Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Пояснительная записка**

К курсовому проектированию

По курсу «Логика и основы алгоритмизации

в инженерных задачах»

на тему «Реализация алгоритма Дейкстры»

Выполнил:

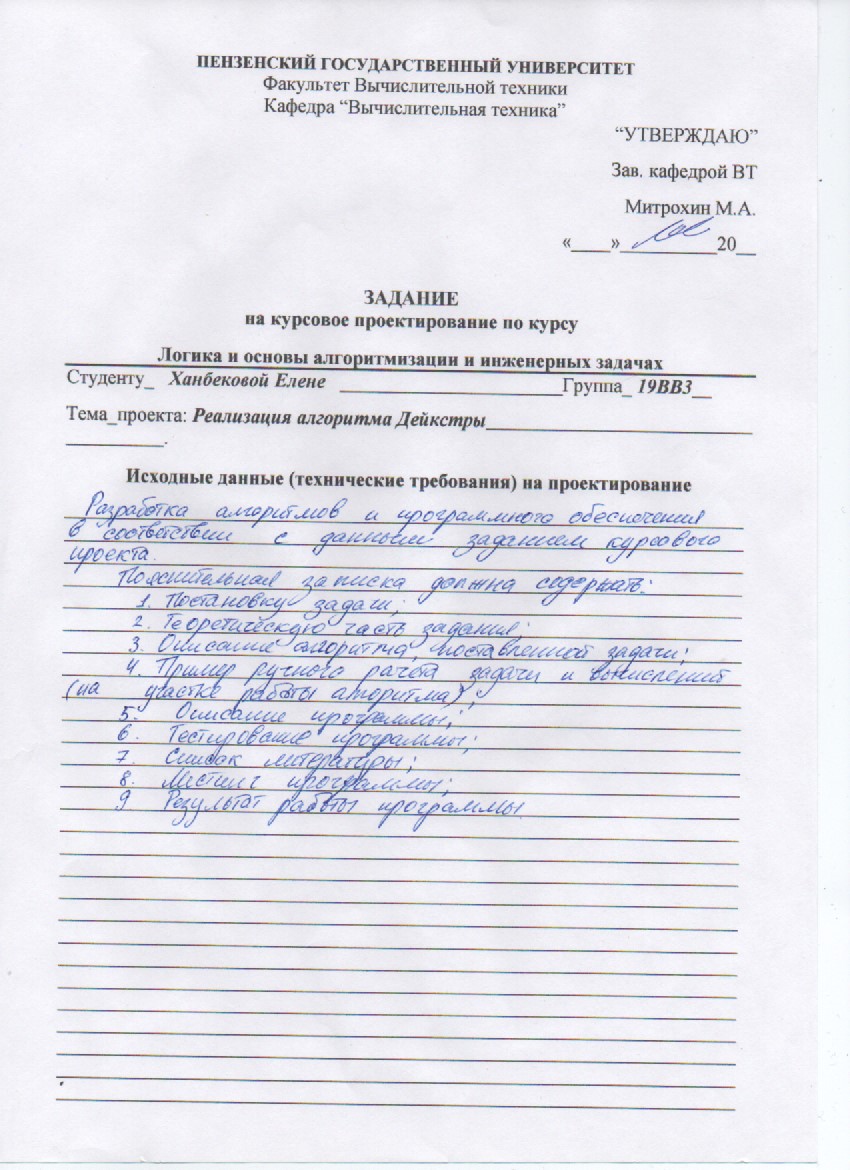
студентка группы 19ВВ3

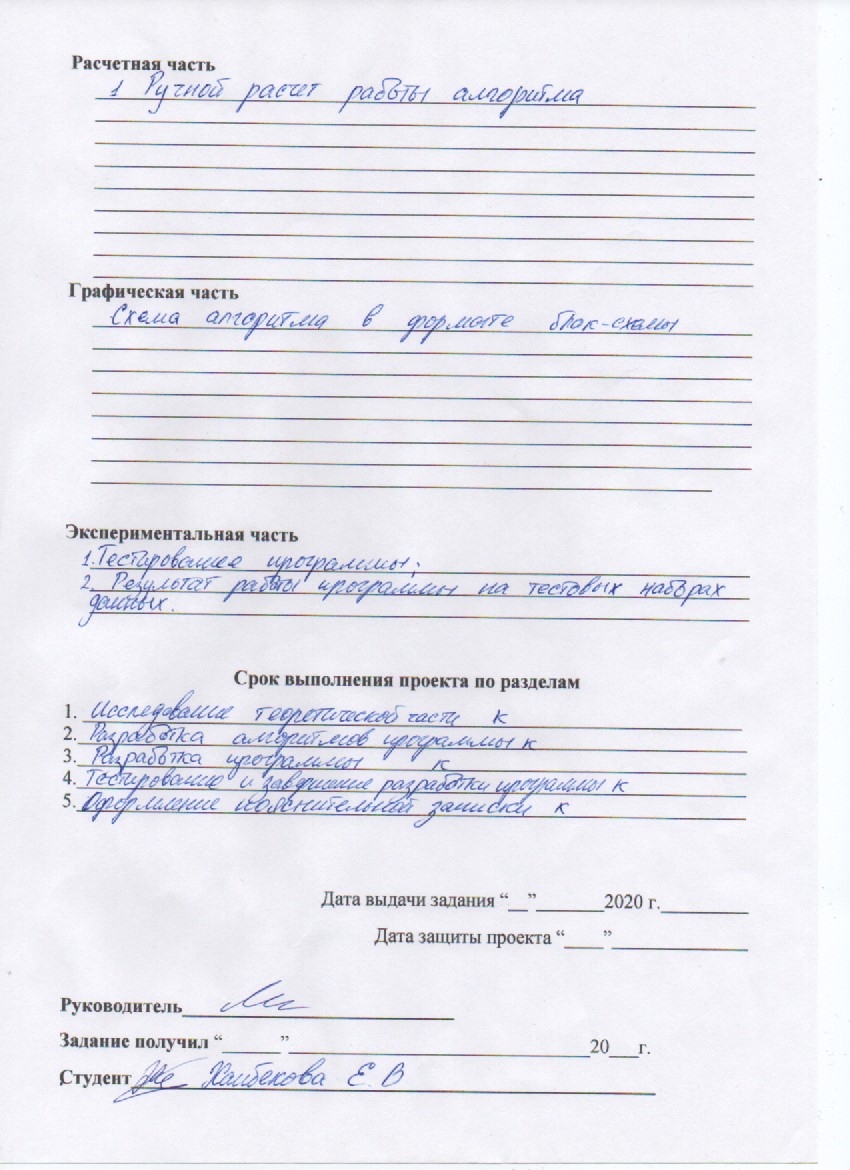
Ханбекова Е. В.

Принял:

Митрохин М.А.

Пенза 2020

****

****

**Содержание**

[Реферат 5](#_Toc59446621)

[Введение 6](#_Toc59446622)

[1. Постановка задачи 7](#_Toc59446623)

[2. Теоретическая часть задания 8](#_Toc59446624)

[3. Описание алгоритма программы 10](#_Toc59446625)

[4. Описание программы 13](#_Toc59446626)

[5. Тестирование 22](#_Toc59446627)

[6. Ручной расчёт задачи 25](#_Toc59446628)

[Заключение 27](#_Toc59446629)

[Список использованных источников 28](#_Toc59446630)

[Приложения 29](#_Toc59446631)

# Реферат

Отчет 34 стр, 16 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ДЕЙКСТРЫ, КРАТЧАЙШИЙ ПУТЬ, ДЛИНА ПУТИ.

Цель исследования – разработка программы, реализующуей алгоритм Дейкстры.

В работе рассмотрен алгоритм Дейкстры. Данный алгоритм находит кратчайшее расстояние между вершинами ориентированного и неориентированного графа.

# Введение

Алгоритм Дейкстры ищет кратчайшие пути, соединяющие исходную вершину графа с остальными вершинам этого графа (если между ними есть связь). Минусом данного метода является невозможность обработки графов, в которых имеются ребра с отрицательным весом.

Каждой вершине сопоставляется метка — минимальное известное расстояние от этой вершины до другой. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены. В противном случае из еще не посещенных вершин выбирается вершина, имеющая минимальную метку. Для каждой соседней вершины рассмотрим новую длину пути, равную сумме текущей метки и длины ребра, соединяющего данную вершину с этим соседней. Если полученная длина меньше метки соседней, заменим метку этой длиной. Рассмотрев все соседние вершины, пометим исходную вершину как посещенную и повторим шаг.

В результате работы алгоритма Дейкстры строится Дерево кратчайших путей.

В качестве среды разработки была выбрана среда MicrosoftVisualStudio2019, языки программирования – C и C++.

Целью данной курсовой работы является разработка программы на языке Си. С его помощью в данном курсовом проекте реализуется алгоритм Дейкстры.

# Постановка задачи

Требуется разработать программу, которая осуществит поиск кратчайшего расстояния между двумя вершинами, использую алгоритм Дейкстры.

Исходный граф в программе должен задаваться матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь выбирал способ создания матрицы смежности (вручную или автоматически, с помощью случайного заполнения) или мог загрузить ее из файла, выбирал тип создаваемого графа (ориентированный или неориентированный), вводил количество вершин для генерации матрицы смежности. На экран должна выводиться матрица смежности графа. Для поиска кратчайшего расстояния между вершинами пользователь должен ввести номер стартовой вершины. Если пользователь в ходе работы программы введет некорректные данные, программа должна вывести на экран соответствующее предупреждение. Необходимо предусмотреть различные входные данные, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

# Теоретическая часть задания

**Теория графов**

**Графом** называется конечное множество вершин и множество ребер. Каждому ребру сопоставлены две вершины – концы ребра.

Бывают различные варианты определения графа. В данном определении концы у каждого ребра – равноправны. В этом случае нет разницы, где начало, а где – конец у ребра. Но, например, в транспортных сетях бывают случаи одностороннего движения по ребру, тогда говорят об ориентированном графе – графе, у ребер которого одна вершина считается начальной, а другая – конечной. Если некоторое ребро u соединяет две вершины A и B графа, то говорят, что ребро u инцидентно вершинам A и B, а вершины в свою очередь инцидентны ребру u. Вершины, соединенные ребром, называются смежными.

