宽度优先搜索(BFS)与图论入门

主讲人 令狐冲
课程版本 v7.0

BFS 的适用场景

分层遍历
连通块问题
拓扑排序

BFS 的适用场景

- 分层遍历
    - 一层一层的遍历一个图、树、矩阵
    - 简单图最短路径
        - 简单图的定义是，图中所有的边长都一样
- 连通块问题
    - 通过图中一个点找到其他所有连通的点
    - 找到所有方案问题的一种非递归实现方式
- 拓扑排序
    - 实现容易度远超过 DFS

BFS 的三种实现方法

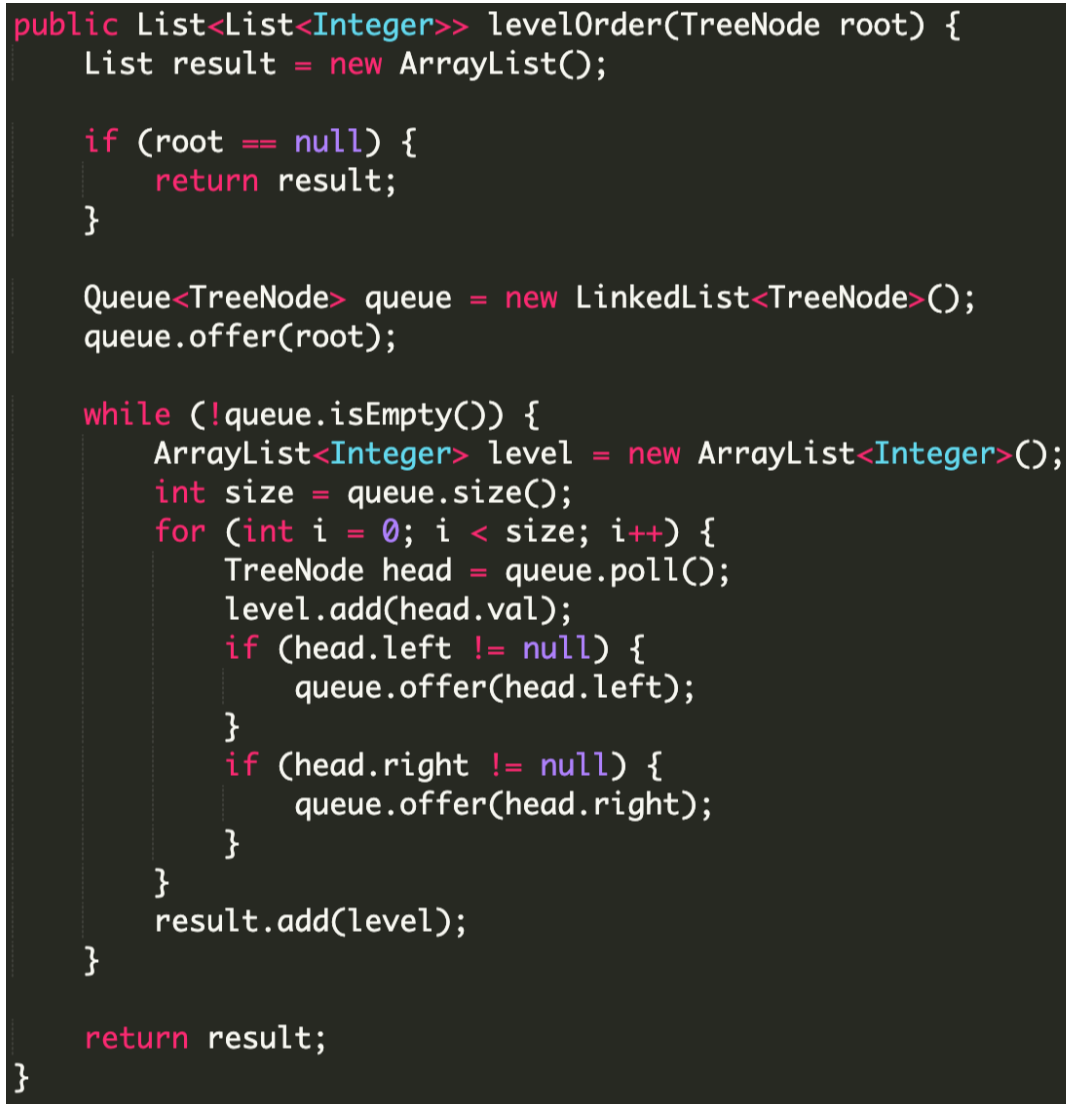
https://www.lintcode.com/problem/binary-tree-level-order-traversal

