МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Математический факультет

Кафедра вычислительной математики

**КУРСОВАЯ РАБОТА**

О числе золотого сечения

Выполнил студент Русин Владислав Сергеевич

Группы МП-203

очной формы обучения

направления подготовки (специальности)

01.03.02 Прикладная математика и информатика \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022г.

|  |  |
| --- | --- |
|  | Научный руководитель  Овчинников Михаил Алексеевич  Должность доцент  Ученая степень канд. физ.-мат. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022г. |

Челябинск

2022

Содержание

[1. Число золотого сечения 3](#_Toc74082996)

[1.1. Некоторые свойства числа золотого сечения 3](#_Toc74082997)

[1.2. Арифметика чисел, порождаемых числом золотого сечения 4](#_Toc74082998)

[2. Задача о матрицах 5](#_Toc74083002)

[2.1. Условия задачи 5](#_Toc74083003)

[2.2. Программа, реализующая решение поставленной задачи 6](#_Toc74083004)

[2.2.1. Внешний вид вывода программы 6](#_Toc74083009)

[2.2.2. Частичный разбор кода программы 7](#_Toc74083009)

[2.3. Анализ полученных результатов 11](#_Toc74083005)

[Список литературы 12](#_Toc74083017)

[Приложение 13](#_Toc74083018)

1. **Число золотого сечения**

Золотое сечение – это такое соотношение двух величин, при котором отношение большей величины к меньшей равно отношению суммы этих величин к большей, т. е. . Принимая за неизвестное , получим уравнение или

(1)

Числом золотого сечения называется иррациональное число , являющееся положительным корнем уравнения (1), приблизительно равное 1,618.

* 1. **Некоторые свойства числа золотого сечения**
* Из квадратного уравнения (1) следует, что или

(2)

* Число представимо в виде цепной дроби:

(3)

Это единственная цепная дробь, содержащая только единицы. Она получается записью квадратного уравнения (1) в виде  с последующей заменой в правой части уравнения на его значение . Таким образом, получим . Продолжая применять к правой части уравнения замену , получим бесконечную цепную дробь (3).

* Число представимо в виде бесконечной цепочки квадратных уравнений:

(4)

Эта запись получается при поиске квадратного корня уравнения (1) с последующей заменой в правой части уравнения на его значение (т. е. аналогично способу, используемому для представления в виде бесконечной цепной дроби (3)):

* 1. **Арифметика чисел, порождаемых числом золотого сечения**

Самой известной последовательностью чисел, порождаемых числом , являются числа Фибоначчи (обозначим эту последовательность как ), названные так в честь математика XII–XIII веков Леонардо Пизанского (известного как Фибоначчи). Числа в последовательности образуются по следующему правилу: пусть , и каждое последующее число равно сумме двух предшествующих ему чисел, т. е. , где   
.

Первые члены последовательности Фибоначчи: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Интересно вот что: отношение любого члена последовательности к члену является приближённым значением числа , причём тем приближение будет точнее, чем больше будет n. Так, , ,   
, , … Таким образом, .

1. **Задача о матрицах**

Рассматриваемая в данной работе задача состоит в том, чтобы выяснить, по какому принципу устроено множество матриц, получаемых в результате перемножений исходных матриц, которые были представлены в качестве примера ТКТП (топологическая квантовая теория поля) британским математиком Майклом Атья.

* 1. **Условия задачи**

Дана группа из 20 пар матриц порядка 3 (будем называть эти пары «базовыми»), разбитых на две подгруппы: пары, пронумерованные как , – , , и пары, пронумерованные , – , . Также дана формула, по которой эти пары перемножаются:

(5)

здесь и представляют пары с индексом , и представляют пары с индексом , где . Из свойства (2) числа золотого сечения следует:

(6)

В результате перемножения пар по этой формуле получается новая пара матриц, ( задаёт первую матрицу, – вторую.

Примечательно то, что при перемножении по формуле двух пар одной подгруппы с индексами результатом будет являться пара матриц той же подгруппы с индексом ( – взятие остатка от деления). Например, при перемножении по формуле (5) пар , и , получим пару , , а при перемножении пар , и , получим , .

