A

A. Alice's Nim

Determine the winning player of a Nim game with an additional condition: moving to a position with XOR of heap sizes $= 0$ is prohibited.

A. Alice's Nim

Recall the usual Nim game winning condition: a player can win the game iff the XOR of heap sizes $\neq 0$. Proof sketch (induction by the total number of stones):

- The terminal state (all empty heaps) has XOR $= 0$.
- For a position with XOR $= x$ there are moves to positions with XOR $= 0, \ldots, x - 1$, and no move to XOR $= x$.

A. Alice's Nim

Proposition

In the modified Nim game, a position is a win for the first player if XOR > 1, and a lose if XOR $= 1$.

Proof by induction. The terminal states (without any legal moves) consist of an odd number of heaps of one stone each, XOR $= 1$.

From the properties of the usual Nim game, any legal move from a XOR $= 1$ position makes XOR > 1 (since XOR $= 0$ is disallowed), and from a XOR > 1 position it is always possible to move to XOR $= 1$.

A. Alice's Nim

The final detail is that an initial position with XOR $= 0$ is not automatically a win, since the rules only prohibit to *move* to a XOR $= 0$ position.

If the initial position has XOR $= 0$, one has to check if it is possible to move to a XOR $= 1$ position, for example, by trying to modify each heap to obtain XOR $= 1$ and checking if this move is legal (the size decreases).

An easier criterion: there has to be an odd-sized heap (proof is an easy exercise).

Complexity $O(n)$.

B

B. Bag with Gifts

There are n kinds of gifts, each with a size and a price. For each k from 1 to M, determine the largest total price of a collection of gifts with total size $\leqslant C$ such that a gift of each kind is either present k times or not present at all.

B. Bag with Gifts

$k = 1$: standard backpack problem.

- $maxPrice[i][j]$ = the largest total price of a set of unique gifts among kinds $1, \ldots, i$ with total size $\leqslant j$.
- $maxPrice[0][\text{any value}] = 0$.
- $maxPrice[i][j] = \max(maxPrice[i-1][j], maxPrice[i-1][j - s[i]] + c[i])$ (either leave the gifts of kind i alone, or take a single one).
- The answer to the problem is $maxPrice[n][C]$.

B. Bag with Gifts

Consider an arbitrary $k > 1$. Any valid answer for this k can be obtained by taking a set of unique gifts of total size $\leqslant C/k$ and multiplying each gift k times…

…hence the answer for k is $maxPrice[n][\lfloor C/k \rfloor] \times k$.

Luckily, we have already computed all required $maxPrice[n][\ldots]$ while solving the $k = 1$ problem, thus no extra work is required.

Complexity $O(nC + M)$.

C

C. Colored Drinks

There are n kinds of liquids, each one has certain color and density. For a given sequence of colors, determine if it possible to create a non-mixing drink with this color sequence.

C. Colored Drinks

For each color, consider all liquids of this color and sort them by increasing of density.

If the top layer in the drink should have color c, it is optimal to choose the lightest liquid with color c (indeed, every other liquid of this color can be replaced with the lightest one without violating the non-mixing requirement).

Similarly, the next layer should be as light as possible without mixing with the first layer, and so on.

Each step can be done with binary search in the sorted liquid list for the respective color. Complexity $O(n \log n)$.

D

Drawing Magic Triangles

Find the area of intersection of n given triangles.

Drawing Magic Triangles

A halfplane is a region determined by an inequality $ax+by+c \geqslant 0$. Each triangle can be represented as an intersection of three *halfplanes*. That is, the problem boils down to intersecting $3n$ halfplanes and finding the area.

Drawing Magic Triangles

This is a fairly standard geometry task. Here's a rough description of one of the faster methods:

- Divide the half-planes into two groups: $a > 0$ and $a < 0$ (ignore $a = 0$ case for the sake of simplicity).
- In each group, sort the halfplanes by angle of the corresponding line. We will compute the intersection of first i halfplanes iteratively.
- The intersection is going to be a convex region, with its border represented as a sequence of sides (that can be either segments or rays).
- Maintain a stack of vertices of the boundary. When adding a new halfplane h, remove the vertices from stack while they do not belong to h, and introduce a new point of intersection with the last region side (a lot similar to iterative convex hull construction).

Drawing Magic Triangles

- The last remaining step is to intersect the two resulting regions for two halfplane groups (assume that neither of them is empty, nor is the whole plane).
- One can show that the border of each of the two regions contains exactly one point (x, y) for each value of y; introduce two functions $f_{a>0}(y)$ and $f_{a<0}(y)$ as the x-coordinate of the respective point for each of the two regions.

