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 −

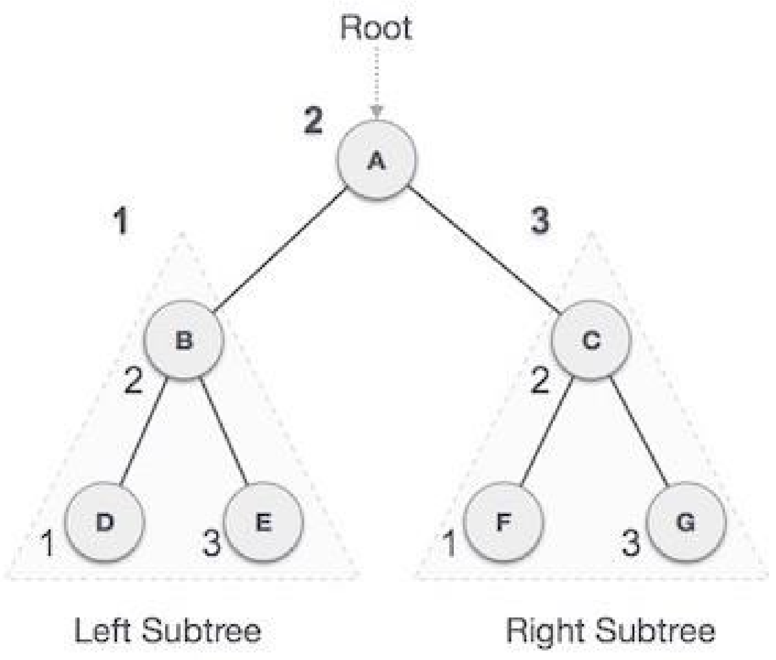
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

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.
```

Example

Following are the implementations of this operation in various programming languages −

C | C++ | Java | Python

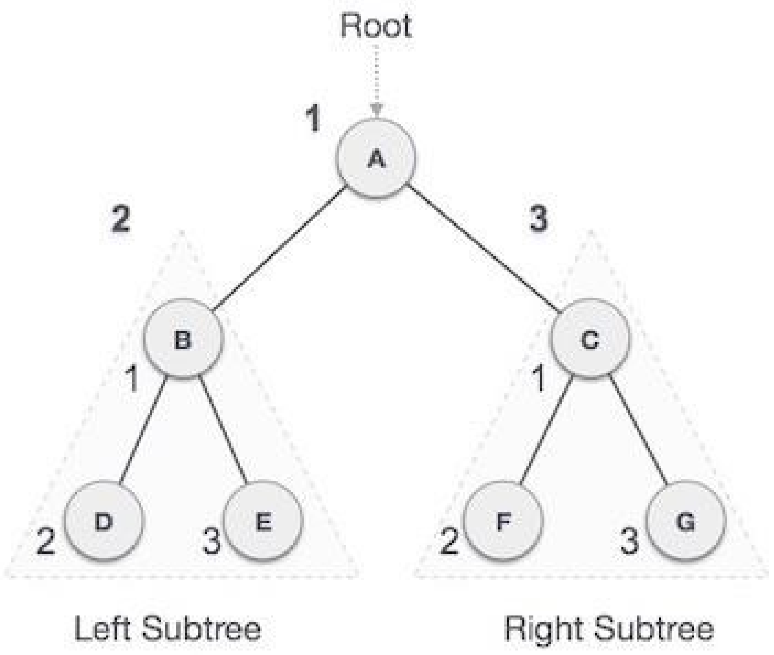
```c
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node *leftChild;
   struct node *rightChild;
};
struct node *root = NULL;
void insert(int data){
   struct node *tempNode = (struct node*) malloc(sizeof(struct n
   struct node *current;
   struct node *parent;
   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;

   //if tree is empty
   if(root == NULL) {
      root = tempNode;
   } else {
      current = root;
      parent = NULL;
      while(1) {
         parent = current;

         //go to left of the tree
         if(data < parent->data) {
            current = current->leftChild;

            //insert to the left
            if(current == NULL) {
               parent->leftChild = tempNode;
               return;
            }
         }//go to right of the tree
         else {
            current = current->rightChild;

            //insert to the right
            if(current == NULL) {
               parent->rightChild = tempNode;
               return;
            }
         }
      }
   }
}
void inorder_traversal(struct node* root){
   if(root != NULL) {
      inorder_traversal(root->leftChild);
      printf("%d ",root->data);
      inorder_traversal(root->rightChild);
   }
}
int main(){
   int i;
   int array[7] = { 27, 14, 35, 10, 19, 31, 42 };
   for(i = 0; i < 7; i++)
      insert(array[i]);
   printf("Inorder traversal: ");
   inorder_traversal(root);
   return 0;
```

Output

```
Inorder traversal: 10 14 19 27 31 35 42
```

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.
```

Example

Following are the implementations of this operation in various programming languages −