Отметим, что матрицы в парах с индексом 0 являются по сути единичными матрицами, и результатом умножения на них пары с индексом будет та же пара с индексом , поэтому для дальнейших вычислений пары с индексом 0 применяться не будут.

Обозначим результат перемножения по формуле (5) пар , и , как  
. Назовём это «цепочками» длины 2. Тогда результатом перемножения и , будет . Это цепочки длины 3. Таким образом, будем составлять цепочки длины n, причём элементами цепочки на нечётных позициях будут базовые пары , , а на чётных - базовые пары , .

Вопрос задачи: как устроено множество пар матриц, получаемых при составлении цепочек длины n?

* 1. **Программа, реализующая решение поставленной задачи**

Нужно сказать, что в данной работе используются не совсем те базовые пары матриц, которые были представлены Майклом Атья. Дело в том, что "настоящие" матрицы содержат в себе элементы - полуцелые числа, из-за чего результаты вычислений получаются более громоздкими. Чтобы решить эту проблему, в работе используются удвоенные "настоящие" матрицы, содержащие в себе только целые числа. Однако в результате перемножений по формуле (5) такие удвоенные "настоящие" удваиваются снова, становясь уже учетверёнными, поэтому все элементы матриц в парах, получаемых по формуле, должны быть поделены на 2, чтобы они снова стали удвоенными.

**2.2.1. Внешний вид вывода программы**

Результатом работы программы являются два текстовых файла. В одном содержатся пары матриц, полученные по формуле, в виде:

*Название пары :*

*{*

*[Строка 1 матрицы 1] [Строка 1 матрицы 2]*

*[Строка 2 матрицы 1] [Строка 2 матрицы 2]*

*[Строка 3 матрицы 1] [Строка 3 матрицы 2]*

*};*

Элементы в строках разделяются запятыми (название файла: matrices.txt). В другом файле – записи об одинаковых парах в виде:

*Название пары 1 - копия Название пары 2*

(название файла: knownDuplicates.txt).

Программа вычисляет пары матриц и ищет среди них совпадающие во всех цепочках длины 2–5, т. е. всего в файл matrices.txt будет занесено   
 записей о парах матриц.

**2.2.2. Частичный разбор кода программы**

В программе постоянно повторяются следующие три шага:

1. вычисление очередной пары матриц по формуле (5);
2. запись полученной пары в файл matrices.txt;
3. поиск среди предыдущих вычисленных пар такой, которая будет одинаковой с последней. При нахождении одной такой пары поиск прекращается, а в файл knownDuplicates.txt заносится запись о найденном совпадении.

Для реализации этих шагов в программе есть три соответствующие функции:

1. calculate\_by\_formula;
2. print\_to\_file;
3. search\_for\_duplicate.

Рассмотрим эти функции подробнее, начиная с calculate\_by\_formula:

* Возвращаемый тип: ***void***
* Список параметров: \*a, \*b, \*c, \*d (указатели типа ***const*** vector<vector<**int**>>\*), \*result1, \*result2 (указатели типа vector<vector<**int**>>\*)

Тип vector<vector<**int**>> представляет матрицу. \*a, \*b, \*c, \*d – указатели на матрицы, \*a, \*b – первая пара, \*c, \*d – вторая пара.

Функция представляет собой реализацию формулы (6). Сначала сохраняются результаты обычного матричного перемножения нужных матриц:

vector<vector<int>> ac = multiply(a, c);

vector<vector<int>> ad = multiply(a, d);

vector<vector<int>> bc = multiply(b, c);

vector<vector<int>> bd = multiply(b, d);

Затем составляются необходимые суммы матриц, которые записываются в \*result1 и \*result2:

\*result1 = add(&ac, &ad, &bc);

\*result2 = add(&ac, &bd);

Как было замечено ранее, по формуле получаются удвоенные матрицы, поэтому не забываем делить результаты на 2:

divide\_by\_2(result1);

divide\_by\_2(result2);

Полученная в результате вычислений пара матриц записана в   
\*result1, \*result2, причём т. к. это указатели, то результат вычислений доступен по адресам, на которые эти указатели указывают, после завершения работы функции, что позволяет передать получившиеся матрицы назад, в функцию, из которой вызывалась calculate\_by\_formula, и оттуда вызвать следующую функцию - print\_to\_file:

* Возвращаемый тип: ***void***
* Список параметров: fstream \*matrices, string pairHeader, \*res1, \*res2 (указатели типа **const** vector<vector<**int**>>\*)

\*matrices - ссылка на поток для чтения/редактирования файла matrices.txt; pairHeader - название, которым будет обозначена пара, записываемая в файл; \*res1, \*res2 - указатели на пару матриц, которую необходимо записать в файл.

