# 题目

在一个 m\*n 的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物？

**示例 1:**

输入:

[

  [1,3,1],

  [1,5,1],

  [4,2,1]

]

输出: 12

解释: 路径 1→3→5→2→1 可以拿到最多价值的礼物

**提示：**

0 < grid.length <= 200

0 < grid[0].length <= 200

# 分析

## 方法一：动态规划

**思路：**

状态表示：dp[i][j]表示到达i,j位置处的最大价值

转移方程：dp[i][j] = max{dp[i-1][j],dp[i][j-1]} + grid[i][j];

dp数组初始状态：dp[0][j] = {0}, dp[i][0] = {0} （就是将第一行和第一列全部设为0）

其实这里的dp数组我们可以将它优化成一维的，优化之后的一维dp保存的是当前上一行的最大价值，然后我们从做到有去更新这个数组即可

**代码：**

class Solution {

public:

int maxValue(vector<vector<int>>& grid) {

int row = grid.size();

int col = grid[0].size();

vector<int> dp(col+1,0);

for(int i=0;i<row;i++)

{

for(int j=0;j<col;j++)

{

dp[j+1] = max(dp[j],dp[j+1]) + grid[i][j];

}

}

return dp[col];

}

};

或（推荐这种写法，类似LeetCode 64）：

class Solution {

public:

    int maxValue(vector<vector<int>>& grid) {

        int m = grid.size();

        int n = grid[0].size();

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

        dp[0][0] = grid[0][0];

        for(int i=0;i<m;i++)

        {

            for(int j=0;j<n;j++)

            {

                if(i==0 && j!=0)

                {

                    dp[i][j] = dp[i][j-1] + grid[i][j];

                }

                else if(i!=0 && j==0)

                {

                    dp[i][j] = dp[i-1][j] + grid[i][j];

                }

                else if(i!=0 && j!=0)

                {

                    dp[i][j] = max(dp[i-1][j],dp[i][j-1])+ grid[i][j];

                }

            }

        }

        return dp[m-1][n-1];

    }

};