Day 5 Contest

October 1st, 2018

Gleb Evstropov, Mikhail Tikhomirov

3rd Hello Barcelona Workshop

A. A Lot

Solve many equations $X^Z \equiv Y \pmod{p}$ for Y, find the smallest solution.

A. A Lot

r is a *primitive root* modulo p if $r^0, r^1, \ldots, r^{p-2}$ are all distinct integers modulo p (note that $r^{p-1} \equiv 1 \pmod{p}$ — Fermat's little theorem).

A. A Lot

r is a *primitive root* modulo p if $r^0, r^1, \ldots, r^{p-2}$ are all distinct integers modulo p (note that $r^{p-1} \equiv 1 \pmod{p}$ — Fermat's little theorem).

One can solve $r^x \equiv t$ for x as follows (Shanks' baby step-giant step algorithm):

- choose an integer k, set $l = \lfloor \frac{p-1}{k} \rfloor$.
- generate two sets: $A = (x^0, \ldots, x^{k-1})$ and $B = (x^0, x^k, x^{2k}, \ldots, x^{lk})$.
- let $x = ky + z$ with $z < k$, therefore $t \equiv (r^y)^k \cdot r^z \equiv a \cdot b$ with $a \in A$ and $b \in B$.
- iterate over $b \in B$ and check if $tb^{-1} \in A$.

A. A Lot

r is a *primitive root* modulo p if $r^0, r^1, \dots, r^{p-2}$ are all distinct integers modulo p (note that $r^{p-1} \equiv 1 \pmod{p}$ — Fermat's little theorem).

One can solve $r^x \equiv t$ for x as follows (Shanks' baby step-giant step algorithm):

- choose an integer k, set $l = \lfloor \frac{p-1}{k} \rfloor$.
- generate two sets: $A = (x^0, \dots, x^{k-1})$ and $B = (x^0, x^k, x^{2k}, \dots, x^{lk})$.
- let $x = ky + z$ with $z < k$, therefore $t \equiv (r^y)^k \cdot r^z \equiv a \cdot b$ with $a \in A$ and $b \in B$.
- iterate over $b \in B$ and check if $tb^{-1} \in A$.

The complexity of this algorithm is $O(k + l)$ to generate A and B, and $O(l \log k)$ for finding the answer. Note that A and B can be generated only once, and choosing $k \sim 10^6$, $l \sim 100$ processes each query very fast.

A. A Lot

Cases when X or Y are zeros are trivial.

A. A Lot

Cases when X or Y are zeros are trivial.

If both X and Y are non-zero, there are unique solutions to $r^x \equiv X$ and $r^y \equiv Y$. The initial equation now becomes

$$xZ \equiv y \pmod{(p-1)}.$$

A. A Lot

Cases when X or Y are zeros are trivial.

If both X and Y are non-zero, there are unique solutions to $r^x \equiv X$ and $r^y \equiv Y$. The initial equation now becomes

$$xZ \equiv y \pmod{(p-1)}.$$

There are a bunch of methods for solving this; one of them is to obtain answers for all prime powers dividing $p-1$, and unite the answers with Chinese remainder theorem. A straightforward solution is also possible (with some case analysis).

B. Almost Average

Given an array of n numbers, process queries "given a and b, find

$$\max_{a \leqslant l < r \leqslant b} \frac{\sum_{i=l}^{r} a_i}{r - l}."$$

B. Almost Average

Let $p_i = \sum_{j=1}^{i-1} a_j$.

B. Almost Average

Let $p_i = \sum_{j=1}^{i-1} a_j$.

Geometric interpretation: let us create two sequences of red and blue points in the plane with coordinates $r_i = (i, p_{i-1})$ and $b_i = (i, p_i)$. A query can then be restated as: find a pair of points r_a, b_b with $l \leqslant a < b \leqslant r$ with the largest slope of the segment $r_a b_b$.

B. Almost Average

Let $p_i = \sum_{j=1}^{i-1} a_j$.

Geometric interpretation: let us create two sequences of red and blue points in the plane with coordinates $r_i = (i, p_{i-1})$ and $b_i = (i, p_i)$. A query can then be restated as: find a pair of points r_a, b_b with $l \leqslant a < b \leqslant r$ with the largest slope of the segment $r_a b_b$.

How could we solve this problem if we knew that $a \in [l_a, r_a]$ and $b \in [l_b, r_b]$ with $r_a \leqslant l_b$? Construct convex hull of red points in $[l_a, r_a]$ and blue points in $[l_b, r_b]$. The answer is then the *common inward tangent* of the two convex hulls. This tangent can be found in $O(\log n)$ provided that both convex hulls are already constructed.

B. Almost Average

Another trick we will is merging the convex hulls. If two convex hulls are built on non-intersecting segments, we can merge them in $O(\log n)$: locate the two common tangents and cut the parts between them. This may require storing the hulls in an efficient data structure such as a treap.

