63. Unique Paths II

Oct. 22, 2018 | 64.8K views

★★★★★ Average Rating: 4.64 (58 votes)

A robot is located at the top-left corner of a *m* x *n* grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as `1` and `0` respectively in the grid.

Note: *m* and *n* will be at most 100.

Example 1:

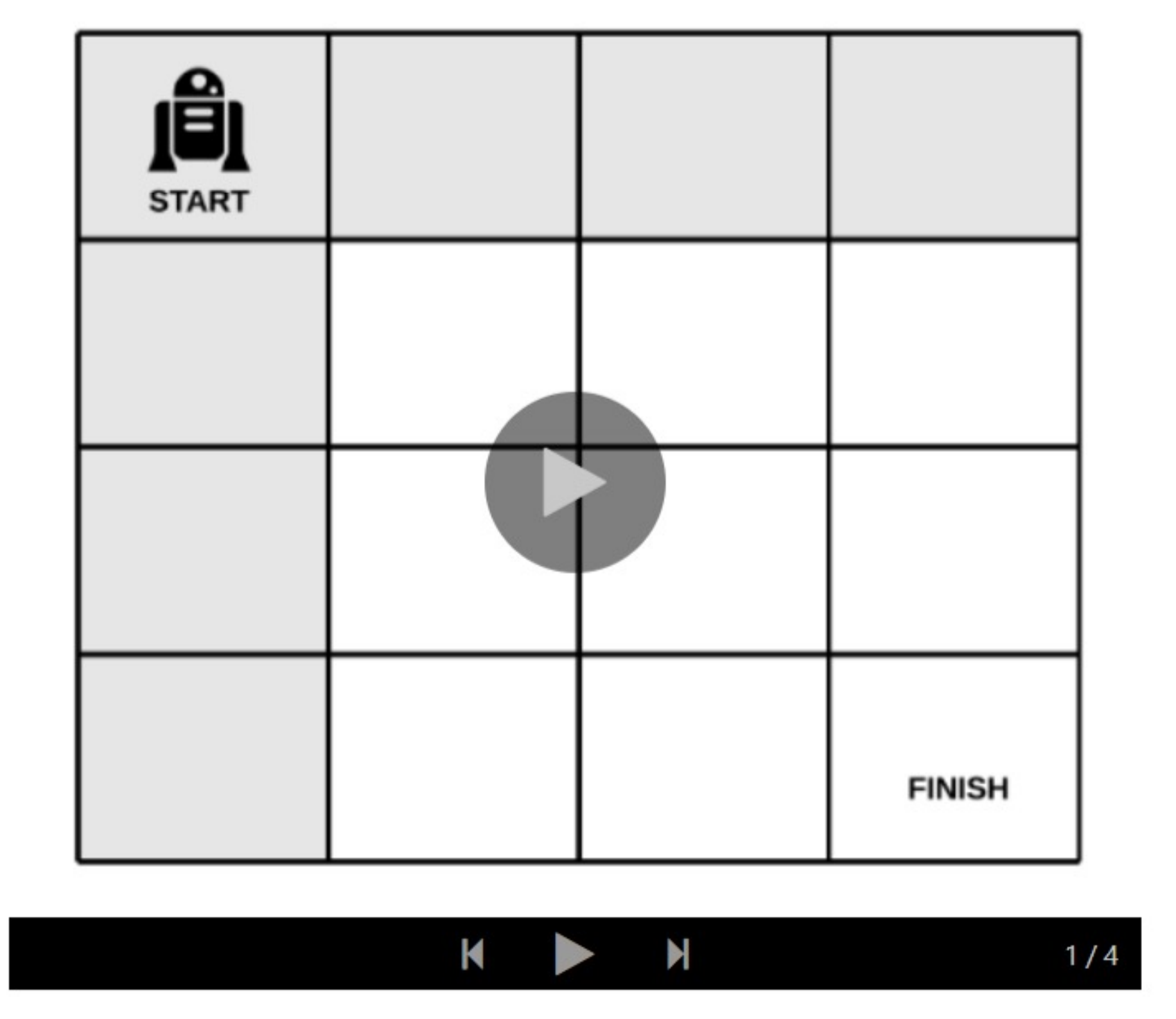
```
Input:
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
Output: 2
Explanation:
There is one obstacle in the middle of the 3x3 grid above.
There are two ways to reach the bottom-right corner:
1. Right -> Right -> Down -> Down
2. Down -> Down -> Right -> Right
```

