Long Contest Editorial

March 27, 2016

by Mikhail Tikhomirov (MIPT)

Moscow Pre-Finals ACM ICPC Workshop, MIPT, 2016

A. As Easy As Possible

For a given string s, answer several queries of the form "find maximal k such that $\texttt{easy}\ldots\texttt{easy}$ (k times) is a subsequence of $s[l \ldots r]$".

A. As Easy As Possible

For a given string s, answer several queries of the form "find maximal k such that $\texttt{easy}\ldots\texttt{easy}$ (k times) is a subsequence of $s[l \ldots r]$".
It is easy to answer a single query in $O(n)$ time.

A. As Easy As Possible

For a given string s, answer several queries of the form "find maximal k such that $\texttt{easy}\ldots\texttt{easy}$ (k times) is a subsequence of $s[l \ldots r]$".
It is easy to answer a single query in $O(n)$ time.
In fact, we will find the longest prefix of the infinite string `(easy)*` (infinite repeats of `easy`) which is a subsequence of the substring t: start with the empty prefix, iterate over characters of t and increase the length of the prefix whenever the current character of t matches the next character of `(easy)*`.

A. As Easy As Possible

How do we answer a query fast?

A. As Easy As Possible

How do we answer a query fast?

Let us use the "binary shift" approach: for a substring $[l; r]$, divide the range $[l; r]$ into $O(\log n)$ parts with powers of 2 as lengths, and combine the answer from precomputed information for the parts.

A. As Easy As Possible

How do we answer a query fast?

Let us use the "binary shift" approach: for a substring $[l;r]$, divide the range $[l;r]$ into $O(\log n)$ parts with powers of 2 as lengths, and combine the answer from precomputed information for the parts.

Which information do we have to precompute? For the current part, we would like to know how much the current prefix can be extended.

A. As Easy As Possible

How do we answer a query fast?

Let us use the "binary shift" approach: for a substring $[l; r]$, divide the range $[l; r]$ into $O(\log n)$ parts with powers of 2 as lengths, and combine the answer from precomputed information for the parts.

Which information do we have to precompute? For the current part, we would like to know how much the current prefix can be extended.

Since `(easy)*` is periodical, it suffices to know the next character of `easy` to match with. Thus, we precompute $a_{k,i,j}$ — how much the prefix can be extended inside the substring $s[i \ldots i + 2^k)$ if the current character to match is `easy`[j].

A. As Easy As Possible

Time and memory needed to precompute $O(nm \log n)$, where $n = |s|$, $m = |\texttt{easy}|$. It is then possible to answer a query in $O(\log n)$ time, for the total $O((q + nm) \log n)$ time complexity.

A. As Easy As Possible

Time and memory needed to precompute $O(nm \log n)$, where $n = |s|$, $m = |\mathtt{easy}|$. It is then possible to answer a query in $O(\log n)$ time, for the total $O((q + nm) \log n)$ time complexity. There is also a similar approach with a segment tree, with the same time complexity and slightly less memory usage.

B. Be Friends

Define distance between integers as their XOR. Find the weight of MST on a set of integers.

B. Be Friends

Define distance between integers as their XOR. Find the weight of MST on a set of integers.
We will assume that all numbers are distinct since we can erase all duplicates.

B. Be Friends

Let k be the position of the greatest binary digit in every number.

B. Be Friends

Let k be the position of the greatest binary digit in every number.

Proposition

There will be at most one edge in MST with 1 in k'th digit of XOR.

B. Be Friends

Let k be the position of the greatest binary digit in every number.

Proposition

There will be at most one edge in MST with 1 in k'th digit of XOR.

This hints at the recursive approach: divide all numbers into two groups according to their k'th digit, solve recursively for these groups, then add minimal possible edge between the groups.

B. Be Friends

How do we find the minimal edge between the groups?

B. Be Friends

How do we find the minimal edge between the groups?
We will find the trie structure useful. The question asks about minimal XOR between integers in two subtrees of a trie.

B. Be Friends

How do we find the minimal edge between the groups?
We will find the trie structure useful. The question asks about minimal XOR between integers in two subtrees of a trie.
We will use a greedy approach.

B. Be Friends

How do we find the minimal edge between the groups?
We will find the trie structure useful. The question asks about minimal XOR between integers in two subtrees of a trie.
We will use a greedy approach. Do parallel DFS of the subtrees.
When we descend into a child of the first vertex, choose a child of the second vertex that minimizes XOR in the current digit (if possible).

B. Be Friends

How do we find the minimal edge between the groups?
We will find the trie structure useful. The question asks about minimal XOR between integers in two subtrees of a trie.
We will use a greedy approach. Do parallel DFS of the subtrees. When we descend into a child of the first vertex, choose a child of the second vertex that minimizes XOR in the current digit (if possible). For each pair of leaves reached, try to improve the answer with their XOR. Clearly, an optimal answer will be found at some leaves pair.

B. Be Friends

How fast does this work?

