**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

# по дисциплине «Методы трансляции»

на тему: «**Организация таблиц идентификаторов**»

Выполнил: студент гр. ИП-32

Кирпиченко Д.Д.

Принял: доцент

Кравченко О.А.

Дата сдачи отчета:

Гомель 2022

***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.

3. Первый метод: таблица идентификаторов – линейный односвязный нециклический список.

Второй метод: таблица идентификаторов – массив, полученный методом хеширования. Тип ключа – целое число из отрезка  
[0, +1 000 000 000]. Метод хеширования – выбор цифр. Метод разрешения коллизий – двойное хеширование.

Приложение было разработано на языке C#

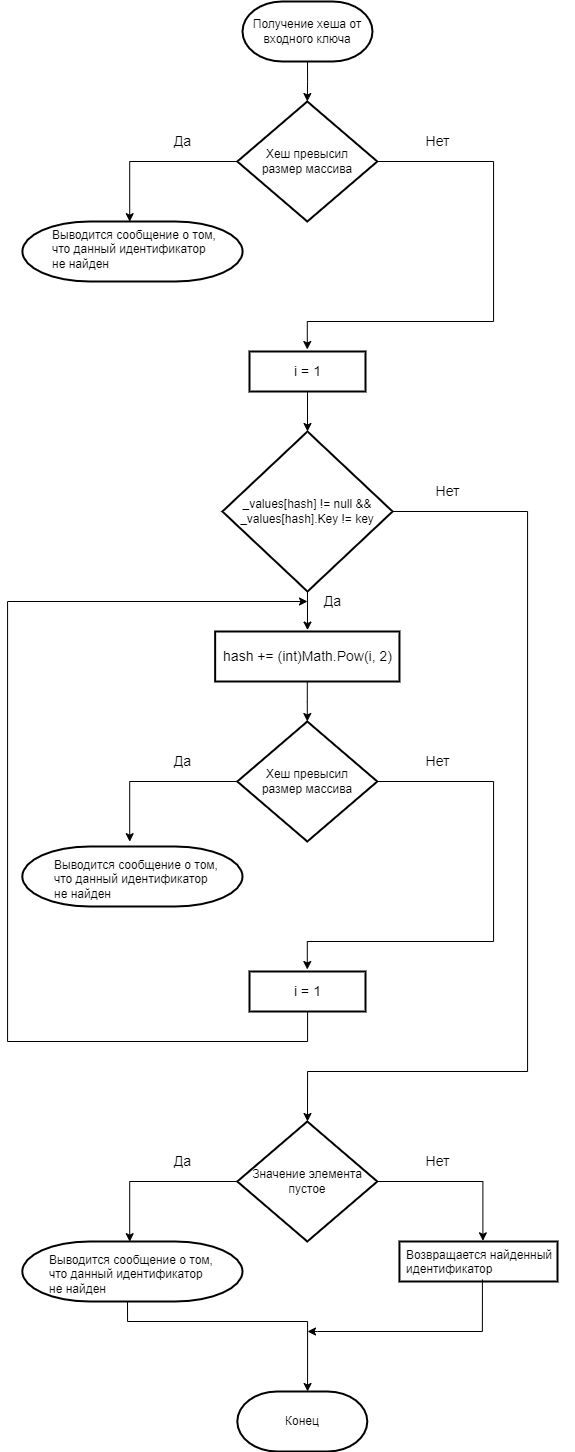


Рисунок 1 –Алгоритм поиска в хеш таблицы

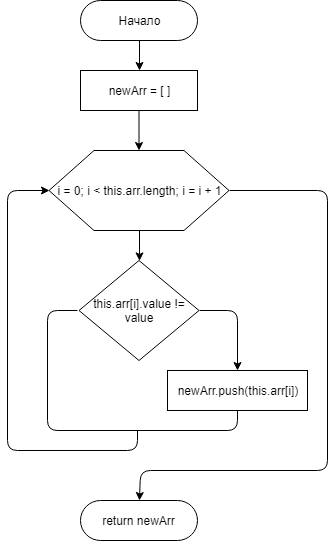
****

Рисунок 2 –Алгоритм поиска в списке

***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 списка и таблицы идентификаторов пользователь необходимо нажать на кнопку и запалняется список и хэш-таблица “Считать из файла”. После чего вызовутся функции ReadFile\_Click и запалняется список и хэш-таблица.

2. Ввод нового идентификатора.

Для добавления нового идентификатора необходимо ввести его название и нажать на кнопку “Добавить”. После чего вызывается функция AddButton\_Click, после чего происходит добавление в список и хэш-таблицу.

