Division AB, Opening Contest: Western Subregional 2014

Discover World + Baikal 2020

September 9, 2020

A. Arithmetic Derivative

Keywords: number theory, factorization.

If we know the factorization of $n = p_1 \cdot \ldots \cdot p_k$, the arithmetic derivative n' can be found pretty easily. In fact $n' = n/p_1 + n/p_2 + \ldots + n/p_k$.

Factorization is easy to find in $O(\sqrt{n})$ time: try all $k = 2, \ldots, \lfloor\sqrt{n}\rfloor$ and divide n by k while possible (each new division corresponds to an extra prime divisor k). If at the end $n > 1$, then the final value of n is also a largest prime divisor.

In this problem $O(\sqrt{n})$ per test case is a bit too slow ($T \times \sqrt{\max n} \sim 10^9$). We can instead create a list of primes up to $\sqrt{10^9}$ and only use them in the above process. There are only ~ 3000 primes up to $\sqrt{10^9}$, and the new method is now fast enough.

Alternatively, if you have a fast factorization routine implemented (e.g. Pollard rho), just use that and you'll be fine.

B. Banner Rotation

Keywords: probabilities, dynamic programming, more dynamic programming.

It helps to divide the process into two levels.

On the top level, we keep track of how many different banners were shown until the end of each day. If Vasya have seen a different banners at the start of the day, let $P(a, b)$ be the probability that he will have seen b banners at the end of that day. Note that $P(a, b)$ can be positive only if $b \geqslant a$ (since we can't unsee banners).

Let $E(a)$ be the expected number of days until Vasya have seen all banners if he'd seen a banners before the first day starts. We have $E(n) = 0$ (by definition), and $E(a) = 1 + \sum_{b \geqslant a} P(a, b)E(b)$ for $a < n$ (by the law of total probability and linearity of expectation).

The last relation can be rewritten as $E(a) = \frac{1+\sum_{b>a} P(a,b)E(b)}{1-P(a,a)}$. This allows us to compute $E(n-1), E(n-2), \ldots, E(0)$ in this order in $O(n^2)$ time, provided we know all values of $P(a, b)$.

We now only need to compute all $P(a, b)$.

The number of shown banners is equidistributed in $[A, B]$. If we denote $P(a, b, k)$ to be the respective probability if exactly k banners are shown in a day, then $P(a, b) = \frac{\sum_{k=A}^{B} P(a,b,k)}{B-A+1}$.

Let us fix a. Assuming that Vasya had seen a different banners before the current day started, we will compute $DP_a(k, s, c)$ — the probability of arriving in such a situation that:

- since the start of the day a total k banners were shown to Vasya (not necessarily distinct, and possibly including some banners Vasya had seen before);
- up to this point, Vasya has seen s distinct banners;
- during this day, Vasya has clicked c banners (naturally, all of them were shown to Vasya, and all of them are distinct).

Initialize with $DP_a(0, a, 0) = 1$. Possible forward transitions from this state include:

- if $c = n$, then no banners will be shown. Let us just go to $DP_a(k + 1, s, c)$ with probability 1.
- Otherwise, with probability $\frac{s-c}{n-c}$ Vasya will be shown a banner he'd seen before. Then, with probability $P \cdot \frac{s-c}{n-c}$ we go to $DP_a(k + 1, s, c + 1)$, and with probability $(1 - P) \cdot \frac{s-c}{n-c}$ to $DP_a(k + 1, s, c)$.
- Similarly, with probability $\frac{n-s}{n-c}$ Vasya will be shown a new banner. With probability $P \cdot \frac{n-s}{n-c}$ we go to $DP_a(k + 1, s + 1, c + 1)$, and with probability $(1 - P) \cdot \frac{n-s}{n-c}$ to $DP_a(k + 1, s + 1, c)$.

This allows us to compute $DP_a(k, s, c)$ for all a, k, s, c in $O(n^4)$ time. Taking $P(a, b, k) = \sum_c DP_a(k, b, c)$, we reconstruct all $P(a, b)$ and plug them in the previous relations.

C. Code of the Tree

Keywords: tree diameter/centroid, subtree DP/DFS.

