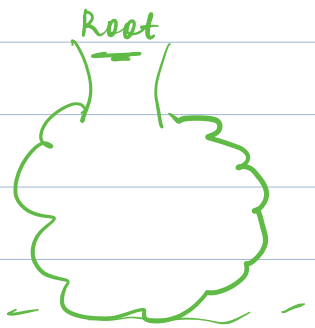
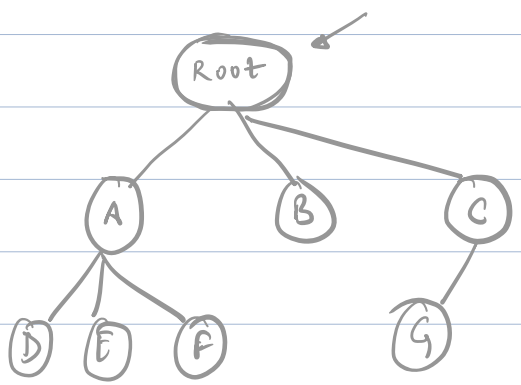
Agenda.

- Trees basics → Use cases of trees
- Terminologies:
- Traversals
- Height of tree
- Size of tree.



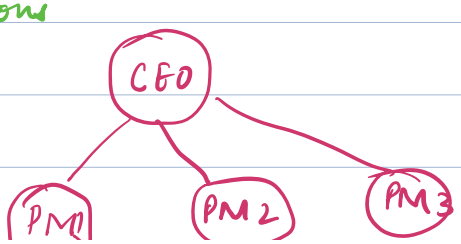
DS → Linear / Sequential.

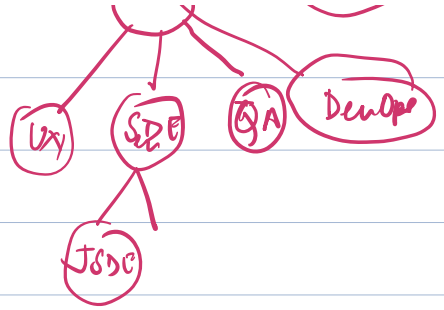
Trees → Heirarchial DS



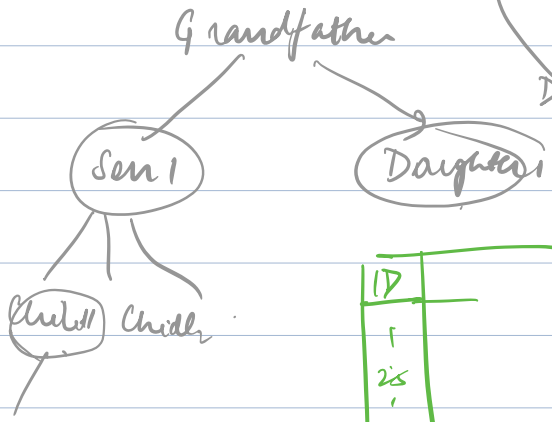
Use cases:

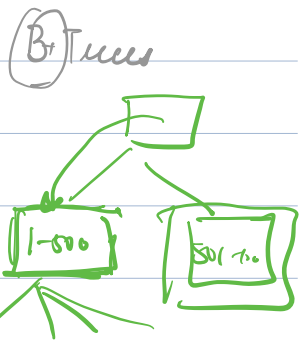
1. Organisations



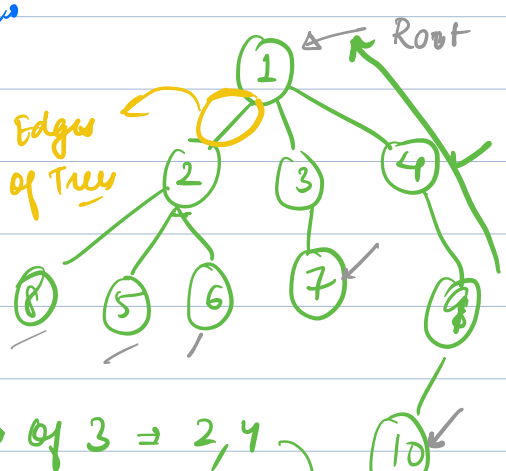


(2) Family Tree





Terminologies



Parent node of 7 ?
⇓
3

Children of 2
⇓
8, 5, 6

Siblings → of 3 ⇒ 2, 4

ancestors → of 9 = 4, 1 (common parent)

