Todays Content

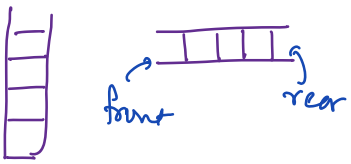
- Trees Intro
- Naming Convention
- Tree Traversal
- Basic Tree Problems

Linear:

↳ arrays

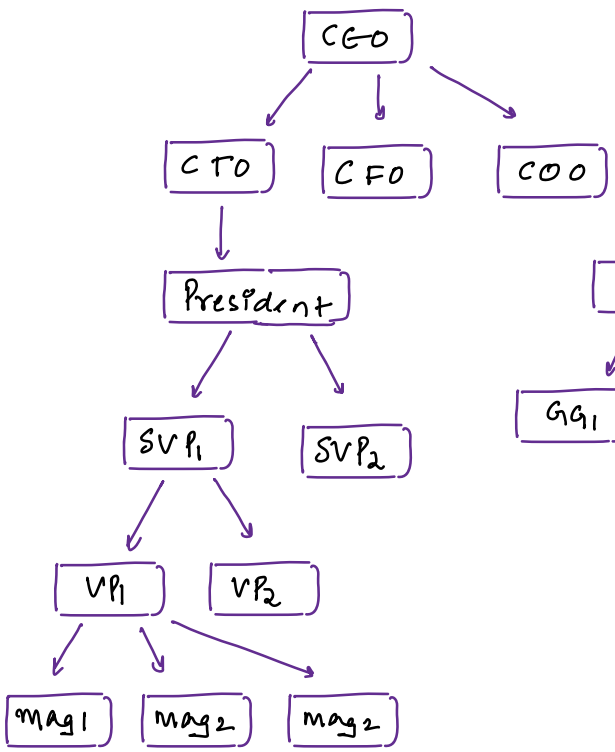
Linked list

stacks & Queues

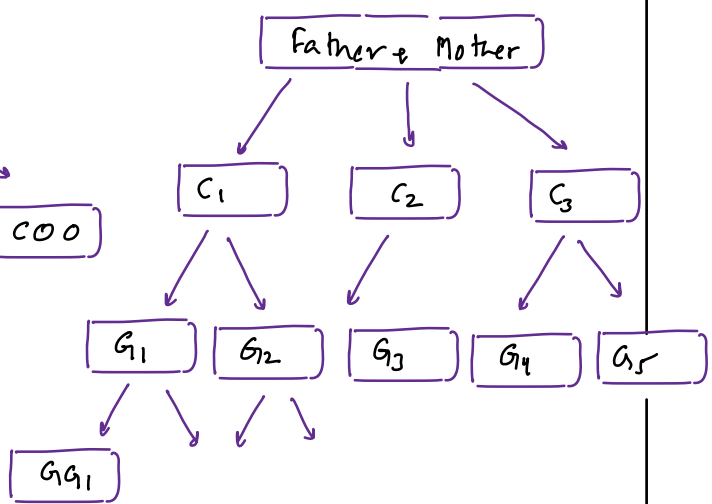


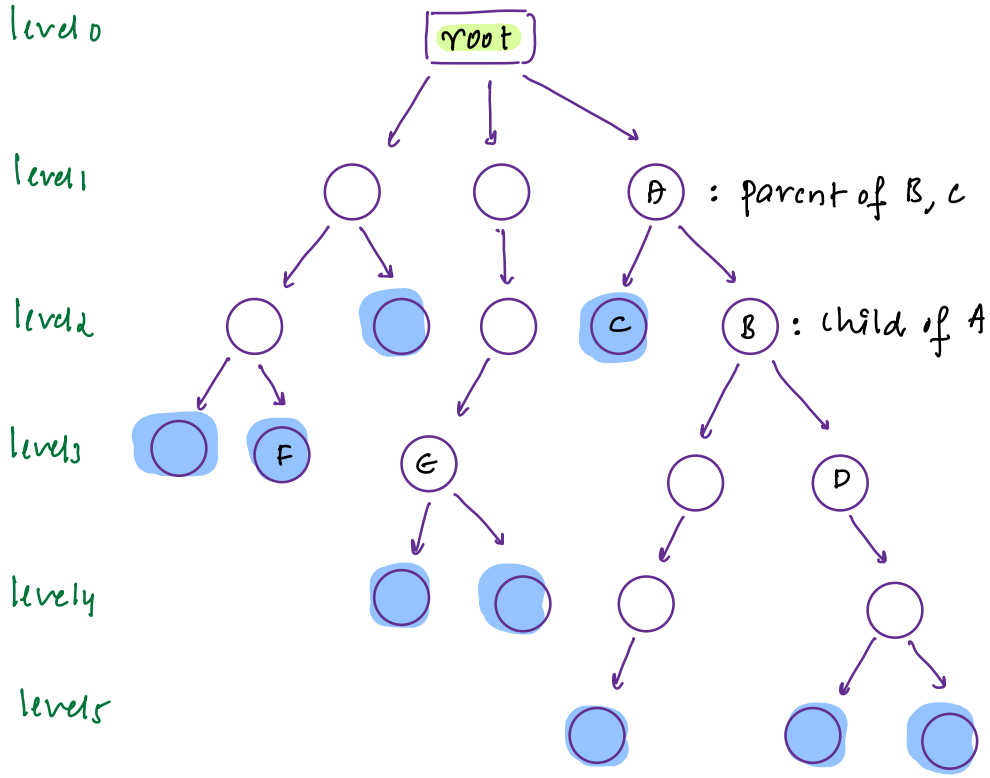
Hirarchial Data

↳ Eg: Company organisation



Eg2: Family Tree:





Naming

A → D : A is ancestor of D & D is decedent of A

B → C : Sibiling node, share same parent

