104. Maximum Depth of Binary Tree

link

dfs非递归相当于多加一个数据结构进去

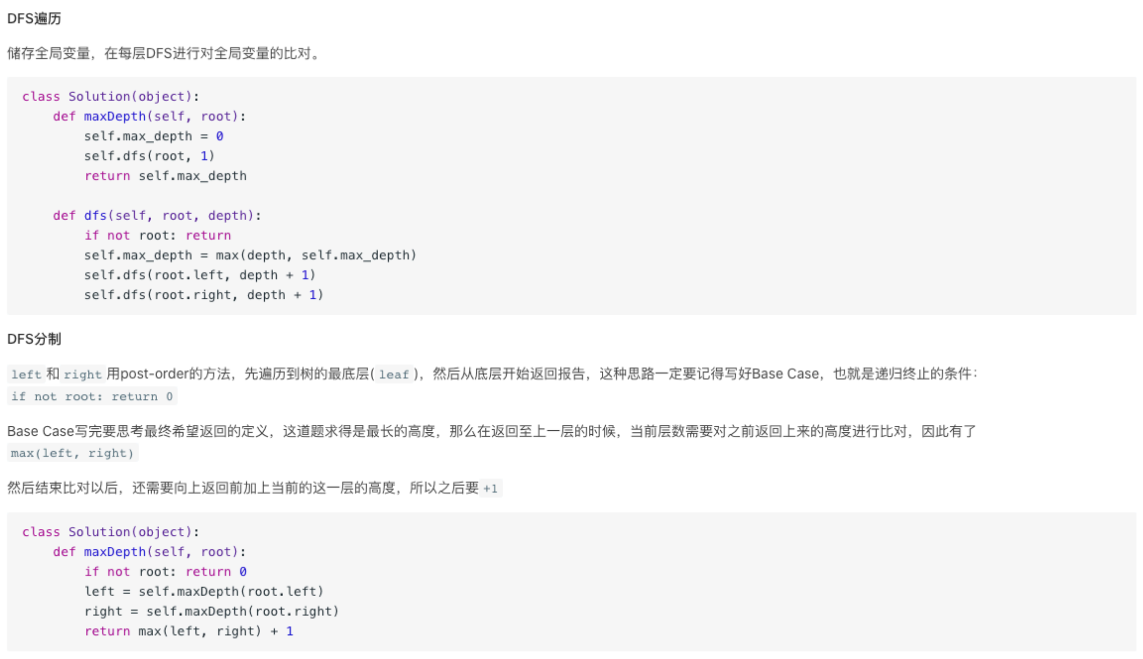
```
public int maxDepth(TreeNode root) {
    int max = 0;
    if (root == null) {return 0;}
    Stack<TreeNode> path = new Stack<>();
    Stack<Integer> sub = new Stack<>();
    path.push(root);
    sub.push(1);
    while (!path.isEmpty()) {
        TreeNode temp = path.pop();
        int tempVal = sub.pop();
        if (temp.left == null && temp.right == null) {max = Math.max(max, tempVal);}
        else {
            if (temp.left != null) {
                path.push(temp.left);
                sub.push(tempVal + 1);
            }
            if (temp.right != null) {
                path.push(temp.right);
                sub.push(tempVal + 1);
            }
        }
    }
    return max;
}
```

bfs

```
    public int bfs(TreeNode node){
        if(root == null) return 0;
        Queue<TreeNode> queue = new LinkedList();
        queue.offer(root);
        int level = 0;
        while(!queue.isEmpty()){
            int size = queue.size();
            for(int i = 0; i < size; ++i){
                TreeNode current = queue.poll();
                if(current.left != null) queue.offer(current.left);
                if(current.right != null) queue.offer(current.right);
            }
            level++;
        }
        return level;
    }
```

递归

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        return root ? 1 + max(maxDepth(root -> left), maxDepth(root -> right)) : 0;
    }
};
```



DFS遍历

储存全局变量，在每层DFS进行对全局变量的比对。

```
class Solution(object):
    def maxDepth(self, root):
        self.max_depth = 0
        self.dfs(root, 1)
        return self.max_depth

    def dfs(self, root, depth):
        if not root: return
        self.max_depth = max(depth, self.max_depth)
        self.dfs(root.left, depth + 1)
        self.dfs(root.right, depth + 1)
```

DFS分制

`left` 和 `right` 用post-order的方法，先遍历到树的最底层(`leaf`)，然后从底层开始返回报告，这种思路一定要记得写好Base Case，也就是递归终止的条件：`if not root: return 0`

Base Case写完要思考最终希望返回的定义，这道题求得是最长的高度，那么在返回至上一层的时候，当前层数需要对之前返回上来的高度进行比对，因此有了 `max(left, right)`

然后结束比对以后，还需要向上返回前加上当前的这一层的高度，所以之后要 `+1`

```
class Solution(object):
    def maxDepth(self, root):
        if not root: return 0
        left = self.maxDepth(root.left)
        right = self.maxDepth(root.right)
        return max(left, right) + 1
```