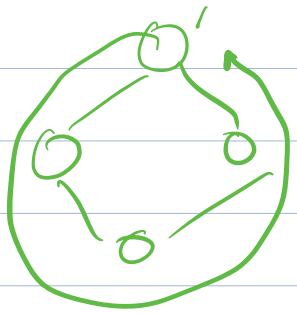
Successor of 4 ⇒ 9, 10

Leaf Nodes
⇓
8, 5, 6, 7, 10

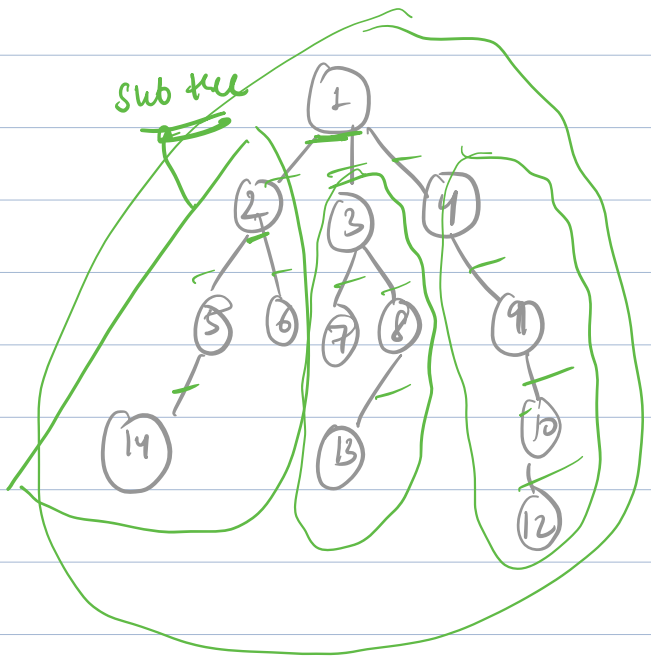
Node is combination of data, edges / connection to the children node.

① 1 node can have how many parents ⇒ 1

② No cycles



Reaching back same node without repeating any node



count of nodes = 13

Every node has 1 parent except root.

Total edges = 12

n nodes ⇒ n-1 edges

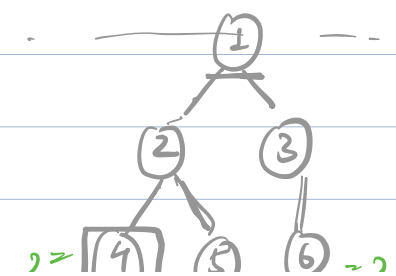
Sub trees are the reason we can seamlessly apply recursion to trees problems.

Properties of Tree:

① Height of a node:

⇓

Distance from the farthest leaf node



in the subtree

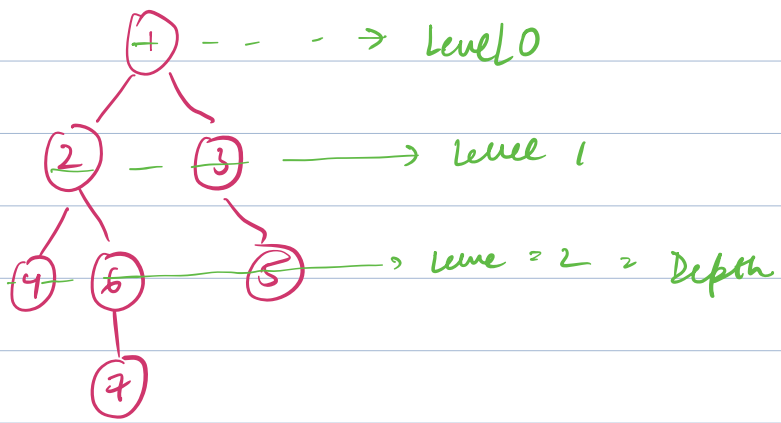
height(1) = height(tree) height(7)

= 3.

