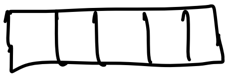
Welcome ☺

Agenda :
1. Trees Intro
2. Naming conventions
3. Traversal
4. Basic Tree Problems.

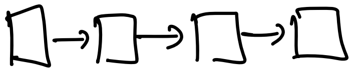
Advance classes → 21st August 9:00 pm

Linear

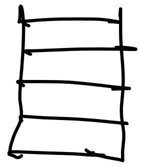
arrays



linked list



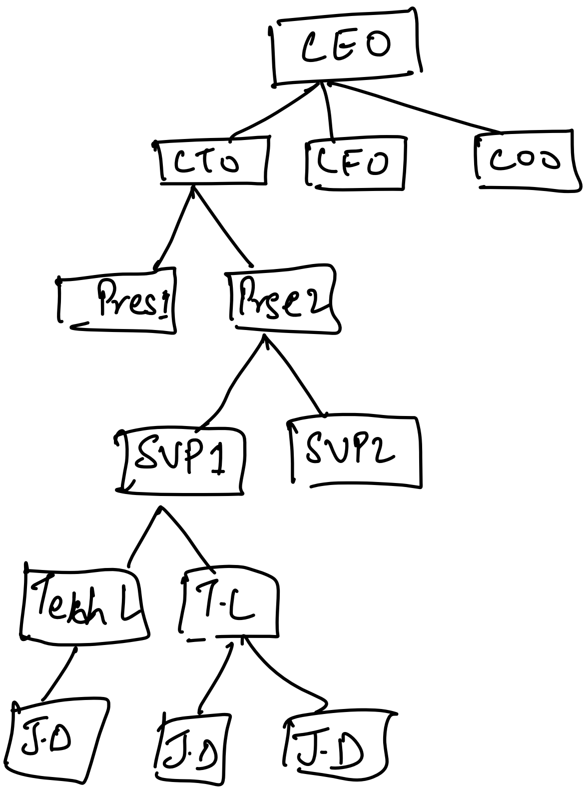
Stacks

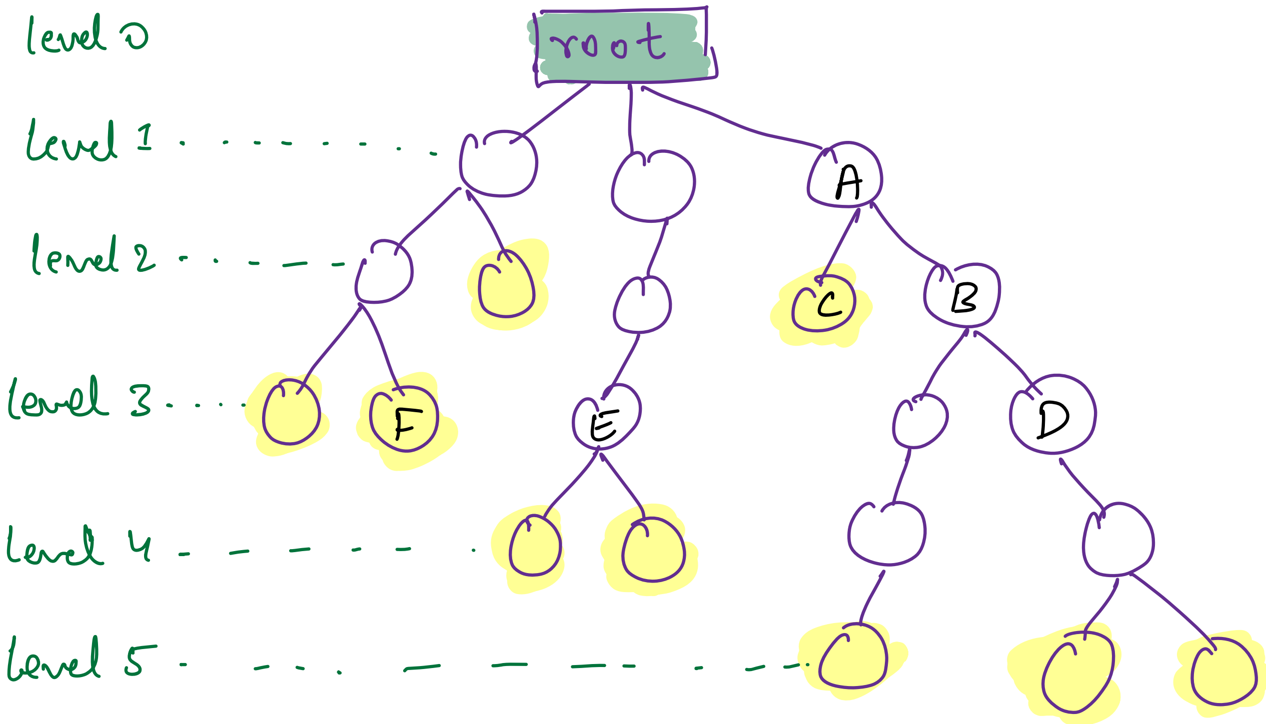


Queue.

Heirarchical D.S

eg: Company Organisa^{n}





Naming

A → D → A is ancestor to D & D is descendant of A

A → B → A is parent to B & B is child of A

B → C → B & C are siblings.

F, E, D → nodes at same level

root → Node without a parent

leaf → Node without a children

