TREE TRAVERSALS

- Traversing means visiting each node only once.
- Tree traversal is a method for visiting all the nodes in the tree exactly once.

There are three types of tree traversal techniques

a) Inorder Traversal

b) Preorder Traversal

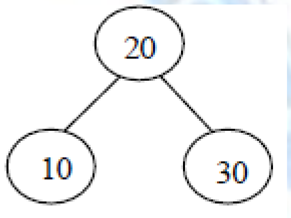
c) Postorder Traversal

a) Inorder Traversal

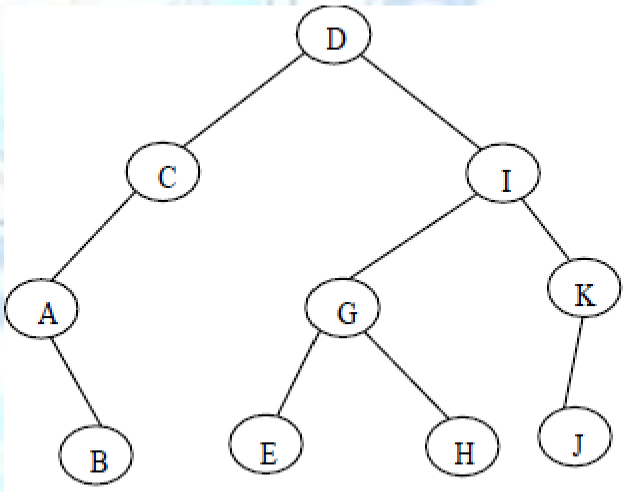
The inorder traversal of a binary tree is performed as

- Traverse the left subtree in inorder
- Visit the root
- Traverse the right subtree in inorder.

Example :



Inorder 10, 20, 30



A B C D E G H I J K

Recursive routine for inorder traversal

def inorder_traversal_recursive

(root): result = []

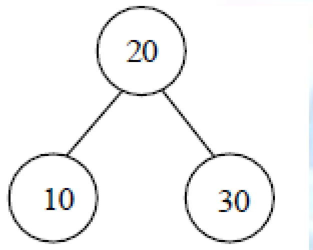
```
    def traverse (node):
    if node is None:
      return
  # 1. Traverse the left subtree
   traverse (node.left)
  # 2. Visit the current node
   result.append (node.val)
  # 3. Traverse the right subtree
   traverse
   (node.right)
   traverse (root)
return result
```

b) Preorder Traversal

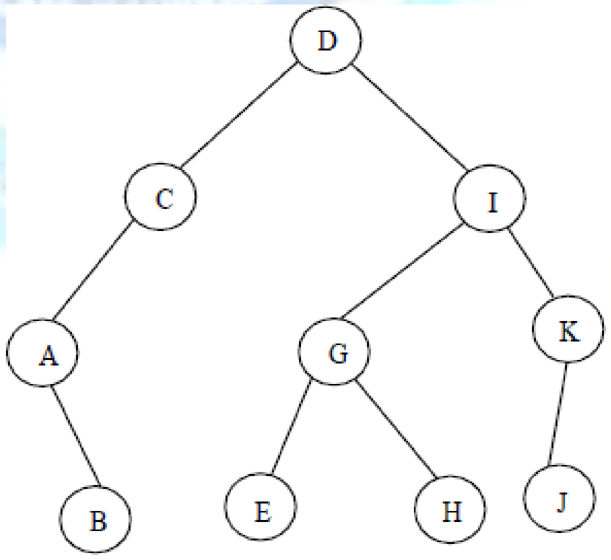
The preorder traversal of a binary tree is performed as follows,

- Visit the root
- Traverse the left subtree in preorder
- Traverse the right subtree in preorder.

Example:



Preorder : 20, 10, 30



Preorder D C A B I G E H K J

Recursive Routine For Preorder Traversal

```
def preorder_traversal_recursive(root,
        result=None): if result is None:
        result = []
    if root:
        # 1. Visit the Root Node
        result.append(root.value)  # Or
        print(root.value) # 2. Traverse the Left
        Subtree
        preorder_traversal_recursive(root.left,
        result) # 3. Traverse the Right Subtree
        preorder_traversal_recursive(root.right,
        result)
    return result
```

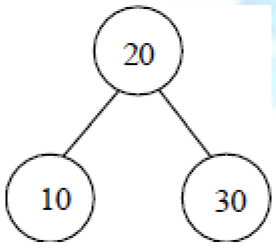
c)Postorder Traversal

The postorder traversal of a binary tree is performed by the following steps.

- Traverse the left subtree in postorder.
- Traverse the right subtree in postorder.
- Visit

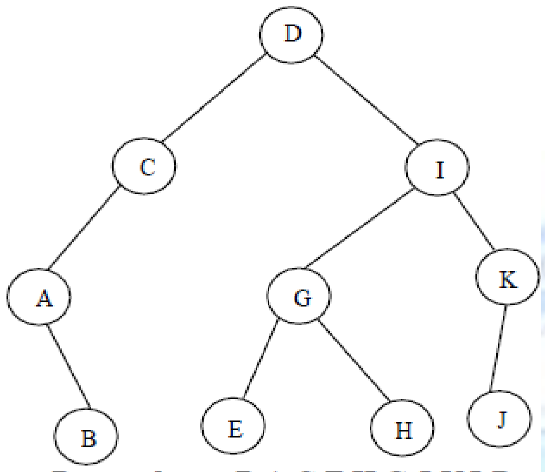
the root.

Example : 1



Postorder : - 10, 30, 20

Example : 2



Post order : - B A C E H G J K I D

Recursive Routine For Postorder Traversal

```
def postorder_traversal(node, result):
    # Base case: if the node is None, return (end of a
    branch) if node is None:
        return
    # Recursively traverse the left subtree
    postorder_traversal(node.left, result)
    # Recursively traverse the right subtree
    postorder_traversal(node.right, result)
    # Process the current node (add its data to the result list)
    result.append(node.data)
```