Yuan Gao z5239220 Q2

You are given a $2D$ map consisting of an $R \times C$ grid of squares; in each square there is a number representing the elevation of the terrain at that square. Find a path going from square $(1, R)$ which is the top left corner of the map to square $(C, 1)$ in the lower right corner which from every square goes only to the square immediately below or the square immediately to the right so that the number of moves from lower elevation to higher elevation along such a path is as small as possible.

1. Create dynamic programming array

> Create dp is a $m \times n$ matrix and `dp[i][j]` means that the miminum number of moves from lower elevation to higher elevation at (i, j)

> > Every element is "0"
> > Because the path is from the top left corner to the lower right bottom corner, in every moving, there are only two options(moving down or moving right).

> > Hence, if current position is `grid[i][j]`:
> >
> > > -Compare the value of terrain `grid[i - 1][j]` and `grid[i][j - 1]` and store the value in `left` and `top`
> > > -`dp[i][j] = min(dp[i - 1][j] + left, dp[i][j - 1] + top)`

> The result is `dp[R - 1][C - 1]`

2. Example

Grid

1	2	3
1	3	2
2	5	1

dp Array

0	0	0
0	0	0
0	0	0

$\longrightarrow$

0	1	2
0	1	1
1	2	1

result = 1

3. Code

```
int findBestPath(vector<vector<int>> grid){
    //get length and wide of grid
    int R = grid.size();
    int C = grid[0].size();
    //create dp matrix
    vector<vector<int>> dp(R, vector<int>(C, 0));
    //deal with first column
    for(int i = 1; i < R; ++i){
        if(grid[i - 1][0] < grid[i][0]){
            dp[i][0] += dp[i - 1][0] + 1;
        }
    }
    //deal with first raw
    for(int j = 1; j < C; ++j){
        if(grid[0][j - 1] < grid[0][j]){
            dp[0][j] += dp[0][j - 1] + 1;
        }
    }
    //deal with remainder
    for(int i = 1; i < R; ++i){
        for(int j = 1; j < C; ++j){
            int top = 0;
            int left = 0;
            if(grid[i - 1][j] < grid[i][j]){
                ++left;
            }
            if(grid[i][j - 1] < grid[i][j]){
                ++top;
            }
            dp[i][j] = min(dp[i - 1][j] + left, dp[i][j - 1] + top);
        }
    }

    return dp[R - 1][C - 1];
}
```

4. Time complexity

> Traverse a two-dimensional array
> Hence, total time complexity is $O(R \times C)$