**Министерство науки и высшего образования Российской Федерации** **Федеральное государственное бюджетное образовательное учреждение высшего образования** **«Московский государственный технический университет** **имени Н. Э. Баумана** **(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Радиотехнический»**

**Кафедра «Системы обработки информации и управления»**

Отчет по домашнему заданию

по курсу БКИТ

Выполнила:

Студентка Группы РТ5-31Б   
Веденеева Екатерина Ильинична

подпись, дата

Проверил:

подпись, дата

2020г.

1. **Задание**

Разработать программу, реализующую вычисление расстояния Левенштейна с использованием алгоритма Вагнера-Фишера.

1. Программа должна быть разработана в виде библиотеки классов на языке C#.
2. Использовать самый простой вариант алгоритма без оптимизации.
3. Дополнительно возможно реализовать вычисление расстояния Дамерау-Левенштейна (с учетом перестановок соседних символов).
4. Модифицировать предыдущую лабораторную работу, вместо поиска подстроки используется вычисление расстояния Левенштейна.
5. Предусмотреть отдельное поле ввода для максимального расстояния. Если расстояние Левенштейна между двумя строками больше максимального, то строки считаются несовпадающими и не выводятся в список результатов.
6. **Код**

Файл Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab4Sem3

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

Файл Form1.cs

using System;

using System.IO; // Для работы с файлами

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Diagnostics;

namespace Lab4Sem3

{

public partial class Form1 : Form

{

/// <summary>

/// Выполняется в параллельном потоке для поиска строк

/// </summary>

public static List<PSearchRes> ArrayThreadTask(object paramObj)

{

PSTP param = (PSTP)paramObj;

//Слово для поиска в верхнем регистре

string wordUpper = param.wordPattern.Trim().ToUpper();

//Результаты поиска в одном потоке

List<PSearchRes> Result = new List<PSearchRes>();

//Перебор всех слов во временном списке данного потока

foreach (string str in param.tempList)

{

//Вычисление расстояния Дамерау-Левенштейна

int dist = EditDistance.Distance(str.ToUpper(), wordUpper);

//Если расстояние меньше порогового, то слово добавляется врезультат

if (dist <= param.maxDist)

{

PSearchRes temp = new PSearchRes()

{

word = str,

dist = dist,

ThreadNum = param.ThreadNum

};

Result.Add(temp);

}

}

return Result;

}

public Form1()

{

InitializeComponent();

}

/// <summary>

/// Список слов

/// </summary>

List<string> list = new List<string>();

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

{

OpenFileDialog fd = new OpenFileDialog();

fd.Filter = "текстовые файлы|\*.txt";

if (fd.ShowDialog() == DialogResult.OK)

{

Stopwatch t = new Stopwatch();

t.Start();

//Чтение файла в виде строки

string text = File.ReadAllText(fd.FileName);

//Разделительные символы для чтения из файла

char[] separators = new char[] { ' ', '.', ',', '!', '?', '/', '\t', '\n' };

string[] textArray = text.Split(separators);

foreach (string strTemp in textArray)

{

//Удаление пробелов в начале и конце строки

string str = strTemp.Trim();

//Добавление строки в список, если строка не содержится в списке

if (!list.Contains(str)) list.Add(str);

}

t.Stop();

this.textBoxFileReadTime.Text = t.Elapsed.ToString();

this.textBoxFileReadCount.Text = list.Count.ToString();

}

else

{

MessageBox.Show("Необходимо выбрать файл");

}

}

private void label1\_Click(object sender, EventArgs e)

{

}

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

{

// Чтение слова для поиска

string word = this.textBoxFind.Text.Trim();

// Если поле поиска (и слово соответственно) не пусто

if (!string.IsNullOrWhiteSpace(word) && list.Count > 0)

{

string wordUpper = word.ToUpper();

// Для хранения временных рещультатов поиска

List<string> tempList = new List<string>();

// Запуск таймера

Stopwatch t = new Stopwatch();

t.Start();

foreach (string str in list)

{

if (str.ToUpper().Contains(wordUpper))

{

tempList.Add(str);

}

}

t.Stop();

this.textBoxExactTime.Text = t.Elapsed.ToString();

this.listBoxResult.BeginUpdate();

//Очистка списка

this.listBoxResult.Items.Clear();

//Вывод результатов поиска

foreach (string str in tempList)

