Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 1 з дисципліни
«Мультипарадигменне програмування»

«Імперативне програмування»

Виконав: ІП-02 Гончар О. О.

Київ 2021

Лабораторна робота 1

Практична робота складається із трьох завдань, які самі по собі є досить простими. Але, оскільки задача - зрозуміти, як писали код наші славні пращури у 1950-х, ми введемо кілька обмежень:

- Заборонено використовувати функції
- Заборонено використовувати цикли
- Для виконання потрібно взяти мову, що підтримує конструкцію GOTO

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.

Завдання 1

Алгоритм

Для реалізації даної задачі необхідно:

1. Посимвольно зчитати увесь текст із файлу вводу;
2. В умовах задачі було зазначено, що необхідно ігнорувати регістр, тому до поточного слова ми додаємо символ, якщо він знаходиться у нижньому регістрі, або додаємо переведений у нижній регістр символ. Потім зчитуємо наступний символ. Робимо так до тих пір, поки зчитаний символ є літерою;
3. Додаємо поточне слово до масиву слів, якщо воно не порожнє. Якщо його довжина менша за вказану користувачем, також ігноруємо це слово;
4. Збільшуємо кількість слів, якщо воно вже є у масиві, або створюємо нове;
5. Сортування бульбашкою за спаданням (за кількістю повторів слів);
6. Виведення у файл.

Реалізація

```
//=====================
//Some settings
//=====================
int stopWordMinLenght = 3; // https://t.me/multiparadigm_labs/425
int countCommonWords = 25; // count of the most common words (25 by default)
string inputFilePath = "input.txt"; // name of the input file
string outputFilePath = "output.txt"; // name of the output file

//=====================
//Variables initialization
//=====================
string[] wordsArray = new string[1], resultWordsArray = new
string[countCommonWords]; // arrays of words
int[] wordsCount = new int[1], resultWordsCount = new int[countCommonWords]; //
arrays of count of words with corresponding index
int currentChar; // ascii code of the current char
int lastWordPointer = 0, currentWordIndex; // {lastWordPointer} index of the last
filled word in array, {currentWordIndex} used for adding new word and copy array
int i = 0, j = 0; // indexes used everywhere
string currentWord = ""; // current word

//=====================
//Reading the input file
//=====================
StreamReader streamReader = new StreamReader(inputFilePath);
readingFile:
```

