Day 2 Contest

Bytedance Moscow Workshops Camp 2020

May 2, 2020

A. Allocation

Keywords: implementation, standard data structures

Store all free blocks as a sequence of segments in linked list. Each allocated block contains at least 100 of 10^5 blocks; then, number of segments will not be more than 1000, and each query can be implemented in $O(1000)$ by straightforward way. Values of variables can be stored in map or hashmap (for example, std::unordered_map).

B. Binary Trees

Keywords: binary tree, SQRT-decomposition.

The solution will be offline. More precisely, we split queries on blocks; each block contains $BLOCK = \sqrt{n}$ queries.

Consider one block. Let $v_1, v_2, ..., v_{k'}$ be vertices which change in the block queries; let W be a set of $v_1, v_2, \ldots, v_{k'}$ and its pairwise LCAs; let call vertices of W white, and others - black.

Before performing queries, run DFS in $O(n)$ and check whether each vertex can be root of BST or not in assumption that all white vertices are empty, without any numbers. Then, erase all vertices which can't be roots of BST, and also all vertices which are roots of BST with only black vertices. Remaining vertices can be split in trees with white leafs; let its roots be white too. It can be proven that number of white vertices is not more than $2 * BLOCK$.

Then, fill white vertices with their current values and try to find a number of good subtrees in our trees. We do it with second DFS; let's do it in more details.

Each tree can be described as binary tree on white vertices, connected by black paths. Consider such a "metaedge" consisting of vertices $u = u_l, u_{l-1}, \ldots, u_0 = v$, u and v are white, while others are black; u is a parent of v. Each edge (u_j, u_{j-1}) can be left (u_{j-1} is left child of u_j) or right. Let j_0 be a maximum possible j such that all edges between u_{j_0} and v are of one type. Then, all vertices from u_0 to u_{l-1} are roots of BST iff u_0, u_1 and u_{j_0+1} (if it exists) are roots of BST, as it can be easily proven; moreover, if u_0 is a root of BST, but u_l is not, number of roots of BST in the metaedge can be found in $O(1)$, and u and its ancestors are not roots of BST.

During second DFS, precalculate all needed information about each metaedge, and also minimals and maximal black numbers in each subtree. After that, perform query. For each query, change corresponding white vertex and then calculate the answer using DFS which works with each metaedge in $O(1)$ using an information described above. Such a DFS will work in $O(|W|) = O(\sqrt{n})$ time, and for each block, we run two $O(n)$ DFS. Then, total complexity of the solution is $O(q\sqrt{n})$.

C. Counting Bit String

Keywords: dynamic programming.

Suppose for simplicity then $P = 5$ (other cases are simpler).

Let $dp[i][p1][p2][p3][p4]$, $1 \leqslant i \leqslant n$, $0 \leqslant p1 \leqslant p2 \leqslant p3 \leqslant p4 \leqslant L$ is a number of ways to fill first i symbols of S, or $S[1..i]$, in such a way that prefix of length i is "good" (any substring of length L or less of the prefix contains no more than P 1's, and in substring $S[i - L + 1..i]$, S contains 1's at positions $i - p1$, $i - p2$, $i - p3$, $i - p4$ for such pj that $pj < L$; if $pj = L$ then we assume that there are no more than $j - 1$ ones in last substring of length L.

Our dynamic will be forward. To go from state $(i, p1, p2, p3, p4)$, we should try to put 0 or 1 to $S[i + 1]$ In case of 0 we move to $(i + 1, p1 + 1, p2 + 1, p3 + 4, p4 + 1)$ (more precisely, $\min(pj + 1, L)$ for each j). In case

of 1, if $p4 < L$ then we should fill all symbols between $i+2$ and $i-p4+L-1$ with zeroes, so we move to $(p3+L-p4, p2+L-p4, p1+L-p4, L-p4-1, i+L-p4)$. Case $p4 = L$ is left to the reader.

