Algorithm Assignment 2

201828018670050 王纪宝

2018 年 11 月 1 日

目录

1 Problem 1: Money robbing **1**
1.1 The Optimal Substructure and DP Equation . . . 1
1.2 pseudo-code . . . 1
1.3 Prove the Correctness . . . 2
1.4 The complexity of this algorithm . . . 2

2 Problem 2: Node selection **3**
2.1 The Optimal Substructure and DP Equation . . . 3
2.2 Pseudo-code . . . 3
2.3 Prove the Correctness . . . 4
2.4 The complexity of this algorithm . . . 4

3 Problem 3: Decoding **4**
3.1 The Optimal Substructure and DP Equation . . . 4
3.2 Pseudo-code . . . 4
3.3 Prove the Correctness . . . 5
3.4 The complexity of this algorithm . . . 5

1 Problem 1: Money robbing

1.1 The Optimal Substructure and DP Equation

对于第 i 家，有两种选择，抢或者不抢。如果抢第 i 家，那么最大收益为：value[i] + 抢前 i-2 家的最优收益；如果第 i 家不抢，那么最大收益为：抢前 i-1 家的最优收益。

状态转移方程可写为: $dp[i] = max(dp[i-1], dp[i-2] + value[i])$

1.2 pseudo-code

LeetCode 198 题

```
class Solution {
public:
    int rob(vector<int>& nums){
```

```
        if( nums.size()==0 )
            return 0;
        else if( nums.size()==1 )
            return nums[0];
        vector<int> dp(nums.size(),0);
        dp[0] = nums[0];
        dp[1] = max( nums[0], nums[1] );
        for( int i=2; i<nums.size(); i++ ){
            dp[i] = max( dp[i-1], dp[i-2]+nums[i] );
        }
        return dp[nums.size()-1];
    }
};
```

1.3 Prove the Correctness

如果暴力求解的话，每一家都有可能抢或者不抢，枚举每一种可能性，那么总的时间复杂度为 $O(2^n)$，且最后算出来的最优解一定是正确的。现在，我们采用动态规划思想中的 memory 数组，记录前 i-1 家的最优值，当计算第 i 家最优值时，其依赖的 OPT[i-1] 和 OPT[i-2] 已经求出，可直接计算当前最优值，故算法是正确的。

1.4 The complexity of this algorithm

代码中的 dp 数组，每个元素只计算了一遍，故算法的时间复杂度为 $O(n)$。

注：

当房屋变成一个环的时候，也就是相当于在原有问题上添加了一个限制条件：第 ‘1‘家和第 ‘n‘家不能同时抢。那么，分别计算抢第二家到最后一家与抢第一家到倒数第二家的最大值，取两个值中更大的那个就是结果。

LeetCode 213 题

```
class Solution {
public:
    int rob(vector<int>& nums) {
        if( nums.size()==0 )
            return 0;
        else if( nums.size()==1 )
            return nums[0];
        return max( func(nums, 0), func(nums, nums.size()-1 ));
    }
    int func( vector<int> nums, int pos ){
        nums.erase(nums.begin()+pos);
        if( nums.size()==0 )
            return 0;
        else if( nums.size()==1 )
            return nums[0];
```

```
            vector<int> dp(nums.size(),0);
            dp[0] = nums[0];
            dp[1] = max( nums[0], nums[1] );
            for( int i=2; i<nums.size(); i++ ){
                dp[i] = max( dp[i-1], dp[i-2]+nums[i] );
            }
            return dp[nums.size()-1];
        }
    };
```

2 Problem 2: Node selection

2.1 The Optimal Substructure and DP Equation

对于当前根节点，有两种可能性，选取或者不选取。如果选取了当前的根节点，那么其孩子节点就不能选取，即最优值变为：root->value + 4 个孙子的最优值之和；如果当前根节点没有选取，最优值变为：两个孩子节点最优值之和。

给定任意一个根，假设我们能够得到当前根选取与不选取的最优值，结果保存在一个长度为 2 的数组里，array[0] 代表根节点不选取，array[1] 代表根节点选取，那么最优解为：