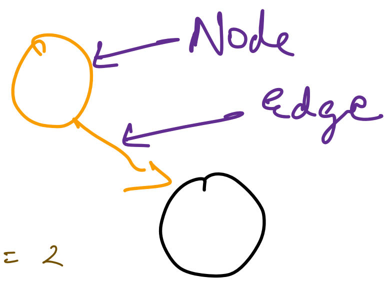
Trees → ① will only have 1 root node

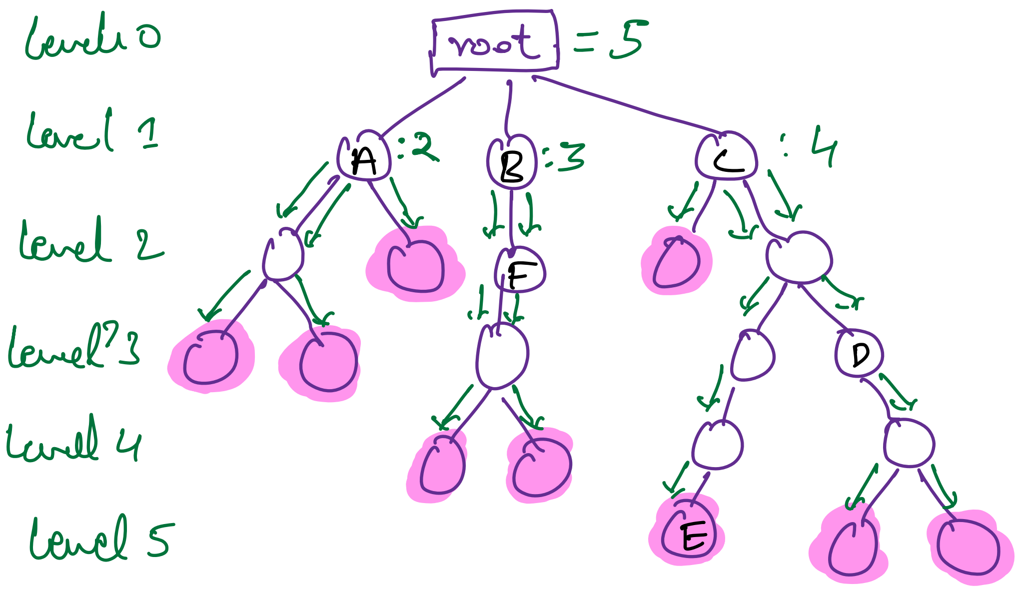
② For every node, only 1 parent can be there

Height of a Tree

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

→ Height is calculated based on no. of edges





Obs 1 :

$H(\text{node}) = 1 + \max(\text{height of its child nodes})$

Obs 2

$H(\text{leaf}) = 0$

$H_A = 2$

$H_B = 3$

$H_C = 4$

$H_{root} = \max(2, 3, 4) + 1$

$H_E = 0$

Depth (Node)

length of path from root to node

$d_A = 1$

$d_F = 2$

$d_E = 5$

obs 1

If depth of node $= d$,
depth of child node $= d+1$

obs 2

depth(root node) = 0

obs 3

depth of node = level of node.

