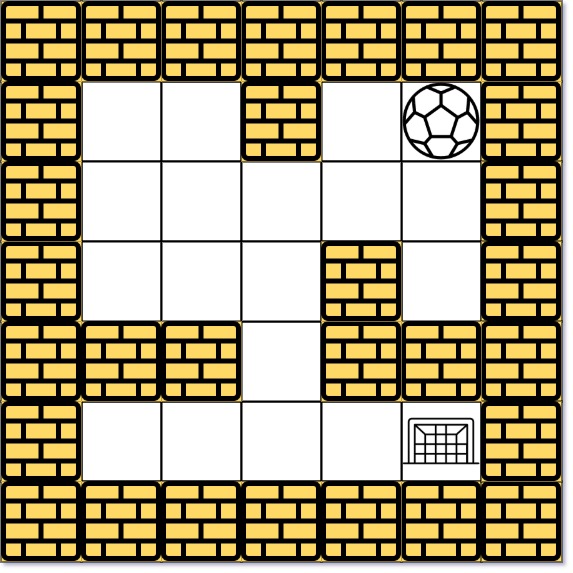
There is a ball in a maze with empty spaces (represented as 0) and walls (represented as 1). The ball can go through the empty spaces by rolling **up, down, left or right**, but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction.

Given the m x n maze, the ball's start position and the destination, where start = [startrow, startcol] and destination = [destinationrow, destinationcol], return *the shortest****distance****for the ball to stop at the destination*. If the ball cannot stop at destination, return -1.

The **distance** is the number of **empty spaces** traveled by the ball from the start position (excluded) to the destination (included).

You may assume that **the borders of the maze are all walls** (see examples).

**Example 1:**



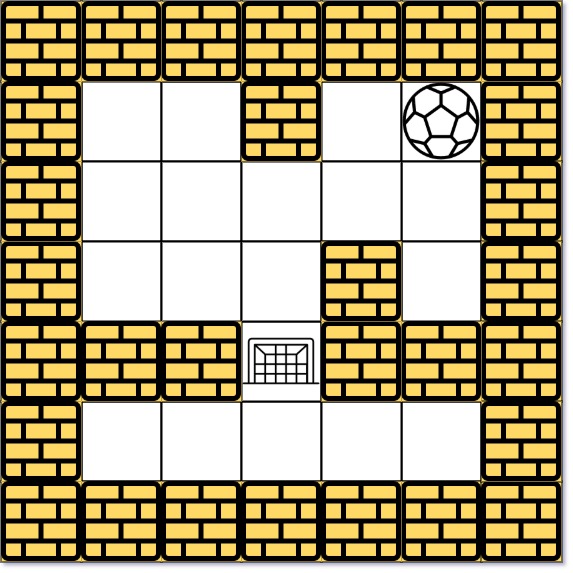
**Input:** maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]], start = [0,4], destination = [4,4]

**Output:** 12

**Explanation:** One possible way is : left -> down -> left -> down -> right -> down -> right.

The length of the path is 1 + 1 + 3 + 1 + 2 + 2 + 2 = 12.

**Example 2:**



**Input:** maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]], start = [0,4], destination = [3,2]

**Output:** -1

**Explanation:** There is no way for the ball to stop at the destination. Notice that you can pass through the destination but you cannot stop there.

**Example 3:**

**Input:** maze = [[0,0,0,0,0],[1,1,0,0,1],[0,0,0,0,0],[0,1,0,0,1],[0,1,0,0,0]], start = [4,3], destination = [0,1]

**Output:** -1

**Constraints:**

* m == maze.length
* n == maze[i].length
* 1 <= m, n <= 100
* maze[i][j] is 0 or 1.
* start.length == 2
* destination.length == 2
* 0 <= startrow, destinationrow <= m
* 0 <= startcol, destinationcol <= n
* Both the ball and the destination exist in an empty space, and they will not be in the same position initially.
* The maze contains **at least 2 empty spaces**.