**Московский государственный технический университет**

**им. Н.Э. Баумана**

Кафедра ИУ–5

**Отчет по ДЗ по курсу «Базовые компоненты интернет технологий»**

Выполнил:

студент гр. ИУ5-34

Окопный М.О.

\_\_\_\_\_\_\_\_

Проверил:

к.т.н., доцент

Гапанюк Ю.Е.

\_\_\_\_\_\_\_\_

**Москва – 2017 г**

1. **Цель выполнения лабораторной работы**

Разработать программу, реализующую многопоточный поиск в файле.

1. **Листинг программы**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Threading.Tasks;

using System.IO;

using System.Diagnostics;

namespace dz

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

List<string> list = new List<string>(); // Список слов

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); //Чтение файла в виде строки

char[] separators = new char[] { ' ', '.', ',', '!', '?', '/', '\t', '\n', '—', ')', '(' }; //Разделительные символы для чтения из файла

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

foreach (string strTemp in textArray)

{

string str = strTemp.Trim(); //Удаление пробелов в начале и конце строки

if (!list.Contains(str) && str.Length != 0) //Добавление строки в список, если строка не содержится в списке

list.Add(str);

}

t.Stop();

this.text\_box2.Text = t.Elapsed.ToString();

add\_to\_list\_box(list);

}

else

{

MessageBox.Show("Необходимо выбрать файл");

}

}

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

{

string word = this.text\_box1.Text.Trim(); //Слово для поиска

if (!string.IsNullOrWhiteSpace(word) && list.Count > 0) //Если слово для поиска не пусто

{

string wordUpper = word.ToUpper(); //Слово для поиска в верхнем регистре

List<string> tempList = new List<string>(); //Временные результаты поиска

Stopwatch t = new Stopwatch();

t.Start();

foreach (string str in list)

{

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

{

tempList.Add(str);

}

}

if (tempList.Count == 0)

{

MessageBox.Show("Искомое слово не найдено!");

}

t.Stop();

this.text\_box4.Text = t.Elapsed.ToString();

list\_box1.SelectedIndex = list\_box1.FindStringExact(text\_box1.Text);

Lev\_distanse(text\_box1.Text);

}

else

{

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

}

}

public static int Distance(string str1Param, string str2Param)

{

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

int str1Len = str1Param.Length;

int str2Len = str2Param.Length;

if ((str1Len == 0) && (str2Len == 0)) return 0; //Если хотя бы одна строка пустая, возвращается длина другой строки

if (str1Len == 0) return str2Len;

if (str2Len == 0) return str1Len;

string str1 = str1Param.ToUpper(); //Приведение строк к верхнему регистру

string str2 = str2Param.ToUpper();

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

for (int i = 0; i <= str1Len; i++) matrix[i, 0] = i; //Инициализация нулевой строки и нулевого столбца матрицы

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

for (int i = 1; i <= str1Len; i++) //Вычисление расстояния Дамерау-Левенштейна

{

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

{

int symbEqual = ((str1.Substring(i - 1, 1) == str2.Substring(j - 1, 1)) ? 0 : 1); //Эквивалентность символов, переменная symbEqual соответствует m(s1[i],s2[j])

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

matrix[i, j] = Math.Min(Math.Min(ins, del), subst); //Элемент матрицы вычисляется как минимальный из трех случаев

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))) //Дополнение Дамерау по перестановке соседних символов

{

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

}

}

}

return matrix[str1Len, str2Len]; //Возвращается нижний правый элемент матрицы

}

void add\_to\_list\_box(List<string> arr)

{

Stopwatch t = new Stopwatch();

t.Start();

list\_box1.Items.Clear();

list\_box1.BeginUpdate();

foreach (string l in arr)

{

list\_box1.Items.Add(l);

}

list\_box1.EndUpdate();

t.Stop();

this.text\_box3.Text = t.Elapsed.ToString();

}

int Lev\_distanse(string word)

{

list\_box2.Items.Clear();

int a = 0;

bool f = int.TryParse(text\_box5.Text, out a);

if (!f || a < 0 || a % 1 != 0)

{

MessageBox.Show("Параметр введен неверно!");

list\_box2.Items.Clear();

return 0;

}

list\_box2.BeginUpdate();

foreach (string s in list\_box1.Items)

{

if (Distance(word, s) <= a)

list\_box2.Items.Add(s);

}

list\_box2.EndUpdate();

return 0;

}

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

{

string word = this.text\_box1.Text.Trim(); //Слово для поиска

if (!string.IsNullOrWhiteSpace(word) && list.Count > 0) //Если слово для поиска не пусто

{

int maxDist;

if (!int.TryParse(this.text\_box5.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.text\_box6.Text.Trim(), out ThreadCount))

