**M. Ali. Arslan**

**19F-0348**

**Assignment-03**

**Code:**

#include<iostream>

#define SPACE 10

using namespace std;

class Node

{

public:

Node\* left;

Node\* right;

int data;

Node()

{

left = NULL;

right = NULL;

data = 0;

}

Node(int value)

{

left = NULL;

right = NULL;

data = value;

}

};

class BST

{

public:

Node\* root;

BST()

{

root = NULL;

}

bool isEmpty()

{

if (root == NULL)

{

return true;

}

else

{

return false;

}

}

bool addNode(Node\* new\_node)

{

if (isEmpty())

{

root = new\_node;

cout << "Value is added as root node of Binary Search Tree!" << endl;

return true;

}

else

{

Node\* temp = root;

while (temp != NULL)

{

if (new\_node->data == temp->data)

{

cout << "Value is already present in Binary Search Tree!" << endl;

return false;

}

else if ((new\_node->data < temp->data) && (temp->left == NULL))

{

temp->left = new\_node;

cout << "Value is added to left side of Binary Search Tree!" << endl;

return true;

break;

}

else if ((new\_node->data < temp->data))

{

temp = temp->left;

}

else if ((new\_node->data > temp->data) && (temp->right == NULL))

{

temp->right = new\_node;

cout << "Value is added to right side of Binary Search Tree!" << endl;

return true;

break;

}

else

{

temp = temp->right;

}

}

}

}

void destroy(Node\* present\_node)

{

if (present\_node != NULL)

{

destroy(present\_node->left);

destroy(present\_node->right);

delete present\_node;

present\_node = NULL;

}

}

~BST()

{

destroy(root);

}

int height(Node\* present\_node)

{

if (present\_node == NULL)

{

return -1;

}

int left\_height = height(present\_node->left);

int right\_height = height(present\_node->right);

if (left\_height > right\_height)

{

return left\_height + 1;

}

else

{

return right\_height + 1;

}

}

bool isBalanced(Node\* p\_node)

{

if (p\_node == NULL)

{

return true;

}

if (isBalanced(p\_node->left) == false)

{

return false;

}

if (isBalanced(p\_node->right) == false)

{

return false;

}

int leftheight = height(p\_node->left);

int rigthheight = height(p\_node->right);

if (abs(leftheight - rigthheight) <= 1)

{

return true;

}

else

{

return false;

}

}

void display(Node\* p\_node, int space)

{

if (p\_node == NULL)

{

return;

}

space = space + SPACE;

display(p\_node->right, space);

cout << endl;

int i = SPACE;

while (i < space)

{

cout << " ";

i++;

}

cout << p\_node->data << endl;

display(p\_node->left, space);

}

void successor(Node\* s)

{

if (root == NULL)

{

return;

}

if (s->left == NULL && s->right == NULL)

{

cout << endl;

cout << " No Successor!" << endl;

}

else if (s->left == NULL && s->right != NULL)

{

cout << endl;

cout << s->right->data << " is Successor!" << endl;

}

else if (s->right == NULL && s->left != NULL)

{

cout << endl;

cout << s->left->data << " is Successor!" << endl;

}

else

{

cout << endl;

cout << s->left->data << " & " << s->right->data << " are Successors!" << endl;

}

}

Node\* search(Node\* r, int s)

{

if (r == NULL)

{

return r;

}

Node\* temp = root;

Node\* ptr = NULL;

while (temp != NULL)

{

if (s < temp->data)

{

temp = temp->left;

}

else if (s > temp->data)

{

temp = temp->right;

}

else

{

ptr = temp;

return ptr;

}

}

}

void predecessor(Node\* s)

{

{

if (root == NULL)

{

return;

}

Node\* temp = root;

while (temp != NULL)

{

if (temp->right == s || temp->left == s)

{

cout << temp->data << " is Predessor!" << endl;

return;

}

else if (temp == s)

{

cout << "No Predecessor!" << endl;

return;

}

else

{

if (s->data < temp->data)

{

temp = temp->left;

}

else if (s->data > temp->data)

{

temp = temp->right;

}

}

}

}

}

};

int main()

{

BST b;

Node\* n = new Node;

int choice, value;

do

{

cout << "1) Add Value in BST" << endl;

cout << "2) Check if BST is Empty or Not" << endl;

cout << "3) Height of BST" << endl;

cout << "4) Check if BST is isBalanced or Not" << endl;

cout << "5) Print tree" << endl;

cout << "6) Find the successor of BST " << endl;

cout << "7) Find the predecessor of BST " << endl;

cout << "0) Exit " << endl;

cout << "Enter Choice: ";

cin >> choice;

cout << endl;

switch (choice)

{

case 0:

exit(0);

case 1:

cout << "Enter value to be added in BST: ";

cin >> value;

n->data = value;

if (b.addNode(n))

{

cout << "Added Sucessfully!" << endl << endl;

}

else

{

cout << "Operation Unsucessfull!" << endl << endl;

}

break;

case 2:

if (b.isEmpty())

{

cout << "BST is Empty!" << endl << endl;

break;

}

else

{

cout << "BST is Not Empty!" << endl << endl;

break;

}

case 3:

cout << "Height of BST is: " << b.height(b.root);

cout << endl << endl;

break;

case 4:

if (b.isBalanced(b.root))

{

cout << "BST is Balancd!" << endl;

}

else

{

cout << "Not isBalanced!" << endl;

}

break;

case 5:

cout << "The Tree is: " << endl;

b.display(b.root, 8);

cout << endl << endl;

break;

case 6:

cout << "Enter node to find its successor: ";

cin >> value;

b.successor(b.search(b.root, value));

break;

case 7:

cout << "Enter node to find its predessor: ";

cin >> value;

b.predecessor(b.search(b.root, value));

break;

default:

cout << "Invalid Input!" << endl;

break;

}

} while (choice != 0);

system("pause");

return 0;

}



