Stranded Far From Home (island)

BY TÄHVEND UUSTALU AND TAAVET KALDA (ESTONIA)

In this task, we are given a graph with weighted nodes—the nodes are the villages, the weights are the numbers of inhabitants, the edges are the roads between the villages. We say that the convincing node eats the convinced one.

Subtask 1. $N \le 2\,000$ and $M \le 2\,000$.

In this subtask, we have enough time to simulate the process. For each node, we check whether its tie color could win. This can be done by subsequently eating the neighboring nodes that are small enough—we add all nodes adjacent to one that we already ate into a priority queue and repeatedly eat the smallest entry of the priority queue, if this is small enough. If this is not possible, we cannot win.

Subtask 2. $s_1 \ge \ldots \ge s_N$, and every village b with $b > 1$ is directly connected to exactly one village a with $a < b$.

In this subtask, the graph is a tree rooted at the greatest node. The node weights are descending from the root to the leaves. A node can always eat its whole subtree. We can solve this subtask using dynamic programming: first, we compute the weight sum for the subtree of each node. This can be done in linear time by recursively computing the sum for the children, and then adding their sum and the weight of the current node to get the weight for the whole subtree. Then, we traverse the tree again from top to bottom. The root can always win, any other node can win iff the sum of its subtree is at least as big as its parent and the parent can win.

Subtask 3. Villages a and b are directly connected if and only if $|a - b| = 1$.

In this subtask, the graph is a line. The greatest node can always win. It 'splits' the line into two halves—if the sum of one half is smaller than the greatest node, none of the nodes in this half can win. Otherwise, the greatest node of this half can win, and we can recursively proceed with the computation. Thus, we can solve this subtask by using data structures for range minimum and range sum, for example a segment tree and prefix sums.

Subtask 4. There are at most 10 distinct numbers of inhabitants.

Here, there are only at most ten distinct node weights. We create a new graph and first add all nodes with minimal weight. In every component of this graph, every node can eat every other node. Then, we do a depth first search to compute the sum for each component of this graph. All components which are too small to be able to eat a node with the next largest node weight cannot win—thus, we mark all such nodes. Next, we proceed by adding the nodes with the next largest node weight to the new graph, again compute the sum for each component (as the new nodes can now eat all other nodes in their component), and proceed like this.

Subtask 5. No further constraints.

For the full solution, we follow a similar idea as in Subtask 4. However, with more possible node weights, we can not do a depth first search every time, but need some data structure to efficiently manage the current components and mark the nodes that can not win anymore. This can be done for example with a slightly modified union find structure—we mark the representative of a component if this component can not win anymore. However, now an adaption to the path compression optimization is necessary such that it does not destroy those markings: When searching for the representative in the union find structure, we remember whether we passed a marked node, and if so, we also mark the current node. Afterwards, we can safely do the path compression.

There are several other possibilities for managing the set of nodes that can not win anymore. For example, we could, instead of modifying the union find structure, additionally hold node sets for the components and, when merging, merge the smaller one into the bigger one. When marking a component as 'cannot win,' we go through all the nodes in its set. This is fast enough, as every node is marked only logarithmically often (every time it is marked, the total sum of the component at least doubles).