

二叉树的深度与高度

2022年9月9日 10:19

此题我们以 平衡二叉树 作为引入点

110. 平衡二叉树

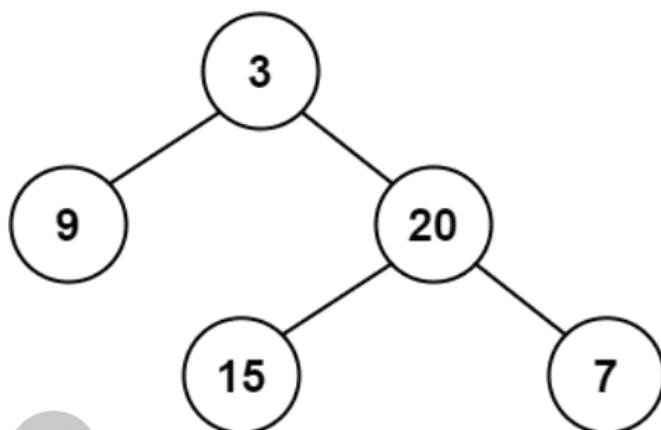
难度 简单  1131     


给定一个二叉树，判断它是否是高度平衡的二叉树。

本题中，一棵高度平衡二叉树定义为：

一个二叉树每个节点的左右两个子树的高度差的绝对值不超过 1。

示例 1:



 输入: `root = [3,9,20,null,null,15,7]`
输出: `true`

如图所示：平衡二叉树定义为：一个二叉树每个节点的左右两个子树的高度差的绝对值不超过1

那么我们需要明白 什么是高度？而高度与深度在计算逻辑上有什么区别吗？

二叉树节点的深度：指从根节点到该节点的最长简单路径边的条数或者节点数（取决于深度从0开始还是从1开始，或者说以节点数定义，还是以边定义）

二叉树节点的高度：指从该节点到叶子节点的最长简单路径边的条数或者节点数（取决于深度从0开始还是从1开始，或者说以节点数定义，还是以边定义）

关于根节点的深度究竟是1 还是 0，不同的地方有不一样的标准，leetcode的题目中都是以节点为一度，即根节点深度是1。但维基百科上定义用边为一度，即根节点的深度是0，我们暂时以leetcode为准（毕竟要在这上面刷题）。

求深度可以看作是从上至下去查找，即对应前序遍历

而求高度必须从下至上去查找，即对应后序遍历

我们在求二叉树的最小深度时，使用后序遍历（从叶子节点出发，找到最短的路径）

```
1  class Solution {
2  public:
3      int getDepth(TreeNode* node) {
4          if (node == NULL) return 0;
5          int leftDepth = getDepth(node->left);        // 左
6          int rightDepth = getDepth(node->right);      // 右
7                                                    // 中
8          // 当一个左子树为空，右不为空，这时并不是最低点
9          if (node->left == NULL && node->right != NULL) {
10             return 1 + rightDepth;
11         }
12         // 当一个右子树为空，左不为空，这时并不是最低点
13         if (node->left != NULL && node->right == NULL) {
14             return 1 + leftDepth;
15         }
16         int result = 1 + min(leftDepth, rightDepth);
17         return result;
18     }
19
20     int minDepth(TreeNode* root) {
21         return getDepth(root);
22     }
23 };
```

而在求二叉树的最大深度时，可以使用后序遍历

这是因为，二叉树的最大深度，实际上是根节点的高度，所以才可以使用
后序遍历

若真正求取二叉树的最大深度，代码应该写成如下（前序遍历）

```

1  class Solution {
2  public:
3      int result;
4      void getDepth(TreeNode* node, int depth) {
5          result = depth > result ? depth : result; // 中
6
7          if (node->left == NULL && node->right == NULL) return ;
8
9          if (node->left) { // 左
10             depth++; // 深度+1
11             getDepth(node->left, depth);
12             depth--; // 回溯, 深度-1
13         }
14         if (node->right) { // 右
15             depth++; // 深度+1
16             getDepth(node->right, depth);
17             depth--; // 回溯, 深度-1
18         }
19         return ;
20     }
21     int maxDepth(TreeNode* root) {
22         result = 0;
23         if (root == NULL) return result;
24         getDepth(root, 1);
25         return result;
26     }
27 };

```

现在回归本题，判断二叉树是否为平衡二叉树
 题目中所给的类型是bool类型，没办法反映子树的高度
 因此另设一个函数 返回值为int类型

求取子树的高度采用后序遍历

核心代码：

```

    int leftHeight = getHeight(node->left); // 左
    if (leftHeight == -1) return -1;
    int rightHeight = getHeight(node->right); // 右
    if (rightHeight == -1) return -1;

    int result;
    if (abs(leftHeight - rightHeight) > 1) { // 中
        result = -1;
    } else {
        result = 1 + max(leftHeight, rightHeight); // 以当前节点为根节点的树的最大高度
    }

    return result;

```

整体代码：

```
class Solution {
public:
    // 返回以该节点为根节点的二叉树的高度，如果不是平衡二叉树了则返回-1
    int getHeight(TreeNode* node) {
        if (node == NULL) {
            return 0;
        }
        int leftHeight = getHeight(node->left);
        if (leftHeight == -1) return -1;
        int rightHeight = getHeight(node->right);
        if (rightHeight == -1) return -1;
        return abs(leftHeight - rightHeight) > 1 ? -1 : 1 + max(leftHeight, rightHeight);
    }
    bool isBalanced(TreeNode* root) {
        return getHeight(root) == -1 ? false : true;
    }
};
```