**Курсовая работа на тему «Самая длинная общая подпоследовательность»   
по дисциплине  
 «Алгоритмы и структуры данных»**

|  |
| --- |
| Выполнил  студент гр. 3530904/90002 |
| Руководитель |

|  |
| --- |
| М.С.Драцкая |
| Е.А. Павлов |

Оглавление

[1. Постановка задачи 3](#_Toc39032904)

[2. Описание алгоритма решения и пример выполнения 3](#_Toc39032905)

[2.1. Нахождение самой длинной общей подпоследовательности двух последовательностей 3](#_Toc39032906)

[2.2. Пример выполнения 5](#_Toc39032907)

[3. Описание используемых структур данных 8](#_Toc39032908)

[3.1. Класс String 8](#_Toc39032909)

[3.2. Класс Matrix (шаблон) 9](#_Toc39032910)

[3.3. Базовый класс Exception 10](#_Toc39032912)

[3.4. Класс FileIsNotOpenException 10](#_Toc39032913)

[3.5. Класс FileWithEmptyStringException 11](#_Toc39032914)

[3.6. Класс IndexOutOfRangeException 11](#_Toc39032915)

[4.Анализ алгоритма (вычислительная сложность) 11](#_Toc39032916)

[5. Описание программы 11](#_Toc39032917)

[5.1. Структура программы 11](#_Toc39032918)

[5.2. Форматы входных и выходных данных 12](#_Toc39032919)

[6. Описание спецификации программы (детальные требования) 13](#_Toc39032920)

[6.1 Спецификация программы 13](#_Toc39032921)

[6.2. Детальные требования (данные для чтения из файлов) 15](#_Toc39032927)

# 1. Постановка задачи

1. В соответствии с выбранным вариантом (3.5) разработать алгоритм нахождения самой длинной общей подпоследовательности двух последовательностей.

2. Для разработки алгоритма создать следующие структуры данных:

* Класс String – объекты этого класса будут являться последовательностями, которые будут задействованы в работе алгоритма.
* Класс Matrix (шаблон)– объект этого класса (двумерный массив) будет использован в реализации алгоритма.
* Базовый класс Exception и производные классы исключений для обнаружения ошибок в программе.

3. Провести тестирование алгоритмов и отразить результаты в отчёте.

# 2. Описание алгоритма решения и пример выполнения

## 2.1. Нахождение самой длинной общей подпоследовательности двух последовательностей

Задача нахождения наибольшей общей подпоследовательности ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *longest common subsequence*, LCS) — задача поиска [последовательности](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C), которая является подпоследовательностью двух последовательностей.   
 *Пусть имеются последовательности X=⟨,,…,⟩ и Y=⟨,,…,⟩. Необходимо найти LCS(X, Y).*

**Что такое подпоследовательность?**

Последовательность  Z=⟨,,…,⟩ является **подпоследовательностью** последовательности X=⟨,,…,⟩, если существует строго возрастающая последовательность ⟨,,…,⟩ индексов X таких, что для всех j=1,2,…,k выполняется соотношение  =.

Другими словами, подпоследовательность данной последовательности — это последовательность, из которой удалили ноль или больше элементов. Подпоследовательность также может быть и пустой (если удалены все элементы исходной последовательности). Например, Z=⟨B,C,D,B⟩ является подпоследовательностью последовательности X=⟨A,B,C,B,D,A,B⟩, а соответствующая последовательность индексов имеет вид ⟨2,3,5,7⟩.

**Что такое общая подпоследовательность?**

Последовательность Z является **общей подпоследовательностью** последовательностей X и Y, если Z является подпоследовательностью как X, так и Y.

**Решение**

Данная задача решается с применением принципов *динамического программирования (метод решения задачи путём её разбиения на несколько одинаковых подзадач, рекуррентно связанных между собой)*

Нахождение длины наибольшей общей подпоследовательности

Рассматриваются две строки (или числовые последовательности) — X и Y. Пусть первая строка состоит из n символов ,,…,, вторая строка состоит из m символов ,,…,.

Рассматриваются последние символы данных строк  и .

* Если эти символы совпадают, то они обязательно войдут последними символами и в наибольшую общую подпоследовательность данных строк. Тогда можно свести задачу нахождения наибольшей общей подпоследовательности для строк X =,,…, и Y =,,…, к задаче нахождения наибольшей общей подпоследовательности для строк, полученных отбрасыванием от данных строк последнего символа, то есть для ,,…, и ,,…,. Затем к ответу для «укороченных» строк добавим последние (равные) символы исходных строк ( или ) и получим ответ для исходных строк.
* Если же последние символы исходных строк не совпадают, то эти символы ( и ) не могут одновременно входить в наибольшую общую подпоследовательность, поэтому можно один из них отбросить. Тогда задача сводится к нахождению наибольшей общей подпоследовательности для одного из двух случаев - для строк ,,…, и ,,…, или для строк ,,…, и ,,…,.

