Solution

Approach 1: Dynamic Programming

**Intuition**

The robot can only move either down or right. Hence any cell in the first row can only be reached from the cell left to it.

1 / 4

And, any cell in the first column can only be reached from the cell above it.

1 / 4

For any other cell in the grid, we can reach it either from the cell to left of it or the cell above it.

If any cell has an obstacle, we won't let that cell contribute to any path.

We will be iterating the array from left-to-right and top-to-bottom. Thus, before reaching any cell we would have the number of ways of reaching the predecessor cells. This is what makes it a Dynamic Programming problem. We will be using the obstacleGrid array as the DP array thus not utilizing any additional space.

Note: As per the question, cell with an obstacle has a value 1. We would use this value to make sure if a cell needs to be included in the path or not. After that we can use the same cell to store the number of ways to reach that cell.

**Algorithm**

1. If the first cell i.e. obstacleGrid[0,0] contains 1, this means there is an obstacle in the first cell. Hence the robot won't be able to make any move and we would return the number of ways as 0.
2. Otherwise, if obstacleGrid[0,0] has a 0 originally we set it to 1 and move ahead.
3. Iterate the first row. If a cell originally contains a 1, this means the current cell has an obstacle and shouldn't contribute to any path. Hence, set the value of that cell to 0. Otherwise, set it to the value of previous cell i.e. obstacleGrid[i,j] = obstacleGrid[i,j-1]
4. Iterate the first column. If a cell originally contains a 1, this means the current cell has an obstacle and shouldn't contribute to any path. Hence, set the value of that cell to 0. Otherwise, set it to the value of previous cell i.e. obstacleGrid[i,j] = obstacleGrid[i-1,j]
5. Now, iterate through the array starting from cell obstacleGrid[1,1]. If a cell originally doesn't contain any obstacle then the number of ways of reaching that cell would be the sum of number of ways of reaching the cell above it and number of ways of reaching the cell to the left of it.

obstacleGrid[i,j] = obstacleGrid[i-1,j] + obstacleGrid[i,j-1]

1. If a cell contains an obstacle set it to 0 and continue. This is done to make sure it doesn't contribute to any other path.

Following is the animation to explain the algorithm's steps:

1 / 18

**Complexity Analysis**

* Time Complexity: O(M \times N)*O*(*M*×*N*). The rectangular grid given to us is of size M \times N*M*×*N* and we process each cell just once.
* Space Complexity: O(1)*O*(1). We are utilizing the obstacleGrid as the DP array. Hence, no extra space.