B. Almost Average

Now for the queries part: construct the segment tree on the initial array. A vertex of the tree corresponding to the segment $[l, r]$ will contain convex hulls of both red and blue points, as well as the answer for the query $[l, r]$.

B. Almost Average

Now for the queries part: construct the segment tree on the initial array. A vertex of the tree corresponding to the segment $[l, r]$ will contain convex hulls of both red and blue points, as well as the answer for the query $[l, r]$.

This answer can be obtained as the largest of three values: answers for the left and the right half, as well as the largest slope starting in the left half and ending in the right one, that can be found as described above.

B. Almost Average

Now for the queries part: construct the segment tree on the initial array. A vertex of the tree corresponding to the segment $[l, r]$ will contain convex hulls of both red and blue points, as well as the answer for the query $[l, r]$.

This answer can be obtained as the largest of three values: answers for the left and the right half, as well as the largest slope starting in the left half and ending in the right one, that can be found as described above.

Now, how do we answer an arbitrary query? Divide the segment $[l, r]$ into $k = O(\log n)$ segments $s_1, \ldots, s_k$ corresponding to vertices of the segment tree. The answer for each of the individual parts is already known.

B. Almost Average

Now for the queries part: construct the segment tree on the initial array. A vertex of the tree corresponding to the segment $[l, r]$ will contain convex hulls of both red and blue points, as well as the answer for the query $[l, r]$.

This answer can be obtained as the largest of three values: answers for the left and the right half, as well as the largest slope starting in the left half and ending in the right one, that can be found as described above.

Now, how do we answer an arbitrary query? Divide the segment $[l, r]$ into $k = O(\log n)$ segments $s_1, \ldots, s_k$ corresponding to vertices of the segment tree. The answer for each of the individual parts is already known.

It suffices to find the largest slope in partitions $s_1 : s_2$, $(s_1 + s_2) : s_3$, $\ldots$, $(s_1 + \ldots + s_{k-1}) : s_k$. Since we are able to merge convex hulls fast, the whole procedure takes $O(\log^2 n)$ per query in total.

C. Amoeba

You are given an undirected tree. For each d from A to B, find the largest number t_d such that it is possible to draw t_d disjoint paths of length d from a vertex of the tree.

C. Amoeba

First, for each edge of the tree we want to find the longest path starting with this edge in both directions. We can do this in two stages: first solve for all edges directed from the root with the standard subtree DP, then use these values to obtain answers for edges directed towards the root.

C. Amoeba

First, for each edge of the tree we want to find the longest path starting with this edge in both directions. We can do this in two stages: first solve for all edges directed from the root with the standard subtree DP, then use these values to obtain answers for edges directed towards the root.

Now, consider a vertex v, and let $l_1 \geqslant \ldots \geqslant l_k$ be the lengths of longest paths leaving v for each edge incident to v. For each d from 1 to k we can now obtain a solution with d paths of any length not exceeding l_d.

C. Amoeba

Let us store the largest d we can obtain for each length, and improve them with all this information for each "center" vertex. Note further that $t_d \leqslant t_{d-1}$, hence for each d we can improve t_{d-1} with t_d.

C. Amoeba

Let us store the largest d we can obtain for each length, and improve them with all this information for each "center" vertex. Note further that $t_d \leqslant t_{d-1}$, hence for each d we can improve t_{d-1} with t_d.

The resulting numbers t_d are the answer. Complexity is $O(n \log n)$ since we have to sort the lists $l_1, \ldots, l_k$ in each vertex.

D. Automaton

Find the largest number of vertices in a minimal DFA that only accepts binary strings of length L, and the largest number of strings accepted by such DFA.

D. Automaton

When is a DFA minimal?

D. Automaton

When is a DFA minimal?

- All states have to be reachable from s — the initial state.

D. Automaton

When is a DFA minimal?

- All states have to be reachable from s — the initial state.
- All states have to have distinct *right contexts* — sets of strings leading to a terminal state starting from the current state. In particular, there should be only one terminal state since its right context consists only of the empty string.

D. Automaton

When is a DFA minimal?

- All states have to be reachable from s — the initial state.
- All states have to have distinct *right contexts* — sets of strings leading to a terminal state starting from the current state. In particular, there should be only one terminal state since its right context consists only of the empty string.
- For each state, the terminal vertex has to be reachable from it.

D. Automaton

When does a minimal DFA accept strings of length L exclusively?

D. Automaton

When does a minimal DFA accept strings of length L exclusively?

The states have to be divided into layers $S_0, \ldots, S_L$, with vertices in the layer S_i at distance i from the starting state (and at distance $L - i$ from the terminal state).

D. Automaton

When does a minimal DFA accept strings of length L exclusively?

The states have to be divided into layers $S_0, \ldots, S_L$, with vertices in the layer S_i at distance i from the starting state (and at distance $L - i$ from the terminal state).

Some upper bounds on sizes of layers: clearly $|S_{i+1}| \leqslant 2|S_i|$ since there are most two transitions leaving each state in S_i.
Consequently, $|S_i| \leqslant 2^i$.