- If $f_{a>0}(y) > f_{a<0}(y)$ for any y, then the intersection is empty. Otherwise, find the two roots L and R of $f_{a>0}(y) - f_{a<0}(y)$ with binary search. The border of the whole intersection should now be limited to the region $L \leqslant y \leqslant R$.
- The total complexity should be $O(n \log n)$.

E

E. Enlarge Circles

Given n points (x_i, y_i) in the plane, assign them non-negative values $r_i \geqslant 0$ such that $\sum r_i \to \max$ and $r_i + r_j \leqslant \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ for each pair $i < j$.

E. Enlarge Circles

We start by building matrix of pairwise distances $d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ and place ∞ on main diagonal ($d_{i,i} = \infty$).

Consider optimal answer $r_1, r_2, \ldots, r_n$, one can prove that $2 \cdot \sum_{i=1}^{n} r_i \leqslant C$, where C is the minimum weight assignment $p_1, p_2, \ldots, p_n$, $C = \sum_{i=1}^{n} d_{i,p_i}$. One can think of it as a perfect matching of minimum total cost.

The above can be show by simple summation of $r_i + r_{p_i} \leqslant d_{i,p_i}$. However, is it true that value of minimum cost matching can be achieved?

E. Enlarge Circles

Recall the Hungarian algorithm to solve assingment problem. Given $n \times n$ matrix $a_{i,j}$ it finds permutation $p_1, p_2, \ldots, p_n$ that minimizes $\sum_{i=1}^{n} a_{i,p_i}$. Moreover, it solves dual problem and finds two arrays u_i and v_i that satisfy $u_i + v_j \leqslant a_{i,j}$ for every pair (i, j) and $\sum_{i=1}^{n} u_i + v_i$ is maximum possible.

However, for symmetric a ($a_{i,j} = a_{j,i}$) there is no need to have two arrays u_i and v_i as we can replace them with $w_i = \frac{u_i + v_i}{2}$. Thus, minimum weight matching is the answer for w_i maximization.

The last question remaining is, how we deal with $r \geqslant 0$ condition? Consider some optimal answer and point (x_i, y_i) such that $r_i < 0$. There is only one circle j that covers this point. We can reduce its radius by x and respectively increase r_i by x.

F

F. Formula

You are given a string s of L digits and + characters. Transform this string into a valid arithmetic expression with value at most N by replacing as few characters as possible.

F. Formula

Let $f_{i,j}$ be the smallest value of a valid expression obtained from the first i characters by j replacements, or ∞ if no valid expression can be produced this way.

General idea of calculating $f_{i,j}$: choose the last summand length k, and the number c of replacements we dedicate to changing this summand (note that we may need to use one of these replacements to make the character immediately before the summand equal to `+`).

Then

$$f_{i,j} = \min_{k,c} f_{i-k-1,j-c} + \mathit{summand}(i-k, i-1, c),$$

where $\mathit{summand}(l, r, c)$ is the smallest number we can obtain on the segment $[l, r]$ using c replacements.

F. Formula

A few observations:

- Since the total sum can never exceed 10^9, we should never use $k > 10 = \log_{10} \max N + 1$.
- If $k < i$ and $s_{i-k} \neq +$, we have to transform s_{i-k} to `+`. Also if $s_{i-k+1} = 0$, we have to transform it to 1. Let c_0 be the number of these "obligatory" transformations.
- Let $c' = c - c_0$ be the number of remaining transformations we make in this summand. If $k > 2$, then there is an optimal solution with $c' = 0$. Indeed, if we make any extra replacements, we may transform s_{i-2} to `+` instead, resulting in a smaller value; this case is going to be covered by another DP transition.

F. Formula

To sum up, $f_{i,j}$ can be found in one of the following ways:

- Choose $1 \leqslant k \leqslant 10$, and take the summand $s[i-k, i-1]$ without changes when it's valid (possibly paying to s_{i-k-1} to `+`, and s_{i-k} from 0 to 1).
- Take $k = 1$, change s_{i-1} to 0 and s_{i-2} to `+`.
- Take $k = 2$, change $s[i-2, i-1]$ to "10" and s_{i-3} to `+`.

The total complexity is $O(L^2 \log N)$ ($O(L^2 \log^2 N)$ is fine too?).

G

G. GuruGuru

Given a vehicle originally facing north and a sequence of clockwise and counter-clockwise $\frac{\phi}{2}$ rotations, find out the number of times it made a full turn in counter-clockwise direction.

G. GuruGuru

For each prefix of the given sequence compute the difference between number of 'R' and 'L', denote the i-th difference as b_i.

Now, $b_i \mod 4 = 0$ stands for position where vehicle faces north. Consider all such (i, j) that $b_i \mod 4 = 0$, $b_j \mod 4 = 0$ and $b_k \mod 4 \neq 0$ for any $i < k < j$.