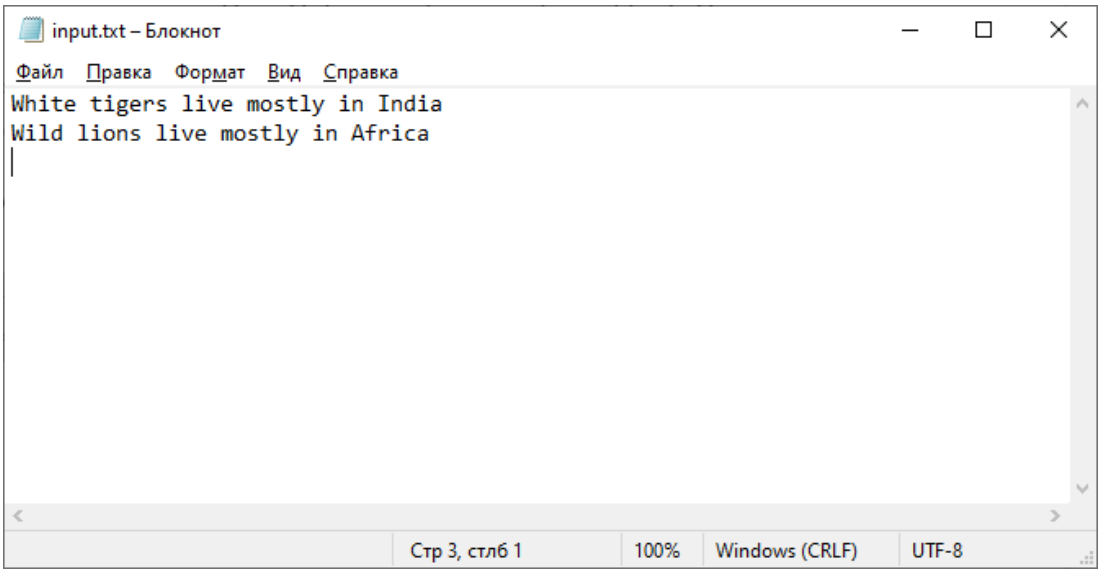
```
{
    // keep reading, if it isn't end of the file
    if (!streamReader.EndOfStream)
    {
        currentChar = streamReader.Read();
    }
    else
    {
        goto stopReading;
    }

    if (97 <= currentChar && currentChar <= 122) // lowercase symbols codes in
dec
    {
        currentWord = currentWord + (char)currentChar;
        goto nextChar;
    }
    else if (65 <= currentChar && currentChar <= 90) // uppercase symbols codes
in dec
    {
        currentWord = currentWord + (char)(currentChar + 32); // converting to
lowercase
        goto nextChar;
    }
    if (currentWord != "")
    {
        if (currentWord.Length < stopWordMinLenght) // length of the current word
couldn't be lower than {stopWordMinLenght}
        {
            currentWord = "";
        }
        else
        {
            currentWordIndex = 0;
            checkWords:
            {
                if (currentWordIndex == lastWordPointer) // we've checked all
words and didn't find any matches
                {
                    goto addNewWord;
                }
                if (currentWord == wordsArray[currentWordIndex]) // check if
current work is same with current word in array
                {
                    currentWord = "";
                    wordsCount[currentWordIndex]++;
                    goto nextChar;
                }
                currentWordIndex++;
                goto checkWords;
            }
```

```
            addNewWord:
            {
                if (lastWordPointer == wordsArray.Length) // if the last word is
in the last position we have to expand the array
                {
                    string[] newWordsArray = new string[wordsArray.Length * 2];
// optimized resizing
                    int[] newWordsCountArray = new int[wordsArray.Length * 2];
                    currentWordIndex = 0;
                    copyTheNextElement:
                    {
                        if (currentWordIndex == lastWordPointer) // we've copied
all the elements
                        {
                            wordsArray = newWordsArray;
                            wordsCount = newWordsCountArray;
                            goto addTheLastWord;
                        }
                        newWordsArray[currentWordIndex] =
wordsArray[currentWordIndex];
                        newWordsCountArray[currentWordIndex] =
wordsCount[currentWordIndex];
                        currentWordIndex++;
                        goto copyTheNextElement;
                    }
                }
            }
            addTheLastWord:
            {
                wordsArray[lastWordPointer] = currentWord; // if we find a new
word, it is single so far :(
                wordsCount[lastWordPointer] = 1;
                lastWordPointer++;
                currentWord = "";
            }
        }
    }

    nextChar:
    if (!streamReader.EndOfStream)
    {
        goto readingFile;
    }
    stopReading:
    {
        streamReader.Close();
    }
}

//====================
```

