**Task # 1:** Implement the AVL Tree by performing searching.

**Solution**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab11\_AVL

{

class AVL

{

class Node

{

public int data;

public Node left;

public Node right;

public Node(int data)

{

this.data = data;

}

}

Node root;

public AVL()

{

}

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)

{//and here

root = Delete(root, target);

}

private Node Delete(Node current, int target)

{

Node parent;

if (current == null)

{ return null; }

else

{

//left subtree

if (target < current.data)

{

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

if (balance\_factor(current) == -2)//here

{

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

{

current = RotateRR(current);

}

else

{

current = RotateRL(current);

}

}

}

//right subtree

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

{ //if current.left != null

return current.left;

}

}

}

return current;

}

public void Find(int key)

{

if (Find(key, root).data == key)

{

Console.WriteLine("{0} was found!", key);

}

else

{

Console.WriteLine("Nothing found!");

}

}

private Node Find(int target, Node current)

{

if (target < current.data)

{

if (target == current.data)

{

return current;

}

else

return Find(target, current.left);

}

else

{

if (target == current.data)

{

return current;

}

else

return Find(target, current.right);

}

}

public void DisplayTree()

{

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);

}

static void Main(string[] args)

{

Console.WriteLine("The AVL tree is:");

AVL tree = new AVL();

tree.Add(33);

tree.Add(22);

tree.Add(44);

tree.Add(66);

tree.Add(99);

tree.Add(100);

tree.Delete(22);

tree.DisplayTree();

}

}

}

Text

Description automatically generated**Output**