Traversals of Tree

Traversing means visiting each node at once. Tree traversal is a method for visiting all the nodes in thetree exactly once.

There are three types of tree traversal techniques, namely

1. Inorder Traversal or symmetric order
2. Preorder Traversal or depth-first order
3. Postorder Traversal

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.

* **Recursive Routine for Inorder Traversal**

```
void Inorder (Tree T)
{
if (T!=NULL)
{
Inorder (T->left);
printf("%d",T->data);
Inorder (T->right);
}
}
```

We start from **A**, and following in-order traversal, we move to its left subtree **B**. **B** is also traversed in-order. The process goes on until all the nodes are visited. The output of inorder traversal of this tree will be −

$$D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$$

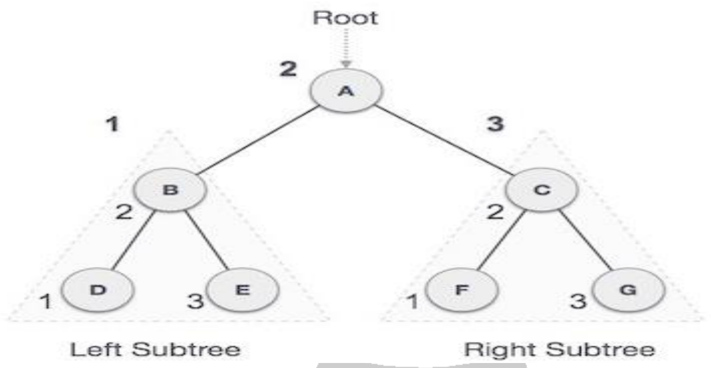
Algorithm

```
Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Visit root node.
Step 3 − Recursively traverse right subtree.
```

In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

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



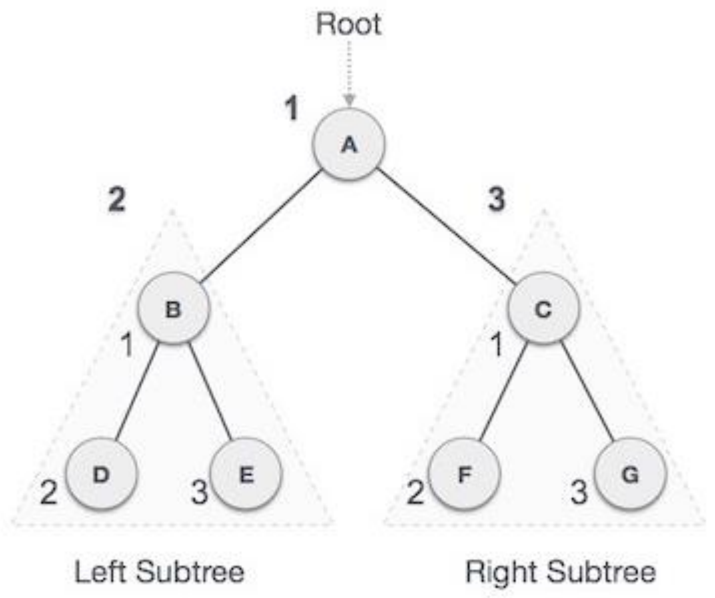
Preorder Traversal

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.
*

Pre-order Traversal

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



We start from **A**, and following pre-order traversal, we first visit **A** itself and then move to its left subtree **B**. **B** is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be −

$$A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$$

*

Recursive Routine for Preorder Traversal

```
void Preorder (Tree T)
{
if (T ! = NULL)
{
printf("%d",T->data);
Preorder (T ->left);
Preorder (T ->right);
}
```

Postorder Traversal

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

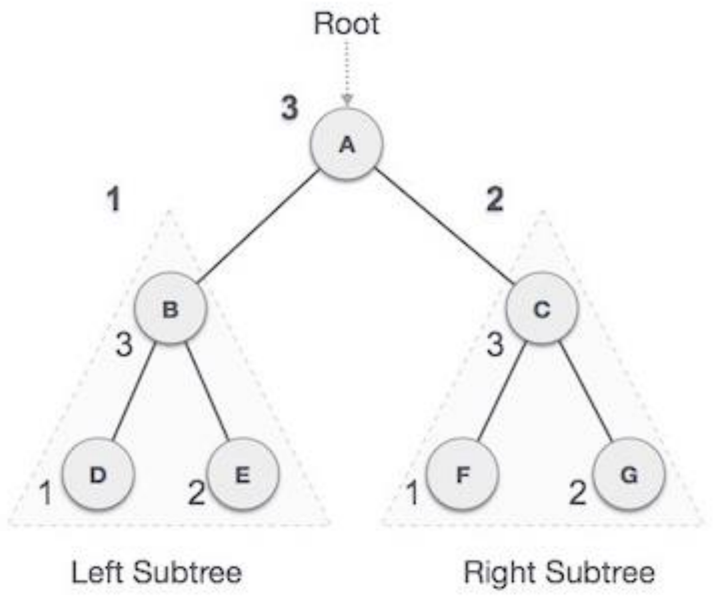
* Traverse the left subtree in postorder.
* Traverse the right subtree in postorder.
* Visit the root.

Recursive Routine for Postorder Traversal

```
void Postorder (Tree T)
{
if (T ! = NULL)
{
Postorder (T -
>Left); Postorder
(T ->Right);
printf("%d",T-
>data);
}
```

Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.



We start from **A**, and following Post-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be −

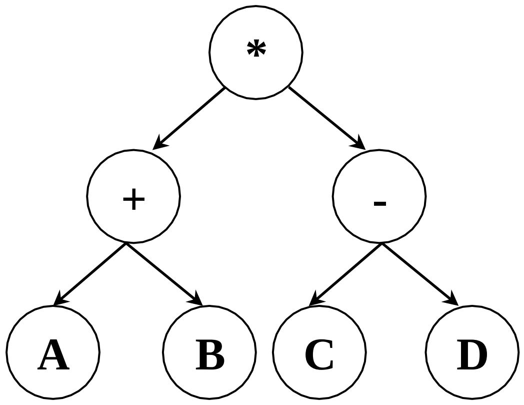
$$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$$

EXPRESSION TREE

An expression tree is tree in which left nodes have the operands and interior node have the operators. Like binary tree, expression tree can also be traversed by inorder, preorder and postorder traversal.

Example:

(A+B)*(C-D)



Constructing an Expression Tree

Let us consider postfix expression given as an input for constructing an expression tree by performing thefollowing steps Read one symbol at a time from the postfix expression and then scan the expression from left to right manner.

1. Check whether the symbol is an operand or operator.
    a. If the symbol is an operand, create a one - node tree and push a pointer on to the stack.
    b. If the symbol is an operator pop two pointers from the stack namely T_1 and T_2 and form a new tree with root as the operator and T_2 as a left child and T_1 as a right child. A pointer to this new tree is then pushed onto the stack.
2. Repeated the above steps until reach the end of the expression.
3. The final pointer in the stack is complete expression tree.

Example:

ABC*+DE*F+G*+