Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

# Лабораторная работа " Бинарные деревья."

Выполнил студент гр. РИС-24-3б

Жиряков Леонид Антонович

Проверил:

Доцент кафедры ИТАС   
Ольга Андреевна Полякова

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

г. Пермь, 2024

Вариант 8.

Постановка задачи:

Общая:

Требуется реализовать алгоритмы для собственного варианта бинарного дерева поиска, имеющего не менее трёх уровней.

Алгоритмы:

1. Необходимо реализовать функции для редактирования дерева:

- Вставка узла.

- Удаление узла.

- Поиск элемента по ключу.

2. Реализовать алгоритмы обхода дерева:

2.1 Прямой

2.2 Симметричный

2.3 Обратный

3. Реализовать алгоритм балансировки дерева.

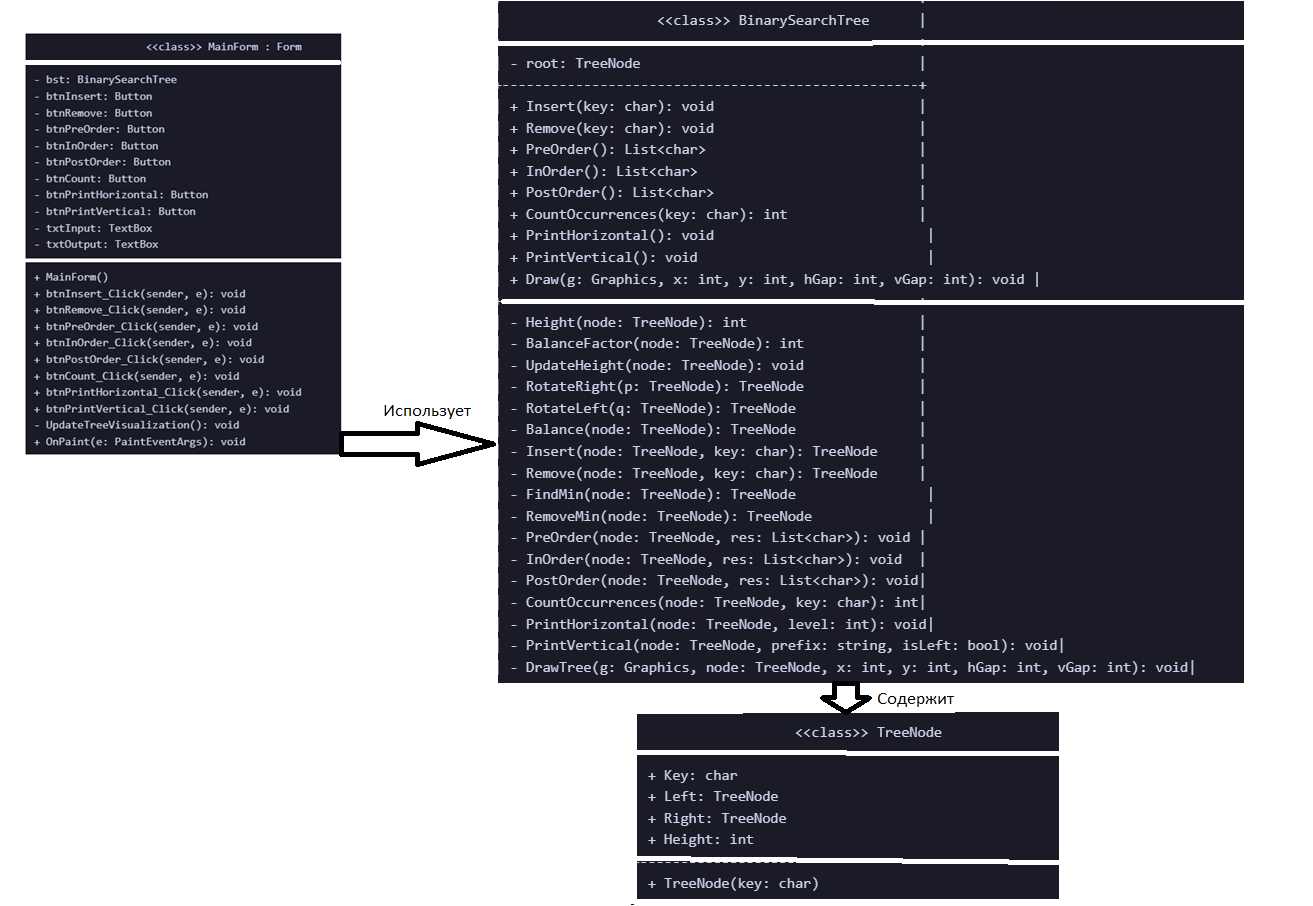
4. Реализовать вертикальную и горизонтальную печать.

