Міністерство освіти і науки України
Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Мультипарадигменне програмування

ЗВІТ

до лабораторної роботи №1

Виконав студент ІП-01 Возовиков Данило

Прийняв ас. Очеретяний О. К.

Київ 2021

1. Завдання лабораторної роботи

Завдання 1:

Обчислювальна задача тут тривіальна: для текстового файлу ми хочемо

відобразити N (наприклад, 25) найчастіших слів і відповідну частоту їх повторення, упорядковано за зменшенням. Слід обов'язково нормалізувати використання великих літер і ігнорувати стоп-слова, як «the», «for» тощо. Щоб все було просто, ми не піклуємося про порядок слів з однаковою частотою повторень. Ця обчислювальна задача відома як term frequency.

Ось такий вигляд матимуть ввід і відповідно вивід результату програми: Input:
White tigers live mostly in India
Wild lions live mostly in Africa

Output:
live - 2
mostly - 2
africa - 1
india - 1
lions - 1
tigers - 1
white - 1
wild - 1

Завдання 2:

Тепер, нам потрібно виконати задачу, що називається словниковим індексуванням. Для текстового файлу виведіть усі слова в алфавітному порядку разом із номерами сторінок, на яких Ці слова знаходяться. Ігноруйте всі слова, які зустрічаються більше 100 разів. Припустимо, що сторінка являє собою послідовність із 45 рядків.

Наприклад, якщо взяти книгу Pride and Prejudice, перші кілька записів індексу будуть:

abatement - 89
abhorrence - 101, 145, 152, 241, 274, 281
abhorrent - 253
abide - 158, 292

2. Опис алгоритму

Завдання 1

1. Зчитати всі рядки з текстового файлу
2. Зчитати наступне слово, привівши всі літери до нижнього регістру. Якщо слів не залишилося - перейти до кроку 6
3. Якщо слово є стоп-словом - перейти до кроку 2
4. Якщо масив повторень містить зчитане слово - збільшити кількість повторень на 1, інакше - додати слово до масиву 5. Перейти до кроку 2
6. Відсортувати масив алгоритмом бульбашки
7. Записати перші 25 слів відсотрованого масиву до вихідного файлу

Завдання 2

1. Зчитати всі рядки з текстового файлу
2. Зчитати наступне слово, привівши всі літери до нижнього регістру. Якщо слів не залишилося - перейти до кроку 6 3. Знайти сторінку, на якій зустрілося слово
4. Якщо слово вже є в масиві слів, а сторінки нема - додати сторінку до масиву сторінок
5. Якщо слова нема в масиві слів - додати слово до масиву
6. Відсортувати масив слів в алфавітному порядку
7. Обрати наступне слово з масиву відсортованих слів. Якщо слів не залишилося - перейти до кроку 9
8. Якщо слово зустрілося менше 100 разів - записати слово й сторінки з повтореннями у файл. Перейти до кроку 7
9. Кінець алгоритму

3. Вихідний код

Завдання 1

```
using System;
using System.IO;

string inputFile = "input.txt";
string outputFile = "output.txt";
string inputString = File.OpenText(inputFile).ReadToEnd().ToLower();
string[] stopWords = { "in", "the", "for", "a", "to", "on", "at", "with",
"about", "before" };

int termPointer = 0;
Term[] terms = new Term[7];

using var outputStream = new StreamWriter(outputFile);
int characterPointer = 0;

string word = "";

ReadInput:
if (characterPointer < inputString.Length)
{
    if (inputString[characterPointer] == ' ')
    {
        int stopWordIndexer = 0;
        TestWord:
        if (stopWordIndexer < stopWords.Length)
        {
            if(stopWords[stopWordIndexer] == word)
                goto StartNextWord;

            stopWordIndexer++;
            goto TestWord;
        }

        bool isFound = false;
        int termIndexer = 0;
        TryFind:
        if (termIndexer < termPointer && !isFound)
        {
            if (terms[termIndexer].Word == word)
```

```
            {
                isFound = true;
            }
            else
                termIndexer++;
            goto TryFind;
        }

        if (!isFound)
        {
            termPointer++;
            terms[termIndexer].Word = word;

            if (termPointer > terms.Length * .7f)
            {
                var expandedArr = new Term[(int)(terms.Length * 1.35f)];
                int copyIndexer = 0;
                CopyArray:
                if (copyIndexer < terms.Length)
                {
                    expandedArr[copyIndexer] = terms[copyIndexer];
                    copyIndexer++;
                    goto CopyArray;
                }

                terms = expandedArr;
            }
        }

        terms[termIndexer].Frequency++;

        StartNextWord:
        word = "";
    }
    else
        word += inputString[characterPointer];

    characterPointer++;
    goto ReadInput;
}

int num = terms.Length;
int i = 0;
int j = 0;
Sort1:
if(i < num - 1)
{
    j = 0;

    Sort2:
    if (j < num - i - 1)
    {
        if (terms[j + 1].Frequency > terms[j].Frequency)
        {
            var tmp = terms[j];
            terms[j] = terms[j + 1];
            terms[j + 1] = tmp;
        }

        j++;
        goto Sort2;
    }

    i++;
```

```
            goto Sort1;
        }

        bool end = false;
        int wordPointer = 0;
        PrintWord:
        if (wordPointer < terms.Length && !end)
        {
            if (terms[wordPointer].Frequency == 0)
            {
                end = true;
                goto PrintWord;
            }

            Console.WriteLine($"{terms[wordPointer].Word} -
        {terms[wordPointer].Frequency}");
            outputStream.WriteLine($"{terms[wordPointer].Word} -
        {terms[wordPointer].Frequency}");
            wordPointer++;
            goto PrintWord;
        }

        public struct Term
        {
            public string Word;
            public int Frequency;
        };
```

