Long Contest Editorial

November 14, 2015

Moscow International Workshop ACM ICPC, MIPT, 2015

A. Automorphism

Given a tree, count the number of its automorphisms.

A. Automorphism

First, can we solve the problem for a rooted tree?

A. Automorphism

First, can we solve the problem for a rooted tree?
The general idea resembles the algorithm for checking rooted trees isomorphism.

A. Automorphism

First, can we solve the problem for a rooted tree?
The general idea resembles the algorithm for checking rooted trees isomorphism.
We would like to assign a number to each subtree in such a way that subtrees are isomorphic iff their numbers are equal.

A. Automorphism

First, can we solve the problem for a rooted tree?
The general idea resembles the algorithm for checking rooted trees isomorphism.
We would like to assign a number to each subtree in such a way that subtrees are isomorphic iff their numbers are equal.
This can be done as follows: recursively assign a number to each of the childs' subtrees, then sort all the obtained numbers. Call the resulting vector a *profile* of the subtree.

A. Automorphism

First, can we solve the problem for a rooted tree?
The general idea resembles the algorithm for checking rooted trees isomorphism.
We would like to assign a number to each subtree in such a way that subtrees are isomorphic iff their numbers are equal.
This can be done as follows: recursively assign a number to each of the childs' subtrees, then sort all the obtained numbers. Call the resulting vector a *profile* of the subtree.
Note that different profiles correspond to non-isomorphic trees, and vice versa.

A. Automorphism

First, can we solve the problem for a rooted tree?
The general idea resembles the algorithm for checking rooted trees isomorphism.
We would like to assign a number to each subtree in such a way that subtrees are isomorphic iff their numbers are equal.
This can be done as follows: recursively assign a number to each of the childs' subtrees, then sort all the obtained numbers. Call the resulting vector a *profile* of the subtree.
Note that different profiles correspond to non-isomorphic trees, and vice versa.
Store a global list of different profiles, with the mapping from profiles to unique numbers. If the current profile was not met before, assign it a new number and remember the profile.

A. Automorphism

How to count the number of isomorphisms?

A. Automorphism

How to count the number of isomorphisms?
For a subtree with a given profile, the answer (number of isomorphisms) is the product of answers for subtrees, multiplied by the number of ways to permute the children.

A. Automorphism

How to count the number of isomorphisms?
For a subtree with a given profile, the answer (number of isomorphisms) is the product of answers for subtrees, multiplied by the number of ways to permute the children.
If there are k isomorphic subtrees of some type, we multiply the answer by $k!$.

A. Automorphism

How to count the number of isomorphisms?
For a subtree with a given profile, the answer (number of isomorphisms) is the product of answers for subtrees, multiplied by the number of ways to permute the children.
If there are k isomorphic subtrees of some type, we multiply the answer by $k!$.
Together we mapping numbers and building profiles, this algorithm works in $O(n \log n)$.

A. Automorphism

How to deal with general (unrooted) trees?

A. Automorphism

How to deal with general (unrooted) trees?

Definition

A vertex v is called a *centroid* of the tree if all its subtrees contain at most $n/2$ vertices.

A. Automorphism

How to deal with general (unrooted) trees?

Definition

A vertex v is called a *centroid* of the tree if all its subtrees contain at most $n/2$ vertices.

Fact

A tree may only contain a unique centroid, or a pair of adjacent centroids.

A. Automorphism

How to deal with general (unrooted) trees?

Definition

A vertex v is called a *centroid* of the tree if all its subtrees contain at most $n/2$ vertices.

Fact

A tree may only contain a unique centroid, or a pair of adjacent centroids.

Observation

Any automorphism maps a centroid to a centroid (possibly the same).

A. Automorphism

- If the tree contains a unique centroid, then it is a fixed point of any automorphism, and the problem is reduced to the rooted tree problem.

A. Automorphism

- If the tree contains a unique centroid, then it is a fixed point of any automorphism, and the problem is reduced to the rooted tree problem.
- If there are two centroids, they may stay in their places, or swap their positions.

A. Automorphism

- If the tree contains a unique centroid, then it is a fixed point of any automorphism, and the problem is reduced to the rooted tree problem.
- If there are two centroids, they may stay in their places, or swap their positions.
  If their subtrees (obtained by removing the edge connecting the centroids) are non-isomorphic, then the centroids stay in their places, and each one is associated with the rooted tree problem.

A. Automorphism

- If the tree contains a unique centroid, then it is a fixed point of any automorphism, and the problem is reduced to the rooted tree problem.
- If there are two centroids, they may stay in their places, or swap their positions.
  If their subtrees (obtained by removing the edge connecting the centroids) are non-isomorphic, then the centroids stay in their places, and each one is associated with the rooted tree problem.
  Otherwise, the centroids may be swapped. As their subtrees are isomorphic, the total answer is $2x^2$, where x is the answer for a subtree of a single centroid.

A. Automorphism

- If the tree contains a unique centroid, then it is a fixed point of any automorphism, and the problem is reduced to the rooted tree problem.
- If there are two centroids, they may stay in their places, or swap their positions.
  If their subtrees (obtained by removing the edge connecting the centroids) are non-isomorphic, then the centroids stay in their places, and each one is associated with the rooted tree problem.
  Otherwise, the centroids may be swapped. As their subtrees are isomorphic, the total answer is $2x^2$, where x is the answer for a subtree of a single centroid.

