МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования  
«Гомельский государственный технический университет имени П.О. Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

Специальность 1-40 04 01 «Информатика и технологии программирования»

Отчет  
 по лабораторной работе №2  
по дисциплине «Методы трансляции»  
на тему: «Организация таблиц идентификаторов»

Составил: студент гр. ИП-31 Коваленко А.И.

(подпись, дата)

Принял доцент кафедры ИТ Кравченко О.А.

(подпись, дата)

Гомель 2023

***Цель работы***: *изучить основные методы организации таблиц идентификаторов, получить представление о преимуществах и недостатках, присущих различным методам организации таблиц идентификаторов.*

***Задание***

Разработать алгоритм, написать и отладить программу с графическим интерфейсом поиска в таблице идентификаторов заданного значения двумя методами и сравнения быстродействия этих методов в соответствии с индивидуальным заданием.

Функции программы:

- первоначальное заполнение таблицы идентификаторов (не менее 50 значений, длина идентификаторов – не более 32 символов) посредством ввода информации из текстового файла в заданную структуру данных;

- ввод нового идентификатора;

- вывод информации о месте нахождения введенного идентификатора в существующей таблице или включение в таблицу нового идентификатора, если его нет в таблице;

- вывод информации о времени (скорости) поиска идентификатора в таблице для каждого из рассматриваемых методов.

12. Первый метод: таблица идентификаторов – бинарное дерево, обход Left-Right-Root.

Второй метод: таблица идентификаторов – массив, полученный методом хеширования. Тип ключа – строка текста произвольной длины. Преобразование строки – конкатенация битовых образов символов. Метод хеширования – модульный. Метод разрешения коллизий – квадратичный.

Приложение было разработано на языке C#. Пользовательский интерфейс написан на WPF.

***Описание функций***

1. Первоначальное заполнение таблицы идентификаторов (не менее 50 значений, длина идентификаторов – не более 32 символов) посредством ввода информации из текстового файла в заданную структуру данных.

Для заполнения дерева и таблицы идентификаторов пользователь необходимо нажать на зеленую кнопку “считать из файла”. Файл *identif.txt* находится в корневой папке /*bin/debug*.

Функция *void ReadFile\_Click(object sender, RoutedEventArgs e)* является обработчиком события нажатия на кнопку «считать из файла». После выполнения функции произойдет вызов функций *Hashtable.Add(line), binaryTree.Add(line)*, которые добавляют данные в дерево и таблицу идентификаторов.

2. Ввод нового идентификатора.

Для добавления нового идентификатора необходимо ввести его в соответствующее поле и нажать на кнопку “Добавить”.

В обработчике события кнопки *void AddButton\_Click(object sender, RoutedEventArgs e):*

* вызов функции *hashTable(string text)*. Входным параметром data является идентификатор для добавления. После выполнения функции идентификатор будет добавлен в линейный двусвязный циклический список.
* вызов функции *binaryTree (string text).* Входным параметром data является идентификатор для добавления. После выполнения функции идентификатор будет добавлен в таблица идентификаторов.

Алгоритм добавления идентификатора в таблицу:

1) Ключ получается путем конкатенации символов строки битовых образов символов.

2) В случае возникновения коллизия: шаг q не фиксирован, а изменяется квадратично: q=1,4,9,16.... Соответственно при попытке добавить элемент в занятую ячейку i начинаем последовательно просматривать ячейки i+1,i+4,i+9 и так далее, пока не найдём свободную ячейку.

3) В массив мы добавляем новый элемент в ячейку с индексом хэша.

4) Если эта ячейка занята, то переходим к шагу 2. Если ячейка свободна, то в массив добавляется элемент в новую ячейку.

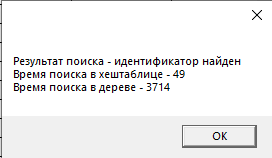
3. Поиск по заданному идентификатору:

Для вывода информации необходимо ввести идентификатор и нажать на кнопку “поиск”

Функция *void SearchButton\_Click(object sender, RoutedEventArgs e)* обрабатывает нажатие на кнопку. После выполнения функции, если существует, будет выведена информация о идентификаторе. В противном случае мы получим сообщение о том, что идентификатора нет.

