打家劫舍问题：动态规划的解题步骤

原创 赵彦庆 LOA算法学习笔记 2021-02-22 20:09

01 什么是动态规划的问题?

动态规划就是把一个大问题一步步降解成越来越小的子问题，直到子问题小到可以用确定的条件来解答，即按照顺序，从基元问题一步步扩大问题的规模，直到问题的规模覆盖了我要求解的问题。每一个规模的问题的解叫做一个状态，每个不同规模的问题的解的关系叫做状态转移方程，实际应用中无非就是利用历史记录，来避免我们的重复计算。而这些历史记录，我们得需要一些变量来保存，一般是用一维数组或者二维数组来保存，综上所述的动态规划问题其实可以简化成为三个步骤，用于解决大部分的动态规划问题。

本期例题：LeetCode 198. House Robber 打家劫舍（Easy）

你是一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。

给定一个代表每个房屋存放金额的非负整数数组，计算你在不触动警报装置的情况下，能够偷窃到的最高金额。

示例：

输入: [1,2,3,1]

输出: 4

解释: 偷窃 1 号房屋 (金额 = 1)，然后偷窃 3 号房屋 (金额 = 3)。

偷窃到的最高金额 = 1 + 3 = 4 。

02 总结如何使用动态规划的解题步骤?

动态规划的的三个解题步骤是：

- 定义子问题
- 写出子问题的递推关系
- 确定 DP 数组的计算顺序

步骤一：定义子问题

子问题是和原问题相似，但规模较小的问题。例如这道打家劫舍问题，原问题是「从全部房子中能偷到的最大金额」，将问题的规模缩小，子问题就是「从 k个房子中能偷到的最大金额」，用 f(k)表示。

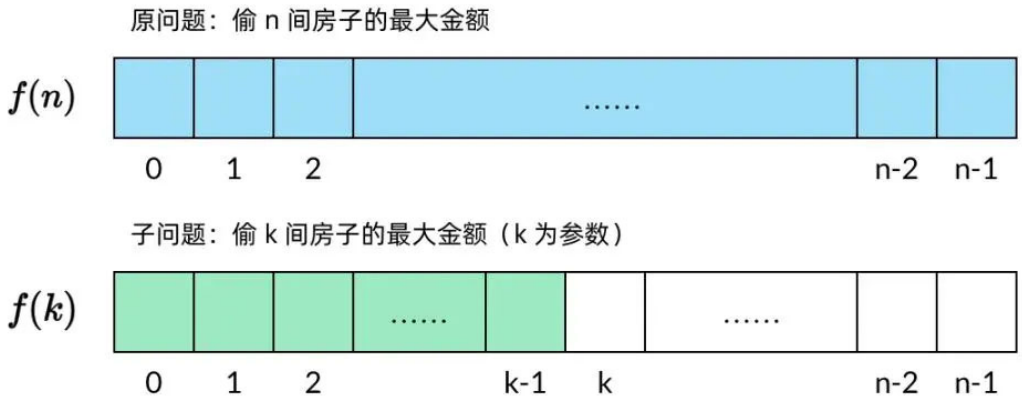

图1 打家劫舍问题的子问题定义

可以看到，子问题是参数化的，我们定义的子问题中有参数k 。假设一共有 n个房子的话，就一共有 n个子问题。动态规划实际上就是通过求这一堆子问题的解，来求出原问题的解。这要求子问题需要具备两个性质：

- **原问题要能由子问题表示。**例如这道题中， k=n时实际上就是原问题。否则，解了半天子问题还是解不出原问题，那子问题岂不是白解了。

- **一个子问题的解要能通过其他子问题的解求出。**例如这道题中， f(k)可以由f(k-1)和 f(k-2)求出，具体原理后面会解释。这个性质就是教科书中所说的「最优子结构」。如果定义不出这样的子问题，那么这道题实际上没法用动态规划解。

