i = \min(p_i, i - l + 1)$.

C. Counting Palindromes

Let us answer several queries $\sum_{i=l}^{m} \min(p_i, i - l + 1)$. We process queries by decreasing of l, and maintain RSQ structure on the changing array $q_i = \min(p_i, i - l + 1)$.

We would have to update a lot of elements q_i if we stored them directly, so instead represent q_i as linear functions of l. Answering a query can be done by querying RSQ of linear functions and substituting current value of l.

C. Counting Palindromes

Let us answer several queries $\sum_{i=l}^{m} \min(p_i, i - l + 1)$. We process queries by decreasing of l, and maintain RSQ structure on the changing array $q_i = \min(p_i, i - l + 1)$.

We would have to update a lot of elements q_i if we stored them directly, so instead represent q_i as linear functions of l. Answering a query can be done by querying RSQ of linear functions and substituting current value of l.

Each linear function will be updated at most twice (initialized with $q_i = i - l + 1$, changed to $q_i = p_i$).

C. Counting Palindromes

Let us answer several queries $\sum_{i=l}^{m} \min(p_i, i - l + 1)$. We process queries by decreasing of l, and maintain RSQ structure on the changing array $q_i = \min(p_i, i - l + 1)$.

We would have to update a lot of elements q_i if we stored them directly, so instead represent q_i as linear functions of l. Answering a query can be done by querying RSQ of linear functions and substituting current value of l.

Each linear function will be updated at most twice (initialized with $q_i = i - l + 1$, changed to $q_i = p_i$).

The total complexity is now $O((q + n) \log n)$.

D. Drawing Trianlges

You are given a triangle. Split it into similar triangles with distinct sizes.

D. Drawing Triangles

Solution doesn't exist for an equilateral triangle.