B. Be Friends

How fast does this work?

The parallel DFS works in $O(w_1 + w_2)$, where w_1 and w_2 are sizes of corresponding subtrees

B. Be Friends

How fast does this work?

The parallel DFS works in $O(w_1 + w_2)$, where w_1 and w_2 are sizes of corresponding subtrees *(it can even be made* $O(\min(w_1, w_2))$*)*.

B. Be Friends

How fast does this work?
The parallel DFS works in $O(w_1 + w_2)$, where w_1 and w_2 are sizes of corresponding subtrees *(it can even be made* $O(\min(w_1, w_2))$*)*.
Each vertex of the trie is present in at most $k + 1$ subtrees, so total complexity is $O(k^2 n)$ since there are $O(kn)$ vertices in the trie.

C. Coprime Heaven

We are given $k \leqslant 4$ numbers l_i. Distribute numbers from 1 to $n = \sum l_i$ into circles of lengths $l_1, \ldots, l_k$ such that each pair of adjacent numbers is coprime.

C. Coprime Heaven

Clearly, even numbers cannot be adjacent. A circle of length $l > 1$ can include at most $\lfloor l/2 \rfloor$ even numbers (or at most 1 if $l = 1$). If there are more even numbers than we can include in all the circles, then no answer exists.

C. Coprime Heaven

Clearly, even numbers cannot be adjacent. A circle of length $l > 1$ can include at most $\lfloor l/2 \rfloor$ even numbers (or at most 1 if $l = 1$). If there are more even numbers than we can include in all the circles, then no answer exists.

Proposition

Otherwise, there is always an answer!

C. Coprime Heaven

Clearly, even numbers cannot be adjacent. A circle of length $l > 1$ can include at most $\lfloor l/2 \rfloor$ even numbers (or at most 1 if $l = 1$). If there are more even numbers than we can include in all the circles, then no answer exists.

Proposition

Otherwise, there is always an answer!

Proof

Can be done by a long and tedious case analysis and constructions.

C. Coprime Heaven

Clearly, even numbers cannot be adjacent. A circle of length $l > 1$ can include at most $\lfloor l/2 \rfloor$ even numbers (or at most 1 if $l = 1$). If there are more even numbers than we can include in all the circles, then no answer exists.

Proposition

Otherwise, there is always an answer!

Proof

Can be done by a long and tedious case analysis and constructions. Key insight: it is convenient to include runs of adjacent numbers into circles, then only ends of the runs should be checked for coprimeness.

C. Coprime Heaven

Is there a simple way to construct an answer (provided it exists)?

C. Coprime Heaven

Is there a simple way to construct an answer (provided it exists)?
A simple (but apparently hard to prove) way is this: construct circles from left to right.

C. Coprime Heaven

Is there a simple way to construct an answer (provided it exists)? A simple (but apparently hard to prove) way is this: construct circles from left to right. Brute-force all ways to put first few numbers, then brute-force all ways to greedily fill the circles with adjacent numbers (that is, if the last used number is x, we will append numbers $x + 1$, $x + 2$, $\ldots$ to the chosen circle until it fills up).

C. Coprime Heaven

Is there a simple way to construct an answer (provided it exists)? A simple (but apparently hard to prove) way is this: construct circles from left to right. Brute-force all ways to put first few numbers, then brute-force all ways to greedily fill the circles with adjacent numbers (that is, if the last used number is x, we will append numbers $x+1$, $x+2$, $\ldots$ to the chosen circle until it fills up). Carefully check for coprimeness each time we put anything, don't forget to handle $l = 1$ case.

C. Coprime Heaven

Is there a simple way to construct an answer (provided it exists)?
A simple (but apparently hard to prove) way is this: construct circles from left to right. Brute-force all ways to put first few numbers, then brute-force all ways to greedily fill the circles with adjacent numbers (that is, if the last used number is x, we will append numbers $x + 1$, $x + 2$, $\ldots$ to the chosen circle until it fills up). Carefully check for coprimeness each time we put anything, don't forget to handle $l = 1$ case.
Always finds a solution (at least, on the jury test cases), and works very fast too (which hints that there are many solutions of similar form).

C. Coprime Heaven

Is there a simple way to construct an answer (provided it exists)?
A simple (but apparently hard to prove) way is this: construct circles from left to right. Brute-force all ways to put first few numbers, then brute-force all ways to greedily fill the circles with adjacent numbers (that is, if the last used number is x, we will append numbers $x + 1$, $x + 2$, ... to the chosen circle until it fills up). Carefully check for coprimeness each time we put anything, don't forget to handle $l = 1$ case.
Always finds a solution (at least, on the jury test cases), and works very fast too (which hints that there are many solutions of similar form).
An unchecked conjecture: starting from trivial configuration (put smallest numbers into first circle, the next into the second circle, and so on) and performing local optimization/simulated annealing should work too.

