Morris Inorder Traversal

Left Root Right

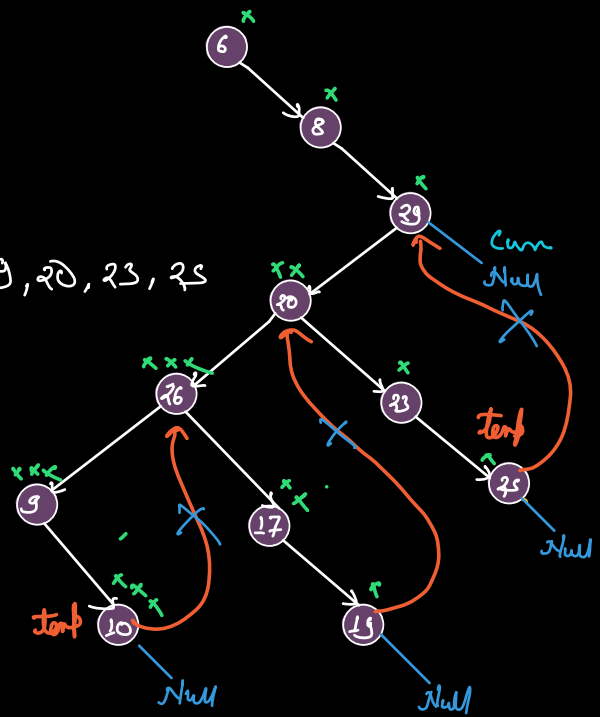
6, 8, 9, 10, 26, 17, 19, 20, 23, 25

Morris

Last node to be visited will be the right most node.

↓

Will not have a right child



```
While ( cun != null ) {

    if( cun. left == null){
        Print(cun.val);
        Cun = cun. right,
    }
    else {
        temp = cun. left;
        While (temp.right != null && temp. right != curr){
            temp = tep.right;
        }
        if ( temp . right == null) {
            temp.right = curr;
            Cun = cun.left;
        }
        else {
            temp. right = null;
            Print(cun. val);
            Cun = Cun. right;
        }
}
```

TC : O(N)

Every node being visited at most 3 times.

SC : O(1)

HW:

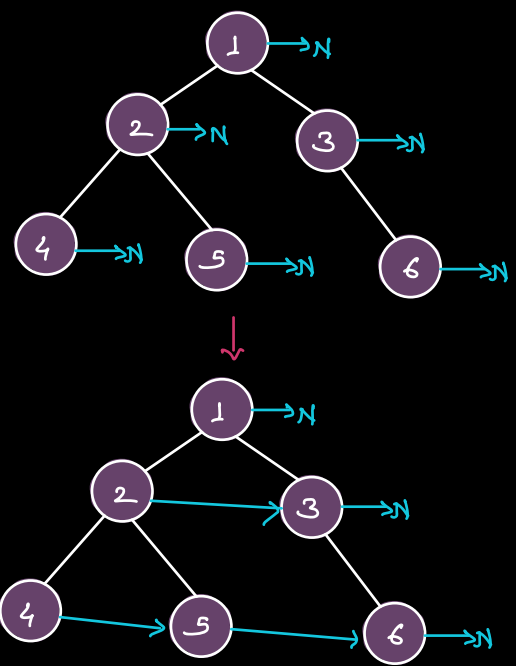
Post & Pre

Oracle
Facebook
Scaler

(Hard)

Q Given a binary tree with 'next' pointers.
Initially the next of all nodes is pointing to Null.
Populate the next pointer such that:
the next of a node points to the next node on the right side.

```
Class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode next;
}
```



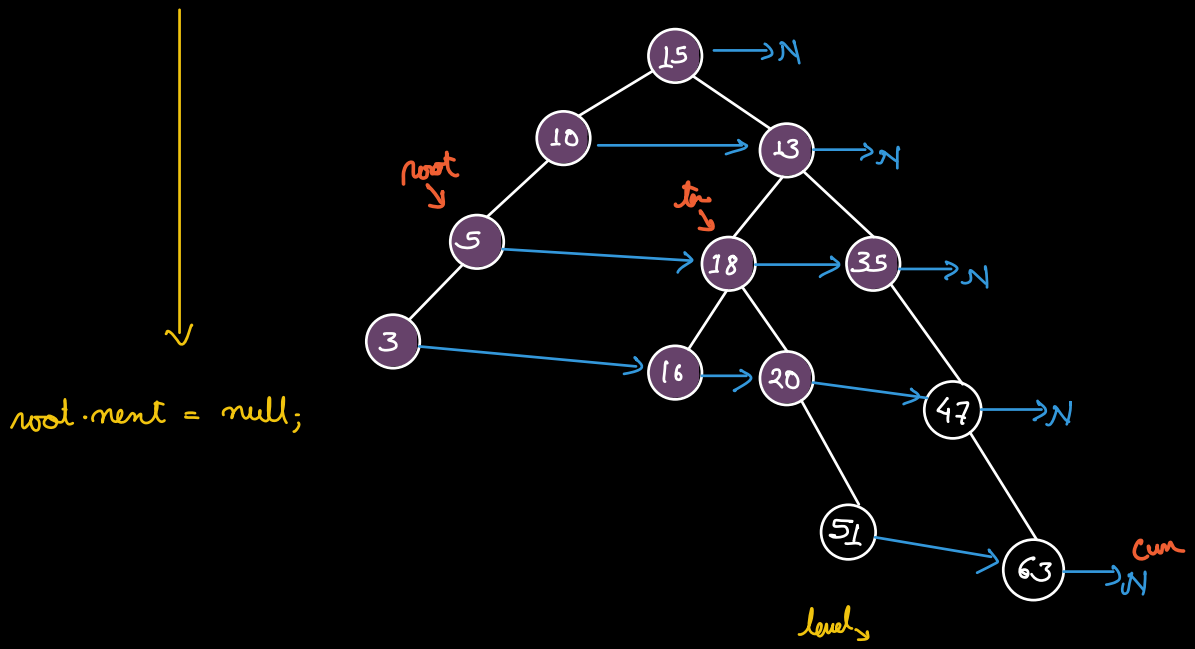
Use level order traversal

TC : O(N)
SC : O(N)