Solution 1. Root the tree at any vertex r. First, for all vertices v compute $down_v$ — the maximum length of a path from v down into its subtree.
We have $down_v = 1 + \max_{u:\text{ child of } v} down_u$, or $down_v = 0$ if v has no children. With subtree DP this can be computed in $O(n)$ for all v.
Now we have to compute up_v — the maximum length of a path from v that starts with a transition to its parent. We compute from top to bottom: suppose that we know up_v, and $down_u$ for all u — children of v. Then we have to update up_u with $1 + \max(up_v, \max_{u' \neq u} down_{u'})$, where u' ranges over all children of v.
$\max_{u' \neq u} down_{u'}$ is just equal to the largest $down_{u'}$ if $down_u$ is not the largest, otherwise it is equal to the second maximum. Thus, we need to find two largest values among $down_{u'}$ and we can find each up_u in $O(1)$ time. After this we proceed to do the same for all children of v, until all up_v are computed.
Finally, the longest path from any vertex v has length $\max(down_v, up_v)$. This solution has complexity $O(n)$.

Solution 2. Proposition. Let v, u be *diameter endpoints* — any pair of vertices at the largest possible distance in the tree. Then, for any vertex w either (w, v) or (w, u) is a longest path starting from w.
Proof. Assume the contrary, and there is a vertex z such that $dist(w, z)$ is larger than both $dist(w, v)$ and $dist(w, u)$. We also have $dist(v, z), dist(u, z) \leqslant dist(v, u)$ by the choice of v, u.
Let a be the vertex closest to z on the path from v to u. WLOG assume that a also lies on the path from w to u (if a instead lies on the path w to v, the analysis is symmetrical).
We have $dist(v, z) = dist(v, a) + dist(a, z)$, $dist(v, u) = dist(v, a) + dist(a, u)$, thus $dist(a, z) \leqslant dist(a, u)$. But then, $dist(w, z) = dist(w, a) + dist(a, z) \leqslant dist(w, a) + dist(a, u) = dist(w, u)$, a contradiction. □
It follows that we can locate any diameter endpoints v, u, compute all distances from v and u, and take the largest of the two for each other vertex w. With a similar argument as above, we can establish the following procedure for finding v, u:

- w — any vertex;
- v — any vertex at maximum distance from w;
- u — any vertex at maximum distance from v.

Note that distances from any vertex w to all others can be found with a single DFS of the tree in $O(n)$ time. This solution performs $O(1)$ DFS runs, thus the complexity is $O(n)$.

D. Decoding Tree Code

Keywords: tree diameter.

Let $D = \max d(v)$ — the diameter length. Let $v_0, \ldots, v_D$ be the vertices along any diameter. We know that $d(v_i) = \max(i, D - i)$. Choose distinct $v_0, \ldots, v_D$ subject to the constraints on $d(v_i)$; if we cannot do this, there can be no answer.
Any other vertex w must have $d(w) > \lceil D/2 \rceil$. If there is such a vertex left unused, there is no answer as well.
Otherwise, we can connect each other vertex w with $v_{d(w)-1}$, and the longest path from w will have length $d(w)$ and end in v_0.

E. Election

Keywords: math, bounds.

If $a \geqslant 0$ and $round(a) = x$, then $x - 1/2 \leqslant a < x + 1/2$ if $x \neq 0$, and $0 \leqslant a < 1/2$ if $x = 0$.
Summing lower and upper bounds for all percentages, we must have $\sum L_i \leqslant 100 < \sum R_i$. If the inequalities hold, then suitable values summing up to 100 can be chosen in each range.
For simplicity we can multiply the bounds by 2 to avoid dealing with fractions.

F. Factorial Number System

Keywords: parsing, numeric systems.

First, parse the expression into numbers and operands. This can be done with an LL-parser, or simply by splitting the string at all occurences of `+` and `-`.
Then convert each each number token into an actual number, and compute the expression. Finally, convert the result back into factorial system.
Note that each number fits in 32 bits, but intermediate results may not. Use 64 bits for safety.

G. Graphs. How Many of Them?

Keywords: combinatorics, DP.

