**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты Интернет-технологий»

Пояснительная записка по выполнению домашнего задания

|  |  |  |
| --- | --- | --- |
| Выполнила: |  | Проверил: |
| студентка группы  ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Бессонова Ксения |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

Описание задания

Пример реализации ДЗ рассмотрен в учебном пособии, глава «Пример многопоточного поиска в текстовом файле с использованием технологии Windows Forms».

Разработать программу, реализующую многопоточный поиск в файле.

1. Программа должна быть разработана в виде приложения Windows Forms на языке C#. По желанию вместо Windows Forms возможно использование WPF.
2. В качестве основы используется макет, разработанный в лабораторных работах №4 и №5.
3. Реализуйте функцию поиска с использованием расстояния Левенштейна в многопоточном варианте. Количество потоков для запуска функции поиска вводится на форме в поле ввода (TextBox).
4. Реализуйте функцию записи результатов поиска в файл отчета. Файл отчета создается в формате .txt или .html.

**Листинг**

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace HW

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

**SubArrays.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HW

{

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;

}

}

}

**MinMax.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HW

{

class MinMax

{

public int Min { get; set; }

public int Max { get; set; }

public MinMax(int pmin, int pmax)

{

this.Min = pmin;

this.Max = pmax;

}

}

}

**EditDistance.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HW

{

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];

}

}

}

**ParallelSearchResult.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HW

{

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;

using System.Threading.Tasks;

namespace HW

{

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; }

}

}

**Form1:**

using System;

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.IO;

using System.Diagnostics;

namespace HW

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

// расстояние Ливенштейна

public static int Distance(string string1, string string2)

{

int diff;

int[,] m = new int[string1.Length + 1, string2.Length + 1];

for (int i = 0; i <= string1.Length; i++) m[i, 0] = i;

for (int j = 0; j <= string2.Length; j++) m[0, j] = j;

for (int i = 1; i <= string1.Length; i++)

for (int j = 1; j <= string2.Length; j++)

{

diff = (string1[i - 1] == string2[j - 1]) ? 0 : 1;

m[i, j] = Math.Min(Math.Min(m[i - 1, j] + 1,

m[i, j - 1] + 1),

m[i - 1, j - 1] + diff);

}

return m[string1.Length, string2.Length];

}

// чтение файла

private void reading\_button\_Click(object sender, EventArgs e)

{

OpenFileDialog fileDialog = new OpenFileDialog

{

InitialDirectory = Directory.GetCurrentDirectory(),

Filter = "Файлы \*.txt|\*.txt"

};

fileDialog.ShowDialog();

if (fileDialog.FileName.Length == 0)

{

return;

}

Stopwatch time = new Stopwatch();

time.Start();

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

foreach (var word in text.Split())

{

if (!list.Contains(word))

{

list.Add(word);

}

}

time.Stop();

Double result = time.Elapsed.TotalMilliseconds;

this.Time\_of\_loading.Text = result.ToString();

}

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

{

this.Close();

}

// поиск

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

{

// берем слово

string word = this.Field\_for\_input.Text.Trim();

// проверка, что слово введено и файл загружен

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

{

int maxDist;

// проверка что указано расстояние

if (!int.TryParse(this.text\_dist.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;

// введено к-во потоков

if (!int.TryParse(this.count\_of\_stream.Text.Trim(), out ThreadCount))

{

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

return;

}

Stopwatch timer = new Stopwatch();

timer.Start();

//Результирующий список

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

//Деление списка на фрагменты для параллельного запуска в потоках

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

int count = arrayDivList.Count;

//Количество потоков соответствует количеству фрагментов массива

Task<List<ParallelSearchResult>>[] tasks = new Task<List<ParallelSearchResult>>[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(arrayDivList[i].Min, arrayDivList[i].Max - arrayDivList[i].Min);

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();

}

// ожидание потоков

Task.WaitAll(tasks);

timer.Stop();

//Объединение результатов

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

{

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

}

//завершение параллельного поиска

timer.Stop();

//Время поиска

this.Time\_of\_searching.Text = timer.Elapsed.ToString();

//Вычисленное количество потоков

//Обновление списка результатов

this.Result\_list.BeginUpdate();

this.Result\_list.Items.Clear();

foreach (var x in Result)

{

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

this.Result\_list.Items.Add(temp);

}

//окончание обновления списка результатов

this.Result\_list.EndUpdate();

}

else

{

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

}

}

public static List<ParallelSearchResult> ArrayThreadTask(object paramObj)

{

ParallelSearchThreadParam param = (ParallelSearchThreadParam)paramObj;

//слово для поиска в верхнем регистре

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

//Результаты поиска в одном потоке

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

//Перебор всех слов во временном списке данного потока

foreach (string str in param.tempList)

{

//Вычисление расстояния Левенштейна

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

//Если расстояние меньше порогового, то слово добавляется в результат

if (dist <= param.maxDist)

