

Trees 2

[Level Order Traversal \(R - L\)](#)

[Left View And Right View](#)

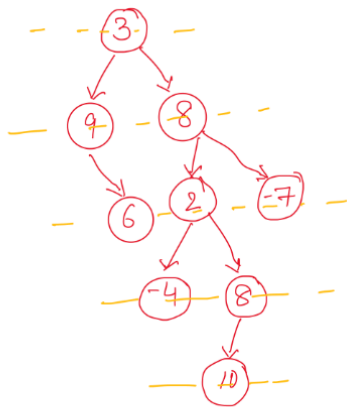
[Vertical Order Traversal](#)

[Top And Bottom View](#)

[Types Of Binary Tree](#)

[Height Balanced Tree](#)

Level Order Traversal (R - L)



Expected ~~dfs~~ \rightarrow 3 9 8 6 2 -7 -4 8 10
Left to Right.

Idea:

Recursion will not work as all subtree cannot be same.

So

Iterative :-

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

Level order Traversal

3 9 8 6 2 -7 - - -

Pseudo Code:-

```

Queue < Node > q;
q.enqueue (root);
while (!q.empty()) {
    Node f = q.front();
    q.dequeue();
    Print (f.data());
    if (q.left != NULL)
        q.push (f.left);
    if (q.right != NULL)
        q.push (f.right);
}
  
```

Normal Inorder

T: $O(N)$

S: $O(N)$

Queue
(FIFO)

```

-> while (q.size() > 1)
-> q.push (NULL);
if (f == NULL)
{
    Print ("in");
    q.push (NULL);
}
else {
    Print (f.data());
}
-> To print inorder
line by line.
  
```

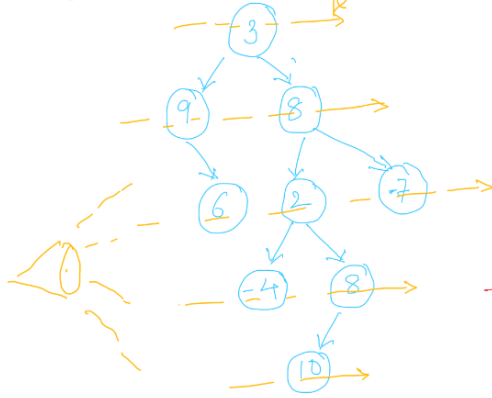
~~3 NULL 9 8 NULL~~

Inorder in different line

R-L \rightarrow first Push Right than left Inside loop

Left View And Right View

Left View of Binary Tree

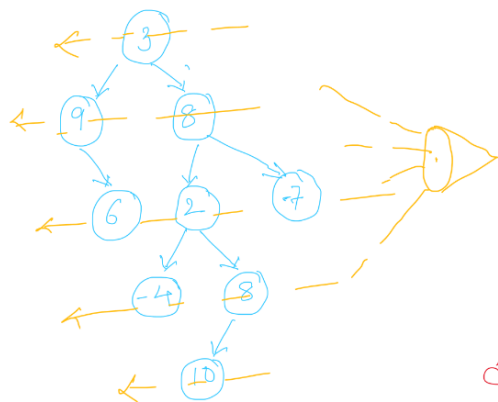


Left view: 3 9 6 -4 10

- Start level order Traversal and in every iteration select the first element to get left view
- For all first element in each level element previous to that, will be NULL except root element. (prev == NULL)

edge: root Node

Right view of a B-Tree

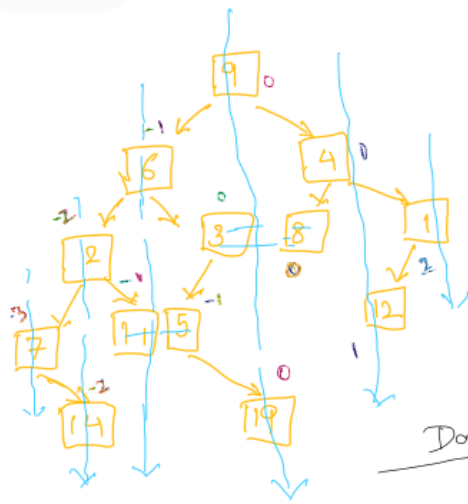


3 8 -7 8 10

- Start level order Traversal Last node of every level will be Right view of B-Tree.
- For R.v node in each level the next will be NULL.

OR
do Level order Traversal from Right to left, & follow approach for left view.