C | C++ | Java | Python

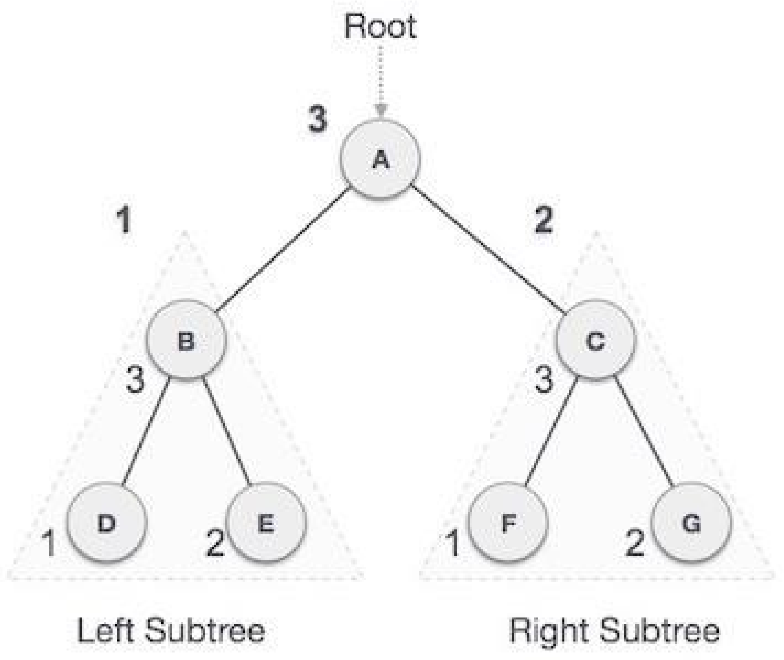
```c
#include <stdlib.h>
struct node {
   struct node *leftChild;
   struct node *rightChild;
};
struct node *root = NULL;
void insert(int data){
   struct node *tempNode = (struct node*) malloc(sizeof(struct n
   struct node *current;
   struct node *parent;
   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;

   //if tree is empty
   if(root == NULL) {
      root = tempNode;
   } else {
      current = root;
      parent = NULL;
      while(1) {
         parent = current;

         //go to left of the tree
         if(data < parent->data) {
            current = current->leftChild;

            //insert to the left
            if(current == NULL) {
               parent->leftChild = tempNode;
               return;
            }
         }//go to right of the tree
         else {
            current = current->rightChild;

            //insert to the right
            if(current == NULL) {
               parent->rightChild = tempNode;
               return;
            }
         }
      }
   }
void pre_order_traversal(struct node* root){
   if(root != NULL) {
      printf("%d ",root->data);
      pre_order_traversal(root->leftChild);
      pre_order_traversal(root->rightChild);
   }
}
int main(){
   int i;
   int array[7] = { 27, 14, 35, 10, 19, 31, 42 };
   for(i = 0; i < 7; i++)
      insert(array[i]);
   printf("Preorder traversal: ");
   pre_order_traversal(root);
   return 0;
}
```

Output

```
Preorder traversal: 27 14 10 19 35 31 42
```

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.
```

Example

Following are the implementations of this operation in various programming languages −

C C++ Java Python

```
#include <stdlib.h>
struct node {
   int data;
   struct node *rightChild;
};
struct node *root = NULL;
void insert(int data){
   struct node *tempNode = (struct node*) malloc(sizeof(struct n
   struct node *current;
   struct node *parent;
   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;

   //if tree is empty
   if(root == NULL) {
      root = tempNode;
   } else {
      current = root;
      parent = NULL;
      while(1) {
         parent = current;

         //go to left of the tree
         if(data < parent->data) {
            current = current->leftChild;

            //insert to the left
            if(current == NULL) {
               parent->leftChild = tempNode;
               return;
            }
         }//go to right of the tree
         else {
            current = current->rightChild;

            //insert to the right
            if(current == NULL) {
               parent->rightChild = tempNode;
               return;
            }
      }
   }
}
void post_order_traversal(struct node* root){
   if(root != NULL) {
      post_order_traversal(root->leftChild);
      post_order_traversal(root->rightChild);
      printf("%d ", root->data);
   }
}
int main(){
   int i;
   int array[7] = { 27, 14, 35, 10, 19, 31, 42 };
   for(i = 0; i < 7; i++)
      insert(array[i]);
   printf("Post order traversal: ");
   post_order_traversal(root);
   return 0;
}
```

Output

```
Post order traversal: 10 19 14 31 42 35 27
```

Complete Implementation

Now let's see the complete implementation of tree traversal in various programming languages −

C | C++ | Java | Python

```
#include <stdlib.h>
struct node {
   int data;
   struct node *leftChild;
```

```
};
struct node *root = NULL;
void insert(int data){
   struct node *tempNode = (struct node*) malloc(sizeof(struct n
   struct node *current;
   struct node *parent;
   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;

//if tree is empty
   if(root == NULL) {
      root = tempNode;
   } else {
      current = root;
      parent = NULL;
      while(1) {
         parent = current;

         //go to left of the tree
         if(data < parent->data) {
            current = current->leftChild;

            //insert to the left
            if(current == NULL) {
               parent->leftChild = tempNode;
               return;
            }
         }//go to right of the tree
         else {
            current = current->rightChild;

            //insert to the right
            if(current == NULL) {
               parent->rightChild = tempNode;
               return;
            }
         }
   }
}
void pre_order_traversal(struct node* root){
   if(root != NULL) {
      printf("%d ",root->data);
      pre_order_traversal(root->leftChild);
      pre_order_traversal(root->rightChild);
   }
}
void inorder_traversal(struct node* root){
   if(root != NULL) {
      inorder_traversal(root->leftChild);
      printf("%d ",root->data);
      inorder_traversal(root->rightChild);
   }
}
void post_order_traversal(struct node* root){
   if(root != NULL) {
      post_order_traversal(root->leftChild);
      post_order_traversal(root->rightChild);
      printf("%d ", root->data);
   }
}
int main(){
   int i;
   int array[7] = { 27, 14, 35, 10, 19, 31, 42 };
   for(i = 0; i < 7; i++)
      insert(array[i]);
   printf("Preorder traversal: ");
   pre_order_traversal(root);
   printf("\nInorder traversal: ");
   inorder_traversal(root);
   printf("\nPost order traversal: ");
   post_order_traversal(root);
   return 0;
}
```

Output

```
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
```