(2) Depth of a node:

Distance from the root node

(3) Levels

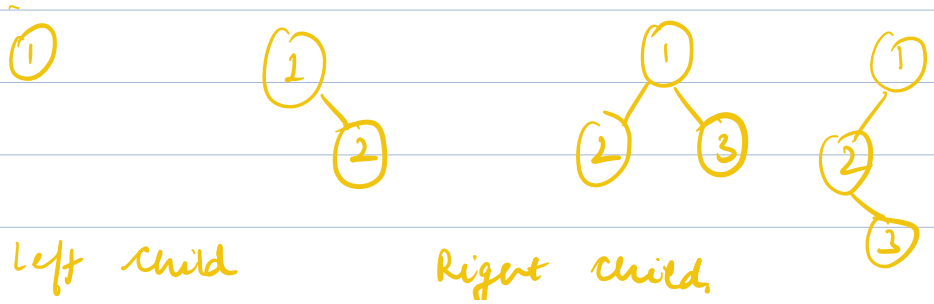


Binary Trees

↳ Every node can have a max of 2 children

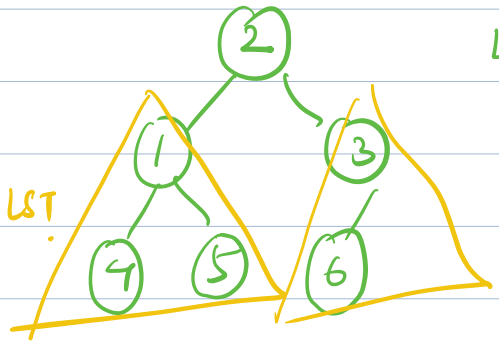
Ternary Trees ⇒ At max (3) children

K-ary Trees ⇒ At max (K) children.

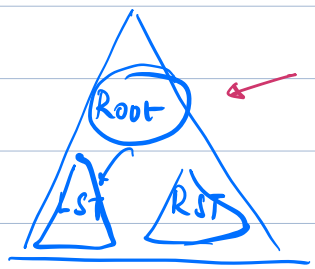


Left child Right child

Left subtree Right subtree



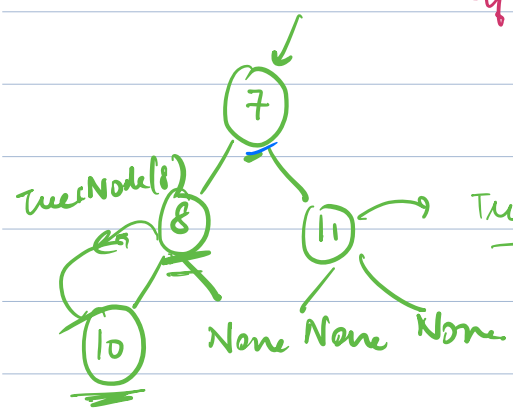
Left subtree of 2



Implementation (Binary Tree)

```
class TreeNode:
    def __init__(self, data):
        self.data = data.
        self.left = None  ⇒
        self.right = None. ⇒
```

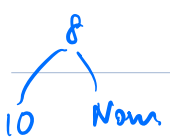
References to the root node of LST, RST



TreeNode(11) :

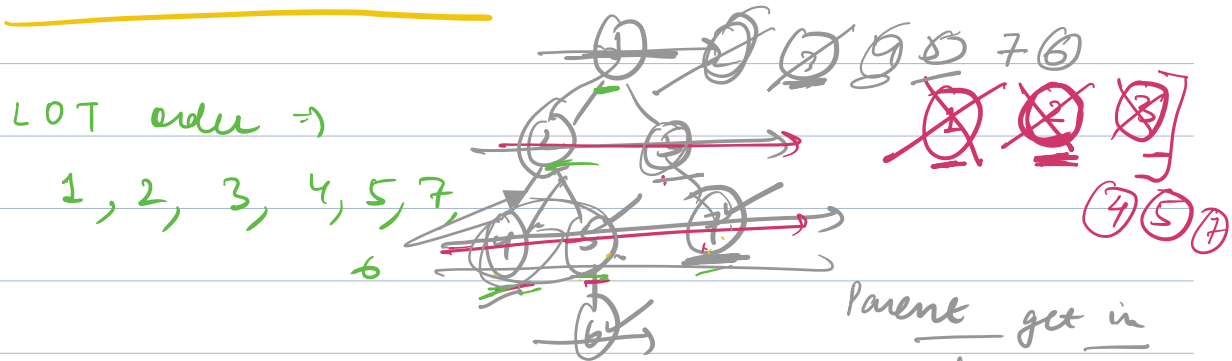
root.data = 7
root.left = TreeNode(8)
root.right = TreeNode(11)

