**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по домашнему заданию

«Многопоточный поиск в файле»

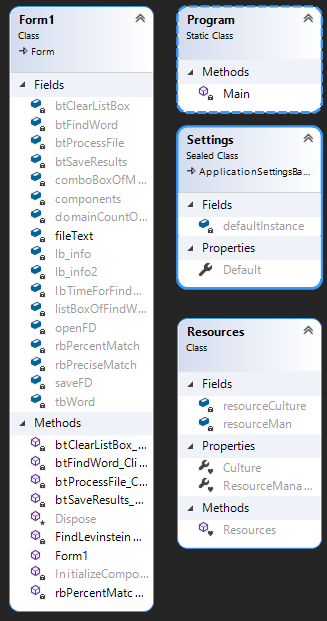
|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32 |  | преподаватель каф. ИУ5 |
| Барышников Михаил |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

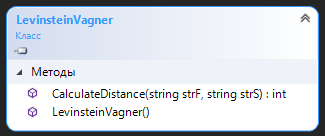
Москва, 2020 г.

**Задача**

Разработать программу, реализующую многопоточный поиск в файле. Программа должна быть разработана в виде приложения Windows Forms на языке C#. По желанию вместо Windows Forms возможно использование WPF. В качестве основы используется макет, разработанный в лабораторных работах №4 и №5. 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#». 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#».

**Диаграмма классов**





**Текст программы**

* Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Diagnostics;

using VagnerFisherAlgorithmlib;

namespace FileSearch

{

public partial class Form1 : Form

{

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

public Form1()

{

InitializeComponent();

}

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

{

fileText.Clear();

Stopwatch fixedTime = new Stopwatch();

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

{

fixedTime.Start();

string[] text = File.ReadAllText(openFD.FileName).Split(' ');

foreach (string i in text)

if (!fileText.Contains(i))

fileText.Add(i);

fixedTime.Stop();

lbTimeForFindWord.Text = fixedTime.Elapsed.ToString();

}

}

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

{

if (tbWord.Text != "" && fileText.Count != 0)

{

Stopwatch fixedTime = new Stopwatch();

int maxDistance;

if (rbPreciseMatch.Checked)

{

fixedTime.Start();

int countOfThreads = Int32.Parse(domainCountOfThreads.Text), countOfWords = fileText.Count / countOfThreads;

Task<bool>[] tasks = new Task<bool>[countOfThreads];

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

{

List<string> tempTaskList = fileText.GetRange(i \* countOfWords, (i == countOfThreads - 1 ? fileText.Count - i \* countOfWords : countOfWords));

tasks[i] = new Task<bool>(() => (tempTaskList.Contains(tbWord.Text)));

tasks[i].Start();

}

Task.WaitAll(tasks);

fixedTime.Stop();

foreach (Task<bool> i in tasks)

{

if (i.Result)

{

listBoxOfFindWords.BeginUpdate();

listBoxOfFindWords.Items.Add(tbWord.Text + " - " + fixedTime.ElapsedMilliseconds.ToString());

listBoxOfFindWords.EndUpdate();

}

}

lbTimeForFindWord.Text = fixedTime.Elapsed.ToString();

}

else if (Int32.TryParse(comboBoxOfMaxDistance.Text, out maxDistance))

{

fixedTime.Start();

int countOfThreads = Int32.Parse(domainCountOfThreads.Text), countOfWords = fileText.Count / countOfThreads;

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

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

{

List<string> tempTaskList = fileText.GetRange(i \* countOfWords, (i == countOfThreads - 1 ? fileText.Count - i \* countOfWords : countOfWords));

tasks[i] = new Task<List<string>>(() => FindLevinsteinDistanceInThread(tempTaskList, tbWord.Text, maxDistance));

tasks[i].Start();

}

Task.WaitAll(tasks);

fixedTime.Stop();

foreach (Task<List<string>> i in tasks)

{

foreach (string j in i.Result)

{

listBoxOfFindWords.BeginUpdate();

listBoxOfFindWords.Items.Add(j + " - " + maxDistance.ToString() + " - " + fixedTime.ElapsedMilliseconds.ToString());

listBoxOfFindWords.EndUpdate();

}

}

lbTimeForFindWord.Text = fixedTime.Elapsed.ToString();

}

}

tbWord.Text = "";

}

private void rbPercentMatch\_CheckedChanged(object sender, EventArgs e)

{

comboBoxOfMaxDistance.Visible = rbPercentMatch.Checked;

}

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

{

listBoxOfFindWords.Items.Clear();

}

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

{

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

{

File.Create(saveFD.FileName).Close();

foreach (string i in listBoxOfFindWords.Items)

File.AppendAllText(saveFD.FileName, i + "\n");

}

}

static List<string> FindLevinsteinDistanceInThread(List<string> strList, string word, int maxDistance)

{

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

foreach (string i in strList)

if (LevinsteinVagner.CalculateDistance(i, word) <= maxDistance)

ResultList.Add(i);

return ResultList;

}

}

}

* Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace FileSearch

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

**Результаты выполнения программы**

