算法设计与分析作业 2

杨树鑫 2019E8013261016

November 13th , 2019

本次作业选做 1、2、3 题。

1 第 1 题 Money Robbing

1.1 最优子结构和 DP 方程

用 $OPT(i)$ 表示对前 i 个房屋抢劫得到的最优解，$value(i)$ 表示第 i 个房屋能抢到的金额，当这条街上的房屋数量为 n 的时候，即求 $OPT(n)$, 将其转换为多步决策，在第 i 步的时候，只考虑 $1, 2, ..., i$ 的房屋，从后往前考虑

1、当选择第 i 间房屋的时候，就不能选择第 $i-1$ 间房屋，因此问题转换为 $OPT(i-2) + value(i)$

2、当不选择第 i 间房屋的时候，就能选择第 $i-1$ 间房屋，因此问题转换为 $OPT(i-1)$

最优子结构就是在上述两种情况中选择解最大的决策

因此 DP 方程可写为

$$OPT(i) = max\{OPT(i-1), OPT(i-2) + value(i)\}$$

1.2 伪代码

伪代码如算法 1 所示

1.3 算法正确性

这个抢劫问题可以看做是多步决策过程，每一步都是选或者不选的决策，第一步首先考虑最后一间房屋

算法 1 房屋抢劫

输入: $Array$ 房屋价值数组

输出: 最大抢劫值

```
function Rob(Array)
    for i = 0 to n + 2 do
        OPT[i] = 0
    end for
    for i = 2 to n + 1 do
        OPT[i] = max{OPT[i − 1], OPT[i − 2] + value[i − 2]}
    end for
    return OPT[n]
end function
```

如果这个房屋选，则不能选与它相邻的第 $i-1$ 间，因此只能在前 $i-2$ 间房屋中再做决策，问题的解就由前 $i-2$ 间房屋的最优决策加上第 i 间房屋的价值组成

如果这个房屋不选，则可以在前 $i-1$ 间房屋中进行决策，问题的解就直接由前 $i-1$ 间房屋的最优决策组成

每个子问题的结构相同，因此对每个子问题进行求解，就可以得到原问题的解

上述算法与此分析相同，因此算法正确

1.4 算法复杂度

原问题有 n 个子问题，每个子问题进行 2 次比较，因此共有 $O(2n)$ 次运算，所以原问题的时间复杂度为

$$T(n)=O(n)$$

由于开了一个 $n+2$ 的数组存储子问题的最优解，因此原问题的空间复杂度为

$$O(n)$$

1.5 附加问题：2、当房屋存在循环的问题

当存在循环时，实际上是增加了一个条件，即第一个房屋和最后一个房屋至多只能选择一个，因此可以将问题分成两个，其中一个是 $\{1, 2, ..., n-1\}$ 房屋，另一个是 $\{2, ..., n\}$ 房屋，对这两个问题分别按上面的算法进行求解，其中的最大值即原问题的解

2 第 2 题 Node Selection

2.1 最优子结构和 DP 方程

用 $OPT(root)$ 表示 $root$ 节点的最优解，$value(root)$ 表示 $root$ 节点的价值，$OPT(root)[1]$ 表示选择了 $root$ 节点时得到的最优解，$OPT(root)[0]$ 表示不选择根节点时得到的最优解，则存在两种选择的可能：

1、选择 $root$ 节点的时候，就不能选择其子节点，因此问题转换为求 $OPT(root->left)[0] + OPT(root->right)[0] + value(root)$

2、不选择 $root$ 节点的时候，就能选择其子节点，因此问题转换为求 $max\{OPT(root->left)[0], OPT(root->left)[1]\} + max\{OPT(root->right)[0], OPT(root->right)[1]\}$

最优子结构就是在上述两种情况中选择解最大的决策

因此 DP 方程可写为

$$OPT(i) = max\{OPT(root->left)[0] + OPT(root->right)[0] + value(root), max\{OPT(root->left)[0], OPT(root->left)[1]\} + max\{OPT(root->right)[0], OPT(root->right)[1]\}\}$$

