Day 4 Contest: Worldwide Selection 2

Moscow International Workshop 2020

December 1, 2020

A. Antiplagiarism

Keywords: parsing.

Carefully read the programs line by line, and replace line breaks with spaces.
Apply the rules to reduce each program. Replace identifiers with unique strings in order of their appearance, and compare the two programs.

B. Block Game

Keywords: BFS.

Let S be the set of positions that end up in the middle square. There are $\binom{24}{8} < 10^6$ possible sets.
Run a reverse BFS from the middle set to find the shortest distance from each possible S, storing optimal transitions.
For a given board, try all three options of S, and choose the smallest distance.

C. Conference

Keywords: subset DP.

Let $f_{i,mask}$ be 1 if it is possible to direct first i cars without collision, and the set of stops of the car i is $mask$.
We can check if $(i, mask_1) \to (i+1, mask_2)$ is a legal transition by explicitly looking for collisions. With this, we can obtain a straightforward $O(mn2^{2n})$ solution.
The factor of n can be eliminated with careful recursion.

This is somewhat close to TL, but we can optimize this further. Note that if $mask_1 \subset mask_2$, and $(i, mask_1)$ is reachable, then the state $(i, mask_2)$ is obsolete, since it's going to be harder to avoid at any point.
By only considering $mask$ minimal by inclusion, the number of relevant $mask$ for any i is $O(2^n/\sqrt{n})$ (see Sperner's theorem).
With careful implementation, this can lead to an optimization by a further factor of n.

D. Decorative Floor

Keywords: profile DP.

Number all cells left to right, top to bottom. Let $dp_{i,mask}$ be the lowest cost to cover all cells up to i, and a subset $mask$ among cells $i+1, \ldots, i+2h+2$.
If the cell $i+1$ is not in the shape, then in $dp_{i,mask}$ we must have $mask[0] = 0$.
If the cell $i+1$ is not in the shape, or $mask[0] = 1$, then the only transition from $dp_{i,mask}$ is to $dp_{i+1,mask>>1}$.
Otherwise, try to cover cell $i+1$ with all possible brick rotations, aligning the bottom left cell of the brick with $i+1$. Make sure that no intersections occur with cells already present in $mask$, and that the brick is inside the boundaries.
The answer is $dp_{nh,0}$. **Complexity** is $O(nhm2^{2h+2})$. This can be optimized if for each $mask$ and vertical position we precompute which blocks are legal to place.

E. Equalities and Matches

Keywords: parsing.

First, obtain all possible expressions by removing a single match. Compress the string if this makes a character empty.
Then, for all such expressions try all options of adding a single match, including in between characters. Parse all resulting expressions to check if they are valid.
Complexity is $O(n^3)$, where n is the length of the expression. Optimizations are possible with DP, but not needed.

F. Fantasy Game

Keywords: combinatorics, number theory.

If $h > m$, we can not kill the monster. Let us precompute answers $ans[h]$ for all $h \leqslant m$.
Try all $d = 1, \ldots, m$. We should take $b = \lfloor m/d \rfloor$.
For these values of d, b, the probability to kill a monster with HP h depends only on $t = \lceil h/d \rceil$, and is equal to $\sum_{i=t}^{b} \binom{b}{i} p^i (1-p)^{b-i}$.
When b is large, only summands with i close to pb are significant. Also, moving from a summand to an adjacent one involves multiplying by a simple fraction (no need to precompute powers or binomials).
On each step, update $ans[td]$. In the end, update $ans[h]$ with all of $ans[h+1], \ldots$.
The total complexity of this solution is $O(\sum_{d=1}^{m} m/d) = O(m \log m)$.

G. Galactic Wormholes

Keywords: DSU.

Solve offline. For simplicity, assume the graph is connected (solution doesn't change, but discussion is easier).
Start by finding all edges that connect components when they appear, obtaining a spanning tree T. This is basically Kruskal's algorithm with the given order of edges.
Now, scan all edges again, and maintain a DSU for biconnected components. One can see that at any point each biconnected component forms a connected subgraph of T. Indeed, components are merged along a path of T when any safe edge appears.
Thus, for each biconnected component c we can keep track of $v(c)$ — vertex closest to the root of T (after initially rooting at any vertex).

