剑指 Offer 47. 礼物的最大价值

Tuesday, June 7, 2022 10:44 AM

https://leetcode.cn/problems/li-wu-de-zui-da-jie-zhi-lcof/

中等

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

0 < grid.length <= 200
0 < grid[0].length <= 200

我的思路（能accept 但有问题）：动规

1. dp[i][j]:到达此下标的最大价值
2. 因为从左上角出发的，故初始化dp数组最左边和最上边数值。
   最左：dp[i][0]
   最上：dp[0][j]
3. 循环更新dp数组
   当前网格最大值 等于 左一格或上一格dp最大值 加上 当前格数值。
   dp[i][j] = Math.max(dp[i][j-1],dp[i-1][j]) + value
   ps：增加变量maxValue, 遍历时更新该值。最后返回。

code：

```
class Solution {
    public int maxValue(int[][] grid) {
        int[][] dp = new int[grid.length][grid[0].length];
        dp[0][0] = grid[0][0];
        int maxValue = grid[0][0];
        // 初始化最左边
        for(int i=1;i<grid.length;i++){
            dp[i][0] = dp[i-1][0]+grid[i][0];
        // 更新最大值
```

```
                maxValue = (maxValue < dp[i][0]) ? dp[i][0] : maxValue;
            }
            // 初始化最右边
            for(int j=1;j<grid[0].length;j++){
                dp[0][j] = dp[0][j-1]+grid[0][j];
                maxValue = (maxValue < dp[0][j]) ? dp[0][j] : maxValue;
            }
            // 更新dp数组、最大值
            for(int i=1;i<grid.length;i++){
                for(int j=1;j<grid[0].length;j++){
                    dp[i][j] = Math.max(dp[i][j-1],dp[i-1][j]) + grid[i][j];
                    maxValue = (maxValue < dp[i][j]) ? dp[i][j] : maxValue;
                }
            }
            return maxValue;
        }
    }
```

问题：

1. 初始化时记得确保 行列均不会越界，
   即 I,j 的边界分别是 grid.length, grid[0].length
2. dp[i][j] 的含义就是**路径至当前格的最大值**，还搞什么 maxValue ?!!!!!

正确code：

```
    class Solution {
        public int maxValue(int[][] grid) {
            int[][] dp = new int[grid.length][grid[0].length];
            // 初始化 00 位置，避免 最左边和最右边 初始化时重复计算该格
            dp[0][0] = grid[0][0];
            // 初始化最左列和最上行，因为dp数组需要从 左或上一格 数值进行更新
            for(int i=1;i<grid.length;i++){
                dp[i][0] = dp[i-1][0]+grid[i][0];
            }
            for(int j=1;j<grid[0].length;j++){
                dp[0][j] = dp[0][j-1]+grid[0][j];
            }
            // 更新dp数组
            for(int i=1;i<grid.length;i++){
                for(int j=1;j<grid[0].length;j++){
                    dp[i][j] = Math.max(dp[i][j-1],dp[i-1][j]) + grid[i][j];
                }
            }
```

```
        return dp[grid.length-1][grid[0].length-1];
    }
}
```