D. Drawing Hell

A set of n points is given in the plane. Two players play a game, a move is to connect two points with a segment if the segment does not contain other points and does not intersect previously drawn segments. Determine the winner of the game if the player who is unable to make a move loses and both players act optimally.

D. Drawing Hell

Turns out the number of moves in the game does not depend on the strategy, but only on the set of points.

D. Drawing Hell

Turns out the number of moves in the game does not depend on the strategy, but only on the set of points.

Observation

If all points lie on a straight line, then the number of moves is $n-1$. Otherwise, the number of moves is $3n-3-c$, where c is the number of points on the border of the set's convex hull.

D. Drawing Hell

Turns out the number of moves in the game does not depend on the strategy, but only on the set of points.

Observation

If all points lie on a straight line, then the number of moves is $n-1$. Otherwise, the number of moves is $3n-3-c$, where c is the number of points on the border of the set's convex hull.

Proof

The first case is obvious. For the second case, we notice that any final configuration is a triangulation of the initial set (that is, every face of the resulting planar graph is a triangle). Euler's formula $V-E+F=2$ implies that any triangulation of a non-collinear set has $3n-3-c$ edges.

D. Drawing Hell

Build the convex hull of the initial set. If it's a line, the answer depends on the parity of $n - 1$. Otherwise, the answer depends on the parity of $3n - 3 - c$.

D. Drawing Hell

Build the convex hull of the initial set. If it's a line, the answer depends on the parity of $n-1$. Otherwise, the answer depends on the parity of $3n-3-c$.
The number of test cases is large, thus one should use an $O(n \log n)$-time algorithm for convex hull.

E. Easiest game

Process several queries: for an $m \times n$ board count number of pairs (s, t) with $1 \leqslant s \leqslant t \leqslant \max(m, n)$ such that an (s, t)-knight can visit all board cells.

E. Easiest game

Process several queries: for an $m \times n$ board count number of pairs (s, t) with $1 \leqslant s \leqslant t \leqslant \max(m, n)$ such that an (s, t)-knight can visit all board cells.

Solution has two parts: finding the criterion for suitability of an (s, t)-pair, and then counting them effectively.

E. Easiest game

Process several queries: for an $m \times n$ board count number of pairs (s, t) with $1 \leqslant s \leqslant t \leqslant \max(m, n)$ such that an (s, t)-knight can visit all board cells.

Solution has two parts: finding the criterion for suitability of an (s, t)-pair, and then counting them effectively.

Both are hard.

E. Easiest game

How to find the condition when an (s, t)-knight can traverse the board? One way is to write a brute-force for small boards and seek for pattern.

E. Easiest game

How to find the condition when an (s, t)-knight can traverse the board? One way is to write a brute-force for small boards and seek for pattern.
After some seeking, one can arrive at

E. Easiest game

How to find the condition when an (s,t)-knight can traverse the board? One way is to write a brute-force for small boards and seek for pattern.

After some seeking, one can arrive at

Proposition

Let $m \leqslant n$ and $s \leqslant t$. Then, an (s,t)-knight can traverse an $m \times n$ board iff three conditions hold:

E. Easiest game

How to find the condition when an (s, t)-knight can traverse the board? One way is to write a brute-force for small boards and seek for pattern.
After some seeking, one can arrive at

Proposition

Let $m \leqslant n$ and $s \leqslant t$. Then, an (s, t)-knight can traverse an $m \times n$ board iff three conditions hold:

1. $GCD(t + s, t - s) = 1$

E. Easiest game

How to find the condition when an (s, t)-knight can traverse the board? One way is to write a brute-force for small boards and seek for pattern.
After some seeking, one can arrive at

Proposition

Let $m \leqslant n$ and $s \leqslant t$. Then, an (s, t)-knight can traverse an $m \times n$ board iff three conditions hold:

1. $GCD(t + s, t - s) = 1$
2. $2t \leqslant n$

E. Easiest game

How to find the condition when an (s, t)-knight can traverse the board? One way is to write a brute-force for small boards and seek for pattern.

After some seeking, one can arrive at

Proposition

Let $m \leqslant n$ and $s \leqslant t$. Then, an (s, t)-knight can traverse an $m \times n$ board iff three conditions hold:

1. $GCD(t+s, t-s) = 1$
2. $2t \leqslant n$
3. $t + s \leqslant m$

E. Easiest game

Proof (partial)

Necessity: if $g = GCD(t+s, t-s) > 1$, then $x+y$ and $x-y$ are always the same modulo g, thus not all cells are reachable from each other.

E. Easiest game

Proof (partial)

Necessity: if $g = GCD(t+s, t-s) > 1$, then $x+y$ and $x-y$ are always the same modulo g, thus not all cells are reachable from each other.

If $2t > n$, then no move can be made from the central cell of the board.

E. Easiest game

Proof (partial)

Necessity: if $g = GCD(t+s, t-s) > 1$, then $x+y$ and $x-y$ are always the same modulo g, thus not all cells are reachable from each other.
If $2t > n$, then no move can be made from the central cell of the board.
If $s+t > m$, then ???

