

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]);

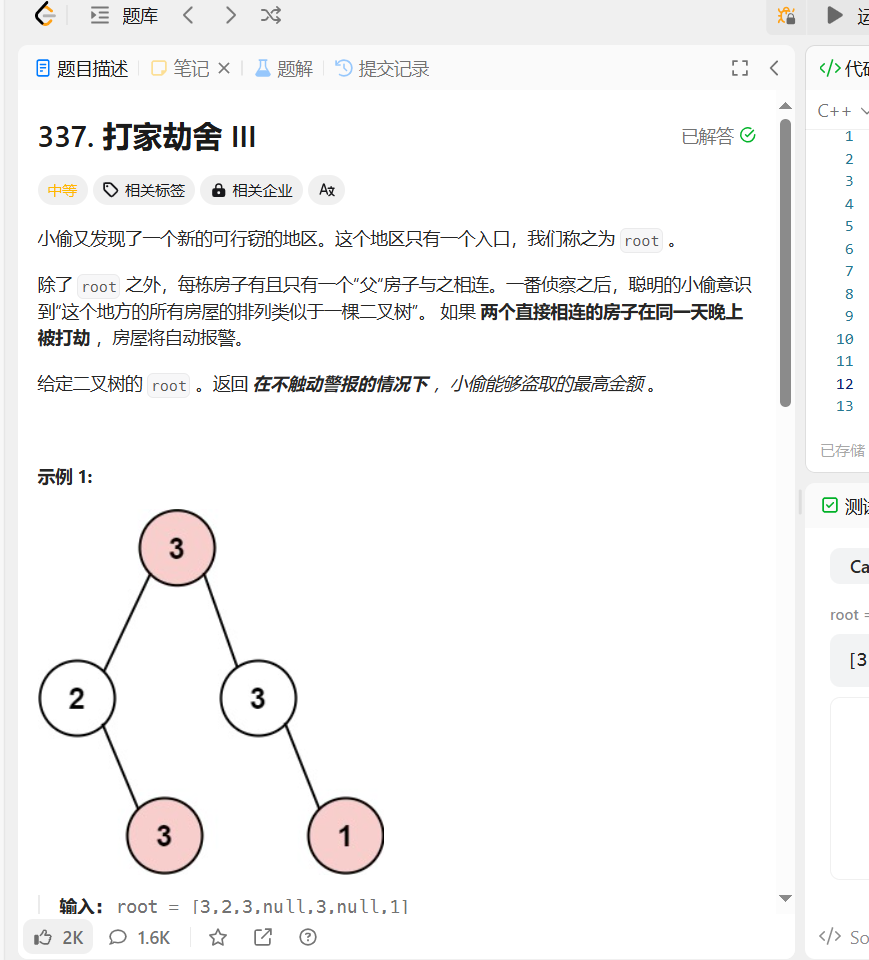
}

return dp[len - 1];

}

};

题目：



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]);

}

}