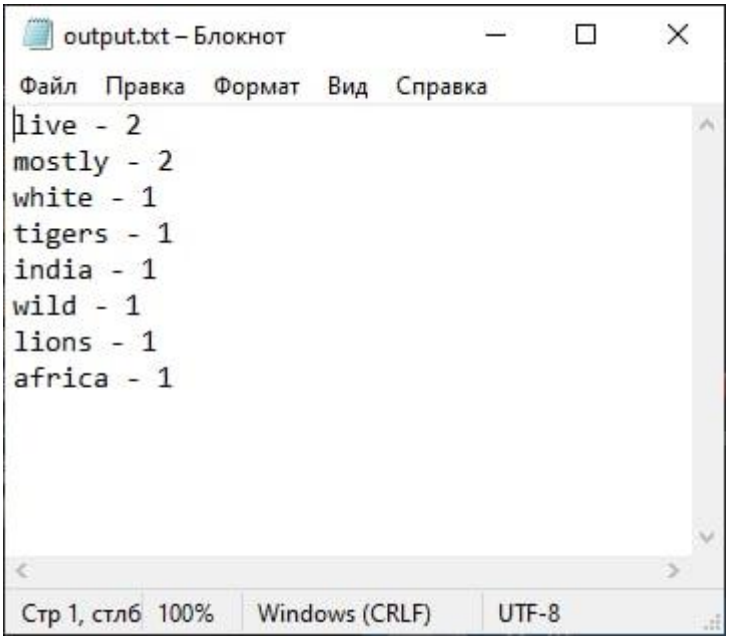
```
//Bubble sort
//=====================
i = lastWordPointer - 1;
outerLoop:
{
    j = 0;
    if (i >= 1)
    {
        innerLoop:
        {
            if (j < i)
            {
                if (wordsCount[j] < wordsCount[j+1])
                {
                    int tempCount = wordsCount[j];
                    string tempWord = wordsArray[j];
                    wordsCount[j] = wordsCount[j + 1];
                    wordsArray[j] = wordsArray[j + 1];
                    wordsCount[j + 1] = tempCount;
                    wordsArray[j + 1] = tempWord;
                }
                j = j + 1;
                goto innerLoop;
            }
        }
        i = i - 1;
        goto outerLoop;
    }
}

//=====================
//Setting limit of {countCommonWords}
//=====================
j = lastWordPointer - 1;
i = 0;
settingLimit:
{
    if (i < countCommonWords && i <= j && wordsCount[i] != 0)
    {
        resultWordsArray[i] = wordsArray[i];
        resultWordsCount[i] = wordsCount[i];
        i = i + 1;
        goto settingLimit;
    }
}

//=====================
//Writing to the file
//=====================
j = lastWordPointer - 1;
i = 0;
```

```
StreamWriter streamWriter = new StreamWriter(outputFilePath);
writeNextElement:
{
    if (i <= j)
    {
        streamWriter.WriteLine($"{resultWordsArray[i]} - {resultWordsCount[i]}");
        i = i + 1;
        goto writeNextElement;
    }
}
streamWriter.Close();
```

Приклад роботи:



Зміст файлу «input.txt»



Зміст файлу «output.txt»

Завдання 2

Алгоритм

Для реалізації даної задачі необхідно:

1. Построково зчитати файл та збільшити кількість строк на 1;
2. Якщо кількість строк дорівнює 45, обнулити її та збільшити номер сторінки;
3. Пройтися кожною строкою посимвольно та аналогічно до першої задачі поділити на слова;
4. Додати нове слово;
5. Відсортувати бульбашкою в алфавітному порядку;
6. Вивести до файлу результати.