单队列

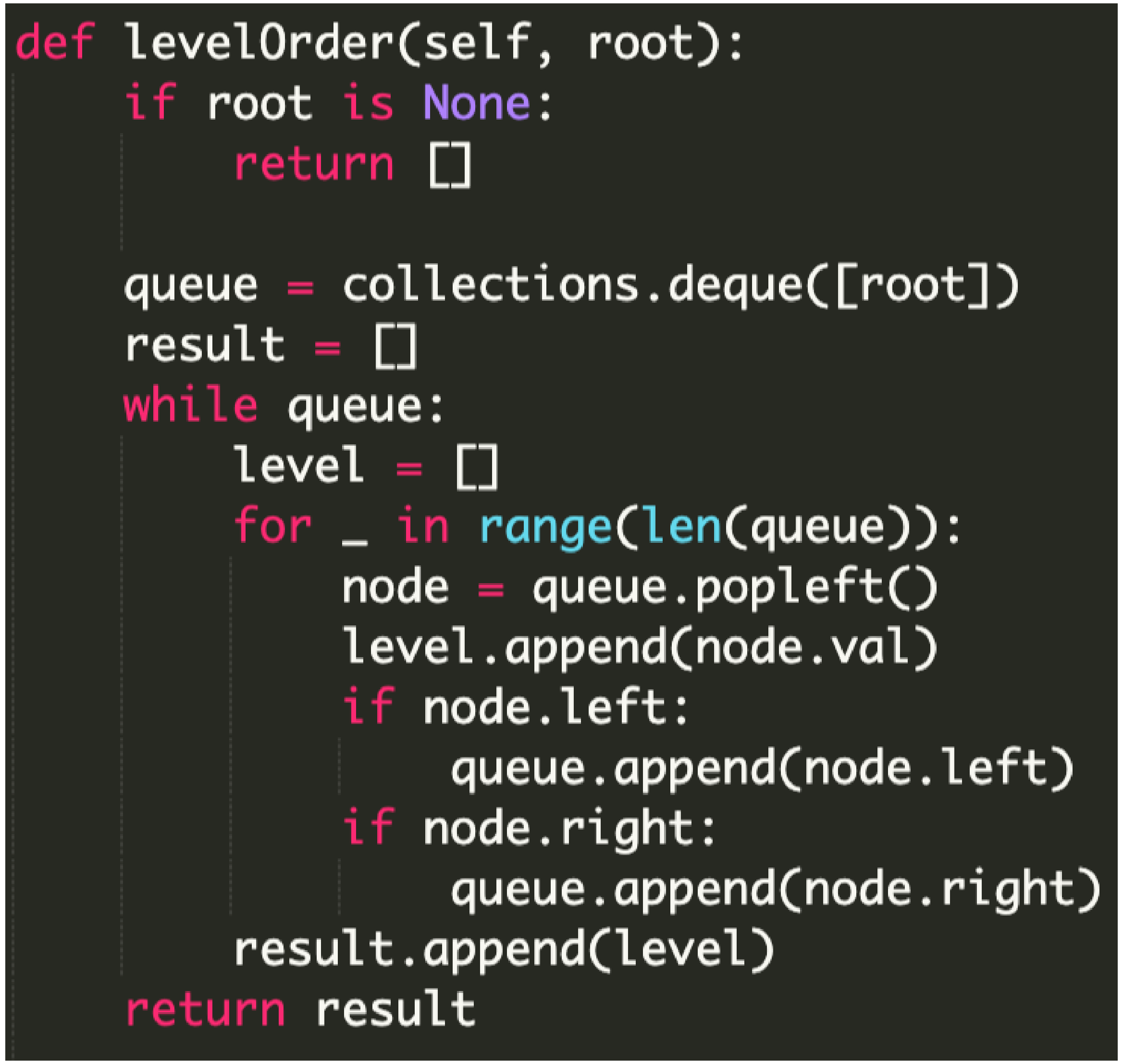
双队列

DummyNode

单队列实现方法

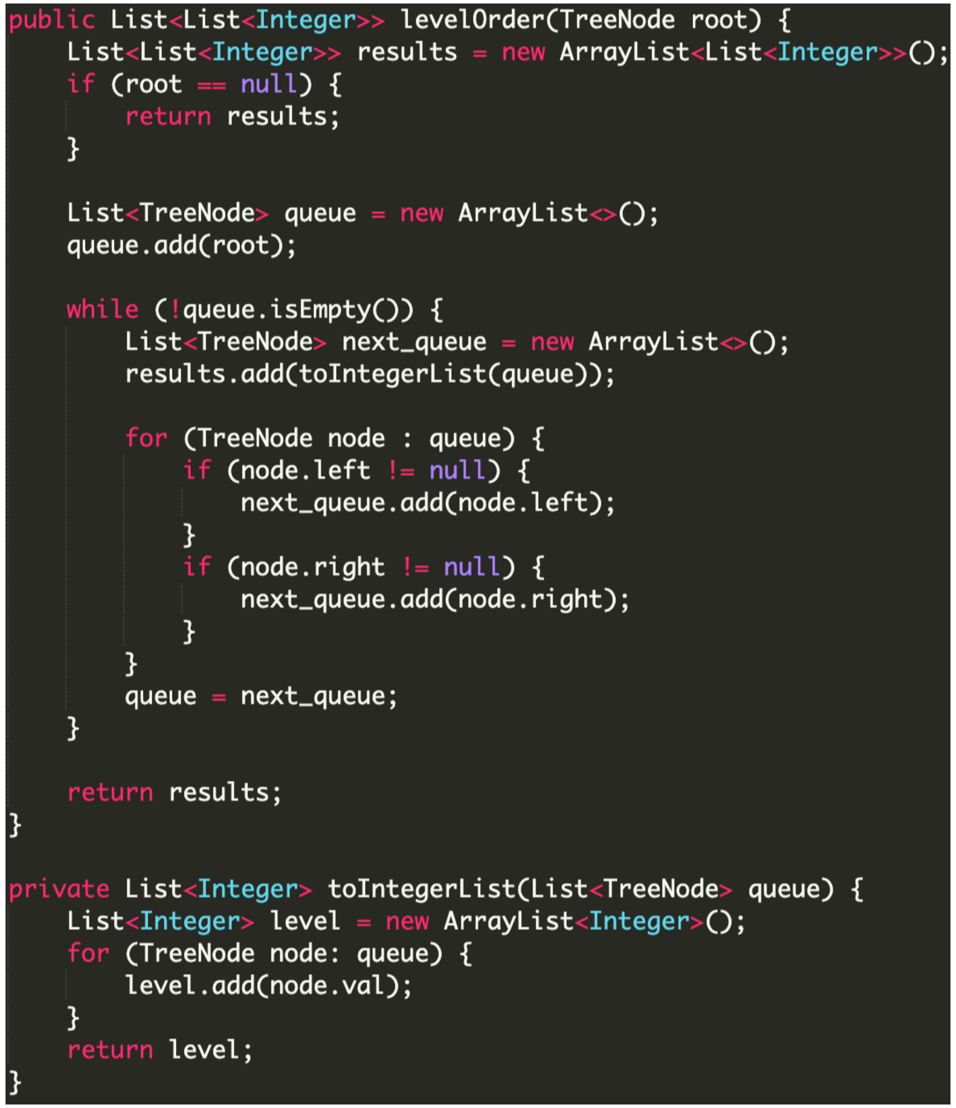


```java
public List<List<Integer>> levelOrder(TreeNode root) {
    List result = new ArrayList();

    if (root == null) {
        return result;
    }

    Queue<TreeNode> queue = new LinkedList<TreeNode>();
    queue.offer(root);

    while (!queue.isEmpty()) {
        ArrayList<Integer> level = new ArrayList<Integer>();
        int size = queue.size();
        for (int i = 0; i < size; i++) {
            TreeNode head = queue.poll();
            level.add(head.val);
            if (head.left != null) {
                queue.offer(head.left);
            }
            if (head.right != null) {
                queue.offer(head.right);
            }
        }
        result.add(level);
    }

    return result;
}
```

