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

Кафедра інформатики та програмної інженерії

Програмування інтелектуальних інформаційних систем

ЗВІТ

до лабораторних робіт

Виконав студент ІТ-04 Мирошниченко А. О _____
(№ групи, прізвище, ім'я, по батькові)

Прийняв ас. Очеретяний О. К. _____
(посада, прізвище, ім'я, по батькові)

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**.

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.

2. Опис використаних технологій

Використовується мова C# з конструкцією goto

3. Опис програмного коду до завдань

Завдання 1:
Спочатку ми зчитуємо дані с файлу task1.txt. Записуємо слова, які нам потрібні, в масив string[] words. Після цього за допомогою імітації роботи циклу while та конструкції goto записуємо кожне слово з readingText у words. Для коректності розрахунку слів переводимо їх у нижній регістр та перевіряємо, чи містяться тексті слова по типу by, the, for і так далі. Також є два масиви:
string[] wordsSeenOnce – сюди записуємо слова, які зустрілись один раз.
int[] wordsSeenOnceCount – сюди записуємо кількість слів, які зустрілись один раз.

Всі вони містять інформацію з words. За допомогою goto проходимось по всьому words та перевіряємо чи всі з wordsSeenOnce записані. Якщо ні, то додаємо їх, а значення wordsSeenOnceCount стає більшим на одиницю.

Після цього йде сортування bubble та вивід у консоль

Завдання 2:
Копіюємо код з Завдання 1. Він нам знадобиться. Також оголошуємо змінну wordsOnPages, де будуть записані слова на сторінках. Кінець сторінки – 45 символ нового рядка. Проходимо через масив унікальних слів. Записуємо в string[] wordLess100Times ті слова, які зустрічаються менше 100 разів в wordsSeenOnceCount. Після цього сортування bubble та вивід слів з їх елементами в консоль.

4. Скріншоти роботи програмного застосунку

Завдання 1:

Завдання 2:

```
a-shooting - 239,

abatement - 73,

abhorrence - 83, 121, 128, 207, 234, 240,

abhorrent - 217,

abide - 133,

abiding - 136,

abilities - 52, 53, 80, 118, 131, 149,

able - 12, 25, 41, 57, 61, 63, 64, 67, 72, 75, 80, 82, 90, 95, 98, 109, 110, 115,
, 159, 169, 171, 176, 177, 180, 182, 186, 190, 192, 198, 199, 204, 205, 207, 210,

ablution - 90,

ably - 109,

abode - 42, 48, 82, 92, 98, 135, 204,

abominable - 22, 35, 51, 91, 122,

abominably - 33, 100, 211, 234,
```

