N. Satya Sai Siva Rama krishna → 2 3/4 years

Todays Content

- Trees Intro
- Naming Convention
- Tree Traversal
- Basic Tree Problems
- LCA
- 1 1/2 y Adv

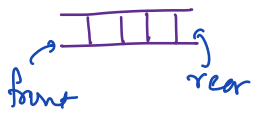
- Recording available students of Scaler
- Notes will be available to everyone
- (Notable to hear/ Notable Sei) Reloud
- language agnostic Pseudocode → [language of choice]

Linear:

↳ arrays

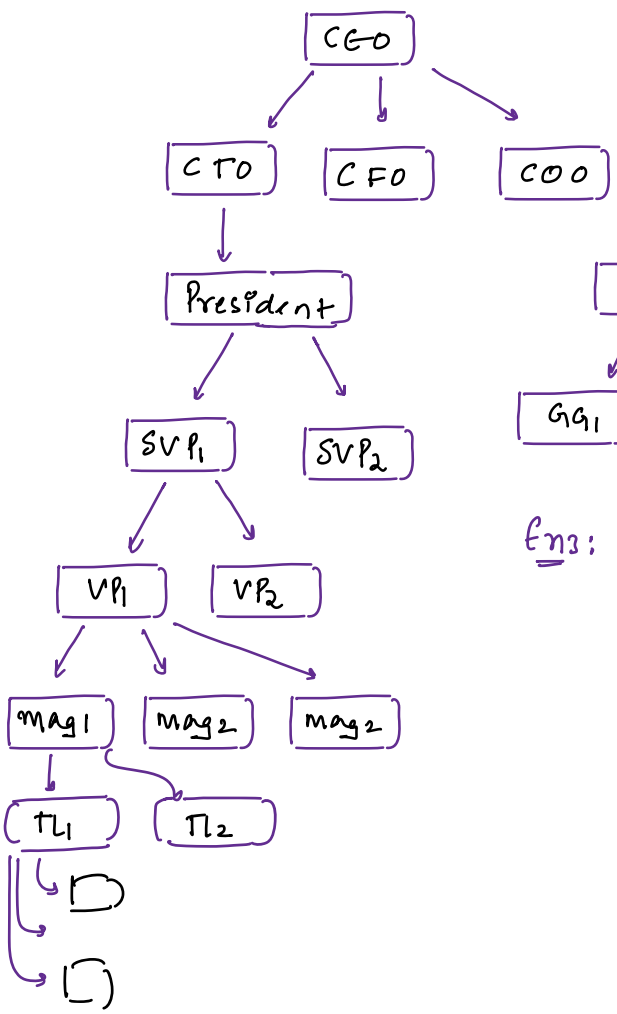
Linked list

stacks & Queues

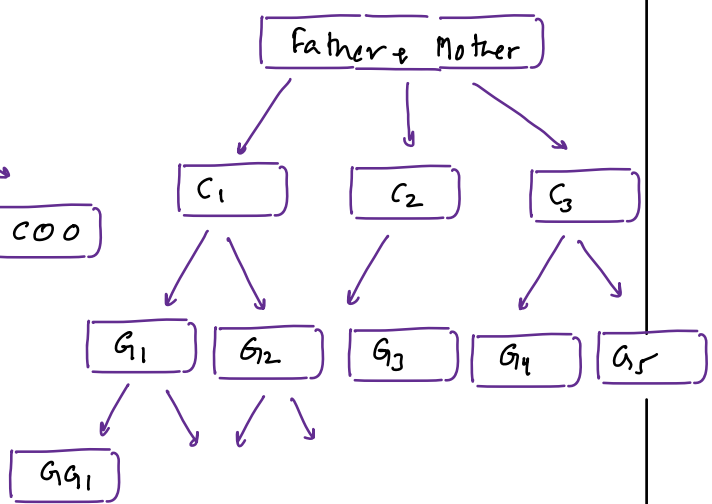


Hirarchial Data

