Jumping to Conclusions

Although this problem is a 2D variation of the array-jumping problem, which has both DP and greedy solutions, it requires a little extra thought. The difficulty here is that we are allowed to move in four directions (up, down, left, right), so DP may not be the best approach. Let's examine a greedy solution instead.

First, note that if a cell can be reached from the origin in the first move, then this is the best possible outcome for that cell. By induction, we can infer that the best outcome for any unvisited cell is to move there directly from a cell in the visited set. Thus, the problem can be reduced to finding shortest paths in an unweighted graph, lending itself to a BFS solution.

Let's keep track of the set of visited cells and queue up the ones we should visit in the next iteration. We start with cell $(1, 1)$. Then, for each cell in the queue, we add all unvisited cells within its reach to a new queue, to be used in the next iteration. Both queues take up at most nm additional memory.

We can stop the algorithm once we reach the destination, in which case the answer will be the number of iterations, or when the queue becomes empty (which means the destination cannot be reached).

To avoid the cost of visiting a cell more than once, we can use a DSU data structure to keep track of visited cells. There will be one DSU for each row and each column, totaling $2nm$ additional memory.

Marking a cell as visited translates to the DSU's `merge` method, which in this case has logarithmic complexity. Hence, the overall time complexity is $O(nm \cdot \log nm)$.