Sri SAI RAM ENGINEERING COLLEGE

INSTITUTE OF TECHNOLOGY

West Tambaram, Chennai - 44

YEAR 2

SEM 3

CS8391

DATA STRUCTURES

UNIT No. 3

3.1.2 TREE TRAVERSAL

Version: 1.XX

3.1.2 TREE TRAVERSAL

Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way. Tree is a nonlinear data structure in which the elements can be traversed in many different ways. There are different algorithms for tree traversals, they are,

- Pre-order Traversal
- Post-order Traversal
- In-order Traversal

PRE-ORDER TRAVERSAL

To traverse a non-empty binary tree in pre-order, the following operations are performed recursively at each node. The algorithm works by:

1. Visiting the root node,
2. Traversing the left sub-tree, and finally
3. Traversing the right sub-tree.

Pre-order traversal is also called depth-first traversal. In this algorithm, the left sub-tree is always traversed before the right sub-tree. The word 'pre' in the pre-order specifies that the root node is accessed prior to any other nodes in the left and right sub-trees.

Pre-order algorithm is also known as the NLR traversal algorithm (Node-Left-Right). Pre-order traversal algorithms are used to extract a prefix notation from an expression tree. When we traverse the elements of a tree using the pre-order traversal algorithm, the expression that we get is a prefix expression.

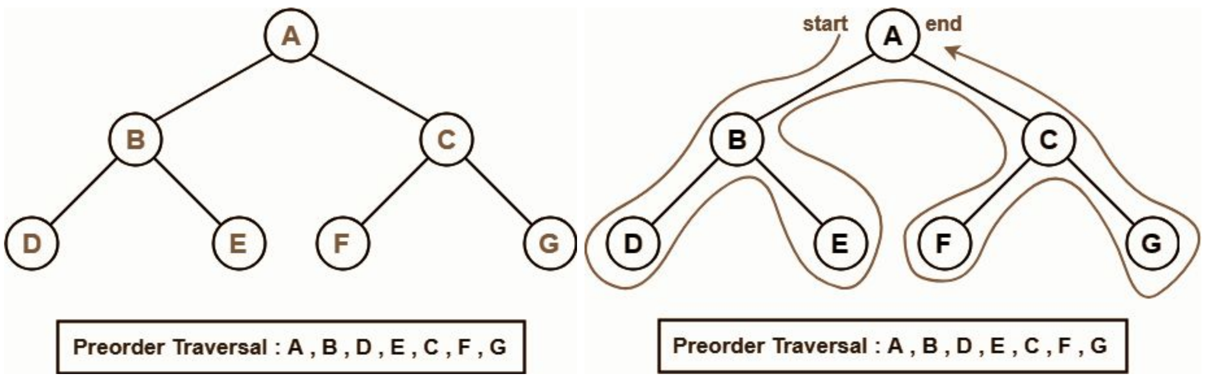
ALGORITHM FOR PRE-ORDER TRAVERSAL

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2: Write TREE → DATA
Step 3: PREORDER(TREE → LEFT)
Step 4: PREORDER(TREE → RIGHT)
            [END OF LOOP]
Step 5: END
```

Procedure:

```
void preorder_traversal(Node* root) {
    if (root == NULL)
        return;
    printf("%d -> ", root->value);
    preorder_traversal(root->left);
    preorder_traversal(root->right);
}
```

Example:


A
B
C
D
E
F
G
Preorder Traversal : A , B , D , E , C , F , G
start
end
A
B
C
D
E
F
G
Preorder Traversal : A , B , D , E , C , F , G

IN-ORDER TRAVERSAL

To traverse a non-empty binary tree in in-order, the following operations are performed recursively at each node. The algorithm works by:

1. Traversing the left sub-tree,
2. Visiting the root node, and finally
3. Traversing the right sub-tree.

In-order traversal is also called symmetric traversal. In this algorithm, the left sub-tree is always traversed before the root node and the right sub-tree. The word ‘in’ in the in-order specifies that the root node is accessed in between the left and the right sub-trees. In-order algorithm is also known as the LNR traversal algorithm (Left-Node-Right).

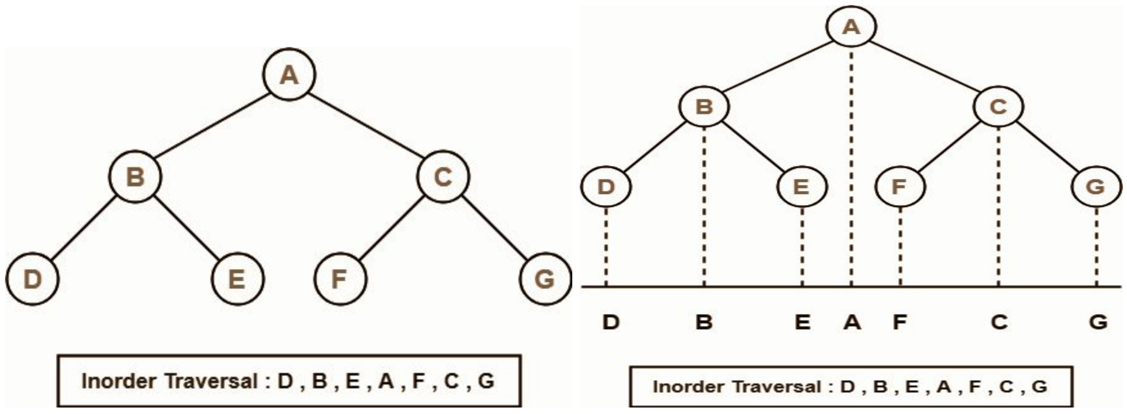
ALGORITHM

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2: INORDER(TREE → LEFT)
Step 3: Write TREE → DATA
Step 4: INORDER(TREE → RIGHT)
            [END OF LOOP]
Step 5: END
```

Procedure:

```
void inorder_traversal(Node* root) {
   if (root == NULL)
      return;
   inorder_traversal(root->left);
   printf("%d\n", root->value);
   inorder_traversal(root->right);
}
```

Example:


A
B
C
D
E
F
G
Inorder Traversal : D , B , E , A , F , C , G
D B E A F C G
Inorder Traversal : D , B , E , A , F , C , G

POST-ORDER TRAVERSAL

To traverse a non-empty binary tree in post-order, the following operations are performed recursively at each node. The algorithm works by:

1. Traversing the left sub-tree,
2. Traversing the right sub-tree, and finally
3. Visiting the root node.

The left sub-tree is always traversed before the right sub-tree and the root node. The word 'post' in the post-order specifies that the root node is accessed after the left and the right sub-trees. Post-order algorithm is also known as the LRN traversal algorithm (Left-Right-Node). Post-order traversals are used to extract postfix notation from an expression tree.

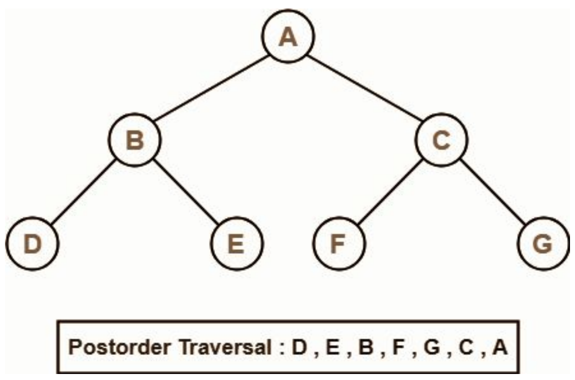
ALGORITHM:

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2: POSTORDER(TREE → LEFT)
Step 3: POSTORDER(TREE → RIGHT)
Step 4: Write TREE → DATA
            [END OF LOOP]
Step 5: END
```

Procedure:

```
void postorder_traversal(Node* root) {
    if (root == NULL)
        return;
    postorder_traversal(root->left);
    postorder_traversal(root->right);
    printf("%d\n", root->value);
}
```

Example:



Constructing a Binary Tree from Traversal Results

A binary tree can be constructed if we are given at least two traversal results. The first traversal must be the in-order traversal and the second can be either pre-order or post-order traversal.The in-order traversal result will be used to determine the left and the right child nodes, and the pre-order/post-order can be used to determine the root node.

Steps:

Step 1 Use the pre-order sequence to determine the root node of the tree. The first element would be the root node.

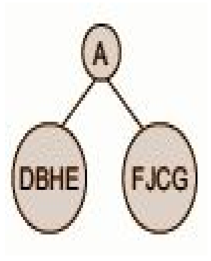
Step 2 Elements on the left side of the root node in the in-order traversal sequence forms the left sub-tree of the root node. Similarly, elements on the right side of the root node in the in-order traversal sequence form the right sub-tree of the root node.

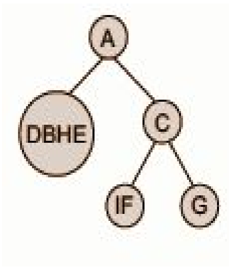
Step 3 Recursively select each element from pre-order traversal sequence and create its left and right sub-trees from the in-order traversal sequence.

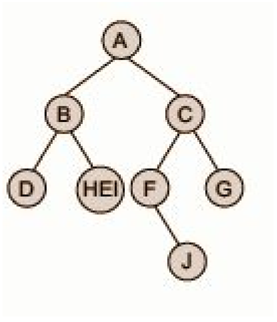
Example:

Given In–order Traversal: D B H E I A F J C G and Post order Traversal: D H I E B J F G C A of a Binary Tree.

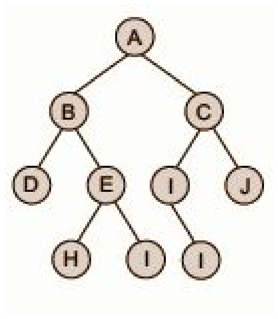
Step 1 **Step 2** **Step 3** **Step 4**





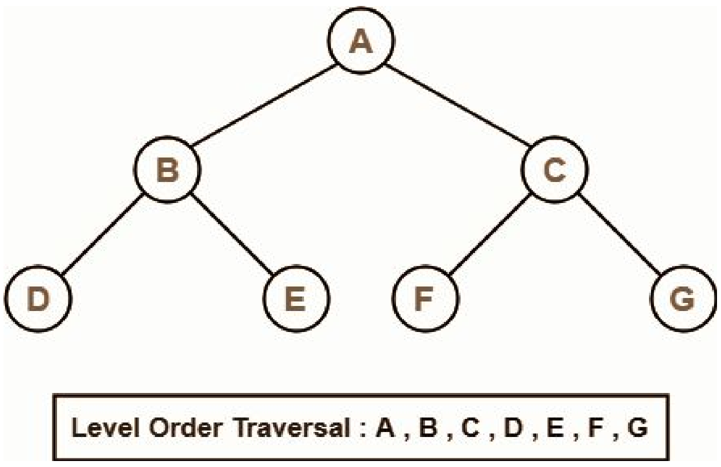


Step 5



LEVEL-ORDER TRAVERSAL

In level-order traversal, all the nodes at a level are accessed before going to the next level. This algorithm is also called as the breadth-first traversal algorithm.



EXPRESSION TREE

