基本解法(BFS)

Java代码

```
class Solution {
    // 定义一棵二叉树
  public class TreeNode {
      // 定义节点的值
      int val;
      // 定义左子树
      TreeNode left;
      // 定义右子树
      TreeNode right;
      // 定义初始化方法
      TreeNode(int x) { val = x; }
 }
  public int maxDepth(TreeNode root) {
        if (root == null) {return 0;} // 考虑树为空的特殊情况 BFS无法自动处理
        Queue<TreeNode> queue = new LinkedList<TreeNode>();// 使用队列来记录各层节
点
        queue.offer(root);// 根节点入队
        int res = 0;// 目标值
        while (!queue.isEmpty()) { // 判断是否还有没有遍历完的节点
            int size = queue.size(); // 开始遍历新一层节点前，队列里即为新一层全部节点
            while (size > 0) {  // 需将这一层节点全都遍历完
                TreeNode node = queue.poll();// 遍历节点
                if (node.left != null) {
                    queue.offer(node.left);// 左子树入队列
                }
                if (node.right != null) {
                    queue.offer(node.right);// 右子树入队列
                }
                size--;
            }
            res++;// 新一层节点遍历完成，目标值 +1
        }
        return res;
    }

}
```

优化解法(DFS)

Java代码

```
class Solution {
    // 递归三要素
    // 1）确定函数等价关系式（参数，返回值）。参数是传入树的根节点，返回值是树的深度
    public int maxDepth(TreeNode root) {
        // 2)确定结束条件。如果节点为空，返回0
        // 且此处可以处理二叉树为空的corner case
        if(root == null) { return 0; }
        // 3)函数主功能。分别求左右子树最大深度，返回左右子树深度最大值 + 1
        //   即为当前节点为根节点的树的最大深度
        int leftTreeDepth=maxDepth(root.left);
        int rightTreeDepth=maxDepth(root.right);
        return Math.max(leftTreeDepth,rightTreeDepth)+1
    }
}
```

C++代码

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
  public:
    int maxDepth(TreeNode* root) {
        if (root == nullptr) return 0;
        return max(maxDepth(root->left), maxDepth(root->right)) + 1;
    }
};
```

Python代码

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def maxDepth(self, root):
        if root == None:
```

```
            return 0
        left_high = self.maxDepth(root.left)
        right_high = self.maxDepth(root.right)

        return max(left_high,right_high) + 1
```