D. Automaton

On the other hand, $|S_L| = 1$. Consider layers S_i and S_{i+1}, and suppose that all states in S_{i+1} are non-equivalent (have distinct right contexts). For the time being, let us include a *rejecting* state R_{i+1} in S_{i+1} as well. If there is no transition from a state of S_i with a particular letter, we create a transition to the rejecting state instead.

D. Automaton

On the other hand, $|S_L| = 1$. Consider layers S_i and S_{i+1}, and suppose that all states in S_{i+1} are non-equivalent (have distinct right contexts). For the time being, let us include a *rejecting* state R_{i+1} in S_{i+1} as well. If there is no transition from a state of S_i with a particular letter, we create a transition to the rejecting state instead.

Two states in S_i are equivalent iff their transitions coincide (the condition of all states of S_{i+1} being non-equivalent is important!). Therefore, there can be as many states as ordered pairs of states of S_{i+1}. However, a pair (R_{i+1}, R_{i+1}) has an empty right context, therefore, it corresponds to the rejecting state R_i.

D. Automaton

On the other hand, $|S_L| = 1$. Consider layers S_i and S_{i+1}, and suppose that all states in S_{i+1} are non-equivalent (have distinct right contexts). For the time being, let us include a *rejecting* state R_{i+1} in S_{i+1} as well. If there is no transition from a state of S_i with a particular letter, we create a transition to the rejecting state instead.

Two states in S_i are equivalent iff their transitions coincide (the condition of all states of S_{i+1} being non-equivalent is important!). Therefore, there can be as many states as ordered pairs of states of S_{i+1}. However, a pair (R_{i+1}, R_{i+1}) has an empty right context, therefore, it corresponds to the rejecting state R_i.

Hence, $1 + |S_i| \leqslant (1 + |S_{i+1}|)^2$ (1 here is for the rejecting states), and $|S_i| \leqslant 2^{2^{L-i}} - 1$ by induction.

D. Automaton

From the above considerations, the best size of the DFA we could hope for is

$$\sum_{i=0}^{L} \min(2^i, 2^{2^{L-i}} - 1).$$

D. Automaton

From the above considerations, the best size of the DFA we could hope for is

$$\sum_{i=0}^{L} \min(2^i, 2^{2^{L-i}} - 1).$$

And we can actually construct such DFA! Let k be the largest number such that $2^k \leqslant 2^{2^{L-k}} - 1$ (note that this is equivalent to $k < 2^{L-k}$).

D. Automaton

From the above considerations, the best size of the DFA we could hope for is

$$\sum_{i=0}^{L} \min(2^i, 2^{2^{L-i}} - 1).$$

And we can actually construct such DFA! Let k be the largest number such that $2^k \leqslant 2^{2^{L-k}} - 1$ (note that this is equivalent to $k < 2^{L-k}$).

Layers $S_0, \ldots, S_k$ will look like a binary tree, and layers $S_{k+1}, \ldots, S_L$ will be constructed greeidly from S_L by taking all possible transition pairs as described above.

D. Automaton

From the above considerations, the best size of the DFA we could hope for is

$$\sum_{i=0}^{L} \min(2^i, 2^{2^{L-i}} - 1).$$

And we can actually construct such DFA! Let k be the largest number such that $2^k \leqslant 2^{2^{L-k}} - 1$ (note that this is equivalent to $k < 2^{L-k}$).

Layers $S_0, \ldots, S_k$ will look like a binary tree, and layers $S_{k+1}, \ldots, S_L$ will be constructed greeidly from S_L by taking all possible transition pairs as described above.

It now suffices to make transitions from S_k to S_{k+1} arbitrarily without violating non-existence requirement.

D. Automaton

From the above considerations, the best size of the DFA we could hope for is

$$\sum_{i=0}^{L} \min(2^i, 2^{2^{L-i}} - 1).$$

And we can actually construct such DFA! Let k be the largest number such that $2^k \leqslant 2^{2^{L-k}} - 1$ (note that this is equivalent to $k < 2^{L-k}$).

Layers $S_0, \ldots, S_k$ will look like a binary tree, and layers $S_{k+1}, \ldots, S_L$ will be constructed greeidly from S_L by taking all possible transition pairs as described above.

It now suffices to make transitions from S_k to S_{k+1} arbitrarily without violating non-existence requirement.

Note further that all possible constructions have to look this way since the bound is tight.

D. Automaton

What about maximizing the number of accepted strings? The only leeway we have is in choosing transitions from S_k to S_{k+1}.

D. Automaton

What about maximizing the number of accepted strings? The only leeway we have is in choosing transitions from S_k to S_{k+1}.

Let $f_{i,j}$ be the number of states at $(L-i)$-th level with exactly j strings in the right context when constructing the automaton from the bottom up (we'll include the rejecting state into $f_{i,0}$ for convenience).

D. Automaton

What about maximizing the number of accepted strings? The only leeway we have is in choosing transitions from S_k to S_{k+1}.

