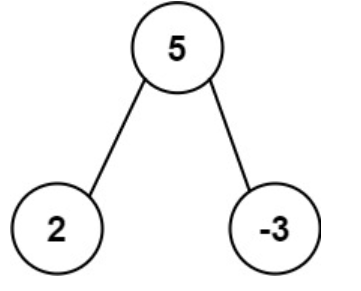
# 题目

给你一个二叉树的根结点root，请返回出现次数最多的子树元素和。如果有多个元素出现的次数相同，返回所有出现次数最多的子树元素和（不限顺序）。

一个结点的「子树元素和」定义为以该结点为根的二叉树上所有结点的元素之和（包括结点本身）。

示例 1：

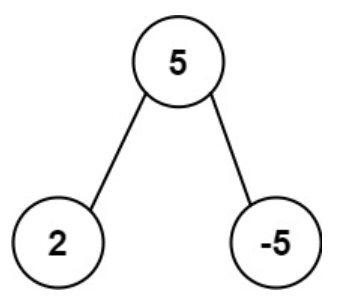


输入: root = [5,2,-3]

输出: [2,-3,4]

说明：以2为根节点的，因为没有子树，因此子树元素和为2，同理-3的子树元素和为-3,5的子树元素和未5+2-3=4。

示例 2：



输入: root = [5,2,-5]

输出: [2]

提示:

节点数在 [1, 104] 范围内

-105 <= Node.val <= 105

# 分析

## 方法一：深度优先搜索

思路：

我们可以从根结点出发，深度优先搜索这棵二叉树。对于每棵子树，其子树元素和等于子树根结点的元素值，加上左子树的元素和，以及右子树的元素和。

用哈希表统计每棵子树的元素和的出现次数，计算出现次数的最大值maxCnt，最后将出现次数等于maxCnt的所有元素和返回。

代码：

class Solution {

private:

unordered\_map<int, int> mp;

int maxCnt =0;

public:

int dfs(TreeNode \*node) {

if (node == nullptr) return 0;

// 计算所有子树的和

int sum = node->val + dfs(node->left) + dfs(node->right);

// 将子树和存储在map中，并且返回当前的最大值

maxCnt = max(maxCnt, ++mp[sum]);

//经典错误：mp[sum]++

return sum;

}

vector<int> findFrequentTreeSum(TreeNode\* root) {

// 执行深度优先遍历，完成map的初始化，并且计算出最大值

dfs(root);

vector<int> res;

for (auto &it : mp) {

if (it.second == maxCnt)

res.emplace\_back(it.first);

}

return res;

}

};

复杂度分析：

时间复杂度：O(n)，其中n是二叉树的结点个数。深度优先搜索的时间为O(n)。

空间复杂度：O(n)。哈希表和递归的栈空间均为O(n)。