树的结构

Saturday, July 28, 2018 10:19 AM

Example 1:

```
Input:     1         1
          / \       / \
         2   3     2   3

        [1,2,3],   [1,2,3]

Output: true
```

Example 2:

```
Input:     1         1
          /           \
         2             2

        [1,2],     [1,null,2]

Output: false
```

Example 3:

```
Input:     1         1
          / \       / \
         2   1     1   2

        [1,2,1],   [1,1,2]

Output: false
```

相同的二叉树

将两棵树，左节点对左节点，右节点对右节点，先比较节点的值，再比较结构

```
if (p == null && q == null){
    return true;
}
```

从一开始，如果两棵树的根节点相同，即使都为 null 也成立，再向下搜索

接下来是判断左右孩子是否为空的条件：

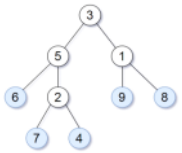
```
if (p == null || q == null){
        return false;
}
```

p 和 q 都为 null 会在一开始被捕获，所以这里判断的是只有一个孩子为空的情况

如果一棵树的某个节点有左孩子，而另一棵树的相同节点却没有左孩子，就导致结构的差异，结果为 false

```
if (p.val == q.val){
    result = isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
}
```

在两个节点的值相同的情况下进行递归



叶节点相同的二叉树

用深搜遍历树，用列表存放叶节点，然后比较列表就能解决

首先是遍历：

```
public void DFS(TreeNode treeNode, List<Integer> list){
    if (treeNode.left == null && treeNode.right == null){
        list.add(treeNode.val);

    }
    if (treeNode.left != null){
        DFS(treeNode.left, list);
    }
    if (treeNode.right != null){
        DFS(treeNode.right, list);
    }
}
```

如果左右孩子都为空，就表示是叶节点，添加至列表

```
    List<Integer> list1 = new ArrayList<>();
    List<Integer> list2 = new ArrayList<>();
    DFS(root1, list1);
    DFS(root2, list2);
    if (list1.size() != list2.size()){
        return false;
    }
    for (int i = 0; i < list1.size(); i++) {
        if (!list1.get(i).equals(list2.get(i))){
            return false;
        }
    }
    return true;
```

For example, this binary tree `[1,2,2,3,4,4,3]` is symmetric:

```
    1
   / \
  2   2
 / \ / \
3  4 4  3
```

But the following `[1,2,2,null,3,null,3]` is not:

```
    1
   / \
  2   2
   \   \
   3    3
```

leftChild.left, rightChild.right 比较的是最左边和最右边的节点

leftChild.right, rightChild.left 比较的是相邻的两个节点

Example 1:

Given the following tree `[3,9,20,null,null,15,7]`:

```
    3
   / \
  9  20
    /  \
   15   7
```

Return true.

Example 2:

Given the following tree `[1,2,2,3,3,null,null,4,4]`:

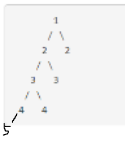
```
      1
     / \
    2   2
   / \
  3   3
 / \
4   4
```

Return false.

关于最后一个判断语句：

```
return left > right ? left : right
```

是用在一个节点的左右子树深度不同的时候，不如我在这棵树的节点 4 添加一个左孩子：



然后在返回的时候，left > right，返回 left

因为要求深度，所以需要搜索到最深一个节点，如果将节点加在右边也是一个样

对称二叉树

对称树要判断一个节点的左孩子和另一个节点的右孩子是否相同，一左一右，一右一左相对应

如果对应节点的不相同就返回 false

一开始判断两个参数是否为空，如果都为空，返回 true，如果有一个不为空，则返回 false

```
public boolean DFS(TreeNode leftChild, TreeNode rightChild){
    if (leftChild == null || rightChild == null){
        return leftChild == rightChild;
    }
    if (leftChild.val != rightChild.val){
        return false;
    }
    //判断一个节点的左孩子和另一个节点的右孩子是否相同
    return DFS(leftChild.left, rightChild.right)
            && DFS(leftChild.right, rightChild.left);
}
```

要判断一开始根节点是否为空，然后用左右孩子进行遍历：

```
public boolean isSymmetric(TreeNode root) {
    //如果根节点为空，要返回 true，烦！
    if(root == null){
        return true;
    }
    return DFS(root.left, root.right);
}
```

平衡二叉树

平衡二叉树指的是，每个节点的两棵子树的深度相差不超过1的二叉树，也就是节点的深度的最大值和最小值相差1

这里的做法是先找到最底层，然后一层一层地 +1，并进行比较

```
    if (root == null){
        return 1;
    }
    int left = DFS(root.left) + 1;
    int right = DFS(root.right) + 1;
```

每一次递归就 +1

针对 Example 2 来分析一下：

首先， left = DFS(root.left) = DFS(2) + 1, right = DFS(root.right) = DFS(2) + 1;

右边的 DFS(2) 已经是叶节点，返回 0，所以 root.right（右边子树）的深度为 1（因为是子树深度，所以一开始的根节点不加入计算）

接下来都是左边子树：

DFS(2).left = DFS(3) + 1, DFS(2).right = DFS(3) + 1

DFS(3).left = DFS(4) + 1, DFS(3).right = DFS(4) + 1

DFS(4) = 1

所以 DFS(2) = 3，左边的子树深度为 3

一级一级返回，这时候不满足

```
    if (Math.abs(left - right) > 1){
        result = false;
    }
```

所以就直接返回 false