E. Easiest game

Proof (partial)

Necessity: if $g = GCD(t+s, t-s) > 1$, then $x+y$ and $x-y$ are always the same modulo g, thus not all cells are reachable from each other.
If $2t > n$, then no move can be made from the central cell of the board.
If $s+t > m$, then ???
(Empirically, for $s+t = m+1$ only half of the board is reachable from a certain cell (Parity of some sort?))

E. Easiest game

Proof (partial)

Necessity: if $g = GCD(t+s, t-s) > 1$, then $x+y$ and $x-y$ are always the same modulo g, thus not all cells are reachable from each other.
If $2t > n$, then no move can be made from the central cell of the board.
If $s+t > m$, then ???
(Empirically, for $s+t = m+1$ only half of the board is reachable from a certain cell (Parity of some sort?))
Sufficiency: ???

E. Easiest game

Let $a = t + s$, $b = t - s$.

E. Easiest game

Let $a = t + s$, $b = t - s$.
Then, the conditions are:

E. Easiest game

Let $a = t + s$, $b = t - s$.
Then, the conditions are:

1. $GCD(a, b) = 1$

E. Easiest game

Let $a = t + s$, $b = t - s$.
Then, the conditions are:

1. $GCD(a, b) = 1$
2. $a + b \leqslant n$

E. Easiest game

Let $a = t + s$, $b = t - s$.
Then, the conditions are:

1. $GCD(a, b) = 1$
2. $a + b \leqslant n$
3. $a \leqslant m$

E. Easiest game

Let $a = t + s$, $b = t - s$.
Then, the conditions are:

1. $GCD(a, b) = 1$
2. $a + b \leqslant n$
3. $a \leqslant m$
4. additionally, $b < a$, and a and b have the same parity.

E. Easiest game

Let $a = t + s$, $b = t - s$.
Then, the conditions are:

1. $GCD(a, b) = 1$
2. $a + b \leqslant n$
3. $a \leqslant m$
4. additionally, $b < a$, and a and b have the same parity.

If we drop the GCD condition, the number of (a, b) pairs can be calculated in $O(1)$ (it's the number of black cells inside a certain region of the infinite chessboard).

E. Easiest game

Let $a = t + s$, $b = t - s$.
Then, the conditions are:

1. $GCD(a, b) = 1$
2. $a + b \leqslant n$
3. $a \leqslant m$
4. additionally, $b < a$, and a and b have the same parity.

If we drop the GCD condition, the number of (a, b) pairs can be calculated in $O(1)$ (it's the number of black cells inside a certain region of the infinite chessboard).
Denote $f(n, m)$ the answer without GCD.

E. Easiest game

Let $a = t + s$, $b = t - s$.
Then, the conditions are:

1. $GCD(a, b) = 1$
2. $a + b \leqslant n$
3. $a \leqslant m$
4. additionally, $b < a$, and a and b have the same parity.

If we drop the GCD condition, the number of (a, b) pairs can be calculated in $O(1)$ (it's the number of black cells inside a certain region of the infinite chessboard).
Denote $f(n, m)$ the answer without GCD.
Also, denote $f'(n, m)$ the number of pairs with same restrictions except for the last one (same parity).

E. Easiest game

To account for GCD, we will use the inclusion-exclusion principle.

E. Easiest game

To account for GCD, we will use the inclusion-exclusion principle. The answer is *almost* $\sum_{k=1}^{\min(n,m)} \mu(k) f(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$, where $\mu(k)$ is the Moebius function.

E. Easiest game

To account for GCD, we will use the inclusion-exclusion principle. The answer is *almost* $\sum_{k=1}^{\min(n,m)} \mu(k) f(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$, where $\mu(k)$ is the Moebius function.

The difference is, if k is even then we shouldn't care about same parity of a and b inside $f(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$, so in this case we replace it with $f'(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$.

E. Easiest game

To account for GCD, we will use the inclusion-exclusion principle.
The answer is *almost* $\sum_{k=1}^{\min(n,m)} \mu(k) f(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$, where $\mu(k)$ is the Moebius function.
The difference is, if k is even then we shouldn't care about same parity of a and b inside $f(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$, so in this case we replace it with $f'(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$.
That's still $O(\min(n, m))$.

E. Easiest game

To account for GCD, we will use the inclusion-exclusion principle.
The answer is *almost* $\sum_{k=1}^{\min(n,m)} \mu(k) f(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$, where $\mu(k)$ is the Moebius function.
The difference is, if k is even then we shouldn't care about same parity of a and b inside $f(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$, so in this case we replace it with $f'(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$.
That's still $O(\min(n, m))$.
To optimize further, note that there are $O(\sqrt{n} + \sqrt{m})$ values of k such that $\lfloor n/k \rfloor$ or $\lfloor m/k \rfloor$ change.