Vertical Order Traversal

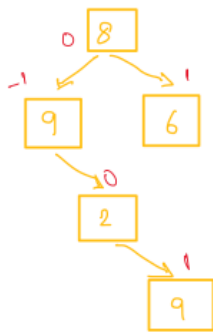


Expected O/P

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

Data Structure
Hashmap

Hashmap <level, list<Nodes>> hm;



Expected hm

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

Inorder (LNR)

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

X

Preorder (NLR)

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

X

Postorder

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

level order

-1 : 9
0 : 8 2
1 : 6 9 ✓

<Node, level>

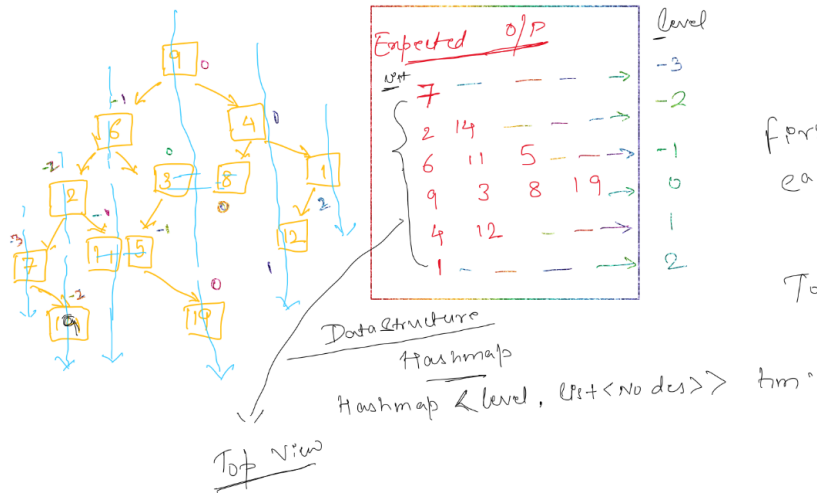
~~<8, 0>~~ ~~<9, -1>~~ ~~<6, 1>~~ ~~<2, 0>~~ ~~<9, 1>~~

keep adding in hashmap

Pseudo Code:

```
Queue <Node, Int> q
HashMap <int, List<Node>> hm
q.push({root, 0});
while(!q.empty())
{
    Pair <Node, int> p = q.front
    Node f = p.first;
    int l = p.second;
    q.dequeue();
    hm[l].add(f); //Insert Node f at level l.
    //Add left & right child in the queue.
    if(f.left != NULL)
        q.push({f.left, l+1});
    if(f.right != NULL)
        q.push({f.right, l+1});
}
```

Top And Bottom View

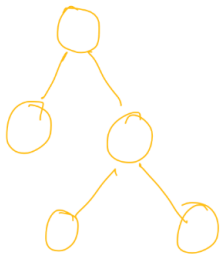


First Node at each level

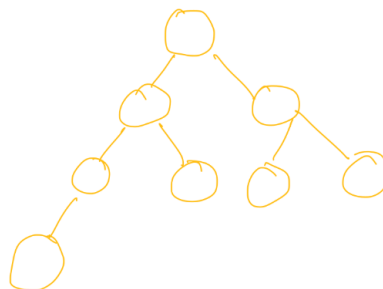
To check - Bottom View

Types Of Binary Tree

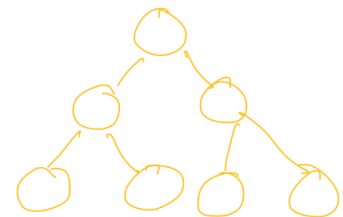
Full BT
- All nodes 0 or 1 child



Complete BT
- every level should be completely filled (exception - last level)
- All nodes in last level should be filled from right to left.



Perfect BT
- All levels should be filled
- All levels should have 0 or 2 child



Perfect BT ✓
PBT ✓

Height Balanced Tree

```

int height (Node root)
{
    if (root == NULL)
        return 0;

    int l = height (root. left)
    int r = height (root. right)
    return max(l, r) + 1;
}

```

Height of a Tree

Q Check whether a given tree is height balanced or not?

$$|h(LST) - h(RST)| \leq 1$$

e.g.



bool ans = true;

bool isBalanced (Node root)

```

{
    height (root)
    return ans;
}

```

```

int height (Node root)
{
    if (root == NULL)
        return 0;

```

```

    int l = height (root. left)
    int r = height (root. right)

```

```

    if (abs (l - r) > 1) {
        ans = false;
    }

```

```

    return max(l, r) + 1;
}

```

}