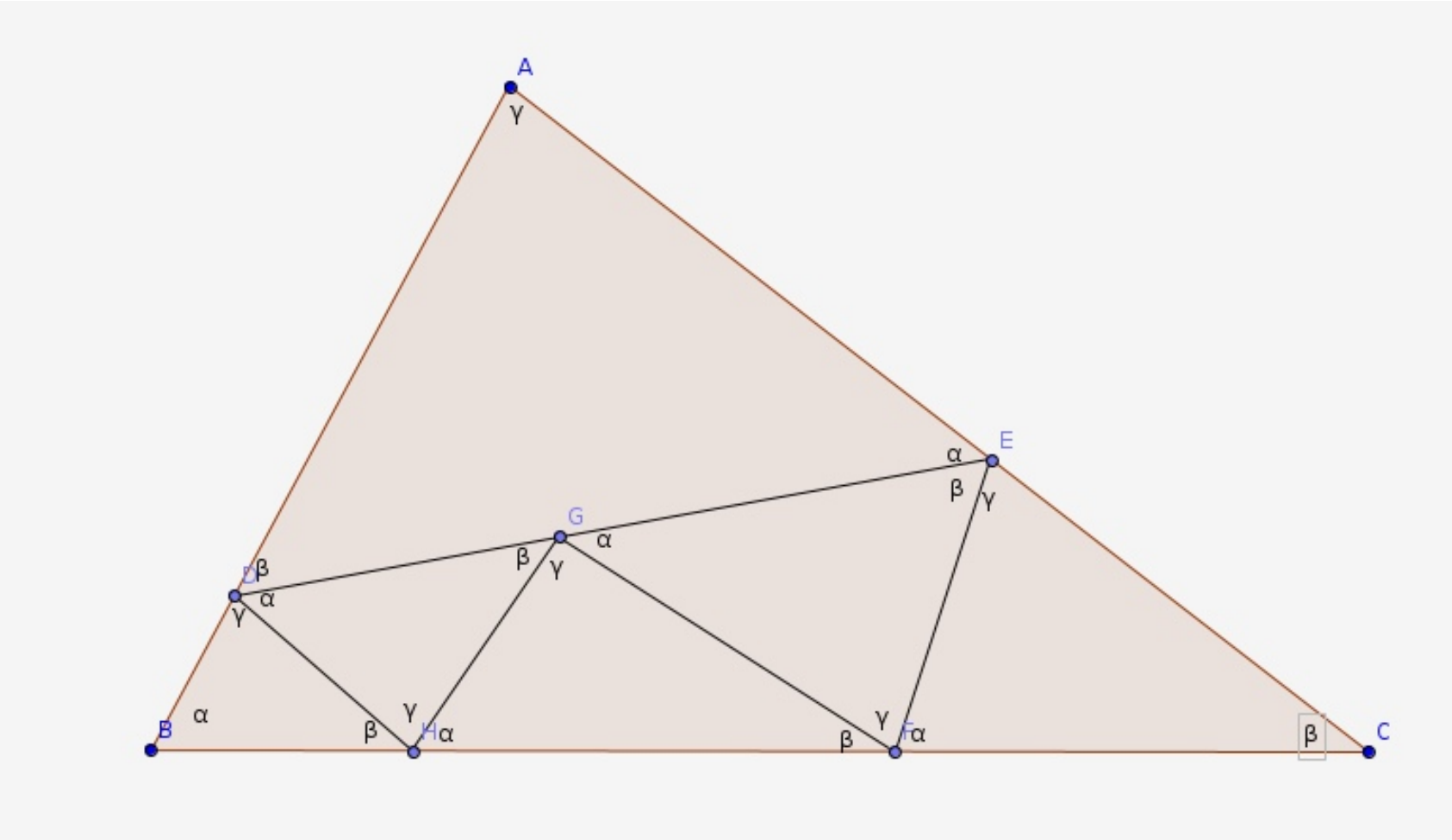
D. Drawing Triangles

Solution doesn't exist for an equilateral triangle.

But it is impossible to draw equilateral triangle on integer grid.

D. Drawing Triangles

Chose two non-equal sides and split into 6 similar triangles at most cases:



E. European Football Championship

Given a shuffled list of a football championship, determine the number of possible tournament schemes agreeing with this list.

E. European Football Championship

By looking at the number of plays for each team, we can divide them into four groups:

- G — eliminated at group stage (8 teams with 3 plays);
- Q — elimiated at quarterfinals (4 teams with 4 plays);
- S — eliminated at semifinals (2 teams with 5 plays);
- F — advanced to finals (2 teams with 6 plays).

E. European Football Championship

By looking at the number of plays for each team, we can divide them into four groups:

- G — eliminated at group stage (8 teams with 3 plays);
- Q — elimiated at quarterfinals (4 teams with 4 plays);
- S — eliminated at semifinals (2 teams with 5 plays);
- F — advanced to finals (2 teams with 6 plays).

If, in addition, we knew which plays happened at group stage and which in the play-off, we could completely determine the scheme.

E. European Football Championship

By looking at the number of plays for each team, we can divide them into four groups:

- G — eliminated at group stage (8 teams with 3 plays);
- Q — elimiated at quarterfinals (4 teams with 4 plays);
- S — eliminated at semifinals (2 teams with 5 plays);
- F — advanced to finals (2 teams with 6 plays).

If, in addition, we knew which plays happened at group stage and which in the play-off, we could completely determine the scheme.

Final observation: there are at most two games between any pair of teams (one at group stage and one at playoff).

E. European Football Championship

Let us try all options for the play-off games:

- final matchup is unambigious;

E. European Football Championship

Let us try all options for the play-off games:

- final matchup is unambigious;
- choose one of 2 matchups in the semifinal between S and F teams;

E. European Football Championship

Let us try all options for the play-off games:

- final matchup is unambigious;
- choose one of 2 matchups in the semifinal between S and F teams;
- choose one of 4! matchups between Q and S+F teams in quarterfinals.

