Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовой работе

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма Форда-Беллмана»

Выполнил:

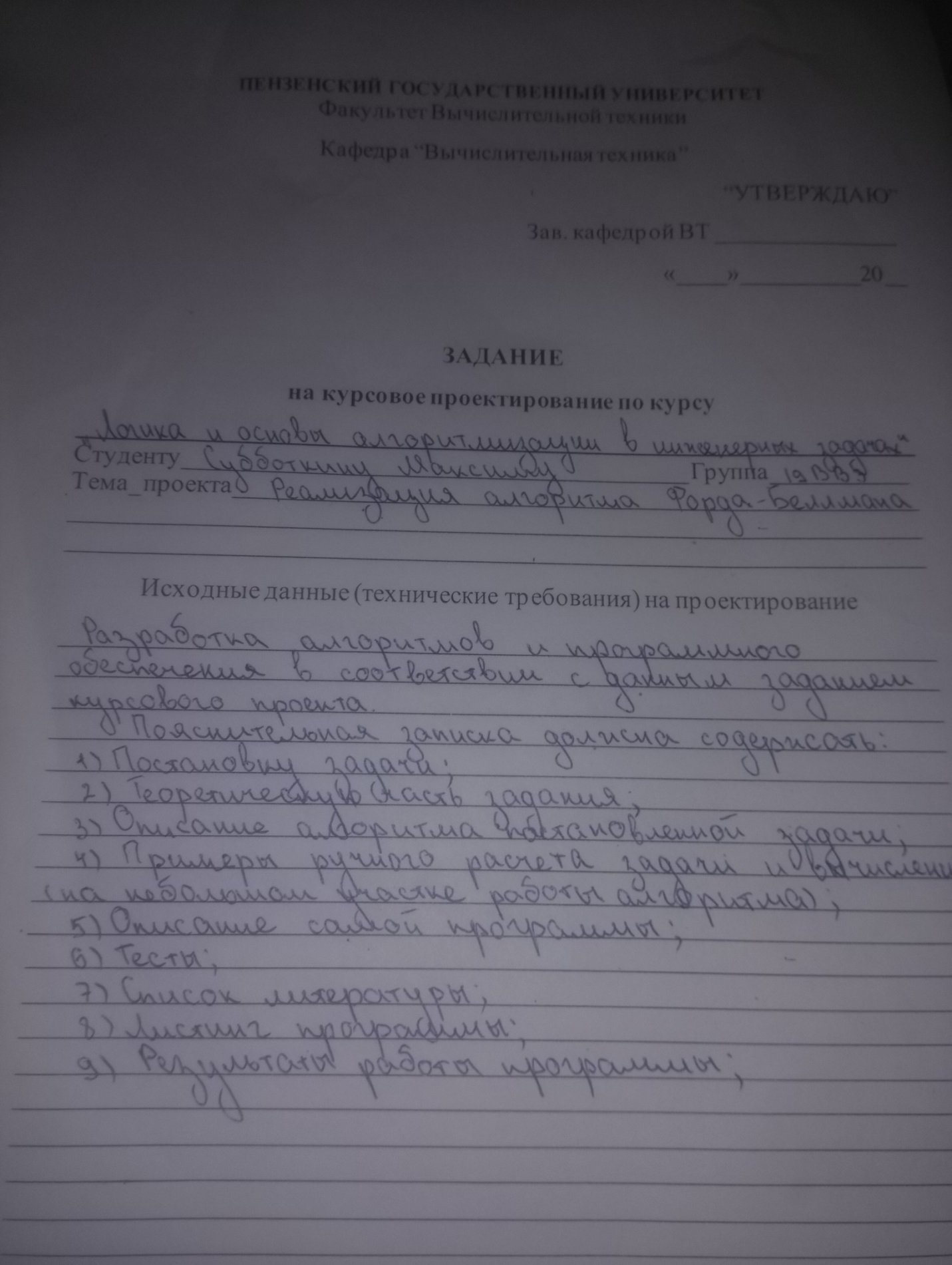
Студент группы 19ВВ3

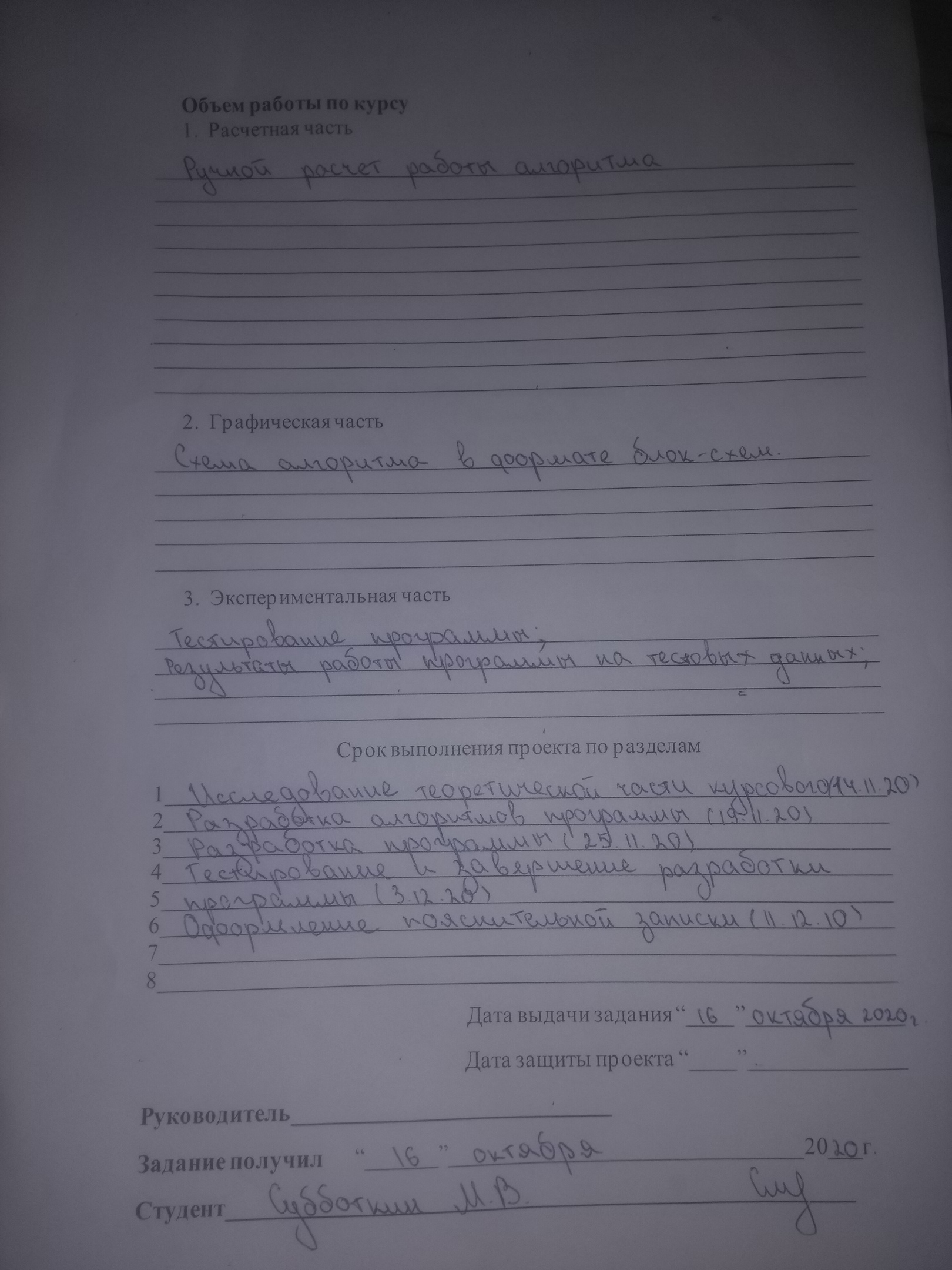
Субботкин М.В.

Принял:

Митрохин М.А.

Пенза 2020





Содержание

[**Реферат** 5](#_Toc59455884)

[**Введение** 6](#_Toc59455887)

[**1 Постановка задачи** 7](#_Toc59455888)

[**2 Теоретическая часть** 8](#_Toc59455889)

[**3 Описание алгоритма программы** 9](#_Toc59455890)

[**4 Пример ручного просчёта задачи** 11](#_Toc59455891)

[**5 Описание программы** 14](#_Toc59455894)

[**6 Тестирование программы** 20](#_Toc59455895)

[**Заключение** 27](#_Toc59455897)

[**Список литературы** 28](#_Toc59455898)

[**Приложение А** 29](#_Toc59455902)

[**Листинг** 29](#_Toc59455903)

**Реферат**

**Отчет 32 стр, 20 рисунков.**

АЛГОРИТМ ФОРДА-БЕЛЛМАНА, КРАТЧАЙШИЕ РАССТОЯНИЯ, ОТРИЦАТЕЛЬНЫЙ ЦИКЛ.

Цель исследования – разработка программы, выполняющей алгоритм Форда-Беллмана.

В работе рассмотрен способ нахождения кратчайших расстояний от одной вершины в графе. Для поиска можно использовать ориентированную взвешенную матрицу смежности. Выявлено, что с помощью данного алгоритма можно найти кратчайшие расстояния от одной вершины в графе, а так же выяснить, содержится ли в графе отрицательный цикл.

**Введение**

**Алгоритм Форда-Беллмана** позволяет найти кратчайшие пути из одной вершины графа до всех остальных, даже для графов, в которых веса ребер могут быть отрицательными. Тем не менее, в графе не должно быть циклов отрицательного веса, достижимых из начальной вершины, иначе вопрос о кратчайших путях является бессмысленным.

Алгоритм вычисляет кратчайшие пути снизу вверх. Сначала он вычисляет самые короткие расстояния, то есть пути длиной не более, чем в одно ребро. Затем он вычисляет кратчайшие пути длиной не более двух ребер и так далее. После *i*-й итерации внешнего цикла вычисляются кратчайшие пути длиной не более*i*ребер. В любом простом пути может быть максимум *|V|-1* ребер, поэтому внешний цикл выполняется именно *|V|-1* раз. Идея заключается в том, что если мы вычислили кратчайший путь с не более чем *i* ребрами, то итерация по всем ребрам гарантирует получение кратчайшего пути с не более чем *i + 1* ребрам

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2019, язык программирования – Си и Си++.

Целью данной курсовой работы является разработка программы на языке Си и С++, которые являются широко используемым. Именно с их помощью в данном курсовом проекте реализуется алгоритм Форда-Беллмана, осуществляющий поиск кратчайших пути из одной вершины до всех остальных вершин графа.

# **1 Постановка задачи**

Требуется разработать программу, которая осуществит поиск кратчайших пути из одной вершины до всех остальных вершин графа, используя алгоритм Форда-Беллмана.

Исходный граф в программе должен задаваться взвешенной и ориентированной матрицей смежности. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После этого он вводит начальную вершину, от которой найдутся все кратчайшие пути до остальных вершин графа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройства ввода- клавиатура и мышь.

**2 Теоретическая часть**

Граф G (рисунок 1) задается множеством вершин X1, X2, ..., Xn. и

множеством ребер , соединяющих между собой определенные вершины. Ребра из множества А ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется ориентированным графом. Так же граф может иметь вес рёбер , который так же может быть отрицательным.



Рисунок 1-Пример графа

При представлении графа матрицей смежности информация о ребрах

графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается весом ребра, иначе нулем.

Идея алгоритма заключается в следующем. В начале инициализируются расстояния от исходной вершины до всех остальных вершин, как бесконечные, а расстояние до самой вершины принимается равным нулю. Вторым шагом вычисляются самые короткие расстояния. Следующие шаги нужно выполнять |V|-1 раз, где |V| — число вершин в данном графе. На этом шаге сообщается, присутствует ли в графе цикл отрицательного веса.

**3 Описание алгоритма программы**

Для программной реализации алгоритма понадобится два массива: SourceMatrix(int) - хранит матрицу смежности, MinPath (int) – содержит минимальные пути. Итак, имеется граф G=(V, E). Расстояние для начальной вершины (src) равняется к нулю , а расстояние до всех вершин равно бесконечности.

Ниже представлен псевдокод функции Ford-Bellman.

