**Міністерство освіти і науки України**

Національний технічний університет України

«Київський політехнічний інститут»

Факультет прикладної математики

Кафедра спеціалізованих комп’ютерних систем

**Лабораторна робота № 4**

З дисципліни

**«Операційні системи»**

Виконали

Студенти групи КВ-82

Павлов Богдан

Іванов Дмитро

Мумряк Сергій

Перевірив

Зайцев В. Г.

**Київ 2011**

**Постановка задачі:**

**Варіант 5.** Побудувати таблицю ідентифікаторів по методу бінарного дерева відповідно до варіанту 1 з наступними змінами: У якості ідентифікаторів використовувати одну букву.

**Код програми:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using System.Drawing;

namespace OS\_Lab4

{

class BalancedTree

{

Node root;

Panel localArea;

public BalancedTree(Panel P)

{

root = null;

localArea = P;

}

public virtual Node Search(IComparable data)

{

return SearchHelper(root, data);

}

protected virtual Node SearchHelper(Node current, IComparable data)

{

if (current == null)

return null; // node was not found

else

{

int result = current.Value.CompareTo(data);

if (result == 0)

// they are equal - we found the data

return current;

else if (result > 0)

{

// current.Value > n.Value

// therefore, if the data exists it is in current's left subtree

return SearchHelper(current.Left, data);

}

else // result < 0

{

// current.Value < n.Value

// therefore, if the data exists it is in current's right subtree

return SearchHelper(current.Right, data);

}

}

}

public String GetSearchPath(IComparable data)

{

if (this.Search(data) == null)

return null;

else

{

String str = new String(' ',1);

Node temp = root;

int result;

while (temp != null)

{

str = str + " " + temp.Value.ToString();

result = temp.Value.CompareTo(data);

if (result == 0)

break;

else if (result > 0)

temp = temp.Left;

else if (result < 0)

temp = temp.Right;

}

return str;

}

}

public virtual void Add(IComparable data)

{

// first, create a new Node

Node n = new Node(data);

int result;

// now, insert n into the tree

// trace down the tree until we hit a NULL

Node current = root, parent = null;

while (current != null)

{

result = current.Value.CompareTo(n.Value);

if (result == 0)

// they are equal - inserting a duplicate - do nothing

return;

else if (result > 0)

{

// current.Value > n.Value

// therefore, n must be added to current's left subtree

current.HeightLeft++;

parent = current;

current = current.Left;

}

else if (result < 0)

{

// current.Value < n.Value

// therefore, n must be added to current's right subtree

current.HeightRight++;

parent = current;

current = current.Right;

}

}

// ok, at this point we have reached the end of the tree

//count++;

if (parent == null)

// the tree was empty

root = n;

else

{

result = parent.Value.CompareTo(n.Value);

if (result > 0)

// parent.Value > n.Value

// therefore, n must be added to parent's left subtree

parent.Left = n;

else if (result < 0)

// parent.Value < n.Value

// therefore, n must be added to parent's right subtree

parent.Right = n;

}

//localConsole.AppendText("-----------------------------\n");

//Print();

Balance();

}

public virtual void Delete(IComparable data)

{

Node parent, temp;

int result = root.Value.CompareTo(data);

if (result == 0)

{

root = this.DeleteHelper(root);

}

else

{

temp = root;

do

{

parent = temp;

result = parent.Value.CompareTo(data);

if ((result > 0) && (parent.Left != null))

{

temp = parent.Left;

parent.HeightLeft--;

}

else if ((result < 0) && (parent.Right != null))

{

temp = parent.Right;

parent.HeightRight--;

}

}

while (temp.Value.CompareTo(data) != 0);

if (temp == parent.Left)

parent.Left = this.DeleteHelper(temp);

else if (temp == parent.Right)

parent.Right = this.DeleteHelper(temp);

}

Balance();

}

protected Node DeleteHelper(Node current)

{

if ((current.Left == null) && (current.Right == null))

current = null;

else if (current.HeightLeft >= current.HeightRight)

{

current = this.RightRotateHelper(current);

current.HeightRight--;

current.Right = current.Right.Right;

}

else

{

current = this.LeftRotateHelper(current);

current.HeightLeft--;

current.Left = current.Left.Left;

}

return current;

}

public void RightRotate()

{

root = this.RightRotateHelper(root);

}

protected Node RightRotateHelper(Node current)

{

Node temp, parent;

if (current.Left.Right == null) // left subtree has no right child -- малый правый поворот

{

temp = current.Left;

temp.Right = current;

}

else // left subtree has right child -- большой правый поворот

{

parent = current.Left;

while (parent.Right.Right != null)

{

parent.HeightRight--;

parent = parent.Right;

}

temp = parent.Right;

parent.HeightRight = temp.HeightLeft;//\*\*

parent.Right = temp.Left;//\*\*

temp.Right = current;

temp.Left = current.Left;

}

current.Left = null;

temp.HeightRight = current.HeightRight + 1;

temp.HeightLeft = current.HeightLeft - 1;

current.HeightLeft = 0;

return temp;

}

public void LeftRotate()

{

root = LeftRotateHelper(root);

}

protected Node LeftRotateHelper(Node current)

{

Node temp, parent;

if (current.Right.Left == null)

{

temp = current.Right;

temp.Left = current;

}

else

{

parent = current.Right;

while (parent.Left.Left != null)

{

parent.HeightLeft--;

parent = parent.Left;

}

temp = parent.Left;

parent.HeightLeft = temp.HeightRight;

parent.Left = temp.Right;

temp.Left = current;

temp.Right = current.Right;

}

current.Right = null;

temp.HeightLeft = current.HeightLeft + 1;

temp.HeightRight = current.HeightRight - 1;

current.HeightRight = 0;

return temp;

}

public void Balance()

{

root = BalanceHelper(root);

}

protected Node BalanceHelper(Node current)

{

while ((current.HeightLeft - current.HeightRight < -1) || (current.HeightLeft - current.HeightRight > 1))

{

if (current.HeightLeft > current.HeightRight)

current = RightRotateHelper(current);

else if (current.HeightLeft < current.HeightRight)

current = LeftRotateHelper(current);

}

if (current.Right != null)

current.Right = BalanceHelper(current.Right);

if (current.Left != null)

current.Left = BalanceHelper(current.Left);

return current;

}

public void Print()

{

localArea.Controls.Clear();

PrintHelper(root, new Point(500,40), 500, Color.Red);

}

protected void PrintHelper(Node current, Point P, int d, Color Col)

{

if (current.Left != null)

PrintHelper(current.Left, new Point(P.X-(int)(d/1.9),P.Y+40),d/2, Color.Green);

Label l = new Label();

l.Text = current.Value.ToString() + "(" + current.HeightLeft + "," + current.HeightRight + ")";

l.ForeColor = Col;

l.Size = new Size(7\*l.Text.Length, 23);

l.Location = P;

localArea.Controls.Add(l);

if (current.Right != null)

PrintHelper(current.Right, new Point(P.X + (int)(d/1.9), P.Y + 40), d/2, Color.Blue);

}

}

}

