Day 5 Contest

Moscow International Workshop 2020

December 3, 2020

A. Ancient Klingon Language

Keywords: parsing, subset DP.

The given formulas essentially consist of min and max of characters' positions in the permutation. For $i = 1, 2$, let $E_i(p)$ be the result of the expression i for a permutation p.
Also, let $E_i(p, k)$ be the result of the expression i is we replace k smallest characters with 0, and the rest with 1. Clearly, if $E_1(p) = E_2(p)$, then $E_1(p, k) = E_2(p, k)$ for any k. Further, if $E_1(p) \neq E_2(p)$, we can find such a k that $E_1(p, k) \neq E_2(p, k)$, for example, $k = \min(E_1(p), E_2(p))$.

Let us call a vector S of 0's and 1's good, if $E_1(S) = E_2(S)$. Then a good permutation p can be represented as a sequence of good vectors $S_0, \ldots, S_n$, each obtained from the previous one by adding one 1 bit.
Find all good S explicitly, and count the number of such sequences with DP. **Complexity**: $O((n + L)2^n)$, where L is the total expression length.

B. Build The Evacuation Site

Keywords: DSU.

Let us add edges by **decreasing** of their strength. If we've added all edges with strength $\geqslant s$, then the sizes of connected components correspond to values $r_s(v)$.
Maintain DSU of current connected components. We can represent all possible components at all possible times with a rooted forest: whenever several components $c_1, c_2, \ldots, c_i$ are merged after edges of strength s are added, create a vertex for the new component c, and add edges of weight s from c to all $c_1, c_2, \ldots, c_i$.
For any vertex v, let p_v be the path from a root component to the atom component with a single vertex v. Then, any two vertices v, u can be compared as follows:

1. Let c_v and c_u be the first components on p_v and p_u (these are root components in the forest).
2. If c_v and c_u have different sizes, the vertex with the larger component is better.
3. Otherwise, if subsequent edges of p_v and p_u have different weights, the vertex with the larger weight is better.
4. If c_v and c_u are atoms, then v and u are equal. Otherwise, move c_v and c_u further in their respective paths, and return to step 2.

We can use this to find all best evactuation sites as follows. Let S be the set of components that can lie on p_v for a best vertex v (initially S consists of all root components).
If S consists of atoms, print all elements of S. Otherwise, erase all elements of S with non-maximum size, and out of these with non-maximum weight of outcoming edges. Then, replace S with all children of S.

This solution can be implemented with **complexity** $O(ndsu(n))$.

C. Covid-19 Tracing

Keywords: 2020, simulation.

Keep track of who's suspicious (inititally only the person p). Process all contacts in chronological order: if one of the two contacting persons is suspicious, mark the other as suspicuous as well.
In the end, print the number of suspicuous people.

D. Delivery Fare

Keywords: factorization.

Factorize p, and list all divisors of p (there will be less than 30 000 of them).
Suppose $a \leqslant b \leqslant c$ and $abc = p$. Try all options for c, then, to minimize $a + b + c$, divisors a and b should be as close as possible under $ab = n/c$. Find b as the smallest divisor $\geqslant \sqrt{n/c}$ in the list that also divides n/c.

In practice, the last part is pretty fast. Naive factorization in $O(\sqrt{p})$ is slow, though.
We can either use fancy factorization algorithms (like Pollard rho), or use more minor optimizations, like precomputing primes under $\sqrt{10^{15}}$.

E. Edge Path (updated)

Keywords: SCC, shortest path.

Consider condensation of the graph. If the graph is already strongly connected, the answer is 0.
If there is more than one initial or terminal SCC, there is no answer, since a path can not visit two of either.
Otherwise, we have to reverse a path from the initial SCC S to the terminal SCC T. Note that if maximum flow from S to T is less than 2, there is no answer. Indeed, after reversing any path from S to T we can't reach T from S anymore.

Otherwise, reversing *any* path from S to T makes the graph strongly connected. Indeed, consider reversing a path p. From any vertex v we can reach a vertex of p, thus we can also reach S. It suffices to show that we can reach T from S.
But since max. flow is at least 2, there is an augmenting path from S to T in the residual network of the 1-flow through p. This is exactly a path we can follow after reversing p.

Using the argument above, we can see that the following solution works: reverse a shortest path p from S to T. If T is still reachable from S, print the length of p. Otherwise, there is no answer.
Complexity: $O(n + m)$ for finding SCC and running BFS and DFS.

F. Far Expedition

Keywords: shortest path, trapezoid decomposition.

