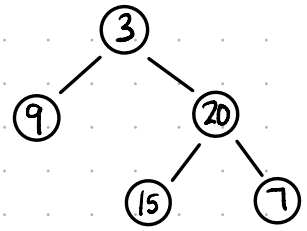
Maximum Depth of Binary Tree

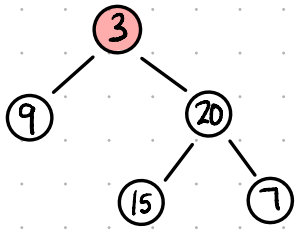
basically looking for height of tree

Method 1: Post-Order DFS - Recursion



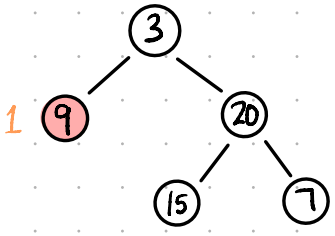

```
def maxDepth(root):
    # Base Case
    if not root:  # for None
        return 0
    # Recursive Case
    left = maxDepth(root.left)
    right = maxDepth(root.right)

    return 1 + max(left, right)
```

Process



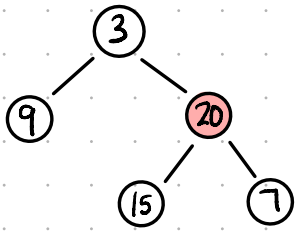
For Node 3:

- Call maxDepth(3), which calls maxDepth(9) and maxDepth(20)



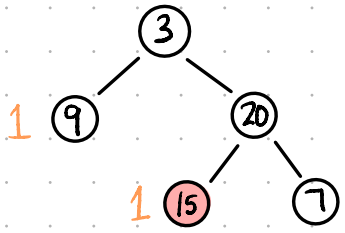
For Node 9:

- Call maxDepth(9), which checks its left and right children. Since (9) has no children, it calls maxDepth(None) for both left and right.
- maxDepth returns 0 because it's a leaf node
- Now, for (9), left = 0 and right = 0. So, the depth of (9) is calculated as 1 + max(0,0) = 1



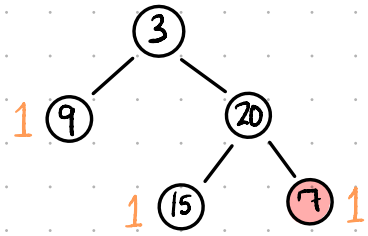
For Node 20:

- Call maxDepth(20), which calls maxDepth(15) & maxDepth(7)



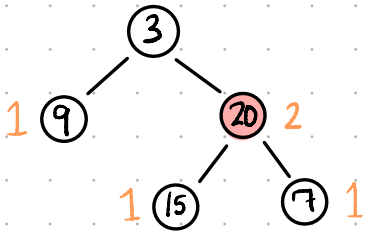
For Node 15:

- Call maxDepth(15), no children on left and right
- maxDepth returns 0
- For (15), 1 + max(0, 0) = 1



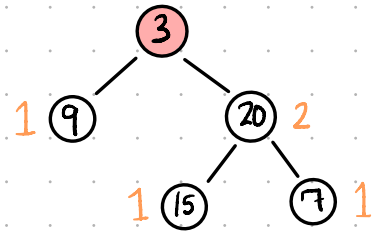
For Node 7:

- Call maxDepth(7), no children on left and right
- maxDepth returns 0
- For (7), 1 + max(0, 0) = 1



Back to Node 20:

- For (20), we have left = 1 and right = 1
- Depth of (20) is 1 + max(1 + 1) = 2

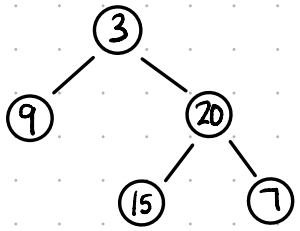


Back to Node 3:

- For (3), we have left = 1 and right = 2
- Depth of (3) is 1 + max(1, 2) = 3

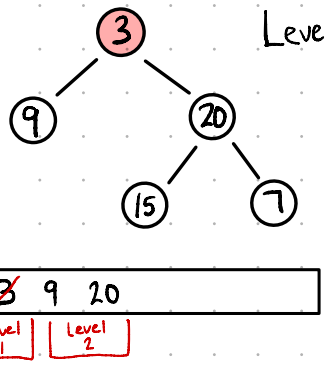
Function returns 3

Method 2: BFS - Iterative



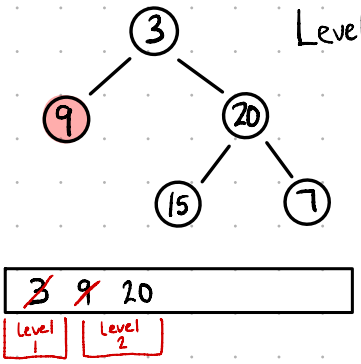

```
def maxDepth(root):

    if not root:
        return 0

    level = 0
    queue = [root]
    while queue:
        for i in range(len(queue)):
            node = queue.pop(0)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        level += 1
    return level
```

Process



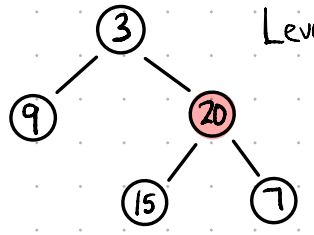
For Node 3:

a root exists so we'll initialize level to 0 and add the root to the queue. Now let's pop it and since it has left and right children lets add them (9) (20). That's the only node at that level so the for loop ends. Increment Level by 1.

