



213. 打家劫舍 II

已解答 

中等

 相关标签

 相关企业

 提示

Aa

你是一个专业的小偷，计划偷窃沿街的房屋，每间房内都藏有一定的现金。这个地方所有的房屋都 **围成一圈**，这意味着第一个房屋和最后一个房屋是紧挨着的。同时，相邻的房屋装有相互连通的防盗系统，**如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警**。

给定一个代表每个房屋存放金额的非负整数数组，计算你 **在不触动警报装置的情况下**，今晚能够偷窃到的最高金额。

示例 1:

输入: `nums = [2,3,2]`

输出: 3

解释: 你不能先偷窃 1 号房屋 (金额 = 2)，然后偷窃 3 号房屋 (金额 = 2)，因为他们是相邻的。

示例 2:

输入: `nums = [1,2,3,1]`

输出: 4

解释: 你可以先偷窃 1 号房屋 (金额 = 1)，然后偷窃 3 号房屋 (金额 = 3)。偷窃到的最高金额 = 1 + 3 = 4。

示例 3:

输入: `nums = [1,2,3]`

输出: 3

解题思路一:

保留每次解,但是一组解只有首没有尾,一组解只有尾没有首,这两组解取最大值。为了节省空间,每次每组解只有两个值,结束后返回这组解的最大值。

代码:

```
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
```

```

        if (n == 0) return 0;
        if (n == 1) return nums[0];

        // 使用类方法实现线性打家劫舍
        return max(robLinear(nums, 0, n - 2), robLinear(nums, 1, n - 1));
    }

private:
    // 类方法，用于对子数组进行线性打家劫舍
    int robLinear(const vector<int>& nums, int start, int end) {
        int prev2 = 0, prev1 = 0, curr = 0;
        for (int i = start; i <= end; ++i) {
            curr = max(prev1, prev2 + nums[i]);
            prev2 = prev1;
            prev1 = curr;
        }
        return curr;
    }
};

```

解题思路二：

保留每次每个解，但是分两组解，一组去头，一组去尾，实际上就是两组的起始点不同，核心思路还是打家劫舍 1.0：设置 **dp** 数组来存储解（即当前能得到的最大值），观察数组可以发现，第一个解一定是 **dp[0]=nums[0]**，第二个解一定是 **dp[1]=max(dp[0],nums[1])**；从第三个解开始，一定是前一个解的值与前一个解的前一项与当前价值之和的较大值，即有方程 **dp[i]=max(dp[i-1],dp[i-2]+nums[i])**，最后输出 **dp[end-start+1]** 即是所求的最大值

代码：

```

class Solution {
public:
    int rob(vector<int>& nums) {
        //异常处理
        if (nums.empty()) {
            return 0;
        }
        if (nums.size() == 1) {
            return nums[0];
        }
        int n = nums.size();
    }
};

```

```

        return max(robLinear(nums, 0, n - 2), robLinear(nums, 1, n - 1));
    }

private:
    int robLinear(vector<int>& nums, int start, int end) {
        //异常处理
        int len = end - start + 1;
        if (len == 1) {
            return nums[start];
        }
        //打家劫舍 1.0
        vector<int> dp(len);
        dp[0] = nums[start];
        dp[1] = max(nums[start], nums[start + 1]);

        for (int i = 2; i < len; ++i) {
            dp[i] = max(dp[i - 1], dp[i - 2] + nums[start + i]);
        }

        return dp[len - 1];
    }
};

```

题目：

题库 < > 叉

题目描述 笔记 × 题解 提交记录

337. 打家劫舍 III 已解答 ✓

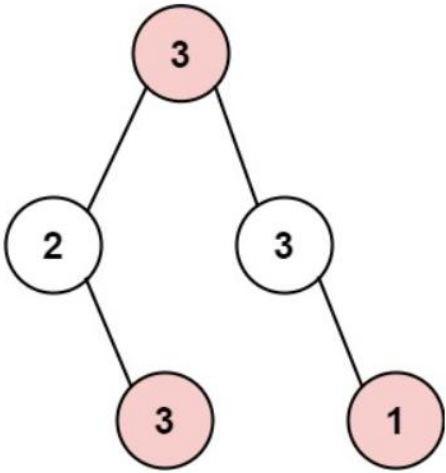
中等 相关标签 相关企业 Ax

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 `root`。

除了 `root` 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。

给定二叉树的 `root`。返回 **在不触动警报的情况下**，小偷能够盗取的最高金额。

示例 1:



```
graph TD; 3((3)) --- 2((2)); 3 --- 3_2((3)); 2 --- 3_1((3)); 3_2 --- 1((1));
```

输入: `root = [3,2,3,null,3,null,1]`

2K 1.6K ☆ 叉 ⓘ

C++

1
2
3
4
5
6
7
8
9
10
11
12
13

已存储

✓ 测试

Ca

root =

[3]

</> So

解题思路一：

要么抢劫当前节点，要么越过当前节点选择其孩子节点，使用键值对来存储抢劫当前节点的最大值和不抢劫当前节点的最大值，对于每个可能存在的节点，采用递归处理，遇到空节点返回(0,0)；自底向上走，最后返回的就是可能存在的最大值

```
class Solution {
public:
    int rob(TreeNode* root) {
        auto result = robSub(root);
        return max(result.first, result.second);
    }
}
```

```

private:
    // 返回一个 pair，第一个值表示不抢劫当前节点的最大金额，第二个值表示抢劫当前节点
    的最大金额
    pair<int, int> robSub(TreeNode* node) {
        if (!node) return { 0, 0 };

        auto left = robSub(node->left);
        auto right = robSub(node->right);

        // 不抢劫当前节点
        int robExcludeCurrent = max(left.first, left.second) + max(right.first,
right.second);

        // 抢劫当前节点
        int robIncludeCurrent = node->val + left.first + right.first;

        return { robExcludeCurrent, robIncludeCurrent };
    }
};

```

解题思路二：

在解题思路一的基础上，使用哈希表来存储每个节点的值，而不是使用键值对，还可以使用 **vector** 来代替，但是由于几个解题思路高度相似，所以这里不做展示

```

class Solution {
public:
    unordered_map <TreeNode*, int> f, g;

    void dfs(TreeNode* node) {
        if (!node) {
            return;
        }
        dfs(node->left);
        dfs(node->right);
        f[node] = node->val + g[node->left] + g[node->right];
        g[node] = max(f[node->left], g[node->left]) + max(f[node->right],
g[node->right]);
    }
}

```

```
int rob(TreeNode* root) {  
    dfs(root);  
    return max(f[root], g[root]);  
}  
}
```