obs 4

~~Height of node = Depth of node.~~ ✗

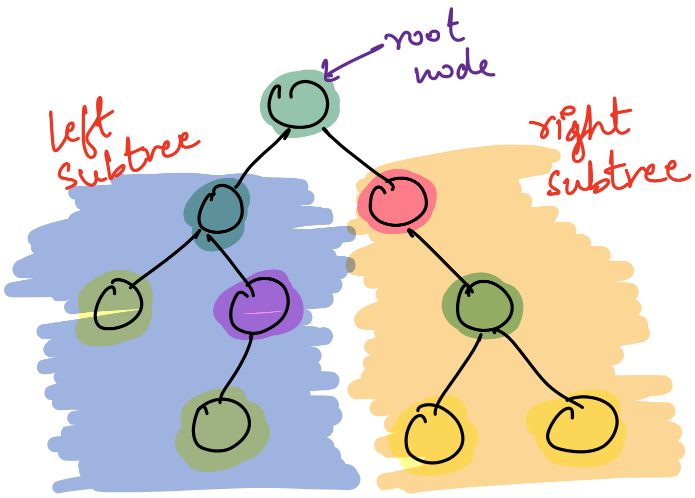
Wrong

Binary Trees

⇒ Every node can have atmax 2 children.

0 ✓ 1 ✓ 2 ✓ 3 ✗ 4 ✗ 5 ✗

Eg:



→ leaf nodes.

→ node with 1 child

→ node with 2 child

D.S of a tree node

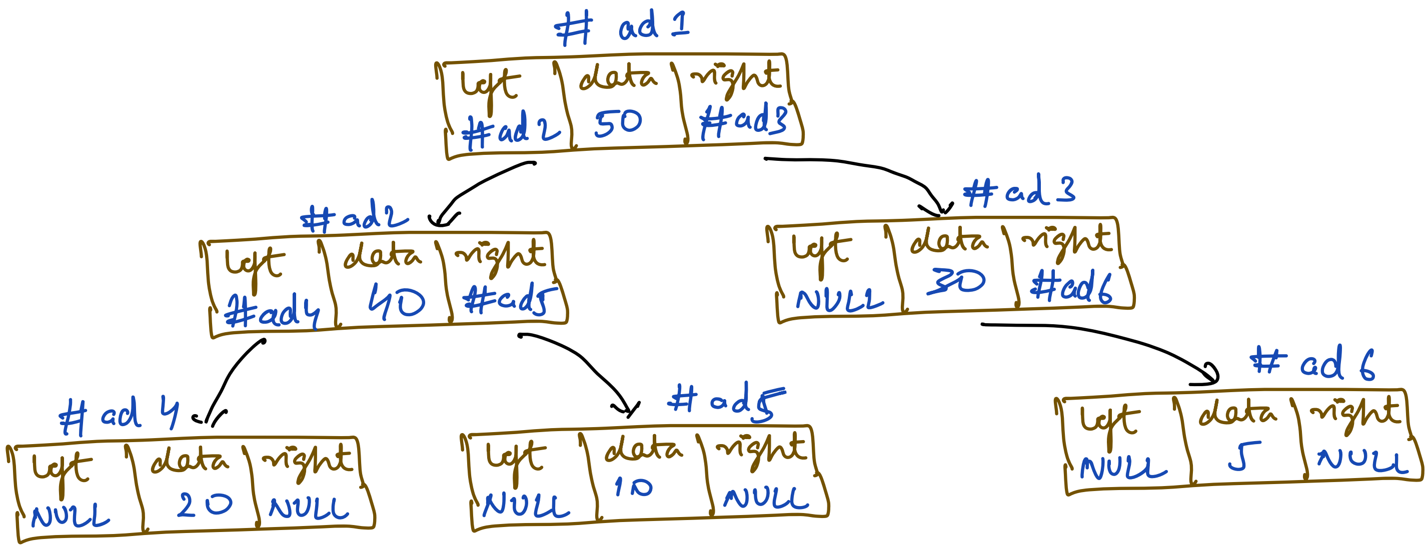
LEFT	DATA	RIGHT

root left ⇒ root node of left subtree

root right ⇒ root node of right subtree.

```
Class Node
{
    int data ;
    Node left ; // object reference -> holds address of left child node
    Node right ; // holds address of right child node.
    Node ( int n )
    {
        data = n ;
        left = NULL ;
        right = NULL ;
    }
}
```