E. European Football Championship

Let us try all options for the play-off games:

- final matchup is unambigious;
- choose one of 2 matchups in the semifinal between S and F teams;
- choose one of 4! matchups between Q and S+F teams in quarterfinals.

For each play-off game, choose one of at most two games in the list.

E. European Football Championship

Let us try all options for the play-off games:

- final matchup is unambigious;
- choose one of 2 matchups in the semifinal between S and F teams;
- choose one of 4! matchups between Q and S+F teams in quarterfinals.

For each play-off game, choose one of at most two games in the list.

There are at most $2 \times 4! \times 2^7 = 6144$ options to consider. For each of them, check if it is consistent and gives rise to a correct scheme according to all the rules.

F. Filling

You are given $W \times H$ field with one cell removed. Tile the remaining cells with L-shaped piece.

F. Filling

Note that we can tile rectangle of size $6 \times n$ for $n \geqslant 2$.

F. Filling

Note that we can tile rectangle of size $6 \times n$ for $n \geqslant 2$.

Can reduce rectangle size to 17×17 or smaller, then run brute-force algorithm with optimizations.

F. Filling

Note that we can tile rectangle of size $6 \times n$ for $n \geqslant 2$.

Can reduce rectangle size to 17×17 or smaller, then run brute-force algorithm with optimizations.

Small tip: save at least two cells in every direction around removed cell.

G. Graph Theory Problem

You have to support three types of offline queries to graph:

- Add an edge
- Remove an edge
- Count number of bridges in the graph

G. Graph Theory Problem

As kindly mentioned in statement, with divide-and-conquer one can reduce to the following problem:

Implement a data structure that supports operations “add an edge”, “find the number of bridges” and “retract several last operations”.

G. Graph Theory Problem

As kindly mentioned in statement, with divide-and-conquer one can reduce to the following problem:

Implement a data structure that supports operations “add an edge”, “find the number of bridges” and “retract several last operations”.

As usual, retractions are treated simply by storing a stack of variable changes (along with old values), and undoing them while necessary.

G. Graph Theory Problem

As kindly mentioned in statement, with divide-and-conquer one can reduce to the following problem:

Implement a data structure that supports operations “add an edge”, “find the number of bridges” and “retract several last operations”.

As usual, retractions are treated simply by storing a stack of variable changes (along with old values), and undoing them while necessary.

Solution 2: use link-cut trees.

G. Graph Theory Problem

Solution 1: the goal is to reduce the graph to size $O(k)$ while processing a segment of k queries.

G. Graph Theory Problem

Solution 1: the goal is to reduce the graph to size $O(k)$ while processing a segment of k queries.

Consider $O(k)$ vertices present among queries in the segment, mark them as active. In each connected component (represented as a tree) T, find the smallest connected subtree T' containing all active vertices.

G. Graph Theory Problem

Solution 1: the goal is to reduce the graph to size $O(k)$ while processing a segment of k queries.

Consider $O(k)$ vertices present among queries in the segment, mark them as active. In each connected component (represented as a tree) T, find the smallest connected subtree T' containing all active vertices.

All edges not in T' are going to be bridges throughout the segment; can erase them.

G. Graph Theory Problem

Solution 1: the goal is to reduce the graph to size $O(k)$ while processing a segment of k queries.

Consider $O(k)$ vertices present among queries in the segment, mark them as active. In each connected component (represented as a tree) T, find the smallest connected subtree T' containing all active vertices.

All edges not in T' are going to be bridges throughout the segment; can erase them.

T' contains $O(k)$ vertices connected by edge chains; contract T while preserving edge multiplicities (can be done in $O(k)$ with one DFS).

G. Graph Theory Problem

Solution 1: the goal is to reduce the graph to size $O(k)$ while processing a segment of k queries.

Consider $O(k)$ vertices present among queries in the segment, mark them as active. In each connected component (represented as a tree) T, find the smallest connected subtree T' containing all active vertices.

All edges not in T' are going to be bridges throughout the segment; can erase them.

