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 Філіповський Данііл
(№ групи, прізвище, ім'я, по батькові)

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

Київ 2021

1. Завдання лабораторної роботи

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

2. Опис використаних технологій

Для виконання завдання використовувалсь мова C#(підтримує конструкцією GOTO).

Завдання 1:

Спочатку зчитуємо дані із текстового файлу і додаємо до змінної типу стрінг, котра буде використовуватись як масив char.

Слова, знайдені в тексті будуть додаватися до arr_words.

За допомогою goto проходимо масив до кінця, і записуємо кожне слово до нового масиву слів. Кінець слово будь який символ окрім малих і великих літер та знаку дефісу.

Для коректного переводу символу до малого регістру використаємо їх ASCII код додаючи 32, якщо символ виявиться великою літерою латиниці.

І ще буде перевірка на те, чи являється це слово стоп-словом з умови завдання.

При обробці слів вони відправляються у масив, в одному зберігаются унікальні слова, в іншому їх кількість. Якщо слово зустрічається не в перший раз, то у другий масив додаємо до елемента на тій же самій позиції одиницю.

В кінці виводимо зміст двох масивів з відповідним форматуванням.

Завдання 2:

Принцип роботи схожий на Завдання 1 за винятком деяких моментів.

Оголошуємо двомірний масив куди будемо записувати масиви слів для кожної сторінки. Кінцем сторінки зазвичай вважається досягненнея 45 символу нового рядка.

Після цього підраховуєм кількість слів.

Створюємо новий масив, перебираємо масив з унікальними входженнями і записуємо у новостворений слова з входженнями менше ніж 100

Далі виводимо слово на екран і перевіряємо його на масиві сторінок, виводимо елемент масиву(сторінки на яких було зафіксовано це слово).