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

A. Automorphism

- If the tree contains a unique centroid, then it is a fixed point of any automorphism, and the problem is reduced to the rooted tree problem.
- If there are two centroids, they may stay in their places, or swap their positions.
  If their subtrees (obtained by removing the edge connecting the centroids) are non-isomorphic, then the centroids stay in their places, and each one is associated with the rooted tree problem.
  Otherwise, the centroids may be swapped. As their subtrees are isomorphic, the total answer is $2x^2$, where x is the answer for a subtree of a single centroid.

The complexity is still $O(n \log n)$.
Note that assigning unique numbers to non-isomorphic trees can be done in linear time (without hashing).

B. Laser Billiards

In a rectangle there are several vertical and horizontal beams. Ball moves inside the rectangle, reflecting from its sides after collisions. For several values of initial position, velocity and time interval, count the number of different intersection moments with beams.

B. Laser Billiards

In a rectangle there are several vertical and horizontal beams. Ball moves inside the rectangle, reflecting from its sides after collisions. For several values of initial position, velocity and time interval, count the number of different intersection moments with beams. We can shrink the ball and expand the beams equally. After that, we can assume that a point flies in the rectangle and collides with vertical and horizontal strips.

B. Laser Billiards

We have to count each intersection once, even when we collide with two beams at one moment.

B. Laser Billiards

We have to count each intersection once, even when we collide with two beams at one moment.
If we should have counted such moments twice, the problem would be independent for vertical and horizontal beams collision.

B. Laser Billiards

We have to count each intersection once, even when we collide with two beams at one moment.
If we should have counted such moments twice, the problem would be independent for vertical and horizontal beams collision.
The problem for one type of beams is fairly easy: for, say, vertical beams we can use reflection principle over vertical edges to obtain a periodical infinite sequence of strips. After that, counting reduces to range queries on the periodical array.

B. Laser Billiards

We have to count each intersection once, even when we collide with two beams at one moment.
If we should have counted such moments twice, the problem would be independent for vertical and horizontal beams collision.
The problem for one type of beams is fairly easy: for, say, vertical beams we can use reflection principle over vertical edges to obtain a periodical infinite sequence of strips. After that, counting reduces to range queries on the periodical array.
However, we counted double collisions twice, and now have to subtract them back.

B. Laser Billiards

There are $O(n + m)$ diagonal segments between borders of the rectangle.

B. Laser Billiards

There are $O(n + m)$ diagonal segments between borders of the rectangle.
Let us count the number of double collisions on some segment.
This reduces to counting the number of pairs of vertical and horizontal strips at offsets x and y such that $x - y = d$ or $x + y = d$ depending on direction of the diagonal.

B. Laser Billiards

There are $O(n+m)$ diagonal segments between borders of the rectangle.
Let us count the number of double collisions on some segment.
This reduces to counting the number of pairs of vertical and horizontal strips at offsets x and y such that $x - y = d$ or $x + y = d$ depending on direction of the diagonal.
These numbers can be counted all at once using FFT for fast polynomial multiplication.

B. Laser Billiards

It suffices to count double collisions along the path.

B. Laser Billiards

It suffices to count double collisions along the path.
We can count the number of segments lying on the path using the reflection principle.

B. Laser Billiards

It suffices to count double collisions along the path.
We can count the number of segments lying on the path using the reflection principle.
As the number of segments can be large, we should use faster summation methods. For example, count the DP "total number of double collisions starting at current segment and following 2^k segments forward" to sum along path of length l in $O(\log l)$ time.

B. Laser Billiards

It suffices to count double collisions along the path.
We can count the number of segments lying on the path using the reflection principle.
As the number of segments can be large, we should use faster summation methods. For example, count the DP "total number of double collisions starting at current segment and following 2^k segments forward" to sum along path of length l in $O(\log l)$ time.
It suffices to account for proper parts of start and finish segments.
This can be done using bitmasks in $\sim ((n+m)/32)$ time.

C. SmartDog

A dog and a cat are placed inside a labyrinth. On every turn cat moves to a random adjacent cell, and a dog either hears nothing or hears a distorted located where the cat is located. After that, dog moves so that to minimize expected shortest path to cat (to its knowledge). Given what dog hears, simulate dog's movements.

C. SmartDog

To simulate movement, we just have to know the probability of the event “cat is in the cell (x, y)” for each cell of the field at any moment.

C. SmartDog

To simulate movement, we just have to know the probability of the event “cat is in the cell (x, y)” for each cell of the field at any moment.
Initially, all the probabilities are equal.

C. SmartDog

To simulate movement, we just have to know the probability of the event "cat is in the cell (x, y)" for each cell of the field at any moment.
Initially, all the probabilities are equal.
Consider a move. Let $p(x, y)$ be the probability for the cat to be in the cell (x, y) before the move, and $p'(x, y)$ be the same probability after the move happened. We would like to express p' using values of p.

C. SmartDog

To simulate movement, we just have to know the probability of the event "cat is in the cell (x, y)" for each cell of the field at any moment.
Initially, all the probabilities are equal.
Consider a move. Let $p(x, y)$ be the probability for the cat to be in the cell (x, y) before the move, and $p'(x, y)$ be the same probability after the move happened. We would like to express p' using values of p.

