# 题目

输入一棵二叉树的根节点，判断该树是不是平衡二叉树。如果某二叉树中任意节点的左右子树的深度相差不超过1，那么它就是一棵平衡二叉树。

示例 1:

给定二叉树 [3,9,20,null,null,15,7]

3

/ \

9 20

/ \

15 7

返回true。

示例 2:

给定二叉树 [1,2,2,3,3,null,null,4,4]

1

/ \

2 2

/ \

3 3

/ \

4 4

返回 false。

**限制：**

1 <= 树的结点个数 <= 10000

**类似题目：Leetcode 110**

# 分析

## 方法一：树的深度

**思路：**

先序遍历每一个节点，并比较左右子树高度，如果有>1则返回false

注意：这里是比较每一个左右子树，不是比较左右子树的最大值和最小值。

错误代码：

class Solution {

private:

int dfs(TreeNode\* root, int maxDepth)

{

if(nullptr==root)return maxDepth;

int leftMaxDepth = dfs(root->left,maxDepth+1);

int rightMaxDepth = dfs(root->right,maxDepth+1);

return abs(leftMaxDepth-rightMaxDepth);

}

public:

bool isBalanced(TreeNode\* root) {

int maxDepth = 0;

if(dfs(root,maxDepth)<=1)

{

return true;

}

return false;

}

};

**代码：**

class Solution {

public:

bool isBalanced(TreeNode\* root) {

if (root == NULL) return true;

//如果该子树为空，则一定是平衡的（因为没有左右子树）

if (abs(getHeight(root->left) - getHeight(root->right)) > 1) return false;

return isBalanced(root->left)&& isBalanced(root->right);

}

int getHeight(TreeNode\* root)

{

if (root == NULL) return 0;

int leftHeight = getHeight(root->left);

int rightHeight = getHeight(root->right);

return leftHeight > rightHeight ? leftHeight + 1 : rightHeight + 1;

}

};

## 方法二：后序遍历

**思路：**

使用后序遍历的方式遍历二叉树的每个节点，那么在遍历到一个节点之前我们就已经

遍历了它的左右子树。只要在每个节点的时候记录它的深度(某一节点的深度等于它到叶节点的路径的长度）就可以判断每个节点是不是平衡的。

**代码：**

class Solution {

public:

bool isBalanced(TreeNode\* root)

{

if (root == nullptr) return true;

int depth = 0;

return isBalanced(root, depth);

}

bool isBalanced(TreeNode\* root,int &pDepth) {

if (root == nullptr) { pDepth = 0; return true; }

int left=0,right=0;

if (isBalanced(root->left, left) && isBalanced(root->right, right))

{

int diff = left - right;

if (abs(diff) <= 1)

{

pDepth = 1 + (left > right ? left : right);

return true;

}

}

return false;

}

};