**ЛАБОРАТОРНАЯ РАБОТА № 2  
«ИССЛЕДОВАНИЕ НЕЛИНЕЙНЫХ СТРУКТУР ДАННЫХ. AVL-ДЕРЕВЬЯ»**

**2.1 Цель работы**

Исследовать возможности применения AVL-деревьев – для хранения, поиска и обработки информации. Приобрести практические использования классов, реализующих AVL-деревья. Оценить эффективность использования AVL-деревья по сравнению с бинарными деревьями поиска.

**2.2 Постановка задачи**

1. В ходе выполнения настоящей лабораторной работы сначала необходимо ознакомиться с организацией и программной реализацией AVL-деревьев.

2. На одном из языков программирования (С++ или Object Pascal) в среде визуального программирования (С++ Builder или Delphi, соответственно), с иcпользованием классов, реализующих бинарные деревья поиска и AVL-деревья (для С++ файлы: classes.h, DATA.H и MYLIST.H; для ObjectPascal: файл Collection.pas) реализовать Windows-приложение, обеспечивающее выполнения следующих функций:

2.1. Построение бинарного дерева поиска и AVL-дерева на основании данных, расположенных в файле(имя файла и имя ключевого поля определяются вариантом задания – таблица 2.1), содержащем построчно записи разделенные символом табуляции;

2.2 Отображение на визуальной форме бинарного дерева поиска и AVL-дерева, используя компонент TTreeView;

2.3. Предоставление интерфейса пользователю для выполнения операций добавления, удаления, изменения и поиска (по ключевому полю) элементов обоих типов деревьев, с отображением результатов выполнения операций на визуальной форме;

2.4. Отображение времени выполнения операций добавления, удаления, изменения и поиска данных по заданному пользователем значению ключевого поля;

2.5. Предусмотреть возможность перестроения деревьев из входного файла с ограничением числа добавляемых элементов (максимальное число элементов должно определяться пользователем – поле ввода на визуальной форме).

2.6. Отображение на визуальной форме актуальной информации о списке и дереве: количество элементов(узлов), количество листьев, глубину дерева.

3. С использованием разработанной программы выполнить исследования структур данных бинарного дерева поиска и AVL-дерева:

3.1. Построить бинарное дерево поиска и AVL- дерево для первых N1 строк таблицы (N1 задается в соответствии с вариантом задания – таблица 1);

3.2. Выполнить по 5 раз операции добавления, удаления и поиска информации (по случайным значениям ключевого поля), фиксируя в отчете время выполнения операций;

3.3. Вычислить среднее время выполнения операций добавления, удаления и поиска информации (по ключевому полю) зафиксированных в п. 3.2.

4. Повторять пп. 3.1 – 3.3 для значений N2, N3, N4 и N5 (N2 – N5 задается в соответствии с вариантом задания – таблица 2.1), фиксируя получаемые значения времени в таблице.

5. На основании данных, полученных при выполнении пп. 3 – 4 построить графики зависимости среднего времени, затрачиваемого на выполнение каждой операции (добавление удаление поиск) от количества элементов N для бинарного дерева поиска и AVL-дерева.

6. Сформулировать выводы.

7. Оформить отчет.

**2.3 Индивидуальный вариант**

Таблица 2.1 – Вариант задания

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № Вар | Файл данных | Ключевое поле | N1 | N2 | N3 | N4 | N5 |
| 3 | Table22.txt | Телефон | 40 | 150 | 1300 | 3000 | 8000 |

**2.4 Ход работы**

2.4.1 Была разработана программа, решающая поставленную задачу (приложение А).

2.4.2 Пользовательский интерфейс приложения представлен на рисунке 2.1.

