**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 домашней работе

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Ветошкин Артём |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

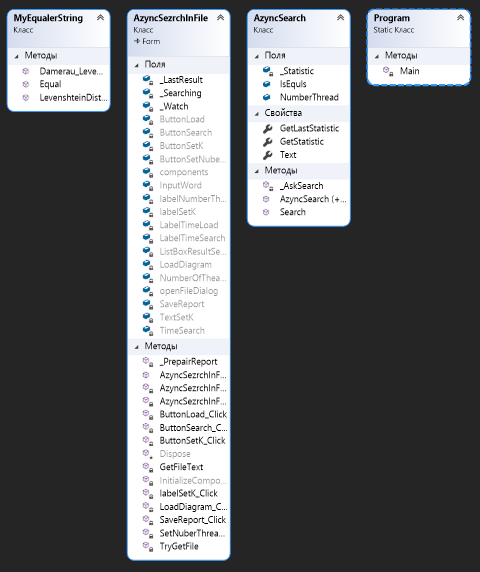
Москва, 2020 г.

**Постановка задачи**

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#».

**Диаграмма классов**



**Текст программы**

Файл Program.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace DZ {

static class Program {

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main() {

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForms());

}

}

}

Файл AzyncSearch.cs:

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Threading.Tasks;

namespace SearchString {

class AzyncSearch {

static public long NumberThread = 10;

public AzyncSearch() {

Text = new List<string>();

\_Statistic = new List<(long parametr, long time)>();

}

public AzyncSearch(List<string> text) {

Text = text;

\_Statistic = new List<(long parametr, long time)>();

}

public List<(long length\_string, long time)> GetStatistic {

get { return \_Statistic; }

}

public (long length\_string, long time) GetLastStatistic {

get { return \_Statistic[\_Statistic.Count - 1]; }

}

public (List<string> found\_string, List<long> time) Search(string word, long save\_parametr) {

var timer = new Stopwatch();

var Tasks = new List<Task<(List<string> found\_string, long time)>>();

int lenSubArray = Text.Count / (int)NumberThread;

for(int i = 0; i < NumberThread; ++i) {

List<string> tmp = Text.GetRange(i \* lenSubArray

, Math.Min(Text.Count - i \* lenSubArray, lenSubArray));

Tasks.Add(new Task<(List<string> found\_string, long time)>(

\_AskSearch, new Tuple<List<string>, string>(tmp, word)));

}

timer.Start();

foreach (var i in Tasks) {

i.Start();

}

Task.WaitAll(Tasks.ToArray());

timer.Stop();

(List<string> found\_string, List<long> time) ans = (new List<string>(), new List<long>());

foreach (var task in Tasks) {

ans.found\_string.AddRange(task.Result.found\_string);

ans.time.Add(task.Result.time);

}

\_Statistic.Add((save\_parametr, timer.ElapsedTicks));

return ans;

}

static public Func<string, string, bool> IsEquls;

private (List<string> found\_string, long time) \_AskSearch(object paramObj) {

List<string> text = ((Tuple<List<string>, string>)paramObj).Item1;

string word = ((Tuple<List<string>, string>)paramObj).Item2;

(List<string> found\_string, long time) ans = (new List<string>(), new long());

var stopwatch = new Stopwatch();

stopwatch.Start();

foreach (var s in text) {

if (IsEquls(s, word)) {

ans.found\_string.Add(s);

}

}

stopwatch.Stop();

ans.time = stopwatch.ElapsedTicks;

return ans;

}

public List<string> Text { get; }

private List<(long parametr, long time)> \_Statistic;

}

}

Файл CompareString.cs:

using System;

namespace CompareString {

class MyEqualerString {

static public long LevenshteinDistance(string s1, string s2, long k = long.MaxValue) {

if (s1.Length is 0) return s2.Length;

if (s2.Length is 0) return s1.Length;

int size = s2.Length + 1;

bool use\_optimaze = !k.Equals(long.MaxValue);

long[][] D = new long[2][] { new long[size], new long[size] };

D[0][0] = 0;

D[1][0] = 1;

for (int j = 1; j <= s2.Length; ++j) {

D[0][j] = j;

}

for (int i = 1; i <= s1.Length; ++i) {

for (int j = 1; j <= s2.Length; ++j) {

D[1][j] = Math.Min(D[0][j] + 1, Math.Min(D[1][j - 1] + 1,

D[0][j - 1] + (s1[i - 1] == s2[j - 1] ? 0 : 1)));

}

D[1].CopyTo(D[0], 0);

D[1][0] = i + 1;

if (use\_optimaze) {

if (D[1][Math.Min(i, size - 1)] > k) {

return k;

}

}

}

return D[Math.Min(1, s1.Length - 1)][size - 1];

