Long Contest Editorial

November 19, 2015

Moscow International Workshop ACM ICPC, MIPT, 2015

A. Another Rubik's Puzzle?

Given a 4×4 grid with 4 red, green, blue and yellow cells each. On each step, we can perform a cyclic shift of a row/column. Find the shortest sequence of moves that places all red, green, blue and yellow cells in rows, in that order. We know that for all configurations 12 moves are enough.

A. Another Rubik's Puzzle?

There are 16 possible different moves from each position. Full brute-force approach would have to try 16^{12} possible sequences, which is too much.

A. Another Rubik's Puzzle?

There are 16 possible different moves from each position. Full brute-force approach would have to try 16^{12} possible sequences, which is too much.
However, meet-in-the-middle approach will work nicely. Let's try all sequences of 6 first moves from the initial position, and record shortest distance for all reachable positions.

A. Another Rubik's Puzzle?

There are 16 possible different moves from each position. Full brute-force approach would have to try 16^{12} possible sequences, which is too much.

However, meet-in-the-middle approach will work nicely. Let's try all sequences of 6 first moves from the initial position, and record shortest distance for all reachable positions.

Similarly, try all sequences of 6 last moves from the final position (we would have to do them backwards, but it doesn't matter since the moves are symmetrical). Record the shortest distances as well.

A. Another Rubik's Puzzle?

There are 16 possible different moves from each position. Full brute-force approach would have to try 16^{12} possible sequences, which is too much.

However, meet-in-the-middle approach will work nicely. Let's try all sequences of 6 first moves from the initial position, and record shortest distance for all reachable positions.

Similarly, try all sequences of 6 last moves from the final position (we would have to do them backwards, but it doesn't matter since the moves are symmetrical). Record the shortest distances as well.

Check all positions reachable from both initial and final positions, and try to improve the answer by the sum of two distances.

A. Another Rubik's Puzzle?

There are 16 possible different moves from each position. Full brute-force approach would have to try 16^{12} possible sequences, which is too much.

However, meet-in-the-middle approach will work nicely. Let's try all sequences of 6 first moves from the initial position, and record shortest distance for all reachable positions.

Similarly, try all sequences of 6 last moves from the final position (we would have to do them backwards, but it doesn't matter since the moves are symmetrical). Record the shortest distances as well.

Check all positions reachable from both initial and final positions, and try to improve the answer by the sum of two distances.

Number of operations will be rouhgly $16^6 \log(16^6)$.

B. Being Solarty Systematic

We are given a set of points inside a 3-dimensional torus, along with their masses and velocites. If at an integer time moment two or more points occupy the same place, they merge into a single point with mass begin sum of all collided points' masses, and velocity becomes (roughly) average of all colliding particles' velocities. Determine the set of points after the last collision happened. All coordinates and velocities are always integer.

B. Being Solarty Systematic

How do we determine when two points will collide?

B. Being Solarty Systematic

How do we determine when two points will collide?

The time t satisfies equations

$x_1 + tv_{1x} \equiv x_2 + tv_{2x} \pmod{n_x}$

$y_1 + tv_{1y} \equiv y_2 + tv_{2y} \pmod{n_y}$

$z_1 + tv_{1z} \equiv z_2 + tv_{2z} \pmod{n_z}$

B. Being Solarty Systematic

How do we determine when two points will collide?
The time t satisfies equations
$x_1 + tv_{1x} \equiv x_2 + tv_{2x} \pmod{n_x}$
$y_1 + tv_{1y} \equiv y_2 + tv_{2y} \pmod{n_y}$
$z_1 + tv_{1z} \equiv z_2 + tv_{2z} \pmod{n_z}$
Each equation is equivalent to a linear modular equation $ax \equiv b \pmod{c}$. This can be solved in a standard manner: divide a, b, c by $GCD(a, c)$ (when possible), then solve the modular inverse problem with Euclid's algorithm.

B. Being Solarty Systematic

How do we determine when two points will collide?
The time t satisfies equations
$x_1 + tv_{1x} \equiv x_2 + tv_{2x} \pmod{n_x}$
$y_1 + tv_{1y} \equiv y_2 + tv_{2y} \pmod{n_y}$
$z_1 + tv_{1z} \equiv z_2 + tv_{2z} \pmod{n_z}$
Each equation is equivalent to a linear modular equation $ax \equiv b \pmod{c}$. This can be solved in a standard manner: divide a, b, c by $GCD(a, c)$ (when possible), then solve the modular inverse problem with Euclid's algorithm.
Each of the equation will either produce a requirement $t \equiv z \pmod{c}$, or will imply that such t doesn't exist. All the requirements can be merged together using Chinese remainder theorem. Thus a single collision can be determined in $O(\log C)$ time, where $C = \max(n_x, n_y, n_z)$.

B. Being Solarty Systematic

How do we handle collisions of all points?

B. Being Solarty Systematic

How do we handle collisions of all points?
There will be at most $n - 1$ pairwise collisions to consider. To determine first collision, for each pair of points determine the first moment of collision, and choose the earliest one.

