Grid Unique Paths II - Maze Obstacles

Created by Chandra Prakash

Problem Statement

We are given an MxN Maze. The maze contains some obstacles. A cell is 'blockage' in the maze if its value is -1. 0 represents non-blockage. There is no path possible through a blocked cell.

We need to count the total number of unique paths from the top-left corner of the maze to the bottom-right corner. At every cell, we can move either down or towards the right.

Editorial

Recursion

Approach

Understand the Movement and Base Rules: A cell `maze[i][j] == -1` is **blocked**: cannot step into it.

Define the Subproblem: Let `f(i, j)` represent the **number of unique paths to reach cell (i, j)** from the top-left, considering blockages.

Note: If `(i, j)` is **blocked**, there is **no path** that can ever end at this cell.

That is, we can return 0 if the element is -1.

```
if (maze[i][j] == -1) return 0;
```

We will be doing a top-down recursion, i.e we will move from the `cell[M-1][N-1]` and try to find our way to the `cell[0][0]`. Therefore at every index, we will try to move up and towards the left.

Code

```
int f(int i, int j, vector<vector<int>>& maze){
    if(i == 0 && j == 0) return 1;
    if(i < 0 || j < 0) return 0;

    if(maze[i][j] == -1) return 0;

    return f(i - 1, j, maze) + f(i, j - 1, maze);
}
```

Complexity Analysis

Time Complexity: `O(2^(m+n))` - because each cell may branch into two recursive calls (up and left), leading to exponential growth.

Space Complexity: `O(m + n)` - due to the maximum depth of the recursion stack from top-left to bottom-right.

Memoization

Approach

General steps:

- Store the result.
- Return value if already calculated.

Code

```
int f(int i, int j, vector<vector<int>>& maze, vector<vector<int>>& dp) {
    if (i < 0 || j < 0) return 0;
    if (maze[i][j] == -1) return 0;
    if (i == 0 && j == 0) return 1;
    if (dp[i][j] != -1) return dp[i][j];
    return dp[i][j] = f(i - 1, j, maze, dp) + f(i, j - 1, maze, dp);
}
```

Complexity Analysis

Time Complexity: `O(m * n)` - each subproblem `(i, j)` is solved only once and stored in the `dp` table.

Space Complexity: `O(m * n)` - for the `dp` table used in memoization and recursion stack depth.

Tabulation

Approach

For each cell `(i, j)`, skip it if it's blocked; otherwise:

- If it's the start cell `(0, 0)`, set `dp[0][0] = 1`
- Otherwise, set `dp[i][j] = dp[i-1][j] + dp[i][j-1]` (handling bounds).

Corner cases: If the starting cell `(0,0)` is blocked (`-1`), return 0 immediately.

Code

```
int f(vector<vector<int>>& maze) {
    int m = maze.size(), n = maze[0].size();
    if (maze[0][0] == -1 || maze[m - 1][n - 1] == -1) return 0;

    vector<vector<int>> dp(m, vector<int>(n, 0));

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (maze[i][j] == -1) {
                dp[i][j] = 0;
                continue;
            }
            if (i == 0 && j == 0) {
                dp[i][j] = 1;
            } else {
                int up = (i > 0) ? dp[i - 1][j] : 0;
```

```
            int left = (j > 0) ? dp[i][j - 1] : 0;
            dp[i][j] = up + left;
        }
    }
  }
  return dp[m - 1][n - 1];
}
```

Complexity Analysis

Time Complexity: `O(m * n)` - each cell in the grid is visited once.

Space Complexity: `O(m * n)` - for storing the DP table.

Space optimization

Approach

prev question code:

```
int f(int m, int n) {
  vector<int> prev(n, 0);
  for (int i = 0; i < m; i++) {
    vector<int> curr(n, 0);
    for (int j = 0; j < n; j++) {
      if (i == 0 && j == 0) {
        curr[j] = 1;
      } else {
        int up = (i > 0) ? prev[j] : 0;
        int left = (j > 0) ? curr[j - 1] : 0;
        curr[j] = up + left;
      }
    }
    prev = curr;
  }
  return prev[n - 1];
}
```

For each cell `(i, j)`:

- If the cell is blocked (`1`), set `curr[j] = 0`

Code

```
int uniquePathsWithObstacles(vector<vector<int>>& maze) {
  int m = maze.size(), n = maze[0].size();
  if (maze[0][0] == -1 || maze[m - 1][n - 1] == -1) return 0;
  vector<int> prev(n, 0);
  for (int i = 0; i < m; i++) {
    vector<int> curr(n, 0);
    for (int j = 0; j < n; j++) {
      if (maze[i][j] == -1) {
        curr[j] = 0;
```

```
            } else if (i == 0 && j == 0) {
                curr[j] = 1;
            } else {
                int up = (i > 0) ? prev[j] : 0;
                int left = (j > 0) ? curr[j - 1] : 0;
                curr[j] = up + left;
            }
        }
        prev = curr;
    }
    return prev[n - 1];
}
```

Complexity Analysis

Time Complexity: `O(m * n)` - every cell is processed once

Space Complexity: `O(n)` - only two 1D arrays of size `n` are used to store state

---

THE END