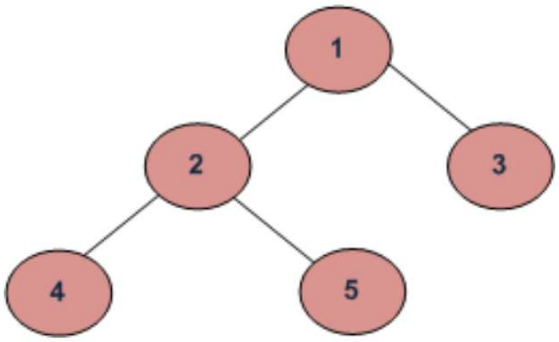
TREE TRAVERSALS

Traversal is a process to visit all the nodes of a tree and may print their values too. Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



Depth First Traversals:
(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1
Breadth-First or Level Order Traversal: 1 2 3 4 5

Depth First Traversal:

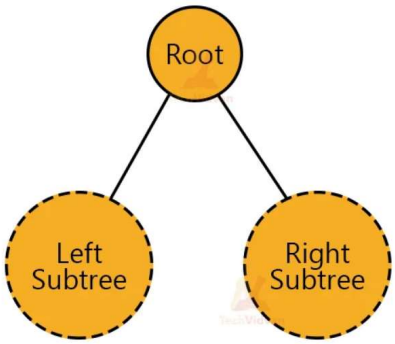
In depth-first traversal, we go in one direction up to the bottom first and then come back and go to the other direction. There are three types of depth-first traversals. These are:

1. Preorder traversal

2. Inorder traversal

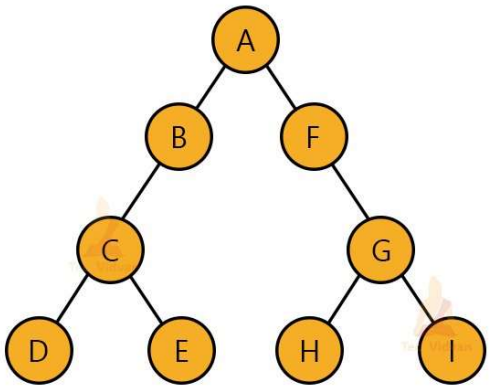
3. Postorder traversal

A binary tree is a recursive data structure. Also, each binary tree has 3 parts: a root node, a left subtree and a right subtree. These left and right subtrees are also trees in themselves and thus, again further described recursively.



Preorder Traversal:

In a preorder traversal, we process/visit the root node first. Then we traverse the left subtree in a preorder manner. Finally, we visit the right subtree again in a preorder manner. For example, consider the following tree:



Here, the root node is A. All the nodes on the left of A are a part of the left subtree whereas all the nodes on the right of A are a part of the right subtree. Thus, according to preorder traversal, we will first visit the root node, so A will print first and then move to the left subtree.

B is the root node for the left subtree. So B will print next and we will visit the left and right nodes of B. In this manner, we will traverse the whole left subtree

and then move to the right subtree. Thus, the order of visiting the nodes will be: A→B→C→D→E→F→G→H→I.

Algorithm for Preorder Traversal:

for all nodes of the tree:

Step 1: Visit the root node.

Step 2: Traverse left subtree recursively.

Step 3: Traverse right subtree recursively.

Pseudo-code for Preorder Traversal:

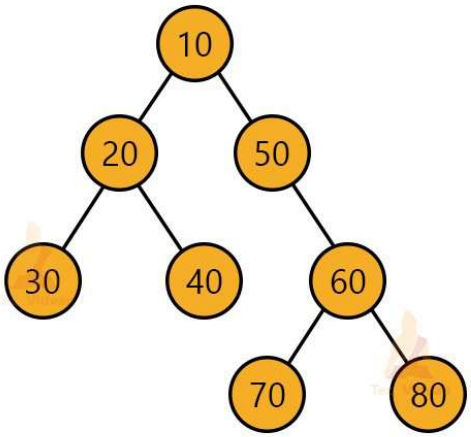
```
void Preorder(struct node* ptr)
{
if(ptr != NULL)
{
printf("%d", ptr->data);
Preorder(ptr->left);
Preorder(ptr->right);
}
}
```

Uses of Preorder Traversal:

- If we want to create a copy of a tree, we make use of preorder traversal.
- Preorder traversal helps to give a prefix expression for the expression tree.

Inorder Traversal:

In an inorder traversal, we first visit the left subtree, then the root node and then the right subtree in an inorder manner. Consider the following tree:



In this case, as we visit the left subtree first, we get the node with the value 30 first, then 20 and then 40. After that, we will visit the root node and print it. Then comes the turn of the right subtree. We will traverse the right subtree in a similar manner. Thus, after performing the inorder traversal, the order of nodes will be 30→20→40→10→50→70→60→80.

Algorithm for Inorder Traversal:

for all nodes of the tree:
Step 1: Traverse left subtree recursively.
Step 2: Visit the root node.
Step 3: Traverse right subtree recursively.

Pseudo-code for Inorder Traversal:

```
void Inorder(struct node* ptr)
{
if(ptr != NULL)
{
```

```
Inorder(ptr->left);
printf("%d", ptr->data);
Inorder(ptr->right);
}
}
```

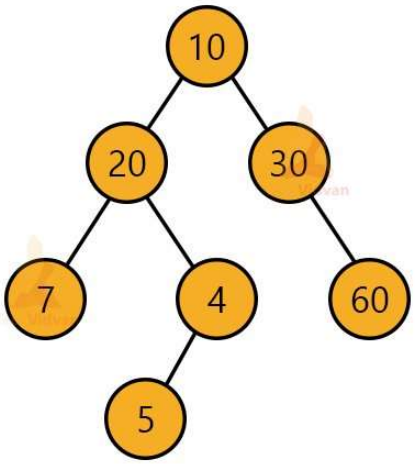
Uses of Inorder Traversal:

- Used to print the nodes of a binary search tree in non-decreasing order.
- We can also use the inorder traversal to get the non-increasing order of a BST.

Postorder Traversal:

Postorder traversal is a kind of traversal in which we first traverse the left subtree in a postorder manner, then traverse the right subtree in a postorder manner and at the end visit the root node.

For example, in the following tree:



The postorder traversal will be 7→5→4→20→60→30→10.

Algorithm for Postorder Traversal:

for all nodes of the tree:

Step 1: Traverse left subtree recursively.

Step 2: Traverse right subtree recursively.

Step 3: Visit the root node.

Pseudo-code for Postorder Traversal:

```
void Postorder(struct node* ptr)
{
if(ptr != NULL)
{
Postorder(ptr->left);
Postorder(ptr->right);
printf("%d", ptr->data);
}
}
```

Uses of Postorder Traversal:

- It helps to get the postfix expression in an expression tree.