# ЛАБОРАТОРНАЯ РАБОТА № 1 «ИССЛЕДОВАНИЕ ЛИНЕЙНЫХ И НЕЛИНЕЙНЫХ СТРУКТУР ДАННЫХ. ЛИНЕЙНЫЕ СПИСКИ И БИНАРНЫЕ ДЕРЕВЬЯ ПОИСКА»

## Цель работы

Исследовать возможности применения линейных и нелинейных структур данных – линейных списков и бинарных деревьев поиска – для хранения, поиска и обработки информации. Приобрести практические навыки создания и использования классов, реализующих списки и бинарные деревья поиска и исследовать эффективность этих структур данных при выполнении операций добавления удаления и поиска данных.

## Постановка задачи

1. В ходе выполнения настоящей лабораторной работы в начале необходимо ознакомиться с организацией и программной реализацией двух типов структур данных – линейного списка и бинарного деревьев поиска (см. п. 2, Приложения A-В).

2. На одном из языков программирования (С++ или Object Pascal) в среде визуального программирования (С++ Builder или Delphi, соответственно), с иcпользованием классов, реализующих линейные списки и бинарные деревья поиска (для С++ файлы: classes.h, DATA.H и MYLIST.H; для ObjectPascal: файл Collection.pas; - описания и программная реализация данных классов приведена в Приложениях A-В) реализовать Windows-приложение(примеры создания подобных приложений приведены в пп. 3.2 и 3.3 для Delphi и С++ Builder, соответственно), обеспечивающее выполнения следующих функций:

2.1. Построение линейного списка и бинарного дерева поиска на основании данных, расположенных в файле(имя файла и имя ключевого поля определяются вариантом задания - таблица 1), содержащем построчно записи разделенные символом табуляции;

2.2. Отображение на визуальной форме линейного списка, используя компонент TListView, и бинарного дерева поиска, используя компонент TTreeView;

2.3. Предоставление интерфейса пользователю для выполнения операций добавления, удаления, изменения и поиска(по ключевому полю) элементов списка и дерева, и отбражать результаты выполнения операций на 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 списка и дерева из входного файла с ограничением числа добавляемых элементов (макимальное 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 линейного списка и бинарного дерева поиска:

3.1. Построить список и бинарное 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 1.1), фиксируя получаемые значения времни в таблице вида:

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 линейного списка и бинарного дерева поиска.

6. Сформулировать выводы.

7. Оформить отчет.

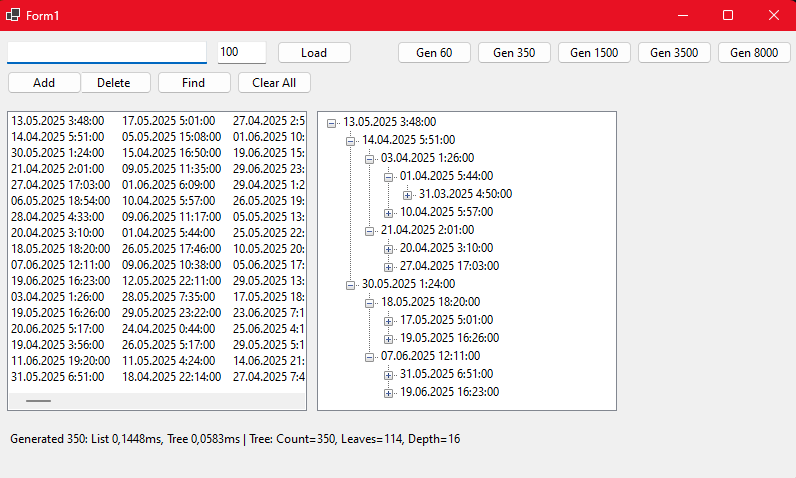
Вариант задания

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Вариант | Файл  данных | Ключевое  поле | N1 | N2 | N3 | N4 | N5 |
| 10 | Table1.txt | Время | 60 | 350 | 1500 | 3500 | 8000 |

## Ход работы

Код программы в Приложении А.

Интерфейс пользователя показан на рисунке 1.1. 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.

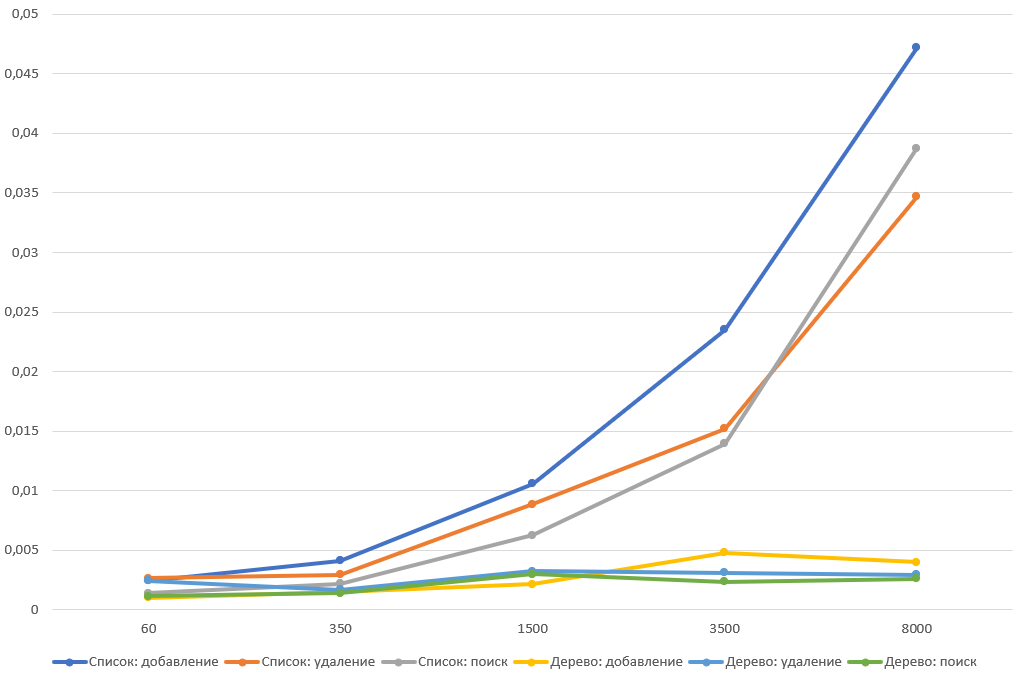


* + - * 1. Интерфейс пользователя

Таблица 1.2 – Время выполнения операций в списке и дереве

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во  элементов |  | Список | | | Дерево | | |
| Добавление | Удаление | Поиск | Добавление | Удаление | Поиск |
| 60 | 1 | 0,0025 | 0,0022 | 0,0011 | 0,0009 | 0,0015 | 0,0011 |
| 2 | 0,0023 | 0,0031 | 0,001 | 0,0012 | 0,0011 | 0,0008 |
| 3 | 0,0024 | 0,0019 | 0,0015 | 0,001 | 0,0071 | 0,0012 |
| 4 | 0,0026 | 0,0038 | 0,0012 | 0,0011 | 0,001 | 0,001 |
| 5 | 0,0024 | 0,0023 | 0,0022 | 0,0009 | 0,0014 | 0,0015 |
| Ср | 0,00244 | 0,00266 | 0,00140 | 0,00102 | 0,00242 | 0,00112 |
| 350 | 1 | 0,0038 | 0,0033 | 0,0011 | 0,001 | 0,0023 | 0,0012 |
| 2 | 0,0034 | 0,0022 | 0,0013 | 0,0015 | 0,0017 | 0,0012 |
| 3 | 0,0039 | 0,0023 | 0,0023 | 0,0011 | 0,0146 | 0,0014 |
| 4 | 0,0042 | 0,004 | 0,0033 | 0,0017 | 0,0014 | 0,0018 |
| 5 | 0,0052 | 0,0028 | 0,0028 | 0,0023 | 0,1837 | 0,0013 |
| Ср | 0,00410 | 0,00292 | 0,00216 | 0,00152 | 0,00167 | 0,00138 |
| 1500 | 1 | 0,0094 | 0,0089 | 0,0018 | 0,0018 | 0,0034 | 0,001 |
| 2 | 0,0122 | 0,0092 | 0,0056 | 0,0023 | 0,0016 | 0,0027 |
| 3 | 0,0111 | 0,0104 | 0,0081 | 0,0022 | 0,0032 | 0,0032 |
| 4 | 0,0096 | 0,0071 | 0,0076 | 0,0022 | 0,0056 | 0,0056 |
| 5 | 0,0106 | 0,0086 | 0,0082 | 0,0022 | 0,0023 | 0,0023 |
| Ср | 0,01058 | 0,00884 | 0,00626 | 0,00214 | 0,00322 | 0,00296 |
| 3500 | 1 | 0,0273 | 0,0016 | 0,0121 | 0,0043 | 0,0024 | 0,002 |
| 2 | 0,0201 | 0,0184 | 0,0179 | 0,0032 | 0,0026 | 0,0023 |
| Кол-во  элементов |  | Список | | | Дерево | | |
| Добавление | Удаление | Поиск | Добавление | Удаление | Поиск |
| 3500 | 3 | 0,0201 | 0,0176 | 0,0125 | 0,0028 | 0,0035 | 0,0018 |
| 4 | 0,029 | 0,0230 | 0,0066 | 0,0095 | 0,0023 | 0,0032 |
| 5 | 0,0211 | 0,0154 | 0,0206 | 0,0041 | 0,0047 | 0,0024 |
| Ср | 0,02352 | 0,01520 | 0,01394 | 0,00478 | 0,00310 | 0,00234 |
| 8000 | 1 | 0,0498 | 0,0207 | 0,0561 | 0,0058 | 0,0023 | 0,0023 |
| 2 | 0,0501 | 0,0494 | 0,0294 | 0,0042 | 0,0032 | 0,0032 |
| 3 | 0,0413 | 0,0388 | 0,0324 | 0,0029 | 0,0040 | 0,0029 |
| 4 | 0,0466 | 0,0452 | 0,0264 | 0,0047 | 0,0029 | 0,0021 |
| 5 | 0,0482 | 0,0194 | 0,0494 | 0,0023 | 0,0022 | 0,0024 |
| Ср | 0,04720 | 0,03470 | 0,03874 | 0,00398 | 0,00292 | 0,00258 |

Изобразим эти данные на графике(рисунок 1.2).



* + - * 1. График времени выполнения операций

## Вывод

В ходе выполнения лабораторной работы были изучены линейные и нелинейные структуры данных на примере реализации линейного списка и бинарного дерева поиска. Основное внимание было уделено сравнительному исследованию эффективности выполнения операций добавления, удаления и поиска данных в этих структурах. Для проведения экспериментов было разработано Windows-приложение на языке программирования C# с использованием визуального интерфейса, в котором реализована возможность построения и визуализации списка и дерева на основе данных из файла, а также выполнения операций с отображением их результатов и времени выполнения. Программа позволяла работать с различным количеством элементов, загружаемых из файла, а также отображала актуальную информацию о количестве элементов, глубине дерева и числе листьев.

В процессе работы были проведены замеры времени выполнения операций для различных объёмов данных (60, 350, 1500, 3500 и 8000 элементов). Результаты замеров показали, что бинарное дерево поиска в большинстве случаев обеспечивает более высокую эффективность по сравнению с линейным списком, особенно при увеличении объёма данных. Особенно заметна разница во времени выполнения операций поиска и удаления: в дереве они происходят быстрее благодаря логарифмической структуре, тогда как в списке время выполнения растёт почти линейно. Наименьшие значения времени выполнения операций наблюдаются при работе с малым количеством элементов, однако с ростом объема данных бинарное дерево сохраняет стабильную производительность, в то время как список демонстрирует ухудшение.

# пРИЛОЖЕНИЕ А кОД ПРОГРАММЫ

Файл MainForm

using System.Diagnostics;

using Lb1C\_.List;

using Lb1C\_.Tree;

namespace Lb1C\_;

public partial class MainForm : Form

{

private readonly LinkedList list = new();

private Button gen60Btn, gen350Btn, gen1500Btn, gen3500Btn, gen8000Btn;

private Label infoLabel;

private TextBox inputBox;

private ListView listView;

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

private TextBox maxBox;

private BinarySearchTree tree = new();

private TreeView treeView;

public MainForm()

{

InitializeComponent();

InitControls();

}

private void InitControls()

{

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

maxBox = new TextBox { Left = 220, Top = 10, Width = 50, Text = "100" };

loadBtn = new Button { Text = "Load", Left = 280, Top = 10 };

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

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

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

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

gen60Btn = new Button { Text = "Gen 60", Left = 400, Top = 10 };

gen350Btn = new Button { Text = "Gen 350", Left = 480, Top = 10 };

gen1500Btn = new Button { Text = "Gen 1500", Left = 560, Top = 10 };

gen3500Btn = new Button { Text = "Gen 3500", Left = 640, Top = 10 };

gen8000Btn = new Button { Text = "Gen 8000", Left = 720, Top = 10 };

listView = new ListView { Left = 10, Top = 80, Width = 300, Height = 300, View = View.List };

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

infoLabel = new Label { Left = 10, Top = 400, Width = 600, Height = 50 };

Controls.AddRange(new Control[]

{

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

gen60Btn, gen350Btn, gen1500Btn, gen3500Btn, gen8000Btn,

listView, treeView, infoLabel

});

loadBtn.Click += LoadBtn\_Click;

addBtn.Click += AddBtn\_Click;

deleteBtn.Click += DeleteBtn\_Click;

findBtn.Click += FindBtn\_Click;

clearBtn.Click += ClearBtn\_Click;

gen60Btn.Click += (s, e) => GenerateData(60);

gen350Btn.Click += (s, e) => GenerateData(350);

gen1500Btn.Click += (s, e) => GenerateData(1500);

gen3500Btn.Click += (s, e) => GenerateData(3500);

gen8000Btn.Click += (s, e) => GenerateData(8000);

}

private void GenerateData(int count)

{

list.Clear();

tree = new BinarySearchTree();

var baseDate = DateTime.Today;

var random = new Random();

var times = Enumerable.Range(0, count)

.Select(i => baseDate.AddMinutes(random.Next(0, 100000)).AddDays(random.Next(-15, 10)))

.ToList();

var sw = Stopwatch.StartNew();

foreach (var time in times)

list.Add(time);

var listTime = sw.Elapsed;

sw.Restart();

foreach (var time in times)

tree.Add(time);

var treeTime = sw.Elapsed;

infoLabel.Text =

$"Generated {count}: List {listTime.TotalMilliseconds:F4}ms, Tree {treeTime.TotalMilliseconds:F4}ms";

RefreshUI();

}

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

{

list.Clear();

tree = new BinarySearchTree();

RefreshUI();

infoLabel.Text = "Cleared All";

}

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

{

var dialog = new OpenFileDialog();

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

list.Clear();

tree = new BinarySearchTree();

var max = int.TryParse(maxBox.Text, out var m) ? m : 100;

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

foreach (var line in lines)

{

var parts = line.Split('\t');

if (DateTime.TryParse(parts[0], out var time))

{

list.Add(time);

tree.Add(time);

}

}

RefreshUI();

}

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

{

if (!DateTime.TryParse(inputBox.Text, out var time)) return;

var sw = Stopwatch.StartNew();

list.Add(time);

var t1 = sw.Elapsed;

sw.Restart();

tree.Add(time);

var t2 = sw.Elapsed;

infoLabel.Text = $"Added: List {t1.TotalMilliseconds:F4}ms, Tree {t2.TotalMilliseconds:F4}ms";

RefreshUI();

}

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

{

if (!DateTime.TryParse(inputBox.Text, out var time)) return;

var sw = Stopwatch.StartNew();

list.Delete(time);

var t1 = sw.Elapsed;

sw.Restart();

tree.Delete(time);

var t2 = sw.Elapsed;

infoLabel.Text = $"Deleted: List {t1.TotalMilliseconds:F4}ms, Tree {t2.TotalMilliseconds:F4}ms";

RefreshUI();

}

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

{

if (!DateTime.TryParse(inputBox.Text, out var time)) return;

var sw = Stopwatch.StartNew();

var n1 = list.Find(time);

var t1 = sw.Elapsed;

sw.Restart();

var n2 = tree.Find(time);

var t2 = sw.Elapsed;

infoLabel.Text =

$"Found: List {t1.TotalMilliseconds:F4}ms ({n1 != null}), Tree {t2.TotalMilliseconds:F4}ms ({n2 != null})";

}

private void RefreshUI()

{

listView.Items.Clear();

var node = list.Head;

while (node != null)

{

listView.Items.Add(node.Time.ToString("G"));

node = node.Next;

}

treeView.Nodes.Clear();

DisplayTree(tree.Root, null, treeView);

infoLabel.Text += $" | Tree: Count={tree.Count()}, Leaves={tree.Leaves()}, Depth={tree.Depth()}";

}

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

{

if (node == null) return;

var treeNode = new TreeNode(node.Time.ToString("G"));

if (parent == null)

view.Nodes.Add(treeNode);

else

parent.Nodes.Add(treeNode);

DisplayTree(node.Left, treeNode, view);

DisplayTree(node.Right, treeNode, view);

}

private void ListView\_ItemActivate(object sender, EventArgs e)

{

if (listView.SelectedItems.Count > 0)

{

var selectedTime = listView.SelectedItems[0].Text;

Clipboard.SetText(selectedTime);

}

}

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

{

listView.ItemActivate += ListView\_ItemActivate;

}

}

Файл LinkedList

namespace Lb1C\_.List;

public class LinkedList

{

public Node Head { get; private set; }

public int Count { get; private set; }

public void Add(DateTime time)

{

var node = new Node(time);

if (Head == null)

{

Head = node;

}

else

{

var current = Head;

while (current.Next != null)

current = current.Next;

current.Next = node;

}

Count++;

}

public bool Delete(DateTime time)

{

Node current = Head;

Node prev = null;

while (current != null)

{

if (current.Time == time)

{

if (prev == null)

Head = current.Next;

else

prev.Next = current.Next;

Count--;

return true;

}

prev = current;

current = current.Next;

}

return false;

}

public Node Find(DateTime time)

{

var current = Head;

while (current != null)

{

if (current.Time == time)

return current;

current = current.Next;

}

return null;

}

public void Clear()

{

Head = null;

Count = 0;

}

}

Файл Node

namespace Lb1C\_.List;

public class Node

{

public Node(DateTime time)

{

Time = time;

}

public DateTime Time { get; set; }

public Node Next { get; set; }

}

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; }

}