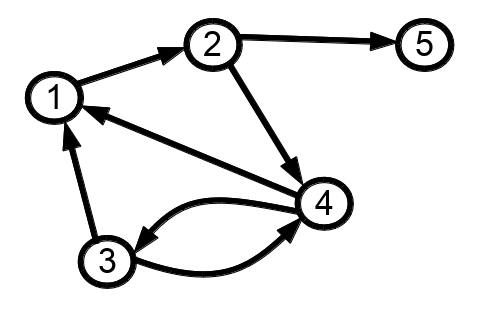


Рисунок 1 – Граф

Ребра называются кратными, если они соединяют одну и ту же пару вершин (а в случае ориентированного графа – если у них совпадают начала и концы). Ребро называется петлей, если у него совпадают начало и конец.

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице (двумерном списке), где элемент A[i][j] равен 1, если ребра i и j соединены ребром и равен 0 в противном случае. Если граф неориентированный, то матрица смежности всегда симметрична относительно главной диагонали. При использовании матрицы смежности удобно проверять соединены ли две вершины ребром – это просмотр одного элемента матрицы A[i][j], но сложнее перебирать все ребра, исходящие из данной вершины (для этого необходимо перебрать все оставшиеся вершины и проверить, соединены ли они ребром).

Очень часто рассматриваются графы, в которых каждому ребру приписана некоторая числовая характеристика – вес. Вес может означать длину дороги или стоимость проезда по данному маршруту. Соответствующие графы называются взвешенными.

При представлении графа матрицей смежности вес ребра можно хранить в матрице, то есть A[i][j] в данном случае будет равно весу ребра из i в j.

**Путем** на графе называется последовательность ребер u1, u2, …, uk, в которой конец одного ребра является началом следующего ребра. Начало первого ребра называется началом пути, конец последнего ребра - концом пути.

# Описание алгоритма программы

Таблица 1 – Описание переменных

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Роль** |
| i, j, l, k | int | Счётчики |
| start | int | Стартовая вершина |
| N | int | Число вершин графа |
| p | int | Номер постоянной вершины |
| komanda, Selection, kom | int | Номер команды меню |
| vis | int | Динамический массив, содержащий посещенные вершины |
| A | int | Массив, содержащий матрицу смежности |
| DIS | int | Массив, содержащий всю длину пути |
| way[100][100] | char | Массив, содержащий все пути из стартовой вершины |

**Описание работы функции DIJKSTRA()**

Алгоритм начинается с того, что всем элементам массива vis присваивается значение 0, т.е. все вершины помечаются, как не посещённые. Также расстояние до каждой вершины становится бесконечным, равным 8000, которое означает бесконечность и превосходит все возможные длины путей. Начальная вершина помечается как уже посещенная (vis [start] = 1) и общее расстояние (DIS[start]) становится равным 0, так как мы еще не начали перебирать остальные вершины. Затем начинается основной цикл. Если встречается вершина i, расстояние из вершины p до которой будет меньше, чем известное ранее расстояние из начальной вершины в i (DIS[i] > DIS[p] + A[p][i]), тогда это расстояние DIS[p] + A[p][i] присваивается DIS[p]. После чего переменной p присваивается значение i. Этой вершине присваивается значение посещенной vis[р] = 1. Алгоритм работает до тех пор, пока все вершины не будут посещены.

Ниже представлена схема части функции Dijkstra()

Схема 1 - Алгоритм Дейкстры

k = N

DIS[start] = 0 vis[start] = 1

k = 1,p = start

\_itoa(start, way[start], 10)

i = 0

Начало

i < N

(A[p][i] != 0) && (!vis[i]) && (i != p)&& (DIS[i] > DIS[p] + A[p][i])

DIS[i] = DIS[p] + A[p][i]

strcat(way[i], way[p])

min = 8000

i = 0

i < N

Конец

i++

нет

нет

нет

да

да

да

min = i

(DIS[i] < min) && (vis[i] = 0)

p = min

vis[p] = 1

k++

да

да

нет

i++

# Описание программы

Для написания данной программы использован язык программирования Си.

Проект был создан в виде консольного приложения Visual C++ .

Данная программа состоит из нескольких функций: main(), Title(), Dijkstra (), Menu(), Saverez(), Save(), Handvvod(), Avtovvod(), Loadvvod().

Работа программы начинается с заглавного экрана, содержащего информацию о работе. Данные выводится с помощью функции Title(), код которой представлен ниже.

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

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

printf("\nНа тему: \"Реализация алгоритма Дейкстры\"");

printf("\n\nВыполнила студентка группы 19ВВ3");

printf("\nХанбекова Е. В.");

printf("\n\nПринял:");

printf("\nМитрохин М.А.\n\n");

printf("\nНажмите любую клавишу для продолжения");

\_getch();

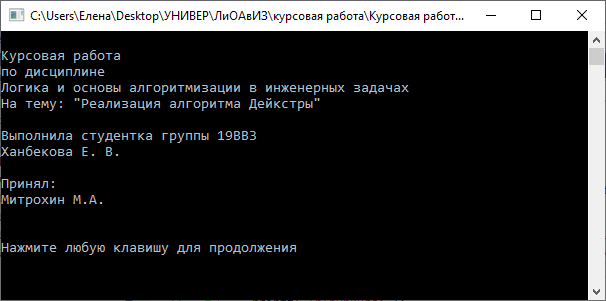


Рисунок 2 – Заглавный экран

Для продолжения работы нужно нажать любую клавишу.

Затем открывается меню программы, которое предлагает выбрать способ задания графа (вручную, автоматически, загрузить из файла). Меню программы реализовано в функции Menu(), код которой представлен ниже.

int komanda = 0;

system("cls");

printf("1| Ручной ввод графа ");

printf("\n2| Автоматический ввод графа"); printf("\n3| Загрузить граф из файла");

printf("\n0| Завершение работы\n\n");

printf("Ведите номер команды ");

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

system("cls");

return(komanda);

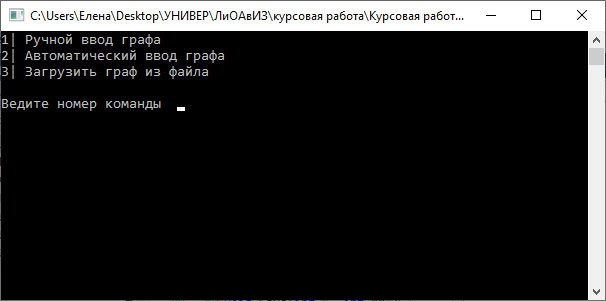


Рисунок 3 – Главное меню

При выборе первого пункта, пользователь попадет в меню ручного ввода матрицы. Данный пункт меню реализован в функции Handvvod(). Внутри данной функции с помощью меню пользователь так же выбирает задать ориентированный или неориентированный граф. Часть кода, отвечающая за создание матрицы, представлена ниже.

int komanda = 0, i, j;

printf("1| Ориентированный граф");

printf("\n2| Неориентированный граф");

printf("\n\nВедите номер команды ");

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

while ((komanda < 1) || (komanda > 2)) {

printf("\n\nНеверно введена команда. Попробуйте еще раз. ");

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

}

switch (komanda) {

case 1:

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

for (j = 0; j < N; j++) {

if (i != j) {

printf("Расстояние от %d до %d ", i + 1, j + 1);

scanf\_s("%d", &A[i][j]);

while (A[i][j] < 0) {

printf("\nТолько положительные числа! Введите другое число ");

scanf\_s("%d", &A[i][j]);

}

}

else

A[i][j] = 0;

}

}

break;

case 2:

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

for (j = 0; j < N; j++) {

if (i == j) {

A[i][j] = 0;

}

if (i < j) {

printf("Расстояние от %d до %d ", i + 1, j + 1);

scanf\_s("%d", &A[i][j]);

while (A[i][j] < 0) {

printf("\nТолько положительные числа! Введите другое ");

scanf\_s("%d", &A[i][j]);

}

A[j][i] = A[i][j];

}

}

}

break;

}

printf("\n\n ");

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

printf("%d ", i + 1);

}

printf("\n\n");

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

printf("%d| ", i + 1);

for (j = 0; j < N; j++) {

printf("%d ", A[i][j]);

}

printf("\n");

}

printf("\nДля сохранения графа нажмите 1 ");

char kom;

kom = getch();

if (kom == '1') {

Save(N, A);

}

return(A);

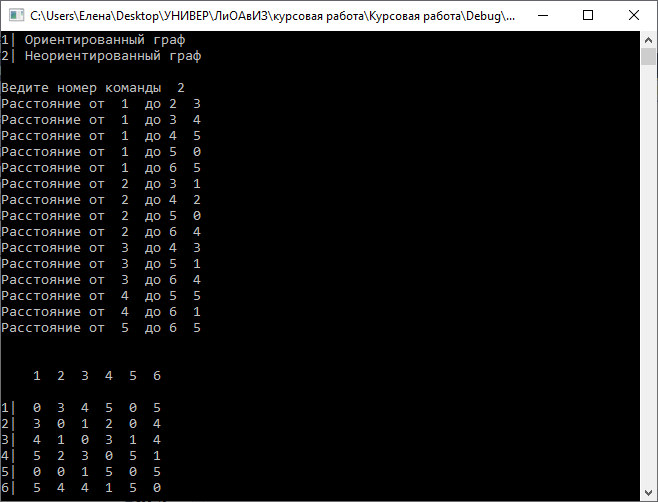


Рисунок 4 – Ручной ввод графа(неориентированного)