5. Визуализация дерева должна быть выполнена с использованием любой доступной графической библиотеки – SFML, SDL, OpenGL и подобных.

Персональная:

Тип информационного поля char. Найти количество элементов с заданным ключом.

UML – диаграмма:



Программная реализация:

MainForm.cs:

using System;

using System.Drawing;

using System.Windows.Forms;

namespace BinarySearchTreeApp

{

public partial class MainForm : Form

{

private BinarySearchTree bst;

public MainForm()

{

InitializeComponent();

bst = new BinarySearchTree();

// Для примера добавим несколько узлов

foreach (char c in "FBADCEGIH")

bst.Insert(c);

UpdateTreeVisualization();

}

private void btnInsert\_Click(object sender, EventArgs e)

{

if (!string.IsNullOrEmpty(txtInput.Text))

{

char key = txtInput.Text[0];

bst.Insert(key);

txtInput.Clear();

UpdateTreeVisualization();

}

}

private void btnRemove\_Click(object sender, EventArgs e)

{

if (!string.IsNullOrEmpty(txtInput.Text))

{

char key = txtInput.Text[0];

bst.Remove(key);

txtInput.Clear();

UpdateTreeVisualization();

}

}

private void btnPreOrder\_Click(object sender, EventArgs e)

{

var list = bst.PreOrder();

txtOutput.Text = "Прямой обход: " + string.Join(" ", list);

}

private void btnInOrder\_Click(object sender, EventArgs e)

{

var list = bst.InOrder();

txtOutput.Text = "Симметричный обход: " + string.Join(" ", list);

}

private void btnPostOrder\_Click(object sender, EventArgs e)

{

var list = bst.PostOrder();

txtOutput.Text = "Обратный обход: " + string.Join(" ", list);

}

private void btnCount\_Click(object sender, EventArgs e)

{

if (!string.IsNullOrEmpty(txtInput.Text))

{

char key = txtInput.Text[0];

int count = bst.CountOccurrences(key);

txtOutput.Text = $"Количество '{key}': {count}";

}

}

private void btnPrintHorizontal\_Click(object sender, EventArgs e)

{

txtOutput.Clear();

using (var sw = new System.IO.StringWriter())

{

Console.SetOut(sw);

bst.PrintHorizontal();

txtOutput.Text = sw.ToString();

}

}

private void btnPrintVertical\_Click(object sender, EventArgs e)

{

txtOutput.Clear();

using (var sw = new System.IO.StringWriter())

{

Console.SetOut(sw);

bst.PrintVertical();

txtOutput.Text = sw.ToString();

}

}

private void UpdateTreeVisualization()

{

this.Invalidate();

}

protected override void OnPaint(PaintEventArgs e)

{

base.OnPaint(e);

bst.Draw(e.Graphics, this.ClientSize.Width / 2, 50, 200, 100);

}

}

}

Program.cs:

using System;

using System.Windows.Forms;

namespace BinarySearchTreeApp

{

static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForm());

}

}

}

MainForm.Designer.cs:

namespace BinarySearchTreeApp

{

partial class MainForm

{

private System.ComponentModel.IContainer components = null;

private System.Windows.Forms.Button btnInsert;

private System.Windows.Forms.Button btnRemove;

private System.Windows.Forms.Button btnPreOrder;

private System.Windows.Forms.Button btnInOrder;

private System.Windows.Forms.Button btnPostOrder;

private System.Windows.Forms.Button btnCount;

private System.Windows.Forms.Button btnPrintHorizontal;

private System.Windows.Forms.Button btnPrintVertical;

private System.Windows.Forms.TextBox txtInput;

private System.Windows.Forms.TextBox txtOutput;

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

private void InitializeComponent()