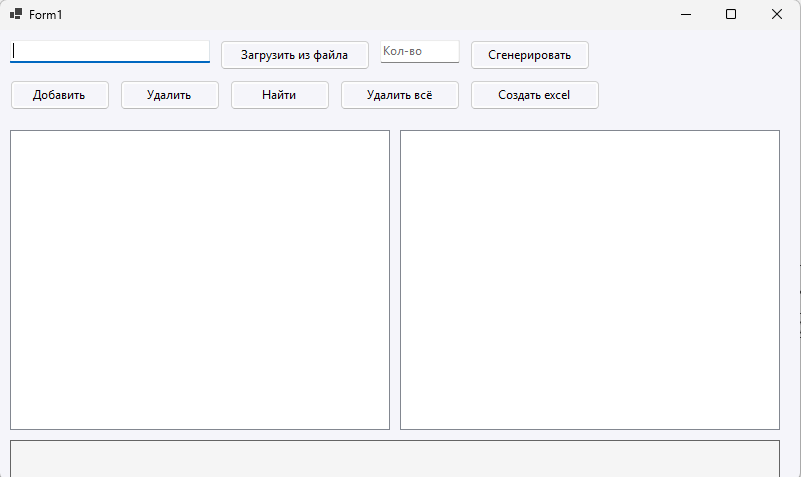


Рисунок 2.1 – Пользовательский интерфейс

Интерфейс сверху содержит панель управления, где можно производить действия с деревьями. В средней части находится блок для отображения деревьев, данные в деревьях одинаковые. Снизу находится блок с информацией, здесь показано последнее действие, а также состояние деревьев на данный момент.

2.4.3 Была сформирована таблица о времени выполнения операций над деревьями (таблица 2.2).

Таблица 2.2 – Время выполнения операций над деревьями

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер | BST - Добавление | BST - Удаление | BST - Поиск | AVL - Добавление | AVL - Удаление | AVL - Поиск |
| 40 |  |  |  |  |  |  |
| 1 | 0,0249 | 0,2893 | 0,1232 | 0,0152 | 0,225 | 0,118 |
| 2 | 0,0013 | 0,0012 | 0,0009 | 0,0012 | 0,0015 | 0,0005 |
| 3 | 0,0015 | 0,0012 | 0,0008 | 0,0011 | 0,0012 | 0,0002 |
| 4 | 0,0015 | 0,0011 | 0,0007 | 0,0011 | 0,0011 | 0,0005 |
| 5 | 0,0014 | 0,0014 | 0,0004 | 0,0012 | 0,0011 | 0,0003 |
| Среднее | 0,00612 | 0,05884 | 0,0252 | 0,00396 | 0,04598 | 0,0239 |
| 150 |  |  |  |  |  |  |
| 1 | 0,0013 | 0,001 | 0,0012 | 0,0013 | 0,0011 | 0,0005 |
| 2 | 0,002 | 0,0017 | 0,001 | 0,0013 | 0,0015 | 0,0007 |
| 3 | 0,0018 | 0,0019 | 0,0005 | 0,0011 | 0,0013 | 0,0006 |
| 4 | 0,0017 | 0,002 | 0,0008 | 0,0013 | 0,0014 | 0,0006 |
| 5 | 0,0016 | 0,0016 | 0,0009 | 0,0013 | 0,0012 | 0,0006 |
| Среднее | 0,00168 | 0,00164 | 0,00088 | 0,00126 | 0,0013 | 0,0006 |
| 1300 |  |  |  |  |  |  |
| 1 | 0,0023 | 0,0013 | 0,0015 | 0,0017 | 0,0017 | 0,001 |
| 2 | 0,0027 | 0,0022 | 0,0014 | 0,0018 | 0,0017 | 0,001 |
| 3 | 0,0027 | 0,0023 | 0,0008 | 0,0017 | 0,0018 | 0,001 |
| 4 | 0,0029 | 0,0018 | 0,0017 | 0,0016 | 0,002 | 0,0009 |
| 5 | 0,0032 | 0,0019 | 0,0017 | 0,0016 | 0,0017 | 0,001 |
| Среднее | 0,00276 | 0,0019 | 0,00142 | 0,00168 | 0,00178 | 0,00098 |
| 3000 |  |  |  |  |  |  |
| 1 | 0,0025 | 0,002 | 0,0019 | 0,0017 | 0,002 | 0,0013 |
| 2 | 0,004 | 0,0019 | 0,0017 | 0,002 | 0,0021 | 0,0013 |
| 3 | 0,0022 | 0,0026 | 0,0021 | 0,0016 | 0,002 | 0,0009 |
| 4 | 0,0031 | 0,002 | 0,001 | 0,0018 | 0,0021 | 0,0008 |
| 5 | 0,0033 | 0,0019 | 0,0012 | 0,0016 | 0,002 | 0,001 |
| Среднее | 0,00302 | 0,00208 | 0,00158 | 0,00174 | 0,00204 | 0,00106 |
| 8000 |  |  |  |  |  |  |