Solution

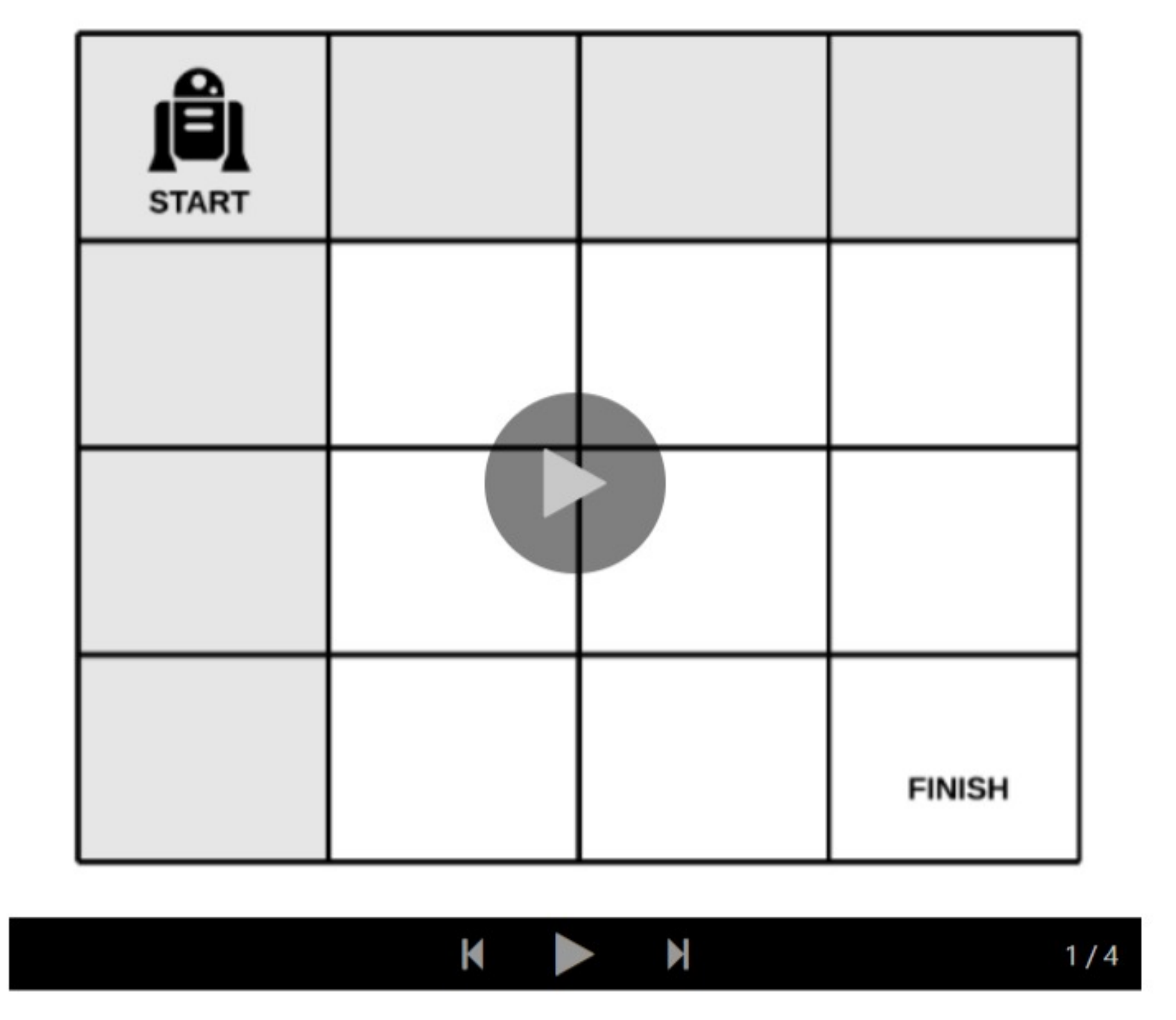
Approach 1: Dynamic Programming

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.



