Today's Content:

- ✓ → Level order traversal without extra space
- ✓ → Fill Right pointer
- ✓ → Level order Serialization
- ✓ → De-Serialize
- → Other doubts

Class Node {

    int data
    Node left
    Node right
    Node next

}

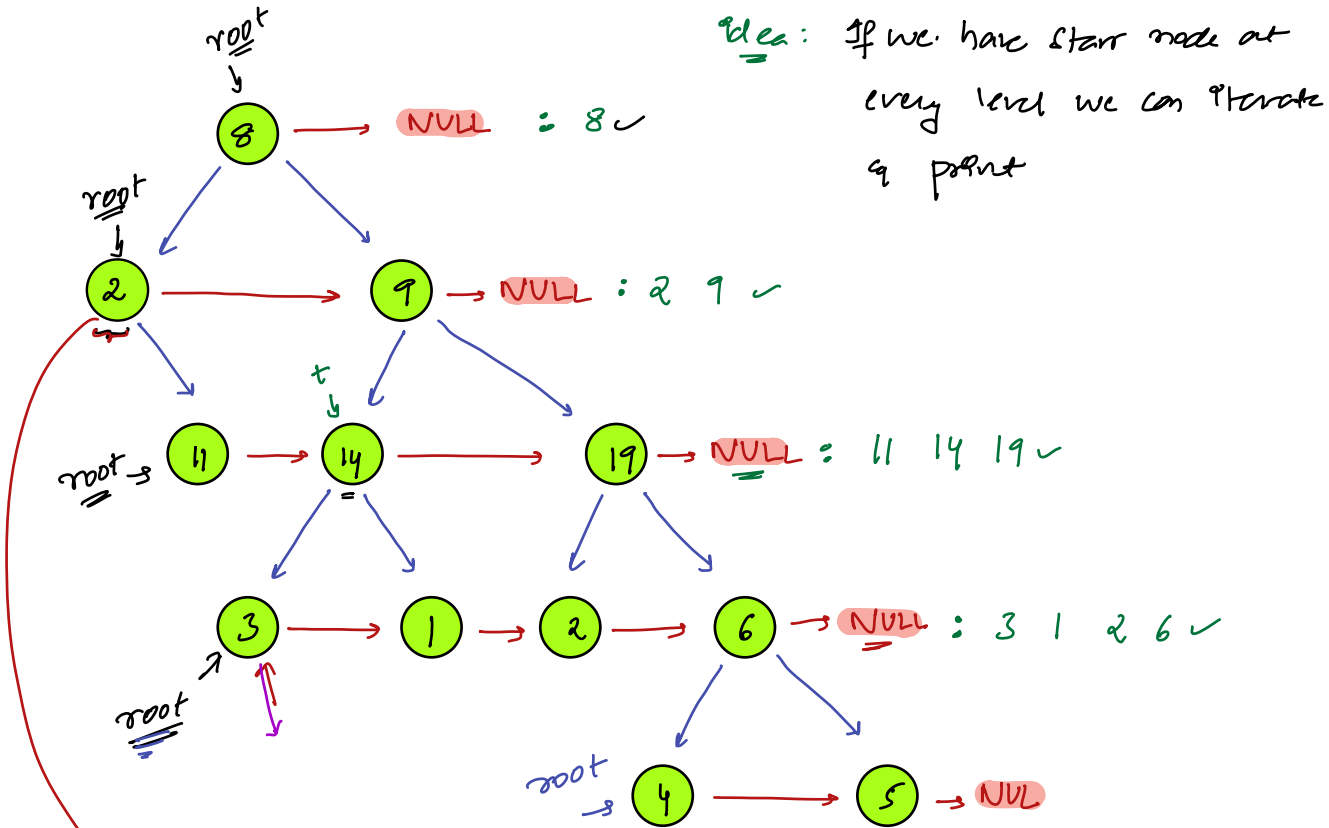
node points to next node in same level

Q) print BT in level order without extra space

→ SC: O(1)

level: 8 2 9 11 14 19 3 1 2 6 4 5

Idea: If we have start node at every level we can iterate & print

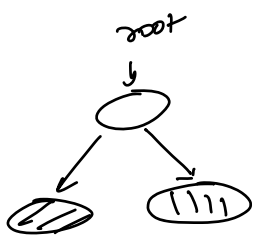


root = NextChild(2) : return 11

root = NextChild(3) : return 4

root = NextChild(4) : return NULL

```
void levelOrder(Node root) {

    while( root != NULL ) {

        Node temp = root

        while (temp != NULL) {          // we iterate all
            print(temp.data)            // all node 1 time
            temp = temp.next
        }

        if (root.left != NULL) {
            root = root.left
        }
        else if (root.right != NULL) {      or
            root = root.right
        }
        else {
            root = nextChild(root)
        }

        root = nextChild(root)          // we iterate n all node 1 times
    }
}
```



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

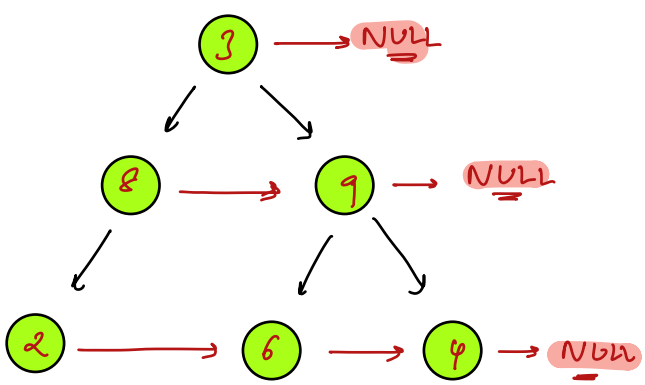
```
Node NextChild (Node root) {
    while (root != NULL)
        if (root.left != NULL) { return root.left }
        if (root.right != NULL) { return root.right }
        root = root.next
    return root
}
```

Preferably if NULL

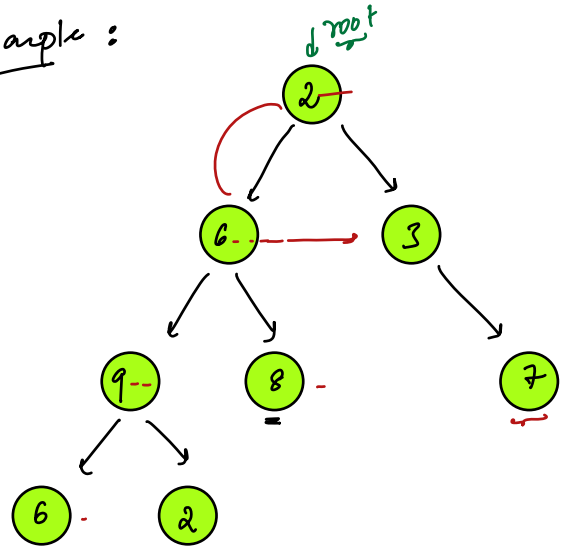
2.8) Fill right Pointer for a given B.T → Can be anything

- S.C: O(1)
- T.C: O(N)

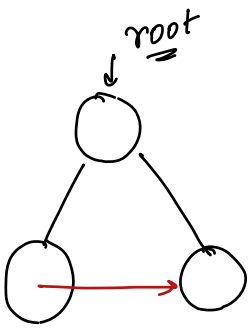
```
Class Node {
    int data;
    Node left
    Node right
    Node next
    Node (int n) {
        data = n
        left = NULL
        right = NULL
        next = NULL
    }
}
```



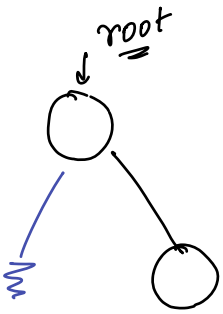
Example :



Idea : fill next pointers level by level



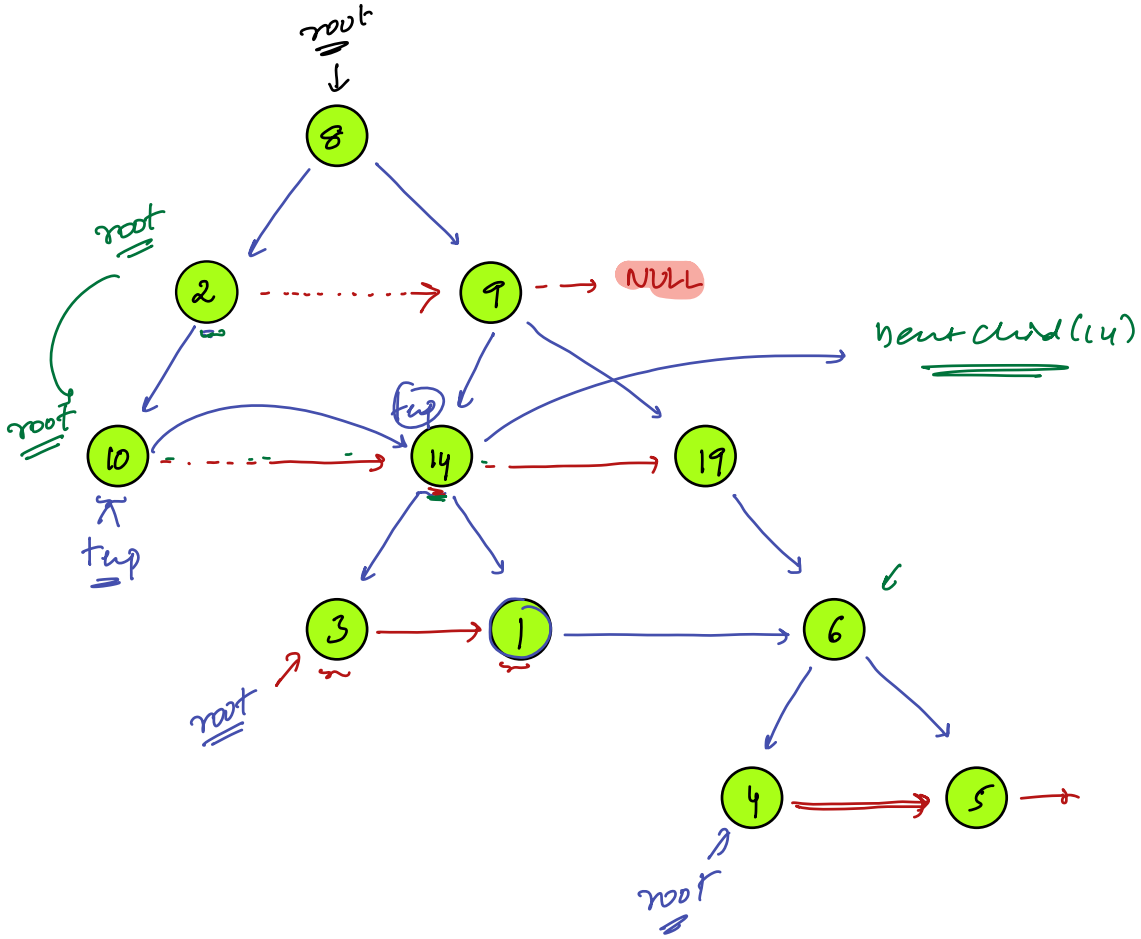
root.left.next == root.right



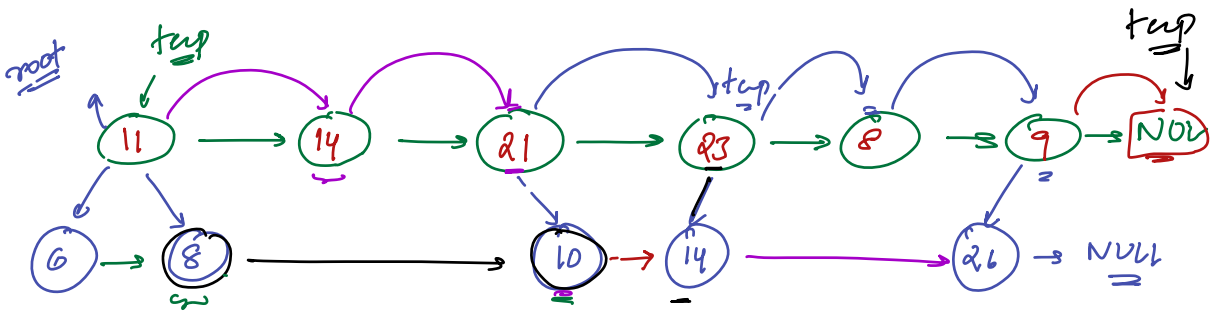
wrong

// root.left.next == root.right

root = 8



queue:



nextchild (9, null)

nextchild (null)

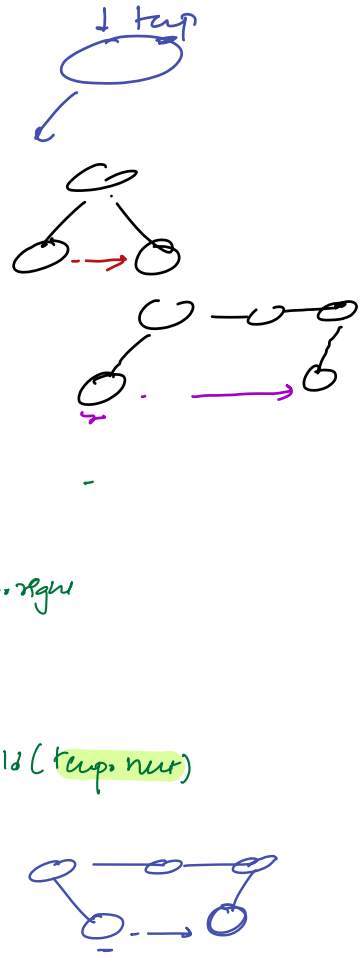
{ nextchild (21, null)

match (21) = 14

{ nextchild (21, null)

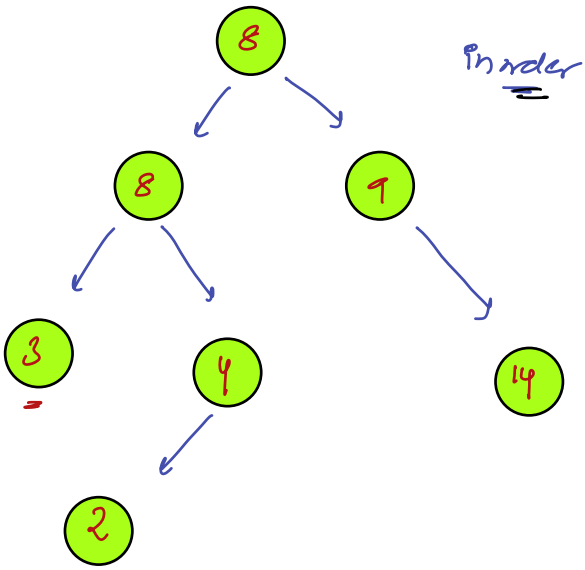
match (8) = 26

```
void fillRightPointer(Node root) {
    while(root != NULL) {
        Node temp = root;
        while(temp != NULL) {
            if(temp.left != NULL) {
                if(temp.right != NULL) {
                    temp.left.next = temp.right
                }
                else {
                    temp.left.next = nextChild(temp.next)
                }
            }
            if(temp.right != NULL) {
                temp.right.next = nextChild(temp.next)
            }
            temp = temp.next
        }
        root = NextChild(root);  // root is going to first node of next level
    }
}
```



TC: O(N) SC: O(1) } 10:40pm break

Ex: // duplicates :



preorder: 8 8 3 4 2 9 14

Inorder: 3 8 2 9 8 9 14

LST

RST

8

// Serializable :

⇒ We need to store in such a manner that we need to construct back original Tree

DeSerialization

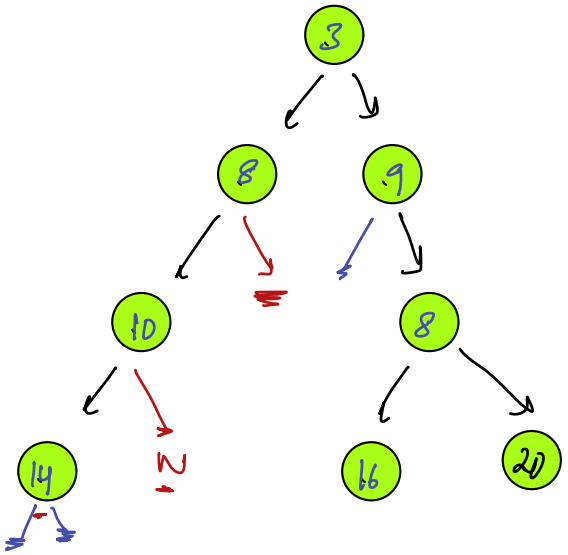
// Serialization:

preorder | postorder | levelorder (Inorder)

Assignment

Software

// Serialization using level order: → {(+ve) data} (Array +ve data)

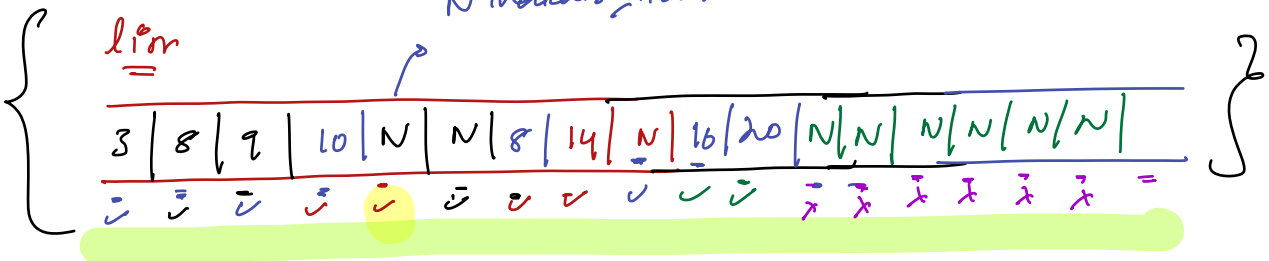


Idea:
Simply go level by level & push child of every node.

Only change: In a list

N nodes, null

list =



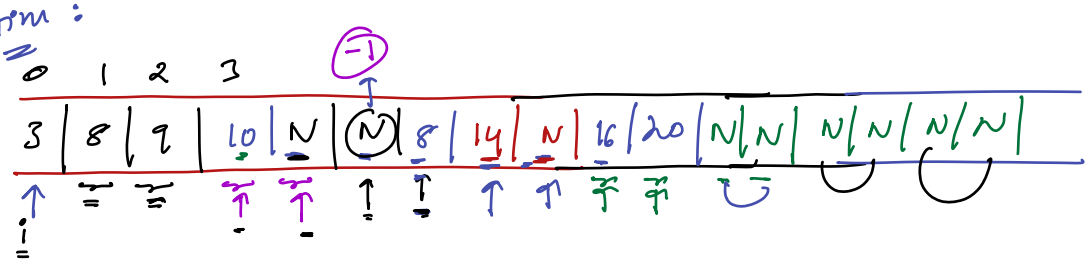
```
Serialization(Node root){
    ArrayList<Integer> li;   } Use queue to store nodes & value in lpm }
    Queue<Node> q;
    q.push(root);
    while( q.size()>0 )
    {
        Node (t) = q.front(); q.pop();
        if( t != NULL) { li.add ( t.data)}
        else { li.add( -1 ) }   -1 indicates NULL
        if( t != NULL) {  // if t == NULL in that case
            q.push( t.left)      we won't push children
            q.push( t.right)
        }
    }
    return li
}
```

TC: $O(N)$

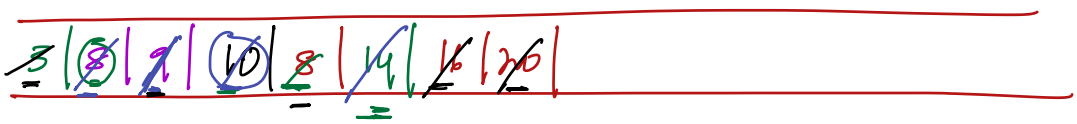
SC: $O($ max Nodes in a level $) \rightarrow O(N)$

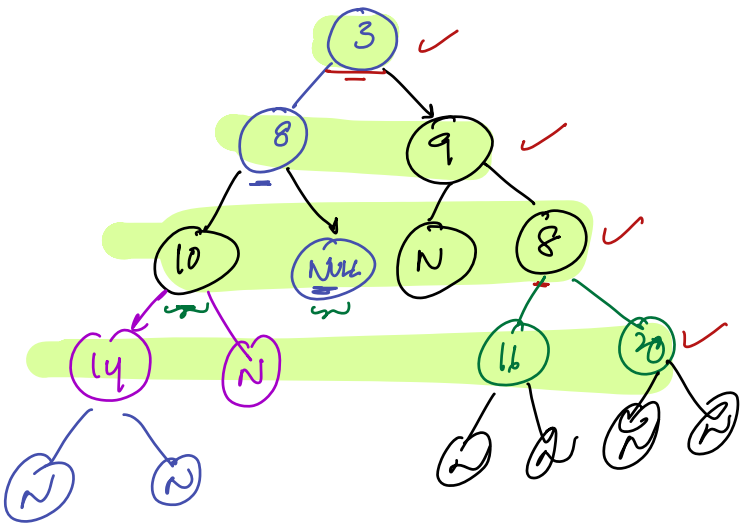
↳ $N/2$

Deserialization :



obs: Fill the child nodes of every node level by level.