Node r = new Node (30)



⇒ Tree construction can be explained using serialization & deserialization

Advance module

Tree Traversal

1) Inorder
2) Preorder
3) Post Order
4) level order
5) Vertical level order

Adv. module.

Preorder

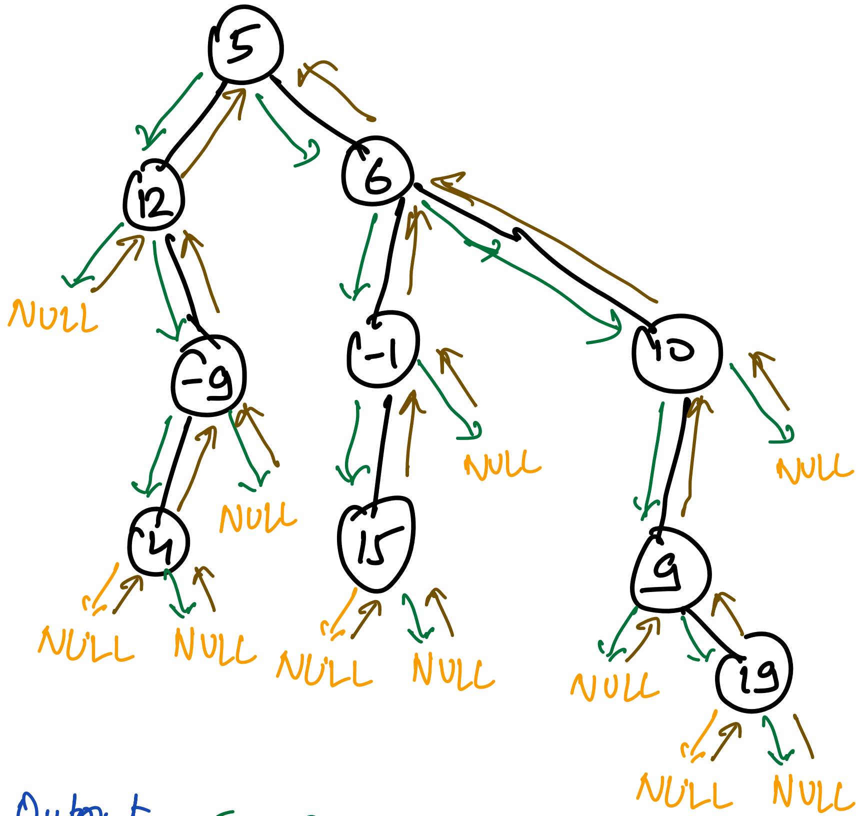
D L R

data · left subtree · right subtree

Step 1 : print (data)

Step 2 : Goto left subtree and print entire left subtree in preorder

Step 3 : Goto right subtree and print entire right subtree in preorder



Output 5, 12, -9, 4, 6, -1, 15, 10, 9, 19

Preorder → DLR 5, 12, -9, 4, 6, -1, 15, 10, 9, 19

Inorder → LDR 12, 4, -9, 5, 15, -1, 6, 9, 19, 10

Postorder → LRD 4, -9, 12, 15, -1, 19, 9, 10, 6, 5

Pseudocode

Ass: Given root node, print entire tree in preorder.

```
void preOrder ( Node root )
{
  ① if( root == NULL) return;
  ② print ( root.data)
  ③ preOrder ( root.left)
  ④ preOrder ( root.right)
}
```

T.C ⇒ O(N)

S.C ⇒ O(H)

