02/03/22

DSML Intermediate – DSA

Tree – Basics 1

✓ ① Tree Basics & Terminologies

✓ ② Tree Traversals

✓ ③ Finding Size of Tree

Next ④ Finding Height of Tree

Content

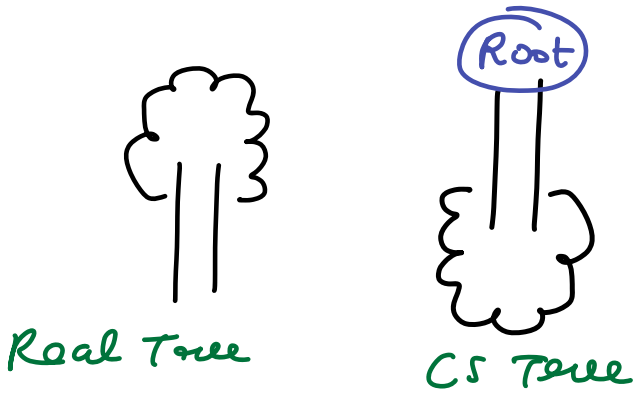
Today → Tree Basics 1 (2nd March)

· Tree-2 Basics (4th March)

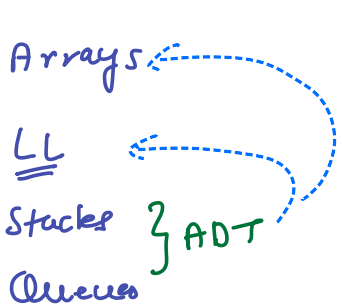
Contest Discussion – 6th March

7, 9, 11 March Break | Problem Solving Sessions

DSML Adv Content: 15th March Onwards


Root
Real Tree
CS Tree

① DS Till Now



start end

Linear & Sequential

Direction — 5th element.

A[4]

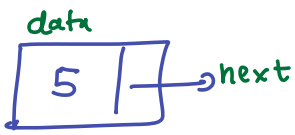
head.next.next.next ___.

---

② Hierarchical DS {Non-linear} DS

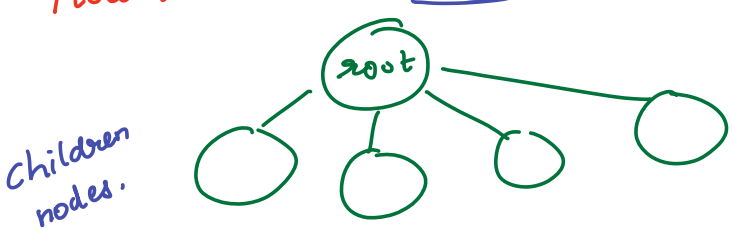
- Family Trees
- Inheritance
- Organisations
- File system in OS

No notion of x^{th} elements.



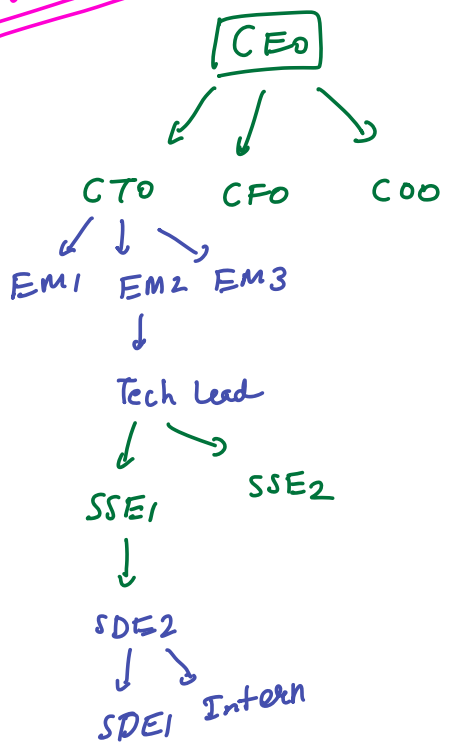
---

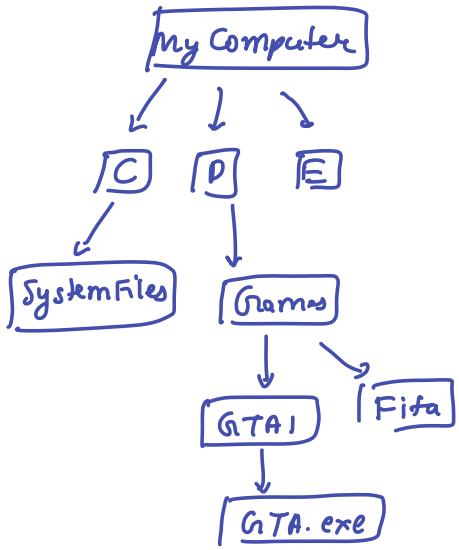
(1) How to describe nodes coming out of root?



Use Cases

Relationships





HTML DOM Tree

Nesting of Tags.

Tree is subset of Graph.

Social Network : Graph

My SQL — Indexing → internally implementing.

