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:

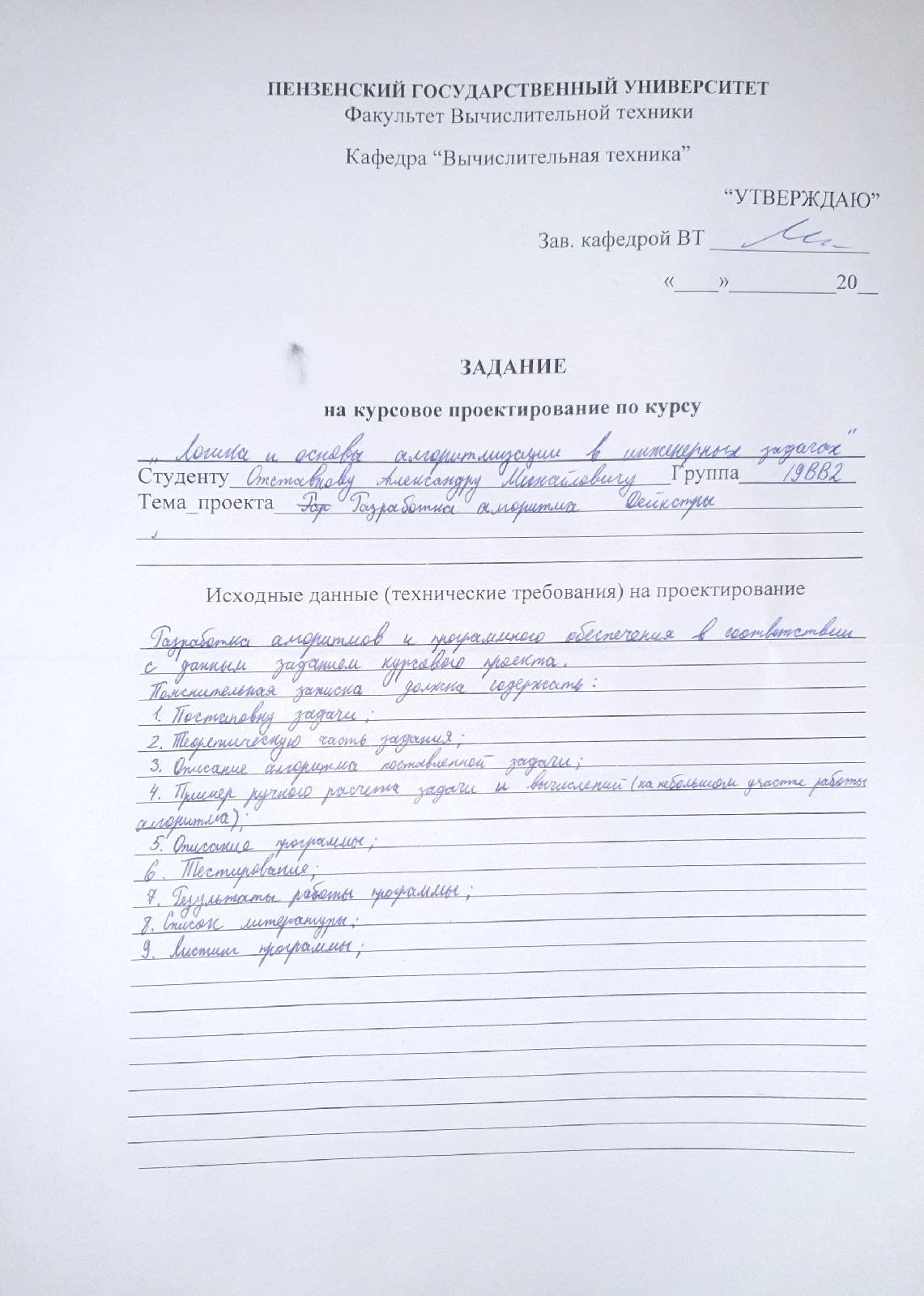
студент группы 19ВВ2

Отставнов А.М.

Принял:

Митрохин М.А.

Пенза 2020



**Содержание**

[1. Реферат 5](#_Toc58799884)

[2. Введение 6](#_Toc58799885)

[3. Постановка задачи 6](#_Toc58799886)

[4. Теоретическая часть задания 7](#_Toc58799887)

[5. Описание алгоритма программы 9](#_Toc58799888)

[6. Описание программы 11](#_Toc58799889)

[7. Тестирование 17](#_Toc58799890)

[8. Ручной расчёт задачи 20](#_Toc58799891)

[9. Заключение 22](#_Toc58799892)

[10. Список литературы 23](#_Toc58799893)

[11. Приложения 24](#_Toc58799894)

# Реферат

Отчет 34 стр, 12 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ДЕЙКСТРЫ, РАССТОЯНИЕ.

Цель исследования – разработка программы, реализующую алгоритм Дейкстры.

В работе рассмотрен алгоритм Дейкстры. Установлено, что с помощью данного алгоритма можно находить кратчайшее расстояние между вершинами ориентированного и неориентированного графа.

# Введение

Алгоритм Дейкстры строит кратчайшие пути, ведущие из исходной вершины графа к остальным вершинам этого графа (если таковые имеются).

В процессе работы алгоритма последовательно помечаются рассмотренные вершины графа. Причем вершина, помеченная последней (на данный момент) расположена ближе к исходной вершине, чем все непомеченные, но дальше, чем все помеченные.

Сначала помечается исходная вершина; следующей, очевидно, будет помечена вершина, ближайшая к исходной, и смежная с ней.

Пусть на каком-то шаге уже помечено несколько вершин. Известны кратчайшие пути, ведущие из исходной вершины к помеченным. Для каждой из непомеченных вершин проделаем следующее:

1. Рассмотрим все дуги, ведущие из помеченных вершин в одну непомеченную. Каждая такая дуга является последней дугой на пути из исходной вершины в эту непомеченную.
2. Выберем из этих путей кратчайший. А затем выберем среди них самый короткий ко всем непомеченным вершинам, и пометим вершину, к которой он ведет.

Алгоритм завершится, когда будут помечены все достижимые вершины.

В результате работы алгоритма Дейкстры строится Дерево кратчайших путей.

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

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Дейкстры, осуществляющий поиск кратчайшего расстояния между вершинами.

# Постановка задачи

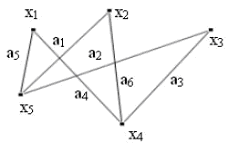
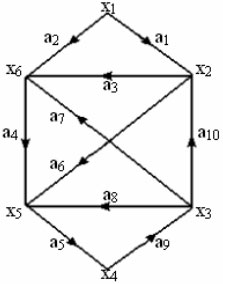
Требуется разработать программу, которая осуществит поиск кратчайшего расстояния между двумя вершинами, использую алгоритм Дейкстры.

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 рандомного заполнения), выбирал тип создаваемого графа (ориентированный или неориентированный), вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности графа. Для поиска кратчайшего расстояния между двумя существующими вершинами пользователь должен ввести номера этих вершин. Если пользователь введет номер несуществующей команды или выберет для поиска расстояния несуществующую вершину, программа должна вывести соответствующее сообщение с просьбой заново ввести требуемые данные. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

# Теоретическая часть задания

**Сведения о графах**

Граф G задается множеством точек (вершин) х1, х2,..., хn. (которое обозначается через Х) и множеством линий (ребер) а1, а2,...,аm. (которое обозначается символом А), соединяющих между собой все или часть этих точек. Таким образом, граф G полностью задается (и обозначается) парой (Х, А). Если ребра из множества А ориентированы, что обычно показывается стрелкой, то они называются дугами, и граф с такими ребрами называется ориентированным графом.

*а б*

Рисунок 1 – Примеры графов: неорграф(*а*) и орграф(*б*)

Например, если дорога имеет не двух-, а одностороннее движение то направление этого движения будет показано стрелкой.

Если ребра не имеют ориентации, то граф называется неориентированным, (двухстороннее движение).

В ориентированном графе дуга обозначается упорядоченной парой, состоящей из начальной и конечной вершин, ее направление предполагается заданным от первой вершины ко второй.

Путем (или ориентированным маршрутом) ориентированного графа называется последовательность дуг, в которой конечная вершина всякой дуги, отличной от последней, является начальной вершиной следующей.

Так, путями являются последовательности дуг:

а6, а5, а9, а8, а4. (1)

а1, а6, а5, а9. (2)

а1, а6, а5, а9, а10, а6, а4. (3)

Ориентированной цепью (орцепью) называется такой путь, в котором каждая дуга используется не больше одного раза.

Простой орцепью называется такой путь, в котором каждая вершина используется не более одного раза. Например, путь (2).

Маршрут есть неориентированный “двойник” пути, и это понятие рассматривается в тех случаях, когда можно пренебречь направленностью дуг в графе. Таким образом, маршрут есть последовательность ребер ä1, ä2,..., äq, в которой каждое ребро аi, за исключением первого и последнего ребер, связано с ребрами аi-1 и аi+1 своими концевыми вершинами. Последовательности дуг:

ä2, ä4, ä8, ä10, (4)

ä2, ä7, ä8, ä4, ä3, (5)

ä10 , ä7 , ä4 , ä8 , ä7 , ä2. (6)

Также путь или маршрут можно изображать последовательностью вершин. Иногда дугам графа приписываются числа, называемыевесом, стоимостью, или длиной этой дуги. В этом случае граф называется графом с взвешенными дугами*.* А если вес приписывается вершинам графа, то тогда получается граф с взвешенными вершинами. Если в графе веса приписаны и дугам, и вершинам, то он называется просто взвешенным*.*

# Описание алгоритма программы

Таблица 1 – Описание переменных

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Роль** |
| i, j | int | Счётчики |
| x\_start, x\_end | int | Начальная и конечная вершины |
| N | int | Количество вершин графа |
| P | int | Номер кратчайшего расстояния |
| CMND | int | Номер команды главного меню |
| Type | int | Переменная для выбора типа графа (1 – ориент., 2 – неориент.) |
| visited[200] | int | Массив, содержащий непостоянные и постоянные пути и расстояния  (0 – непостоянные; 1 – постоянные) |
| graph[200][200] | unsigned int | Массив, содержащий расстояние между вершинами |
| dist[200][200] | unsigned int | Массив, содержащий всю длину |
| way[200][200] | char | Массив, содержащий весь |
| Temp[200] | char | Массив, содержащий промежуточные пути |
| save[5], save1[5], ex[5] | char | Переменные, в которые записывается реакция пользователя на запрос программы |
| R | int | Переменная, которая имеет случайное значение, показывает вероятность существования обратного пути. |

Описание работы функции DIJKSTRA()

Алгоритм начинается с того, что несуществующим ребрам матрицы присваивается значение 50 000, которое означает бесконечность и заведомо превосходит все возможные длины путей. Далее всем элементам массива visited присваивается значение 0, т.е. все вершины помечаются, как не посещённые. Также расстояние до каждой вершины становится бесконечным, равным 50 000. Начальная вершина помечается как уже посещенная (visited[x\_start] = 1) и общее расстояние(dist[x\_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 (dist[i] > dist[p] + graph[p][i]), тогда это расстояние dist[p] + graph[p][i] присваивается dist[p]. После чего переменной p присваивается значение i.Этой вершине присваивается значение посещенной visited[р] = 1 Весь цикл работает до тех пор, пока p не станет вершиной х\_еnd, кратчайший путь до которой мы и пытались найти.

Ниже представлен псевдокод части функций DIJKSTRA()

**DIJKSTRA()**

для i = 0, пока i < n, выполнять i=i+1

для j = 0, пока j < n, выполнять j=j+1

если graph[i][j] == 0

graph[i][j] = 50000; //очень большое значение

для i = 0 пока i < n выполнять i= i+1

visited[i] = 0

dist[i] = 50000

dist[x\_start] = 0

visited[x\_start] = 1

p = x\_start;

преобразовать x\_start+ 1 в строку и записать в Temp

для i = 0 пока i <= n выполнять i= i+1

скопировать "X" в way[i]

записать Temp в way[i]

выполнять

для i = 0 пока i < n выполнять i= i+1

если graph[p][i] != 50000 и !visited[i] и i != p

если dist[i] > dist[p] + graph[p][i]

преобразовать i + 1 в строку и записать в Temp

скопировать way[p + 1] в way[i + 1]

записать " -> X" в way[i + 1]

записать Temp в way[i + 1]

если dist[i] > dist[p] + graph[p][i]

dist[i] = dist[p] + graph[p][i]

иначе

dist[i] = dist[i] + 0;

для i = 0 пока i < n выполнять i= i+1

если !(visited[i]

p = i;

для i = 0 пока i < n выполнять i= i+1

если dist[p] > dist[i]) и !visited[i]

p = i;

visited[p] = 1;

пока p != x\_end

# Описание программы

Для написания данной программы использован язык программирования Си. Язык программирования Си - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

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

Данная программа является многомодульной, поскольку состоит из нескольких функций: main(), RAND(), HAND(), TITLE(), MENU(), DIJKSTRA().

Работа программа начинается с вывода краткой информации о работе.