{

ParallelSearchResult temp = new ParallelSearchResult()

{

word = str,

dist = dist,

ThreadNum = param.ThreadNum

};

Result.Add(temp);

}

}

return Result;

}

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

{

}

private void label\_dist\_Click(object sender, EventArgs e)

{

}

private void button\_report\_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.Filter = "Text files|\*.txt|HTML Reports|\*.html;";

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

{

string ReportFileName = fd.FileName;

if (Path.GetExtension(fd.FileName) == ".txt")

{

StreamWriter sw = new StreamWriter(fd.FileName);

sw.WriteLine("Отчет: " + ReportFileName);

sw.WriteLine("Время чтения из файла: " + this.Time\_of\_loading.Text);

sw.WriteLine("Слово для поиска: " + this.Field\_for\_input.Text);

sw.WriteLine("Максимальное расстояние для нечеткого поиска: " + this.text\_dist.Text);

sw.WriteLine("Время нечеткого поиска: " + this.Time\_of\_searching.Text);

sw.WriteLine("Результаты поиска: ");

foreach (var x in this.Result\_list.Items)//запись результатов поиска

{

sw.WriteLine(x.ToString());

}

sw.Close();

}

else

{

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.Time\_of\_loading.Text + "</td>");

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

b.AppendLine("<tr>");

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

b.AppendLine("<td>" + this.Field\_for\_input.Text + "</td>");

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

b.AppendLine("<tr>");

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

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

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

b.AppendLine("<tr>");

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

b.AppendLine("<td>" + this.Time\_of\_searching.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.Result\_list.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 успешно сформирован. Файл: " + ReportFileName);

}

}

private void groupBox2\_Enter(object sender, EventArgs e)

{

}

private void Field\_for\_input\_TextChanged(object sender, EventArgs e)