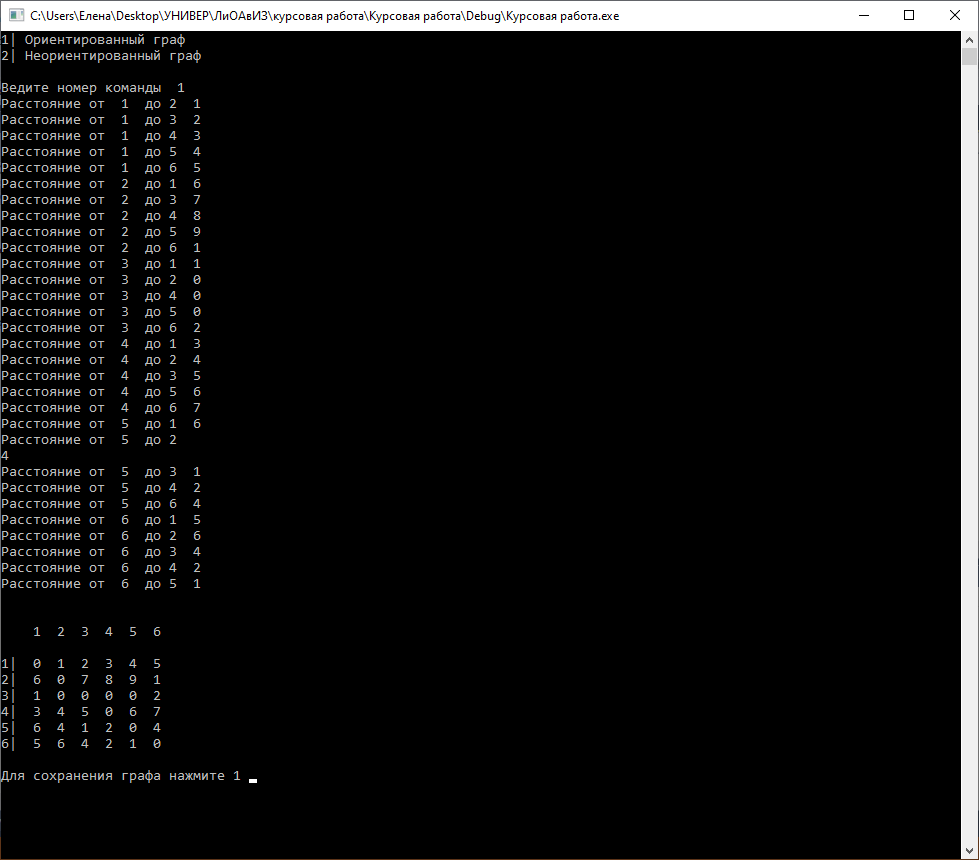


Рисунок 5 – Ручной ввод графа(ориентированного)

При выборе второго пункта, пользователь попадет в меню автоматического создания матрицы. Данный пункт меню реализован в функции Avtovvod(), часть кода, отвечающая за создание матрицы, представлена ниже.

int komanda = 0, i, j;

printf("1| Ориентированный граф");

printf("\n2| Неориентированный граф");

printf("\n\nВедите номер команды ");

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

while ((komanda > 2) || (komanda < 1)) {

printf("\n\nНеверно введена команда. Попробуйте еще раз. ");

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

}

switch (komanda) {

case 1:

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

for (j = 0; j < N; j++) {

A[i][j] = rand() % 5;

if (i == j) {

A[i][j] = 0;

}

}

}

break;

case 2:

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

for (j = 0; j < N; j++) {

A[i][j] = rand() % 6;

A[j][i] = A[i][j];

if (i == j) {

A[i][j] = 0;

}

}

}

break;

}

printf("\n\n ");

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

printf("%d ", i + 1);

}

printf("\n\n");

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

printf("%d| ", i + 1);

for (j = 0; j < N; j++) {

printf("%d ", A[i][j]);

}

printf("\n");

}

printf("\nДля сохранения графа введите 1 ");

char kom;

kom = getch();

if (kom == '1') {

Save(N, A);

}

return(A);

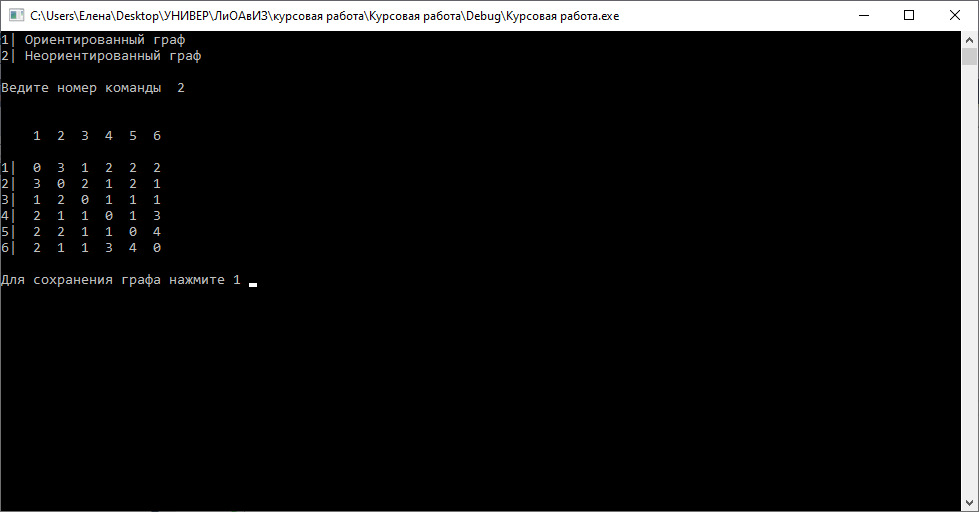


Рисунок 6 - Автоматическое создание графа

Третий пункт меню нужно выбрать, если пользователь желает загрузить матрицу из файла. Данный пункт меню реализован в функции Loadvvod().

Часть кода отвечающего за создание матрицы представлена ниже.

FILE \*F;

int i, j;

char nazvanie[10];

printf("\nВедите название файла: ");

scanf("%s", nazvanie);

if ((F = fopen(nazvanie, "r")) == NULL)

{

fprintf(stderr, "Невозможно открыть для чтения файл\n Нажмите любую клавишу для выхода\n");

\_getch();

return 0;

}

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

for (j = 0; j < N; j++) {

fscanf(F, "%d", &A[i][j]);

}

}

fclose(F);

if (A[N - 1][N - 1] < 0) {

printf("Количество вершин было введено неверно. Для выхода нажмите любую клавишу ");

\_getch();

err = 1;

return 0;

}

printf("\n\n ");

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

printf("%d ", i + 1);

}

printf("\n\n");

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

printf("%d| ", i + 1);

for (j = 0; j < N; j++) {

printf("%d ", A[i][j]);

}

printf("\n");

}

return(A);

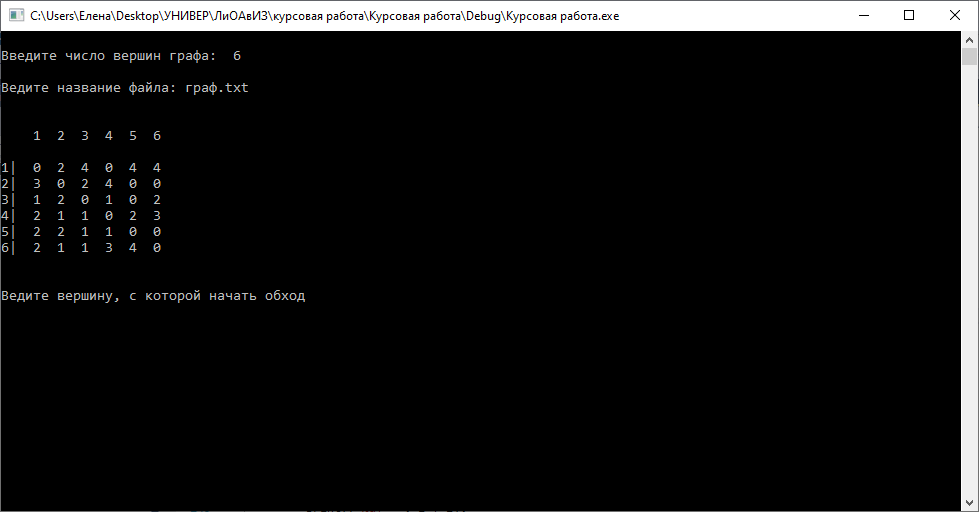


Рисунок 7 - создание графа из файла

Выбор последнего пункта меню приведет к завершению работы.

После вывода матрицы на экран появляется предложение сохранить сгенерированную матрицу в файл. Если пользователь хочет это сделать, то требуется ввести «1». В противном случае можно написать любой другой символ. За сохранением в файл матрицы отвечает функция Save(). Ее код представлен ниже.

char nazvanie[10];

printf("Ведите название файла: ");

scanf("%s", nazvanie);

FILE \*F;

if ((F = fopen(nazvanie, "w+")) == NULL)

{

fprintf(stderr, "Невозможно открыть для записи файл\n");

return;

}

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

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

fprintf(F, "%d\n", A[i][j]);

}

}

fclose(F);

Далее пользователю нужно будет ввести стартовую вершину. Если начальная вершина больше чем количество вершин в матрице, то программа выведет соответсвующее собщение и попросит ввести ее заново.

int start;

printf("\n\nВедите вершину, с которой начать обход ");

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

while (start > N) {

printf("\n Ошибка. Введите вершину заново ");

scanf("%d", &start);

}

В ином случае программа, используя алгоритм Дейкстры выведет кратчайшее расстояние между вершинами и путь, который нужно пройти, чтобы получить такой результат. Код функции, выполняющей алгоритм Дейкстры представлен ниже.

int i;

start--;

DIS[start] = 0;

vis[start] = 1;

int k = 1;

p = start;

\_itoa(start + 1, way[start], 10);// для пути в каждую вершину ставим в начало стартовую

do {

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

if ((A[p][i] != 0) && (vis[i] == 0) && (i != p) && (DIS[i] > DIS[p] + A[p][i])) {

\_itoa(i + 1, post, 10);

strcpy(way[i], way[p]);

strcat(way[i], " -> ");

strcat(way[i], post);

DIS[i] = DIS[p] + A[p][i];

}

}

int min = 8000;

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

if ((DIS[i] < min) && (vis[i] == 0)) {

min = i;

}

p = min;

vis[p] = 1;

k++;

} while (k != N);

Если пути не существует, то программа уведомит пользователя о невозможности попадания из начальной вершины в конечную.

Далее программа также предложит сохранить уже результат работы алгоритма в файл. Если пользователь хочет это сделать, то нужно ввести «1». Код представлен ниже.

printf("\nДля сохранения результата нажмите 1 ");

char kom;

kom = getch();

if (kom == '1') {

Saverez(DIS);

}

void Saverez(int\*DIS) {

char nazvanie[10];

printf("\nВедите название файла: ");

scanf("%s", nazvanie);

FILE \*F;

if ((F = fopen(nazvanie, "a+")) == NULL)

{

fprintf(stderr, "Невозможно открыть для записи файл\n");

return;

}

for (p = 0; p < N; p++) {

if (start - 1 != p) {

if (DIS[p] != 8000) {

fprintf(F, "\n Путь: %s\n", way[p]);

fprintf(F, " Расстояние = %d \n", DIS[p]);

}

else

fprintf(F, "\n Пути из %d в %d нет\n\n ", start + 1, p + 1);

}

}

fclose(F);

}

В противном случае можно написать любой другой символ. После чего программа вновь выведет главное меню.

