

Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет Кафедра «Вычислительной
техники»

Пояснительная записка
к курсовому проектированию
по дисциплине: «Логика и основы алгоритмизации в инженерных задачах»
на тему: «Реализация алгоритма Дейкстры»

27.12.24
хорошо
фва

Выполнил студент группы 23ВВВ2:
Федоров Б.М.
Приняли:
Митрохин М.А.
Юрова О.В.

Пенза 2024

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ _____

« _____ » _____ 20__

ЗАДАНИЕ

на курсовое проектирование по курсу

"Логика и основы алгоритмизации в инженерных задачах"

Студенту Федорову Борису Михайловичу

Группа 23BAВ2

Тема проекта Реализация алгоритма Дейкстры

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данными заданием курсового проекта.

Пояснительная записка должна содержать:

1. Постановку задачи;
2. Теоретическую часть задания;
3. Описание алгоритма решаемой задачи;
4. Пример ручного расчета задачи и вычисления (на небольшом участке работы алгоритма);
5. Описание самой программы;
6. Тесты;
7. Список литературы;
8. Истории программы;
9. Результаты работы программы.

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

Работа алгоритма в формате блок-схемы

3. Экспериментальная часть

Тестирование программы

Результат работы программы на тестовых данных

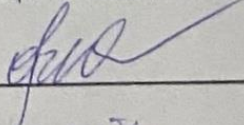
Срок выполнения проекта по разделам

- 1 Исследование теоретической части курсового
- 2 Разработка алгоритмов программы
- 3 Разработка программы
- 4 Тестирование и завершение разработки программы
- 5 Оформление пояснительной записки
- 6
- 7
- 8

Дата выдачи задания “ 6 ” сентября 2024

Дата защиты проекта “ ”

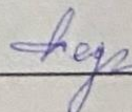
Руководитель Юрова О.В.



Задание получил “ 6 ” сентября

20 24 г.

Студент Федоров Б.И.



Содержание

Реферат.....	3
Введение	4
1 Постановка задачи.....	6
2 Теоретическая часть программы.....	7
3 Описание алгоритма программы.	8
4 Описание программы.....	10
5 Тестирование.	15
6 Ручной расчет задачи.	19
Список литературы	22
Приложение А.....	23
Код программы.....	23
Приложение В.	39
Результаты работы программы.	39

Реферат.

Отчет 35 стр., 17 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ДЕЙКСТРЫ.

Цель исследования – разработка программы, способная находить кратчайшие расстояния до всех вершин графа с помощью алгоритма Дейкстры.

В работе рассматривается алгоритм, который позволяет находить кратчайшее расстояние от одной вершины графа до остальных.

Введение.

Теория графов — раздел дискретной математики, изучающий графы, одна из ветвей топологии. В самом общем смысле граф — это множество точек (вершин, узлов), которые соединяются множеством линий (рёбер, дуг).

Теория графов (то есть систем линий, соединяющих заданные точки) включена в учебные программы для начинающих математиков, поскольку:

1 как и геометрия, обладает наглядностью;

2 как и теория чисел, проста в объяснении и имеет сложные нерешённые задачи;

3 не имеет громоздкого математического аппарата («комбинаторные методы нахождения нужного упорядочения объектов существенно отличаются от классических методов анализа поведения систем с помощью уравнений»);

4 имеет выраженный прикладной характер.

На протяжении более сотни лет развитие теории графов определялось в основном проблемой четырёх красок. Решение этой задачи в 1976 году оказалось поворотным моментом истории теории графов, после которого произошло её развитие как основы современной прикладной математики.

Универсальность графов незаменима при проектировании и анализе коммуникационных сетей.

Теория графов, как математическое орудие, приложима как к наукам о поведении (теории информации, кибернетике, теории игр, теории систем, транспортным сетям), так и к чисто абстрактным дисциплинам (теории множеств, теории матриц, теории групп и так далее).

Несмотря на разнообразные, усложнённые, малопонятные и специализированные термины многие модельные (схемные, структурные) и конфигурационные проблемы переформулируются на языке теории графов, что позволяет значительно проще выявить их концептуальные трудности.

Часто требуется производить различные операции над графами, например нахождение кратчайшего расстояния между вершинами. Для этого придуманы различные алгоритмы, один из которых Алгоритм Дейкстры.

Алгоритм Дейкстры (англ. Dijkstra's algorithm) — алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании, например, его используют протоколы маршрутизации OSPF и IS-IS.

В то время Дейкстра искал способ продемонстрировать возможности нового компьютера ARMAC и искал задачу, которую мог бы решить ARMAC и при этом понятную незнакомым с компьютерами людям.

Дейкстра взял задачу поиска кратчайшего пути и разработал алгоритм её решения. На базе алгоритма он разработал программу построения маршрутов между городами по транспортной карте Нидерландов.

Алгоритм Дейкстры относится к так называемым «жадным» алгоритмам. То есть, алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, язык программирования – Си.

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 пути от одной вершины графа до всех остальных.

1 Постановка задачи.

Разработка программы, которая выполняет поиск минимального пути от одной вершины к другим, с помощью алгоритма Дейкстры.

В программе должно быть текстовое меню.

Также должна быть реализована возможность выбора случайного или ручного задания графа.

Ещё пользователь должен иметь возможность выбрать, ориентированный это граф или нет.

Должна быть предусмотрена возможность сохранить результат работы программы в файл.

2 Теоретическая часть программы.

Данный алгоритм способен найти кратчайшее расстояние от одной вершины до всех других, при условии, что граф не содержит рёбер с отрицательным весом.

Изначально каждой вершине приписывается вес – это вес пути от начальной вершины до данной. Исследуя граф, алгоритм находит для каждой вершины маршрут, и, если он оказывается кратчайшим, выделяет вершину. Весом данной вершины становится вес пути. Для всех соседних вершин алгоритм также рассчитывает вес, при этом не выделяя их. Алгоритм заканчивает свою работу, когда доходит до последней вершины.

Теоретическое описание работы алгоритма Дейкстры по шагам:

Шаг 1. Всем вершинам, кроме начальной, присваивается вес равный 10000, а первой вершине – 0.

Шаг 2. каждая вершина отмечается как не посещённая.

Шаг 3. Первая выбранная (начальная) вершина «s» объявляется текущей.

Шаг 4. Вес всех невыделенных вершин пересчитывается по формуле: вес невыделенной вершины это

минимальное число из старого веса данной вершины, суммы веса текущей вершины и веса ребра, соединяющего текущую вершину с невыделенной.