Прежде всего, конец файла устанавливается как позиция, с которой начинается запись в файл:

(\*matrices).clear();

(\*matrices).seekp(0, ios::end);

Затем в файл заносится запись, оформленная так, как было описано в   
пункте **2.2.1.**:

*(\*matrices) << multiplicationName << "****\n****{" << endl;*

***for*** *(****int*** *i = 0; i < N; i++)*

*{*

*(\*matrices) << "****\t****[ ";*

***for*** *(****int*** *j = 0; j < N; j++)*

*(\*matrices) << setw(numWidth) << (\*res1)[i][j] << (j < 2 ? ", " : " ]****\t****");*

*(\*matrices) << "****\t****[ ";*

***for*** *(****int*** *j = 0; j < N; j++)*

*(\*matrices) << setw(numWidth) << (\*res2)[i][j] << (j < 2 ? ", " : " ]****\n****");*

*}*

*(\*matrices) << "};" << endl << endl;*

Это конец функции print\_to\_file. Управление далее передаётся в функцию search\_for\_duplicate:

* Возвращаемый тип: **void**
* Список параметров: *string pairHeader, fstream \*matrices, ofstream \*knownDuplicates,* \*res1, \*res2 (указатели типа **const** vector<vector<**int**>>\*)

*\*knownDuplicates* – ссылка на поток для записи в файл knownDuplicates.txt; \*res1, \*res2, pairHeader и \*matrices те же, что и для функции print\_to\_file.

Сначала создаём переменные, в которые будем впоследствии записывать название и матрицы каждой пары, с которой будет сравниваться пара \*res1, \*res2:

*string comparedPairHeader;*

*vector<vector<****int****>> comparedM1(N, vector<****int****>(N));*

*vector<vector<****int****>> comparedM2(N, vector<****int****>(N));*

Создаём переменную типа ***bool***, в которую будет записываться результат сравнения с каждой отдельной парой:

***bool*** *isDuplicate;*

Устанавливаем начало файла matrices.txt в качестве точки, с которой начнётся считывание пар:

*(\*matrices).clear();*

*(\*matrices).seekg(0, ios::beg);*

В цикле ***while*** считываются пары матриц и проводится сравнение. Записываем название каждой пары в переменную comparedPairHeader. Поиск совпадений прекратится, если в comparedPairHeader будет записано название пары, для которой совпадения ищутся:

***while*** *(getline((\*matrices), comparedPairHeader) && comparedPairHeader != pairHeader)*

Очередной шаг цикла начинаем с установки значения isDuplicate (изначально считаем, что рассматриваемая на данном шаге цикла пара является идентичной паре, для которой таковые ищутся) и считывания пары из файла matrices.txt:

isDuplicate = true;

read\_pair(matrices, &comparedM1, &comparedM2);

Сравниваем пары и, если хотя бы один элемент с индексом i, j хотя бы одной матрицы из пары отличается от элемента с теми же индексами соответствующей матрицы другой пары, прекращаем сопоставление этих пар и переходим к следующему шагу цикла **while**:

***for*** *(****int*** *i = 0; i < N; i++)*

*{*

***for*** *(****int*** *j = 0; j < N; j++)*

*{*

***if*** *((\*res1)[i][j] != comparedM1[i][j] ||*

*(\*res2)[i][j] != comparedM2[i][j])*

*{*

*isDuplicate = false;*

***break****;*

*}*

*}*

***if*** *(!isDuplicate)*

***break****;*

*}*

Если же пара, рассматриваемая на текущем шаге цикла **while**, оказывается такой же, как пара с названием pairHeader, то в файле knownDuplicates.txt делается пометка об обнаруженном совпадении, и поиск прекращается:

***if*** *(isDuplicate)*

*{*

*(\*knownDuplicates) << pairHeader.substr(0, pairHeader.length() - 2) << " - копия "*

*<< comparedPairHeader.substr(0, comparedPairHeader.length() - 2) << endl*

*<< endl;*

***break****;*

*}*

На этом заканчивается работа функции search\_for\_duplicate. Дальше программа либо начнёт выполнять все три шага заново, либо, если последняя обработанная пара матриц называлась , завершится.