Продолжение таблицы 2.2 – Время выполнения операций над деревьями

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер | BST - Добавление | BST - Удаление | BST - Поиск | AVL - Добавление | AVL - Удаление | AVL - Поиск |
| 1 | 0,0033 | 0,003 | 0,0027 | 0,0019 | 0,0022 | 0,0012 |
| 2 | 0,0034 | 0,0028 | 0,0015 | 0,002 | 0,002 | 0,0014 |
| 3 | 0,0034 | 0,0031 | 0,0024 | 0,0019 | 0,0021 | 0,0011 |
| 4 | 0,0046 | 0,004 | 0,002 | 0,0025 | 0,0028 | 0,0015 |
| 5 | 0,003 | 0,0035 | 0,0019 | 0,0024 | 0,003 | 0,002 |
| Среднее | 0,00354 | 0,00328 | 0,0021 | 0,00214 | 0,00242 | 0,00144 |

2.4.4 Полученные данные отобразили на графиках (рисунки 2.2–2.4).

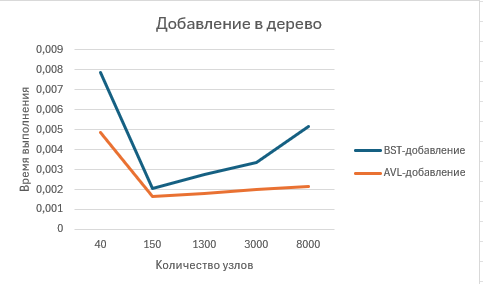


Рисунок 2.2 – Графики времени операции вставки

Как видно на рисунке 2.2, время вставки в AVL-дерево ниже. BST-дерево при большом наборе данных, может иметь достаточно большую глубину, а AVL-дерево, благодаря операциям балансировки имеет более низкую глубину, что и способствует более быстрым операциям.

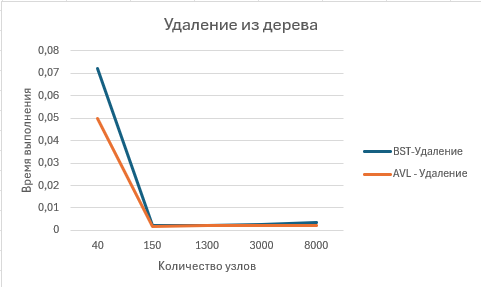


Рисунок 2.3 – Графики времени удаления из дерева

Как видно на рисунке 2.3, время вставки в AVL-дерево ниже. Как в случае с вставкой, скорость операций выше из-за меньшей глубины.

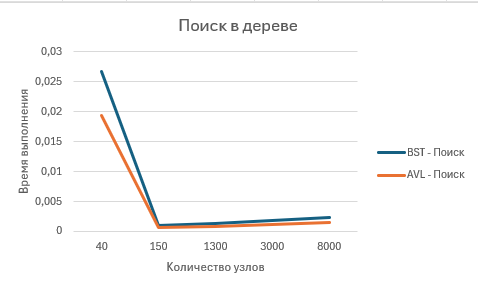


Рисунок 2.4 – Графики времени операции поиска

Как видно на рисунке 2.4, операции поиска в AVL-дереве выполняются быстрее.

**Вывод**

В ходе выполнения работы были реализованы и протестированы структуры данных — бинарное дерево поиска (BST) и сбалансированное AVL-дерево. На основе проведённых экспериментов можно сделать следующие выводы:

AVL-дерево демонстрирует более высокую производительность по сравнению с BST при операциях вставки, удаления и поиска, особенно на больших объёмах данных.

