Пензенский государственный университет

Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовой работе

По курсу «Логика и основы алгоритмизации в инженерных задачах»

На тему «Реализация алгоритма Дейкстры»

Выполнил студенты группы 22ВВВ3 (22ВВП2):

Тельнов И.В.

19.12.23
отлично
Орл

Приняли:

Юрова О.В.

Акифьев И.В.

Пенза 2023

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 Тельнова Нина Владимировна Группа 22ВВВ3122ВВВП2)

Тема_проекта Реализация алгоритма Дейкстры

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данным заданием курсового проекта.
Пояснительная записка должна содержать:
1 Постановку задачи;
2 Теоретическую часть задания;
3 Описание алгоритма поставленной задачи;
4 Пример ручного расчета задачи и вычислений (на небольшом участке работы алгоритма);
5 Описание общей программы;
6 Тесты;
7 Список литературы;
8 Листинг программы;
9 Результаты работы программы;

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схем

3. Экспериментальная часть

Тестирование программы.
Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

1 Исследование теоретической части курсовой
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

Дата защиты проекта "____" _____

Руководитель Юрова О.В

Задание получил "6" сентября 2023 г.

Студент Тельнов Илья Владимирович

Содержание

Реферат

Отчёт 33 стр., 18 рисунков, 1 таблица, 1 приложение.
ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ДЕЙКСТРЫ.

Цель исследования – разработка программы, способной эффективно находить кратчайшие пути между вершинами взвешенного связного ориентированного графа с использованием алгоритма Дейкстры.

В работе рассмотрены основные шаги алгоритма Дейкстры для нахождения минимального пути в взвешенном графе.

Введение

Алгоритм Дейкстры, также известный как алгоритм нахождения кратчайшего пути, применяется для нахождения кратчайшего пути от одной из вершин графа до всех остальных. Алгоритм Дейкстры работает на ориентированных (с некоторыми дополнениями и на неориентированных) графах, и призван искать кратчайшие пути между заданной вершиной и всеми остальными вершинами в графе. Основная цель алгоритма состоит в том, чтобы найти кратчайший путь, при этом минимизируя общий вес рёбер.

В начале алгоритма расстояние для начальной вершины полагается равным нулю, а все остальные расстояния заполняются большим положительным числом (бóльшим максимального возможного пути в графе). Массив флагов заполняется нулями. Затем запускается основной цикл.

На каждом шаге цикла мы ищем вершину с минимальным расстоянием и флагом равным нулю. Затем мы устанавливаем в ней флаг в 1 и проверяем все соседние с ней вершины . Если в них расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, 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 нахождение минимального пути.

1. Постановка задачи

Требуется разработать программу, которая найдет кратчайшие пути от одной из вершин графа до всех остальных, используя алгоритм Дейкстры.

Исходный граф в программе должен задаваться матрицей весов рёбер, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы взвешенного графа. После обработки этих данных на экран должно выводиться меню с выбором дальнейший действий, таких как просмотр связей между вершинами или выполнение алгоритма Дейстры. Необходимо предусмотреть различные исходы, такие как случаи отсутствия связности в графе или некорректного ввода данных, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

.

2. Теоретическая часть задания

Граф G, изображенный на рисунке 1, состоит из множества вершин X1,X2,…,Xn и множества ребер, соединяющих определенные вершины. Ребра в этом графе имеют веса, показывающие стоимость прохождения между вершинами. Граф с такими ребрами называется взвешенным.

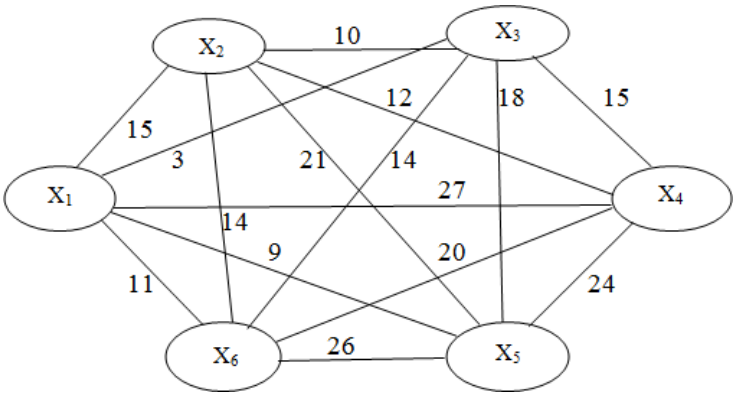

Рисунок 1 – Пример взвешенного графа

Представление графа в виде матрицы смежности позволяет хранить информацию о ребрах графа в квадратной матрице, где присутствие пути из одной вершины в другую обозначается числом, представляющим вес ребра, а отсутствие пути – специальным обозначением, например, нулём.

Существует множество алгоритмов для работы с взвешенными графами, и одним из них является алгоритм Дейкстры. Этот алгоритм направлен на нахождение минимального пути между вершинами, работу с ориентированными графами. Он начинается из определенной вершины и последующей проверкой новых ребер и вершин в графе, выбирает ребро минимального веса, связывающее нас со следующими ребрами и вершинами. Процесс продолжается до тех пор, пока не будет найден минимальный вес пути.