Шаг 5. Среди невыделенных вершин ищется вершина с минимальным весом. Если такая не найдена, значит вес всех вершин равен 10000, то данный маршрут не существует. После этого происходит выход. Если найдена, то текущей становится найденная вершина и она выделяется.

Шаг 6. Если текущей вершиной оказывается конечная, то путь найден, и его вес становится весом конечной вершины.

Шаг 7. Возвращение на 4 шаг.

3 Описание алгоритма программы.

Задаётся ориентированный либо же неориентированный взвешенный граф с «n» вершинами. Обязательно учтём, что вес всех рёбер неотрицателен. Указана некоторая стартовая вершина «s». Необходимо определить длины кратчайших путей из вершины «s» во все остальные вершины, а также предоставить способ вывода самих кратчайших путей.

При представлении графа матрицей смежности веса рёбер хранятся в матрице, где наличие пути из одной вершины в другую обозначается числом

>0 . Если же в матрице значение $= 0$, то такого пути нет.

Расстояние от начальной вершины «s» до вершины «i» хранится в массиве

«minimalrast[i]». Начальные значения «minimalrast[s]=0», min=10000 для всех остальных вершин «i». В самом начале алгоритм знает путь только из вершины

«s» до вершины «s» $= 0$, до остальных вершин кратчайшие пути пока неизвестны. Алгоритм пошагово изменяет (улучшает) значения в массиве

«minimalrast», в итоге находит кратчайшие расстояния до всех вершин. Алгоритм заканчивается, когда все вершины посещены или если до них нельзя дойти (вершины изолированы).

```
функция Алгоритм(G, start_index, n, min, s, pos, minimalrast, temp):
minimalrast[start_index] = 0
повторять:
min = 10000
s = 10000
для i от 0 до n:
если (pos[i] == -1) И (minimalrast[i] < min): min = minimalrast[i]
s = i
если (s != 10000):
для i от 0 до n:
```

```
если (G[s][i] > 0):  
temp = min + G[s][i]  
если (temp < minimalrast[i]): minimalrast[i] = temp  
pos[s] = 0  
пока (s < 10000)
```

```
печать("\nКратчайшие расстояния от вершины ", start_index + 1, " до остальных  
вершин\n")  
записьВФайл("\nКратчайшие расстояния от вершины ", start_index + 1, " до остальных  
вершин\n")
```

```
для i от 0 до n:  
печать("%7d ", minimalrast[i])  
записьВФайл("%7d ", minimalrast[i])
```

4 Описание программы.

Для написания программы использован язык программирования Си.

Данная программа является многомодульной, поскольку состоит из нескольких функций:

Работа программы начинается с вывода титульного листа на экран, что видно из рисунка 1:

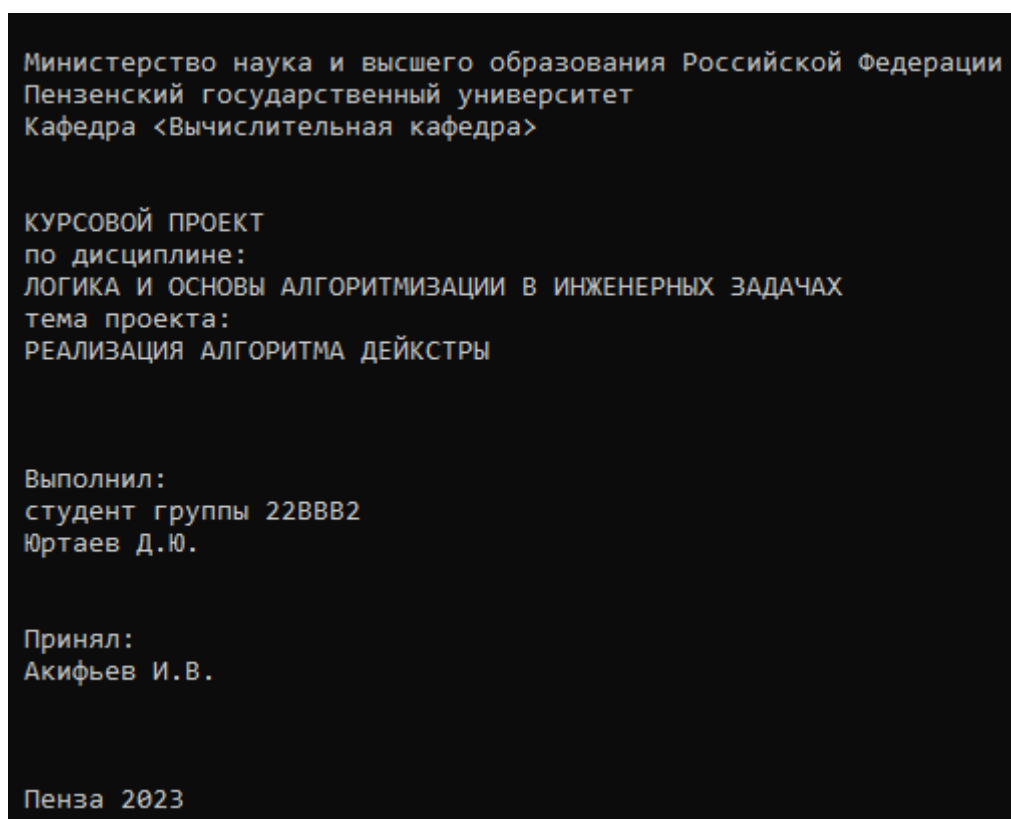


Рисунок 1 - Титульный лист работы

Далее на экран выводится текстовое меню (Рисунок 2):

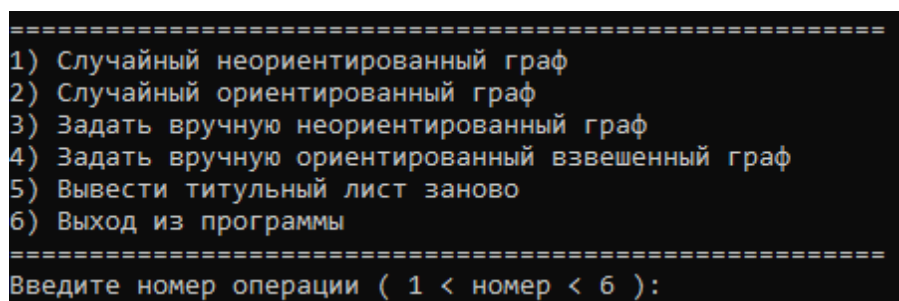


Рисунок 2 - Меню программы

Пользователь может сгенерировать граф случайно или ввести вручную.