B. Being Solarty Systematic

How do we handle collisions of all points?
There will be at most $n - 1$ pairwise collisions to consider. To determine first collision, for each pair of points determine the first moment of collision, and choose the earliest one.
To repeatedly find the next collision, maintain an `std::set` of collisions. On each step, choose the earliest collision, erase all collisions which are no longer concerned with existing points, add new collisions for the new merged point.

B. Being Solarty Systematic

How do we handle collisions of all points?

There will be at most $n-1$ pairwise collisions to consider. To determine first collision, for each pair of points determine the first moment of collision, and choose the earliest one.

To repeatedly find the next collision, maintain an `std::set` of collisions. On each step, choose the earliest collision, erase all collisions which are no longer concerned with existing points, add new collisions for the new merged point.

At the start, we have to add $O(n^2)$ collisions into the `std::set`. For each collision, we have to do $O(n \log n)$ amount of work, for $O(n)$ operations with the `std::set`. That yields an $O(n^2(\log n + \log C))$ solution.

B. Being Solarty Systematic

How do we handle collisions of all points?

There will be at most $n-1$ pairwise collisions to consider. To determine first collision, for each pair of points determine the first moment of collision, and choose the earliest one.

To repeatedly find the next collision, maintain an `std::set` of collisions. On each step, choose the earliest collision, erase all collisions which are no longer concerned with existing points, add new collisions for the new merged point.

At the start, we have to add $O(n^2)$ collisions into the `std::set`. For each collision, we have to do $O(n \log n)$ amount of work, for $O(n)$ operations with the `std::set`. That yields an $O(n^2(\log n + \log C))$ solution.

Since n is small, a more straightforward $O(n^3 \log C)$ solution can pass. On each step, we can simply find pairwise collisions and choose the earliest.

C. Cyber-ZOO

A robot is standing in each of n intersections. If we press a red/green button, all robots standing on i-th intersection move to r_i-th/g_i-th intersection. Determine if we can gather all the robots at a single intersection.

C. Cyber-ZOO

This is basically the synchronizing word problem for DFA (deterministic finite automaton).

C. Cyber-ZOO

This is basically the synchronizing word problem for DFA (deterministic finite automaton).

A *synchronizing word* of a DFA is a word such that sends all starting states to the same state. We have to determine if a synchronizing word exists.

C. Cyber-ZOO

Statement

A synchronizing word exists iff for any pair of states there is a word that sends them to the same state.

C. Cyber-ZOO

Statement

A synchronizing word exists iff for any pair of states there is a word that sends them to the same state.

Proof

Suppose there is a pair of states which are not sent to the same state with any word. Then, clearly, no synchronizing word exists.

C. Cyber-ZOO

Statement

A synchronizing word exists iff for any pair of states there is a word that sends them to the same state.

Proof

Suppose there is a pair of states which are not sent to the same state with any word. Then, clearly, no synchronizing word exists. Assume the contrary, each pair of states can be synchronized. Let S be the set of different states after following the current word w.

C. Cyber-ZOO

Statement

A synchronizing word exists iff for any pair of states there is a word that sends them to the same state.

Proof

Suppose there is a pair of states which are not sent to the same state with any word. Then, clearly, no synchronizing word exists. Assume the contrary, each pair of states can be synchronized. Let S be the set of different states after following the current word w. While $|S| > 1$, choose two different states from S and append the synchronizing word for this pair w' to w. All states from S must follow w', and the size of S will decrease. Eventually, S will contain only one element, and thus w is a synchronizing word.

C. Cyber-ZOO

How to check if all pairs of states can be synchronized?

C. Cyber-ZOO

How to check if all pairs of states can be synchronized?

Construct a graph with vertices in pairs (v, u), where v and u are DFA states. For every symbol c, add edge from (v, u) to $(f(c, v), f(c, u))$.

C. Cyber-ZOO

How to check if all pairs of states can be synchronized?
Construct a graph with vertices in pairs (v, u), where v and u are DFA states. For every symbol c, add edge from (v, u) to $(f(c, v), f(c, u))$.
A pair (v, u) can be synchronized iff a vertex of the form (w, w) is reachable from it.

C. Cyber-ZOO

How to check if all pairs of states can be synchronized?
Construct a graph with vertices in pairs (v, u), where v and u are DFA states. For every symbol c, add edge from (v, u) to $(f(c, v), f(c, u))$.
A pair (v, u) can be synchronized iff a vertex of the form (w, w) is reachable from it.
To determine this, run a DFS from all states of the form (w, w) using reversed edges. The reachable states are exactly the synchronizable pairs.

C. Cyber-ZOO

How to check if all pairs of states can be synchronized?

Construct a graph with vertices in pairs (v, u), where v and u are DFA states. For every symbol c, add edge from (v, u) to $(f(c, v), f(c, u))$.

A pair (v, u) can be synchronized iff a vertex of the form (w, w) is reachable from it.