E. Easiest game

To account for GCD, we will use the inclusion-exclusion principle.
The answer is *almost* $\sum_{k=1}^{\min(n,m)} \mu(k) f(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$, where $\mu(k)$ is the Moebius function.
The difference is, if k is even then we shouldn't care about same parity of a and b inside $f(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$, so in this case we replace it with $f'(\lfloor n/k \rfloor, \lfloor m/k \rfloor)$.
That's still $O(\min(n, m))$.
To optimize further, note that there are $O(\sqrt{n} + \sqrt{m})$ values of k such that $\lfloor n/k \rfloor$ or $\lfloor m/k \rfloor$ change.
Thus, the solution can be optimized to $O(\sqrt{n} + \sqrt{m})$ per test by processing segments of k where $\lfloor n/k \rfloor$ and $\lfloor m/k \rfloor$ are fixed.

F. Fibonacci of Fibonacci

Find $F_{F_n} \bmod 20160519$.

F. Fibonacci of Fibonacci

Proposition

For each integer m Fibonacci numbers modulo m eventually loop, that is, there exists $p > 0$ such that $F_n \equiv F_{n+p} \pmod{m}$ for each n.

F. Fibonacci of Fibonacci

Proposition

For each integer m Fibonacci numbers modulo m eventually loop, that is, there exists $p > 0$ such that $F_n \equiv F_{n+p} \pmod{m}$ for each n.

For the given modulo 20160519 the period is small enough to find explicitly.

F. Fibonacci of Fibonacci

Proposition

For each integer m Fibonacci numbers modulo m eventually loop, that is, there exists $p > 0$ such that $F_n \equiv F_{n+p} \pmod{m}$ for each n.

For the given modulo 20160519 the period is small enough to find explicitly.
Since there are many queries to answer, one should use matrix exponentiation to find the answer in $O(\log n)$ per query:

Proposition

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

G. Global Warming

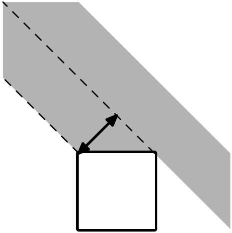
Given two convex polygons — the planet and the moon, and the radiation direction from the sun, find the total heat from the sun assuming that the light can reflect from the moon.

G. Global Warming

The model of effective heat absorption in the statement is somewhat obscure. However, it helps to think of it this way: assume that the planet is illuminated by a single strip of light. Then the resulting amount of heat from this strip of light is the *width* of the strip part that falls on the surface of the planet.

G. Global Warming

The model of effective heat absorption in the statement is somewhat obscure. However, it helps to think of it this way: assume that the planet is illuminated by a single strip of light. Then the resulting amount of heat from this strip of light is the *width* of the strip part that falls on the surface of the planet.



For example, on this picture the equivalent length is equal to half the square diagonal length.

G. Global Warming

Let us solve a subproblem: find the equivalent length for a given light strip and a convex polygon.

G. Global Warming

Let us solve a subproblem: find the equivalent length for a given light strip and a convex polygon.

Each ray of light is described by a line equation $ax + by + c = 0$, where the vector (a, b) is orthogonal to the light direction.

G. Global Warming

Let us solve a subproblem: find the equivalent length for a given light strip and a convex polygon.
Each ray of light is described by a line equation $ax + by + c = 0$, where the vector (a, b) is orthogonal to the light direction.
Let's assume that $a^2 + b^2 = 1$. Then the strip is described by two border rays, which are described by numbers $\underline{c}$ and $\overline{c}$ from corresponding equations.

G. Global Warming

Let us solve a subproblem: find the equivalent length for a given light strip and a convex polygon.

Each ray of light is described by a line equation $ax + by + c = 0$, where the vector (a, b) is orthogonal to the light direction.

Let's assume that $a^2 + b^2 = 1$. Then the strip is described by two border rays, which are described by numbers $\underline{c}$ and $\overline{c}$ from corresponding equations.

For the polygon we can similarly find border rays which fall on its surface. It is evident that their c's (denote them $\underline{c'}$ and $\overline{c'}$) are simply the extremal values of $ax + by$ over all vertices of polygon.

G. Global Warming

Let us solve a subproblem: find the equivalent length for a given light strip and a convex polygon.
Each ray of light is described by a line equation $ax + by + c = 0$, where the vector (a, b) is orthogonal to the light direction.
Let's assume that $a^2 + b^2 = 1$. Then the strip is described by two border rays, which are described by numbers $\underline{c}$ and $\overline{c}$ from corresponding equations.
For the polygon we can similarly find border rays which fall on its surface. It is evident that their c's (denote them $\underline{c'}$ and $\overline{c'}$) are simply the extremal values of $ax + by$ over all vertices of polygon.
The equivalent length of the illuminated part is then the length of the $[\underline{c}; \overline{c}] \cap [\underline{c'}; \overline{c'}]$ range.

G. Global Warming

The most complex part is to find extremal values of $ax + by$ over the vertices of a convex polygon.