T' contains $O(k)$ vertices connected by edge chains; contract T while preserving edge multiplicities (can be done in $O(k)$ with one DFS).

To add edges, need to connect different components (explicitly), or contract all edges on a path (need LCA).

G. Graph Theory Problem

Solution 1: the goal is to reduce the graph to size $O(k)$ while processing a segment of k queries.

Consider $O(k)$ vertices present among queries in the segment, mark them as active. In each connected component (represented as a tree) T, find the smallest connected subtree T' containing all active vertices.

All edges not in T' are going to be bridges throughout the segment; can erase them.

T' contains $O(k)$ vertices connected by edge chains; contract T while preserving edge multiplicities (can be done in $O(k)$ with one DFS).

To add edges, need to connect different components (explicitly), or contract all edges on a path (need LCA).

Complexity between $O(n \log^2 n)$ and $O(n \log n)$ depending on LCA structure.

H. Hatto Nero

You are given a tree. Each edge is controlled by an official with rank r_i and greedines c_i.

You are going from the root to every vertex. On you way you pay each official either c_i or *median* of pays to officials-*ancestors* with *smaller* rank (if there are any).

H. Hatto Nero

Let x_i denotes a pay of i-th official.

We need a data structure supporting following operations:

- add/delete point (r_i, x_i),
- For given R, B, find number of points (r_i, x_i) with $r_i \leqslant R, x_i \leqslant B$.

H. Hatto Nero

Let x_i denotes a pay of i-th official.

We need a data structure supporting following operations:

- add/delete point (r_i, x_i),
- For given R, B, find number of points (r_i, x_i) with $r_i \leqslant R, x_i \leqslant B$.

Can use BIT-tree (know also as Fenwick tree) by r_i of treaps by x_i.

H. Hatto Nero

Let x_i denotes a pay of i-th official.

We need a data structure supporting following operations:

- add/delete point (r_i, x_i),
- For given R, B, find number of points (r_i, x_i) with $r_i \leqslant R, x_i \leqslant B$.

Can use BIT-tree (know also as Fenwick tree) by r_i of treaps by x_i.

Both operations are performed in $O(\log^2 n)$ time.

H. Hatto Nero

Let x_i denotes a pay of i-th official.

We need a data structure supporting following operations:

- add/delete point (r_i, x_i),
- For given R, B, find number of points (r_i, x_i) with $r_i \leqslant R, x_i \leqslant B$.

Can use BIT-tree (know also as Fenwick tree) by r_i of treaps by x_i.

Both operations are performed in $O(\log^2 n)$ time.

To find i-th official pay, we can first calculate number of ancestors with smaller rank using $(r_i, +\infty)$ query.

H. Hatto Nero

Let x_i denotes a pay of i-th official.

We need a data structure supporting following operations:

- add/delete point (r_i, x_i),
- For given R, B, find number of points (r_i, x_i) with $r_i \leqslant R, x_i \leqslant B$.

Can use BIT-tree (know also as Fenwick tree) by r_i of treaps by x_i.

Both operations are performed in $O(\log^2 n)$ time.

To find i-th official pay, we can first calculate number of ancestors with smaller rank using $(r_i, +\infty)$ query.

Then perform binary search by B using (r_i, B) queries.

H. Hatto Nero

Let x_i denotes a pay of i-th official.

We need a data structure supporting following operations:

- add/delete point (r_i, x_i),
- For given R, B, find number of points (r_i, x_i) with $r_i \leqslant R, x_i \leqslant B$.

Can use BIT-tree (know also as Fenwick tree) by r_i of treaps by x_i.

Both operations are performed in $O(\log^2 n)$ time.

To find i-th official pay, we can first calculate number of ancestors with smaller rank using $(r_i, +\infty)$ query.

Then perform binary search by B using (r_i, B) queries.

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

I. Alien Invasion

You have n planes in 3-dimensional space, any three intersect in exactly one point. Find a ray from the origin that intersects maximal number of planes.