Draw a horizontal line through each vertex of an obstacle in both directions, until it (properly) intersects an obstacle, or reaches $x = \pm\infty$.
It is free to travel along any horizontal line, so we can make them vertices of the graph.
(The interesting part of) the plane is now divided into trapezoids, with lower and upper side contained in one of the horizontal lines. Note that one of the sides can have zero length (thus, trapezoids can be triangles).
For each trapezoid create weighted edges between lines containing top and bottom sides (0 for downward edge, Δy for upward edge).
The resulting graph is planar, thus it contains $O(V)$ vertices and edges, where V is the total number of vertices of all obstacles.

For each query locate the trapezoids containing start and finish. If this is the same trapezoid, the answer is trivial to find. Otherwise, the answer can be found with $O(1)$ shortest path queries.
Complexity is $O(QV \log V)$. $O(V^2 \log V + Q \log V)$ is possible with precomputing pairwise distances, and offline point location for putting start and finish of each query in a trapezoid.

G. Graph Game

Keywords: network flows, linear programming.

Let $f(e)$ be the assigned probability of the edge e. The vertex player will always choose a vertex v with minimum $\sum_{e \to v} f(e) - \sum_{e \leftarrow v} f(e)$. The edge player has to maximize that minimum.
Let's minimize the negative of the result, $\max \Delta(v)$, there $\Delta(v) = \sum_{e \leftarrow v} f(e) - \sum_{e \to v} f(e)$.
Clearly $\sum_v \Delta(v) = 0$, thus $\max \Delta(v) \geqslant 0$.
If the graph contains a cycle $e_1, \ldots, e_k$, we can assign $f(e_i) = 1/k$, then all $\Delta(v) = 0$.
Let's consider the case when the graph is acyclic.

Hand-wavy solution: If $f(e)$ is the flow through the edge e, then $\Delta(v)$ is flow "appearing" at the vertex v.
Decompose the flow into paths, assuming each vertex receives flow $\Delta(v)$ from the source if $\Delta(v) \geqslant 0$, or gives $-\Delta(v)$ flow to the sink if $\Delta(v) \leqslant 0$.

Let $l(v)$ be the maximum path length starting at v, then the flow sum over all edges in paths starting at v does not exceed $\Delta(v) \cdot l(v)$.
It follows $\sum_v \Delta(v) \cdot l(v) \geqslant \sum_e f(e) = 1$, thus $\max \Delta(v) \geqslant 1/\sum_v l(v)$ (assume the contrary, and conclude that the left-hand side is less than 1).
On the other hand, for each vertex v consider a path of length $l(v)$ starting at v, and push $1/\sum_v l(v)$ flow through this path. We can verify that $\max \Delta(v) = 1/\sum_v l(v)$, thus this is exactly the answer.

Big brain solution: consider the linear program $\Delta(v) = \sum_{e \leftarrow v} f(e) - \sum_{e \to v} f(e) \leqslant 1$ for all v, $f(e) \geqslant 0$ for all e, $\sum f(e) \to \min$. If the optimum of this program is x, then the answer is $1/x$.
The dual of this program is $\Delta(v) - \Delta(u) \geqslant 1$ for all $e = v \to u$, $\Delta(v) \geqslant 0$ for all v, $\sum \Delta(v) \to \min$. This describes longest paths from all vertices, thus we immediately have the same answer.
We only need to find cycles and compute longest paths, thus **complexity** is $O(n + m)$.

H. Highways

Keywords: a bit of linear algebra, ad-hoc tree shenanigans?
Let $A_{ij} = 1$ if the edge is initially red, and 0 otherwise.
A resulting tree has at least one leaf (citation needed). Try all possible options of a leaf v and its neighbour u.
If B_{ij} is the resulting color matrix, we must have $B_{vu} = 1$, and $B_{vw} = 0$ for $w \neq u$.
For any v, u, w, any color switch operation preserves $A_{vu} \oplus A_{vw} \oplus A_{uw}$, where $\oplus$ is XOR. Thus, we can determine any $B_{ij} = A_{vi} \oplus A_{vj} \oplus A_{ij} \oplus B_{vi} \oplus B_{vj}$.

If we reconstruct B_{ij} for all v, u, and check if the resulting red graph is a tree, we obtain an $O(n^4)$ solution.
To speed-up, observe that changing u while v is fixed modifies $O(n)$ elements of B_{ij}. Let us in addition maintain the collection E of currently red edges, as well as their total cost.
If the size of E is not $n - 1$, there's not need to check if it's a tree. Otherwise, the check is done in $O(n)$ with DFS.
The solution is now $O(n^3)$, up to the complexity of maintaining E as edges are added and removed.
If E is represented e.g. as a doubly-linked list, we can insert/remove elements in $O(1)$, and list elements of E in $O(n)$. This yields true $O(n^3)$ complexity.