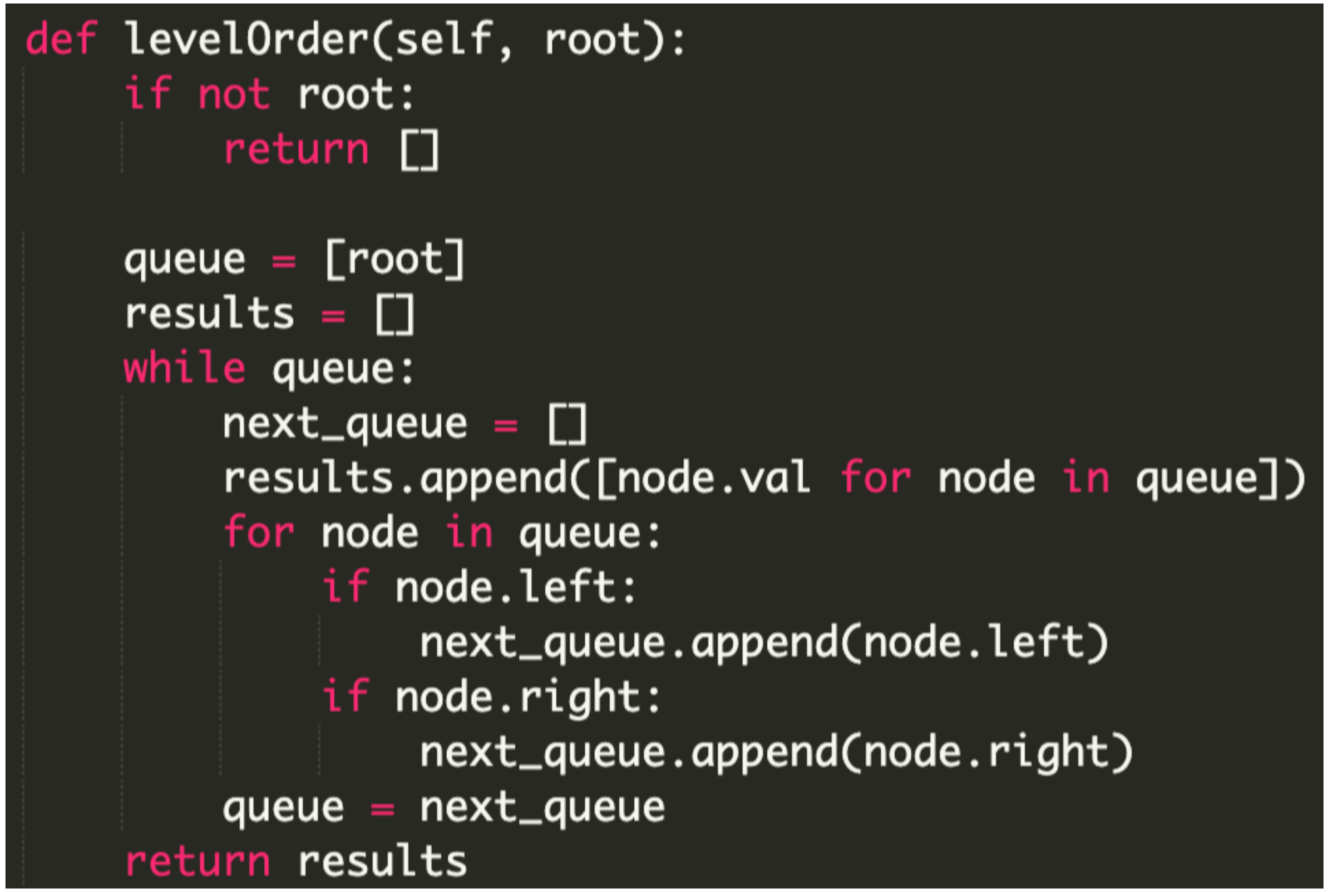


```python
def levelOrder(self, root):
    if root is None:
        return []

    queue = collections.deque([root])
    result = []
    while queue:
        level = []
        for _ in range(len(queue)):
            node = queue.popleft()
            level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        result.append(level)
    return result
```

