Kick Start 2021 - Round B

Analysis: Truck Delivery

Test Set 1

Let us root the tree at the capital city 1. For the small test set, we can answer each query by simply iterating the path from the given city $\mathbf{C_j}$ up to the capital city. We start out with the answer $ans = 0$, and whenever we encounter a road on the path with $\mathbf{L_i} \le \mathbf{W_j}$, we update the answer to $ans = \gcd(ans, \mathbf{A_i})$.

The time complexity of the Greatest Common Divisor (GCD) operation $\gcd(a, b)$ is $O(\log(\min(a, b)))$ or $O(\log(MaxA))$ in our case, where $MaxA$ is the largest toll among all roads. A short proof of GCD time complexity is provided here, and it can be generalized to show that the amortized time complexity of a sequence of K GCD operations, where the result of a previous operation is fed into the next one, is $O(K + \log(MaxA))$ as opposed to $O(K \times \log(MaxA))$. Since a path in the tree can have up to $\mathbf{N}$ cities, the overall time complexity of the algorithm for all $\mathbf{Q}$ days is therefore $O(\mathbf{Q} \times (\mathbf{N} + \log(MaxA)))$.

Test Set 2

Since all queries are known in advance, we do not have to answer them in the given order, so let's group the queries by city $\mathbf{C_j}$.

Let's look at how we can answer all queries for a particular city C efficiently. First, we need to build a list of roads on the path from city C to the capital city 1 and sort them by the load-limits $\mathbf{L_i}$ in an increasing order. Let's also sort the queries for city C by weight $\mathbf{W_j}$ in a non-decreasing order. Now we can answer the queries by iterating these two lists in parallel and calculating GCD of all roads with load-limit up to and including the weight $\mathbf{W_j}$ of the current query.

The time complexity of this approach is $O(\mathbf{N}^2 \log(\mathbf{N}) + \mathbf{N} \log(MaxA) + \mathbf{Q} \log(\mathbf{Q}))$ as we need to sort the list of roads from each city to the capital city 1, perform a series of GCD operations for each of these $\mathbf{N}$ paths, and also have the queries sorted by loads.

Rather than building paths to the capital for each city independently, we can perform a Depth-first search (DFS) of the tree starting at the capital city 1 and answer all queries of a city C as we visit the city for the first time. That way, the cities and roads on the path from C to 1 are conveniently stored in the DFS stack. All we need is an efficient data structure that would store the toll $\mathbf{A_i}$ of precisely these roads and support GCD queries of tolls in the load-limit range $[1, \mathbf{W_j}]$.

That data structure happens to be a segment tree ST with load-limits $\mathbf{L_i}$ as keys (recall that all load-limits are unique), the tolls $\mathbf{A_i}$ as values, and GCD as the merge operation. Initially, the segment tree ST is empty, namely, the values of all its nodes are 0. Whenever we traverse the i -th road, we perform a point update operation $ST.update(\mathbf{L_i}, \mathbf{A_i})$, and, when we backtrack along this road in the DFS traveral, we cancel the value $\mathbf{A_i}$ by calling $ST.update(\mathbf{L_i}, 0)$. By doing so, we ensure that at the time of answering queries for a particular city, the segment tree ST contains the tolls of precisely the roads on the path to the capital city 1, and the answer to a query is $ST.query(1, \mathbf{W_j})$.

Let $MaxQ$ be the maximum load-limit among all roads. Each update or query of the segment tree involves $O(\log(MaxQ))$ GCD operations so the amortized time complexity of a single update or query operation is $O(\log(MaxA) + \log(MaxQ))$. Since we have two update operations per road and one query operation per each day, the overall time complexity of the algorithm is $O((\mathbf{N} + \mathbf{Q}) \times (\log(MaxA) + \log(MaxQ)))$.