3. Опис програмного коду

```
using System;
using System.IO;

namespace Task1
{
    class Program
    {
        static void Main(string[] args)
        {
            string text = File.ReadAllText(@"Text_Example.txt");
            int text_length = text.Length;
            int words_show = 5;
            int i = 0;
            string current_word = "";

            int word_count = 0;

            int insertPos = 0;
            int j = 0;
            int dubs = 0;
            string[] arr_words = new string[5000];



        while_loop:
            if ((text[i] >= 65) && (text[i] <= 90) || (text[i] >= 97) && (text[i] <= 122) ||
text[i] == 45)
            {
                if ((text[i] >= 65) && (text[i] <= 90))
                {
                    current_word += (char)(text[i] + 32);
                }
                else
                {
                    current_word += text[i];
                }
            }
            else
            {
                if (current_word != "" && current_word != null && current_word != "-" &&
current_word != "no"  && current_word != "\r\n" && current_word != "\n\r")
                {
                    arr_words[word_count] = current_word;
                    word_count++;
                }
                current_word = "";
            }
            i++;
            if (i < text_length)
            {
                goto while_loop;
            }
            else
            {
                if (current_word != "" && current_word != null && current_word != "-" &&
current_word != "no" && current_word != "\r\n" && current_word != "\n\r")
                {
```

```
            arr_words[word_count] = current_word;
            word_count++;
        }
    }
    string[] word_once_arr = new string[5000];
    int[] words_once_count_arr = new int[5000];

    int amount_of_words = arr_words.Length;

    i = 0;



while_loop_counter:
    insertPos = 0;
    int current_length = word_once_arr.Length;
    j = 0;




for_loop:
    if (j < current_length && word_once_arr[j] != null)
    {
        if (word_once_arr[j] == arr_words[i])
        {
            insertPos = j;
            goto for_loop_end;

        }
        j++;
        goto for_loop;
    }



for_loop_end:
    if (insertPos == 0)
    {
        word_once_arr[i - dubs] = arr_words[i];
        words_once_count_arr[i - dubs] = 1;
    }
    else
    {
        words_once_count_arr[insertPos] += 1;
        dubs++;
    }
    i++;
    if (i < amount_of_words && arr_words[i] != null)
    {
        goto while_loop_counter;
    }
    int length = words_once_count_arr.Length;
    j = 0;
    int inner_i = 0;



sort_loop:
    if (j < length && words_once_count_arr[j] != 0)
    {
        inner_i = 0;
    sort_inner_loop:
        if (inner_i < length - j - 1 && words_once_count_arr[inner_i] != 0)
        {
            if (words_once_count_arr[inner_i] < words_once_count_arr[inner_i + 1])
            {
                int temp = words_once_count_arr[inner_i];
```

```
                        words_once_count_arr[inner_i] = words_once_count_arr[inner_i + 1];
                        words_once_count_arr[inner_i + 1] = temp;



                        string temp2 = word_once_arr[inner_i];
                        word_once_arr[inner_i] = word_once_arr[inner_i + 1];
                        word_once_arr[inner_i + 1] = temp2;
                    }
                    inner_i++;
                    goto sort_inner_loop;
                }

                j++;
                goto sort_loop;
            }
            int f = 0;



        print_loop:
            if (f < length && word_once_arr[f] != null && f < words_show)
            {
                Console.WriteLine("{0} - {1}", word_once_arr[f], words_once_count_arr[f]);
                f++;
                goto print_loop;
            }
        }
    }
}




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

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {



            string text = File.ReadAllText(@"Text_Example.txt");

            int text_length = text.Length;

            int i = 0;

            string curr_word = "";

            string[] words_arr = new string[99000];

            string[,] page_word_arr = new string[9000, 9000];

            int word_count = 0;

            int row_count = 0;

            int page_count = 0;

            int page_word_counter = 0;
```

```
        while_loop:
            if ((text[i] >= 65) && (text[i] <= 90) || (text[i] >= 97) && (text[i] <= 122) ||
text[i] == 45 || text[i] == 234 || text[i] == 225 || text[i] == 224)
            {
                if ((text[i] >= 65) && (text[i] <= 90))
                {
                    curr_word += (char)(text[i] + 32);
                }
                else
                {
                    curr_word += text[i];
                }
            }
            else
            {
                if (text[i] == '\n')
                {
                    row_count++;
                }
                if (row_count > 45)
                {
                    page_count++;
                    page_word_counter = 0;
                    row_count = 0;
                }
                if (curr_word != "" && curr_word != null && curr_word != "-" && curr_word !=
"no" && curr_word != "from" && curr_word != "the" && curr_word != "by" && curr_word != "and" &&
curr_word != "i" && curr_word != "in" && curr_word != "or" && curr_word != "any" && curr_word
!= "for" && curr_word != "to" && curr_word != "\"" && curr_word != "a" && curr_word != "on" &&
curr_word != "of" && curr_word != "at" && curr_word != "is" && curr_word != "\n" && curr_word
!= "\r" && curr_word != "\r\n" && curr_word != "\n\r")
                {

                    words_arr[word_count] = curr_word;
                    word_count++;
                    page_word_arr[page_count, page_word_counter] = curr_word;
                    page_word_counter++;
                }
                curr_word = "";
            }
            i++;
            if (i < text_length)
            {
                goto while_loop;
            }
            else
            {
                if (curr_word != "" && curr_word != null && curr_word != "-" && curr_word !=
"no" && curr_word != "from" && curr_word != "the" && curr_word != "by" && curr_word != "and" &&
curr_word != "i" && curr_word != "in" && curr_word != "or" && curr_word != "any" && curr_word
!= "for" && curr_word != "to" && curr_word != "\"" && curr_word != "a" && curr_word != "on" &&
curr_word != "of" && curr_word != "at" && curr_word != "is" && curr_word != "\n" && curr_word
!= "\r" && curr_word != "\r\n" && curr_word != "\n\r")
                {
                    words_arr[word_count] = curr_word;
                    word_count++;
                }
            }




            string[] words_once_arr = new string[99000];

            int[] words_once_count_arr = new int[99000];
```