5. Код програми до завдань

Завдання 1:

```
using System;
using System.IO;

namespace Task1
{
    class Task1
    {
        static void Main(string[] args)
        {
            string readingText = File.ReadAllText(@"..\..\..\..\..\task1.txt");
            int textLength = readingText.Length;
            int highNumber = 500;
            bool mustBe = true;
            int iterator = 0;
            string readingWord = "";
            string[] words = new string[1000000];
            int wordCount = 0;
            whileLoopTry:
            if (readingText[iterator] >= 65 && readingText[iterator] <=90 || reading-
Text[iterator] >= 97 && readingText[iterator] <=122 ||
                readingText[iterator] == 45)
            {
                if (readingText[iterator] >= 65 && readingText[iterator] <= 90)
                {
                    readingWord += (char)(readingText[iterator] + 32);
                }
                else
```

```
                {
                    readingWord += readingText[iterator];
                }
            }
            else
            {
                if (readingWord != "" && readingWord != null && readingWord != "-" &&
readingWord != "no" && readingWord != "from"
                    && readingWord != "the" && readingWord != "by" && readingWord !=
"and" && readingWord != "i"
                    && readingWord != "in" && readingWord != "or" && readingWord != "any"
&& readingWord != "for"
                    && readingWord != "to" && readingWord != "\"" && readingWord != "a"
&& readingWord != "on"
                    && readingWord != "of" && readingWord != "at" && readingWord != "is"
&& readingWord != "\n"
                    && readingWord != "\r" && readingWord != "\r\n" && readingWord !=
"\n\r")
                {

                    words[wordCount] = readingWord;
                    wordCount++;
                }

                readingWord = "";
            }
            iterator++;

            if (iterator < textLength)
            {
                goto whileLoopTry;
            }
            else
            {
                if (readingWord != "" && readingWord != null && readingWord != "-" &&
readingWord != "no" && readingWord != "from"
                    && readingWord != "the" && readingWord != "by" && readingWord !=
"and" && readingWord != "i"
                    && readingWord != "in" && readingWord != "or" && readingWord != "any"
&& readingWord != "for"
                    && readingWord != "to" && readingWord != "\"" && readingWord != "a"
&& readingWord != "on"
                    && readingWord != "of" && readingWord != "at" && readingWord != "is"
&& readingWord != "\n"
                    && readingWord != "\r" && readingWord != "\r\n" && readingWord !=
"\n\r")
                {
                    words[wordCount] = readingWord;
                    wordCount=wordCount+1;
                }
            }

            string[] wordsSeenOnce= new string[1000000];
            int[] wordsSeenOnceCount=new int[1000000];


            int wordsAmount = words.Length;
            iterator = 0;
            int positionInsert;
            int j;
            int duplicates = 0;

            whileCount:
                positionInsert = 0;
                int lengthNow = wordsSeenOnce.Length;
```

```
                j = 0;

                loopForImitation:
                if(wordsSeenOnce[j] != null && j > lengthNow)
                {
                    if (wordsSeenOnce[j]==words[iterator])
                    {
                        positionInsert = j;
                        mustBe = false;
                        goto forEnd;
                    }
                    j++;
                    goto loopForImitation;
                }

                forEnd:
                if (mustBe)
                {
                    wordsSeenOnce[iterator - duplicates] = words[iterator];
                    wordsSeenOnceCount[iterator - duplicates] = 1;

                }
                else
                {
                    wordsSeenOnceCount[positionInsert] += 1;
                    duplicates++;
                }
                iterator++;

                if(words[iterator]!=null && iterator<wordsAmount)
                {
                    goto whileCount;
                }

            int length = wordsSeenOnceCount.Length;
            j = 0;

            int innerIterator;
            sort:
            if (wordsSeenOnceCount[j] != 0 && j <length)
            {
                innerIterator = 0;
            sortInner:
                if (wordsSeenOnceCount[innerIterator] != 0 && innerIterator < length - j
- 1)
                {
                    if (wordsSeenOnceCount[innerIterator] < wordsSeenOnceCount[innerIter-
ator + 1])
                    {

                        string temporaryWord = wordsSeenOnce[innerIterator];
                        wordsSeenOnce[innerIterator] = wordsSeenOnce[innerIterator + 1];
                        wordsSeenOnce[innerIterator + 1] = temporaryWord;

                        int temporaryNumber = wordsSeenOnceCount[innerIterator];
                        wordsSeenOnceCount[innerIterator] = wordsSeenOnceCount[innerIter-
ator + 1];
                        wordsSeenOnceCount[innerIterator + 1] = temporaryNumber;

                    }
                    innerIterator++;
                    goto sortInner;
                }
                j++;
                goto sort;
```

```
            }

        int printWords = 0;
        print:
            if (wordsSeenOnce[printWords] != null && printWords < highNumber && print-
Words < length)
            {
                Console.WriteLine("{0} - {1}", wordsSeenOnce[printWords],
wordsSeenOnceCount[printWords]);
                printWords++;
                goto print;
            }
        }
    }
}
```