//根节点不选

$$opt\_ans[0] = max(left\_ans[0], left\_ans[1]) + max(right\_ans[0], right\_ans[1])$$

//选择根节点

$$opt\_ans[1] = root \to val + left\_ans[0] + right\_ans[0]$$

$$ANS = max(opt\_ans[0], opt\_ans[1])$$

2.2 Pseudo-code

LeetCode 337 题

```
    class Solution {
    public:
        int rob(TreeNode* root) {
            vector<int> ans = solve( root );
            return max( ans[0], ans[1] );
        }
        vector<int> solve( TreeNode* root ){
            vector<int> ans(2,0);
            if( !root )
                return ans;
            vector<int> left_ans = solve( root->left);
            vector<int> right_ans = solve( root->right );
            //根节点不选
            ans[0] = max( left_ans[0], left_ans[1] ) + max( right_ans[0], right_ans[1] );
            //选择根节点
```

```
            ans[1] = root->val + left_ans[0] + right_ans[0];
            return ans;
        }
    };
```

2.3 Prove the Correctness

当前节点的最优值，依赖于其儿子节点的最优值（当前节点没选，最优值等于选其左孩子节点最优值 + 选其右孩子节点最优值）或其孙子节点的最优值（选取了当前节点，最优值等于根节点值 + 其左孩子不选左根的最优值 + 右孩子不选右根的最优值），由此可知算法的正确性。

2.4 The complexity of this algorithm

整个算法相当于一个深度优先搜索（DFS），树中的每个节点都会遍历一次，故算法的时间复杂度为 $O(n)$。

3 Problem 3: Decoding

3.1 The Optimal Substructure and DP Equation

dp[i] 表示从字符 0–i 的字符串包含最多的编码种数。不考虑特殊情况，该题的递推式是 $dp[i] = dp[i-1] + dp[i-2]$，因为一个数字可以表示一个编码，两个数字也有可能表示一个编码，所以 dp[i] 应该等于 0–i-1 的字符串包含的最多编码种数加上 0–i-2 的字符串包含的最多编码种数。但是考虑到一共只有 26 种基础编码加上特殊情况 0，所以递推式可以表示为：

$$dp[i] = \begin{cases} 1 & i == 0 \\ 1 & i == 1 \;\&\&\; (s[1] ==' 0' \;||\; (s[0] -' 0') * 10 + (s[1] -' 0') > 26) \\ 2 & i == 1 \;\&\&\; (s[1]! =' 0' \;\&\&\; (s[0] -' 0') * 10 + (s[1] -' 0') <= 26) \\ dp[i-1] & s[i-1] ==' 0' \;||\; (s[i-1] -' 0') * 10 + (s[i] -' 0') > 26 \\ dp[i-2] & s[i] ==' 0' \\ dp[i-1] + dp[i-2] & \text{其他情况} \end{cases}$$

3.2 Pseudo-code

LeetCode 91 题

```
class Solution {
public:
    int numDecodings(string s) {
        if( s.length()==0 || s[0]=='0' ) return 0;
        vector<int> dp(s.length()+1,0);
        dp[0] = dp[1] = 1;
        for( int i=1; i<s.length(); i++ ){
```

```
8                   if( s[i]!='0' )
9                       dp[i+1] += dp[i];
10                  if( s[i-1]!='0' && (s[i-1]-'0')*10+(s[i]-'0')<=26 )
11                      dp[i+1] += dp[i-1];
12              }
13              return dp[s.length()];
14          }
15      };
```

3.3 Prove the Correctness

首先该问题包含最优子结构的性质，求 0–n 的字符串包含最多的编码种数包含了求 0–n-1 的字符串包含最多的编码种数或 0–n-2 的字符串包含最多的编码种数。其次该问题包含重叠子性质，大量子问题会重复计算，如求 dp[5] 需要计算 dp[4] 或 dp[3], 求 dp[4] 需要求 dp[3] 或 dp[2]，因此 dp[3] 被重复计算。综上该问题可以用动态规划方法解且求解方程正确。

3.4 The complexity of this algorithm

整个求解过程只遍历了一遍字符串，故算法时间复杂度为 $O(n)$。