Реалізація

```
//=====================
//Some settings
//=====================
string inputFilePath = "input.txt"; // name of the input file
string outputFilePath = "output.txt"; // name of the output file
int amountToIgnore = 100; // we should ignore all words that are more than
{countToIgnore}
int linesPerPage = 45;

//=====================
//Variables initialization
//=====================

string[] wordsArray = new string[1]; // array of words
int[] wordsCount = new int[1]; // array of count of words with corresponding
index
char currentChar; // current char
int lastWordPointer = 0, currentWordIndex; // {lastWordPointer} index of the last
filled word in array, {currentWordIndex} used for adding new word and copy array
int i = 0, j = 0; // indexes used everywhere
string currentWord = ""; // current word
int linesCount = 0; // count lines on the current page
string currentString = ""; // current line
int[][] pagesContent = new int[1][];
int currentPage = 0; // number of the current page

//=====================
//Reading the input file
//=====================
StreamReader streamReader = new StreamReader(inputFilePath);
readingFile:
{
```

```
    if (!streamReader.EndOfStream)
    {
        currentString = streamReader.ReadLine();
        linesCount = linesCount + 1;
    }
    else
    {
        goto stopReading;
    }

    if (linesCount == linesPerPage)
    {
        linesCount = 0;
        currentPage = currentPage + 1;
    }

    i = 0; // iterator for the current line
    iteratingInString:
    {
        if (i == currentString.Length)
        {
            goto nextChar;
        }

        currentChar = currentString[i];

        if (97 <= currentChar && currentChar <= 122) // lowercase symbols codes
in dec
        {
            currentWord = currentWord + currentChar;
            if (i + 1 < currentString.Length)
                goto nextChar;
        }
        else if (65 <= currentChar && currentChar <= 90) // uppercase symbols
codes in dec
        {
            currentWord = currentWord + (char)(currentChar + 32); // converting
to lowercase
            if (i + 1 < currentString.Length)
                goto nextChar;
        }

        if (currentWord != "")
        {
            currentWordIndex = 0;
            checkWords:
            {
                if (currentWordIndex == lastWordPointer) // we've checked all
words and didn't find any matches
                {
                    goto addNewWord;
```

```
                }
                if (currentWord == wordsArray[currentWordIndex]) // check if
current work is same with current word in array
                {
                    currentWord = "";
                    wordsCount[currentWordIndex] = wordsCount[currentWordIndex] +
1;
                    if (pagesContent[currentWordIndex].Length <
wordsCount[currentWordIndex]) // check if array length is enough
                    {
                        j = 0;
                        int[] newPages = new int[wordsCount[currentWordIndex] *
2]; // optimized resizing
                        nextCopy:
                        {
                            newPages[j] = pagesContent[currentWordIndex][j];
                            j = j + 1;
                            if (j < wordsCount[currentWordIndex] - 1)
                                goto nextCopy;
                        }
                        pagesContent[currentWordIndex] = newPages;
                        pagesContent[currentWordIndex][wordsCount[currentWordInde
x] - 1] = currentPage;
                    }
                    pagesContent[currentWordIndex][wordsCount[currentWordIndex] -
1] = currentPage;
                    goto nextChar;
                }
                currentWordIndex++;
                goto checkWords;
            }

            addNewWord:
            if (lastWordPointer == wordsArray.Length) // if the last word is in
the last position we have to expand the array
            {
                string[] newWordsArray = new string[wordsArray.Length * 2]; //
optimized resizing
                int[] newWordsCountArray = new int[wordsArray.Length * 2];
                int[][] newPagesArray = new int[wordsArray.Length * 2][];

                currentWordIndex = 0;
                copyTheNextElement:
                {
                    if (currentWordIndex == lastWordPointer)
                    {
                        wordsArray = newWordsArray;
                        wordsCount = newWordsCountArray;
                        pagesContent = newPagesArray;
                        goto addTheLastWord;
                    }
```

