LeetCode 第 104 号问题：二叉树的最大深度

> *本文首发于公众号「图解面试算法」，是 图解 LeetCode 系列文章之一。*
>
> *同步博客：https://www.algomooc.com*

今天分享的题目来源于 LeetCode 上第 104 号问题：二叉树的最大深度。题目难度为 Easy 。

题目描述

给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

说明: 叶子节点是指没有子节点的节点。

示例：

给定二叉树 `[3,9,20,null,null,15,7]`，

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

返回它的最大深度 3 。

题目解析 - DFS

最直接的办法就是使用DFS (深度优先搜索) 策略计算树的高度. 具体算法流程如下:

- **终止条件**：当前节点为空
- **返回值**：
  - 节点为空时，所以返回 0
  - 节点不为空时, 返回左右子树高度的最大值 + 1

动画描述

代码实现

```
/**
 * JavaScript 描述
 * DFS
 */
var maxDepth = function(root) {
    if (root == null) {
        return 0;
    }
    let leftHeight = maxDepth(root.left);
    let rightHeight = maxDepth(root.right);
```

```
    return Math.max(leftHeight, rightHeight) + 1;
};
```

精简版

```
var maxDepth = function(root) {
    return !root ? 0 : Math.max(maxDepth(root.left) + 1, maxDepth(root.right) + 1) ;
};
```

复杂度分析

- 时间复杂度：**O(n)**, 我们每个结点只访问一次，因此时间复杂度为 O(N)
- 空间复杂度：
  - 最坏情况下，树是完全不平衡的，例如每个结点只剩下左子结点，递归将会被调用 N 次（树的高度），因此保持调用栈的存储将是 O(N)。
  - 最好情况下（树是完全平衡的），树的高度将是 log(N)。因此，在这种情况下的空间复杂度将是 O(log(N))

题目解析 - BFS

求二叉树的深度也就是求二叉树有几层了, 采用 BFS (广度优先搜索) 策略对二叉树按层遍历.

实现 BFS 就要用到 '先进先出' 的队列了, 具体算法流程如下:

- 遍历二叉树节点，依次将当前节点 和它的左右子节点入队
- 依次出队, 出队子节点重复上一步操作

动画描述

代码实现

```
/**
 * JavaScript 描述
 * BFS
 */
var maxDepth = function(root) {
    let level = 0;
    if (root == null) {
        return level;
    }
    let queue = [root];
    while (queue.length) {
        let len = queue.length;
        while (len--) {
            let curNode = queue.pop();
            curNode.left && queue.unshift(curNode.left);
            curNode.right && queue.unshift(curNode.right);
        }
        level++;
    }
```

```
    return level;
};
```

复杂度分析

- 时间复杂度：**O(n)**
- 空间复杂度：**O(N)**