3. Описание алгоритма программы

Цель алгоритма Дейкстры в графе заключается в нахождении короткого пути путем пошагового добавления ребер с наименьшим весом. Начиная с начальной вершины, мы постепенно проходим граф, выбирая ребро минимального веса, которое связывает уже выбранные вершины с вершиной из оставшихся. Этот процесс продолжается, пока не будет найден самый короткий путь. Ниже представлен псевдокод:

```
функция Алгоритм Дейкстры(граф[MAX_VERTICES][ MAX_VERTICES], вершины,
начало, конец, длина кратчайшего пути)

    посещено [MAX_VERTICES]
    расстояние [MAX_VERTICES]

    для i = 0 пока i < вершины
        расстояние[i] = INT_MAX
        посещено[i] = 0

    расстояние[старт] = 0
    родитель[старт] = -1

    для i = 0 пока i < вершины – 1
        минРасстояние = INT_MAX
        минИндекс = -1


        для j = 0 пока j < вершины
            если не посещено[j] и расстояние[j] <  минРасстояние
                минРасстояние = расстояние[j]
                минИндекс = j


        посещено[минИндекс] = 1

        для j = 0 пока j < вершины
            если не посещено[j] и граф[ минИндекс][j] и
        расстояние[минИндекс] != INT_MAX и расстояние[минИндекс] +
        граф[минИндекс][j] < расстояние[j]

                расстояние[j] = расстояние[минИндекс] +
граф[минИндекс][j]

                родитель[j] = минИндекс
    *длина кратчайшего пути = расстояние[конец]
```

4. 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 Win64 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: dijkstra, fillRandomGraph, fillUndirectedGraphRandom, loadGraphFromFile, saveGraphToFile, printShortestPath, printGraph, getInputInteger, getInputString, main.

Работа программы начинается с вывода общей информации о курсовой работе (рис. 2).

```
printf("Пензенский государственный университет\n");
printf("Кафедра «Вычислительная техника»\n");
printf("Приняли: Юрова О.В. и Акифьев И.В.\n");
printf("Курсовая работа\n");
printf("По курсу \"Логика и основы алгоритмизации в инженерных задачах\"\n");
printf("На тему \"Реализация алгоритма Дейкстры\"\n");
printf("Выполнил: Тельнов Илья Владимирович, группа: 22ВВВ3 (22ВВП2)\n");
```

Далее выводятся основные действия программы, которые пользователь может выбрать (рис. 3).

```
while (1) {
    printf("\nМеню:\n");
    printf("1. Заполнение графа\n");
    printf("2. Загрузка графа из файла\n");
    printf("3. Сохранение графа в файл\n");
    printf("4. Выполнение алгоритма Дейкстры\n");
    printf("5. Просмотр связей вершин в графе\n");
    printf("0. Выход\n");
    printf("\nВаш выбор: ");
```

Если пользователь выбрал первый пункт (рис. 4 и рис. 5), то программа просит выбрать ориентированный или неориентированный граф и просит ввести количество вершин. Затем она заполняет этот граф случайными числами (вызывая функцию fillRandomGraph или fillUndirectedGraphRandom) и выводит его.

```
case 1: {
    printf("Выберите тип графа:\n");
    printf("1. Ориентированный\n");
    printf("2. Неориентированный\n");
    printf("\nВаш выбор: ");
    int graphTypeChoice;
    if (scanf("%d", &graphTypeChoice) != 1) {
        printf("Ошибка ввода. Пожалуйста, введите целое число.\n");
        fflush(stdin);
        break;
    }
    vertices = getInputInteger("Введите количество вершин (max 100): ");
    if (vertices <= 0 || vertices > MAX_VERTICES) {
        printf("Ошибка: количество вершин должно быть в пределах от 1 до %d\n",
MAX_VERTICES);
        break;
    }
    if (graphTypeChoice == 1 || graphTypeChoice == 2) {
        if (graphTypeChoice == 1) {
            fillRandomGraph(graph, vertices, 1);
        }
        else {
            fillUndirectedGraphRandom(graph, vertices);
        }
        graphFilled = 1;
        printGraph(graph, vertices, 1);
    }
    else {
        printf("Ошибка: неверный выбор типа графа.\n");
    }
    break;
}
```

Если пользователь выбрал второй пункт (рис. 6), то сначала программа просит ввести название файла. Затем она выгружает этот файл с графом (вызывая функцию loadGraphFromFile) и выводит меню на экран, позволяя выбрать необходимые дальнейшие действия.

```
case 2:
    if (!graphFilled) {
        // Оставьте запрос на имя файла
        char filename_load[MAX_FILENAME_LENGTH];
        printf("Введите имя файла для загрузки: ");
        scanf("%s", filename_load);
        if (loadGraphFromFile(graph, &vertices, filename_load)) {
            graphFilled = 1;
        }
        else {
            printf("Ошибка при загрузке графа из файла.\n");
        }
    }
    else {
        printf("Граф уже заполнен. Выберите другую опцию.\n");
    }
    break;
```

Если пользователь выбрал третий пункт (рис. 7), то программа попросит ввести название файла, куда произойдет сохранение графа (только если вы уже заполнили его). Затем она загружает этот файл (вызывая функцию saveGraphToFile) и выводит меню на экран, позволяя выбрать необходимые дальнейшие действия.

```
case 3:
    if (graphFilled) {
        saveGraphToFile(graph, vertices, getInputString("Введите имя файла для сохранения: ",
filename, MAX_FILENAME_LENGTH));
    }
    else {
        printf("Граф не был заполнен. Выберите способ заполнения графа перед сохранением
в файл.\n");
    }
    break;
```