B+ Trees

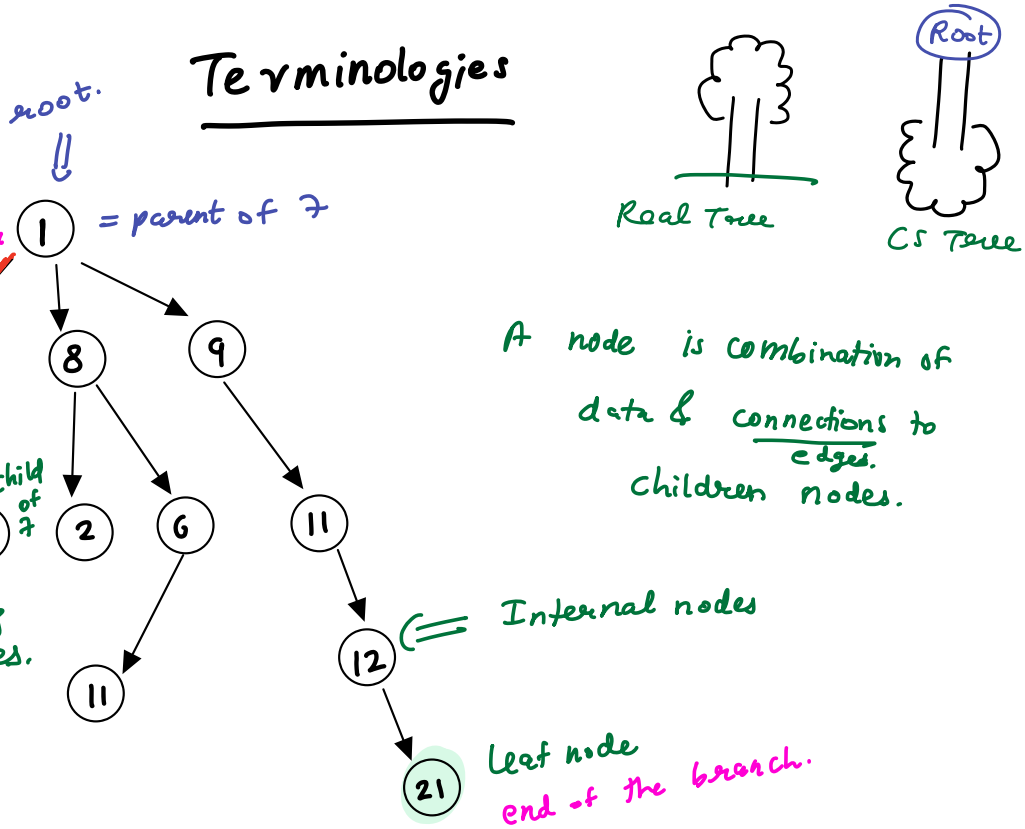
Trie

type of Tree.

DSML : Decision Trees

Recursion Tree to analyze TC & SC.

Terminologies



Real Tree

CS Tree

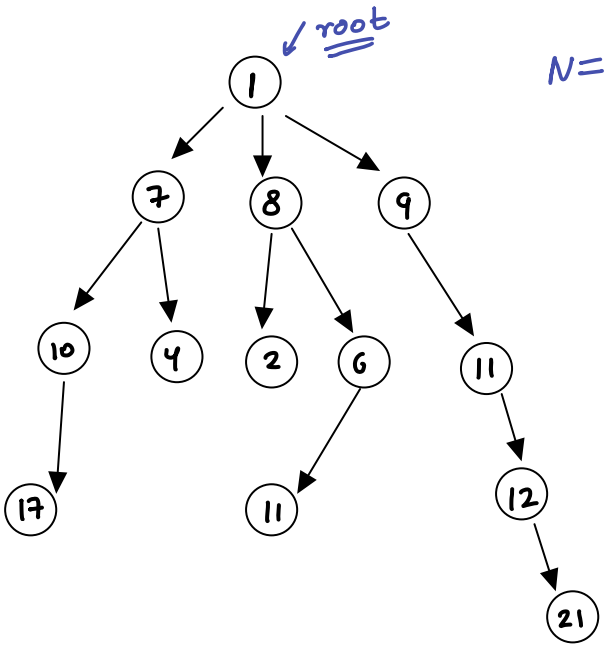
A node is combination of data & connections to edges.
Children nodes.

Root + leaf

→ 1 node can have only 1 parent node

→ No cycles can be in a tree.