x = TreeNode(10)




```
y = TreeNode(8)
y.left = x
root = TreeNode(7)
root.left = y
root.right = TreeNode(11)
```

Level Order Traversal



LOT order →

1, 2, 3, 4, 5, 7, 6

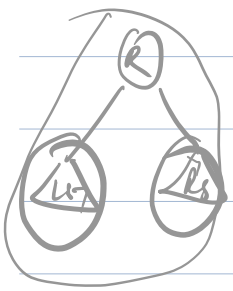
Parent get in first

children get in first

```
def printLevelOrder(TreeNode root):
    qu = collections.deque([root])
    while not qu.empty():
        x = qu.popleft()
        if x.left:
            qu.append(x.left)
        if x.right:
            qu.append(x.right)
        print(x.data)
```

1 2 3 4 5 7 6

Q Given the root node of a Binary tree. Find the total number of nodes in the tree?




```
def size( root: ):
    # Assumption: This fn returns me
                  size of tree rooted at root node
    if root == None:
        return 0
    return 1 + size(root.left) + size(root.right)
```

BC

ML

```
def size(root):
    # BC => if a leaf Node
    if !root.left and !root.right:
        return 1
    => cnt = 1
    if root.left:
        cnt += size(root.left)
    if root.right
        cnt += size(root.right)
```

return cnt.

1+2+1 = 4

Break - 10 min

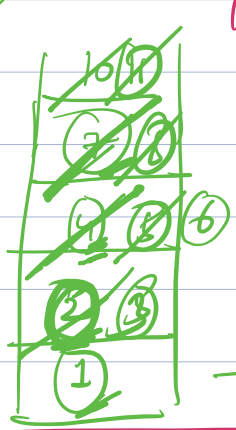
Traversal:

- Preorder Traversal → Root | Left | Right
- Inorder Traversal
- Post Order Traversal

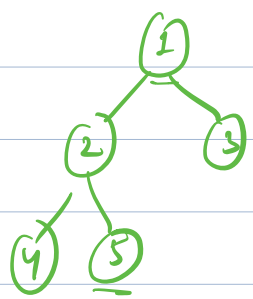
Root left Right

1 2 4 7 5 8 3 6

```
def preorder(root):
    print(root.data)
    if root.left:
        preorder(root.left)
    if root.right:
        preorder(root.right)
```



```
def preorder(root)
1   if root == None:
2       return
3   print(root.data)
4   preorder(root.left)
5   preorder(root.right)
```



preorder(1)
preorder(2)
preorder(4)

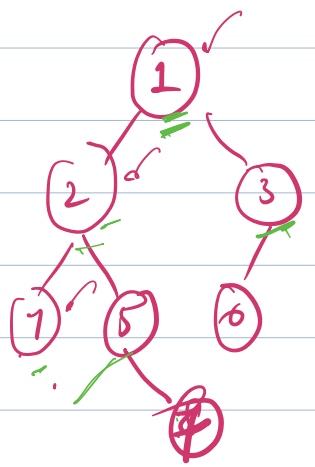
1 2 4 5

Post Order

Left Right Root

Inorder

Left Root Right



4 7 5 2 6 3 1

4 2 5 7 1 6 3

postorder(left)
postorder(right)
print root

inorder(left)
print root
inorder(right)

H.W find height of tree

```
def height(root):
    pass
```