* 1. **Анализ полученных результатов**

После завершения работы программы файл knownDuplicates.txt содержит 43500 записей о совпадениях, т. е. из 66420 пар матриц 43500 являются копиями оставшихся 22920.

# **Список литературы**

1. Воробьев Н. Н. Числа Фибоначчи. - 5-е изд. - М.: Наука, 1984. - ([Популярные лекции по математике](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BF%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D0%B8_%D0%BF%D0%BE_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B5)) - 144 с.
2. Бендукидзе А. Д. Золотое сечение // Квант. - 1973. - №8. - С. 22–27.
3. Аракелян Г. Математика и История золотого сечения / Г.Аракелян. – М.: Логос, 2014. – 404 с
4. Золотое сечение // Википедия URL: https://ru.wikipedia.org/wiki/Золотое\_сечение (дата обращения: 18.05.2022).
5. Числа Фибоначчи // Википедия URL: https://ru.wikipedia.org/wiki/Числа\_Фибоначчи (дата обращения: 19.05.2022).

# **Приложение**

*#include <iostream>*

*#include <vector>*

*#include <fstream>*

*#include <iomanip>*

*#include <string>*

***using******namespace*** *std;*

***const******int*** *N = 3;*

***const******int*** *numWidth = 7;*

***const*** *vector<pair<vector<vector<****int****>>, vector<vector<****int****>>>> AB =*

*{*

*{*

*{{-1, 1, 1}, {-1, -1, -1}, {-2, 0, 0}}, // A1*

*{{1, -1, -1}, {-2, 2, 0}, {1, -1, 1}} // B1*

*},*

*{*

*{{-1, 1, 1}, {2, -2, -2}, {-1, 1, 1}}, // A2*

*{{0, -2, -2}, {-1, 3, 1}, {3, -3, -1}} // B2*

*},*

*{*

*{{1, 1, 1}, {2, -2, -2}, {-1, 3, 3}}, // A3*

*{{0, -2, -2}, {-1, 5, 3}, {3, -5, -3}} // B3*

*},*

*{*

*{{1, 1, 1}, {-1, -3, -3}, {-2, 4, 4}}, // A4*

*{{-1, -1, -1}, {-2, 6, 4}, {1, -7, -5}} // B4*

*},*

*{*

*{{0, 0, 0}, {0, -4, -4}, {0, 4, 4}}, // A5*

*{{-2, 0, 0}, {0, 6, 4}, {0, -8, -6}} // B5*

*},*

*{*

*{{1, -1, -1}, {1, -3, -3}, {2, 4, 4}}, // A6*

*{{-1, 1, 1}, {2, 6, 4}, {-1, -7, -5}} // B6*

*},*

*{*

*{{1, -1, -1}, {-2, -2, -2}, {1, 3, 3}}, // A7*

*{{0, 2, 2}, {1, 5, 3}, {-3, -5, -3}} // B7*

*},*

*{*

*{{-1, -1, -1}, {-2, -2, -2}, {1, 1, 1}}, // A8*

*{{0, 2, 2}, {1, 3, 1}, {-3, -3, -1}} // B8*

*},*

*{*

*{{-1, -1, -1}, {1, -1, -1}, {2, 0, 0}}, // A9*

*{{1, 1, 1}, {2, 2, 0}, {-1, -1, 1}} // B9*

*}/\*,*

*{*

*{{0, 0, 0}, {0, 0, 0}, {0, 0, 0}}, // A0*

*{{2, 0, 0}, {0, 2, 0}, {0, 0, 2}} // B0*

*}\*/*

*};*

***const*** *vector<pair<vector<vector<****int****>>, vector<vector<****int****>>>> FG =*

*{*

*{*

*{{-1, 1, -1}, {-1, -1, 1}, {2, 0, 0}}, // F1*

*{{1, -1, 1}, {-2, 2, 0}, {-1, 1, 1}} // G1*

*},*

*{*