Generalized Cayley's theorem. Let G be a graph with n vertices such that its connected components have sizes $c_1, \ldots, c_k$. Then the number of ways to add $k - 1$ edges to make the graph connected is equal to $n^{k-2} \prod_i c_i$.

If n is the number of vertices, denote:

- $G(n)$ — the number of all graphs;
- $C(n)$ — the number of connected graphs;
- $P(n, k)$ — the sum of $\prod_i c_i$ (see above) over all graphs with k connected components and no bridges;
- $B(n, k)$ — the number of connected graphs with k bridges.

We have that:

1) $G(n) = 2^{n(n-1)/2}$;
2) $C(n) = \sum_{k=0}^{n-1} B(n, k)$;
3) $G(n) = \sum_{m=1}^{n} \binom{n-1}{m-1} C(m) G(n-m)$ (split all graphs by the size m of the connected component containing the first vertex);
4) for $k > 1$ $P(n, k) = \sum_{m=1}^{n-1} \binom{n-1}{m-1} P(n-m, k-1) P(m, 1)$ (same as above, note that $m < n$ since the graphs are not connected);
5) $B(n, k) = n^{k-1} P(n, k+1)$ (generalized Cayley's thm);

With these relations we can compute all the values by increasing of n:

- Express $C(n)$ from 3) using $C(0), \ldots, C(n-1), G(0), \ldots, G(n)$.
- Find $P(n, 2), \ldots, P(n, n)$ from 4) using $P(0, k), \ldots, P(n-1, k)$.
- Find $B(n, 1), \ldots, B(n, n-1)$ from 5) using $P(n, 2), \ldots, P(n, n)$.
- Find $B(n, 0)$ from 2) using $C(n), B(n, 1), \ldots, B(n, n-1)$.
- Find $P(n, 1)$ from 5) using $B(n, 0)$.

This leads to an $O(n^3)$ DP solution.

H. Halloween

Keywords: harmonic series.

Let $M = \max a_i, b_i$, and let c_x, d_x be the number of occurences of x among a_i, b_i respectively.
We can now find the answer as

$$\sum_{x=1}^{M} \sum_{y=1}^{\lfloor M/x \rfloor} c_x \cdot d_{x \cdot y} \cdot y.$$

The compexity is bounded by $\sum_{x=1}^{M} M/x = O(M \log M)$.

I. Intelligent System

Keywords: greedy.

For each of the n stages find a_i, b_i — two fastest robots available on the stage i (if there is only one robot on a stage, put $b_i = \infty$).
If we double operating time for a robot on the stage i, we have to double the lowest time (i.e. a_i). The actual time then will be equal to $\min(2a_i, b_i)$.
If we don't double anyone on the stage i, the time is just a_i.
If the stage i is the one we double someone on, the answer is $(\sum_j a_j) + \Delta_i$, where $\Delta_i = \min(2a_i, b_i) - a_i$.
To maximize the answer, choose i so that Δ_i is largest.

J. Joining Transformation

Keywords: greedy, data structures.

If a subsequence of s is lexicographically largest, it must begin with all occurences of `z` in s.
Similarly, after taking all `z`'s, we must take all `y`'s **to the right** of the rightmost `z` (or just anywhere is there were no `z`'s). After considering each letter `z`, `y`, `x`, ..., `a` in this way, we obtain the answer.
Note that even if the resulting subsequence is very long, we can describe it simply by listing multiplicities of each letter.
To find the multiplicities we'll need a data structure that, given a range $[l, r]$ in the string s produces the number of occurences of a letter c, as well the rightmost occurence of c (if there are any).
This is easily done with a separate segment tree for each letter.
The complexity is $O(\alpha \log n)$ ($\alpha = 26$) for a find query, and $O(\log n)$ for an update query.

K. King of Byteland

Keywords: data structures, centroid decomposition.

Centroid decomposition of a tree T with n vertices is constructed as follows:

- locate a *centroid* vertex c such that removing c leaves several connected components, each with at most $n/2$ vertices;
- remove c, and construct centroid decompositions for each of the resulting connected components recursively.

Naturally, the resulting trees on all stages are nested, and can be divided into layers of nestedness. Since the size of the largest tree is halved on each subsequent layer, there will be $O(\log n)$ layers in total.

