Long Contest Editorial

November 17, 2015

Moscow International Workshop ACM ICPC, MIPT, 2015

A. Bus Routes

Count the number of graphs on n vertices such that:

- the graph is connected
- the graph contains at least one cycle
- each edge can be colored into one of m colors

The graphs are equivalent iff the set of their edges coincide as well as their colors.

A. Bus Routes

We have to count m-colored connected graphs, and then subtract the number of m-colored trees.

A. Bus Routes

We have to count m-colored connected graphs, and then subtract the number of m-colored trees.
From Cayley's formula, the number of m-colored trees is $n^{n-2}m^{n-1}$.

A. Bus Routes

We have to count m-colored connected graphs, and then subtract the number of m-colored trees.

From Cayley's formula, the number of m-colored trees is $n^{n-2}m^{n-1}$.

Let's count the number of m-colored connected graphs (in the following we will omit m-colored).

A. Bus Routes

We have to count m-colored connected graphs, and then subtract the number of m-colored trees.
From Cayley's formula, the number of m-colored trees is $n^{n-2}m^{n-1}$.
Let's count the number of m-colored connected graphs (in the following we will omit m-colored).
Denote $conn_n$ the number of connected graphs on n vertices. By considering every disconnected graph, and its component containing vertex 1, we obtain the formula:

$$conn_n = (m+1)^{\frac{n(n-1)}{2}} - \sum_{k=1}^{n-1} \binom{n-1}{k-1} conn_k (m+1)^{\frac{(n-k)(n-k-1)}{2}}$$

A. Bus Routes

We have to count m-colored connected graphs, and then subtract the number of m-colored trees.
From Cayley's formula, the number of m-colored trees is $n^{n-2}m^{n-1}$.
Let's count the number of m-colored connected graphs (in the following we will omit m-colored).
Denote $conn_n$ the number of connected graphs on n vertices. By considering every disconnected graph, and its component containing vertex 1, we obtain the formula:

$$conn_n = (m+1)^{\frac{n(n-1)}{2}} - \sum_{k=1}^{n-1} \binom{n-1}{k-1} conn_k (m+1)^{\frac{(n-k)(n-k-1)}{2}}$$

Unfortunately, straightforward calculation of this recurrence takes too long.

A. Bus Routes

We have to use exponential generating functions.

A. Bus Routes

We have to use exponential generating functions.

Definition

An *exponential generating function* $f(x)$ of a sequence $a_0, a_1, \ldots$ is defined as a formal sum

$$f(x) = \sum_{k=0}^{\infty} x^k \frac{a_k}{k!}$$

A. Bus Routes

We have to use exponential generating functions.

Definition

An *exponential generating function* $f(x)$ of a sequence $a_0, a_1, \ldots$ is defined as a formal sum

$$f(x) = \sum_{k=0}^{\infty} x^k \frac{a_k}{k!}$$

Let $f(x)$ be the exp. g. f. of numbers of connected m-colored graphs, and $g(x)$ be the exp. g. f. of numbers of all m-colored graphs.

A. Bus Routes

We have to use exponential generating functions.

Definition

An *exponential generating function* $f(x)$ of a sequence $a_0, a_1, \ldots$ is defined as a formal sum

$$f(x) = \sum_{k=0}^{\infty} x^k \frac{a_k}{k!}$$

Let $f(x)$ be the exp. g. f. of numbers of connected m-colored graphs, and $g(x)$ be the exp. g. f. of numbers of all m-colored graphs.

Observation

$g(x) = e^{f(x)}$

A. Bus Routes

Proof of the observation

$$e^{f(x)} = \sum_{k=0}^{\infty} \frac{f(x)^k}{k!} =$$

A. Bus Routes

Proof of the observation

$$e^{f(x)} = \sum_{k=0}^{\infty} \frac{f(x)^k}{k!} = \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{a_1,\dots,a_k \geqslant 0} x^{a_1+\dots+a_k} \frac{conn_{a_1} \dots conn_{a_k}}{a_1! \dots a_k!}$$

A. Bus Routes

Proof of the observation

$$e^{f(x)} = \sum_{k=0}^{\infty} \frac{f(x)^k}{k!} = \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{a_1,\ldots,a_k \geqslant 0} x^{a_1+\ldots+a_k} \frac{conn_{a_1} \ldots conn_{a_k}}{a_1! \ldots a_k!}$$

$$= \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{0 \leqslant a_1 \leqslant \ldots \leqslant a_k} \binom{a_1 + \ldots + a_k}{a_1, \ldots, a_k} x^{a_1+\ldots+a_k} \frac{conn_{a_1} \ldots conn_{a_k}}{(a_1 + \ldots + a_k)!}$$

A. Bus Routes

Proof of the observation

$$e^{f(x)} = \sum_{k=0}^{\infty} \frac{f(x)^k}{k!} = \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{a_1,\ldots,a_k \geqslant 0} x^{a_1+\ldots+a_k} \frac{conn_{a_1} \ldots conn_{a_k}}{a_1! \ldots a_k!}$$

$$= \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{0 \leqslant a_1 \leqslant \ldots \leqslant a_k} \binom{a_1 + \ldots + a_k}{a_1, \ldots, a_k} x^{a_1+\ldots+a_k} \frac{conn_{a_1} \ldots conn_{a_k}}{(a_1 + \ldots + a_k)!}$$

$$= \sum_{s=0}^{\infty} x^s \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{a_1+\ldots+a_k=s} \binom{s}{a_1, \ldots, a_k} \frac{conn_{a_1} \ldots conn_{a_k}}{s!}$$

