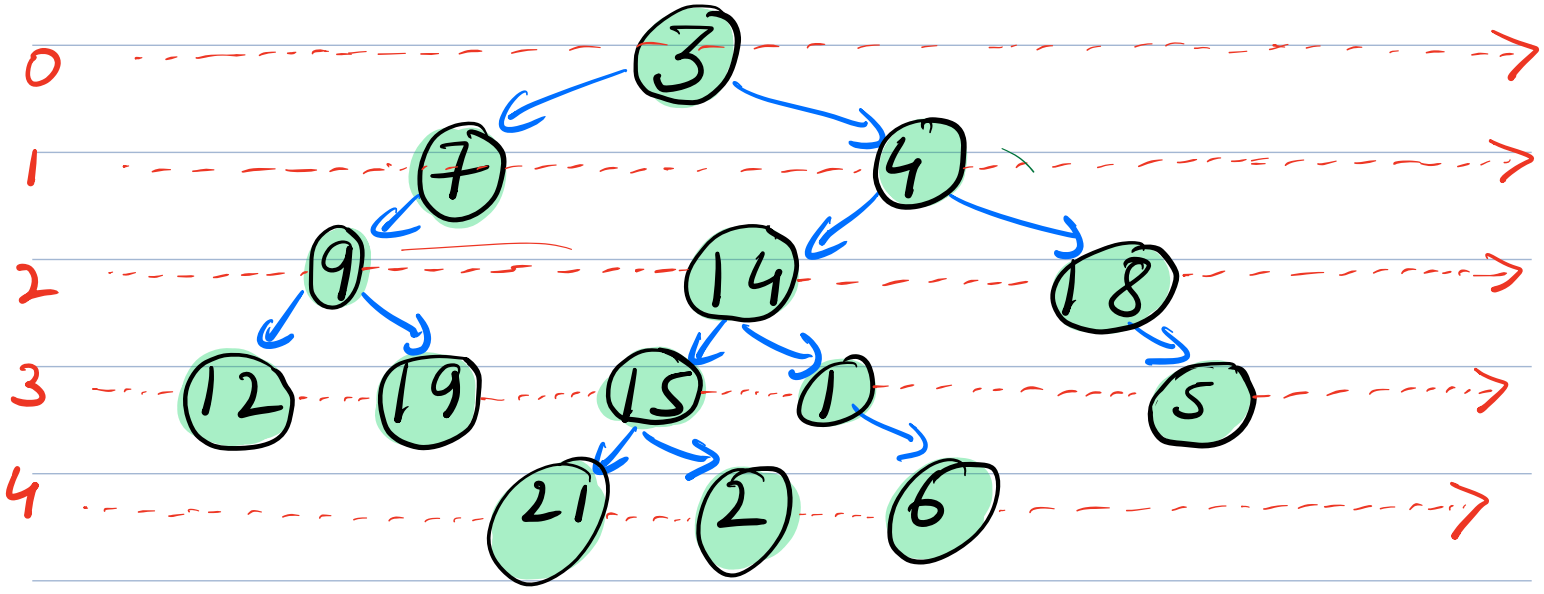
Level Order Traversal (left to right)