}

static public long Damerau\_LevenshteinDistance(string s1, string s2, long k = long.MaxValue) {

if (s1.Length is 0) return s2.Length;

if (s2.Length is 0) return s1.Length;

int size = s2.Length + 1;

bool use\_optimaze = !k.Equals(long.MaxValue);

long[][] D = new long[3][] { new long[size],

new long[size], new long[size] };

D[0][0] = 0;

D[1][0] = 1;

D[2][0] = 2;

for (int j = 1; j <= s2.Length; ++j) {

D[0][j] = j;

}

for (int j = 1; j <= s2.Length; ++j) {

D[1][j] = Math.Min(D[0][j] + 1, Math.Min(D[1][j - 1] + 1,

D[0][j - 1] + (s1[0] == s2[j - 1] ? 0 : 1)));

}

for (int i = 2; i <= s1.Length; ++i) {

for (int j = 1; j <= s2.Length; ++j) {

if (j == 0) continue;

D[2][j] = Math.Min(D[1][j] + 1, Math.Min(D[2][j - 1] + 1,

Math.Min(D[1][j - 1] + (s1[i - 1] == s2[j - 1] ? 0 : 1)

, ((j - 2 >= 0 && s1[i - 1] == s2[j - 2]

&& s2[j - 1] == s1[i - 2])

? D[0][j - 2] + 1 : int.MaxValue))));

}

D[1].CopyTo(D[0], 0);

D[2].CopyTo(D[1], 0);

D[2][0] = i + 1;

if (use\_optimaze) {

if (D[2][Math.Min(i, size - 1)] > k) {

return k;

}

}

}

return D[s1.Length < 2 ? 1 : 2][size - 1];

}

static public bool Equal(string s1, string s2,

Func<string, string, long, long> get\_distance,

long k = 0) {

return get\_distance(s1, s2, k + 1) <= k;

}

}

}

Файл AsyncSearchInFile.cs:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using CompareString;

using SearchString;

namespace DZ {

public partial class AzyncSezrchInFile : Form {

private Stopwatch \_Watch = new Stopwatch();

private SearchString.AzyncSearch \_Searching = new SearchString.AzyncSearch();

private (List<string> found\_string, List<long> times) \_LastResult;

public AzyncSezrchInFile() {

InitializeComponent();

TextSetK.Visible = false;

labelSetK.Visible = false;

InputWord.Visible = false;

ButtonSetK.Visible = false;

SaveReport.Visible = false;

LoadDiagram.Visible = false;

ButtonSearch.Visible = false;

LabelTimeLoad.Visible = false;

NumberOfTheard.Visible = false;

LabelTimeSearch.Visible = false;

labelNumberThreads.Visible = false;

ListBoxResultSearch.Visible = false;

ButtonSetNuberThreads.Visible = false;

SearchString.AzyncSearch.IsEquls = (string a, string b)

=> MyEqualerString.Equal(a, b, MyEqualerString.Damerau\_LevenshteinDistance, 0);

Height = 90;

}

private bool TryGetFile(string filter) {

openFileDialog.InitialDirectory = "c:\\";

openFileDialog.Filter = filter;

openFileDialog.FilterIndex = 2;

openFileDialog.RestoreDirectory = true;

return openFileDialog.ShowDialog() == DialogResult.OK;

}

private List<string> GetFileText() {

if (!TryGetFile("txt files (\*.txt)|\*.txt")) {

return null;

}

\_Watch.Start();

var fileStream = openFileDialog.OpenFile();

var fileContent = string.Empty;

using (StreamReader reader = new StreamReader(fileStream)) {

fileContent = reader.ReadToEnd();

}

var got\_list = fileContent.Split(" ,.;?!\"\'\n\t«»)(".ToCharArray());

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

foreach (string word in got\_list) {

if (!list.Contains(word)) {

list.Add(word);

}

}

return list;

}

private void ButtonSearch\_Click(object sender, EventArgs e) {

string word = InputWord.Text;

\_LastResult = \_Searching.Search(word, long.Parse(TextSetK.Text));

ListBoxResultSearch.BeginUpdate();

ListBoxResultSearch.Items.Clear();

foreach (var s in \_LastResult.found\_string) {

ListBoxResultSearch.Items.Add(s);

}

ListBoxResultSearch.EndUpdate();

LabelTimeSearch.Text = \_Searching.GetLastStatistic.time.ToString() + "ticks";

LabelTimeSearch.Visible = true;

ListBoxResultSearch.Visible = true;

SaveReport.Visible = true;

LoadDiagram.Visible = true;

Height = 345;

}

private void ButtonLoad\_Click(object sender, EventArgs e) {

List<string> GotText = GetFileText();

\_Watch.Stop();

if (GotText is null) {

return;

}

LabelTimeLoad.Text = \_Watch.ElapsedTicks.ToString() + "ticks";

\_Searching = new SearchString.AzyncSearch(GotText);

if (Height > 150) return;

ButtonSetK.Visible = true;

LabelTimeLoad.Visible = true;

