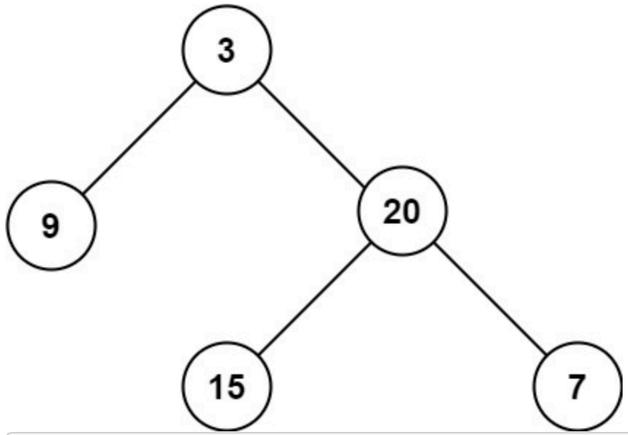
Maximum Depth of Binary Tree

Given the `root` of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: 3
```

Example 2:

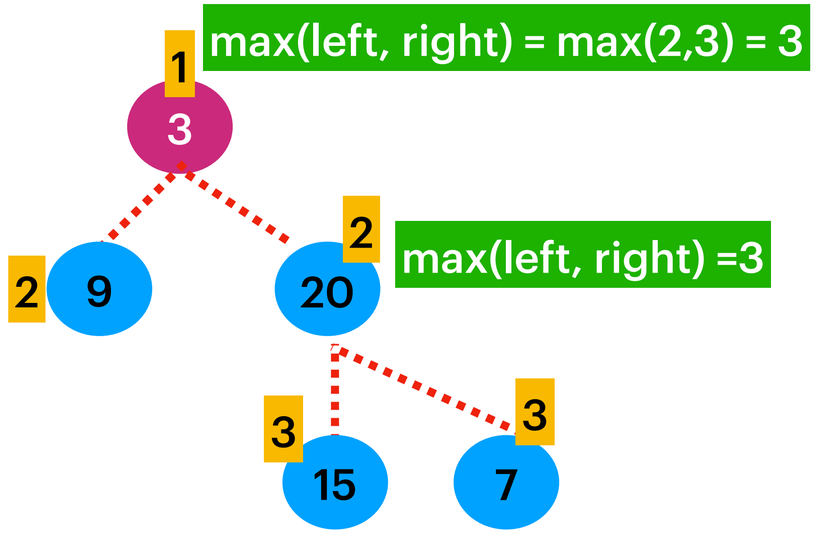
```
Input: root = [1,null,2]
Output: 2
```

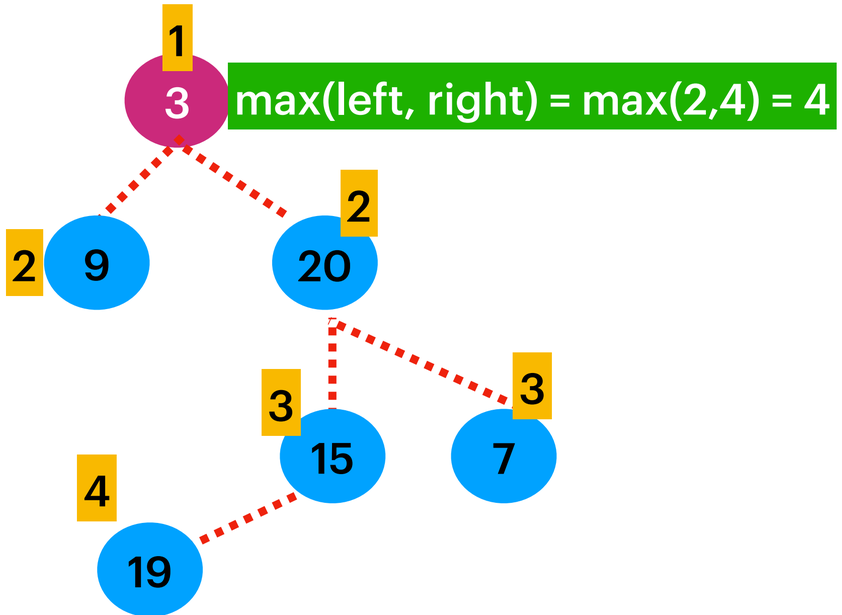
Constraints:

- The number of nodes in the tree is in the range `[0, 10^4]`.
- `-100 <= Node.val <= 100`

Maximum Depth of Binary Tree Recursive

From Root Find the depth of leftSubTree & Find the depth of Right Subtree, take max(leftDept, rightDepth)



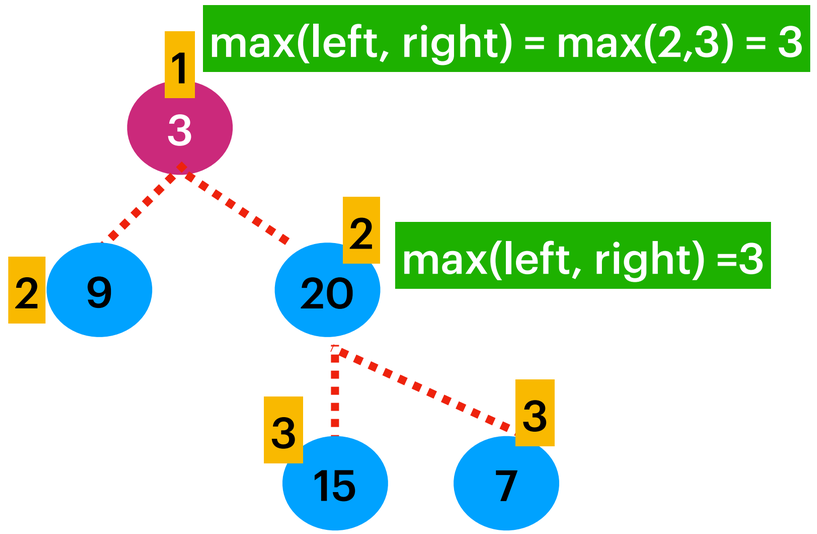


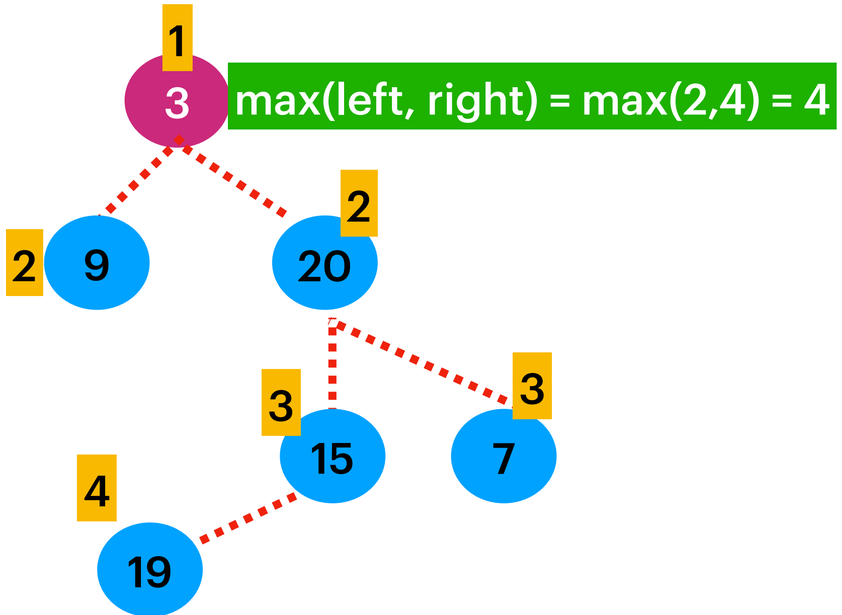