{

this.btnInsert = new System.Windows.Forms.Button();

this.btnRemove = new System.Windows.Forms.Button();

this.btnPreOrder = new System.Windows.Forms.Button();

this.btnInOrder = new System.Windows.Forms.Button();

this.btnPostOrder = new System.Windows.Forms.Button();

this.btnCount = new System.Windows.Forms.Button();

this.btnPrintHorizontal = new System.Windows.Forms.Button();

this.btnPrintVertical = new System.Windows.Forms.Button();

this.txtInput = new System.Windows.Forms.TextBox();

this.txtOutput = new System.Windows.Forms.TextBox();

this.SuspendLayout();

//

// btnInsert

//

this.btnInsert.Location = new System.Drawing.Point(12, 12);

this.btnInsert.Name = "btnInsert";

this.btnInsert.Size = new System.Drawing.Size(75, 23);

this.btnInsert.TabIndex = 0;

this.btnInsert.Text = "Вставить";

this.btnInsert.UseVisualStyleBackColor = true;

this.btnInsert.Click += new System.EventHandler(this.btnInsert\_Click);

//

// btnRemove

//

this.btnRemove.Location = new System.Drawing.Point(12, 41);

this.btnRemove.Name = "btnRemove";

this.btnRemove.Size = new System.Drawing.Size(75, 23);

this.btnRemove.TabIndex = 1;

this.btnRemove.Text = "Удалить";

this.btnRemove.UseVisualStyleBackColor = true;

this.btnRemove.Click += new System.EventHandler(this.btnRemove\_Click);

//

// btnPreOrder

//

this.btnPreOrder.Location = new System.Drawing.Point(12, 70);

this.btnPreOrder.Name = "btnPreOrder";

this.btnPreOrder.Size = new System.Drawing.Size(120, 23);

this.btnPreOrder.TabIndex = 2;

this.btnPreOrder.Text = "Прямой обход";

this.btnPreOrder.UseVisualStyleBackColor = true;

this.btnPreOrder.Click += new System.EventHandler(this.btnPreOrder\_Click);

//

// btnInOrder

//

this.btnInOrder.Location = new System.Drawing.Point(12, 99);

this.btnInOrder.Name = "btnInOrder";

this.btnInOrder.Size = new System.Drawing.Size(120, 23);

this.btnInOrder.TabIndex = 3;

this.btnInOrder.Text = "Симметричный обход";

this.btnInOrder.UseVisualStyleBackColor = true;

this.btnInOrder.Click += new System.EventHandler(this.btnInOrder\_Click);

//

// btnPostOrder

//

this.btnPostOrder.Location = new System.Drawing.Point(12, 128);

this.btnPostOrder.Name = "btnPostOrder";

this.btnPostOrder.Size = new System.Drawing.Size(120, 23);

this.btnPostOrder.TabIndex = 4;

this.btnPostOrder.Text = "Обратный обход";

this.btnPostOrder.UseVisualStyleBackColor = true;

this.btnPostOrder.Click += new System.EventHandler(this.btnPostOrder\_Click);

//

// btnCount

//

this.btnCount.Location = new System.Drawing.Point(12, 157);

this.btnCount.Name = "btnCount";

this.btnCount.Size = new System.Drawing.Size(120, 23);

this.btnCount.TabIndex = 5;

this.btnCount.Text = "Подсчитать вхождения";

this.btnCount.UseVisualStyleBackColor = true;

this.btnCount.Click += new System.EventHandler(this.btnCount\_Click);

//

// btnPrintHorizontal

//

this.btnPrintHorizontal.Location = new System.Drawing.Point(12, 186);

this.btnPrintHorizontal.Name = "btnPrintHorizontal";

this.btnPrintHorizontal.Size = new System.Drawing.Size(120, 23);

this.btnPrintHorizontal.TabIndex = 6;

this.btnPrintHorizontal.Text = "Печать горизонтально";

this.btnPrintHorizontal.UseVisualStyleBackColor = true;

this.btnPrintHorizontal.Click += new System.EventHandler(this.btnPrintHorizontal\_Click);

//

// btnPrintVertical

//

this.btnPrintVertical.Location = new System.Drawing.Point(12, 215);

this.btnPrintVertical.Name = "btnPrintVertical";

this.btnPrintVertical.Size = new System.Drawing.Size(120, 23);

this.btnPrintVertical.TabIndex = 7;

this.btnPrintVertical.Text = "Печать вертикально";

this.btnPrintVertical.UseVisualStyleBackColor = true;

this.btnPrintVertical.Click += new System.EventHandler(this.btnPrintVertical\_Click);

