**TITLE 31**

Create a Binary Search Tree(BST) and search for a given value in BST

**OBJECTIVE:**

By the end of this problem we will be able to create a Binary Search Tree and search for a given value in BST

**PROBLEM STATEMENT:**

In this problem we create a BST and search for a given value in BST.

Once the input is stored then the output is printed.

**ALGORITHM:**

START

Define variable: data

INPUT: Read from the keyboard

COMPUTATION: Computing the tree traversal of the given data

DISPLAY: Displaying the tree traversal of the data and also if the element entered is found or not

STOP

**PROGRAM:**

#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 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("Visiting 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;

}

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]);

i = 31;

struct node \* temp = search(i);

if(temp != NULL) {

printf("[%d] Element found.", temp->data);

printf("\n");

}else {

printf("[ x ] Element not found (%d).\n", i);

}

i = 15;

temp = search(i);

if(temp != NULL) {

printf("[%d] Element found.", temp->data);

printf("\n");

}else {

printf("[ x ] Element not found (%d).\n", i);

}

printf("\nPreorder traversal: ");

pre\_order\_traversal(root);

printf("\nInorder traversal: ");

inorder\_traversal(root);

printf("\nPost order traversal: ");

post\_order\_traversal(root);

return 0;

}

**CONCLUSION:**

The simulation of the above C program helped me understand how a BST is created and tree traversal of the data considered.

**OUTPUT:**

Visiting elements: 27 35 [31] Element found

Visiting elements: 27 14 19 [x] Element not found (15)

Preorder traversal: 27 14 10 19 35 31 42

Inorder traversal: 10 14 19 27 31 35 42

Postorder traversal: 10 19 14 31 42 35 27