

Tree Traversal

Tree Traversals (Inorder, Preorder and Postorder):

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree –

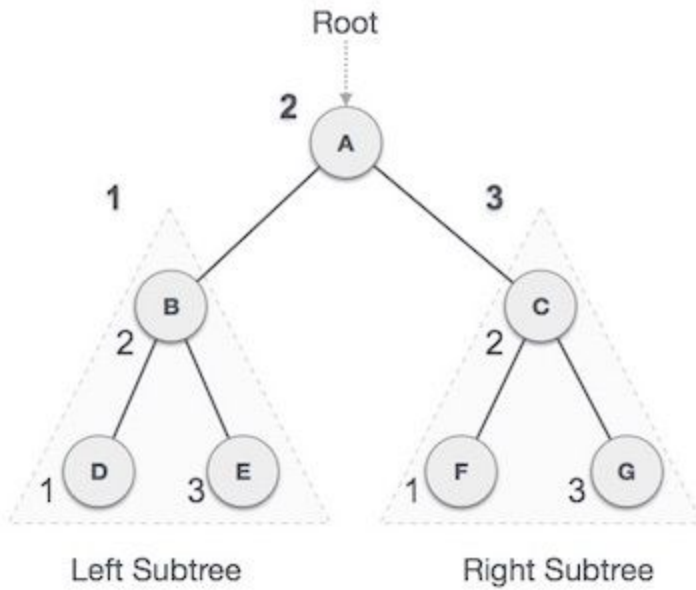
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

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.



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 –

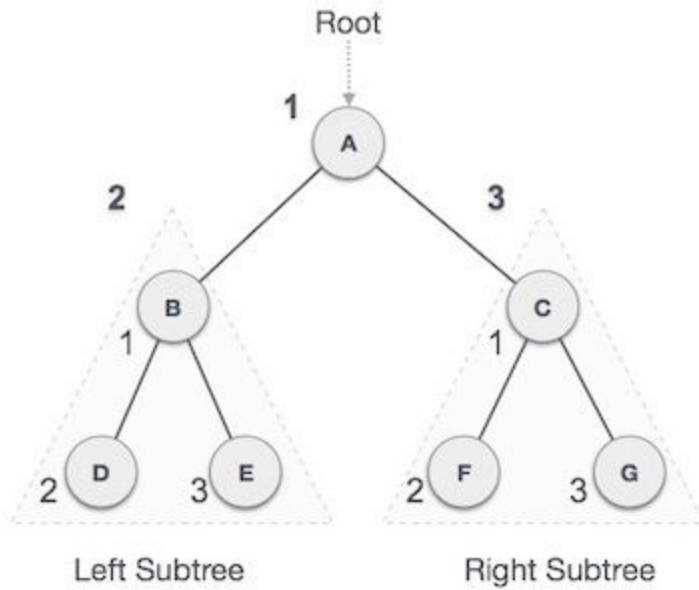
Step 1 – Recursively traverse left subtree.

Step 2 – Visit root node.

Step 3 – Recursively traverse right subtree.

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 –

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

Algorithm

Until all nodes are traversed –

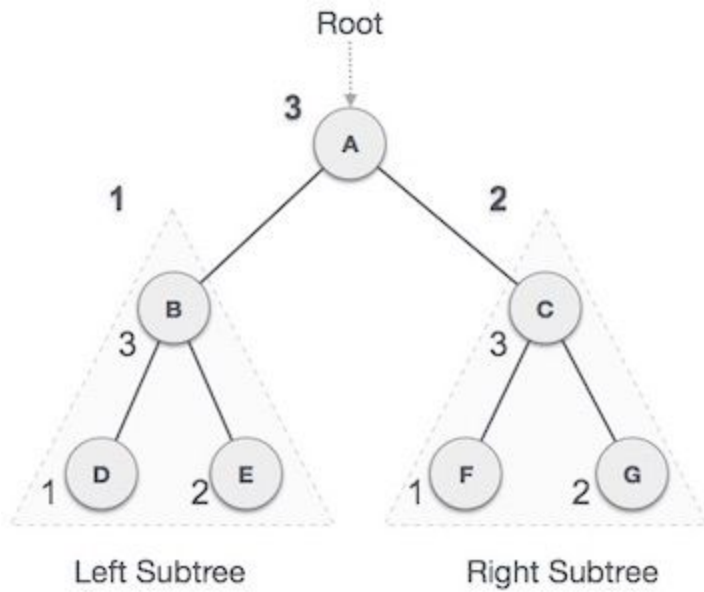
Step 1 – Visit root node.