Задача сведена к меньшей задаче - нахождения наибольшей общей подпоследовательности для строк, полученных отбрасыванием последних символов от исходных строк, то есть для префиксов исходных строк.

- LCS префиксов данных последовательностей, заканчивающихся элементами с номерами и соответственно. Получается следующее рекуррентное соотношение:

Далее необходимо завести двумерный массив размером (n+1)×(m+1) и заполнить его значениями по указанным рекуррентным соотношениям. Сначала весь массив заполняется нулями, затем двумя вложенными циклами по  и по  заполняется оставшаяся часть массива.

Нахождение самой наибольшей общей подпоследовательности

Для нахождения самой общей подпоследовательности необходимо восстановить ответ. Для этого выполняется «обратный проход» по массиву  , начиная с последнего элемента. В каждой рассматриваемой ячейке  выясняется, как было получено значение в этой ячейке. Это зависит от последних символов рассматриваемых префиксов.

* Если , то тогда ответ для элемента  получен из  добавлением 1, поэтому осуществляется переход к элементу , а к ответу добавляется символ =.
* Если  нужно перейти к тому элементу  или , значение в котором совпадает со значением .

## 2.2. Пример выполнения

Имеются две строки **s1=abcdaf** и **s2=acbcf**. Таким образом, самая длинная общая подпоследовательность этих двух строк равна **«abcf»**, она имеет длину 4.   
Создаётся таблица **F** (двумерный массив), содержащая все символы **s1** в строке и все символы **s2** в столбце. Таблица будет выглядеть так:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  |  |  | a | b | c | d | a | f |
| 0 |  |  |  |  |  |  |  |  |
| 1 | a |  |  |  |  |  |  |  |
| 2 | c |  |  |  |  |  |  |  |
| 3 | b |  |  |  |  |  |  |  |
| 4 | c |  |  |  |  |  |  |  |
| 5 | f |  |  |  |  |  |  |  |

0-й столбец представляет собой пустую подпоследовательность **s1**. Точно так же 0-я строка представляет собой пустую подпоследовательность **s2**. Если взять пустую подпоследовательность строки и попытаться сопоставить ее с другой строкой, то общая подпоследовательность будет иметь длину 0. Таким образом, 0-й ряд и 0-й столбцы заполняются нулями. Получается:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  |  |  | a | b | c | d | a | f |
| 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | a | 0 |  |  |  |  |  |  |
| 2 | c | 0 |  |  |  |  |  |  |
| 3 | b | 0 |  |  |  |  |  |  |
| 4 | c | 0 |  |  |  |  |  |  |
| 5 | f | 0 |  |  |  |  |  |  |

Заполняется **F [1] [1]**. Имеется строка **a** и другая строка **a**, тогда длина LCS будет равна 1. Теперь рассматривается **F [1] [2]**. У нас есть строка **ab** и строка **a**. Длина LCS будет равна 1. Остальные значения будут также равны 1 для первой строки, так как рассматриваются только строка **a** со строками **abc, abcd**, **abcda**, **abcdaf**. Таблица будет выглядеть так:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  |  |  | a | b | c | d | a | f |
| 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | a | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | c | 0 |  |  |  |  |  |  |
| 3 | b | 0 |  |  |  |  |  |  |
| 4 | c | 0 |  |  |  |  |  |  |
| 5 | f | 0 |  |  |  |  |  |  |

При заполнении **F [2] [1]** имеется строка **ac** с одной стороны, а **с** другой стороны. Таким образом, длина LCS равна 1. По существующей закономерности: если **s1 [2]** и **s2 [1]** не совпадают, то длина LCS будет максимальной длиной LCS **сверху** или **слева**. Мы получаем:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  |  |  | a | b | c | d | a | f |
| 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | a | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | c | 0 | 1 |  |  |  |  |  |
| 3 | b | 0 |  |  |  |  |  |  |
| 4 | c | 0 |  |  |  |  |  |  |
| 5 | f | 0 |  |  |  |  |  |  |