**Вход**: Матрица смежности SourceMatrix графа G, источник src

**Выход:** Минимальное расстояние от вершины src до всех остальных

**Алгоритм ПОШ**

1. Для всех i выполнять
2. MinPath[i] = INF
3. MinPath[src] = 0
4. Цикл от k до V выполнять
   1. Цикл от i до V выполнять

4.2 Цикл от j до V выполнять

4.3 Если SouceMatrix не равно 0 выполнять

4.5 Если MinPath[j] больше чем MinPath[j]+SourceMatrix[i][j] выполнять

* 1. MinPath[j] = MinPath[j] + SourceMatrix[i][j]
  2. Конец цикла

4.8 Конец цикла

4.9 Конец цикла

5 Цикл от i до V выполнять

5.1 Цикл от j до V выполнять

5.2 Если SouceMatrix не равно 0 выполнять

5.3 Если MinPath[j] больше чем MinPath[j]+SourceMatrix[i][j] выполнять

5.4 Вывести ошибку

5.5 Выйти

5.6 Конец цикла

5.7Конец цикла

**4 Пример ручного просчёта задачи**

Пусть задан следующий граф.

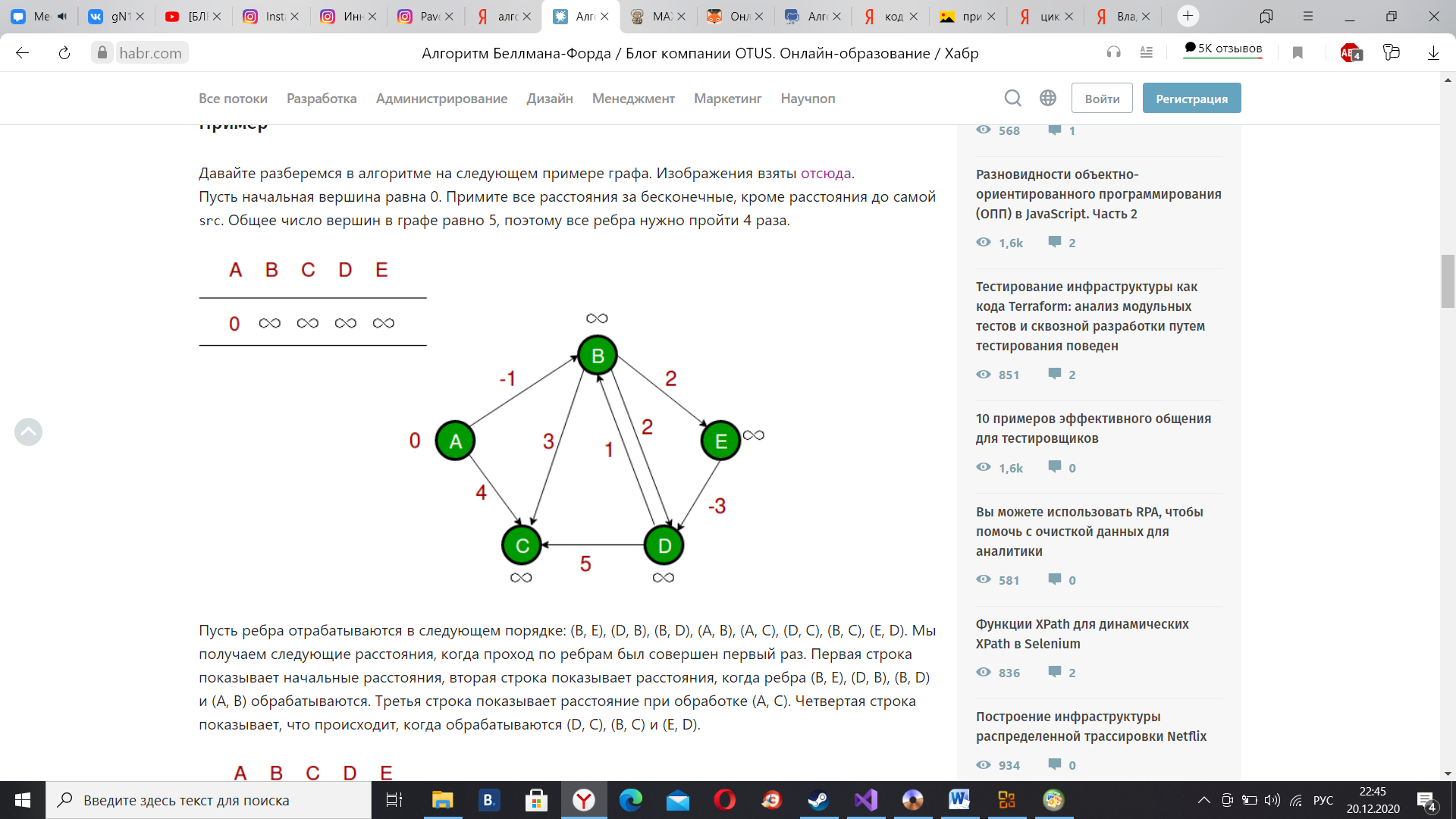


Рисунок 2- Граф для ручного расчёта

Пусть начальная вершина равна 0. Примем все расстояния за бесконечные, кроме расстояния до самой src. Общее число вершин в графе равно 5, поэтому все ребра нужно пройти 4 раза.

Пусть ребра отрабатываются в следующем порядке: (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D). Мы получаем следующие расстояния, когда проход по ребрам был совершен первый раз. Первая строка показывает начальные расстояния, вторая строка показывает расстояния, когда ребра (B, E), (D, B), (B, D) и (A, B) обрабатываются. Третья строка показывает расстояние при обработке (A, C). Четвертая строка показывает, что происходит, когда обрабатываются (D, C), (B, C) и (E, D).

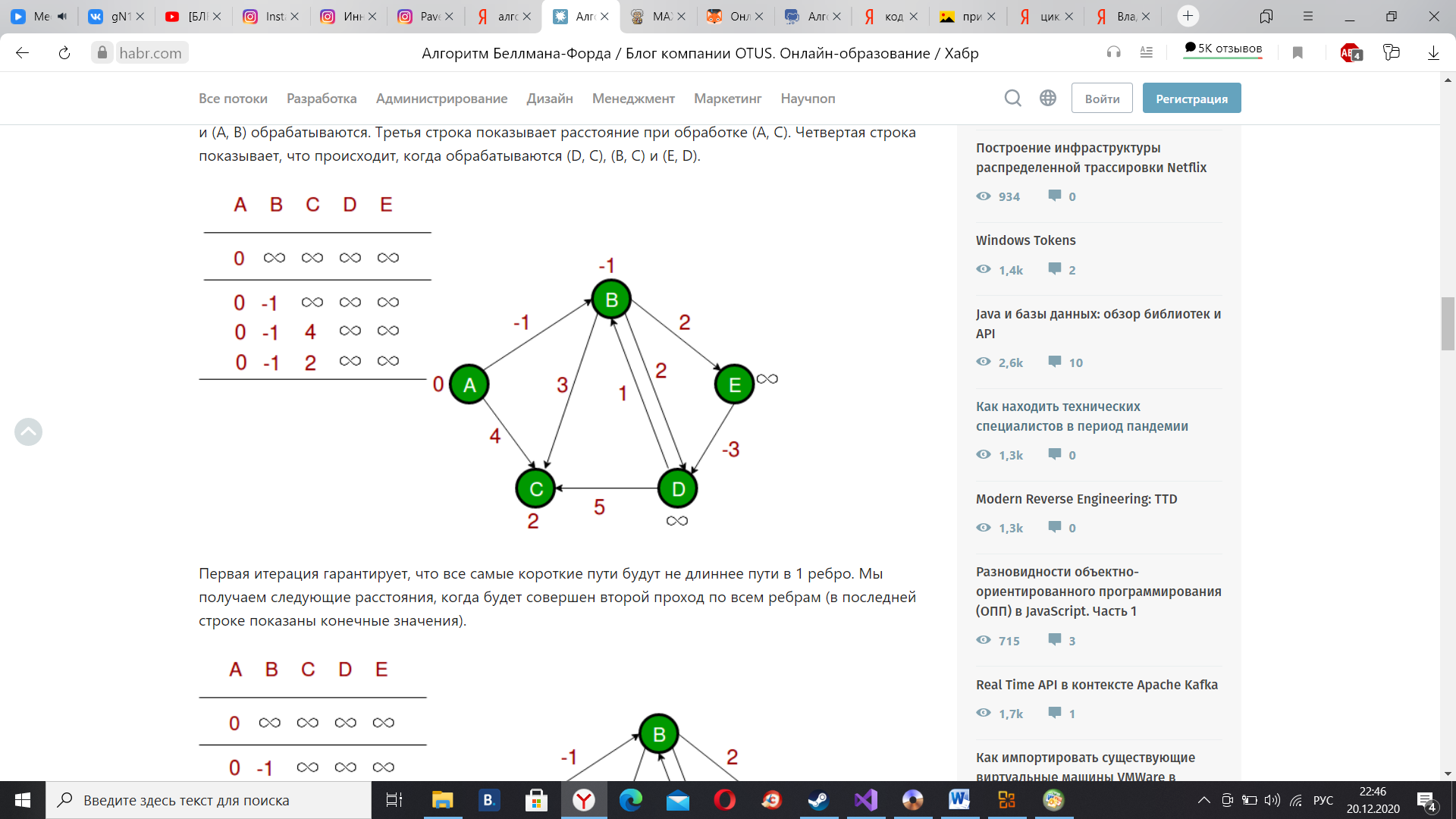


Рисунок 3 - первая итерация

Первая итерация гарантирует, что все самые короткие пути будут не длиннее пути в 1 ребро. Мы получаем следующие расстояния, когда будет совершен второй проход по всем ребрам (в последней строке показаны конечные значения).

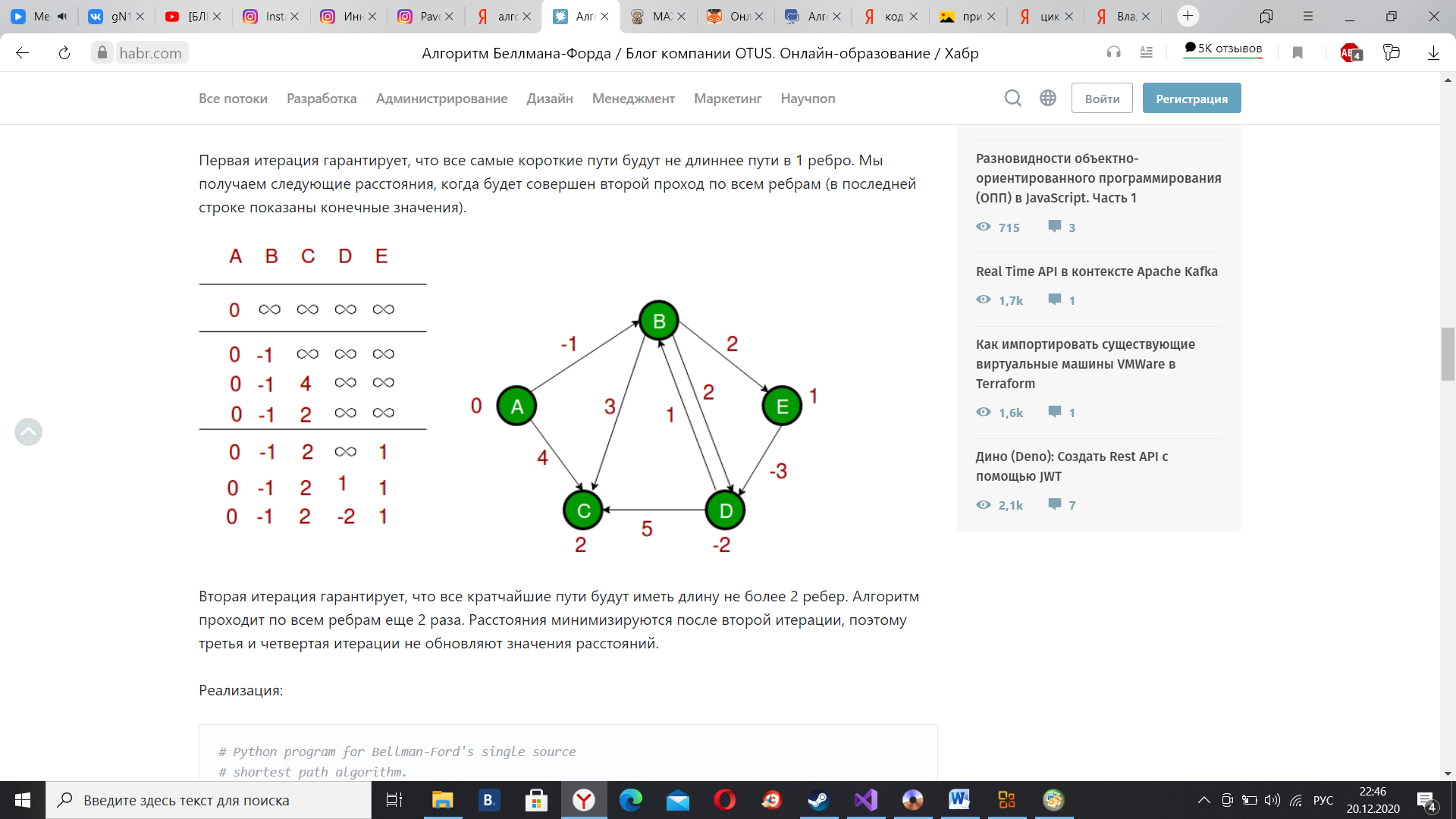


Рисунок 4 - Вторая итерация

Вторая итерация гарантирует, что все кратчайшие пути будут иметь длину не более 2 ребер. Алгоритм проходит по всем ребрам еще 2 раза. Расстояния минимизируются после второй итерации, поэтому третья и четвертая итерации не обновляют значения расстояний.