//

// txtInput

//

this.txtInput.Location = new System.Drawing.Point(138, 14);

this.txtInput.Name = "txtInput";

this.txtInput.Size = new System.Drawing.Size(50, 20);

this.txtInput.TabIndex = 8;

//

// txtOutput

//

this.txtOutput.Location = new System.Drawing.Point(12, 244);

this.txtOutput.Multiline = true;

this.txtOutput.Name = "txtOutput";

this.txtOutput.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;

this.txtOutput.Size = new System.Drawing.Size(400, 150);

this.txtOutput.TabIndex = 9;

//

// MainForm

//

this.ClientSize = new System.Drawing.Size(800, 450);

this.Controls.Add(this.txtOutput);

this.Controls.Add(this.txtInput);

this.Controls.Add(this.btnPrintVertical);

this.Controls.Add(this.btnPrintHorizontal);

this.Controls.Add(this.btnCount);

this.Controls.Add(this.btnPostOrder);

this.Controls.Add(this.btnInOrder);

this.Controls.Add(this.btnPreOrder);

this.Controls.Add(this.btnRemove);

this.Controls.Add(this.btnInsert);

this.Name = "MainForm";

this.Text = "Сбалансированное бинарное дерево поиска";

this.ResumeLayout(false);

this.PerformLayout();

}

}

}

BinarySearchTree.cs:

using System;

using System.Collections.Generic;

using System.Drawing;

namespace BinarySearchTreeApp

{

public class TreeNode

{

public char Key;

public TreeNode Left, Right;

public int Height;

public TreeNode(char key)

{

Key = key;

Left = Right = null;

Height = 1;

}

}

public class BinarySearchTree

{

private TreeNode root;

private int Height(TreeNode node) => node?.Height ?? 0;

private int BalanceFactor(TreeNode node) => Height(node.Right) - Height(node.Left);

private void UpdateHeight(TreeNode node)

{

node.Height = Math.Max(Height(node.Left), Height(node.Right)) + 1;

}

private TreeNode RotateRight(TreeNode p)

{

TreeNode q = p.Left;

p.Left = q.Right;

q.Right = p;

UpdateHeight(p);

UpdateHeight(q);

return q;

}

private TreeNode RotateLeft(TreeNode q)

{

TreeNode p = q.Right;

q.Right = p.Left;

p.Left = q;

UpdateHeight(q);

UpdateHeight(p);

return p;

}

private TreeNode Balance(TreeNode node)

{

UpdateHeight(node);

int balance = BalanceFactor(node);

if (balance == 2)

{

if (BalanceFactor(node.Right) < 0)

node.Right = RotateRight(node.Right);

return RotateLeft(node);

}

if (balance == -2)

{

if (BalanceFactor(node.Left) > 0)

node.Left = RotateLeft(node.Left);

return RotateRight(node);

}

return node;

}

public void Insert(char key)

{

root = Insert(root, key);

}

private TreeNode Insert(TreeNode node, char key)

{

if (node == null) return new TreeNode(key);

if (key < node.Key)

node.Left = Insert(node.Left, key);

else if (key > node.Key)

node.Right = Insert(node.Right, key);

else

return node; // дубликаты не добавляем

return Balance(node);

}

public void Remove(char key)

{

root = Remove(root, key);

}

private TreeNode FindMin(TreeNode node)

{

return node.Left == null ? node : FindMin(node.Left);

}

private TreeNode RemoveMin(TreeNode node)

{

if (node.Left == null)

return node.Right;

node.Left = RemoveMin(node.Left);

return Balance(node);

}

private TreeNode Remove(TreeNode node, char key)

{

if (node == null) return null;

if (key < node.Key)

node.Left = Remove(node.Left, key);

else if (key > node.Key)

node.Right = Remove(node.Right, key);

else

{

TreeNode left = node.Left;

TreeNode right = node.Right;

if (right == null) return left;

TreeNode min = FindMin(right);

min.Right = RemoveMin(right);

min.Left = left;

return Balance(min);

}

return Balance(node);

}

// Обходы

public List<char> PreOrder()