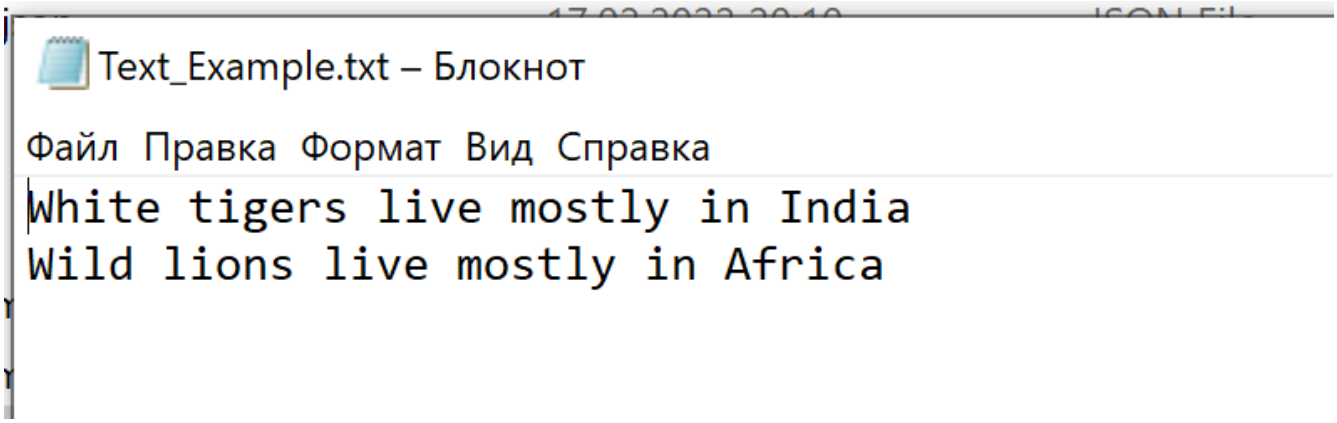
```
    int words_amount = words_arr.Length;

    i = 0;

    int insertPos = 0;

    int j = 0;

    int dubs = 0;


while_loop_count:
    insertPos = 0;

    int current_length = words_once_arr.Length;

    j = 0;




for_loop:
    if (j < current_length && words_once_arr[j] != null)
    {
        if (words_once_arr[j] == words_arr[i])
        {
            insertPos = j;
            goto end_for_loop;

        }
        j++;
        goto for_loop;
    }
end_for_loop:
    if (insertPos == 0)
    {
        words_once_arr[i - dubs] = words_arr[i];
        words_once_count_arr[i - dubs] = 1;
    }
    else
    {
        words_once_count_arr[insertPos] += 1;
        dubs++;
    }
    i++;
    if (i < words_amount && words_arr[i] != null)
    {
        goto while_loop_count;
    }


    int length = words_once_count_arr.Length;

    int k = 0;

    string[] words_once_arr_less_than_100 = new string[99000];

    int LastInsert = 0;



words_less_than_100_loop:
    if (k < length && words_once_arr[k] != null)
    {
        if (words_once_count_arr[k] <= 100)
        {
            words_once_arr_less_than_100[LastInsert] = words_once_arr[k];
            LastInsert++;
```

```
                }
                k++;
                goto words_less_than_100_loop;
            }
            int write = 0;
            int sort = 0;
            bool toSwapWords = false;
            int counter = 0;
            int word_lenth_cur = 0;
            int word_lenth_next = 0;
        sort_loop:
            if (write < words_once_arr_less_than_100.Length &&
words_once_arr_less_than_100[write] != null)
            {
                sort = 0;
            inner_sort_loop:
                if (sort < words_once_arr_less_than_100.Length - write - 1 &&
words_once_arr_less_than_100[sort + 1] != null)
                {
                    word_lenth_cur = words_once_arr_less_than_100[sort].Length;
                    word_lenth_next = words_once_arr_less_than_100[sort + 1].Length;

                    int compare_lenth = word_lenth_cur > word_lenth_next ? word_lenth_next :
word_lenth_cur;

                    toSwapWords = false;
                    counter = 0;
                alphabet_check:

                    if (words_once_arr_less_than_100[sort][counter] >
words_once_arr_less_than_100[sort + 1][counter])
                    {
                        toSwapWords = true;
                        goto alphabet_check_end;
                    }
                    if (words_once_arr_less_than_100[sort][counter] <
words_once_arr_less_than_100[sort + 1][counter])
                    {
                        goto alphabet_check_end;
                    }
                    counter++;
                    if (counter < compare_lenth)
                    {
                        goto alphabet_check;
                    }
                alphabet_check_end:
                    if (toSwapWords)
                    {
                        string temp = words_once_arr_less_than_100[sort];
                        words_once_arr_less_than_100[sort] = words_once_arr_less_than_100[sort
+ 1];
                        words_once_arr_less_than_100[sort + 1] = temp;
                    }
                    sort++;
                    goto inner_sort_loop;
                }
                write++;
                goto sort_loop;
            }
            k = 0;
            int less_than_100_length = words_once_arr_less_than_100.Length;
        print_loop:
            if (k < less_than_100_length && words_once_arr_less_than_100[k] != null)
            {
                Console.Write("{0} - ", words_once_arr_less_than_100[k]);
                int first_dim = 0;
                int second_dim = 0;
                int[] word_pages = new int[100];
                int pageInsert = 0;
```