Глубина дерева в AVL остаётся логарифмической благодаря балансировке, что обеспечивает стабильное время выполнения операций.

BST без балансировки может деградировать в линейную структуру, что приводит к увеличению времени выполнения операций.

Графики и таблицы подтверждают, что при росте количества узлов AVL-дерево сохраняет эффективность, тогда как BST теряет производительность.

Разработанное приложение позволило визуализировать деревья, проводить операции и фиксировать метрики, что обеспечило наглядность и точность исследования.

**ПРИЛОЖЕНИЕ А**

**КОД ПРОГРАММЫ**

Файл BSTreeNode

public class BSTreeNode

{

public BSTreeNode(string phone)

{

Phone = phone;

}

public string Phone { get; set; }

public BSTreeNode Left { get; set; }

public BSTreeNode Right { get; set; }

}

Файл BinarySearchTree

public class BinarySearchTree

{

public BSTreeNode Root { get; private set; }

public void Add(string phone)

{

if (!IsValidPhone(phone)) return;

Root = Add(Root, phone);

}

private BSTreeNode Add(BSTreeNode node, string phone)

{

if (node == null) return new BSTreeNode(phone);

if (string.Compare(phone, node.Phone) < 0)

node.Left = Add(node.Left, phone);

else if (string.Compare(phone, node.Phone) > 0)

node.Right = Add(node.Right, phone);

return node;

}

public bool Delete(string phone)

{

if (!IsValidPhone(phone)) return false;

var found = false;

Root = Delete(Root, phone, ref found);

return found;

}

private BSTreeNode Delete(BSTreeNode node, string phone, ref bool found)

{

if (node == null) return null;

var cmp = string.Compare(phone, node.Phone);

if (cmp < 0)

{

node.Left = Delete(node.Left, phone, ref found);

}

else if (cmp > 0)

{

node.Right = Delete(node.Right, phone, ref found);

}

else

{

found = true;

if (node.Left == null) return node.Right;

if (node.Right == null) return node.Left;

var min = FindMin(node.Right);

node.Phone = min.Phone;

node.Right = Delete(node.Right, min.Phone, ref found);

}

return node;

}

private BSTreeNode FindMin(BSTreeNode node)

{

while (node.Left != null)

node = node.Left;

return node;

}

public BSTreeNode Find(string phone)

{

return Find(Root, phone);

}

private BSTreeNode Find(BSTreeNode node, string phone)

{

if (node == null || node.Phone == phone) return node;

return string.Compare(phone, node.Phone) < 0

? Find(node.Left, phone)

: Find(node.Right, phone);

}

public int Count()

{

return Count(Root);

}

public int Leaves()

{

return Leaves(Root);

}

public int Depth()

{

return Depth(Root);

}

private int Count(BSTreeNode node)

{

return node == null ? 0 : 1 + Count(node.Left) + Count(node.Right);

}

private int Leaves(BSTreeNode node)

{

return node == null ? 0 :

node.Left == null && node.Right == null ? 1 :

Leaves(node.Left) + Leaves(node.Right);

}

private int Depth(BSTreeNode node)

{

return node == null ? 0 : 1 + Math.Max(Depth(node.Left), Depth(node.Right));

}

private bool IsValidPhone(string phone)

{

return phone.StartsWith("7") && phone.All(char.IsDigit) && phone.Length >= 10;

}

}

Файл AvlTreeNode

public class AvlTreeNode

{

public int Height;

public AvlTreeNode Left, Right;

public string Phone;

public AvlTreeNode(string phone)

{

Phone = phone;

Height = 1;

}

}

Файл AvlTree

public class AvlTree

{

public AvlTreeNode Root { get; private set; }

public void Add(string phone)

{

if (!IsValidPhone(phone)) return;

Root = Insert(Root, phone);

}

public bool Delete(string phone)

{

if (!IsValidPhone(phone)) return false;

var found = false;

Root = Delete(Root, phone, ref found);

return found;

}

public AvlTreeNode Find(string phone)

{

return Find(Root, phone);

}

public int Count()

{

return Count(Root);

}

public int Leaves()

{

return Leaves(Root);