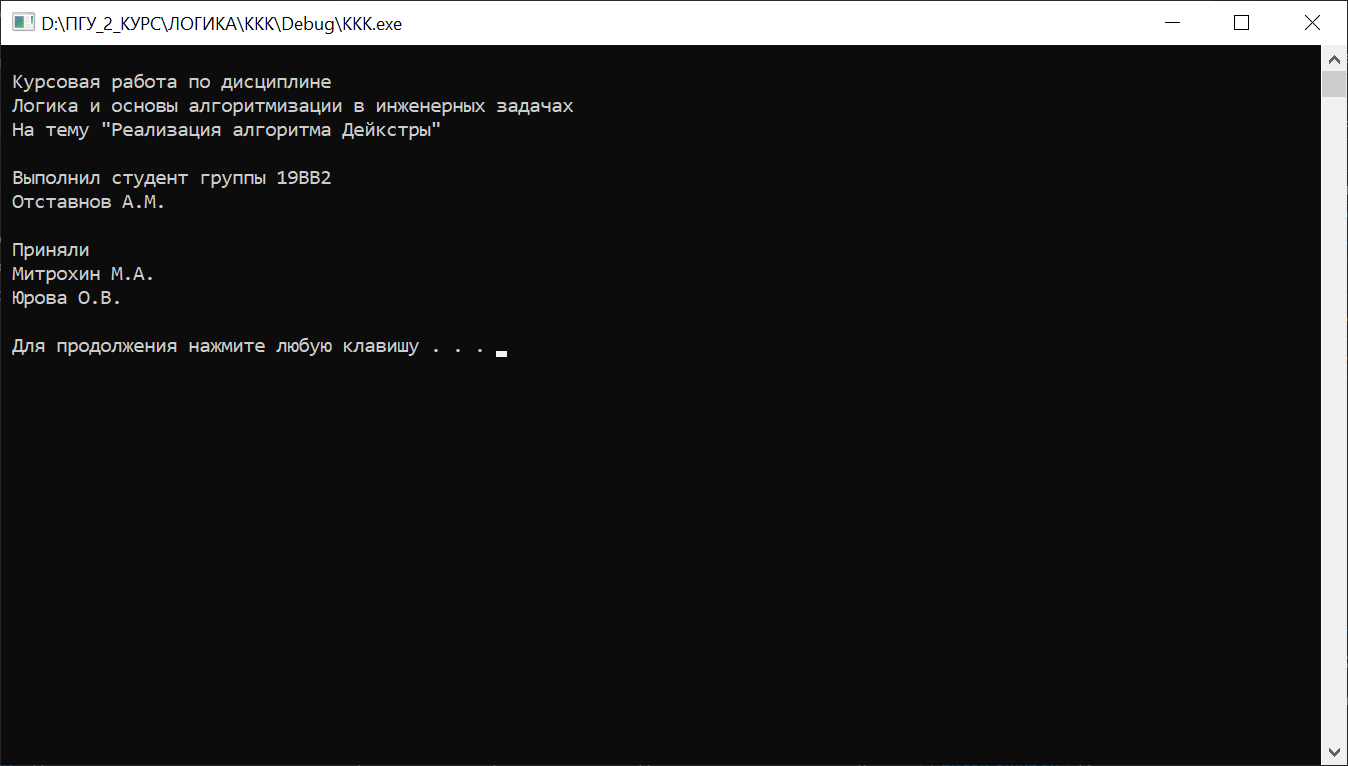


Рисунок 2 – Начальная страница

Для продолжения работы нужно нажать любую клавишу. После чего открывается основное меню программы. При выборе первого пункта, пользователь попадет в меню автоматического создания матрицы. При выборе второго пункта, пользователь попадет в меню создания матрицы. Выбор третьего пункта меню приведет к завершению работы.

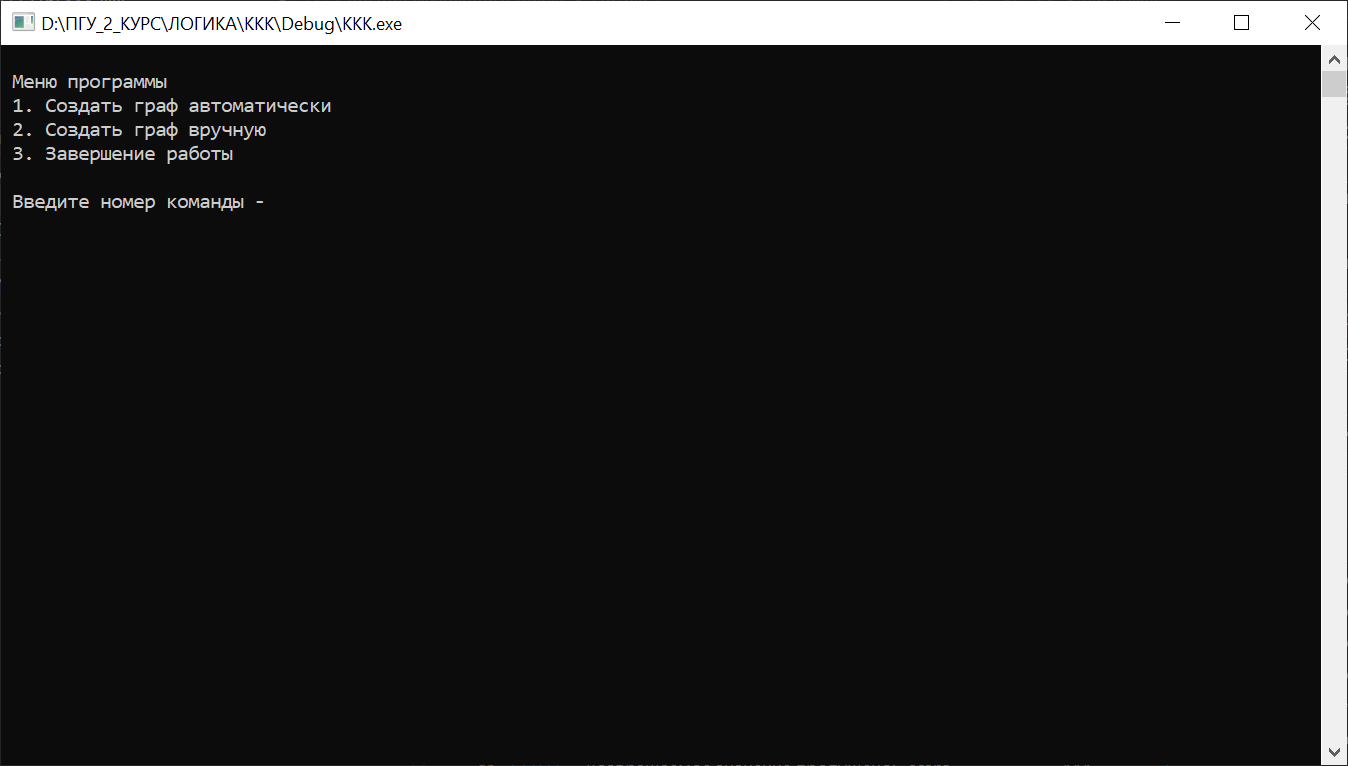


Рисунок 3 – Главное меню

При выборе первого или второго пункта пользователю предлагается выбрать тип создаваемого – ориентированный или неориентированный. После выбора одного из двух типов пользователю требуется ввести количество вершин создаваемого графа. В случае создания матрицы с помощью автозаполнения матрица смежности графа будет выведена на экран (Рисунок 3). В случае создания матрицы вручную пользователь должен будет ввести расстояние между вершинами, причем в случае ориентированного граф нужно будет вводить расстояние как из i-й вершины в j-ю, так и из j-й вершины в i-ю. В неориентированном же графе нужно будет вводить расстояние только из i-й вершины в j-ю, так как в таком графе graph[i][j] = graph[j][i].

printf(" Матрица смежности для ориентированного графа\n");

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

{

for (j = i; j < n; ++j)

{

R = rand() % 100;

if (R > 30)

{

graph[i][j] = rand() % 10;

graph[j][i] = 0;

graph[i][i] = graph[j][j] = 0; // чтобы петля(узел) не создавалась

}

else

graph[i][j] = graph[j][i] = rand() % 10;

graph[i][i] = graph[j][j] = 0; // чтобы петля(узел) не создавалась

}

} }

if (type == 2) {

printf(" Матрица смежности для неориентированного графа\n");

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

{

for (j = i; j < n; ++j)

{

graph[i][j] = graph[j][i] = rand() % 10;

graph[i][i] = graph[j][j] = 0; // чтобы петля(узел) не создавалась

}

}}

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

{

printf("%4d ", j + 1); //горизонт

}

printf("\n\n");

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

{

printf(" %d ", i + 1); // вертик

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

{

printf("%4d ", graph[i][j]);

}

printf("\n\n"); }

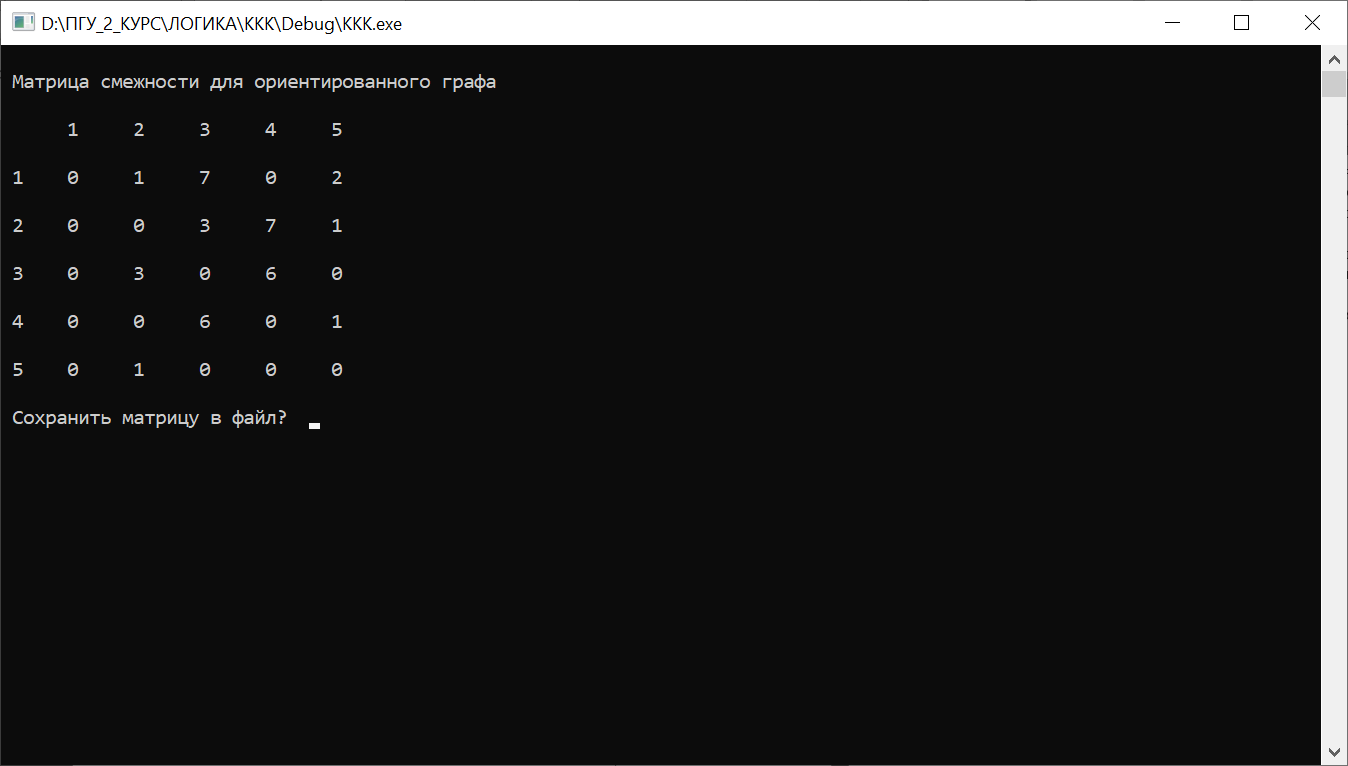
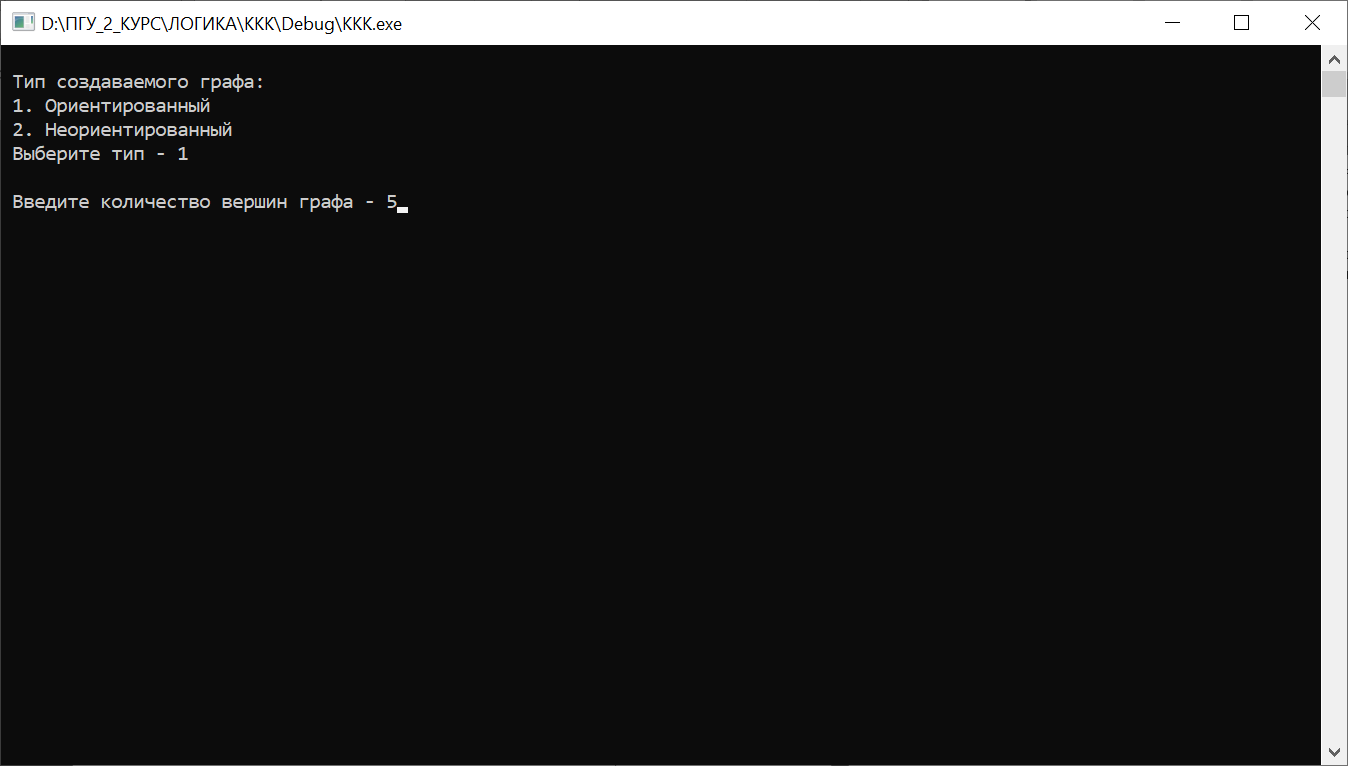