```
public int maxDepth(TreeNode root) {

    if(root == null)
    {
    return 0;
    }

    int leftDepth = 1 + maxDepth(root.left);
    int rightDepth = 1 + maxDepth(root.right);

    return Math.max(leftDepth, rightDepth);
    }
```

Time Complexity : O(n)
Space Complexity : O(n)

Maximum Depth of Binary Tree Iterative LevelOrder [BFS]

From Root Find the depth of leftSubTree & Find the depth of Right Subtree, take max(leftDept, rightDepth)



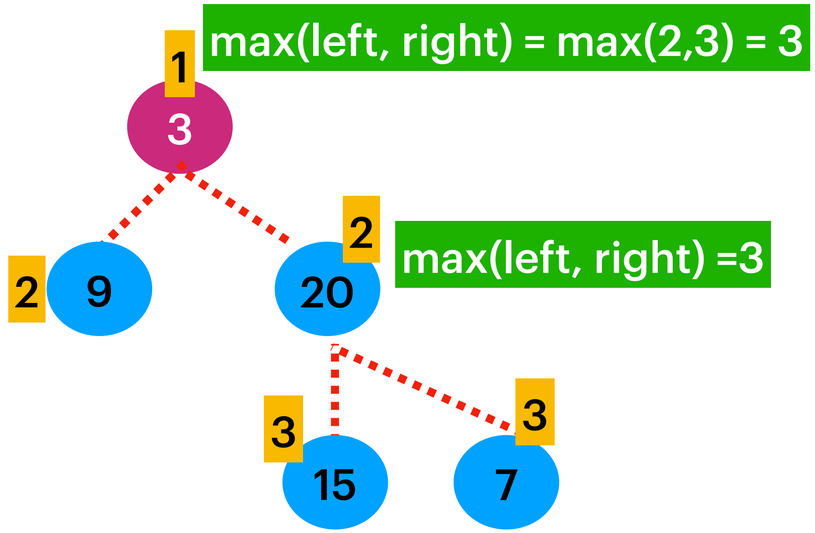


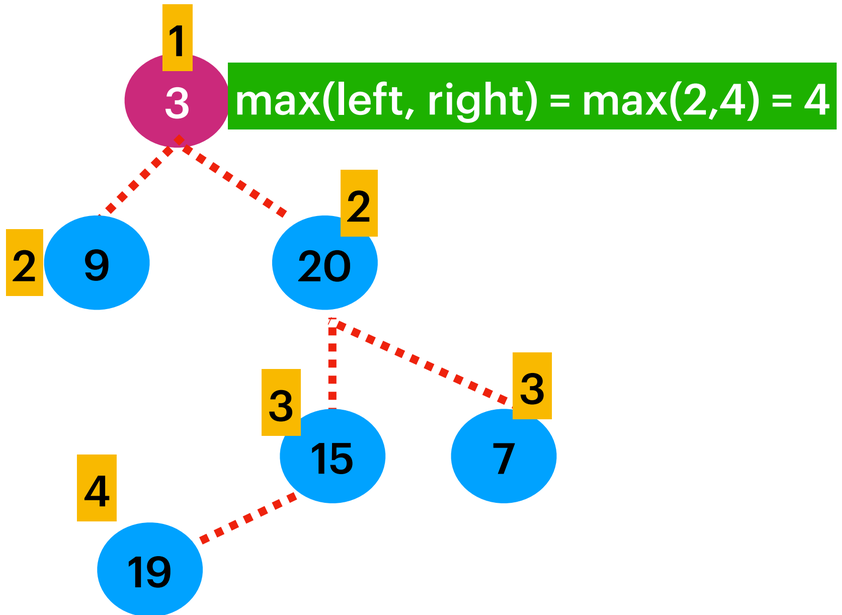
Time Complexity : O(n)
Space Complexity : O(n)

```
public int maxDepth(TreeNode root) {

if(root == null)
{
return 0;
}
Queue<TreeNode> queue = new LinkedList<>();
queue.add(root);
int depth = 0;

while(!queue.isEmpty())
{
depth++;

int levelElementSize = queue.size();
while(levelElementSize > 0)
{
TreeNode current = queue.poll();
if(current.left != null)
{
queue.add(current.left);
}

if(current.right != null)
{
queue.add(current.right);
}
levelElementSize --;
}

}

return depth;
}
```

Maximum Depth of Binary Tree Iterative LevelOrder [DFS]

From Root Find the depth of leftSubTree & Find the depth of Right Subtree, take max(leftDept, rightDepth)





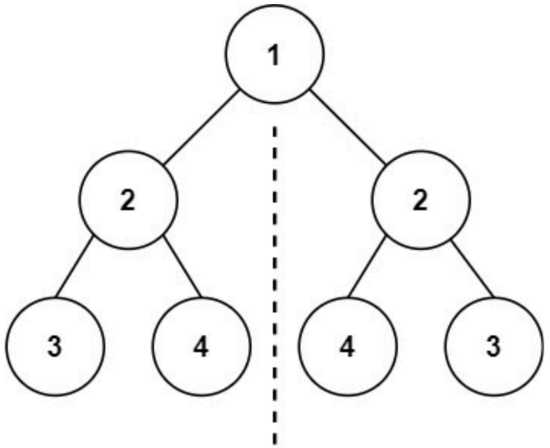
Time Complexity : O(n)
Space Complexity : O(n)

```
public int maxDepth(TreeNode root) {

if(root == null)
{
return 0;
}

Stack<TreeNode> treeStack = new Stack<>();
Stack<Integer>  depthStack = new Stack<>();
treeStack.push(root);
depthStack.push(1);

int depth = 0 ; int currentDepth = 0;

while(!treeStack.isEmpty())
{
TreeNode current = treeStack.pop();
currentDepth = depthStack.pop();

if(current != null)
{
depth = Math.max(depth, currentDepth);
treeStack.push(current.left);
treeStack.push(current.right);
depthStack.push(currentDepth+1);
depthStack.push(currentDepth+1);

