337. House Robber III

Medium 3635 63 Add to List Share

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can rob tonight without alerting the police.

Example 1:

```
Input: [3,2,3,null,3,null,1]

     3
    / \
   2   3
    \   \
     3   1

Output: 7
Explanation: Maximum amount of money the thief can rob = 3 + 3 + 1 = 7.
```

Example 2:

```
Input: [3,4,5,1,3,null,1]

     3
    / \
   4   5
  / \   \
 1   3   1

Output: 9
Explanation: Maximum amount of money the thief can rob = 4 + 5 = 9.
```

Definition:

rob: the maximum money you can rob tonight if you rob this node.

not_rob: the maximum money you can rob tonight if not rob this node.

Recurrence relation:

For each root:

rob = root.val + root.left.not_rob + root.right.not_rob

not_rob = max(root.left.rob, root.left.not_rob) + max(root.right.rob, root.right.not_rob)

Time complexity: O(N)

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int rob(TreeNode* root) {
        int rob;
        int not_rob;
        dp(root, &rob, &not_rob);
        return max(rob, not_rob);
    }

private:
    void dp(TreeNode* root, int *rob, int *not_rob) {
        if (root == NULL) {
            *rob = 0;
            *not_rob = 0;
            return;
        }

        int left_rob;
        int left_not_rob;
        dp(root->left, &left_rob, &left_not_rob);

        int right_rob;
        int right_not_rob;
        dp(root->right, &right_rob, &right_not_rob);

        *rob = root->val + left_not_rob + right_not_rob;
        *not_rob = max(left_rob, left_not_rob) + max(right_rob, right_not_rob);

        return;
    }
};
```