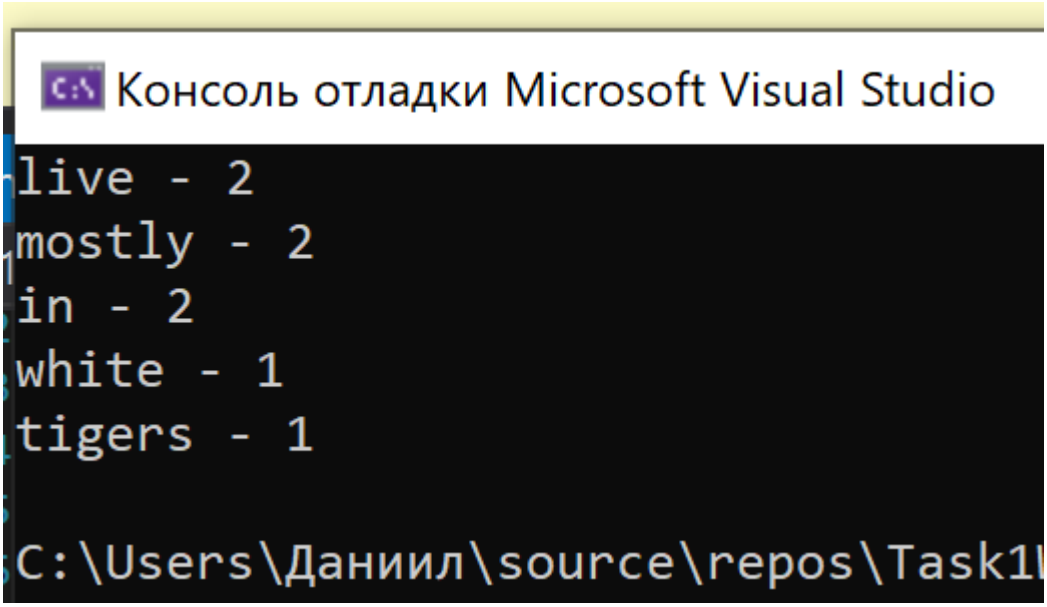
```
            check_page:
                if (first_dim < 10000 && page_word_arr[first_dim, 0] != null)
                {
                    second_dim = 0;
                check_page_word:
                    if (second_dim < 10000 && page_word_arr[first_dim, second_dim] != null)
                    {
                        if (page_word_arr[first_dim, second_dim] ==
words_once_arr_less_than_100[k])
                        {
                            word_pages[pageInsert] = first_dim + 1;
                            pageInsert++;
                            first_dim++;
                            goto check_page;
                        }
                        second_dim++;
                        goto check_page_word;
                    }

                    first_dim++;
                    goto check_page;
                }
                int tired_counte = 0;
            pagination_loop:
                if (tired_counte < 100 && word_pages[tired_counte] != 0)
                {
                    if (tired_counte != 99 && word_pages[tired_counte + 1] != 0)
                    {
                        Console.Write("{0}, ", word_pages[tired_counte]);
                    }
                    else
                    {
                        Console.Write("{0}", word_pages[tired_counte]);
                    }
                    tired_counte++;
                    goto pagination_loop;
                }
                Console.WriteLine();
                k++;
                goto print_loop;
            }
        }
    }
}
```

4. Скріншоти роботи програмного застосунку

Завдання 1:
Input:


Text_Example.txt – Блокнот
Файл Правка Формат Вид Справка
White tigers live mostly in India
Wild lions live mostly in Africa

Output:


Консоль отладки Microsoft Visual Studio
live - 2
mostly - 2
in - 2
white - 1
tigers - 1
C:\Users\Даниил\source\repos\Task1

Завдання 2:
Input:

Text_Example.txt – Блокнот

Ф Отмена невозможна. т Вид Справка

```
Pride and Prejudice


Jane Austen




Chapter 1

It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife

However little known the feelings or views of such a man may be on his first entering a neighbourhood, this truth is

"My dear Mr. Bennet," said his lady to him one day, "have you heard that Netherfield Park is let at last?"

Mr. Bennet replied that he had not.

"But it is, returned she; "for Mrs. Long has just been here, and she told me all about it.

Mr. Bennet made no answer.
```

Output:

Выбрать C:\Users\Даниил\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\net5.0\ConsoleApp1.exe

```
abatement - 33
abhorrence - 36, 52, 53, 79, 93, 95
abhorrent - 84
abide - 54
abiding - 55
abilities - 26, 27, 35, 50, 53, 61
able - 7, 14, 22, 28, 30, 31, 33, 34, 35, 36, 39, 41, 42, 46, 49, 51, 54, 55, 57, 58, 59, 61, 64, 66, 67, 68, 69, 70, 72
, 74, 75, 77, 79, 81, 82, 87, 89, 92, 93, 96, 99
ablution - 39
abode - 23, 24, 36, 40, 42, 55, 79
abominable - 12, 20, 26, 40, 52
abominably - 18, 43, 82, 93
abominate - 79, 92
abound - 34
abouts - 77
above - 4, 12, 49, 56, 61, 63, 65, 66, 67, 70, 72, 78, 79, 85, 87
abroad - 61, 70, 89
abruptly - 16, 50
abruptness - 61, 62
abrupt - 64
absence - 21, 22, 24, 27, 28, 31, 33, 34, 35, 36, 41, 48, 53, 61, 64, 68, 70, 72, 87
absent - 11, 62, 68, 69
absolutely - 6, 9, 12, 31, 32, 41, 47, 53, 59, 64, 67, 73, 79, 82, 93, 94
absolute - 28, 68, 77, 96
absurd - 23, 52, 53, 92, 94
absurdities - 41, 66
absurdity - 59
abundantly - 24, 30, 41
abundant - 69
abuse - 2, 53
```