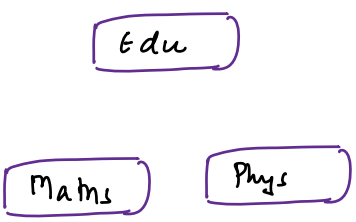
↳ Ex: Company organisation

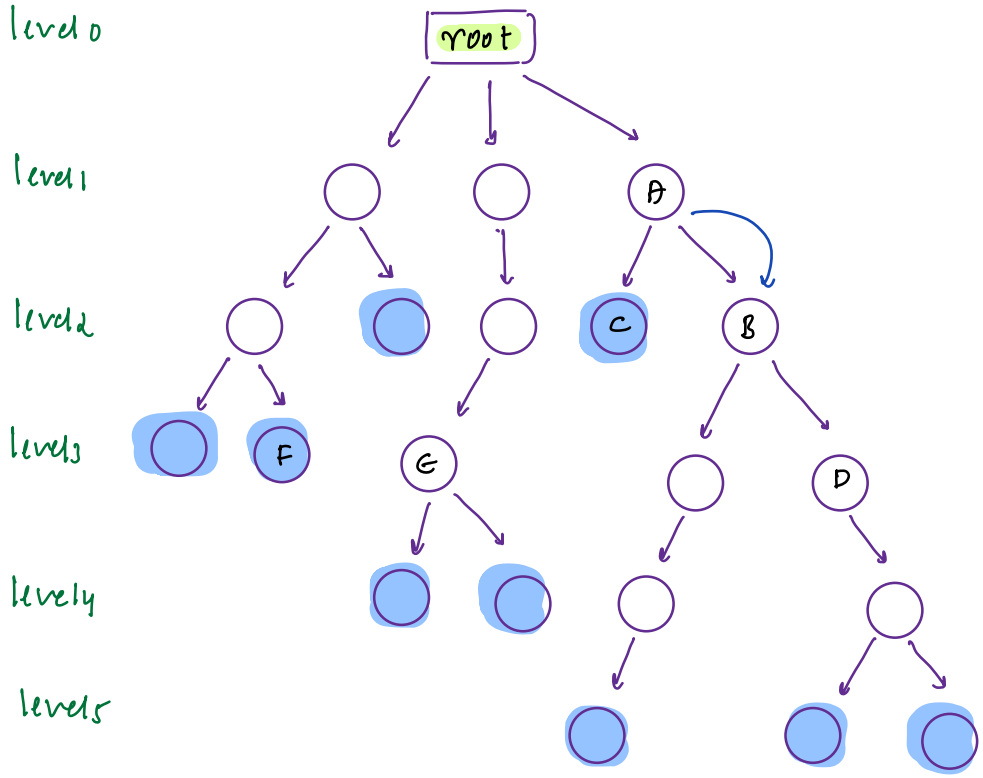


Ex2: Family Tree:



Ex3:





Naming

A → B : A is parent of B, B is child of A

A → D : A is ancestor of D, D is descendent of A

B → C : siblings

F, E, D : nodes at same level

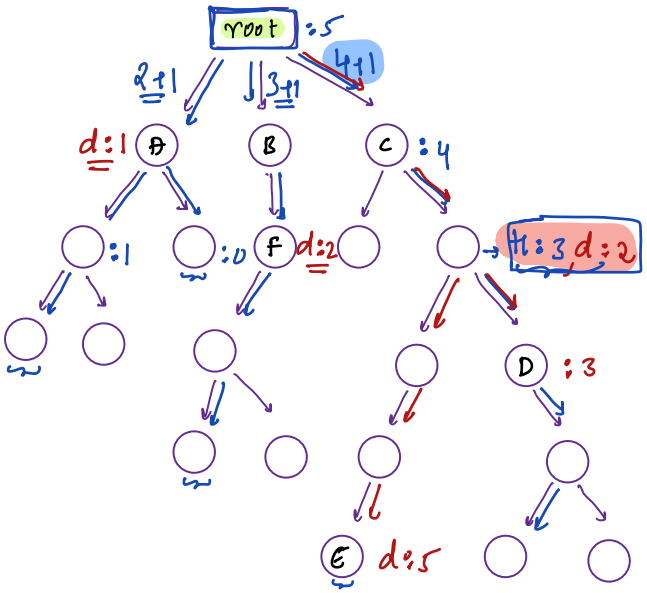
root : node without parent

leaf : node without child

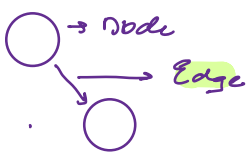
Tree : other than root, every node only single parent

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)
H(F)