Рисунок 4 – Автоматическое создание графа

if (type == 1)

{

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

{

for (j = i + 1; j < n; j++)

{

printf(" Введите расстояние из %d в %d - ", i + 1, j + 1);

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

printf(" Введите расстояние из %d в %d - ", j + 1, i + 1);

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

}

}

}

}

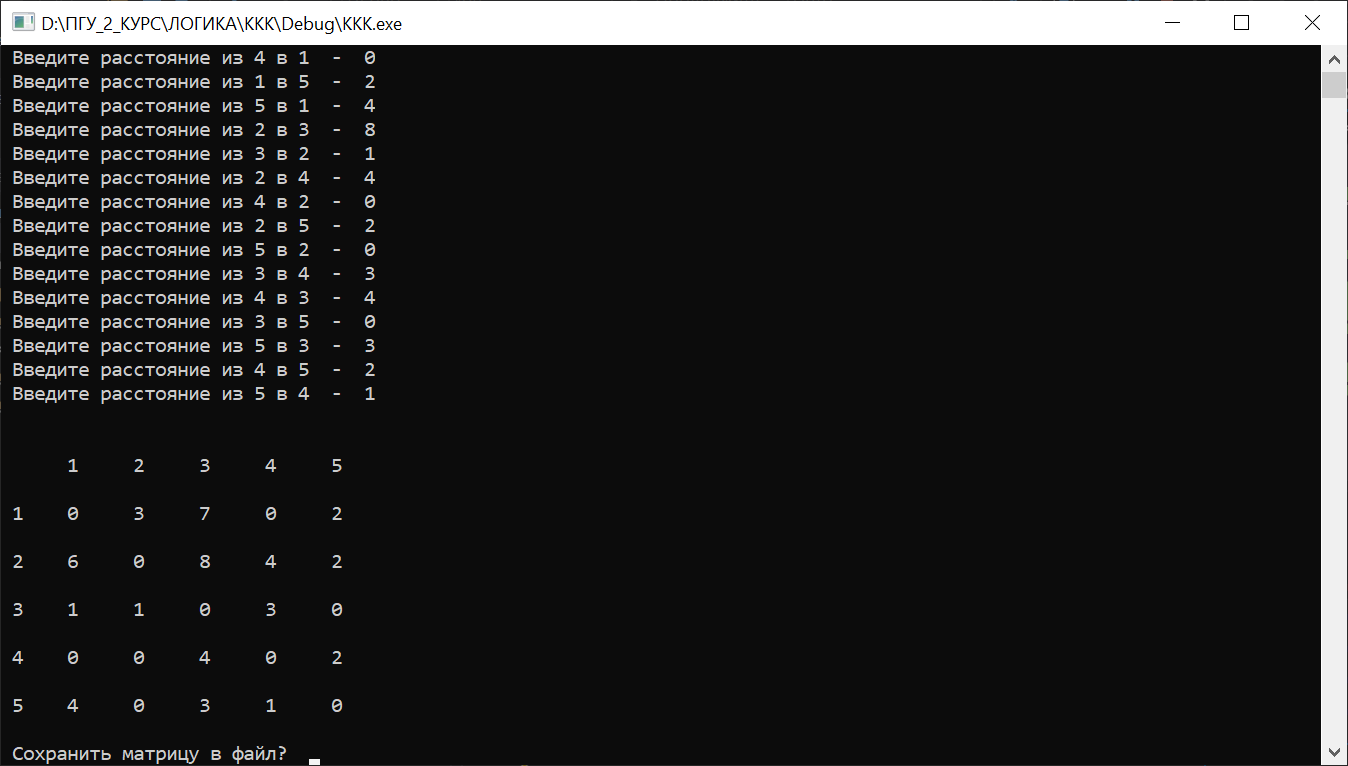
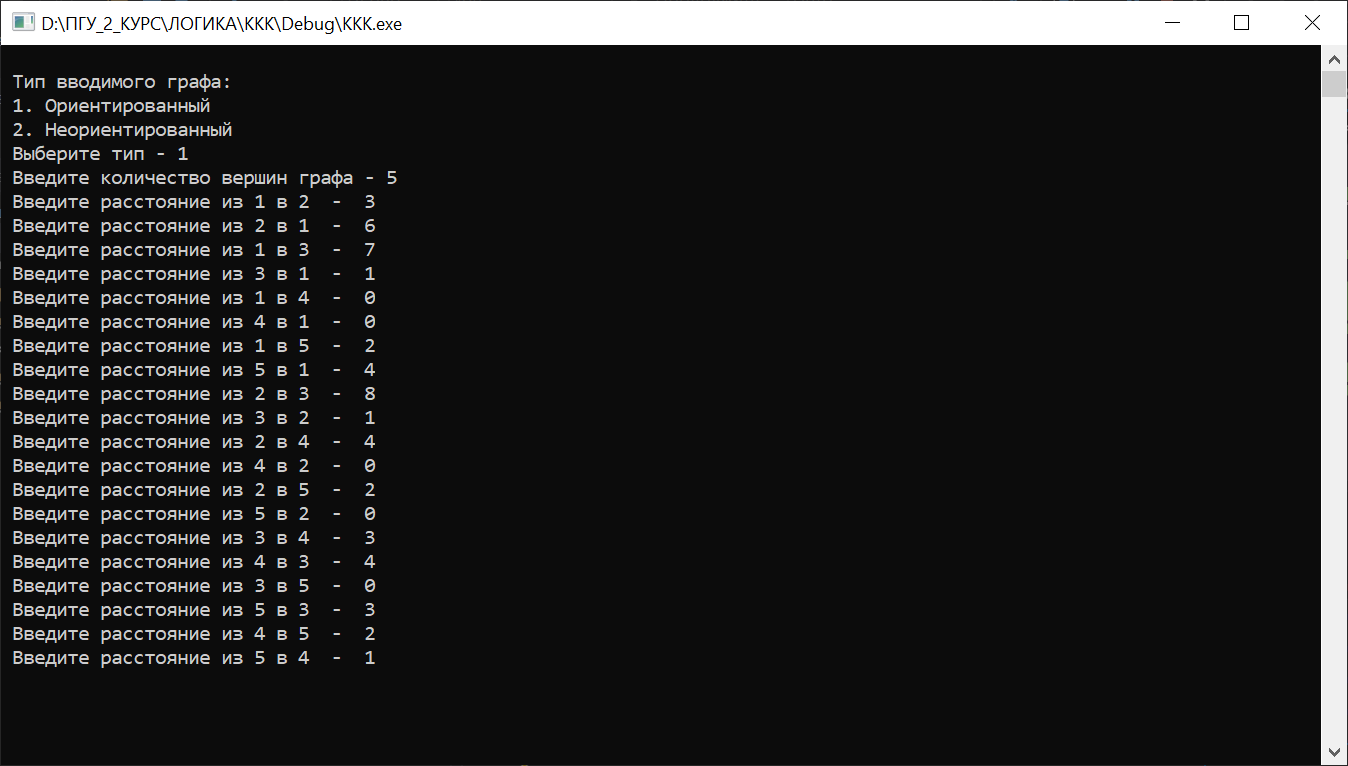


Рисунок 5 - Ручное создание графа (ориентированного)

if (type == 2)

{

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

{

for (j = i + 1; j < n; j++)

{

printf(" Введите расстояние из %d в %d - ", i + 1, j + 1);

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

graph[j][i] = graph[i][j];

}

}

}

printf("\n\n ");

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

{

printf("%4d ", j + 1); //горизонт

}

printf("\n\n");

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

{

printf(" %d ", i + 1); // вертик

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

{

printf("%4d ", graph[i][j]);

}

printf("\n\n");

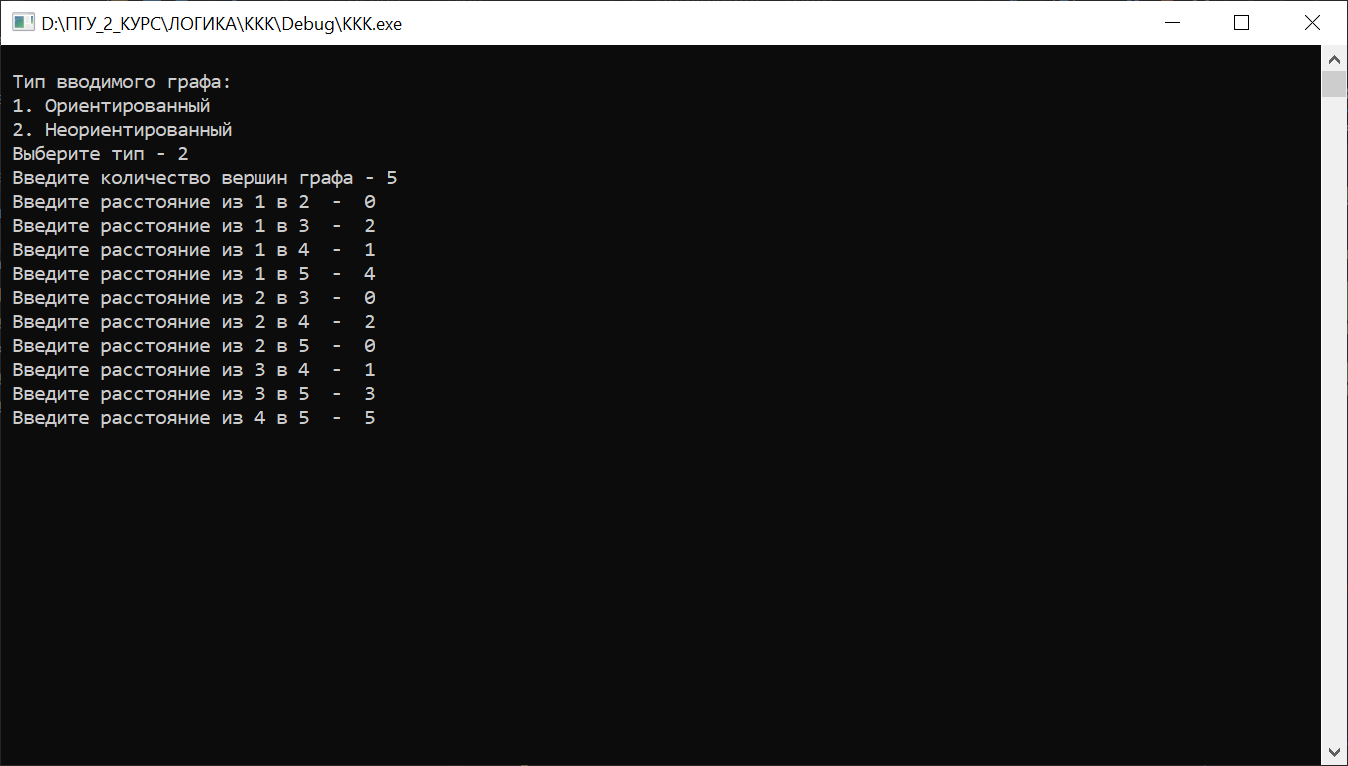
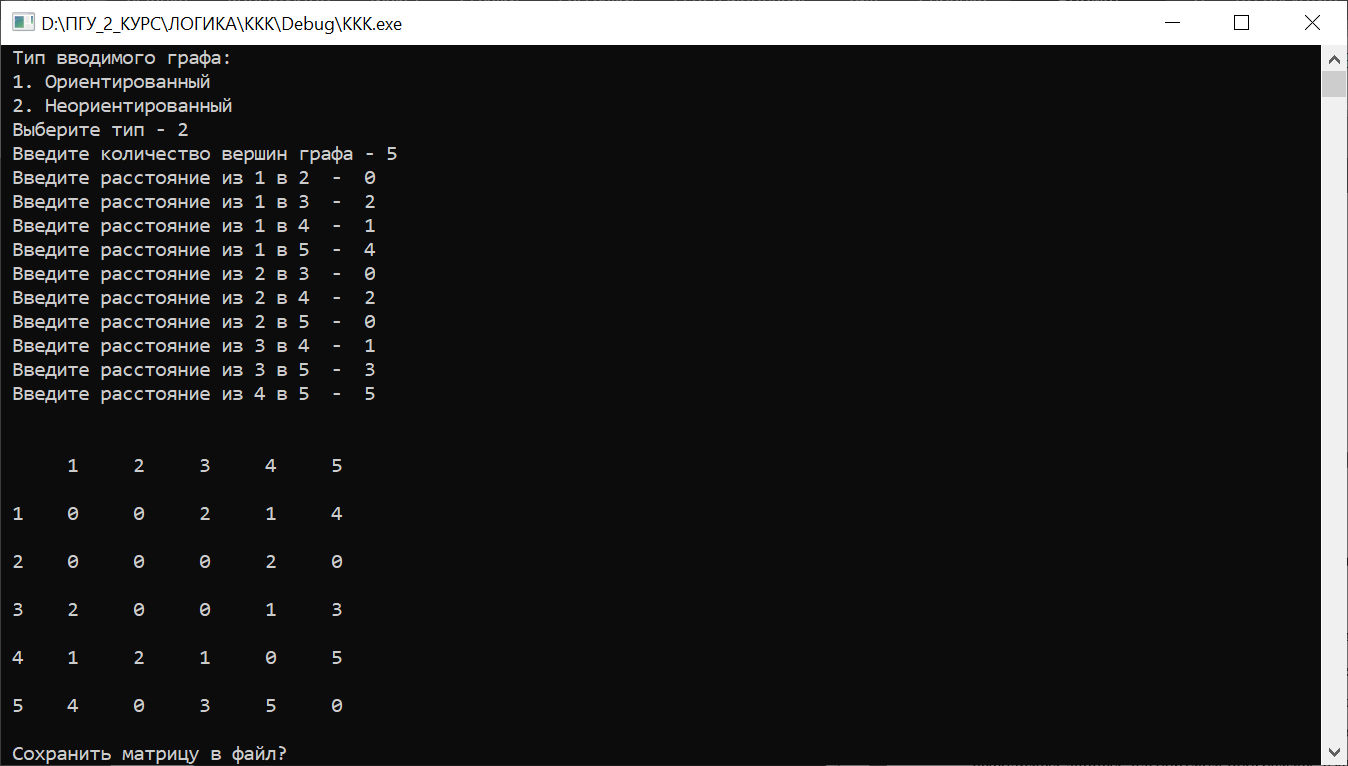
 