Пример ручного просчета в случае отрицательного цикла

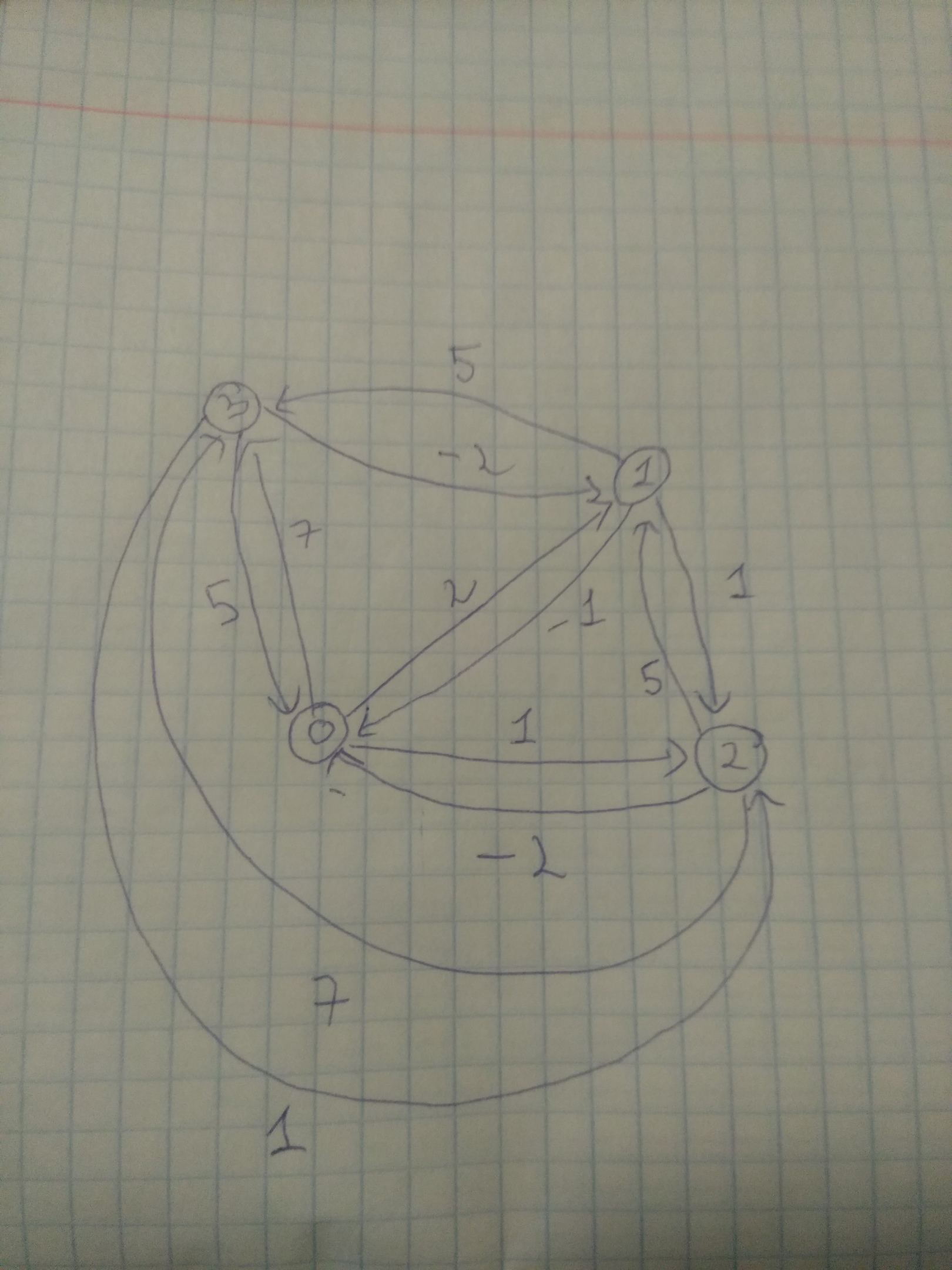


Рисунок 5 - Пример графа с отрицательным циклом

В данном графе ищутся расстояния от вершины 3 до всех остальных вершин. Так как мы разобрали алгоритм поиска кратчайших расстояний, то можно утверждать, что кратчайшее расстояние из вершины 3 в вершину 0 проходит из вершины 0 в вершину 1 – расстояние -2 и далее из вершины 1 в вершину 0 – расстояние -1. Получается, что кратчайшее расстояние равно -3.

Но при следующей итерации выясняется, что кратчайшее расстояние равное -3 больше, чем расстояние -3 +(-2) из-за чего цикл поиска выполняется снова, и так при каждой следующей итерации. Таким образом, выходные данные будут неверные, но алгоритм Форда-Беллмана позволяется нам предупредить отрицательный цикл и вывести сообщение об его наличии.

**5 Описание программы**

Для написания данной программы использованы языки программирования Си и Си++.

Проект был создан в виде консольного приложения в программе Visual studio 2019

Данная программа является многомодульной, поскольку состоит из нескольких файлов: main.cpp, Menu.cpp, fileOutput.cpp, Ford\_Ballman.cpp, Generation.cpp, outputFord.cpp, outputMass.cpp, Title.cpp, Header.h.

Файл Header.h содержит библиотеки и объявление некоторых функций.

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#define INF 10000000;

#include <conio.h>

#include <time.h>

#include <random>

#include <stdio.h>

#include <Windows.h>

#include <locale.h>

void outputFord(int count, int\* result, int apex);

void fileOutput(int count, int\* result, int apex, FILE\* file);

void Ford\_Ballman(int CountApex, int\*\* SourceMatrix, int Start, FILE\* fin, int status);

void Title();

void Menu();

void output(int\*\* mass, int f);

void randomGen(int f, int\*\* mass);

main.cpp включает в себя функции работы с матрицами и файлами.

Menu.cpp осуществляет вывод меню в консоль и обработку событий при выборе пункта меню.

Title.cpp осуществляет вывод титульной страницы.

fileOutput.cpp осуществляет сохранение результата в файл.

Ford\_Ballman.cpp реализует сам алгоритм.

Generation.cpp осуществляет генерацию графа.

outputFord.cpp осуществляет вывод результата алгоритма на экран.

outputMass.cpp осуществляет вывод графа на экран.

В программе имеются следующие функции: outputFord, fileOutput, Ford-Ballman, Title, Menu, output, selected, randomGen, main.

Работа программы начинается с вывода в консоли меню.

#include "Header.h"

void Menu() {

printf("\nВвыберите операцию");

printf("\n1) Сгенерировать граф");

printf("\n2) Запустить алгоритм");

printf("\n3) Сохранить результат в файл");

printf("\n4) Выйти\n");

\_getch();

}

Где пользователь выбирает один из пунктов: Генерацию графа, Алгоритм, Сохранение или Выход.

Если пользователь выбирает пункт “Cоздание”, то программа попросит ввести его размер графа, после чего сгенерируется и выведется матрица смежности взвешенного неориентированного графа.

#include "Header.h"

void randomGen(int f, int\*\* mass) {

for (int i = 0; i < f; i++) {

mass[i] = (int\*)malloc(sizeof(int) \* f);

}

for (int i = 0; i < f; i++) {

for (int j = 0; j < f; j++) {

int c = (rand() % 10 - 2);

mass[i][j] = c;

if (i == j) {

mass[i][j] = 0;

}

}

}

for (int i = 0; i < f; i++) {

for (int j = 0; j < f; j++) {

printf("|%d|\t ", mass[i][j]);

}

printf("\n");

}

}

При выборе пункта “Алгоритм”, программа попросит пользователя ввести вершину , от которой будут искаться расстояния. В зависимости от вершин и наличия сгенерированной матрицы программа выдаст различные результаты.

else if (input == 2) {

if (mass == 0) {

system("cls");

printf("Граф не найден!\n");

printf("\nВведите размер графа : ");

scanf\_s("%d", &f);

mass = (int\*\*)malloc(sizeof(int) \* f);

randomGen(f, mass);

}

system("cls");

output(mass, f);

printf("Введите стратовую веришу : ");

scanf("%d", &start);

if (start > f - 1) {

printf("Введена неправильная вершина!");

\_getch();

system("cls");

Menu();

}

else {

Ford\_Ballman(f, mass, start - 1, 0, 1);

\_getch();

system("cls");

Menu();

}

}

“Сохранение” выполняет запись результатов работы программы в текстовый файл.

FILE\* fout;

if (input == 3) {

char name[] = "data.txt";

if ((fout = fopen(name, "w+")) == NULL) {

printf("Файл не найден!");

\_getch();

}

else {

system("cls");

printf("Файл сохранен\n");

fout = fopen("C:\\new\\data.txt", "w");

for (int i = 0; i < f; i++) {

for (int j = 0; j < f; j++) {

fprintf(fout, "|%d|\t ", mass[i][j]);

if (j == f - 1)

fprintf(fout, "\n");

}

}

Ford\_Ballman(f, mass, start - 1, fout, 2);

printf("\n");

Menu();

void fileOutput(int count, int\* result, int apex, FILE\* file) {

int i;

fprintf(file, "Результат : \n");

for (i = 0; i < count; i++)

{

if (result[i] < 10000) {

fprintf(file, "Минимальное расстояние между вершинами %d и %d = %d\n\n", apex, \

i, result[i]);

}

else {

fprintf(file, "Расстояние из %d = INF\n", apex, \

i);

}

}

}

И последний пункт “Выход”, завершает работу программы с кодом 0.

Ниже можно увидеть оформление после запуска программы и его последующие изменения, после выбора соответствующих пунктов (Рисунки 6 – 10).