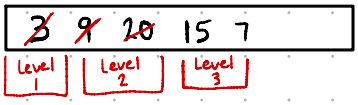


For Node 9 and 20:

At this level only 2 nodes exist. We Pop 9 and it has no children so nothing is appended.

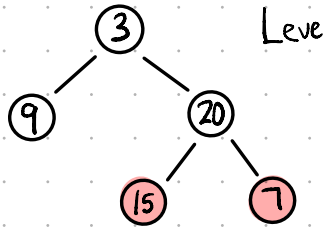


Level = ~~0~~ ~~1~~ 2

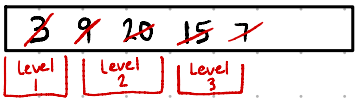


For Node 20:

Now for node 20, we'll pop it and add it's children (15) and (7) to the queue. No more nodes, so for loop ends. Increment level by 1.



Level = ~~0~~ ~~1~~ ~~2~~ 3

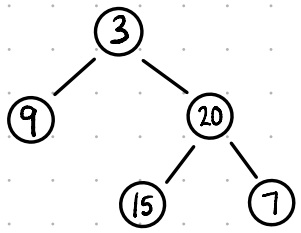


For Node 15 & 7:

Pop Node 15, then Pop Node 7. No Children to append, so for loop ends. Increment Level by 1. No more nodes in the queue, so while loop ends.

Function returns 3

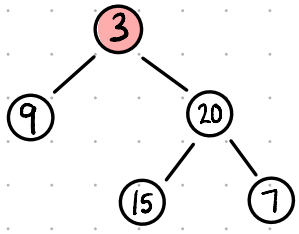
Method 3: Pre Order DFS - Iterative




```
def maxDepth(root):

    stack = [[root, 1]] # root node & it's depth level
    res = 0

    while stack:
        node, depth = stack.pop()

        if node: # if current node not null
            res = max(depth, res)
            stack.append([node.left, depth+1])
            stack.append([node.right, depth+1])
    return res
```

Process

20,2
9, 2
~~3,1~~

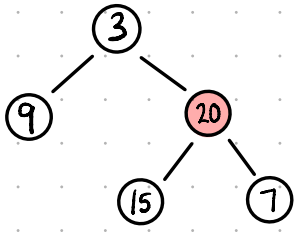


res = ~~0~~ 1

For Node 3:

We initialize the stack w/ node 3 - the root. We enter the while loop and pop node 3, update res to node 3 depth = 1, and append it's children (9) and (20).

7, 3
15, 3
~~20,2~~
9, 2
~~3,1~~

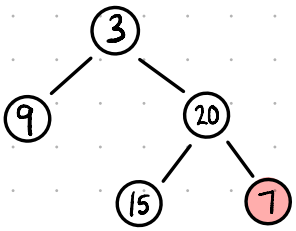


res = ~~0~~ ~~1~~ 2

For Node 20:

It's on top of the stack, so it's going to get popped and add it's children (15) and (7). After we update res to node 20 depth = 2.

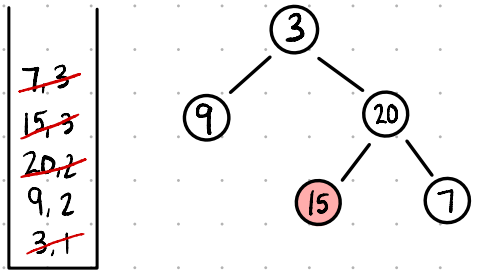
~~7, 3~~
15, 3
~~20,2~~
9, 2
~~3,1~~



res = ~~0~~ ~~1~~ ~~2~~ 3

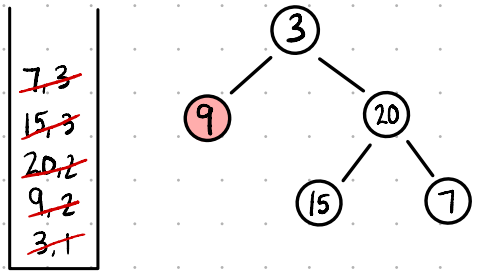
For Node 7:

Pop 7, no children to add. Update res to 3.



res = ~~0~~ ~~1~~ ~~2~~ 3

For Node 15:
Pop 15, no children to add. Res stays the same.



res = ~~0~~ ~~1~~ ~~2~~ 3

For Node 9:
Pop 9, no children to add. Res stays the same.

Function returns 3