4. Вывод информации о времени (скорости) поиска идентификатора в таблице.

Информация о времени выводится в модальном окне, в котором указано время поиска в бинарном дереве и таблице идентификаторов. Например:



**Тесты для добавления:**

|  |  |  |
| --- | --- | --- |
| Идентификатор | Результат | Ожидаемый результат |
| компилятор | Идентификатор добавлен | Идентификатор добавлен |
| строка | Идентификатор добавлен | Идентификатор добавлен |

**Тесты для поиска**

|  |  |  |
| --- | --- | --- |
| Идентификатор | Результат | Ожидаемый результат |
| бутерброд | Идентификатор не найден | Идентификатор не найден |
| строка | Идентификатор найден | Идентификатор найден |

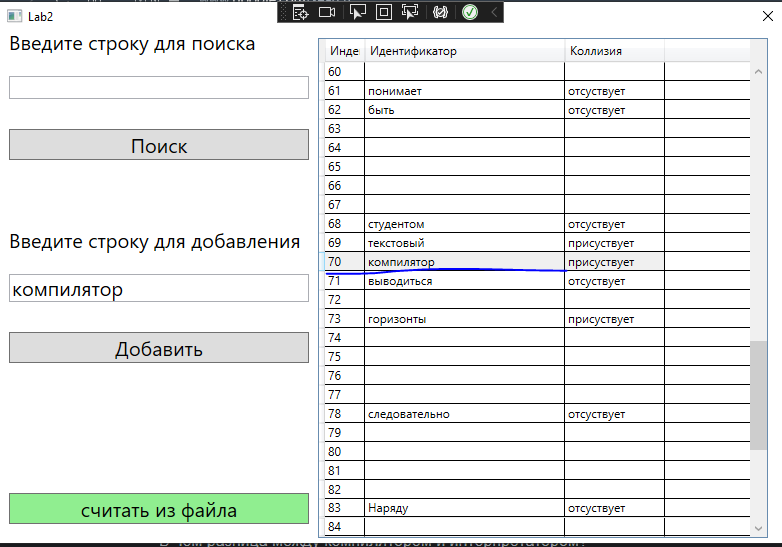


Рисунок 1 – Добавление нового идентификатора

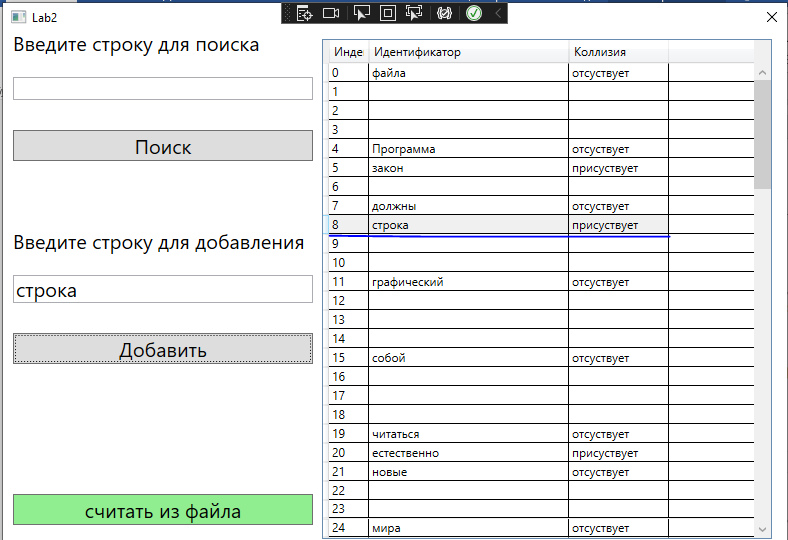


Рисунок 2 – Добавление нового идентификатора

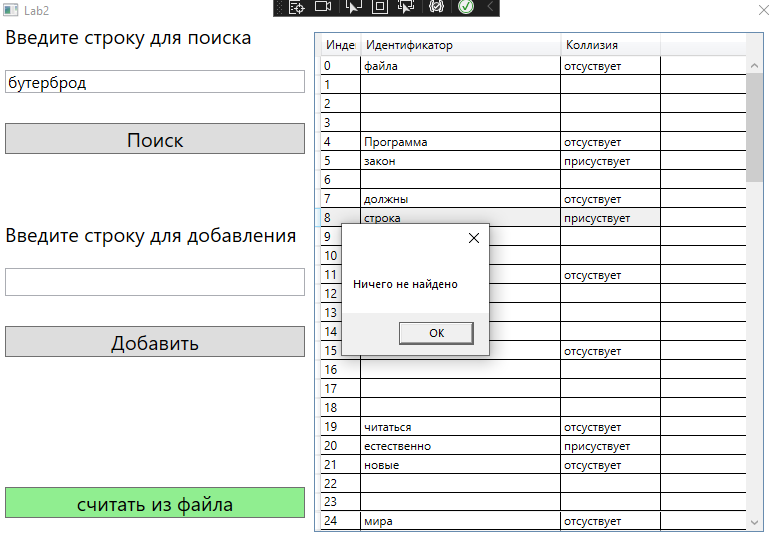


Рисунок 3 – Поиск идентификатора

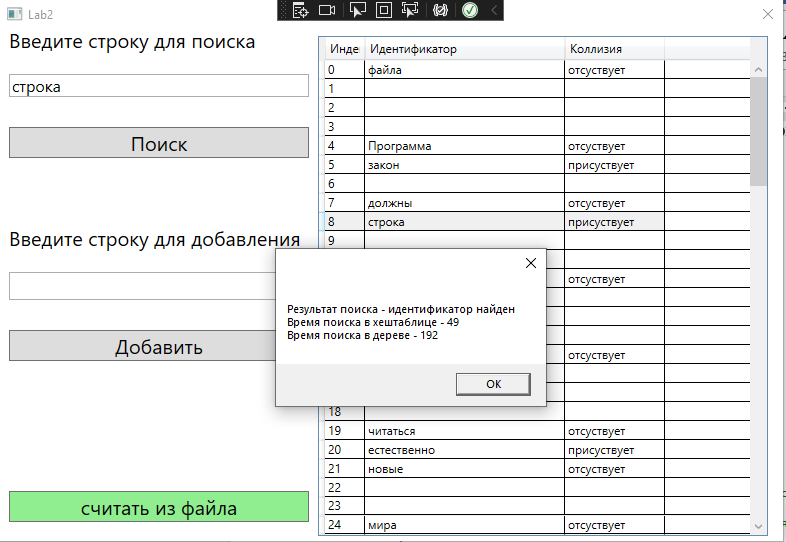
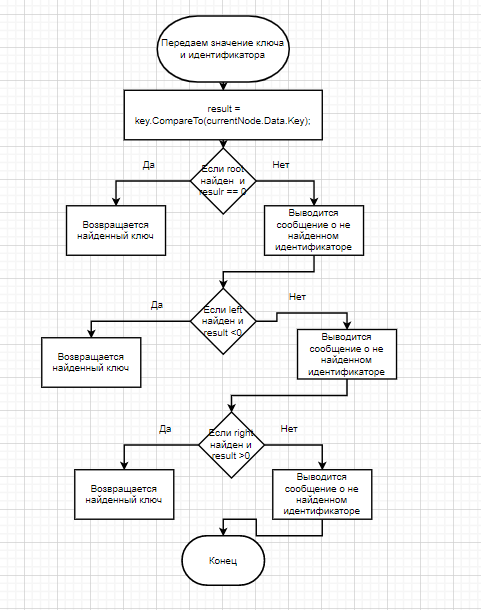