Если пользователь выбрал четвертый пункт (рис. 8), то программа возьмет граф (только если вы уже заполнили его) и начнет выполнять алгоритм Дейкстры (вызвав функцию dijkstra). На экране появится возможность ввода начальной вершины для нашего алгоритма и рекомендуемый самый короткий путь от начальной до конечной вершины(вызов функции printShortestPath)

```
case 4:
    if (!graphFilled) {
        printf("Граф не был заполнен. Выберите способ заполнения графа перед выполнением алгоритма Дейкстры.\n");
    }
    else {
        int startVertex = getInputInteger("Введите начальную вершину для алгоритма Дейкстры: ");
        dijkstra(graph, vertices, startVertex, vertices - 1, &shortestPathLength);
        printf("Рекомендуемый самый короткий путь от вершины %d до вершины %d: ", startVertex, vertices - 1);
        printShortestPath(graph, startVertex, vertices - 1, shortestPathLength);
        printf("\n");
    }
break;
```

Если пользователь выбрал пятый пункт (рис. 9), то программа возьмет граф (вызвав функцию printGraph, только если вы уже заполнили его) и выведет связи вершин в нем.

```
case 5:
   if (graphFilled) {
      //printf("Связи вершин в графе:\n");
      printGraph(graph, vertices, 0);
   }
   else {
      printf("Граф не был заполнен. Выберите способ заполнения графа перед просмотром связей.\n");
   }
   break;
```

Если пользователь выбрал пункт «0», то программа завершит свою работу (рис. 10).

Ниже представлена работа программы:

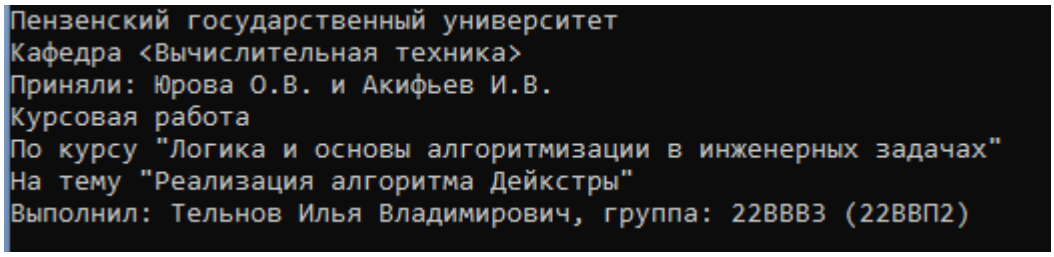

Рисунок 2 – Общая информация

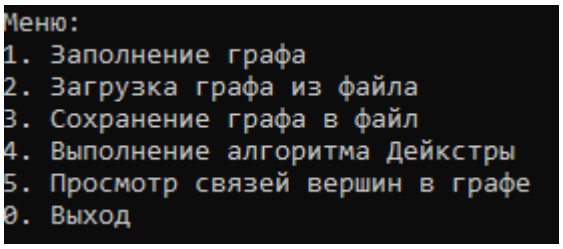

Рисунок 3 – Основные действия

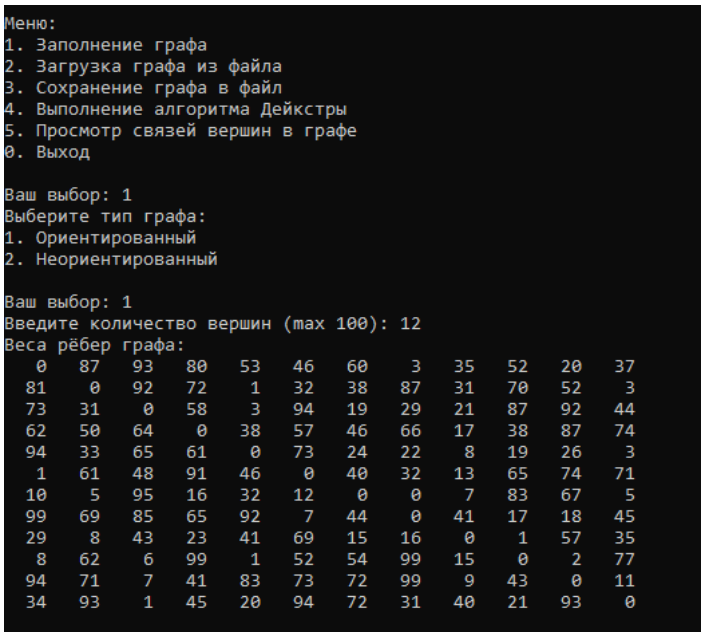

Рисунок 4 – Заполнение ориентированного графа

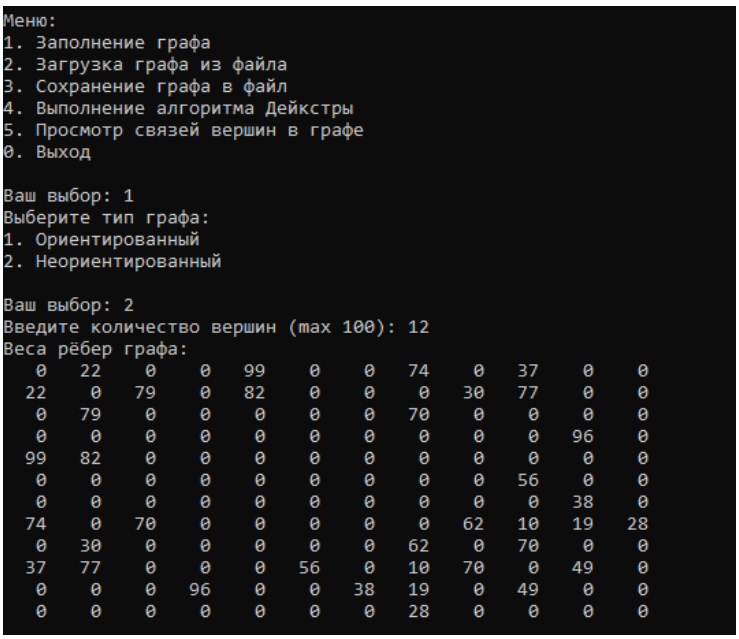

Рисунок 5 – Заполнение неориентированного графа

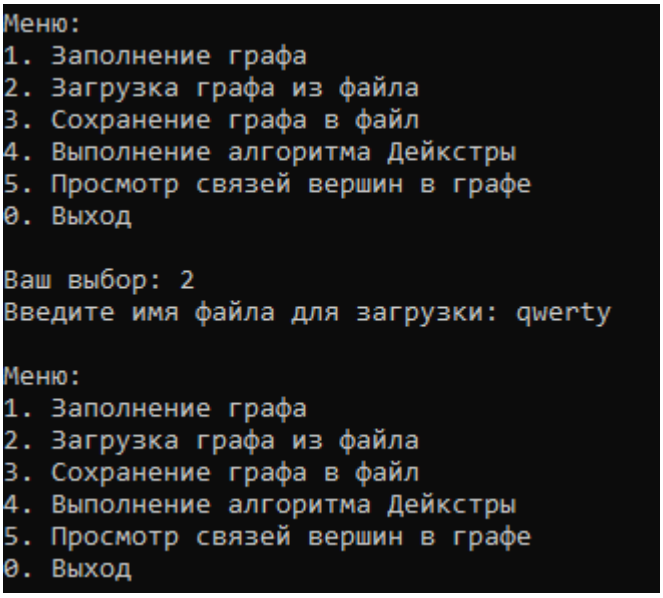

Рисунок 6 – Загрузка графа из файла

```
Меню:
1. Заполнение графа
2. Загрузка графа из файла
3. Сохранение графа в файл
4. Выполнение алгоритма Дейкстры
5. Просмотр связей вершин в графе
0. Выход

Ваш выбор: 1
Выберите тип графа:
1. Ориентированный
2. Неориентированный

Ваш выбор: 1
Введите количество вершин (max 100): 12
Веса рёбер графа:
   0   65   45    7   84   82   95   46   23    3    7    2
  19    0   14   35   77   63   40   10   76   80   66   73
   6   76    0    2   45   36   71   85   40   24   96   49
  52   93   92    0   57   14   29   50   42   71   78   67
  43   77   84   72    0   31   37   67   85   98   43   16
  99   37   40   22   12    0   29   32   67   63   77   97
   4   61   37   65   68   34    0    6   78   73   19   42
  62   76   87   31   39   35   29    0   24   29   79   81
  69   11   28   20   84   81   41   11    0   34   77    1
  60   58   70   55   24   36   37   29   55    0   59   72
  59   26   38   80   81   47   94   67   99   73    0   35
  67   50   85   64   78   63   14   74    0   35   89    0

Меню:
1. Заполнение графа
2. Загрузка графа из файла
3. Сохранение графа в файл
4. Выполнение алгоритма Дейкстры
5. Просмотр связей вершин в графе
0. Выход

Ваш выбор: 3
Введите имя файла для сохранения: qwerty
```

Рисунок 7 – Сохранение графа в файл

```
Меню:
1. Заполнение графа
2. Загрузка графа из файла
3. Сохранение графа в файл
4. Выполнение алгоритма Дейкстры
5. Просмотр связей вершин в графе
0. Выход

Ваш выбор: 1
Выберите тип графа:
1. Ориентированный
2. Неориентированный

Ваш выбор: 1
Введите количество вершин (max 100): 12
Веса рёбер графа:
   0   72   69   99    0   60   81   68   25   48  100   30
  14    0    1    0   36    0   18   43   84   67   31   86
   1   98    0   62   54   89   71    1   85   53   89   49
  45   91   94    0   59   61   16   41   88   99   71    6
  73   66   94   19    0   85   77   65   46   61   17   18
  94   97   95   54   61    0   18   65    1   26   59   31
  72   93   11   20   21   25    0   41   86    9   90   97
  18   89   46   18    1   88   61    0   13   56   46   25
  64   97   42   92   41   54   26   46    0   25   31   93
  82   69   26    4   70   46   20   51   16    0   38   69
  10   70   54   76   58   76    8   68   87   53    0   12
  31   70   69   91   27   84   48   51   72   93   23    0

Меню:
1. Заполнение графа
2. Загрузка графа из файла
3. Сохранение графа в файл
4. Выполнение алгоритма Дейкстры
5. Просмотр связей вершин в графе
0. Выход

Ваш выбор: 4
Введите начальную вершину для алгоритма Дейкстры: 0
Рекомендуемый самый короткий путь от вершины 0 до вершины 11: 0 -> 11 -> 0 (суммарный вес: 30)
```