3 7 4 9 14 18 12 19 15, 1, 5

21 2 6

queue

3 7 4 9 14 18 12 19 15

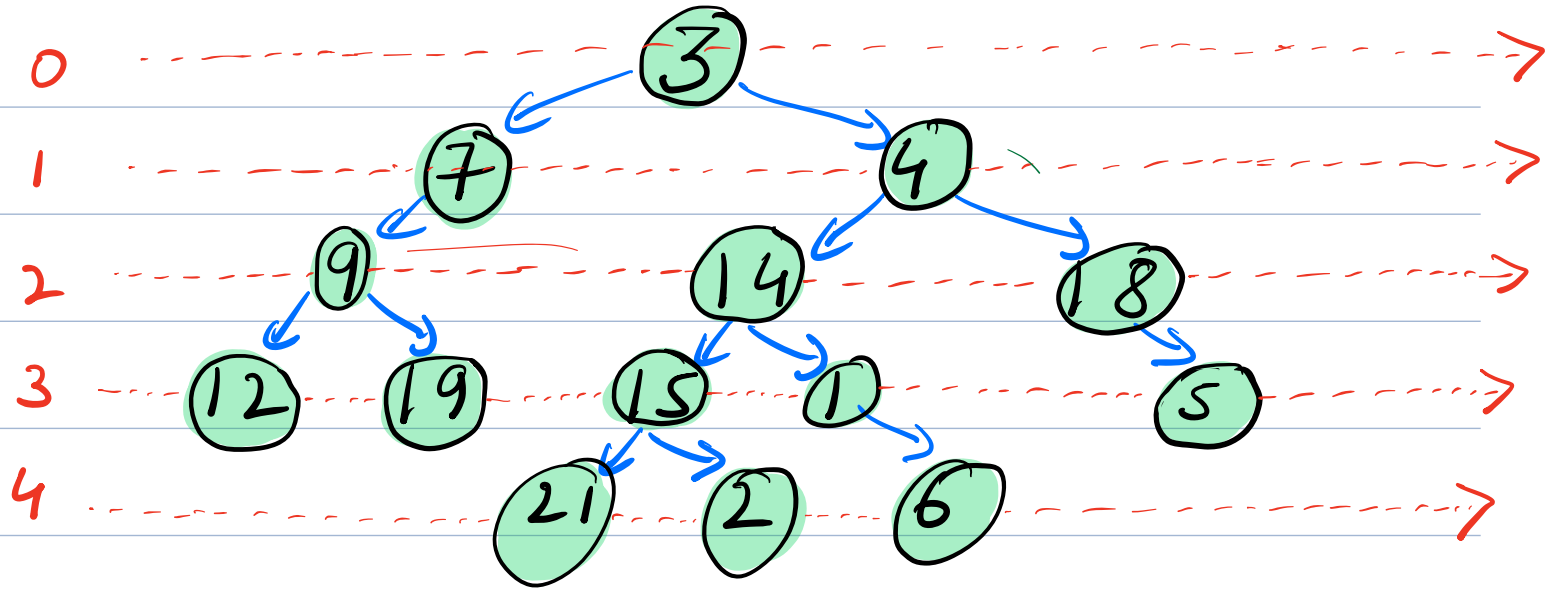
1 5 21 2 6

Code

```
void levelOrder(root){
    queue<TreeNode> q.
    q.enque(root)
    while(!q.empty()){
        x = q.front()
        print(x.data)
        q.deque()
        if(x.left != null)
            q.enque(x.left)
        if(x.right != null)
            q.enque(x.right)
    }
}
```

TC: O(n)

SC: O(n)