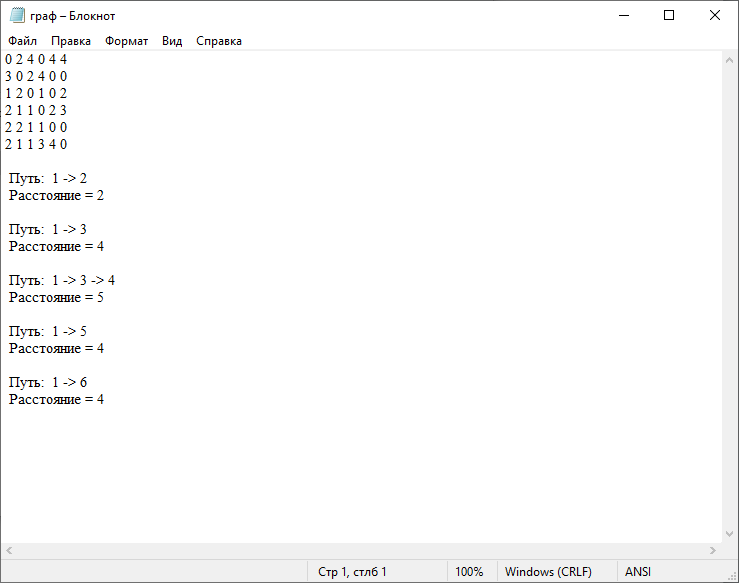
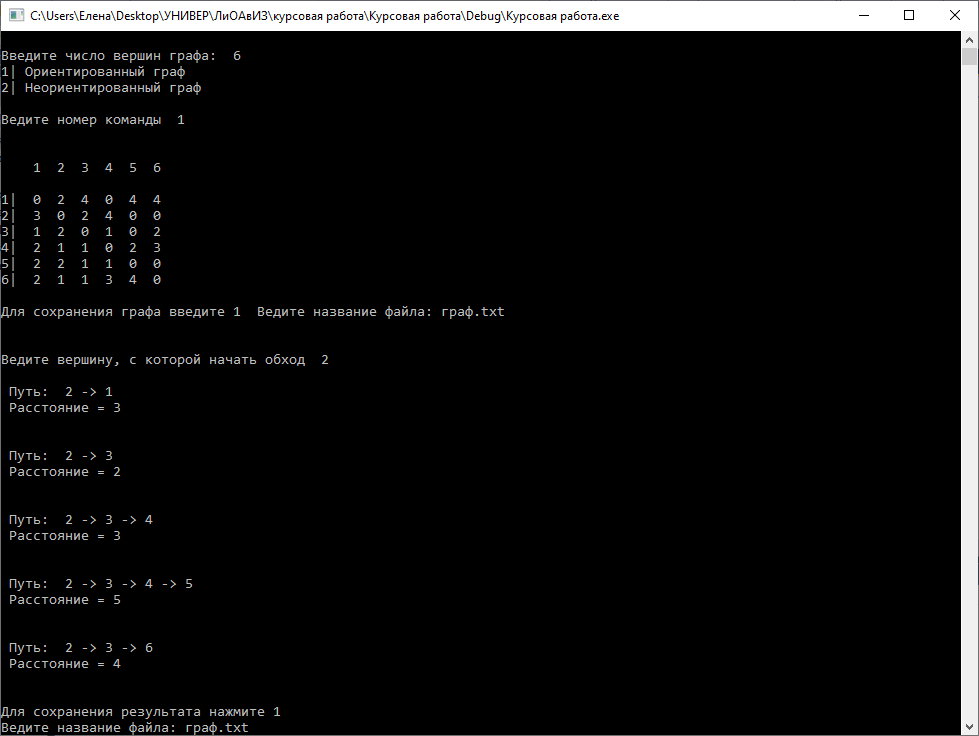


Рисунок 8 – Результат работы алгоритма и сохраненные данные в файле

# Тестирование

Таблица 2 – План тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Ожидаемый результат | Выполненное действие | Полученный результат |
| Запуск программы | Вывод начального сообщения, содержащего краткую информацию о работе | Запустить файл программы тип .exe | Верно |
| Основное меню | Вывод меню с методами создания графа | Ввод размера матрицы | Верно |
| Создание с помощью функции автоматического заполнения ориентированного графа с 6 вершинами | Вывод ориентированного графа размера 6 на 6, который был создан с помощью функции автоматического заполнения | В главном меню выбрать «Автоматический ввод графа».  В открывшемся меню выбрать ориентированный тип графа. | Верно |
| Создание с помощью функции ручного заполнения неориентированного графа с 6 вершинами | Вывод ориентированного графа размера 6 на 6, который был создан с помощью ручного заполнения | В главном меню выбрать «Ручной ввод графа».  В открывшемся меню выбрать неориентированный тип граф. Ввести веса ребер. | Верно |
| Создание графа с помощью загрузки из файла | Вывод графа, построенного с помощью значений, содержащихся в файле. | В главном меню выбрать «Загрузить граф из файла».  Ввести количество вершин графа и название файла. | Верно |
| Сохранение созданного графа в файл | Обнаружение созданного графа в файле | После вывода матрицы смежности программой будет предложено сохранить матрицу в файл. Чтобы сохранить ее, ввести «1». | Верно |
| Поиск кратчайшего расстояния между вершинами | Вывод порядка обхода вершин и кратчайшего расстояния между вершинами. Результат должен совпасть с результатом ручного просчета. | Ввести стартовую вершины. | Верно |
| Сохранение результата поиска в файл | Обнаружение полученного результата в файле | После вывода результата программой будет предложено сохранить его в файл. Чтобы сохранить, ввести «1». | Верно |
| Выход из программы | Завершение работы программы | В главном меню выбрать «Завершение работы». | Верно |

Среда разработки Microsoft Visual Studio 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 исправлены проблемы, связанных с вводом данных, алгоритмом программы, взаимодействием функций, а также с правильностью и неправильностью выводимых данных.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных данных.

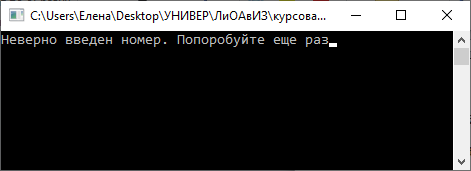
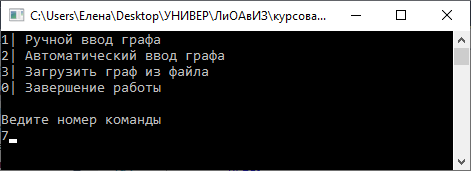


Рисунок 9 – тестирование меню создания матрицы при неправильном вводе команды

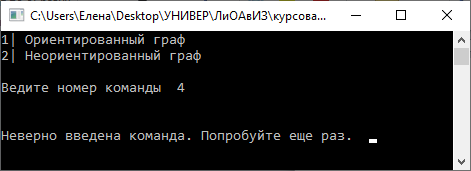


Рисунок 10 – тестирование меню вида графа при неправильном вводе команды

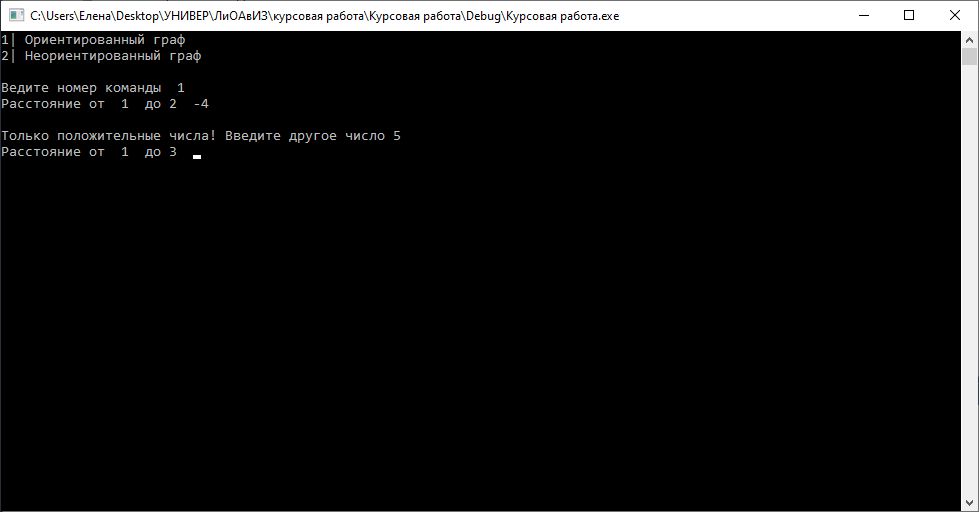


Рисунок 11 - тестирование при вводе вводе отрицательного веса ребра.

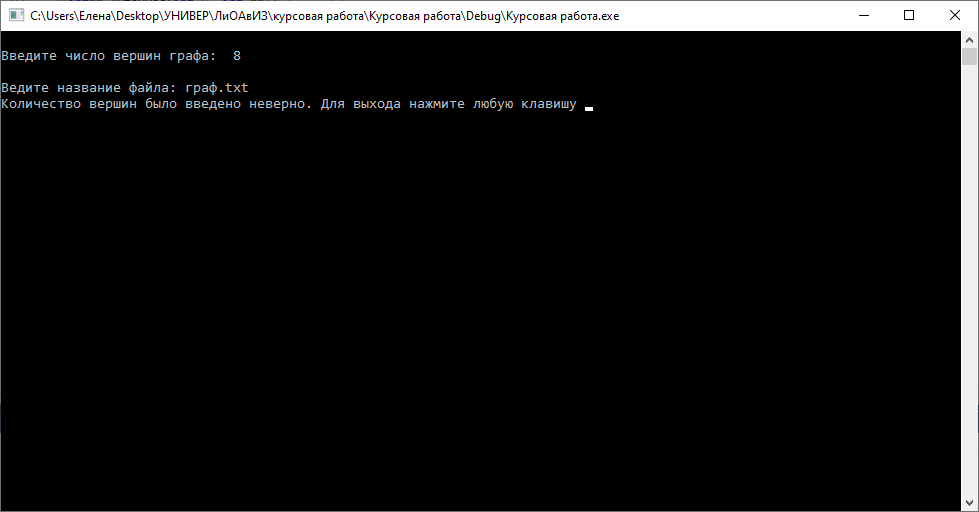


Рисунок 12 – тестирование ввода неправильного количества вершин при загрузке из файла