To determine this, run a DFS from all states of the form (w, w) using reversed edges. The reachable states are exactly the synchronizable pairs.

The complexity is $O(\alpha n^2)$, where α is the size of the alphabet. In our problem, $\alpha = 2$.

D. Diversity of Tree

We are given a tree on n vertices, each vertex has a color. We choose k random vertices and build the diameter of the induced tree (among several possible diameters minimize indices of its ends). What is the expected number of different colors among vertices lying on the diameter?

D. Diversity of Tree

First, for all pairs of vertices v, u find d_{vu} — the distance between v and u, and c_{vu} — number of different colors on the path between v and u.

D. Diversity of Tree

First, for all pairs of vertices v, u find d_{vu} — the distance between v and u, and c_{vu} — number of different colors on the path between v and u.
All these values can be computed in $O(n^2 \log n)$ with a DFS from each vertex that maintains a set of all colors met on the path.

D. Diversity of Tree

Now, what is the probability that (v, u) is minimal among the diameters? Clearly, v and u should be among the chosen vertices.

D. Diversity of Tree

Now, what is the probability that (v, u) is minimal among the diameters? Clearly, v and u should be among the chosen vertices. Also, for every chosen vertex w $d_{vw} \leqslant d_{vu}$ and $d_{wu} \leqslant d_{vu}$ must hold.

D. Diversity of Tree

Now, what is the probability that (v, u) is minimal among the diameters? Clearly, v and u should be among the chosen vertices. Also, for every chosen vertex w $d_{vw} \leqslant d_{vu}$ and $d_{wu} \leqslant d_{vu}$ must hold.

If these conditions hold for all chosen vertices w, then vu is indeed a diameter, but probably not a lexicographically minimal one.

D. Diversity of Tree

Now, what is the probability that (v, u) is minimal among the diameters? Clearly, v and u should be among the chosen vertices. Also, for every chosen vertex w $d_{vw} \leqslant d_{vu}$ and $d_{wu} \leqslant d_{vu}$ must hold.

If these conditions hold for all chosen vertices w, then vu is indeed a diameter, but probably not a lexicographically minimal one.

If more strict restrictions $w > v || d_{wu} > d_{vu}$ and $w > u || d_{vw} > d_{vu}$ hold, then (v, u) is the lexicographically minimal diameter.

D. Diversity of Tree

Now, what is the probability that (v, u) is minimal among the diameters? Clearly, v and u should be among the chosen vertices. Also, for every chosen vertex w $d_{vw} \leqslant d_{vu}$ and $d_{wu} \leqslant d_{vu}$ must hold.
If these conditions hold for all chosen vertices w, then vu is indeed a diameter, but probably not a lexicographically minimal one.
If more strict restrictions $w > v || d_{wu} > d_{vu}$ and $w > u || d_{vw} > d_{vu}$ hold, then (v, u) is the lexicographically minimal diameter.
If the number of vertices satisfying all these conditions is m, then we should add $c_{vu}\binom{m}{k-2}/\binom{n}{k}$ to the answer.

D. Diversity of Tree

Now, what is the probability that (v, u) is minimal among the diameters? Clearly, v and u should be among the chosen vertices. Also, for every chosen vertex w $d_{vw} \leqslant d_{vu}$ and $d_{wu} \leqslant d_{vu}$ must hold.
If these conditions hold for all chosen vertices w, then vu is indeed a diameter, but probably not a lexicographically minimal one.
If more strict restrictions $w > v || d_{wu} > d_{vu}$ and $w > u || d_{vw} > d_{vu}$ hold, then (v, u) is the lexicographically minimal diameter.
If the number of vertices satisfying all these conditions is m, then we should add $c_{vu} \binom{m}{k-2} / \binom{n}{k}$ to the answer.
m can be computed straightforwardly in $O(n)$ by checking all possible w. Thus we obtain an $O(n^3)$ solution.

D. Diversity of Tree

Now, what is the probability that (v, u) is minimal among the diameters? Clearly, v and u should be among the chosen vertices. Also, for every chosen vertex w $d_{vw} \leqslant d_{vu}$ and $d_{wu} \leqslant d_{vu}$ must hold.

If these conditions hold for all chosen vertices w, then vu is indeed a diameter, but probably not a lexicographically minimal one.

If more strict restrictions $w > v || d_{wu} > d_{vu}$ and $w > u || d_{vw} > d_{vu}$ hold, then (v, u) is the lexicographically minimal diameter.

If the number of vertices satisfying all these conditions is m, then we should add $c_{vu}\binom{m}{k-2}/\binom{n}{k}$ to the answer.

m can be computed straightforwardly in $O(n)$ by checking all possible w. Thus we obtain an $O(n^3)$ solution.

The final, most heavy part can be optimized ~ 32 times by using bit operations.

E. Expectation

We are given n points in the plane. We choose a random point q inside a rectangle $[0; X] \times [0; Y]$. Find the expectation of squared distance to the second closest point.

E. Expectation

Consider the case when p_i is the closest point, and p_j is the second closest point. Where can q lie under these conditions?

