Day 3 Contest

Moscow Pre-finals Workshop 2020

April 20, 2020

A. Artifact Autentification

Keywords: DFS.

Reconstruct the graph interactively with DFS. After this step no queries are needed, just use the two graphs you know.
Try all options of the starting vertex in the map.
For two properly edge-labeled graphs G and H with chosen starting vertices we can identify vertices of G and H with parallel DFS using edges with matching labels.
Compare sets of neighbour labels for matching vertices.
Total complexity: $O(cn^2)$ time and $O(cn)$ queries, where $c = 26$.

B. Balanced Bouldering

Keywords: shortest path, DP.

s can be bounded by $9hw$.
Create a graph with vertices (x, y, s) — currently at the hold (x, y) with stamina s, and edges leading to all nearby holds (and decreased stamina).
The graph has $\sim 9h^2w^2$ vertices and $\sim 9r^2h^2w^2$ edges. Dijkstra may be slow, but we can compute the distances for all vertices by decreasing of s.
To avoid memory issues, to find distances for s only store distances for $s + 1, \ldots, s + 9$.
Total complexity: $O(9r^2h^2w^2)$ time and $O(9h^2w^2)$ memory.

C. Counting Creatures

Keywords: math, casework.

$B[i] = A[i] - A[i + 1] = \#$ of creatures with $> i$ legs.
$C[i] = B[i] - B[i + 1] = \#$ of creatures with $i + 1$ legs.
From this, for creatures with $\geqslant K$ legs we can find out:
• the total number of creatures X;
• the total number of legs Y;
• the number of types Z.
Replace Y with $Y - KX$ and assume non-negative number of legs.
Cases $Z = 0, 1$ are trivial. For $Z = 2$ try all solutions.
For $Z > 2$ in any solution we must have $X \geqslant Z$ and $Y \geqslant L = 0 + \ldots + Z - 1$.
With casework we can prove that the solution is unique only if $Y = L$, or $X = Z$ and $Y = L + 1$.

D. Drunk Deer

Keywords: Markov chains, algebraic graph theory (?).

Let $4A$ be the adjacency matrix of the $n \times m$ grid graph. With relations for the expected number of steps $N = (I_{nm} - A)^{-1}\mathbf{1}$ and eigenvalues of A equal to $\cos(\frac{i\pi}{n+1})\sin(\frac{j\pi}{m+1})$ can ultimately reconstruct the answer **without** the missing cell.
Apparently can adapt this even with the missing cell. We'll do our best to update this later.

E. Eastern Empire

Keywords: complicated profile DP.

Process cells in row-major order, and consider placing walls adjacent to it. We need to maintain that:
• walls form several chains with their ends on the profile,
• each connected region separated by walls contains either only `c`'s or only `e`'s,

• regions with `e`'s are never enclosed by walls, and there is ultimately exactly one enclosed region with `c`'s.

Do curved profile DP, store the following: • pairs of positions where wall chains meet the profile;
• partition of the profile cells into connected components,
• for each component, if it contains `c`'s, `e`'s, or none of them.
The information is enough to keep track of all conditions.
Only keep the reachable states.

F. Funny Frogs

Keywords: math. Note that each breeding changes all $x_i \bmod (y+1)$ by -1.
Since ultimately all x_i for $i \neq c$ become 0, then all initial $x_i \bmod (y+1)$ (except x_c) must be the same.
Clearly, at least $T = \max_{i \neq c} x_i$ steps are needed. Let's try to win in T steps.
For each $i \neq c$ determine t_i — the number of times frogs i are bred. We must have $x_i + t_i \cdot y - (T - t_i) = 0$, thus $t_i = \frac{T - x_i}{y+1}$.
Conditions ensure that all divisions are possible, $t_i \geqslant 0$, $\sum t_i \leqslant T$, and it is possible to choose a suitable order of operations.
The remaining operations are spent on breeding frogs c.
Since we know the number of operations, the final number of frogs is $\sum x_i + T(y - n + 1)$.

G. Grid Game

Keywords: shortest path, meet-in-the-middle. The number of possible positions is too large for a simple BFS.
Instead, let us run find shortest paths both from the initial position, and from all possible final positions (backwards).
Maintain two BFS queues Q_1 and Q_2. If $|Q_1| \leqslant |Q_2|$, process do all forward transitions from the next state in Q_1, otherwise do all backwards transitions from the next state in Q_2.
When $Q_1 \cap Q_2$ is non-empty, we can restore the answer. In practice this reduces the number of considered states significantly.
Depending on the implementation, constant optimizations may be needed.

H. Hillary's Hobby

Keywords: scanline.
Let G and B be gray and black regions. Their borders are concentric circles.
If we know the radii of all circles, we can find the answer easily.
To find the boundary of $G \cup B$, for each rectangle find d_{min}, d_{max} — the distances to the closest and the furthest point to the origin, then unite the segments $[d_{min}, d_{max}]$.
To find the boundary of B, perform a scanline by increasing r. For each black cell c, create events when the circle starts and stops intersecting the cell c.
The number of cells intersected by a circle of radius r is $8 \cdot \lfloor r \rfloor + 4$. Create events for integer r when this value changes.
Finally, observe that all r in B are $O(n(w+h))$. Do the scanline, maintaining the number of intersected cells.
Can group squares to avoid costly sorting.
Total complexity: $O(n \log n + nwh)$.

I. Interesting Integers

Keywords: math, brute-force.
If n is even, or $2 \leqslant n \leqslant 7$, there is no answer.
A solution for $n = 9$ can be found with brute-force.
We can find that $\frac{1}{3} = \frac{1}{5} + \frac{1}{9} + \frac{1}{45}$.
Thus, to go from n to $n+2$, take any $3x$ in the solution and replace it with $5x$, $9x$, $45x$.

J. John's Jigsaw

Keywords: greedy.
Let S be the total area of the given squares. Try all options of $S = h \times w$.
For any k we must have that the total area of squares with side at least 2^k should not exceed $(h - h \bmod 2^k) \times (w - w \bmod 2^k)$.

Since after placing $2^k \times 2^k$ squares the remaining space in the truncated rectangle can always be split into smaller squares, this condition is sufficient.

K. King's Keyword

Keywords: number theory, binary search. If $xy = z$ in base b, then $xy \leqslant z$ in all bases $> b$, and $xy \geqslant z$ in all bases $< b$.
Binary search on b. Note that b is bounded $n^3 A \sim 10^{18}$, where A is the largest digit.

L. Lazy Managers

Keywords: DP, string matching.

Solution 1. Construct a trie of all strings s_i, and build suffix links with Aho-Corasick.
Now, we want to compute dp_i — the largest weight of a chain ending at s_i. To recompute we need to update $dp_i = w_i + \max_{j<i} dp_j$, where s_j is a substring of s_i.
Suppose that the occurence of s_j in s_i has its last position at p. Then s_j is reachable from the prefix $s_i[0..p]$ in the Aho-Corasick trie by suffix links.
It follows that dp_i should be updated with path maximums in the tree of suffix links.
We can update values and find path maximums in the suffix link tree with a segment tree on Euler tour. **Total complexity**: $O(S \log S)$.

Solution 2. Let S be the total length of all strings. Observe that there can be only $O(\sqrt{S})$ different lengths among the strings.
Hash all the strings, and maintain $dp[hash]$ for all hashes.
Do compute $dp[s_i]$, try all different lengths l occuring in the set, and update $dp[s_i]$ with $dp[hash(t)]$ for all substrings t of length l.
Total complexity: $O(S\sqrt{S}\log n)$, or $O(S\sqrt{S})$ with a hash-map.

M. Modern Methodics

Keywords: binary search, greedy, segment tree.

Use binary search on k. We need to check if the segments can be ordered in such a way that intersecting segments are at most k positions apart.

Construct the ordering from left to right.

Let's call a segment **active** if it has not been used in the ordering yet. For each active segment s, let r_s be the upper bound on its position in the ordering. Initially, $r_s = n$ for any s. Once a segment s' intersecting with segment s is used in the ordering at position i, r_s becomes equal to $\min(n, i+k)$ and never changes afterwards.

Suppose we have determined the first i segments of the ordering and want to decide on the $i+1$-th.

If there exists j such that **more** than $j-i$ active segments s have $r_s \leqslant j$, a valid ordering does not exist.

Otherwise, find the smallest j such that **exactly** $j-i$ active segments s have $r_s \leqslant j$. Note that j always exists, since $j = n$ satisfies the condition. The $i+1$-th segment in the ordering must be an active segment with $r_s \leqslant j$. (In fact, the next $j-i$ segments must be exactly the active segments with $r_s \leqslant j$, in some order.)

Out of active segments with $r_s \leqslant j$, it's best to put a segment that ends earlier (i.e. has the smallest b_s). It can be proven based on two facts:
• non-intersecting segments must be ordered from left to right;
• segments s' having $a_{s'} \leqslant b_s$ and $r_s = n$ will have r_s changed to $\min(n, i+k)$, and we would rather do that for fewer segments to be less constrainted in the future.

Implementation:
• one segment tree for minimum contains, in cell x, the value of x minus the number of active segments with $r_s \leqslant x$;
• another segment tree contains, in cell x, the smallest right end b_s of some active segment s having $r_s = x$;
• it is also useful to have an array of priority queues, with segments having $r_s = x$ ordered by their right ends, to help maintaining the second segment tree.

Total complexity: $O(n \log^2 n)$, where n is the number of segments.