{

this.listBoxResult.Items.Add(str);

}

this.listBoxResult.EndUpdate();

}

else

{

MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");

}

}

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

{

}

private void button1\_Click(object sender, EventArgs e)

{

// Слово для поиска

string searchWord = this.textBoxFind.Text.Trim();

this.listBoxResult.Items.Add(searchWord);

this.listBoxResult.Items.Add(list.Count);

//Если поисковая строка не пустая

if (!string.IsNullOrWhiteSpace(searchWord) && list.Count > 0)

{

int maxL;

// Проверка на дуррака

if (!int.TryParse(this.textBoxMaxL.Text.Trim(), out maxL))

{

MessageBox.Show("Не указано максимальное расстояние. Заполните поле.");

return;

}

// Проверка на дурака (2)

if (maxL < 1 || maxL > 5)

{

MessageBox.Show("Максимальное расстояние должно быть в диапазоне от 1 до 5");

return;

}

//Проверка на слепого дурака (заполнение количества потоков)

int sQuant= Convert.ToInt32(this.textBoxStreams.Text.Trim());

// Таймер. Старт.

Stopwatch timer = new Stopwatch();

timer.Start();

/////////////////////////// НАЧАЛО ПОИСКА //////////////////////////

//Список для хранения результатов.

List<PSearchRes> Ress = new List<PSearchRes>();

// Разбиваем список на фрагменты для хапуска в разных потоках

List<MinMax> arrDividedList = SubArrays.DivideSubArrays(0, list.Count, sQuant);

int count = arrDividedList.Count;

//Количество потоков соответствует количеству фрагментов массива

Task<List<PSearchRes>>[] tasks = new Task<List<PSearchRes>>[count];

//Запуск потоков

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

{

//Создание временного списка, чтобы потоки не работали параллельно с одной коллекцией

List<string> tempTaskList = list.GetRange(arrDividedList[i].Min,

arrDividedList[i].Max - arrDividedList[i].Min);

tasks[i] = new Task<List<PSearchRes>>(

//Метод, который будет выполняться в потоке

ArrayThreadTask,

//Параметры потока

new PSTP()

{

tempList = tempTaskList,

maxDist = maxL,

ThreadNum = i,

wordPattern = searchWord

});

//Запуск потока

tasks[i].Start();

}

Task.WaitAll(tasks);

timer.Stop();

//Объединение результатов

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

{

Ress.AddRange(tasks[i].Result);

}

///////////////////////// ЗАВЕРШЕНИЕ ПОИИСКА /////////////////////////

timer.Stop();

///////////////////// ВЫВОД РЕЗУЛЬТАТОВ /////////////////////////////

// Врепя поиска совпадений

this.textBoxSTimer.Text = timer.Elapsed.ToString();

// Вычисленное кол-во потоков

this.textBoxSQuant.Text = sQuant.ToString();

//Очистка списка

this.listBoxResult.Items.Clear();

// Вывод результатов поиска в форму

foreach (var x in Ress)

{

string tmp = x.word + "(расстояние=" + x.dist.ToString() + " поток=" + x.ThreadNum.ToString() + ")";

this.listBoxResult.Items.Add(tmp);

}

// Окончание обновления списка результатов

this.listBoxResult.EndUpdate();

}

else

{

MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");

}

}

private void textBoxStreams\_TextChanged(object sender, EventArgs e)

{

}

}

}

Файл PSTP.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab4Sem3

{

class PSTP

{

/// <summary>

/// Массив для поиска

/// </summary>

public List<string> tempList { get; set; }

/// <summary>

/// Слово для поиска

/// </summary>

public string wordPattern { get; set; }

/// <summary>

/// Максимальное расстояние для нечеткого поиска

/// </summary>

public int maxDist { get; set; }

/// <summary>

/// Номер потока

/// </summary>

public int ThreadNum { get; set; }

}

}

Файл EditindDistance.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab4Sem3

{

public static class EditDistance

{

/// <summary>

/// Вычисление расстояния Дамерау-Левенштейна

/// </summary>

public static int Distance(string str1Param, string str2Param)

{

if ((str1Param == null) || (str2Param == null)) return -1;

int str1Len = str1Param.Length;

int str2Len = str2Param.Length;

//Если хотя бы одна строка пустая, возвращается длина другой строки

if ((str1Len == 0) && (str2Len == 0)) return 0;

if (str1Len == 0) return str2Len;

if (str2Len == 0) return str1Len;