Real Truths About Trees

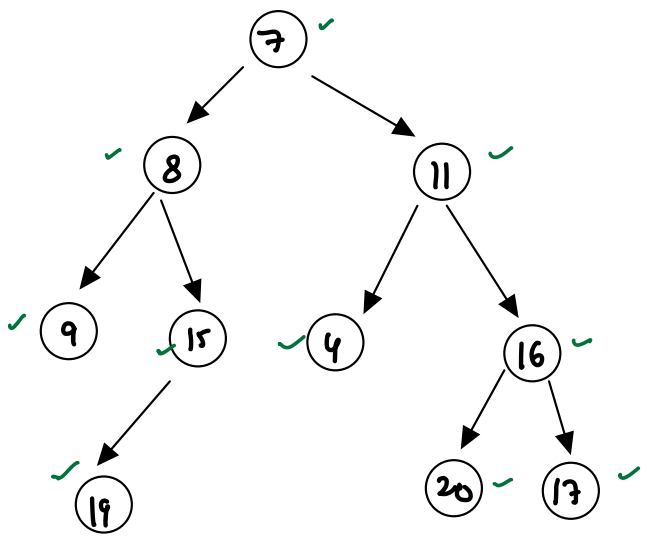


N = Size of Tree

N = Count of nodes = 13

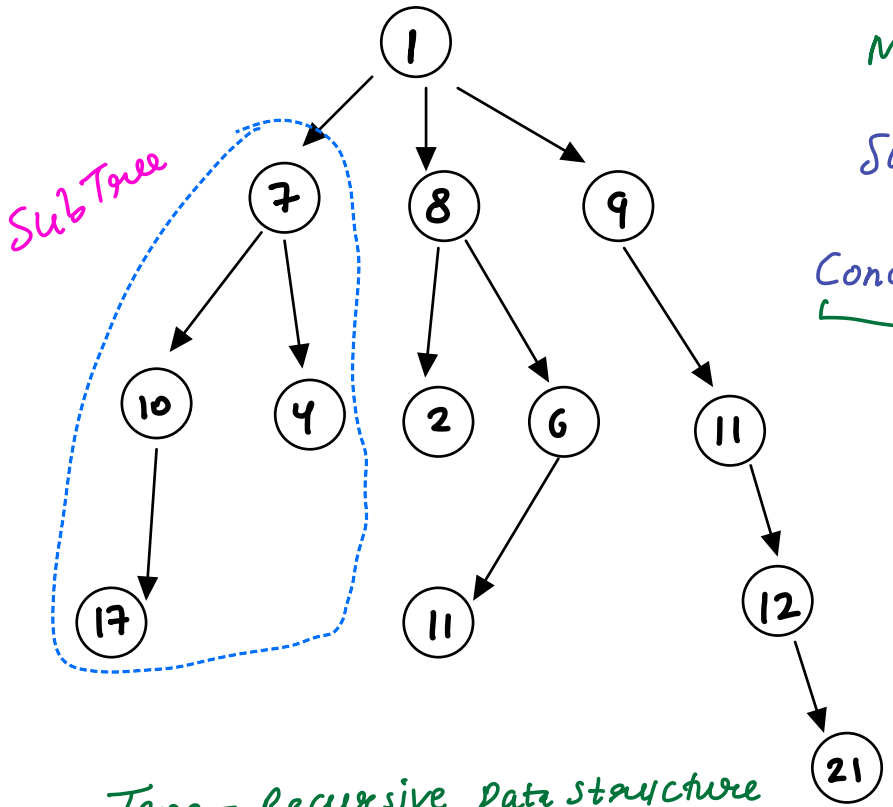
Except root, all other nodes have exactly 1 parent node.

Edges/Link = N − 1
= 12.



N = 10

edges = N − 1 = 9



Multiple families.