Рисунок 6 – Ручное создание графа (неориентировванный)

После вывода матрицы на экран появляется предложение сохранить сгенерированную матрицу в файл. Если пользователь хочет это сделать, то требуется написать либо «да», либо «Да». В противном случае можно написать любой другой символ. Далее пользователю нужно будет ввести начальную вершину, после – конечную. Если начальная и конечная вершина совпадут, то программа выведет соответсвующее собщение и попросит ввести их заново. Если пользователь введет несуществующую вершину, тогда программа также выведет соотвествующее сообщение и попросит ввести вершину заново. В противном случае программа, используя алгоритм Дейкстры выведет кратчайшее расстояние между вершинами и путь, который нужно пройти, чтобы получить такой результат. Конечно , если путь существует. Иначе программа уведомит пользователя о невозможности попадания из начальной вершины в конечную. Тут программа также предложит сохранить уже результат работы алгоритма в файл. Если пользователь хочет это сделать, то требуется написать либо «да», либо «Да». В противном случае можно написать любой другой символ. После чего пользователь попадет в главное меню.

Printf(« Введите начальную вершину – «);

scanf(«%d», &x\_start);

while (x\_start > n) {

printf(« Такой вершины не существует! \n Введите начальную вершину заново – «);

scanf(«%d», &x\_start);

}

printf(« Введите конечную вершину – «);

scanf(“%d”, &x\_end);

while (x\_end > n)

{

printf(« Такой вершины не существует!\n Введите конечную вершину заново – «);

scanf(“%d”, &x\_end);

}

while (x\_start == x\_end)

{

printf(« Начальная и конечная вершины совпадают\n»);

printf(« Введите начальную вершину заново – «);

scanf(“%d”, &x\_start);

while (x\_start > n)

{

printf(« Такой вершины не существует!\n Введите начальную вершину заново – «);

scanf(«%d», &x\_start);

}

printf(« Введите конечную вершину заново – «);

scanf(“%d”, &x\_end);

while (x\_end > n)

{

printf(« Такой вершины не существует!\n Введите конечную вершину заново – «);

scanf(“%d”, &x\_end);

}

}

x\_end--;

x\_start--;

DIJKSTRA();

if (dist[p] != 50000)

{

printf(“\n Путь - %s”, way[p + 1]);

printf(“\n”);

printf(“ Расстояние - %u \n\n”, dist[p]);

}

else

printf(«\n Пути из %d в %d не существует\n\n «, x\_start + 1, x\_end + 1);

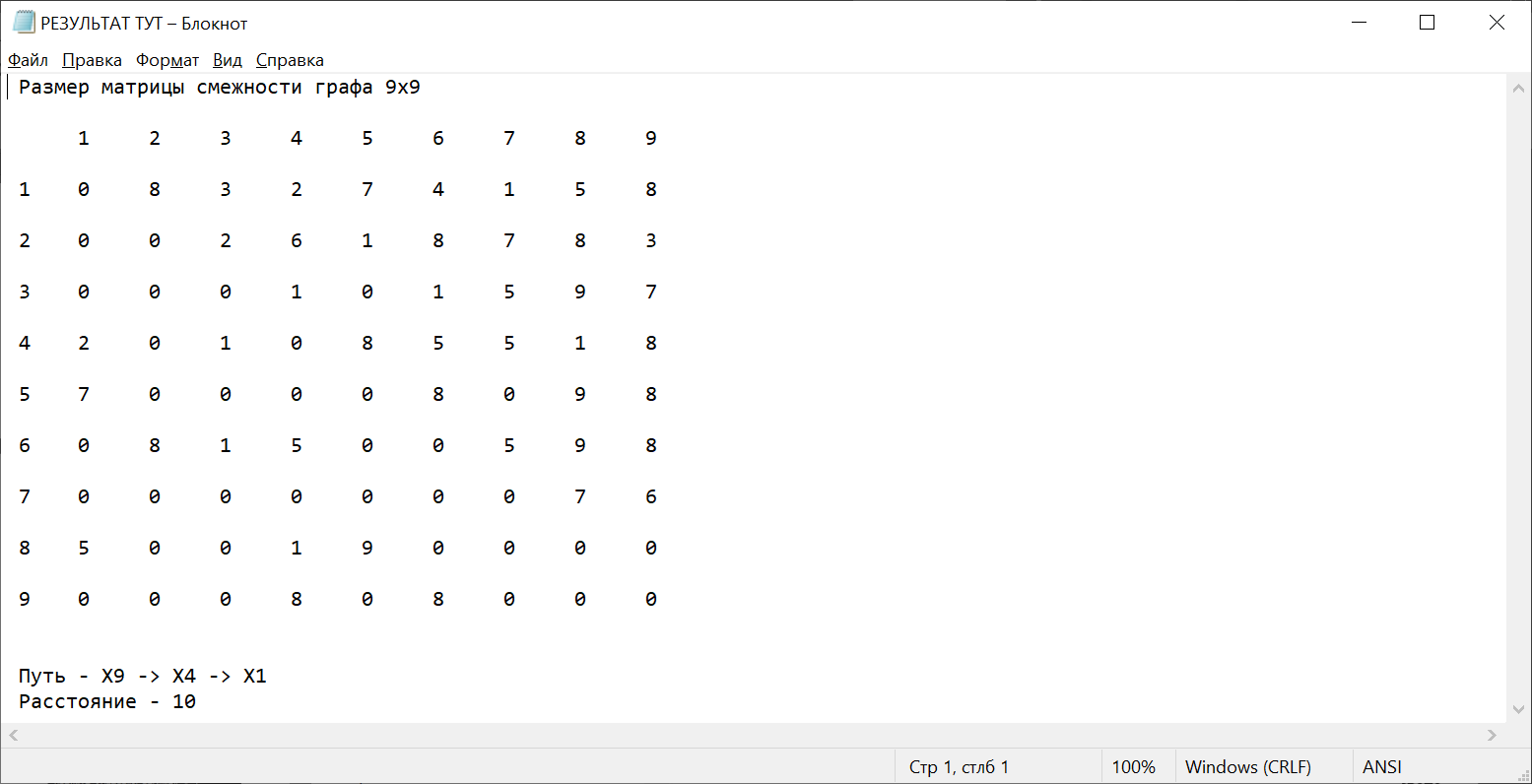
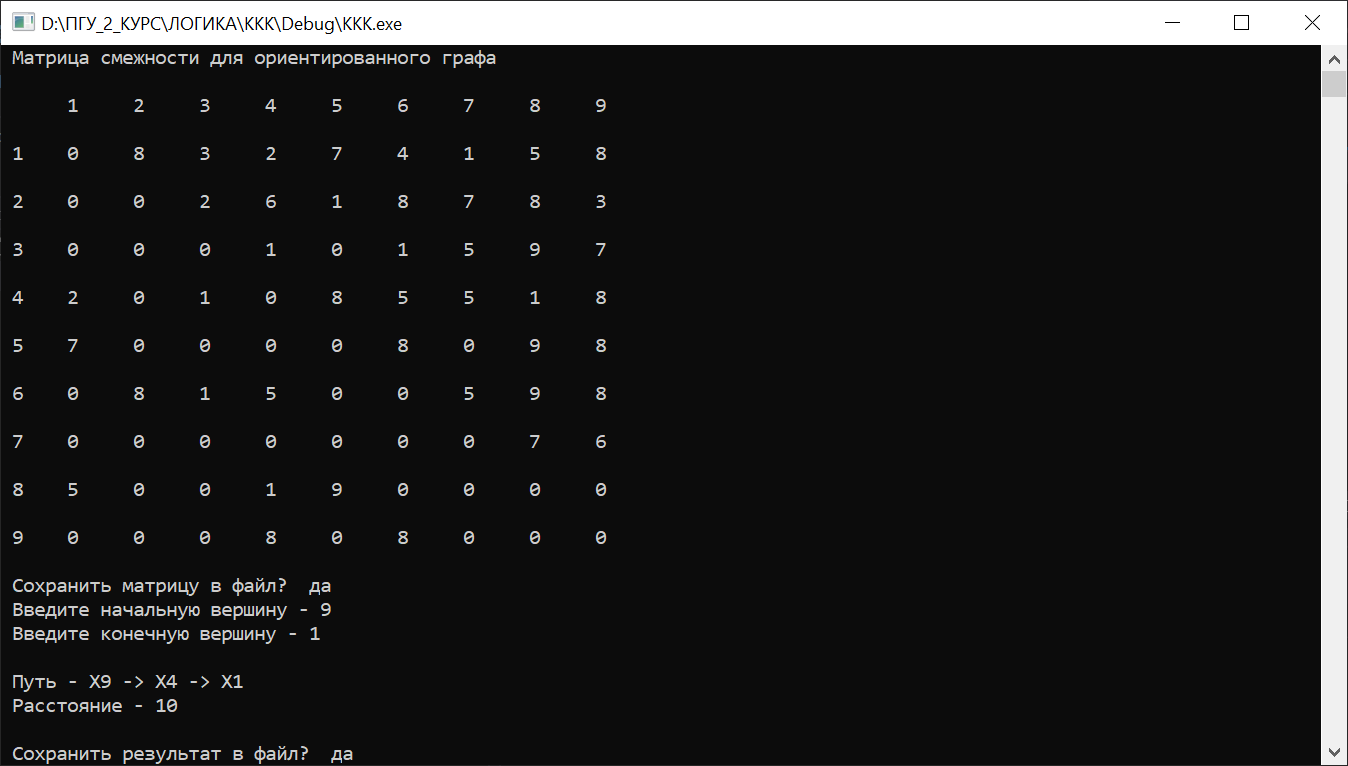