- Suppose that on that current move dog heard nothing. Consider a cell (x, y).

C. SmartDog

To simulate movement, we just have to know the probability of the event "cat is in the cell (x, y)" for each cell of the field at any moment.
Initially, all the probabilities are equal.
Consider a move. Let $p(x, y)$ be the probability for the cat to be in the cell (x, y) before the move, and $p'(x, y)$ be the same probability after the move happened. We would like to express p' using values of p.

- Suppose that on that current move dog heard nothing.
  Consider a cell (x, y).
  Then, consider all adjacent cells (let their number be d). For each adjacent cell (x', y'), add $p(x, y)/d$ to $p'(x', y')$; this corresponds to the uniform choosing of direction.

C. SmartDog

- Suppose that the dog just heard the location (x, y), and the current time is t. Let us thoroughly analyse the new probabilities.

C. SmartDog

- Suppose that the dog just heard the location (x, y), and the current time is t. Let us thoroughly analyse the new probabilities.
  Let $L(x_i, y_i, t_i)$ be the event "the cat is at the position (x_i, y_i) at the moment t_i"

C. SmartDog

- Suppose that the dog just heard the location (x, y), and the current time is t. Let us thoroughly analyse the new probabilities.
  Let $L(x_i, y_i, t_i)$ be the event "the cat is at the position (x_i, y_i) at the moment t_i"
  Let $H(x_i, y_i, t_i)$ be the event "the dog heard noise at the position (x_i, y_i) at the moment t_i".

C. SmartDog

For some cell (X, Y), $p'(X, Y)$ should be equal to $\mathrm{Prob}(L(X, Y, t) | H(x, y, t), H(x_1, y_1, t_1), \ldots)$, where (x_i, y_i, t_i) are all previous events "dog heard noise".

C. SmartDog

For some cell (X, Y), $p'(X, Y)$ should be equal to $\mathrm{Prob}(L(X, Y, t) | H(x, y, t), H(x_1, y_1, t_1), \ldots)$, where (x_i, y_i, t_i) are all previous events "dog heard noise".

$$\mathrm{Prob}(L(X, Y, t) | H(x, y, t), H(x_1, y_1, t_1), \ldots) = \frac{\mathrm{Prob}(L(X, Y, t), H(x, y, t), H(x_1, y_1, t_1), \ldots)}{\mathrm{Prob}(H(x, y, t), H(x_1, y_1, t_1), \ldots)}$$

C. SmartDog

For some cell (X, Y), $p'(X, Y)$ should be equal to $\mathrm{Prob}(L(X, Y, t) | H(x, y, t), H(x_1, y_1, t_1), \ldots)$, where (x_i, y_i, t_i) are all previous events "dog heard noise".

$$\mathrm{Prob}(L(X, Y, t) | H(x, y, t), H(x_1, y_1, t_1), \ldots) = \frac{\mathrm{Prob}(L(X, Y, t), H(x, y, t), H(x_1, y_1, t_1), \ldots)}{\mathrm{Prob}(H(x, y, t), H(x_1, y_1, t_1), \ldots)}$$

Note that denominator is the same for all cells (X, Y), so we can omit it and normalize the distribution later.

C. SmartDog

$$\mathrm{Prob}(L(X,Y,t), H(x,y,t), H(x_1,y_1,t_1),\ldots) =$$

C. SmartDog

$$\mathrm{Prob}(L(X,Y,t),H(x,y,t),H(x_1,y_1,t_1),\ldots)=$$

$$\mathrm{Prob}(L(X,Y,t),H(x_1,y_1,t_1),\ldots)\cdot$$
$$\mathrm{Prob}(H(x,y,t)|L(X,Y,t),H(x_1,y_1,t_1),\ldots)$$

C. SmartDog

$$\mathrm{Prob}(L(X,Y,t), H(x,y,t), H(x_1,y_1,t_1),\ldots) =$$

$$\mathrm{Prob}(L(X,Y,t), H(x_1,y_1,t_1),\ldots)\cdot$$
$$\mathrm{Prob}(H(x,y,t)|L(X,Y,t), H(x_1,y_1,t_1),\ldots)$$

$$p(X,Y)\mathrm{Prob}(H(x_1,y_1,t_1),\ldots)\cdot\mathrm{Prob}(H(x,y,t)|L(X,Y,t))$$

C. SmartDog

$$\mathrm{Prob}(L(X,Y,t), H(x,y,t), H(x_1,y_1,t_1),\ldots) =$$

$$\mathrm{Prob}(L(X,Y,t), H(x_1,y_1,t_1),\ldots)\cdot$$
$$\mathrm{Prob}(H(x,y,t)|L(X,Y,t), H(x_1,y_1,t_1),\ldots)$$

$$p(X,Y)\mathrm{Prob}(H(x_1,y_1,t_1),\ldots)\cdot\mathrm{Prob}(H(x,y,t)|L(X,Y,t))$$

Again, $\mathrm{Prob}(H(x_1,y_1,t_1),\ldots)$ can be omitted.