3

7 4

9 14 18

12 19 15 1 5

21 2 6

---

queue

3

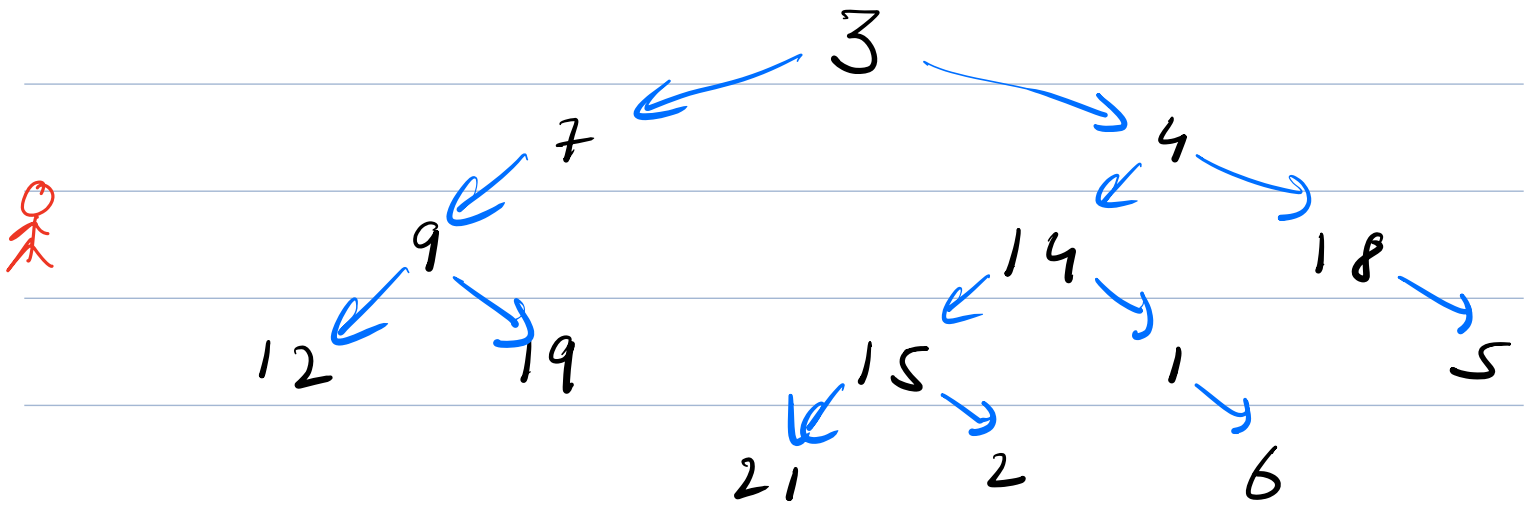
7 4

9 14 18

12 19 15 1 5

21 2 6

```
void levelOrder(root){
    queue<TreeNode> q.
    q.enque(root), q.enque(null)
    while(!q.empty()){
        x = q.front()
        if(x == null && q.size == 1)
            break
        if(x == null)
            1) goto new line  2) remove this null
            3) insert null at end 4) continue
        print(x.data)
        q.deque()
        if(x.left != null)
            q.enque(x.left)
        if(x.right != null)
            q.enque(x.right)
    }
    cout << "\n"
```



Left view: first node of each level

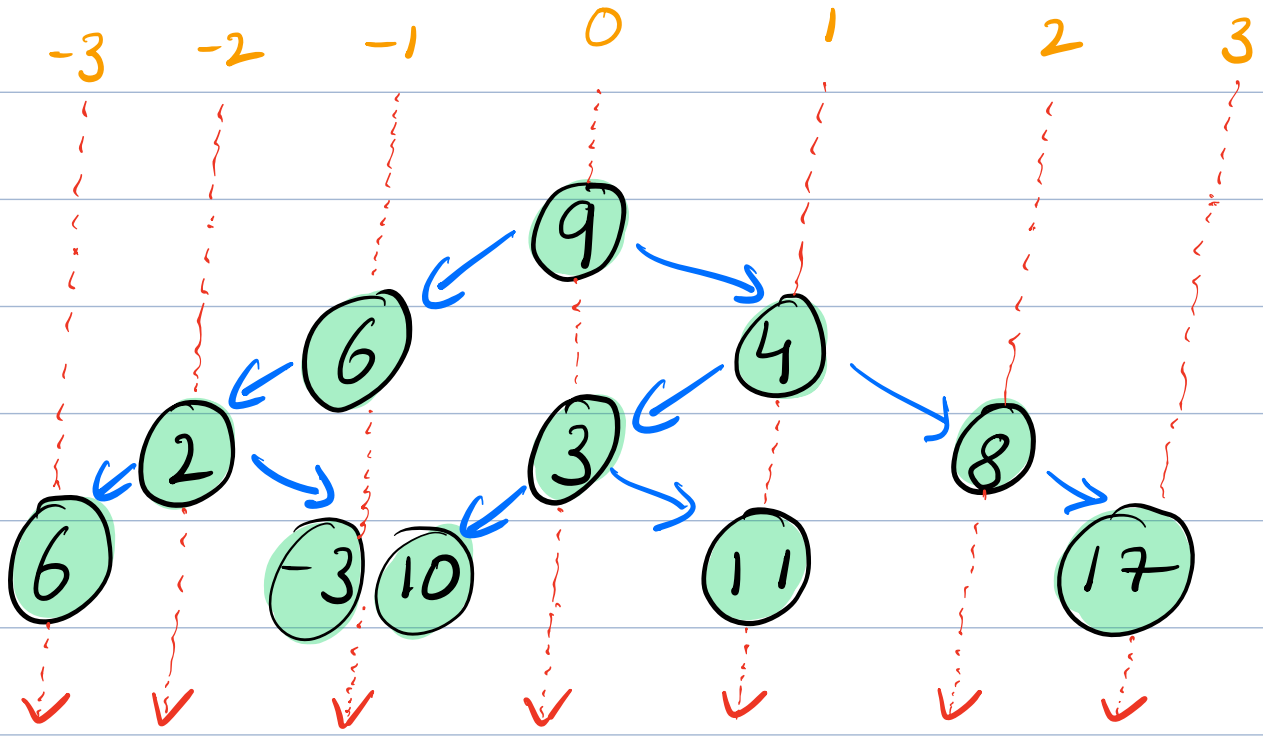
Right view: last node of each level.

OR first node going right to left.

LV: 3 7 9 12 21

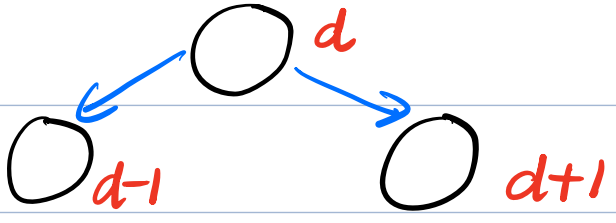
RV: 3 4 18 5 6

Vertical order traversal



top to bottom.

