Московский государственный университет имени Н. Э. Баумана

Отчет по лабораторным работам №4-5

«**Работа с файлами. Вычисление расстояния Левенштейна с использованием алгоритма Вагнера-Фишера**»

по предмету Базовые компоненты интернет-технологий

Выполнил:

студент группы ИУ5-34Б

Прус Ева

Москва, 2020 г.

Описание задания

**Лабораторная работа №4**

Разработать программу, реализующую работу с файлами.

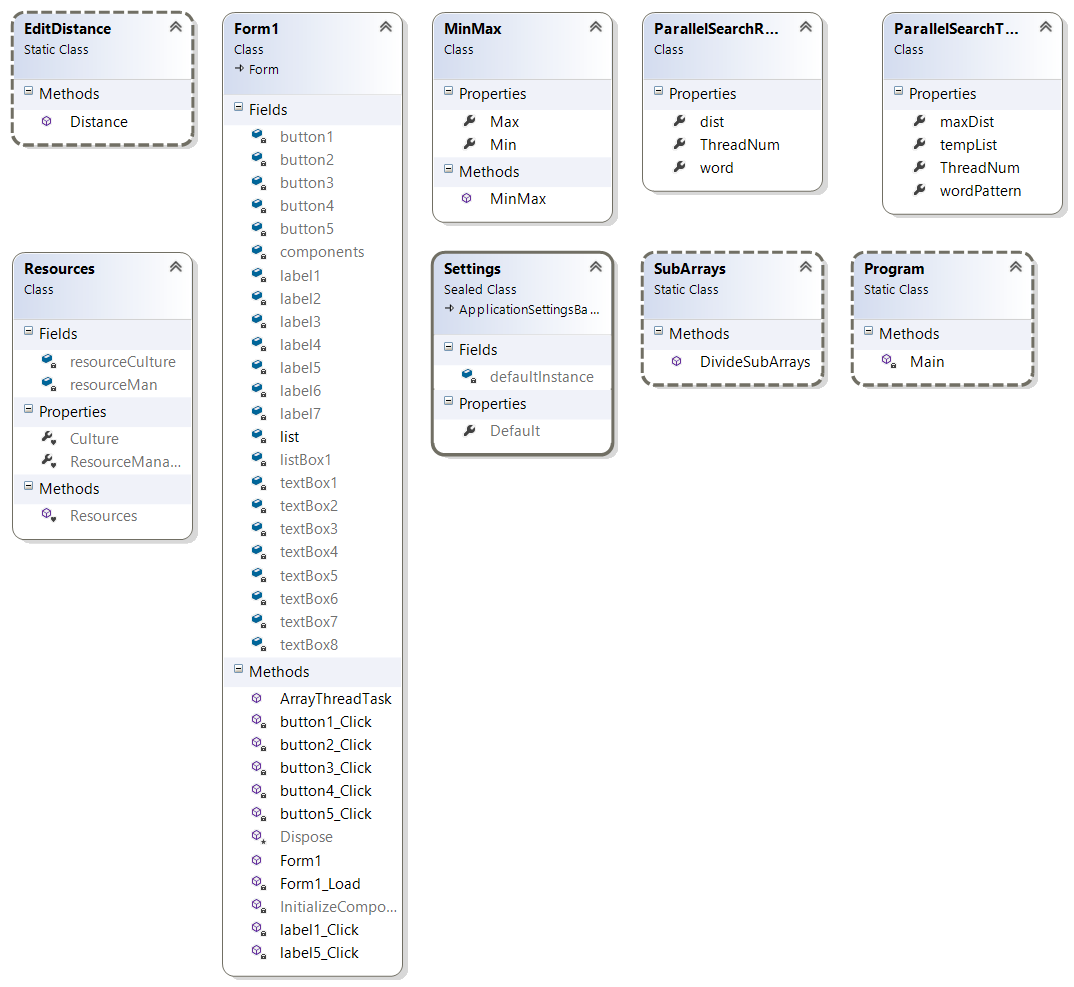
1. Программа должна быть разработана в виде приложения Windows Forms на языке C#. По желанию вместо Windows Forms возможно использование WPF.
2. Добавить кнопку, реализующую функцию чтения файла в список слов List<string>.
3. Для выбора имени файла используется класс OpenFileDialog, который открывает диалоговое окно с выбором файла. Ограничить выбор только файлами с расширением «.txt».
4. Для чтения из файла рекомендуется использовать статический метод ReadAllText() класса File (пространство имен System.IO). Содержимое файла считывается методом ReadAllText() в виде одной строки, далее делится на слова с использованием метода Split() класса string. Слова сохраняются в список List<string>.
5. При сохранении слов в список List<string> дубликаты слов не записываются. Для проверки наличия слова в списке используется метод Contains().
6. Вычислить время загрузки и сохранения в список с использованием класса Stopwatch (пространство имен System.Diagnostics). Вычисленное время вывести на форму в поле ввода (TextBox) или надпись (Label).
7. Добавить на форму поле ввода для поиска слова и кнопку поиска. При нажатии на кнопку поиска осуществлять поиск введенного слова в списке. Слово считается найденным, если оно входит в элемент списка как подстрока (метод Contains() класса string).
8. Добавить на форму список (ListBox). Найденные слова выводить в список с использованием метода «название\_списка.Items.Add()». Вызовы метода «название\_списка.Items.Add()» должны находится между вызовами методов «название\_списка.BeginUpdate()» и «название\_списка. EndUpdate()».
9. Вычислить время поиска с использованием класса Stopwatch. Вычисленное время вывести на форму в поле ввода (TextBox) или надпись (Label).