Let $f_{i,j}$ be the number of states at $(L - i)$-th level with exactly j strings in the right context when constructing the automaton from the bottom up (we'll include the rejecting state into $f_{i,0}$ for convenience).

Choosing transitions from S_k to S_{k+1} is the same thing as constructing the layer S'_k by taking all pairs of states in S_{k+1}, and then identifying vertices of S_k with some of the vertices in S'_k.

D. Automaton

What about maximizing the number of accepted strings? The only leeway we have is in choosing transitions from S_k to S_{k+1}.

Let $f_{i,j}$ be the number of states at $(L-i)$-th level with exactly j strings in the right context when constructing the automaton from the bottom up (we'll include the rejecting state into $f_{i,0}$ for convenience).

Choosing transitions from S_k to S_{k+1} is the same thing as constructing the layer S'_k by taking all pairs of states in S_{k+1}, and then identifying vertices of S_k with some of the vertices in S'_k.

Clearly, we should do this greedily: choose the largest possible j with $f_{L-k,j} > 0$, and let $z = \min(f_{L-k,j}, |S_k|)$. Identify z vertices and decrease $|S_k|$ by z. Proceed while $|S_k| > 0$.

D. Automaton

By construction, $f_{0,0} = f_{0,1} = 1$, and

$$f_{i,j} = \sum_{j'=0}^{j} f_{i-1,j'} f_{i-1,j-j'}.$$

D. Automaton

By construction, $f_{0,0} = f_{0,1} = 1$, and

$$f_{i,j} = \sum_{j'=0}^{j} f_{i-1,j'} f_{i-1,j-j'}.$$

Note that since k is the largest number subject to $k < 2^{L-k}$, we have that $k \sim n - \log_2 n$. Therefore, there are only $O(L)$ values of $f_{i,j}$ we have to compute (however, these values are large).

D. Automaton

By construction, $f_{0,0} = f_{0,1} = 1$, and

$$f_{i,j} = \sum_{j'=0}^{j} f_{i-1,j'} f_{i-1,j-j'}.$$

Note that since k is the largest number subject to $k < 2^{L-k}$, we have that $k \sim n - \log_2 n$. Therefore, there are only $O(L)$ values of $f_{i,j}$ we have to compute (however, these values are large).

In total, the complexity is $O(L^3)$, assuming a generous bound $O(L)$ on the size of all intermediate numbers.

E. Average Palindromes

Given a string of n letters and question marks, find $1/n$ times the expected number of odd-length palindromes when replacing question marks with random letters. Output the answer with precision 10^{-6}.

E. Average Palindromes

How would we solve this problem in $O(n^2)$ time? Consider each center position, and expand boundaries l and r of the palindrome.

E. Average Palindromes

How would we solve this problem in $O(n^2)$ time? Consider each center position, and expand boundaries l and r of the palindrome.

Let P be the probability of $s[l+1, r-1]$ being a palindrome, and P' be the same probability for $s[l, r]$. If s_l and s_r are both letters, then $P' = P$ if $s_l = s_r$, and $P' = 0$ if $s_l \neq s_r$. If at least one of s_l and s_r is $\widehat{?}$, then $P' = P/26$.

E. Average Palindromes

Now, a few optimizations:

E. Average Palindromes

Now, a few optimizations:

- If at some stage of this process P is less than, say, 10^{-15}, then we can stop processing this center position since the total impact of subsequent palindromes will be less than required precision. Hence, if we've encountered > 15 question marks, we're free to stop.

E. Average Palindromes

Now, a few optimizations:

- If at some stage of this process P is less than, say, 10^{-15}, then we can stop processing this center position since the total impact of subsequent palindromes will be less than required precision. Hence, if we've encountered > 15 question marks, we're free to stop.
- We can optimize processing indices without question marks. For the current positions l and r, let s be the number of the following steps without meeting question marks. We can then determine $s' \leqslant s$ — the number of steps where no mismatch occurs. This adds $s' \cdot P$ to the answer, and if $s' < s$, the whole process should be aborted.

E. Average Palindromes

Now, a few optimizations:

- If at some stage of this process P is less than, say, 10^{-15}, then we can stop processing this center position since the total impact of subsequent palindromes will be less than required precision. Hence, if we've encountered > 15 question marks, we're free to stop.
- We can optimize processing indices without question marks. For the current positions l and r, let s be the number of the following steps without meeting question marks. We can then determine $s' \leqslant s$ — the number of steps where no mismatch occurs. This adds $s' \cdot P$ to the answer, and if $s' < s$, the whole process should be aborted.
  This part can be implemented, for example, with binary search on s' and rolling hashes for substring comparison.

E. Average Palindromes

The above optimizations yield an $O(n \log n)$ solution: for each center process blocks without question marks in a bunch, and process each question mark directly.