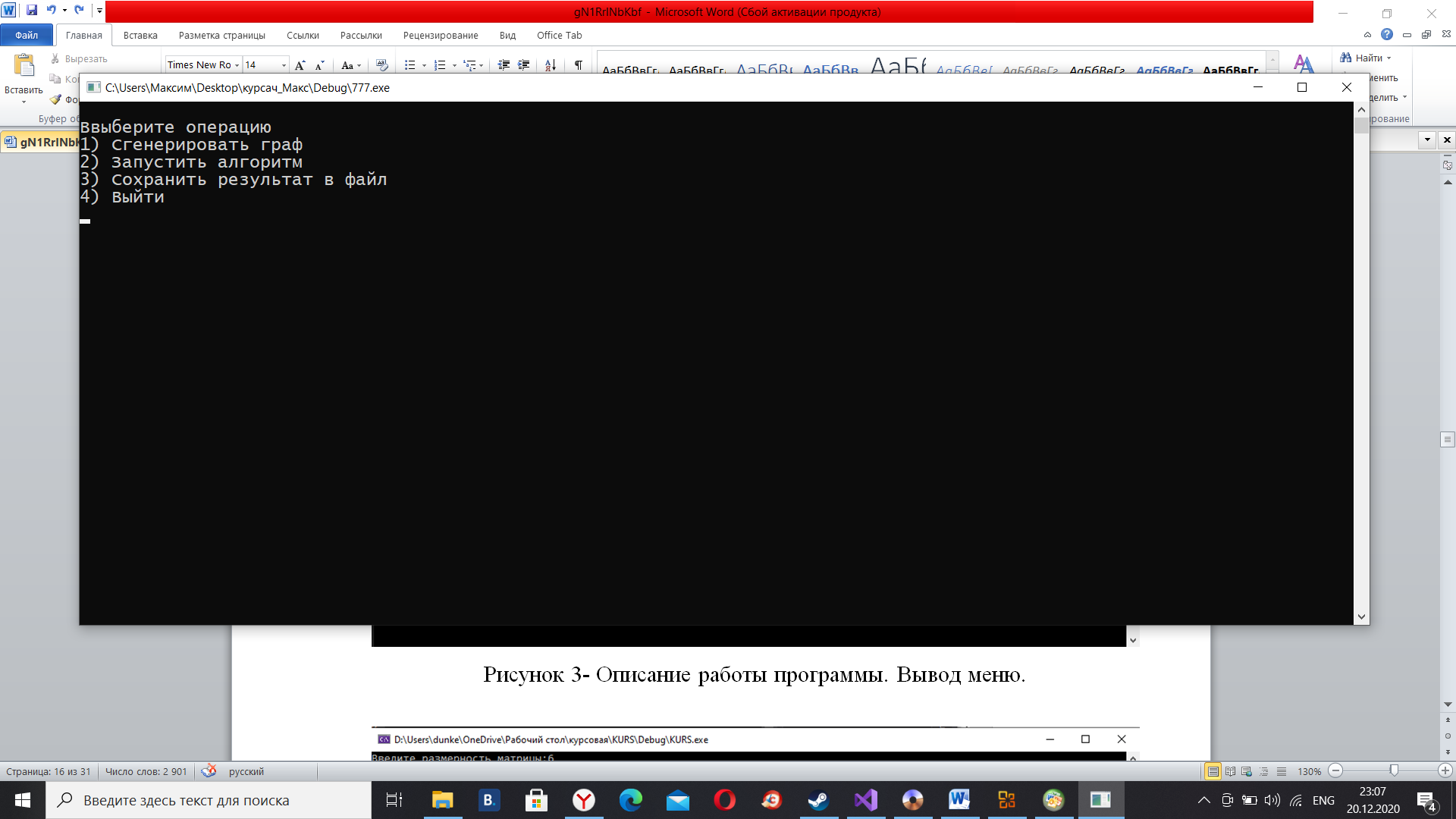


Рисунок 6- Описание работы программы. Вывод меню.

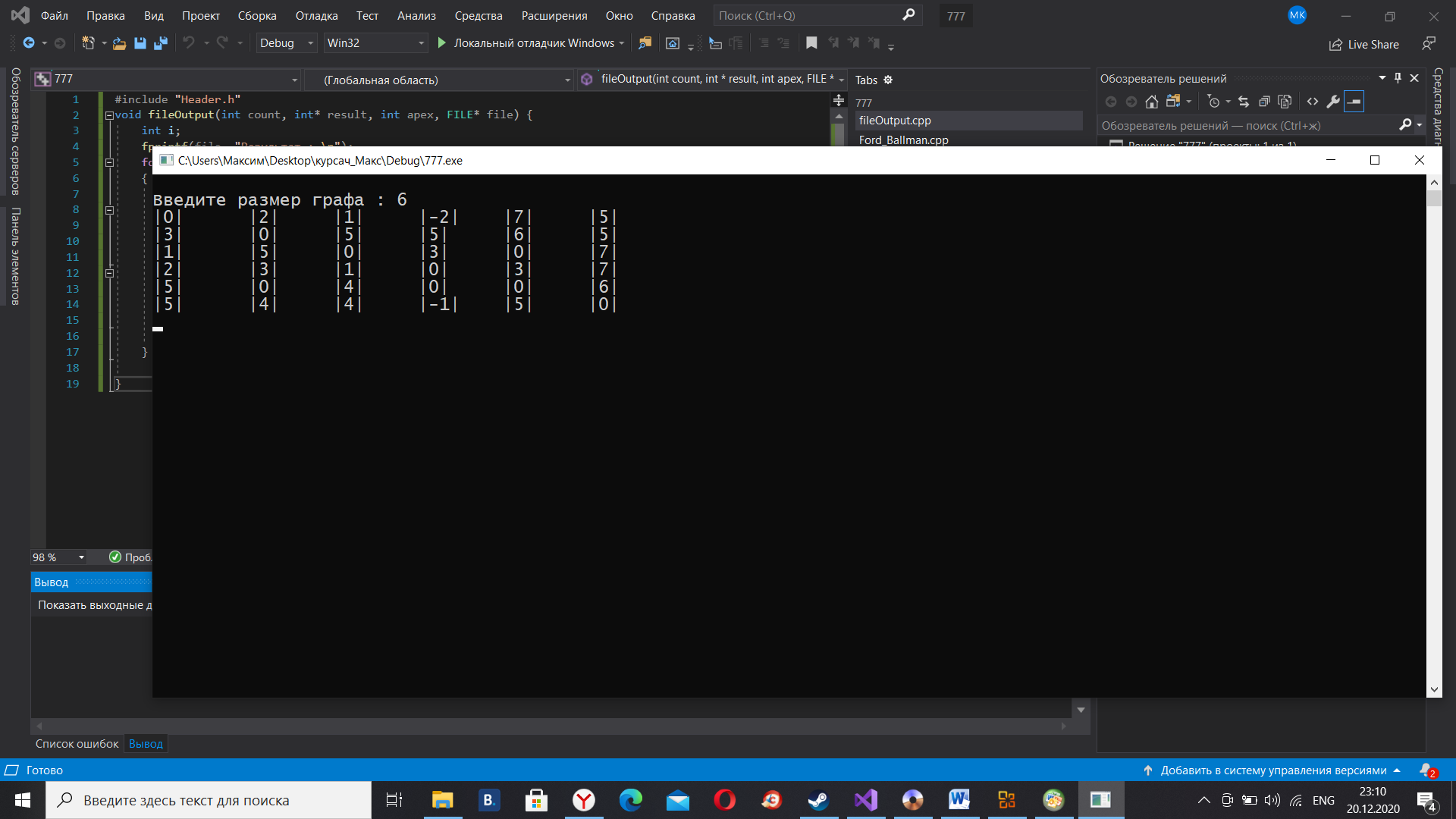


Рисунок 7-Описание работы программы. Генерация матрицы

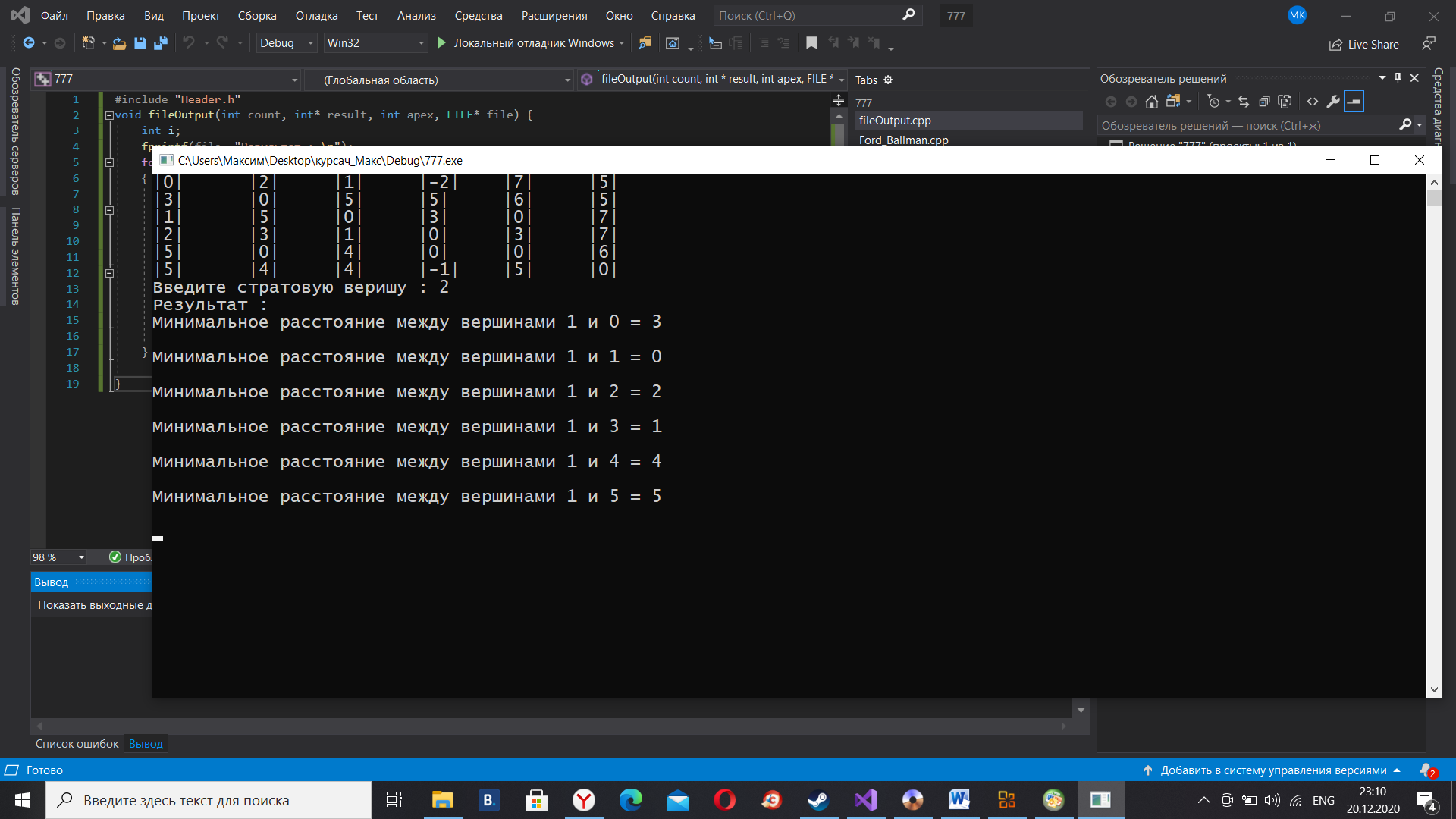


Рисунок 8-Описание работы программы. Результат работы алгоритма.