双队列的实现方法

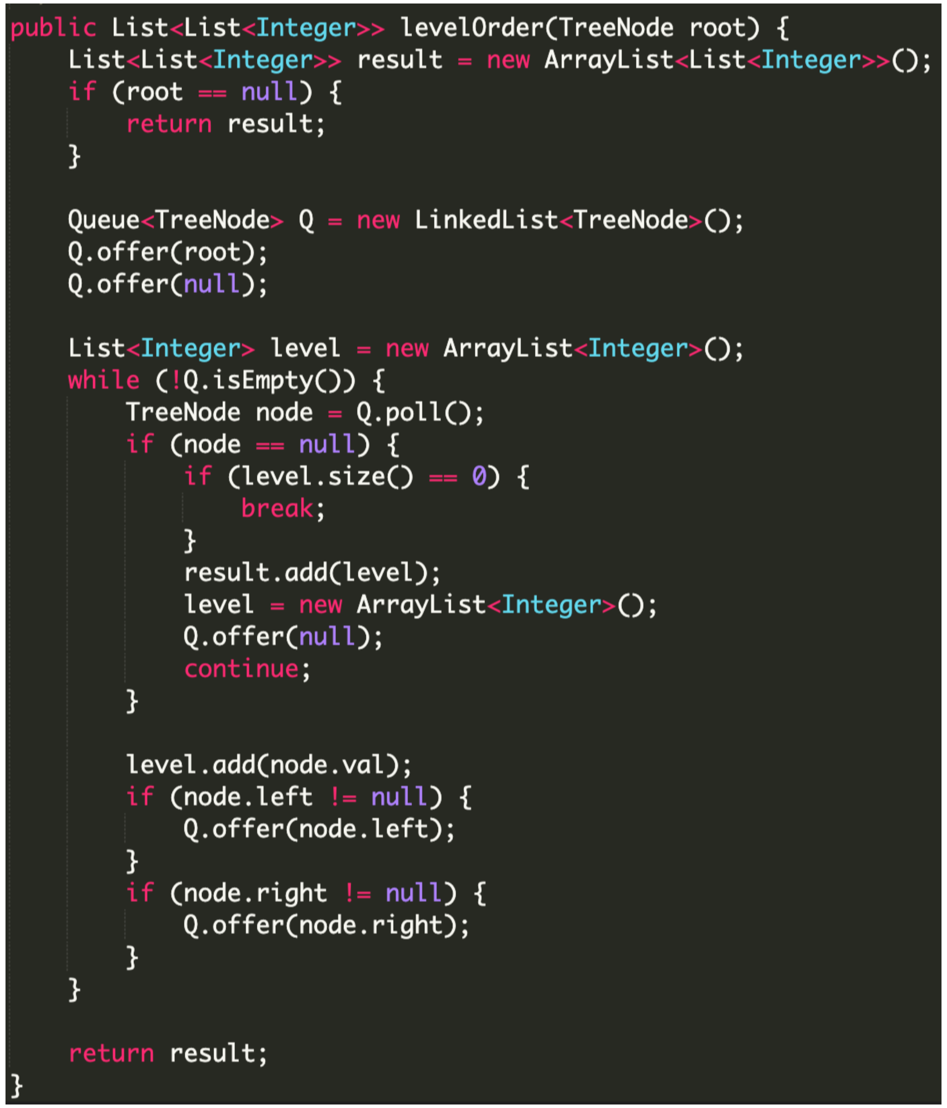


```java
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> results = new ArrayList<List<Integer>>();
    if (root == null) {
        return results;
    }

    List<TreeNode> queue = new ArrayList<>();
    queue.add(root);

    while (!queue.isEmpty()) {
        List<TreeNode> next_queue = new ArrayList<>();
        results.add(toIntegerList(queue));

        for (TreeNode node : queue) {
            if (node.left != null) {
                next_queue.add(node.left);
            }
            if (node.right != null) {
                next_queue.add(node.right);
            }
        }
        queue = next_queue;
    }

    return results;
}

private List<Integer> toIntegerList(List<TreeNode> queue) {
    List<Integer> level = new ArrayList<Integer>();
    for (TreeNode node: queue) {
        level.add(node.val);
    }
    return level;
}
```