C. SmartDog

$$\mathrm{Prob}(L(X,Y,t), H(x,y,t), H(x_1,y_1,t_1), \dots) =$$

$$\mathrm{Prob}(L(X,Y,t), H(x_1,y_1,t_1), \dots) \cdot$$
$$\mathrm{Prob}(H(x,y,t) | L(X,Y,t), H(x_1,y_1,t_1), \dots)$$

$$p(X,Y)\mathrm{Prob}(H(x_1,y_1,t_1), \dots) \cdot \mathrm{Prob}(H(x,y,t) | L(X,Y,t))$$

Again, $\mathrm{Prob}(H(x_1,y_1,t_1), \dots)$ can be omitted.
$\mathrm{Prob}(H(x,y,t)|L(X,Y,t))$ is by definition $Ne^{-((x-X)^2+(y-Y)^2)/2\sigma^2}$.

C. SmartDog

$$\mathrm{Prob}(L(X,Y,t),H(x,y,t),H(x_1,y_1,t_1),\ldots)=$$

$$\mathrm{Prob}(L(X,Y,t),H(x_1,y_1,t_1),\ldots)\cdot$$
$$\mathrm{Prob}(H(x,y,t)|L(X,Y,t),H(x_1,y_1,t_1),\ldots)$$

$$p(X,Y)\mathrm{Prob}(H(x_1,y_1,t_1),\ldots)\cdot\mathrm{Prob}(H(x,y,t)|L(X,Y,t))$$

Again, $\mathrm{Prob}(H(x_1,y_1,t_1),\ldots)$ can be omitted.
$\mathrm{Prob}(H(x,y,t)|L(X,Y,t))$ is by definition
$Ne^{-((x-X)^2+(y-Y)^2)/2\sigma^2}$.
To sum up, $p'(X,Y)$ is just $p(x,y)$ multiplied by probability of hearing noise at (x,y) if the cat is at (X,Y) (which makes sense).

C. SmartDog

Finally, for each option of movement calculate expected shortest path to the cat and choose the best. Pay attention to tolerance of comparison between probabilities.

C. SmartDog

Finally, for each option of movement calculate expected shortest path to the cat and choose the best. Pay attention to tolerance of comparison between probabilities.
Complexity is $O(knm + (nm)^3)$ (the latter summand for calculating all shortest paths).

D. Expression

Given an arithmetic expression, erase as much bracket pairs as possible so that the expression keeps its original meaning.
More specifically, we can assume that multiplication/division has higher priority than addition/subtraction, and addition and multiplication are associative. Also, subtraction can be combined with addition if it is a second operation, and similarly division can be combined with miltiplication.
No more assumptions can be made about the operations.

D. Expression

Let's decide for each bracket pair whether we can remove it or not, starting from the inner ones.

D. Expression

Let's decide for each bracket pair whether we can remove it or not, starting from the inner ones.
We will use following notation (context-free grammar):

- term $:=$ $a..z|(\text{expression})$
- product $:=$ $\text{term}|\text{term} * \text{product}|\text{term}/\text{product}$
- expression $:=$ $\text{product}|\text{product} + \text{expression}|\text{product} - \text{expression}$

D. Expression

Consider a term t of the form (expression). We assume that all brackets inside the term have been removed when possible.

D. Expression

Consider a term t of the form (expression). We assume that all brackets inside the term have been removed when possible.
There is a unique maximal (unextendable) product containing the term, and unique maximal expression containing the product as an operand of + or –.

D. Expression

Consider a term t of the form (expression). We assume that all brackets inside the term have been removed when possible.
There is a unique maximal (unextendable) product containing the term, and unique maximal expression containing the product as an operand of + or –.
There are several cases to consider.

D. Expression

Consider a term t of the form (expression). We assume that all brackets inside the term have been removed when possible.
There is a unique maximal (unextendable) product containing the term, and unique maximal expression containing the product as an operand of + or −.
There are several cases to consider.

- If the expression inside of brackets is a letter, then the brackets can be removed indefinitely.

D. Expression

Consider a term t of the form (expression). We assume that all brackets inside the term have been removed when possible.
There is a unique maximal (unextendable) product containing the term, and unique maximal expression containing the product as an operand of + or –.
There are several cases to consider.

- If the expression inside of brackets is a letter, then the brackets can be removed indefinitely.
- If the expression inside of brackets is a product (according to the described context-free grammar), the brackets can be removed iff there is no division (/) immediately before the term t.

D. Expression

Consider a term t of the form (expression). We assume that all brackets inside the term have been removed when possible.
There is a unique maximal (unextendable) product containing the term, and unique maximal expression containing the product as an operand of + or –.
There are several cases to consider.

- If the expression inside of brackets is a letter, then the brackets can be removed indefinitely.
- If the expression inside of brackets is a product (according to the described context-free grammar), the brackets can be removed iff there is no division (/) immediately before the term t.
- Otherwise, the brackets can be removed if the term t is not a proper part of the product (that is, does not take part in the multiplication or division immediately), and also there is no $-$ immediately before the term t.

D. Expression

All these conditions are concerned with symbols immediately next to the brackets, or presence of + or - inside the brackets, but not inside inner pairs of brackets.

D. Expression

All these conditions are concerned with symbols immediately next to the brackets, or presence of + or - inside the brackets, but not inside inner pairs of brackets.
These can be checked easily along with parsing the expression in linear time.

