Documentation for Minimum Path Sum Solution

Problem Statement

Given an (m times n) grid filled with non-negative numbers, find a path from the top-left to the bottom-right corner which minimizes the sum of all numbers along its path. You can only move either down or right at any point in time.

Examples

Example 1

Input:

grid = [

    [1, 3, 1],

    [1, 5, 1],

    [4, 2, 1]

]

Output: 7

Explanation: The path that minimizes the sum is 1 → 3 → 1 → 1 → 1, with a sum of 7.

Example 2

Input:

grid = [

[1, 2, 3],

[4, 5, 6]

]

Output: 12

Explanation: The path that minimizes the sum is 1 → 2 → 3 → 6, with a sum of 12.

Constraints

- (m == text{grid.length})
- (n == text{grid[i].length})
- (1 leq m, n leq 200)
- (0 leq text{grid[i][j]} leq 200)

Solution

The solution uses dynamic programming to compute the minimum path sum. It maintains a 2D list `dp` where `dp[i][j]` represents the minimum path sum to reach the cell `(i, j)`.

Algorithm

1. **Initialization:**
    - If the grid is empty, return 0.
    - Initialize `m` and `n` to be the number of rows and columns of the grid, respectively.
    - Create a 2D list `dp` of size `m x n` initialized to 0.

2. **Base Case:**
    - Set `dp[0][0]` to `grid[0][0]` because this is the starting point.

3. **Fill the First Row:**
    - For each column `j` from 1 to `n-1`, set `dp[0][j]` to `dp[0][j-1] + grid[0][j]` since you can only move right in the first row.

4. **Fill the First Column:**
    - For each row `i` from 1 to `m-1`, set `dp[i][0]` to `dp[i-1][0] + grid[i][0]` since you can only move down in the first column.

5. **Fill the Rest of the dp Table:**
    - For each cell `(i, j)` where `i > 0` and `j > 0`, set `dp[i][j]` to the minimum of the values from the cell above (`dp[i-1][j]`) and the cell to the left (`dp[i][j-1]`), plus the value of the current cell (`grid[i][j]`).

6. **Result:**
    - The value at `dp[m-1][n-1]` will be the minimum path sum to reach the bottom-right corner of the grid.