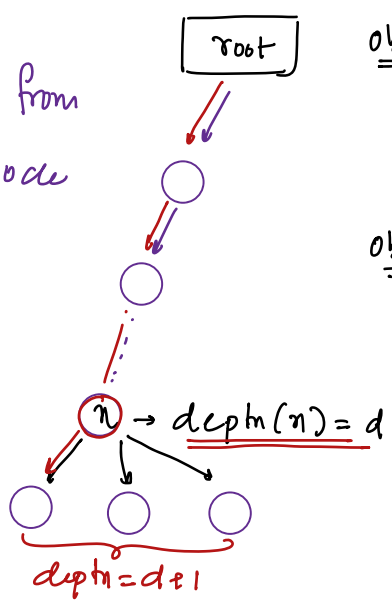
Obs1:
H(node) = 1 + max (Height of child nodes)

Obs2:
H(leafNode) = 0

depth (Node):

length of path from root to the node

d(A) : 1
d(F) : 2
d(E) : 5
d(D) :



Obs: depth(node) = d

depth(child node) = d+1

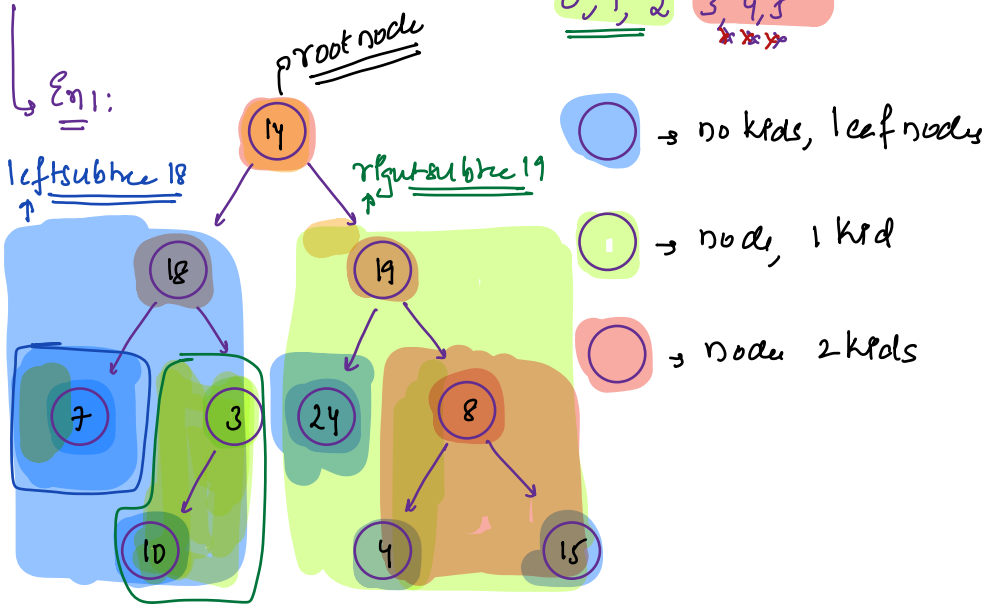
Obs2: depth(root) = 0

Obs: Height(node) = depth(node)

Binary Tree : Every node can at max have 2 children

0, 1, 2 | 3, 4, 5

Eg1:

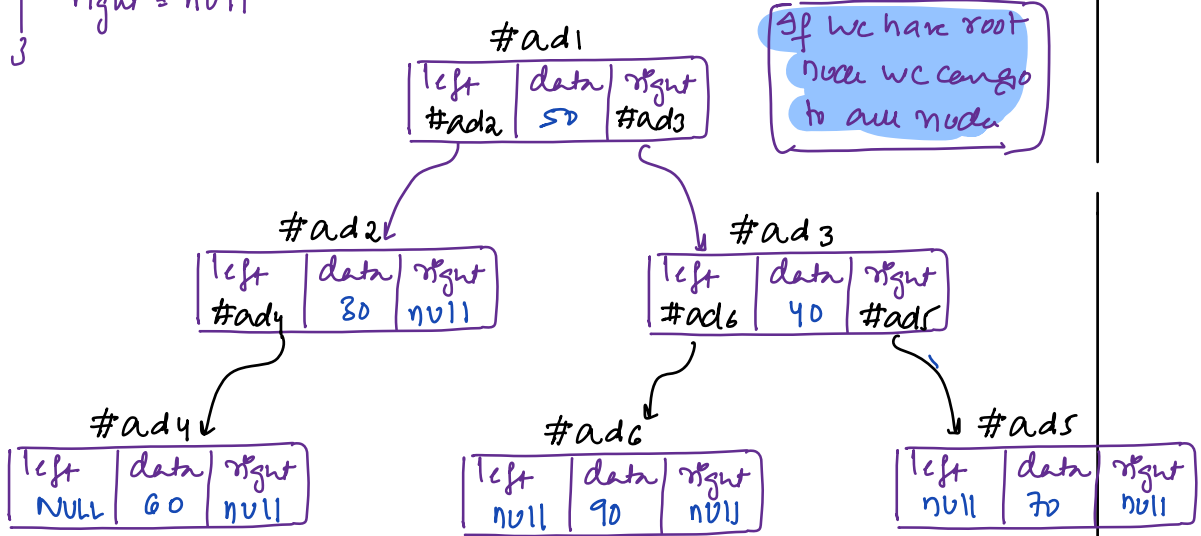



```
Class node {
    int d; // data
    node left; //
    node right; //
    node(int n) {
        d = n
        left = null
        right = null
    }
}
```

