Your Ultimate Guide To Landing Top AI roles

DECODE AiML

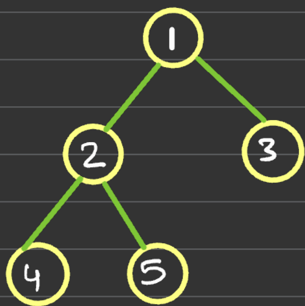
2.14.1

Binary Tree

→ A binary Tree is a tree such that

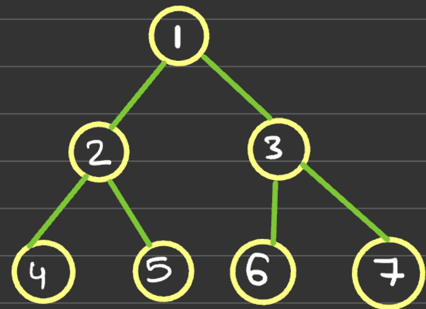
① Every node has atmost 2 children.

② These children are referred to as the left child and right child.



Full BT

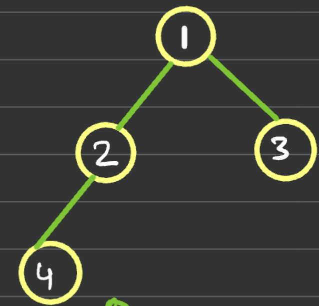
Complete BT



Full BT

Complete BT

Perfect BT



Complete BT

Left skewed BT

Right Skewed BT

Structural Classification

→ **Full Binary Tree**: Each node has 0 or 2 children

→ **Complete Binary Tree**: All levels are Completely filled except the last level. In last level, nodes are filled from left to right.

→ **Perfect Binary Tree**: All internal nodes have 2 childrens. and all leaves are at the same level.

→ **Skewed Binary Tree**: Each Parent node has only 1 child. Looks like Linked List

Property based classification

- BST
- AVL
- Heap
- Red-black

Binary Tree Implementation in Python

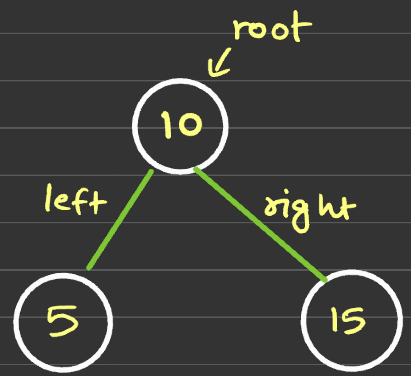
```
Class TreeNode:
    def __init__(self,value):
        Self.value = value
        self.left = None
        Self.right = None
```

Value	
Left R	right R

Node structure

Example Usecase

```
root = TreeNode(10)
root.left = TreeNode(5)
root.right = TreeNode(15)
```

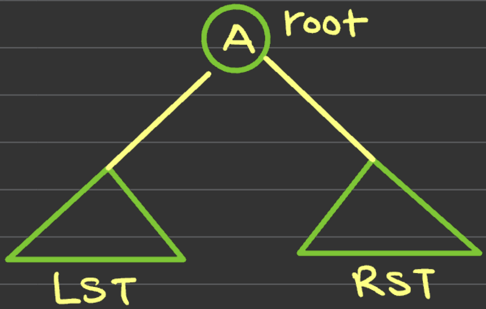


2.14.2

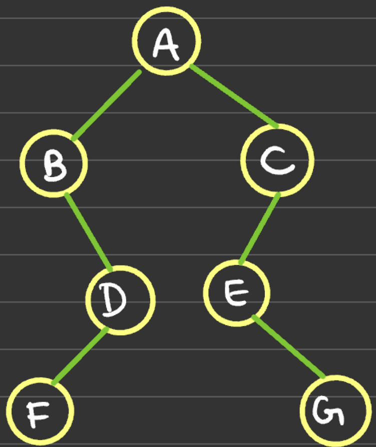
Traversal in a Binary Tree

→ Traversal means visiting each and every node of the tree

→ There are mainly 3 ways of Tree traversal.



1. Preorder : Root LST RST
2. Inorder : LST Root RST
3. Postorder : LST RST Root



Preorder T : ABDFCEG
Inorder T : BFDAEGC
Postorder T : FDBGECA

Logic building for Traversal Code (Inorder)

```
def traversal(node):
    if node is None:
        return
    traversal(node.left)
    print(node.value, end=" ")
    traversal(node.right)
```

← base case

← Recursive Case

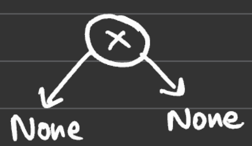
→ base case or Trivial case

node is None

→ Recursive case

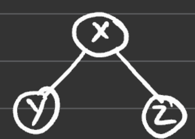
↳ build solution bottom up.