При создании случайного графа у пользователя запрашивается его размер. Размер можно ввести, используя клавиатуру. После вывода графа на экран запрашивается номер стартовой вершины, как указано на рисунке 3:

```
Введите размер графа: 5
  1  2  3  4  5
1  0  0  0 10  9
2  0  0  8  7 13
3  0  8  0  0  0
4 10  7  0  0  0
5  9 13  0  0  0
Введите номер вершины: 3
```

Рисунок 3 - Вывод случайно сгенерированного неориентированного взвешенного графа на экран

Как видно из рисунка 4, выводятся кратчайшие расстояния от стартовой вершины до всех остальных:

```
Введите размер графа: 5
  1  2  3  4  5
1  0  0  0 10  9
2  0  0  8  7 13
3  0  8  0  0  0
4 10  7  0  0  0
5  9 13  0  0  0
Введите номер вершины: 3
Кратчайшие расстояния от вершины 3 до остальных вершин
  25   8   0  15  21
Для продолжения нажмите любую кнопку... █
```

Рисунок 4 - Результат случайно сгенерированного неориентированного взвешенного графа

При задании графа вручную сначала нужно ввести размер графа(Рисунок 5):

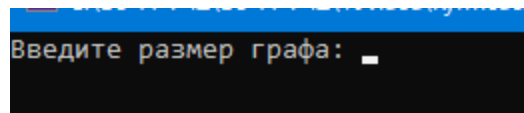


Рисунок 5 - Ввод размера графа

После запрашиваются расстояния между вершинами, что видно на рисунке 6:

```
Введите размер графа: 5
Введите расстояние 1 - 2: 3
Введите расстояние 1 - 3: 7
Введите расстояние 1 - 4: 0
Введите расстояние 1 - 5: 3
Введите расстояние 2 - 3: 5
Введите расстояние 2 - 4: 7
Введите расстояние 2 - 5: 0
Введите расстояние 3 - 4: 8
Введите расстояние 3 - 5: 2
Введите расстояние 4 - 5: 0
```

	1	2	3	4	5
1	0	3	7	0	3
2	3	0	5	7	0
3	7	5	0	8	2
4	0	7	8	0	0
5	3	0	2	0	0

Рисунок 6 - Ввод веса ребер и вывод неориентированного взвешенного графа

После вывода графа на экран запрашивается стартовая вершина и происходит вывод кратчайшие расстояния на экран(Рисунок 7):

```

Введите размер графа: 5
Введите расстояние 1 - 2: 3
Введите расстояние 1 - 3: 7
Введите расстояние 1 - 4: 0
Введите расстояние 1 - 5: 3
Введите расстояние 2 - 3: 5
Введите расстояние 2 - 4: 7
Введите расстояние 2 - 5: 0
Введите расстояние 3 - 4: 8
Введите расстояние 3 - 5: 2
Введите расстояние 4 - 5: 0
    1    2    3    4    5

    1    0    3    7    0    3
    2    3    0    5    7    0
    3    7    5    0    8    2
    4    0    7    8    0    0
    5    3    0    2    0    0

Введите номер вершины: 3
Кратчайшие расстояния от вершины 3 до остальных вершин
    5        5        0        8        2
Для продолжения нажмите любую кнопку...

```

Рисунок 7 - Результат неориентированного взвешенного графа, введенного вручную

На рисунке 8 показано, как после успешного завершения программы происходит сохранение результата в текстовый документ. Данные в нем обновляются каждый раз после завершения программы.

результат.txt – Блокнот

Файл	Правка	Формат	Вид	Справка			
	1	2	3	4	5	6	7
1	0	5	0	4	4	3	1
2	5	0	0	13	8	0	0
3	0	0	0	0	0	7	0
4	4	13	0	0	0	12	11
5	4	8	0	0	0	3	0
6	3	0	7	12	3	0	0
7	1	0	0	11	0	0	0

Кратчайшие расстояния от вершины 5 до остальных вершин

4	8	10	8	0	3	5
---	---	----	---	---	---	---

Рисунок 8 - Сохранение графа в файл

Также пользователь может вывести титульный лист заново, используя 5 пункт программы.

5 Тестирование.

Тестирование проводилось и в процессе разработки, и после написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, выводом данных в файл, алгоритмом программы, взаимодействием функций.

Ниже на рисунках 9 - 13 представлены результаты работы программы.

```
Введите размер графа: 5
Введите расстояние 1 - 2: 2
Введите расстояние 1 - 3: 5
Введите расстояние 1 - 4: 8
Введите расстояние 1 - 5: 0
Введите расстояние 2 - 3: 3
Введите расстояние 2 - 4: 5
Введите расстояние 2 - 5: 8
Введите расстояние 3 - 4: 2
Введите расстояние 3 - 5: 0
Введите расстояние 4 - 5: 3
    1    2    3    4    5

    1    0    2    5    8    0
    2    2    0    3    5    8
    3    5    3    0    2    0
    4    8    5    2    0    3
    5    0    8    0    3    0

Введите номер вершины: 3
Кратчайшие расстояния от вершины 3 до остальных вершин
    5        3        0        2        5
Для продолжения нажмите любую кнопку..._
```

Рисунок 9 - Результат неориентированного графа, введенного вручную

```
Введите размер графа: 5
    1    2    3    4    5

    1    0    6    2    0   10
    2    6    0    0    0   11
    3    2    0    0    0    8
    4    0    0    0    0    1
    5   10   11    8    1    0

Введите номер вершины: 3
Кратчайшие расстояния от вершины 3 до остальных вершин
    2        8        0        9        8
Для продолжения нажмите любую кнопку...
```

Рисунок 10 - Результат случайно сгенерированного неориентированного графа


```

Введите размер графа: 5
      1   2   3   4   5
1   0   7   6   0   0
2   0   0   8   0   5
3   6   8   0   3   0
4   0   0   3   0   0
5   2   5   9   0   0

Введите номер вершины: 3
Кратчайшие расстояния от вершины 3 до остальных вершин
      6       8       0       3       13
Для продолжения нажмите любую кнопку...