Just check that vehicle performs a full turn in counter-clockwise direction. That means $b_j - b_i = 4$.

H

H. Hand Pattern

We have a deck of $n \times m$ cards, with each rank from 1 to m present n times. We choose l random cards from the deck. We also have l patterns, each pattern is either `*` or (`a`/`b`/`c`)+k. If `a`+k is present, then `a`+$(k-1)$ is present as well. Find the probability that the chosen cards can be matched with patterns so that for some values of a, b, c the ranks match with the respective pattern values (`*` can be any card).

H. Hand Pattern

First, how do we check if a given hand $r_1, \ldots, r_l$ matches the patterns? Try all options of $a, b, c \in \{r_1, \ldots, r_l\}$, for each of them check if there enough cards of required ranks to match with patterns. This procedure takes $\sim O(L^4)$ operations.

Let $r_1 \leqslant \ldots \leqslant r_l$ be the hand sorted by ranks. Clearly, the matchability only depends on the differences $\Delta_1, \ldots, \Delta_{l-1}$, with $\Delta_i = r_{i+1} - r_i$.

Furthermore, let $\Delta'_i = \min(\Delta_i, 2)$ be the *reduced hand differences.* One can see that the matchability only depends on Δ'_i: indeed, all the cards matching $a, a+1, \ldots, a+k$ have to be within a consecutive rank segment present among $r_1, \ldots, r_l$, hence if $\Delta_i \geqslant 2$ then no two cards in $r_1, \ldots, r_i$ and in $r_{i+1}, \ldots, r_l$ can be matched to $a+i$ and $a+j$ simultaneously.

H. Hand Pattern

It now suffices to try all possible sequences of reduced differences $0 \leqslant \Delta'_i \leqslant 2$ (there are 3^{l-1} of them) and check the matchability for each of them. It now suffices to compute the probability that given reduced differences occur.

For clarity: we will identify each card with a pair (r, c) with $1 \leqslant r \leqslant m$ and $1 \leqslant c \leqslant n$.

This probability is $1/\binom{nm}{l}$ times the product of the following values:

- the number of unordered hands $1 \leqslant r_1 \leqslant \ldots \leqslant r_l \leqslant m$ giving rise to the given reduced differences.
- the number of ways to choose cards of given ranks: $\prod \binom{n}{a_i}$, where a_i are numbers of cards of each particular rank.

H. Hand Pattern

To compute the first quantity, we choose the smallest rank r_1, as well as $\delta_i = \Delta_i - \Delta'_i \geqslant 0$. We have to have $\delta_i = 0$ whenever $\Delta'_i < 2$, as well as

$$r_1 + \sum_{i=1}^{l-1} \delta_i \leqslant m - \sum_{i=1}^{l-1} \Delta'_i.$$

Let x be the number of $\Delta'_i = 2$. With a standard combinatorial argument about ordered partitions, then answer is then

$$\binom{m - \sum \Delta'_i + x}{x+1}$$

H. Hand Pattern

The total complexity per test case is $O(3^l l^4)$. We also use binomial coefficients and factorials, but they can be precomputed (and the parameters are small anyway).

I

I. Infinite Multiplication Table

Given an integer S compute the number of quadriples (a, b, c, d) where all values are integer and $0 < a \leqslant b$, $0 < c \leqslant d$ and $\sum_{i=a}^{b} \sum_{j=c}^{d} i \cdot j = S$.

I. Infinite Multiplication Table

$\sum_{i=a}^{b} \sum_{j=c}^{d} i \cdot j = S$ can be represented as $\sum_{i=a}^{b} = A$, $\sum_{j=c}^{d} = B$ and $S = A \cdot B$.

We now should try all representations of S as a product of $A \cdot B$, and for each valid pair (A, B) increase the answer by $f(A) \cdot f(B)$, where $f(x)$ is the number of pairs (i, j) such that sum of all integers from i to j inclusive equals x.

Values of $f(x)$ can be precomputed by trying all reasonable i and j. For fixed i there is no need to try $j > i + \frac{S}{i}$ so the total number of pairs is $O(S \log S)$.

J

J. Jiro's Job

Given undirected unweighted graph $G = (V, E)$, starting vertex s and target vertex t, find out the minimum possible prime length of a path connecting s and t (not necessary simple).

J. Jiro's Job

First test whether there exists a path of length exactly 2. That can be done in linear time by trying every vertex as an intermediate.

Now we know that the length we are looking for is odd. Moreover, if we have a path of length x, we can obtain a path of length $x + 2$ by just traversing some edge two more times (there and back).