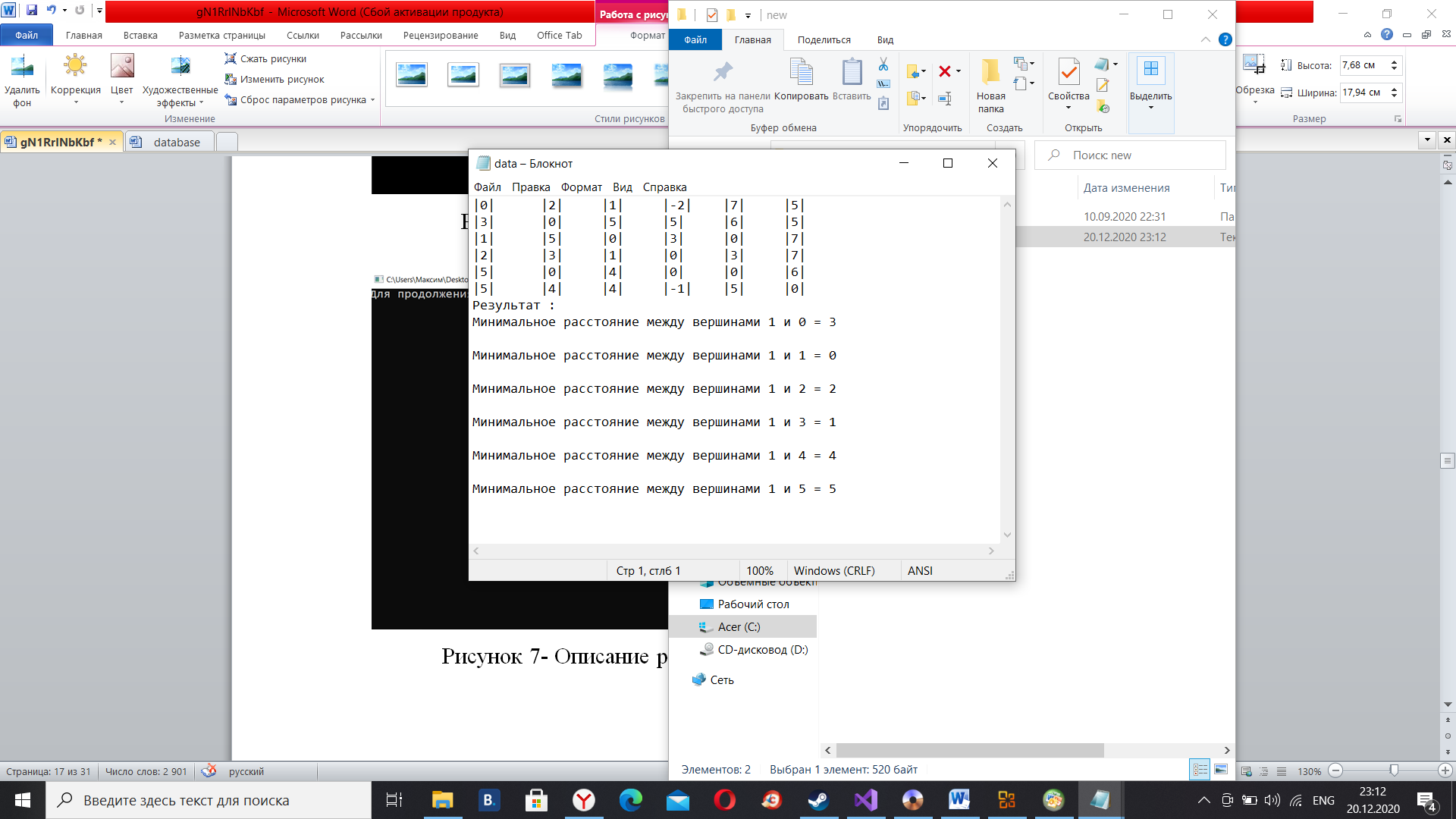


Рисунок 9-Описание работы программы. Сохранение файла

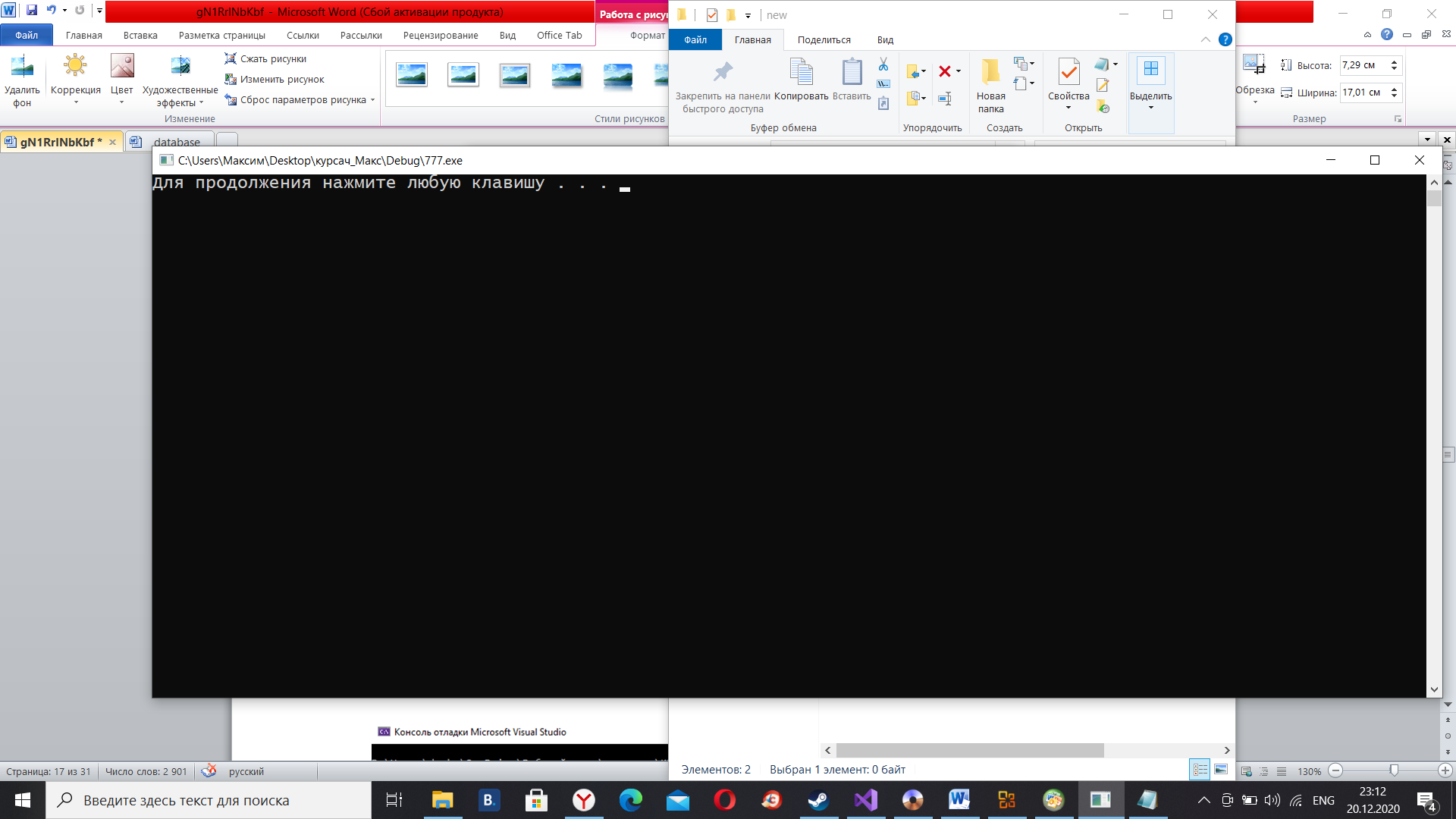


Рисунок 10- Описание работы программы. Выход из программы.

**6 Тестирование программы**

Среда разработки MicrosoftVisualStudio 2019 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Таблица 1- Описание поведения программы при тестировании

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер  теста | Описание теста | Последовательность действий(как достичь результата) | Ожидаемый результат. | Полученный результат |
| 1 | Запуск программы | Запустить «777.exe» | Вывод в консоль меню | Верно |
| 2 | Генерация матрицы | Выбор пункта меню «Сгенерировать граф» | Вывод сгенерированной матрицы смежности. | Верно |
| 3 | Ошибка при выборе пункта «Алгоритм» | Выбрать пункт «Запустить алгоритм» с отсутствующей матрицей | Вывод сообщения в консоль «Граф не найден! Введите размер» | Верно |
| 4 | Ошибка при введении вершин | Ввести вершину, которой нет в графе | Вывод сообщения в консоль «Введена неправильная вершина!» | Верно |
| 5 | Корректный вывод путей | Ввести вершину, из которой есть связи | Вывод в консоль минимальных путей | Верно |
| 6 | Отрицательный цикл | Ввести вершину, при которой найдется отрицательный цикл | Вывод сообщения “Ошибка! Найден отрицательный цикл” | Верно |
| 7 | Сохранение в файл | Выбор пункта меню «Сохранение» | Создание файла, содержащего результаты работы программы | Верно |
| 8 | Выход из программы | Выбор пункта меню «Выход» | Завершение работы программы с кодом 0 | Верно |
| 9 | Результат работы программы | Ввести вершину из которой есть пути | Совпадение результатов работы программы с ручными рассчётами | Верно |
|  |  |  |  |  |

Ниже представлены результаты тестирования(Рисунки 11-20). Изначально после запуска программы выведется меню в консоль.



Рисунок 11- Результат 1 теста

После выбора Пункта «Создание» программа предложит пользователю ввести размерность матрицы смежности взвешенного ориентированного графа. После чего она будет сгенерирована и выведена на экран.

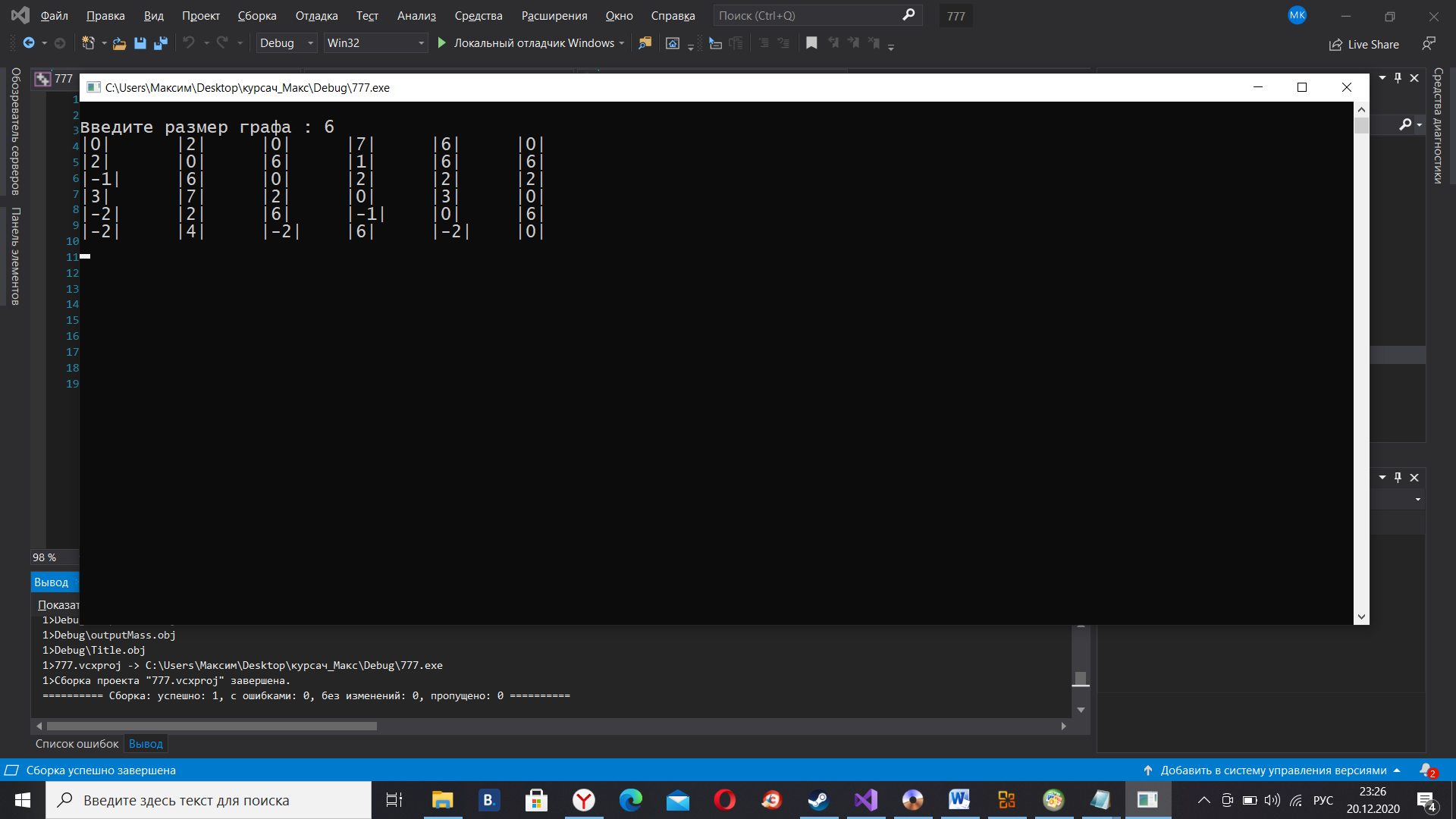


Рисунок 12-Результат работы 2 теста

Если же пользователь не задаст размер матрицы и выберет пункт «Алгоритм», то программа выведет ошибку и попросит его сгенерировать матрицу.