**Лабораторная работа №5**

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.

Диаграмма классов

****

Текст программы

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

}

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

private void button1\_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();

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

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

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

foreach (string strTemp in textArray)

{

string str = strTemp.Trim();

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

}

t.Stop();

this.textBox1.Text = t.Elapsed.ToString();

this.textBox2.Text = list.Count.ToString();

}

else

{

MessageBox.Show("Необходимо выбрать файл");

}

}

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

{

}

private void button2\_Click(object sender, EventArgs e)

{

string word = this.textBox3.Text.Trim();

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

{

string wordUpper = word.ToUpper();

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

Stopwatch t = new Stopwatch();

t.Start();

foreach (string str in list)

{

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

{

tempList.Add(str);

}

}

t.Stop();

this.textBox4.Text = t.Elapsed.ToString();

this.listBox1.BeginUpdate();

this.listBox1.Items.Clear();

foreach (string str in tempList)

{

this.listBox1.Items.Add(str);

}

this.listBox1.EndUpdate();

}

else

{

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

}

}

private void button3\_Click(object sender, EventArgs e)

{

string word = this.textBox3.Text.Trim();

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

{

int maxDist;

if (!int.TryParse(this.textBox5.Text.Trim(), out maxDist))

{

MessageBox.Show("Необходимо указать максимальное расстояние");

return;

}

if (maxDist < 1 || maxDist > 5)

{

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

return;

}

int ThreadCount;

if (!int.TryParse(this.textBox6.Text.Trim(), out ThreadCount))

{

MessageBox.Show("Необходимо указать количество потоков");

return;

}

Stopwatch timer = new Stopwatch();

timer.Start();

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

List<MinMax> arrayDivList = SubArrays.DivideSubArrays(0, list.Count, ThreadCount);

int count = arrayDivList.Count;

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

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

{

List<string> tempTaskList = list.GetRange(arrayDivList[i].Min, arrayDivList[i].Max - arrayDivList[i].Min);

tasks[i] = new Task<List<ParallelSearchResult>>(ArrayThreadTask,

new ParallelSearchThreadParam()

{

tempList = tempTaskList,

maxDist = maxDist,

ThreadNum = i,

wordPattern = word

});

tasks[i].Start();

}

Task.WaitAll(tasks);

timer.Stop();

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

{

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

}

timer.Stop();

this.textBox8.Text = timer.Elapsed.ToString();

//Вычисленное количество потоков

this.textBox7.Text = count.ToString();

//Начало обновления списка результатов

this.listBox1.BeginUpdate();

//Очистка списка

this.listBox1.Items.Clear();

foreach (var x in Result)

{

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

this.listBox1.Items.Add(temp);

}

this.listBox1.EndUpdate();

}

else

{

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

}

}

public static List<ParallelSearchResult> ArrayThreadTask(object paramObj)

{

ParallelSearchThreadParam param = (ParallelSearchThreadParam)paramObj;

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

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

foreach (string str in param.tempList)

{

//Вычисление расстояния Дамерау-Левенштейна

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

if (dist <= param.maxDist)

{

ParallelSearchResult temp = new ParallelSearchResult()

{

word = str,

dist = dist,

ThreadNum = param.ThreadNum

};

Result.Add(temp);

}

}

return Result;

