Assignment 02

肖阳 201918008629001

1 Robbing Money

1. (a)Assume that there are N houses along the street, I use $\mathrm{OPT(N)}$ to represent the maximum amount of money I can rob at one night and $\mathrm{A_i}(1 \leq i \leq N)$ to represent the amount of money in each house. If N equal to 1, I only have one house to rob , so $\mathrm{OPT(N)} = \mathrm{OPT}(1) = 1$. If N equal to 2, I will rob the house which has larger amount of money because I can't rob both at one night, so $\mathrm{OPT(N)} = \mathrm{OPT}(2) = \max(\mathrm{A_1}, \mathrm{A_2})$. Considering more common cases, the maximum amount of money I can get from i houses determined by whether I rob the N^{th} house. If I rob the i^{th} house , $\mathrm{OPT(i)} = \mathrm{OPT(i-2)} + \mathrm{A_i}$, or $\mathrm{OPT(i)} = OPT(i-1)$ because I can't rob two adjacent houses.

   DP EQUATION:

$$\mathrm{OPT(N)} = \begin{cases} A_1, N = 1 \\ \max(A_1, A_2), N = 2 \\ \max\big(OPT(N-2) + A_N, OPT(N-1)\big), N > 2 \end{cases}$$

   (b) **Pseudo-code**

```
算法 1 robbing money
输入: Array数组, n数组大小
输出: 抢到的最大钱数
function RobbingMoney(Array, n)
    result ← 0
    if n == 1 then
        return Array[0]
    end if
    if n == 2 then
        return max(Array[0], Array[1])
    end if
    pre ← Array[0]
    cur ← max(Array[0], Array[1])
    for i = 2 → n − 1 do
        result ← max(pre, cur + Array[i])
        pre ← cur
        cur ← result
    end for
    return result
end function
```

   (c) **Prove the correctness**

   Assume that OPT(i) have a better value OPT'(i), and I rob the i^{th} house so that $\mathrm{OPT(i)} = \mathrm{OPT(i-2)} + \mathrm{A_i}$. Since $\mathrm{OPT'(i)}$ has a different choice, $\mathrm{OPT'}(i) = OPT(i-1)$. What's more,

$OPT(i-1)$ is greater than $\mathrm{OPT(i-2)+A_i}$. That is a contradiction to the definition of OPT(i). So the correctness is proved!

(d) **Time complexity**: O(n) **Space complexity**: O(1)

2. (a) The dissimilarity between this question to the last one is that the first house and the last house can't be robbed at one night, so that the question can be splited into two situation: (1)The first house was not robbed. OPT(N) can be solved by the same way in Q1 with A[2:N].(2)The last house was not robbed. OPT(N) can be solved by the same way in Q1 with A[1:N-1].

DP EQUATION:

$$\mathrm{OPT\_CIRCLE(1{:}\,N)} = \max\begin{cases} OPT(1{:}\,N-1) \\ OPT(2{:}\,N) \end{cases}$$

$\mathrm{OPT(i,j)}$ means the maximum amount of money I can rob from the i^{th} house to the j^{th} house.

(b) **Pseudo-code**

```
算法 1 robbing money
输入: Array数组，n数组大小
输出: 抢到的最大钱数
function RobbingMoney(Array, n)
    result ← 0
    if n == 1 then
        return Array[0]
    end if
    if n == 2 then
        return max(Array[0], Array[1])
    end if
    pre ← Array[0]
    cur ← max(Array[0], Array[1])
    for i = 2 → n − 1 do
        result ← max(pre, cur + Array[i])
        pre ← cur
        cur ← result
    end for
    return result
end function
function RobbingMoney2(Array, n)
    ResultNoFirst ←RobbingMoney(Array + 1, n − 1)
    ResultNoLast ←RobbingMoney(Array, n − 1)
    return max(ResultNoFirst, ResultNoLast)
end function
```

(c)**Prove the correctness**

The question contains the two special cases of Q1, without the first house or without the last house .This will not affect the correctness of the original algorithm.

(d)**Time complexity**: O(n) **Space complexity**: O(1)

2 Node Selection

(a) Assume R is the root of any subtree of the entire tree. $\mathrm{OPT(R)}$ represents the maximum sum of weight I can get from the subtree whose root of R. So there are only two situation: choose R or not choose R. If R is chosen, $\mathrm{OPT(R)} = OPT(R \to left \to left) + OPT(R \to left \to right) + OPT(R \to right \to left) + OPT(R \to right \to right) + R \to value$ .If R is not chosen ,that means its child can be chosen, $\mathrm{OPT(R)} = OPT(R \to left) + OPT(R \to right)$.if R is null, then $\mathrm{OPT(R)} = OPT(R \to left) = OPT(R \to right) = 0$.