⇒



left ⇒ -1

right ⇒ +1

How: Hashmap:

-1 ----

0 ----

1 ----

line no → list of nodes

Code

```
hashmap<int, list<Node>> hm
queue < pair<Node, int>> q
q.enque ( { root, 0 } )
while (! q.empty ()) {
    pair<Node, int> p = q.front ()
    q.pop()
    int d= p.second , Node node = p.first
    hm[d].insert(node)        // do null-check
    q.enque ( { left_child, d-1 } )
    q.enque ( { right_child, d+1 } )

}
```

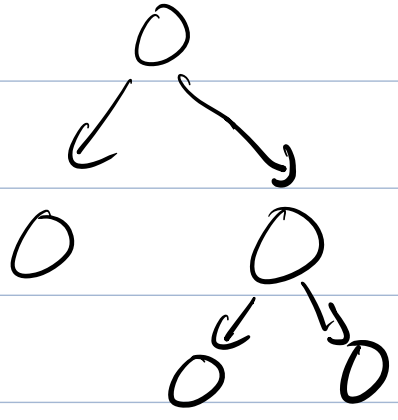
TC: } O(n)
SC: }

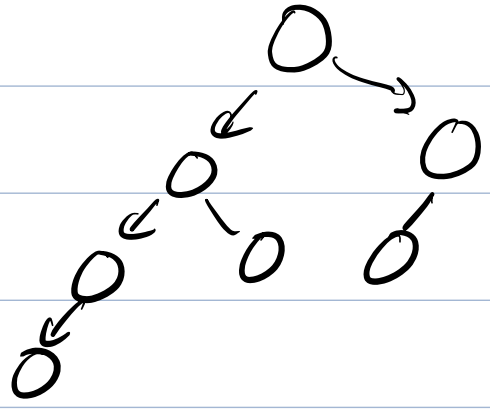
Top view ⇒ First node of each list

Types of binary tree

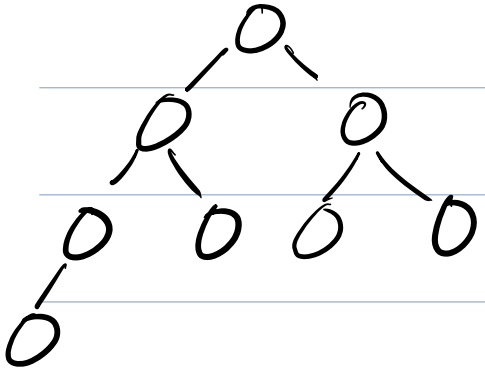
- Proper binary tree: either 0 or 2 children
- Complete binary tree: each level completely filled except may be last

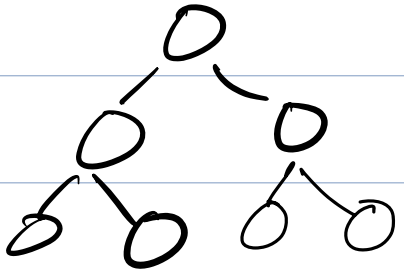
  Exception: Last level. should be filled L → R
- Perfect binary tree: every level completely filled





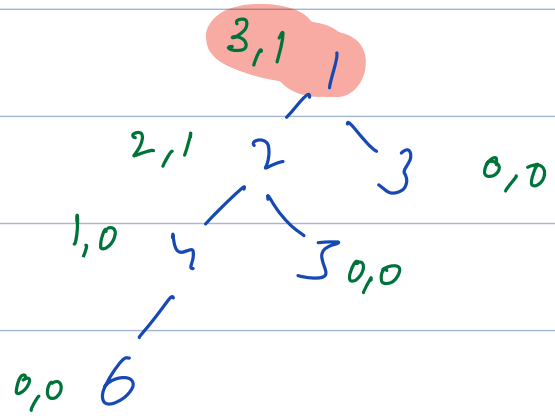
⇒ every perfect tree is proper & complete



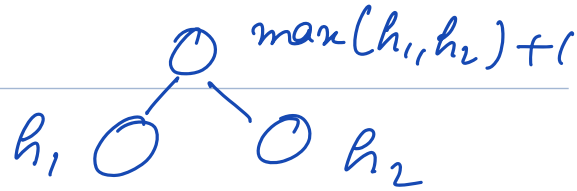


Q Check if balanced tree

for all nodes |height (left_child) - height (right_child)| ≤ 1



= false (not balanced)



Idea ⇒ for each node check if

|height (left_child) - height (right_child)| ≤ 1

Code

```
bool ans
int height (node) {
    if (node == null)   return 0
    l = height (node.left)
    r = height (node.right)
    if (abs(l-r) > 1)   ans = false
    return max(l,r) + 1
}
```

```
bool checkTreeBalanced(TreeNode root){
    ans = true
    height(root)
    return ans
}
```