{

var res = new List<char>();

PreOrder(root, res);

return res;

}

private void PreOrder(TreeNode node, List<char> res)

{

if (node == null) return;

res.Add(node.Key);

PreOrder(node.Left, res);

PreOrder(node.Right, res);

}

public List<char> InOrder()

{

var res = new List<char>();

InOrder(root, res);

return res;

}

private void InOrder(TreeNode node, List<char> res)

{

if (node == null) return;

InOrder(node.Left, res);

res.Add(node.Key);

InOrder(node.Right, res);

}

public List<char> PostOrder()

{

var res = new List<char>();

PostOrder(root, res);

return res;

}

private void PostOrder(TreeNode node, List<char> res)

{

if (node == null) return;

PostOrder(node.Left, res);

PostOrder(node.Right, res);

res.Add(node.Key);

}

// Подсчёт вхождений

public int CountOccurrences(char key)

{

return CountOccurrences(root, key);

}

private int CountOccurrences(TreeNode node, char key)

{

if (node == null) return 0;

int count = (node.Key == key) ? 1 : 0;

return count + CountOccurrences(node.Left, key) + CountOccurrences(node.Right, key);

}

// Печать дерева в консоль (горизонтально)

public void PrintHorizontal()

{

PrintHorizontal(root, 0);

}

private void PrintHorizontal(TreeNode node, int level)

{

if (node == null) return;

PrintHorizontal(node.Right, level + 1);

Console.WriteLine(new string(' ', 4 \* level) + node.Key);

PrintHorizontal(node.Left, level + 1);

}

// Печать дерева в консоль (вертикально)

public void PrintVertical()

{

PrintVertical(root, "", false);

}

private void PrintVertical(TreeNode node, string prefix, bool isLeft)

{

if (node == null) return;

Console.WriteLine(prefix + (isLeft ? "├── " : "└── ") + node.Key);

PrintVertical(node.Left, prefix + (isLeft ? "│ " : " "), true);

PrintVertical(node.Right, prefix + (isLeft ? "│ " : " "), false);

}

// Визуализация (оставим как есть)

public void Draw(Graphics g, int x, int y, int hGap, int vGap)

{

DrawTree(g, root, x, y, hGap, vGap);

}

private void DrawTree(Graphics g, TreeNode node, int x, int y, int hGap, int vGap)

{

if (node == null) return;

if (node.Left != null)

{

g.DrawLine(Pens.Black, x, y, x - hGap, y + vGap);

DrawTree(g, node.Left, x - hGap, y + vGap, hGap / 2, vGap);

}

if (node.Right != null)

{

g.DrawLine(Pens.Black, x, y, x + hGap, y + vGap);

DrawTree(g, node.Right, x + hGap, y + vGap, hGap / 2, vGap);

}

g.FillEllipse(Brushes.White, x - 20, y - 20, 40, 40);

g.DrawEllipse(Pens.Black, x - 20, y - 20, 40, 40);