So, each state can be performed in $O(1)$. Total number of states is $n*(\binom{L}{4}+\binom{L}{3}+\binom{L}{2}+\binom{L}{1}+\binom{L}{0}) \leqslant 100*251176$ ($L \leqslant 50$) which satisfies time and memory limits.

D. Deep Red

Keywords: dynamic programming

We can assume that we work with not numbers, but segments $[l_i, r_i]$ of numbers. Each segment $[l, r]$ means that on this place, only numbers $x \in [l, r]$ can occur, and l, r are possible. Initially, we have segments $[x, x]$ and $[y, y]$. Sum of $[l_1, r_1]$ and $[l_2, r_2]$ is $[l_1 + l_2 - 1, r_1 + r_2 + 1]$, and product is $[l_1 * l_2 - 1, r_1 * r_2 + 1]$ (it can be proven that making of segments with $l \leqslant 0$ doesn't have a sense).

Note that if $[l_1, r_1] \subseteq [l_2, r_2]$ then making of $[l_2, r_2]$ doesn't have a sense (if in all calculations we'll use $[l_1, r_1]$ instead of $[l_2, r_2]$ then some future segments will be replaced by their subsegments, and it'll make answer better (or at least not worse). One can get the fact about $l \leqslant 0$ as a consequence.

Suppose for simplicity that $x, y < z$. Then, the other fact we note that answer is not more then $2z$ (just add y to x while subsegment is less than $z \leqslant 1000$); so it doesn't make a sense to make a segment $[l, r]$ with $2z < l$. Let $minr[l]$, $1 \leqslant 2z$ is minimal possible r such that $[l, r]$ is a possible answer. Initially, $minr[x] = x$, $minr[y] = y$, and all other $minr$-s are inf. Then, we note that during the operations with "sensible" segments, all l-s and r-s strictly increase (multiplication on $[1, r]$ is not sensible); so iterate over all $l = 1, 2, ..., z$ and if $minr[l] <$ inf then try to add and/or multiply $[l, minr[l]]$ to $[l', minr[l']]$ for all $l' \leqslant l$ and try to relax future $minr$-s. Then, the answer can found among segments $[l, minr[l]]$, $l = 1, 2, \ldots, 2z$. Total complexity is $O(z^2)$.

E. Edge Sets

Keywords: DFS, LCA.

Let 1 be a root of tree. Then, run DFS and calculate $timeIn[v]$, $v \in V$ - times of entering of DFS in a vertex. Also calculate $depth[v]$ — distance between root and v on edges ($depth[v] = 0$).

Suppose then we have a query - vertices $v_1, v_2, \ldots, v_k$. Sort them by $timeIn[v]$. Then, let $v_0 = LCA(v_1, v_k)$. Then it's known from properties of DFS that the following pairwise non-intersection paths contain exactly all needed edges:

- path from v_0 to v_1;
- path from $lca(v_1, v_2)$ to v_2;
- path from $lca(v_2, v_3)$ to v_3;
- ...
- path from $lca(v_{k-1}, v_k])$ to v_k.

Lengths of paths can be found as a differences of depths (these paths are all "from ancestor to descendant"); after $O(n\ log\ n)$ precalculation, one can calculate any $lca(u, v)$ in $O(1)$. So, the complexity of our algorithm is $O(n\ log\ n + \sum_{i=1}^{q} k_i)$.

F. Fractal Enumeration

Keywords: math, DP, fractals.

Let $f(n, k)$ be the answer for the problem, k is even, $-2^n < k < 2^n$. Let also be $f(n, k, i)$ is a sum of values in points with $x + y = k$ in i-th quadrant, $i = 1, 2, 3, 4$, and $num(n, k, i)$ is a number of such a points in i-th quadrant. Obviously, $num(n, k, i)$ can be found in $O(1)$ for any n, k, i. Then:

- $f(n, k) = f(n, k, 1) + f(n, k, 2) + f(n, k, 3) = f(n, k, 4)$;

- $f(n,k,2) = f(n,k+2^n,1) + 4^{n-1} * num(n,k,1)$;
- $f(n,k,3) = f(n,k+2*n+1,1) + 2*4^{n-1} * num(n,k,1)$;
- $f(n,k,4) = f(n,k+2^n,1) + 3*4^{n-1} * num(n,k,1)$;
- $f(n,k,1) = f(n-1,k-2^n)$.

