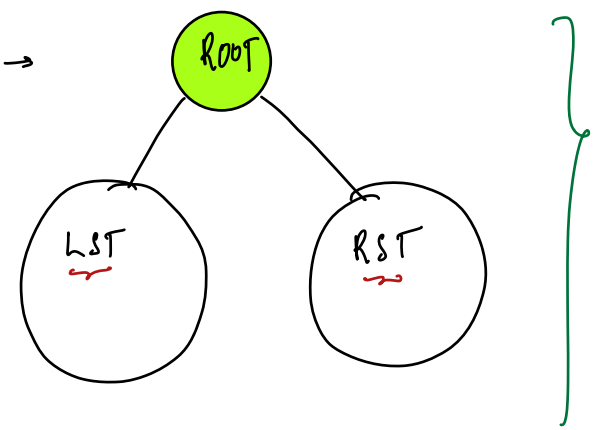
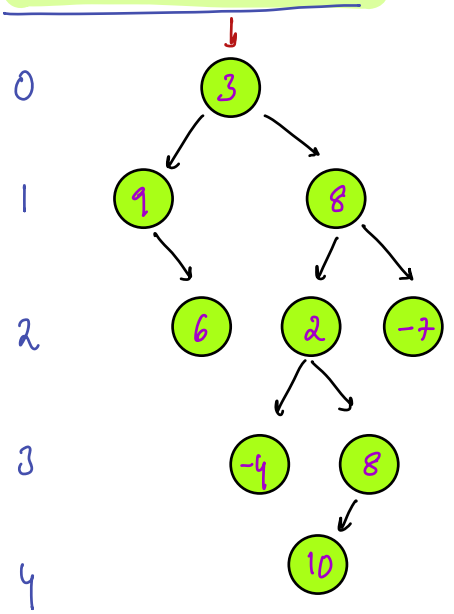
Today's Content :

- → level order traversal
  - a) leftview b) RightView
- → vertical level order traversal
  - a) Top view b) Bottom view c) Diagnal view
- → Tree Construction Problems. [Next Session]

→



level order traversal 1:



Expected Output:

3 9 8 6 2 -7 -4 8 10

TC: O(N)

SC: ~~—~~ ? Comments

Ideal: (Recursion) → * Subproblems

Ideal: (Iterative) ?

~~3~~ | ~~9~~ | ~~8~~ | ~~6~~ | ~~2~~ | ~~-7~~ | ~~-4~~ | ~~8~~ | ~~10~~

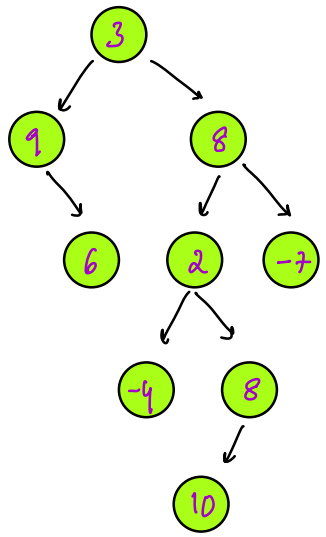
Output:

3 9 8 6 2 -7 -4 8 10

Pseudocode:

```
Queue<Node> q;
q.insert(root);
while(q.size()>0){
    Node f = q.front();
    q.remove();
    print(f.data)
    // insert children
    if(f.left != NULL){
        q.insert(f.left)
    }
    if(f.right != NULL){
        q.insert(f.right)
    }
}
```

level order order 2:



Expected Output:

3 \n
9 8 \n
6 2 -7 \n
-4 8 \n
10 \n

leftview : To view Tree from left

→ 1st Element of every level :

Hint: for every 1st Elem in level, prev == NULL

Edge: Root Node Print at start

NULL → Indicate end of a level

3 | N | 9 | 8 | N | 6 | 2 | -7 | N | -4 | 8 | N | 10 | N | N | N | N

∞ loop

3 \n
9 8 \n
6 2 -7 \n
-4 8 \n
10 \n \n \n

Idea:

```
Node f = q.front()
q.remove()
if (f == NULL) {
    print("\n")
    q.insert(NULL)
}
```

q.size() > 1

Even if single node is present

node | NULL

Pseudocode:

```
Queue<Node> q;
q.insert(root);
q.insert(NULL)

while(q.size() > 1){
    Node f = q.front();
    q.remove();
    if(f == NULL){
        print("\n")
        q.insert(NULL)
    }
    else{
        print(f.data)
        // insert children
        if(f.left != NULL){
            q.insert(f.left)
        }
        if(f.right != NULL){
            q.insert(f.right)
        }
    }
}
```

valid value node can hold = NULL

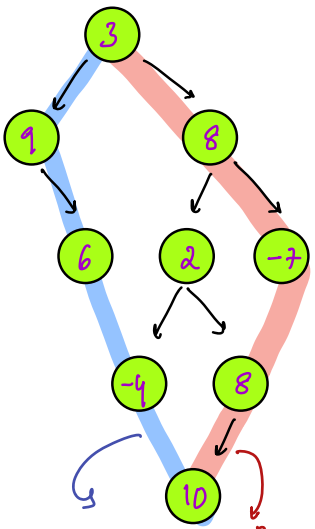
Node temp = NULL
Node nent = NULL

TC: O(N)

