## 题目地址(62. 不同路径)

https://leetcode-cn.com/problems/unique-paths/

## 入选理由

1. 二维网格 dp

## 标签

- 动态规划

## 难度

- 中等

## 题目描述

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?


例如，上图是一个7 x 3 的网格。有多少可能的路径?

示例 1:

```
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 -> 向右 -> 向下
2. 向右 -> 向下 -> 向右
3. 向下 -> 向右 -> 向右
示例 2：

输入：m = 7, n = 3
输出：28


提示：

1 <= m, n <= 100
题目数据保证答案小于等于 2 * 10 ^ 9
```

## 前置知识

- 排列组合
- 动态规划

## 公司

- 阿里
- 腾讯
- 百度
- 字节

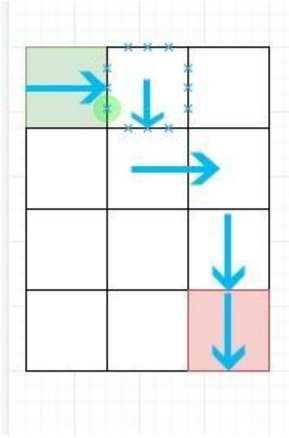
## 思路

首先这道题可以用排列组合的解法来解，需要一点高中的知识。

$$C_n^r = \frac{n!}{r!(n-r)!}$$

而这道题我们也可以用动态规划来解。其实这是一道典型的适合使用动态规划解决的题目，它和爬楼梯等都属于动态规划中最简单的题目，因此也经常会被用于面试之中。

读完题目你就能想到动态规划的话，建立模型并解决恐怕不是难事。其实我们很容易看出，由于机器人只能右移动和下移动，因此第[i, j]个格子的总数应该等于[i - 1, j] + [i, j -1]， 因为第[i,j]个格子一定是从左边或者上面移动过来的。



这不就是二维平面的爬楼梯么？和爬楼梯又有什么不同呢?

代码大概是：

Python Code:

```
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        d = [[1] * n for _ in range(m)]

        for col in range(1, m):
            for row in range(1, n):
                d[col][row] = d[col - 1][row] + d[col][row - 1]

        return d[m - 1][n - 1]
```

**复杂度分析**

- 时间复杂度：$O(M * N)$
- 空间复杂度：$O(M * N)$

由于 dp[i][j] 只依赖于左边的元素和上面的元素，因此空间复杂度可以进一步优化， 优化到 O(n).


具体代码请查看代码区。

当然你也可以使用记忆化递归的方式来进行，由于递归深度的原因，性能比上面的方法差不少：

> 直接暴力递归的话可能会超时。

Python3 Code:

```
class Solution:

    @lru_cache
    def uniquePaths(self, m: int, n: int) -> int:
        if m == 1 or n == 1:
            return 1
        return self.uniquePaths(m - 1, n) + self.uniquePaths(m, n - 1)
```

## 关键点

- 排列组合原理
- 记忆化递归
- 基本动态规划问题
- 空间复杂度可以进一步优化到 O(n), 这会是一个考点

## 代码

代码支持 JavaScript，Python3, CPP

JavaScript Code:

```javascript
/*
 * @lc app=leetcode id=62 lang=javascript
 *
 * [62] Unique Paths
 *
 * https://leetcode.com/problems/unique-paths/description/
 */
/**
 * @param {number} m
 * @param {number} n
 * @return {number}
 */
var uniquePaths = function (m, n) {
  const dp = Array(n).fill(1);

  for (let i = 1; i < m; i++) {
    for (let j = 1; j < n; j++) {
      dp[j] = dp[j] + dp[j - 1];
    }
  }

  return dp[n - 1];
};
```

Python3 Code:

```python
class Solution:

    def uniquePaths(self, m: int, n: int) -> int:
        dp = [1] * n
        for _ in range(1, m):
            for j in range(1, n):
                dp[j] += dp[j - 1]
        return dp[n - 1]
```

CPP Code:

```cpp
class Solution {
public:
    int uniquePaths(int m, int n) {
        vector<int> dp(n + 1, 0);
        dp[n - 1] = 1;
        for (int i = m - 1; i >= 0; --i) {
            for (int j = n - 1; j >= 0; --j) dp[j] += dp[j + 1];
        }
```

```
        return dp[0];
    }
};
```

**复杂度分析**

- 时间复杂度：$O(M * N)$
- 空间复杂度：$O(N)$

## 扩展

你可以做到比$O(M * N)$更快，比$O(N)$更省内存的算法么？这里有一份资料可供参考。

> 提示： 考虑数学

## 相关题目

- **70. 爬楼梯**
- **63. 不同路径 II**
- **【每日一题】- 2020-09-14 -小兔的棋盘**

上一页 下一页