Consider any query: find the k-th vertex among vertices at distance d from a vertex v.
Consider the top layer of the decomposition (before any vertices are removed), and let $T_1, \ldots, T_s$ be the subtrees after removing the centroid c. Suppose that $v \in T_i$. Before we solve the original problem, let us find the k-th vertex at distance d from v, excluding all vertices of T_i.
We can instead search for vertices at distance $d' = d - dist(v, c)$ from the centroid c. On the construction stage, let us compute sets of vertices $L_{x,j}$ in the subtree T_j at distance x from c, for all relevant j and x.
If we didn't have to exclude T_i, we could simply precompute $U_x = \cup_j L_{x,j}$ for each x, and output the k-th element of $U_{d'}$. This can be done fast enough if U_x are stored in segment trees/treaps. Note that sets U_x are enough to answer the query if $v = c$.
To exclude T_i, let us instead compute sets $P_{x,k}$ and $S_{x,k}$ — the union of the first/last k sets $L_{x,j}$ for all $k = 0, \ldots, s$. Then, the answer to the query is the k-th element of $P_{d',i-1} \cup S_{d',s-i}$. Note that at this point $P_{x,k}$ and $S_{x,k}$ need to be persistent segment trees/treaps since their total size can be large.
Instead of finding the union explicitly, we can binary search for the answer/parallel descent in segment trees representing $P_{d',i-1}$ and $S_{d',s-i}$. This is possible to do in $O(\log n)$ time per query.

To answer the original query, we have to introduce vertices of T_i. This is done exactly the same since T_i is a node in the decomposition, and we can precompute all necessary information there in the same way.
We have to do this recursively, going down the decomposition layers, until v is the centroid of the current tree. Now the answer to the query can be found as the k-th element of the union of all $P_{\ldots,\ldots}$ and $S_{\ldots,\ldots}$ produced from all visited layers. The number of sets in the union is clearly $O(\log n)$, since at most two are produced from each layer.
The k-th element can be found just as well with binary search/parallel descent, and $O(\log^2 n)$ per query is possible (yet $O(\log^3 n)$ is probably enough to get AC).

L. Linear Programmer

Keywords: shortest path.

Put $x_1 = 0$, and maximize x_n.
Create a directed weighted graph with n vertices. For each inequality $x_v - x_u \leqslant l$, create an arc from u to v of length l.
Put x_v to be the shortest path length from 1 to v. If v is unreachable from 1, put $d_v = \infty$.
This assignment of x_v satisfies all inequalities (otherwise, one of the shortest paths can be shortened further).
Further, the assigned value of x_v is largest possible among all solutions. Indeed, consider a shortest path $1 = u_0, u_1, \ldots, u_k = v$. For each arc in the path, we have an inequality $x_{u_{i+1}} - x_{u_i} \leqslant l_i$. Adding all these inequalities together, we obtain $x_v - x_1 \leqslant \sum l_i = L$ — the shortest path length.
All arc lengths in the graph are positive, thus we can use Dijkstra's algorithm for finding shortest paths.

M. Masha and Dasha

Keywords: geometry, binary search.

Let us introduce a coordinate system on the border of the triangle: $p(d)$ is the point at distance d from the vertex A when travelling counterclockwise around the border. Note that $p(0) = p(L)$, where L is the perimeter length.
Define the function $f(x)$ as follows. Cut the triangle with a straight directed segment from $p(x)$ to $p(x + L/2)$, let S_1 and S_2 be the areas of the parts to the left and to the right of the segment respectively. Then $f(x) = S_1 - S_2$.
Computing $f(x)$ can be done in $O(1)$ with rudimentary geometrical primitives and elbow grease.
We can observe that:

- $f(x)$ is continuous;
- $f(0) = -f(L/2)$, since the cut is the same in both cases, and only the order of the parts is swapped.

This implies that a point $x_0 \in [0, L/2]$ such that $f(x_0) = 0$ always exists, and can be found with binary search.
The cut through $p(x_0)$ and $p(x_0 + L/2)$ divides the cake into two halves of equal border length $L/2$ (by definition of coordinate system), and equal area (by definition of f and choice of x_0).