{

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

return;

}

Stopwatch timer = new Stopwatch();

timer.Start();

//-------------------------------------------------

// Начало параллельного поиска

//-------------------------------------------------

List<ParallelSearchResult> Result = new List<ParallelSearchResult>(); //Результирующий список

List<MinMax> arrayDivList = SubArrays.DivideSubArrays(0, list.Count, ThreadCount); //Деление списка на фрагменты для параллельного запуска в потоках

int count = arrayDivList.Count;

Task<List<ParallelSearchResult>>[] tasks = new Task<List<ParallelSearchResult>>[count]; //Количество потоков соответствует количеству фрагментов массива

for (int i = 0; i < count; i++) //Запуск потоков

{

List<string> tempTaskList = list.GetRange(arrayDivList[i].Min, arrayDivList[i].Max - arrayDivList[i].Min); //Создание временного списка, чтобы потоки не работали параллельно с одной коллекцией

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

//Метод, который будет выполняться в потоке

ArrayThreadTask,

//Параметры потока

new ParallelSearchThreadParam()

{

tempList = tempTaskList,

maxDist = maxDist,

ThreadNum = i,

wordPattern = word

});

//Запуск потока

tasks[i].Start();

}

Task.WaitAll(tasks);

timer.Stop();

//Объединение результатов

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

{

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

}

//-------------------------------------------------

// Завершение параллельного поиска

//-------------------------------------------------

timer.Stop();

//Вывод результатов

//Время поиска

this.text\_box7.Text = timer.Elapsed.ToString();

//Вычисленное количество потоков

this.text\_box8.Text = count.ToString();

//Начало обновления списка результатов

this.list\_box2.BeginUpdate();

//Очистка списка

this.list\_box2.Items.Clear();

//Вывод результатов поиска

foreach (var x in Result)

{

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

this.list\_box2.Items.Add(temp);

}

//Окончание обновления списка результатов

this.list\_box2.EndUpdate();

}

else

{

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

}

}

public class ParallelSearchResult

{

public string word { get; set; }// Найденное слово

public int dist { get; set; }// Расстояние

public int ThreadNum { get; set; }// Номер потока

}

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;

}

}

public static class SubArrays

{

// Деление массива на последовательности

//beginIndex - Начальный индекс массива</param>

//endIndex- Конечный индекс массива</param>

//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;

}

}

public static List<ParallelSearchResult> ArrayThreadTask(object paramObj)

{

/// Выполняется в параллельном потоке для поиска строк

ParallelSearchThreadParam param = (ParallelSearchThreadParam)paramObj;

string wordUpper = param.wordPattern.Trim().ToUpper(); //Слово для поиска в верхнем регистре

List<ParallelSearchResult> Result = new List<ParallelSearchResult>();//Результаты поиска в одном потоке

foreach (string str in param.tempList) //Перебор всех слов во временном списке данного потока

{

int dist = Distance(str.ToUpper(), wordUpper);//Вычисление расстояния Дамерау-Левенштейна

if (dist <= param.maxDist)

{

ParallelSearchResult temp = new ParallelSearchResult()

{

word = str,

dist = dist,

ThreadNum = param.ThreadNum

};

Result.Add(temp);

}

}

return Result;

}

class ParallelSearchThreadParam

{

// Параметры которые передаются в поток

// для параллельного поиска

public List<string> tempList { get; set; } // Массив для поиска

public string wordPattern { get; set; } // Слово для поиска>

public int maxDist { get; set; } // Максимальное расстояние для нечеткого поиска

public int ThreadNum { get; set; } // Номер потока

}

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

{

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

//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.text\_box2.Text + "</td>");

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

b.AppendLine("<tr>");

b.AppendLine("<td>Время сохранения слов из файла в ListBox</td>");

b.AppendLine("<td>" + this.text\_box3.Text + "</td>");

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

b.AppendLine("<tr>");

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

b.AppendLine("<td>" + this.text\_box1.Text + "</td>");

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

b.AppendLine("<tr>");

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

b.AppendLine("<td>" + this.text\_box5.Text + "</td>");

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

b.AppendLine("<tr>");

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

b.AppendLine("<td>" + this.text\_box4.Text + "</td>");

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

b.AppendLine("<tr>");

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

b.AppendLine("<td>" + this.text\_box7.Text + "</td>");

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

b.AppendLine("<tr>");

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

b.AppendLine("<td>" + this.text\_box6.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.list\_box2.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 text\_box6\_TextChanged(object sender, EventArgs e)

{

}

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

{

}

private void list\_box2\_SelectedIndexChanged(object sender, EventArgs e)

{

}

private void label6\_Click(object sender, EventArgs e)

{

}

}

}

1. **Результаты работы программы**



