简

最小子问题

费波那契

简单 836,609 提交 | 433,992 通过

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
```

⇒ 找最近子问题

思路：参数时：

1、暴力枚举

2、此题虽然明显无法枚举完，

但可以先 $n=1,2,3\cdots$ 看看，数学归纳法

$n=1$：直接上1阶 → 1

$n=2$：直接2阶
1阶+1阶 → 2

$n=3$：1阶+2阶
2阶+1阶
1阶+1阶+1阶 → 3

$n=4$：1阶+3阶
2阶+2阶 → 3+2=5

1阶+3阶：定下第一步的1阶，看3阶可参考 $f(3)$

2阶+2阶：定下第一步2阶，看2阶可参考 $f(2)$

那可发现 $f(n)=f(n-1)+f(n-2)$

面试题10- I. 斐波那契数列（动态规划，清晰图解） 精选

Krahets L6
2020-02-19

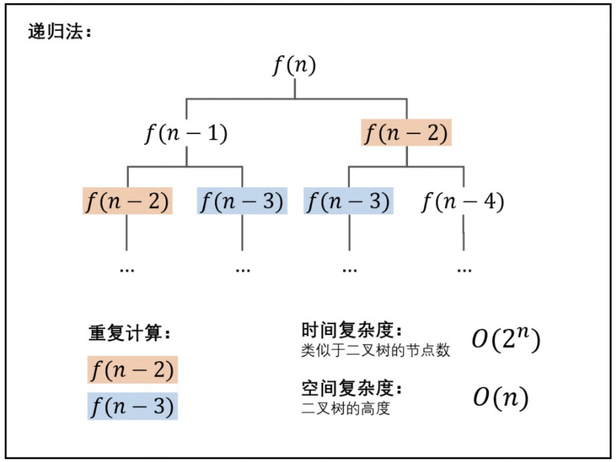
动态规划 Java Python3

解题思路：

斐波那契数列的定义是 $f(n+1) = f(n) + f(n-1)$ ，生成第 n 项的做法有以下几种：

1. **递归法：**
   - **原理：** 把 $f(n)$ 问题的计算拆分成 $f(n-1)$ 和 $f(n-2)$ 两个子问题的计算，并递归，以 $f(0)$ 和 $f(1)$ 为终止条件。
   - **缺点：** 大量重复的递归计算，例如 $f(n)$ 和 $f(n-1)$ 两者向下递归需要 **各自计算** $f(n-2)$ 的值。
2. **记忆化递归法：**
   - **原理：** 在递归法的基础上，新建一个长度为 n 的数组，用于在递归时存储 $f(0)$ 至 $f(n)$ 的数字值，重复遇到某数字则直接从数组取用，避免了重复的递归计算。
   - **缺点：** 记忆化存储需要使用 $O(N)$ 的额外空间。
3. **动态规划：**
   - **原理：** 以斐波那契数列性质 $f(n+1) = f(n) + f(n-1)$ 为转移方程。
   - 从计算效率、空间复杂度上看，动态规划是本题的最佳解法。

> 下图帮助理解递归法的 “重复计算” 概念。



动态规划解析：

- **状态定义：** 设 dp 为一维数组，其中 $dp[i]$ 的值代表 斐波那契数列第 i 个数字 。
- **转移方程：** $dp[i+1] = dp[i] + dp[i-1]$ ，即对应数列定义 $f(n+1) = f(n) + f(n-1)$ ；
- **初始状态：** $dp[0] = 0, dp[1] = 1$ ，即初始化前两个数字；
- **返回值：** $dp[n]$ ，即斐波那契数列的第 n 个数字。

空间复杂度优化：

> 若新建长度为 n 的 dp 列表，则空间复杂度为 $O(N)$ 。

- 由于 dp 列表第 i 项只与第 $i-1$ 和第 $i-2$ 项有关，因此只需要初始化三个整形变量 `sum` , `a` , `b` ，利用辅助变量 sum 使 a, b 两数字交替前进即可 *（具体实现见代码）* 。
- 节省了 dp 列表空间，因此空间复杂度降至 $O(1)$ 。

循环求余法：

> **大数越界：** 随着 n 增大, $f(n)$ 会超过 `Int32` 甚至 `Int64` 的取值范围，导致最终的返回值错误。

- **求余运算规则：** 设正整数 x, y, p ，求余符号为 $\odot$ ，则有 $(x+y) \odot p = (x \odot p + y \odot p) \odot p$ 。
- **解析：** 根据以上规则，可推出 $f(n) \odot p = [f(n-1) \odot p + f(n-2) \odot p] \odot p$ ，从而可以在循环过程中每次计算 $sum = (a+b) \odot 1000000007$ ，此操作与最终返回前取余等价。

> 图解基于 Java 代码绘制，Python 由于语言特性可以省去 sum 辅助变量和大数越界处理。