Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Фізико-технічний інститут

**ЛАБОРАТОРНА РОБОТА №5**

З предмету

«Алгоритми та структури даних»

Виконав:

Студент ФІ-23 групи

Булавінцев Юрій

Київ 2024

***Варіант № 1***

**Мета роботи**: отримати навики застосування двійкових дерев,

реалізувати основні операції над деревами: обхід дерев, включення,

виключення та пошук вузлів.

***Завдання для лабораторної роботи №5***

1. Побудувати двійкове дерево пошуку з букв рядка, що вводиться.

Вивести його на екран у вигляді дерева. Знайти букви, що зустрічаються

більше одного разу. Видалити з дерева ці літери. Вивести елементи дерева, що залишилися, при його постфіксному обході.

**1. Вирішення**

using System;

using System.Collections.Generic;

namespace lab5

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введіть:");

string input = Console.ReadLine();

Tree tree = new Tree();

foreach (char c in input)

{

tree.insertNode(c.ToString());

}

Console.WriteLine("Дерево:");

tree.printTree();

Console.WriteLine("Видаляємо повтори:");

tree.removeDuplicates();

tree.printTree();

Console.WriteLine("Пост обхід:");

tree.postOrderTraversal();

Console.ReadLine();

}

}

class Node

{

public string Value { get; set; }

public Node LeftChild { get; set; }

public Node RightChild { get; set; }

public Node(string value)

{

Value = value;

LeftChild = null;

RightChild = null;

}

}

class Tree

{

private Node rootNode;

public Tree()

{

rootNode = null;

}

public void insertNode(string value)

{

Node newNode = new Node(value);

if (rootNode == null)

{

rootNode = newNode;

}

else

{

Node currentNode = rootNode;

Node parentNode;

while (true)

{

parentNode = currentNode;

if (string.Compare(value, currentNode.Value) < 0)

{

currentNode = currentNode.LeftChild;

if (currentNode == null)

{

parentNode.LeftChild = newNode;

return;

}

}

else if (string.Compare(value, currentNode.Value) > 0)

{

currentNode = currentNode.RightChild;

if (currentNode == null)

{

parentNode.RightChild = newNode;

return;

}

}

else

{

return;

}

}

}

}

public void removeDuplicates()

{

Dictionary<string, int> valueCount = new Dictionary<string, int>();

countValues(rootNode, valueCount);

foreach (var item in valueCount)

{

if (item.Value > 1)

{

deleteNode(item.Key);

}

}

}

private void countValues(Node node, Dictionary<string, int> valueCount)

{

if (node == null) return;

if (valueCount.ContainsKey(node.Value))

{

valueCount[node.Value]++;

}

else

{

valueCount[node.Value] = 1;

}

countValues(node.LeftChild, valueCount);

countValues(node.RightChild, valueCount);

}

public bool deleteNode(string value)

{

rootNode = deleteRec(rootNode, value);

return true;

}

private Node deleteRec(Node root, string key)

{

if (root == null) return root;

if (string.Compare(key, root.Value) < 0)

{

root.LeftChild = deleteRec(root.LeftChild, key);

}

else if (string.Compare(key, root.Value) > 0)

{

root.RightChild = deleteRec(root.RightChild, key);

}

else

{

if (root.LeftChild == null)

{

return root.RightChild;

}

else if (root.RightChild == null)

{

return root.LeftChild;

}

root.Value = minValue(root.RightChild);

root.RightChild = deleteRec(root.RightChild, root.Value);

}

return root;

}

private string minValue(Node root)

{

string minv = root.Value;

while (root.LeftChild != null)

{

minv = root.LeftChild.Value;

root = root.LeftChild;

}

return minv;

}

public void postOrderTraversal()

{

postOrderTraversal(rootNode);

}

private void postOrderTraversal(Node node)

{

if (node == null) return;

postOrderTraversal(node.LeftChild);

postOrderTraversal(node.RightChild);

Console.Write(node.Value + " ");

}

public void printTree()

{

if (rootNode == null)

{

Console.WriteLine("Tree is empty.");

return;

}

Stack<Node> globalStack = new Stack<Node>();

globalStack.Push(rootNode);

int gaps = 32;

bool isRowEmpty = false;

string sep = "------";

Console.WriteLine(sep);

while (!isRowEmpty)

{

Stack<Node> localStack = new Stack<Node>();

isRowEmpty = true;

for (int j = 0; j < gaps; j++)

{

Console.Write(" ");

}

while (globalStack.Count > 0)

{

Node temp = globalStack.Pop();

if (temp != null)

{

Console.Write(temp.Value);

localStack.Push(temp.LeftChild);

localStack.Push(temp.RightChild);

if (temp.LeftChild != null || temp.RightChild != null)

{

isRowEmpty = false;

}

}

else

{

Console.Write("\_\_\_");

localStack.Push(null);

localStack.Push(null);

}

for (int j = 0; j < gaps \* 2 - 2; j++)

{

Console.Write(" ");

}

}

Console.WriteLine();

gaps /= 2;

while (localStack.Count > 0)

{

globalStack.Push(localStack.Pop());

}

Console.WriteLine(sep);

}

}

}

}