

## Experiment No. 8: Binary Search Tree Operations

**Name : Harsh M. Upadhyay**

**Roll No: 57(S3)**

**Aim : Implementation of Binary Search Tree ADT using Linked List.**

**Objective:**

- 1) Understand how to implement a BST using a predefined BST ADT.
- 2) Understand the method of counting the number of nodes of a binary tree.

**Theory:**

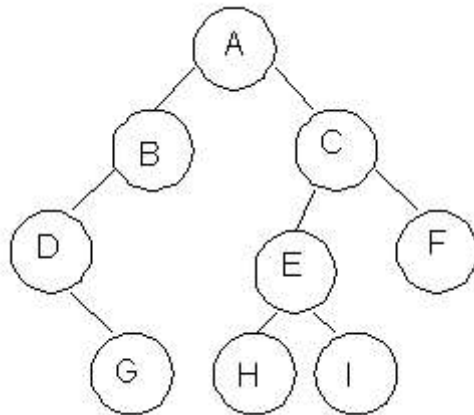
A binary tree is a finite set of elements that is either empty or partitioned into disjoint subsets. In other words node in a binary tree has at most two children and each child node is referred as left or right child.

Traversals in tree can be in one of the three ways : preorder, postorder, inorder.

### Preorder Traversal

Here the following strategy is followed in sequence

1. Visit the root node R
2. Traverse the left subtree of R
3. Traverse the right sub tree of R



Description	Output
Visit Root	A

Traverse left sub tree – step to B then D	ABD
Traverse right sub tree – step to G	ABDG
As left subtree is over. Visit root , which is already visited so go for right subtree	ABDGC
Traverse the left subtree	ABDGCEH
Traverse the right sub tree	ABDGCEHI F

### Inorder Traversal

Here the following strategy is followed in sequence

1. Traverse the left subtree of R
2. Visit the root node R
3. Traverse the right sub tree of R

<b>Description</b>	<b>Output</b>
Start with root and traverse left sub tree from A-B-D	D
As D doesn't have left child visit D and go for right subtree of D which is G so visit this.	DG
Backtrack to D and then to B and visit it.	DGB
Backtrack to A and visit it	DGBA
Start with right sub tree from C-E-H and visit H	DGBAH
Now traverse through parent of H which is E and then I	DGBAHEI
Backtrack to C and visit it and then right subtree of E which is F	DGBAHEICF

### Postorder Traversal

Here the following strategy is followed in sequence

1. Traverse the left subtree of R
2. Traverse the right sub tree of R
3. Visit the root node R

<b>Description</b>	<b>Output</b>
Start with left sub tree from A-B-D and then traverse right sub tree to get G	G
Now Backtrack to D and visit it then to B and visit it.	GD
Now as the left sub tree is over go for right sub tree	GDB
In right sub tree start with leftmost child to visit H followed by I	GDBHI
Visit its root as E and then go for right sibling of C as F	GDBHIEF
Traverse its root as C	GDBHIEFC
Finally a root of tree as A	GDBHIEFCA

### **Algorithm**

Algorithm: PREORDER(ROOT)

Input : Root is a pointer to root node of binary tree

Output : Visiting all the nodes in preorder fashion.

Description : Linked structure of binary tree

1. ptr=ROOT
2. if ptr!=NULL then
  - visit(ptr)
  - PREORDER(LSON(ptr))\
  - PREORDER(RSON(ptr))
  - End if
3. Stop

Algorithm: INORDER(ROOT)

Input : Root is a pointer to root node of binary tree

Output : Visiting all the nodes in inorder fashion.

Description : Linked structure of binary tree

1. ptr=ROOT
2. if ptr!=NULL then

```

        INORDER (LSON(ptr))
        visit(ptr)
        INORDER (RSON(ptr))
    End if
3. Stop

```

Algorithm: POSTORDER(ROOT)

Input : Root is a pointer to root node of binary tree

Output : Visiting all the nodes in postorder fashion.

Description : Linked structure of binary tree

```

1. ptr=ROOT
2. if ptr!=NULL then
    PREORDER(LSON(ptr))
    PREORDER(RSON(ptr))
    visit(ptr)
End if
3. Stop

```

### Code:

```

#include <stdio.h>

#include <stdlib.h>

struct node {
    int data;

    struct node *leftChild, *rightChild;
};

struct node *root = NULL;

struct node *newNode(int item){

```

```

    struct node *temp = (struct node *)malloc(sizeof(struct node));

    temp->data = item;

    temp->leftChild = temp->rightChild = NULL;

    return temp;
}

void insert(int data){

    struct node *tempNode = (struct node*) malloc(sizeof(struct node));

    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;
    }
}
}
}

struct node* search(int data){
    struct node *current = root;
    printf("\n\nVisiting elements: ");
    while(current->data != data) {

```

```

if(current != NULL) {
    printf("%d ",current->data);

    //go to left tree
    if(current->data > data) {
        current = current->leftChild;
    } //else go to right tree
    else {
        current = current->rightChild;
    }

    //not found
    if(current == NULL) {
        return NULL;
    }
}

return current;
}

```

```

// Inorder Traversal
void inorder(struct node *root){
    if (root != NULL) {
        inorder(root->leftChild);

```

```
        printf("%d -> ", root->data);  
        inorder(root->rightChild);  
    }  
}
```

// Preorder Traversal

```
void preorder(struct node *root){  
    if (root != NULL) {  
        printf("%d -> ", root->data);  
        preorder(root->leftChild);  
        preorder(root->rightChild);  
    }  
}
```

// Postorder Traversal

```
void postorder(struct node *root){  
    if (root != NULL) {  
        printf("%d -> ", root->data);  
        postorder(root->leftChild);  
        postorder(root->rightChild);  
    }  
}
```

```
int main(){  
    insert(10);
```



```
insert(14);
insert(19);
insert(26);
insert(27);
insert(31);
insert(33);
insert(35);
insert(42);
insert(44);
printf("Insertion done\n");
printf("\nPreorder Traversal: ");
preorder(root);
printf("\nInorder Traversal: ");
inorder(root);
printf("\nPostorder Traversal: ");
postorder(root);
struct node* k;
k = search(35);
if(k != NULL)
    printf("\nElement %d found", k->data);
else
    printf("\nElement not found");
return 0;
}
```

## Output:

```
C:\Users\Student\Desktop\binary traversal.exe
Insertion done

Preorder Traversal: 10 -> 14 -> 19 -> 26 -> 27 -> 31 -> 33 -> 35 -> 42 -> 44 ->
Inorder Traversal: 10 -> 14 -> 19 -> 26 -> 27 -> 31 -> 33 -> 35 -> 42 -> 44 ->
Postorder Traversal: 10 -> 14 -> 19 -> 26 -> 27 -> 31 -> 33 -> 35 -> 42 -> 44 ->

Visiting elements: 10 14 19 26 27 31 33
Element 35 found
-----
Process exited after 0.01166 seconds with return value 0
Press any key to continue . . .
```

**Conclusion:** Binary trees have many applications in computer science, including data storage and retrieval, expression evaluation, network routing, and game AI.