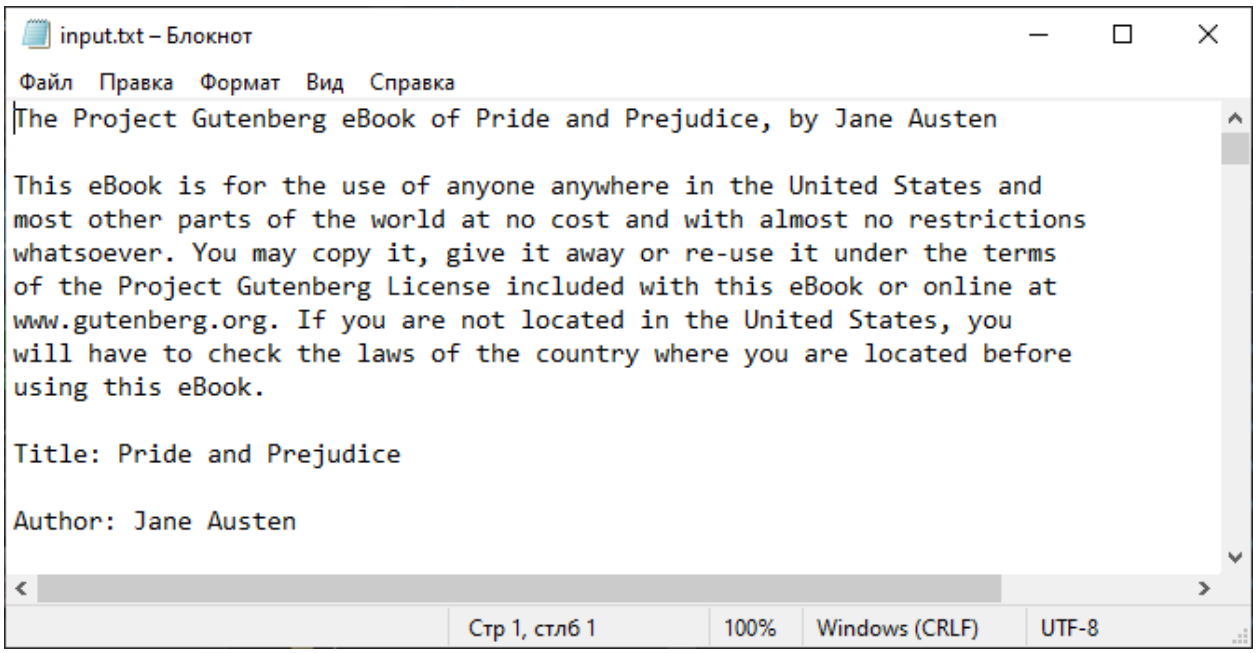
```
                    newWordsArray[currentWordIndex] =
wordsArray[currentWordIndex];
                    newWordsCountArray[currentWordIndex] =
wordsCount[currentWordIndex];
                    newPagesArray[currentWordIndex] =
pagesContent[currentWordIndex];
                    currentWordIndex++;
                    goto copyTheNextElement;
                }
            }
            addTheLastWord:
            {
                wordsArray[lastWordPointer] = currentWord;
                wordsCount[lastWordPointer] = 1;
                pagesContent[lastWordPointer] = new int[1];
                pagesContent[lastWordPointer][0] = currentPage;
                lastWordPointer++;
                currentWord = "";
            }
        }

        nextChar:
        {
            i = i + 1;
            if (i >= currentString.Length)
            {
                goto readNextLine;
            }
            goto iteratingInString;
        }
    }

    readNextLine:
    if (!streamReader.EndOfStream)
    {
        goto readingFile;
    }
    stopReading:
    {
        streamReader.Close();
    }
}

//=====================
//Bubble sort
//=====================
i = lastWordPointer - 1;
outerLoop:
{
    j = 0;
    if (i >= 1)
```

```
    {
        innerLoop:
        {
            if (j < i)
            {
                int currentCharIndex = 0;
                checkTheNextChar:
                {
                    if (wordsArray[j][currentCharIndex] ==
wordsArray[j+1][currentCharIndex]
                        && currentCharIndex+1 != wordsArray[j].Length &&
currentCharIndex+1 != wordsArray[j+1].Length)
                    {
                        currentCharIndex += 1;
                        goto checkTheNextChar;
                    }
                }

                if (currentCharIndex+1 == wordsArray[j].Length &&
currentCharIndex+1 != wordsArray[j+1].Length
                    && wordsArray[j][currentCharIndex] ==
wordsArray[j+1][currentCharIndex])
                {
                    goto compareNextWord;
                } else if (currentCharIndex+1 == wordsArray[j+1].Length &&
currentCharIndex+1 != wordsArray[j].Length
                            && wordsArray[j][currentCharIndex] ==
wordsArray[j+1][currentCharIndex])
                {
                    goto changeWords;
                }

                if (wordsArray[j][currentCharIndex] >
wordsArray[j+1][currentCharIndex])
                {
                    goto changeWords;
                }
                else
                {
                    goto compareNextWord;
                }
                changeWords:
                {
                    int tempCount = wordsCount[j];
                    string tempWord = wordsArray[j];
                    var tempPages = pagesContent[j];
                    wordsCount[j] = wordsCount[j + 1];
                    wordsArray[j] = wordsArray[j + 1];
                    pagesContent[j] = pagesContent[j + 1];
                    wordsCount[j + 1] = tempCount;
                    wordsArray[j + 1] = tempWord;
```

