爬楼梯系列问题

1、基本爬楼问题

力扣题号：70 爬楼梯

题目描述

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

这道题其实也是一个非常基础的斐波那契数，经常被用作递归，递推的入门级题目。现在，我们来研究一下如何使用动态规划来完成本题。

首先，还是先来明确dp数组及下标的含义：
dp[i]代表爬到第i层楼梯共有多少种方法。

其次，确定递推公式。
很容易想到到达最后一层(i)共有两种走法，第一种是从i的前一层跨一步上来，第二种就是从i的前两层跨一步上来。因此，想求到达第i层共有多少方法就转化成为了到达第i-1层和i-2层总共有多少种方法，所以，最终的状态转移方程为：dp[i] = dp[i-1] + dp[i-2]

第三、初始化dp数组。
在这里存在一点点小的争议，当我们处在第0层的时候，dp[i]到底是多少呢？网上有很多争论，有的说是1，有的说是0，各有各的道理。但是，在本题中，实际上dp[0]的意义并不重要，我们更能清楚确定的是dp[1]=1,dp[2]=2。所以，我们不如放弃dp[0]，转而从第三个台阶开始推敲。

第四、确定遍历顺序。
根据递推公式，很明显可以看出，该题是从前向后进行遍历。

第五、距离推导dp数组。
当存在5个台阶时，dp数组的值应该为：1,2,3,5,8

编码

本题与之前的斐波那契数列没有什么本质的差异，除了dp[0]舍弃。另外，本题要从第三个数字开始推导，具体代码如下：

```
#include <bits/stdc++.h>

using namespace std;

class Solution {
public:
    //初始化数组前三位
    //因为dp[0]不参与运算，所以无需关心他的初始值
    int dp[3] = {0, 1, 2};

    int climbStairs(int n) {
        //边界条件
       if (n <= 1) {
            return n;
        }
        //从前向后遍历
        for (int i = 3; i <= n; ++i) {
            //后一个数字等于前两个数字之和
           int sum = dp[1] + dp[2];
            //计算出的结果向前偏移
           dp[1] = dp[2];
            dp[2] = sum;
        }
        return dp[2];
    }
};


int main() {
    Solution s;
    int n;
    cin >> n;
    cout << s.climbStairs(n);
    return 0;
}
```

接下来，我们来看一道稍微复杂一些的题目。

力扣题号：746 使用最小花费爬楼梯

题目描述

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 。

解析

本题的重点还是在于题意的理解，注意，题目中的原文描述是：“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”。

也就是说，我们最后一步到达顶层时候是不需要花费的，你读懂了吗？

我们将示意稍微改动一下，使其更加便于理解。

示例1

10	15	20	楼顶

示例2

1	100	1	1	1	100	1	1	100	1	楼顶

接下来，我们继续按照标准的五步执行：

1、明确dp数组及下标的含义。
dp[i]代表到达第i个台阶所需要花费的最小体力值。

2、确定递推公式
通过前面的分析，我们知道到达第i层的方法来源于第i-1和第i-2层。那么，到底选择i-1还是i-2呢？当然是选择消耗最小的一个。
因此，状态转移方程为dp[i] = min(dp[i-1],dp[i-2])+cost[i]。
为什么这里要加cost[i]呢，因为我们每上一个台阶都将消耗一定的体力值。

3、初始化dp数组

很明显，dp数组在这里不可能完全等同于cost，我们唯一能确定的就是前两层的消耗，因此：dp[0]=cost[0],dp[1]=cost[1]。

4、确定遍历顺序

与普通的爬楼梯一样，这道题是从前向后遍历。

5、举例推导dp数组

我们以输入2为例，模拟dp数组变化，如下图所示：

索引	0	1	2	3	4	5	6	7	8	9
消耗	1	100	2	3	3	103	4	5	104	6

同普通上楼梯一样，因为我最后一步可以选择走一步，也可以选择走两步，所以我需要在最后的两个数值当中选择最小的一个。

编码

```
#include <bits/stdc++.h>

using namespace std;

class Solution {
public:
    int minCostClimbingStairs(vector<int>& cost) {
        int sizeValue = cost.size();
        vector<int> dp(sizeValue);
        //初始化前两层的体力消耗
       dp[0] = cost[0];
        dp[1] = cost[1];
        //从前向后遍历
       for (int i = 2; i < sizeValue; ++i) {
            dp[i] = min(dp[i-1],dp[i-2]) +cost[i];
        }
        //前往最后一层的时候选择消耗最小的。
       return min(dp[sizeValue-1],dp[sizeValue-2]);
    }
};


int main() {
    Solution s;
    vector<int> g = {0, 0, 0, 0};
    cout << s.minCostClimbingStairs(g);
    return 0;
}
```