PES UNIVERSITY

Department of Computer Science and Engineering

PES UNIVERSITY

UE19CS202: Data Structures and its Applications (4-0-0-4-4)

Trees
Binary Trees: Traversal

Dr. Shylaja S S
Ms. Kusuma K V

Binary Tree Traversal:

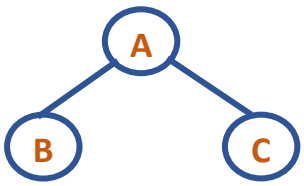
Traversal is the process of moving through all the nodes in a binary tree and visiting each one in turn. The action taken when each node is visited depends on the application; here we shall print the content of each node on visiting.

Tree being a non linear data structure, there are many different ways in which we could traverse all the nodes. When we write an algorithm to traverse a binary tree, we shall almost always wish to proceed so that the same rules are applied at each node, and we thereby adhere to a general pattern.

At a given node, then, there are three tasks we shall wish to do in some order. They are:
1) Visiting a node, let us denote it by V
2) Traversing the left subtree, let us denote it by L
3) Traversing the right subtree, let us denote it by R
The key distinction in traversal orders is to decide if we are to visit the node itself before traversing either subtree, between the subtrees, or after traversing both the subtrees.
The three tasks can be done in six different ways: VLR, LVR, LRV, VRL, RVL, RLV.

By Standard convention, these 6 ways are reduced to three by permitting only the ways in which the left subtree is traversed before the right. The three mirror images are clearly similar. The three ways with left subtree before right subtree are given the following names:
VLR – Preorder, LVR – Inorder, LRV – Postorder
These three names are chosen according to the step at which the given node is visited. With prorder traversal, the node is visited before the subtrees, with inorder traversal, node is visited between the left and right subtree, and with postorder traversal, the node is visited after both the subtrees.

Eg1: Consider a simple example of a Binary Tree shown below:



Preorder Traversal: A B C
Inorder Traversal: B A C
Postorder Traversal: B C A

Binary Tree Recursive Traversal

Preorder Traversal:

- Root Node is visited before the subtrees
- Left subtree is traversed in preorder
- Right subtree is traversed in preorder

```
void preorder(NODE* root)
{
        if(root!=NULL)
        {
                printf("%d ",root->info);
                preorder(root->left);
                preorder(root->right);
        }
}
```

Inorder Traversal:

- Left subtree is traversed in Inorder
- Root Node is visited
- Right subtree is traversed in Inorder

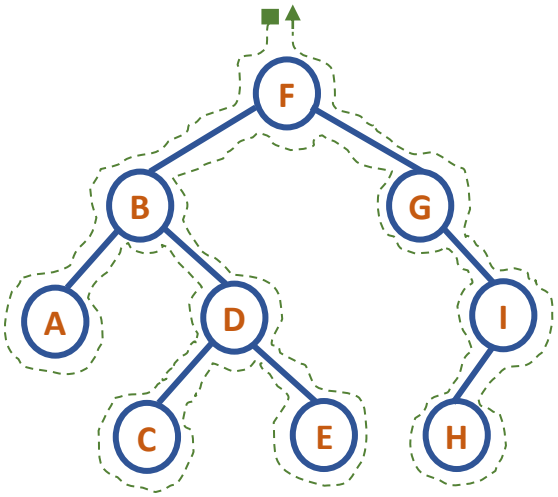
```
void inorder(NODE* root)
{
        if(root!=NULL)
        {
                inorder(root->left);
                printf("%d ",root->info);
                inorder(root->right);
        }
}
```

Postorder Traversal:

- Left subtree is traversed in postorder
- Right subtree is traversed in postorder
- Root Node is visited

```
void postorder(NODE* root)
{
        if(root!=NULL)
        {
                postorder(root->left);
                postorder(root->right);
                printf("%d ",root->info);
        }
}
```

Eg2: Write the Preorder, Inorder and Postorder Traversal for the Binary Tree given below:



Preorder Traversal: F B A D C E G I H
Inorder Traversal: A B C D E F G H I
Postorder Traversal: A C E D B H I G F

Binary Tree Iterative Traversal

1) Inorder Traversal

```
//Iterative Inorder Traversal
iterativeInorder(root)
 s = emptyStack
current = root
do {
                while(current != null)
                {       /* Travel down left branches as far as possible saving pointers to
nodes passed in the stack*/
                        push(s, current)
                        current = current->left
                } //At this point, the left subtree is empty
                poppedNode = pop(s)
                print poppedNode ->info                 //visit the node
                current = poppedNode ->right            //traverse right subtree
} while(!isEmpty(s) or current != null)
```

2) Preorder Traversal

```
//Iterative Preorder Traversal
iterativePreorder(root)
current=root
if (current == null)
   return
s = emptyStack
push(s, current)
```

```
while(!isEmpty(s)) {
        current = pop(s)
        print current->info
        //right child is pushed first so that left is processed first
        if(current->right !=NULL)
                push(s, current->right)
        if(current->left !=NULL)
                push(s, current->left)
}
```

3) Postorder Traversal

```
//Iterative Postorder Traversal
iterativePostorder(root)
s1 = emptyStack
s2 = emptyStack
push(s1, root)

while(!isEmpty(s1)) {
        current = pop(s1)
        push(s2,current)
        if(current->left !=NULL)
                push(s1, current->left)
        if(current->right !=NULL)
                push(s1, current->right)
}
while(!isEmpty(s2)) {           //Print all the elements of stack2
        current = pop(s2)
        print current->info
}
```