↳ depth=0



print (x)

depth=1



print(y)
print(x)
print(z)

depth=2 root

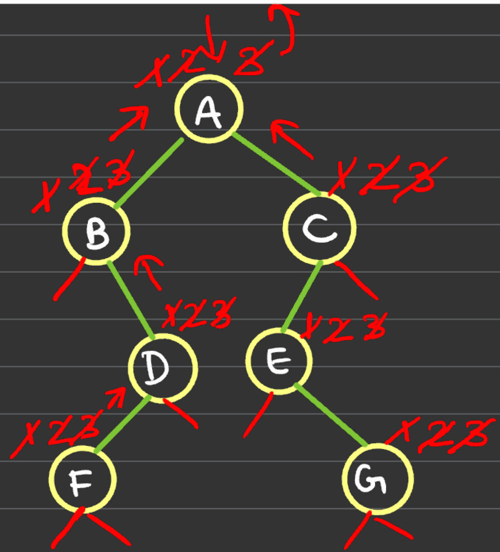
ay x bz

traversal (root.left)
print (x)
traversal (root.right)

Dry Run: Inorder Code

→ Inorder Traversal

```
def inorder(node):
    if node:
        1 inorder(node.left)
        2 print(node.value, end=" ")
        3 inorder(node.right)
```



Inorder output:

BFDAEGC

Traversal Code in Python ← Recursive

Value	
LeftR	rightR

Node structure

→ Preorder Traversal

```
def preorder(node):
    if node:
        print(node.value, end=" ")
        preorder(node.left)
        preorder(node.right)
```

→ Inorder Traversal

```
def inorder(node):
    if node:
        inorder(node.left)
        print(node.value, end=" ")
        inorder(node.right)
```

→ Postorder Traversal

```
def postorder(node):
    if node:
        postorder(node.left)
        postorder(node.right)
        print(node.value, end=" ")
```

Time and Space Complexity of Traversal

```
def traversal(node):          ← T(n)
    if node is None:
        return
    traversal(node.left)      ← T(LST)
    print(node.value, end=" ")  ← O(1)
    traversal(node.right)     ← T(RST)
```

Time Complexity = O(n)
Space Complexity = O(n)

→ Recurrence Relation

$$T(n) = T(LST) + T(RST) + O(1)$$
$$= O(1), \quad n = 0$$

↳ Since LST + RST = n-1 nodes

$$\boxed{T(n) = O(n)}$$

→ Space complexity = max depth of Recursion stack

$$\boxed{\text{Space} = O(n)}$$

Intuition: Every node is visited 3 times and constant work during each visit

2.14.3

Properties of Binary Tree

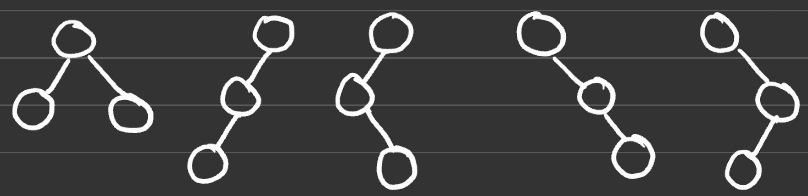
1. Number of binary Tree structure possible for n nodes

↑ unlabeled

$$\boxed{\# BTS = \frac{{}^{2n}C_n}{n+1}}$$

→ $n = 1$  → 1

→ $n = 2$  → 2

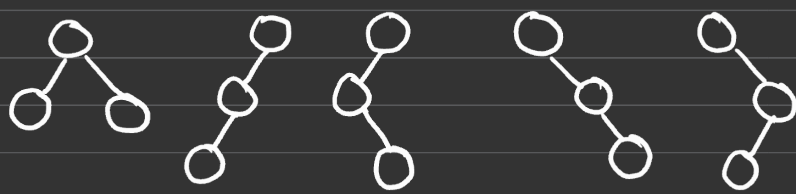
→ $n = 3$  → 5

② Number of binary Tree possible with n labeled node.

$$\# \text{BTS} = \frac{{}^{2n}C_n \times \lfloor\underline{n}}{n+1}$$

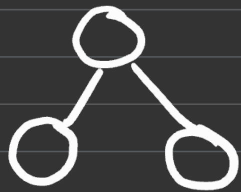
→ $n=1$  → 1

→ $n=2$  → 2

→ $n=3$  → 5

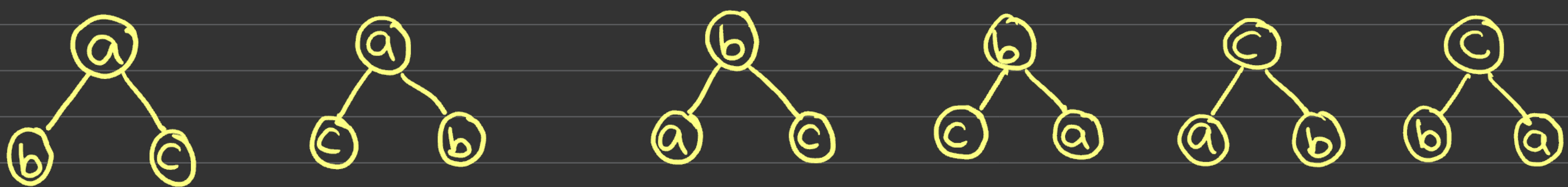
→ Every structure is capable of generating $\lfloor\underline{n}$ labeled Trees

→ Let's check no of labelled binary tree for a given binary Tree structure.