Завдання 2:

```
using System;
using System.Collections.Generic;
using System.IO;

namespace Task2
{
    class Task2
    {
        static void Main(string[] args)
        {
            string readingText = File.ReadAllText(@"..\..\..\..\..\task2.txt");
            int iterator = 0;
            string readingWord = "";
            bool mustBe;
            int textLength = readingText.Length;
            string[] words = new string[100000];
            string[,] wordsOnPages = new string[10000, 10000];
            int pages = 0;
            int wordCount = 0;
            int rows = 0;
            int wordsOnPageCount = 0;

            whileImitation:
            if (readingText[iterator] >= 65 && readingText[iterator] <= 90 || reading-
Text[iterator] >= 97 && readingText[iterator] <= 122 ||
                readingText[iterator] == 45 || readingText[iterator] == 225 || reading-
Text[iterator] == 234 || readingText[iterator] == 224)
            {
                if (readingText[iterator] >= 65 && readingText[iterator] <= 90)
                {
                    readingWord += (char)(readingText[iterator] + 32);
                }
                else
                {
                    readingWord += readingText[iterator];
                }
            }
            else
            {
                if (readingText[iterator] == '\n')
                {
                    rows++;

                }
```

```
                if (rows > 45)
                {
                    pages++;
                    wordsOnPageCount = 0;
                    rows = 0;
                }

                if (readingWord != "" && readingWord != null && readingWord != "-" &&
readingWord != "no" && readingWord != "from"
                    && readingWord != "the" && readingWord != "by" && readingWord !=
"and" && readingWord != "i"
                    && readingWord != "in" && readingWord != "or" && readingWord != "any"
&& readingWord != "for"
                    && readingWord != "to" && readingWord != "\"" && readingWord != "a"
&& readingWord != "on"
                    && readingWord != "of" && readingWord != "at" && readingWord != "is"
&& readingWord != "\n"
                    && readingWord != "\r" && readingWord != "\r\n" && readingWord !=
"\n\r")
                {
                    words[wordCount] = readingWord;
                    wordCount++;
                    wordsOnPages[pages, wordsOnPageCount] = readingWord;
                    wordsOnPageCount++;
                }

                readingWord = "";

            }

            iterator++;

            if (iterator < textLength)
            {
                goto whileImitation;
            }
            else
            {
                if (readingWord != "" && readingWord != null && readingWord != "-" &&
readingWord != "no" && readingWord != "from"
                    && readingWord != "the" && readingWord != "by" && readingWord !=
"and" && readingWord != "i"
                    && readingWord != "in" && readingWord != "or" && readingWord != "any"
&& readingWord != "for"
                    && readingWord != "to" && readingWord != "\"" && readingWord != "a"
&& readingWord != "on"
                    && readingWord != "of" && readingWord != "at" && readingWord != "is"
&& readingWord != "\n"
                    && readingWord != "\r" && readingWord != "\r\n" && readingWord !=
"\n\r")
                {
                    words[wordCount] = readingWord;
                    wordCount++;
                }
            }

            int[] wordsSeenOnceCount = new int[100000];
            string[] wordsSeenOnce = new string[100000];


            int wordsAmount = words.Length;
            iterator = 0;
            int j;
            int positionInsert;
```

```
    int duplicates = 0;

whileCountImitation:
    positionInsert = 0;
    int lengthNow = wordsSeenOnce.Length;
    j = 0;
    mustBe = true;

    forImitation:
    if (wordsSeenOnce[j] != null && j < lengthNow)
    {
        if (wordsSeenOnce[j] == words[iterator])
        {
            positionInsert = 1;
            mustBe = false;
            goto endForImitation;
        }
        j++;
        goto forImitation;
    }

    endForImitation:
    if (mustBe)
    {
        wordsSeenOnce[iterator - duplicates] = words[iterator];
        wordsSeenOnceCount[iterator - duplicates] = 1;
    }
    else
    {
        wordsSeenOnceCount[positionInsert] += 1;
        duplicates++;
    }

    iterator++;
    if(words[iterator] != null && iterator < wordsAmount)
    {
        goto whileCountImitation;
    }


    int length = wordsSeenOnceCount.Length;
