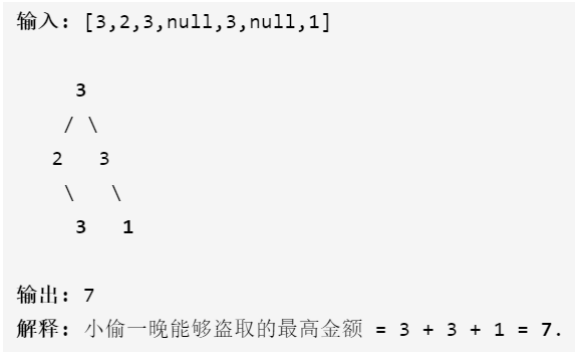
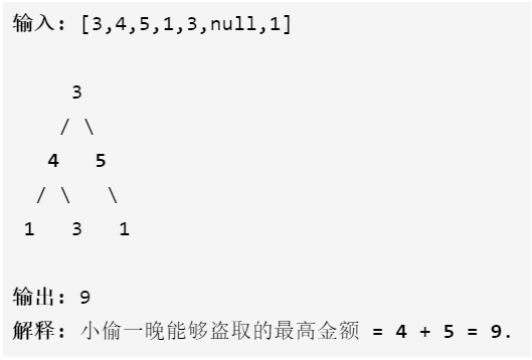
337 打家劫舍Ⅲ

在上次打劫完一条街道之后和一圈房屋后，小偷又发现了一个新的可行窃的地区。这个地区只有一个入口，我们称之为“根”。 除了“根”之外，每栋房子有且只有一个“父“房子与之相连。一番侦察之后，聪明的小偷意识到“这个地方的所有房屋的排列类似于一棵二叉树”。 如果两个直接相连的房子在同一天晚上被打劫，房屋将自动报警。
计算在不触动警报的情况下，小偷一晚能够盗取的最高金额。





- dfs 动态规划

```
class Solution {
    public int rob(TreeNode root) {
        int[] res = dfs(root);
        return Math.max(res[0],res[1]);
    }

     private int[] dfs(TreeNode r){
        if(r == null)  return new int[2];//边界条件，r为null时，跳出

        int[] left = dfs(r.left);
        int[] right = dfs(r.right);

        int[] res = new int[2];

        //计算不抢劫当前根节点可获得的最大金额(那么其左右子树可以随便抢)
        res[0] = Math.max(left[0],left[1]) + Math.max(right[0],right[1]);
        // 计算若抢劫根节点可获得的最大金额(此时,其左右子树的根节点不能被抢)
        res[1] = r.val + left[0] + right[0];

        return res;
    }
}
```

- 动态规划 HashMap存储

```
class Solution {
    // 偷当前节点
    Map<TreeNode, Integer> f = new HashMap<TreeNode, Integer>();
    // 不偷当前节点
    Map<TreeNode, Integer> g = new HashMap<TreeNode, Integer>();

    public int rob(TreeNode root) {
        dfs(root);
        return Math.max(f.getOrDefault(root, 0), g.getOrDefault(root, 0));
    }

    public void dfs(TreeNode node) {
        if (node == null) {
            return;
        }
        // 先递归到底层，类似数组也是从0开始
        dfs(node.left);
        dfs(node.right);

        f.put(node, node.val + g.getOrDefault(node.left, 0) +
g.getOrDefault(node.right, 0));

        // 不偷当前节点的话，左右子树随便偷
        g.put(node, Math.max(f.getOrDefault(node.left, 0),
g.getOrDefault(node.left, 0)) + Math.max(f.getOrDefault(node.right, 0),
g.getOrDefault(node.right, 0)));
    }
}
```

- 仅递归

```
class Solution {

    public int rob(TreeNode root) {
        if(root == null){
            return 0;
        }
        int sum = root.val;

        if(root.left != null){
            sum += rob(root.left.left) + rob(root.left.right); // 第一级 第三级
        }

        if(root.right != null){
            sum += rob(root.right.left) + rob(root.right.right);
        }

       return Math.max(sum, rob(root.left) + rob(root.right));  // 和第二级比较
    }
}
```