Рисунок 8 – Выполнение алгоритма Дейкстры

```
Выберите тип графа:
1. Ориентированный
2. Неориентированный

Ваш выбор: 1
Введите количество вершин (max 100): 12
Веса рёбер графа:
   0   72   69   99    0   60   81   68   25   48  100   30
  14    0    1    0   36    0   18   43   84   67   31   86
   1   98    0   62   54   89   71    1   85   53   89   49
  45   91   94    0   59   61   16   41   88   99   71    6
  73   66   94   19    0   85   77   65   46   61   17   18
  94   97   95   54   61    0   18   65    1   26   59   31
  72   93   11   20   21   25    0   41   86    9   90   97
  18   89   46   18    1   88   61    0   13   56   46   25
  64   97   42   92   41   54   26   46    0   25   31   93
  82   69   26    4   70   46   20   51   16    0   38   69
  10   70   54   76   58   76    8   68   87   53    0   12
  31   70   69   91   27   84   48   51   72   93   23    0

Меню:
1. Заполнение графа
2. Загрузка графа из файла
3. Сохранение графа в файл
4. Выполнение алгоритма Дейкстры
5. Просмотр связей вершин в графе
0. Выход

Ваш выбор: 4
Введите начальную вершину для алгоритма Дейкстры: 0
Рекомендуемый самый короткий путь от вершины 0 до вершины 11: 0 -> 11 -> 0 (суммарный вес: 30)

Меню:
1. Заполнение графа
2. Загрузка графа из файла
3. Сохранение графа в файл
4. Выполнение алгоритма Дейкстры
5. Просмотр связей вершин в графе
0. Выход

Ваш выбор: 5
Связи вершин в графе:
Вершина 0 связана с: 1, 2, 3, 5, 6, 7, 8, 9, 10, 11
Вершина 1 связана с: 0, 2, 4, 6, 7, 8, 9, 10, 11
Вершина 2 связана с: 0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11
Вершина 3 связана с: 0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11
Вершина 4 связана с: 0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11
Вершина 5 связана с: 0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11