}
}

return depth;
}
```

Symmetric/Mirror Tree

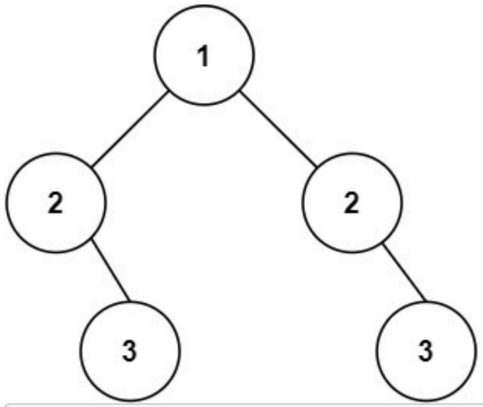
Given the `root` of a binary tree, *check whether it is a mirror of itself* (i.e., symmetric around its center).

Example 1:




```
Input: root = [1,2,2,3,4,4,3]
Output: true
```

Example 2:



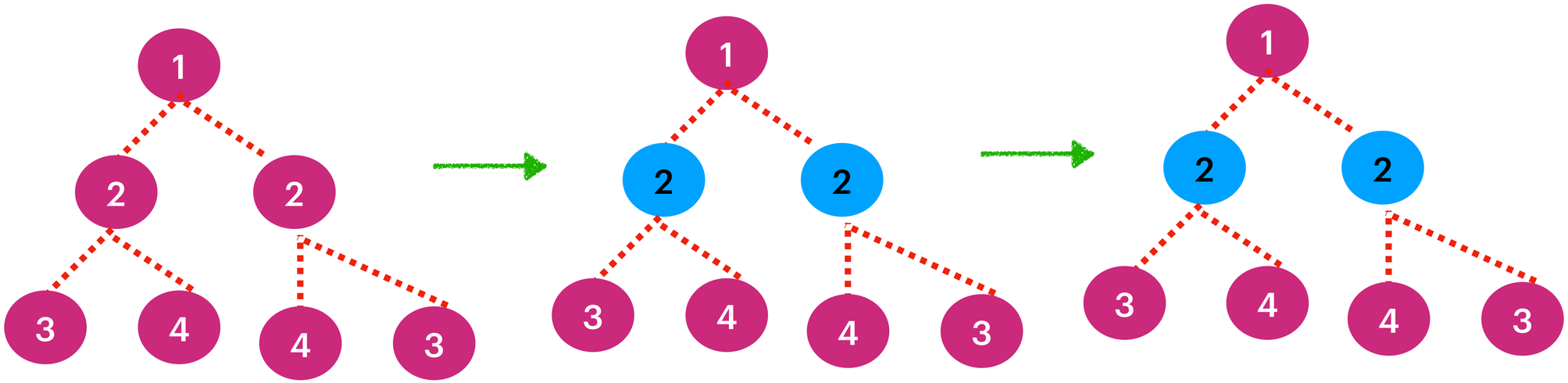