Рисунок 7 – Результат работы алгоритма и сохраненные данные в файле

# Тестирование

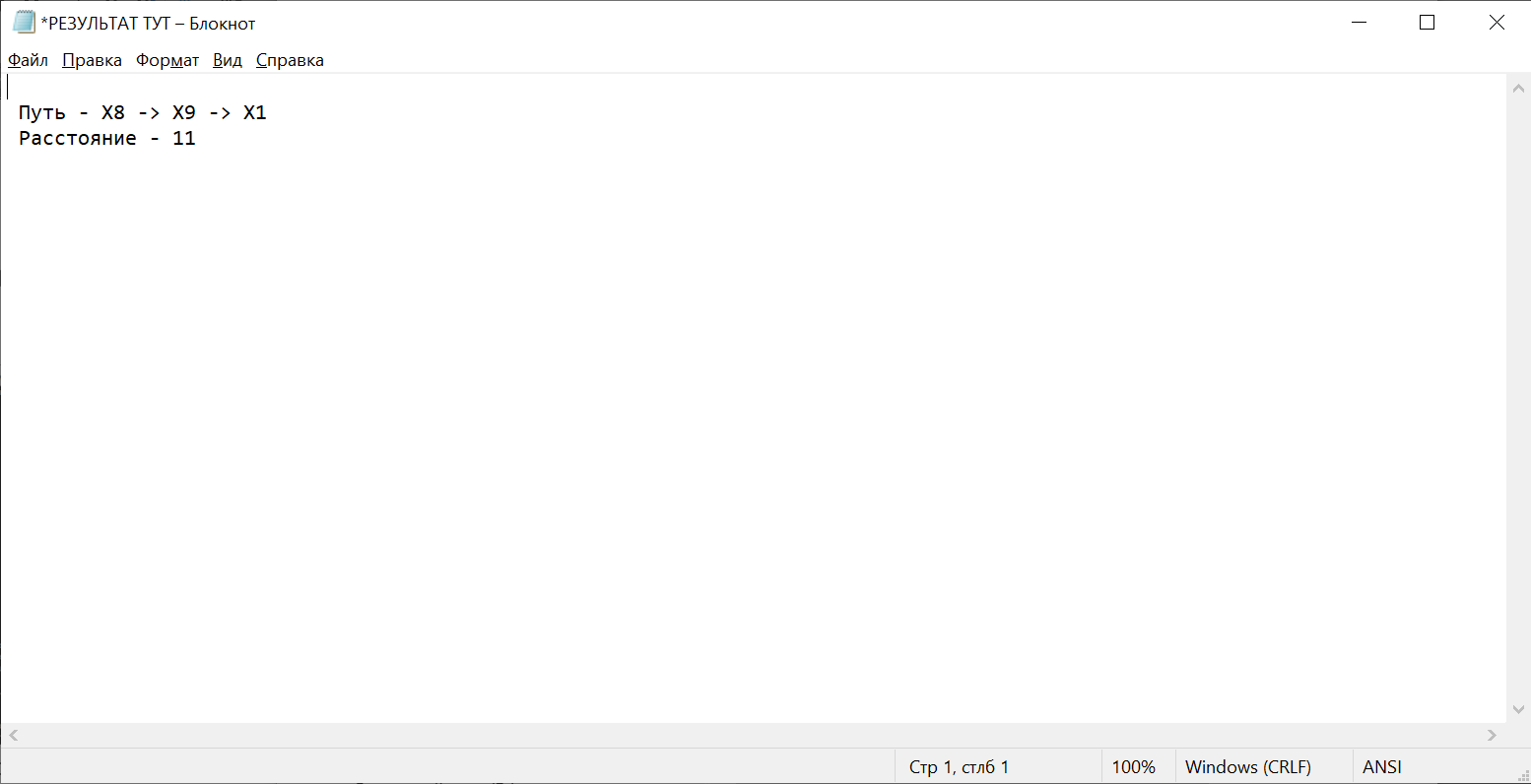
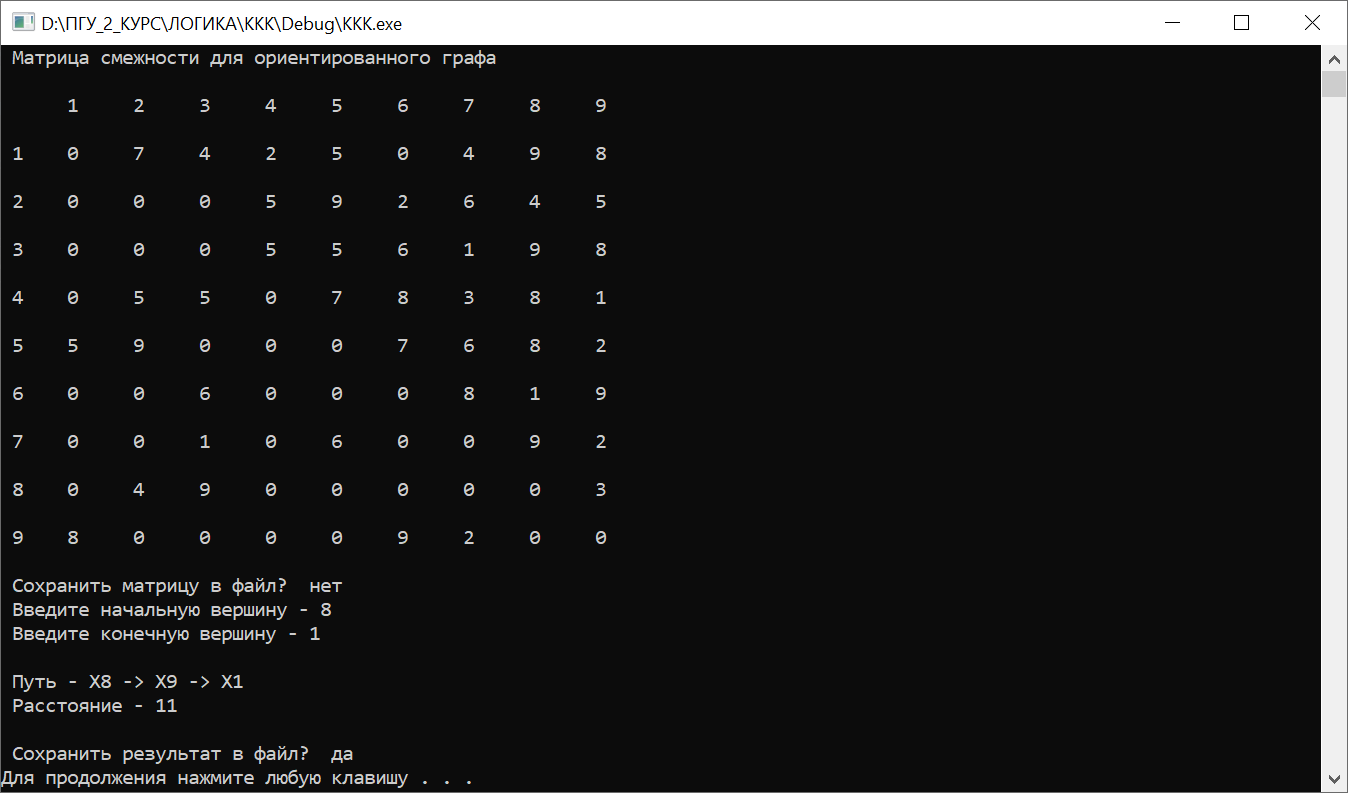
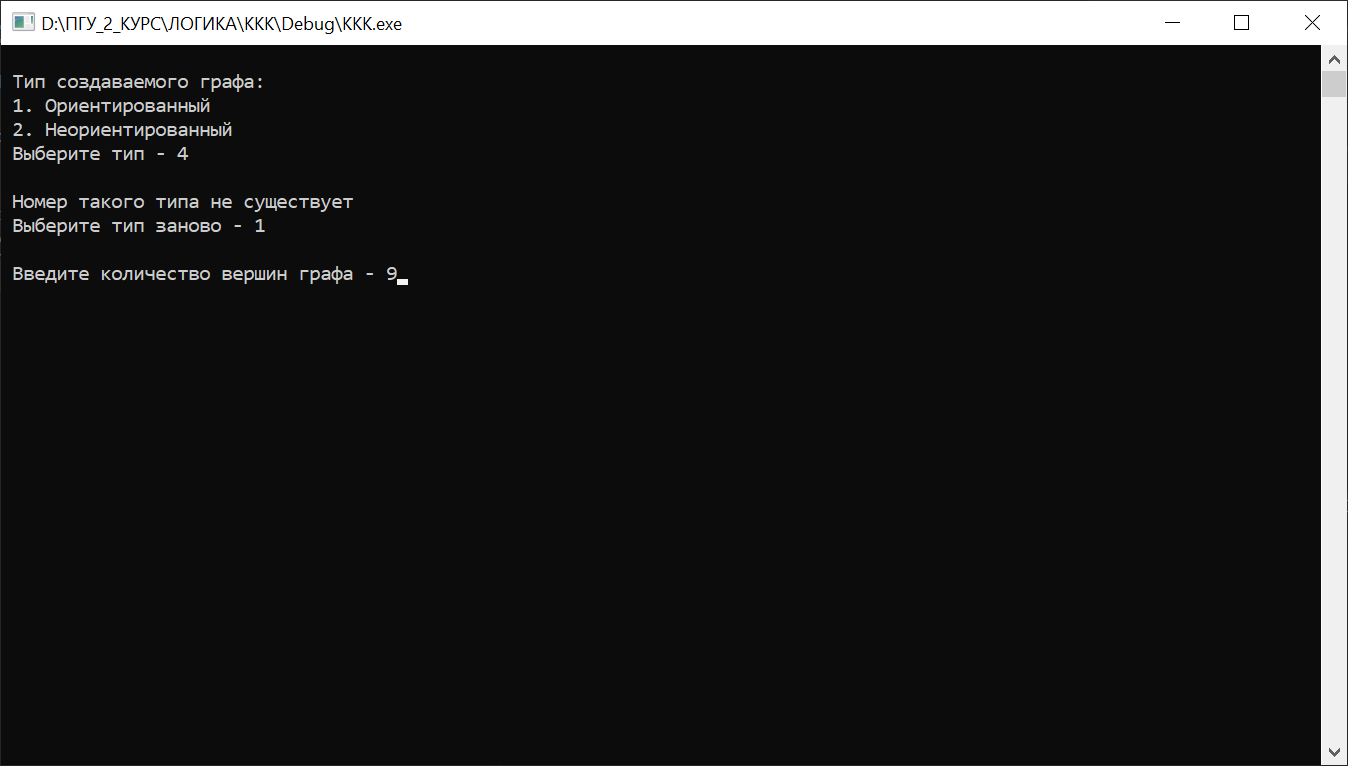
|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Ожидаемый результат | Выполненное действие | Полученный результат |
| Запуск программы | Вывод начального сообщения, содержащего краткую информацию о работе | Запустить файл программы тип .exe | Верно |
| Основное меню | Вывод меню с методами создания графа | Нажатие любой кнопки при выводе начального сообщения | Верно |
| Создание с помощью функции автоматического заполнения ориентированного графа с 9 вершинами | Вывод ориентированного графа размера 5х5, который был создан с помощью функции автоматического заполнения | В главном меню выбрать «Создать граф автоматически» (пункт 1).  В открывшемся меню выбрать ориентированный тип графа (пункт 1). Ввести количество вершин графа – 9. | Верно |
| Создание с помощью функции ручного заполнения неориентированного графа с 5 вершинами | Вывод ориентированного графа размера 8х8, который был создан с помощью ручного заполнения | В главном меню выбрать «Создать граф вручную» (пункт 2).  В открывшемся меню выбрать неориентированный тип граф (пункт 1). Ввести количество вершин графа – 5. | 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, введите слово «да» или «Да». | Верно |
| Поиск кратчайшего расстояния между двумя существующими вершинами | Вывод порядка обхода вершин и кратчайшего расстояния между двумя выбранными вершинами. Результат должен совпасть с результатом ручного просчета. | После сохранения\несохранения матрицы в файл нужно ввести начальную и конечную вершины. | Верно |
| Сохранение результата поиска в файл | Обнаружение полученного результата в файле | После вывода результата программой будет предложено сохранить его в файл. Чтобы сохранить, введите слово «да» или «Да». | Верно |
| Выход из программы | Завершение работы программы | После поиска кр. расстояния пользователь попадает в главное меню. Выбрать «Завершение работы»(пункт 3) | Верно |

Таблица 2 – План тестирования

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 исправлено множество проблем, связанных с вводом данных, изменением дизайна, алгоритмом программы, взаимодействием функций, а также с правильностью и неправильностью выводимых данных.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных данных.

