本日学习内容

1. 完成管理系统初步
2. dp的深化，随想录写到动态规划 - 11
3. acwing动态规划部分
4. 反转链表的四种方法

本日分享内容

题目1.

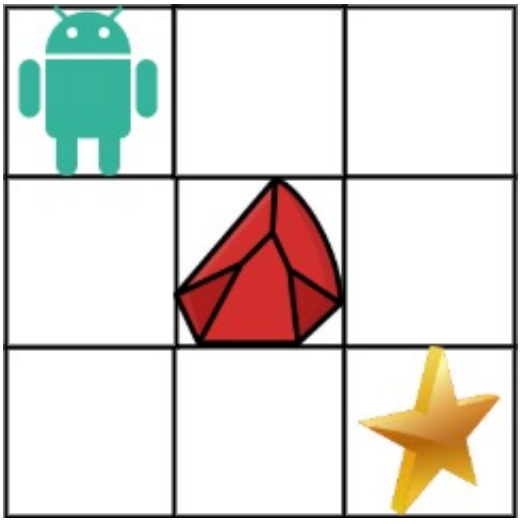
给定一个 `m x n` 的整数数组 `grid`。一个机器人初始位于 **左上角**（即 `grid[0][0]`）。机器人尝试移动到 **右下角**（即 `grid[m - 1][n - 1]`）。机器人每次只能向下或者向右移动一步。

网格中的障碍物和空位置分别用 `1` 和 `0` 来表示。机器人的移动路径中不能包含 **任何** 有障碍物的方格。

返回机器人能够到达右下角的不同路径数量。

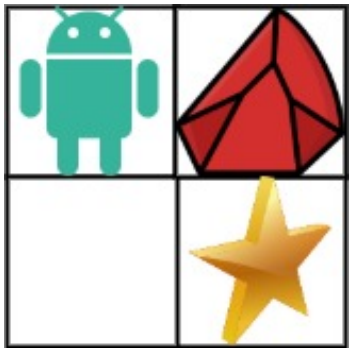
测试用例保证答案小于等于 `2 * 109`。

示例 1：



```
输入：obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]
输出：2
解释：3x3 网格的正中间有一个障碍物。
从左上角到右下角一共有 2 条不同的路径：
1. 向右 -> 向右 -> 向下 -> 向下
2. 向下 -> 向下 -> 向右 -> 向右
```

示例 2：



```
输入：obstacleGrid = [[0,1],[0,0]]
输出：1
```

提示：

m == obstacleGrid.length `

`n == obstacleGrid[i].length`

1 <= m, n <= 100 `

`obstacleGrid[i][j]` 为 `0` 或 `1`

思路

整体与不同路径I相似，只需将障碍点dp设为0即可

代码

```
class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
        if (obstacleGrid[0][0] == 1) {
            return 0;
        }
        vector<int> dp(obstacleGrid[0].size());
        for (int j = 0; j < dp.size(); ++j) {
            if (obstacleGrid[0][j] == 1) {
                dp[j] = 0;
            } else if (j == 0) {
                dp[j] = 1;
            } else {
                dp[j] = dp[j-1];
            }
        }

        for (int i = 1; i < obstacleGrid.size(); ++i) {
            for (int j = 0; j < dp.size(); ++j) {
                if (obstacleGrid[i][j] == 1) {
```

```
                    dp[j] = 0;
                } else if (j != 0) {
                    dp[j] = dp[j] + dp[j-1];
                }
            }
        }
        return dp.back();
    }
};
```

题目2.

给定一个正整数 `n` ，将其拆分为 `k` 个 **正整数** 的和（ `k >= 2` ），并使这些整数的乘积最大化。

返回 *你可以获得的最大乘积* 。

示例 1:

```
输入: n = 2
输出: 1
解释: 2 = 1 + 1, 1 × 1 = 1。
```

示例 2:

```
输入: n = 10
输出: 36
解释: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36。
```

提示:

`2 <= n <= 58`

思路

dp[i]表示i拆分后的最大乘积

转移方程： dp[i] = max(dp[i], j * (i - j), j * dp[i - j]) (1 < j < i)

代码

```cpp
class Solution {
public:
    int integerBreak(int n) {
        vector<int> dp(n + 1);
        dp[1] = 1;
        for (int i = 2;i <= n; i++) {
            for (int j = 1; j < i; j++) {
                dp[i] = max(dp[i], max(j * (i - j), j * dp[i - j]));
            }
        }
        return dp[n];
    }
};
```

本日遇到的问题

1. 最初未能理解迭代法反转链表
2. 不能在01背包问题的各种变式中准确识别出01背包模型

明日学习内容

1. 继续dp
2. 完成反转链表四种方法的博客
3. Leetcode每日一题