Subtrees

Concept of recursion

Tree = Recursive Data structure

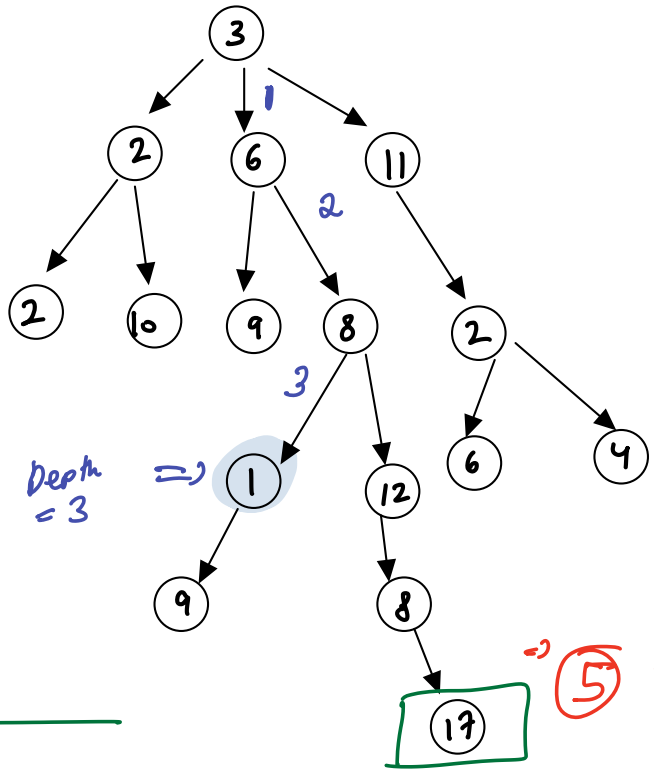
Describing Trees – Properties

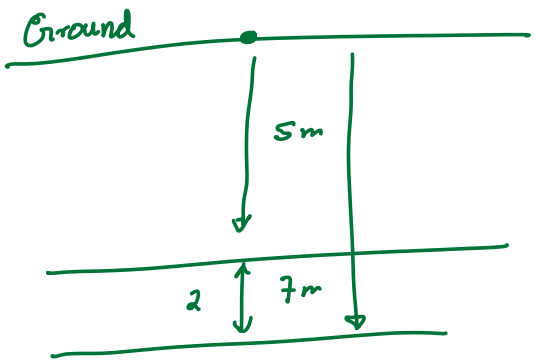
Height of Tree

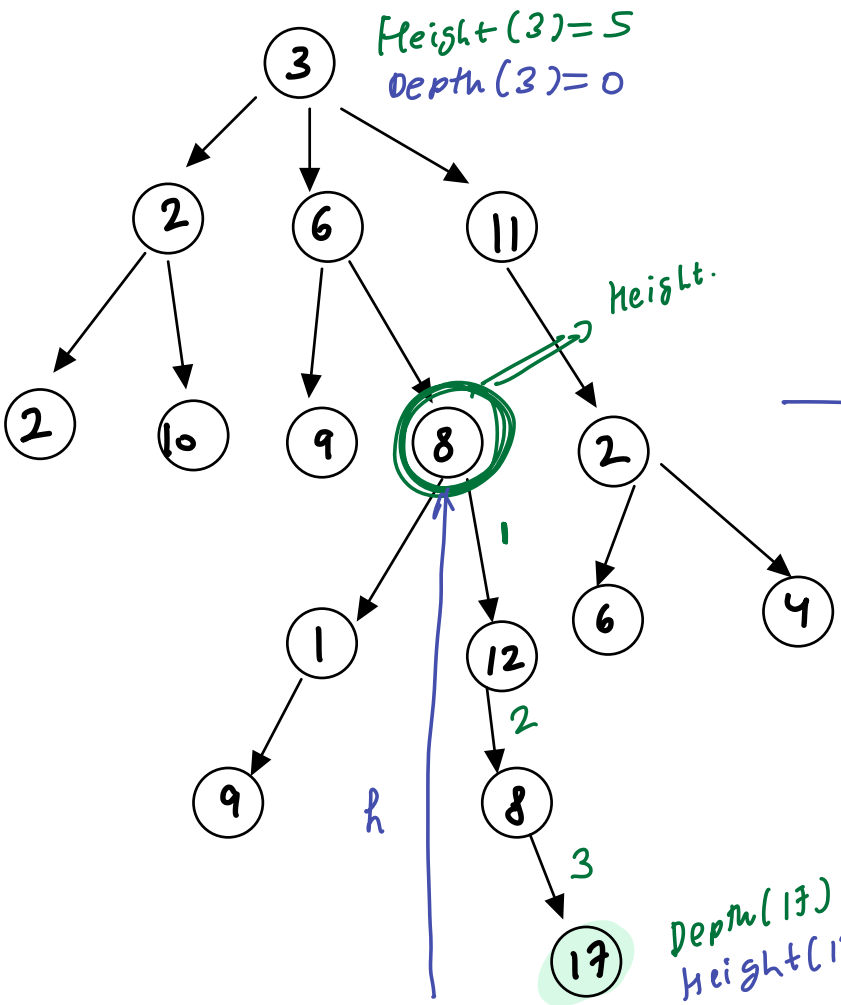
i) Depth of node

= Distance of node from root

= No. of edges/links from root to given node







Height

Height of node
= distance from node to the farthest leaf node

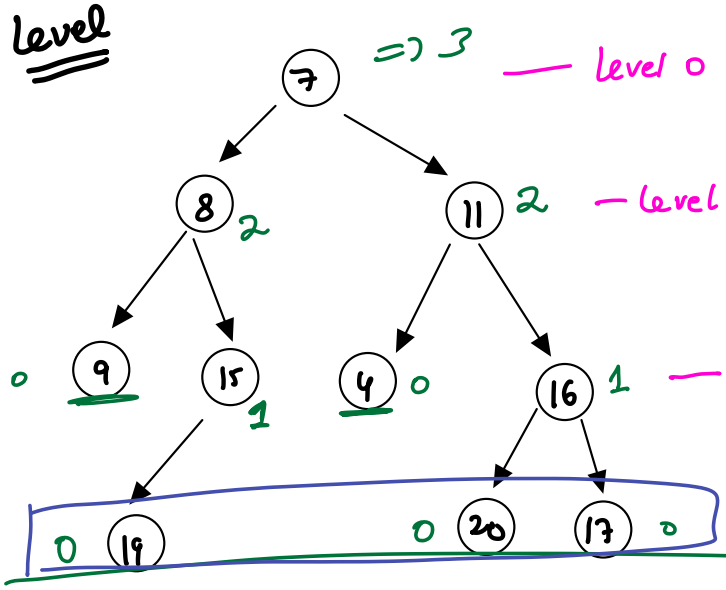
length of longest path from node to a leaf

☆ Height of Tree = Height of root

Depth of Tree = Depth of farthest leaf from root

= Height of Tree

Level



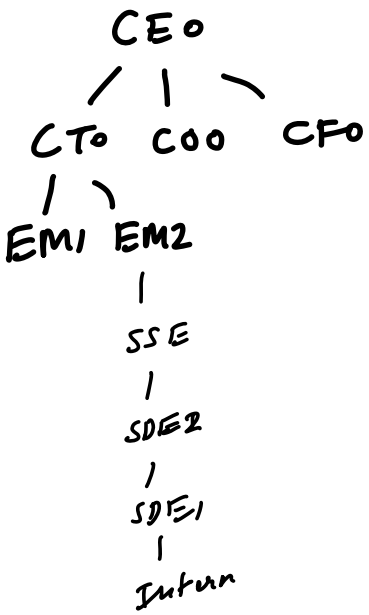
=> 3 — Level 0 Height of leaf = 0

Height of root, 7 => 3

— Level 1

— Level 2 = Depth (Distance from root)

— Level 3 = max. level = Depth of Tree.



Small scale Company.

CEO is actually responsible for all other people directly

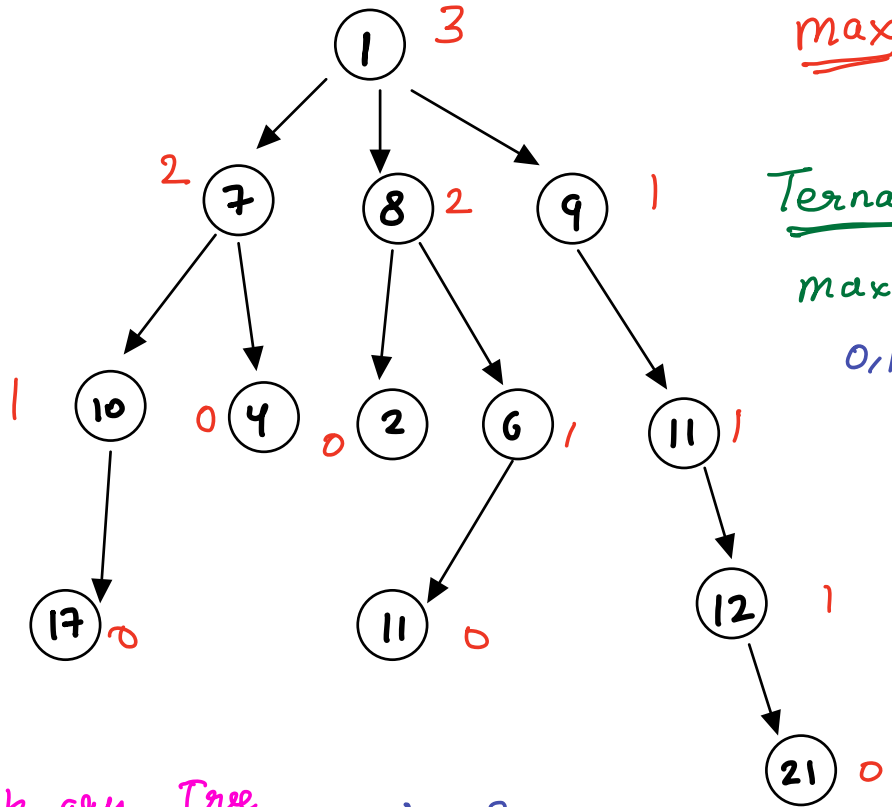
=> Binary Tree

2

Restrict no. of children for every node

Max (atmost, not more than) 2

=> 0, 1, or 2.