Рисунок 8 – Ориентированный граф из 9 вершин, созданный автоматически, в файл сохранен только результат

(пример неправильно выбранного типа создаваемого графа)

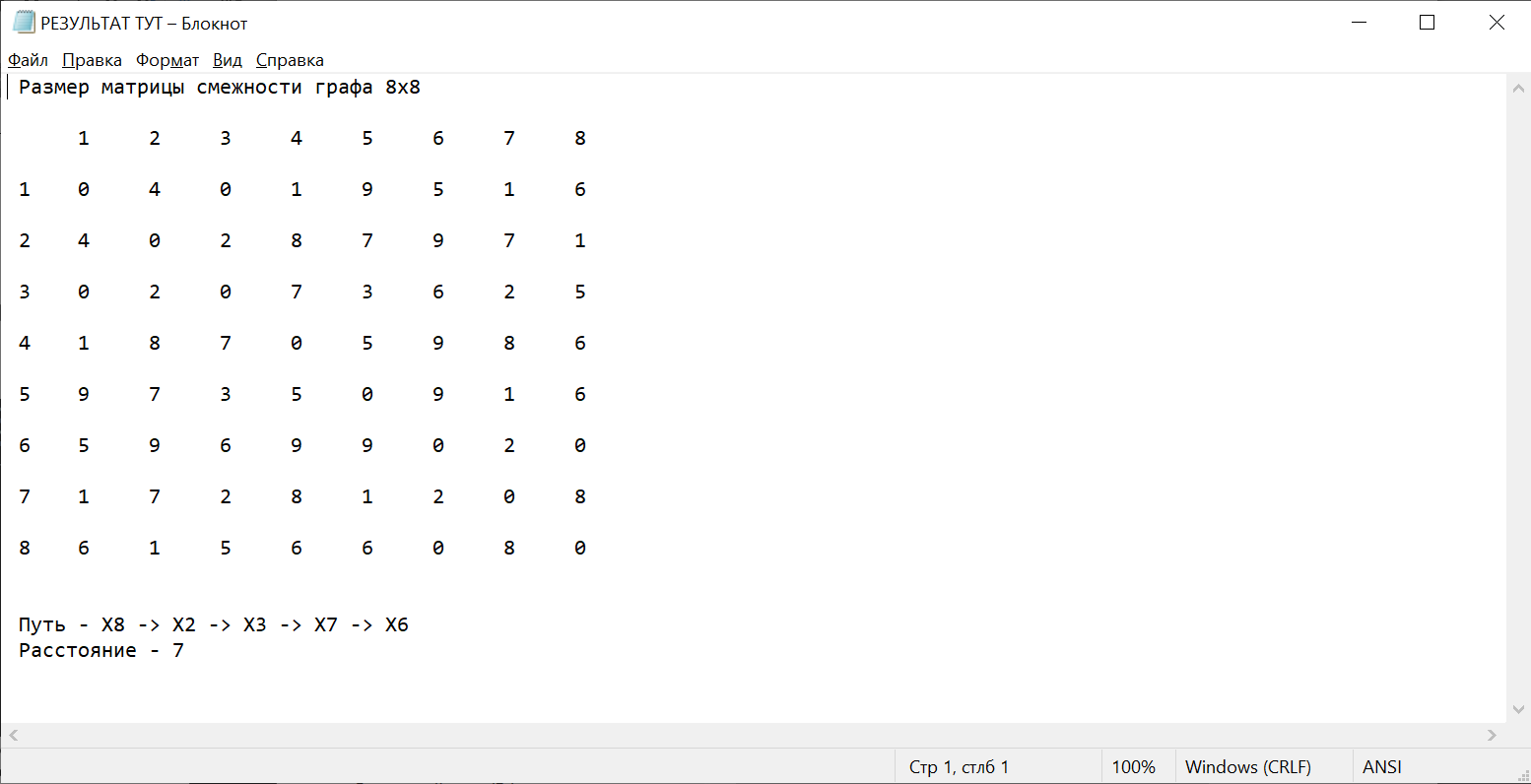
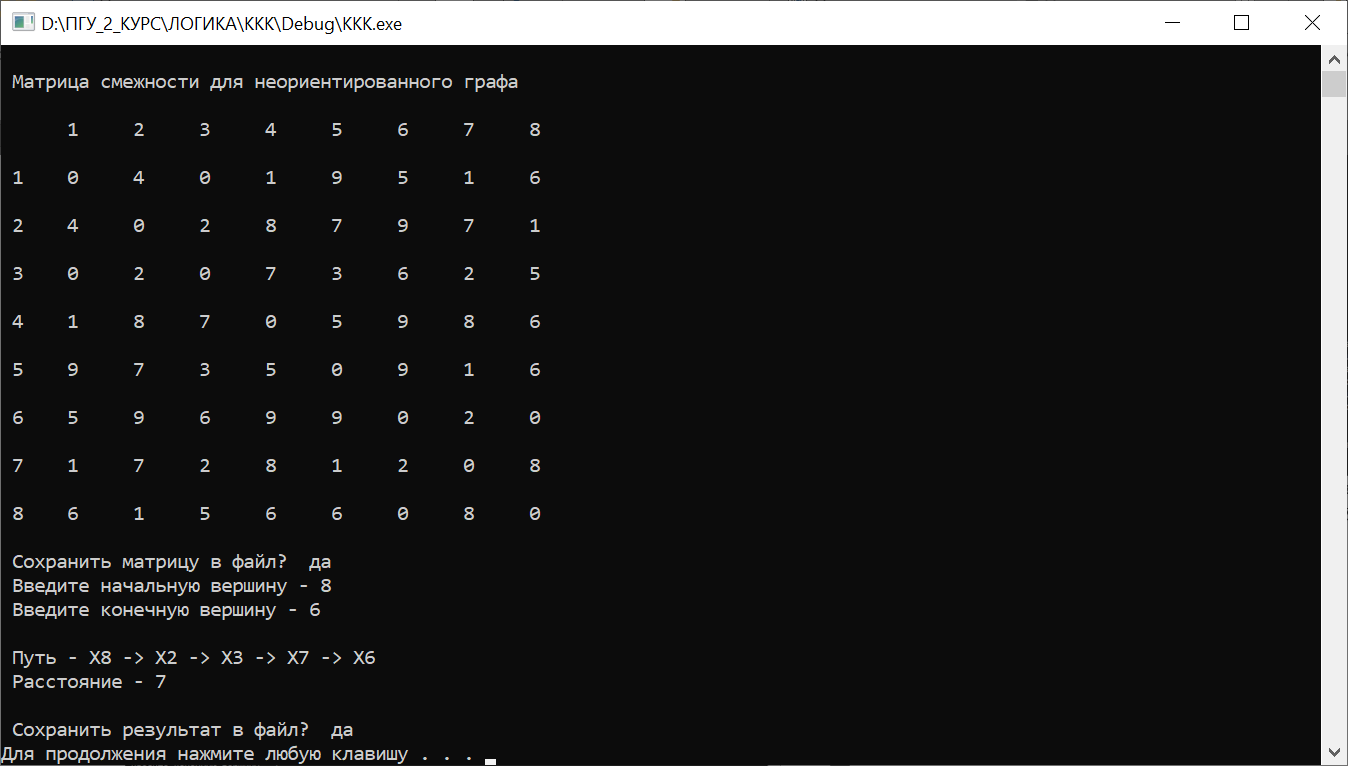
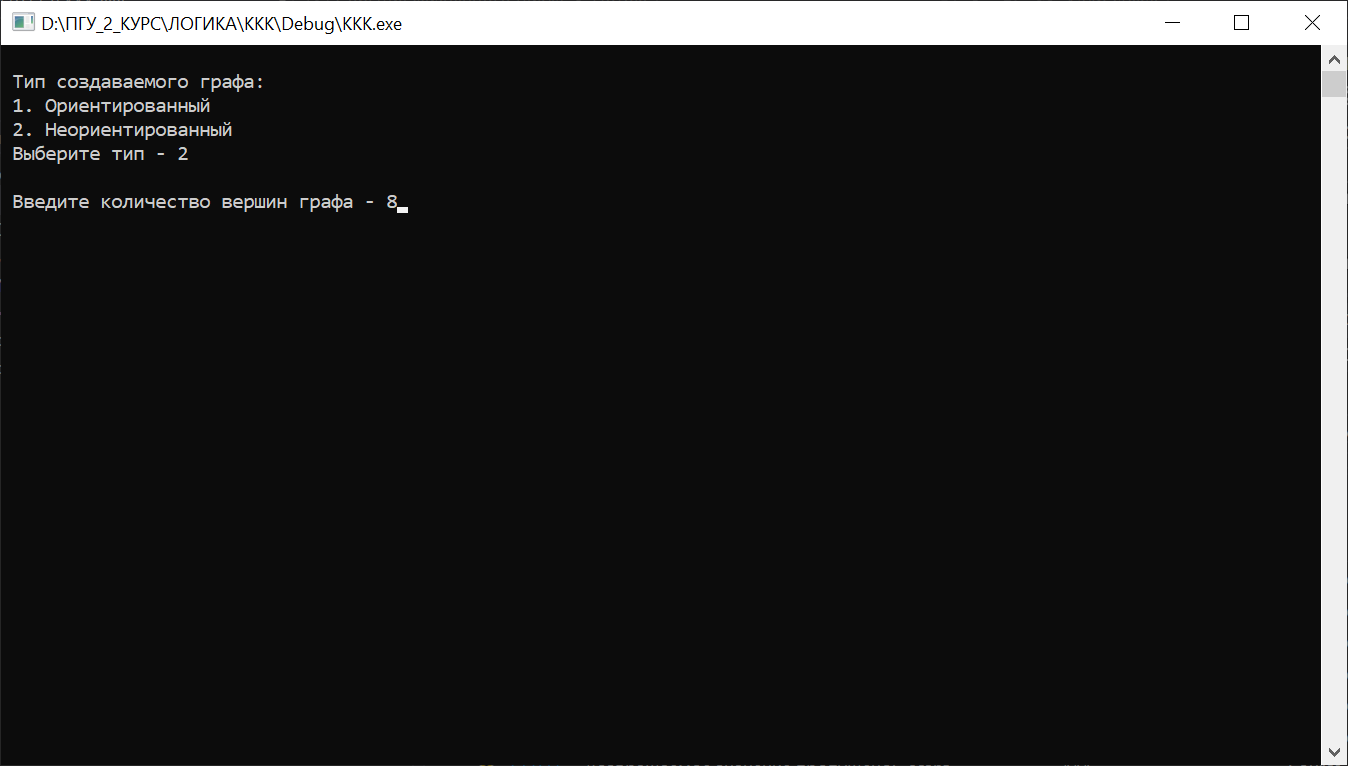


Рисунок 9 – Неориентированный граф из 8 вершин, созданный автоматически, матрица и результат сохранены в файле

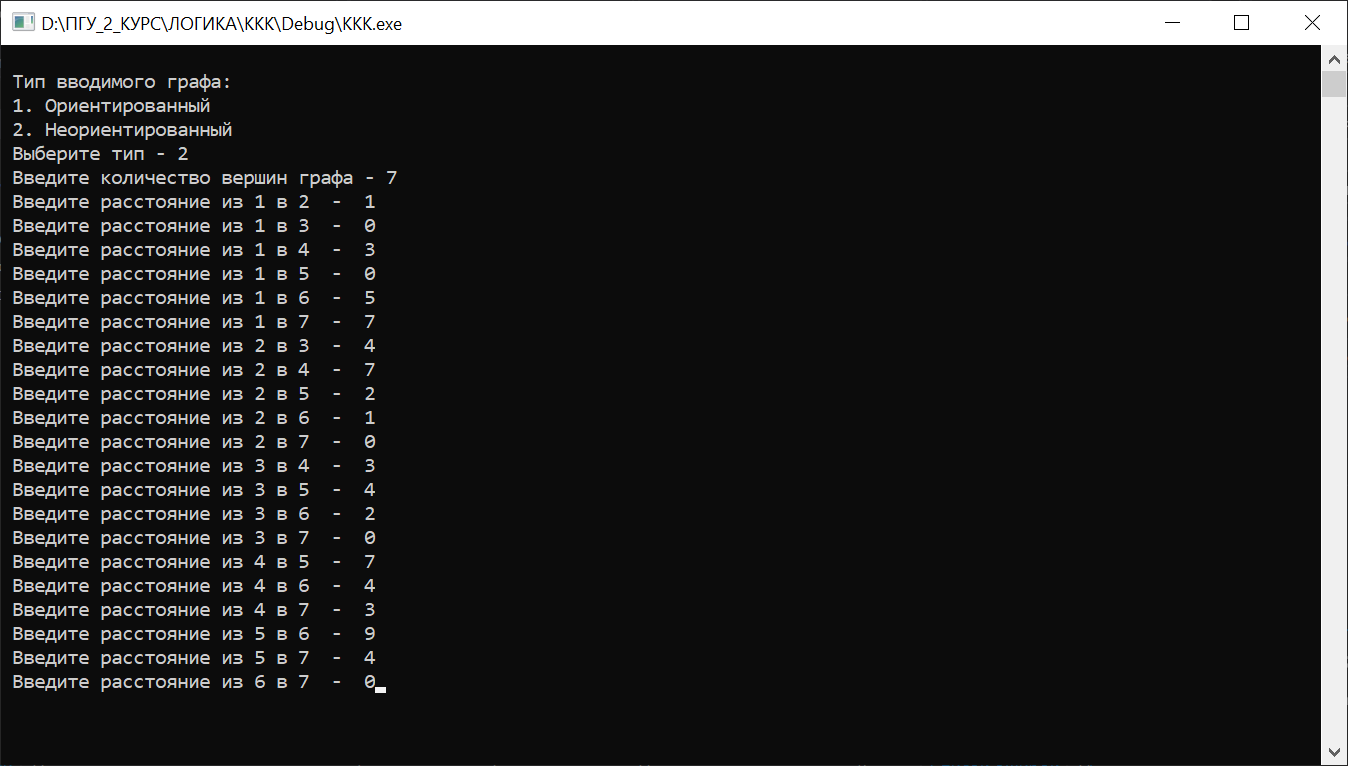
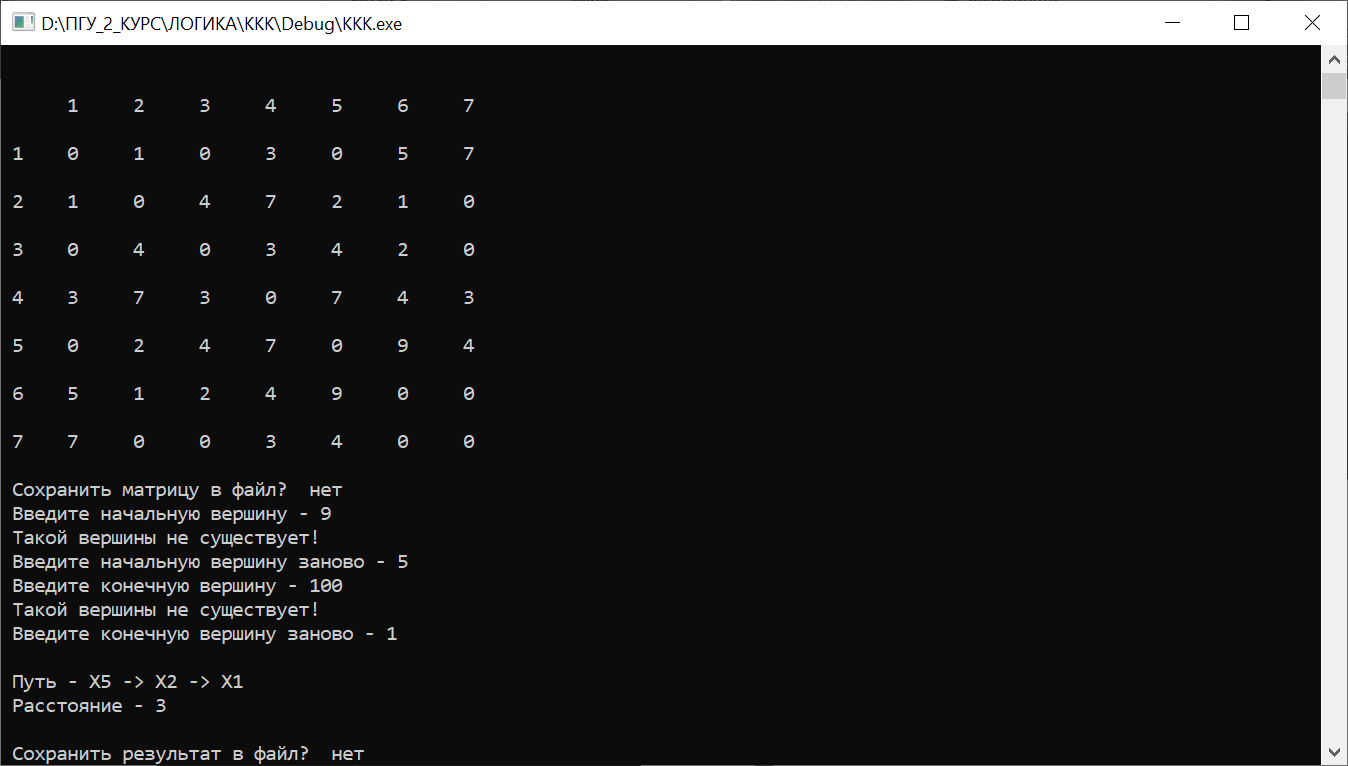
 