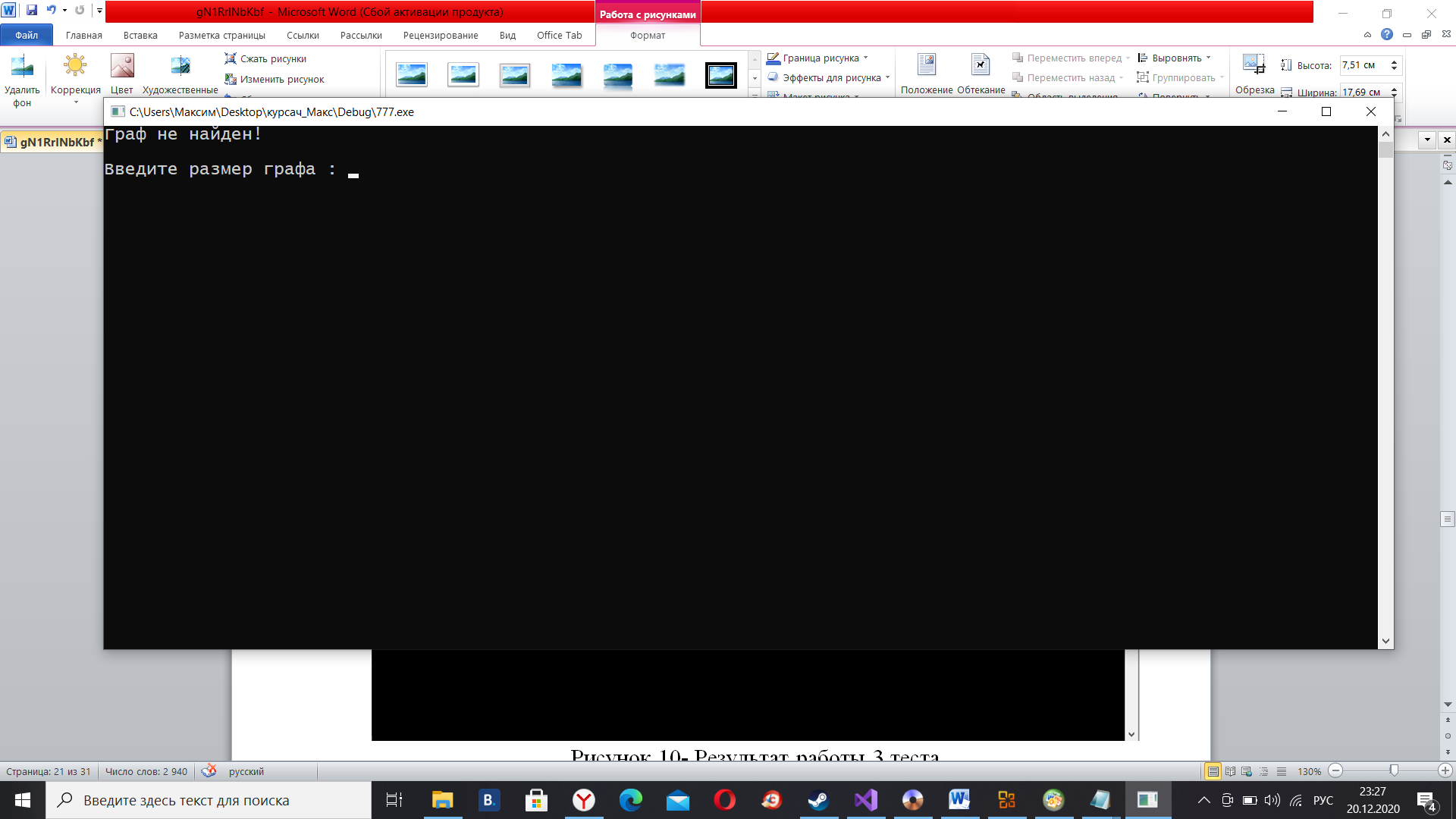


Рисунок 13- Результат работы 3 теста

При введение вершины , которой не существует в графе, программа выведет ошибку и попросит ввести другую вершину

