[104. Maximum Depth of Binary Tree]

https://leetcode.com/problems/maximum-depth-of-binary-tree/description/

Problem Summary

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

- The **maximum depth** is the number of nodes **along the longest path** from the root down to the farthest leaf.

Note: In this problem, depth = number of nodes on the path (not number of edges), so the root itself contributes 1.

🧠 Recursive Approach

👉 Base Case:

- If the tree is empty (root == None), return 0.

👉 Recursive Step:

- For any node, compute the depth of its left subtree: left = maxDepth(root.left)
- Compute the depth of its right subtree: right = maxDepth(root.right)
- Return max(left, right) + 1 because:
    - You want the deeper side.
    - And you add 1 to include the current node.

💻 Python Code

```
# Definition for a binary tree node.
class TreeNode:
  def __init__(self, val=0, left=None, right=None):
    self.val = val
    self.left = left
    self.right = right
```

```
        class Solution:
def maxDepth(self, root: Optional[TreeNode]) -> int:
    if not root:
        return 0
    left = self.maxDepth(root.left)
    right = self.maxDepth(root.right)
    return max(left, right) + 1
```

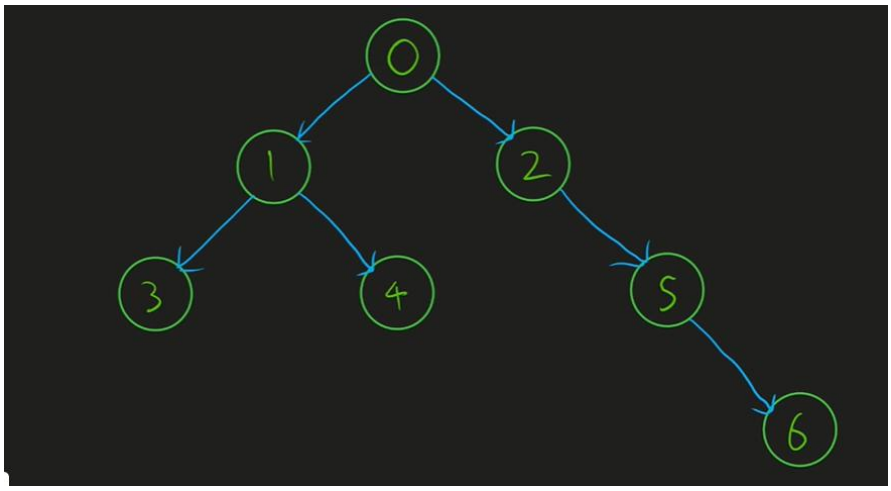
⏱️ Time and Space Complexity

Metrics	Values
Time	O(n) — every node is visited once
Space	O(h) — recursion stack goes up to height of the tree (h)

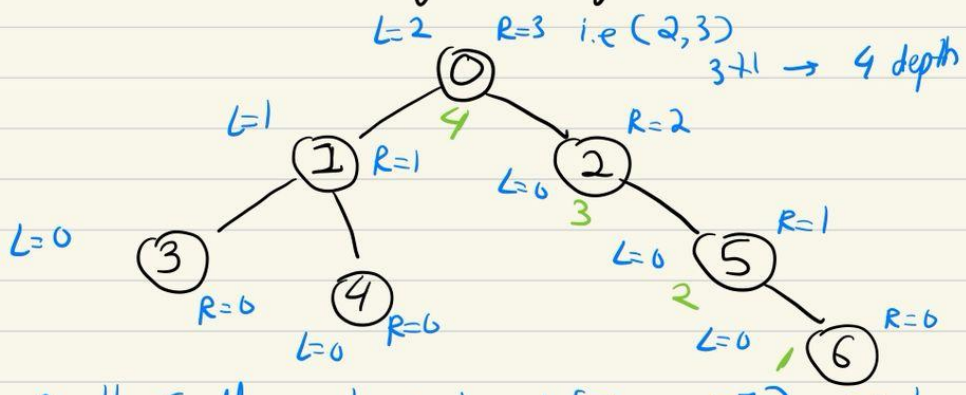
🧠 Conceptual Notes

- The function is **recursive**, and every call is isolated with its own left and right values.
- This is why you can think of every subtree as a small problem being solved independently — a key recursive idea in tree problems.

Dry Run following Tree for example:



② Maximum Depth of Binary Tree




```
def maxDepth(self, root: Optional[TreeNode]) -> int

    if not root:          — check 1
        return 0

    left = self.maxDepth(root.left)  → check 2
    right = self.maxDepth(root.right) → check 3
    return max(left, right)
```

#1 root i.e 0 ≠ null check1

left = maxDepth(root.left i.e 0.left) check2
→ 1

#1.1
root i.e 1 ≠ null

left = maxDepth(root.left i.e 1.left) check 2
1 → 3

1.1.1 root i.e 3 maxDepth(self, 3) → root

3 ≠ null ✓

left = self.maxDepth(3.left)
3 → null

1.1.1.L
root i.e null

null = null ✓
return 0

Goes back to # 1.1.1 left=0

right = self.maxDepth(3.right)
3 → null

1.1.1.R
root null i.e null = null ✓ return

Goes back to 1.1.1 Right=0

return max(left, right) + 1
(0, 0) + 1
return 1

Goes back to 1.1 left=1 i.e by the whole above flow we got this

Right = self.maxDepth(self.right)
1 → 4

1.1.R root=4

4 ≠ null

left = self.maxDepth(root.left)
4 → null

1.1.R.L

root null
i.e null = null ✓ return 0

Goes back to 1.1.R left=0

right = self.maxDepth(root.right)
4 → null

1.1.R.R

root => null i.e null = null return 0

Goes back to 1.1.R Right=0

return max(left, right) + 1
0, 0 + 1

return 1

Goes back to 1.1 Right=1 because it got its call from right

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

return 2

Goes back to 1 left = 2 → from the whole detailed recursive call

right = self.maxDepth(self.right)
1 → 2

# 1.R
root = 2 i.e 2 ≠ null

left = self.maxDepth(root.left)
2 → null

1.R.1
root → null = null return 0

Goes back to 1.R left = 0

right = self.right(2.right)
2 → 5

1.R.R
root i.e 5 ≠ null

left = self.left(5.left)
5 → null)

1.R.R.1
root i.e null = null

Goes back to 1.RR left = 0

right = self.right(5.right)
5 → 6

1.R.R.R root = 6 i.e 6 ≠ null

left = self.maxDepth (6.root)
6 → null

1.R.R.R.1 null = null return 0

Goes back to 1.R.R.R left = 0

right = self.maxDepth (6.root)
6 → null

1.R.R.R.1

null = null return 0

Goes back to 1.R.R.R right = 0

return max(right, left) + 1
0, 0
1

Goes back to 1.R.R right = 1

return max(0, 1) + 1
1

return 2

Goes back to 1.R right = 2

return max(left, right) + 1
0, 2

return 3

Goes back to 1

return max(left, right) + 1

2, 3

3 + 1

return 4

max depth = 4