Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

\_\_10\_\_\_\_

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 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: 6/1/2022

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

**trees.**

**Solution:**

**Class Tree**

class Node

{

public int item;

public Node leftchild;

public Node rightchild;

public void display()

{

Console.Write("[");

Console.Write(item);

Console.Write("]");

}

}

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 + " ");

}

}

}

**Main Method**

static void Main(string[] args)

{

Tree BST = new Tree();

BST.Insert(99);

BST.Insert(42);

BST.Insert(11);

BST.Insert(2);

BST.Insert(3);

BST.Insert(8);

BST.Insert(90);

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();

}

**Output:**

Text

Description automatically generated

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

**Solution:**

**Class**

public class Node

{

public int data, height;

public Node left, right;

public Node(int d)

{

data = d;

height = 1;

}

}

public class AVLTree

{

Node root;

public void Insert(int data)

{

root = InsertRecursion(root, data);

}

public Node InsertRecursion(Node curr, int data)

{

if (curr == null)

return (new Node(data));

if (data < curr.data)

curr.left = InsertRecursion(curr.left, data);

else if (data > curr.data)

curr.right = InsertRecursion(curr.right, data);

else

return curr;

curr.height = 1 + Max(Height(curr.left), Height(curr.right));

return BalanceAVL(curr);

}

public Node MinValueNode(Node node)

{

Node curr = node;

/\* loop down to find the leftmost leaf \*/

while (curr.left != null)

curr = curr.left;

return curr;

}

public void Delete(int data)

{

root = DeleteRecursion(root, data);

}

public Node DeleteRecursion(Node curr, int data)

{

if (curr == null)

return root;

if (data < curr.data)

curr.left = DeleteRecursion(curr.left, data);

else if (data > curr.data)

curr.right = DeleteRecursion(curr.right, data);

else if (curr.data == data)

{

//One or No child

if ((curr.left == null) || (curr.right == null))

{

Node temp = null;

if (temp == curr.left)

temp = curr.right;

else

temp = curr.left;

// No child case

if (temp == null)

{

curr = null;

}

else // One child case

curr = temp;

}

//Two children

else

{

Node temp = MinValueNode(curr.right);

curr.data = temp.data;

curr.right = DeleteRecursion(curr.right, temp.data);

}

}

if (curr == null)

return curr;

curr.height = Max(Height(curr.left), Height(curr.right)) + 1;

return BalanceAVL(curr);

}

public Node BalanceAVL(Node curr)

{

int balance = GetBalanceFactor(curr);

// Left Left Case

if (balance > 1 && GetBalanceFactor(curr.left) >= 0)

return RightRotate(curr);

// Left Right Case

if (balance > 1 && GetBalanceFactor(curr.left) < 0)

{

curr.left = LeftRotate(curr.left);

return RightRotate(curr);

}

// Right Right Case

if (balance < -1 && GetBalanceFactor(curr.right) <= 0)

return LeftRotate(curr);

// Right Left Case

if (balance < -1 && GetBalanceFactor(curr.right) > 0)

{

curr.right = RightRotate(curr.right);

return LeftRotate(curr);

}

return curr;

}

public int Height(Node N)

{

if (N == null)

return 0;

return N.height;

}

public int GetBalanceFactor(Node N)

{

if (N == null)

return 0;

return Height(N.left) - Height(N.right);

}

public int Max(int a, int b)

{

return (a > b) ? a : b;

}

public Node RightRotate(Node y)

{

Node x = y.left;

Node T2 = x.right;

x.right = y;

y.left = T2;

y.height = Max(Height(y.left), Height(y.right)) + 1;

x.height = Max(Height(x.left), Height(x.right)) + 1;

return x;

}

public Node LeftRotate(Node x)

{

Node y = x.right;

Node T2 = y.left;

y.left = x;

x.right = T2;

x.height = Max(Height(x.left), Height(x.right)) + 1;

y.height = Max(Height(y.left), Height(y.right)) + 1;

return y;

}

public void Search(int data)

{

Console.WriteLine("{0} exists in the tree: {1}", data, SearchRecursion(root, data));

}

public bool SearchRecursion(Node curr, int data)

{

if (curr == null)

return false;

if (data == curr.data)

return true;

else if (data < curr.data)

return SearchRecursion(curr.left, data);

else

return SearchRecursion(curr.right, data);

}

public void Display()

{

Console.WriteLine("Inorder AVL:");

InorderTraversal(root);

Console.WriteLine();

}

public void InorderTraversal(Node curr)

{

if (curr != null)

{

InorderTraversal(curr.left);

Console.Write(curr.data + " ");

InorderTraversal(curr.right);

}

}

**Main method**

public static void Main(String[] args)

{

AVLTree obj = new AVLTree();

obj.Insert(1);

obj.Insert(2);

obj.Insert(3);

obj.Insert(4);

obj.Insert(5);

obj.Insert(6);

Console.WriteLine("Ininitial Tree: ");

obj.Display();

Console.WriteLine("After del '4': ");

obj.Delete(4);

obj.Display();

Console.WriteLine("Searching '4': ");

obj.Search(4);

}

}

Text

Description automatically generated

**Output:**