The above means we should look for the minimum odd x such that there exists a path of length exactly x.

J. Jiro's Job

The problem of finding shortest odd path can be solved with the following trick. Create two copies of vertices of graph G, denote them as G_e and G_o. Each edge $(u, v) \in E(G)$ is replaced with two directed arcs: one from u_o to v_e and one from u_e to v_o.

The above transformation implies that every path from s_e to t_o is of odd length. Moreover, for every odd path from s to t in G there exists a path from s_e to t_o in the new graph.

Overall complexity is $O(|V| + |E|)$.

K

K. K-rough Sorting

You are given a permutation $p_1, p_2, \ldots, p_n$. Perform minimum number of swaps of adjacent elements to obtain a permutation with no more than k inversions. If there are several optimal answers pick lexicographically smallest resulting permutation.

K. K-rough Sorting

Each swap of adjacent elements changes (decreases or increases) the number of inversions by 1. Thus, the number of swaps we should perform is $s = \max(0, I(p) - k)$, where $I(p)$ stands for the number of inversions in permutation p.

To decrease number of inversions we should only perform reasonable swaps, i.e. select such i that $p_i > p_{i+1}$.

Any sequence of s reasonable swaps solves the first part of the problem. Now we should find a way to construct lexmin resulting permutation.

K. K-rough Sorting

What is the minimum possible first element? To push some value x at position i to position 1 we need to spend $i - 1$ swaps. All these swaps will be reasonable unless there is some element $y < x$ at position $j < i$. In this case we should push element y to position 1.

Thus, we should use the following procedure. For every position i from 1 to n we select minimum possible x in range from i to n such that we have enough swaps remaining to push it to position i. Naive implementation of this algorithm runs in $O(n^2)$ time.

K. K-rough Sorting

How do we speed up this algorithm? If there are more than $n-i$ swaps remaining in stock we just take the minimum element. Otherwise, let's say there are s swaps left. The minimum element we can put at position i is the minimum value in range from i to $i+s$.

We need a data structure that is able to remove elements and tell the minimum among first s remaining elements. That can be done with a segment tree that stores a pair $(1, p_i)$ at the i leaf and each subtree contains a pair (cnt, min) — the number of active elements on this fundamental segment and the minimum value among them.

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

L

L. Lucky Tickets

You have a number x with $2n$ digits (possibly with leading zeros). Find the smallest y such that $x \leqslant y < 10^{2n} - 1$ such that y and $y+1$ are lucky according to at least one of two definitions.

L. Lucky Tickets

The general idea is standard: let t be the length of the common prefix of x, y and $y+1$. The $(t+1)$-th digit of y is either greater than $(t+1)$-th digit of x, or the digits are equal, in which case all digits of x following $(t+1)$-th are 9's.

Let us choose the types of luckiness for y and $y+1$ in one of the four ways (in fact, the types have to be different, hence there are only two ways). The suffixes of y and $y+1$ following the t-th digit have to satisfy conditions of sort "the difference of odd and even digits is z_1" or "the difference of digits in left and right half is z_2", where z_1 and z_2 are defined by first t digits of x (therefore, fixed).

L. Lucky Tickets

Let $f(z_1, z_2)$ be the smallest number with $2n$ digits satisfying the difference conditions. If first t digits of $f(z_1, z_2)$ are zero, then we can paste it together with the first t digits of x to obtain a solution, otherwise there is no solution for this particular t.

The problem boils down to finding $f(z_1, z_2)$. We reduce it to a variation of the shortest path problem as follows. Let $g(z_1, z_2)$ be the length of $f(z_1, z_2)$ without leading zeros. We then seek to extend $f(z_1, z_2)$ in all possible ways by prepending it with more digits, arriving at candidates for other values $f(z_1', z_2')$.

L. Lucky Tickets

There are a few concerns:

- y and $y+1$ may have suffixes consisting of 9's and 0's respectively, which we cannot obtain by prepending equal digits. We will resolve this by initializing $g(z_1, z_2)$ with $y = d \cdot 10^k - 1$ for all values of $1 \leqslant d \leqslant 9$ and $0 \leqslant k \leqslant 2n$.
- We may have to skip a few zeros before appending the digit. However, at any position each digit d can change z_1 and z_2 in one of four ways ($z_1 \pm d$, $z_2 \pm d$). For each of these modifications we should locate the rightmost digit to the left of $f(z_1, z_2)$ and place d there. This way there should be at most 4×9 transitions leaving each state.
- Since $|z_1|, |z_2| \leqslant 9n$, the number of states can be roughly estimated as $(18n)^2$, which is a lot. However, a lot of these states are unreachable, thus lazy DP computation of $g(z_1, z_2)$ should work within reasonable time.