Вершина 6 связана с: 0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11
Вершина 7 связана с: 0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11
Вершина 8 связана с: 0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11
Вершина 9 связана с: 0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11
Вершина 10 связана с: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11
Вершина 11 связана с: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Рисунок 9 – Просмотр связей вершин

5. Тестирование

Среда разработки Microsoft Visual Studio 2019 предоставляет все средства, необходимые при разработке и отладке многомодульной программы. Тестирование проводилось в процессе разработки, после завершения написания программы. В ходе тестирования проблем не было выявлено.

Таблица 1 – Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Проверка запуска программы	Вывод общей информации и основных действий	Верно
Проверка на неправильный выбор основного действия	Программа должна оповестить пользователя об ошибке, предложить снова варианты	Верно
Проверка на введение несуществующего файла	Программа должна оповестить пользователя об ошибке, предложить снова варианты	Верно
Проверка на введение неправильного размера графа	Программа должна оповестить пользователя об ошибке, предложить снова варианты	Верно

Проверка на выполнение действий без графа	Программа должна оповестить пользователя об ошибке, предложить снова варианты	Верно
Проверка на неправильный выбор типа графа	Программа должна оповестить пользователя об ошибке, предложить снова варианты	Верно

Ниже приведены результаты тестов:

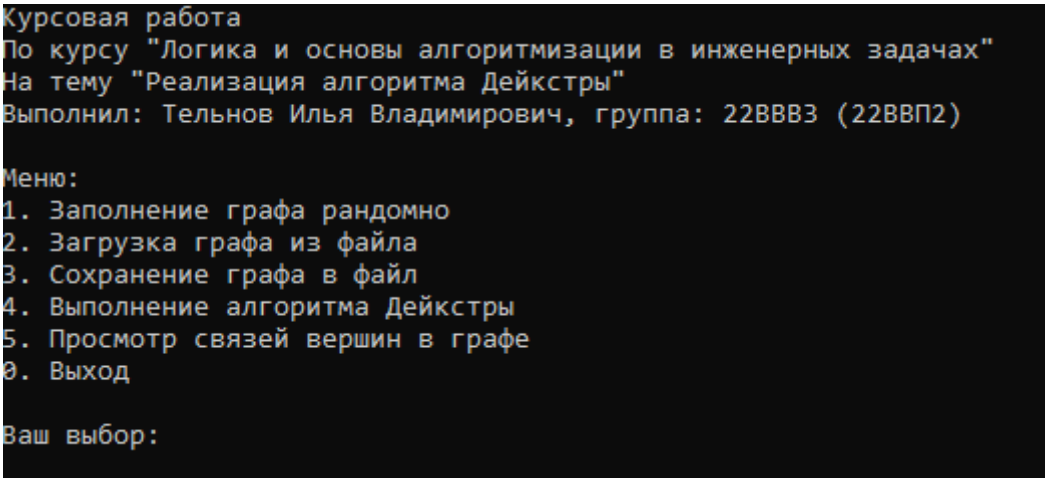

Рисунок 11 – Проверка запуска программы

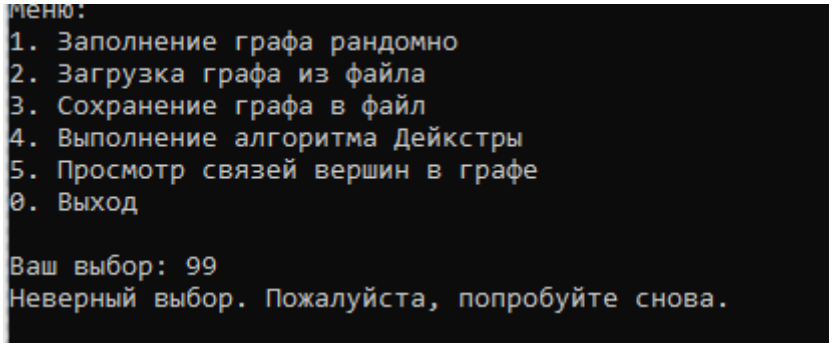

Рисунок 12 – Проверка на неправильный выбор основного действия

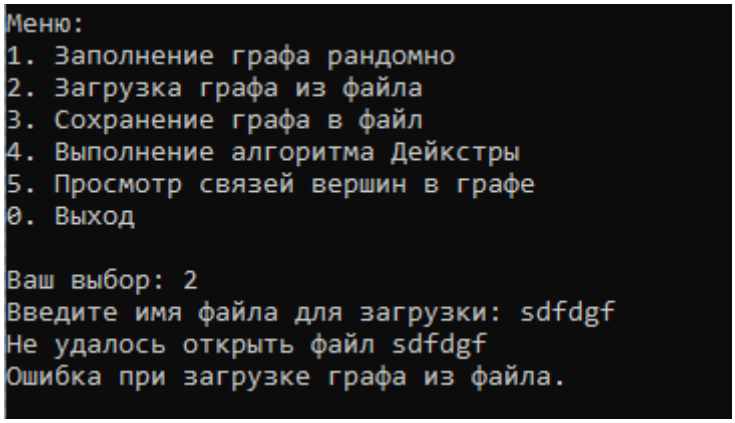

Рисунок 13 – Проверка на введение несуществующего файла

