> 我们正式开启动态规划系列,我们先讲动态规划的理论基础

1、贪心算法与动态规划的区别

动态规划中每一个状态一定是由上一个状态推导出来的

这一点就区分于贪心,贪心没有状态推导，而是从局部直接选最优的。

2、动态规划的五步：

2.1.确定dp数组(dp table)以及下标的含义

2.2.确定递推公式

2.3.dp数组如何初始化

2.4确定遍历顺序

2.5举例推导dp数组

有些时候递推公式会决定dp数组的初始化以及遍历顺序.

3、动态规划的debug最好方式就是把dp数组打印出来,看看究竟是不是按照自己思路推导。

首先要思考自己举例推导状态转移公式了吗？打印dp数组的日志了么？打印出来dp数组和我想的一样么？

70. 爬楼梯

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

注意：给定 n 是一个正整数。

示例 1：
输入：2
输出：2
解释：有两种方法可以爬到楼顶。

1. 1 阶 + 1 阶
2. 2 阶

示例 2：
输入：3
输出：3
解释：有三种方法可以爬到楼顶。

1. 1 阶 + 1 阶 + 1 阶
2. 1 阶 + 2 阶
3. 2 阶 + 1 阶

> 爬楼梯有两种方法一个就是回溯法另一个就是动态规划法
> 1、**确定dp数组下标**及其**含义**
> dp[i]表示是否能到达第i个台阶的方法数.
> 2、**递推公式**
> dp[i]=dp[i-1]+dp[i-2] (i>=2)
> 3、对数组如何初**始化**
> dp[0]=1,dp[1]=1,dp[2]=1
> 4、确定遍历顺序
> 由于该数组只有一维所以用**从左向右遍历**即可实现
> 5、举例说明
> 当i=5时

```
class Solution {
public:
    int climbStairs(int n) {
 vector<int> dp(n+1);
 dp[1]=1;
 dp[2]=2;
```

```
 for(int i=3;i<dp.size();i++){
 dp[i]=dp[i-1]+dp[i-2];
 }
  return dp[n];
     }
};
```

对于**爬楼梯**问题,我们还可以继续深化,就是**一步一个楼梯,两个台阶，三个台阶**,直到**m个台阶，有多少种方法爬到n阶楼顶**.其实就是一个**完全背包**问题.

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

为什么dp[0]初始化为1呢?

746. 使用最小花费爬楼梯

题目链接：https://leetcode-cn.com/problems/min-cost-climbing-stairs/

数组的每个下标作为一个阶梯，第 i 个阶梯对应着一个非负数的体力花费值 cost[i]（下标从 0 开始）。

每当你爬上一个阶梯你都要花费对应的体力值，一旦支付了相应的体力值，你就可以选择向上爬一个阶梯或者爬两个阶梯。

请你找出达到楼层顶部的最低花费。在开始时，你可以选择从下标为 0 或 1 的元素作为初始阶梯。

示例 1：

输入：cost = [10, 15, 20] 输出：15 解释：最低花费是从 cost[1] 开始，然后走两步即可到阶梯顶，一共花费 15 。 示例 2：

输入：cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1] 输出：6 解释：最低花费方式是从 cost[0] 开始，逐个经过那些 1 ，跳过 cost[3] ，一共花费 6 。

cost[i]表示一个非负数的体力值，每当你爬上一个阶梯你都要花费对应的体力值，一旦支付了相应的体力值,你就可以选择向上爬**一个阶梯**或者爬**两个阶梯，最终结果为找到达到楼层顶部的最低花费.**

1、定义dp数组及其含义

dp[i]表示达到第i个阶梯的最低花费

2、递推公式

对于获得dp[i]途径有一个是dp[i-1],一个是dp[i-2]

dp[i]=max(dp[i-1]+cost[i-1],dp[i-2]+cost[i-2])

3、数组的初始化

dp[0]=0;dp[1]=0;

4、遍历顺序:从左向右遍历

5、举例说明

```
class Solution {
public:
    int minCostClimbingStairs(vector<int>& cost) {
 vector<int> dp(cost.size()+3);
 dp[0]=0;
 dp[1]=0;
 for(int i=2;i<=cost.size();i++){
```

```
8   dp[i]=min(dp[i-1]+cost[i-1],dp[i-2]+cost[i-2]);
9   }
10   return dp[cost.size()];
11       }
12  };
```