max = 3

Ternary Tree

max 3

0, 1, 2 or 3.

k-ary Tree

max k children

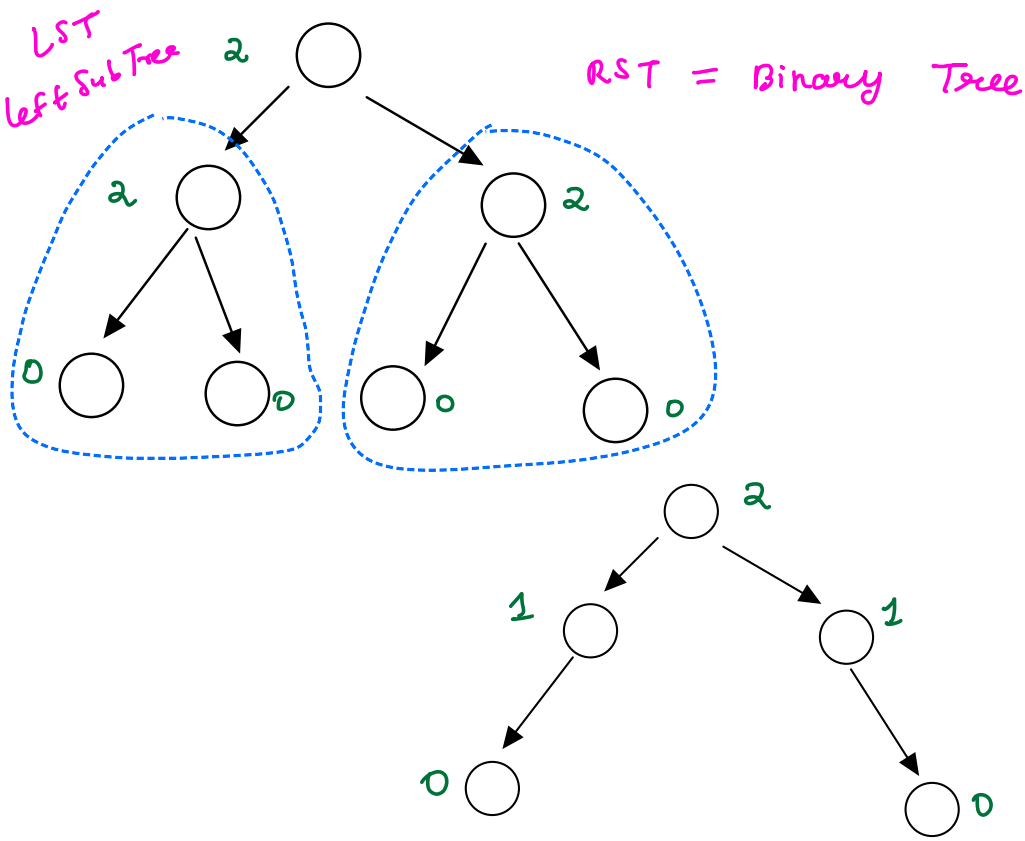
k = 3

k = 4

k = 5

k = 26

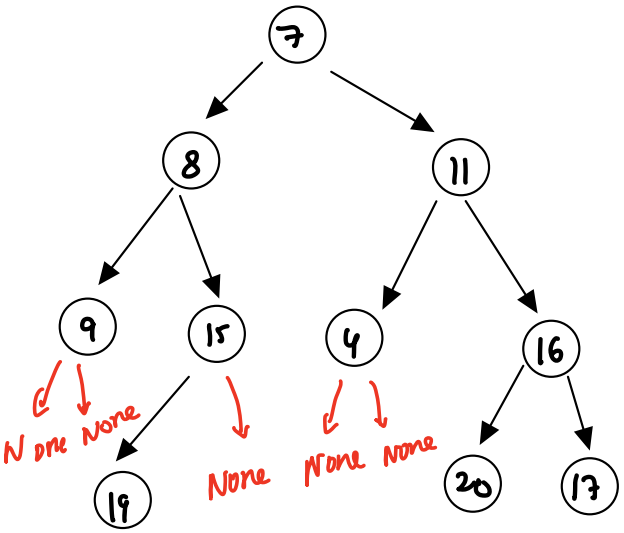
k = 100



Next Class - Binary Search Trees.

⇒ Implementation (Binary Tree)

max 2 children. (references)



max 1 connection

∴ Singly LL.

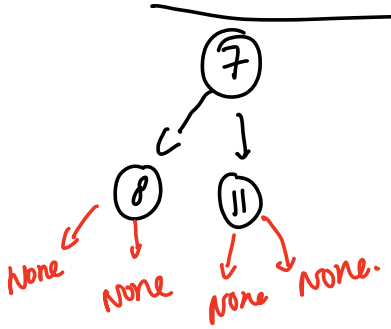
```
Class TreeNode:

    def __init__(self, data):
        self.data = data
        self.left = None    # reference
        self.right = None   # reference.
```

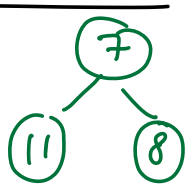
```
Class BinaryTree:

    def __init__(
        self)

    self.root
        = None
```