F. Continued Fraction

Convert the fraction $\frac{10^N}{10^M-1}$ to a continued fraction

$$a_0 + \cfrac{1}{a_1 + \frac{1}{a_2 + \ldots}}.$$

F. Continued Fraction

General algorithm: we convert x/y to a continued fraction starting from a_0: $x/y = \lfloor x/y \rfloor + \frac{1}{y/(x \bmod y)}$. We stop when the denominator is 0.

F. Continued Fraction

General algorithm: we convert x/y to a continued fraction starting from a_0: $x/y = \lfloor x/y \rfloor + \frac{1}{y/(x \bmod y)}$. We stop when the denominator is 0.

Let $a \geqslant b$.

$$\frac{10^a}{10^b - 1} = 10^{a-b} + \frac{10^{a-b}}{10^b - 1}.$$

F. Continued Fraction

General algorithm: we convert x/y to a continued fraction starting from a_0: $x/y = \lfloor x/y \rfloor + \frac{1}{y/(x \bmod y)}$. We stop when the denominator is 0.

Let $a \geqslant b$.

$$\frac{10^a}{10^b - 1} = 10^{a-b} + \frac{10^{a-b}}{10^b - 1}.$$

If $a < b$,

$$\frac{10^a}{10^b - 1} = 0 + \frac{1}{\frac{10^b - 1}{10^a}}.$$

F. Continued Fraction

General algorithm: we convert x/y to a continued fraction starting from a_0: $x/y = \lfloor x/y \rfloor + \frac{1}{y/(x \bmod y)}$. We stop when the denominator is 0.

Let $a \geqslant b$.

$$\frac{10^a}{10^b - 1} = 10^{a-b} + \frac{10^{a-b}}{10^b - 1}.$$

If $a < b$,

$$\frac{10^a}{10^b - 1} = 0 + \frac{1}{\frac{10^b - 1}{10^a}}.$$

The resulting fraction only requires two stages:

$$\frac{10^b - 1}{10^a} = 10^{b-a} - 1 + \frac{1}{\frac{10^a}{10^a - 1}} = 10^{b-1} - 1 + \frac{1}{1 + \frac{1}{10^a - 1}}.$$

G. K-plets

You are given a set of n elements and m forbidden subsets of size k. Your goal is to find the number of ways to split the set in subsets of size k with no forbidden subsets appearing among them.

G. K-plets

Consider the case when no two subsets intersect. Using inclusion-exclusion principle we can express the answer as $f(n) - f(n-k) \cdot m + f(n - 2 \cdot k) \cdot \frac{m(m-1)}{2} - \ldots$.

G. K-plets

Consider the case when no two subsets intersect. Using inclusion-exclusion principle we can express the answer as $f(n) - f(n-k) \cdot m + f(n - 2 \cdot k) \cdot \frac{m(m-1)}{2} - \ldots$.

The above is a medium-easy combinatorics. However, in general subsets may intersect. Consider the graph of size m where vertices i and j are connected iff banned subsets i and j do not intersect.

G. K-plets

Consider the case when no two subsets intersect. Using inclusion-exclusion principle we can express the answer as $f(n) - f(n-k) \cdot m + f(n - 2 \cdot k) \cdot \frac{m(m-1)}{2} - \ldots$.

The above is a medium-easy combinatorics. However, in general subsets may intersect. Consider the graph of size m where vertices i and j are connected iff banned subsets i and j do not intersect.

The original formula is transferred to $f(n) - c(1) \cdot f(n-k) + c(2) \cdot f(n-2k) - \ldots$, where $c(x)$ is the number of cliques of size x in this graph.

G. K-plets

How do we compute the number of cliques? The most naive algorithm simply check all subsets in $O(2^m \cdot m^2)$ time. It can be upgraded to $O(2^m \cdot m)$ by using unsigned long long for a bitmask of neighbors. Then we can get $O(2^m)$ by using recursive backtracking.

G. K-plets

How do we compute the number of cliques? The most naive algorithm simply check all subsets in $O(2^m \cdot m^2)$ time. It can be upgraded to $O(2^m \cdot m)$ by using unsigned long long for a bitmask of neighbors. Then we can get $O(2^m)$ by using recursive backtracking.

However, 2^{54} is a lot and we need a significant improvement. Use meet-in-the-middle approach. Split nodes in two subsets of size 27. Let us have a fixed subset of vertices S_l in the left part. We can intersect their list of neighbors to get a valid set of possible elements in the right part N_r.

G. K-plets

How do we compute the number of cliques? The most naive algorithm simply check all subsets in $O(2^m \cdot m^2)$ time. It can be upgraded to $O(2^m \cdot m)$ by using unsigned long long for a bitmask of neighbors. Then we can get $O(2^m)$ by using recursive backtracking.

However, 2^{54} is a lot and we need a significant improvement. Use meet-in-the-middle approach. Split nodes in two subsets of size 27. Let us have a fixed subset of vertices S_l in the left part. We can intersect their list of neighbors to get a valid set of possible elements in the right part N_r.