I. Ideal Container (updated)

Keywords: nasty geometry, binary search.
In any answer the polygons can be separated with a straight line, so that each polygon is contained in a circle segment.
Let us binary search on the radius r, and for each polygon find the smallest segment of the circle of radius r containing it.
If both segments can fit together in the circle, then the answer is $\leqslant r$.

Consider the polygon P contained in a segment. WLOG assume that the segment chord is vertical, and the polygon is on RHS of the chord.
Try moving the chord to the right, shifting and rotating P at contact points if needed. There are two cases when we can't move forward:

- there are at least two vertices of P on the border of the circle;
- one of the vertices of P is at the rightmost point $(r, 0)$, and also one of the sides of P is on the chord.

This gives $O(n^2)$ interesting positions for P. For each of them figure out the optimal segment containing P.
The chord of an optimal segment either contains a side, or it contains a vertex v and is orthogonal to Ov, where O is the circle center.
Thus, the smallest segment can found in $O(n^3)$. **Total complexity** if $O(n^3 \log \varepsilon^{-1})$, where ε is required accuracy.
Trigonometric functions are *very* slow, so avoid using them too frequently. `long double` helps with precision issues.

J. Jigsaw Puzzle

Keywords: brute-force.
Brute-force all possible arrangements.
Possible optimizations include:

- fixing position and rotation of one of the pieces,
- checking intersections in sides and corners with masks,
- pruning in recursion, etc.

K. Knight Move

Keywords: profile DP.

Connect white cells (i, j) with $(i-2, j \pm 1)$. The problem is to count the number of independent sets in this graph.
The graph splits into four independent parts: with even/odd i and even/odd $j - 2i$.
Each of these components has $\leqslant n/4$ vertices in each column. This allows to apply standard profile DP to count independent sets, and multiply answers for all parts.

Complexity: $O(nm2^{n/4})$.

L. Linear Combination of Polygons (fixed)

Keywords: linear algebra, suddenly Euclid's algorithm?.

Lemma. Consider counter-clockwise borders of convex polygons A and B, and suppose they contain directed sides $x \cdot v$ and $y \cdot v$ for some vector v, $x, y > 0$. Then the border of $A + B$ contains a side $(x + y) \cdot v$.

We are given polygons $X = aP + bQ$ and $Y = cP + dQ$, under $ad - bc = 1$, $a, b, c, d \geqslant 0$.
Observation. Suppose $a, b, c, d \geqslant 0$, $ad - bc = 1$. Then exactly one of the following holds:
- $a \geqslant c$, $b \geqslant d$ (and at least one of the inequalities is strict);
- $a \leqslant c$, $b \leqslant d$ (and at least one of the inequalities is strict);
- $a = d = 1$, $b = c = 0$.

Suppose that for any vector v the side xv of X is longer than the side yv of Y. In this case, we can *subtract* Y from X, that is, replace X with $X' = (a-c)P + (b-d)Q$ by shortening each side of X by the respective side of Y.
The observation above implies that if subtraction is possible, then it is because $a \geqslant c$, $b \geqslant d$ for minimal P, Q.

Similar to Euclid's algorithm, we can speed up by subtracting the largest possible multiple of Y (taking care not to make any of X, Y empty).
Enumerate all possible vectors v, and keep track of coefficients of respective sides of X and Y. Then, **complexity** is $O((n+m)\log \max c)$, where $\max c$ is the largest possible coordinate. This follows directly from Euclid's algorithm analysis.

M. Modifications

Keywords: data structures, centroid decomposition/Euler tour.

Build centroid decomposition, and in each node rooted at r maintain $\sum_{a,b} dist(a, b)v_a v_b$ over all pairs such that a, b are in different subtrees of r.
$dist(a, b)v_a v_b = dist(a, r)v_a v_b + dist(b, r)v_a v_b$. For each subtree, construct an Euler tour and maintain an RSQ structure of vectors $(1, v_a, dist(a, r), dist(a, r)v_a)$.
For each subtree T of r, put $x_T = \sum_{a \in T} v_A$, $y_T = \sum_{a \in T} dist(a, r)v_a$. The answer is $\sum_{T \neq T'} x_T y_{T'} = (\sum_T x_T)(\sum_T y_T) - \sum_T (x_T y_T)$. Maintain all three sums as values in subtrees change.
Changing any weight in a subtree can be done with a linear transformation of vectors in a range for Euler tour of the subtree. For example, adding x to v_a performs $(1, v_a, dist(a, r), dist(a, r)v_a) \to (1, v_a + x \cdot 1, dist(a, r), dist(a, r)v_a + x \cdot dist(a, r))$.

A solution with a single Euler tour is possible, with much larger vectors.

Complexity: $O(n + q \log^2 n)$, or $O(n + q \log n)$ with huge constant.