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



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.

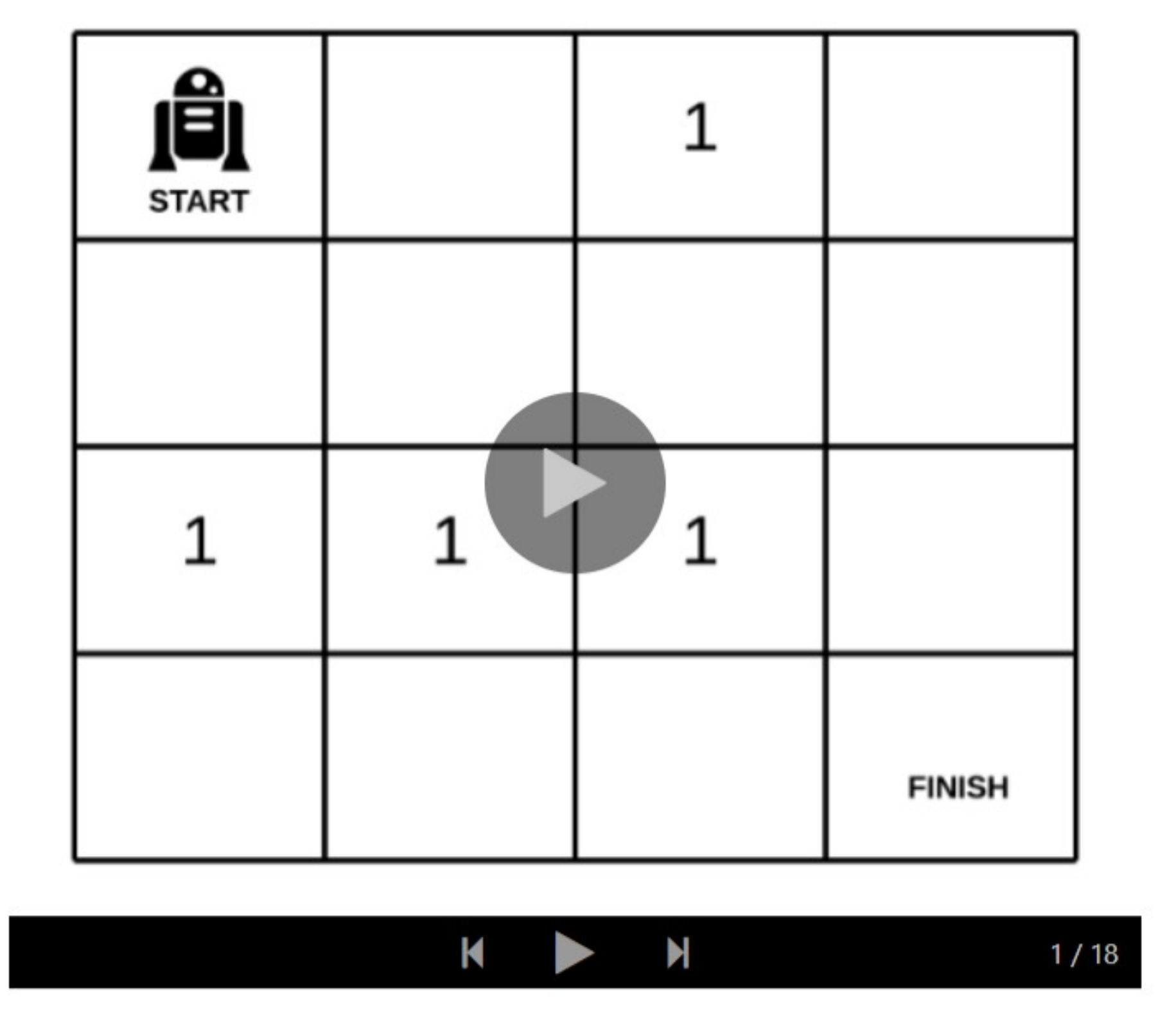
Algorithm

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]
```

6. 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:



Java Python Copy