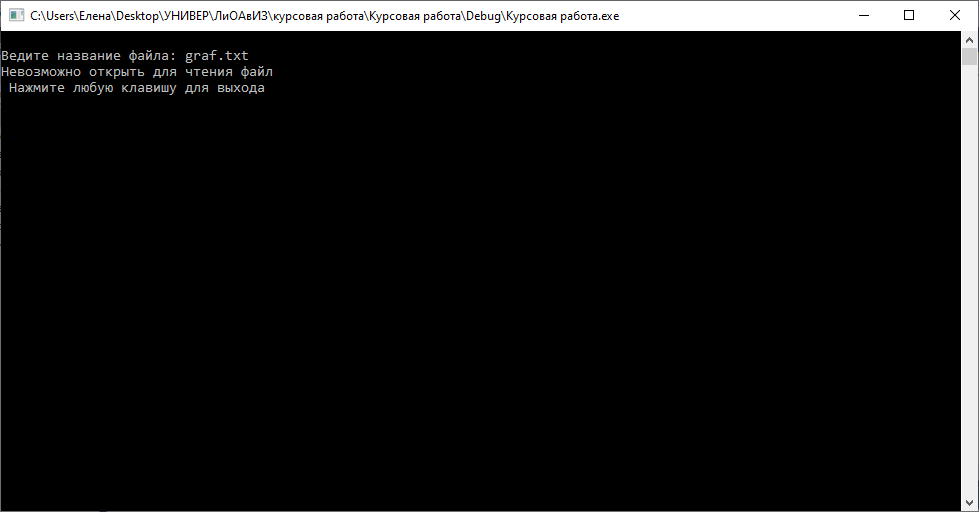


Рисунок 13 – тестирование неправильного ввода названия файла

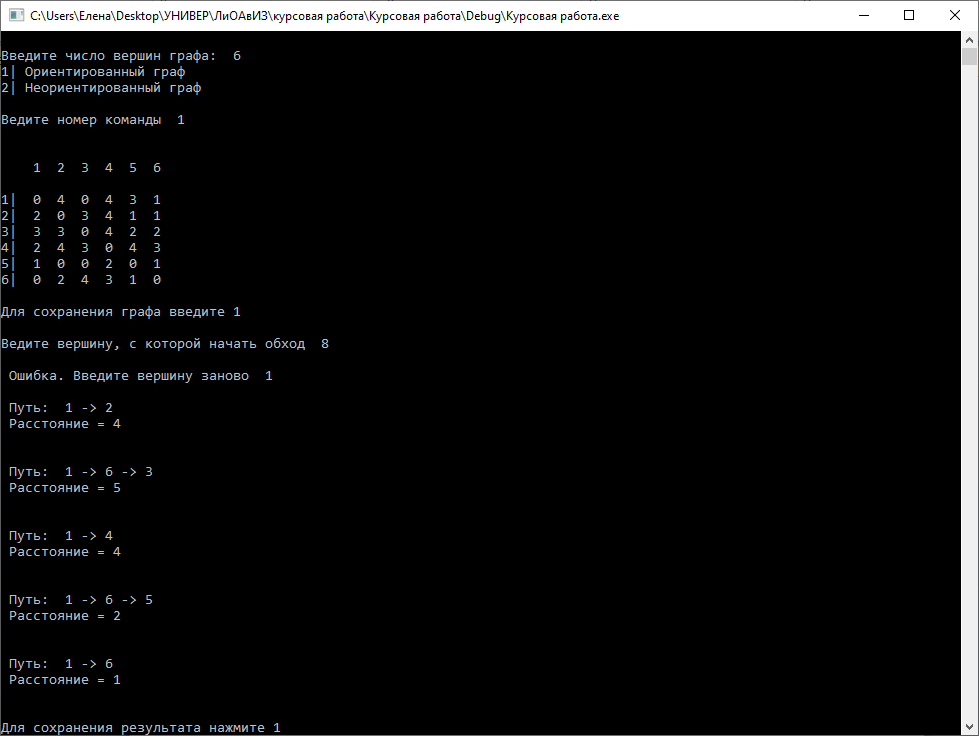


Рисунок 14 – тестирование неправильного ввода стартовой вершины

# Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере ориентированного графа с 6 вершинами. Граф создадим автоматически.

Начальная вершина – 1.

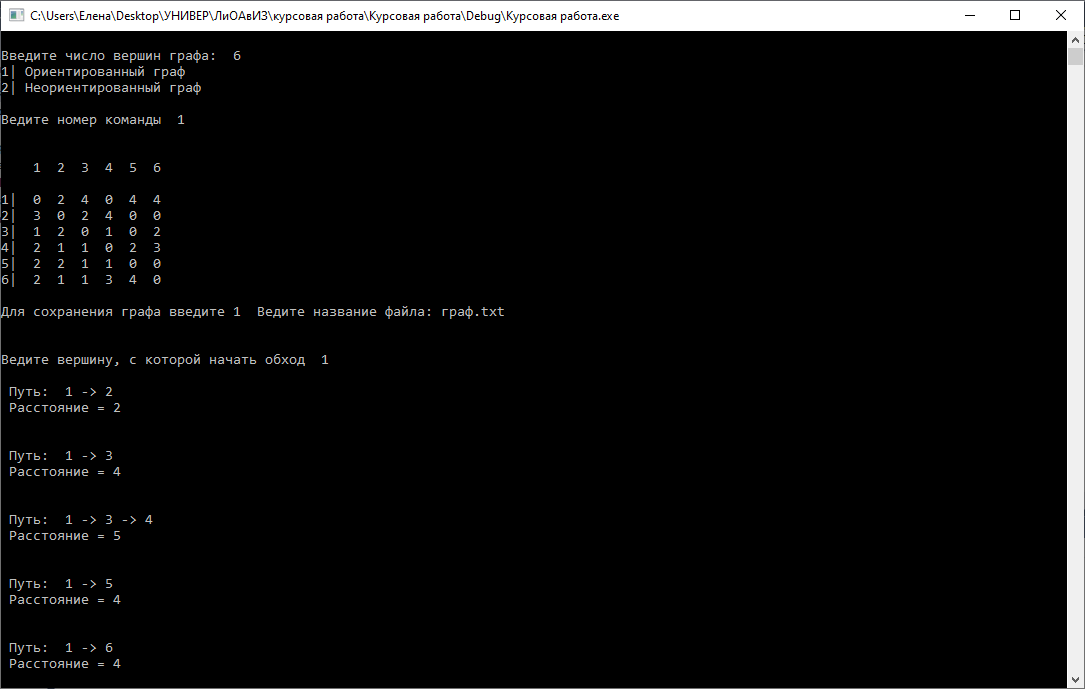


Рисунок 15 – Пример матрицы для ручного расчета

В ручном просчете постоянную вершину помечаем скобками. В скобках находится длина кратчайшего пути до вершины.

Таблица 3 –ручной расчет

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вершина 1 | Вершина 2 | Вершина 3 | Вершина 4 | Вершина 5 | Вершина 6 |
| (0) | 2 | 4 | 8000 | 4 | 4 |
|  | (2) |  | 2 + 4 = 6 |  |  |
|  |  | (4) | 4 + 1 = 5 |  |  |
|  |  |  |  | (4) |  |
|  |  |  |  |  | (4) |
|  |  |  | (5) |  |  |

Результаты работы программы сошлись с результатами ручного просчета.

Расчет второго тестового случая проведем с помощью приложения graphonline.ru.

