The 4th ProgNova Programming Contest

Editorial

Nov 12, 2018

A Long Swaps

It can be observed that via a sequence of swaps, we can swap any pair of letters from the first and last k positions. There is no way to change any of the middle $n-2k$ letters. When $n-2k \leq 0$, the answer is always possible. When $n-2k > 0$, we can sort the first and last k letters together, and write them back to the first and last k positions. Then check if the resulting letters appear sorted. The running time is $O(|s| \log |s|)$.

B Bracket Matrix

We iterate the columns from left to right, and arrange the brackets in each column. When we are at column j, let the number of unmatched left brackets in row i be $open_i$. We want to greedily place right brackets to the rows with larger $open_i$, and place left brackets to the rows with smaller $open_i$. Suppose the current column has k left brackets and $n-k$ right brackets. We may sort the rows based on their $open_i$, and assign the k left brackets to the first k rows, and the $n-k$ right brackets to the last $n-k$ rows. At any point, if some $open_i$ drops below zero, the answer is `no`. At the end, all $open_i$ should be exactly zero for the answer to be `yes`. The time complexity is $O(n^2 \log n)$.

Additionally, the values of $open_i$ may differ by at most one in any step, i.e. the values can be represented by v and $v+1$ for some integer v. We can optimize the $\log n$ factor by keeping track of how many rows have $open_i = v$ and how many rows have $open_i = v+1$. This makes our solution $O(n^2)$.

C XOR Equation

As there are at most 10 question marks in the expression, there are at most 10^{10} ways to fill the digits. Enumerating all 10^{10} possibilities would cost too much time. We observe that once we have finished enumerating the digits for two of the numbers, the third number can be immediately determined uniquely. Therefore we can rewrite the equation into a' xor $b' = c'$, where a' is the number with the fewest question marks and b' is the number of the second fewest question marks. Then we only enumerate the choices of digits for a' and b'. There are at most 6 question marks in a' and b', so we only have 10^6 possibilities.

D Auto Completion

Build a Trie tree for all dictionary words, and preprocess the number of words inside each subtree of the Trie tree. For each query we walk down the Trie tree starting from the root.

If we see a letter, just follow the edge to the corresponding child. If we see a group of hash signs, count the number of hash signs (let it be k), and find the k-th smallest word inside the current subtree. We can base on the preprocessed number of words in each children to decide which child to go to. If the current subtree only has $k_s < k$ words, then we are finding the $((k-1) \ mod \ k_s + 1)$-th smallest word within the subtree. We would at most spend $O(26 \cdot L)$ time walking down the Trie, where L is the total length of the text produced by all keystroke sequences. The input guarantees that $L \leq 10^6$.

E Food Carts

Sort the food carts ranges increasingly based on their left endpoints. We iterate each starting index l, and use a pointer to keep track of the minimum ending index r, so that the sum of $p[l..r] \geq k$. Every food cart we choose must include at least $p[l..r]$ so that the intersection of all ranges would have at least k passengers. We can use a data structure (priority queue or BBST) to maintain the right endpoints of all food cart ranges that start before or at l. Using the data structure we can determine for each l the number of food cart ranges s_l that contain $p[l..r]$ and start strictly before l. We can also determine the number of food cart ranges s'_l that contain $p[l..r]$ and start right at l. We have $2^{s_l} \cdot (2^{s'_l} - 1)$ ways to choose from those food carts. The total answer is $\sum_l 2^{s_l} \cdot (2^{s'_l} - 1)$. The time complexity is $O((n+m)\log(n+m))$.

F Triplet-Free Queens

Use complete search to place the queens. Keep track of the number of queens in each row, column, diagonal, and anti-diagonal. These numbers can be updated in constant time after placing each queen. We can choose to place zero, one, or two queens in each row. The total time complexity is $O(C(m,2)^n)$, which is a bit too big when $n = 5, m = 10$. Yet we can rotate the chessboard so that $m \geq n$. The worst case then happens when $n = m = 7$ and $C(7,2)^7 \approx 1.8 \times 10^9$ which is affordable. As we only care about the maximum number of queens and their placement, we may prune the search when it is impossible to reach a maximum number of queens.

G Manhattan Shopping

If we rotate the 2D plane by 45 degrees clockwise, then Peter may go to a point j from point i using one horizontal move if j is in the top-right quadrant of i. We say point i and point j belong to a same component in this case. We can sort the points after rotation by increasing x and then increasing y, and then figure out all the components using one linear scan with a stack. The stack keeps track of the lowest y coordinate of each component, and stores a sequence of components that have decreasing lowest y's. Finding the component takes $O(n \log n)$ time.

Each component offers a subset of the m items. The remaining task is to choose a minimum number of components so that their union has the full set of items. This can be solved using dynamic programming. First, in $O(2^m)$ process which subset of items is available in some component. For each subset, we enumerate all subsets of it to take a DP transition. The total time complexity of finding the minimum number of components is

$O(3^m)$. When Peter's initial component has m items already, the answer is zero. Otherwise, the answer is the minimum number of components to take plus one.

H Magical Crystals

Firstly, we convert the input tree T to a new binary tree B such that each leaf of B is a vertex of T, and the distance between the two leaves in B corresponds to the congestion factor between their corresponding vertices in T. The transformation is as follows:

- At the beginning, each vertex in B is a single vertex from T. B has no edges.
- Each time we find an edge $e = (x, y, w)$ with the smallest w from T. Contract two endpoints x and y of e. Merge the two subtrees b_1 and b_2 in B (b_1 contains x, b_2 contains y). Create a new vertex z and connect z to b_1 and b_2. Set the height of the new tree rooted at z to be the weight w of the contracted edge. That is, determine the weight of the edge between b_1 and z based on the height of the subtree rooted at b_1 and w. Do the same for the edge between b_2 and z.

The transformation takes $O(n \log n)$ time.

After the transformation, consider the min-cost matching in B, where the cost of matching two vertices is the sum of weights on the path between these two vertices. Each query corresponding to find a min-cost matching. Notice that, if the difference between the demand and the supply in a subtree is s, and the weight of the edge between the root of this subtree and the parent of the root is w, then the contribution of this edge to the cost of the min-cost matching is $w \cdot s$. For each query, we can first sort all the query vertices in its DFS order. Then we can compute all the LCAs of every two adjacent vertices in DFS order. Construct a small tree (linear in query size) that only contains all the query vertices and the corresponding LCAs. Finally count the difference between demand and supply in each subtree of the small tree of this query. Answering a query with n_i vertices takes $O(n_i)$ time.