A

A. Alice and Maze

We are standing in a graph, possessing a certain amount of treasure. Traversing each edge infers a cost (when we don't have enough treasure, we can't use that edge). Each time we arrive at a vertex, we gain additional treasure. Determine the largest amount of treasure we can have when arriving at the exit (or that this amount can be arbitrarily large, or that we can't get to the exit at all).

A. Alice and Maze

First, remove parts of the graph unreachable from the start even with unlimited treasure. If the exit is unreachable this way, the answer is -2.

Otherwise, we can use Bellman-Ford-like technique: let d_v be the largest amount of treasure we can have while at the vertex v. Initially $d_1 = q$ (the initial amount), $d_v = -\infty$ for all others.

Next, perform n relaxation phases: for every edge $x \to y$, if $d_x \geqslant cost(x, y)$, make $d_y = \max(d_y, d_x - cost(x, y) + gain_y$.

A. Alice and Maze

The only situation when we couldn't have found the correct answer at the end of this procedure is when there is a cyclic route reachable from the start that can be used indefinitely to increase the amount of treasure (this would imply answer -1).

To recognize this situation, perform one more relaxation phase. If *any* of the d_v values change after this, then -1 is the correct answer, otherwise output d_n.

The complexity is $O(nm)$.

B

B. Buggy Stack Machine

Perform operations with a stack machine. Whenever the smallest bits of the top k entries match a pattern, erase them all.

B. Buggy Stack Machine

The only problem is how to match the top of the stack against the pattern fast enough. Possible approaches include:

- deamortized KMP (prefix function automaton) or Z-function (regular KMP shouldn't work since it's $O(n)$ for adding a single character in the worst case);

- rolling hashes.

None of these approaches infer asymptotically significant work. The total complexity is still $O(n)$.

C

C. Constellation

We have an $m \times m$ matrix with pairwise distances between some points in the plane. Find the number of ways to choose m points out of a given set S of n points so that to match with the given distances.

C. Constellation

Suppose that we've chosen positions for the first two points p_1 and p_2 (out of m). Then there are at most two ways to place all the other points.

Indeed, any other point p_i has to satisfy $dist(p_1, p_i) = m_{1,i}$, $dist(p_2, p_i) = m_{2,i}$, that is, it has to lie on a two circles intersection. Since $p_1 \neq p_2$, the intersection consists of at most two points.

Further, as long as we choose a location for p_i not collinear with p_1 and p_2, all other points' locations are unambiguous.

This allows for an $O(n^2 m)$ solution: try all locations for p_1 and p_2, then check if other points can be placed using a hash set S.

C. Constellation

Now, let us instead try all locations for p_1. p_2 has to satisfy $p_2 = p_1 + v$, with $|v|^2 = m_{1,2}$, where $m_{1,2} \leqslant 10^8$.

It can be checked explicitly that for any value of $m_{1,2} \leqslant 10^8$ there are less than $K = 300$ suitable vectors v.

Thus, the complexity of the optimized solution becomes $O(nKm)$.

D

D. Domination

In a tournament (a directed graph with each pair of vertices connected with exactly one arc in one of the possible direction), find the smallest set of vertices S so that (x, y) is an arc for any $x \in S$ and $y \notin S$.

D. Domination

The condensation of a tournament is a bamboo (that is, strongly connected components of a tournament can be ordered $c_1, \ldots, c_k$ so that c_i dominates c_j whenever $i < j$).

By definition, the answer is $|c_1|$.

D. Domination

Even shorter solution: order all vertices by decreasing of their out-degree: $v_1, \ldots, v_n$. One can see from the above considerations that c_1 consists of several first vertices in this order.

Let $x_i = \max j : (v_j, v_i) \in E$. Then $|c_1| = \min k : x_i \leqslant k$ for all $i = 1, \ldots, k$.

In any case, the complexity is $O(n^2)$.

E

E. Elevator

There are n people riding an elevator. When an elevator stops on a floor x, all people going to the floor x, as well the floor $x - 1$ leave. Find the smallest number of stops so that everyone gets to their desired floor.

E. Elevator

Let x be the highest floor anyone wants to go to. We will have to stop on x eventually (the only other option to satisfy people going to x would be to stop on $x + 1$, but there is no sense in doing that).