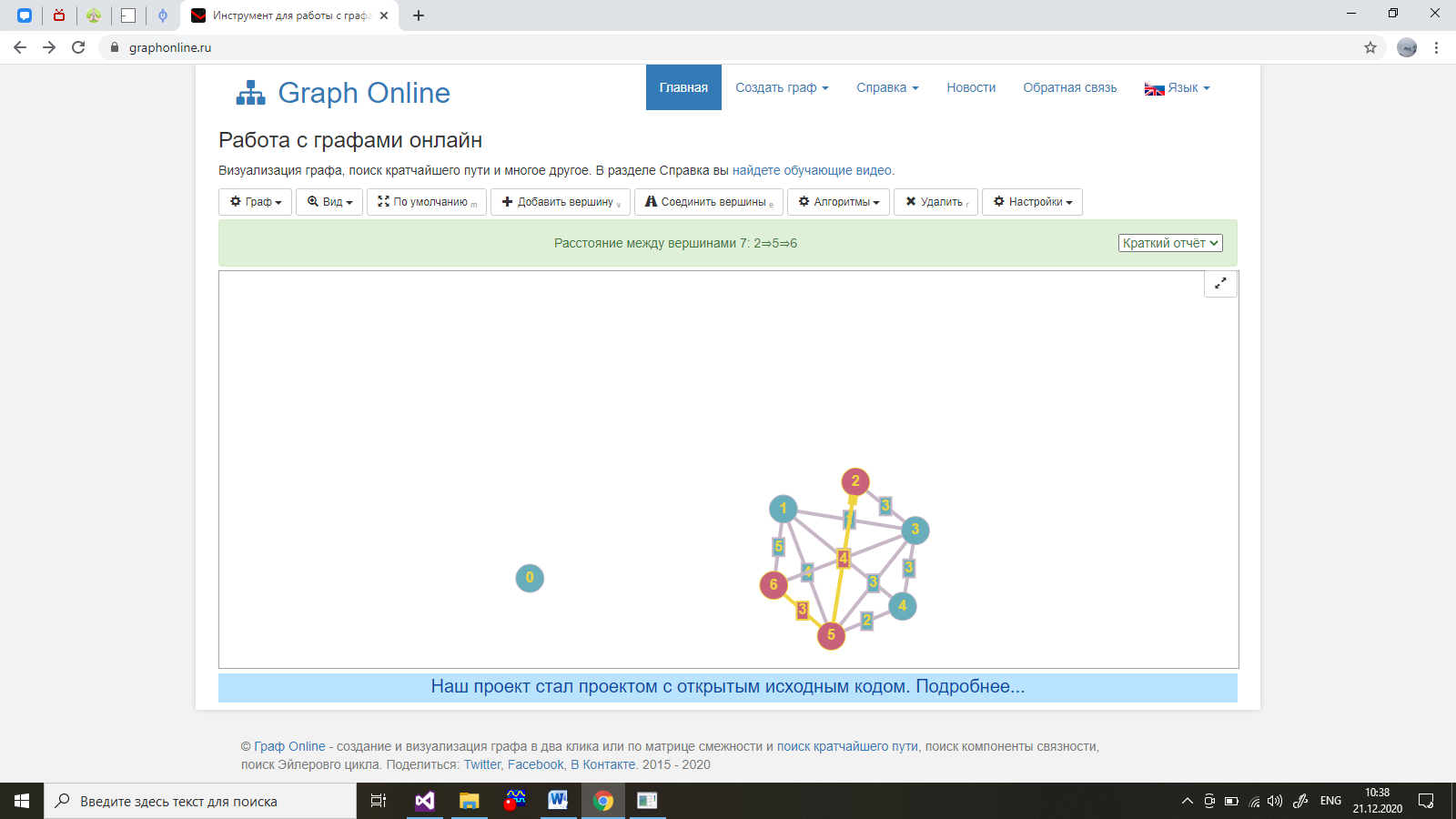
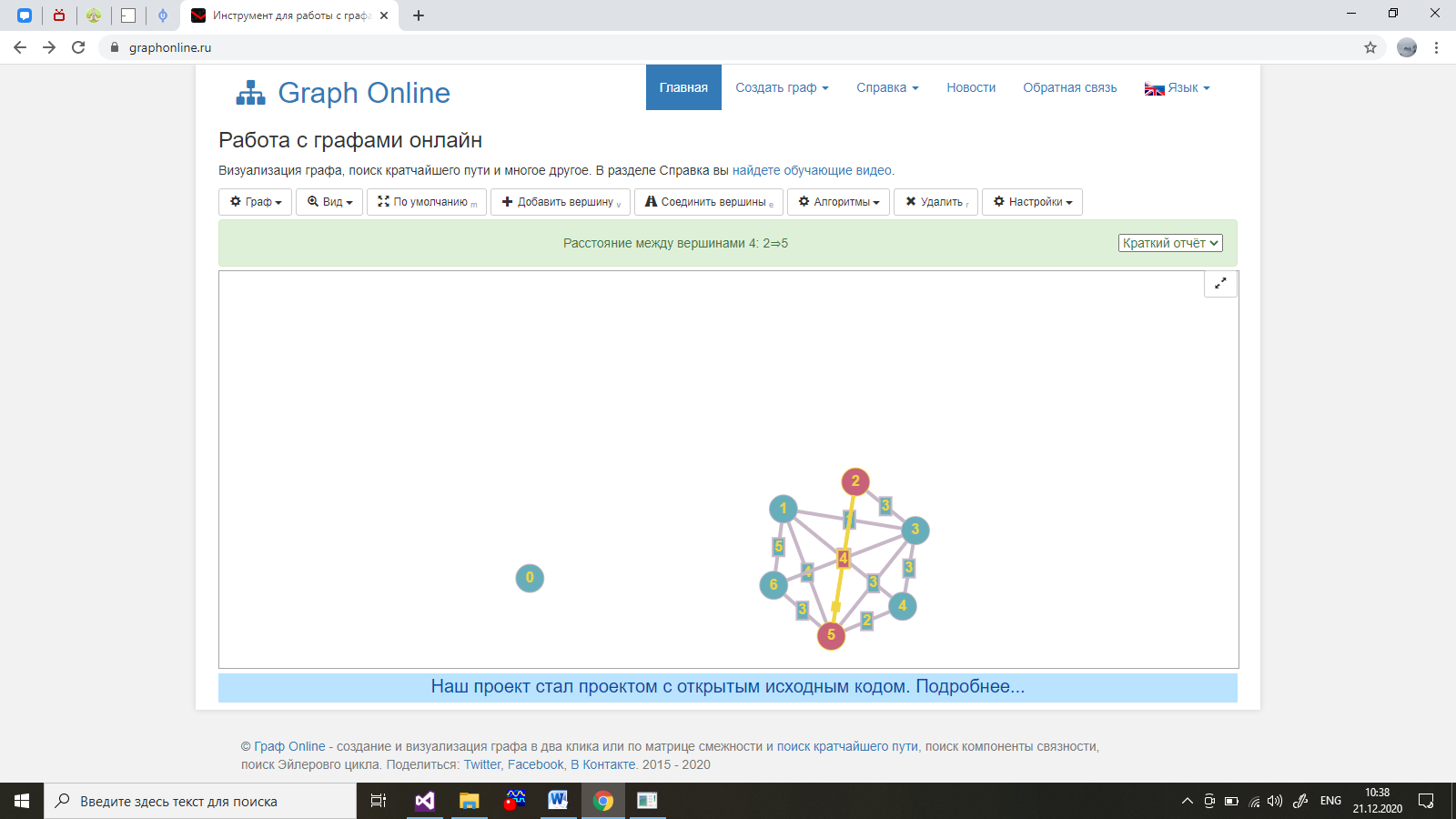
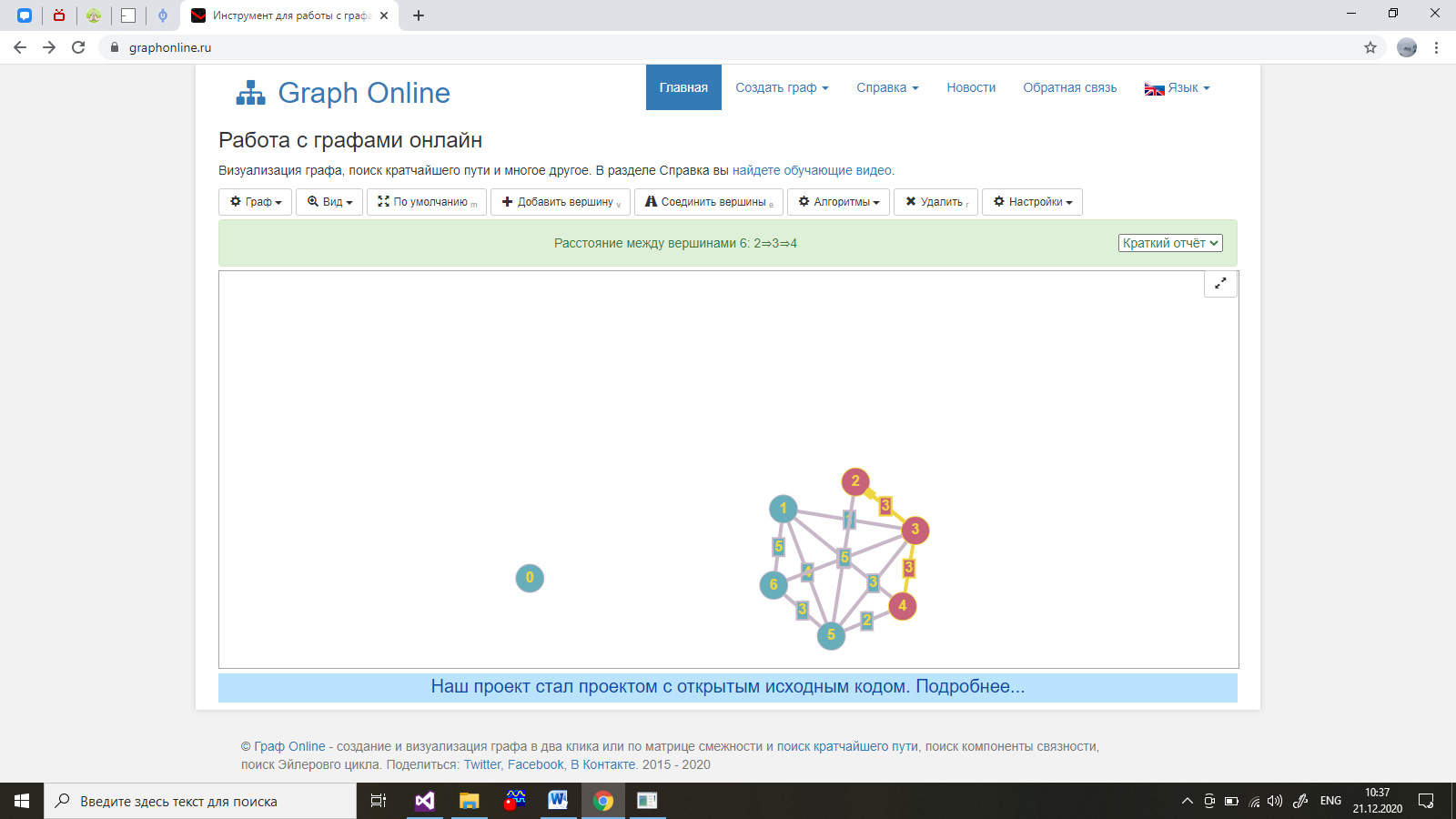
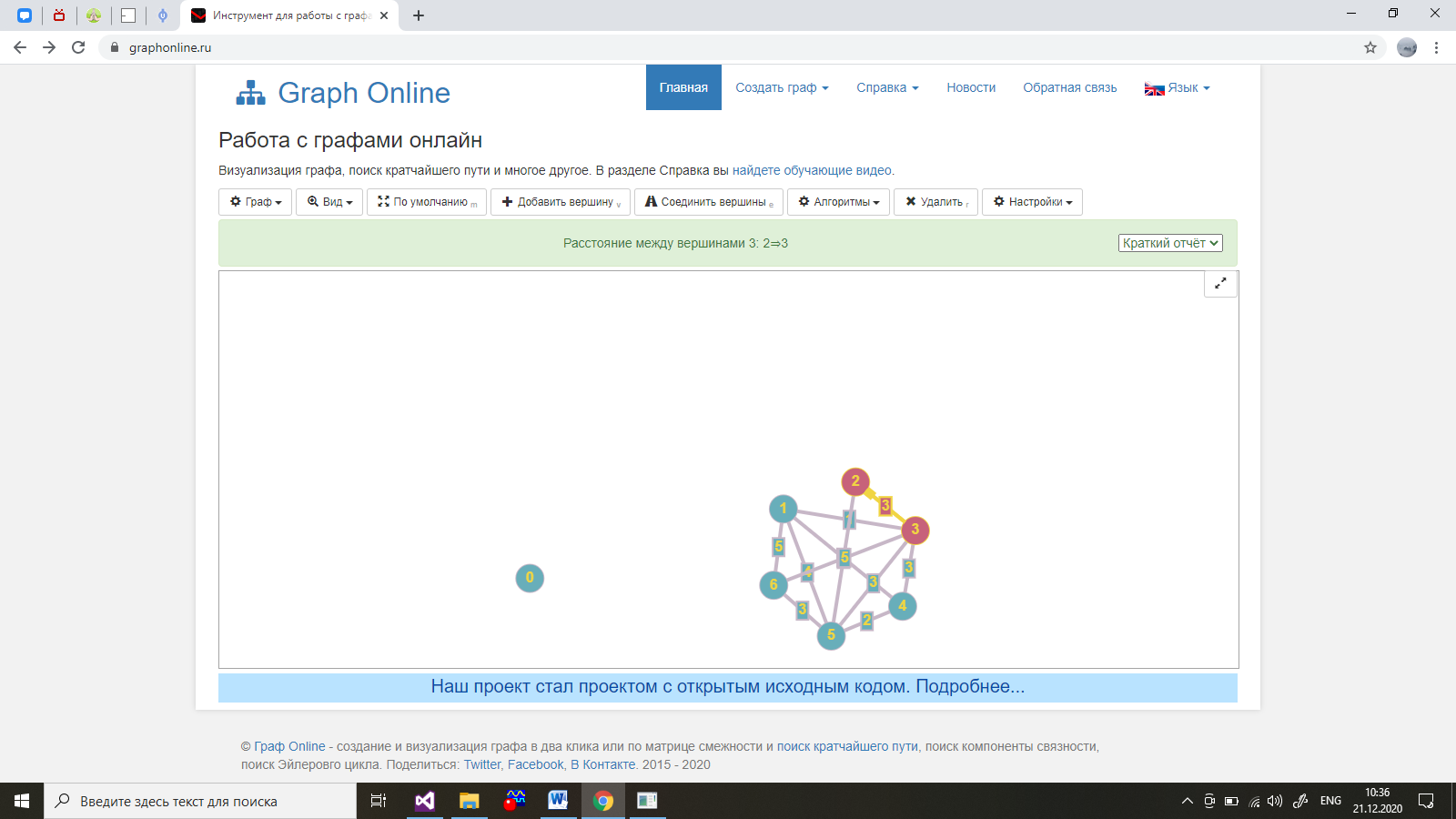
