Tracks in the snow (Spoiler)

1 Greedy Approach

Remark 1. *The animal that crossed the meadow last can be deduced directly from the input: it is simply the animal that left the topmost tracks in the upper left (or, alternatively, lower right) corner.*

Remark 2. *In an optimal solution (using the minimum number of animals) no two animals of the same type will cross the meadow directly after each other.*

Proof. We could simply merge the two paths (go along the first one from the upper left to the lower right corner, return on the same way and take the second path down) to get a new one, thereby reducing the number of animals used. □

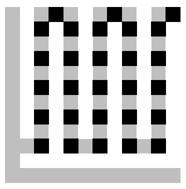
In any step the current animal can visit all the cells reachable from the lower left corner using only already visited cells and cells containing his respective tracks. Now consider the following greedy algorithm: at any step we simply visit all reachable cells, as long as there are unvisited cells (this algorithm processes the animals in *reverse* order). Note that the animal we choose at each step is given by the remarks above.

Proposition 3. *The described algorithm is correct and uses the minimum number of animals possible.*

Proof. We can clearly visit all those cells and since there is a way from source to target completely covered with tracks from the last animal we can simply go back to the source and follow that way. Thus the result of the algorithm corresponds to a legal sets of animal moves.

We now show by induction that, for any n, after having used n animals we have visited all the cells possible. The case $n = 1$ follows by definition. Now assume $n \geq 2$. Let S be the set of all cells that could be reached by using n animals but isn't visited by the above algorithm using only n animals. If $S = \emptyset$ we're finished. Otherwise take the cell s with the minium distance to the upper left corner (measured in the minimum number of cells on any valid path to this cell). Then by choice all other cells on this path are visited by our algorithm. If the cell considered were reachable by less than n animals, we would have already visited it. So according to the remarks above the topmost track on the cell is the same we are currently considering in our algorithm, so our algorithm will visit it contradicting the choice of s. So S is indeed empty and thus our algorithm is indeed optimal. □

A simple way to implement this algorithm is to set all cells visited so far to a special "Don't care" character (e.g.'$\star$'). This algorithm needs $O(n^2m^2)$ steps for an $n \times m$-meadow and suffices for the first subtask. The figure on the right shows that there are indeed up to $\Omega(mn)$ animals needed.



2 Linear solution

Remove from the graph considered before all edges between cells with different topmost tracks. Now we can contract all those components to single nodes and add the deleted edges thereby obtaining a new graph H. For simplicity number the nodes from 1 to k where node 1 represents the component containing the upper left corner.

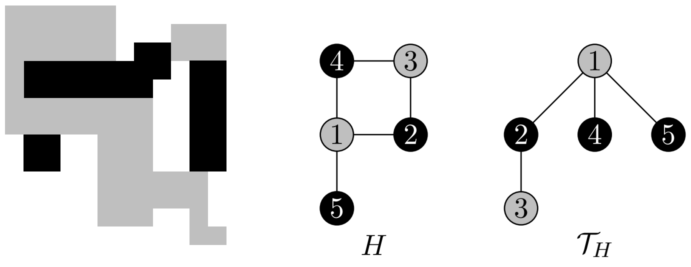


Figure 1: An example of tracks, the respective graph and a possible breadth-first-tree

This graph is clearly bipartite with a bicoloring given by the assignment component $\mapsto$ topmost track. Consider the breadth-first tree $\mathcal{T}_H$ (starting at node 1). Note that both H and $\mathcal{T}_H$ can be calculated in time $O(mn)$.

Proposition 4. *The minimum number a of animals needed to create the given pattern equals the depth of $\mathcal{T}_H$, i.e. $a = 1 + \max_{v \in V(H)} \operatorname{dist}(1, v)$.*

Proof. We show by induction that the cells visited by the previously described greedy algorithm after using n animals are exactly the cells represented by the nodes within distance $n - 1$ from the root. Again the case $n = 1$ is trivial. For $n \geq 2$ we can visit all the nodes "in the layer below": since we have $E(H) \supseteq E(\mathcal{T}_H)$ the bicoloring of H gives a bicoloring of $\mathcal{T}_H$ and thus all the nodes in the next layer have the same, namely the currently active, color, and are visited by the greedy algorithm.

Furthermore the greedy algorithm doesn't reach any other nodes: the nodes visited so far are the previous layers. Assume component $\mathcal{C}$ can be visited and take any path from the root to $\mathcal{C}$ and throw away all the parts up to the last node that is already visited. The remaining nodes have to be of the currently active color, i.e. they belong the component $\mathcal{C}$. Thus if $\mathcal{C}$ hasn't been visited before it is child of a node visited before and will be visited in this step. $\square$

This suffices for an $O(mn)$ solution (i.e. linear in the size of the input). However, instead of explicitely calculating components one can assign weights to all the edges in the original graph: any edge between two nodes of the same color gets weight 0 and any other edge weight 1. Then clearly the maximum distance of any node to the root (the upper left corner) equals a. The distances can be calculated by Dijksta's algorithm where we can use a **deque** instead of a **priority_queue** (0/1-Dijkstra) to achieve a linear running time.