F, E, D : Node at same level

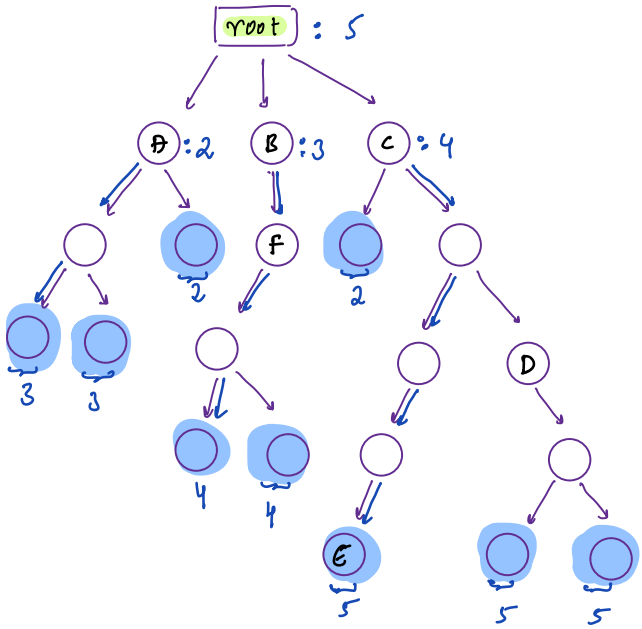
root : Node without a parent

leaf : Nodes without children

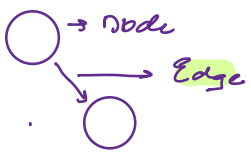
True : [We will have only 1 root node
For every node only 1 single parent

Height(Node):

[length of longest path from node, to any of its descendent leaf nodes]



Note: Path is calculated Based on no: of edges



H(A) = 2
H(B) = 3
H(C) = 4
H(D) = 2
H(E) = 0
H(F) = 2

Obs1:

H(node) = 1 + max(Height of it's child Nodes)

Obs2:

