Problem A. Adding IQs

Topics: Dynamic Programming

Since the range of the IQs is small, you can do an inclusion-exclusion dp maintaining the amount of subsets that have specific sums, as such the complexity of the problem becomes n^3.

Complexity: $\mathcal{O}(n^3)$

Problem B. Braga's Problem

Topics: Math

Use the sieve of Eratosthenes to figure out the primes and use prefix sum to maintain the sum of primes in a range.

Complexity: $\mathcal{O}(n \log \log n)$

Problem C. Circulating Misinformation

Topics: sortings

Add all strings from the original list into a set, and then as you get the strings from the altered list, remove the ones present in the original list. Then simply print out the modified original list.

Complexity: $\mathcal{O}(n \log n)$

Problem D. Do The g, Man

Topics: BFS/DFS

You can search for cycles of length 4 the following way: for each node u, you can check its adjacency in pairs two-by-two, checking if some other node is also connected to that same pair. Once found, you simply check if it's possible for you to have the leg of length two. It seems like you need n^3 to check all pair-node-node combinations, but if such an answer exists, you can see that you'll only have to do up to n^2 operations. Now we need to figure out how to find out whether there is a tail of size two. For that matter, for each node you need to find out where paths of length two lead; you can do this in n^2 also. It works sort of like this: for each pair of nodes (i, j), where there's an edge between them, you find out from j where it can lead to without returning to i. The catch here is that you don't necessarily need to look at every path, you can simply store at most 3 paths here. And in addition to this, from every path you found starting with $i- > j$, you store it in another array indicating the nodes that can be of distance 2 away from i. In this other array, you can choose to only store up to 10 separate nodes inside (not necessarily distinct). Let's suppose we find a cycle of length 4 consisting of (i, j, k, l), and that the tail will be coming out of node i. Then, let's check the possible length 2 paths with the starts $(i, j), (i, k), (i, l)$, and remove them from the array of distance 2. If there's still a node present there, then for sure you can have a tail that doesn't intersect j, k or l.

Complexity: $\mathcal{O}(n^2)$

Problem E. Escaping Cokeman

Topics: Graphs, Shortest Paths

The minimum time to reach the exit E will be either the time to reach E evading both walls and the marked path, or the time to reach the badge B evading both walls and marked path + the time to reach E from B evading only the walls. To achieve that, let's perform two kinds of BFS, BFS_1 evades the marked path and BFS_2 evades only the walls. Thus the answer will be $min(BFS_1(S, E), BFS_1(S, B) + BFS_2(B, E))$.

Complexity: $\mathcal{O}(N * M)$

Problem F. Fraud Detection

Topics: ad-hoc

Just count how many times each digit appears as the leading one.

Complexity: $\mathcal{O}(n)$

Problem G. Gruesome Polynomials

Topics: binary search

You can binary search the root within each range of size 0.5 inside of $[-30, 30]$.

Complexity: $\mathcal{O}(n \log n)$, where $n = 0.5/10^{-6}$

Problem H. Heaviside Step Function

Topics: Ad-hoc

Simply check if the number given is negative or not; since it exceeds the size of an integer, read it as string.

Complexity: $\mathcal{O}(1)$

Problem I. IME1000 project

Topics: Shortest Paths, Graphs, Dynamic Programming

Let's store a $dist[N][K]$ table with $dist[u][r]$ being the cost to reach node u starting from 1 and ending with a reputation of r. Now to minimize the cost to reach all nodes with every reputation from 0 to K, we run Dijkstra's algorithm starting from node 1 and reputation 0. Here, a reputation greater than K is equivalent to having a reputation of K, so we can consider those equal. So, traversing through some edge either increases its reputation by 1 until the maximum of K or resets to 0 if its already K. Therefore, the transition from some node u and reputation r to some adjacent node v with w_i the cost of the edge (u, v) is given by one of these conditions: $dist[u][r] + w > dist[v][max(K, r+1)]$ or $r = k$ and $dist[u][r] > dist[u][0]$. After that, we only need to maximize $v[i] - dist[i][K]$, the profit minus the minimum cost with reputation K, ignoring the negative values.

Complexity : $\mathcal{O}((N + M) \cdot K \cdot log(N \cdot K))$

Problem J. Johannes Loves Games

Topics: Segment tree

Turns out that just making a segment tree with a custom node is the solution to the problem. The custom node will contain the maximum left sum, right sum, full sum, and current value for that. As such, you must figure out a way to merge the nodes when iterating the segment tree, which is left to the reader as too much information has been given out already :)

Complexity: $\mathcal{O}(n \log n + q \times \log n)$

Problem K. Kowtowing Our Leader

Topics: binary search

You can binary search the maximum happiness product necessary for the amount of pairs whose product is greater than or equal to that which is greater than or equal to m. This can be found by sorting the initial array in decreasing order and then finding out the furthest element, such that $a_i \times a_j >= x$, where x is the maximum happiness product.

Complexity: $\mathcal{O}(n \log^2 n)$

Problem L. Legendary Duty Scheduler

Topics: Simulation, STL, Schedules

To determine the student responsible for duty on each of the T days, we simulate the process by maintaining an ordered set of the active students, each of them is included in the set at the beginning. Each query can be stored in an array indexed by its corresponding day as either an insert or removal query, so they can be processed offline later. At each step of the simulation you need to keep track of the current element in the set and the day we are currently on. If there are some operations for the current day, process the inserting ones first and then the removals. To achieve the next step of the simulation, increase the time and get the next element of the ordered set. If the set is empty, no one is pulling duty on that day.

Complexity: Each insertion/removal/iteration takes $\mathcal{O}(\log N)$, and we do it at most $\mathcal{O}(T + Q)$ times, so the total complexity is: $\mathcal{O}((T + Q) \log N)$.