Для **F [2] [2]** имеются строки **ab** и **ac** . Поскольку **c** и **b** не являются одинаковыми, в ячейку помещается максимум сверху или слева. В этом случае это 1. После этого для **F [2] [3]** имеются строки **abc** и **ac** . На этот раз текущие значения обеих строк и столбцов совпадают. Теперь длина LCS будет равна максимальной длине LCS + 1. Мы получаем:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  |  |  | a | b | c | d | a | f |
| 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | a | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | c | 0 | 1 | 1 | 2 |  |  |  |
| 3 | b | 0 |  |  |  |  |  |  |
| 4 | c | 0 |  |  |  |  |  |  |
| 5 | f | 0 |  |  |  |  |  |  |

Дальше поиск длины LCS происходит аналогичным образом. После заполнения таблица будет выглядеть так:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  |  |  | a | b | c | d | a | f |
| 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | a | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | c | 0 | 1 | 1 | 2 | 2 | 2 | 2 |
| 3 | b | 0 | 1 | 2 | 2 | 2 | 2 | 2 |
| 4 | c | 0 | 1 | 2 | 3 | 3 | 3 | 3 |
| 5 | f | 0 | 1 | 2 | 3 | 3 | 3 | 4 |

Длина LCS между **s1** и **s2** будет равна **F [5] [6] = 4**. Здесь 5 и 6 - длина **s2** и **s1** соответственно.

Для того, чтобы восстановить LCS, выполняется «обратный проход» по массиву **F,** начиная с последнего элемента. В каждой рассматриваемой ячейке **F** выясняется, как было получено значение в этой ячейке. Получаем LCS = **«abcf».**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  |  |  | a | b | c | d | a | f |
| 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | a | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | c | 0 | 1 | 1 | 2 | 2 | 2 | 2 |
| 3 | b | 0 | 1 | 2 | 2 | 2 | 2 | 2 |
| 4 | c | 0 | 1 | 2 | 3 | 3 | 3 | 3 |
| 5 | f | 0 | 1 | 2 | 3 | 3 | 3 | 4 |

# 3. Описание используемых структур данных

## 3.1. Класс String

Назначение: объекты класса String будет являться последовательностями, самую длинную общую подпоследовательность которых необходимо найти.

class String

{

public:

//конструктор без парметров (итог: пустая строка, длина которой равна 0)

String();

//конструктор, который в качестве параметра принимает размер строки (итог: пустая строка указанного размера)

String(int newLength);

//конструктор, который в качестве параметра принимает указатель на массив символов (итог: строка с символами)

String(const char \* newString);

//конструктор копирования

String(const String & otherString);

//деструктор

~String();

//возвращает длину строки

int getLength() const;

//возвращает указатель на массив символов строки

char \* getOurString() const;

//оператор индексирования: возвращает символ с указанным индексом

char & operator [] (int index) const;

//оператор присваивания

String & operator = (const String & otherString);

//оператор сравнения строк (равны ли они)

bool const operator == (const String & otherString) const;

//добавляет символ в конец строки и увеличивает её размер (используется в функции поиска LCS)

void pushBack(const char &newElement);

//переставляет элементы массива в обратном порядке (используется в функции поиска LCS)

void reverse();

//дружественная функция: оператор вывода строки

friend std::ostream & operator << (std::ostream & out, const String & string);

//дружественная функция: оператор ввода строки

friend std::istream & operator >> (std::istream & in, String & string);

private:

//указатель на массив символов строки

char \*ourString\_;

//длина строки

int length\_;

};

## 3.2. Класс Matrix (шаблон)

Назначение: объект класса Matrix будет использован в функции поиска LCS (в матрице храним количество совпадений элементов).

template<typename T>

class Matrix

{

public:

//конструктор без парметров (итог: пустая матрица, размер которой 0\*0)

Matrix();

//конструктор, который в качестве параметра принимает количество строк и слобцов (итог: пустая строка указанного размера)

Matrix(int newRows, int newColumns);

//конструктор копирования

Matrix(const Matrix & otherMatrix);

//деструктор

~Matrix();

//возвращает количество строк матрицы

int getRows() const;

//возвращает количество столбцов матрицы

int getColumns() const;

//возвращает указатель на матрицу

T \* getMatrix() const;

//оператор индексирования: возвращает строку с указанным индексом

T \* operator [] (int rowIndex);

//оператор присваивания

Matrix & operator = (const Matrix & otherMatrix);

//заполняет всю матрицу указанным значением (используется в функции поиска LCS)

void fillWholeMatrix(T value);

//дружественная функция: оператор вывода матрицы

template<typename T>

friend std::ostream & operator << (std::ostream & cout, const Matrix<T> & matrix);

//дружественная функция: оператор ввода матрицы

template<typename T>

friend std::istream & operator >> (std::istream & cin, Matrix<T> & matrix);

private:

//указатель на матрицу

T \*\* ourMatrix\_;

//количество строк матрицы

int rows\_;

//количество столбцов матрицы

int columns\_;

# };

## 3.3. Базовый класс Exception

Назначение: является базовым классом для классов, содержащих сообщение о конкретной ошибке.