→ [object reference of node class, can hold address of node object]

p→ data node = 50

Node root = new Node(50)



Tree Traversals:

- (given root node)
- Tree Traversals → {Iterate in complete Tree}
  1) Preorder
  2) Inorder
  3) Postorder

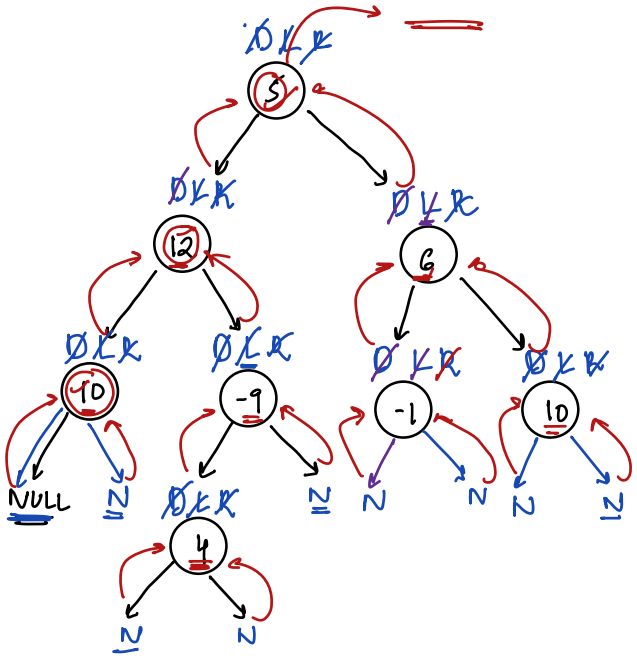
Tree Traversals :

→ Preorder : D L R

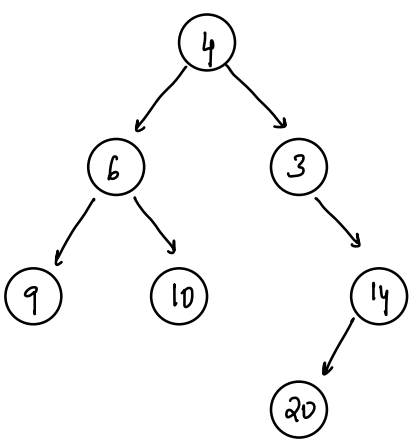
Step1 : Print root data

Step2 : Goto leftsubtree & print entire leftsubtree in preorder

Step3 : Goto rightsubtree & print entire rightsubtree in preorder



Output: 5 12 10 -9 4 6 -1 10



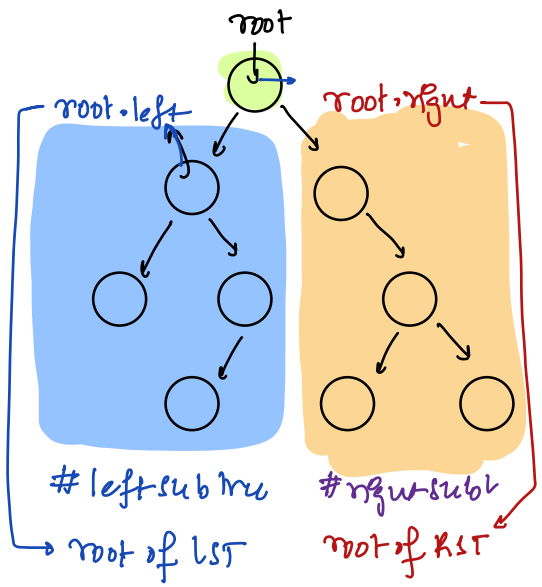
preorder: 4 6 9 10 3 14 20

preorder:

Pseudo Code:

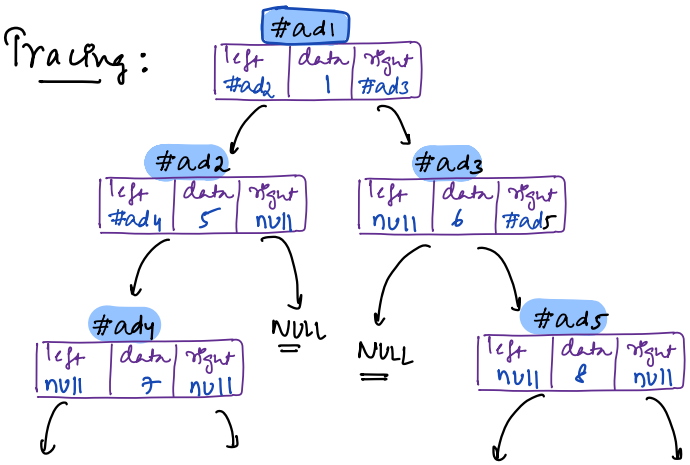
Given root node, print entire in pre-order

```
void preorder(Node r) {        // r → root node
1   if(r == null) {return}
2   print(r.data)
3   preorder(r.left)
        // print preorder of LST
4   preorder(r.right)
        // print preorder of RST
3
```



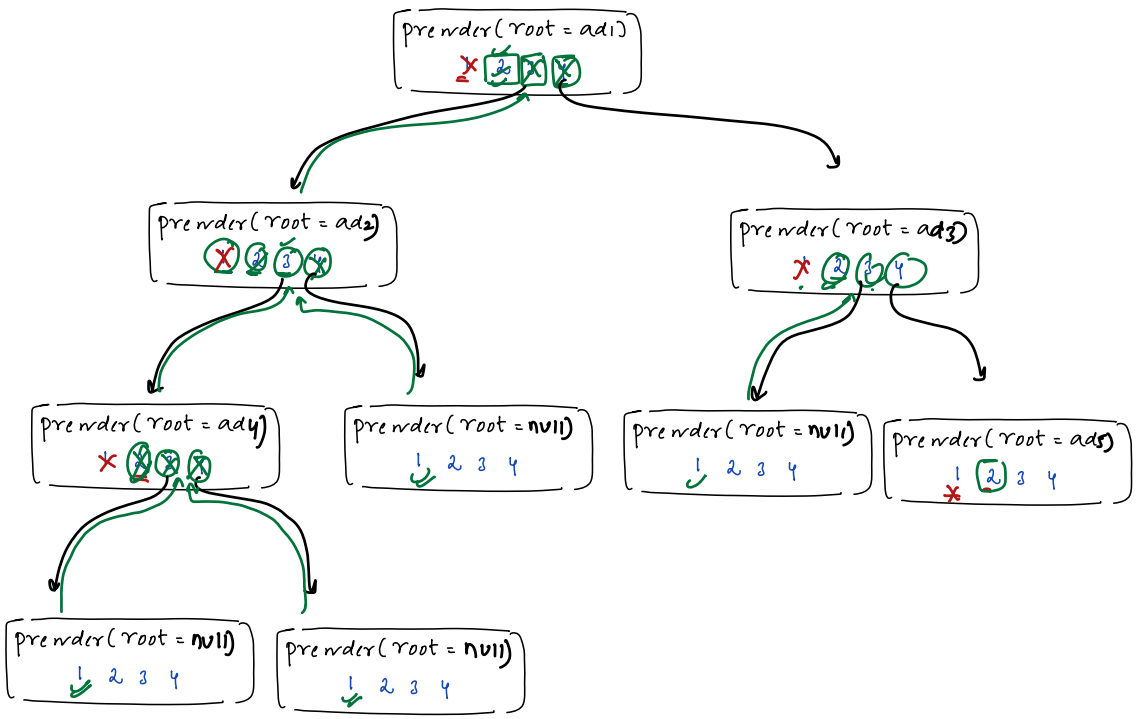
Pseudo Code:

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



// Note: When a function is completely executed, it will return back.