62.不同路径

题目链接：https://leetcode-cn.com/problems/unique-paths/

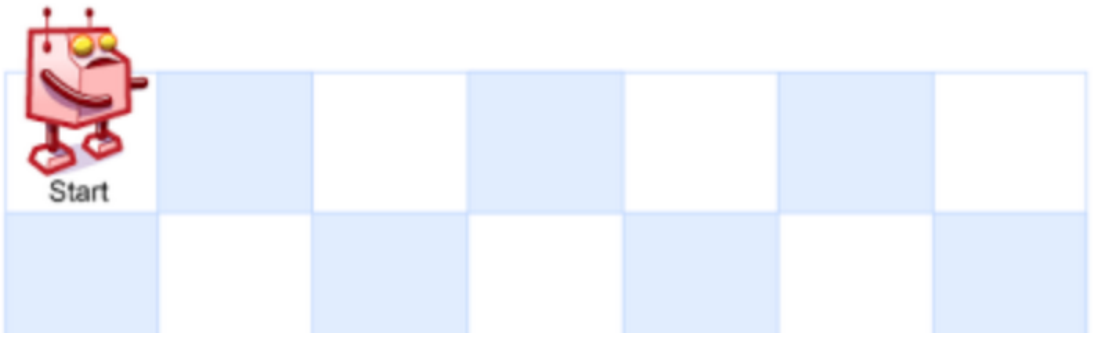
一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start”）。

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：

示例 1：



该题目是二维动态规划

1、dp数组的定义及其下标含义

dp[i][j]表示到达(i,j)表格的路径数

2、递推公式

dp[i][j]=dp[i-1][j]+dp[i][j-1];(i>=1&&j>=1)

3、dp数组初始化

dp[i][0]=1;

dp[0][j]=1;

4、dp数组遍历顺序

从上到下,从左到右

返回值为dp[m-1][n-1];

5、举例说明

重点理解为什么只要用一个一维数组就可以实现，状态压缩?

dp[i]表示第i列的路径数，看过去好像少了行的确定，其实由于遍历顺序是从上到下，从左到右的顺序，那么迭代顺序就是行的确定

递推公式:dp[i]=dp[i]+dp[i-1](如果第j次迭代，那么实际为dp[j][i]=dp[j-1][i]+dp[j][i-1]);

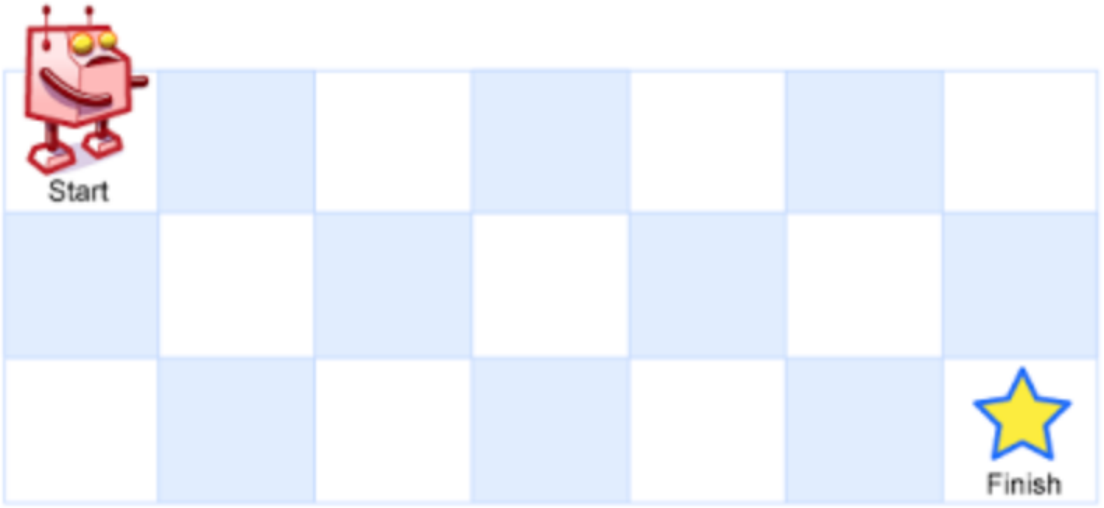
63. 不同路径 II

题目链接：https://leetcode-cn.com/problems/unique-paths-ii/

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”）。

现在考虑网格中有障碍物。那么从左上角到右下角将会有多少条不同的路径？



> 该道题就是上面那道题的基础上改一下数组的初始化和数组的递推公式
> 递推公式:
> 以下公式的前提条件:i>=1&&j>=1
> dp[i][j]=dp[i-1][j]+dp[i][j-1]((i-1, j)和(i, j-1)表格上都没有障碍物)
> dp[i][j]=0((i-1, j)或(i, j-1)表格上存在障碍物)