Every possible right part of the clique will be a subset $S_r \subset N_r$. We can compute values of dynamic programming $d(A, i)$, $A \subset V_r$: how many cliques of size i are subsets of A. The complexity of such solution is $O(2^{\frac{m}{2}} \cdot m^2)$

G. K-plets

That is still too much. How can we speed things up? Notice, that if we split graph in masks of size a and b ($a + b = m$), first part that tries all cliques in the left part works in $O(2^a)$, while second part that computes the exact values runs in $O(2^b \cdot b^2)$.

G. K-plets

That is still too much. How can we speed things up? Notice, that if we split graph in masks of size a and b ($a + b = m$), first part that tries all cliques in the left part works in $O(2^a)$, while second part that computes the exact values runs in $O(2^b \cdot b^2)$.

We can reduce the running time by rebalancing values of a and b to get complexity $O(2^a + 2^b \cdot b^2)$.

G. K-plets

That is still too much. How can we speed things up? Notice, that if we split graph in masks of size a and b ($a + b = m$), first part that tries all cliques in the left part works in $O(2^a)$, while second part that computes the exact values runs in $O(2^b \cdot b^2)$.

We can reduce the running time by rebalancing values of a and b to get complexity $O(2^a + 2^b \cdot b^2)$.

Using some further backtracking optimization tricks we can reduce right part running time down to $O(2^b \cdot b)$ to get $O(2^a + 2^b \cdot b)$ overall.

H. NIMG

You are playing yet another version of game of Nim. During the player's turn she selects any pile of size at least F and splits them in some number of piles x (chosen by the player) such that the difference between the size of the smallest new pile and the size of the largest new pile is as small as possible (she fairly distributes stones between new piles).

H. NIMG

Assuming that piles do not affect each other and after the split we can consider that each pile is an individual game, we can compute Sprague-Grundy function $f(x)$ for each possible size x of some pile. Then we obtain the outcome of the game by simple computing value $f(a_1) \oplus f(a_2) \oplus \ldots \oplus f(a_n)$, where a_i is the initial size of the i-th pile. Here and later $\oplus$ stands for XOR function.

H. NIMG

Assuming that piles do not affect each other and after the split we can consider that each pile is an individual game, we can compute Sprague-Grundy function $f(x)$ for each possible size x of some pile. Then we obtain the outcome of the game by simple computing value $f(a_1) \oplus f(a_2) \oplus \ldots \oplus f(a_n)$, where a_i is the initial size of the i-th pile. Here and later $\oplus$ stands for XOR function.

We will compute values of $f(x)$ in order of increasing x. For x between 0 and $F - 1$ we set $f(x) = 0$. For some fixed $x \geqslant F$ we can try all possible values of k from 2 to k to split the pile in $x \bmod k$ piles of size $\lceil \frac{x}{k} \rceil$ and $k - x \bmod k$ piles of size $\lfloor \frac{x}{k} \rfloor$.

H. NIMG

That naive computation runs in $O(x)$ time per single x and in $O(s^2)$ in total, where s is the upper bound on the size of a single pile ($s = 10^5$).

H. NIMG

That naive computation runs in $O(x)$ time per single x and in $O(s^2)$ in total, where s is the upper bound on the size of a single pile ($s = 10^5$).

First we observe that for $k > \sqrt{x}$ the sizes of resulting piles are $\leqslant \sqrt{x}$.

H. NIMG

That naive computation runs in $O(x)$ time per single x and in $O(s^2)$ in total, where s is the upper bound on the size of a single pile ($s = 10^5$).

First we observe that for $k > \sqrt{x}$ the sizes of resulting piles are $\leqslant \sqrt{x}$.

Moreover, assuming that we compute Sprague-Grundy value of the move as $f(a) \oplus f(x) \oplus \ldots \oplus f(a) \oplus f(a+1) \oplus \ldots \oplus f(a+1)$, we only care about the parity of the number of piles of size a and the number of piles of size $a+1$.

H. NIMG

The above observations result in the following algorithm. Compute $f(x)$ in order of increasing x.

H. NIMG

The above observations result in the following algorithm. Compute $f(x)$ in order of increasing x.

For each x first try to divide it in k piles for all $k \leqslant \sqrt{x}$.

H. NIMG

The above observations result in the following algorithm. Compute $f(x)$ in order of increasing x.

For each x first try to divide it in k piles for all $k \leqslant \sqrt{x}$.

For every $a \leqslant \sqrt{x}$ and $r_1, r_2 \in Z_2$, we should decide whether there exists some k such that $\lfloor \frac{x}{k} \rfloor = a$, $x \bmod k$ has remainder r_1 modulo 2 and $k - x \bmod k$ has remainder r_2 modulo 2.

H. NIMG

First we consider the maximum possible number of piles of size a, equal to $\lfloor \frac{x}{a} \rfloor$. Then we spread the remaining $x \bmod a$ elements among these piles.

H. NIMG

First we consider the maximum possible number of piles of size a, equal to $\lfloor \frac{x}{a} \rfloor$. Then we spread the remaining $x \bmod a$ elements among these piles.