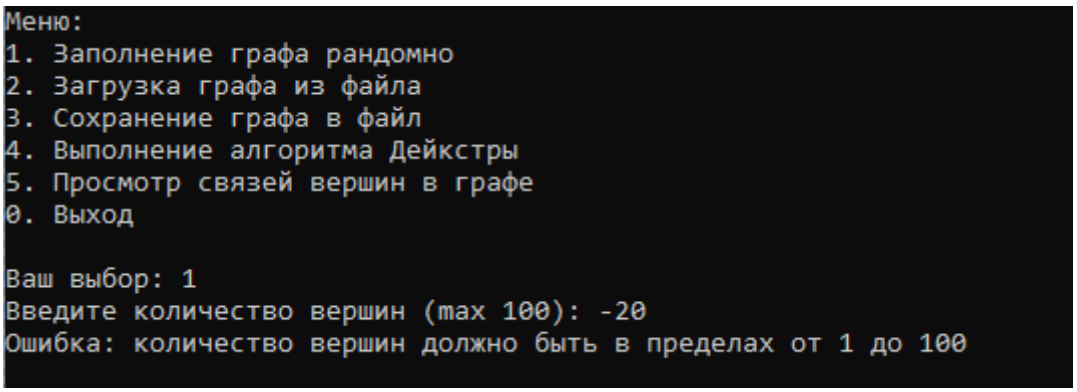

Рисунок 14 – Проверка на введение неправильного размера графа

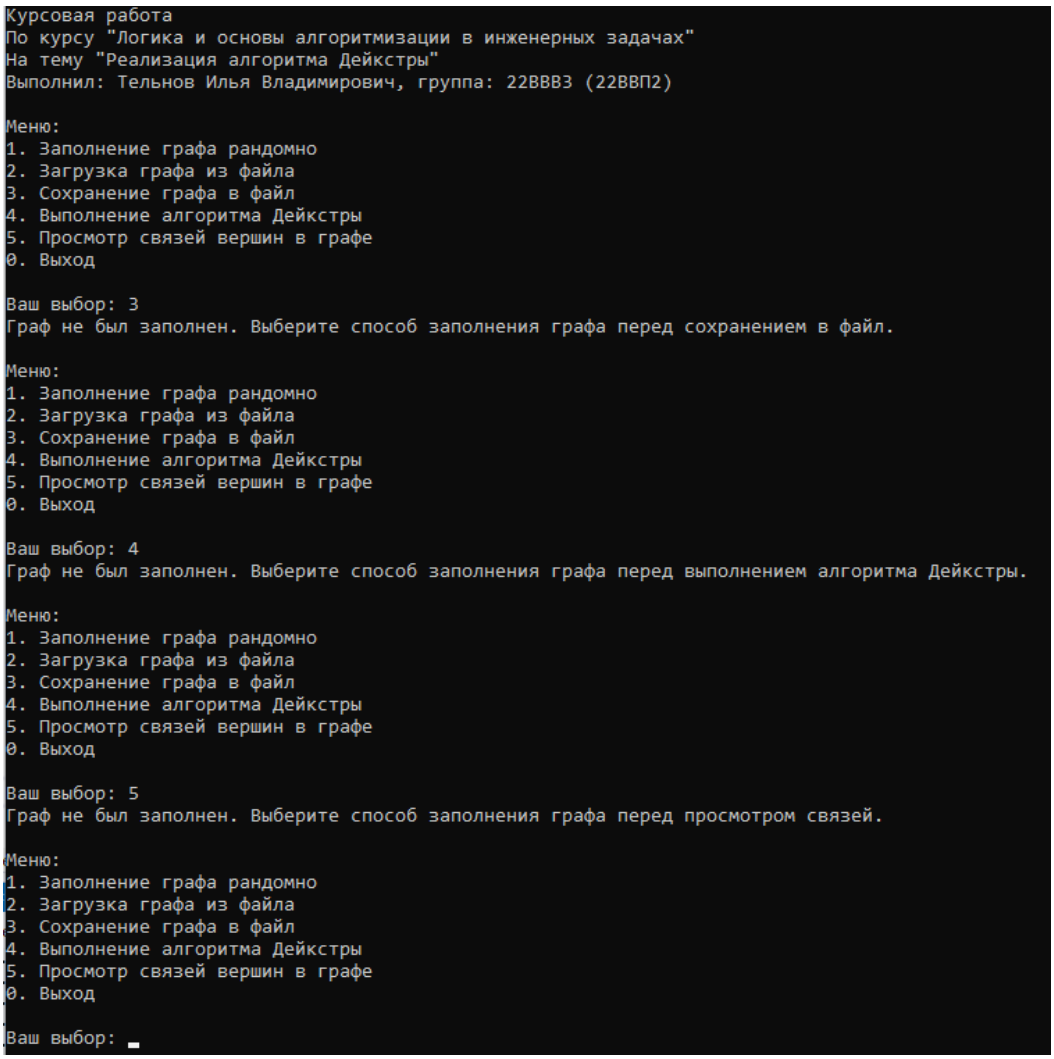

Рисунок 15 – Проверка на выполнение действий без графа

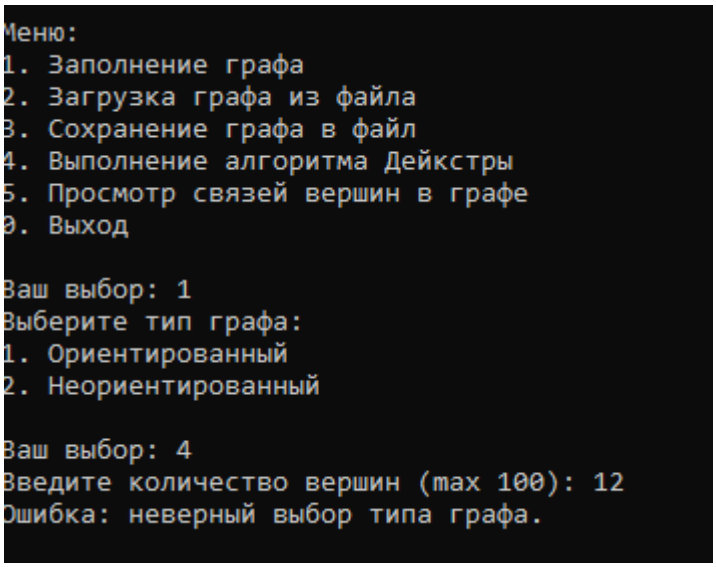

Рисунок 16 – Проверка на выбор неправильного типа графа

6. Ручной расчёт задачи

Проведём проверку программы посредством ручных вычислений:

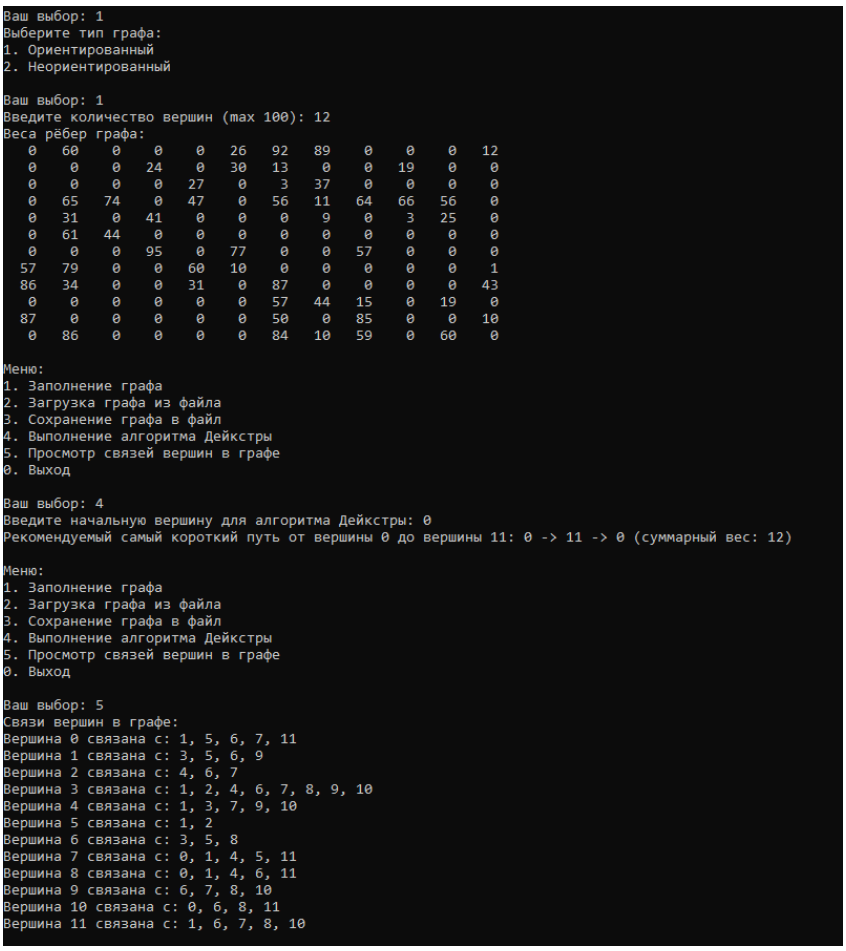

Рисунок 17 – Ориентированный граф и результат

Начинаем с вершины 0. Выбираем ребро с минимальным весом: ребро 0 - 11 с весом 12.

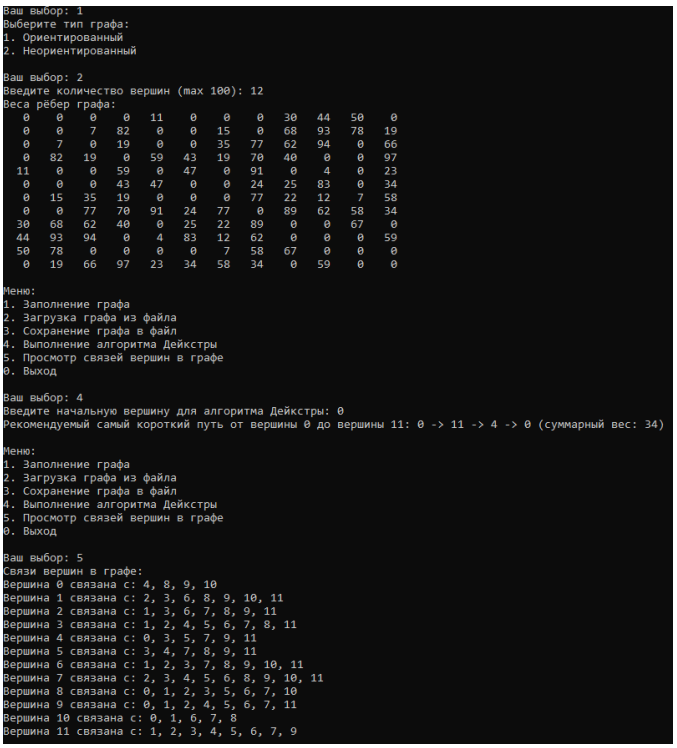

Рисунок 18 – Неориентированный граф и результат

Далее ищем рёбра с минимальным весом по такому же принципу. Из вершины 0 можно пройти в вершину 4 и получить следующее ребро: ребро 0 - 4 с весом 32. Из вершины 4 можно перейти в вершину 11 получаем ребро: ребро 4 - 11 с весом 23. В итоге мы получаем самый короткий путь с суммарным весом 34: 0 - 4 - 11. Таким образом, сделав ручной подсчет на данном примере, мы удостоверились, что программа работает верно.

Заключение

Таким образом, в процессе создания данного проекта была разработана программа, осуществляющая алгоритм Дейкстры для поиска минимального расстояния во взвешенном связном графе при помощи среды Microsoft Visual Studio 2019.

При выполнении курсовой работы были получены практические навыки по разработке программного обеспечения и освоены техники создания матриц смежности, фундаментально опирающиеся на теорию графов. Также были углублены знания языка программирования Си.

Программа обладает ограниченным, но достаточным для использования функционалом.

Список литературы

1. Брайан Керниган, Деннис Ритчи «Язык программирования Си»
2. М. Уэит, С. Прата, Д. Мартин «Язык Си. Руководство для начинающих»
3. Кормен Т., Лейзерсон Ч., Ривест Р. «Алгоритмы: Построение и анализ» - М.: МЦНМО, 2001. - 960 с.
