0070. 爬楼梯

ITCharge 大约 3 分钟

- 标签：记忆化搜索、数学、动态规划
- 难度：简单

题目链接

- 0070. 爬楼梯 - 力扣

题目大意

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。现在给定一个整数 n。

要求：计算出有多少种不同的方法可以爬到楼顶。

说明：

- $1 \le n \le 45$。

示例：

- 示例 1：

```py
输入：n = 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：

```py
输入：n = 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
```

解题思路

思路 1：递归（超时）

根据我们的递推三步走策略，写出对应的递归代码。

1. 写出递推公式：$f(n) = f(n-1) + f(n-2)$。
2. 明确终止条件：$f(0) = 0, f(1) = 1$。
3. 翻译为递归代码：
   1. 定义递归函数：`climbStairs(self, n)` 表示输入参数为问题的规模 n，返回结果为爬 n 阶台阶到达楼顶的方案数。
   2. 书写递归主体：`return self.climbStairs(n - 1) + self.climbStairs(n - 2)`。
   3. 明确递归终止条件：
      1. `if n == 0: return 0`
      2. `if n == 1: return 1`

思路 1：代码

```py
class Solution:
    def climbStairs(self, n: int) -> int:
        if n == 1:
            return 1
        if n == 2:
            return 2
        return self.climbStairs(n - 1) + self.climbStairs(n - 2)
```

思路 1：复杂度分析

- **时间复杂度**：$O((\frac{1+\sqrt{5}}{2})^n)$。
- **空间复杂度**：$O(n)$。每次递归的空间复杂度是 $O(1)$，调用栈的深度为 n，所以总的空间复杂度就是 $O(n)$。

思路 2：动态规划

1. 划分阶段

按照台阶的层数进行划分为 $0 \sim n$。

2. 定义状态

定义状态 $dp[i]$ 为：爬到第 i 阶台阶的方案数。

3. 状态转移方程

根据题目大意，每次只能爬 1 或 2 个台阶。则第 i 阶楼梯只能从第 $i-1$ 阶向上爬 1 阶上来，或者从第 $i-2$ 阶向上爬 2 阶上来。所以可以推出状态转移方程为 $dp[i] = dp[i-1] + dp[i-2]$。

4. 初始条件

- 第 0 层台阶方案数：可以看做 1 种方法（从 0 阶向上爬 0 阶），即 $dp[0] = 1$。
- 第 1 层台阶方案数：1 种方法（从 0 阶向上爬 1 阶），即 $dp[1] = 1$。
- 第 2 层台阶方案数：2 种方法（从 0 阶向上爬 2 阶，或者从 1 阶向上爬 1 阶）。

5. 最终结果

根据状态定义，最终结果为 $dp[n]$，即爬到第 n 阶台阶（即楼顶）的方案数为 $dp[n]$。

思路 2：代码

```py
class Solution:
    def climbStairs(self, n: int) -> int:
        dp = [0 for _ in range(n + 1)]
        dp[0] = 1
        dp[1] = 1
```

```python
        for i in range(2, n + 1):
            dp[i] = dp[i - 1] + dp[i - 2]

        return dp[n]
```

思路 2：复杂度分析

- **时间复杂度**：$O(n)$。一重循环遍历的时间复杂度为 $O(n)$。
- **空间复杂度**：$O(n)$。用到了一维数组保存状态，所以总体空间复杂度为 $O(n)$。因为 $dp[i]$ 的状态只依赖于 $dp[i - 1]$ 和 $dp[i - 2]$，所以可以使用 3 个变量来分别表示 $dp[i]$、$dp[i - 1]$、$dp[i - 2]$，从而将空间复杂度优化到 $O(1)$。