If the current edge $e = xy$ is in T, or x and y are in the same biconnected component, no changes are needed.
Otherwise, look at $v(c_x)$ and $v(c_y)$. If $v(c_x)$ is farther from the root than $v(c_y)$, then c_x should be merged with the component of $parent(v(c_x))$ in T.
Proceed until x and y are in the same component.
The newly safe edges is the number of bridges removed, plus 1 if e was not a bridge.

The complexity is $O((n+m)dsu(n))$, where $dsu(n)$ is the cost of one operation with DSU with n atoms.

H. Hack the Lock

Keywords: suffix structures.

Consider the unchanged part $u = s_i \ldots s_{i+l-1}$ of s. In the end it has to match against an equal substring $t_j \ldots t_{j+l-1}$ of t.
There are two possible optimal ways to match these substrings. One of them is to move u to the left i times, then to the right $n-l$ times, the to the left $n-j-l$ times. The other one is symmetrical. Consider only the first way.
For any choice of u it is optimal to choose the leftmost occurence of u in s, and the rightmost occurence of u in t.

Construct a suffix automaton/tree of $s\$t$. Consider only the states such that ends of both strings are reachable (common substrings of s and t).
If following u leads from the root to a state x, then u should be the longest such path (otherwise, u can be safely extended to the left).
Leftmost occurence of any such u in s is obtained from the longest path from x to any state with \$ transition. Similarly, the rightmost occurence in t is obtained from the shortest path from x to the final state. All of these can be found with DP on automaton states.

Complexity is $O(n)$ for building SA, computing DPs and obtaining answers from each state.

I. Islands

Keywords: subset DP.

Construct a path by adding vertices one by one. Let $(mask, v, u)$ be a state when the path covers a vertex subset $mask$, and the last two vertices and v and u in this order.
Let $score_{mask,v,u}$ be the largest possible score when we arrive to $(mask, v, u)$, and $cnt_{mask,v,u}$ is the number of optimal ways.
Transitions from $(mask, v, u)$ are to $(mask + \{w\}, u, w)$ for all vertices w. Validity and extra score for any such transition are straightforward to find.
The answer is obtained from all values of $score_{V,v,u}$ and $cnt_{V,v,u}$, where V is the set of all vertices. Since paths are

reversable, divide the count by 2.
Complexity: $O(2^n n^3)$ per test case, with a good constant factor.

J. Joy Of The Speed

Keywords: math, numerical methods.

Depending on the speed v, we want to choose transmission i such that $P(c_i \frac{v}{v_0})$ is smallest, where $P(x) = 1 - px + qx^3$.
One can see that when $c_1 > c_2$, then $P(c_1 x) = P(c_2 x)$ has a single positive solution x_0, and $P(c_1 x) < P(c_2 x)$ when $x < x_0$, and $P(c_1 x) > P(c_2 x)$ when $x > x_0$.
x_0 can be found with binary search, or analytically since $P(c_1 x) - P(c_2 x) = x(a - bx^2)$ for some a, b.
At this point we know velocities v_i when we should switch from transmission i to $i + 1$.

Now we can binary search for the final velocity V. Within binary search, sum up distances travelled between transmission changes.
Between changes we have $dv/dt = 1/P(cv)$ with constant c, thus formally $dt = P(cv)dv$.
The distance travelled until velocity V is reached from starting velocity v_0 is $\int_{v_0 \leqslant v(t) \leqslant V} v(t)dt = \int_{v_0}^{V} vP(cv)dv$, which can be computed analytically.
The time to reach velocity V from v_0 is $\int_{v_0 \leqslant v(t) \leqslant V} dt = \int_{v_0}^{V} P(cv)dv$.
Solutions using numerical integration are also possible.

K. Kingdom of Onliners

Keywords: DFS.

Run a DFS on the given graph. If an influencer exists, than the vertex v with largest *tout* is an influencer. Check that explicitly.
If v is an influencer, than any other vertex u is an influencer if v is reachable from u. Find all of them by running a reverse DFS from v.
Complexity: $O(n + m)$.