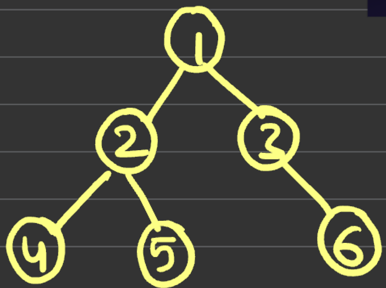
Your Ultimate Guide To Landing Top AI roles

DECODE AiML

2.15.1

Count no of Nodes in a Binary Tree

→ Input: root node of binary Tree
Output: integer n ← no of nodes in a BT



→ Let's look for Recursive Solution

① Base Case

→ root is None → return 0

② Recursive Case

→ recurrence relation.

$$\boxed{\begin{aligned} NN(T) &= 1 + NN(LST) + NN(RST) \\ &= 0, \text{ if } T \text{ is None} \end{aligned}}$$

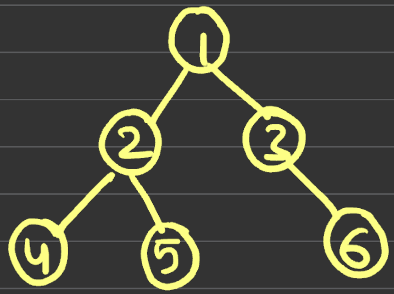
Time & Space Complexity

Time Complexity $= O(n)$ ← whenever we visit a node, we spend constant time $(n \times 3 \times c)$

Space Complexity $= O(n)$ ← depth of recursion tree = depth of binary Tree.

Code and Dry Run

```python
def count_nodes(root):
    if root is None:
        return 0
    return 1+ count_nodes(root.left)
              + count_nodes(root.right)
```

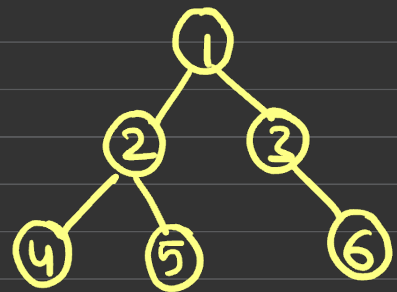


2.15.2

Count no of leaves in a Binary Tree

→ Input : root node of binary Tree
Output : integer Cnt → no of leaf nodes in a Binary Tree

→ Let's look for Recursive Solution



① **Base case**
- → T is a leaf node → return 1
- → T is empty → return 0

② **Recursive Case**
- → recurrence relation.

$$NL(T) = \begin{cases} NL(LST) + NL(RST) \\ 1, \text{ if } T \text{ is a leaf} \\ 0, \text{ if } T \text{ is Empty} \end{cases}$$

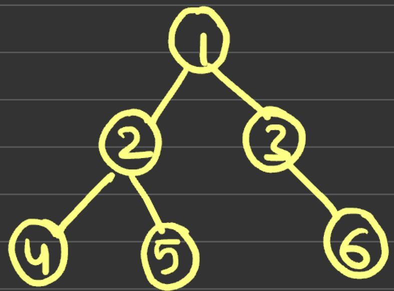
Time & Space Complexity

Time Complexity $= O(n)$ ← whenever we visit a node, we spend constant time $(n \times 3 \times c)$

Space Complexity $= O(n)$ ← depth of recursion tree = depth of binary Tree.

Code and Dry Run

```
def count_leaves(root):
    if root is None:
        return 0
    if root.left==None and root.right==None
        return 1
    return count_leaves(root.left)
        + count_leaves(root.right)
```



2.15.3

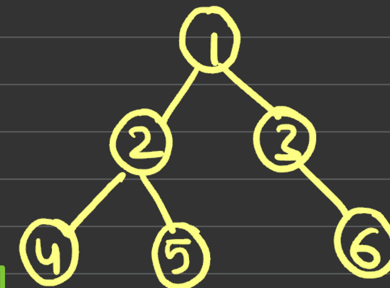
Find the height of a Binary Tree

→ Input : root node of binary Tree
Output : integer h ← max depth/height of Binary Tree

→ Let's look for Recursive Solution

① <u>Base case</u>
→ root is None → return 0

② <u>Recursive Case</u>
→ recurrence relation.



$$\boxed{\text{Height}(T) = \begin{cases} 1 + \max(\text{Height}(LST), \text{Height}(RST)) \\ 0 \quad \text{if } T \text{ is empty.} \end{cases}}$$

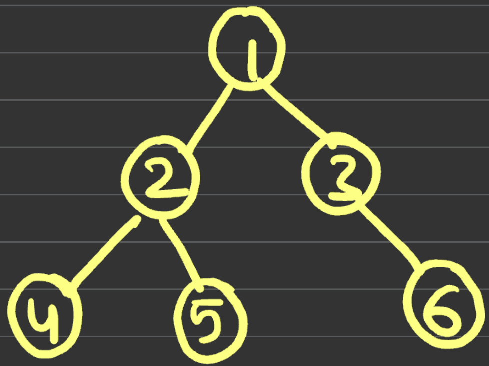
Time & Space Complexity

Time Complexity = O(n) ← whenever we visit a node, we spend constant time (n x 3 x c)

Space Complexity = O(n) ← depth of recursion tree = depth of binary Tree.

Code and Dry Run

```
def height(root):
    if root is None:
        return 0
    return 1+ max(height(root.left),
                  height(root.right))
```



DECODE AiML

Like Subscribe