TextSetK.Visible = true;

labelSetK.Visible = true;

Height = 150;

}

private void ButtonSetK\_Click(object sender, EventArgs e) {

if (TextSetK.Text.Length is 0) {

MessageBox.Show("Поле ввода пусто!");

return;

}

long k = 0;

if (!long.TryParse(TextSetK.Text, out k)) {

MessageBox.Show("Вы ввели вместо пороговой длины неизвестный символ!");

return;

}

SearchString.AzyncSearch.IsEquls = (string a, string b)

=> MyEqualerString.Equal(a, b, MyEqualerString.Damerau\_LevenshteinDistance, k);

if (Height > 180) return;

ButtonSetNuberThreads.Visible = true;

NumberOfTheard.Visible = true;

labelNumberThreads.Visible = true;

Height = 180;

}

private void AzyncSezrchInFile\_Load(object sender, EventArgs e) {

Owner.Visible = false;

}

private void AzyncSezrchInFile\_FormClosed(object sender, FormClosedEventArgs e) {

Owner.Visible = true;

}

private void labelSetK\_Click(object sender, EventArgs e) {

}

private void SetNuberThreads\_Click(object sender, EventArgs e) {

if (NumberOfTheard.Text.Length is 0) {

MessageBox.Show("Поле ввода пусто!");

return;

}

if (!long.TryParse(NumberOfTheard.Text, out AzyncSearch.NumberThread)) {

MessageBox.Show("Вы ввели вместо числа потоков неизвестный символ!");

}

if (Height > 210) return;

ButtonSearch.Visible = true;

InputWord.Visible = true;

Height = 210;

}

private void LoadDiagram\_Click(object sender, EventArgs e) {

var Diagram = new ResultDiagram(\_LastResult.times, "Ticks");

Diagram.ShowDialog();

}

private string \_PrepairReport(string ReportFileName) {

string stringReport = "";

stringReport += "<html>";

stringReport += "<head>";

stringReport += "<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'/>";

stringReport += "<title>" + "Отчет: " + ReportFileName + "</title>";

stringReport += "</head>";

stringReport += "<body>";

stringReport += "<h1>" + "Отчет: " + ReportFileName + "</h1>";

stringReport += "<table border='1'>";

stringReport += "<tr>";

stringReport += "<td>Время чтения из файла</td>";

stringReport += "<td>" + LabelTimeLoad.Text + "</td>";

stringReport += "</tr>";

stringReport += "<tr>";

stringReport += "<td>Количество уникальных слов в файле</td>";

stringReport += "<td>" + \_Searching.Text.Count + "</td>";

stringReport += "</tr>";

stringReport += "<tr>";

stringReport += "<td>Слово для поиска</td>";

stringReport += "<td>" + InputWord.Text + "</td>";

stringReport += "</tr>";

stringReport += "<tr>";

stringReport += "<td>Пороговая длина поиска</td>";

stringReport += "<td>" + TextSetK.Text + "</td>";

stringReport += "</tr>";

stringReport += "<tr>";

stringReport += "<td>Время общего поиска</td>";

stringReport += "<td>" + LabelTimeSearch.Text + "</td>";

stringReport += "</tr>";

stringReport += "<tr>";

stringReport += "<td>Максимальнео время работы потока</td>";

stringReport += "<td>" + \_LastResult.times.Max().ToString() + "</td>";

stringReport += "</tr>";

stringReport += "<tr valign='top'>";

stringReport += "<td>Результаты поиска</td>";

stringReport += "<td>";

stringReport += "<ul>";

foreach (var word in this.\_LastResult.found\_string) {

stringReport += "<li>" + word.ToString() + "</li>";

}

stringReport += "</ul>";

stringReport += "</td>";

stringReport += "</tr>";

stringReport += "<tr valign='top'>";

stringReport += "<td>Время по потокам в тиках</td>";

stringReport += "<td>";

stringReport += "<ul>";

long i = 1;

foreach (var word in this.\_LastResult.times) {

stringReport += "<li> " + i++.ToString() + " : " + word.ToString() + "</li>";

}

stringReport += "</ul>";

stringReport += "</td>";

stringReport += "</tr>";

stringReport += "</table>";

stringReport += "</body>";

stringReport += "</html>";

return stringReport;

}

private void SaveReport\_Click(object sender, EventArgs e) {

SaveFileDialog ReportFile = new SaveFileDialog();

ReportFile.FileName = "report\_" + DateTime.Now.ToString("hh;mm;ss\_dd\_MM\_yy"); ;

ReportFile.DefaultExt = ".html";

ReportFile.Filter = "HTML Reports|\*.html";

if (ReportFile.ShowDialog() == DialogResult.OK) {

File.AppendAllText(ReportFile.FileName, \_PrepairReport(ReportFile.FileName));

MessageBox.Show("Отчет готов. Сохранён в файле: " + ReportFile.FileName);

}

}

}

}

**Анализ результатов**