```
class Solution(object):
    def uniquePathsWithObstacles(self, obstacleGrid):
        """
        :type obstacleGrid: List[List[int]]
        :rtype: int
        """

        m = len(obstacleGrid)
        n = len(obstacleGrid[0])

        # If the starting cell has an obstacle, then simply return as there would be
        # no paths to the destination.
        if obstacleGrid[0][0] == 1:
            return 0

        # Number of ways of reaching the starting cell = 1.
        obstacleGrid[0][0] = 1

        # Filling the values for the first column
        for i in range(1,m):
            obstacleGrid[i][0] = int(obstacleGrid[i][0] == 0 and obstacleGrid[i-1][0] == 1)

        # Filling the values for the first row
        for j in range(1, n):
            obstacleGrid[0][j] = int(obstacleGrid[0][j] == 0 and obstacleGrid[0][j-1] == 1)

        # Starting from cell(1,1) fill up the values
```

Complexity Analysis

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

Rate this article: ★★★★★

Previous Next

Comments: 25

Sort By

Type comment here... (Markdown is supported)

Preview Post

calvinchankf ★ 2917 February 20, 2019 10:09 AM

I think the article should include more alternatives cos an interviewer must ask you for more than one approach during an interview and (s)he might then want you to compare the approaches.

so like, apart from your iterative approach, here is another approach with recursions

Read More

30 Share Reply

SHOW 2 REPLIES

orioncyngus ★ 95 March 23, 2019 9:24 AM

The O(1) space solution doesn't work if you have to consider the possibility of integer overflow, which one of the test cases does. To avoid integer overflow, you can't have a constant space solution in this case.

18 Share Reply

SHOW 5 REPLIES

ab_rai ★ 26 December 1, 2019 12:03 PM

Please help!!!
If we use the given matrix then it is showing integer overflow.
Line 21: Char 60: runtime error: signed integer overflow: 1053165744 + 1579748616 cannot be represented in type 'int' (solution.cpp)

Read More

12 Share Reply

SHOW 1 REPLY

snedden27 ★ 12 January 16, 2019 10:06 PM

Nice Explanation. Would love to have unique path 3 with not constraint of only moving left and bottom

6 Share Reply

Hurrymomo ★ 10 August 10, 2019 12:59 PM

The solution provided has a precision problem. Loosely speaking, when we go along the diagonal in the matrix, for each move, we at least doubled number of paths. so if we have a few obstacle in 100x100 board, we need more than 100 bits to track the number of paths.

4 Share Reply

sima ★ 13 October 23, 2018 11:08 PM Report

Really fancy demo!!!!
Could you share how you made the vedio?

2 Share Reply

SHOW 1 REPLY

voyangli ★ 1 October 26, 2019 9:05 AM

The solution provided by Java has a signed integer overflow problem. The value of obatacleGrid[21][28] cannot be represented in type 'int' in the 27 test case. It is more rigorous to use type 'long' although using type 'int' is accepted.

1 Share Reply

SHOW 1 REPLY

xxtt ★ 58 July 2, 2019 8:44 AM

Nice solution, but maybe we should also consider whether modifying the given matrix is appropriate. If we don't want to spend extra time recovering the original matrix, then I think we have to copy the whole thing with O(m*n) space still.

1 Share Reply

JerryWuX ★ 101 November 14, 2018 3:22 AM

Hi godayaldivya
Thank you for you crystal clear explanation! However I am confused about the space complexity of the dynamic programming. Take this specific question as an example, we use the obstacleGrid to memorize the previous result, but why doesn't it count as an extra space? This algorithm still request a piece of memory to store the matrix, which has m * n elements.

Read More

1 Share Reply

SHOW 2 REPLIES

k_spring ★ 0 June 17, 2019 5:42 PM

666

0 Share Reply

‹ 1 2 3 ›