Tree Problems:

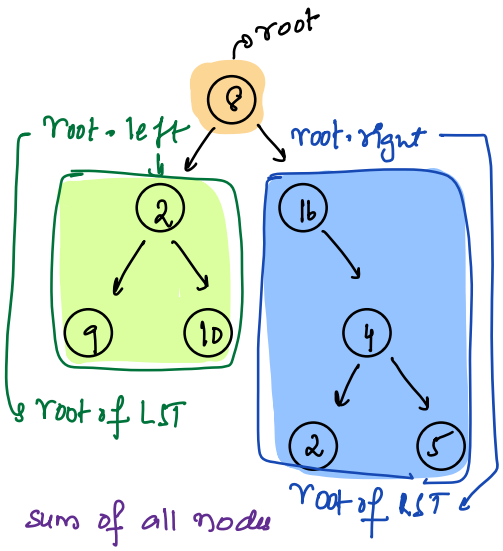
a) Size(Node root) ✓

b) Sum(Node root) ✓

Ass: Given root node, return size of Tree

Size(root) = // Size of LST + Size of RST + 1

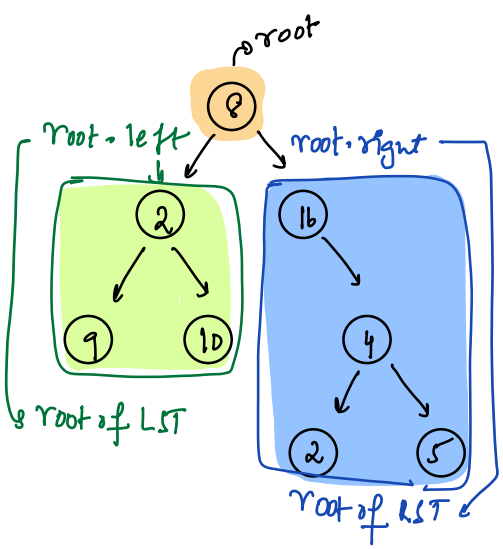
```
int size(Node root)
{
    if (root == null) { return 0; }
    int l = size(root.left);
    int r = size(root.right);
    return l + r + 1;
}
```



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

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

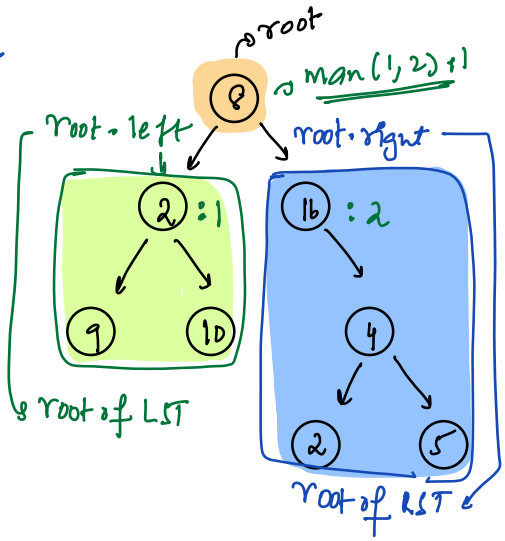
```
int sum(Node root) {
    if(root == null) { return 0; }
    int ls = sum(root.left);
    int rs = sum(root.right);
    return ls + rs + root.data;
}
```



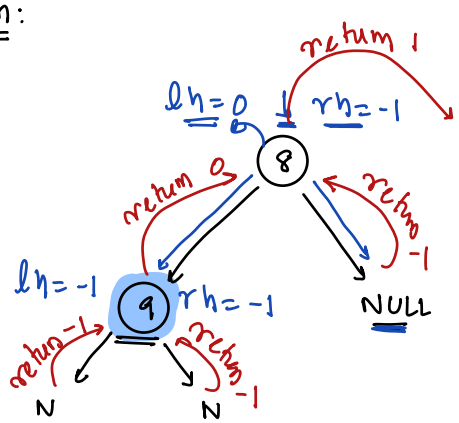
// Given root node, get height of node

```
int height(Node root){
    if(root==null){return -1;}
    int lh = height(root.left);
    int rh = height(root.right);
    return max(lh, rh)+1;
}
```

change

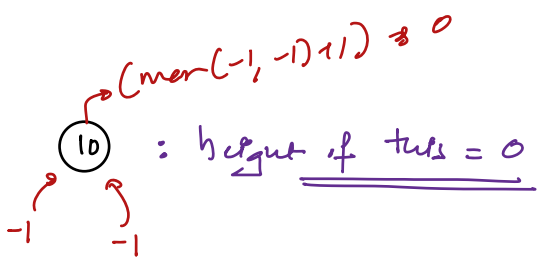


Eg:

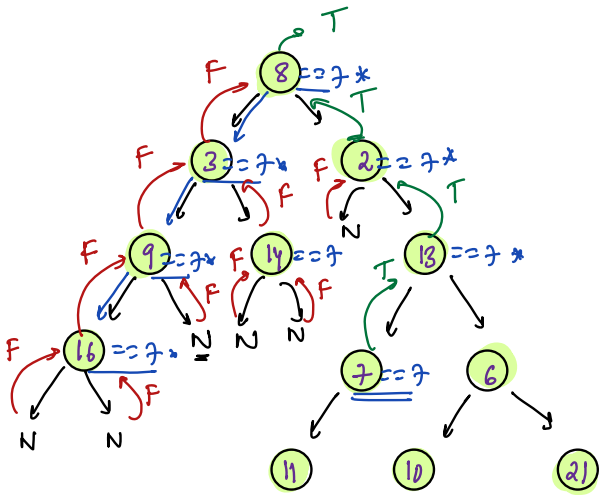


Issue: height of leaf node = 0
but we are returning 1

Eg:



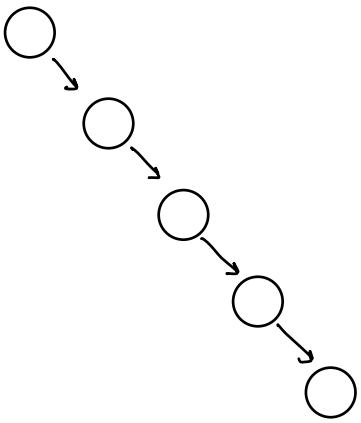
// Given BT & root node, all value are distinct, search for a node with value = k., k=7 k=25



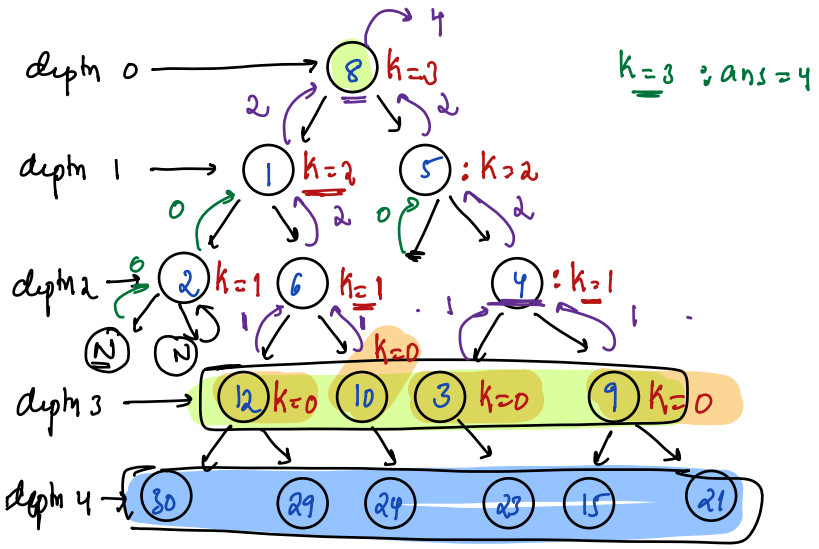

```
bool search(Node r, int k){
    if(root==NULL){ return false}
    if(r.data==k){ return True}
    if( search(r.left, k) ||
            search(r.right, k){
        return true
    }
    return false
}
```

// If N node in tree, max height = (N-1)

N=5 : H=4



// Given root node & k, calculate no: of node at depth k.



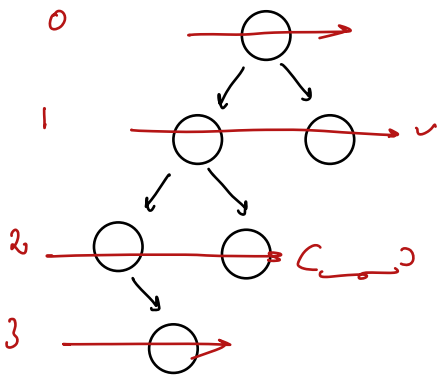

```
int countdepth(Node root, int k){
    if(root==null){return 0;}
    if(k==0){return 1;}        // node at depth k from root node
    int l = countdepth(root.left, k-1);
    int r = countdepth(root.right, k-1);
    return l+r;
}
```