E. Paint

Paint the edges of a graph into white, blue and red in such a way that every white edge is adjacent to a red edge. Minimize total cost if a white edge costs 0, a blue edge costs 1, and a red edge costs 2.

E. Paint

Suppose that we have chosen all the edges that will be red.

E. Paint

Suppose that we have chosen all the edges that will be red.
It now makes sense to paint blue only those edges which are not adjacent to any red edge.

E. Paint

Suppose that we have chosen all the edges that will be red.
It now makes sense to paint blue only those edges which are not adjacent to any red edge.
Note that only the set of vertices covered by red edges matters here.

E. Paint

Suppose that we have chosen all the edges that will be red.
It now makes sense to paint blue only those edges which are not adjacent to any red edge.
Note that only the set of vertices covered by red edges matters here.
For each of the 2^n subsets we will count the minimal number of red edges needed to cover this set. This can be done with a classical subset DP, with transitions "add a single edge".

E. Paint

Suppose that we have chosen all the edges that will be red.
It now makes sense to paint blue only those edges which are not adjacent to any red edge.
Note that only the set of vertices covered by red edges matters here.
For each of the 2^n subsets we will count the minimal number of red edges needed to cover this set. This can be done with a classical subset DP, with transitions "add a single edge".
Having computed this, for each subset count the number of blue edges explicitly.

E. Paint

Suppose that we have chosen all the edges that will be red.
It now makes sense to paint blue only those edges which are not adjacent to any red edge.
Note that only the set of vertices covered by red edges matters here.
For each of the 2^n subsets we will count the minimal number of red edges needed to cover this set. This can be done with a classical subset DP, with transitions "add a single edge".
Having computed this, for each subset count the number of blue edges explicitly.
In total, we obtain a $O(m2^n)$ solution.

F. Parliament

Given a graph with “friend” and “enemy” edges, find the maximal size of a subset S such that friends of elements of S are also in S, and no enemy of an element of S is in S. Also, count the number of different maximal subsets.

F. Parliament

Given a graph with “friend” and “enemy” edges, find the maximal size of a subset S such that friends of elements of S are also in S, and no enemy of an element of S is in S. Also, count the number of different maximal subsets.
The number of “friend” edges is at least $\frac{n(n-1)}{3}$.

F. Parliament

Clearly, we can treat connected components over friend edges as single vertices with sizes as weights.

F. Parliament

Clearly, we can treat connected components over friend edges as single vertices with sizes as weights.
What is the maximal number of such connected components?

F. Parliament

Clearly, we can treat connected components over friend edges as single vertices with sizes as weights.
What is the maximal number of such connected components?

Observation

Maximal number of components is achieved when all the edges are in the single “clique” component.

F. Parliament

Clearly, we can treat connected components over friend edges as single vertices with sizes as weights.
What is the maximal number of such connected components?

Observation

Maximal number of components is achieved when all the edges are in the single “clique” component.

The size of the clique is at least $\sim n\sqrt{\frac{2}{3}}$.

F. Parliament

Clearly, we can treat connected components over friend edges as single vertices with sizes as weights.
What is the maximal number of such connected components?

Observation

Maximal number of components is achieved when all the edges are in the single “clique” component.

The size of the clique is at least $\sim n\sqrt{\frac{2}{3}}$.

Thus, the maximal number of components is $\sim n(1 - \sqrt{\frac{2}{3}})$.

F. Parliament

Clearly, we can treat connected components over friend edges as single vertices with sizes as weights.
What is the maximal number of such connected components?

Observation

Maximal number of components is achieved when all the edges are in the single "clique" component.

The size of the clique is at least $\sim n\sqrt{\frac{2}{3}}$.

Thus, the maximal number of components is $\sim n(1 - \sqrt{\frac{2}{3}})$.
For $n \leqslant 250$, this number does not exceed 46.

F. Parliament

After compressing components, the problem becomes basically equal to the maximal independent set problem, but with weights.

F. Parliament

After compressing components, the problem becomes basically equal to the maximal independent set problem, but with weights. This problem can solved in $O(n2^{n/2})$ on a graph with n vertices with a “meet-in-the-middle” approach and fairly standard subset DP.

F. Parliament

After compressing components, the problem becomes basically equal to the maximal independent set problem, but with weights. This problem can solved in $O(n2^{n/2})$ on a graph with n vertices with a “meet-in-the-middle” approach and fairly standard subset DP.

The total complexity is roughly $O(n2^{(1-\sqrt{\frac{2}{3}})/2})$.

G. Sequences

We are given an array of n numbers. Consider all p-element decreasing sequences and order them lexicographically by sequences of indices. Find k-th lexicographically p-element decreasing sequence for several values of k.

G. Sequences

First, we will count $A_{q,i}$ the number of q-element decreasing sequences starting at the index i for all indices, and all values of q from 1 to p.

G. Sequences

First, we will count $A_{q,i}$ the number of q-element decreasing sequences starting at the index i for all indices, and all values of q from 1 to p.
This is a standard application of Fenwick tree (BIT) or segment tree.

