The thief has found himself a new place for his thievery again. There is only one entrance to this area, called 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 form a binary tree. It will automatically contact the police if **two directly-linked houses were broken into on the same night**.

Given the root of the binary tree, return *the maximum amount of money the thief can rob****without alerting the police***.

**Example 1:**

**Input:** root = [3,2,3,null,3,null,1]

**Output:** 7

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

**Example 2:**

**Input:** root = [3,4,5,1,3,null,1]

**Output:** 9

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

Solution:

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public int rob(TreeNode root) {

return rob(root, new HashMap<>());

}

public int rob(TreeNode root, Map<TreeNode, Integer> map) {

if (root == null) return 0;

if (map.containsKey(root)) return map.get(root);

int ans = 0;

if (root.left != null) {

ans += rob(root.left.left, map) + rob(root.left.right, map);

}

if (root.right != null) {

ans += rob(root.right.left, map) + rob(root.right.right, map);

}

ans = Math.max(ans + root.val, rob(root.left, map) + rob(root.right, map));

map.put(root, ans);

return ans;

}

}

T.C : O(n)  
S.C. : O(n)  
**Explanation:** If you observe the recursive approach we have overlapping subproblems like for root node we are calling on its grandchildren (root.left.left, root.left.right, root.right.left, root.right.right) and when we are on child node of root (root.left, root.right) then again we will need the data of those four nodes so again calling on it. So here you have **Recusrion+Overlapping SubProblems** which can make you think of DP Approach.  
So what we are doing is just store the calculated answer for each node int the HashMap and if we need the value for that node again at any point we will just do the map.get(node) and get the value. Rest Recursive logic is absolutely same.