Code Jam 2020 - Virtual World Finals

Analysis: Pack the Slopes

Test Set 1

A translation of the problem statement is that we wish to find the minimum cost maximum flow on a directed tree. The rest points can be represented by nodes, and the ski slopes can be represented by edges. A commonly used algorithm to solve minimum cost maximum flow problems is successive shortest paths. In the case of our directed tree, a simplified successive shortest paths approach can be applied.

Observe that we can add skiers greedily. Of all the nodes that can be reached from the top of the mountain (the root of the tree), we should always send a skier to a node with a minimum cost path using only edges that still have capacity remaining. Since we are working with a tree, there is only one path from the root to each node. With this in mind, we can find the path costs to all the nodes using any tree traversal from the root while calculating the cost to each node as the cost of its parent (which we calculated first) plus the cost of the edge connecting them. Then, we can sort the nodes by their path cost in ascending order. We should always use the first node in the list that still has some capacity along its path.

Let us maintain the number of skiers sent through each edge. We can work through our sorted list of nodes in order. We check if the current node in the list still has some capacity by checking the edges in the path, and either update the capacities if we can push another skier, or move on to the next node in the list if we cannot. Doing this for a single skier takes O(**N**) time. We know we will keep using the same path until some edge reaches capacity, so we can compute the minimum capacity left in the path with a linear pass over it, and simulate sending that many skiers at once. Each such a step takes O(**N**) time overall, and since we have to try up to O(**N**) possible destinations, the overall algorithm takes O($\mathbf{N}^2$) time, which is efficient enough to solve Test Set 1.

Test Set 2

For the large input, we need to do more. We can use a heavy light decomposition of the tree to manage the remaining capacities of the edges in the tree. The decomposition will need to support updates and queries on paths from the root to an internal node. The updates involve subtracting an amount from the capacity of each edge. The queries involve finding the minimum capacity of all the edges on a path.

A heavy light decomposition breaks a tree up into a collection of disjoint paths such that the number of decomposition paths from any node to the root is at most O(log **N**). If we keep a segment tree for each decomposition path, then we can support both the query and update operations in O($\log^2$ **N**). Note that the segment tree we use must support lazy range updates. This improves the running time of each step of the algorithm for Test Set 2 to take only O($\log^2$ **N**) time instead of O(**N**) time. This makes the algorithm run in O(**N** $\log^2$ **N**) time in total, which is sufficient to solve the Test Set 2.