It means that to calculate $f(n,k)$ for some k, we should calculate $f(n-1,k \mod (2^n))$ and $f(n,k \mod (2^n) - 2^n)$; to calculate these values, it's enough to calculate $f(n-2,k \mod (2^{n-1}))$ and $f(n-2,k \mod (2^{n-1}) - 2^{n-1})$ and so on. So, one can calculate answer in $O(n)$.

The only detail is that the answer can be long, so use long arithmetics or Python/Java BigInteger for this problem.

G. Good Integers

Keywords: run-twice, matrices, eigenvalues, constructive.

First question we have is: how many possible paths of length n exist in given graph G?

Obviously, it's a sum of elements in matrix M^n, for M as adjacency matrix of the graph (more precise, $m^n[u][v]$ is a number of paths of length n between u and v).

Maximal eigenvalue of the matrix is 2; so, number of paths grows as 2^n so we don't have asymptotic reserve.

But we can find and consider eigenvector $Vec = [3,6,4,2,5,4,3,2,1]^T$ with eigenvalue 2. According to this vector, we build the following graph G':

- vertices of G' are pairs $v' = (v, num)$, $v \in V = 1,2,\ldots,9$ is a vertex of G, num is a natural number, $1 \leqslant num \leqslant Vec[v]$ (i.e. pairs $(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(2,4),(2,5),(2,6),(3,1),(3,2)$ and so on — 30 vertices in total);
- G' is **directed**;
- each vertex of G' has exactly two outgoing edges; mark these edge by letters A and B;
- for any edge $((v,num_v),(u,num_u))$ in G', edge (v,u) should exist in G;
- for any edge (v,u) of G, and for any $num_u \in \{1,2,\ldots,Vec[u]\}$ there should be edge from (v,num_v) to (u,num_u) for some num_v.

So we can consider G' as DFA. To encode the given string S, just start from, for example, vertex $(1,1)$ and go in DFA using symbols S. For each vertex (v,num) we visit, just add v to the answer. Length of encoded result is exactly $|S|$ if not print 1 at the beginning. But how to decode this path?

One can try to decode string just by another iterating over S and DFA; but it's possible that for some (v,num) there are tow outgoing edges to (u,num_1) and (u,num_2) so we can't understand whether we should go to (u,num_1) or not. We can't restore next vertex in V' - but if we have two neighbouring vertex $v,u \in G$ in encoded string and know state (u,num_u) in which we occur after corresponding move then we can restore previous (v,num_v) — as it follows from construction, there cannot be two ingoing edges to any (u,num_u) from (v,num_1) and (v,num_2) with one v and different num_1, num_2. So, the only thing we need is to find last state (w,num_w).

w is known as last symbol of encoded string; to code num_w, just add some path to $(1,1)$. num_v is from 1 to 6, so we can add strings 1, 21, 521, 6521, 76521 or 876521 for $num_v = 1,2,...,6$, respectively, to the beginning of answer during encoding phase. So, length of encoded string will be not more than $|S|+6$.

The complexity of both phases of the solution is $O(n)$.

H. Hoodies

Keywords: Fenwick tree.

In this task we need to maintain the scores for green and violet participants and ask, how many participants of another color have a lesser score.

As maximal score doesn't exceed $3 \cdot 10^5$, we can keep the number of contestants with all scores in Fenwick tree. Using Fenwick tree we can both modify their scores and answer the queries.

Another, an easier method is using gnu c++ extension for the ordered set. You can read more about that here.

I. Integer Points

Keywords: Euclid algorithm.

We need to find the $\sum_{x=1}^{n} 1 + \lfloor \sqrt{\frac{p}{q}} x \rfloor$.