A. Bus Routes

Proof of the observation

$$e^{f(x)} = \sum_{k=0}^{\infty} \frac{f(x)^k}{k!} = \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{a_1,\ldots,a_k \geqslant 0} x^{a_1+\ldots+a_k} \frac{conn_{a_1} \ldots conn_{a_k}}{a_1! \ldots a_k!}$$

$$= \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{0 \leqslant a_1 \leqslant \ldots \leqslant a_k} \binom{a_1 + \ldots + a_k}{a_1, \ldots, a_k} x^{a_1+\ldots+a_k} \frac{conn_{a_1} \ldots conn_{a_k}}{(a_1 + \ldots + a_k)!}$$

$$= \sum_{s=0}^{\infty} x^s \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{a_1+\ldots+a_k=s} \binom{s}{a_1, \ldots, a_k} \frac{conn_{a_1} \ldots conn_{a_k}}{s!}$$

The coefficient at x^s is exactly the number of ways to partite s vertices into subsets and build a connected graph on each of them, divided by $s!$.

A. Bus Routes

Corollary

$f(x) = \ln g(x)$, or $f'(x) = \frac{1}{g'(x)}$.

A. Bus Routes

Corollary

$f(x) = \ln g(x)$, or $f'(x) = \frac{1}{g'(x)}$.

We know $g(x)$, and therefore $g'(x)$. It suffices to find several first coefficients of an inverse function to $g'(x)$.

A. Bus Routes

Suppose we know first k coefficients of the inverse function.

A. Bus Routes

Suppose we know first k coefficients of the inverse function.
That is, $f(x) \equiv A \pmod{x^k}$, $f^{-1}(x) \equiv B \pmod{x^k}$,
$AB \equiv 1 \pmod{x^k}$.

A. Bus Routes

Suppose we know first k coefficients of the inverse function. That is, $f(x) \equiv A \pmod{x^k}$, $f^{-1}(x) \equiv B \pmod{x^k}$, $AB \equiv 1 \pmod{x^k}$.
We would like to find next k coefficients. That is, $f(x) \equiv A + Cx^k \pmod{x^{2k}}$, $f^{-1} \equiv B + Dx^k \pmod{x^{2k}}$, $(A + Cx^k)(B + Dx^k) \equiv 1 \pmod{x^{2k}}$ (here A, B, C, D — polynomials of degree $k - 1$). We have to find D.

A. Bus Routes

Suppose we know first k coefficients of the inverse function. That is, $f(x) \equiv A \pmod{x^k}$, $f^{-1}(x) \equiv B \pmod{x^k}$, $AB \equiv 1 \pmod{x^k}$.
We would like to find next k coefficients. That is, $f(x) \equiv A + Cx^k \pmod{x^{2k}}$, $f^{-1} \equiv B + Dx^k \pmod{x^{2k}}$, $(A + Cx^k)(B + Dx^k) \equiv 1 \pmod{x^{2k}}$ (here A, B, C, D — polynomials of degree $k - 1$). We have to find D.
Transform: $AB + (AD + BC)x^k \equiv 1 \pmod{x^{2k}}$.

A. Bus Routes

Suppose we know first k coefficients of the inverse function.
That is, $f(x) \equiv A \pmod{x^k}$, $f^{-1}(x) \equiv B \pmod{x^k}$,
$AB \equiv 1 \pmod{x^k}$.
We would like to find next k coefficients. That is,
$f(x) \equiv A + Cx^k \pmod{x^{2k}}$, $f^{-1} \equiv B + Dx^k \pmod{x^{2k}}$,
$(A + Cx^k)(B + Dx^k) \equiv 1 \pmod{x^{2k}}$ (here A, B, C, D — polynomials of degree $k - 1$). We have to find D.
Transform: $AB + (AD + BC)x^k \equiv 1 \pmod{x^{2k}}$.
Since $AB \equiv 1 \pmod{x^k}$, we can write $AB \equiv 1 + Zx^k \pmod{x^{2k}}$.

A. Bus Routes

Suppose we know first k coefficients of the inverse function.
That is, $f(x) \equiv A \pmod{x^k}$, $f^{-1}(x) \equiv B \pmod{x^k}$,
$AB \equiv 1 \pmod{x^k}$.
We would like to find next k coefficients. That is,
$f(x) \equiv A + Cx^k \pmod{x^{2k}}$, $f^{-1} \equiv B + Dx^k \pmod{x^{2k}}$,
$(A + Cx^k)(B + Dx^k) \equiv 1 \pmod{x^{2k}}$ (here A, B, C, D — polynomials of degree $k - 1$). We have to find D.
Transform: $AB + (AD + BC)x^k \equiv 1 \pmod{x^{2k}}$.
Since $AB \equiv 1 \pmod{x^k}$, we can write $AB \equiv 1 + Zx^k \pmod{x^{2k}}$.
It follows that $AD \equiv -(Z + BC) \pmod{x^k}$.

A. Bus Routes

Suppose we know first k coefficients of the inverse function.
That is, $f(x) \equiv A \pmod{x^k}$, $f^{-1}(x) \equiv B \pmod{x^k}$,
$AB \equiv 1 \pmod{x^k}$.
We would like to find next k coefficients. That is,
$f(x) \equiv A + Cx^k \pmod{x^{2k}}$, $f^{-1} \equiv B + Dx^k \pmod{x^{2k}}$,
$(A + Cx^k)(B + Dx^k) \equiv 1 \pmod{x^{2k}}$ (here A, B, C, D — polynomials of degree $k - 1$). We have to find D.
Transform: $AB + (AD + BC)x^k \equiv 1 \pmod{x^{2k}}$.
Since $AB \equiv 1 \pmod{x^k}$, we can write $AB \equiv 1 + Zx^k \pmod{x^{2k}}$.
It follows that $AD \equiv -(Z + BC) \pmod{x^k}$.
Multiply both sides by B, obtain $D \equiv -B(Z + BC) \pmod{x^k}$.