Stop on x to satisfy everyone going to x and $x - 1$. If not all people are satisfied, let x be the highest floor among all remaining people and do the same, and so on.

The easiest implementation is to sort all the given floors and go from right to left. $O(n \log n)$ with quick sort, or $O(n + \max a_i)$ with counting sort.

F

F. Flow Strikes Back

Given a graph G and excess function $exc(v)$ find out any flow function f that produces given excesses. Minimize maximum value of $f(u, v)$.

F. Flow Strikes Back

First of all we transform the graph to a classic flow network by introducing two new nodes: source s and sink t. For each node v such that $exc(v) > 0$ we add an arc from s to v of capacity $exc(v)$. For each node v such that $exc(v) < 0$ we add an arc from v to t of capacity $-exc(v)$.

As we aim to minimize maximum $f(u, v)$ for all arcs where $u, v \notin \{s, t\}$. Go for a binary search, for a fixed value of arcs capacity c we should check whether there exists a flow that saturates all arc going out of s.

F. Flow Strikes Back

To achieve this taks we implement some very fast maximum flow algorithm and hope for the best. Maximum flow algorithms are known for achieving the actual running times orders of magnitude below their theoretical bounds. Though, in this particular problem the constraints are really harsh.

Algorithms to try are: Dinic, Dinic combined with scaling technique, pre-flow push algorithms family, preferrable with all major heuristics.

F. Flow Strikes Back

What was the author's intent?

We are not sure, but a brief investigation of test cases gave us an extra constraint that $\sum |b_i| \leqslant 2 \cdot 10^6$. It is not clear whether it is a coincidence or a mistake in the problem statement. However, introducing this new constrain gives an extra space to apply Karzanov bounds for the number of Dinic's algorthm phases.

There are few chances we can prove some better time bound with this extra constraint but it still will be significantly above the reasonable.

G

G. Good Substring

Given a string of length n and q consecutive change queries, after each query determine what is the longest good substring of the current version of the string. Substring is considered to be good if none of its substring of length at least two is a palindrome.

G. Good Substring

First we would like to solve the problem with no queries at all and establish a simple criteria for string goodness.

Notice, that if substring from l to r is a palindrome, substring from $l + 1$ to $r - 1$ is a palindrome as well. Thus, we only need to check that string has no palindromic-substrings of length 2 or 3.

There are $O(n)$ palindromes of length 2 or 3. If s_i and s_{i+1} form a palindrome, they can't be present in a good substring simultaneously. If s_i, s_{i+1} and s_{i+2} form a palindrome, only s_i and s_{i+1} or s_{i+1} and s_{i+2} can be present in any good substring.

G. Good Substring

What can we say about the form of any good substring? Imagine we fixed some position l as the leftmost element of a good substring. What is the maximum r we can use as a rightmost element? It is the maximum r such that there are no bad pairs or bad triples on the segment from l to r.

The above remark gives us an understanding that the string s can be represented as a decomposition to almost non-overlapping good substring. "Almost", because if a triple breaks a good substring in two, its middlepoint can be present in both of them.

We maintain a set of all maximal good substrings and a separate multiset of their lengths. Constant number of changes are required per each query giving the overall running time $O((n+q)\log n)$.

H

H. House Selection

Given an unweighted tree find out the number of triples (u, v, w) such that $\rho(u, v) = p$, $\rho(v, w) = q$ and these shortest paths share no common edges.

H. House Selection

In other words we have to find the number of triples (u, v, w) such that $\rho(u, v) = p$, $\rho(v, w) = q$ and $\rho(u, w) = p + q$. Or simply the number of simple paths of length $p + q$.

Consider any node as a root of the tree. Now, for each node v we would like to compute the number of paths of length $p + q$ that have this node as a point of minimal depth.

For each subtree we compute $c(u, x)$ — the number of nodes in a subtree of u that have $depth(v) - depth(u) = x$ (relative depth).

H. House Selection

What is the number of simple paths of length $p + q$ that have v as a heighest point? It is no greater than $\sum_{i=0} p + qc(v, i) \cdot c(v, p + q - i)$. However, this way we might have selected paths that go down to the same subtree, so we should subtract $\sum_{i=0} p + q - 2c(u, i) \cdot c(u, p + q - 2 - i)$ for each $u \in Children(v)$.