Рисунок 3 – Вывод времени поиска

**Блок-схемы**

****Рисунок 4 – функция поиска в бинарном дереве

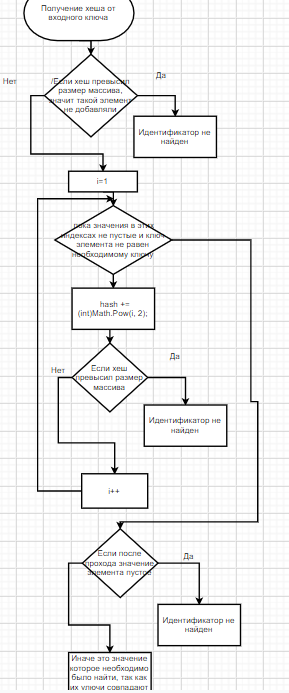
****

Рисунок 5 –функция поиска в хешированном массиве

**Листинг программы**

BinarySearchThree.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab2

{

/// <summary>

/// Расположения узла относительно родителя

/// </summary>

public enum Side

{

Left,

Right

}

// класс, представляющий обход дерева

public class BinaryTreeNode

{

public BinaryTreeNode(string data)

{

Data = data;

}

public string Data { get; set; }

// левый узел дерева

public BinaryTreeNode LeftNode { get; set; }

// правый узел дерева

public BinaryTreeNode RightNode { get; set; }

// корень дерева

public BinaryTreeNode ParentNode { get; set; }

public Side? NodeSide =>

ParentNode == null

? (Side?)null

: ParentNode.LeftNode == this

? Side.Left

: Side.Right;

public override string ToString() => Data.ToString();

}

public class BinaryTree

{

public BinaryTreeNode RootNode { get; set; }

public BinaryTreeNode Add(BinaryTreeNode node, BinaryTreeNode currentNode = null)

{

if (RootNode == null)

{

node.ParentNode = null;

return RootNode = node;

}

currentNode = currentNode ?? RootNode;

node.ParentNode = currentNode;

int result;

return (result = node.Data.CompareTo(currentNode.Data)) == 0 ? currentNode : result < 0 ? currentNode.LeftNode == null

? (currentNode.LeftNode = node)

: Add(node, currentNode.LeftNode)

: currentNode.RightNode == null

? (currentNode.RightNode = node)

: Add(node, currentNode.RightNode);

}

// функция добавления значения в дерево

public BinaryTreeNode Add(string data)

{

return Add(new BinaryTreeNode(data));

}

public BinaryTreeNode FindNode(string data, BinaryTreeNode startWithNode = null)

{

startWithNode = startWithNode ?? RootNode;

int result;

return (result = data.CompareTo(startWithNode.Data)) == 0

? startWithNode

: result < 0

? startWithNode.LeftNode == null

? null

: FindNode(data, startWithNode.LeftNode)

: startWithNode.RightNode == null

? null

: FindNode(data, startWithNode.RightNode);

}

}

}

Hashtable.cs

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab2

{

class Hashtable

{

private const int Size = 100;

private readonly string[] Data = new string[Size];

private List<string> colisions = new List<string>();

static public int indTable = 0;

// число хэш-значений (входов хэш-таблицы).

private const int A = 5;

public Hashtable()

{

for (int i = 0; i < Size; i++)

colisions.Add("");

}

// модульный метод хэширования

private int GetHash(string value)

{

// массив кодов символов строки

var chars = value.ToCharArray();

// сумма кодов юникода

var h = 0;

foreach (var c in chars)

{

h += c;

}

h = (h % Size);

return h;

}

// разоешение колизий

// value - квадрат числа (1,4,9...)

private int SolveCollision(int index,int value)

{

var result = (index + value) % Size;

return result;

}

public void Add(string value, bool clr = false)

{

var index = GetHash(value);

int i = 1;

if (Data[index] == null)

{

Data[index] = value;

colisions[index] = ("отсуствует");

indTable = index;

}

else

{

do

{

index = SolveCollision(index, i \* i);

colisions[index] = ("присуствует");

i++;

}

while (Data[index] != null);

Data[index] = value;

indTable = index;

}

}

public long Find(string value)

{

var stopwatch = Stopwatch.StartNew();

var index = GetHash(value);

int i = 1;

if (Data[index] == null)

{

return -1;