↓

Height of tree

Steps

Preorder ⇒	1	2	3	4
Inorder ⇒	1	3	2	4
Post Order ⇒	1	3	4	2

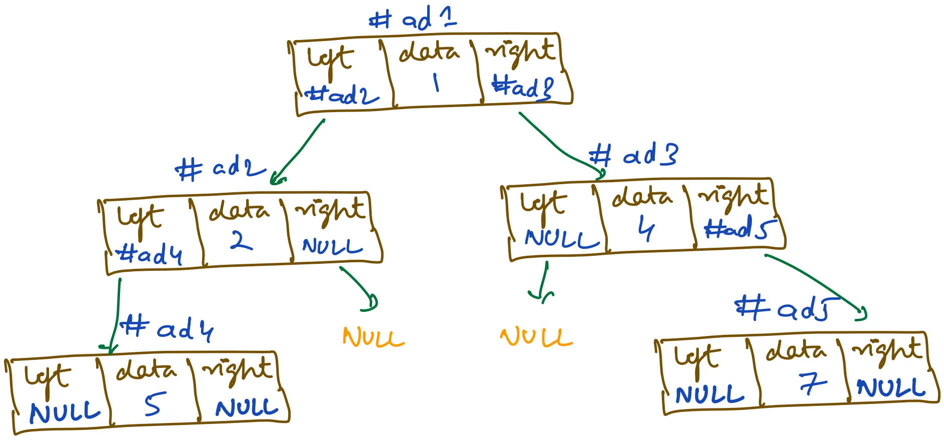
① → base

② → data print

③ → goto left

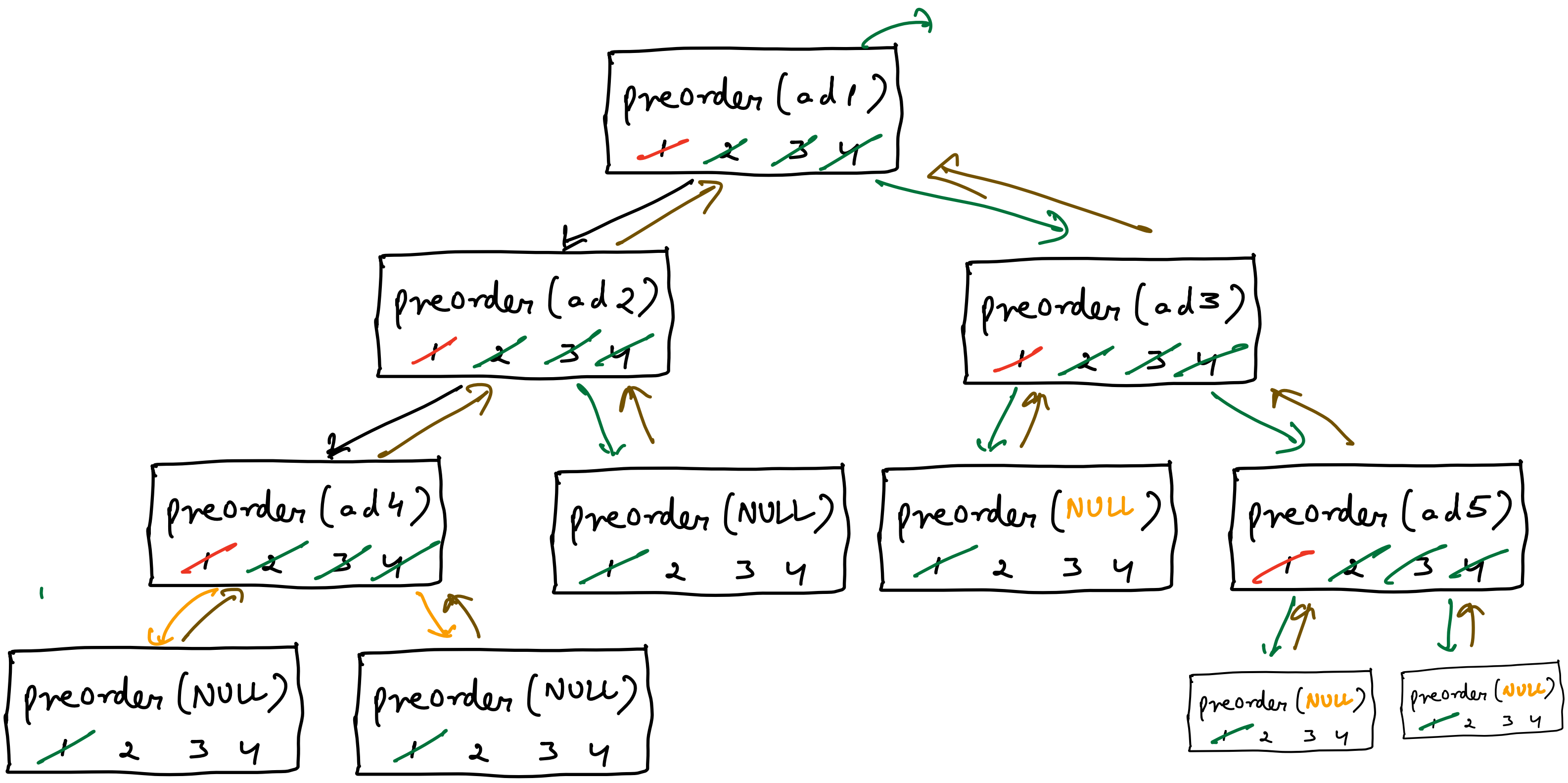
④ → goto right.