//

root = deserialization(int ar[], int n){

Node root = new Node(ar[0])

QueueNode> q; q.push(root);

i = 1;

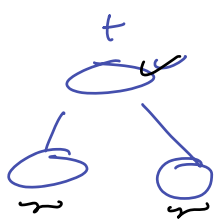
while (q.size() > 0) {

Node t = q.front();

q.pop();

// t -> left child at index i

-> right child at index i+1



if (ar[i] != -1) {

t.left = new node(ar[i])

q.push(t.left)

}

if (ar[i+1] != -1) {

t.right = new node (ar[i+1])

q.push(t.right)

}

i = i+2

}

return root;

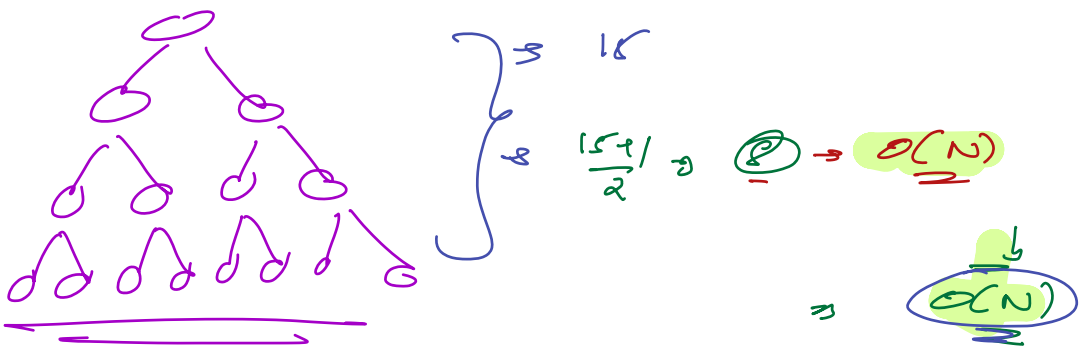
TC: O(N) SC: O(max node in level)

(Data) → Number

↳ Is number → string arr[]

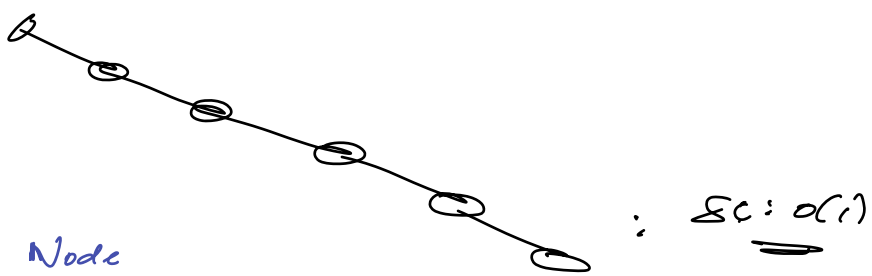
↳ We can use char to indicate that pointer is NULL

→ : O(N/2)



→ (Equal Tree Partition)

→ Tree: Check if there exists a subtree with Sum?



∴ SC: O(1)

H → Node

H == N : Skewed → less SC: O(1)

H == log(N): Balance → SC: O(N)

→ LCA without parent ptr

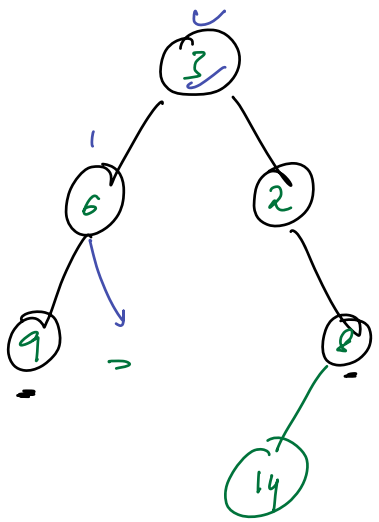
↳ node which return both True is your LCA

→ Space Complexity

CBT → Complete Binary Tree :

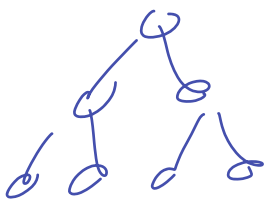
- Nodes level by level
- 0 nodes / 2 nodes

Heap ?



0	1	2	3	4	5	6	7	8	9	10	11	12
3	6	2	9	NULL	NULL	8	NULL	NULL	14	NULL	NULL	NULL

→



Period of String

↳ (N - Lps[7])