}

private void label5\_Click(object sender, EventArgs e)

{

}

private void button4\_Click(object sender, EventArgs e)

{

string TempReportFileName = "Report\_" + DateTime.Now.ToString("dd\_MM\_yyyy\_hhmmss");

//Диалог сохранения файла отчета

SaveFileDialog fd = new SaveFileDialog();

fd.FileName = TempReportFileName;

fd.DefaultExt = ".html";

fd.Filter = "HTML Reports|\*.html";

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

{

string ReportFileName = fd.FileName;

//Формирование отчета

StringBuilder b = new StringBuilder();

b.AppendLine("<html>");

b.AppendLine("<head>");

b.AppendLine("<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'/>");

b.AppendLine("<title>" + "Отчет: " + ReportFileName + "</title>");

b.AppendLine("</head>");

b.AppendLine("<body>");

b.AppendLine("<h1>" + "Отчет: " + ReportFileName + "</h1>");

b.AppendLine("<table border='1'>");

b.AppendLine("<tr>");

b.AppendLine("<td>Время чтения из файла</td>");

b.AppendLine("<td>" + this.textBox1.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Количество уникальных слов в файле</td>");

b.AppendLine("<td>" + this.textBox2.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Слово для поиска</td>");

b.AppendLine("<td>" + this.textBox3.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Максимальное расстояние для нечеткого поиска</td>");

b.AppendLine("<td>" + this.textBox5.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Время четкого поиска</td>");

b.AppendLine("<td>" + this.textBox4.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Время нечеткого поиска</td>");

b.AppendLine("<td>" + this.textBox8.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr valign='top'>");

b.AppendLine("<td>Результаты поиска</td>");

b.AppendLine("<td>");

b.AppendLine("<ul>");

foreach (var x in this.listBox1.Items)

{

b.AppendLine("<li>" + x.ToString() + "</li>");

}

b.AppendLine("</ul>");

b.AppendLine("</td>");

b.AppendLine("</tr>");

b.AppendLine("</table>");

b.AppendLine("</body>");

b.AppendLine("</html>");

//Сохранение файла

File.AppendAllText(ReportFileName, b.ToString());

MessageBox.Show("Отчет сформирован. Файл: " + ReportFileName);

}

}

private void button5\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

EditDistance.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApplication1

{

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];

}

}

}

MinMax.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApplication1

{

/// <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;

}

}

}

ParallelSearchResult.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace WindowsFormsApplication1

{

/// <summary>

/// Результаты параллельного поиска

/// </summary>

public class ParallelSearchResult

{

/// <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; }

}

}

ParallelSearchThreadParam.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace WindowsFormsApplication1

{

/// <summary>

/// Параметры которые передаются в поток для параллельного поиска

/// </summary>

class ParallelSearchThreadParam

{

/// <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; }

}

}

SubArrays.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace WindowsFormsApplication1

{

/// <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;

}

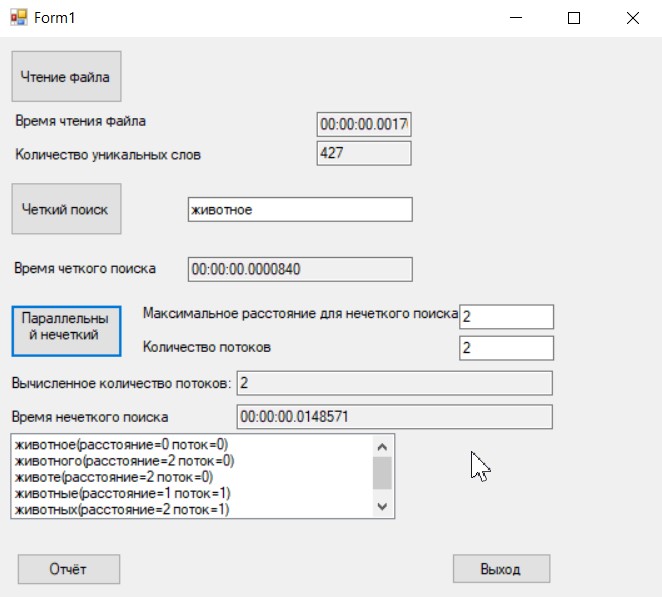
}

}

Примеры

Был прочитан фрагмент из статьи из Википедии «Еж обыкновенный»

Нечеткий поиск слова «животное»



Отчет по нечеткому поиску слова «животное»



Четкий поиск слова «животное»