Let's take $S(n, \alpha) = \sum_{k=1}^{n} \lfloor k\alpha \rfloor$. We can prove that

- $S(n, \alpha) = S(n, \alpha + 1) - \frac{n(n+1)}{2}$ if $0 < \alpha < 1$,
- $S(n, \alpha) = S(n, \alpha - 1) + \frac{n(n+1)}{2}$ if $2 < \alpha$
- $S(n, \alpha) = \frac{(n+n')(n+n'+1)}{2} - S(n', \beta)$, where $n' = \lfloor (\alpha - 1)n \rfloor, \beta = \frac{\alpha}{\alpha - 1}$, if $1 < \alpha < 2$.

Only the last case is not obvious. Let's look at α and $\beta = \frac{\alpha}{\alpha-1}$. We can see that $\frac{1}{\alpha} + \frac{1}{\beta} = 1$. By **Rayleigh theorem**, for such α and β the sequences $\lfloor \alpha n \rfloor$ and $\lfloor \beta n \rfloor$ don't intersect and contain all natural numbers in their union. So, we can take $m = \lfloor \alpha n \rfloor$, $n' = \lfloor \frac{m}{\beta} \rfloor$, and see at the sum of first m natural numbers as $S(\alpha, n) + S(\alpha, n') = \sum_{i=1}^{m} i = \frac{m(m+1)}{2}$, as these two sequences give the numbers $1, \ldots, m$ in their union.

So, we can directly calculate the answer using these 3 formulas. For some reasons, seems like there is enough precision to perform all calculations ($\alpha \to \frac{\alpha}{\alpha-1}$ don't seem to lead to high precision loss until $\alpha \approx 1$, but in this case, n is multiplied by the same $\alpha - 1$ and converges to zero very fast.)

J. Jury Troubles

Keywords: Independent set in a bipartite graph, Kuhn algorithm.

Let's find the maximal possible number of satisfied jury members. What jury members can't be satisfied together? Two jury members with votes (a_i, b_i), (a_j, b_j) can't be satisfied together if and only if $a_i = b_j$ or $a_j = b_i$. Any set of jury members without such pair can be satisfied (e.g. all a_i participants may qualify). It means that there is a bipartite graph on jury members, and we need to find the maximal independent set in this graph. It is the classical task, which can be solved by the Kuhn algorithm in $\mathcal{O}((n + m)k)$ time.

When we know, which jury members should be satisfied, we can satisfy them simply by qualifying the skaters which they want to qualify, as they don't contradict each other.

K. Kate The Robot

Keywords: constructive, heuristics.

Let us generate random mazes according to the rules, and keep a program P that solves them.
If a random maze is not solved by P, extend P in the shortest way to solve it. Repeat until random mazes are solved with large probability.
This does not give P of reasonable length, so we can use the following optimizations:
• Instead of solving mazes one by one, keep a pool of mazes M that are not solved by P. To extend P, choose a maze from M such that the extension to P is shortest. When some mazes in M are solved, re-generate them.
• After a short while, the shortest extension in the pool will always be 1. To optimize at this point, choose the next command of P to maximize the number of solved pools in M.

This may have to run for a while, but we can run this locally and only submit P.

L. Longest Arc

Keywords: geometry, convex hull, tangents.

In this task you are given a set of disjoint circles and you need to find the length of a longest circular arc in their convex hull.

How does their convex hull look like? It is a set of circular arcs, connected by some straight segments. We can see that any two consecutive circular arcs in convex hull should be connected by the common tangent line to these circles and that all interesting points in this convex hull are the tangent endpoints.

First, we can get rid of circles which are contained in any other circle. After that, we can build a usual convex hull on a set of endpoints of all common tangents to all pairs of circles. If two consecutive points at this convex hull belong to one circle, there is a circular arc between them, and a straight segment otherwise. The answer is the maximal sum of lengths of consecutive circular arcs.