```

Рисунок 11 - Результат случайно сгенерированного ориентированного графа

```

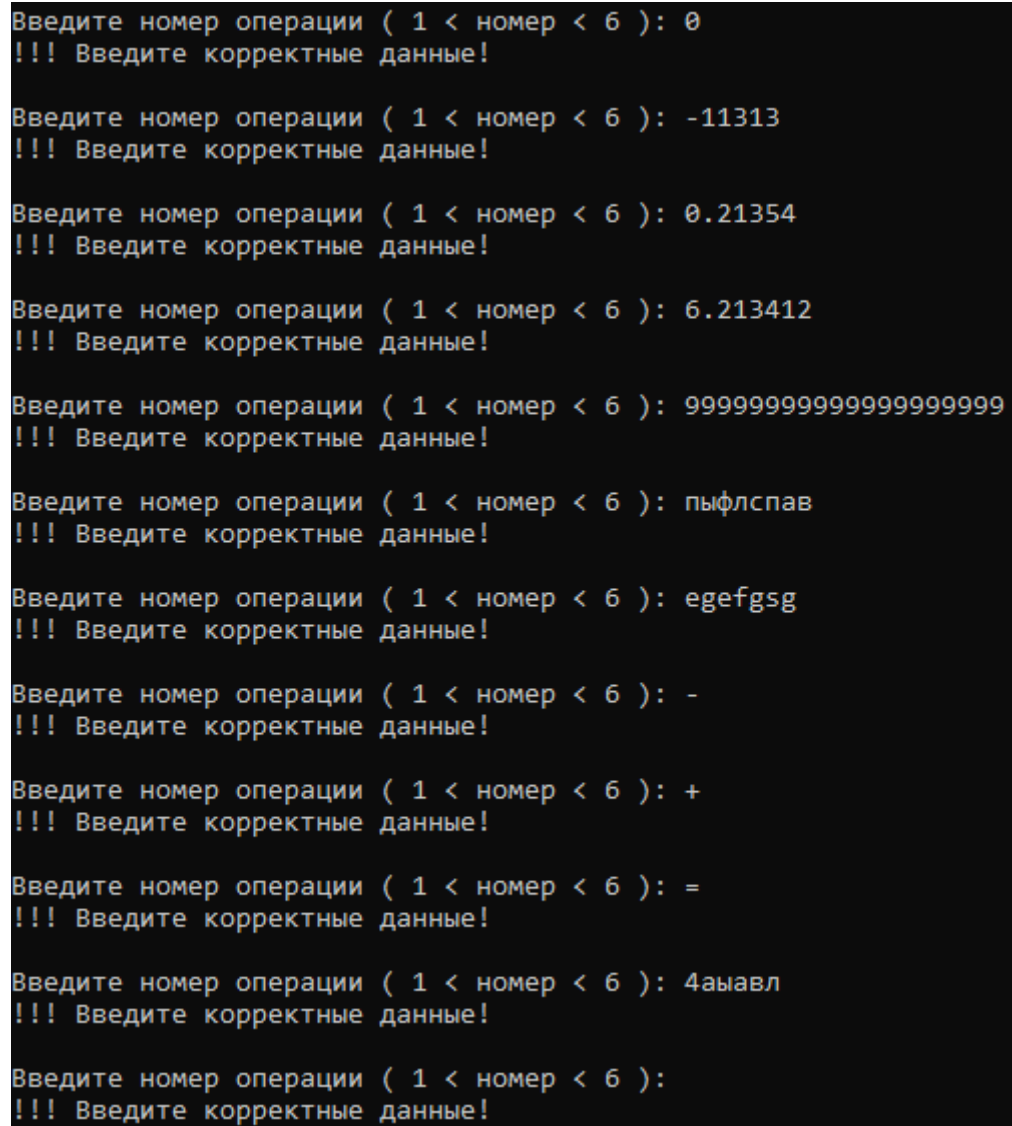
Введите размер графа: 5
Введите расстояние 1 - 2: 3
Введите расстояние 1 - 3: 0
Введите расстояние 1 - 4: 9
Введите расстояние 1 - 5: 8
Введите расстояние 2 - 1: 0
Введите расстояние 2 - 3: 4
Введите расстояние 2 - 4: 6
Введите расстояние 2 - 5: 3
Введите расстояние 3 - 1: 0
Введите расстояние 3 - 2: 5
Введите расстояние 3 - 4: 2
Введите расстояние 3 - 5: 0
Введите расстояние 4 - 1: 1
Введите расстояние 4 - 2: 7
Введите расстояние 4 - 3: 8
Введите расстояние 4 - 5: 5
Введите расстояние 5 - 1: 0
Введите расстояние 5 - 2: 3
Введите расстояние 5 - 3: 2
Введите расстояние 5 - 4: 0
      1   2   3   4   5
1   0   3   0   9   8
2   0   0   4   6   3
3   0   5   0   2   0
4   1   7   8   0   5
5   0   3   2   0   0

Введите номер вершины: 3
Кратчайшие расстояния от вершины 3 до остальных вершин
      3       5       0       2       7
Для продолжения нажмите любую кнопку...

```

Рисунок 12 - Результат ориентированного графа, введенного вручную

Также в программе предусмотрена проверка на некорректный ввод данных. При попытке ввода пользователем некорректного значения (такого, с каким программа не должна работать) на экране отображается сообщение об ошибке и запрашивается повторный ввод в соответствии с рисунком 15.



```
Введите номер операции ( 1 < номер < 6 ): 0
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ): -11313
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ): 0.21354
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ): 6.213412
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ): 99999999999999999999
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ): пыфлспав
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ): egefgsg
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ): -
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ): +
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ): =
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ): 4аывавл
!!! Введите корректные данные!

Введите номер операции ( 1 < номер < 6 ):
!!! Введите корректные данные!
```

Рисунок 13 - Проверка ввода некорректных значений.

Таблица 1 - описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Появление меню на экране и сообщения о выборе нужной операции	Верно
Выбор генерации графа	Запрос ввода количества вершин	Верно
Ручной ввод графа	Запрос ввода расстояния между вершинами	Верно
Сохранение результата	Результат записан в файл	Верно
Результат работы программы	Верный вывод кратчайших расстояний	Верно

В результате тестирования было выявлено, что программа работает успешно.

6 Ручной расчет задачи.

Проверим точность выполнения программы посредством ручного расчёта на примере неориентированного графа с 5 вершинами (рисунок 14). Начнём ручной расчёт с вершины номер 3.

Метка самой вершины 3 обозначается равной 0, метки остальных вершин – бесконечность (10000). Бесконечность обозначает, что путь до всех остальных вершин пока неизвестен. На данном этапе все вершины помечаются как не посещённые.

Начинаем обход из вершины 3 в последующие. Если путь из 3 в другие вершины присутствует, то идем дальше по графу. В данном случае перемещаемся в вершину 4, так как длина пути до нее минимальна. Длина пути в нее через вершину 3 равна $0 + 5 = 5$. 5 меньше текущей метки вершины 2 (10000), поэтому новая метка 4-ой вершины равна 5.