}

else if (Data[index] == value)

{

stopwatch.Stop();

return stopwatch.ElapsedTicks;

}

else

{

while (Data[index] != value)

{

index = SolveCollision(index,i\*i);

i++;

if (Data[index] == null)

{

return -1;

}

}

stopwatch.Stop();

return stopwatch.ElapsedTicks;

}

}

public List<MyDictionary> Output()

{

var list = new List<MyDictionary>();

for (var i = 0; i < Size; i++)

{

if (Data[i] == null)

{

list.Add(new MyDictionary()

{

Key = i,

Value = string.Empty,

});

}

else

{

list.Add(new MyDictionary()

{

Key = i,

Value = Data[i],

Colisions = colisions[i],

});

}

}

return list;

}

public class MyDictionary

{

public int Key { get; set; }

public string Value { get; set; }

public string Colisions { get; set; }

}

}

}

MainWindow.xaml.cs

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using static lab2.Hashtable;

namespace lab2

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

BinaryTree binaryTree = new BinaryTree();

private const string FilePath = "identif.txt";

private Hashtable Hashtable = new Hashtable();

public MainWindow()

{

InitializeComponent();

//taskTekst.Text = "9. Первый метод: таблица идентификаторов – бинарное дерево, обход Left-Right-Root." +

// "Второй метод: таблица идентификаторов – массив, полученный методом хеширования.Тип ключа – строка текста произвольной длины." +

// " Преобразование строки – конкатенация битовых образов символов." +

// " Метод хеширования – модульный.Метод разрешения коллизий – квадратичный.";

}

private void ReadFile\_Click(object sender, RoutedEventArgs e)

{

using (var reader = new StreamReader(FilePath))

{

string line;

Hashtable = new Hashtable();

binaryTree = new BinaryTree();

while ((line = reader.ReadLine()) != null)

{

Hashtable.Add(line);

binaryTree.Add(line);

}

}

dataGrid.ItemsSource = Hashtable.Output();

dataGrid.Columns[0].Width = 40;

dataGrid.Columns[1].Width = 200;

dataGrid.Columns[2].Width = 100;

dataGrid.Columns[0].Header = "Индекс";

dataGrid.Columns[1].Header = "Идентификатор";

dataGrid.Columns[2].Header = "Колизия";

}

private void SearchButton\_Click(object sender, RoutedEventArgs e)

{

var text = name.Text;

if (text == string.Empty)

{

MessageBox.Show("Введите имя идентификатора");

return;

}

var first = Hashtable.Find(text);

var stopwatch = Stopwatch.StartNew();

var second = binaryTree.FindNode(text);

var end = stopwatch.ElapsedTicks;

var result = string.Empty;

if (first > 0 && end > 0)

{

result = $"Результат поиска - идентификатор найден\r\nВремя поиска в хештаблице - {first}\r\nВремя поиска в дереве - {end}";

var index = dataGrid.ItemsSource.Cast<MyDictionary>().First(item => item.Value == text).Key;

dataGrid.SelectedIndex = index;

}

else

{

result = "Ничего не найдено";

}

MessageBox.Show( result);

}

private void AddButton\_Click(object sender, RoutedEventArgs e)

{

var text = AddBlock.Text;

if (text == string.Empty)

{

MessageBox.Show("Введите имя идентификатора");

return;

}

Hashtable.Add(text, true);

binaryTree.Add(text);

dataGrid.ItemsSource = Hashtable.Output();

dataGrid.Columns[0].Width = 40;

dataGrid.Columns[1].Width = 200;

dataGrid.Columns[2].Width = 100;

//dataGrid.Columns[3].Width = 15;

dataGrid.Columns[0].Header = "Индекс";

dataGrid.Columns[1].Header = "Идентификатор";

dataGrid.Columns[2].Header = "Коллизия";

dataGrid.SelectedIndex = Hashtable.indTable;

}

}

}

**Вывод**: изучил основные методы организации таблиц идентификаторов, получил представление о преимуществах и недостатках, присущих различным методам организации таблиц идентификаторов.