Take one pile of size a and distribute it among other piles of size a. This changes the parity of piles of size a by $(1 + a) \bmod 2$ and the parity of the number of piles of size $a + 1$ by $a \bmod 2$, so it makes sense to perform this operation only once.

H. NIMG

First we consider the maximum possible number of piles of size a, equal to $\lfloor \frac{x}{a} \rfloor$. Then we spread the remaining $x \bmod a$ elements among these piles.

Take one pile of size a and distribute it among other piles of size a. This changes the parity of piles of size a by $(1 + a) \bmod 2$ and the parity of the number of piles of size $a + 1$ by $a \bmod 2$, so it makes sense to perform this operation only once.

Overall running time is $O(s\sqrt{s})$.

I. Semi-cool points

Given a convex polygon with integer vertices in the plane, compute the number of half-integer points inside it or on its border.

I. Semi-cool points

Recall that according to Pick's theorem $S = a + \frac{b}{2} - 1$, where S is the area of the polygon with integer vertices, a is the number of integer points (lattice points) inside the polygon and b is the number of such points on this polygon's border.

I. Semi-cool points

Recall that according to Pick's theorem $S = a + \frac{b}{2} - 1$, where S is the area of the polygon with integer vertices, a is the number of integer points (lattice points) inside the polygon and b is the number of such points on this polygon's border.

Value of S can be found in linear time with one of many methods available. To compute b we consider each side segment. Let the i-th side be defined by vector $(vx_i = x_{i+1} - x_i, vy_i = y_{i+1} - y_i)$, the number of integer points inside this segment is $gcd(vx, vy) - 1$ (plus two more endpoints).

I. Semi-cool points

Recall that according to Pick's theorem $S = a + \frac{b}{2} - 1$, where S is the area of the polygon with integer vertices, a is the number of integer points (lattice points) inside the polygon and b is the number of such points on this polygon's border.

Value of S can be found in linear time with one of many methods available. To compute b we consider each side segment. Let the i-th side be defined by vector $(vx_i = x_{i+1} - x_i, vy_i = y_{i+1} - y_i)$, the number of integer points inside this segment is $gcd(vx, vy) - 1$ (plus two more endpoints).

I. Semi-cool points

Multiply both x and y coordinates by 2 and compute the number of integer points inside the polygon or on its border using the formulas above. Denote this value as ans_{22}.

I. Semi-cool points

Multiply both x and y coordinates by 2 and compute the number of integer points inside the polygon or on its border using the formulas above. Denote this value as ans_{22}.

The value of ans_{22} is not the answer yet as we also counted integer points (x, y), and points of kind $(x + 0.5, y)$ and $(x, y + 0.5)$. Denote as $ans_1 1$ the number of integer points for original polygon, ans_{12} — the number of integer points if we multiply ony y's by 2 and ans_{21} — the number of integer points if we multiply only x's by 2.

I. Semi-cool points

Multiply both x and y coordinates by 2 and compute the number of integer points inside the polygon or on its border using the formulas above. Denote this value as ans_{22}.

The value of ans_{22} is not the answer yet as we also counted integer points (x, y), and points of kind $(x + 0.5, y)$ and $(x, y + 0.5)$. Denote as $ans_1 1$ the number of integer points for original polygon, ans_{12} — the number of integer points if we multiply ony y's by 2 and ans_{21} — the number of integer points if we multiply only x's by 2.

So the answer is $ans_{22} - ans_{12} - ans_{21} + ans_{11}$.

J. Stairs

You are given integers n, m, k and a subset V of k valid jumps. Your goal is to compute $m^a \bmod 1234567891$, where a is the number of sequences $a_0, b_0, a_1, b_1, a_2, b_2, \ldots, a_l$ such that $a_0 = 0$, $a_l = n$, $b_i \leqslant a_i$ and $a_{i+1} - a_i$ is present in the set of valid jumps.

J. Stairs

Denote $f(x)$ as a number of valid sequences ending with x. By definition $f(x) = \sum_{d \in V} f(x-d) \cdot (x-d+1)$.

J. Stairs

Denote $f(x)$ as a number of valid sequences ending with x. By definition $f(x) = \sum_{d \in V} f(x-d) \cdot (x-d+1)$.

Using the fact that $d \leqslant s$ ($s = 10$) we construct a vector $v_s = f(0), f(1), \ldots, f(s)$ and matrix A_s of size $(s+1) \times (s+1)$ such that $A_s \cdot v_s = v_{s+1}$, where $v_{s+1} = f(1), f(2), \ldots, f(s+1)$. Now we compute v_n as $A_s \cdot A_{s+1} \cdot \ldots \cdot A_n \cdot v_s = v_n$.

J. Stairs

Denote $f(x)$ as a number of valid sequences ending with x. By definition $f(x) = \sum_{d \in V} f(x-d) \cdot (x-d+1)$.