↓

Solve without using any extra space

SC : O(1)




```
TreeNode getNextRight(root) {
    if(root == null)
        ret null;
    TreeNode temp = root.next;
    While(temp != null) {
        if(temp.left != null){
            ret temp.left;
        }
        if(temp.right != null){
            ret temp.right;
        }
        temp = temp.next;
    }
    ret null;
}
```

```
level = root;
while (level != null) {
    cur = level;
    while (cur != null) {
        if (cur.left != null) {
            if (cur.right != null) {
                cur.left.next = cur.right;
            }
            else {
                cur.left.next = getNextRight(cur);
            }
        }
        if (cur.right != null) {
            cur.right.next = getNextRight(cur);
        }
        cur = cur.next;
    }
    if (level.left != null) {
        level = level.left;
    }
    else if (level.right != null) {
        level = level.right;
    }
    else
        level = getNextRight(level);
}
```

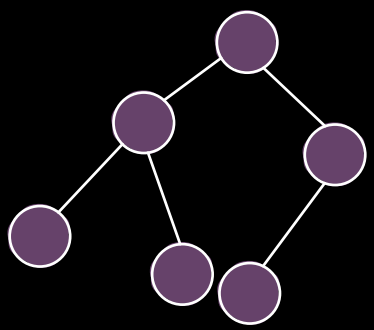
TC : O(N)
SC : O(1)

Google ✓
Facebook ✓
Direct i ✓
Walmart ✓

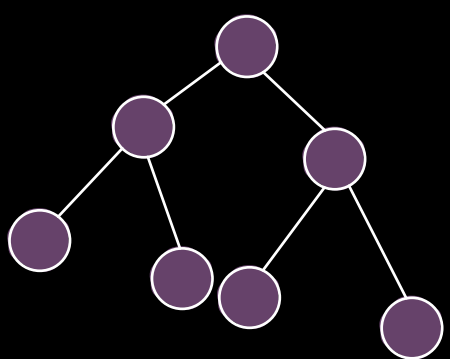
Q Given a complete binary tree. Count the no. of nodes in it.

Complete BT

All levels are completely filled except possibly the last level.
In the last level all nodes have to be left aligned.



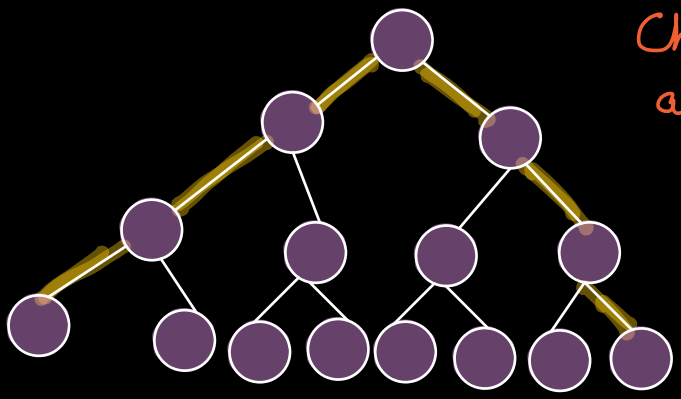
⇒ 6



$2^h - 1$

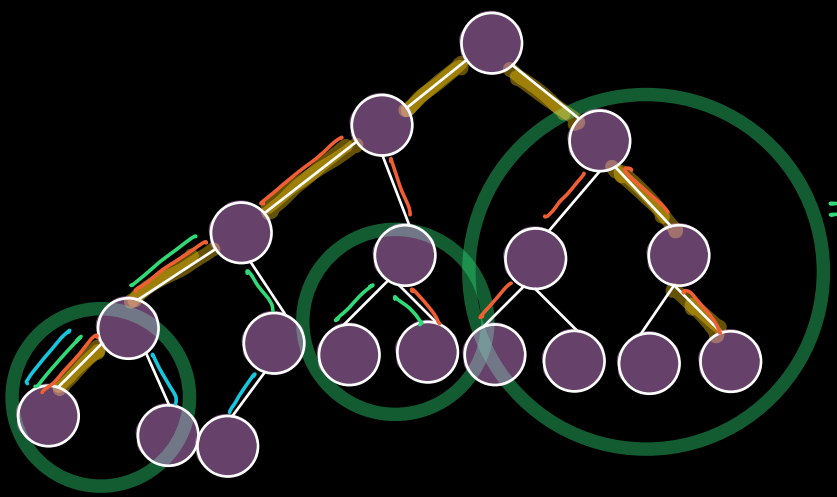
$N = 2^{(h+1)} - 1$ → refer to Tree II session

↑
if all levels are completely filled



Check if all levels are completely filled:

$lh == rh$



$\Rightarrow (2^{(h+1)} - 1)$

For any node one of the two sub trees will always be a completely filled tree.

```
int countNodes(root) {
    if(root == null) ret 0;
    int lh = leftHt(root); => O(log N)
    int rh = rightHt(root); => O(log N)
    if(lh == rh) {
        ret 2^(h+1) - 1;
    }
    ret
        1 + CountNodes(root.left)
          + CountNodes(root.right);
}
```

TC : $O(\log N \times \log N)$