Step 2 – Recursively traverse left subtree.

Step 3 – Recursively traverse right subtree.

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 –

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

Algorithm

Until all nodes are traversed –

Step 1 – Recursively traverse left subtree.

Step 2 – Recursively traverse right subtree.

Step 3 – Visit root node.

(Inorder, preorder and post order traversal)

traversal ‘codes’:

Code in 'c':

```

// C program for different tree traversals
#include <iostream>
using namespace std;

/* A binary tree node has data, pointer to left child
and a pointer to right child */
struct Node
{
    int data;
    struct Node* left, *right;
    Node(int data)
    {
        this->data = data;
        left = right = NULL;
    }
};

/* Given a binary tree, print its nodes according to the
"bottom-up" postorder traversal. */
void printPostorder(struct Node* node)
{
    if (node == NULL)
        return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

    // now deal with the node
    cout << node->data << " ";
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct Node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    cout << node->data << " ";

    /* now recur on right child */
    printInorder(node->right);
}

```

```

}

/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct Node* node)
{
    if (node == NULL)
        return;

    /* first print data of node */
    cout << node->data << " ";

    /* then recur on left subtree */
    printPreorder(node->left);

    /* now recur on right subtree */
    printPreorder(node->right);
}

/* Driver program to test above functions*/
int main()
{
    struct Node *root = new Node(1);
    root->left          = new Node(2);
    root->right          = new Node(3);
    root->left->left      = new Node(4);
    root->left->right     = new Node(5);

    cout << "\nPreorder traversal of binary tree is \n";
    printPreorder(root);

    cout << "\nInorder traversal of binary tree is \n";
    printInorder(root);

    cout << "\nPostorder traversal of binary tree is \n";
    printPostorder(root);

    return 0;
}

```

Code in “Java”:


```

// Java program for different tree traversals

/* Class containing left and right child of current
node and key value*/
class Node
{
    int key;
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}

class BinaryTree
{
    // Root of Binary Tree
    Node root;

    BinaryTree()
    {
        root = null;
    }

    /* Given a binary tree, print its nodes according to the
    "bottom-up" postorder traversal. */
    void printPostorder(Node node)
    {
        if (node == null)
            return;

        // first recur on left subtree
        printPostorder(node.left);

        // then recur on right subtree
        printPostorder(node.right);

        // now deal with the node
        System.out.print(node.key + " ");
    }

    /* Given a binary tree, print its nodes in inorder*/
    void printInorder(Node node)
    {
        if (node == null)
            return;

```

```

        /* first recur on left child */
        printInorder(node.left);

        /* then print the data of node */
        System.out.print(node.key + " ");

        /* now recur on right child */
        printInorder(node.right);
    }

    /* Given a binary tree, print its nodes in preorder*/
    void printPreorder(Node node)
    {
        if (node == null)
            return;

        /* first print data of node */
        System.out.print(node.key + " ");

        /* then recur on left subtree */
        printPreorder(node.left);

        /* now recur on right subtree */
        printPreorder(node.right);
    }

    // Wrappers over above recursive functions
    void printPostorder() {    printPostorder(root); }
    void printInorder() {    printInorder(root); }
    void printPreorder() {    printPreorder(root); }

    // Driver method
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        System.out.println("Preorder traversal of binary tree is ");
        tree.printPreorder();

        System.out.println("\nInorder traversal of binary tree is ");
        tree.printInorder();
    }
}

```

```
        System.out.println("\nPostorder traversal of binary tree is ");
        tree.printPostorder();
    }
}
```