    int a = 0;
    string[] wordLess100Times=new string[100000];
    int lastWordInsert = 0;
    Less100Times:
    if (wordsSeenOnce[a] != null && a < length)
    {
        if (wordsSeenOnceCount[a] <= 100)
        {
            wordLess100Times[lastWordInsert] = wordsSeenOnce[a];
            lastWordInsert++;
        }
        a++;
        goto Less100Times;
    }

    int sorted;
    int count;
    bool swapWord;
    int wordWritten = 0;
    int currentWordLength;
    int nextWordLength;

    firstSortImitation:
```

```
            if (wordLess100Times[wordWritten] != null && wordWritten < word-
Less100Times.Length)
            {
                sorted = 0;
                secondSortImitation:

                if (wordLess100Times[sorted+1] != null && sorted < word-
Less100Times.Length - wordWritten - 1)
                {
                    currentWordLength = wordLess100Times[sorted].Length;
                    nextWordLength = wordLess100Times[sorted + 1].Length;

                    int wordLengthComparison;

                    if (nextWordLength < currentWordLength)
                    {
                        wordLengthComparison = nextWordLength;

                    }
                    else
                    {
                        wordLengthComparison = currentWordLength;
                    }

                    swapWord = false;
                    count = 0;

                    alphabetExamination:
                    if (wordLess100Times[sorted+1][count] < word-
Less100Times[sorted][count])
                    {
                        swapWord = true;
                        goto alphabetEnd;

                    }

                    if (wordLess100Times[sorted + 1][count] > word-
Less100Times[sorted][count])
                    {
                        goto alphabetEnd;
                    }

                    count++;
                    if (wordLengthComparison > count)
                    {
                        goto alphabetExamination;
                    }

                    alphabetEnd:
                    if (swapWord)
                    {
                        string temporaryWord = wordLess100Times[sorted];
                        wordLess100Times[sorted] = wordLess100Times[sorted + 1];
                        wordLess100Times[sorted + 1] = temporaryWord;
                    }

                    sorted++;
                    goto secondSortImitation;

                }

                wordWritten++;
                goto firstSortImitation;

            }
```

```
            a = 0;
            int wordLengthSeenLess100 = wordLess100Times.Length;

            print:
            if (wordLess100Times[a] != null && a < wordLengthSeenLess100)
            {
                Console.Write("{0} - ", wordLess100Times[a]);

                int dimensionOne = 0;
                int dimensionTwo;

                int[] pagesWithWords = new int[100];
                int insertIntoPage = 0;

                pageExamination:

                if (wordsOnPages[dimensionOne, 0] != null && dimensionOne < 10000)
                {
                    dimensionTwo = 0;

                    pageCheckOnWord:

                    if (wordsOnPages[dimensionOne, dimensionTwo] != null && dimensionTwo
< 10000)
                    {

                        if (wordsOnPages[dimensionOne, dimensionTwo] == word-
Less100Times[a])
                        {

                            pagesWithWords[insertIntoPage] = dimensionOne+1;
                            insertIntoPage++;
                            dimensionOne++;
                            goto pageExamination;

                        }

                        dimensionTwo++;
                        goto pageCheckOnWord;

                    }

                    dimensionOne++;
                    goto pageExamination;
                }

                int secondCount = 0;

                pagination:

                if (pagesWithWords[secondCount] !=0 && secondCount < 100)
                {

                    if (pagesWithWords[secondCount+1] != 0 && secondCount != 99)
                    {
                        Console.Write("{0}, ", pagesWithWords[secondCount]);
                    }
                    else
                    {
                        Console.Write("{0}, ", pagesWithWords[secondCount]);
                    }

                    secondCount++;
                    goto pagination;
```

```
                }

                Console.WriteLine("\n");
                a++;
                goto print;
            }
        }
    }
}
```