Kick Start 2022 - Round H

Analysis: Electricity

Test Set 1

For this test set, we can simulate providing electricity to each junction, one junction at a time. For each of these $\mathbf{N}$ simulations, we compute how many junctions will end up receiving electricity after the initial junction is powered, and our answer is the maximum of these $\mathbf{N}$ simulation results. Each simulation can be accompolished with depth-first search (DFS); we start our DFS at the junction we are directly providing power to. In each step of our DFS, we iterate over the current junction's neighbors, and recurse into the neighbor if its capacity is strictly less than the current junction's capacity. Because the capacity must be strictly decreasing, we are guaranteed to never go in a cycle or visit a junction more than once. The DFS time complexity is therefore bounded by the total number of edges, making it take $O(\mathbf{N})$ time; with $\mathbf{N}$ simulations, this gives us an $O(\mathbf{N}^2)$ time solution, which is sufficient for this test set.

Test Set 2

We can improve our DFS-based solution from test set 1 to be fast enough for this test set by removing the redundancy among the $\mathbf{N}$ DFS invocations and consolidating the computation into an $O(\mathbf{N})$ solution. To give an example of the computational redundancy from our test set 1 solution that we aim to remove, suppose v_1 and v_2 are directly connected junctions, with v_1 having the greater capacity. Simulating providing electricity to v_1 will also lead to simulating providing electricity to v_2 due to the edge between them, but we will also simulate directly providing electricity to v_2, duplicating our work. We can remove this duplication by treating this as a dynamic programming problem and memoizing our work.

Our solution works similarly to before, where for each junction v we perform a DFS to compute how many junctions would be powered if we provided electricity to v. However, we memoize this DFS; when the DFS is called on a junction v, we first check if v is in our memoization table and return the result if it's present. Otherwise, we recurse and perform DFS as usual, but store the result in the memoization table before returning the result. This memoization implies that among our $\mathbf{N}$ DFS calls, an edge will never be traversed twice, because after the first traversal, the result will be memoized, making further traversals unnecessary. This implies our $\mathbf{N}$ DFS calls will take in total $O(\mathbf{N})$ time, making this approach sufficiently fast for the large test set.