*{{-1, 1, -1}, {2, -2, 2}, {1, -1, 1}}, // F2*

*{{0, -2, 2}, {-1, 3, -1}, {-3, 3, -1}} // G2*

*},*

*{*

*{{1, 1, -1}, {2, -2, 2}, {1, -3, 3}}, // F3*

*{{0, -2, 2}, {-1, 5, -3}, {-3, 5, -3}} // G3*

*},*

*{*

*{{1, 1, -1}, {-1, -3, 3}, {2, -4, 4}}, // F4*

*{{-1, -1, 1}, {-2, 6, -4}, {-1, 7, -5}} // G4*

*},*

*{*

*{{0, 0, 0}, {0, -4, 4}, {0, -4, 4}}, // F5*

*{{-2, 0, 0}, {0, 6, -4}, {0, 8, -6}} // G5*

*},*

*{*

*{{1, -1, 1}, {1, -3, 3}, {-2, -4, 4}}, // F6*

*{{-1, 1, -1}, {2, 6, -4}, {1, 7, -5}} // G6*

*},*

*{*

*{{1, -1, 1}, {-2, -2, 2}, {-1, -3, 3}}, // F7*

*{{0, 2, -2}, {1, 5, -3}, {3, 5, -3}} // G7*

*},*

*{*

*{{-1, -1, 1}, {-2, -2, 2}, {-1, -1, 1}}, // F8*

*{{0, 2, -2}, {1, 3, -1}, {3, 3, -1}} // G8*

*},*

*{*

*{{-1, -1, 1}, {1, -1, 1}, {-2, 0, 0}}, // F9*

*{{1, 1, -1}, {2, 2, 0}, {1, 1, 1}} // G9*

*}/\*,*

*{*

*{{0, 0, 0}, {0, 0, 0}, {0, 0, 0}}, // F0*

*{{2, 0, 0}, {0, 2, 0}, {0, 0, 2}} // G0*

*}\*/*

*};*

*vector<vector<****int****>> multiply(****const*** *vector<vector<****int****>> \*a,* ***const*** *vector<vector<****int****>> \*b)*

*{*

*vector<vector<****int****>> result(N, vector<****int****>(N, 0));*

***for*** *(****int*** *i = 0; i < N; i++)*

*{*

***for*** *(****int*** *j = 0; j < N; j++)*

*{*

***int*** *sum = 0;*

***for*** *(****int*** *k = 0; k < N; k++)*

*sum += (\*a)[i][k] \* (\*b)[k][j];*

*result[i][j] = sum;*

*}*

*}*

***return*** *result;*

*}*

*vector<vector<****int****>> add(****const*** *vector<vector<****int****>> \*a,* ***const*** *vector<vector<****int****>> \*b)*

*{*

*vector<vector<****int****>> result(N, vector<****int****>(N, 0));*

***for*** *(****int*** *i = 0; i < N; i++)*

***for*** *(****int*** *j = 0; j < N; j++)*

*result[i][j] = (\*a)[i][j] + (\*b)[i][j];*

***return*** *result;*

*}*

*vector<vector<****int****>> add(****const*** *vector<vector<****int****>> \*a,* ***const*** *vector<vector<****int****>> \*b,* ***const*** *vector<vector<****int****>> \*c)*

*{*

*vector<vector<****int****>> result(N, vector<****int****>(N, 0));*

***for*** *(****int*** *i = 0; i < N; i++)*

***for*** *(****int*** *j = 0; j < N; j++)*

*result[i][j] = (\*a)[i][j] + (\*b)[i][j] + (\*c)[i][j];*

***return*** *result;*

*}*

***void*** *divide\_by\_2(vector<vector<****int****>> \*a)*

*{*

***for*** *(****int*** *i = 0; i < N; i++)*

***for*** *(****int*** *j = 0; j < N; j++)*

*(\*a)[i][j] /= 2;*

*}*

***void*** *calculate\_by\_formula(****const*** *vector<vector<****int****>> \*a,* ***const*** *vector<vector<****int****>> \*b,*

