```
        next[1] = 0;
        int pos = 2;
        int cn = 0;
        while (pos < next.length) {
                if (ms[pos - 1] == ms[cn]) {
                        next[pos++] = ++cn;
                } else if (cn > 0) {
                        cn = next[cn];
                } else {
                        next[pos++] = 0;
                }
        }
        return next;
}
```

判断二叉树是否为平衡二叉树

【题目】

平衡二叉树的性质为：要么是一棵空树，要么任何一个节点的左右子树高度差的绝对值不超过 1。给定一棵二叉树的头节点 head，判断这棵二叉树是否为平衡二叉树。

【要求】

如果二叉树的节点数为 N，要求时间复杂度为 $O(N)$。

【难度】

士 ★☆☆☆

【解答】

解法的整体过程为二叉树的后序遍历，对任何一个节点 node 来说，先遍历 node 的左子树，遍历过程中收集两个信息，node 的左子树是否为平衡二叉树，node 的左子树最深到哪一层记为 lH。如果发现 node 的左子树不是平衡二叉树，无须进行任何后续过程，此时返回什么已不重要，因为已经发现整棵树不是平衡二叉树，退出遍历过程；如果 node 的左子树是平衡二叉树，再遍历 node 的右子树，遍历过程中再收集两个信息，node 的右子树是否为平衡二叉树，node 的右子树最深到哪一层记为 rH。如果发现 node 的右子树不是平衡二叉树，无须进行任何后续过程，返回什么也不重要，因为已经发现整棵树不是平衡二叉树，退出遍历过程；如果 node 的右子树也是平衡二叉树，就看 lH 和 rH 差的绝对值是否大

于 1，如果大于 1，说明已经发现整棵树不是平衡二叉树，如果不大于 1，则返回 lH 和 rH 较大的一个。

判断的全部过程请参看如下代码中的 isBalance 方法。在递归函数 getHeight 中，一旦发现不符合平衡二叉树的性质，递归过程会迅速退出，此时返回什么根本不重要。boolean[] res 长度为 1，其功能相当于一个全局的 boolean 变量。

```
public boolean isBalance(Node head) {
        boolean[] res = new boolean[1];
        res[0] = true;
        getHeight(head, 1, res);
        return res[0];
}

public int getHeight(Node head, int level, boolean[] res) {
        if (head == null) {
                return level;
        }
        int lH = getHeight(head.left, level + 1, res);
        if (!res[0]) {
                return level;
        }
        int rH = getHeight(head.right, level + 1, res);
        if (!res[0]) {
                return level;
        }
        if (Math.abs(lH - rH) > 1) {
                res[0] = false;
        }
        return Math.max(lH, rH);
}
```

整个后序遍历的过程中，每个节点最多遍历一次，如果中途发现不满足平衡二叉树的性质，整个过程会迅速退出，没遍历到的节点也不用遍历了，所以时间复杂度为 $O(N)$。

根据后序数组重建搜索二叉树

【题目】

给定一个整型数组 arr，已知其中没有重复值，判断 arr 是否可能是节点值类型为整型的搜索二叉树后序遍历的结果。

进阶：如果整型数组 arr 中没有重复值，且已知是一棵搜索二叉树的后序遍历结果，通过数组 arr 重构二叉树。