G. Global Warming

The most complex part is to find extremal values of $ax + by$ over the vertices of a convex polygon.
One possible approach is based on the fact that $ax + by$ function is unimodal (with a unique local extremum) on a convex curve such that its, say, x coordinate does not decrease. Hence, the $ax + by$ function is unimodal on lower and upper halves of the polygon, so the ternary search can be applied to each of them.

G. Global Warming

The most complex part is to find extremal values of $ax + by$ over the vertices of a convex polygon.
One possible approach is based on the fact that $ax + by$ function is unimodal (with a unique local extremum) on a convex curve such that its, say, x coordinate does not decrease. Hence, the $ax + by$ function is unimodal on lower and upper halves of the polygon, so the ternary search can be applied to each of them.
This reasoning yields a way to find extremums of $ax + by$ over convex polygon in $O(\log n)$ time.

G. Global Warming

The total heat is comprised of two parts: direct light and moon-reflected light.

G. Global Warming

The total heat is comprised of two parts: direct light and moon-reflected light.

The direct heat is accounted simply as the effective length for the planet NOT blocked by the moon (that is, the length of the interval for the planet minus the length of the intersection for planet's and moon's intervals).

G. Global Warming

The total heat is comprised of two parts: direct light and moon-reflected light.

The direct heat is accounted simply as the effective length for the planet NOT blocked by the moon (that is, the length of the interval for the planet minus the length of the intersection for planet's and moon's intervals).

The reflected heat is comprised of several strips for each illuminated side of the moon. Each of them can be processed independently as discussed earlier (no blocking can take place now). Take care since some strips may reflect light away from the planet!

G. Global Warming

The total heat is comprised of two parts: direct light and moon-reflected light.

The direct heat is accounted simply as the effective length for the planet NOT blocked by the moon (that is, the length of the interval for the planet minus the length of the intersection for planet's and moon's intervals).

The reflected heat is comprised of several strips for each illuminated side of the moon. Each of them can be processed independently as discussed earlier (no blocking can take place now). Take care since some strips may reflect light away from the planet!

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

H. Hash Collision

Count the pairs of n-letter strings with the same polynomial hash with given base and modulo.

H. Hash Collision

Count the pairs of n-letter strings with the same polynomial hash with given base and modulo.

Polynomial hash is described by the formula $h(s) = \sum_{i=0}^{n-1} s_i p^i \bmod m$,

where p is the *base*, and m is the *modulo*.

H. Hash Collision

Denote $d_{n,h}$ the number of n-letter strings with hash value h. The answer is then $\sum_{h=0}^{m-1} \binom{d_{n,h}}{2}$

H. Hash Collision

Denote $d_{n,h}$ the number of n-letter strings with hash value h. The answer is then $\sum_{h=0}^{m-1} \binom{d_{n,h}}{2}$
By definition, they satisfy the recurrence

$$d_{n_1+n_2,h} = \sum_{h_1=0}^{m-1} d_{n_1,h_1} d_{n_2,(h-h_1 p^{n_1}) \bmod m}$$

H. Hash Collision

Denote $d_{n,h}$ the number of n-letter strings with hash value h. The answer is then $\sum_{h=0}^{m-1} \binom{d_{n,h}}{2}$
By definition, they satisfy the recurrence

$$d_{n_1+n_2,h} = \sum_{h_1=0}^{m-1} d_{n_1,h_1} d_{n_2,(h-h_1 p^{n_1}) \bmod m}$$

Applying this reccurence with $n_1 = 1$, one can obtain a $O(nm\alpha)$ solution (since $d_{1,h} = 1$ iff h is a hash of a single letter).

H. Hash Collision

To speed this up, note that the described recurrence can be computed as coefficients of the polynomial product $P(x) \times Q(x)$, where
$P(x) = \sum_{i=0}^{m-1} d_{n_1,i} x^i$, and $Q(x) = \sum_{i=0}^{m-1} d_{n_2,i} x^{(p^{n_1} i) \bmod m}$.

H. Hash Collision

To speed this up, note that the described recurrence can be computed as coefficients of the polynomial product $P(x) \times Q(x)$, where
$P(x) = \sum_{i=0}^{m-1} d_{n_1,i} x^i$, and $Q(x) = \sum_{i=0}^{m-1} d_{n_2,i} x^{(p^{n_1} i) \bmod m}$.
Now, suppose that we know the numbers $d_{n,h}$. Let
$P(x) = \sum_{i=0}^{m-1} d_{n,i} x^i$, and $P'(x) = \sum_{i=0}^{m-1} d_{n,i} x^{(p^n i) \bmod m}$. Then, after computing $P(x)P'(x)$, we obtain the numbers $d_{2n,h}$.