I. Alien Invasion

Consider unit sphere with center in the origin. Each plance corresponds to an open hemisphere on this sphere.

I. Alien Invasion

Consider unit sphere with center in the origin. Each plance corresponds to an open hemisphere on this sphere.

Need to find point that is covered by maximal number of hemispheres.

I. Alien Invasion

Consider unit sphere with center in the origin. Each plance corresponds to an open hemisphere on this sphere.

Need to find point that is covered by maximal number of hemispheres.

WLOG can assume that chosen point lies near to some hemisphere's border.

I. Alien Invasion

Consider unit sphere with center in the origin. Each plance corresponds to an open hemisphere on this sphere.

Need to find point that is covered by maximal number of hemispheres.

WLOG can assume that chosen point lies near to some hemisphere's border.

Can iterate over that hemisphere — $O(n)$ iterations.

I. Alien Invasion

Consider unit sphere with center in the origin. Each plance corresponds to an open hemisphere on this sphere.

Need to find point that is covered by maximal number of hemispheres.

WLOG can assume that chosen point lies near to some hemisphere's border.

Can iterate over that hemisphere — $O(n)$ iterations.

For each iteration, intersect hemispheres with chosen hemispere border and get $n - 1$ segments.

I. Alien Invasion

Consider unit sphere with center in the origin. Each plance corresponds to an open hemisphere on this sphere.

Need to find point that is covered by maximal number of hemispheres.

WLOG can assume that chosen point lies near to some hemisphere's border.

Can iterate over that hemisphere — $O(n)$ iterations.

For each iteration, intersect hemispheres with chosen hemispere border and get $n - 1$ segments.

Now need to find point covered by maximal number of segmenst — can be done in $O(n \log n)$ time.

I. Alien Invasion

Consider unit sphere with center in the origin. Each plance corresponds to an open hemisphere on this sphere.

Need to find point that is covered by maximal number of hemispheres.

WLOG can assume that chosen point lies near to some hemisphere's border.

Can iterate over that hemisphere — $O(n)$ iterations.

For each iteration, intersect hemispheres with chosen hemispere border and get $n - 1$ segments.

Now need to find point covered by maximal number of segmenst — can be done in $O(n \log n)$ time.

$O(n^2 \log n)$ overall.

J. Jailing Rabbits

There are several rabbit holes in an $n \times m$ grid, each hole containing four rabbits. Initially all grid vertices and edges are uncolored. Holes are activated in random order.

- If the vertex containing the activated hole is colored, nothing happens.
- Otherwise, four rabbits start to run in cardinal directions. A rabbit arriving on the boundary turns right and finishes in a grid angle.
- If a rabbit encounters a colored vertex followed a differently colored edge, he turns right.

Calculate the total expected running distance of all rabbits.

J. Jailing Rabbits

For consistency, let us assume that the grid border is initially colored with a unique color.

J. Jailing Rabbits

For consistency, let us assume that the grid border is initially colored with a unique color.

Let us define a *subproblem* $t = (x_1, y_1, x_2, y_2)$ as a following configuration:

- each side of the rectangle R with corners (x_1, y_1) and (x_2, y_2) is mono-colored (colors of sides may be same or different);

J. Jailing Rabbits

For consistency, let us assume that the grid border is initially colored with a unique color.

Let us define a *subproblem* $t = (x_1, y_1, x_2, y_2)$ as a following configuration:

- each side of the rectangle R with corners (x_1, y_1) and (x_2, y_2) is mono-colored (colors of sides may be same or different);
- no vertex or edge inside R is colored (hence, no hole inside R has been activated);

J. Jailing Rabbits

For consistency, let us assume that the grid border is initially colored with a unique color.

Let us define a *subproblem* $t = (x_1, y_1, x_2, y_2)$ as a following configuration:

- each side of the rectangle R with corners (x_1, y_1) and (x_2, y_2) is mono-colored (colors of sides may be same or different);
- no vertex or edge inside R is colored (hence, no hole inside R has been activated);
- no edges leaving R are colored (except for edge incident to corners of R).