//Приведение строк к верхнему регистру

string str1 = str1Param.ToUpper();

string str2 = str2Param.ToUpper();

//Объявление матрицы

int[,] matrix = new int[str1Len + 1, str2Len + 1];

//Инициализация нулевой строки и нулевого столбца матрицы

for (int i = 0; i <= str1Len; i++) matrix[i, 0] = i;

for (int j = 0; j <= str2Len; j++) matrix[0, j] = j;

//Вычисление расстояния Дамерау-Левенштейна

for (int i = 1; i <= str1Len; i++)

{

for (int j = 1; j <= str2Len; j++)

{

//Эквивалентность символов, переменная symbEqual соответствует m(s1[i],s2[j])

int symbEqual = ((str1.Substring(i - 1, 1) == str2.Substring(j - 1, 1)) ? 0 : 1);

int ins = matrix[i, j - 1] + 1; //Добавление

int del = matrix[i - 1, j] + 1; //Удаление

int subst = matrix[i - 1, j - 1] + symbEqual; //Замена

//Элемент матрицы вычисляется как минимальный из трех случаев

matrix[i, j] = Math.Min(Math.Min(ins, del), subst);

//Дополнение Дамерау по перестановке соседних символов

if ((i > 1) && (j > 1) &&

(str1.Substring(i - 1, 1) == str2.Substring(j - 2, 1)) &&

(str1.Substring(i - 2, 1) == str2.Substring(j - 1, 1)))

{

matrix[i, j] = Math.Min(matrix[i, j], matrix[i - 2, j - 2] + symbEqual);

}

}

}

//Возвращается нижний правый элемент матрицы

return matrix[str1Len, str2Len];

}

/// <summary>

/// Вывод расстояния Дамерау-Левенштейна в консоль

/// </summary>

public static void WriteDistance(string str1Param, string str2Param)

{

int d = Distance(str1Param, str2Param);

Console.WriteLine("'" + str1Param + "','" + str2Param + "' -> " + d.ToString());

}

}

}

Файл MinMax.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab4Sem3

{

/// <summary>

/// Хранение минимального и максимального значений диапазона

/// </summary>

public class MinMax

{

public int Min { get; set; }

public int Max { get; set; }

public MinMax(int pmin, int pmax)

{

this.Min = pmin;

this.Max = pmax;

}

}

/// <summary>

/// Класс для деления массива на последовательности

/// </summary>

public static class SubArrays

{

/// <summary>

/// Деление массива на последовательности

/// </summary>

/// <param name="beginIndex">Начальный индекс массива</param>

/// <param name="endIndex">Конечный индекс массива</param>

/// <param name="subArraysCount">Требуемое количество подмассивов</param>

/// <returns>Список пар с индексами подмассивов</returns>

public static List<MinMax> DivideSubArrays(int beginIndex, int endIndex, int subArraysCount)

{

//Результирующий список пар с индексами подмассивов

List<MinMax> result = new List<MinMax>();

//Если число элементов в массиве слишком мало для деления

//то возвращается массив целиком

if ((endIndex - beginIndex) <= subArraysCount)

{

result.Add(new MinMax(0, (endIndex - beginIndex)));

}

else

{

//Размер подмассива

int delta = (endIndex - beginIndex) / subArraysCount;

//Начало отсчета

int currentBegin = beginIndex;

//Пока размер подмассива укладывается в оставшуюся

//последовательность

while ((endIndex - currentBegin) >= 2 \* delta)

{

//Формируем подмассив на основе начала

//последовательности

result.Add(

new MinMax(currentBegin, currentBegin + delta));

//Сдвигаем начало последовательности

//вперед на размер подмассива

currentBegin += delta;

}

//Оставшийся фрагмент массива

result.Add(new MinMax(currentBegin, endIndex));

}

//Возврат списка результатов

return result;

}

}

}

Файл PSearchR.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab4Sem3

{

/// <summary>

/// Результаты параллельного поиска

/// </summary>

public class PSearchRes

{

/// <summary>

/// Найденное слово

/// </summary>

public string word { get; set; }

/// <summary>

/// Расстояние

/// </summary>

public int dist { get; set; }

/// <summary>

/// Номер потока

/// </summary>

public int ThreadNum { get; set; }

}

}

**3. Результат выполнения программы**