H. Hash Collision

To speed this up, note that the described recurrence can be computed as coefficients of the polynomial product $P(x) \times Q(x)$, where
$P(x) = \sum_{i=0}^{m-1} d_{n_1,i} x^i$, and $Q(x) = \sum_{i=0}^{m-1} d_{n_2,i} x^{(p^{n_1} i) \bmod m}$.
Now, suppose that we know the numbers $d_{n,h}$. Let
$P(x) = \sum_{i=0}^{m-1} d_{n,i} x^i$, and $P'(x) = \sum_{i=0}^{m-1} d_{n,i} x^{(p^n i) \bmod m}$. Then, after computing $P(x)P'(x)$, we obtain the numbers $d_{2n,h}$.
This allows us to use a binary exponentiation-line approach:

H. Hash Collision

To speed this up, note that the described recurrence can be computed as coefficients of the polynomial product $P(x) \times Q(x)$, where
$P(x) = \sum_{i=0}^{m-1} d_{n_1,i} x^i$, and $Q(x) = \sum_{i=0}^{m-1} d_{n_2,i} x^{(p^{n_1} i) \bmod m}$.
Now, suppose that we know the numbers $d_{n,h}$. Let
$P(x) = \sum_{i=0}^{m-1} d_{n,i} x^i$, and $P'(x) = \sum_{i=0}^{m-1} d_{n,i} x^{(p^n i) \bmod m}$. Then, after computing $P(x)P'(x)$, we obtain the numbers $d_{2n,h}$.
This allows us to use a binary exponentiation-line approach:

- for an odd n recursively find the answers for $n-1$ and use a single step of recurrence

H. Hash Collision

To speed this up, note that the described recurrence can be computed as coefficients of the polynomial product $P(x) \times Q(x)$, where
$P(x) = \sum_{i=0}^{m-1} d_{n_1,i}x^i$, and $Q(x) = \sum_{i=0}^{m-1} d_{n_2,i}x^{(p^{n_1}i) \bmod m}$.
Now, suppose that we know the numbers $d_{n,h}$. Let
$P(x) = \sum_{i=0}^{m-1} d_{n,i}x^i$, and $P'(x) = \sum_{i=0}^{m-1} d_{n,i}x^{(p^n i) \bmod m}$. Then, after computing $P(x)P'(x)$, we obtain the numbers $d_{2n,h}$.
This allows us to use a binary exponentiation-line approach:

- for an odd n recursively find the answers for $n-1$ and use a single step of recurrence
- and for an even n find the answers for $n/2$ and use the method above.

H. Hash Collision

To speed this up, note that the described recurrence can be computed as coefficients of the polynomial product $P(x) \times Q(x)$, where
$P(x) = \sum_{i=0}^{m-1} d_{n_1,i} x^i$, and $Q(x) = \sum_{i=0}^{m-1} d_{n_2,i} x^{(p^{n_1} i) \bmod m}$.
Now, suppose that we know the numbers $d_{n,h}$. Let
$P(x) = \sum_{i=0}^{m-1} d_{n,i} x^i$, and $P'(x) = \sum_{i=0}^{m-1} d_{n,i} x^{(p^{n} i) \bmod m}$. Then, after computing $P(x)P'(x)$, we obtain the numbers $d_{2n,h}$.
This allows us to use a binary exponentiation-line approach:

- for an odd n recursively find the answers for $n-1$ and use a single step of recurrence
- and for an even n find the answers for $n/2$ and use the method above.

We obtain a solution that does $O(\log n)$ m-degree polynomial multiplications.

H. Hash Collision

Of course, to speed up we should use FFT to multiply polynomials. Since the answer should be found modulo $10^6 + 3$, the `double` precision should be enough. The resulting complexity is $O(m \log m \log n)$.

I. Increasing or Decreasing

Process many queries of the form: count the numbers inside the range $[L; R]$ which decimal representations are monotonous (that is, teh digits are either increasing or decreasing).

I. Increasing or Decreasing

First approach: compute DP of sort "how many non-increasing/non-decreasing l-digit prefixes of n-digit numbers exist such that they are already less than/still equal to corresponding part of R/L" (this is messy to code and will probably need some optimization).

I. Increasing or Decreasing

First approach: compute DP of sort "how many non-increasing/non-decreasing l-digit prefixes of n-digit numbers exist such that they are already less than/still equal to corresponding part of R/L" (this is messy to code and will probably need some optimization).
Second approach: generate all suitable numbers and note that there are less than 20 million of them. Hence, each query is simply several binary searches in precomputed lists.

J. Just Convolution

Given $a_0, \ldots, a_{n-1}$ and $b_0, \ldots, b_{n-1}$ — random permutations of $\{0, \ldots, n-1\}$, find $c_0, \ldots, c_{n-1}$, where

$$c_k = \max_{i=0}^{n-1}(a_i + b_{(k-i) \bmod n})$$

.

J. Just Convolution

Given $a_0, \ldots, a_{n-1}$ and $b_0, \ldots, b_{n-1}$ — random permutations of $\{0, \ldots, n-1\}$, find $c_0, \ldots, c_{n-1}$, where

$$c_k = \max_{i=0}^{n-1}(a_i + b_{(k-i) \bmod n})$$

.