***const*** *vector<vector<****int****>> \*c,* ***const*** *vector<vector<****int****>> \*d,*

*vector<vector<****int****>> \*result1, vector<vector<****int****>> \*result2)*

*{*

*vector<vector<****int****>> ac = multiply(a, c);*

*vector<vector<****int****>> ad = multiply(a, d);*

*vector<vector<****int****>> bc = multiply(b, c);*

*vector<vector<****int****>> bd = multiply(b, d);*

*\*result1 = add(&ac, &ad, &bc);*

*\*result2 = add(&ac, &bd);*

*divide\_by\_2(result1);*

*divide\_by\_2(result2);*

*}*

***void*** *print\_to\_file(fstream \*matrices, string pairHeader,* ***const*** *vector<vector<****int****>> \*res1,* ***const*** *vector<vector<****int****>> \*res2)*

*{*

*(\*matrices).clear();*

*(\*matrices).seekp(0, ios::end);*

*(\*matrices) << pairHeader << "****\n****{" << endl;*

***for*** *(****int*** *i = 0; i < N; i++)*

*{*

*(\*matrices) << "****\t****[ ";*

***for*** *(****int*** *j = 0; j < N; j++)*

*(\*matrices) << setw(numWidth) << (\*res1)[i][j] << (j < 2 ? ", " : " ]****\t****");*

*(\*matrices) << "****\t****[ ";*

***for*** *(****int*** *j = 0; j < N; j++)*

*(\*matrices) << setw(numWidth) << (\*res2)[i][j] << (j < 2 ? ", " : " ]****\n****");*

*}*

*(\*matrices) << "};" << endl << endl;*

*}*

***void*** *read\_pair(fstream \*matrices, vector<vector<****int****>> \*readTo1, vector<vector<****int****>> \*readTo2)*

*{*

*string line;*

*getline((\*matrices), line); // считать строку с открывающей скобкой '{'*

***for*** *(****int*** *i = 0; i < N; i++)*

*{*

*getline((\*matrices), line);*

*line = line.substr(3, line.length());*

***for*** *(****int*** *j = 0; j < N; j++)*

*{*

*(\*readTo1)[i][j] = stoi(line.substr(0, numWidth));*

*line = line.substr(numWidth + 2, line.length());*

*}*

*line = line.substr(4, line.length());*

***for*** *(****int*** *j = 0; j < N; j++)*

*{*

*(\*readTo2)[i][j] = stoi(line.substr(0, numWidth));*

*line = line.substr(numWidth + 2, line.length());*

*}*

*}*

*getline((\*matrices), line); // считать строку с закрывающей скобкой "};"*

*getline((\*matrices), line); // считать пустую строку*

*}*

***void*** *search\_for\_duplicate(string pairHeader, fstream \*matrices, ofstream \*knownDuplicates,* ***const*** *vector<vector<****int****>> \*res1,* ***const*** *vector<vector<****int****>> \*res2)*

*{*

*string comparedPairHeader;*

*vector<vector<****int****>> comparedM1(N, vector<****int****>(N));*

*vector<vector<****int****>> comparedM2(N, vector<****int****>(N));*

***bool*** *isDuplicate;*

*(\*matrices).clear();*

*(\*matrices).seekg(0, ios::beg);*

***while*** *(getline((\*matrices), comparedPairHeader) && comparedPairHeader != pairHeader)*

*{*

*isDuplicate = true;*

*read\_pair(matrices, &comparedM1, &comparedM2);*

***for*** *(****int*** *i = 0; i < N; i++)*

*{*

***for*** *(****int*** *j = 0; j < N; j++)*

*{*

***if*** *((\*res1)[i][j] != comparedM1[i][j] ||*

*(\*res2)[i][j] != comparedM2[i][j])*

*{*

*isDuplicate = false;*

***break****;*

*}*

*}*

***if*** *(!isDuplicate)*

***break****;*

*}*

***if*** *(isDuplicate)*

*{*

*(\*knownDuplicates) << pairHeader.substr(0, pairHeader.length() - 2) << " - копия "*