Рисунок 10 - Неориентированный граф из 5 вершин, созданный вручную, ни матрица, ни результат не были сохранены (пример выбора несуществующих вершин )

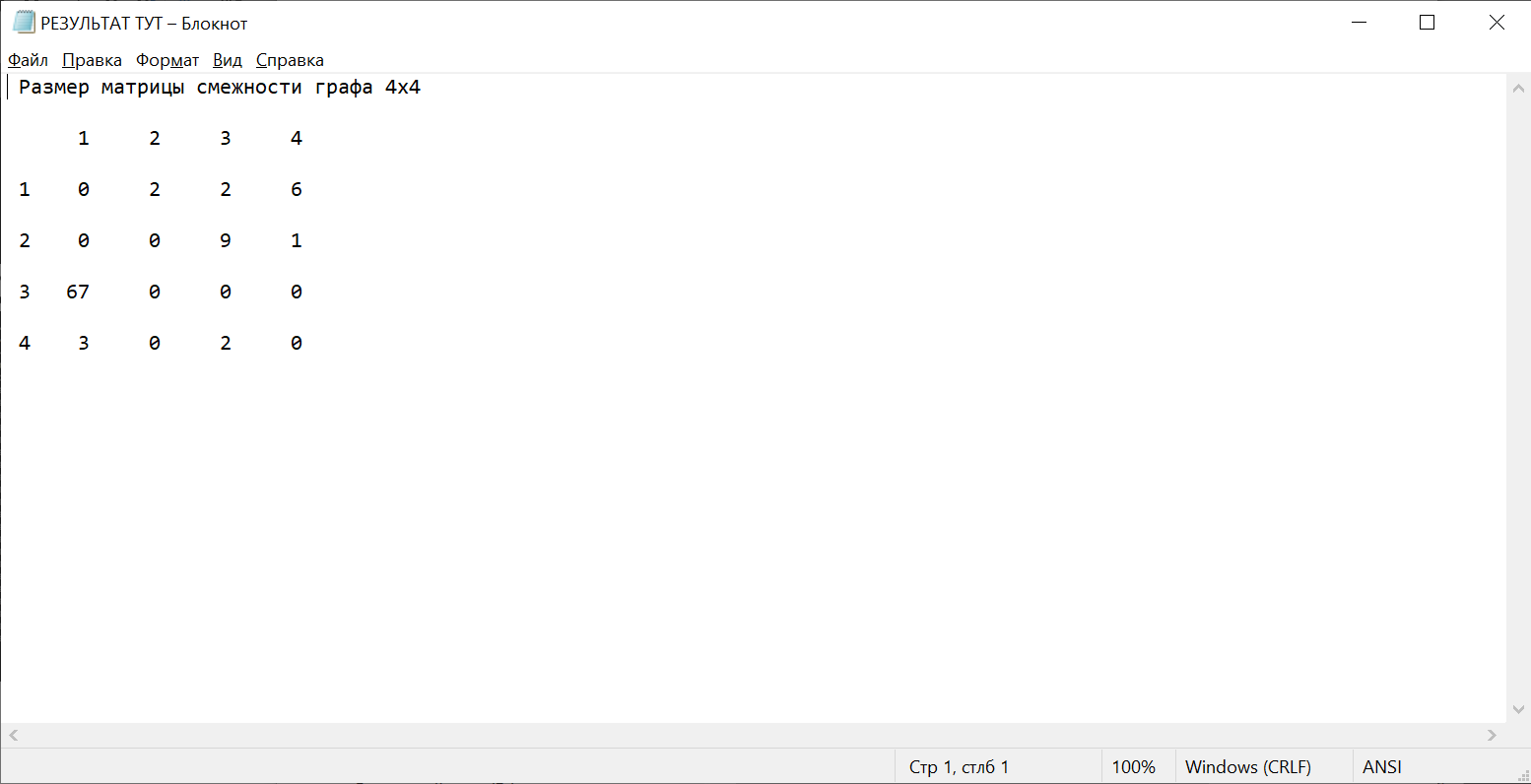
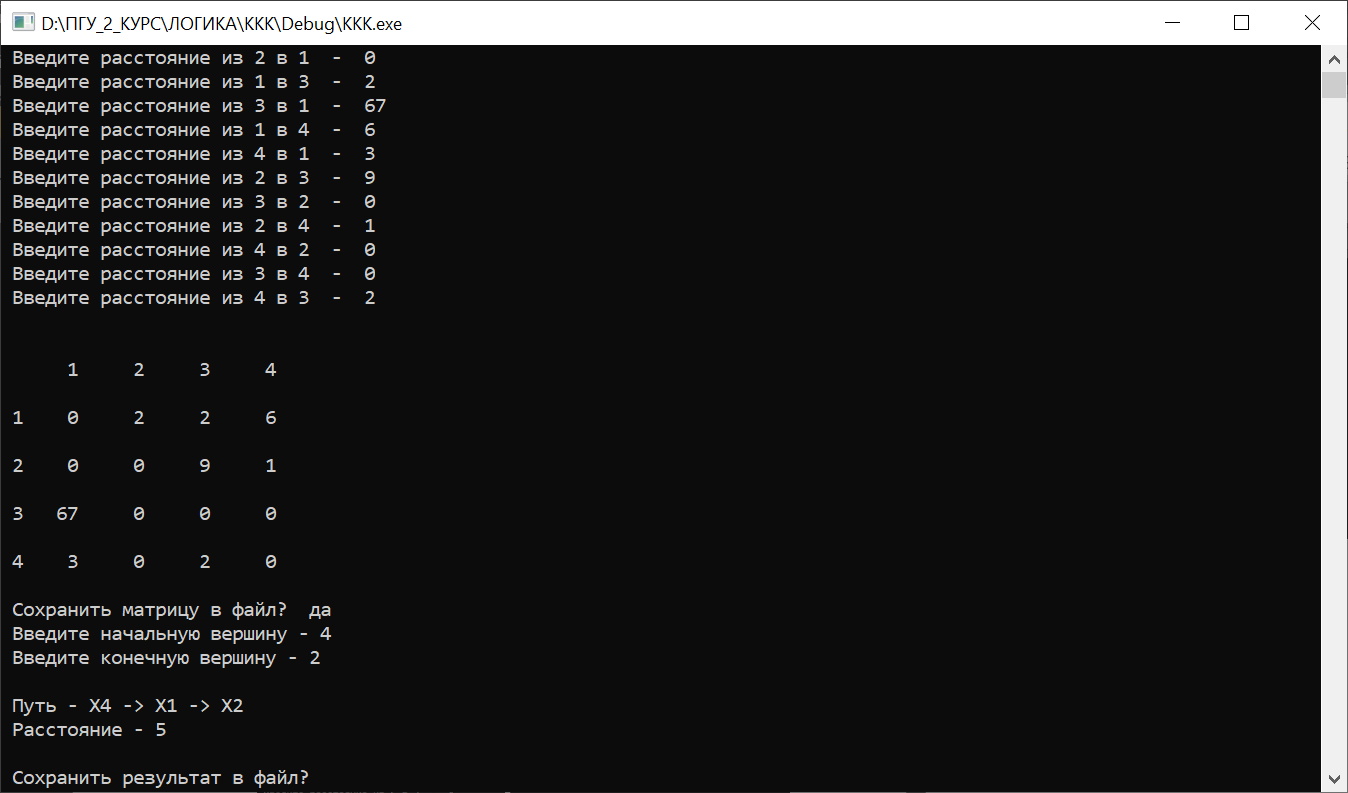
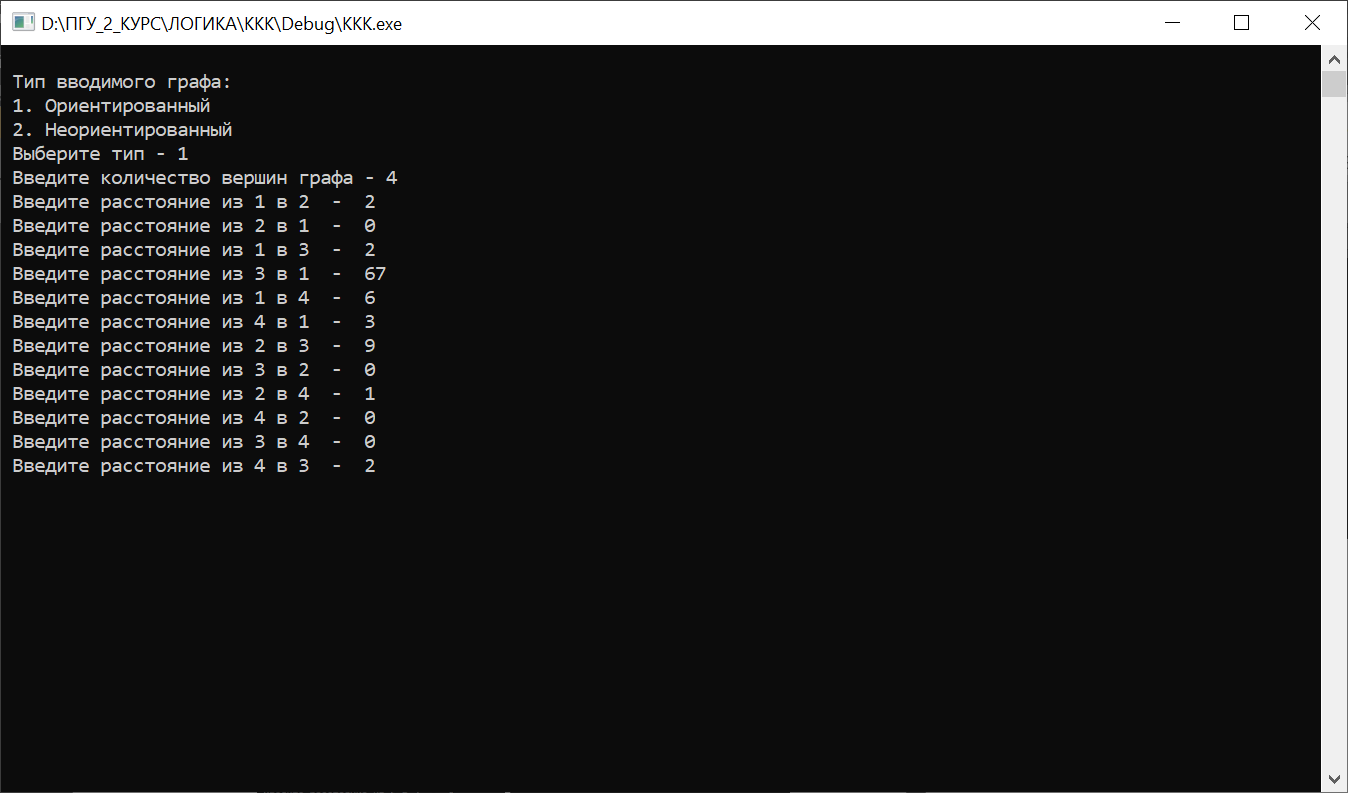


Рисунок 10 - Ориентированный граф из 4 вершин, созданный вручную, в файл была сохранена только матрица

# Ручной расчёт задачи

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

Начальная вершина – 2. Конечная вершина – 5.



