Bahria University,

Karachi Campus

A picture containing text, room

Description automatically generated

LAB EXPERIMENT NO.

**10**

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
|  | **BST AND AVL TREE** |
| 1 | Write a program to implement concept of Binary Search Tree using dynamic trees |
| 2 | Implement the AVL Tree by performing searching |

Submitted On:

(Date: DD/MM/YY)

**Task No. 1:** Write a program to implement concept of Binary Search Tree using dynamic trees

**Solution:**

static void Main(string[] args){

Tree BST = new Tree();

BST.Insert(20);

BST.Insert(25);

BST.Insert(45);

BST.Insert(15);

BST.Insert(67);

BST.Insert(43);

BST.Insert(80);

BST.Insert(33);

BST.Insert(67);

BST.Insert(99);

BST.Insert(91);

Console.WriteLine("Inorder Traversal of BST: ");

BST.Inorder(BST.ReturnRoot());

Console.WriteLine(" ");

Console.WriteLine();

Console.WriteLine("Preorder Traversal of BST: ");

BST.Preorder(BST.ReturnRoot());

Console.WriteLine(" ");

Console.WriteLine();

Console.WriteLine("Postorder Traversal of BST: ");

BST.Postorder(BST.ReturnRoot());

Console.WriteLine(" ");

Console.ReadLine();

Console.WriteLine("Have a nice Day !!!");}

//Node Class

class Node{

public int item;

public Node leftchild;

public Node rightchild;

public void display(){

Console.Write("[");

Console.Write(item);

Console.Write("]");}}

// Tree Class

class Tree{

public Node root;

public Tree(){

root = null;}

public Node ReturnRoot(){

return root;}

public void Insert(int id){

Node newNode = new Node();

newNode.item = id;

if (root == null)

root = newNode;

else{

Node current = root;

Node parent;

while (true){

parent = current;

if (id < current.item){

current = current.leftchild;

if (current == null){

parent.leftchild = newNode;

return;}}

else{

current = current.rightchild;

if (current == null){

parent.rightchild = newNode;

return;}}}}}

public void Preorder(Node Root){

if (Root != null){

Console.Write(Root.item + " ");

Preorder(Root.leftchild);

Preorder(Root.rightchild);}}

public void Inorder(Node Root){

if (Root != null){

Inorder(Root.leftchild);

Console.Write(Root.item + " ");

Inorder(Root.rightchild);}}

public void Postorder(Node Root){

if (Root != null){

Postorder(Root.leftchild);

Postorder(Root.rightchild);

Console.Write(Root.item + " ");}}

**Output:**

Text

Description automatically generated

**Task No. 2:** Implement the AVL Tree by performing searching.

**Solution:**

//Main Method

static void Main(string[] args){

AVLtree tree = new AVLtree();

Console.WriteLine("data that you stored in the nodes of avl tree are as follows: ");

tree.Add(11);

tree.Add(4);

tree.Add(6);

tree.Add(1);

tree.Add(9);

tree.Add(12);

tree.display();

tree.Delete(4);

tree.Delete(1);

Console.WriteLine("after deleting nodes containing a data 4 & 1 from avl tree: ");

tree.display();

Console.Write("Enter the node you want to search in AVL Tree: ");

int node = int.Parse(Console.ReadLine());

tree.search(node);

Console.WriteLine("");

Console.ReadLine();

Console.WriteLine("Have a nice Day !!!");}}

//AVL Tree class

class AVLtree{

class Node{

public int data;

public Node left;

public Node right;

public Node(int data){

this.data = data;}}

Node root;

public AVLtree(){}

public void Add(int data){

Node newItem = new Node(data);

if (root == null){

root = newItem;}

else{

root = RecursiveInsert(root, newItem);}}

private Node RecursiveInsert(Node current, Node n){

if (current == null){

current = n;

return current;}

else if (n.data < current.data){

current.left = RecursiveInsert(current.left, n);

current = balance\_tree(current);}

else if (n.data > current.data){

current.right = RecursiveInsert(current.right, n);

current = balance\_tree(current);}

return current;}

private Node balance\_tree(Node current){

int b\_factor = balance\_factor(current);

if (b\_factor > 1){

if (balance\_factor(current.left) > 0){

current = RotateLL(current);}

else{

current = RotateLR(current);}}

else if (b\_factor < -1){

if (balance\_factor(current.right) > 0){

current = RotateRL(current);}

else{

current = RotateRR(current);}}

return current;}

public void Delete(int target){

root = Delete(root, target);}

private Node Delete(Node current, int target){

Node parent;

if (current == null)

{ return null; }

else{

if (target < current.data){

current.left = Delete(current.left, target);

if (balance\_factor(current) == -2){

if (balance\_factor(current.right) <= 0){

current = RotateRR(current);}

else{

current = RotateRL(current);}}}

else if (target > current.data){

current.right = Delete(current.right, target);

if (balance\_factor(current) == 2){

if (balance\_factor(current.left) >= 0){

current = RotateLL(current);}

else{

current = RotateLR(current);}}}

//if target is found

else{

if (current.right != null){

//delete its inorder successor

parent = current.right;

while (parent.left != null){

parent = parent.left;}

current.data = parent.data;

current.right = Delete(current.right, parent.data);

if (balance\_factor(current) == 2)//rebalancing{

if (balance\_factor(current.left) >= 0){

current = RotateLL(current);}

else { current = RotateLR(current); }}}

else{

return current.left;}}}

return current;}

public void search(int key){

if (search(key, root).data == key){

Console.WriteLine(key + " was found in avl tree!");}

else if (search(key, root).data != key){

Console.WriteLine("Nothing found!");}}

private Node search(int target, Node current){

if (target < current.data){

if (target == current.data){

return current;}

else

return search(target, current.left);}

else{

if (target == current.data){

return current;}

else

return search(target, current.right);}}

public void display(){

if (root == null){

Console.WriteLine("Tree is empty");

return;}

InOrderDisplayTree(root);

Console.WriteLine();}

private void InOrderDisplayTree(Node current){

if (current != null){

InOrderDisplayTree(current.left);

Console.Write("{0} ", current.data);

InOrderDisplayTree(current.right);}}

private int max(int l, int r){

return l > r ? l : r;}

private int getHeight(Node current)

{

int height = 0;

if (current != null){

int l = getHeight(current.left);

int r = getHeight(current.right);

int m = max(l, r);

height = m + 1;}

return height;}

private int balance\_factor(Node current){

int l = getHeight(current.left);

int r = getHeight(current.right);

int b\_factor = l - r;

return b\_factor;}

private Node RotateRR(Node parent){

Node pivot = parent.right;

parent.right = pivot.left;

pivot.left = parent;

return pivot;}

private Node RotateLL(Node parent){

Node pivot = parent.left;

parent.left = pivot.right;

pivot.right = parent;

return pivot;}

private Node RotateLR(Node parent){

Node pivot = parent.left;

parent.left = RotateRR(pivot);

return RotateLL(parent); }

private Node RotateRL(Node parent{

Node pivot = parent.right;

parent.right = RotateLL(pivot);

return RotateRR(parent); }

Text

Description automatically generated**Output:**