}

public int Depth()

{

return Height(Root);

}

private AvlTreeNode Insert(AvlTreeNode node, string phone)

{

if (node == null) return new AvlTreeNode(phone);

var cmp = string.Compare(phone, node.Phone);

if (cmp < 0)

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

else if (cmp > 0)

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

else

return node;

UpdateHeight(node);

return Balance(node);

}

private AvlTreeNode Delete(AvlTreeNode node, string phone, ref bool found)

{

if (node == null) return null;

var cmp = string.Compare(phone, node.Phone);

if (cmp < 0)

{

node.Left = Delete(node.Left, phone, ref found);

}

else if (cmp > 0)

{

node.Right = Delete(node.Right, phone, ref found);

}

else

{

found = true;

if (node.Left == null) return node.Right;

if (node.Right == null) return node.Left;

var min = FindMin(node.Right);

node.Phone = min.Phone;

node.Right = Delete(node.Right, min.Phone, ref found);

}

UpdateHeight(node);

return Balance(node);

}

private AvlTreeNode Find(AvlTreeNode node, string phone)

{

if (node == null || node.Phone == phone) return node;

return string.Compare(phone, node.Phone) < 0

? Find(node.Left, phone)

: Find(node.Right, phone);

}

private AvlTreeNode FindMin(AvlTreeNode node)

{

while (node.Left != null)

node = node.Left;

return node;

}

private int Count(AvlTreeNode node)

{

return node == null ? 0 : 1 + Count(node.Left) + Count(node.Right);

}

private int Leaves(AvlTreeNode node)

{

return node == null ? 0 :

node.Left == null && node.Right == null ? 1 :

Leaves(node.Left) + Leaves(node.Right);

}

private int Height(AvlTreeNode node)

{

return node?.Height ?? 0;

}

private void UpdateHeight(AvlTreeNode node)

{

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

}

private int BalanceFactor(AvlTreeNode node)

{

return Height(node.Left) - Height(node.Right);

}

private AvlTreeNode Balance(AvlTreeNode node)

{

var balance = BalanceFactor(node);

if (balance > 1)

{

if (BalanceFactor(node.Left) < 0)

node.Left = RotateLeft(node.Left);

return RotateRight(node);

}

if (balance < -1)

{

if (BalanceFactor(node.Right) > 0)

node.Right = RotateRight(node.Right);

return RotateLeft(node);

}

return node;

}

private AvlTreeNode RotateRight(AvlTreeNode y)

{

var x = y.Left;

var T2 = x.Right;

x.Right = y;

y.Left = T2;

UpdateHeight(y);

UpdateHeight(x);

return x;

}

private AvlTreeNode RotateLeft(AvlTreeNode x)

{

var y = x.Right;

var T2 = y.Left;

y.Left = x;

x.Right = T2;

UpdateHeight(x);

UpdateHeight(y);

return y;

}

private bool IsValidPhone(string phone)

{

return phone.StartsWith("7") && phone.All(char.IsDigit) && phone.Length >= 10;

}

}

Файл TreeToExcel

public class TreeToExcel

{

private double Measure(Action action)

{

var sw = Stopwatch.StartNew();

action();

sw.Stop();

return sw.Elapsed.TotalMilliseconds;

}

private string GetProjectDirectory()

{

var path = AppContext.BaseDirectory;

while (!Directory.EnumerateFiles(path).Any(f => f.EndsWith(".csproj")))

path = Directory.GetParent(path)!.FullName;

return path;

}

public void RunBenchmarksAndExportToExcel()

{

var sizes = new[] { 40, 150, 1300, 3000, 8000 };

var rand = new Random();

ExcelPackage.License.SetNonCommercialOrganization("My Noncommercial organization");

var filePath = Path.Combine(GetProjectDirectory(), "TreeBenchmark.xlsx");

var data1 = Enumerable.Range(0, 1000)

.Select(\_ => rand.NextInt64(70000000000, 79999999999).ToString())

.ToList();

using (var package = new ExcelPackage())

{

var sheet = package.Workbook.Worksheets.Add("Benchmark");

var row = 1;

sheet.Cells[row, 1].Value = "Размер";

sheet.Cells[row, 2].Value = "BST - Добавление";

sheet.Cells[row, 3].Value = "BST - Удаление";

sheet.Cells[row, 4].Value = "BST - Поиск";

sheet.Cells[row, 5].Value = "AVL - Добавление";

sheet.Cells[row, 6].Value = "AVL - Удаление";

sheet.Cells[row, 7].Value = "AVL - Поиск";

foreach (var size in sizes)

{

row++;

sheet.Cells[row, 1].Value = size;

double bstAddSum = 0, bstDeleteSum = 0, bstFindSum = 0;

double avlAddSum = 0, avlDeleteSum = 0, avlFindSum = 0;

for (var i = 1; i <= 5; i++)

{

var data = Enumerable.Range(0, size)

.Select(\_ => rand.NextInt64(70000000000, 79999999999).ToString())

.ToList();

var bst = new BinarySearchTree();

var avl = new AvlTree();

foreach (var phone in data)

bst.Add(phone);

foreach (var phone in data)

avl.Add(phone);

var newPhone = rand.NextInt64(70000000000, 79999999999).ToString();

var bstAdd = Measure(() => { bst.Add(newPhone); });

var avlAdd = Measure(() => { avl.Add(newPhone); });

var toDelete = data.OrderBy(\_ => rand.Next()).First();

var bstDelete = Measure(() => bst.Delete(toDelete));

var avlDelete = Measure(() => avl.Delete(toDelete));

// Find по одному элементу

var remaining = data.Where(x => x != toDelete).ToList();

var toFind = remaining.OrderBy(\_ => rand.Next()).First();

var bstFind = Measure(() => bst.Find(toFind));

var avlFind = Measure(() => avl.Find(toFind));

// Запись текущего прохода

row++;

sheet.Cells[row, 2].Value = bstAdd;

sheet.Cells[row, 3].Value = bstDelete;

sheet.Cells[row, 4].Value = bstFind;

sheet.Cells[row, 5].Value = avlAdd;

sheet.Cells[row, 6].Value = avlDelete;

sheet.Cells[row, 7].Value = avlFind;

bstAddSum += bstAdd;

bstDeleteSum += bstDelete;

bstFindSum += bstFind;

avlAddSum += avlAdd;

avlDeleteSum += avlDelete;

avlFindSum += avlFind;

}

// Средние значения

row++;

sheet.Cells[row, 1].Value = "Среднее";

sheet.Cells[row, 2].Value = bstAddSum / 5;

sheet.Cells[row, 3].Value = bstDeleteSum / 5;

sheet.Cells[row, 4].Value = bstFindSum / 5;

sheet.Cells[row, 5].Value = avlAddSum / 5;

sheet.Cells[row, 6].Value = avlDeleteSum / 5;

sheet.Cells[row, 7].Value = avlFindSum / 5;

}

sheet.Cells[sheet.Dimension.Address].AutoFitColumns();

package.SaveAs(new FileInfo(filePath));

}

MessageBox.Show("Бенчмарк завершён. Результаты сохранены на рабочем столе в TreeBenchmark.xlsx");

}

}

Файл MainForm

using System.Diagnostics;

using Lb1C\_.AVLTree;

using Lb1C\_.Tree;

namespace Lb1C\_;

public partial class MainForm : Form

{

private AvlTree avlTreeHead = new();

private TreeView avlTreeView;

private TreeView bstTreeView;

private TextBox genCountBox;

private Button generateBtn;

private Label infoLabel;

private TextBox inputBox;

private Button loadBtn, addBtn, deleteBtn, findBtn, clearBtn, toExcelBtn;

private BinarySearchTree treeHead = new();

public MainForm()

{

InitializeComponent();

InitControls();

}

private void InitControls()

{

BackColor = Color.FromArgb(245, 245, 250); // светло-серый фон

inputBox = new TextBox { Left = 10, Top = 10, Width = 200 };

loadBtn = new Button { Text = "Загрузить из файла", Left = 220, Top = 10, Height = 30 };

addBtn = new Button { Text = "Добавить", Left = 10, Top = 40, Height = 30 };

deleteBtn = new Button { Text = "Удалить", Left = 80, Top = 40, Height = 30 };

findBtn = new Button { Text = "Найти", Left = 160, Top = 40, Height = 30 };

clearBtn = new Button { Text = "Удалить всё", Left = 240, Top = 40, Height = 30 };

toExcelBtn = new Button { Text = "Создать excel", Left = 320, Top = 40, Height = 30 };

genCountBox = new TextBox{ Left = 620, Top = 10, Width = 100, Height = 30, PlaceholderText = "Кол-во"};

generateBtn = new Button{ Text = "Сгенерировать", Left = 730, Top = 10, Width = 120, Height = 30 };

bstTreeView = new TreeView { Left = 10, Top = 80, Width = 280, Height = 300 };

avlTreeView = new TreeView { Left = 320, Top = 80, Width = 280, Height = 300 };

infoLabel = new Label { Left = 10, Top = 400, Width = 900, Height = 50 };

genCountBox.BackColor = Color.White;

genCountBox.ForeColor = Color.Black;

inputBox.BackColor = Color.White;

inputBox.ForeColor = Color.Black;

infoLabel.BackColor = Color.WhiteSmoke;

infoLabel.ForeColor = Color.DarkSlateGray;

infoLabel.BorderStyle = BorderStyle.FixedSingle;

//Верхняя зона

inputBox.SetBounds(10, 10, 200, 30);

loadBtn.SetBounds(220, 10, 150, 30);

genCountBox.SetBounds(380, 10, 80, 30);

generateBtn.SetBounds(470, 10, 120, 30);

//Средняя зона

addBtn.SetBounds(10, 50, 100, 30);

deleteBtn.SetBounds(120, 50, 100, 30);

findBtn.SetBounds(230, 50, 100, 30);

clearBtn.SetBounds(340, 50, 120, 30);

toExcelBtn.SetBounds(470, 50, 130, 30);

//Деревья

bstTreeView.SetBounds(10, 100, 380, 300);

avlTreeView.SetBounds(400, 100, 380, 300);

bstTreeView.BackColor = Color.White;

avlTreeView.BackColor = Color.White;

//Нижняя зона

infoLabel.SetBounds(10, 410, 770, 40);

var bstContextMenu = new ContextMenuStrip();

var avlContextMenu = new ContextMenuStrip();

bstContextMenu.Items.Add("Копировать", null, CopyToClipboard\_BST);

avlContextMenu.Items.Add("Копировать", null, CopyToClipboard\_AVL);

bstTreeView.ContextMenuStrip = bstContextMenu;

avlTreeView.ContextMenuStrip = avlContextMenu;

Controls.AddRange(new Control[]

{

inputBox, clearBtn, loadBtn, addBtn, deleteBtn, findBtn,

avlTreeView, bstTreeView, infoLabel, toExcelBtn, genCountBox,

generateBtn

});

loadBtn.Click += LoadBtn\_Click;

addBtn.Click += AddBtn\_Click;

deleteBtn.Click += DeleteBtn\_Click;

findBtn.Click += FindBtn\_Click;

clearBtn.Click += ClearBtn\_Click;

toExcelBtn.Click += (s, e) => new TreeToExcel().RunBenchmarksAndExportToExcel();

generateBtn.Click += (s, e) =>

{

if (int.TryParse(genCountBox.Text.Trim(), out int count) && count > 0)

{

GenerateData(count);

}

else

{

MessageBox.Show("Введите корректное число для генерации", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

};

}

private void CopyToClipboard\_BST(object sender, EventArgs e)

{

var treeData = GetTreeData(treeHead.Root);

Clipboard.SetText(treeData);

}

private void CopyToClipboard\_AVL(object sender, EventArgs e)

{

var treeData = GetTreeData(avlTreeHead.Root);

Clipboard.SetText(treeData);

}

private string GetTreeData(dynamic node)

{

if (node == null) return string.Empty;

var result = node.Phone + Environment.NewLine;

result += GetTreeData(node.Left);

result += GetTreeData(node.Right);

return result;

}

private void GenerateData(int count)

{

avlTreeHead = new AvlTree();

treeHead = new BinarySearchTree();

var random = new Random();

var phones = Enumerable.Range(0, count)

.Select(i => random.NextInt64(70000000000, 79999999999))

.ToList();

var sw = Stopwatch.StartNew();

foreach (var phone in phones)

treeHead.Add(phone.ToString());

var treeTime = sw.Elapsed.TotalMicroseconds;

sw.Restart();

foreach (var phone in phones)

avlTreeHead.Add(phone.ToString());

var avlTreeTime = sw.Elapsed.TotalMicroseconds;

infoLabel.Text = $"Generated {count}: Tree {treeTime:F4}ms, AvlTree {avlTreeTime:F4}ms";

RefreshUI();

}

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

{

var dialog = new OpenFileDialog();

if (dialog.ShowDialog() != DialogResult.OK) return;

avlTreeHead = new AvlTree();

var lines = File.ReadAllLines(dialog.FileName);

foreach (var line in lines)

{

var number = line.Trim();

if (!string.IsNullOrEmpty(number))

{

avlTreeHead.Add(number);

treeHead.Add(number);

}

}

RefreshUI();

}

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

{

var phone = inputBox.Text.Trim();

if (string.IsNullOrWhiteSpace(phone)) return;

var sw = Stopwatch.StartNew();

treeHead.Add(phone);

var elapsed = sw.Elapsed;

sw.Restart();

avlTreeHead.Add(phone);

var elapsedByAvl = sw.Elapsed;

infoLabel.Text = $"Added: Tree {elapsed.TotalMilliseconds:F4}ms AvlTree {elapsedByAvl.TotalMilliseconds:F4}ms";

RefreshUI();

}

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

{

var phone = inputBox.Text.Trim();

if (string.IsNullOrWhiteSpace(phone)) return;

var sw = Stopwatch.StartNew();

treeHead.Delete(phone);

var elapsed = sw.Elapsed;

sw.Restart();

avlTreeHead.Delete(phone);

var elapsedByAvl = sw.Elapsed;

infoLabel.Text =

$"Deleted: Tree {elapsed.TotalMilliseconds:F4}ms AvlTree {elapsedByAvl.TotalMilliseconds:F4}ms";

RefreshUI();

}

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

{

var phone = inputBox.Text.Trim();

if (string.IsNullOrWhiteSpace(phone)) return;

var sw = Stopwatch.StartNew();

var found = treeHead.Find(phone);

var elapsed = sw.Elapsed;

sw.Restart();

var foundInAvlTree = avlTreeHead.Find(phone);

var elapsedByAvl = sw.Elapsed;

infoLabel.Text =

$"Found: Tree {elapsed.TotalMilliseconds:F4}ms ({found != null}) AvlTree {elapsedByAvl.TotalMilliseconds:F4}ms ({foundInAvlTree != null})";

}

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

{

avlTreeHead = new AvlTree();

treeHead = new BinarySearchTree();

RefreshUI();

infoLabel.Text = "Cleared All";

}

private void RefreshUI()

{

avlTreeView.Nodes.Clear();

bstTreeView.Nodes.Clear();

DisplayTree(avlTreeHead.Root, null, avlTreeView);

DisplayTree(treeHead.Root, null, bstTreeView);

infoLabel.Text +=

$" | BST: Count={treeHead.Count()}, Leaves={treeHead.Leaves()}, Depth={treeHead.Depth()} | " +

$"AVL: Count={avlTreeHead.Count()}, Leaves={avlTreeHead.Leaves()}, Depth={avlTreeHead.Depth()}";

}

private void DisplayTree(dynamic node, TreeNode parent, TreeView view)

{

if (node == null) return;

var treeNode = new TreeNode(node.Phone);

if (parent == null)

view.Nodes.Add(treeNode);

else

parent.Nodes.Add(treeNode);

DisplayTree(node.Left, treeNode, view);

DisplayTree(node.Right, treeNode, view);

}

private void Form1\_Load(object sender, EventArgs e)

{

}

}