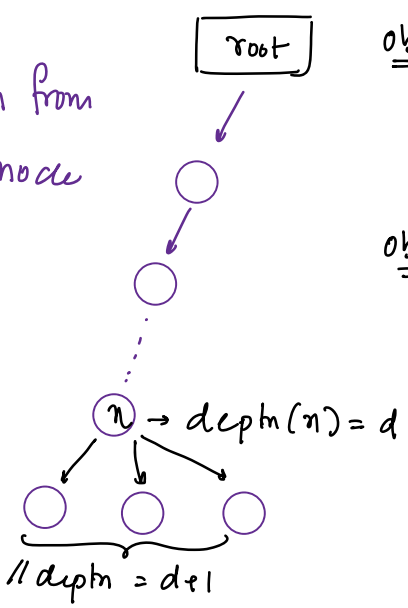
H(leafNode) = 0

[Terminologies
Height(Tree) = Height(root node)
depth(Tree) = max depth of leaf node]

depth(Node):

length of path from root to the node

d(A) = 1
d(F) = 2
d(E) = 5
d(D) = 3



Obs: If depth of Node = d, depth of it's child node = d+1

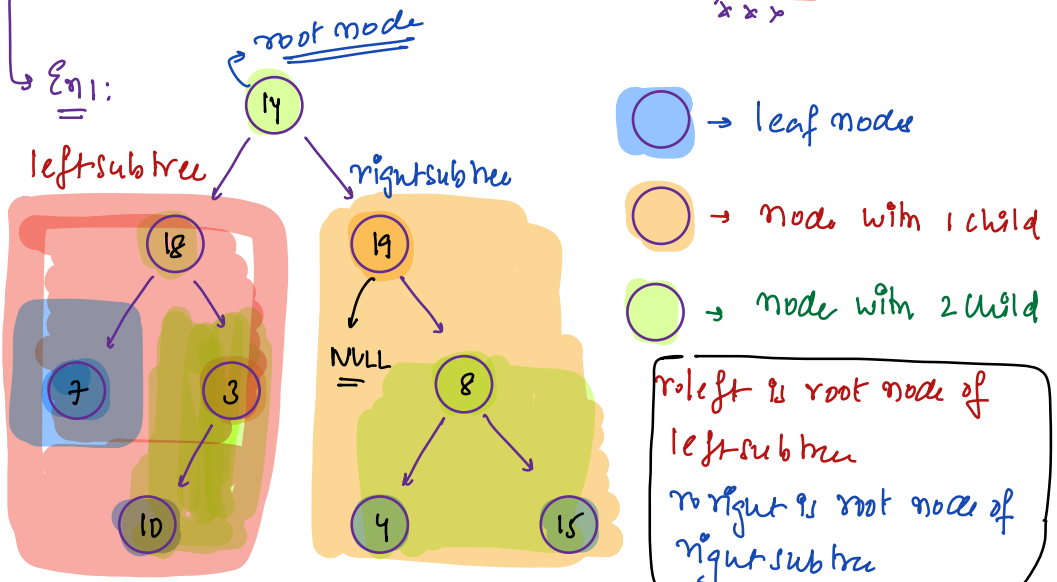
Obs2: depth(root) = 0

[Note: wrong
Height(node) = depth(node)]

Binary Tree : Every node can at max have 2 children

0, 1, 2 | 3, 4, 5 ✗ ✗ ✗

↳ Eg1:

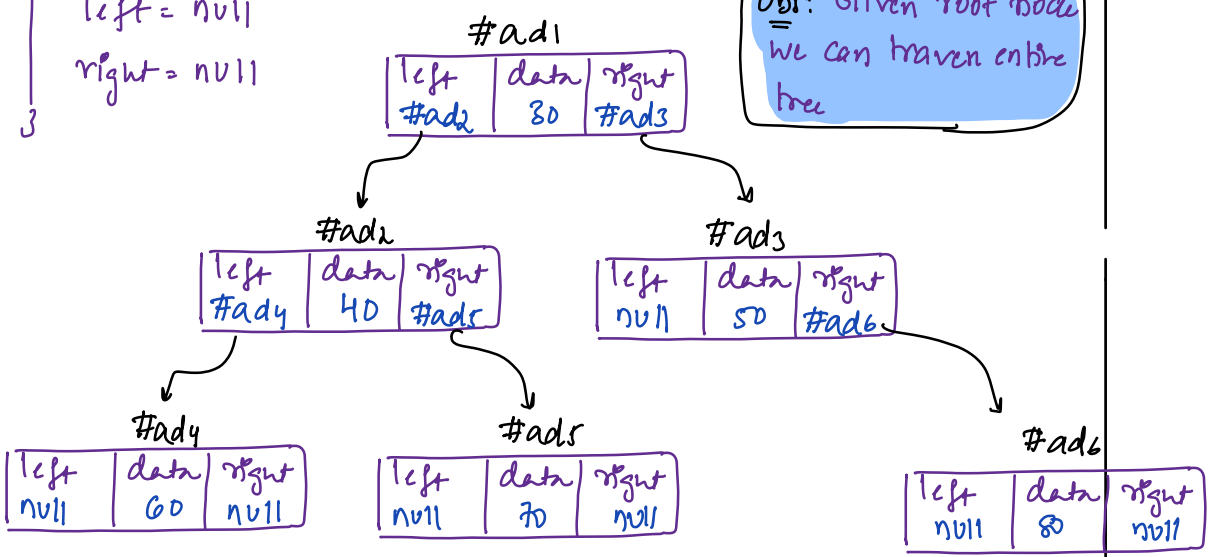


root.left is root node of leftsubtree
root.right is root node of rightsubtree

```
Class node {
    int d;
    node left; // obj refena, which can hold addres of node object
    node right; // obj refena, which can hold addres of node object
    node(int n) {
        d = n
        left = null
        right = null
    }
}
```

Node r = new Node(30)

Obs: Given root node we can traven entire tree



// Tree Construction, can be explained by Serialization & deserialization in adv context

↳ Note: For all Tree problems, tree is already constructed we are just given root node

Tree Traversals:

- → preorder
- → Inorder
- → postorder

- → level order
- → vertical level order

Adv

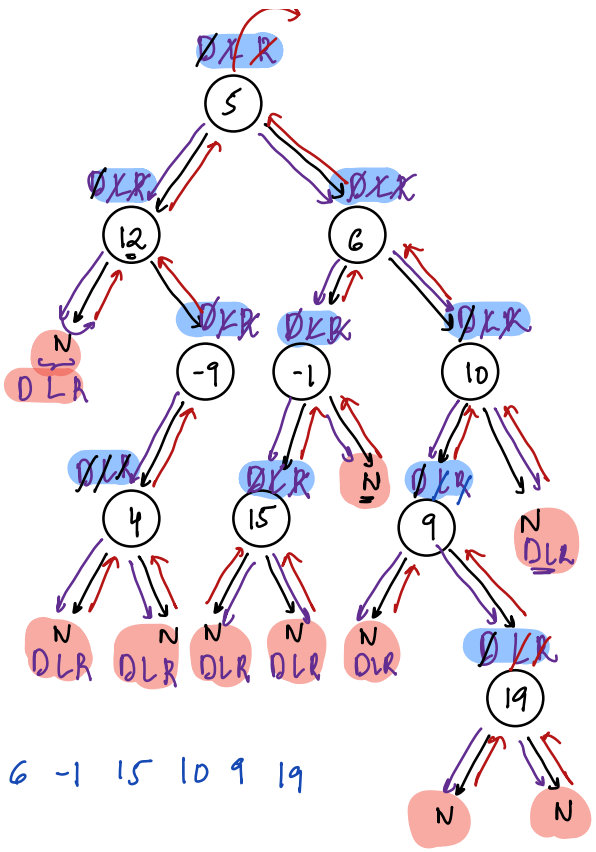
Tree Traversals :