DP EQUATION:

$$\mathrm{OPT(root)} = \max\begin{cases} OPT(root \to left \to left) + OPT(root \to left \to right) + OPT(root \to right \to left) + \\ OPT(root \to right \to right) + root \to value \\ OPT(root \to left) + OPT(root \to right) \end{cases}$$

(b) Pseudo-code

算法 1 select nodes

输入: $root$树根节点

输出: 节点的最大权重和

```
function DFS(root)
    if root == NULL then
        return [0, 0]
    else
        OPT(root−>left) = DFS(root−>left)
        OPT(root−>right) = DFS(root−>right)
        ResultChooseRoot = max(OPT(root−>left)) + max(OPT(root−>right)
        ResultNotChooseRoot = root−>value + OPT(root−>left)[0] + OPT(root−>right)[0]
        return [ResultChooseRoot, ResultNotChooseRoot]
    end if
end function
function NodeSelection(root)
    return max(DFS(root))
end function
```

(c) Prove the correctness

Assume that $\mathrm{OPT(node_i)}$ have a better value $\mathrm{OPT}'(node_i)$, and I chosen $\mathrm{node_I}$ so that $\mathrm{OPT(node_i)} = OPT(node_i \to left \to left) + OPT(node_i \to left \to right) + OPT(node_i \to right \to left) + OPT(node_i \to right \to right) + node_i \to value$.

Since $\mathrm{OPT}'^{(\mathrm{node_I})}$ has the different value, $\mathrm{OPT}'(node_i) = OPT(node_i \to left) + OPT(node_i \to right)$. $\mathrm{OPT}'(node_i)$ is larger than $\mathrm{OPT}(node_i)$, that is to say $OPT(node_i \to left) + OPT(node_i \to right)$ is larger than $OPT(node_i \to left \to left) + OPT(node_i \to left \to right) + OPT(node_i \to right \to left) + OPT(node_i \to right \to right) + node_i \to value$. This is a contradiction of the definition of $\mathrm{OPT(node_i)}$. The prove complete!

(d) Every node in the tree will be visited only once , so time complexity of the algorithm is O(n).

3 Unique Binary Search Tree

(a) Assume OPT(N) is the number of unique binary search tree that 1~N can generate, F(i,N) is the number of unique binary search tree whose root is $\mathrm{i}(1 \le \mathrm{i} \le \mathrm{N})$ that 1~N can generate. So $\mathrm{OPT(n)} = \sum_{i=1}^{n} F(i,n)$. What's more , any BST whose root is i can be separated into two parts, left subtree is generated by number 1~(i-1) , right subtree is generated by number (i+1)~n. So the number of unique BST whose root is i equal to the number of unique left subtree times the number of unique right subtree, that is $\mathrm{F(i,N)} = \mathrm{OPT(i-1)OPT(n-i)}$.

DP EQUATION:

$$\mathrm{OPT(n)} = \sum_{i=1}^{n} F(i,n) = \sum_{i=1}^{n} OPT(i-1)OPT(n-i)$$

What's more , $\mathrm{OPT(n)} = \sum_{i=1}^{n} OPT(i-1)OPT(n-i)$ is a catalan number, so the equation can be transformed into

$$\mathrm{OPT(n)} = \frac{4n-2}{n+1} OPT(n-1)$$

(b) Pseudo-code

算法 1 Unique BST

输入: n

输出: 整数1-n产生不同二叉搜索树的个数

```
function DFS(n)
    if n == 0 or n == 1 then
        return 1
    end if
    OPT[0] = OPT[1] = 1
    for i = 2 → n do
        for j = 1 → i do
            OPT[i]+ = OPT[j − 1] ∗ OPT[i − j]
        end for
    end for
    return OPT[n]
end function
```

算法 1 Unique BST 02

输入: n

输出: 整数1-n产生不同二叉搜索树的个数

```
function UniqueBST-v2(n)
    if n == 0 or n == 1 then
        return 1
    end if
    pre ← 1
    for i = 1 → n do
        cur ← (4 * i − 2)/(i + 1) * pre
        pre ← cur
    end for
    return cur
end function
```

(c) Prove the correctness

If n equal to 0,which means there are no positive integer to generate BST, we can only get a empty tree, OPT[0]=1; If n equal to 1, which means there are one integer 1 to generate BST, we can get a BST with one node, OPT[1]=1. Given N, there will be N situations of the root, So $\mathrm{OPT(N)} = \sum_{i=1}^{n} F(i, N)$ contain every cases of BST generated by N integer, and

$$\mathrm{OPT(n)} = \sum_{i=1}^{n} F(i, n) = \sum_{i=1}^{n} OPT(i-1)OPT(n-i)\ (2 \le \mathrm{n})$$

has already been proved to be right.

(d) Unique BST 01: Time complexity : O(n^2) Space complexity: O(n)

Unique BST 02: Time complexity : O(n) Space complexity: O(1)