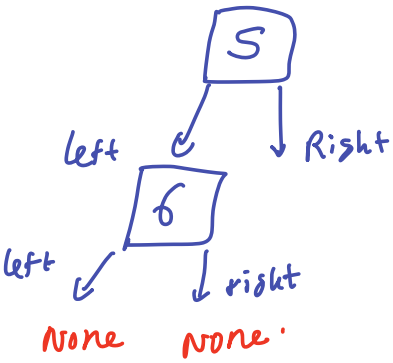


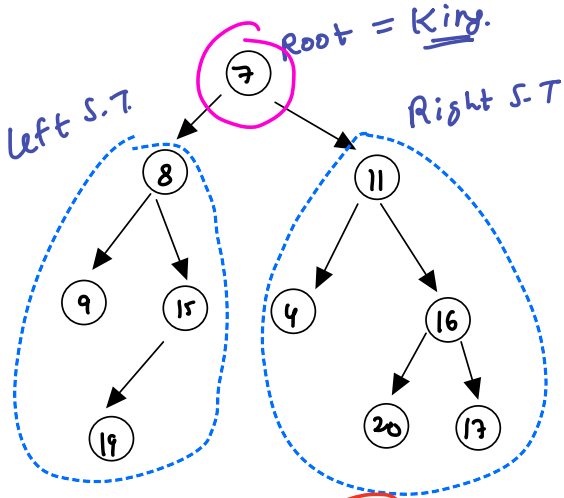

```
root = TreeNode(7)
root.left = TreeNode(8)
root.right = TreeNode(11)
```



① Recursively

Traversals





Visit

- Print the node's data

$3! = 6.$

$3 * 2 * 1 = 6.$

King.

Root.data	Root.val	Left	Right	left	Right
Left	Right	Root.val	Root.val	Right	left
Right	Left	Right	Left	Root.val	Root.val
✓	✗	✓	✗	✓	✗

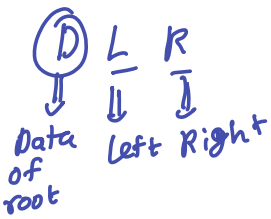
left should be visited before right.

PreOrder

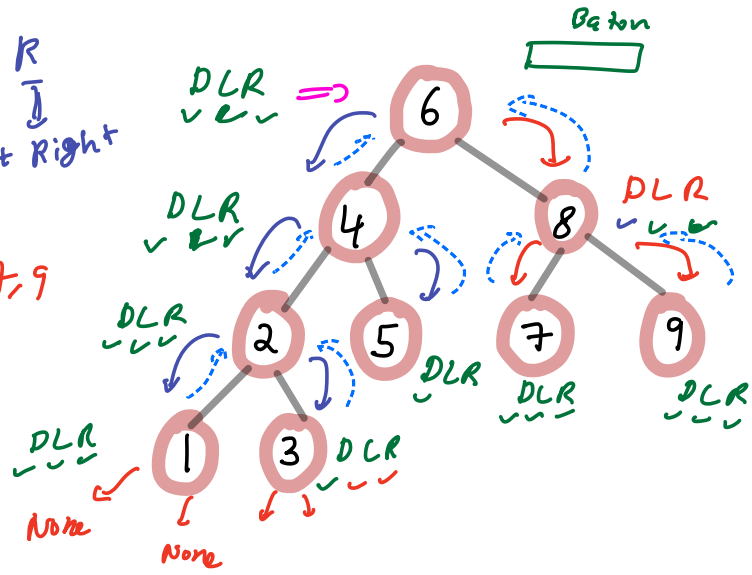
Inorder

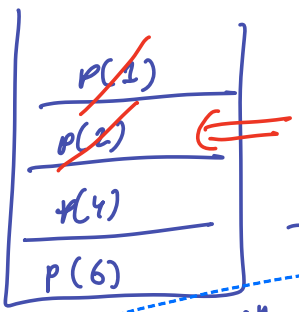
post Order.

Preorder



6, 4, 2, 1, 3, 5, 8, 7, 9





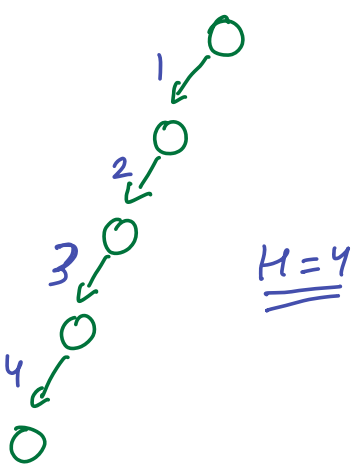
Assumption = Faith.
Main logic = Logic
& Base condition = BC.

FLBC

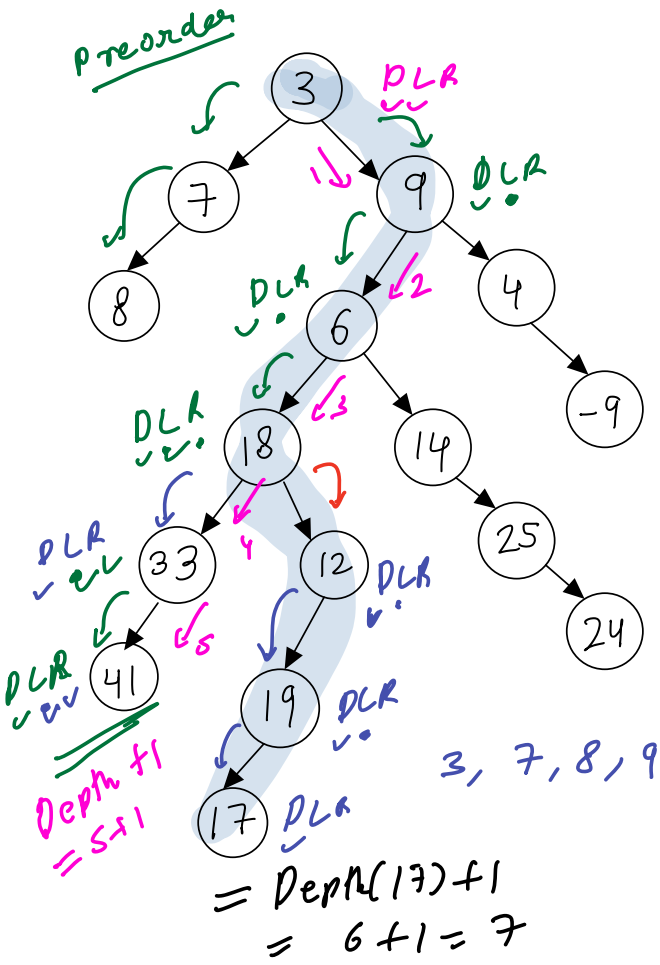
```
def preorder(root):
    if root == None:
        return
    print(root.data)
    preorder(root.left)
    preorder(root.right)
```