Аналогично находим пути для других соседей, в данном случае вершина 1, 2 и 5. После того, как все связанные с 3 вершины проверены, минимальное расстояние до 3 вершины считается окончательным, а вершина 3 помечается «посещенной».

Далее идём в вершину 4. Расстояние до неё (как уже известно) равно 5. Соседей кроме 3 вершины у неё нет, поэтому расстояние до неё является кратчайшим и вершина считается посещённой.

Далее идем в вершину 1. Она связана с вершинами 2, 3. Проверяем соседей (вершины 2 и 3). Но 3 вершина уже посещённая, поэтому осталось определить расстояние до вершины 2. Путь до неё равен $6 + 1 = 7$. Это меньше, чем было до этого (11), значит обновляем минимальное расстояние до этой вершины. Отмечаем вершину 1 посещённой.

Из 2 вершины есть путь в 1, 3, и 5. Непосещённая из них только 5. Расстояние до неё равно минимальному расстоянию до вершины 2

+ расстояние от неё до 5-ой вершины. Это равно $7 + 13 = 20$. Это больше, чем было до этого(13), значит оставляем прежний маршрут.

Перемещаемся в вершину 5. Её соседи 2 и 3. Вершина номер 3 – начальная, до неё путь = 0. Проверим путь до вершины 2. Это равно $13 + 13 = 26$. Это больше чем 7, значит не обновляем значение.

Таким образом получилось, что кратчайшие расстояния до вершин при ручном расчёте равны(Рисунок 14):

До вершины 1 = 6

До вершины 2 = 7

До вершины 3 = 0 (стартовая)

До вершины 4 = 5

До вершины 5 = 13

Таким образом, можно сделать вывод, что программа работает корректно.

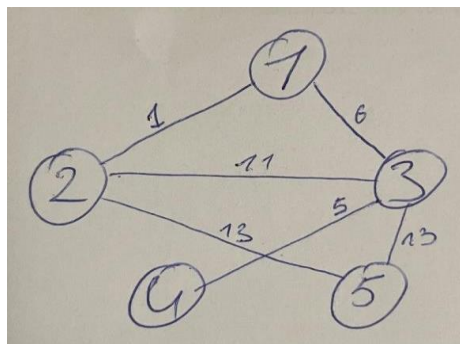


Рисунок 14 - Визуальное отображение графа.

```
Введите размер графа: 5
  1  2  3  4  5
1  0  1  6  0  0
2  1  0 11  0 13
3  6 11  0  5 13
4  0  0  5  0  0
5  0 13 13  0  0

Введите номер вершины: 3

Кратчайшие расстояния от вершины 3 до остальных вершин
  6    7    0    5   13
Для продолжения нажмите любую кнопку...
```

Рисунок 15 - Тестирование программы.

Заключение.

В процессе создания данного проекта разработана программа, реализующая поиск кратчайшего пути с помощью алгоритма Дейкстры в MicrosoftVisualStudio 2022.

При выполнении данной курсовой работы были получены навыки разработки программ и работы с матрицами смежности. Приобретены навыки по осуществлению алгоритма Дейкстры.

Недостатком программы является отсутствие графического сопровождения алгоритма, позволяющего наглядно представить результаты работы.

Список литературы.

1. Д.В. Парфенов. Язык Си: кратко и ясно. Учебное пособие.
2. А.А. Тюгашев. Языки программирования. Учебное пособие. 2018 г.
3. Уилсон Р. Введение в теорию графов. Пер. с англ. 1977. 208 с.
4. Харви Дейтел, Пол Дейтел. Как программировать на С/С++. 2009 г.
5. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.
6. Левитин А. В. Глава 9. Жадные методы: Алгоритм Дейкстры
7. Зыков А. А. Основы теории графов.

Приложение А. Листинг программы.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <time.h>
#include <locale.h>
#include <string.h>
#include <string>

FILE* f;
void menu() {

    printf("===== \n");
    printf("1) Случайный неориентированный граф\n");
    printf("2) Случайный ориентированный граф\n");
    printf("3) Задать вручную неориентированный граф\n");
    printf("4) Задать вручную ориентированный граф\n");
    printf("5) Вывести титульный лист заново\n");
    printf("6) Выход из программы\n");
    printf("===== \n");
}

void RandNeOrientGraf(int** matrix, int n) {
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                matrix[i][j] = 0;
            }
            if (i < j) {
                if (rand() % 100 > 50) {
                    matrix[i][j] = rand() % 15;
                }
                else {
                    matrix[i][j] = 0;
                }
                matrix[j][i] = matrix[i][j];
            }
        }
    }
}

void titulniyList() {
    printf("\n");
    printf(" Министерство наука и высшего образования Российской Федерации\n");
    printf(" Пензенский государственный университет\n");
    printf(" Кафедра «Вычислительная кафедра»\n");

    printf("\n\n");
    printf(" КУРСОВОЙ ПРОЕКТ\n");
    printf(" по дисциплине:\n ЛОГИКА И ОСНОВЫ АЛГОРИТМИЗАЦИИ В ИНЖЕНЕРНЫХ\n ЗАДАЧАХ\n");
    printf(" тема проекта:\n РЕАЛИЗАЦИЯ АЛГОРИТМА ДЕЙКСТРЫ\n ");

    printf("\n\n\n");
    printf(" Выполнил:\n");
    printf(" студент группы 23BVB2\n");
    printf(" Федоров Б.М.\n");
}
```