Expression Tree is used to represent expressions. There are different types of expression formats:

- Prefix expression
- Infix expression and
- Postfix expression

Expression Tree is a special kind of binary tree with the following properties:

- Each leaf is an operand.
- The root and internal nodes are operators.

- Subtrees are subexpressions with the root being an operator.

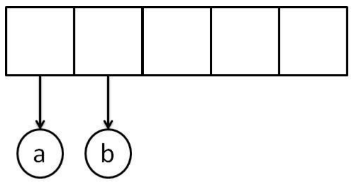
CONSTRUCTION OF EXPRESSION TREE

Consider that a postfix expression is given as an input for constructing an expression tree. Following are the step to construct an expression tree:

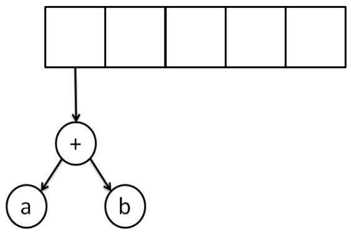
1. Read one symbol at a time from the postfix expression.
2. Check if the symbol is an operand or operator.
3. If the symbol is an operand, create a one node tree and pushed a pointer onto a stack
4. If the symbol is an operator, pop two pointer from the stack namely T1 & T2 and form a new tree with root as the operator, T1 & T2 as a left and right child
5. A pointer to this new tree is pushed onto the stack

Example

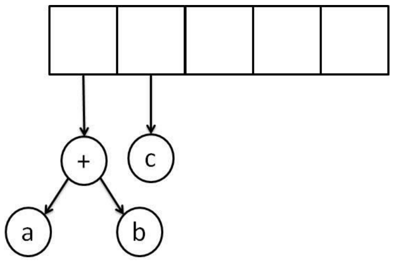
The input is: a b + c * The first two symbols are operands, create one-node tree and push a pointer to them onto the stack.



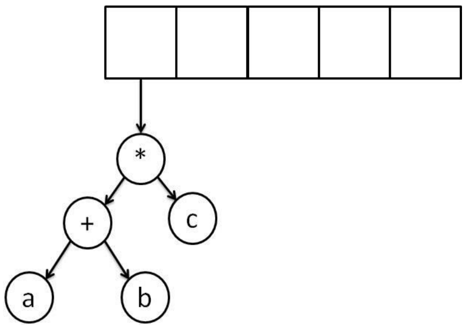
Next, read a'+' symbol, so two pointers to tree are popped,a new tree is formed and push a pointer to it onto the stack.



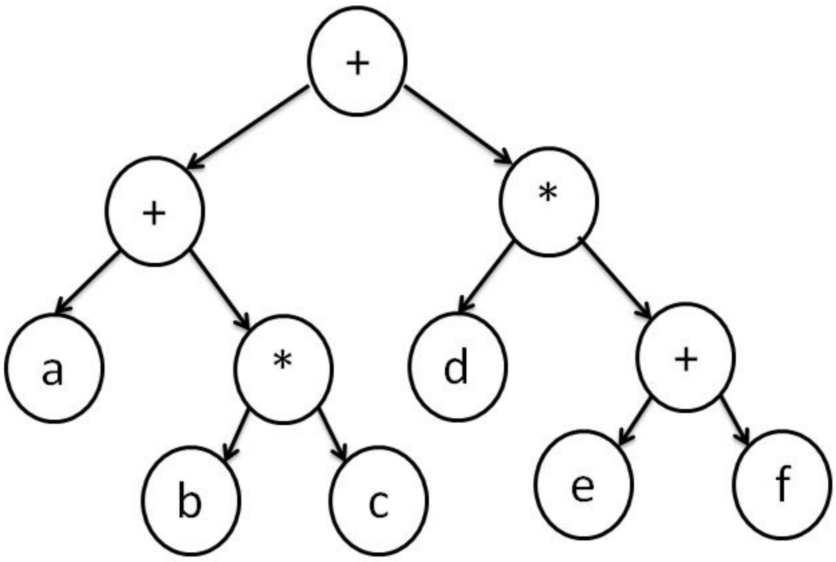
Next, 'c' is read, we create one node tree and push a pointer to it onto the stack.



Finally, the last symbol is read ' * ', we pop two tree pointers and form a new tree with a, ' * ' as root, and a pointer to the final tree remains on the stack.



EXAMPLE



- Infix expression: (a+(b*c))+(d*(e + f))
- Postfix Expression: a b c * + d e f + * +
- Prefix Expression: + + a * b c * d + e f