Завдання 2

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;

string inputFile = "input.txt";
string outputFile = "output.txt";
List<string> stopWords = new() { "in", "the", "for", "a", "to", "on", "at", "with",
"about", "before" };

Dictionary<string, List<int>> tearmFrequancy = new();
var bookStream = File.OpenText(inputFile);
using var outputStream = new StreamWriter(outputFile);

int pageIndex = 1;
int lineIndex = 1;
int linesPrePage = 45;

ReadBook:
if (!bookStream.EndOfStream)
{
    if (lineIndex % linesPrePage == 0)
        pageIndex++;

    string line = bookStream.ReadLine()?.ToLower() ?? string.Empty;
    int characterPointer = 0;
    string word = "";

    ReadLine:
    if (characterPointer < line.Length)
    {
        if (line[characterPointer] == ' ')
```

```
        {
            word = Regex.Replace(word.ToLower(), "[^a-zA-Z]", "");
            if (string.IsNullOrEmpty(word))
            {
                goto StartNextWord;
            }

            int stopWordIndexer = 0;
            TestWord:
            if (stopWordIndexer < stopWords.Count)
            {
                if(stopWords[stopWordIndexer] == word)
                    goto StartNextWord;

                stopWordIndexer++;
                goto TestWord;
            }

            if (!tearmFrequancy.ContainsKey(word))
            {
                tearmFrequancy.Add(word, new List<int> {pageIndex});
            }
            else
            {
                tearmFrequancy[word].Add(pageIndex);
                if (tearmFrequancy[word].Count >= 100)
                {
                    tearmFrequancy.Remove(word);
                    stopWords.Add(word);
                }
            }

            StartNextWord:
            word = "";
        }else
            word += line[characterPointer];

        characterPointer++;
        goto ReadLine;
    }

    lineIndex++;
    goto ReadBook;
}

var wordsArr = tearmFrequancy.Keys.ToArray();
int num = wordsArr.Length;
int i = 0;
int j = 0;
Sort1:
if(i < num - 1)
{
    j = 0;

    Sort2:
    if (j < num - i - 1)
    {
        if (String.Compare(wordsArr[j], wordsArr[j + 1], StringComparison.Ordinal)
> 0)
        {
            var tmp = wordsArr[j];
            wordsArr[j] = wordsArr[j + 1];
            wordsArr[j + 1] = tmp;
        }
```

```
        j++;
        goto Sort2;
    }

    i++;
    goto Sort1;
}

int wordPointer = 0;
PrintWord:
if (wordPointer < wordsArr.Length && wordPointer < 25)
{
    Console.Write($"{wordsArr[wordPointer]} - ");
    outputStream.Write($"{wordsArr[wordPointer]} - ");
    int dataIndex = 0;
    PrintPage:
    if (dataIndex < tearmFrequancy[wordsArr[wordPointer]].Count)
    {
        Console.Write(tearmFrequancy[wordsArr[wordPointer]][dataIndex] + " ");
        outputStream.Write(tearmFrequancy[wordsArr[wordPointer]][dataIndex] + " ");
        dataIndex++;
        goto PrintPage;
    }
    Console.Write('\n');
    outputStream.Write('\n');

    wordPointer++;
    goto PrintWord;
}
```

4. Приклади роботи

```
1  live - 2
2  mostly - 2
3  white - 1
4  tigers - 1
5  india - 1
6  wild - 1
7  lions - 1
8
```

```
abatement - 99
abhorrence - 111 160 167 263 306
abhorrent - 276
abide - 174 318
abiding - 177
abilities - 72 107 155 171 194
able - 20 37 58 78 84 86 88 92 98 101 107 107 109 110 120 126 130 131 145 152 156 172 177 178 184 186 187
226 227 231 233 238 243 246 252 253 260 260 261 263 264 268 269 283 287 297 298 308
abode - 59 60 66 110 130 176 260
abominable - 32 51 71 71 122 161
abominably - 48 133 269 299
abominate - 263 296
abound - 101
above - 11 11 32 153 179 195 202 210 212 213 214 214 218 220 232 237 256 257 262 278 284
abroad - 194 233
abrupt - 203
abruptly - 155
abruptness - 198 198
absence - 54 56 64 77 78 90 99 99 100 106 106 111 127 150 172 172 195 195 197 207 224 232 239 283
absent - 31 199 225 229
absolute - 78 253
absolutely - 17 25 32 92 147 166 167 171 190 203 242 260 269 299
absurd - 61 296 302
absurdities - 127 217
absurdity - 189
abundant - 227

```