```

printf("\n\n");
printf(" Приняла:\n");
printf(" Юрова О.В.\n");
printf("\n\n\n");
printf(" Пенза 2024\n");

time_t startTime = time(nullptr);

while (difftime(time(nullptr), startTime) < 5) {}
system("cls");
}

void RandOrientGraf(int** matrix, int n) {
    srand(time(NULL));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            if (i == j) {
                matrix[i][j] = 0;
            }
            if (i < j) {
                if (rand() % 100 > 50) {
                    matrix[i][j] = rand() % 10;
                    if (rand() % 100 > 50)
                        matrix[j][i] = 0;
                    else
                        matrix[j][i] = matrix[i][j];
                }
            }
            else
                if (rand() % 100 > 50) {
                    matrix[j][i] = rand() % 10;
                    matrix[i][j] = 0;
                }
                else {
                    matrix[i][j] = 0;
                    matrix[j][i] = 0;
                }
        }
    }
}

void NeOrGraf(int** matrix, int n, int temp) {
    using namespace std;
    for (int i = 0; i < n; i++)
    {
        for (int j = i; j < n; j++) {

            bool flag = 0;

            do
            {
                string value;
                value.resize(5);
                printf("Введите расстояние %d - %d (от 0 до 9999): ", i + 1, j + 1);

                getline(cin, value);

                flag = 0;

                for (int i = 0; i < value.size(); i++)

```

```

        {
            if (!isdigit(value[i]) || isspace(value[i]))
                flag = 1;
        }

        if (!flag)
        {
            if (std::atoi(value.c_str()) > 9999)
                flag = 1;
            if (value.size() == 0)
                flag = 1;
        }

        if (flag == 1)
        {
            std::cout << "!!! Введите корректные данные!" << std::endl << endl;
        }
        else
        {
            temp = std::atoi(value.c_str());
            break;
        }
    } while (flag);

    matrix[i][j] = temp;
    matrix[j][i] = temp;
}
}
}

```

```

void OrGraf(int** matrix, int n, int temp) {
    using namespace std;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {

            bool flag = 0;

            do
            {
                string value;
                value.resize(5);
                printf("Введите расстояние %d - %d (от 0 до 9999): ", i + 1, j + 1);

                getline(cin, value);

                flag = 0;

                for (int i = 0; i < value.size(); i++)
                {
                    if (!isdigit(value[i]) || isspace(value[i]))
                        flag = 1;
                }

                if (!flag)
                {
                    if (std::atoi(value.c_str()) > 9999)
                        flag = 1;
                    if (value.size() == 0)
                        flag = 1;
                }
            }
        }
    }
}

```

```

        if (flag == 1)
        {
            std::cout << "!!! Введите корректные данные!" << std::endl << endl;
        }
        else
        {
            temp = std::atoi(value.c_str());
            break;
        }
    } while (flag);

    matrix[i][j] = temp;

    }
}
}

void vivodGrafa(int** G, int n) {
    f = fopen("pez.txt", "w");
    printf(" ");
    fprintf(f, " ");
    for (int i = 0; i < n; i++) {
        printf("%4d", i + 1);
        fprintf(f, "%5d", i + 1);
    }
    printf("\n\n");
    fprintf(f, "\n\n");
    for (int i = 0; i < n; i++) {
        printf("%3d", i + 1);
        fprintf(f, "%3d", i + 1);
        for (int j = 0; j < n; j++) {

            printf("%4d", G[i][j]);
            fprintf(f, "%5d", G[i][j]);
        }
        printf("\n");
        fprintf(f, "\n");
    }
    printf("\n");
}

void Vershini(int** G, int n, int* minimalrast, int* pos) {
    for (int i = 0; i < n; i++) {
        minimalrast[i] = 10000;
        pos[i] = -1;    //вершины помечаются непосещенными
    }
}

void Algoritm(int** G, int start_index, int n, int min, int s, int* pos, int* minimalrast, int temp) {
    minimalrast[start_index] = 0;
    do {
        min = 10000; // значение минимального веса расстояния
        s = 10000; // значение для хранения индекса вершины с минимальным весом расстояния
        for (int i = 0; i < n; i++) {
            if ((pos[i] == -1) && (minimalrast[i] < min)) { // Если вершину ещё не обошли и вес меньше min
                min = minimalrast[i]; // Обновляем min на минимальный вес
                s = i; // Запоминаем индекс вершины с минимальным расстоянием
            }
        }
    }
}

```

```

        if (s != 10000) { // если нашли путь с минимальным расстоянием
            for (int i = 0; i < n; i++) { // обновляем вес для соседних вершин
                if (G[s][i] > 0) { // наличие ребра между вершиной s и вершиной i
                    temp = min + G[s][i]; // вычисление веса для вершины i
                    if (temp < minimalrast[i]) { // если новый вес меньше текущего веса i
                        minimalrast[i] = temp; // обновляем минимальный вес i
                    }
                }
            }
            pos[s] = 0; //вершина помечается посещенной
        }
    } while (s < 10000); //пока не будут обработаны все вершины
    printf("\nКратчайшие расстояния от вершины %d до остальных вершин\n", start_index + 1);
    fprintf(f, "\nКратчайшие расстояния от вершины %d до остальных вершин\n", start_index + 1);
    for (int i = 0; i < n; i++) {
        if (minimalrast[i] == 10000) { // Проверяем, является ли минимальный вес для вершины i
            //недостижимым
            printf("(недостижима) "); // Если недостижима, выводим сообщение в консоль
        }
        else {
            printf("%7d ", minimalrast[i]); // Если достижима, выводим минимальный вес
        }
        if (minimalrast[i] == 10000) {
            fprintf(f, "(недостижима) "); // Если недостижима, записываем сообщение в файл
        }
        else {
            fprintf(f, "%7d ", minimalrast[i]); // Если достижима, записываем минимальный вес в файл
        }
    };
    printf("\n\nПроизведена запись в файл(pez.txt)\n", start_index + 1);
}

int main() {
    int num = 0, min = NULL, s = NULL;
    int** G; // указатель на указатель на строку элементов
    int* minimalrast, * pos, start_index;
    int n = 0, temp = NULL, start = NULL, save;
    int value = 0;

    int count;
    setlocale(LC_ALL, "rus");

    using namespace std;

    titulniyList();

    while (true) {
        system("cls");
        menu();

        bool flag = 0;

        do
        {
            string value;
            value.resize(5);
            cout << "Введите номер операции ( 1 < номер < 6 ): ";

```

```

getline(cin, value);

flag = 0;

if (value[0] == '0')
    flag = 1;

for (int i = 0; i < value.size(); i++)
{
    if (!isdigit(value[i]) || isspace(value[i]))
        flag = 1;
}

if (!flag)
{
    if (value.size() == 0 && std::atoi(value.c_str()) == 0)
        flag = 1;
    if (std::atoi(value.c_str()) > 6)
        flag = 1;
}

if (flag == 1)
{
    std::cout << "!!! Введите корректные данные!" << std::endl << endl;
}
else
{
    num = std::atoi(value.c_str());
    break;
}
} while (flag);

if (num == 1) {
    system("cls");

    bool flag = 0;

    do
    {
        string value;
        value.resize(5);
        cout << "Введите размер графа (от 2 до 500): ";

        getline(cin, value);

        flag = 0;

        if (value[0] == '0')
            flag = 1;

        for (int i = 0; i < value.size(); i++)
        {
            if (!isdigit(value[i]) || isspace(value[i]))
                flag = 1;
        }

        if (!flag)
        {

```



```

        if (value.size() == 1 && std::atoi(value.c_str()) == 1)
            flag = 1;
        if (std::atoi(value.c_str()) > 500)
            flag = 1;
        if (value.size() == 0)
            flag = 1;
    }

    if (flag == 1)
    {
        std::cout << "!!! Введите корректные данные!" << std::endl << endl;
    }
    else
    {
        n = std::atoi(value.c_str());
        break;
    }
} while (flag);

```