A. Bus Routes

Suppose we know first k coefficients of the inverse function.
That is, $f(x) \equiv A \pmod{x^k}$, $f^{-1}(x) \equiv B \pmod{x^k}$, $AB \equiv 1 \pmod{x^k}$.
We would like to find next k coefficients. That is, $f(x) \equiv A + Cx^k \pmod{x^{2k}}$, $f^{-1} \equiv B + Dx^k \pmod{x^{2k}}$, $(A + Cx^k)(B + Dx^k) \equiv 1 \pmod{x^{2k}}$ (here A, B, C, D — polynomials of degree $k-1$). We have to find D.
Transform: $AB + (AD + BC)x^k \equiv 1 \pmod{x^{2k}}$.
Since $AB \equiv 1 \pmod{x^k}$, we can write $AB \equiv 1 + Zx^k \pmod{x^{2k}}$.
It follows that $AD \equiv -(Z + BC) \pmod{x^k}$.
Multiply both sides by B, obtain $D \equiv -B(Z + BC) \pmod{x^k}$.
To sum up, we can find D by $O(1)$ multiplications of polynomials of degree $k-1$.

A. Bus Routes

Suppose we know first k coefficients of the inverse function.
That is, $f(x) \equiv A \pmod{x^k}$, $f^{-1}(x) \equiv B \pmod{x^k}$,
$AB \equiv 1 \pmod{x^k}$.
We would like to find next k coefficients. That is,
$f(x) \equiv A + Cx^k \pmod{x^{2k}}$, $f^{-1} \equiv B + Dx^k \pmod{x^{2k}}$,
$(A + Cx^k)(B + Dx^k) \equiv 1 \pmod{x^{2k}}$ (here A, B, C, D — polynomials of degree $k - 1$). We have to find D.
Transform: $AB + (AD + BC)x^k \equiv 1 \pmod{x^{2k}}$.
Since $AB \equiv 1 \pmod{x^k}$, we can write $AB \equiv 1 + Zx^k \pmod{x^{2k}}$.
It follows that $AD \equiv -(Z + BC) \pmod{x^k}$.
Multiply both sides by B, obtain $D \equiv -B(Z + BC) \pmod{x^k}$.
To sum up, we can find D by $O(1)$ multiplications of polynomials of degree $k - 1$.
If we start from $k = 1$, we will find n-th coefficient of f^{-1} in $O(\log n)$ iterations, and in $O(n \log n)$ time (if we use FFT for polynomial multiplication).

A. Bus Routes

How do we use FFT modulo a prime number?

A. Bus Routes

How do we use FFT modulo a prime number?

Notice that $P = A2^k + 1$, where $2^k > n$.

A. Bus Routes

How do we use FFT modulo a prime number?
Notice that $P = A2^k + 1$, where $2^k > n$.
Find ω such that $\omega^{2^k} \equiv 1 \pmod{P}$, $\omega^{2^k} \not\equiv 1 \pmod{P}$. It exists since there exists a prime root g such that $g^x \equiv m \pmod{P}$ is solvable for all m from 1 to $P - 1$.

A. Bus Routes

How do we use FFT modulo a prime number?
Notice that $P = A2^k + 1$, where $2^k > n$.
Find ω such that $\omega^{2^k} \equiv 1 \pmod{P}$, $\omega^{2^k} \not\equiv 1 \pmod{P}$. It exists since there exists a prime root g such that $g^x \equiv m \pmod{P}$ is solvable for all m from 1 to $P - 1$.
Note that we can use ω as a unity root of order 2^k without changing the formulas.

B. The Robot on the Plane

A robot stands in the cell (x, y) of the grid. On i-th move he can move 3^{i-1} cells in one of four cardinal directions, or stays still. Construct any way to reach the cell (x', y'), or determine that this is impossible.

B. The Robot on the Plane

Denote $\delta_x = x' - x$, $\delta_y = y' - y$, where (x', y') is the target cell, (x, y) is the current cell.

B. The Robot on the Plane

Denote $\delta_x = x' - x$, $\delta_y = y' - y$, where (x', y') is the target cell, (x, y) is the current cell.

All the steps starting from the second one change coordinates by multiples of 3, therefore after the first step δ_x and δ_y must be divisible by 3.

B. The Robot on the Plane

Denote $\delta_x = x' - x$, $\delta_y = y' - y$, where (x', y') is the target cell, (x, y) is the current cell.
All the steps starting from the second one change coordinates by multiples of 3, therefore after the first step δ_x and δ_y must be divisible by 3.

- If δ_x and δ_y are initially divisible by 3, then, clearly, the only possible first step is to stand still.

B. The Robot on the Plane

Denote $\delta_x = x' - x$, $\delta_y = y' - y$, where (x', y') is the target cell, (x, y) is the current cell.
All the steps starting from the second one change coordinates by multiples of 3, therefore after the first step δ_x and δ_y must be divisible by 3.

- If δ_x and δ_y are initially divisible by 3, then, clearly, the only possible first step is to stand still.
- If both δ_x and δ_y are not divisible by 3, then there is no way to make them both divisible by 3 in one step, so there's no way.