E. Expectation

Consider the case when p_i is the closest point, and p_j is the second closest point. Where can q lie under these conditions?
We have inequalities $d(q, p_i) < d(q, p_j)$, and $d(q, p_j) < d(q, p_k)$ for all k different from i and j.

E. Expectation

Consider the case when p_i is the closest point, and p_j is the second closest point. Where can q lie under these conditions?
We have inequalities $d(q, p_i) < d(q, p_j)$, and $d(q, p_j) < d(q, p_k)$ for all k different from i and j.
Each of these inequalities correspond to a half-plane given by midperpendicular of two concerned points.

E. Expectation

Consider the case when p_i is the closest point, and p_j is the second closest point. Where can q lie under these conditions?
We have inequalities $d(q, p_i) < d(q, p_j)$, and $d(q, p_j) < d(q, p_k)$ for all k different from i and j.
Each of these inequalities correspond to a half-plane given by midperpendicular of two concerned points.
Thus, for each p_i and p_j, q can lie inside the intersection of the original rectangle and several half-planes. This region is a convex polygon (or an empty set) and can be built in $O(n^2)$ time by repeated intersection of the current polygon and the next half-plane.

E. Expectation

It suffices to count the expectation of squared distance from q to p_j if q is inside the polygon (denote it P).

E. Expectation

It suffices to count the expectation of squared distance from q to p_j if q is inside the polygon (denote it P).
This can be considered a two-dimensional integral of $d^2(q, p_j)$ over P.

E. Expectation

It suffices to count the expectation of squared distance from q to p_j if q is inside the polygon (denote it P).
This can be considered a two-dimensional integral of $d^2(q, p_j)$ over P.
The integral can be separated into sum of integrals over directed trapezoids lying under sides of the polygon.

E. Expectation

It suffices to count the expectation of squared distance from q to p_j if q is inside the polygon (denote it P).
This can be considered a two-dimensional integral of $d^2(q, p_j)$ over P.
The integral can be separated into sum of integrals over directed trapezoids lying under sides of the polygon.
The squared distance is a sum of quadratic polynomials in x and y, and can be integrated directly.

E. Expectation

It suffices to count the expectation of squared distance from q to p_j if q is inside the polygon (denote it P).

This can be considered a two-dimensional integral of $d^2(q, p_j)$ over P.

The integral can be separated into sum of integrals over directed trapezoids lying under sides of the polygon.

The squared distance is a sum of quadratic polynomials in x and y, and can be integrated directly.

The total complexity is $O(n^4)$.

F. Flight Cage

We are given a set of n rectangles in the plane, one of which is a *main* rectangle. Also there is a line segment. Determine the total length of the segments' parts from which the view of the main rectangle is not obstructed by any other rectangles.

F. Flight Cage

If a rectangle A is partly obstructed by another rectangle B from some point of view, then one of the sides of B partly obstructs one of the sides of A.

F. Flight Cage

If a rectangle A is partly obstructed by another rectangle B from some point of view, then one of the sides of B partly obstructs one of the sides of A.

- Thus, we can find the subset of the point-of-view segment such that a side of the main rectangle is not obstructed by sides of all other rectangles.

F. Flight Cage

If a rectangle A is partly obstructed by another rectangle B from some point of view, then one of the sides of B partly obstructs one of the sides of A.

- Thus, we can find the subset of the point-of-view segment such that a side of the main rectangle is not obstructed by sides of all other rectangles.
- Then, we will intersect the subsets for all sides of the main rectangle.

F. Flight Cage

If a rectangle A is partly obstructed by another rectangle B from some point of view, then one of the sides of B partly obstructs one of the sides of A.

- Thus, we can find the subset of the point-of-view segment such that a side of the main rectangle is not obstructed by sides of all other rectangles.
- Then, we will intersect the subsets for all sides of the main rectangle.

We have thus reduced the problem to finding the subset of the point-of-view segment from which a segment obstructs the view of the other segment.

F. Flight Cage

How to determine from which points of view one segment obstructs the other segment?

F. Flight Cage

How to determine from which points of view one segment obstructs the other segment?

- A segment AB obstructs a point C from the point of view D iff segments AB and CD intersect.

F. Flight Cage

How to determine from which points of view one segment obstructs the other segment?

- A segment AB obstructs a point C from the point of view D iff segments AB and CD intersect.
- A segment AB obstructs a segment CD from the point of view E iff it obstructs both C and D.

F. Flight Cage

How to determine from which points of view one segment obstructs the other segment?

- A segment AB obstructs a point C from the point of view D iff segments AB and CD intersect.
- A segment AB obstructs a segment CD from the point of view E iff it obstructs both C and D.
- When moving along the point-of-view segment EF, the fact of obstruction between segments AB and CD may change only at points where lines AC, AD, BC, BD intersect the segment EF. Thus, we can build all the intersection points and obtain subsegments on which the result doesn't change.

F. Flight Cage

How to determine from which points of view one segment obstructs the other segment?

- A segment AB obstructs a point C from the point of view D iff segments AB and CD intersect.
- A segment AB obstructs a segment CD from the point of view E iff it obstructs both C and D.
- When moving along the point-of-view segment EF, the fact of obstruction between segments AB and CD may change only at points where lines AC, AD, BC, BD intersect the segment EF. Thus, we can build all the intersection points and obtain subsegments on which the result doesn't change.
- For each subsegment choose a point inside of it (e.g. the middle of the segment) and check the obstruction directly; thus we will obtain the result for the whole subsegment.

F. Flight Cage

Performing this procedure for all rectangles, we will obtain $O(n)$ subsegments from which the main rectangle is obstructed.

F. Flight Cage

Performing this procedure for all rectangles, we will obtain $O(n)$ subsegments from which the main rectangle is obstructed.
Find their union in a standard sort-and-sweep manner. The total complexity is $O(n \log n)$.

G. Game Physics

Three balls 1, 2, and 3 are located on the table, and also there are two holes at the corners of the table. We have to locate a cue ball on a fixed horizontal line on the table and hit it so that:

- the cue ball hits the ball 1, which in turn hits the ball 3, which hits the top right hole
- after reflecting from the ball 1, the cue ball hits the ball 2, which hits the top left hole

Determine if this is possible to perform. See the problem statement for description how balls are reflected on collision.

G. Game Physics

Without much detail and formulas, the solution goes as follows:

G. Game Physics

Without much detail and formulas, the solution goes as follows:

- The direction which ball 3 (that is, directly to the top right hole) has to follow uniquely determines the point where the ball 1 should hit it, and therefore determines the direction of the ball 1. (Note that the point may be impossible to hit)

G. Game Physics

Without much detail and formulas, the solution goes as follows:

- The direction which ball 3 (that is, directly to the top right hole) has to follow uniquely determines the point where the ball 1 should hit it, and therefore determines the direction of the ball 1. (Note that the point may be impossible to hit)
- Similarly, the direction of the ball 1 determines the point where the cue ball hits it.

G. Game Physics

Without much detail and formulas, the solution goes as follows:

- The direction which ball 3 (that is, directly to the top right hole) has to follow uniquely determines the point where the ball 1 should hit it, and therefore determines the direction of the ball 1. (Note that the point may be impossible to hit)
- Similarly, the direction of the ball 1 determines the point where the cue ball hits it.
- Additionally, the direction of the ball 2 determines the point where the cue ball hits it, and the direction of the cue ball after reflection from the ball 1.

G. Game Physics

Without much detail and formulas, the solution goes as follows:

- The direction which ball 3 (that is, directly to the top right hole) has to follow uniquely determines the point where the ball 1 should hit it, and therefore determines the direction of the ball 1. (Note that the point may be impossible to hit)
- Similarly, the direction of the ball 1 determines the point where the cue ball hits it.
- Additionally, the direction of the ball 2 determines the point where the cue ball hits it, and the direction of the cue ball after reflection from the ball 1.

Note that reflections can be traced backwards. We can start following the cue ball back from the collision with the ball 2, check that it hits the ball 1 where it has to, and finally obtain its direction before all collisions.

G. Game Physics

How to determine the point of collision of two balls moving with given velocities?

G. Game Physics

How to determine the point of collision of two balls moving with given velocities?
One of the balls may be considered static.

G. Game Physics

How to determine the point of collision of two balls moving with given velocities?

One of the balls may be considered static.

Let the moving ball have radius r. Shrink the moving ball and expand the static ball by r.

G. Game Physics

How to determine the point of collision of two balls moving with given velocities?

One of the balls may be considered static.

Let the moving ball have radius r. Shrink the moving ball and expand the static ball by r.

The moving ball is now a point, so it suffices to intersect a straight ray and a circle, which is done in a standard manner.

H. Herrings

Given two large numbers n and l, count the number of ways to represent $n = x + y + z$, such that $x, y, z \geqslant l$ and none of x, y and z contain 3 in their decimal representation.

H. Herrings

This is a standard application of digit-wise DP.

H. Herrings

This is a standard application of digit-wise DP.
Let us construct x, y and z from the least significant digits.

H. Herrings

This is a standard application of digit-wise DP.

Let us construct x, y and z from the least significant digits.

Suppose that we have placed k least digits so that the k least digits of $x + y + z$ match those of n.

H. Herrings

This is a standard application of digit-wise DP.
Let us construct x, y and z from the least significant digits.
Suppose that we have placed k least digits so that the k least digits of $x + y + z$ match those of n.
In order to place the next digit, we have to know the carry c from the previous digit, as well as which of the numbers x, y and z are less than the number formed by the k least digits of l.

H. Herrings

This is a standard application of digit-wise DP.
Let us construct x, y and z from the least significant digits.
Suppose that we have placed k least digits so that the k least digits of $x + y + z$ match those of n.
In order to place the next digit, we have to know the carry c from the previous digit, as well as which of the numbers x, y and z are less than the number formed by the k least digits of l.
Make all these parameters of DP. That is, we count $dp_{k,m,c}$, where k is the number of considered digits, $m \in [0; 7]$ encodes the comparisons between x, y and z with suffix of l, and c is the amount of carry from the least k digits when computing $x + y + z$.

H. Herrings

Try all possible options of choosing d_1, d_2 and d_3 (but don't forget to forbid the 3's!) such that the $(k+1)$-th digit matches, that is,
$d_1 + d_2 + d_3 + c \equiv n_{k+1} \pmod{10}$,
where n_{k+1} is the $(k+1)$-th least digit of n.

H. Herrings

Try all possible options of choosing d_1, d_2 and d_3 (but don't forget to forbid the 3's!) such that the $(k+1)$-th digit matches, that is, $d_1 + d_2 + d_3 + c \equiv n_{k+1} \pmod{10}$,
where n_{k+1} is the $(k+1)$-th least digit of n.
Consider the number $\overline{d_1 x}$ (x prepended with the digit d_1). If $d_1 \neq l_{k+1}$ ($(k+1)$-th least digit of l), then the result of comparison between x and suffix of l is determined only by d_1.

H. Herrings

Try all possible options of choosing d_1, d_2 and d_3 (but don't forget to forbid the 3's!) such that the $(k+1)$-th digit matches, that is, $d_1 + d_2 + d_3 + c \equiv n_{k+1} \pmod{10}$,
where n_{k+1} is the $(k+1)$-th least digit of n.
Consider the number $\overline{d_1 x}$ (x prepended with the digit d_1). If $d_1 \neq l_{k+1}$ ($(k+1)$-th least digit of l), then the result of comparison between x and suffix of l is determined only by d_1.
If $d_1 = l_{k+1}$, then the result of comparison is the same as comparing the shorter suffix, which is stored as a parameter.

H. Herrings

Try all possible options of choosing d_1, d_2 and d_3 (but don't forget to forbid the 3's!) such that the $(k+1)$-th digit matches, that is,
$d_1 + d_2 + d_3 + c \equiv n_{k+1} \pmod{10}$,
where n_{k+1} is the $(k+1)$-th least digit of n.
Consider the number $\overline{d_1 x}$ (x prepended with the digit d_1). If
$d_1 \neq l_{k+1}$ ($(k+1)$-th least digit of l), then the result of comparison between x and suffix of l is determined only by d_1.
If $d_1 = l_{k+1}$, then the result of comparison is the same as comparing the shorter suffix, which is stored as a parameter.
The answer is in the state when all digits are considered, all of x, y and z are not less than l, and no further carry is needed.

H. Herrings

Try all possible options of choosing d_1, d_2 and d_3 (but don't forget to forbid the 3's!) such that the $(k+1)$-th digit matches, that is,
$d_1 + d_2 + d_3 + c \equiv n_{k+1} \pmod{10}$,
where n_{k+1} is the $(k+1)$-th least digit of n.
Consider the number $\overline{d_1 x}$ (x prepended with the digit d_1). If $d_1 \neq l_{k+1}$ ($(k+1)$-th least digit of l), then the result of comparison between x and suffix of l is determined only by d_1.
If $d_1 = l_{k+1}$, then the result of comparison is the same as comparing the shorter suffix, which is stored as a parameter.
The answer is in the state when all digits are considered, all of x, y and z are not less than l, and no further carry is needed.
In the general case of d-based numeral system, and k summands instead of 3, the complexity of this solution is $O(\log n 2^k k \cdot d^{k-1})$.

I. Integer Sequence

Given a sequence of integers, answer the queries of two types:

- assign x to all $a_l, \dots, a_r$
- for all i from l to r, assign $GCD(a_i, x)$ if $a_i > x$, else leave a_i as it is

I. Integer Sequence

Queries of type 1 can be processed with a standard segment tree with lazy propagation.

I. Integer Sequence

Queries of type 1 can be processed with a standard segment tree with lazy propagation.

To process queries of type 2 we will do the following:

I. Integer Sequence

Queries of type 1 can be processed with a standard segment tree with lazy propagation.
To process queries of type 2 we will do the following:

- Maintain segments of consecutive equal elements (*blocks*) after every modification of the array (this can be done in amortized $O(\log n)$ time per modification if we use an `std::set`-like structure).

I. Integer Sequence

Queries of type 1 can be processed with a standard segment tree with lazy propagation.

To process queries of type 2 we will do the following:

- Maintain segments of consecutive equal elements (*blocks*) after every modification of the array (this can be done in amortized $O(\log n)$ time per modification if we use an `std::set`-like structure).
- Store maximal elements in the same segment tree we use to process queries of type 1.

I. Integer Sequence

Queries of type 1 can be processed with a standard segment tree with lazy propagation.
To process queries of type 2 we will do the following:

- Maintain segments of consecutive equal elements (*blocks*) after every modification of the array (this can be done in amortized $O(\log n)$ time per modification if we use an `std::set`-like structure).
- Store maximal elements in the same segment tree we use to process queries of type 1.
- To process a single query of type 2 on the segment $[l; r]$:

I. Integer Sequence

Queries of type 1 can be processed with a standard segment tree with lazy propagation.
To process queries of type 2 we will do the following:

- Maintain segments of consecutive equal elements (*blocks*) after every modification of the array (this can be done in amortized $O(\log n)$ time per modification if we use an `std::set`-like structure).
- Store maximal elements in the same segment tree we use to process queries of type 1.
- To process a single query of type 2 on the segment $[l; r]$:
  - Extract maximum a_i on the segment $[l; r]$

I. Integer Sequence

Queries of type 1 can be processed with a standard segment tree with lazy propagation.
To process queries of type 2 we will do the following:

- Maintain segments of consecutive equal elements (*blocks*) after every modification of the array (this can be done in amortized $O(\log n)$ time per modification if we use an `std::set`-like structure).
- Store maximal elements in the same segment tree we use to process queries of type 1.
- To process a single query of type 2 on the segment $[l; r]$:
  - Extract maximum a_i on the segment $[l; r]$
  - If $a_i \leqslant x$, then all elements on the segment are at most x, and nothing should be changed. Halt.

I. Integer Sequence

Queries of type 1 can be processed with a standard segment tree with lazy propagation.
To process queries of type 2 we will do the following:

- Maintain segments of consecutive equal elements (*blocks*) after every modification of the array (this can be done in amortized $O(\log n)$ time per modification if we use an `std::set`-like structure).
- Store maximal elements in the same segment tree we use to process queries of type 1.
- To process a single query of type 2 on the segment $[l; r]$:
  - Extract maximum a_i on the segment $[l; r]$
  - If $a_i \leqslant x$, then all elements on the segment are at most x, and nothing should be changed. Halt.
  - Otherwise, locate the block $[L; R]$ which contains a_i. Assign $GCD(a_i, x)$ to all elements inside $[l; r] \cap [L; R]$.

I. Integer Sequence

Queries of type 1 can be processed with a standard segment tree with lazy propagation.
To process queries of type 2 we will do the following:

- Maintain segments of consecutive equal elements (*blocks*) after every modification of the array (this can be done in amortized $O(\log n)$ time per modification if we use an `std::set`-like structure).
- Store maximal elements in the same segment tree we use to process queries of type 1.
- To process a single query of type 2 on the segment $[l; r]$:
  - Extract maximum a_i on the segment $[l; r]$
  - If $a_i \leqslant x$, then all elements on the segment are at most x, and nothing should be changed. Halt.
  - Otherwise, locate the block $[L; R]$ which contains a_i. Assign $GCD(a_i, x)$ to all elements inside $[l; r] \cap [L; R]$.
  - Repeat until $a_i \leqslant x$ condition on the step 2 is met.

I. Integer Sequence

This algorithm works in $O(T \log n)$ per query, where T is the number of blocks with elements greater than x.

I. Integer Sequence

This algorithm works in $O(T \log n)$ per query, where T is the number of blocks with elements greater than x.
A single query can clearly take a long time, but can we bound the total working time?

I. Integer Sequence

This algorithm works in $O(T \log n)$ per query, where T is the number of blocks with elements greater than x.
A single query can clearly take a long time, but can we bound the total working time?
Let $[L_i; R_i]$ be the blocks with elements x_i.

I. Integer Sequence

This algorithm works in $O(T \log n)$ per query, where T is the number of blocks with elements greater than x.
A single query can clearly take a long time, but can we bound the total working time?
Let $[L_i; R_i]$ be the blocks with elements x_i.
Introduce the following quantity:

$$P = \sum \log_2 x_i,$$

where the sum is taken over all blocks.

I. Integer Sequence

Observation

Let A be the maximal possible value of a_i. Then, every query of type 1 increases P by at most $2\log_2 A$.

I. Integer Sequence

Observation

Let A be the maximal possible value of a_i. Then, every query of type 1 increases P by at most $2\log_2 A$.

Proof

Some blocks can be deleted, at most one block is divided into two parts, and at most one new block is created.

I. Integer Sequence

Observation

If T blocks were considered while processed a query of type 2, P decreases by at least $T - 2\log_2 A$.

I. Integer Sequence

Observation

If T blocks were considered while processed a query of type 2, P decreases by at least $T - 2\log_2 A$.

Proof

At most two new blocks are created at the ends of $[l;r]$ (the query segment).

I. Integer Sequence

Observation

If T blocks were considered while processed a query of type 2, P decreases by at least $T - 2\log_2 A$.

Proof

At most two new blocks are created at the ends of $[l; r]$ (the query segment).
On each considered segment, the value of x_i is changed to a proper divisor of x_i, and thus is divided by at least 2.

I. Integer Sequence

Together with the fact that P is always non-negative, we obtain

Statement

The sum of T over all queries of type 2 is at most $O(m \log A)$, where m is the total number of queries.

I. Integer Sequence

Together with the fact that P is always non-negative, we obtain

Statement

The sum of T over all queries of type 2 is at most $O(m \log A)$, where m is the total number of queries.

It follows that the complexity of this solution is $O(m \log A \log n)$.

J. Johnny's Quest

We are given a sequence of n integers a_i. We can choose two non-empty subsets S and T such that:

- all elements of S lie to the left of all elements of T
- XOR of all elements of S is equal to XOR of all elements of T

In how many ways S and T can be chosen?

J. Johnny's Quest

Count DP $c_{k,x}$ — the number of subsets of $a_1, \ldots, a_k$ having XOR $= x$.

J. Johnny's Quest

Count DP $c_{k,x}$ — the number of subsets of $a_1, \ldots, a_k$ having XOR $= x$.
Similarly, count DP $d_{k,x}$ — the number of subsets of $a_k, \ldots, a_n$ having XOR $= x$.

J. Johnny's Quest

Count DP $c_{k,x}$ — the number of subsets of $a_1, \ldots, a_k$ having XOR $= x$.
Similarly, count DP $d_{k,x}$ — the number of subsets of $a_k, \ldots, a_n$ having XOR $= x$.
The answer is

$$\sum_{k=1}^{n} \sum_{x} (c_{k,x} - c_{k-1,x}) d'_{k+1,x},$$

where $d'_{k+1,x}$ is the number of *non-empty* subsets of $a_{k+1}, \ldots, a_n$ with XOR x (that is, $d_{k+1,x}$ if $x \neq 0$, and $d_{k+1,x} - 1$ if $x = 0$).

J. Johnny's Quest

Count DP $c_{k,x}$ — the number of subsets of $a_1, \ldots, a_k$ having XOR $= x$.
Similarly, count DP $d_{k,x}$ — the number of subsets of $a_k, \ldots, a_n$ having XOR $= x$.
The answer is

$$\sum_{k=1}^{n} \sum_{x} (c_{k,x} - c_{k-1,x}) d'_{k+1,x},$$

where $d'_{k+1,x}$ is the number of *non-empty* subsets of $a_{k+1}, \ldots, a_n$ with XOR x (that is, $d_{k+1,x}$ if $x \neq 0$, and $d_{k+1,x} - 1$ if $x = 0$).
The complexity is $O(nA)$, where A is maximal value of a_i.

K. KenKen You Do It?

We are given a connected set of cells on a grid, an operation (+, -, *, /) and the target result S. Count the number of ways to place digits into cells so that the operation applied to all the numbers gives result S, and no two cells in the same row or column contain equal digits.

K. KenKen You Do It?

Cases - and /: trivial.

K. KenKen You Do It?

Cases - and /: trivial.
Cases + and *: optimized brute-force.

L. Leprechaun Hunt

We are given a graph. Two players are playing a game.

- First player chooses V vertices of the graph and places his tokens on them.
- Second player chooses a free vertex and places his token.
- On each turn, a player can move one of his tokens from a vertex to an adjacent vertex.
- The first player wins if he captures second player's token.

Determine if the first player can win, and find the minimal number of moves required to win.

L. Leprechaun Hunt

A fairly standard game analysis.

L. Leprechaun Hunt

A fairly standard game analysis.
The state of the game is fully given by the set of vertices occupied by first player's tokens and the vertex of the second player's token.

L. Leprechaun Hunt

A fairly standard game analysis.
The state of the game is fully given by the set of vertices occupied by first player's tokens and the vertex of the second player's token.
The number of states is $O(n2^n)$, and the total number of transitions is $O(nm2^n)$.

L. Leprechaun Hunt

A fairly standard game analysis.
The state of the game is fully given by the set of vertices occupied by first player's tokens and the vertex of the second player's token.
The number of states is $O(n2^n)$, and the total number of transitions is $O(nm2^n)$.
If we consider only sets of size V, the number of states becomes $O(\sqrt{n}2^n)$.

M. Mosaic

A grid is given, with some cells colored black and some of the black cells containing numbers.
We have to fill some of the white cells with black triangles so that

- Each white region is a rectangle (possibly rotated by 45 $^\circ$)
- Each black cell with number x has exactly x adjacent cells with triangles

We know that such coloring is unique. Find the number of triangles in it.

M. Mosaic

Assume that the board is surrounded by non-numbered black cells.

M. Mosaic

Assume that the board is surrounded by non-numbered black cells. For a given coloring, how can we check if the coloring is valid?

M. Mosaic

Assume that the board is surrounded by non-numbered black cells.
For a given coloring, how can we check if the coloring is valid?
It suffices to look only at 2×2 subrectangles. If the coloring is invalid, then in some subrectangle an invalid situation occurs (a side of a rectangle is discontinued or continued at invalid angle).

M. Mosaic

Assume that the board is surrounded by non-numbered black cells. For a given coloring, how can we check if the coloring is valid? It suffices to look only at 2×2 subrectangles. If the coloring is invalid, then in some subrectangle an invalid situation occurs (a side of a rectangle is discontinued or continued at invalid angle). After precomputing all valid 2×2 simple brute-force approach works fast enough.