// базовый класс исключений

class Exception

{

public:

//конструктор, который в качестве параметра принимает указатель на массив символов (итог: строка с символами)

Exception(const char \* message);

//возвращает сообщение об ошибки

String getMessage() const;

private:

//строка, содержащая сообщение об ошибке

String message\_;

};

## 3.4. Класс FileIsNotOpenException

Назначение: содержит сообщение об ошибке (открыт ли файл для чтения).

//производный класс исключений (открыт ли файл для чтения)

class FileIsNotOpenException : public Exception

{

public:

//конструктор без параметров

FileIsNotOpenException();

};

## 3.5. Класс FileWithEmptyStringException

Назначение: содержит сообщение об ошибке (не содержит ли файл для чтения пустые строки).

//производный класс исключений (не содержит ли файл для чтения пустые строки)

class FileWithEmptyStringException : public Exception

{

public:

//конструктор без параметров

FileWithEmptyStringException();

};

## 3.6. Класс IndexOutOfRangeException

Назначение: содержит сообщение об ошибке (не выходит ли индекс за границы допустимого диапазона).

//производный класс исключений (не выходит ли индекс за границы допустимого диапазона)

class IndexOutOfRangeException : public Exception

{

public:

//конструктор без параметров

IndexOutOfRangeException();

};

# 4.Анализ алгоритма (вычислительная сложность)

Сложность алгоритма (относится и ко времени выполнения, и к требованиям к памяти) составит O(n\*m), где n и m – длины последовательностей.  
Стоит заметить, что если сравнивать, например, два файла из 100 000 строк, то нужно будет хранить в памяти матрицу из 1010 элементов. Таким образом, главный недостаток алгоритма – возможное переполнение памяти.

# 5. Описание программы

## 5.1. Структура программы

Программа содержит следующие файлы:

* functions.h – файл с прототипами функций://выводит длину LCS, возвращает LCS

String LCS(String const & str1, String const & str2);

//выводит матрицу в необходимом для нас виде  
void outputMatrix(String const & str1, String const & str2, Matrix<int> matrix);

* matrix.h – описание и реализация класса Matrix
* string.h – описание класса String
* exceptions.h – реализация классов исключений

Также программа содержит соответствующие файлы реализации:

* functions.cpp
* string.cpp
* exceptions.cpp

Также программа содержит файл main.cpp, в котором происходит проверка работы программы.

Программа предполагает чтение 2-х последовательностей из файлов string1.txt и string2.txt.   
Во время работы программы осуществляются проверки на наличие исключений   
(см. 6.2).   
Если проверки выполнена успешно, последовательности записываются в переменные типа String.  
После этого для каждой пары последовательностей вызывается функция, в которую передаётся эта пара, а также её номер:  
void serchOfLCS(String str1, String str2, int number)  
Данная функция записывает в файл output.txt номер пары, саму пару последовательностей, а также вызывает функция LCS(str1, str2), которая осуществляет поиск LCS.

## 5.2. Форматы входных и выходных данных

Входные данные:  
Программа предполагает чтение 2-х последовательностей из файлов string1.txt и string2.txt.  
В каждом файле записаны несколько строк типа String (их количество неизвестно), которые при чтении будут записаны в следующие переменные:  
 String string1;

String string2;

Чтение происходит до конца файлов в цикле while (!(file1.eof()) && !(file2.eof())).

Выходные данные:  
Для каждой пары строк будет выполнена проверка правильности работы необходимых функций.   
Результат будет записан в файл output.txt

Для каждой пары будут записаны:

* Номер пары в формате:  
  -----STRINGS NUMBER *1*-----  
  (запись из функции serchOfLCS)
* Сами последовательности   
  (запись из функции serchOfLCS)  
    
  ЕСЛИ ЕСТЬ LCS
* Длина LCS в формате:  
  Length of common subsequence:

*4*(запись из функции LCS)

* Матрица, с помощью которой осуществлялся поиск LCS (но только в случае, если произведение строк и столбцов матрицы меньше 300 – это сделано для того, чтобы не загромождать пространство в файле). Формат:

MATRIX:

a b c d a f

0 0 0 0 0 0 0

a 0 1 1 1 1 1 1

c 0 1 1 2 2 2 2

b 0 1 2 2 2 2 2

c 0 1 2 3 3 3 3

f 0 1 2 3 3 3 4  
(запись из функции LCS)  
Если матрица содержит 300 или более элементов, то записать сообщение:  
MATRIX TOO LARGE FOR OUTPUT!  
(запись из функции LCS)