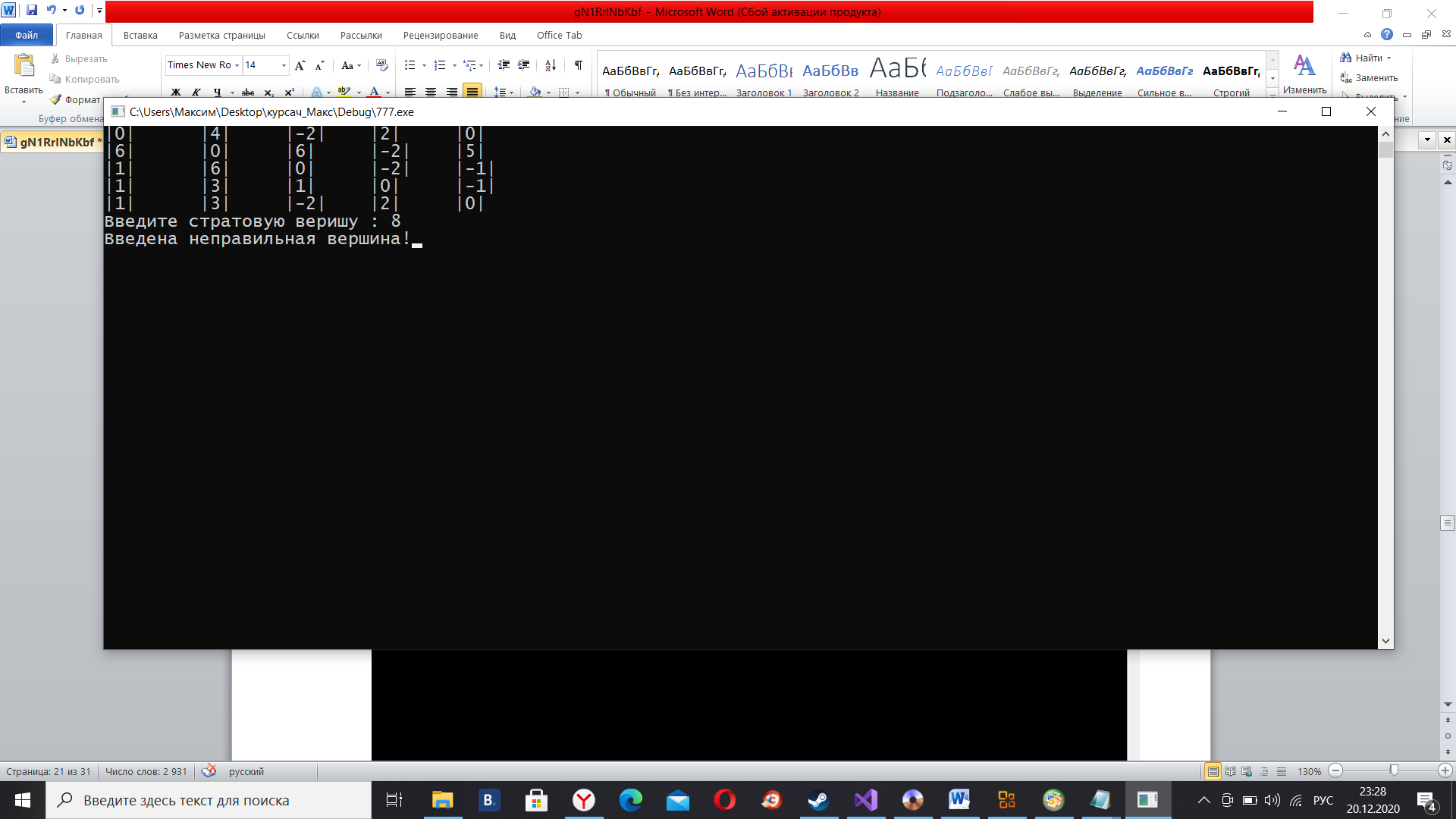


Рисунок 14- Результат работы 4 теста

При корректном вводе программа выдает правильный результат

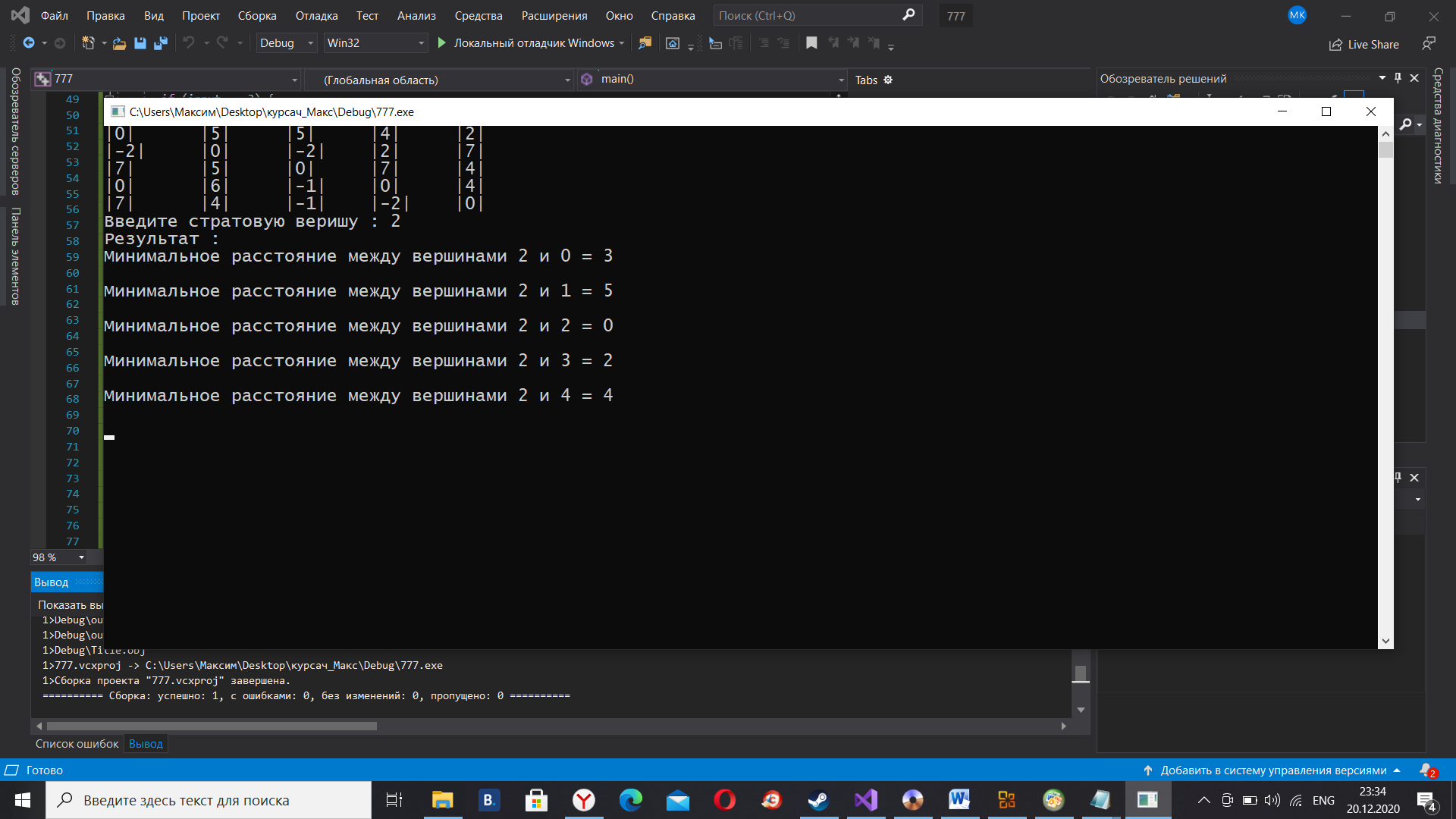


Рисунок 15- Результат работы 5 теста

При наличии отрицательного цикла программа сообщит об этом

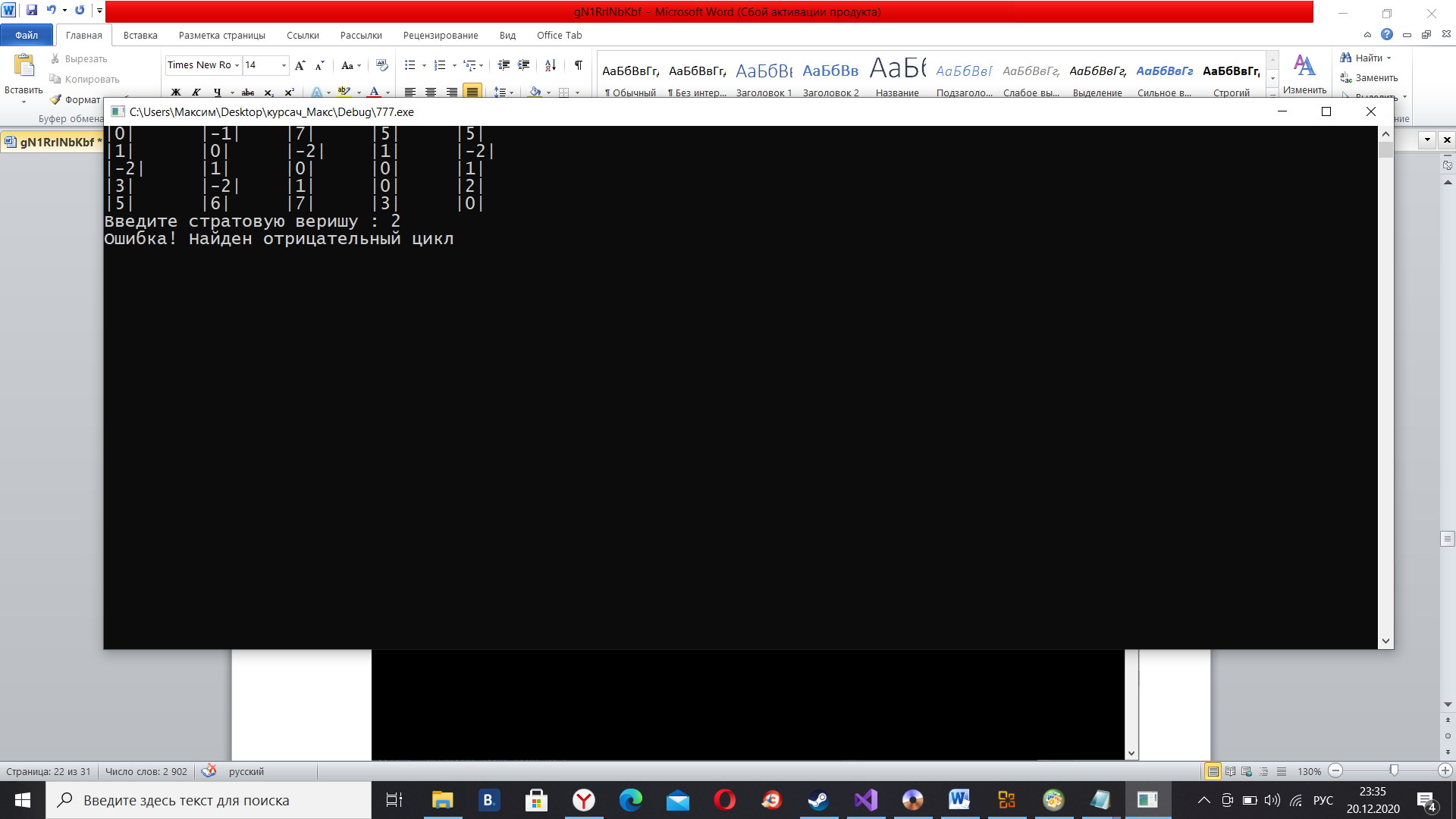


Рисунок 16- Результат работы 6 теста

Когда пользователь выбирает пункт «Сохранение» программа записывает результаты в файл, если его не существует, то создаёт.

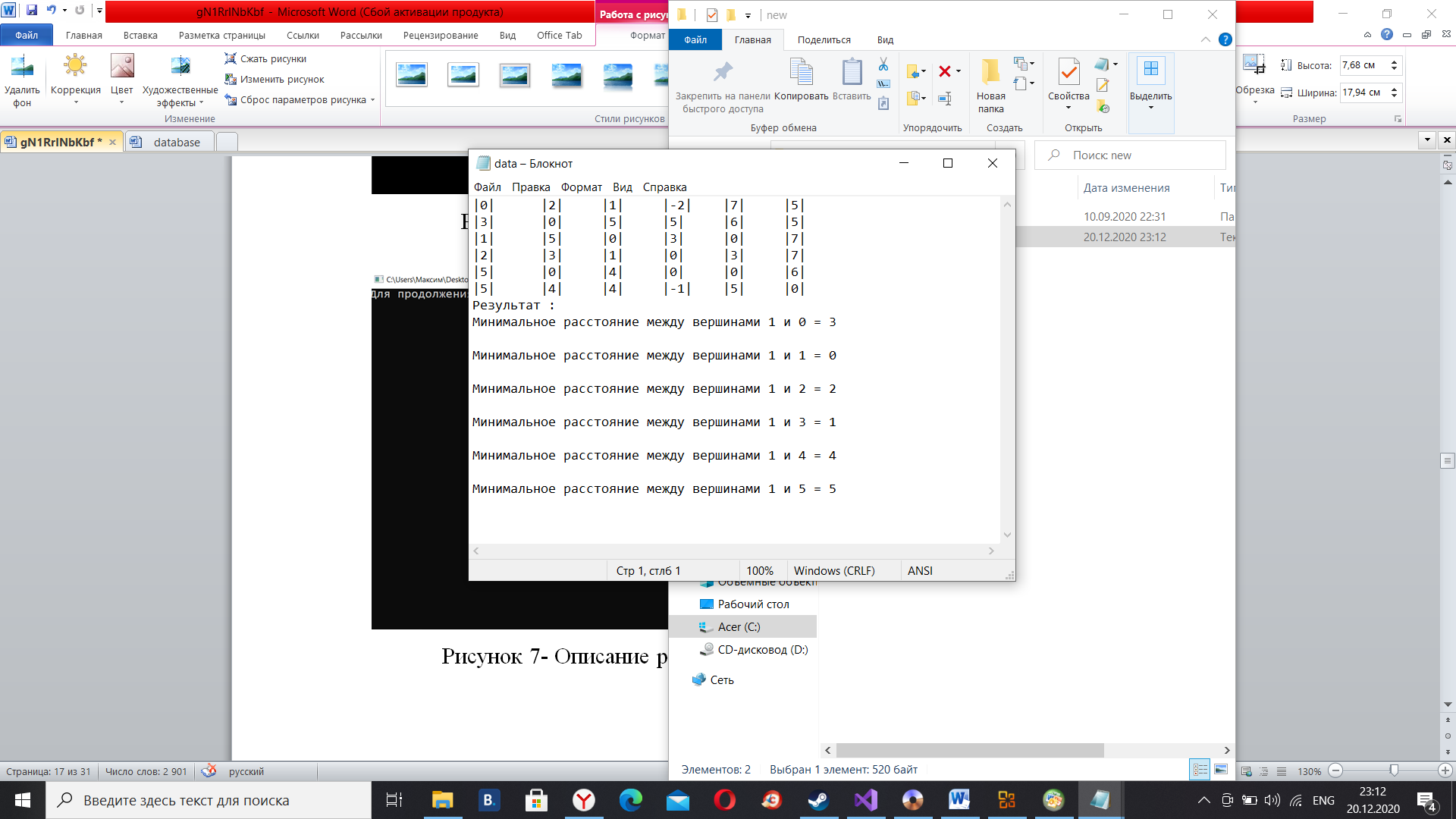


Рисунок 17- Результат работы 7 теста

При выборе пункта «Выход» программа завершит работу с кодом 0.

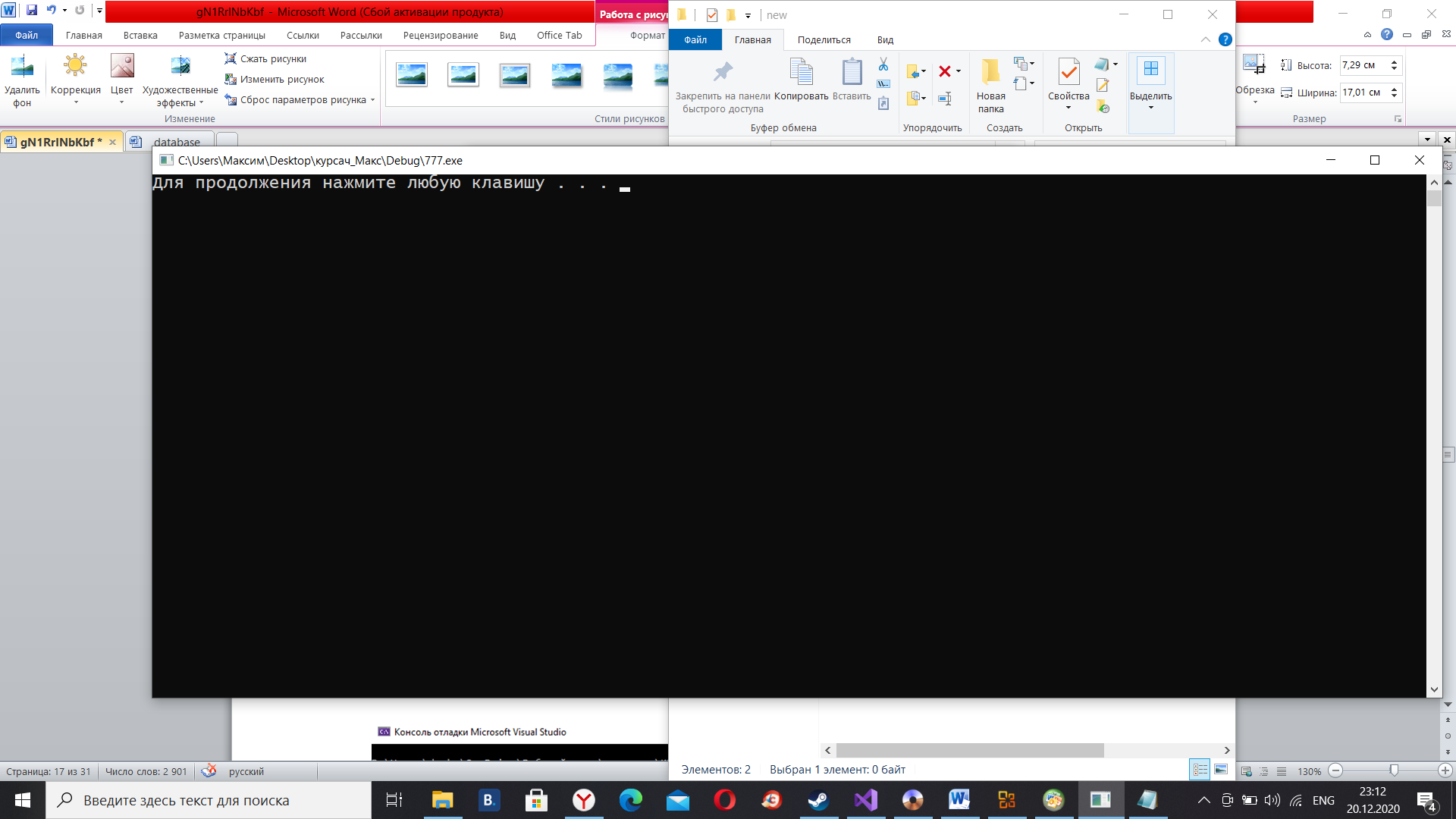


Рисунок 18- Результат работы 8 теста

Сгенерировали граф, выполнили алгоритм и сравнили его с ручными рассчётами.

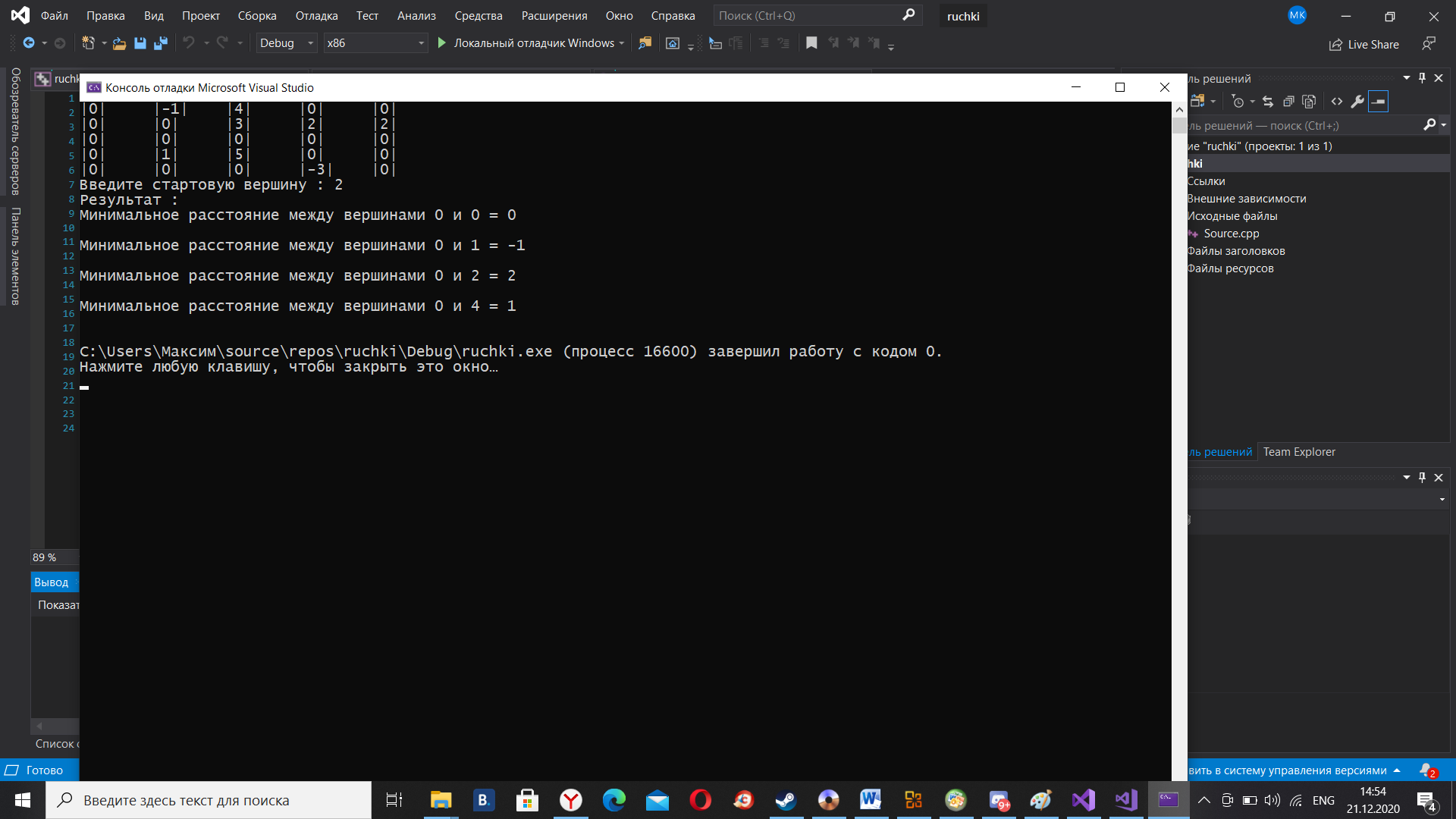


Рисунок 19- Проверка ручного расчёта.