```

G = (int**)malloc(n * sizeof(int*));
for (int i = 0; i < n; i++) {
    G[i] = (int*)malloc(n * sizeof(int));
}
minimalrast = (int*)malloc(n * sizeof(int));
pos = (int*)malloc(n * sizeof(int));

```

```

RandNeOrientGraf(G, n);
vivodGrafa(G, n);

```

```

flag = 0;

```

```

do
{
    string value;
    value.resize(5);
    cout << "Введите номер вершины: ";

    getline(cin, value);

    flag = 0;

    if (value[0] == '0')
        flag = 1;

    for (int i = 0; i < value.size(); i++)
    {
        if (!isdigit(value[i]) || isspace(value[i]))
            flag = 1;
    }

    if (!flag)
    {
        if (value.size() == 0 && std::atoi(value.c_str()) == 0)
            flag = 1;
        if (std::atoi(value.c_str()) > n)
            flag = 1;
    }

    if (flag == 1)
    {

```

```

        std::cout << "!!! Вершина отсутствует. Введите другую!" << std::endl << endl;

    }
    else
    {
        start_index = std::atoi(value.c_str());
        break;
    }
} while (flag);

start_index--;
Vershini(G, n, minimalrast, pos);
Algoritm(G, start_index, n, min, s, pos, minimalrast, temp);

printf("\nДля продолжения нажмите любую кнопку...");
for (int i = 0; i < n; i++) {
    free(G[i]);
}
free(G);
free(minimalrast);
free(pos);
getchar();
getchar();
}

if (num == 2) {
    system("cls");

    bool flag = 0;

    do
    {
        string value;
        value.resize(5);
        cout << "Введите размер графа (от 2 до 500): ";

        getline(cin, value);

        flag = 0;

        if (value[0] == '0')
            flag = 1;

        for (int i = 0; i < value.size(); i++)
        {
            if (!isdigit(value[i]) || isspace(value[i]))
                flag = 1;
        }

        if (!flag)
        {
            if (value.size() == 1 && std::atoi(value.c_str()) == 1)
                flag = 1;
            if (std::atoi(value.c_str()) > 500)
                flag = 1;
            if (value.size() == 0)
                flag = 1;
        }

        if (flag == 1)
        {

```

```

        std::cout << "!!! Введите корректные данные!" << std::endl << endl;
    }
    else
    {
        n = std::atoi(value.c_str());
        break;
    }
} while (flag);

G = (int**)malloc(n * sizeof(int*));
for (int i = 0; i < n; i++) {
    G[i] = (int*)malloc(n * sizeof(int));
}
minimalrast = (int*)malloc(n * sizeof(int));
pos = (int*)malloc(n * sizeof(int));

RandOrientGraf(G, n);
vivodGrafa(G, n);

flag = 0;

do
{
    string value;
    value.resize(5);
    cout << "Введите номер вершины: ";

    getline(cin, value);

    flag = 0;

    if (value[0] == '0')
        flag = 1;

    for (int i = 0; i < value.size(); i++)
    {
        if (!isdigit(value[i]) || isspace(value[i]))
            flag = 1;
    }

    if (!flag)
    {
        if (value.size() == 0 && std::atoi(value.c_str()) == 0)
            flag = 1;
        if (std::atoi(value.c_str()) > n)
            flag = 1;
    }

    if (flag == 1)
    {
        std::cout << "!!! Вершина отсутствует. Введите другую!" << std::endl << endl;
    }
    else
    {
        {
            start_index = std::atoi(value.c_str());
            break;
        }
    }
} while (flag);

```

```

start_index--;
Vershini(G, n, minimalrast, pos);
Algoritm(G, start_index, n, min, s, pos, minimalrast, temp);

printf("\nДля продолжения нажмите любую кнопку...");
for (int i = 0; i < n; i++) {
    free(G[i]);
}
free(G);
free(minimalrast);
free(pos);
getchar();
getchar();
}

if (num == 3) {
    system("cls");
    bool flag = 0;

    do
    {
        string value;
        value.resize(5);
        cout << "Введите размер графа (от 2 до 500): ";

        getline(cin, value);

        flag = 0;

        if (value[0] == '0')
            flag = 1;

        for (int i = 0; i < value.size(); i++)
        {
            if (!isdigit(value[i]) || isspace(value[i]))
                flag = 1;
        }

        if (!flag)
        {
            if (value.size() == 1 && std::atoi(value.c_str()) == 1)
                flag = 1;
            if (std::atoi(value.c_str()) > 500)
                flag = 1;
            if (value.size() == 0)
                flag = 1;
        }

        if (flag == 1)
        {
            std::cout << "!!! Введите корректные данные!" << std::endl << endl;
        }
        else
        {
            n = std::atoi(value.c_str());
            break;
        }
    } while (flag);

    G = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {

```

```

    G[i] = (int*)malloc(n * sizeof(int));
}
minimalrast = (int*)malloc(n * sizeof(int));
pos = (int*)malloc(n * sizeof(int));

NeOrGraf(G, n, temp);
vivodGrafa(G, n);

flag = 0;

do
{
    string value;
    value.resize(5);
    cout << "Введите номер вершины: ";

    getline(cin, value);

    flag = 0;

    if (value[0] == '0')
        flag = 1;

    for (int i = 0; i < value.size(); i++)
    {
        if (!isdigit(value[i]) || isspace(value[i]))
            flag = 1;
    }

    if (!flag)
    {
        if (value.size() == 0 && std::atoi(value.c_str()) == 0)
            flag = 1;
        if (std::atoi(value.c_str()) > n)
            flag = 1;
    }

    if (flag == 1)
    {
        std::cout << "!!! Вершина отсутствует. Введите другую!" << std::endl << endl;
    }
    else
    {
        start_index = std::atoi(value.c_str());
        break;
    }
} while (flag);

start_index--;
Vershini(G, n, minimalrast, pos);
Algoritm(G, start_index, n, min, s, pos, minimalrast, temp);

printf("\nДля продолжения нажмите любую кнопку...");
for (int i = 0; i < n; i++) {
    free(G[i]);
}
free(G);
free(minimalrast);
free(pos);
getchar();

```

