INDEX

Ex. No.: 13

Date :

Tree Traversals in BST

Aim: To write a C-program to demonstrate tree traversals in Binary Search Tree.

1. Binary Search Tree:

A **binary search tree** is a rooted binary tree, whose internal nodes each store a key (and optionally, an associated value) and each have two distinguished sub-trees, commonly denoted *left* and *right*. The tree additionally satisfies the binary search property, which states that the key in each node must be greater than or equal to any key stored in the left sub-tree, and less than or equal to any key stored in the right sub-tree. The leaves (final nodes) of the tree contain no key and have no structure to distinguish them from one another.

Frequently, the information represented by each node is a record rather than a single data element. However, for sequencing purposes, nodes are compared according to their keys rather than any part of their associated records. The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient

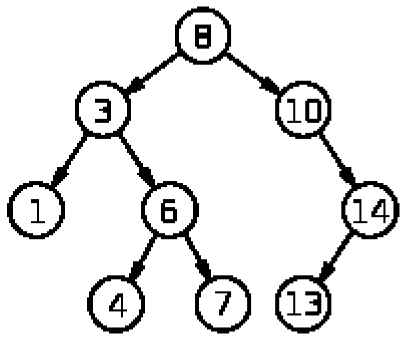

Fig: binary search tree

2. Tree Traversal

There are three popular methods of traversal

2.1.1 Pre-order traversal

The preorder traversal of a nonempty binary tree is defined as follows:

i. Visit the root node

ii. Traverse the left sub-tree in preorder

iii. Traverse the right sub-tree in preorder

2.1.2 In-order traversal

The inorder traversal of a nonempty binary tree is defined as follows:

i. Traverse the left sub-tree in inorder

ii. Visit the root node

iii. Traverse the right sub-tree in inorder

2.1.3 Post-order traversal

The post-order traversal of a nonempty binary tree is defined as follows:

i. Traverse the left sub-tree in post-order

ii. Traverse the right sub-tree in post-order

iii. Visit the root node

1.2 Program

```
#include<stdio.h>
#include<stdlib.h>

typedef struct BST
{
        int data;
        struct BST *left;
        struct BST *right;
}node;

node *create();
void insert(node *,node *);
void preorder(node *);
void postorder(node *);
void inorder(node *);

int main()
{
        char ch;
        node *root=NULL,*temp;

    while(1)
    {
        printf("\nOPERATIONS ---");
        printf("\n1 - Insert an element into tree\n");
        printf("2 - Delete an element from the tree\n");
        printf("3 - Inorder Traversal\n");
        printf("4 - Preorder Traversal\n");
        printf("5 - Postorder Traversal\n");
        printf("6 - Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
                temp=create();
                if(root==NULL)
                    root=temp;
                else
                    insert(root,temp);
                break;
        case 2:
           // delete();
            break;
        case 3:
            printf("\nInorder Traversal: ");
            inorder(root);
            break;
        case 4:
            printf("\nPreorder Traversal: ");
            preorder(root);
            break;
        case 5:
            printf("\nPostorder Traversal: ");
            postorder(root);
            break;
        case 6:
            exit(0);
        default :
            printf("Wrong choice, Please enter correct choice  ");
            break;
        }

    }


        return 0;
}

node *create()
{
        node *temp;
        printf("\nEnter data:");
        temp=(node*)malloc(sizeof(node));
        scanf("%d",&temp->data);
        temp->left=temp->right=NULL;
        return temp;
}

void insert(node *root,node *temp)
{
        if(temp->data<root->data)
        {
                if(root->left!=NULL)
                        insert(root->left,temp);
                else
                        root->left=temp;
        }

        if(temp->data>root->data)
        {
                if(root->right!=NULL)
                        insert(root->right,temp);
                else
                        root->right=temp;
        }
}

void preorder(node *root)
{
        if(root!=NULL)
        {
                printf("%d ",root->data);
                preorder(root->left);
                preorder(root->right);
        }
}
void postorder(node *root)
{
        if(root!=NULL)
        {

                postorder(root->left);
                postorder(root->right);
                printf("%d ",root->data);
        }
}
void inorder(node *root)
{
        if(root!=NULL)
        {

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

Algorithm	**Average**	**Worst case**
Space	O(*n*)	O(*n*)
Search	O(log *n*)	O(*n*)
Insert	O(log *n*)	O(*n*)
Delete	O(log *n*)	O(*n*)