← Structure

(a, b, c) ← node value



→ Total # Labeled BT = 6

③ No of binary tree with a given inorder, preorder and postorder Traversal is 1.

④ Given a Preorder and Inorder, we can Construct a unique Labeled Binary Tree

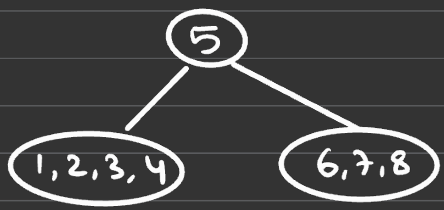
⑤ Given a Postorder and Inorder, we can Construct a unique Labeled Binary Tree

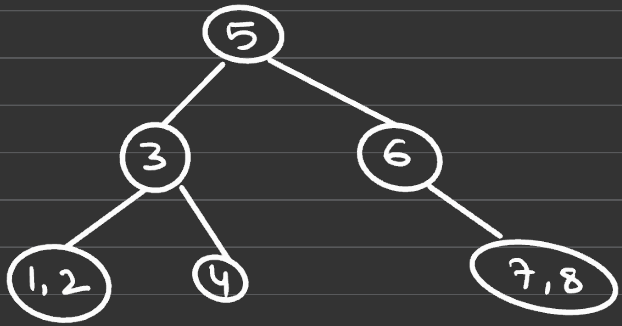
⑥ We may not get a unique tree with given inorder and Preorder.

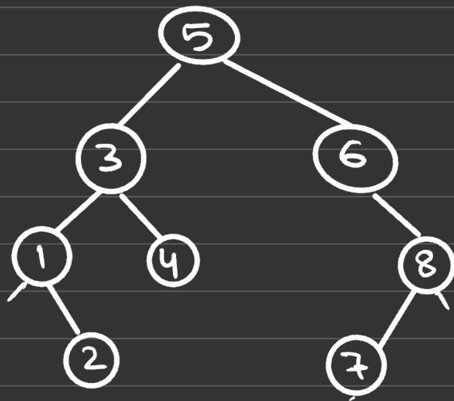
Ex: Construct a Binary Tree with given Inorder and Preorder.

In : 1, 2, 3, 4, 5, 6, 7, 8

Pre : 5, 3, 1, 2, 4, 6, 8, 7



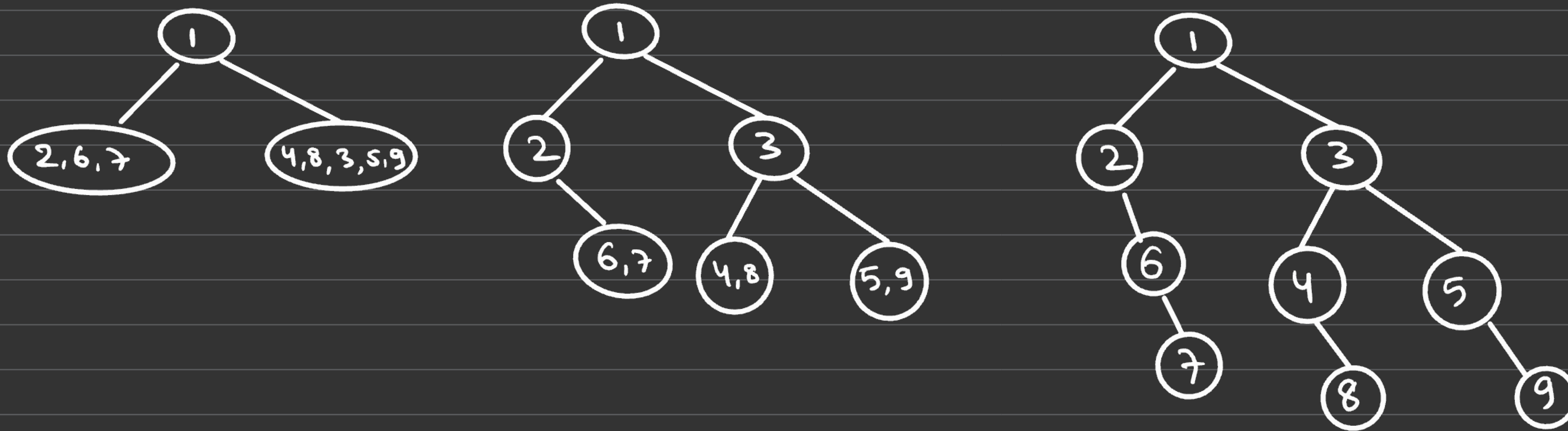




Ex:- Construct a Binary Tree with given Inorder and Postorder

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

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



→ Next Lecture

↳ Problems on Binary Tree

Like Subscribe