动态规划02

62.不同路径

中等 8min

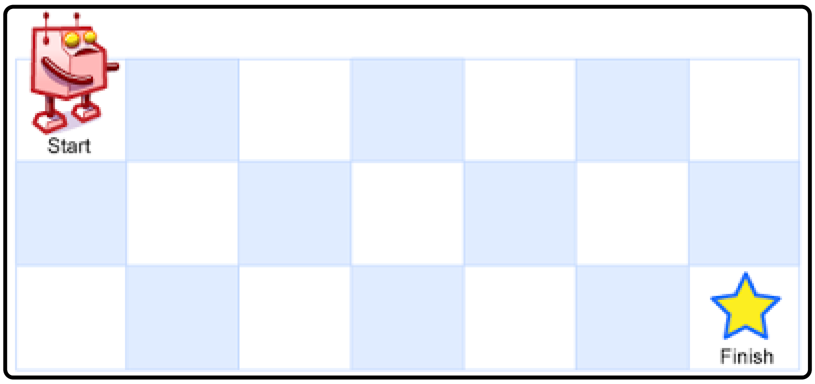
题目

一个机器人位于一个 `m x n` 网格的左上角 （起始点在下图中标记为 “Start” ）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为 “Finish” ）。

问总共有多少条不同的路径？

示例 1：




```
输入：m = 3, n = 7
输出：28
```

示例 2：

```
输入：m = 3, n = 2
输出：3
解释：
从左上角开始，总共有 3 条路径可以到达右下角。
1. 向右 -> 向下 -> 向下
2. 向下 -> 向下 -> 向右
3. 向下 -> 向右 -> 向下
```

示例 3：

```
输入：m = 7, n = 3
输出：28
```

示例 4：

```
输入：m = 3, n = 3
输出：6
```

提示：

- `1 <= m, n <= 100`
- 题目数据保证答案小于等于 `2 * 109`

题解

```cpp
class Solution {
public:
    int uniquePaths(int m, int n) {
        vector<vector<int>> dp(m,vector<int>(n,0));
        for(int i=0;i<m;i++)dp[i][0]=1;//初始化边界都是1;
        for(int j=0;j<n;j++)dp[0][j]=1;
        for(int i=1;i<m;i++){//从左上角到右下角依次求解
            for(int j=1;j<n;j++){
                dp[i][j]=dp[i-1][j]+dp[i][j-1];
            }
        }
        return dp[m-1][n-1];//输出最末尾的值
    }
};
```

63.不同路径 II

中等 5min

题目

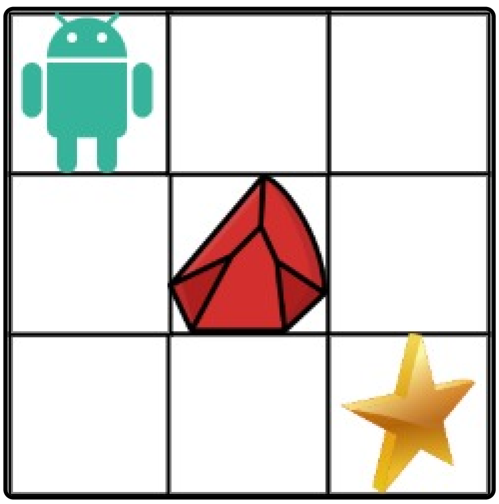
给定一个 `m x n` 的整数数组 `grid`。一个机器人初始位于 **左上角**（即 `grid[0][0]`）。机器人尝试移动到 **右下角**（即 `grid[m - 1][n - 1]`）。机器人每次只能向下或者向右移动一步。

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。

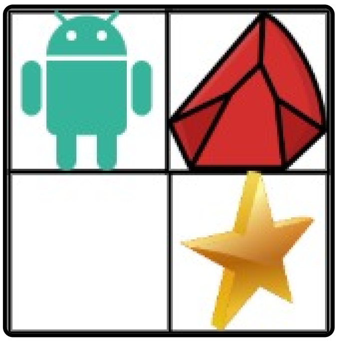
测试用例保证答案小于等于 `2 * 109`。

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`

题解

```
class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
        int m=obstacleGrid.size();
        int n=obstacleGrid[0].size();
        vector<vector<int>> dp(m,vector<int>(n,0));
        for(int i=0;i<m&&obstacleGrid[i][0]==0;i++)dp[i][0]=1;//前方无障碍填1
        for(int j=0;j<n&&obstacleGrid[0][j]==0;j++)dp[0][j]=1;
        for(int i=1;i<m;i++){
            for(int j=1;j<n;j++){
                if(obstacleGrid[i][j]==0)dp[i][j]=dp[i-1][j]+dp[i][j-1];
                //没有障碍再更新，有障碍就不变还是0
            }
        }
        return dp[m-1][n-1];
    }
};
```

整数拆分

中等15min

题目

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`

题解

这道题本来也是没什么想法，但是知道要用dp解决，很自然的就想到，可以把一个数字分成两个数相乘，而这两个数的整数拆分的乘积就是这样拆的最大乘积（?)感觉我表述的不是很明白但是dddd

对于的正整数 n，当 n≥2 时，可以拆分成至少两个正整数的和。令 k 是拆分出的第一个正整数，则剩下的部分是 n−k，n−k 可以不继续拆分，或者继续拆分成至少两个正整数的和。

- 递推公式

$$dp[i] = max\{dp[i], dp[i-j] \times j, (i-j) \times j\}$$

- 初始化状态

  0 不是正整数，1 是最小的正整数，0 和 1 都不能拆分，因此 $dp[0] = dp[1] = 0$。

- 遍历顺序

  由状态转移方程知道$dp[i]$ 是从$j \times (i-j)$ 和 $j \times dp[i-j]$ 中取，所以需要从前往后遍历所有的$dp[i]$,且 j 的取值范围是 1到 $i-1$。

```
class Solution {
public:
    int integerBreak(int n) {
        vector<int> dp(n + 1);
        dp[2] = 1;//初始化，2拆分等于1，从3开始迭代
        for (int i = 3; i <= n ; i++) {
            for (int j = 1; j < i - 1; j++) {
                dp[i] = max(dp[i], max((i - j) * j, dp[i - j] * j));
                //前面提到的递推公式
```

```
            }
        }
        return dp[n];
    }
};
```