对树的操作的算法总结：

除了熟知的迭代方式：

1. root 结点的处理
2. root.left 结点的处理
3. root.right 结点的处理

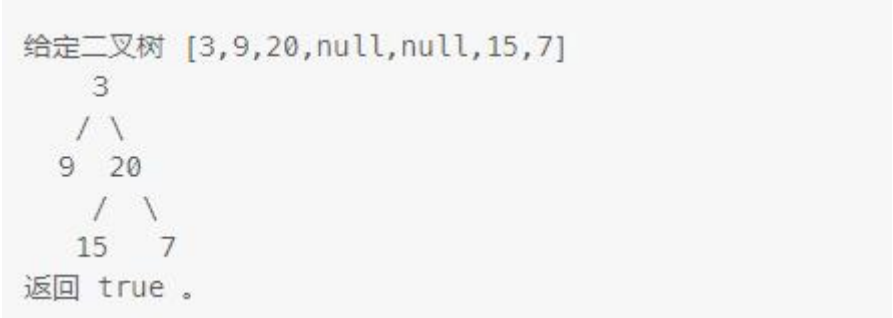
精简的求高度的算法

```
private int height(TreeNode root) {
    if (root == null) {
        return 0;
    }
    return Math.max(height(root.left),height(root.right))+1;
}
```

题目：

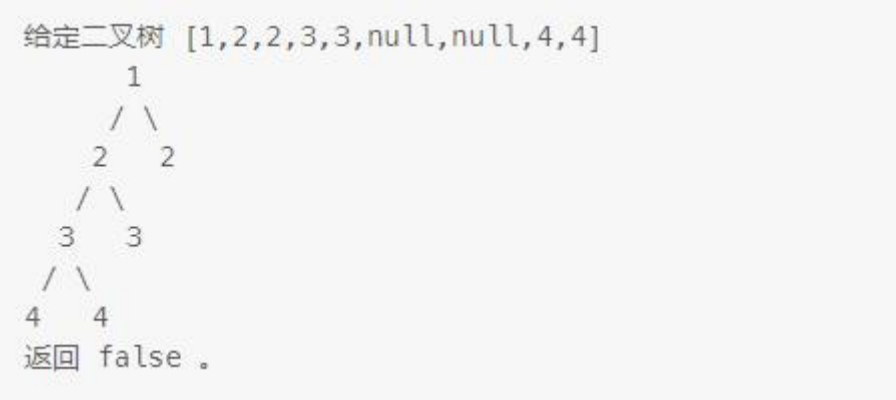
实现一个函数，检查二叉树是否平衡。在这个问题中，平衡树的定义如下：任意一个节点，其两棵子树的高度差不超过 1。

示例 1:



返回 true 。

示例 2:



返回 false 。

题目解析:

使用自底向上的做法，则对于每个节点，函数 height 只会被调用一次。

自底向上递归的做法类似于后序遍历，对于当前遍历到的节点，先递归地判断其左右子树是否平衡，再判断以当前节点为根的子树是否平衡。如果一棵子树是平衡的，则返回其高度（高度一定是非负整数），否则返回 −1-1−1。如果存在一棵子树不平衡，则整个二叉树一定不平衡。

```
class Solution {
    public boolean isBalanced(TreeNode root) {
        return height(root) >= 0;
    }

    public int height(TreeNode root) {
```

```
            if (root == null) {
                return 0;
            }
            int leftHeight = height(root.left);
            int rightHeight = height(root.right);
            if (leftHeight == -1 || rightHeight == -1 || Math.abs(leftHeight
- rightHeight) > 1) {
                return -1;
            } else {
                return Math.max(leftHeight, rightHeight) + 1;
            }
        }
}
```