Алгоритм добавления идентификатора в таблицу:

1) Ключ получается путем преобразования символов строки в число и суммирование четных чисел.

2) Решение коллизии считается по формуле: , где *size* это размер хэш таблицы, valueэто значение *GetHash* и *GetHash2* это функции хеширования, которые считают сумму четных и нечетных элементов соответственно, i это количество раз произошедшей коллизии .

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

4) Если эта ячейка занята, то запускается цикл в котором к текущему индексу добавляется единица и проверяется новая ячейка. Если ячейка свободна , то в массив добавляется элемент в новую ячейку.

3. Вывод информации о месте нахождения введенного идентификатора в существующей таблице или включение в таблицу нового идентификатора, если его нет в таблице;

Для поиска информации необходимо ввести идентификатор и нажать на кнопку “Поиск”

Функция SearchButton\_Click обрабатывает нажатие на кнопку. После выполнения функции будет выведена информация о идентификаторе если он существует, иначе добавит в список и таблицу. Также функция высчитывает время поиска в хэш-таблице и списке.

**Тесты для добавления:**

|  |  |  |
| --- | --- | --- |
| Идентификатор | Результат | Ожидаемый результат |
| Test1 | Идентификатор добавлен | Идентификатор добавлен |
| Test2 | Идентификатор добавлен | Идентификатор добавлен |

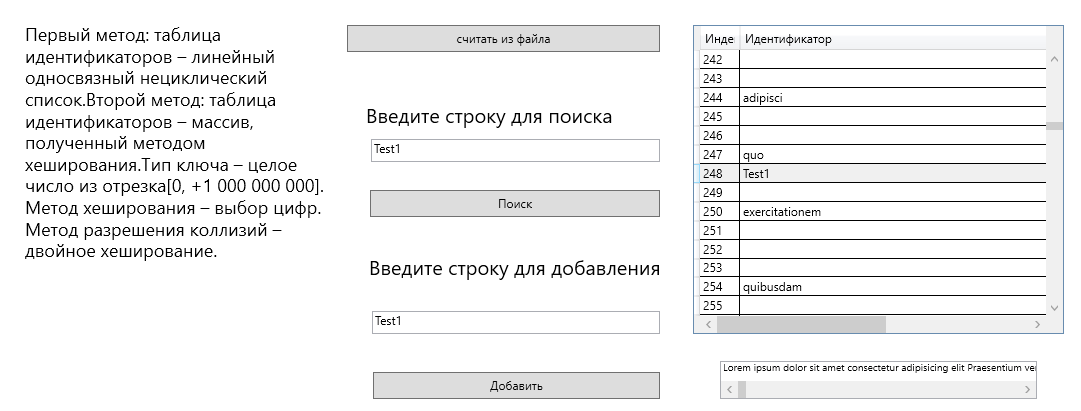


Рисунок 3 – Добавление идентификатора

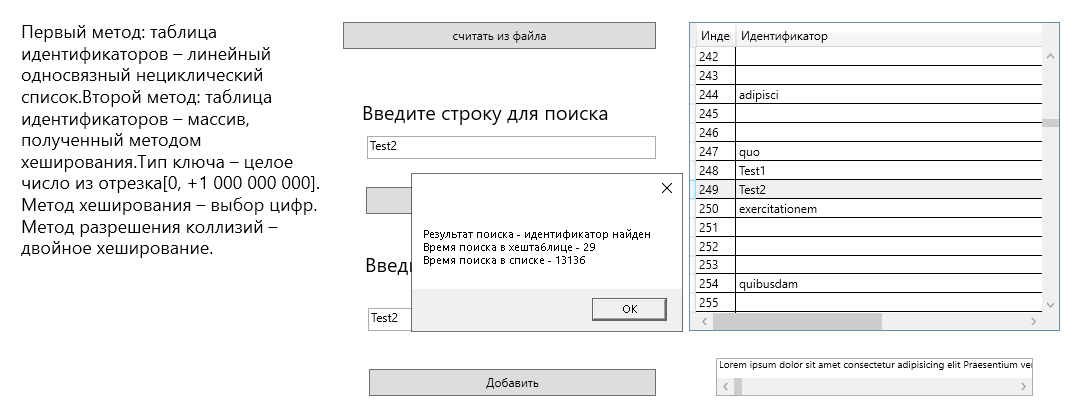


Рисунок 4 – Вывод времени поиска

**Листинг программы**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab2

{

class Hashtable

{

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

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

private const int Size = 1000;

private const int A = 5;

public Hashtable()