2.2 伪代码

伪代码如算法 2 所示

2.3 算法正确性

这个二叉树节点选择的问题可以看做是多步决策过程，每一步都是选或者不选根节点的决策，只是在做出根节点选择决策后，下一步决策与根节点的决策相关，第一步首先考虑根节点的选择问题

算法 2 二叉树选择

输入: $root$ 根节点

输出: $res[2]$ 选择和不选择 $root$ 节点时分别得到的最优解

function SelectNodes($root$)
  if $root == NULL$ **then**
    return $\{0, 0\}$
  end if
  $left \leftarrow$ SelectNodes($root- > left$)
  $right \leftarrow$ SelectNodes($root- > right$)
  $res[0] \leftarrow (max\{left[0], left[1]\} + max\{right[0], right[1]\})$
  $res[1] \leftarrow (left[0] + right[0] + root- > value)$
  return res
end function

如果根节点选，则不能选择它的两个子节点，因此只能在它的孙子节点中再做决策，问题的解就由左子节点的两个子节点的最优决策加上右子节点的两个子节点的最优决策组成

如果这个房屋不选，则可以在它的两个子节点中进行决策，问题的解就直接由左子节点的最优决策、右子节点的最优决策和它本身的值组成

每个子问题的结构相同，因此对每个子问题进行求解，就可以得到原问题的解, 上述算法与此分析相同，因此算法正确

2.4 算法复杂度

原问题有 n 个子问题，每个子问题进行 3 次比较，因此共有 $O(3n)$ 次运算，所以原问题的时间复杂度为

$$T(n)=O(n)$$

由于采用了递归，最优的递归深度为 $logn$, 但如果二叉树极度不平衡，成为斜二叉树时，最深的递归深度为 n, 每次递归中开了一个大小为 2 的数组存储子问题的最优解，因此原问题的空间复杂度为

$$\text{最优：} O(logn) \ \text{最坏：} O(n)$$

3 第 3 题 Unique Binary Search Trees

3.1 最优子结构和 DP 方程

因为这个二叉搜索树的每个元素都是唯一的，因此可以用 $OPT(i)$ 表示由 i 个元素组成的二叉搜索树的数量，所以原问题即求解 $OPT(n)$

原问题中的每个元素都可以作为根节点，原问题可以转换为更小的子问题，即求以每个元素为根节点时能组成的二叉搜索树的数量的和

以 i 表示当前元素，可以从 i 将这 n 个元素分为两部分，左边有 $i-1$ 个元素，右边有 $n-i$ 个元素, 子问题的解即左右两边元素的笛卡尔积

因此 DP 方程可写为

$$OPT(n) = sum(OPT(i-1) * OPT(n-i))\ i = 1, 2, ...n$$

3.2 伪代码

伪代码如算法 3 所示

算法 3 二叉树的数量

输入: n 二叉搜索树的元素数量

输出: res 可以组成的二叉搜索树数量

```
function uBST(n)
    OPT(0) ← 1
    OPT(1) ← 1
    if n < 2 then
        return OPT[n]
    end if
    for i = 2 to n do
        num ← 0
        for j = 1 to n do
            num ← OPT[j − 1] + OPT[i − j]
        end forOPT[i] ← num
    end for
    return OPT[n]
end function
```

3.3 算法正确性

这个二叉搜索树数量的问题可以看做是多步决策过程，每一步是以给定的所有元素中的一个为分界点，将原问题分解为两个相同的子问题

子问题是将这 n 个元素分为两部分，然后求解左边 $i-1$ 个元素能组成的数量，和右边 $n-i$ 个元素能组成的数量的笛卡尔积

每个子问题的结构相同，因此对每个子问题进行求解，就可以得到原问题的解上述算法与此分析相同，因此算法正确

3.4 算法复杂度

原问题有 n 个子问题，每个子问题进行 n 次计算，因此共有 $O(n^2)$ 次运算，所以原问题的时间复杂度为

$$T(n){=}O(n^2)$$

由于开了一个 $n+1$ 的数组存储子问题的最优解，因此原问题的空间复杂度为

$$O(n)$$