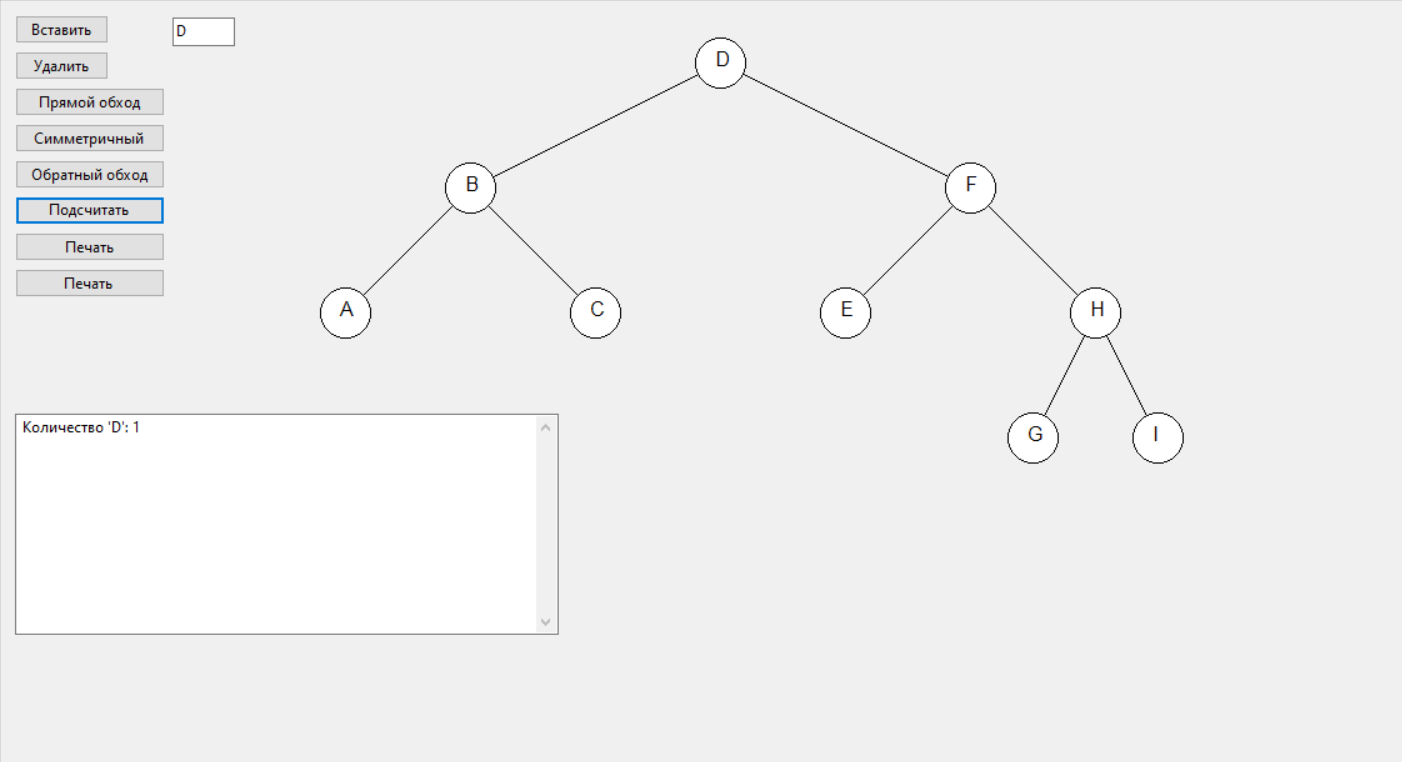
g.DrawString(node.Key.ToString(), new Font("Arial", 12), Brushes.Black, x - 7, y - 12);