* Сама LSC в формате:

ANSWER:

аbcf  
(запись из функции serchOfLCS)

ЕСЛИ НЕТ LCS

* Будет записано сообщение:  
  There is not common subsequence!  
  (запись из функции serchOfLCS)

# 6. **Описание спецификации программы (детальные требования)**

## **6.1 Спецификация программы**

1. Реализован класс String, объекты которого являются последовательностями, LCS которых необходимо найти.   
В данной программе: последовательностью считается набор символов, содержащий любые символы (например, пробелы, служебные символы, буквы английского, русского алфавитов и т.д.). При этом последовательностью не может являться пустая строка.

2. Реализован класс Matrix, объект которого используется при нахождении LCS.  
В данной программе: объекты данного класса представляют собой двумерные массивы, основная функция которых – хранение информации.

3. Реализован базовый класс Exception, а также производные классы:

* FileIsNotOpenException
* FileWithEmptyStringException
* IndexOutOfRangeException

Объекты данных класса – строки (класс String), которые содержат информацию о соответствующих ошибках.

4. Реализованы следующие функции:

* LCS – функция, возвращающая наибольшую общую подпоследовательность, а также записывает в файл output.txt длину этой подпоследовательности и матрицу, в которой хранилась информация и последовательности.
* outputMatrix – функция, которая записывает в файл output.txt матрицу (двумерный массив) в удобном виде. В данном случае выводится не только сам массив в виде таблицы, но и последовательности, с которым работала функция.
* serchOfLCS – функция, реализованная в файле main.cpp, которая записывает в файл output.txt номер пары последовательностей, сами последовательности, а также вызывает функцию LCS

5. Чтение последовательностей происходит из файлов string1.txt и string2.txt. Запись происходит в файл output.txt.

6. При чтении из файлов реализованы необходимые проверки:

* открыты ли файлы (см. 6.2.1). В случае ошибки сообщение должно быть выведено в консоль.
* не содержат ли файлы пустую строку (см. 6.2.2). В случае ошибки сообщение должно быть записано в файл output.txt.
* не происходит ли во время работы программы выход индекса за пределы допустимого диапазона (см. 6.2.3). Данная проверка необходима для оператора индексирования ([]), который реализован как в классе String, так и в классе Matrix. Оператор индексирования используется в функции LCS. В случае ошибки сообщение должно быть выведено в консоль.   
    
  Во всех случаях программа прекращает свою работу.

Чтение происходит до конца файлов (до символа конца файла). Предусмотрена ситуация, при которой в одном файле может содержаться меньше строк, чем в другом (см. 6.2.4).

7. Для каждой пары последовательностей происходит поиск LCS, результаты которого записываются в файл output.txt.

8. После выполнения основной части программы происходит закрытие всех использованных файлов (string1.txt, string2.txt, output.txt).

## 6.2. Детальные требования (данные для чтения из файлов)

1. Инициировать проверку на то, открыты файл для чтения или нет. Использовать класс FileIsNotOpenException  
В случае, если файлы не открыты, вывести сообщение об ошибке "Error! Input file is not open."   
в консоль.   
В случае успешного прохождения данной проверки вывести сообщение "Files are open!" в консоль.   
Функционал обнаружения данной ошибки реализован в файле main.cpp.

2. Инициировать проверку пустых строк в файле. Использовать класс FileWithEmptyStringException  
Согласно спецификации, входные файлы string1.txt и string2.txt не могут содержать пустых строк. В случае их обнаружения программа должна завершить работу и в файл output.txt записать сообщение об ошибке: "Error! Input file contains an empty string.". Функционал обнаружения данной ошибки реализован в дружественной функции ввода строки в классе String.

3. Инициировать проверку на предмет того, не происходит ли во время работы программы выход индекса за пределы допустимого диапазона. Использовать класс IndexOutOfRangeException. В случае, если индекс выходит за пределы допустимого диапазона, вывести сообщение об ошибке "Error! Index out of range." в консоль.   
Функционал обнаружения данной ошибке содержится в реализации оператора индексирования классов String и Matrix.

4. Предусмотреть ситуацию, при которой в файлах string1.txt и string2.txt содержится различное количество строк.   
В такой ситуации из обоих файлов считывается количество строк, содержащихся в файле с наименьшим количеством строк. Данный функционал реализован с помощью цикла while (!(file1.eof()) && !(file2.eof())), чтение в котором происходит только в том случае, если оба файла ещё не кончились.

5. Реализовать проверку функций для всех пар строк (с учётом требований в предыдущих пунктах).