动态规划之打家劫舍问题的思考

原创 黄谋潇 LOA算法学习笔记 2021-01-23 22:42

01 问题描述

1.1 打家劫舍原问题

一个劫匪计划在一条街上偷窃该路上沿街的房屋。每一间房屋都有一定的现金，但是因为相邻的房屋有防盗系统，若**相邻房屋被偷窃的话就会自动触发警报系统报警**，所以劫匪**不能同时偷窃两间相邻的房屋。**

给定一个非负整数的数组表示每一间房屋藏有的现金，计算**在不触发警报系统的情况下所能偷窃到的最大金额。**

1.2 打家劫舍拓展问题

一个劫匪计划在一条街上偷窃该路上沿街的房屋，这条街上的**房屋连接成环状。**每一间房屋都有一定的现金，但是因为相邻的房屋有防盗系统，若**相邻房屋被偷窃的话就会自动触发警报系统报警**，所以劫匪**不能同时偷窃两间相邻的房屋。**

给定一个非负整数的数组表示每一间房屋藏有的现金，计算**在不触发警报系统的情况下所能偷窃到的最大金额。**

02 打家劫舍原问题

2.1 问题分析

本题问题可总结为"在一条街上，抢劫两两不相邻的房屋，使得所能获取的金额最大化"。若只有一间房屋，那就抢劫这一间；若有两间房屋，那就抢劫其中金额较大的一间；若有三间房屋及以上，那就需要求出各决策情况下所能获取的金额数目，然后选择金额数较大的方案。

这是十分典型的多步决策的优化问题，因此可以构造最优子结构 $OPT[i]$ 表示所能抢劫到的最大金额， $M[i]$ 表示第 i 间房屋藏有的现金数。故各情况可分析如下：

1) 当只有一间房屋，则抢劫这一间， $OPT[1] = M[i]$ 。

2) 当只有两间房屋，则抢劫其中金额数较大的一间， $OPT[2] = max(M[1], M[2])$ 。

3) 当有三间房屋及以上时，由于不能抢劫相邻的两间，所以有如下情况：

- 如果抢劫了第 i 间房屋，则不能抢劫第 $i-1$ 间，故最大金额是第 i 间房屋和前 $i-2$ 间房屋的最大金额之和；
- 如果不抢第 i 间房屋，则最大金额为前 $i-1$ 间房屋的最大金额。

故 $OPT[i] = max(OPT[i-2] + M[i], OPT[i-1])$ 。

2.2 状态转移方程

根据对问题的分析，可以总结地列出动态规划状态转移方程，如下：

$$OPT[i] = \begin{cases} M[i] & i = 1 \\ max(M[i], M[i-1]) & i = 2 \\ max(OPT[i-2] + M[i], OPT[i-1]) & i > 2 \end{cases}$$

2.3 正确性证明

- 对只有一间和两间房屋的情况，很显然，满足上述的状态转移方程。

- 当有三间房屋及以上时，1）抢第 i 间，则不能抢第 $i-1$ 间，最大金额为前 $i-2$ 间最大金额和第 i 间之和；2）不抢第 i 间，则最大金额为前 $i-1$ 间最大金额；选取这两种情况的最大金额更大的方案，即对应动态方程 $i>2$ 的情况，满足状态转移方程，证毕。

2.4 代码实现

```
int MoneyRobbing(vector<int>& M){
    int N = M.size();
    if(N==0){
      return 0;
    }
    if(N==1){
      return M[0];
    }
    vector<int> OPT = vector<int>(N, 0);
    OPT[0] = M[0];
    OPT[1] = max(M[0], M[1]);
    for(int i=2; i<N; i++){
      OPT[i] = max(OPT[i-2]+M[i], OPT[i-1]);
    }
    return OPT[N-1];
  }
```

03 打家劫舍拓展问题一

3.1 问题分析

本题问题和打家劫舍原问题唯一的不同就在于房屋排列不再是单排排列的顺序了，而是连接成环状、首尾相连。自然地，因为成环了不会做，那就会有将此问题和原问题联系起来的想法，考虑如何将其分解或降低难度为打家劫舍原问题，然后就可以做了。

我们可以人为地规定这个环状结构的起点和终点，进而编号排序，也就是说第一间房屋和最后一间房屋不能同时偷窃，只能够择其一来偷窃，所以问题可以分解为如下两个子问题：

- 不偷窃第一间房屋时（即去除第一个数组元素， $M[1:]$ ），最大金额为 $Money1$ ；
- 不偷窃最后一间房屋时（即去除最后一个数组元素， $M[:n-1]$ ），最大金额为 $Money2$ 。

故，问题变为取这两种子问题情况的最大金额，即 $max(Money1, Money2)$ 。

3.2 状态转移方程

根据以上的分析，此题就变成了典型的动态规划题目，转换为两个打家劫舍的原问题。因此，构造最优子结构 $OPT[i]$ 表示所能抢劫到的最大金额， $M[i]$ 表示第 i 间房屋藏有的现金数，同样地，最终的动态规划状态转移方程为：

$$OPT[i]=max(OPT[i-2]+M[i], OPT[i-1])$$

3.3 正确性证明

成环状排列就意味着在人为指定起始点后，第一间房屋和最后一间房屋只能够选择其中一个进行偷窃，所以显而易见地将其分解为两个原问题进行求解。1）不偷窃第一间房屋情况下，得到最大金额为 $Money1$ ； 2）不偷窃最后一间房屋情况下，得到最大金额为 $Money2$ 。最后取两种情况下的最大值即为该问题的最大金额。且每种情况，即两种子问题都是和打家劫舍原问题相同，容易证得满足上述动态规划状态转移方程。

3.4 代码实现

```
int MoneyRobbing2(vector<int>& M){
    int N = M.size();
    if(N==1){
      return M[0];
    }
    int Money1, Money2;
    Money1 = MoneyRobbingRange(M, 0, N-2);
    Money2 = MoneyRobbingRange(M, 1, N-1);
    return max(Money1, Money2);
  }
  int MoneyRobbingRange(vector<int>& M, int start, int end){
    int n = M.size();
    int OPTi = 0;
    int OPTi_1 = 0, OPTi_2 = 0;
    for(int i=end; i>=start; i--){
      OPTi = max(OPTi_1, M[i]+OPTi_2);
      OPTi_2 = OPTi_1;
      OPTi_1 = OPTi;
    }
    return OPTi;
  }
```

04 总结

动态规划是多步决策最优化问题，其关键在于寻找最优子结构。一般地，对于一个优化问题，可以考虑求解该问题是否可以描述为多步决策，如果可以则可以考虑使用动态规划方法，去构造问题的最优子结构。先看第一步决策，看简单的情况，再考察所有的决策项，根据子问题总结，得到子问题的一般形式。本人算法水平十分有限，有任何错误或需改进的地方，还请大家不吝赐教，谢谢！