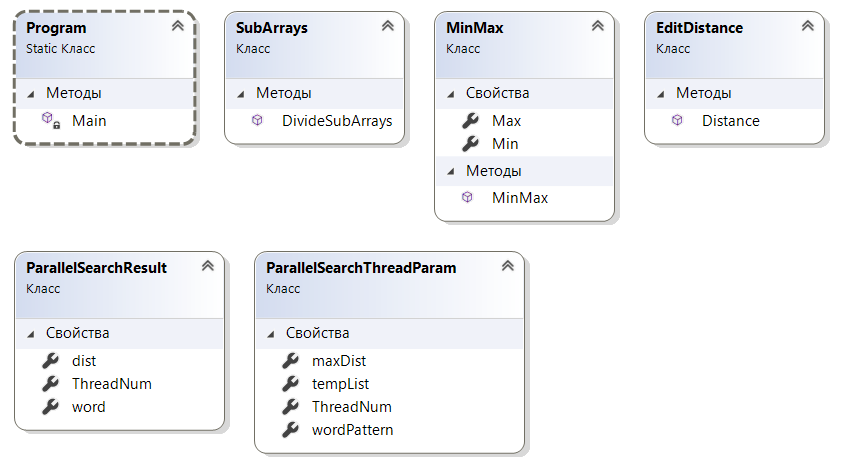
{

}

}

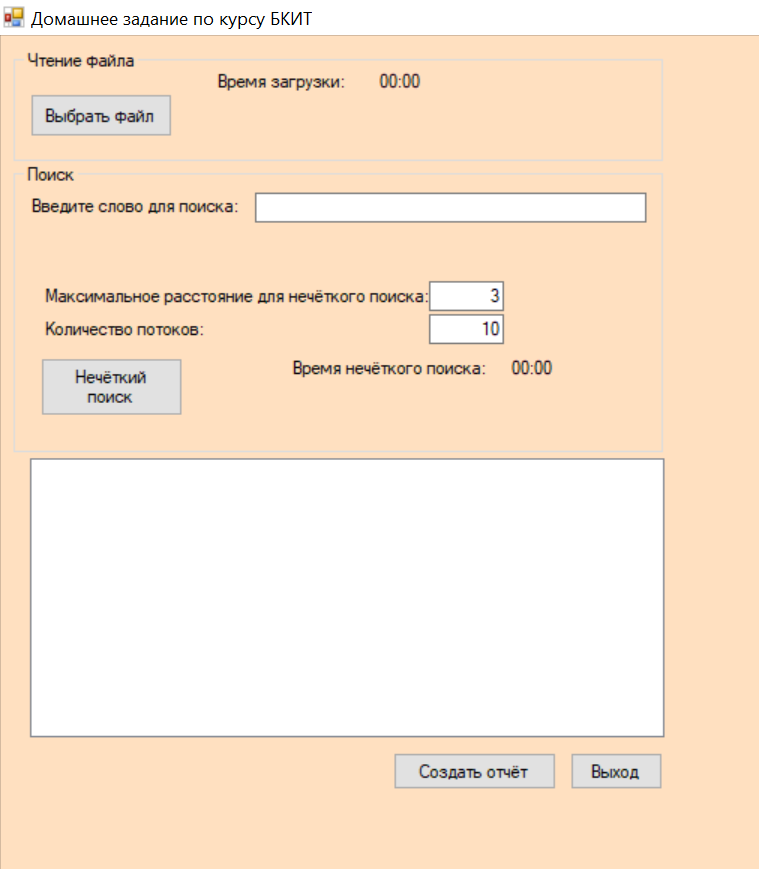
}

**Диаграмма классов**

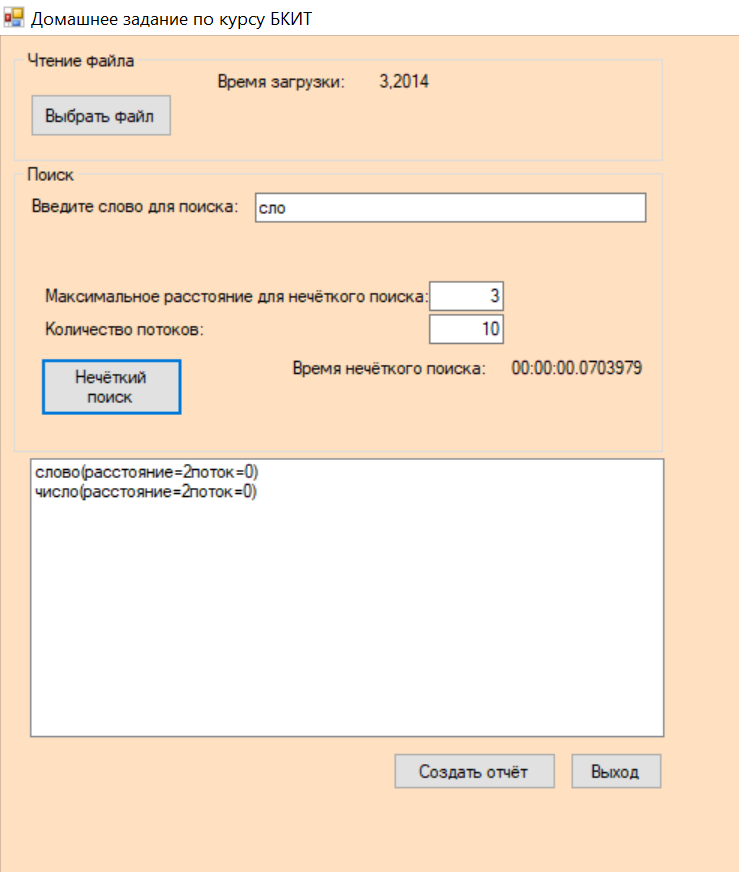


Пример выполнения программы

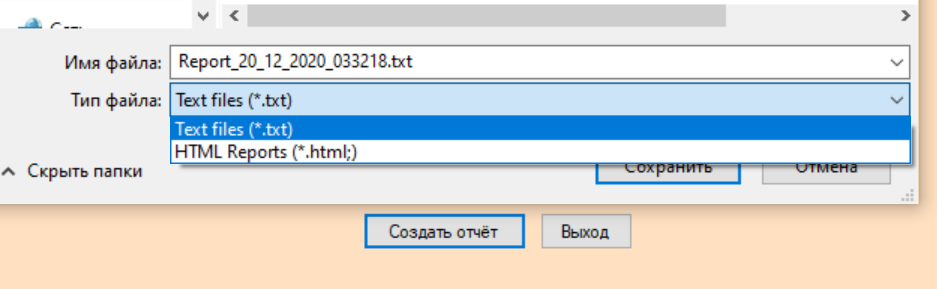
Исходный вид:



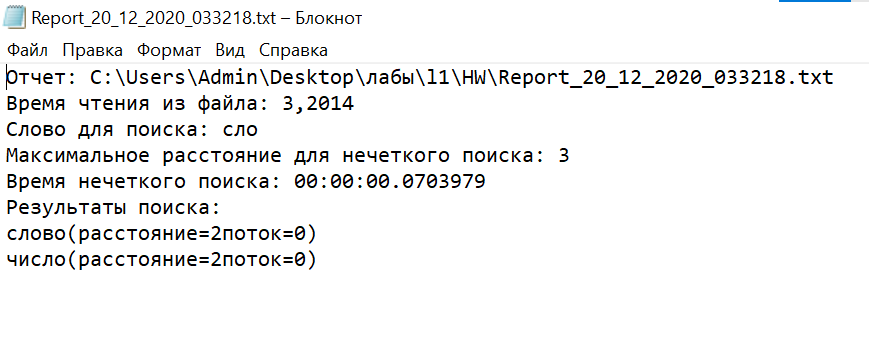
Работа программы:



Возможность выбора сохранения отчёта в одном из двух форматов: \*.txt и \*.html



Отчёт в формате \*.txt



Отчёт в формате \*.html