→ Preorder : D L R (D → data, L → LST, R → RST)

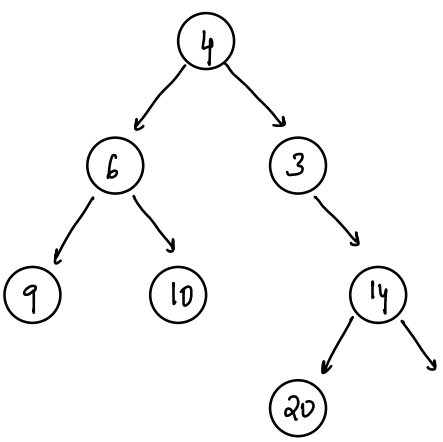
Step1 : print(root.data)

Step2 : goto leftsubtree & print entire leftsubtree in preorder

Step3 : goto rightsubtree & print entire rightsubtree in preorder



→ output : 5, 12 -9 4 6 -1 15 10 9 19



preorder : D L R : 4 6 9 10 3 14 20

Inorder : L D R : 9 6 10 4 3 20 14

postorder : L R D : 9 10 6 20 14 3 4

Pseudo Code:

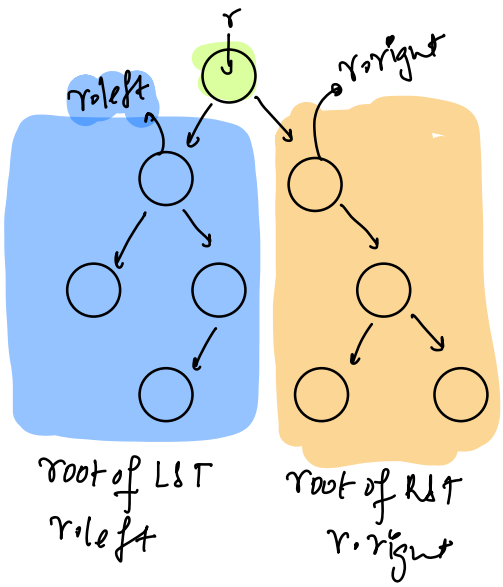
Ass: Given root node, print entire in pre-order

```
void preorder(Node r) {
    1 if(r == null) {return}
    2 print(r.data)
    3 preorder(r.left)
    4 preorder(r.right)
}
```

main logic

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

- Height of Tree
- recursive stack



Pseudo Code:

Preorder: 1 2 3 4

Inorder: 1 3 2 4

Postorder: 1 3 4 2

1 → base

2 → print

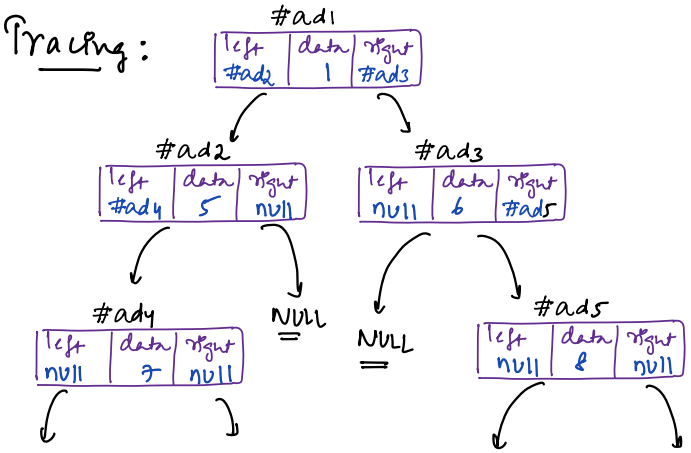
3 → goto left

4 → goto right

TODO

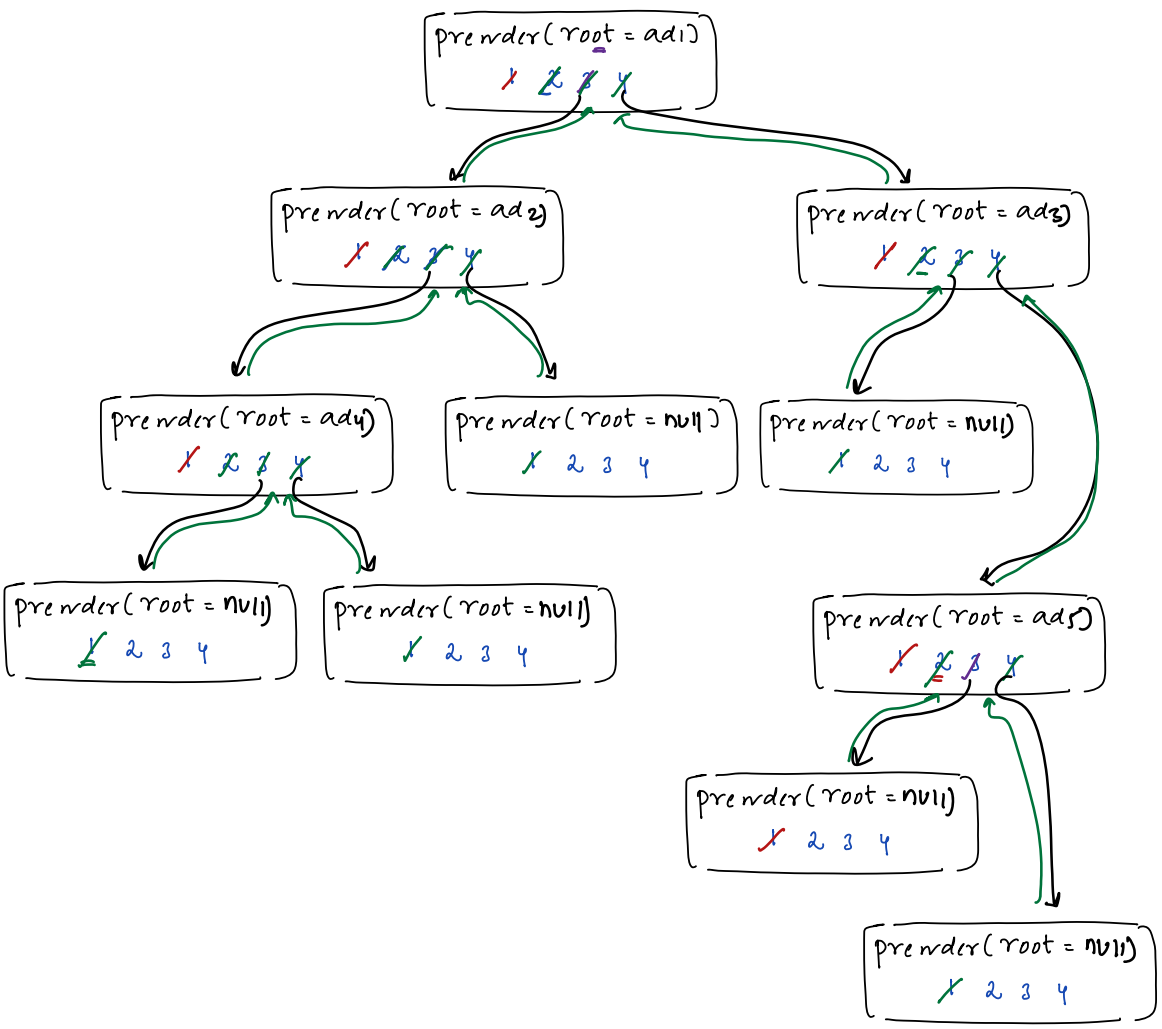
Assignments: Use Recursion

Tracing:




```
void preorder(Node r) {
1  if(r == null) {return}
2  print(r.data)
3  preorder(r.left)
4  preorder(r.right)
}
```

output: 1 5 7 6 8



Tree Problems:

// All below problems solve with recursion & no global variables

a) Size(Node root) ✓

b) Sum(Node root) ✓

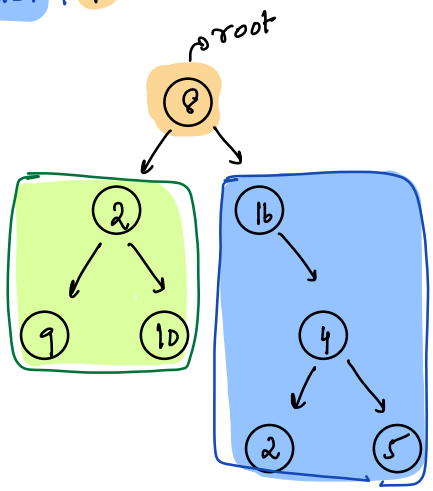
c) Height(Node root) ✓

d) FillDepth() TODO

Ass: Given root node, return size of Tree

size(root) = size of LST + size of RST + 1

```
int size(Node root)
    if(root == null) { return 0; }
    int l = size(root.left)
    int r = size(root.right)
    return l + r + 1
}
```



Ass: Given root node, it will return sum of all nodes

Sum(root) = sum of node LST + sum of node RST + root.data

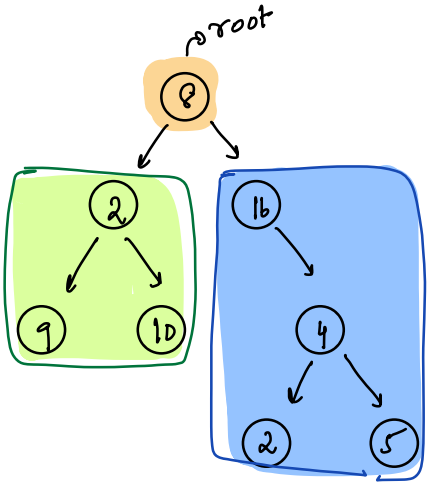
```
int sum(Node root) {
    if(root == null) { return 0; }
    int ls = sum(root.left)
    int rs = sum(root.right)
    return ls + rs + root.data
}
```

↳ value of root node

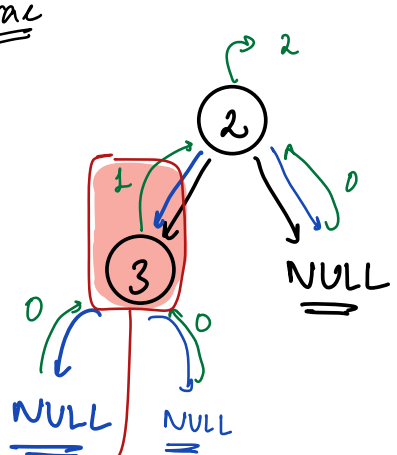
Ass: Given root node, it will height of tree

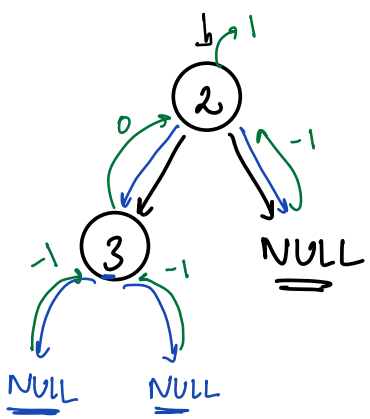
Height(root) = 1 + max(Height of LST, Height of RST)

```
int height(root){
    if(root == NULL) { return -1;
    int hl = height(root.left)
    int hr = height(root.right)
    return max(hl, hr) + 1
}
```



Trace





Issue: leaf node height = 0,
it is returning 1

Note: In your assignment, length of path calculate based on no: of nodes