Рисунок 11 – Пример матрицы для ручного расчета

Из 2 есть путь в 1(6), в 3(7) и в 4(7). Из 4 нельзя попасть ни в одну из вершин. Ее мы не рассматриваем. Остаются 2 смежные вершины – 1 и 3.

Допустим мы пойдем через вершину 3. Из 3 вершины есть 3 вершины, в которые можно попасть – 2(7), 4(9) и 5(2). Из вершины 2 мы держим путь, из 4 вершины не путей нет, значит можем пойти только в 5. Вот мы в пункте назначения. Путь 2 – 3 – 5; Расстояние 7+2 = 9. Теперь мы знаем, что из 2 в 5 путь точно есть.

Допустим теперь мы пойдем через вершину 1. Из 1 вершины есть 4 вершины, в которые можно попасть – 2(6), 3(5), 4(8) и 5(2). Из вершины 2 мы держим путь, из 4 вершины не путей нет. Значит, мы можем пойти напрямую в 5 или через 3. Если идти через 3 вершину, то получим следующие результаты. Путь 2 – 1 – 3 – 5; Расстояние 6+5+2 = 13. Теперь напрямую в 5 вершину. Путь 2 – 1 – 5; Расстояние 6+2 = 8. Делаем вывод, что путь 2 - 1 – 5 является самым коротким. Вот результат работы программы.



Рисунок 12 – результат работы

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Дейкстры для поиска кратчайшего расстояние 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. Приобретены навыки по осуществлению алгоритма Дейкстры. Углублены знания языков программирования Cи и Си++.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Данной программе не хватает визуальной составляющей, чтобы граф мог быть выведен на экран в виде вершин и ребер. Дело в том, 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 использованных источников

1. https://habr.com/ru/post/65367/
2. http://kspt.icc.spbstu.ru/media/files/2014/course/cpp/slides/18\_Graphs\_v2.pdf
3. http://cppstudio.com
4. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978

## Приложения

## Приложение 1. Листинг программы

## 

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <windows.h>

#include <time.h>

using namespace std;

int i, j, n, p, x\_start, x\_end, CMND, type, visited[ 200];

unsigned int graph[200][200], dist[200];

char Temp[200], way[200][200], save[5], save1[5], ex[5];

char da[] = "да"; char Da[] = "Да";

FILE\* res;

void DIJKSTRA()

{

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

{

visited[i] = 0;

dist[i] = 50000;

}

dist[x\_start] = 0;

visited[x\_start] = 1;

p = x\_start;

\_itoa\_s(x\_start + 1, Temp, 10);

for (i = 1; i <= n; i++)

{

strcpy\_s(way[i], "X");

strcat\_s(way[i], Temp);

}

do

{

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

{

if ((graph[p][i] != 50000) && (!visited[i]) && (i != p))

{

if (dist[i] > dist[p] + graph[p][i])

{

\_itoa\_s(i + 1, Temp, 10);

strcpy\_s(way[i + 1], way[p + 1]);

strcat\_s(way[i + 1], " -> X");

strcat\_s(way[i + 1], Temp);

}

if (dist[i] > dist[p] + graph[p][i])

dist[i] = dist[p] + graph[p][i];

else

dist[i] = dist[i] + 0;

}

}

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

if (!(visited[i]))

p = i;

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

if ((dist[p] > dist[i]) && (!visited[i]))

p = i;

visited[p] = 1;

} while (p != x\_end);

}

void HAND()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("\n Тип вводимого графа:\n 1. Ориентированный\n 2. Неориентированный \n Выберите тип - ");

scanf("%d", &type);

while (type > 2)

{

printf("\n Номер такого типа не существует\n Выберите тип заново - ");

scanf("%d", &type);

}

printf(" Введите количество вершин графа - ");

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

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

{

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

{

graph[i][j] = 0;

}

}

if (type == 1)

{

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

{

for (j = i + 1; j < n; j++)

{

printf(" Введите расстояние из %d в %d - ", i + 1, j + 1);

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

printf(" Введите расстояние из %d в %d - ", j + 1, i + 1);

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

}

}

}

if (type == 2)

{

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

{

for (j = i + 1; j < n; j++)

{

printf(" Введите расстояние из %d в %d - ", i + 1, j + 1);

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

graph[j][i] = graph[i][j];

}

}

}

printf("\n\n ");

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

{

printf("%4d ", j + 1); //горизонт

}

printf("\n\n");

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

{

printf(" %d ", i + 1); // вертик

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

{

printf("%4d ", graph[i][j]);

}

printf("\n\n");

}

printf(" Сохранить матрицу в файл? ");

scanf("%s", &save);

if ((strcmp(da, save) == 0) || (strcmp(Da, save) == 0))

{

res = fopen("РЕЗУЛЬТАТ ТУТ.txt", "w");

fprintf(res, " Размер матрицы смежности графа %dx%d \n\n ", n, n);

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

{

fprintf(res, "%4d ", j + 1); //горизонт

}

fprintf(res, "\n\n");

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

{

fprintf(res, " %d ", i + 1); // вертик

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

{

fprintf(res, "%4d ", graph[i][j]);

}

fprintf(res, "\n\n");

}

fclose(res);

}

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

{

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

{

if (graph[i][j] == 0) graph[i][j] = 50000;

}

}

printf(" Введите начальную вершину - ");

scanf("%d", &x\_start);

while (x\_start > n)

{

printf(" Такой вершины не существует! \n Введите начальную вершину заново - ");

scanf("%d", &x\_start);

}

printf(" Введите конечную вершину - ");

scanf("%d", &x\_end);

while (x\_end > n)

{

printf(" Такой вершины не существует!\n Введите конечную вершину заново - ");

scanf("%d", &x\_end);

}

while (x\_start == x\_end)

{

printf(" Начальная и конечная вершины совпадают\n");

printf(" Введите начальную вершину заново - ");

scanf("%d", &x\_start);

while (x\_start > n)

{

printf(" Такой вершины не существует!\n Введите начальную вершину заново - ");

scanf("%d", &x\_start);

}

printf(" Введите конечную вершину заново - ");

scanf("%d", &x\_end);

while (x\_end > n)

{

printf(" Такой вершины не существует!\n Введите конечную вершину заново - ");

scanf("%d", &x\_end);

}

}

x\_end--;

x\_start--;

DIJKSTRA();

if (dist[p] != 50000)

{

printf("\n Путь - %s", way[p + 1]);

printf("\n");

printf(" Расстояние - %u \n\n", dist[p]);

}

else

printf("\n Пути из %d в %d не существует\n\n ", x\_start + 1, x\_end + 1);

printf(" Сохранить результат в файл? ");

scanf("%s", &save1);

if ((strcmp(da, save1) == 0) || (strcmp(Da, save1) == 0))

{

res = fopen("РЕЗУЛЬТАТ ТУТ.txt", "a");

fprintf(res, "\n Путь - %s", way[p + 1]);

fprintf(res, "\n Расстояние - %u", dist[p]);

fclose(res);

}

\_getch;

}

void RAND()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

int R;

printf("\n Тип создаваемого графа:\n 1. Ориентированный\n 2. Неориентированный \n Выберите тип - ");

scanf("%d", &type);

while (type > 2)

{

printf("\n Номер такого типа не существует\n Выберите тип заново - ");

scanf("%d", &type);

}

printf("\n Введите количество вершин графа - ");

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

system("cls");

printf("\n");

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

{

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

{

graph[i][j] = 0;

}

}

if (type == 1) {

printf(" Матрица смежности для ориентированного графа\n");

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

{

for (j = i; j < n; ++j)

{

R = rand() % 100;

if (R > 30)

{

graph[i][j] = rand() % 10;

graph[j][i] = 0;

graph[i][i] = graph[j][j] = 0; // чтобы петля(узел) не создавалась

}

else

graph[i][j] = graph[j][i] = rand() % 10;

graph[i][i] = graph[j][j] = 0; // чтобы петля(узел) не создавалась

}

}

}

if (type == 2) {

printf(" Матрица смежности для неориентированного графа\n");

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

{

for (j = i; j < n; ++j)

{

graph[i][j] = graph[j][i] = rand() % 10;

graph[i][i] = graph[j][j] = 0; // чтобы петля(узел) не создавалась

}

}

}

printf("\n ");

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

{

printf("%4d ", j + 1); //горизонт

}

printf("\n\n");

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

{

printf(" %d ", i + 1); // вертик

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

{

printf("%4d ", graph[i][j]);

}

printf("\n\n");

}

printf(" Сохранить матрицу в файл? ");

scanf("%s", &save);

if ((strcmp(da, save) == 0) || (strcmp(Da, save) == 0))

{

res = fopen("РЕЗУЛЬТАТ ТУТ.txt", "w");

fprintf(res, " Размер матрицы смежности графа %dx%d \n\n ", n, n);

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

{

fprintf(res, "%4d ", j + 1); //горизонт

}

fprintf(res, "\n\n");

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

{

fprintf(res, " %d ", i + 1); // вертик

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

{

fprintf(res, "%4d ", graph[i][j]);

}

fprintf(res, "\n\n");

}

fclose(res);

}

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

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

if (graph[i][j] == 0) graph[i][j] = 50000; //бесконечность

printf(" Введите начальную вершину - ");

scanf("%d", &x\_start);

while (x\_start > n)

{

printf(" Такой вершины не существует! \n Введите начальную вершину заново - ");

scanf("%d", &x\_start);

}

printf(" Введите конечную вершину - ");

scanf("%d", &x\_end);

while (x\_end > n)

{

printf(" Такой вершины не существует!\n Введите конечную вершину заново - ");

scanf("%d", &x\_end);

}

while(x\_start == x\_end)

{

printf(" Начальная и конечная вершины совпадают\n");

printf(" Введите начальную вершину заново - ");

scanf("%d", &x\_start);

while (x\_start > n)

{

printf(" Такой вершины не существует!\n Введите начальную вершину заново - ");

scanf("%d", &x\_start);

}

printf(" Введите конечную вершину заново - ");

scanf("%d", &x\_end);

while (x\_end > n)

{

printf(" Такой вершины не существует!\n Введите конечную вершину заново - ");

scanf("%d", &x\_end);

}

}

x\_end--;

x\_start--;

DIJKSTRA();

if (dist[p] != 50000)

{

printf("\n Путь - %s", way[p + 1]);

printf("\n");

printf(" Расстояние - %u \n\n", dist[p]);

}

else

printf("\n Пути из %d в %d не существует\n\n", x\_start + 1, x\_end + 1);

printf(" Сохранить результат в файл? ");

scanf("%s", &save1);

res = fopen("РЕЗУЛЬТАТ ТУТ.txt", "a");

if((strcmp(da, save1) == 0) || (strcmp(Da, save1) == 0))

{

fprintf(res, "\n Путь - %s", way[p + 1]);

fprintf(res, "\n Расстояние - %u", dist[p]);

}

fclose(res);

\_getch;

}

void MENU()

{

system("cls");

printf("\n Меню программы ");

printf("\n 1. Создать граф автоматически");

printf("\n 2. Создать граф вручную");

printf("\n 3. Завершение работы");

printf("\n\n Введите номер команды - ");

scanf("%d", &CMND);

while( CMND > 3)

{

system("cls");

printf("\n Меню программы ");

printf("\n 1. Создать граф автоматически");

printf("\n 2. Создать граф вручную");

printf("\n 3. Завершение работы");

printf("\n\n Введите номер команды - ");

scanf("%d", &CMND);

}

system("cls");

return;

}

void TITLE()

{

printf("\n Курсовая работа по дисциплине");

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

printf("\n На тему \"Реализация алгоритма Дейкстры\"");

printf("\n\n Выполнил студент группы 19ВВ2");

printf("\n Отставнов А.М.");

printf("\n\n Приняли");

printf("\n Митрохин М.А.\n Юрова О.В.");

printf("\n\n ");

system("pause");

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

TITLE();

do {

MENU();

switch (CMND)

{

case 1:

RAND();

break;

case 2:

HAND();

break;

case 3:

return 0;

}

if (CMND != 3)

system("pause");

} while (CMND != 3);

}