Московский государственный технический университет

им. Н. Э. Баумана

**Отчет**

**по Домашнему Заданию**

Выполнил:

Студент группы ИУ5-35Б

Богданова В.В.

Проверил:

Преподаватель

Гапанюк Ю.Е.

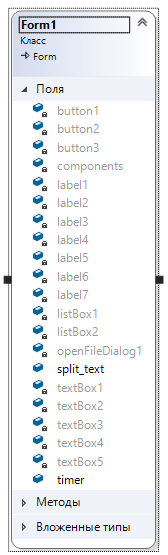
2020г.

**1). Описание задания**

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). В качестве примера используйте проект «Parallel» из примера «Введение в C#».
4. Реализуйте функцию записи результатов поиска в файл отчета. Файл отчета создается в формате .txt или .html. В качестве примера используйте проект «WindowsFormsFiles» (обработчик события кнопки «Сохранение отчета») из примера «Введение в C#».

**2). Диаграмма классов**



**3). Текст программы**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Threading;

using System.IO;

namespace lab4\_сишарп

{

public partial class Form1: Form

{

public static List<string> split\_text = new List<string>();

public static Stopwatch timer = new Stopwatch();

public Form1()

{

InitializeComponent();

}

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

{

OpenFileDialog ofd1 = new OpenFileDialog();

ofd1.Filter = "Text files(\*.txt)|\*.txt";

if (ofd1.ShowDialog() == DialogResult.Cancel)

{

MessageBox.Show("Необходимо выбрать файл");

return;

}

Stopwatch stopWatch = new Stopwatch();

stopWatch.Start();

// читаем файл в строку

string fileText = System.IO.File.ReadAllText(ofd1.FileName);

//добавляем разделители

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

//получаем список слов

string[] split\_text\_temp = fileText.Split(separators);

//убираем дубликаты

foreach (string str in split\_text\_temp)

{

if (!split\_text.Contains(str))

split\_text.Add(str);

}

stopWatch.Stop();

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

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

}

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());

}

/// <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;

}

}

/// <summary>

/// Выполняется в параллельном потоке для поиска слова

/// </summary>

/// <param name="param"></param>

public static List<string> ArrayThreadTask(object paramObj)

{

//Получение параметров

Tuple<List<string>, string, int> param = (Tuple<List<string>, string, int>)paramObj;

int listCount = param.Item1.Count;

//Временный список для результата

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

//Перебор нужных элементов в списке данных

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

{

//Текущее значение из массива

string temp = param.Item1[i];

int dist = Distance(temp.ToLower(), param.Item2);

if (dist <= param.Item3)

tempData.Add(temp);

}

//Возврат массива данных

return tempData;

}

/// <summary>

/// Многопоточный поиск в массиве

/// </summary>

public List<string> ArrayThreadExample(int ArrayLength, int ThreadCount, string word, int max\_dist)

{

//Результирующий список чисел

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

//Создание и заполнение временного списка данных

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

foreach (string str in split\_text)

tempList.Add(str);

//Деление списка на фрагменты для параллельного запуска в потоках

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

int count = arrayDivList.Count;

//Очистка списка

this.listBox2.Items.Clear();

//Вывод диапазонов деления исходного массива

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

{

//Вывод результатов, найденных в каждом потоке

this.listBox2.Items.Add("Диапазон " + i.ToString() + ": " + arrayDivList[i].Min + " - " + arrayDivList[i].Max);

}

Console.WriteLine();

//Обновление таймера

timer.Reset();

//Запуск таймера

timer.Start();

//Количество потоков соответствует количеству фрагментов массива

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

//Запуск потоков

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

{

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

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

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

//Метод, который будет выполняться в потоке

ArrayThreadTask,

//Параметры потока передаются в виде кортежа, чтобы не создавать временный класс

new Tuple<List<string>, string, int>(tempTaskList, word, max\_dist));

//Запуск потока

tasks[i].Start();

}

//Ожидание завершения всех потоков

Task.WaitAll(tasks);

//Остановка таймера

timer.Stop();

//Объединение результатов полученных из разных потоков

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

{

//Вывод результатов, найденных в каждом потоке

Console.Write("Поток " + i.ToString() + ": ");

foreach (var x in tasks[i].Result) Console.Write(x.ToString() + " ");

Console.WriteLine();

//Добавление результатов конкретного потока в общий массив результатов

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

}

//Вывод общего массива результатов

return Result;

}

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

{

//определяем слово для поиска в нижнем регистре

string word = this.textBox2.Text.Trim().ToLower();

int max\_dist = int.Parse(this.textBox4.Text);

int potok\_kol = int.Parse(this.textBox5.Text);

if (string.IsNullOrWhiteSpace(word) || split\_text.Count == 0)

{

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

return;

}

List<string> find\_list = new List<string>();

find\_list = ArrayThreadExample(split\_text.Count, potok\_kol, word, max\_dist);

this.textBox3.Text = timer.Elapsed.ToString();

this.listBox1.BeginUpdate();

//Очистка списка

this.listBox1.Items.Clear();

//Вывод результатов поиска

foreach (string str in find\_list)

{

this.listBox1.Items.Add(str);

}

this.listBox1.EndUpdate();

}

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

{

//Имя файла отчета

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>" + split\_text.Count + "</td>");

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

b.AppendLine("<tr>");

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

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

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

b.AppendLine("<tr>");

b.AppendLine("<td>Максимальное расстояние для нечеткого 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.textBox3.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);

}

}

}

}

**4). Пример выполнения**

