16. Binary trees

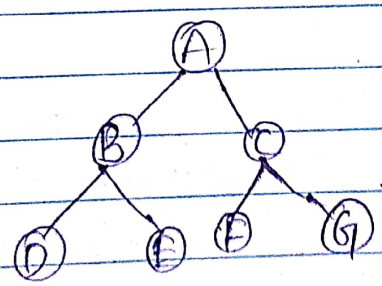
A tree can be defined as a non-linear data structre that stores data in the form of nodes and nodes are connected to each other with the help of edges. There are different ways of traversing a tree depending upon the order in which the trees nodes are visited and the types of data structres used for traversing the tree. There are various data structres involved in traversing a tree as traversing a tree involves iterating over all nodes in some manner. Types of traversals are inorder, preorder, & post order traversal.

- inorder traversal (left, root, right)
- preorder traversal (root, left, right)
- postorder traversal (left, right, root)

Example:-

Inorder traversal:-

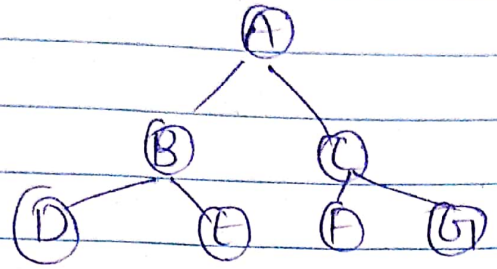
If a binary tree is traversed in order the output will produce sorted key values in ascending order.



D → B → E → A → F → C → G

Preorder traversal:-

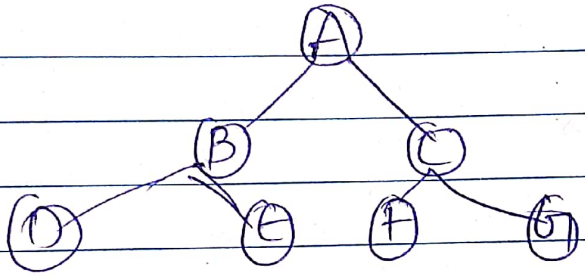
In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.



A → B → D → E → C → F → G

post order traversal-

The root node is visited last, hence the same first we traverse the left subtree, then the right subtree & finally the root.



D → E → B → F → G → C → A

Code:-

Inorder:-

```
class treenode:
    def __init__(self,val):
        self.val=val
        self.left=None
        self.right=None
def inordertraversal(root):
    answer=[]
    inordertraversaluntil(root,answer)
    return answer
def inordertraversaluntil(root,answer):
```

```
        if root is None:
            return
        inordertraversaluntil(root.left, answer)
        answer.append(root.value)
        inordertraversal until(root.right, answer)
    root = Treenode(1)
    root.left = Treenode(2)
    root.right = Treenode(3)
    root.left.left = Treenode(4)
    root.left.right = Treenode(5)
    print(inorder traversal(root))
```

Pre-order:

```
def preordertraversal(root):
    answer = []
    preordertraversaluntil(root, answer)
    return answer
def preordertraversaluntil(root, answer):
    if root is None:
        return
    answer.append(root.val)
    preordertraversaluntil(root.left, answer)
    preordertraversal until(root.right, answer)
    return
root = Treenode(1)
root.left = Treenode(2)
root.right = Treenode(3)
root.left.left = Treenode(4)
root.left.right = Treenode(5)
print(preordertraversal(root))
```

Post Order:-

```
def postordertraversal(root):
    answer=[]
    postordertraversal_until(root,answer)
    return answer
def postordertraversal(root,answer):
    if root is None:
        return
    postordertraversaluntil(root.left,answa)
    postordertraversal until(root.right,answer)
    answer.append(root.val)
    return
root=Treenode(1)
root.left=Treenode(2)
root.right=Tree node(3)
root.left.left=Tree node(4)
root.left.right=Treenode(5)
print(postordertraversal(root))
```

Output-

Inorder - 4 2 5 1 3
preorder - 1 2 4 5 3
postorder - 4 5 2 3 1