The above computations can be done in $O(|Children(v)| \cdot (p + q))$ time for each node v that sums to $O(n \cdot (p + q))$ total running time.

I

I. Intelligence and Magic

You are given a collection of n permutation-functions f_i, i.e. $f_i(1), f_i(2), \ldots, f_i(m)$ form a permutation of integers from 1 to m for each i from 1 to n. You goal is to find any permutation $p_1, p_2, \ldots, p_m$, such that function g_i defined as $g_i(x) = f_i(p_x)$ is unimodal for every i.

I. Intelligence and Magic

Consider the element x of lowest priority for f_1, i.e. $f_1(x) = 1$. It should be a leftmost or a rightmost element for this function. Without loss of generality we can place $p_x = 1$.

Now, if $f_i(x) = 1$ for any $i > 1$ we simply forget about this element and proceed without it. Otherwise, let $f_i(x) = y > 1$ for some i. For all values z such that $f_i(z) < y$ we know they should be granted highest values in p. If you consider a plot of function f_i, these values go to the right, while x stays as the leftmost element. Moreover, we know the order, $p_n = f_i(1), p_{n-1} = f_i(2)$ and so one.

I. Intelligence and Magic

Notice that we made a decision to put x to the left at the very beginning (and it was of no difference due to the overall symmetry) and all later moves were just determined consequences.

Continue to apply all implications till any. We either face a contradiction or a situation, that there exists some j such that for every i we have already placed first j elements of function i. It means that these prefixes match as sets and do not affect values of larger priority in any way.

In the latter case we treat the situation as the beginning of the algorithm and without loss of generality place $f_1(x) = j + 1$ to the leftmost available position.

J

J. Jack Barmer

Given a field of size $n \times m$ with some cells selected as special, select the minimum subset of non-special cells such that there exists a path starting at $(1, 1)$ and ending at (n, m) that passes only through special and selected non-special cells and visits every special cell exactly once.

J. Jack Barmer

Author's solution is bad and he should feel bad.

K

K. Knockout Tournament

A tournament among n players is held, and an outcome matrix is known. Count the number of

tournament schemes so that a specified player wins, and the height of the tournament is smallest possible (that is, equal to $\lceil \log_2 n \rceil$).

K. Knockout Tournament

Dynamic programming: $number_{A,i,h}$ = the number of sub-tournaments of height $\leqslant h$ among the players in the set A so that i is the winner.

Base cases: $number_{\{i\},i,h} = 1$.

Recalculation:

$$number_{A,i,h} = \sum_{\substack{B \subset A \\ i \in B}} \sum_{\substack{j \in B \setminus A \\ i \text{ wins } j}} number_{B,i,h-1} \cdot number_{A \setminus B,j,h-1}$$

The answer is $number_{\{1,\ldots,n\},M,\lceil \log_2 n \rceil}$.

Complexity $O(3^n n^2 \log n)$, but a lot of states can be skipped (for instance, when $2^h < |A|$, or $M \in A$ but $i \neq M$).

L

L. Love Power Plant

Choose as many disjoint `LOVE` subsequences in a given string as possible.

L. Love Power Plant

A greedy approach works: for the first subsequence take the earliest `L` followed by the earliest subsequent `O`, and so on. Mark these letters as used and repeat, until no more subsequences can be found.

Why does this work? Let $l_1 < \ldots < l_k$ be the positions of `L`'s used in an optimal answer, similarly $o_1 < \ldots < o_k$, $v_1 < \ldots < v_k$, $e_1 < \ldots < e_k$.

We must have $l_1 < o_1 < v_1 < e_1$, otherwise there would be no way to choose the subsequences (for instance, if $o_1 > l_1$, then either o_1 can not be preceded by any `L` at all).

L. Love Power Plant

Proceeding in a similar way, we must have that $l_i < o_i < v_i < e_i$ for any $i = 1, \ldots, k$, which implies that a greedy algorithm would obtain an answer with the same cardinality (if not better).

Implementation: store sorted lists of occurences of each letter, and maintain pointers for the currently used letter of each type. Iterate over l_i, move pointers for other letters forward to ensure $l_i < o_i < v_i < e_i$.

Complexity $O(n)$.