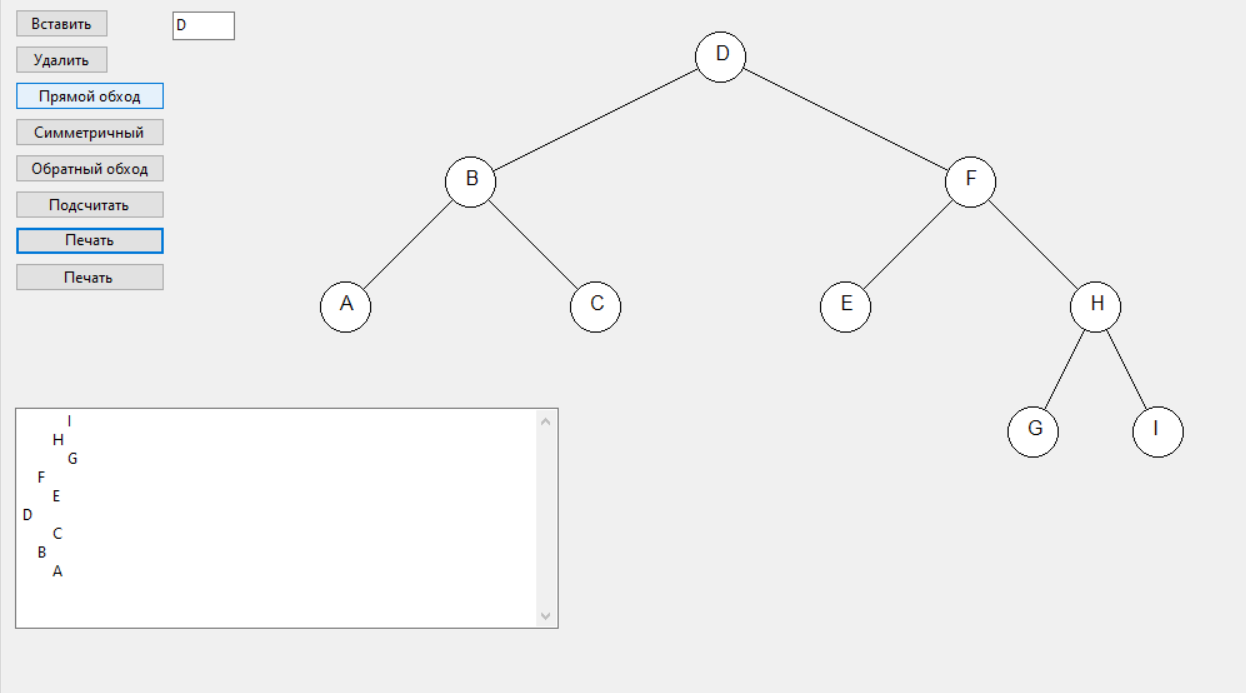
}

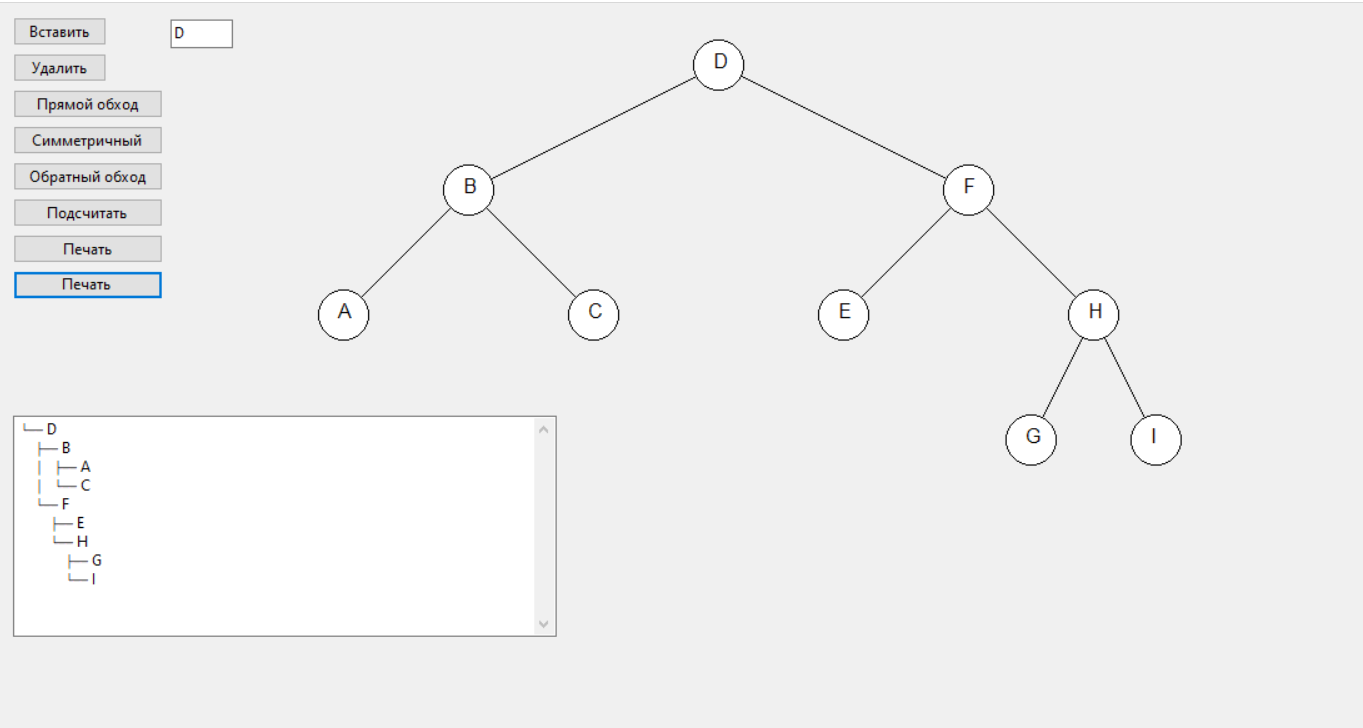
}

}

Результат работы:







Ссылка на GitHub:

https://github.com/LeonidZhir/-