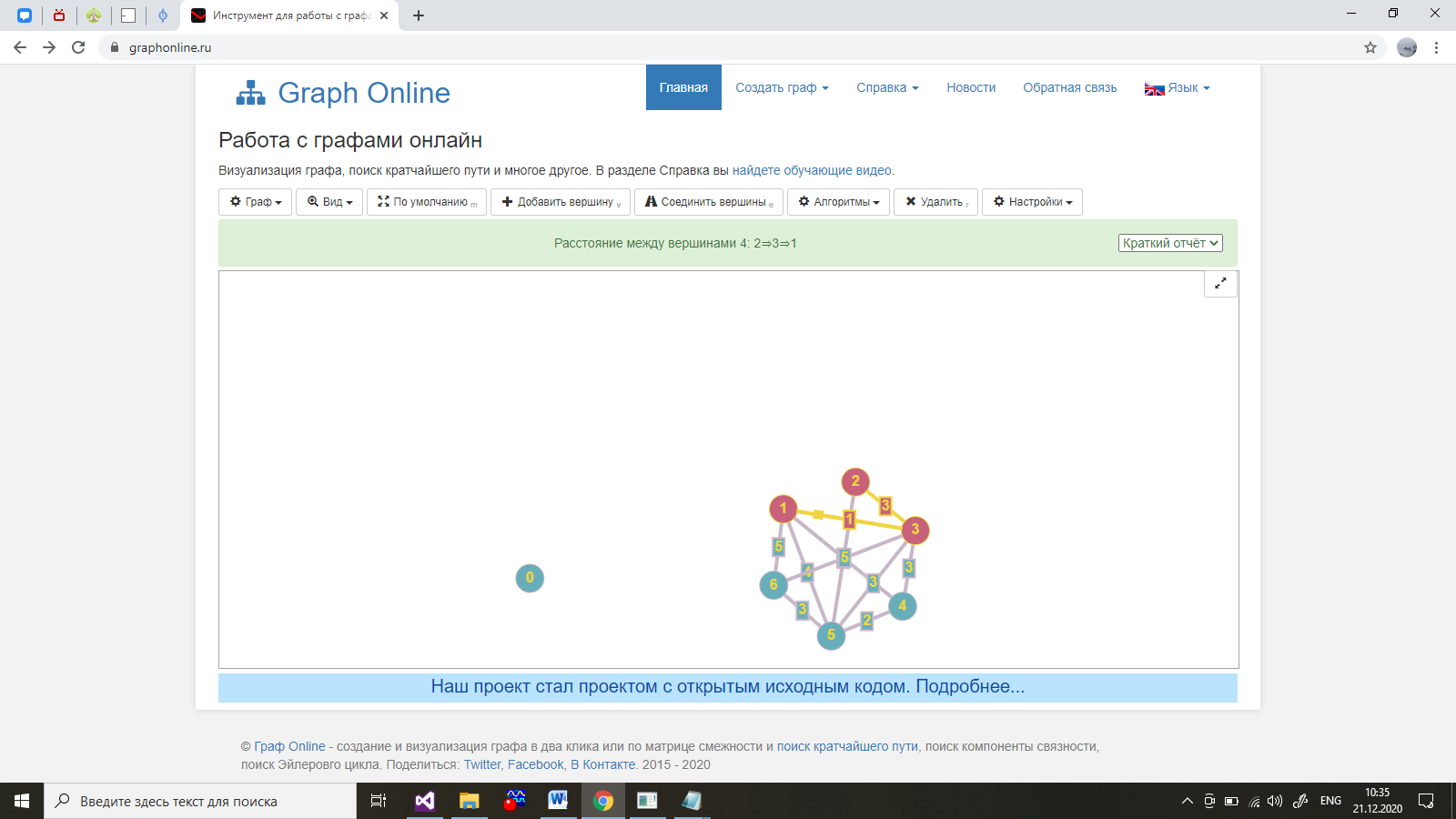
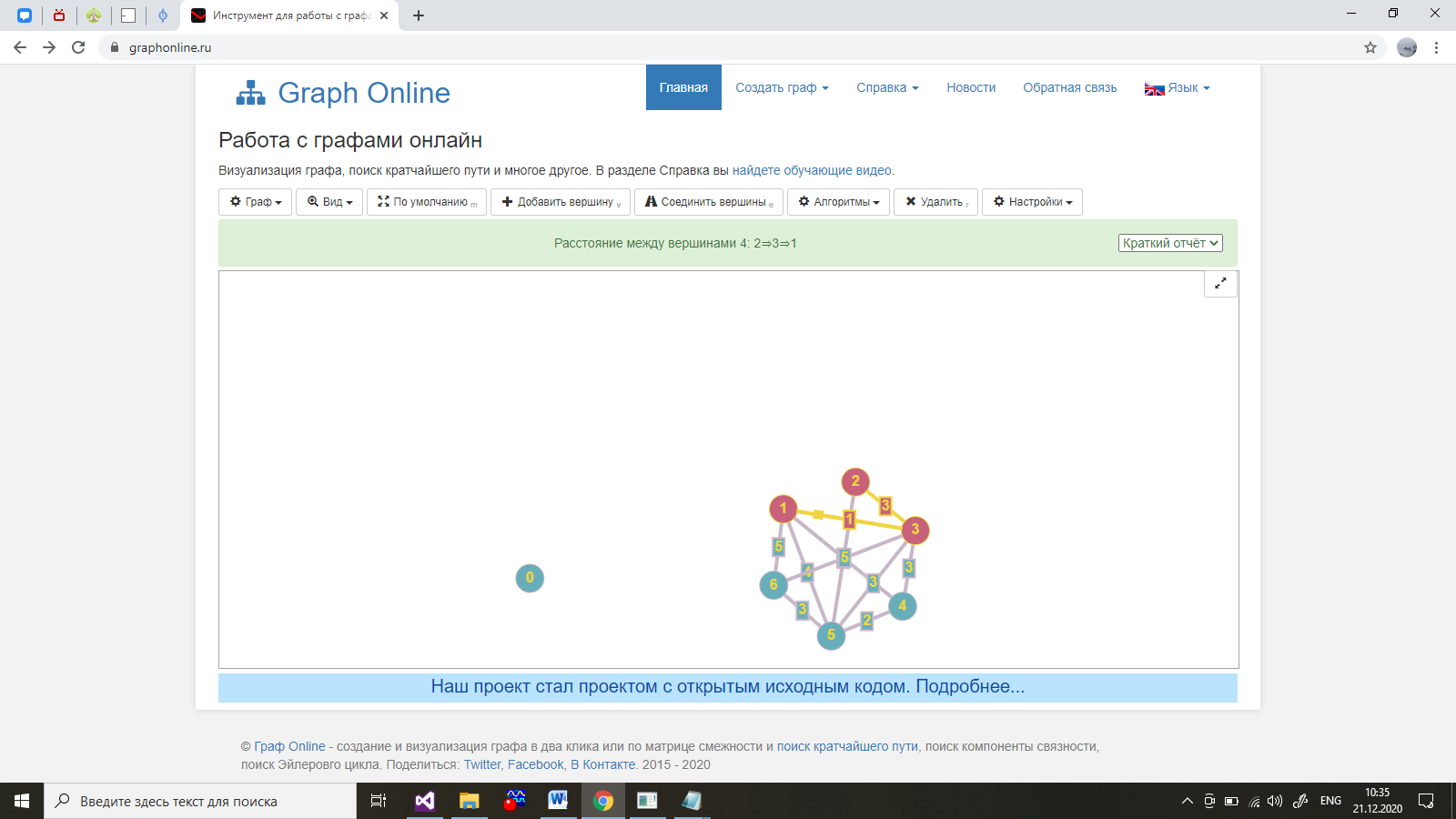
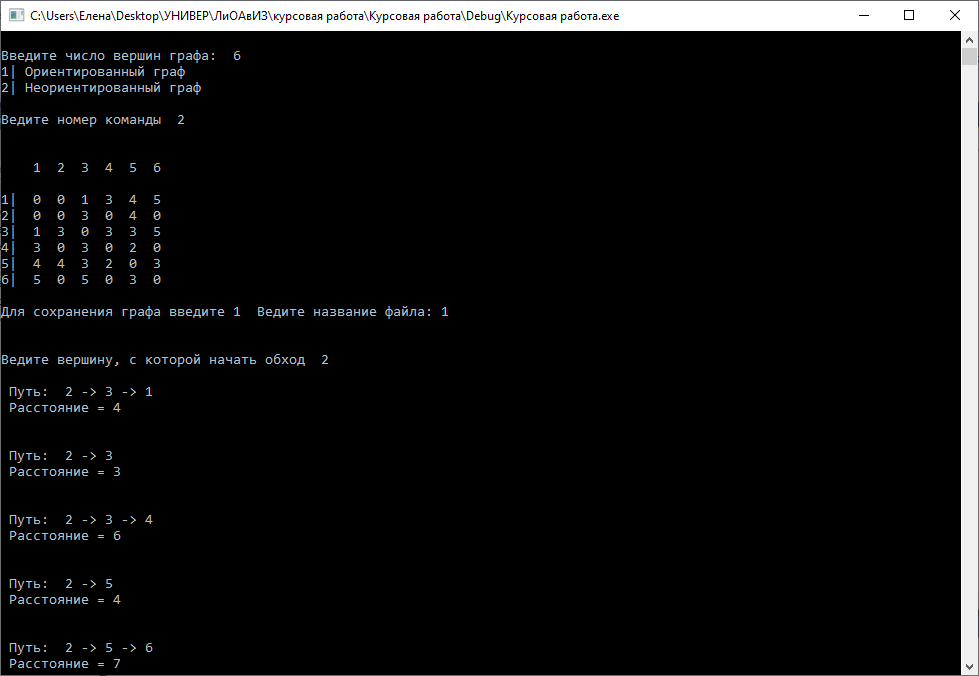


