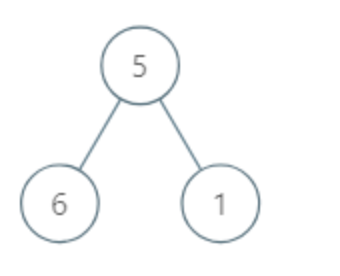
# 题目

给你一棵二叉树的根节点root，找出这棵树的每一棵子树的平均值中的 最大值。

子树是树中的任意节点和它的所有后代构成的集合。

树的平均值是树中节点值的总和除以节点数。

示例：



输入：[5,6,1]

输出：6.00000

解释：

以 value = 5 的节点作为子树的根节点，得到的平均值为 (5 + 6 + 1) / 3 = 4。

以 value = 6 的节点作为子树的根节点，得到的平均值为 6 / 1 = 6。

以 value = 1 的节点作为子树的根节点，得到的平均值为 1 / 1 = 1。

所以答案取最大值 6。

提示：

树中的节点数介于 1 到 5000之间。

每个节点的值介于 0 到 100000 之间。

如果结果与标准答案的误差不超过 10^-5，那么该结果将被视为正确答案。

# 分析

要解决“找出二叉树所有子树平均值中的最大值”的问题，核心思路是后序遍历二叉树，在遍历过程中计算每个子树的“节点总和”和“节点数量”，进而推导平均值，并实时更新最大值。

解题思路：

1、子树的定义与平均值计算：

子树是“任意节点及其所有后代”，因此每个节点都对应一棵子树（以该节点为根）。子树的平均值 = 子树所有节点值的总和 / 子树的节点数量。

2、后序遍历的必要性：

计算某节点为根的子树时，需先知道其左、右子树的“总和”和“节点数”（左子树贡献 + 右子树贡献 + 当前节点自身）。后序遍历（左 → 右 → 根）恰好满足这一依赖关系。

3、信息传递与最大值更新：

遍历每个节点时，返回该子树的“总和”和“节点数”（可用结构体或数组封装），同时计算当前子树的平均值，与全局最大值比较并更新。

代码：

class Solution {

public:

double maximumAverageSubtree(TreeNode\* root) {

double max\_avg = 0.0; // 存储最大平均值

postOrder(root, max\_avg); // 后序遍历，更新max\_avg

return max\_avg;

}

private:

// 后序遍历辅助函数：返回值为vector<int>{子树总和, 子树节点数}

vector<int> postOrder(TreeNode\* node, double& max\_avg) {

// 递归终止条件：空节点贡献0总和、0节点数

if (node == nullptr) {

return {0, 0};

}

// 1. 遍历左子树，获取左子树的总和与节点数

vector<int> left = postOrder(node->left, max\_avg);

int left\_sum = left[0];

int left\_count = left[1];

// 2. 遍历右子树，获取右子树的总和与节点数

vector<int> right = postOrder(node->right, max\_avg);

int right\_sum = right[0];

int right\_count = right[1];

// 3. 计算当前节点为根的子树的总和、节点数与平均值

int curr\_sum = left\_sum + right\_sum + node->val; // 总和 = 左 + 右 + 当前节点值

int curr\_count = left\_count + right\_count + 1; // 节点数 = 左 + 右 + 1（当前节点）

double curr\_avg = (double)curr\_sum / curr\_count; // 平均值（转为double避免整数除法）

// 4. 更新全局最大平均值

max\_avg = max(max\_avg, curr\_avg);

// 5. 返回当前子树的总和与节点数，供父节点计算

return {curr\_sum, curr\_count};

}

};

代码解释：

1、辅助函数postOrder：

- 输入：当前遍历的节点node、引用传递的全局最大平均值max\_avg（确保递归中能更新最大值）。

- 输出：vector<int>{子树总和, 子树节点数}，封装当前子树的核心信息，供父节点使用。

2、递归逻辑：

- 空节点处理：返回{0, 0}，表示无贡献。

- 左/右子树遍历：分别获取左、右子树的总和与节点数。

- 当前子树计算：整合左、右子树信息与当前节点，得到当前子树的总和、节点数，进而计算平均值。

- 最大值更新：将当前子树的平均值与全局最大值比较，取较大值更新max\_avg。

3、类型转换：

计算平均值时，将curr\_sum强制转为double，避免整数除法导致的精度丢失（如(5+6+1)/3若用整数除法会得到4，而非4.0）。

复杂度分析

- 时间复杂度：O(n)，其中n是二叉树的节点数。每个节点仅被遍历一次（后序遍历），每个节点的操作（计算总和、节点数、平均值）均为O(1)。

- 空间复杂度：O(h)，其中h是二叉树的高度。递归调用栈的深度取决于树的高度（平衡树h=log n，最坏情况（链状树）h=n）。