TODO → recursion




```
void preOrder ( Node root )
{
  ① if( root == NULL) return;
  ② print ( root . data)
  ③ preOrder ( root . left)
  ④ preOrder ( root . right)
}
```

o/p ⇒ 1, 2, 5, 4, 7

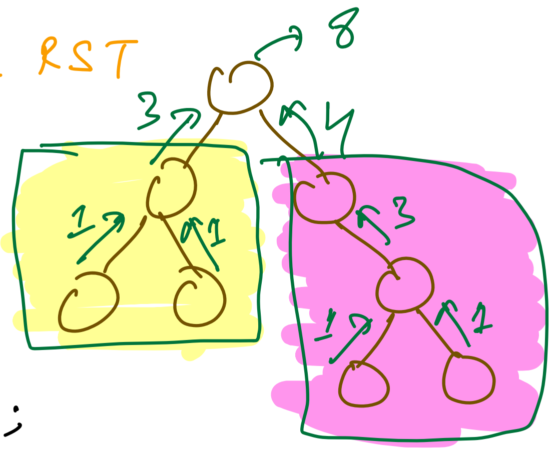


Tree Problems

⇒ No global variable. Solve with recursion

① Given root node, find and return size of tree

Size of node = size of LST + size of RST + 1




```
int size( Node root)
{
    if( root == NULL) return 0;
    int l = size( root.left) ⇒ size of LST
    int r = size( root.right) ⇒ size of RST
    return l + r + 1
}
```

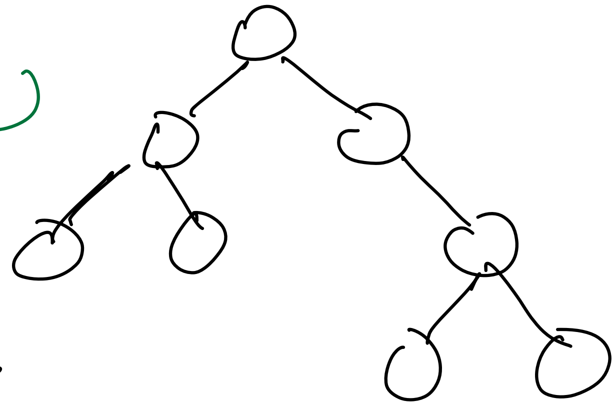
Q2 Given root node, return sum of all nodes.

```
int sum( Node root)
{
    if( root == NULL) return 0;
    int l = sum( root.left) ⇒ sum of LST
    int r = sum( root.right) ⇒ sum of RST
    return l + r + root.data
}
```

↳ value of node

Q3 Given root node, return height of tree.

Height(node) = 1 + max(height LST , height RST)



```
int height( Node root)
{
    if( root == NULL) return -1;
    int lh = height( root.left)   ⇒ sum of LST
    int rh = height( root.right)  ⇒ sum of RST
    return max( lh, rh) + 1;
}
```

eg: