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

## 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.

## Постановка задачи

1. В ходе выполнения настоящей лабораторной работы сначала необходимо ознакомиться с организацией и программной реализацией AVL-деревьев (см. п. 2, Приложения A-Б).

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 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.

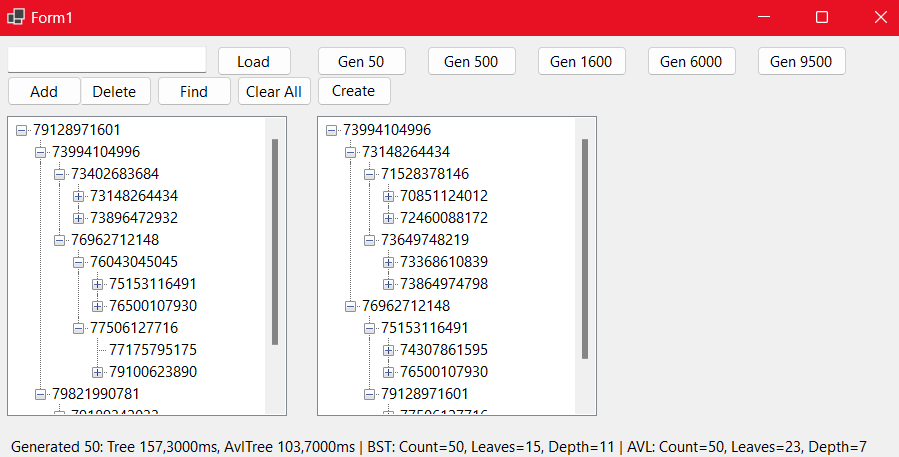
Вариант задания

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Вариант | Файл  данных | Ключевое  поле | N1 | N2 | N3 | N4 | N5 |
| 10 | Table22.txt | Телефон | 50 | 500 | 1600 | 6000 | 9000 |

## Ход работы

Код программы в Приложении А.

Интерфейс пользователя показан на рисунке 2.1. Он содержит 5 кнопок для генерации данных, кнопки добавления, удаления и поиска данных. В левом окне отображаются данные в списке, а в правом в дереве. Также есть возможность загрузить данные из файла. Внизу расположена информация о времени выполнения операции, кол-во элементов в дереве, количество листов и глубина.

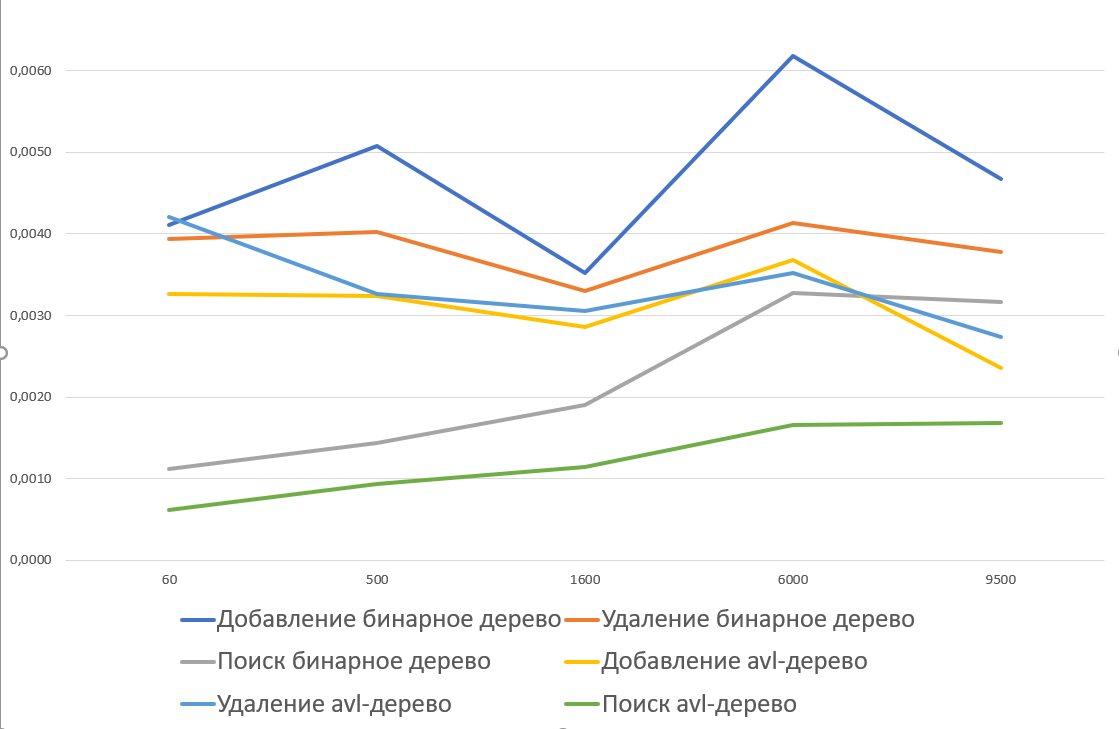


* + - * 1. Интерфейс пользователя

Таблица 2.2 – Время выполнения операций в списке и дереве

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во  элементов |  | Бинарное дерево | | | AVL-дерево | | |
| Добавление | Удаление | Поиск | Добавление | Удаление | Поиск |
| 60 | 1 | 0,0065 | 0,0037 | 0,0021 | 0,0044 | 0,0029 | 0,0002 |
| 2 | 0,0035 | 0,0073 | 0,0011 | 0,0029 | 0,0046 | 0,0006 |
| 3 | 0,0029 | 0,0022 | 0,0007 | 0,0025 | 0,0746 | 0,0006 |
| 4 | 0,0038 | 0,0033 | 0,0003 | 0,0031 | 0,0031 | 0,0008 |
| 5 | 0,0039 | 0,0032 | 0,0014 | 0,0034 | 0,0030 | 0,0009 |
| Ср | 0,0041 | 0,0039 | 0,0011 | 0,0033 | 0,0176 | 0,0006 |
| 500 | 1 | 0,0036 | 0,0055 | 0,0021 | 0,0027 | 0,0036 | 0,0007 |
| 2 | 0,0046 | 0,0044 | 0,0014 | 0,0033 | 0,0035 | 0,0011 |
| 3 | 0,0054 | 0,0041 | 0,0018 | 0,0042 | 0,0035 | 0,0013 |
| 4 | 0,0087 | 0,0037 | 0,0011 | 0,0035 | 0,0033 | 0,0009 |
| 5 | 0,0031 | 0,0024 | 0,0008 | 0,0025 | 0,0024 | 0,0007 |
| Ср | 0,0051 | 0,0040 | 0,0014 | 0,0032 | 0,0033 | 0,0009 |
| 1600 | 1 | 0,0024 | 0,0026 | 0,0022 | 0,0026 | 0,0030 | 0,0011 |
| 2 | 0,0035 | 0,0037 | 0,0015 | 0,0029 | 0,0032 | 0,0011 |
| 3 | 0,0033 | 0,0032 | 0,0018 | 0,0026 | 0,0027 | 0,0013 |
| 4 | 0,0039 | 0,0028 | 0,0019 | 0,0026 | 0,0027 | 0,0009 |
| 5 | 0,0045 | 0,0042 | 0,0021 | 0,0036 | 0,0037 | 0,0013 |
| Ср | 0,0035 | 0,0033 | 0,0019 | 0,0029 | 0,0031 | 0,0011 |
| 6000 | 1 | 0,0081 | 0,0044 | 0,0027 | 0,0042 | 0,0039 | 0,0021 |
| 2 | 0,0073 | 0,0058 | 0,0040 | 0,0036 | 0,0046 | 0,0020 |
| 3 | 0,0078 | 0,0044 | 0,0036 | 0,0061 | 0,0042 | 0,0018 |
| 4 | 0,0036 | 0,0025 | 0,0025 | 0,0021 | 0,0024 | 0,0011 |
| 5 | 0,0041 | 0,0036 | 0,0036 | 0,0024 | 0,0025 | 0,0013 |
| Ср | 0,0062 | 0,0041 | 0,0033 | 0,0037 | 0,0035 | 0,0017 |
| 9500 | 1 | 0,0037 | 0,0043 | 0,0030 | 0,0021 | 0,0025 | 0,0022 |
| 2 | 0,0034 | 0,0026 | 0,0031 | 0,0022 | 0,0023 | 0,0018 |
| 3 | 0,0055 | 0,0050 | 0,0041 | 0,0029 | 0,0036 | 0,0016 |
| 4 | 0,0063 | 0,0033 | 0,0032 | 0,0026 | 0,0021 | 0,0016 |
| 5 | 0,0045 | 0,0037 | 0,0024 | 0,0020 | 0,0032 | 0,0012 |
| Ср | 0,0047 | 0,0038 | 0,0032 | 0,0024 | 0,0027 | 0,0017 |

Изобразим эти данные на графике(рисунок 1.2).



* + - * 1. График времени выполнения операций

## Вывод

В ходе выполнения лабораторной работы были изучены особенности использования нелинейных структур данных на примере AVL-деревьев. Особое внимание было уделено сравнительному анализу работы бинарного дерева поиска и самобалансирующегося AVL-дерева. Сначала была изучена теоретическая часть, связанная с принципами построения и балансировки AVL-деревьев, а также реализация операций добавления, удаления и поиска элементов в подобных структурах. Затем было разработано Windows-приложение, которое позволяет визуализировать работу обеих структур, а также обеспечивает удобный интерфейс для загрузки данных, их модификации и анализа эффективности операций.

В процессе работы была реализована возможность отображения дерева и списка, ввода данных из файла, а также отображения различной статистики, такой как количество элементов, листьев и глубина дерева. Также программа замеряет время выполнения основных операций, что позволило провести экспериментальное исследование производительности. На основании проведённых экспериментов было зафиксировано время выполнения операций вставки, удаления и поиска при разном количестве элементов (от 50 до 9500) и построены соответствующие графики.

Анализ полученных данных показал, что AVL-дерево демонстрирует более стабильные и предсказуемые результаты, особенно при больших объёмах данных, благодаря своей самобалансирующейся природе. При этом время выполнения операций в AVL-дереве оказалось либо на уровне, либо меньше по сравнению с бинарным деревом поиска, особенно при увеличении количества элементов. Таким образом, было подтверждено преимущество использования AVL-деревьев для задач, где важна высокая эффективность операций с большими объёмами данных.

# пРИЛОЖЕНИЕ А кОД ПРОГРАММЫ

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 Button gen50Btn, gen500Btn, gen1600Btn, gen6000Btn, gen9500Btn;

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()

{

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

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

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

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

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

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

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

gen50Btn = new Button { Text = "Gen 50", Left = 320, Top = 10, Height = 30, Width = 90 };

gen500Btn = new Button { Text = "Gen 500", Left = 430, Top = 10, Height = 30, Width = 90 };

gen1600Btn = new Button { Text = "Gen 1600", Left = 540, Top = 10, Height = 30, Width = 90 };

gen6000Btn = new Button { Text = "Gen 6000", Left = 650, Top = 10, Height = 30, Width = 90 };

gen9500Btn = new Button { Text = "Gen 9500", Left = 760, Top = 10, Height = 30, Width = 90 };

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 };

var bstContextMenu = new ContextMenuStrip();

var avlContextMenu = new ContextMenuStrip();

bstContextMenu.Items.Add("Copy to Clipboard", null, CopyToClipboard\_BST);

avlContextMenu.Items.Add("Copy to Clipboard", null, CopyToClipboard\_AVL);

bstTreeView.ContextMenuStrip = bstContextMenu;

avlTreeView.ContextMenuStrip = avlContextMenu;

Controls.AddRange(new Control[]

{

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

gen50Btn, gen500Btn, gen1600Btn, gen6000Btn, gen9500Btn,

avlTreeView, bstTreeView, infoLabel, toExcelBtn

});

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();

gen50Btn.Click += (s, e) => GenerateData(50);

gen500Btn.Click += (s, e) => GenerateData(500);

gen1600Btn.Click += (s, e) => GenerateData(1600);

gen6000Btn.Click += (s, e) => GenerateData(6000);

gen9500Btn.Click += (s, e) => GenerateData(9500);

}

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)

{

}

}

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);

}

// ========= Приватные методы ==========

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; // дубликаты игнорируются

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 AvlTreeNode

namespace Lb1C\_.AVLTree;

public class AvlTreeNode

{

public int Height;

public AvlTreeNode Left, Right;

public string Phone;

public AvlTreeNode(string phone)

{

Phone = phone;

Height = 1;

}

}

Файл BinarySearchTree

namespace Lb1C\_.Tree;

public class BinarySearchTree

{

public BSTreeNode Root { get; private set; }

public void Add(DateTime time)

{

Root = Add(Root, time);

}

private BSTreeNode Add(BSTreeNode node, DateTime time)

{

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

if (time < node.Time)

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

else if (time >= node.Time)

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

return node;

}

public bool Delete(DateTime time)

{

var found = false;

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

return found;

}

private BSTreeNode Delete(BSTreeNode node, DateTime time, ref bool found)

{

if (node == null) return null;

if (time < node.Time)

{

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

}

else if (time > node.Time)

{

node.Right = Delete(node.Right, time, 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.Time = min.Time;

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

}

return node;

}

private BSTreeNode FindMin(BSTreeNode node)

{

while (node.Left != null)

node = node.Left;

return node;

}

public BSTreeNode Find(DateTime time)

{

return Find(Root, time);

}

private BSTreeNode Find(BSTreeNode node, DateTime time)

{

if (node == null || node.Time == time) return node;

return time < node.Time

? Find(node.Left, time)

: Find(node.Right, time);

}

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));

}

}

Файл BSTreeNode

namespace Lb1C\_.Tree;

public class BSTreeNode

{

public BSTreeNode(DateTime time)

{

Time = time;

}

public DateTime Time { get; set; }

public BSTreeNode Left { get; set; }

public BSTreeNode Right { get; set; }

}