SC: (max nodes we can have in level)

→

level order traversal3 : (Right → left)



3
8 9
-7 2 6
8 -4
10

while inserting child node

if (not NULL)
→ right child

if (not NULL)
→ left child

Lv + Rv = Boundary

→ rightview : keep your eye on right side of view

output : 3 8 -7 8 10

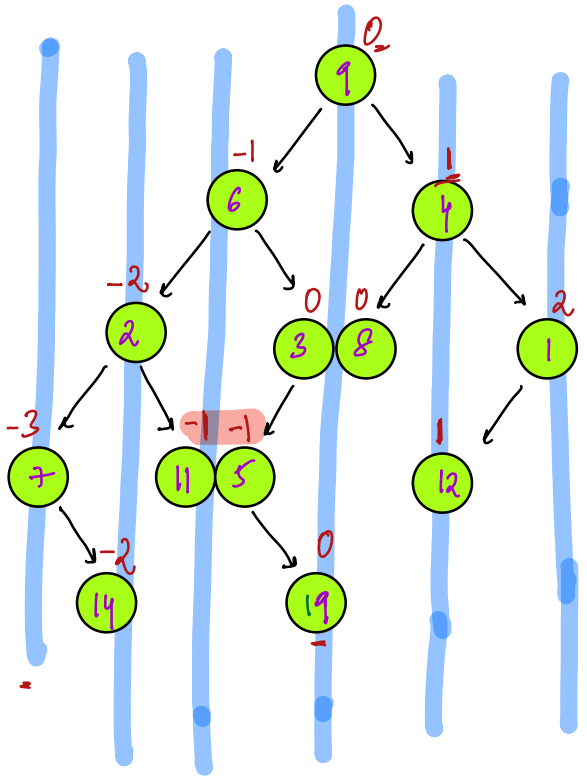
Idea: first node of every level in (level order traversal from R-L)

Edge: Root Node
print at start

→ (Boundary/ Perimeter) : (TODO) / assignment - order

left view + Right View - (Overlapping)
Edge cases

// vertical level order traversal (Left → Right)



Expected output:

7
2 14
6 11 5
9 3 8 19
4 12
1

Top view: first node of every level

Bottom: last node of every level

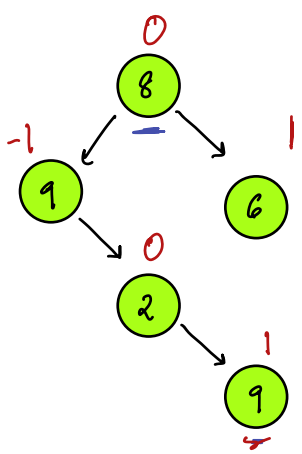
Data Structure:

Map<int, list of nodes>

-3 : 7
-2 : 2 14
-1 : 6 11 5
0 : 9 3 8 19
1 : 4 12
2 : 1

minl = -3
All level
Between
-3 & 2
present
manl = 2

Example:



Expected tm

-1 : 9
0 : 8 2
1 : 6 9

preorder DLF tm

-1 : 9
0 : 8, 2
1 : 9 6

} order not correct

Inorder LDR

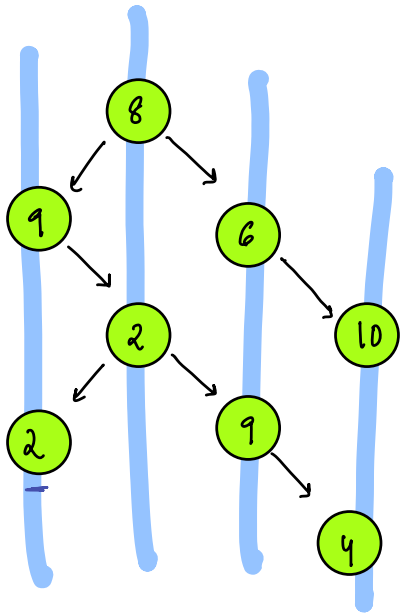
9 2 9 8 6

-1 : 9
0 : 2 8
1 : 9 6

postorder LRD

9 2 9 6 8

-1 : 9
0 : 2 8
1 : 9 6



Fill hashmap Using level order

HM:

-1 : 9 2

2 : 10 4

1 : 6 9

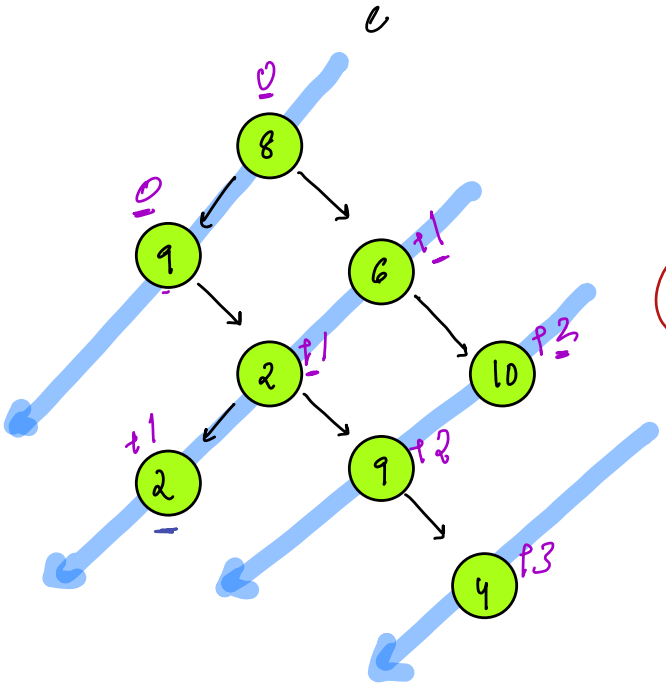
0 : 8 2

<8,0> | <9,-1> | <6,1> | <2,0> | <10,2> | <2,-1> | <9,1> | <4,2>

Pseudo Code

```
HashMap<int, ArrayList> hm;
Queue<{Node, int}> q;
int minL = 1, maxL = -1
q.insert({root, 0})
while( q.size() > 0){
    {Node, int} f = q.front();
    q.remove();
    Node t = f.first; int l = f.second
    minL = min(minL, l), maxL = max(maxL, l)
    // insert node t @ level l
        hm[l].add(t)
    // Add left & right
        if( t.left != NULL)
            q.insert(t.left, l-1)
        if( t.right != NULL)
            q.insert(t.right, l+1)
}
for(i = minL; i<=maxL; i++){
    // hm[i] -> ArrayList of Nodes } print(hm[i].first)
    // print above list          } print(hm[i].last)
}
```

Diagnol view :



output :

8 9

6 2 2

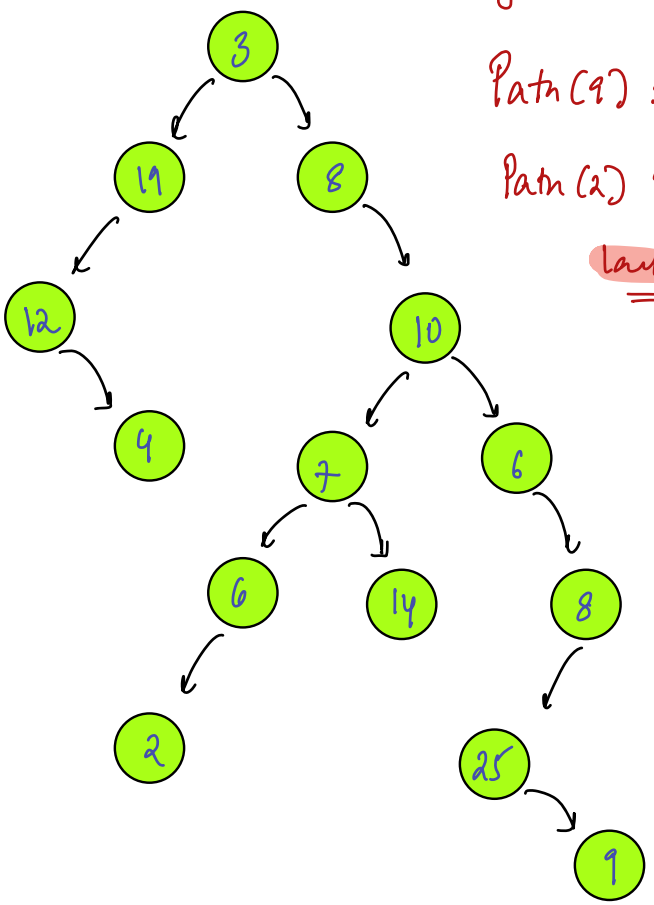
10 9

4

If we go left : (level same)

If we go rig : (level +1)

LCA & Path between 2 Nodes



// given a node get path

Path(9): 3 → 8 → 10 → 6 → 8 → 25 → 9

Path(2): 3 → 8 → 10 → 7 → 6 → 2

last comm node → Least Comm Ancstr

↳ LCA

2 6 7 10 6 8 25 9

// get me path (2-9)

↳

2 → 6 → 7 → 10 → 6 → 8 → 25 → 9

// Idea: LCA(A, B):

→ get path of A from Root Node: List: pA

→ get path of B from Root Node: List: pB

→ last Comm Node ⟶ Least Comm Ancstr

Traversing

}