G. Sequences

First, we will count $A_{q,i}$ the number of q-element decreasing sequences starting at the index i for all indices, and all values of q from 1 to p.
This is a standard application of Fenwick tree (BIT) or segment tree.
Given this information, we can find k-th (0-based, for convenience) decreasing sequence as follows:

G. Sequences

First, we will count $A_{q,i}$ the number of q-element decreasing sequences starting at the index i for all indices, and all values of q from 1 to p.

This is a standard application of Fenwick tree (BIT) or segment tree.

Given this information, we can find k-th (0-based, for convenience) decreasing sequence as follows:

- Determine the first element by moving from left to right and comparing k with $A_{p,i}$. If current $A_{p,i} > k$, then i is the index of the first element, otherwise subtract $A_{p,i}$ from k and continue.

G. Sequences

First, we will count $A_{q,i}$ the number of q-element decreasing sequences starting at the index i for all indices, and all values of q from 1 to p.
This is a standard application of Fenwick tree (BIT) or segment tree.
Given this information, we can find k-th (0-based, for convenience) decreasing sequence as follows:

- Determine the first element by moving from left to right and comparing k with $A_{p,i}$. If current $A_{p,i} > k$, then i is the index of the first element, otherwise subtract $A_{p,i}$ from k and continue.
- Determine all other elements similarly, but skip elements that are not less than the previous chosen element.

G. Sequences

First, we will count $A_{q,i}$ the number of q-element decreasing sequences starting at the index i for all indices, and all values of q from 1 to p.
This is a standard application of Fenwick tree (BIT) or segment tree.
Given this information, we can find k-th (0-based, for convenience) decreasing sequence as follows:

- Determine the first element by moving from left to right and comparing k with $A_{p,i}$. If current $A_{p,i} > k$, then i is the index of the first element, otherwise subtract $A_{p,i}$ from k and continue.
- Determine all other elements similarly, but skip elements that are not less than the previous chosen element.

This allows to answer one query in $O(n)$ time, which is too much to process all queries.

G. Sequences

We will solve the problem off-line and answer all queries simultaneously.

G. Sequences

We will solve the problem off-line and answer all queries simultaneously.
Note that determining the first element can be done in $O(\log n)$ with binary search on the prefix sum array.

G. Sequences

We will solve the problem off-line and answer all queries simultaneously.
Note that determining the first element can be done in $O(\log n)$ with binary search on the prefix sum array.
However, it doesn't work for latter elements since there's no easy way to ignore greater elements.

G. Sequences

We will solve the problem off-line and answer all queries simultaneously.
Note that determining the first element can be done in $O(\log n)$ with binary search on the prefix sum array.
However, it doesn't work for latter elements since there's no easy way to ignore greater elements.
Consider all q from p to 1. We will determine q-th elements of all sequence at once:

G. Sequences

We will solve the problem off-line and answer all queries simultaneously.
Note that determining the first element can be done in $O(\log n)$ with binary search on the prefix sum array.
However, it doesn't work for latter elements since there's no easy way to ignore greater elements.
Consider all q from p to 1. We will determine q-th elements of all sequence at once:

- Sort all events "find the next element of j-th sequence with previous element a_i" and "add element a_i into consideration" so that for each query with element a_i only larger elements are added into BIT.

G. Sequences

We will solve the problem off-line and answer all queries simultaneously.
Note that determining the first element can be done in $O(\log n)$ with binary search on the prefix sum array.
However, it doesn't work for latter elements since there's no easy way to ignore greater elements.
Consider all q from p to 1. We will determine q-th elements of all sequence at once:

- Sort all events "find the next element of j-th sequence with previous element a_i" and "add element a_i into consideration" so that for each query with element a_i only larger elements are added into BIT.
- Process queries with binary search similar to finding the first element (see above).

G. Sequences

This algorithm will answer all queries in $O(np\log n + qp\log^2 n)$.

G. Sequences

This algorithm will answer all queries in $O(np \log n + qp \log^2 n)$. This can be further optimized to $O((n+q)p \log n)$ using tree descent instead of binary search with queries (note that both segment tree and BIT approach can be optimized this way).

H. Squares

Given a set of points in the plane, count different squares with vertices in given points and sides aligned with coordinate axes.

H. Squares

Iterate over x_l — lowest x of vertices of the square. We will count the number of squares with fixed x_l.

H. Squares

Iterate over x_l — lowest x of vertices of the square. We will count the number of squares with fixed x_l.
Actual method of counting depends on number of given points with $x = x_l$.

- If there are few (say, at most k) points with $x = x_l$, we will try all possible pairs of them as left-top and left-bottom vertices.

H. Squares

Iterate over x_l — lowest x of vertices of the square. We will count the number of squares with fixed x_l.
Actual method of counting depends on number of given points with $x = x_l$.

- If there are few (say, at most k) points with $x = x_l$, we will try all possible pairs of them as left-top and left-bottom vertices. If these points are (x_l, y_l) and (x_l, y_r), then the side length of the square is equal to $y_r - y_l$, and other two poitns should be $(x_l + y_r - y_l, y_l)$ and $(x_l + y_r - y_l, y_r)$.

H. Squares

Iterate over x_l — lowest x of vertices of the square. We will count the number of squares with fixed x_l.
Actual method of counting depends on number of given points with $x = x_l$.

- If there are few (say, at most k) points with $x = x_l$, we will try all possible pairs of them as left-top and left-bottom vertices. If these points are (x_l, y_l) and (x_l, y_r), then the side length of the square is equal to $y_r - y_l$, and other two poitns should be $(x_l + y_r - y_l, y_l)$ and $(x_l + y_r - y_l, y_r)$.
  We can check if these two points exist in $O(\log n)$ with binary search or `std::set`-like data structure.

H. Squares

Iterate over x_l — lowest x of vertices of the square. We will count the number of squares with fixed x_l.
Actual method of counting depends on number of given points with $x = x_l$.

- If there are few (say, at most k) points with $x = x_l$, we will try all possible pairs of them as left-top and left-bottom vertices. If these points are (x_l, y_l) and (x_l, y_r), then the side length of the square is equal to $y_r - y_l$, and other two poitns should be $(x_l + y_r - y_l, y_l)$ and $(x_l + y_r - y_l, y_r)$.
  We can check if these two points exist in $O(\log n)$ with binary search or `std::set`-like data structure.
  Complexity for each particular x is $O(k^2 \log n)$, for a total complexity of $O(nk \log n)$, since sum of all k's doesn't exceed n.

H. Squares

- If there are more than k points, we will choose a different strategy. If we choose the right-top point (x_r, y_r) along with x_l, all other points are determined unambigiously. Check if they exist in a similar matter.

H. Squares

- If there are more than k points, we will choose a different strategy. If we choose the right-top point (x_r, y_r) along with x_l, all other points are determined unambigiously. Check if they exist in a similar matter.
  The complexity is $O(\frac{n}{k} n \log n)$, since there are at most $\frac{n}{k}$ x_l's with more than k points.

H. Squares

- If there are more than k points, we will choose a different strategy. If we choose the right-top point (x_r, y_r) along with x_l, all other points are determined unambigiously. Check if they exist in a similar matter.
  The complexity is $O(\frac{n}{k} n \log n)$, since there are at most $\frac{n}{k}$ x_l's with more than k points.

Choosing $k \sim \sqrt{n}$, we obtain the solution with total complexity $O(n\sqrt{n} \log n)$.

I. Tables

We are given k permutations $p_{i,j}$ of n elements. Construct a permutation q on n elements such that

$$\sum_{i=1}^{k}\sum_{j=1}^{n}\min(|p'_{i,j} - q'_j|, 8)$$

is minimized, where p'_i and q' are inverse permutations of p'_i and q'.

I. Tables

This can be viewed as an assignment problem: assign elements of q' with distinct numbers so that to minimize total penalty.

I. Tables

This can be viewed as an assignment problem: assign elements of q' with distinct numbers so that to minimize total penalty.
Of course, general algorithms of solving assignment problem (such as Hungarian algorithm or max-flow min-cost) are inapplicable here, since the number of vertices and edges is too large.

I. Tables

This can be viewed as an assignment problem: assign elements of q' with distinct numbers so that to minimize total penalty.
Of course, general algorithms of solving assignment problem (such as Hungarian algorithm or max-flow min-cost) are inapplicable here, since the number of vertices and edges is too large.
We can, however, reduce the number of edges if we notice that for each position x there are only a few numbers that yield penalty less than $8k$, namely that number within 8 of any $p_{i,x}$.

I. Tables

This can be viewed as an assignment problem: assign elements of q' with distinct numbers so that to minimize total penalty.
Of course, general algorithms of solving assignment problem (such as Hungarian algorithm or max-flow min-cost) are inapplicable here, since the number of vertices and edges is too large.
We can, however, reduce the number of edges if we notice that for each position x there are only a few numbers that yield penalty less than $8k$, namely that number within 8 of any $p_{i,x}$.
Thus we obtain a graph with $O(n)$ vertices and edges. Still, majority of min-cost max-flow algorithms will time out.

I. Tables

This can be viewed as an assignment problem: assign elements of q' with distinct numbers so that to minimize total penalty.
Of course, general algorithms of solving assignment problem (such as Hungarian algorithm or max-flow min-cost) are inapplicable here, since the number of vertices and edges is too large.
We can, however, reduce the number of edges if we notice that for each position x there are only a few numbers that yield penalty less than $8k$, namely that number within 8 of any $p_{i,x}$.
Thus we obtain a graph with $O(n)$ vertices and edges. Still, majority of min-cost max-flow algorithms will time out.
However, this problem can be reduced to regular max-flow with clever network construction.

I. Tables

This can be viewed as an assignment problem: assign elements of q' with distinct numbers so that to minimize total penalty.
Of course, general algorithms of solving assignment problem (such as Hungarian algorithm or max-flow min-cost) are inapplicable here, since the number of vertices and edges is too large.
We can, however, reduce the number of edges if we notice that for each position x there are only a few numbers that yield penalty less than $8k$, namely that number within 8 of any $p_{i,x}$.
Thus we obtain a graph with $O(n)$ vertices and edges. Still, majority of min-cost max-flow algorithms will time out.
However, this problem can be reduced to regular max-flow with clever network construction.
As the network is bipartite, we can use Dinic algorithm to achieve $O(nk\sqrt{n})$ complexity (however, the constant factor is large as the number of edges is roughly $16kn$).

J. Triangle

Count the number of integer points with non-negative coordinates such that $ax + by \leqslant c$ for integer a, b, c.