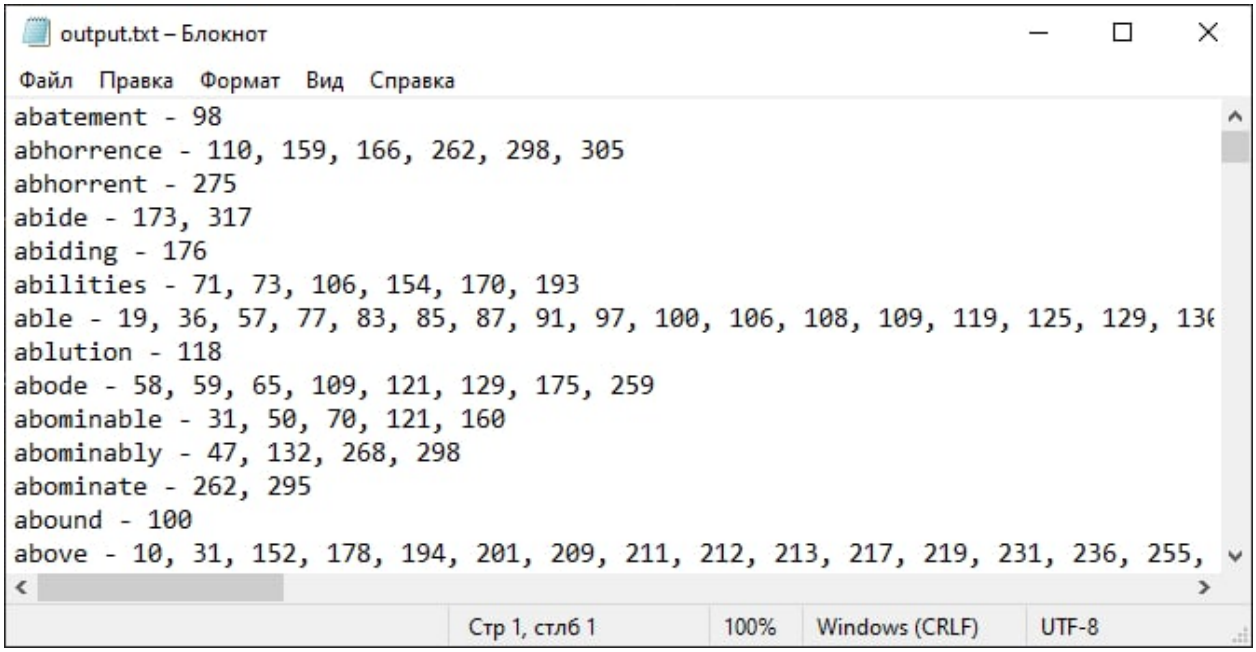
```
                    pagesContent[j + 1] = tempPages;
                }
                compareNextWord:
                j = j + 1;
                goto innerLoop;
            }
        }
        i = i - 1;
        goto outerLoop;
    }
}

//=====================
//Writing to the file
//=====================
StreamWriter streamWriter = new StreamWriter(outputFilePath);
currentWordIndex = 0;
writeTheNextWord:
{
    if (wordsCount[currentWordIndex] < amountToIgnore) { // we should write this
word to the file if its amount does not exceed {amountToIgnore}
        currentPage = 0;
        outPages:
        {
            if (currentPage == 0)
            {
                streamWriter.Write($"{wordsArray[currentWordIndex]} -
{pagesContent[currentWordIndex][currentPage]}"); // writing the word and first
page
            }
            currentPage = currentPage + 1;
            if (wordsCount[currentWordIndex] == currentPage)
            {
                streamWriter.WriteLine();
                goto endOutPages;
            }
            if (pagesContent[currentWordIndex][currentPage] !=
pagesContent[currentWordIndex][currentPage - 1]) // do not duplicate the same
page
            {
                streamWriter.Write($",
{pagesContent[currentWordIndex][currentPage]}");
            }
            goto outPages;
        }
    }
    endOutPages:
    {
        currentWordIndex = currentWordIndex + 1;
        if (currentWordIndex < lastWordPointer)
        {
```

```
            goto writeTheNextWord;
        }
    }
}

streamWriter.Close();
```

Приклад роботи:



Зміст файлу «input.txt»



Зміст файлу «output.txt»

Висновки

В результаті виконання даної лабораторної роботи я реалізував розв’язання 2 задач (term frequency та словникове індексування) за допомогою мови C# та конструкцій GOTO (динамічні структури, цикли та функції за виключенням зчитування з файлу використані не були). Результати були порівняні із даними у прикладі. Результати першої програми співпадають, отже вона вирішена правильно. Результати другої дещо різняться, оскільки в якості вхідного файлу були використані трохи різні дані. Також зрозумів, як писали код наші славні пращури у 1950-х.