Maximum Depth of Binary Tree (LeetCode 104 - Easy)

Solution

Recursive.

```
/**
 * LeetCode 104
 * DFS - recursive
 * @param root
 * @return maximum depth of the tree.
 */
public static int maxDepth_1(TreeNode root) {
	if (root == null) {
		return 0;
	}
	return Math.max(maxDepth_1(root.left), maxDepth_1(root.right)) + 1;
}
```

DFS – iterative

```
/*
 * DFS - iterative
 */
public static int maxDepth_2(TreeNode root) {
	if (root == null) {
		return 0;
	}

	Stack<TreeNode> node_stack = new Stack<TreeNode>();
	Stack<Integer> depth_stack = new Stack<Integer>();

	node_stack.push(root);
	depth_stack.push(1);

	int rv = 1;
	while (!node_stack.isEmpty()) {

		TreeNode cur = node_stack.pop();
		int cur_depth = depth_stack.pop();

		if (cur.left == null && cur.right == null) {
			rv = Math.max(rv, cur_depth);
			continue;
		}
		if (cur.left != null) {
			node_stack.push(cur.left);
			depth_stack.push(cur_depth + 1);
		}
		if (cur.right != null) {
			node_stack.push(cur.right);
			depth_stack.push(cur_depth + 1);
		}
	}
	return rv;
}
```

BFS – iterative

```
/*
 * BFS - iterative
 */
public static int maxDepth_3(TreeNode root) {
	if (root == null) {
		return 0;
	}

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

	int rv = 0;
	while (!queue.isEmpty()) {
		int size = queue.size();
		for (int i = 0; i < size; i++) {
			TreeNode cur = queue.poll();
			if (cur.left != null) {
				queue.add(cur.left);
			}
			if (cur.right != null) {
				queue.add(cur.right);
			}
		}
		rv++;
	}
	return rv;
}
```