步骤二：写出子问题的递推关系

这一步是求解动态规划问题最关键的一步。然而，这一步也是最无法在代码中体现出来的一步,我们来分析一下这道题的递推关系：

假设一共有 n个房子，每个房子的金额分别是 $H_0, H_1, \ldots\ldots\ldots H_{n-1}$ ，子问题 f(k)表示偷前 k个房子（即 $H_0, H_1, \ldots\ldots\ldots H_{k-1}$ ）中能偷到的最大金额。那么，偷 k 个房子有两种偷法：

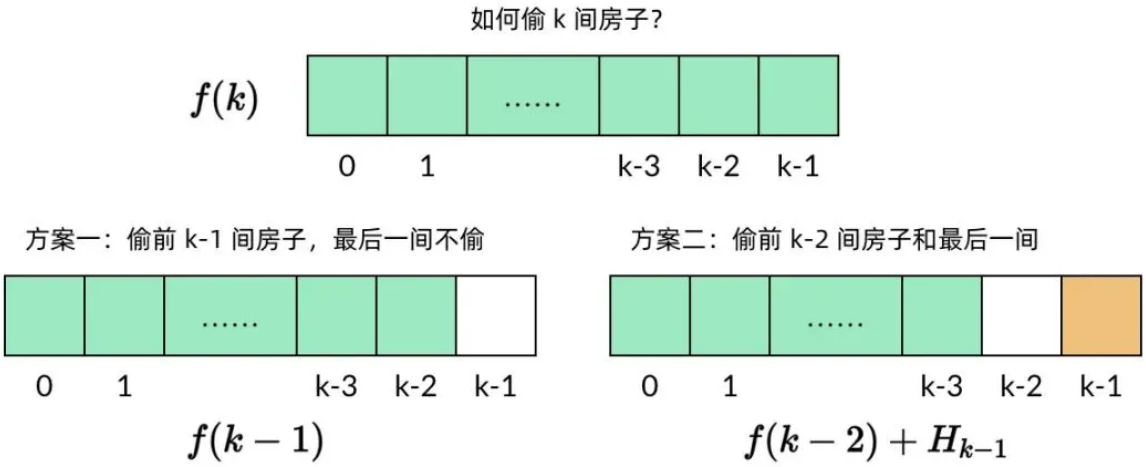

图2 子问题的递推关系

k个房子中最后一个房子是H_{k-1} 。如果不偷这个房子，那么问题就变成在前 k-1个房子中偷到最大的金额，也就是子问题 f(k-1)。如果偷这个房子，那么前一个房子 H_{k-2}显然不能偷，其他房子不受影响。那么问题就变成在前 k-1个房子中偷到的最大的金额。两种情况中，选择金额较大的一种结果。

$$f(k) = \max\begin{cases} f(k-1) \\ f(k-2) + H_{k-1} \end{cases}$$

另外，我们在写递推关系的时候，还要注意递推关系的 base case。这样才能构成完整的递推关系，后面写代码也不容易在边界条件上出错。在这道题中，是k=0 和 k=1时的子问题：

- 当 k=0 时，没有房子，所以 f(0)=0。
- 当k=1时，只有一个房子，偷这个房子即可，所以 f(0)=H 。

步骤三：确定 DP 数组的计算顺序

在确定了子问题的递推关系之后，下一步就是依次计算出这些子问题了。

DP 数组也可以叫「子问题数组」，因为 **DP 数组中的每一个元素都对应一个子问题。**如下图所示，dp[k] 对应子问题 f(k)，即偷前 k 间房子的最大金额。

