**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №5

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-35б |  | преподаватель каф. ИУ5 |
| Зубарева Антонина |  |  |
| Подпись и дата: |  | Подпись и дата: |

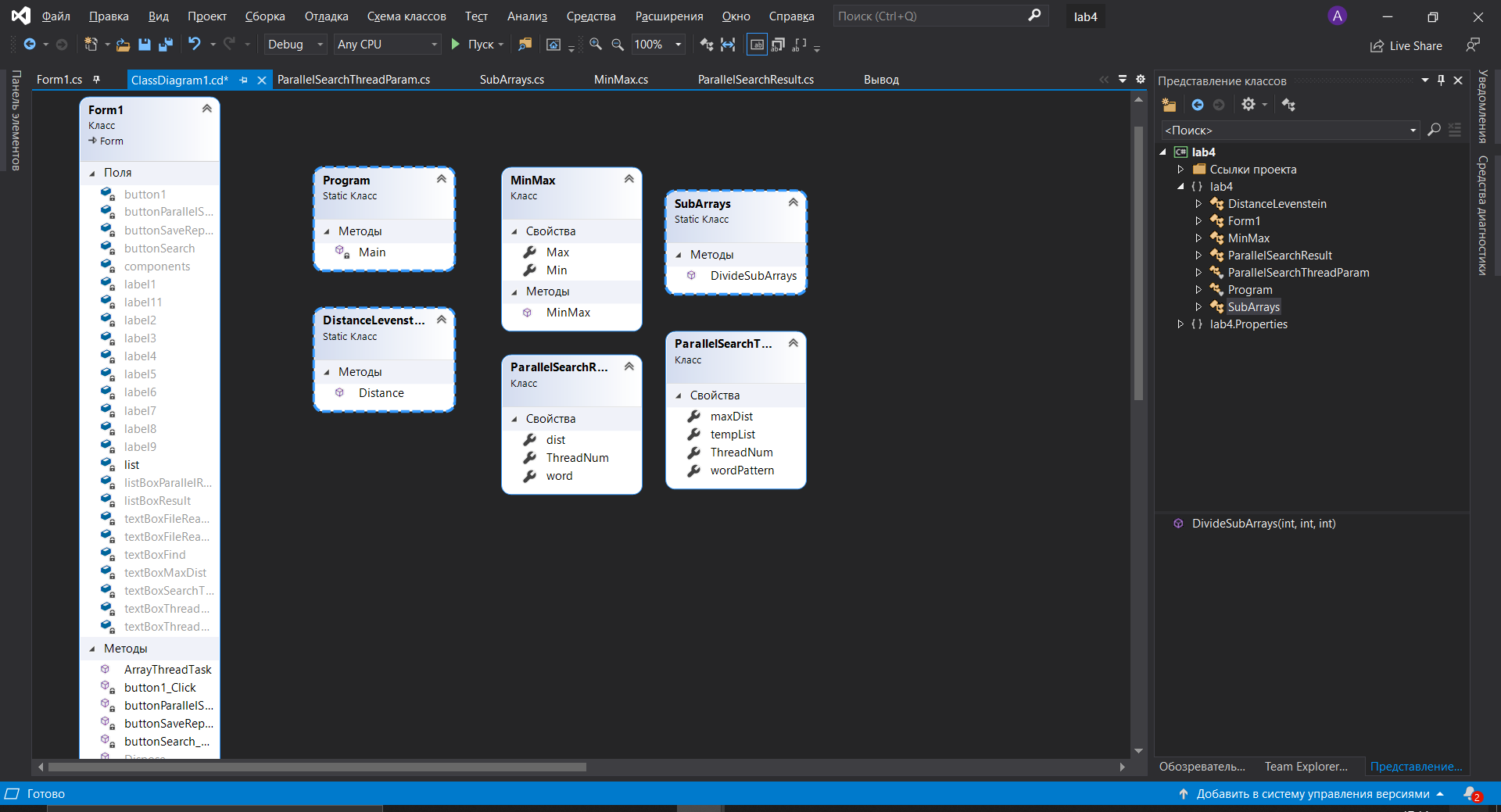
Москва, 2020 г.

# 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.

# Диаграмма классов



# Текст программы

# private void buttonParallelSearch\_Click(object sender, EventArgs e)

# {

# //Слово для поиска

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

# //Если слово для поиска не пусто

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

# {

# //максимальное расстояние

# int maxDist;

# if (!int.TryParse(this.textBoxMaxDist.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;

# }

# //количество потоков

# int ThreadCount;

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

# {

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

# return;

# }

# //время поиска

# Stopwatch timer = new Stopwatch();

# timer.Start();

# //-------------------------------------------------

# // Начало параллельного поиска

# //-------------------------------------------------

# //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();

# }

# 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.textBoxApproxTime.Text = timer.Elapsed.ToString();

# //Вычисленное количество потоков

# this.textBoxThreadCountAll.Text = count.ToString();

# //Начало обновления списка результатов

# this.listBoxParallelResult.BeginUpdate();

# //Очистка списка

# this.listBoxParallelResult.Items.Clear();

# //Вывод результатов поиска

# foreach (var x in Result)

# {

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

# this.listBoxParallelResult.Items.Add(temp);

# }

# //Окончание обновления списка результатов

# this.listBoxParallelResult.EndUpdate();

# }

# else

# {

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

# }

# }

# /// <summary>

# /// Выполняется в параллельном потоке для поиска строк

# /// </summary>

# 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 Дамерау-Левенштейна

# int dist = DistanceLevenstein.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;

# }

# Результаты

