Tree Traversal

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 −

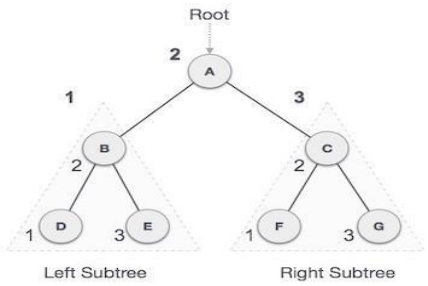
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

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 in-order traversal of this tree will be −

D → B → E → A → F → C → G

Algorithm

Until all nodes are traversed −

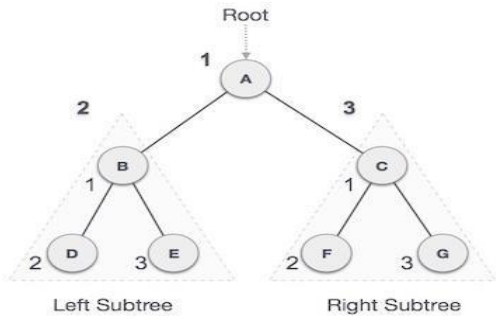
Step 1 − Recursively traverse left subtree.

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 → B → D → E → C → F → G

Algorithm

Until all nodes are traversed −

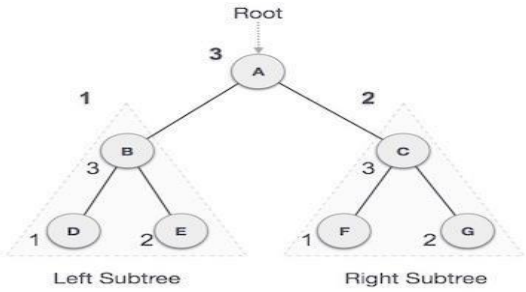
Step 1 − Visit root node.

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 pre-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 → E → B → F → G → C → A

Algorithm

Until all nodes are traversed −

Step 1 − Recursively traverse left subtree.

Step 2 − Recursively traverse right subtree.

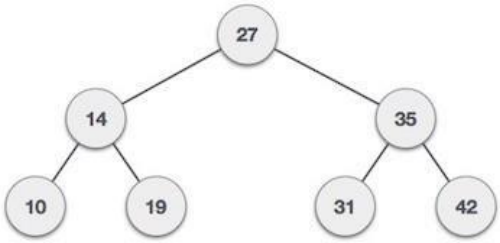
Step 3 − Visit root node.

Implementation

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

We shall now see the implementation of tree traversal in C programming language here using the following binary tree −



Preorder traversal: 27 14 10 19 35 31 42
Inorder traversal: 10 14 19 27 31 35 42
Post order traversal: 10 19 14 31 42 35 27

Implementation of Tree Traversal

PROGRAM:

```
 #include<stdlib.h>
#include<stdio.h>
structbin_tree {
int data;
structbin_tree * right, * left;
};
typedefstructbin_tree node;
void insert(node ** tree, intval)
{
   node *temp = null;
   if(!(*tree))
   {
      temp = (node *)malloc(sizeof(node));
      temp->left = temp->right = null;
      temp->data = val;
      *tree = temp;
      return;
   }
   if(val< (*tree)->data)
   {
      insert(&(*tree)->left, val);
   }
   else if(val> (*tree)->data)
```

```
    {
        insert(&(*tree)->right, val);
    }
}
void print_preorder(node * tree)
{
    if (tree)
    {
printf("%d\n",tree->data);
print_preorder(tree->left);
print_preorder(tree->right);
    }
}
void print_inorder(node * tree)
{
    if (tree)
    {
print_inorder(tree->left);
printf("%d\n",tree->data);
print_inorder(tree->right);
    }
}
void print_postorder(node * tree)
{
    if (tree)
    {
print_postorder(tree->left);
print_postorder(tree->right);
printf("%d\n",tree->data);
    }
}
void deltree(node * tree)
{
    if (tree)
    {
deltree(tree->left);
deltree(tree->right);
        free(tree);
    }
}

node* search(node ** tree, intval)
{
    if(!(*tree))
    {
        return null;
```

```
    }
    if(val< (*tree)->data)
    {
       search(&((*tree)->left), val);
    }
    else if(val> (*tree)->data)
    {
       search(&((*tree)->right), val);
    }
    else if(val == (*tree)->data)
    {
       return *tree;
    }
}
void main()
{
    node *root;
    node *tmp;
    //inti;
    root = null;
    /* inserting nodes into tree */
    insert(&root, 9);
    insert(&root, 4);
    insert(&root, 15);
    insert(&root, 6);
    insert(&root, 12);
    insert(&root, 17);
    insert(&root, 2);
    /* printing nodes of tree */
printf("pre order display\n");
print_preorder(root);
printf("in order display\n");
print_inorder(root);
printf("post order display\n");
print_postorder(root);
    /* search node into tree */
tmp = search(&root, 4);
    if (tmp)
    {
printf("searched node=%d\n", tmp->data);
    }
    else
    {
printf("data not found in tree.\n");
    }
    /* deleting all nodes of tree */
```

```
deltree(root);
}
```

OUTPUT:

```
Pre Order Display
9
4
2
6
15
12
17
In Order Display
2
4
6
9
12
15
17
Post Order Display
2
6
4
12
17
15
9
Searched node=4
```