Рисунок 16 – Пример для расчета

Расчет совпал с результатами программы.

# Заключение

В процессе создания данного курсового проекта разработана программа, реализующая алгоритм Дейкстры для поиска кратчайшего пути из одной вершины в другую в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории графов. Углублены знания языков программирования Cи и Си++.

# Список использованных источников

1. https://foxford.ru/wiki/informatika/algoritm-deykstry
2. https://e-maxx.ru/algo/dijkstra
3. http://comp-science.narod.ru/KPG/Deikstr.htm
4. https://graphonline.ru/

## Приложения

Приложение 1. Листинг программы

#define \_CRT\_NONSTDC\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <conio.h>

#include <Windows.h>

#include <stdlib.h>

#include <queue>

#include <iostream>

char post[100], way[100][100];

int p, err;

void Dijkstra(int\*\*A, int\*DIS, int\*vis, int start, int N) {

int i;

start--;

DIS[start] = 0;

vis[start] = 1;

int k = 1;

p = start;

\_itoa(start + 1, way[start], 10);// для пути в каждую вершину ставим в начало стартовую

do {

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

if ((A[p][i] != 0) && (vis[i] == 0) && (i != p) && (DIS[i] > DIS[p] + A[p][i])) {

\_itoa(i + 1, post, 10);

strcpy(way[i], way[p]);

strcat(way[i], " -> ");

strcat(way[i], post);

DIS[i] = DIS[p] + A[p][i];

}

}

int min = 8000;

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

if ((DIS[i] < min) && (vis[i] == 0)) {

min = i;

}

p = min;

vis[p] = 1;

k++;

} while (k != N);

}

void Title() {

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

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

printf("\nНа тему: \"Реализация алгоритма Дейкстры\"");

printf("\n\nВыполнила студентка группы 19ВВ3");

printf("\nХанбекова Е. В.");

printf("\n\nПринял:");

printf("\nМитрохин М.А.\n\n");

printf("\nНажмите любую клавишу для продолжения");

\_getch();

}

char Menu() {

int komanda = 0;

system("cls");

printf("1| Ручной ввод графа ");

printf("\n2| Автоматический ввод графа");

printf("\n3| Загрузить граф из файла");

printf("\n0| Завершение работы\n\n");

printf("Ведите номер команды ");

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

system("cls");

return(komanda);

}

void Saverez(int\*DIS, int start, int N) {

char nazvanie[10];

printf("\nВедите название файла: ");

scanf("%s", nazvanie);

FILE \*F;

if ((F = fopen(nazvanie, "a+")) == NULL)

{

fprintf(stderr, "Невозможно открыть для записи файл\n");

return;

}

for (p = 0; p < N; p++) {

if (start - 1 != p) {

if (DIS[p] != 8000) {

fprintf(F, "\n Путь: %s\n", way[p]);

fprintf(F, " Расстояние = %d \n", DIS[p]);

}

else

fprintf(F, "\n Пути из %d в %d нет\n\n ", start + 1, p + 1);

}

}

fclose(F);

}

void Save(int N, int \*\*A) {

char nazvanie[10];

printf("Ведите название файла: ");

scanf("%s", nazvanie);

FILE \*F;

if ((F = fopen(nazvanie, "w+")) == NULL)

{

fprintf(stderr, "Невозможно открыть для записи файл\n");

return;

}

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

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

fprintf(F, "%d ", A[i][j]);

}

fprintf(F, "\n");

}

fclose(F);

}

int\*\* Handvvod(int N, int \*\*A) {

int komanda = 0, i, j;

printf("1| Ориентированный граф");

printf("\n2| Неориентированный граф");

printf("\n\nВедите номер команды ");

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

while ((komanda < 1) || (komanda > 2)) {

printf("\n\nНеверно введена команда. Попробуйте еще раз. ");

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

}

switch (komanda) {

case 1:

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

for (j = 0; j < N; j++) {

if (i != j) {

printf("Расстояние от %d до %d ", i + 1, j + 1);

scanf\_s("%d", &A[i][j]);

while (A[i][j] < 0) {

printf("\nТолько положительные числа! Введите другое число ");

scanf\_s("%d", &A[i][j]);

}

}

else

A[i][j] = 0;

}

}

break;

case 2:

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

for (j = 0; j < N; j++) {

if (i == j) {

A[i][j] = 0;

}

if (i < j) {

printf("Расстояние от %d до %d ", i + 1, j + 1);

scanf\_s("%d", &A[i][j]);

while (A[i][j] < 0) {

printf("\nТолько положительные числа! Введите другое ");

scanf\_s("%d", &A[i][j]);

}

A[j][i] = A[i][j];

}

}

}

break;

}

printf("\n\n ");

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

printf("%d ", i + 1);

}

printf("\n\n");

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

printf("%d| ", i + 1);

for (j = 0; j < N; j++) {

printf("%d ", A[i][j]);

}

printf("\n");

}

printf("\nДля сохранения графа нажмите 1 ");

char kom;

kom = getch();

if (kom == '1') {

Save(N, A);

}

return(A);

}

int\*\* Avtovvod(int N, int \*\*A) {

int komanda = 0, i, j;

printf("1| Ориентированный граф");

printf("\n2| Неориентированный граф");

printf("\n\nВедите номер команды ");

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

while ((komanda > 2) || (komanda < 1)) {

printf("\n\nНеверно введена команда. Попробуйте еще раз. ");

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

}

switch (komanda) {

case 1:

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

for (j = 0; j < N; j++) {

A[i][j] = rand() % 5;

if (i == j) {

A[i][j] = 0;

}

}

}

break;

case 2:

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

for (j = 0; j < N; j++) {

A[i][j] = rand() % 6;

A[j][i] = A[i][j];

if (i == j) {

A[i][j] = 0;

}

}

}

break;

}

printf("\n\n ");

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

printf("%d ", i + 1);

}

printf("\n\n");

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

printf("%d| ", i + 1);

for (j = 0; j < N; j++) {

printf("%d ", A[i][j]);

}

printf("\n");

}

printf("\nДля сохранения графа введите 1 ");

char kom;

kom = getch();

if (kom == '1') {

Save(N, A);

}

return(A);

}

int\*\* Loadvvod(int N, int \*\*A) {

FILE \*F;

int i, j;

char nazvanie[10];

printf("\nВедите название файла: ");

scanf("%s", nazvanie);

if ((F = fopen(nazvanie, "r")) == NULL)

{

fprintf(stderr, "Невозможно открыть для чтения файл\n Нажмите любую клавишу для выхода\n");

\_getch();

return 0;

}

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

for (j = 0; j < N; j++) {

fscanf(F, "%d", &A[i][j]);

}

}

fclose(F);

if (A[N - 1][N - 1] < 0) {

printf("Количество вершин было введено неверно. Для выхода нажмите любую клавишу ");

\_getch();

err = 1;

return 0;

}

printf("\n\n ");

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

printf("%d ", i + 1);

}

printf("\n\n");

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

printf("%d| ", i + 1);

for (j = 0; j < N; j++) {

printf("%d ", A[i][j]);

}

printf("\n");

}

return(A);

}

int main(void)

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int Selection;

//Title();

system("cls");

while ((Selection = Menu()) != 0) {

while ((Selection > 3) || (Selection < 0)) {

printf("Неверно введен номер. Попоробуйте еще раз");

\_getch();

Selection = Menu();

}

int N;

printf("\nВведите число вершин графа: ");

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

int \*\*A = (int \*\*)malloc(N \* sizeof(int \*));

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

A[i] = (int \*)malloc(N \* sizeof(int));

}

switch (Selection) {

case 1:

Handvvod(N, A);

break;

case 2:

Avtovvod(N, A);

break;

case 3:

Loadvvod(N, A);

if (err == 1)

return 0;

break;

case 0:

return 0;

break;

}

int \*DIS = (int\*)malloc(N \* sizeof(int));

int \*vis = (int\*)malloc(N \* sizeof(int));

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

DIS[i] = 8000;

vis[i] = 0;

}

int start;

printf("\n\nВедите вершину, с которой начать обход ");

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

while (start > N) {

printf("\n Ошибка. Введите вершину заново ");

scanf("%d", &start);

}

Dijkstra(A, DIS, vis, start, N);

for (p = 0; p < N; p++) {

if (start-1!=p) {

if (DIS[p] != 8000){

printf("\n Путь: %s", way[p]);

printf("\n");

printf(" Расстояние = %d \n\n", DIS[p]);

}

else

printf("\n Пути из %d в %d нет\n\n ", start + 1, p + 1);

}

}

printf("\nДля сохранения результата нажмите 1 ");

char kom;

kom = getch();

if (kom == '1') {

Saverez(DIS, start, N);

}

\_getch();

free(A);

free(DIS);

free(vis);

for (int k = 0; k < 100; k++) {

post[k] = 0;

for (int l = 0; l < 100; l++) {

way[k][l] = 0;

}

}

}

}