J. Jailing Rabbits

For consistency, let us assume that the grid border is initially colored with a unique color.

Let us define a *subproblem* $t = (x_1, y_1, x_2, y_2)$ as a following configuration:

- each side of the rectangle R with corners (x_1, y_1) and (x_2, y_2) is mono-colored (colors of sides may be same or different);
- no vertex or edge inside R is colored (hence, no hole inside R has been activated);
- no edges leaving R are colored (except for edge incident to corners of R).

The first activated hole inside the (rectangle of) a subproblem t splits it into four subproblems.

J. Jailing Rabbits

For consistency, let us assume that the grid border is initially colored with a unique color.

Let us define a *subproblem* $t = (x_1, y_1, x_2, y_2)$ as a following configuration:

- each side of the rectangle R with corners (x_1, y_1) and (x_2, y_2) is mono-colored (colors of sides may be same or different);
- no vertex or edge inside R is colored (hence, no hole inside R has been activated);
- no edges leaving R are colored (except for edge incident to corners of R).

The first activated hole inside the (rectangle of) a subproblem t splits it into four subproblems. Observation: in a subproblem t, all rabbits appearing inside its rectangle leave the rectangle through its corners. Proof by induction: consider the first activated hole and use induction hypothesis.

J. Jailing Rabbits

For each of the possible $O(n^2 m^2)$ subproblems we calculate the following values:

- ED_t — the expected running distance of all rabbits appearing inside t until they reach a corner of t;

J. Jailing Rabbits

For each of the possible $O(n^2m^2)$ subproblems we calculate the following values:

- ED_t — the expected running distance of all rabbits appearing inside t until they reach a corner of t;
- $ER_{t,c}$, where $c \in \{\texttt{TL}, \texttt{TR}, \texttt{BL}, \texttt{BR}\}$ — the expected number of rabbits appearing inside t and reaching (top/bottom) (left/right) corner of t respectively.

J. Jailing Rabbits

For each of the possible $O(n^2m^2)$ subproblems we calculate the following values:

- ED_t — the expected running distance of all rabbits appearing inside t until they reach a corner of t;
- $ER_{t,c}$, where $c \in \{\texttt{TL}, \texttt{TR}, \texttt{BL}, \texttt{BR}\}$ — the expected number of rabbits appearing inside t and reaching (top/bottom) (left/right) corner of t respectively.

Transition: try each hole (strictly) inside t that could be activated first. ED_t = the sum of $ED_{\ldots}$ for four subproblems + the running distance of rabbits from corners of subproblems to corners of t.

J. Jailing Rabbits

For each of the possible $O(n^2m^2)$ subproblems we calculate the following values:

- ED_t — the expected running distance of all rabbits appearing inside t until they reach a corner of t;
- $ER_{t,c}$, where $c \in \{\texttt{TL}, \texttt{TR}, \texttt{BL}, \texttt{BR}\}$ — the expected number of rabbits appearing inside t and reaching (top/bottom) (left/right) corner of t respectively.

Transition: try each hole (strictly) inside t that could be activated first. ED_t = the sum of $ED_{\ldots}$ for four subproblems + the running distance of rabbits from corners of subproblems to corners of t.

By linearity of expectation, $ER_{\ldots}$ is enough to compute the extra distance. $ER_{t,c}$ can be computed as following by considering 16 values $ER_{\ldots}$ for subproblems individually.

J. Jailing Rabbits

The complexity of this solution is $O(n^3 m^3)$ with a very small constant factor.

K. Key Arrays

Maintain array with following queries:

- Addition on a segment
- Reverse on a segment
- Sum on a segment
- Minimum/maximum on a segment

K. Key Arrays

Maintain array with following queries:

- Addition on a segment
- Reverse on a segment
- Sum on a segment
- Minimum/maximum on a segment

Treap with lazy propagation.

$O(\log n)$ per query.