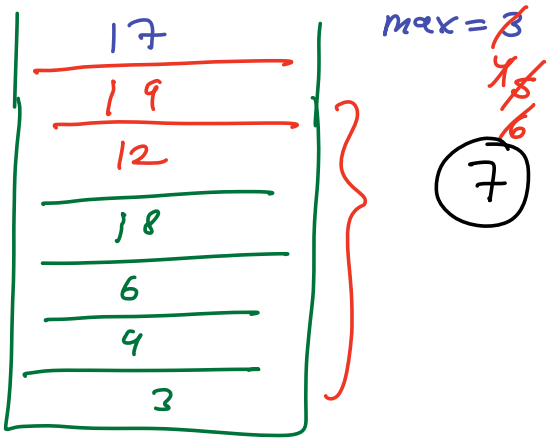
TC: O(N)

SC: O(H+1) = O(H) = Worst case O(N).



H = 4





Space Complexity =

Max elements in stack

= O(Height)

3, 7, 8, 9, 6, 18, 33, 41, 12, 19, 17

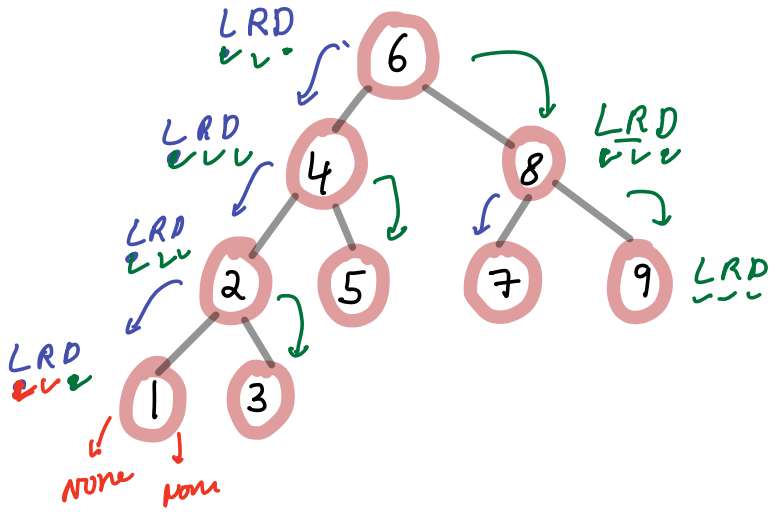
= Depth(17) + 1

= 6 + 1 = 7

Postorder

left, Right, Root

L R D




```
def postorder(root):
    if root == None:
        return
    postorder(root.left)
    postorder(root.right)
    print(root.data)
```

1, 3, 2, 5, 4, 7, 9, 8, 6

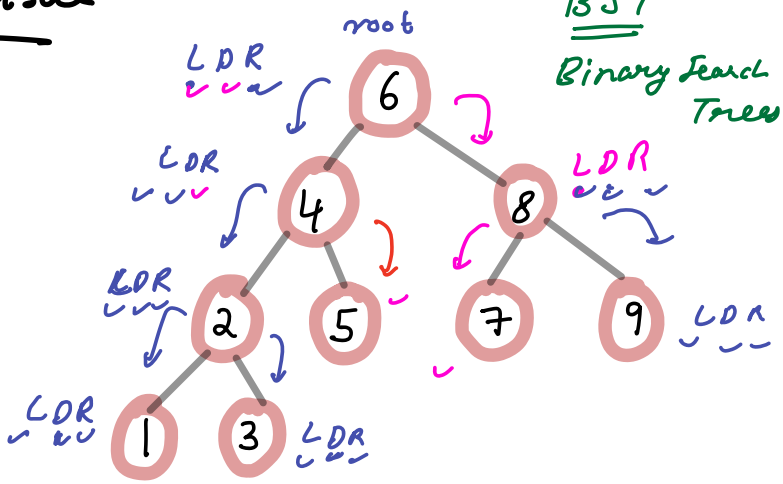
TC: O(N)

SC: O(H)

⇒ Inorder Traversal

L D R

Left, Data, Right



BST

Binary Search Trees.

1, 2, 3, 4, 5, 6, 7, 8, 9

```
def inorder(root):
    if root == None:
        return
    inorder(root.left)
    print(root.data)
    inorder(root.right)
```

How do we traverse the tree if no support for recursion?