B. The Robot on the Plane

Denote $\delta_x = x' - x$, $\delta_y = y' - y$, where (x', y') is the target cell, (x, y) is the current cell.
All the steps starting from the second one change coordinates by multiples of 3, therefore after the first step δ_x and δ_y must be divisible by 3.

- If δ_x and δ_y are initially divisible by 3, then, clearly, the only possible first step is to stand still.
- If both δ_x and δ_y are not divisible by 3, then there is no way to make them both divisible by 3 in one step, so there's no way.
- Otherwise, there is a unique move that makes both coordinates divisible by 3.

B. The Robot on the Plane

After making the first move, we can divide δ_x and δ_y by 3, since all the moves became three times longer. Proceed until $\delta_x = \delta_y = 0$.

B. The Robot on the Plane

After making the first move, we can divide δ_x and δ_y by 3, since all the moves became three times longer. Proceed until $\delta_x = \delta_y = 0$. Complexity is $O(\log(x + y + x' + y'))$.

C. Autopilot System

We are performing random walk on a tree. With probability p_i we move to the vertex v_i, otherwise we move to a random neighbour and pay 1 coin. Find expected number of coints to pay while moving from vertex 1 to vertex n.

C. Autopilot System

If $\sum p_i = 1$, then the answer is:

C. Autopilot System

If $\sum p_i = 1$, then the answer is:

- vertex n is not a teleport vertex — answer is ∞

C. Autopilot System

If $\sum p_i = 1$, then the answer is:

- vertex n is not a teleport vertex — answer is ∞
- vertex n is a teleport vertex v_i — answer is $1/p_i$

C. Autopilot System

Fix vertex n as the root. Compute following values:

C. Autopilot System

Fix vertex n as the root. Compute following values:

- pup_v, eup_v — probability to move to parent of v from v without teleporting, and average number of coins to pay given this condition

C. Autopilot System

Fix vertex n as the root. Compute following values:

- pup_v, eup_v — probability to move to parent of v from v without teleporting, and average number of coins to pay given this condition
- $preach_v$, $ereach_v$ — probability to reach the root from v without teleporting, and average number of coins to pay given this condition

C. Autopilot System

Fix vertex n as the root. Compute following values:

- pup_v, eup_v — probability to move to parent of v from v without teleporting, and average number of coins to pay given this condition
- $preach_v$, $ereach_v$ — probability to reach the root from v without teleporting, and average number of coins to pay given this condition

These values can be computed using simple DFS or two.

C. Autopilot System

Fix vertex n as the root. Compute following values:

- pup_v, eup_v — probability to move to parent of v from v without teleporting, and average number of coins to pay given this condition
- $preach_v$, $ereach_v$ — probability to reach the root from v without teleporting, and average number of coins to pay given this condition

These values can be computed using simple DFS or two.
After this, for every teleport vertex we can write equation $ans_{v_j} = ereach_{v_j} + \sum_{i=1}^{k} p_i ans_{v_i}$. Solve this system using Gauss elimination.

C. Autopilot System

Fix vertex n as the root. Compute following values:

- pup_v, eup_v — probability to move to parent of v from v without teleporting, and average number of coins to pay given this condition
- $preach_v$, $ereach_v$ — probability to reach the root from v without teleporting, and average number of coins to pay given this condition

These values can be computed using simple DFS or two.
After this, for every teleport vertex we can write equation $ans_{v_j} = ereach_{v_j} + \sum_{i=1}^{k} p_i ans_{v_i}$. Solve this system using Gauss elimination.
Complexity $O(n + m^3)$.

D. Immortality of Frog

We are given n segments $[l_i; r_i]$, with $1 \leqslant l_i \leqslant r_i \leqslant n$. Every position k is contained in at most 10 segments which are different from $[1; n]$ (otherwise we have to return 0). Find the number of permutations p_i such that $i \in [l_{p_i}; r_{p_i}]$.

D. Immortality of Frog

For each position k construct a list L_k of indices i such that $k \in [l_i; r_i]$, and $[l_i, r_i] \neq [1; n]$ (call such segments *bad*).

D. Immortality of Frog

For each position k construct a list L_k of indices i such that $k \in [l_i; r_i]$, and $[l_i, r_i] \neq [1; n]$ (call such segments *bad*).
We will consider positions from left to right, and for each position will either choose bad segment $[l_{p_k}; r_{p_k}]$ that contains k, or assign k to one of not-bad segments.

D. Immortality of Frog

For each position k construct a list L_k of indices i such that $k \in [l_i; r_i]$, and $[l_i, r_i] \neq [1; n]$ (call such segments *bad*).
We will consider positions from left to right, and for each position will either choose bad segment $[l_{p_k}; r_{p_k}]$ that contains k, or assign k to one of not-bad segments.
Call a segment $[l_i; r_i]$ *fulfilled* if we have already chosen k such that $p_k = i$.

D. Immortality of Frog

Count DP dp_{k,S_k}, where $S_k \subseteq L_k$ — number of ways to choose p_1, $\dots$, p_k — segments for each of the positions 1, $\dots$, k such that all the segments with $r_i < k$ are fulfilled, and S_k is the set of fulfilled segments among all other bad segments.

D. Immortality of Frog

Count DP dp_{k,S_k}, where $S_k \subseteq L_k$ — number of ways to choose p_1, ..., p_k — segments for each of the positions 1, ..., k such that all the segments with $r_i < k$ are fulfilled, and S_k is the set of fulfilled segments among all other bad segments.
On every transition from k and $k+1$, we either choose $i \in S_{k+1}$ and put $p_{k+1} = i$, or assign k to one of $[1; n]$-segments.

D. Immortality of Frog

Count DP dp_{k,S_k}, where $S_k \subseteq L_k$ — number of ways to choose p_1, ..., p_k — segments for each of the positions 1, ..., k such that all the segments with $r_i < k$ are fulfilled, and S_k is the set of fulfilled segments among all other bad segments.

On every transition from k and $k+1$, we either choose $i \in S_{k+1}$ and put $p_{k+1} = i$, or assign k to one of $[1; n]$-segments.

Also, if for some state dp_{k,S_k} a segment $[l_i, r_i]$ with $r_i = k$ is not in S_k (that is, a segment that is going to finish right now is not assigned with any position k), we have to skip the state and not make any transitions to the next layer.

D. Immortality of Frog

Count DP dp_{k,S_k}, where $S_k \subseteq L_k$ — number of ways to choose p_1, ..., p_k — segments for each of the positions 1, ..., k such that all the segments with $r_i < k$ are fulfilled, and S_k is the set of fulfilled segments among all other bad segments.

On every transition from k and $k+1$, we either choose $i \in S_{k+1}$ and put $p_{k+1} = i$, or assign k to one of $[1; n]$-segments.

Also, if for some state dp_{k,S_k} a segment $[l_i, r_i]$ with $r_i = k$ is not in S_k (that is, a segment that is going to finish right now is not assigned with any position k), we have to skip the state and not make any transitions to the next layer.

Finally, we multiply the answer by $z!$, where z is the number of $[1, n]$-segments, since we have chosen exactly z positions to put into $[1, n]$-segments

D. Immortality of Frog

Count DP dp_{k,S_k}, where $S_k \subseteq L_k$ — number of ways to choose p_1, ..., p_k — segments for each of the positions 1, ..., k such that all the segments with $r_i < k$ are fulfilled, and S_k is the set of fulfilled segments among all other bad segments.
On every transition from k and $k+1$, we either choose $i \in S_{k+1}$ and put $p_{k+1} = i$, or assign k to one of $[1; n]$-segments.
Also, if for some state dp_{k,S_k} a segment $[l_i, r_i]$ with $r_i = k$ is not in S_k (that is, a segment that is going to finish right now is not assigned with any position k), we have to skip the state and not make any transitions to the next layer.
Finally, we multiply the answer by $z!$, where z is the number of $[1, n]$-segments, since we have chosen exactly z positions to put into $[1, n]$-segments
Complexity is $O(n2^k)$

E. Land of farms

We are given an $n \times m$ grid, with several connected regions (ancient farms) in it. We have to choose a subset of cells in the grid, and if we have chosen a cell of an ancient farm, we have to include all other cells of the same ancient farm as well. Maximize the number of connected components of the subset.

E. Land of farms

Choose a subset of ancient farms which we will include in the subset. We have to choose which other cells we include in the subset in the optimal arrangement.

E. Land of farms

Choose a subset of ancient farms which we will include in the subset. We have to choose which other cells we include in the subset in the optimal arrangement.

- We can't include the cells belonging to one of excluded ancient farms.

E. Land of farms

Choose a subset of ancient farms which we will include in the subset. We have to choose which other cells we include in the subset in the optimal arrangement.

- We can't include the cells belonging to one of excluded ancient farms.
- Also, it's clearly not optimal to include cells adjacent to included ancient farms, since this doesn't increase the number of components.

E. Land of farms

Out of other cells, we can choose subset arbitrarily. Clearly, there is an optimal answer such that each component made of non-ancient farms' cells consists of a single cell.

E. Land of farms

Out of other cells, we can choose subset arbitrarily. Clearly, there is an optimal answer such that each component made of non-ancient farms' cells consists of a single cell.
The number of maximal pairwise non-adjacent cells can be found as the independence number of a bipartite graph. It can be expressed using the size of maximal matching, which in turn can be found using Kuhn's algorithm.

E. Land of farms

Out of other cells, we can choose subset arbitrarily. Clearly, there is an optimal answer such that each component made of non-ancient farms' cells consists of a single cell.

The number of maximal pairwise non-adjacent cells can be found as the independence number of a bipartite graph. It can be expressed using the size of maximal matching, which in turn can be found using Kuhn's algorithm.

Complexity is $O(2^d(nm)^3)$, where $d \leqslant 10$ is the number of ancient farms (in practice much faster).

F. Matching Compressed String

We are given a compressed string and a DFA (deterministic finite automaton). Determine if the DFA accepts the compressed string.

F. Matching Compressed String

For convenience, introduce the “sink” state, and add transitions to sink state for all non-existent transitions in the original DFA.

F. Matching Compressed String

For convenience, introduce the "sink" state, and add transitions to sink state for all non-existent transitions in the original DFA.

Parse the compressed string, for each sub-expression s compute a function $\varphi(v)$ that maps a DFA state v to the state $\varphi(v) = f(s_n, f(s_{n-1}, \ldots f(s_1, v)\ldots))$ (that is, start in the state v and feed the expression s, return the resulting state).

F. Matching Compressed String

- If the expression is a letter c, simply map every state v to $f(c, v)$.

F. Matching Compressed String

- If the expression is a letter c, simply map every state v to $f(c, v)$.
- If the expression is concatenation of two or more expressions, construct φ as the composition of corresponding functions for sub-expressions.

F. Matching Compressed String

- If the expression is a letter c, simply map every state v to $f(c, v)$.
- If the expression is concatenation of two or more expressions, construct φ as the composition of corresponding functions for sub-expressions.
- If the expression is of the form n(sub-expression), use binary exponentiation to obtain n-th power of the function for the sub-expression.

F. Matching Compressed String

- If the expression is a letter c, simply map every state v to $f(c, v)$.
- If the expression is concatenation of two or more expressions, construct φ as the composition of corresponding functions for sub-expressions.
- If the expression is of the form n(sub-expression), use binary exponentiation to obtain n-th power of the function for the sub-expression.

Complexity is $O(|s| n \log n)$.

G. Alice's Classified Message

Encode a string as follows:

- start with $i = 0$
- find maximal T such that $s[k; k + T) = s[i; i + T)$ for some $0 \leqslant k < i$ and maximal possible T (for equal T, choose minimal k)
- if T exists, append T and k to the code, add T to i
- else, append -1 and s_i to the code, add 1 to i

G. Alice's Classified Message

Denote s_i the suffix of s starting at i-th character.

G. Alice's Classified Message

Denote s_i the suffix of s starting at i-th character.
We have to answer queries of sort "find maximal LCP (longest common prefix) among pairs (s_i, s_k) for all $0 \leqslant k < i$".

G. Alice's Classified Message

Denote s_i the suffix of s starting at i-th character.
We have to answer queries of sort "find maximal LCP (longest common prefix) among pairs (s_i, s_k) for all $0 \leqslant k < i$".
Construct the suffix array of s, and find LCP's of consecutive suffixes. To find length of the longest LCP of (s_i, s_k) for $0 \leqslant k < i$, find closest positions i_-, i_+ to i in the suffix array, such that $i_-, i_+ < i$, and use RMQ (minimal LCP of consecutive suffix pairs) to find maximal length.

G. Alice's Classified Message

Denote s_i the suffix of s starting at i-th character.
We have to answer queries of sort "find maximal LCP (longest common prefix) among pairs (s_i, s_k) for all $0 \leqslant k < i$".
Construct the suffix array of s, and find LCP's of consecutive suffixes. To find length of the longest LCP of (s_i, s_k) for $0 \leqslant k < i$, find closest positions i_-, i_+ to i in the suffix array, such that $i_-, i_+ < i$, and use RMQ (minimal LCP of consecutive suffix pairs) to find maximal length.
Find minimal k such that corresponding RMQ is maximal. This can be done, for example, using additional RMQ which stores indices of already visited suffixes.

G. Alice's Classified Message

Denote s_i the suffix of s starting at i-th character.
We have to answer queries of sort "find maximal LCP (longest common prefix) among pairs (s_i, s_k) for all $0 \leqslant k < i$".
Construct the suffix array of s, and find LCP's of consecutive suffixes. To find length of the longest LCP of (s_i, s_k) for $0 \leqslant k < i$, find closest positions i_-, i_+ to i in the suffix array, such that $i_-, i_+ < i$, and use RMQ (minimal LCP of consecutive suffix pairs) to find maximal length.
Find minimal k such that corresponding RMQ is maximal. This can be done, for example, using additional RMQ which stores indices of already visited suffixes.
Complexity is $O(n \log^2 n)$.

H. Frog and String

Construct a string of n characters using first k Latin letters such that it contains exactly m distinct subpalindromes.

H. Frog and String

Observation

Number of distinct subpalindromes of string of length n does not exceed n.

H. Frog and String

Observation

Number of distinct subpalindromes of string of length n does not exceed n.

Proof

Proof by induction. We assert that there is at most one suffix-palidnrome of s that has not been occured before.

H. Frog and String

Observation

Number of distinct subpalindromes of string of length n does not exceed n.

Proof

Proof by induction. We assert that there is at most one suffix-palidnrome of s that has not been occured before.
Assume the opposite, let a and b be two longest suffix-palindromes that have not been occured before, $|a| > |b|$.

H. Frog and String

Observation

Number of distinct subpalindromes of string of length n does not exceed n.

Proof

Proof by induction. We assert that there is at most one suffix-palidnrome of s that has not been occured before.
Assume the opposite, let a and b be two longest suffix-palindromes that have not been occured before, $|a| > |b|$.
But since a is a palindrome, b is also a prefix of a, so b occurs earlier in the string.

H. Frog and String

Observation

Number of distinct subpalindromes of string of length n does not exceed n.

Proof

Proof by induction. We assert that there is at most one suffix-palidnrome of s that has not been occured before.
Assume the opposite, let a and b be two longest suffix-palindromes that have not been occured before, $|a| > |b|$.
But since a is a palindrome, b is also a prefix of a, so b occurs earlier in the string.
We arrived at a contradiction, thus the original statement is true.

H. Frog and String

Observation

Number of distinct subpalindromes of string of length n does not exceed n.

Proof

Proof by induction. We assert that there is at most one suffix-palidnrome of s that has not been occured before.
Assume the opposite, let a and b be two longest suffix-palindromes that have not been occured before, $|a| > |b|$.
But since a is a palindrome, b is also a prefix of a, so b occurs earlier in the string.
We arrived at a contradiction, thus the original statement is true.

Thus, if $m > n$, the answer doesn't exist.

H. Frog and String

Consider several cases:

H. Frog and String

Consider several cases:

- If $k = 1$, then, clearly, m must be equal to n.

H. Frog and String

Consider several cases:

- If $k = 1$, then, clearly, m must be equal to n.
- If $n = 1$, m must be 1.

H. Frog and String

Consider several cases:

- If $k = 1$, then, clearly, m must be equal to n.
- If $n = 1$, m must be 1.
- If $n = 2$, m must be 2.

H. Frog and String

Consider several cases:

- If $k = 1$, then, clearly, m must be equal to n.
- If $n = 1$, m must be 1.
- If $n = 2$, m must be 2.
- If $k \geqslant 3$, m must be in between 3 and n. The answer looks as follows: abcabc...(last symbol repeated appropriate number of times)

H. Frog and String

The only case is $k = 2$, $n > 2$. If n is small (say, not greater than 12), solve the problem with brute-force.

H. Frog and String

The only case is $k = 2$, $n > 2$. If n is small (say, not greater than 12), solve the problem with brute-force.
Else, we can construct a block t of size 6 such that we can repeat it arbitrary number of times and the number of distinct subpalindromes stays equal to 8. To obtain greater answers, carefully append appropriate number of equal characters.

I. The Shields

We are given n points in the plane, and m figures obtained by cutting a strip of width w_i from the middle of a rotated square sized d_i (see picture in the problem statement). For each figure, count the number of points inside the figure or on its border.

I. The Shields

Each figure consists of two disjoint right-angled isosceles triangles.

I. The Shields

Each figure consists of two disjoint right-angled isosceles triangles. Let's count the number of points inside each triangle separately and sum them for every figure. Consider each type of triangle separately.

I. The Shields

Each figure consists of two disjoint right-angled isosceles triangles. Let's count the number of points inside each triangle separately and sum them for every figure. Consider each type of triangle separately. Transform coordinates so that each triangle is given by a set of inequalities $x \geqslant x_i$, $y \geqslant y_i$, $x + y \leqslant s_i$.

I. The Shields

Each figure consists of two disjoint right-angled isosceles triangles. Let's count the number of points inside each triangle separately and sum them for every figure. Consider each type of triangle separately. Transform coordinates so that each triangle is given by a set of inequalities $x \geqslant x_i$, $y \geqslant y_i$, $x + y \leqslant s_i$.

A triangle can be represented as a complement of a trapezoid $x + y \leqslant s_i$, $x_i \leqslant x \leqslant s_i - y_i$ to a trapezoid $y < y_i$, $x_i \leqslant x \leqslant s_i - y_i$.

I. The Shields

Each figure consists of two disjoint right-angled isosceles triangles. Let's count the number of points inside each triangle separately and sum them for every figure. Consider each type of triangle separately. Transform coordinates so that each triangle is given by a set of inequalities $x \geqslant x_i$, $y \geqslant y_i$, $x + y \leqslant s_i$.

A triangle can be represented as a complement of a trapezoid $x + y \leqslant s_i$, $x_i \leqslant x \leqslant s_i - y_i$ to a trapezoid $y < y_i$, $x_i \leqslant x \leqslant s_i - y_i$.

For trapezoids of each type we can count number of points inside each trapezoid off-line with sweep-line and RSQ structure in $O(n \log n)$ time (using a suitable coordinate transformation once more).

I. The Shields

Each figure consists of two disjoint right-angled isosceles triangles. Let's count the number of points inside each triangle separately and sum them for every figure. Consider each type of triangle separately. Transform coordinates so that each triangle is given by a set of inequalities $x \geqslant x_i$, $y \geqslant y_i$, $x + y \leqslant s_i$.

A triangle can be represented as a complement of a trapezoid $x + y \leqslant s_i$, $x_i \leqslant x \leqslant s_i - y_i$ to a trapezoid $y < y_i$, $x_i \leqslant x \leqslant s_i - y_i$.

For trapezoids of each type we can count number of points inside each trapezoid off-line with sweep-line and RSQ structure in $O(n \log n)$ time (using a suitable coordinate transformation once more).

Therefore, the whole problem can be solved in $O(n \log n)$ time.

J. Kingdom of Tree

We are given a weighted tree. We can choose number r_i in each vertex i. Road (i, j) is covered if $r_i + r_j \geqslant w(i, j)$, where $w(i, j)$ is the weight of edge between i and j. Minimize $\frac{\sum r_i}{\sum_{\text{edge } (i,j) \text{ is covered}} w(i,j)}$.

J. Kingdom of Tree

Suppose that we want to cover all edges, and minimize sum of r_v.

J. Kingdom of Tree

Suppose that we want to cover all edges, and minimize sum of r_v. Suppose that a leaf vertex v has non-zero r_v, and its only neighbour is u. We can increase r_u by r_v, and set r_v to 0 without changing the sum of r's. Thus, all leaves must have $r_v = 0$.

J. Kingdom of Tree

Suppose that we want to cover all edges, and minimize sum of r_v. Suppose that a leaf vertex v has non-zero r_v, and its only neighbour is u. We can increase r_u by r_v, and set r_v to 0 without changing the sum of r's. Thus, all leaves must have $r_v = 0$. Proceeding in a similar manner, we arrive at the following greedy algorithm:

J. Kingdom of Tree

Suppose that we want to cover all edges, and minimize sum of r_v. Suppose that a leaf vertex v has non-zero r_v, and its only neighbour is u. We can increase r_u by r_v, and set r_v to 0 without changing the sum of r's. Thus, all leaves must have $r_v = 0$. Proceeding in a similar manner, we arrive at the following greedy algorithm:

- Start DFS from arbitrary vertex.

J. Kingdom of Tree

Suppose that we want to cover all edges, and minimize sum of r_v. Suppose that a leaf vertex v has non-zero r_v, and its only neighbour is u. We can increase r_u by r_v, and set r_v to 0 without changing the sum of r's. Thus, all leaves must have $r_v = 0$. Proceeding in a similar manner, we arrive at the following greedy algorithm:

- Start DFS from arbitrary vertex.
- For each vertex v , set r_v so that to cover all edges going to its children in the DFS tree (assuming they can be already partially covered).

J. Kingdom of Tree

Now, we want to minimize $\frac{\sum r_i}{\sum_{\text{edge } (i,j) \text{ is covered}} w(i,j)}$.

J. Kingdom of Tree

Now, we want to minimize $\frac{\sum r_i}{\sum_{\text{edge } (i,j) \text{ is covered}} w(i,j)}$.
Binary search on the answer. We want to check if the resulting fraction is at most x.

J. Kingdom of Tree

Now, we want to minimize $\frac{\sum r_i}{\sum_{\text{edge } (i,j) \text{ is covered}} w(i,j)}$.

Binary search on the answer. We want to check if the resulting fraction is at most x.

This holds iff $\min \sum r_i - x \cdot \sum_{\text{edge } (i,j) \text{ is covered}} w(i,j) \leqslant 0$.

J. Kingdom of Tree

Now, we want to minimize $\frac{\sum r_i}{\sum_{\text{edge } (i,j) \text{ is covered}} w(i,j)}$.

Binary search on the answer. We want to check if the resulting fraction is at most x.

This holds iff $\min \sum r_i - x \cdot \sum_{\text{edge } (i,j) \text{ is covered}} w(i,j) \leqslant 0$.

Thus, we will try to minimize

$P = \sum r_i - x \cdot \sum_{\text{edge } (i,j) \text{ is covered}} w(i,j)$.

J. Kingdom of Tree

Count $dp_{v,l}$ — the minimal value of the P in the subtree of v over all configurations with $r_v = l$, for convenience we will not account for r_v in values of $dp_{v,l}$.

J. Kingdom of Tree

Count $dp_{v,l}$ — the minimal value of the P in the subtree of v over all configurations with $r_v = l$, for convenience we will not account for r_v in values of $dp_{v,l}$.
Consider adding a new subtree of v rooted at u, how should we update $dp_{v,l}$?

J. Kingdom of Tree

Count $dp_{v,l}$ — the minimal value of the P in the subtree of v over all configurations with $r_v = l$, for convenience we will not account for r_v in values of $dp_{v,l}$.
Consider adding a new subtree of v rooted at u, how should we update $dp_{v,l}$?
We can choose not to cover the edge (v, u), then the value of $dp_{v,l}$ should be updated with $dp_{v,l} + dp_{u,l'} + l'$ for all l'.

J. Kingdom of Tree

Count $dp_{v,l}$ — the minimal value of the P in the subtree of v over all configurations with $r_v = l$, for convenience we will not account for r_v in values of $dp_{v,l}$.

Consider adding a new subtree of v rooted at u, how should we update $dp_{v,l}$?

We can choose not to cover the edge (v, u), then the value of $dp_{v,l}$ should be updated with $dp_{v,l} + dp_{u,l'} + l'$ for all l'.

If we choose to cover the edge (v, u), then for each pair of values $dp_{v,l}$ and $dp_{u,r}$ we should update $dp_{v,\max(l,w(u,v)-r)}$ with $dp_{v,l} + dp_{u,r} + r - x \cdot w(v, u)$.

J. Kingdom of Tree

It can be shown that the number of different l among $dp_{v,l}$ does not exceed the size of the subtree rooted at v (if we don't introduce unnecessary states).

J. Kingdom of Tree

It can be shown that the number of different l among $dp_{v,l}$ does not exceed the size of the subtree rooted at v (if we don't introduce unnecessary states).
Indeed, for a leaf vertex there is only $l = 0$.

J. Kingdom of Tree

It can be shown that the number of different l among $dp_{v,l}$ does not exceed the size of the subtree rooted at v (if we don't introduce unnecessary states).

Indeed, for a leaf vertex there is only $l = 0$.

For an inner vertex v, the value of l is either 0, or $w(u, v) - r$ for some child u and r for the existing state $dp_{u,r}$.

J. Kingdom of Tree

It can be shown that the number of different l among $dp_{v,l}$ does not exceed the size of the subtree rooted at v (if we don't introduce unnecessary states).

Indeed, for a leaf vertex there is only $l = 0$.

For an inner vertex v, the value of l is either 0, or $w(u, v) - r$ for some child u and r for the existing state $dp_{u,r}$.

The total number is $1 +$ total size of the subtrees, which is exactly the size of the subtree rooted at v.

J. Kingdom of Tree

It can be shown that the number of different l among $dp_{v,l}$ does not exceed the size of the subtree rooted at v (if we don't introduce unnecessary states).
Indeed, for a leaf vertex there is only $l = 0$.
For an inner vertex v, the value of l is either 0, or $w(u, v) - r$ for some child u and r for the existing state $dp_{u,r}$.
The total number is $1 +$ total size of the subtrees, which is exactly the size of the subtree rooted at v.
It follows that these solution works in $O(n^2 \log n \log^{-1} \varepsilon)$. Here $O(n^2)$ is the total number of transitions (since merging two subtrees of size a and b requires ab transitions), $\log n$ is for set operations (since the set of necessary l in $dp_{v,l}$ is sparse), and $\log^{-1} \varepsilon$ is the number of binary search iterations.