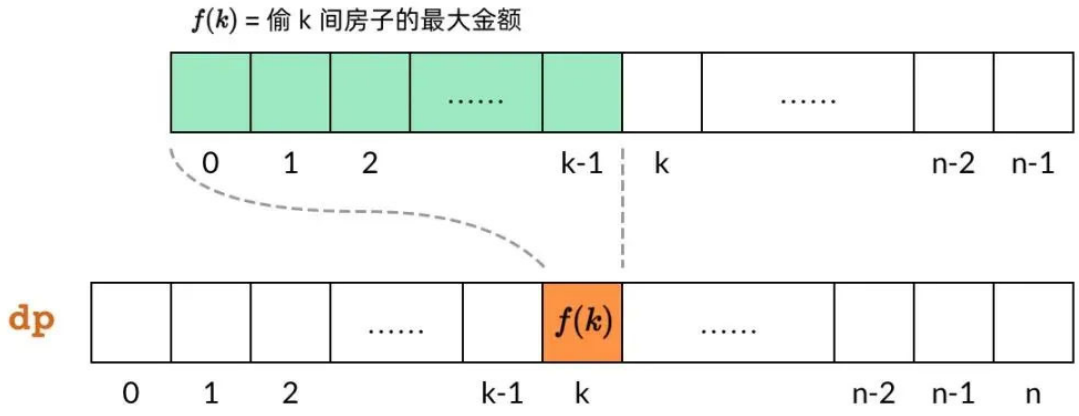

图3 DP 数组与子问题的对应关系

那么，只要搞清楚了子问题的计算顺序，就可以确定 DP 数组的计算顺序。对于打家劫舍问题，我们分析子问题的依赖关系，发现每个 f(k)依赖f(k-1) 和 f(k-2)。也就是说，dp[k] 依赖 dp[k-1] 和 dp[k-2]，如下图所示。

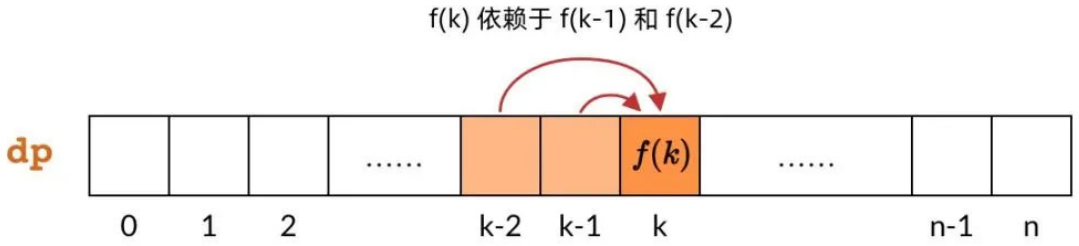

图4 DP 数组的依赖顺序

那么，既然 DP 数组中的依赖关系都是向右指的，DP 数组的计算顺序就是从左向右。这样我们可以保证，计算一个子问题的时候，它所依赖的那些子问题已经计算出来了。

确定了 DP 数组的计算顺序之后，我们就可以写出题解代码了：

```
public int rob(int[] nums) {
    if (nums.length == 0) {
        return 0;
    }
    // 子问题：
    // f(k) = 偷 [0..k) 房间中的最大金额

    // f(0) = 0
    // f(1) = nums[0]
    // f(k) = max{ rob(k-1), nums[k-1] + rob(k-2) }

    int N = nums.length;
    int[] dp = new int[N+1];
    dp[0] = 0;
    dp[1] = nums[0];
    for (int k = 2; k <= N; k++) {
        // 套用子问题的递推关系
        dp[k] = Math.max(dp[k-1], nums[k-1] + dp[k-2]);
    }
    return dp[N];
}
```

03 总结

本文用打家劫舍问题展示了动态规划问题的基本解题三步骤。动态规划问题种类繁多，有些题目非常有难度，但这些问题千变万化也离不开这三个基本解题步骤，只不过是把三步骤的某些步骤变得更难了。例如：

- 在「定义子问题」的步骤，有的题目需要定义二维、三维的子问题。
- 在「子问题递推关系」的步骤，有的题目需要分情况讨论，有复杂的 max、min、sum 表达式。
- 在「DP 数组计算顺序」的步骤，有的题目需要反向计算，甚至斜向计算。