# 题目

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

类似题目：剑指offer 10-II 青蛙跳台阶问题

# 分析

## 方法一：递归

**思路：**

第n阶的方法数，是由前面的n-1和n-2构成的，自然而言的想到递归。

说明：这种方法会超时。

**代码：**

class Solution {

public:

int climbStairs(int n) {

// 递归

if (n < 0) {

return 0;

} else if (n == 1) {

return 1;

} else if (n == 2) {

return 2;

} else {

return climbStairs(n - 1) + climbStairs(n - 2);

}

}

};

## 方法二：动态规划

**思路：**

每一步结果都是前面1步或者2步，即当前的最优解是基于前面的子问题。

我们可以定义状态 dp[i] 表示走到第 i 级台阶的方法数。根据题目，每次可以上一级或者两级台阶，因此走到第 i 级台阶的方法数应该是走到第 i-1 级台阶的方法数加上走到第 i-2 级台阶的方法数（如果 i-2 大于等于0的话）。

要走到第 i 级台阶，你有两种选择：

从第 i-1 级台阶走一步上来。

从第 i-2 级台阶走两步上来（如果 i-2 大于等于 0，即至少有两级台阶）。

因此，走到第 i 级台阶的方法数就是走到第 i-1 级台阶的方法数和走到第 i-2 级台阶的方法数之和。这是因为每一步的选择都是独立的，所以你可以把每种选择的方法数加起来得到总的方法数。

边界条件是：

dp[0] = 1，因为走到第0级台阶只有1种方法，就是不动。

dp[1] = 1 或 dp[1] = 2，取决于是否允许一次性跨过一级台阶。但根据题目描述，每次至少上一级台阶，所以 dp[1] = 1。

**如下：**

**状态转移方程：**

dp[n]=dp[n−1]+dp[n−2]

**初始值：**

dp[0]=1和dp[1]=1

**代码：**

class Solution {

public:

int climbStairs(int n) {

if (n <= 1) {

return n; // 如果n为0或1，直接返回n

}

// 初始化动态规划数组

std::vector<int> dp(n + 1);

dp[0] = 1; // 到达第0阶的方法数为1（尽管实际中不存在第0阶）

dp[1] = 1; // 到达第1阶的方法数为1

// 填充dp数组（因为dp[0]无意义，这里需要循环到i=n的位置）

for (int i = 2; i <= n; ++i) {

// 到达第i阶的方法数等于到达第i-1阶和第i-2阶的方法数之和

dp[i] = dp[i - 1] + dp[i - 2];

}

// 返回到达楼顶（第n阶）的方法数

return dp[n];

}

};

或：

class Solution {

public:

int climbStairs(int n) {

if (n <= 1) return n; // 因为下面直接对dp[2]操作了，防止空指针

vector<int> dp(n + 1); //因为dp[0]无意义，所以多申请一个内存 dp[1] = 1;

dp[2] = 2;

for (int i = 3; i <= n; i++) { // 注意i是从3开始的

dp[i] = dp[i - 1] + dp[i - 2];

}

return dp[n];

}

};

说明：因为这里的dp[0]其实并没有实际意义，所以推荐这里采用dp[1]的方式。

或：

class Solution {

public:

int climbStairs(int n) {

int p = 0, q = 0, r = 1;

for (int i = 1; i <= n; ++i) {

p = q;

q = r;

r = p + q;

}

return r;

}

};

**或：**

class Solution {

public:

int climbStairs(int n) {

vector<int> v(n+1,1);

for(int i=2;i<=n;i++)

{

v[i] = (v[i-1] + v[i-2]);//是否需要考虑溢出问题

}

return v[n];

}

};

时间复杂度：O(n)

或：

class Solution {

public:

int climbStairs(int n) {

if (n <= 2) return n; // 对于0级和1级台阶，直接返回n

int prev = 1; // dp[i-1]

int curr = 2; // dp[i]

for (int i = 3; i <= n; ++i) {

int next = prev + curr; // dp[i+1] = dp[i] + dp[i-1]

prev = curr;

curr = next;

}

return curr; // 最后curr中存储的是dp[n]的值

}

};