{

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

{

colision.Add("");

}

}

private int GetHash(string value)

{

var chars = value.ToCharArray();

var h = 0;

for(int i = 0; i < chars.Length; i = i + 2)

{

h += chars[i];

}

h=(h%Size);

return h;

}

private int GetHash2(string value)

{

var chars = value.ToCharArray();

var h = 0;

for (int i = 0; i < chars.Length; i = i + 1)

{

h += chars[i];

}

h = (h % Size);

return h;

}

private int SolveCollision(int i,string value)

{

var result = GetHash(value) + i \* GetHash2(value);

return result % Size;

}

public void Add(string value)

{

var index = GetHash(value);

int i = 1;

if (Data[index] == null)

{

Data[index] = value;

colision[index] = "Колизии нет";

}

else

{

do

{

int old\_index = index;

index = SolveCollision(i,value);

colision[index] = "Колизии есть: " + Data[old\_index];

i++;

}

while (Data[index] != null);

Data[index] = value;

}

}

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(i,value);

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],

Colision = colision[i]

});

}

}

return list;

}

public class MyDictionary

{

public int Key { get; set; }

public string Value { get; set; }

public string Colision { get; set; }

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab2

{

class List

{

Node Head { get; set; }

private int Count = 0;

public Node this[int index]

{

get => GetNode(index);

}

public void Add(string value)

{

if (Head == null)

{

Head = new Node

{

Value = value,

};

return;

}

else

{

var current = Head;

while (current.Next != null)

{

current = current.Next;

}

current.Next = new Node

{

Value = value,

};

Count++;

}

}

public void Delete(int index)

{

if (index < 0)

{

Console.WriteLine("Wrong index");

return;

}

var current = Head;

var previous = Head;

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

{

current = current.Next;

if (i != 0)

{

previous = previous.Next;

}

}

if (current.Next == null)

{

previous.Next = null;

}

else

{

previous.Next = current.Next;

}

Count--;

}

public string Output()

{

string text = "";

if (Head == null)

{

return "List is empty";

}

var current = Head;

while (current != null)

{

text += current.Value + " ";

current = current.Next;

}

return text;

}

//public void Sort()

//{

// for (int i = 0; i < Count + 1; i++)

// {

// for (int j = 0; j < Count + 1; j++)

// {

// if (this[j].Value > this[j + 1].Value)

// {

// Swap(this[j], this[j + 1]);

// }

// }

// }

//}

public Node GetNode(int index)

{

var current = Head;

for (var i = 0; i < index - 1; i++)

{

current = current.Next;

}

return current;

}

public void Swap(Node left, Node right)

{

Node prevLeft = null;

var currentLeft = Head;

var currentRight = Head;

while (currentLeft != left)

{

prevLeft = currentLeft;

currentLeft = currentLeft.Next;

}

while (currentRight != right)

{

currentRight = currentRight.Next;

}

if (prevLeft != null)

{

left.Next = right.Next;

prevLeft.Next = right;

right.Next = left;

}

else

{

left.Next = right.Next;

Head = right;

Head.Next = left;

}

}

public bool Search(string value)

{

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

{

if(this[i].Value == value)

{

return true;

}

}

return false;

}

}

}

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

{

List lst = new List();

private const string FilePath = "identif.txt";

private Hashtable Hashtable = new Hashtable();

public MainWindow()

{

InitializeComponent();

Task.Text = "Первый метод: таблица идентификаторов – линейный односвязный нециклический список."+

"Второй метод: таблица идентификаторов – массив, полученный методом хеширования.Тип ключа – целое число из отрезка"+

"[0, +1 000 000 000]. Метод хеширования – выбор цифр. Метод разрешения коллизий – двойное хеширование."

;

}

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

{

using (var reader = new StreamReader(FilePath))

{

string line;

Hashtable = new Hashtable();

lst = new List();

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

{

Hashtable.Add(line);

lst.Add(line);

}

}

ListOut.Text = lst.Output();

dataGrid.ItemsSource = Hashtable.Output();

dataGrid.Columns[0].Width = 40;

dataGrid.Columns[1].Width = 150;

dataGrid.Columns[2].Width = 150;

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 = lst.Search(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 списке - {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);

lst.Add(text);

dataGrid.ItemsSource = Hashtable.Output();

dataGrid.Columns[0].Width = 40;

dataGrid.Columns[1].Width = 440;

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

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

}

}

}

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