```

    getchar();
}

if (num == 4) {
    system("cls");
    bool flag = 0;

    do
    {
        string value;
        value.resize(5);
        cout << "Введите размер графа (от 2 до 500): ";

        getline(cin, value);

        flag = 0;

        if (value[0] == '0')
            flag = 1;

        for (int i = 0; i < value.size(); i++)
        {
            if (!isdigit(value[i]) || isspace(value[i]))
                flag = 1;
        }

        if (!flag)
        {
            if (value.size() == 1 && std::atoi(value.c_str()) == 1)
                flag = 1;
            if (std::atoi(value.c_str()) > 500)
                flag = 1;
            if (value.size() == 0)
                flag = 1;
        }

        if (flag == 1)
        {
            std::cout << "!!! Введите корректные данные!" << std::endl << endl;
        }
        else
        {
            n = std::atoi(value.c_str());
            break;
        }
    } while (flag);

    G = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        G[i] = (int*)malloc(n * sizeof(int));
    }
    minimalrast = (int*)malloc(n * sizeof(int));
    pos = (int*)malloc(n * sizeof(int));

    OrGraf(G, n, temp);
    vivodGrafa(G, n);

    flag = 0;

    do
    {

```

```

string value;
value.resize(5);
cout << "Введите номер вершины: ";

getline(cin, value);

flag = 0;

if (value[0] == '0')
    flag = 1;

for (int i = 0; i < value.size(); i++)
{
    if (!isdigit(value[i]) || isspace(value[i]))
        flag = 1;
}

if (!flag)
{
    if (value.size() == 0 && std::atoi(value.c_str()) == 0)
        flag = 1;
    if (std::atoi(value.c_str()) > n)
        flag = 1;
}

if (flag == 1)
{
    std::cout << "!!! Вершина отсутствует. Введите другую!" << std::endl << endl;
}
else
{
    start_index = std::atoi(value.c_str());
    break;
}
} while (flag);

start_index--;
Vershini(G, n, minimalrast, pos);
Algoritm(G, start_index, n, min, s, pos, minimalrast, temp);

printf("\nДля продолжения нажмите любую кнопку...");
for (int i = 0; i < n; i++) {
    free(G[i]);
}
free(G);
free(minimalrast);
free(pos);
getchar();
getchar();
}

if (num == 5) {
    system("cls");
    titulniyList();
}

if (num == 6) {
    break;
}
}
}

```


Приложение В. Результаты работы программы.

```
Введите размер графа (от 2 до 500): 10
  1  2  3  4  5  6  7  8  9 10
1  0 12  0  0  0  0  0  0  0 11
2 12  0  0  0  0 10 10 10 11 13
3  0  0  0  3  0  2  6  0  0  0
4  0  0  3  0  2  3  1  0  1  0
5  0  0  0  2  0  0  5  0  0  0
6  0 10  2  3  0  0 11  0  0  0
7  0 10  6  1  5 11  0  0  1  0
8  0 10  0  0  0  0  0  0  3  0
9  0 11  0  1  0  0  1  3  0 14
10 11 13  0  0  0  0  0  0 14  0

Введите номер вершины: 5
Кратчайшие расстояния от вершины 5 до остальных вершин
  25   13    5    2    0    5    3    6    3   17
Для продолжения нажмите любую кнопку...
```

Рисунок 16 - Результат случайно сгенерированного неориентированного графа

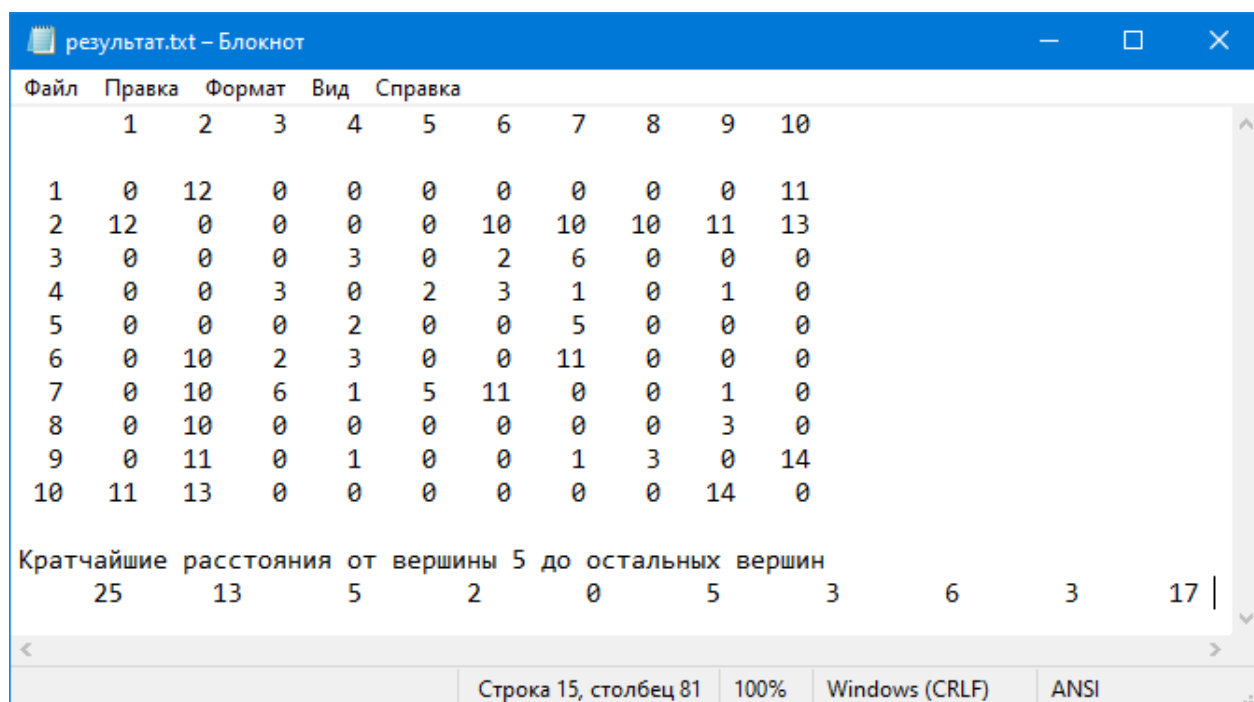


Рисунок 17 - Результат сохранения работы программы в файл