平衡二叉树BalancedBinaryTree

# 平衡二叉树BalancedBinaryTree

## 定义：

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。

二叉树的深度定义：

从根节点到叶节点依次经过的节点(含根节点、叶节点)形成树的一条路径，最长路径的长度为树的深度。

获取二叉树的深度算法见二叉树文档。

## 题目要求

输入一棵二叉树，判断该二叉树是否是平衡二叉树。

## Java代码：

### 算法1：

依次求解当前节点的左子节点和右子节点的深度，判断差值是否在-1和1值之间。然后再依次递归当前节点的左子节点和右子节点。

该算法简单易懂，但是存在一个节点被重复遍历多次，所以时间效率不高。

/\*\*

\* 算法1：简单。

\* 通过分别求解每个节点的左子树和右子树的深度，比较判断。

\* 问题：存在一个节点被重复遍历多次的问题，时间效率不高。

\*/

public boolean isBalancedBinaryTreeByDepth(TreeNode root){

if(root == null) return true;//递归的终止条件

//分别获取左子树和右子树的深度

if(Math.abs(getDepth(root.left)-getDepth(root.right)) > 1){

return false;

}else {

return **isBalancedBinaryTreeByDepth**(root.left) && **isBalancedBinaryTreeByDepth**(root.right);

}

}

/\*\*

\* 根据二叉树的根节点，获取该二叉树的深度(路径的最大长度）

\*/

public int getDepth(TreeNode root){

if(root == null) return 0;

return Math.max(getDepth(root.left),getDepth(root.right))+1;

}

### 算法2：

/\*\*

\* 算法2：从最底层子树判断开始，直到整个二叉树，避免了一个节点被多次遍历的情况

\*/

public boolean isBalancedBinaryTree(TreeNode root){

int[] depth = new int[]{0};//初始值为0,用于存储当前子树的深度

return helper(root,depth);//其实这里的depth就是整个二叉树的深度

}

/\*\*

\*

\*/

public boolean helper(TreeNode root, int[] depth){

if(root == null){

**depth[0] = 0;//空，深度为0**

return true;

}

int[] leftDepth = new int[]{0};//用于存储当前节点的左子树的深度

int[] rightDepth = new int[]{0};//用于存储当前节点的右子树的深度

if(helper(root.left,leftDepth)&&helper(root.right,rightDepth)){

if(leftDepth[0]-rightDepth[0]<=1&&leftDepth[0]-rightDepth[0]>=-1){

//返回true之前，保存当前节点为根节点的二叉树的深度，递归的上一层函数需要

**depth[0] = Math.max(leftDepth[0],rightDepth[0])+1;**

return true;

}

}

return false;

}