```
Input: root = [1,2,2,null,3,null,3]
Output: false
```

Constraints:

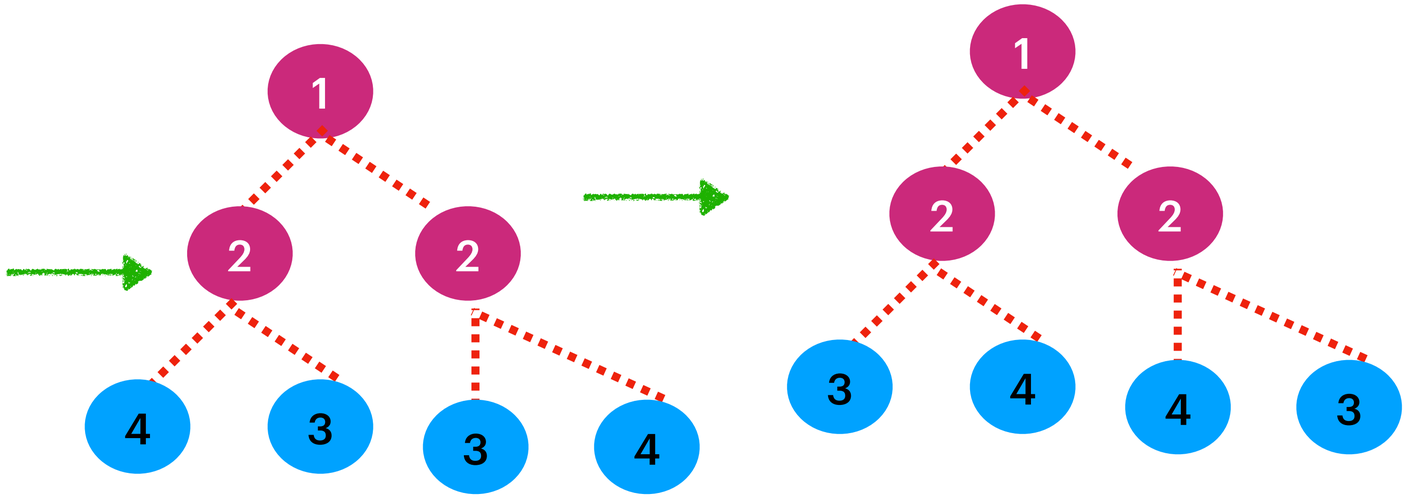
- The number of nodes in the tree is in the range `[1, 1000]`.
- `-100 <= Node.val <= 100`

Follow up: Could you solve it both recursively and iteratively?

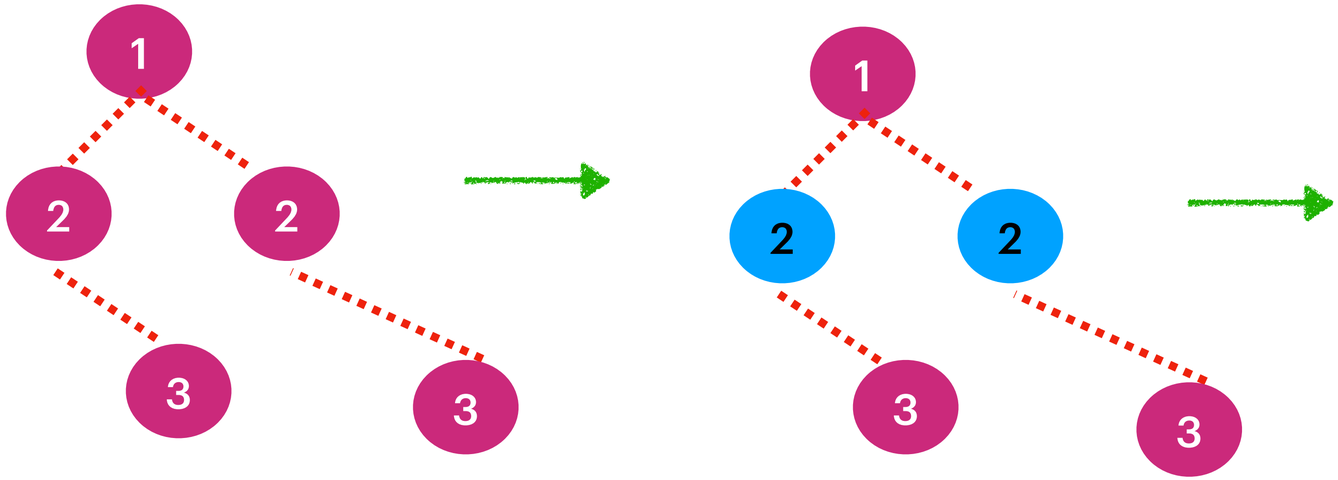
Symmetric/Mirror Tree

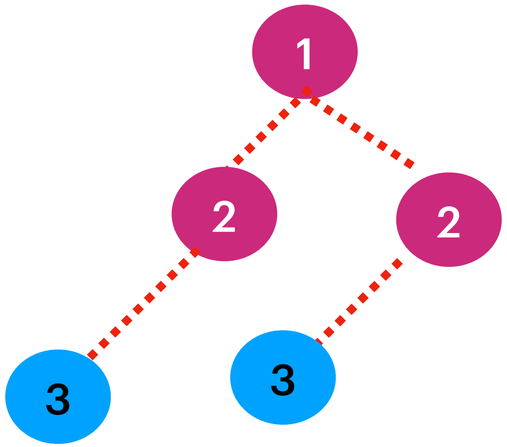


Symmetric/Mirror Tree = True

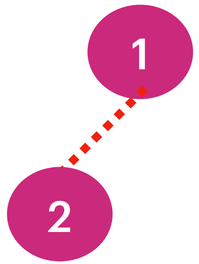


Symmetric/Mirror Tree





Symmetric/Mirror Tree = False



Symmetric/Mirror Tree = False