Using the fact that $d \leqslant s$ $(s = 10)$ we construct a vector $v_s = f(0), f(1), \ldots, f(s)$ and matrix A_s of size $(s+1) \times (s+1)$ such that $A_s \cdot v_s = v_{s+1}$, where $v_{s+1} = f(1), f(2), \ldots, f(s+1)$. Now we compute v_n as $A_s \cdot A_{s+1} \cdot \ldots \cdot A_n \cdot v_s = v_n$.

By definition $A_s(s,i) = 0$ if $s - i$ is not valid and $A_s(s,i) = i + 1$ otherwise.

J. Stairs

To compute m^a mod 1234567891 we should know a or a mod $\varphi(1234567891)$. Integer 1234567891 is prime so $\varphi(1234567891) = 1234567890 = 2 \cdot 5 \cdot 9 \cdot 3607 \cdot 3803$.

J. Stairs

To compute $m^a \bmod 1234567891$ we should know a or $a \bmod \varphi(1234567891)$. Integer 1234567891 is prime so $\varphi(1234567891) = 1234567890 = 2 \cdot 5 \cdot 9 \cdot 3607 \cdot 3803$.

If we find a way to compute $a \bmod p$ for prime $p \in \{2, 5, 9, 3607, 3803\}$ we will be able to restore $a \bmod 1234567890$ using the Chinese remainder theorem.

J. Stairs

To compute $m^a \bmod 1234567891$ we should know a or $a \bmod \varphi(1234567891)$. Integer 1234567891 is prime so $\varphi(1234567891) = 1234567890 = 2 \cdot 5 \cdot 9 \cdot 3607 \cdot 3803$.

If we find a way to compute $a \bmod p$ for prime $p \in \{2, 5, 9, 3607, 3803\}$ we will be able to restore $a \bmod 1234567890$ using the Chinese remainder theorem.

Consider we have some fixed p. Notice that according to definition $A_s(x, y) \bmod p = A_{s+p}(x, y) \bmod p$, so we have $v_n = A_s^{d_s} \cdot A_{s+1}^{d_{s+1}} \cdot \ldots \cdot A_{s+p-1}^{d_{s+p-1}} \cdot v_s$. That means we can find $a \bmod p$ in $O(p \cdot s^3 \cdot \log n)$ time.

K. Number of Zeroes

Given an integer k, consider all integer from 1 to 10^k inclusive and compute the total number of zeroes in their decimal notations modulo p.

K. Number of Zeroes

Consider some fixed position i from 0 to $k-1$. What is the total number of zeroes at this position? We should find the number of integers matching the pattern of some prefix, then 0 at the i-th positon, then sum suffix. There are 10^i possible suffixes for any integer of length less than k. Moreover, there are $10^{k-i-1}-1$ valid prefixes, plus the integer 10^k gives $+1$ for the answer at every position.

K. Number of Zeroes

Consider some fixed position i from 0 to $k-1$. What is the total number of zeroes at this position? We should find the number of integers matching the pattern of some prefix, then 0 at the i-th positon, then sum suffix. There are 10^i possible suffixes for any integer of length less than k. Moreover, there are $10^{k-i-1}-1$ valid prefixes, plus the integer 10^k gives $+1$ for the answer at every position.

From the above we have the following formula

$$ans = k + \sum_{i=0}^{k-1}(10^{k-i-1}-1)\cdot 10^i = (10^{k-1}+1)\cdot k - \sum_{i=0}^{k-1} 10^i$$

K. Number of Zeroes

Using the well-known formula for geometric progression sum we have:

$$\sum_{i=0}^{k-1} 10^i = \frac{10^k - 1}{10 - 1}$$

K. Number of Zeroes

Using the well-known formula for geometric progression sum we have:

$$\sum_{i=0}^{k-1} 10^i = \frac{10^k - 1}{10 - 1}$$

However, to compute the value modulo p by this formula we need to find inverse value modulo p for integer 9. That might turn out to be complicated if p is not prime.

K. Number of Zeroes

What if k is even?

$$\sum_{i=0}^{k-1} 10^i = (1 + 10^{\frac{k}{2}}) \cdot \sum_{i=0}^{\frac{k}{2}-1} 10^i$$

K. Number of Zeroes

What if k is even?

$$\sum_{i=0}^{k-1} 10^i = (1 + 10^{\frac{k}{2}}) \cdot \sum_{i=0}^{\frac{k}{2}-1} 10^i$$

For the odd k he can write:

$$\sum_{i=0}^{k-1} 10^i = 10^{k-1} + \sum_{i=0}^{k-2} 10^i$$

K. Number of Zeroes

What if k is even?

$$\sum_{i=0}^{k-1} 10^i = (1 + 10^{\frac{k}{2}}) \cdot \sum_{i=0}^{\frac{k}{2}-1} 10^i$$

For the odd k he can write:

$$\sum_{i=0}^{k-1} 10^i = 10^{k-1} + \sum_{i=0}^{k-2} 10^i$$

That gives us the algorithm similar to exponentiation by squaring with no division operation to compute the formula. Its running time is $O(\log^2 k)$ for naive implementation but can be easily reduced to $O(\log k)$.