Stack

Advanced Batch

→ Iterative Traversals

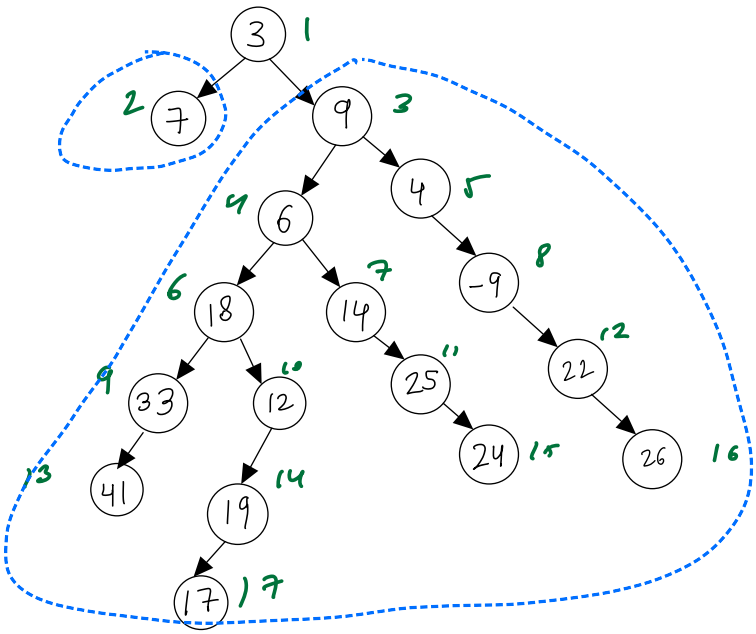
Assignment →

You can't use recursion

But you CAN!

Q1) **Size** of Binary Tree

nodes.

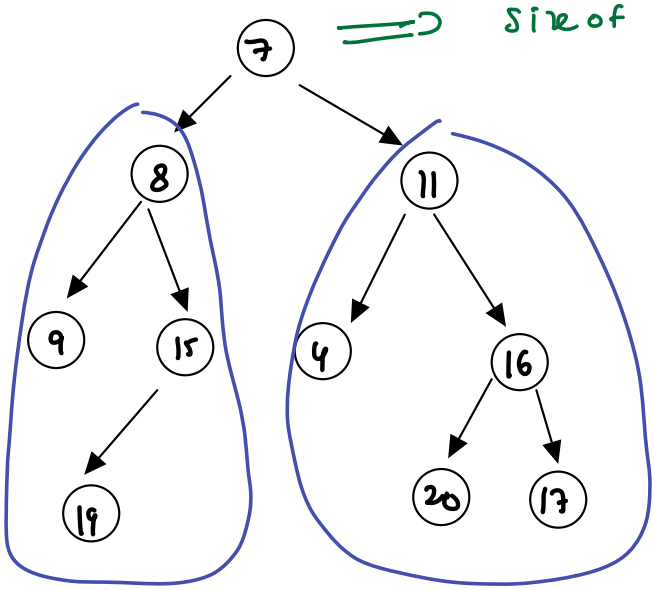


$N = 17$

- max
- min
- sum

① Global Counter

② Return size



⟹ Size of Tree at node 7

$= LST + RST + 1$

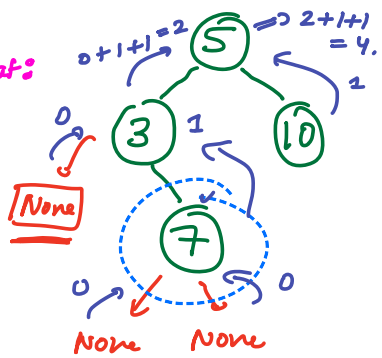
$= 4 + 5 + 1$

$= 10$

LST = 4

RST = 5

```
def size(root):
    if root == None:        # if root is a leaf: return 1
        return 0
    lst = size(root.left)
    rst = size(root.right)
    return lst+rst+1
```



TC: O(N)

SC: O(H) ⟹ O(N) in worst case.

Q2) Height Try it yourself For sure, we will cover in next class.

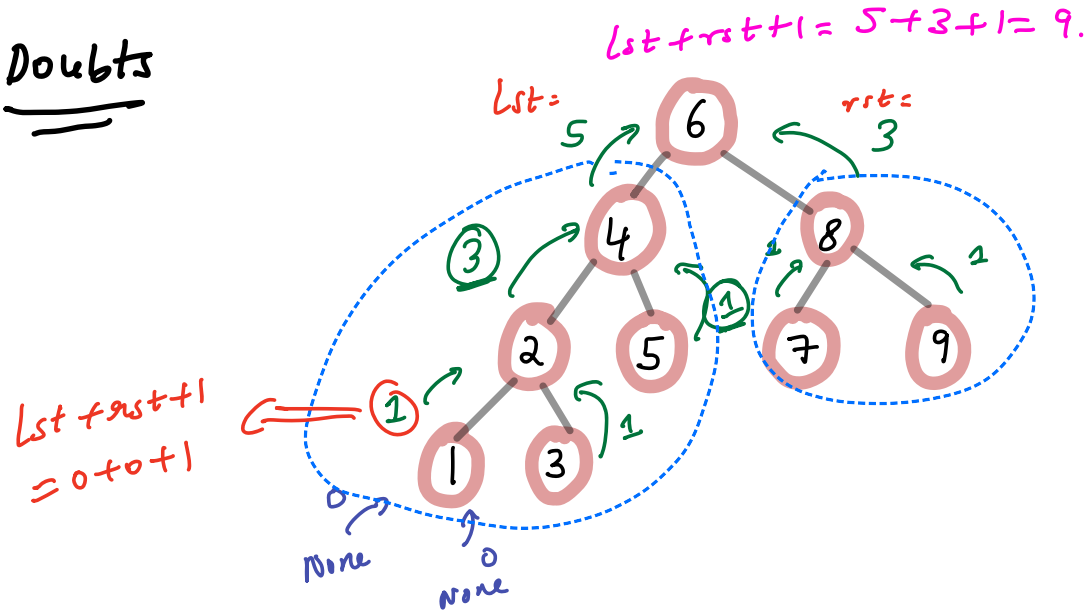
Next Class :

Interview Problems (3-4)

+

Binary Search Trees - Basic

Doubts



Dictionary ⟸ { Balanced BST }