This solution works in $\mathcal{O}(n^2 \log n)$ due to sorting n^2 points to build a convex hull.

M. Minimize the Similarity

Keywords: greedy, tree centroid.

Construct a trie with the given strings $s_1, \ldots, s_n$.

Let $v(s)$ be the vertex of the trie corresponding to string s, and r be the root of the trie. Let $dist(v, u)$ be the shortest path length between v and u in the trie.

Proposition. For any two strings s and t we have $|s| + |t| = dist(v(s), v(t)) + 2LCP(s, t)$, where $LCP(s, t)$ is the maximum common prefix length of s and t.

Thus for any reordering $t_1, \ldots, t_n$ of $s_1, \ldots, s_n$ we have

$$2\sum_{i=1}^{n-1} LCP(t_i, t_{i+1}) = \sum_{i=1}^{n-1}(|t_i| + |t_{i+1}| - dist(v(t_i), v(t_{i+1}))) = 2\sum_{i=1}^{n} |t_i| - (|t_1| + \sum_{i=1}^{n-1} dist(v(t_i), v(t_{i+1})) + |t_n|).$$

Since $2\sum_{i=1}^{n} |t_i|$ is constant, we want to maximize $|t_1| + \sum_{i=1}^{n-1} dist(v(t_i), v(t_{i+1})) + |t_n| = \sum_{i=0}^{n} dist(v(t_i), v(t_{i+1}))$, if we additionally put $t_0 = t_{n+1} = \varepsilon$ — the empty string.

Thus we reduce to a rooted tree problem: reorder vertices $v_1, \ldots, v_n$ so that the total length of the cycle starting at the root $r = v_0 = v_{n+1}$ and visiting $v_0, v_1, \ldots, v_{n+1}$ is maximum.

Proposition. For the optimum ordering $v_1, \ldots, v_n$ all simple paths v_0v_1, …, v_nv_{n+1} have a common vertex.

Proof. Suppose paths v_iv_{i+1} and v_jv_{j+1} do not intersect. Then the ordering $v_0, \ldots, v_i, v_j, v_{j-1}, \ldots, v_{i+1}, v_{j+1}, \ldots$ provides a better answer, since paths v_iv_j and $v_{i+1}v_{j+1}$ do intersect, thus $dist(v_i, v_{i+1}) + dist(v_j, v_{j+1}) < dist(v_i, v_j) + dist(v_{i+1}, v_{j+1})$.

If all pairs of paths have a common vertex, than all paths have a common vertex. □

Let $D(w) = \sum_{i=0}^{n} dist(w, v)$. By the triangle inequality, for any vertex w the answer is bounded from above by $2D(w)$.

But also by the last proposition, the answer is bounded from below by $2\min_w D(w)$.

The minimum is attained at a *centroid vertex* c — a vertex such that any of its (unrooted) subtrees contain at most half of the vertices $v_0, \ldots, v_n$.

To obtain the answer equal to $2D(c)$ we have to have $dist(v_i, v_{i+1}) = dist(v_i, c) + dist(c, v_{i+1})$ for all i, that is, consecutive vertices v_i and v_{i+1} cannot lie in the same (unrooted) subtree of c.

To find such an ordering, divide vertices of $v_0, \ldots, v_n$ into groups $G_0, \ldots, G_k$, where each group contains all vertices from a subtree of c, and there is a separate group for instances of c. Suppose that $v_0 \in G_0$.

We now have to find the ordering that starts with v_0 and does not have consecutive vertices from the same group. Additionally, the last vertex can not to belong to G_0.

We can do this greedily. Ignoring the start and finish constraints, the ordering exists unless there is a group G_i that is larger than all other groups combined. A slight extension of this condition can account for the start and finish.

To produce lex-min ordering, sort all vertices in each group by their string value. On each step, consider all groups such that using them now will still extend to a full ordering according to the criterion above. Out of the suitable groups, choose the one with lex-min unused vertex.

It is possible to solve the problem without using centroid with a very careful greedy traverse of the trie.