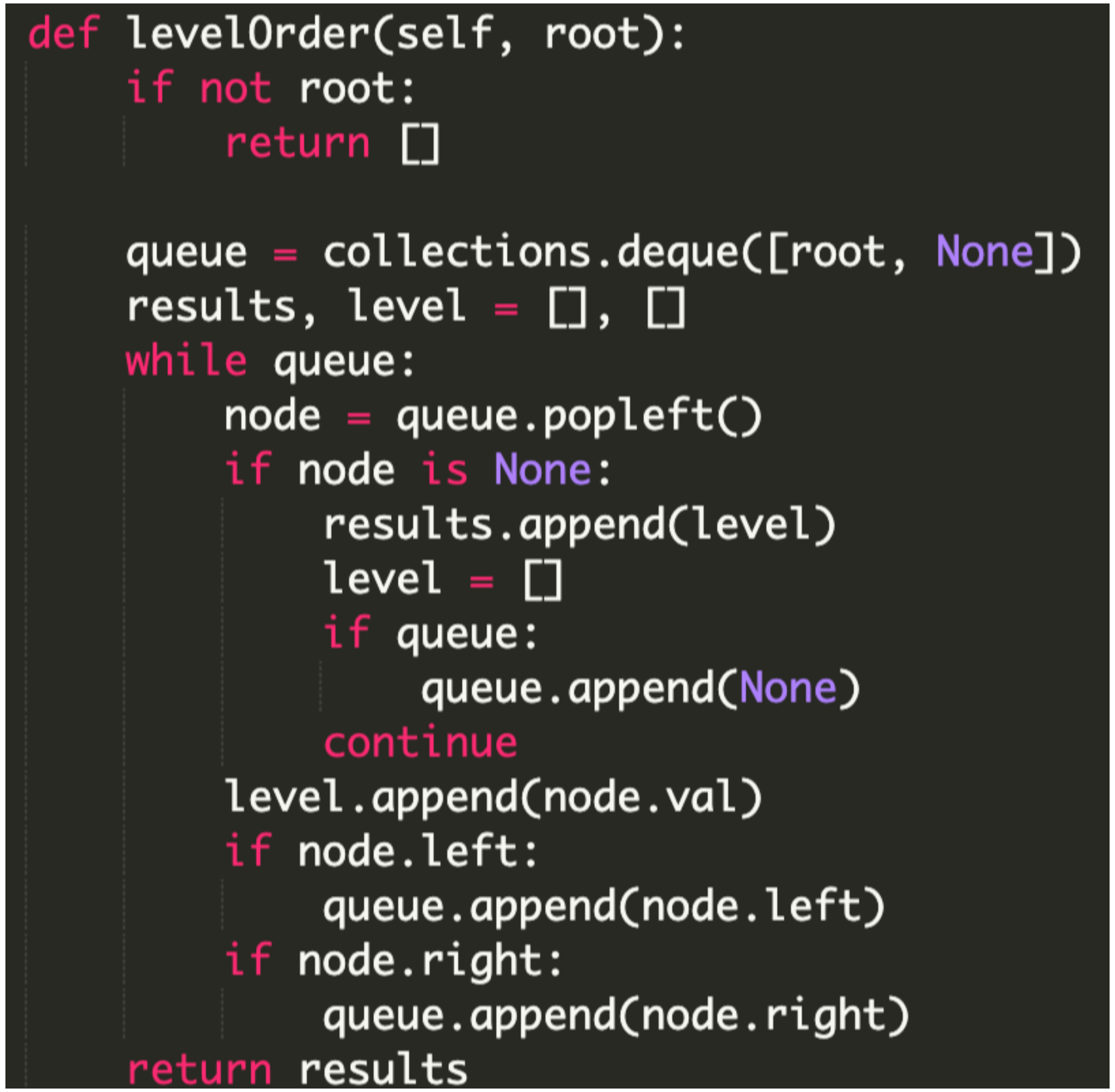


```python
def levelOrder(self, root):
    if not root:
        return []

    queue = [root]
    results = []
    while queue:
        next_queue = []
        results.append([node.val for node in queue])
        for node in queue:
            if node.left:
                next_queue.append(node.left)
            if node.right:
                next_queue.append(node.right)
        queue = next_queue
    return results
```

DummyNode 的实现方法



```java
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> result = new ArrayList<List<Integer>>();
    if (root == null) {
        return result;
    }

    Queue<TreeNode> Q = new LinkedList<TreeNode>();
    Q.offer(root);
    Q.offer(null);

    List<Integer> level = new ArrayList<Integer>();
    while (!Q.isEmpty()) {
        TreeNode node = Q.poll();
        if (node == null) {
            if (level.size() == 0) {
                break;
            }
            result.add(level);
            level = new ArrayList<Integer>();
            Q.offer(null);
            continue;
        }

        level.add(node.val);
        if (node.left != null) {
            Q.offer(node.left);
        }
        if (node.right != null) {
            Q.offer(node.right);
        }
    }

    return result;
}
```



```python
def levelOrder(self, root):
    if not root:
        return []

    queue = collections.deque([root, None])
    results, level = [], []
    while queue:
        node = queue.popleft()
        if node is None:
            results.append(level)
            level = []
            if queue:
                queue.append(None)
            continue
        level.append(node.val)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return results
```