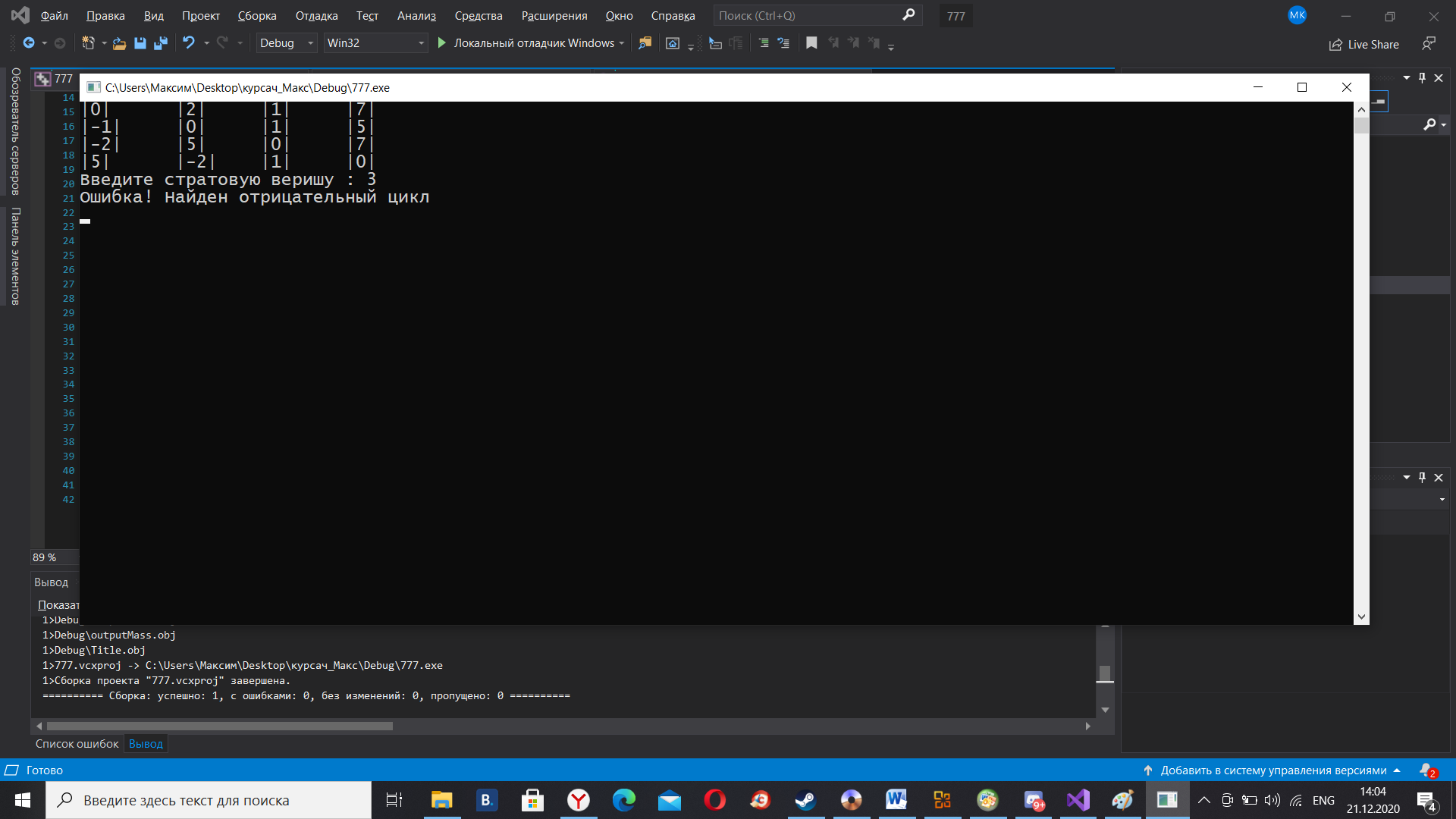


Рисунок 20- Проверка ручного расчёта отрицательного цикла

Результаты работы программы совпадают с ручными расчётами, следовательно программа работает корректно.

**Заключение**

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Форда-Беллмана для поиска максимального потока в MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей. Приобретены навыки по осуществлению алгоритма Форда-Беллмана. Углублены знания языка программирования Cи.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

**Список литературы**

1. https://habr.com/ru/company/otus/blog/484382/

1. <https://ru.wikipedia.org/wiki/Алгоритм_Беллмана_—_Форда>.

# 3.[Стивен С. Скиена – Алгоритмы. Руководство по разработке, 2011 г.](https://vk.com/doc263766508_440149417?hash=2f5bbb2381d7401b60&dl=81c4639ff4ef5e18c4)

**Приложение А**

**Листинг**

**Header.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#define INF 10000000;

#include <conio.h>

#include <time.h>

#include <random>

#include <stdio.h>

#include <Windows.h>

#include <locale.h>

void outputFord(int count, int\* result, int apex);

void fileOutput(int count, int\* result, int apex, FILE\* file);

void Ford\_Ballman(int CountApex, int\*\* SourceMatrix, int Start, FILE\* fin, int status);

void Title();

void Menu();

void output(int\*\* mass, int f);

void randomGen(int f, int\*\* mass);

**fileOutput.cpp**

#include "Header.h"

void fileOutput(int count, int\* result, int apex, FILE\* file) {

int i;

fprintf(file, "Результат : \n");

for (i = 0; i < count; i++)

{

if (result[i] < 10000) {

fprintf(file, "Минимальное расстояние между вершинами %d и %d = %d\n\n", apex, \

i, result[i]);

}

else {

fprintf(file, "Расстояний из %d не существует \n", apex, \

i);

}

}

}

**Ford\_Ballman.cpp**

#include "Header.h"

void Ford\_Ballman(int CountApex, int\*\* SourceMatrix, int Start, FILE\* fin, int status)

{

int\* MinPath; // массив минимальных путей

int i, j, k;

MinPath = (int\*)malloc(CountApex \* sizeof(int));

for (i = 0; i < CountApex; i++)

{

MinPath[i] = INF; //все расстояния делаем бесконечными

}

MinPath[Start] = 0;

for (k = 0; k < CountApex; k++)

{

for (i = 0; i < CountApex; ++i)

{

for (j = 0; j < CountApex; ++j)

{

if (SourceMatrix[i][j] != 0) //если вершина не посещена

if (MinPath[j] > MinPath[i] + SourceMatrix[i][j]) {

MinPath[j] = MinPath[i] + SourceMatrix[i][j];

}

}

}

}

for (i = 0; i < CountApex; ++i) {

for (j = 0; j < CountApex; ++j)

{

if (SourceMatrix[i][j] != 0)

if (MinPath[j] > MinPath[i] + SourceMatrix[i][j]) {

printf("Ошибка! Найден отрицательный цикл\n");

return;

}

}

}

if (status == 1)

outputFord(CountApex, MinPath, Start);

if (status == 2)

fileOutput(CountApex, MinPath, Start, fin);

}

**Generation.cpp**

#include "Header.h"

void randomGen(int f, int\*\* mass) {

for (int i = 0; i < f; i++) {

mass[i] = (int\*)malloc(sizeof(int) \* f);

}

for (int i = 0; i < f; i++) {

for (int j = 0; j < f; j++) {

int c = (rand() % 10 - 2);

mass[i][j] = c;

if (i == j) {

mass[i][j] = 0;

}

}

}

for (int i = 0; i < f; i++) {

for (int j = 0; j < f; j++) {

printf("|%d|\t ", mass[i][j]);

}

printf("\n");

}

}

**Menu.cpp**

#include "Header.h"

void Menu() {

printf("\nВвыберите операцию");

printf("\n1) Сгенерировать граф");

printf("\n2) Запустить алгоритм");

printf("\n3) Сохранить результат в файл");

printf("\n4) Выйти\n");

\_getch();

}

**outputFord.cpp**

#include "Header.h"

void outputFord(int count, int\* result, int apex)

{

int i;

printf("Результат : \n");

for (i = 0; i < count; i++)

{

if (result[i] < 10000) {

printf("Минимальное расстояние между вершинами %d и %d = %d\n\n", apex, \

i, result[i]);

}

else {

printf("Расстояний из вершины %d не существует\n", apex, \

i);

}

}

}

**outputMass.cpp**

#include "Header.h"

void output(int\*\* mass, int f) {

for (int i = 0; i < f; i++) {

for (int j = 0; j < f; j++) {

printf("|%d|\t ", mass[i][j]);

}

printf("\n");

}

}

**Title.cpp**

#include "Header.h"

void Title() {

printf("\nКурсова работа \nпо дисциплине");

printf("\nЛогика и основы алгоритмизации в инженерных задачах");

printf("\nНа тему : \"Реализация алгоритма Форда-Беллмана\"");

printf("\nВыполнил студент группы 19ВВ3");

printf("\nСубботкин М.В.");

printf("\n\nПринял:");

printf("\nМитрохин М.А.\n\n");

printf("\nДля проолжения нажмите любую клавишу");

\_getch();

}

**main.cpp**

#include "Header.h"

void main(){

srand(time(NULL));

setlocale(LC\_ALL, "russian");

int f, start, input = 0;

int\*\* mass = 0;

FILE\* fout;

Title();

system("cls");

Menu();

while (input < 4) {

scanf("\n%d", &input);

if (input == 1) {

system("cls");

printf("\nВведите размер графа : ");

scanf\_s("%d", &f);

mass = (int\*\*)malloc(sizeof(int) \* f);

randomGen(f, mass);

\_getch();

system("cls");

Menu();

}

else if (input == 2) {

if (mass == 0) {

system("cls");

printf("Граф не найден!\n");

printf("\nВведите размер графа : ");

scanf\_s("%d", &f);

mass = (int\*\*)malloc(sizeof(int) \* f);

randomGen(f, mass);

}

system("cls");

output(mass, f);

printf("Введите стратовую веришу : ");

scanf("%d", &start);

if (start > f - 1) {

printf("Введена неправильная вершина!");

\_getch();

system("cls");

Menu();

}

else {

Ford\_Ballman(f, mass, start, 0, 1);

\_getch();

system("cls");

Menu();

}

}

if (input == 3) {

char name[] = "data.txt";

if ((fout = fopen(name, "w+")) == NULL) {

printf("Файл не найден!");

\_getch();

}

else {

system("cls");

printf("Файл сохранен\n");

fout = fopen("C:\\new\\data.txt", "w");

for (int i = 0; i < f; i++) {

for (int j = 0; j < f; j++) {

fprintf(fout, "|%d|\t ", mass[i][j]);

if (j == f - 1)

fprintf(fout, "\n");

}

}

Ford\_Ballman(f, mass, start, fout, 2);

printf("\n");

Menu();

}

}

}

system("cls");

system("pause");

}