CBT: Complete Binary Tree

↳ BT as CBT

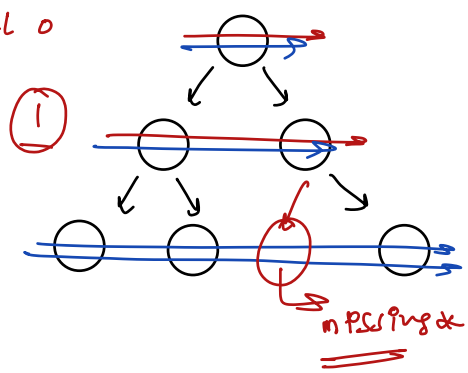
a) Nodes should be filled level by level, untill all nodes at current level are filled, dont go to next level, last level need not be complete

b) Create node left → right in each level



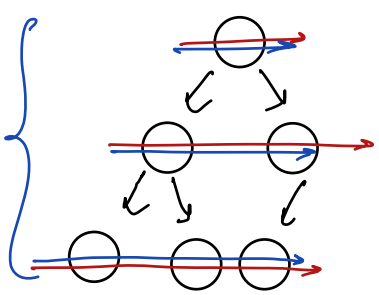
// without completing level 2, we should not goto level 3

level 0

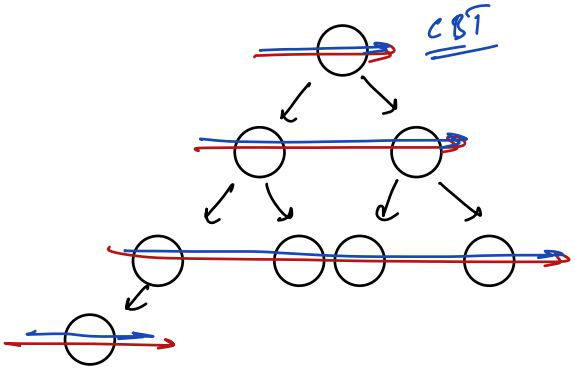


not CBT

CBT

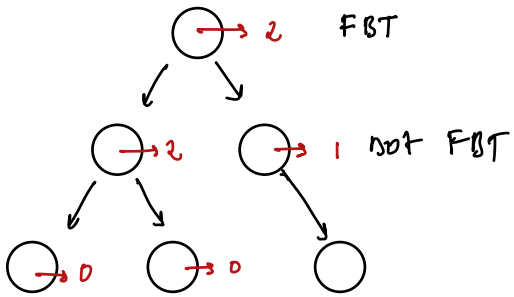


CBT

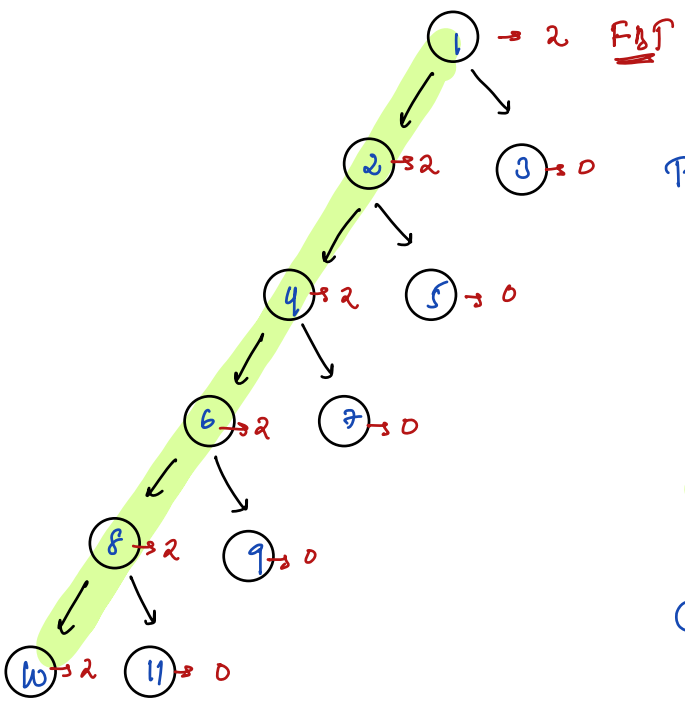


→ FBT: (Full Binary Tree)

↳ Every node 0 or 2 child nodes



→



Html

Tree → evaluate

→ file system

→ CBT

→ Heaps data structure

BST

↳ Balanced Binary Search Tree