*<< comparedPairHeader.substr(0, comparedPairHeader.length() - 2) << endl*

*<< endl;*

***break****;*

*}*

*}*

*}*

***void*** *operate\_on\_length\_two(fstream \*matrices, ofstream \*knownDuplicates)*

*{*

*vector<vector<****int****>> a;*

*vector<vector<****int****>> b;*

*vector<vector<****int****>> f;*

*vector<vector<****int****>> g;*

*vector<vector<****int****>> res1, res2;*

*string pairHeader;*

***for*** *(****int*** *i = 0; i < 9; i++)*

*{*

*a = AB[i].first;*

*b = AB[i].second;*

***for*** *(****int*** *j = 0; j < 9; j++)*

*{*

*f = FG[j].first;*

*g = FG[j].second;*

*pairHeader = "A" + to\_string(i + 1) + ", B" + to\_string(i + 1) +*

*" x F" + to\_string(j + 1) + ", G" + to\_string(j + 1) + " :";*

*calculate\_by\_formula(&a, &b, &f, &g, &res1, &res2);*

*print\_to\_file(matrices, pairHeader, &res1, &res2);*

*search\_for\_duplicate(pairHeader, matrices, knownDuplicates, &res1, &res2);*

*}*

*}*

*}*

***int*** *main()*

*{*

*fstream matrices("matrices.txt", ios::in | ios::out | ios::trunc | ios::binary);*

*ofstream knownDuplicates("knownDuplicates.txt", ios::trunc);*

*matrices << setfill(' ');*

*streampos readBegin = matrices.tellg();*

*streampos nextReadBegin = matrices.tellg();*

*streampos currentGetterPosition = matrices.tellg();*

*string pairName;*

*string basicMultiplierHeader;*

*string currentMultiplierHeader;*

***const*** *vector<pair<vector<vector<****int****>>, vector<vector<****int****>>>> \*basicMultiplier;*

*vector<vector<****int****>> m1(N, vector<****int****>(N));*

*vector<vector<****int****>> m2(N, vector<****int****>(N));*

*vector<vector<****int****>> res1, res2;*

***for*** *(****int*** *l = 2; l <= 4; l++)*

*{*

***if*** *(l == 2)*

*{*

*operate\_on\_length\_two(&matrices, &knownDuplicates);*

*}*

***else***

*{*

***if*** *(l % 2 == 1)*

*{*

*//указатель на AB*

*basicMultiplier = &AB;*

*basicMultiplierHeader = " x A, B :";*

*}*

***else***

*{*

*//указатель на FG*

*basicMultiplier = &FG;*

*basicMultiplierHeader = " x F, G :";*

*}*

*//матрицы начинают считываться с readBegin*

*//пока не дойдём до nextReadBegin (не включая nextReadBegin)*

*matrices.clear();*

*matrices.seekg(readBegin);*

***while*** *(currentGetterPosition != nextReadBegin && getline(matrices, pairName))*

*{*

*pairName = pairName.substr(0, pairName.length() - 2);*

*read\_pair(&matrices, &m1, &m2);*

*// записали ссылку на текущую строку в переменную (currentGetterPosition)*

*currentGetterPosition = matrices.tellg();*

***for*** *(****int*** *i = 0; i < 9; i++)*

*{*

*calculate\_by\_formula(&m1, &m2, &((\*basicMultiplier)[i].first),*

*&((\*basicMultiplier)[i].second), &res1, &res2);*

*currentMultiplierHeader = basicMultiplierHeader;*

*currentMultiplierHeader.insert(6, to\_string(i + 1));*

*currentMultiplierHeader.insert(10, to\_string(i + 1));*

*print\_to\_file(&matrices, pairName + currentMultiplierHeader, &res1, &res2);*

*search\_for\_duplicate(pairName + currentMultiplierHeader, &matrices, &knownDuplicates, &res1, &res2);*

*}*

*matrices.clear();*

*matrices.seekg(currentGetterPosition);*

*}*

*}*

*//операции для этой длины закончены, записываем в readBegin nextReadBegin*

*//записываем ссылку на конец файла в переменную nextReadBegin*

*readBegin = nextReadBegin;*

*matrices.clear();*

*matrices.seekg(0, ios::end);*

*nextReadBegin = matrices.tellg();*

*}*

*//тут конец программы*

*matrices.close();*

*knownDuplicates.close();*

*}*

Примечание: программа никак не сообщает о том, что она выполняется. Отследить это можно только по происходящим изменениям в файлах matrices.txt и knownDuplicates.txt, а остановить её можно только с помощью диспетчера задач.