J. Triangle

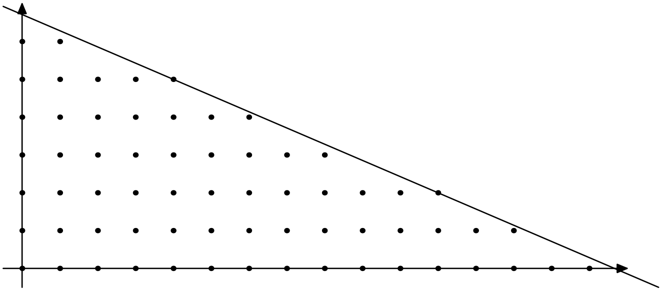
One possible approach is to apply Euclidean-like algorithm.

J. Triangle

One possible approach is to apply Euclidean-like algorithm.
We assume that $GCD(a, b) = 1$, otherwise we can divide A, B and C by $GCD(A, B)$ (with rounding down). Also assume that $A \leqslant B$ and $C \geqslant 0$.

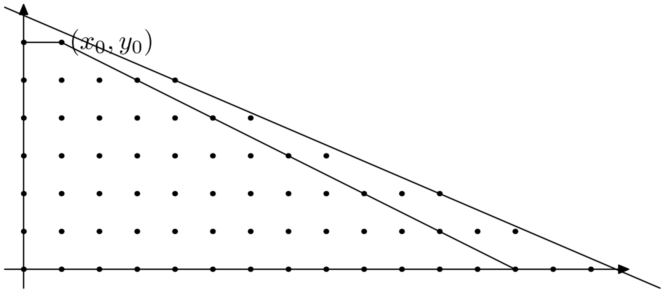
J. Triangle

One possible approach is to apply Euclidean-like algorithm.
We assume that $GCD(a, b) = 1$, otherwise we can divide A, B and C by $GCD(A, B)$ (with rounding down). Also assume that $A \leqslant B$ and $C \geqslant 0$.



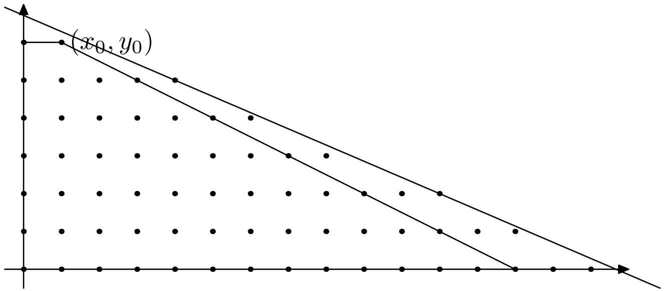
Consider region $ax + by \leqslant c$, $x, y \geqslant 0$.

J. Triangle



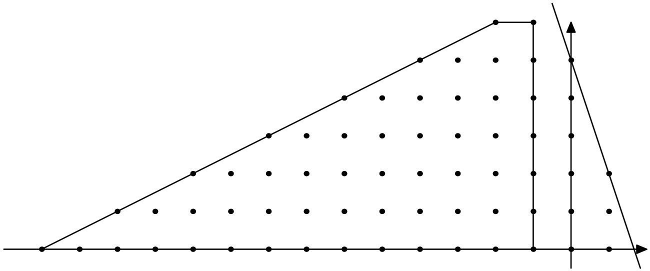
Denote $y_0 = \lfloor \frac{C}{B} \rfloor$, $x_0 = \lfloor \frac{C - By_0}{A} \rfloor$. Point (x_0, y_0) is the rightmost of the top points inside the region.

J. Triangle



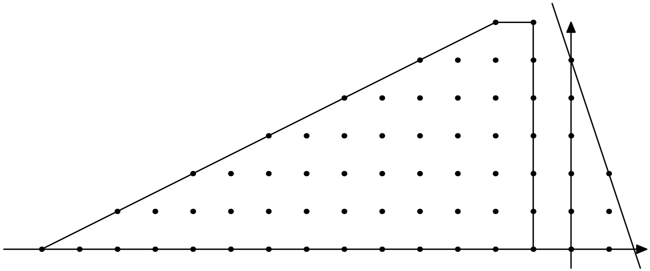
Denote $y_0 = \lfloor \frac{C}{B} \rfloor$, $x_0 = \lfloor \frac{C - By_0}{A} \rfloor$. Point (x_0, y_0) is the rightmost of the top points inside the region.
Let $k = \lfloor \frac{B}{A} \rfloor$. The orthogonal trapezoid with top-right point (x_0, y_0) and right side along the vector $(k, -1)$ is inside the region.

J. Triangle



Apply coordinate tranformation $x \to x - k \cdot (y_0 - y) - x_0 - 1$. This transformation takes the points inside the trapezoid to points with negative x, the number of such points can be counted explicitly.

J. Triangle



Apply coordinate tranformation $x \to x - k \cdot (y_0 - y) - x_0 - 1$. This transformation takes the points inside the trapezoid to points with negative x, the number of such points can be counted explicitly. The rest is a region $a'x + b'y \leqslant c'$, where $b' = b$, $a' = b - ka$. Count the number of points in this region recursively.

J. Triangle

Since we're basically applying Euclid's algorithm to A and B, the number of iterations, and thus the total complexity will be $O(\log(a+b))$.