Problem C. Angry Cows

Subtask 1

We have $n \leq 10$, and so we can afford to try every possible set of walls. For each such set, we can check the (non-)connectivity requirements using a BFS or DFS algorithm: starting from some hiking areas, and making sure not to walk on walls, we check whether we have visited all hiking areas but no cow-areas. We also need to find the remoteness of each area (which can be done using Dijkstra's algorithm) and the overall remoteness of every possible wall set, picking the one with minimum remoteness.

Subtask 2

In this subtask we will make what is possibly the most important observation for the entire task. Namely, the intuition is that, rather than surrounding the hiking areas, it is better to surround the cow areas, as this will give us the highest chance of keeping the hiking areas connected to each other. In general, this may not result in the smallest possible remoteness, but in this subtask we do not care, as everything has remoteness 0.

So: we just place walls on all unused areas that neighbor a cow-area! This wall set, which we call W, will surely separate hikers from cows (unless there is a cow-area adjacent to a hiking area, in which case we should always print `-1`). However, we still need to verify that we have not disconnected some hiking areas. As in subtask 1, we can do this by running a BFS or DFS traversal from some hiking area, avoiding W, and then check whether all hiking areas have been visited. If everything is fine, we can print W. On the other hand, if some hiking areas are disconnected, then W is infeasible. However, in this case in fact no set of walls is feasible, and we can print `-1`! Why is that?

Assume there exists some feasible set W' of walls. Then for any two hiking areas there is some path connecting them that avoids W'. Crucially, this path cannot use any area in W either, because if it did, this would create a W'-avoiding path from either of these hiking areas to a cow-area (that the W-area used by the path is adjacent to). This shows that if there is some feasible set W' of walls, then there are W-avoiding paths between any two hiking areas, which makes W feasible as well.

Subtask 3

One *incorrect* solution that suggests itself here is to place walls in all neighboring areas of the single hiking area h. However, there are two issues with this approach:

- Some of these walls might be unnecessary (since no cow-area can be reached from them without passing through h). Including them might unnecessarily increase the overall remoteness.
- Even taking the above into account, this set of walls might still not have minimal remoteness. This is because there can exist two unused areas such that the first is a neighbor of h whereas the second is not (so the first is closer to h if we look at unweighted distances), but the second is closer in terms of remoteness (weighted distances from h). If walling either of these areas would separate h from cow-areas, then we should select the second rather than the first.

In order to solve this subtask, we need to make the second crucial observation in the task. Namely, we can perform a binary search over the overall remoteness of the solution, which will reduce the task to checking, for a given r, whether there exists a wall set with remoteness at most r.

To do this, in this subtask it is enough to check whether placing walls in all unused areas of remoteness at most r disconnects the hiking area from all cow-areas. Clearly, if this happens for some r, then it also happens for all larger r; in this subtask we are not worried about disconnecting hiking areas from each other. To find the remotenesses of all areas, we run Dijkstra's algorithm.

Subtask 4

Let us root the tree at some hiking area h. We can use dynamic programming (a DFS traversal of the tree) to compute, for each v, a binary value $h_v \in \{0, 1\}$ that is equal to 1 if the subtree rooted at v contains some hiking area. This can be done in linear time. (For hiking areas v we set $h_v = 1$; for other areas, if they are leaves, then $h_v = 0$, and otherwise h_v is the binary disjunction (OR) of all h_w for w

being children of v.)

If there is a cow-area v with $h_v = 1$, then there is no good wall set, as this cow-area lies on the (unique) path between h and some hiking area. Also, as usual, if a cow-area is adjacent to a hiking area, then there is no solution. Otherwise, the set S of areas v with $h_v = 1$ is a connected set, containing the root h, that must be kept connected (i.e. we cannot place any walls there). We should place walls along the outside boundary of S: these are unused areas and we will thus disconnect S (and all hiking areas) from cow-areas.

However, we should not place walls on *all* boundary areas of S, as some of them might be unnecessary (they do not actually separate S from any cow-area). To know which ones to select, we can e.g. compute values $c_v \in \{0, 1\}$ for every area v: $c_v = 1$ if the subtree rooted at v contains some *cow*-area. This is done in exactly the same way as for h_v. Then, we would only place walls on boundary areas of S that have $c_v = 1$.

Subtask 5

Here we need a fully general solution as for Subtask 6 (see below), but it may check all remotenesses iteratively (instead of running a binary search), or its implementation may be inefficient (e.g. Dijkstra or BFS/DFS not running in nearly-linear time).

Subtask 6

We use the binary search idea introduced for Subtask 3. This reduces the problem to determining, for a given r, whether there is a wall set of remoteness at most r. A wall set will have remoteness at most r if and only if it consists of areas with remoteness at most r. Therefore, once we have computed all remotenesses using Dijkstra's algorithm, we can divide all areas into: cow-areas, hiking areas, allowed unused areas, and forbidden unused areas. The task is to find any wall set that only uses allowed unused areas.

Now that we no longer care about remoteness, the remaining task looks very similar to Subtask 2 – with the difference that now there are forbidden areas where we cannot place walls. This prevents us from just walling in the cows (as in the solution to Subtask 2).

However, notice that a forbidden area that is adjacent to a cow-area can be effectively treated like a cow-area (as we must disconnect it from all hiking areas). The same applies to a forbidden area adjacent to a forbidden area adjacent to a cow-area, and so on. We can expand in this way until we find the outside-boundary areas where we should place the walls. More precisely, if we consider the graph obtained by looking only at cow-areas and forbidden areas, then we should place walls at the outside boundary of every connected component of this graph *that contains a cow-area*.

If any of these outside-boundary areas happens to be a hiking area, then there exists a path of the form "cow-area – zero or more forbidden areas – hiking area", so the current choice of r is infeasible. Otherwise, the cows are successfully surrounded, and we need to check whether the hiking areas are still connected (using BFS/DFS – see Subtask 1 or 2). If yes, we return success into the binary search (the current value of r is feasible). If not, we return failure – and we can do this for the same reason as in Subtask 2.