For convenience we will replace max *with* min *in the above formula, which does not change the problem much.*

J. Just Convolution

$O(n^2)$ solution is too ineffective. How do we use randomness of the input permutations?

J. Just Convolution

$O(n^2)$ solution is too ineffective. How do we use randomness of the input permutations?

Let us choose a number K. We can now find all elements of c_i less than K by simply trying all pairs $a_i + b_j < K$ in $O(K^2)$ time.

J. Just Convolution

$O(n^2)$ solution is too ineffective. How do we use randomness of the input permutations?
Let us choose a number K. We can now find all elements of c_i less than K by simply trying all pairs $a_i + b_j < K$ in $O(K^2)$ time.
For all c_i that are still not found we perform an $O(n)$ brute-force computation.

J. Just Convolution

$O(n^2)$ solution is too ineffective. How do we use randomness of the input permutations?

Let us choose a number K. We can now find all elements of c_i less than K by simply trying all pairs $a_i + b_j < K$ in $O(K^2)$ time.

For all c_i that are still not found we perform an $O(n)$ brute-force computation.

Depending on K, how many elements on average will require a brute-force?

J. Just Convolution

Proposition

Suppose $K > \sqrt{n}$. For a given index k, the probability that c_k will require a brute-force is at most $e^{-K^2/2n+O(K^3/n^2)}$.

J. Just Convolution

Proposition

Suppose $K > \sqrt{n}$. For a given index k, the probability that c_k will require a brute-force is at most $e^{-K^2/2n+O(K^3/n^2)}$.

Proof

Denote p_x the index of x in a_i. Then $b_{(k-p_0) \bmod n} \geqslant K$, probability of this is $(n-K)/n$.

J. Just Convolution

Proposition

Suppose $K > \sqrt{n}$. For a given index k, the probability that c_k will require a brute-force is at most $e^{-K^2/2n+O(K^3/n^2)}$.

Proof

Denote p_x the index of x in a_i. Then $b_{(k-p_0) \bmod n} \geqslant K$, probability of this is $(n-K)/n$.
Similarly, $b_{(k-p_1) \bmod n} \geqslant K-1$, probability of this is $(n-1-(K-1))/(n-1)$ (since we have already chosen $b_{(k-p_0) \bmod n}$).

J. Just Convolution

Proposition

Suppose $K > \sqrt{n}$. For a given index k, the probability that c_k will require a brute-force is at most $e^{-K^2/2n+O(K^3/n^2)}$.

Proof

Denote p_x the index of x in a_i. Then $b_{(k-p_0) \bmod n} \geqslant K$, probability of this is $(n-K)/n$.
Similarly, $b_{(k-p_1) \bmod n} \geqslant K-1$, probability of this is $(n-1-(K-1))/(n-1)$ (since we have already chosen $b_{(k-p_0) \bmod n}$).
Proceeding this way, we conclude that the resulting probability is

$$\frac{(n-K)^K}{n!/(n-K)!} = \frac{(1-K/n)^K}{(1-1/n)\dots(1-(K-1)/n)} = \dots$$

J. Just Convolution

Proof (continued)

$$\ldots = \exp(K \ln(1 - K/n) - \ln(1 - 1/n) - \ldots - \ln(1 - (K-1)/n))$$

J. Just Convolution

Proof (continued)

$$\ldots = \exp(K \ln(1 - K/n) - \ln(1 - 1/n) - \ldots - \ln(1 - (K-1)/n))$$

$$= \exp\left(K\left(-\frac{K}{n} + O\left(\frac{K^2}{n^2}\right)\right) + \frac{K(K-1)}{2n} + O\left(\frac{K^3}{n^2}\right)\right)$$

J. Just Convolution

Proof (continued)

$$\ldots = \exp(K \ln(1 - K/n) - \ln(1 - 1/n) - \ldots - \ln(1 - (K-1)/n))$$

$$= \exp\left(K\left(-\frac{K}{n} + O\left(\frac{K^2}{n^2}\right)\right) + \frac{K(K-1)}{2n} + O\left(\frac{K^3}{n^2}\right)\right)$$

$$= \exp\left(-\frac{K^2}{2n} + O\left(\frac{K^3}{n^2}\right)\right)$$

J. Just Convolution

Proof (continued)

$$\ldots = \exp(K\ln(1-K/n) - \ln(1-1/n) - \ldots - \ln(1-(K-1)/n))$$

$$= \exp\left(K\left(-\frac{K}{n} + O\left(\frac{K^2}{n^2}\right)\right) + \frac{K(K-1)}{2n} + O\left(\frac{K^3}{n^2}\right)\right)$$

$$= \exp\left(-\frac{K^2}{2n} + O\left(\frac{K^3}{n^2}\right)\right)$$

For the worst case $n = 2 \cdot 10^5$, choosing $K = 2000$ will produce hardly a hundred of unprocessed elements.