4. Оре О. «Графы и их применение»: Пер. с англ. 1965. 176 с.

Приложение А. Листинг программы.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <locale.h>
#include <string.h>
#include <time.h>

#define MAX_VERTICES 100
#define MAX_FILENAME_LENGTH 100

int parent[MAX_VERTICES];

// Прототипы функций
void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, int start, int end,
int* shortestPathLength);
void fillRandomGraph(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, int
isDirected);
void fillUndirectedGraphRandom(int graph[MAX_VERTICES][MAX_VERTICES], int
vertices);
int loadGraphFromFile(int graph[MAX_VERTICES][MAX_VERTICES], int* vertices, char*
filename);
void saveGraphToFile(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, char*
filename);
void printShortestPath(int graph[MAX_VERTICES][MAX_VERTICES], int start, int end, int
shortestPathLength);
void printGraph(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, int
printWeights);
int getInputInteger(const char* prompt);
char* getInputString(const char* prompt, char* buffer, size_t bufferSize);

int main() {
    setlocale(LC_ALL, "RUS");
    int graph[MAX_VERTICES][MAX_VERTICES] = { 0 };
    int vertices = 0;
    int choice;
    int graphFilled = 0;
    int shortestPathLength = INT_MAX;
    char filename[MAX_FILENAME_LENGTH];

    printf("Пензенский государственный университет\n");
    printf("Кафедра «Вычислительная техника»\n");
    printf("Приняли: Юрова О.В. и Акифьев И.В.\n");
    printf("Курсовая работа\n");
    printf("По курсу \"Логика и основы алгоритмизации в инженерных задачах\"\n");
    printf("На тему \"Реализация алгоритма Дейкстры\"\n");
    printf("Выполнил: Тельнов Илья Владимирович, группа: 22ВВВ3 (22ВВП2)\n");

    while (1) {
```

```
        printf("\nМеню:\n");
        printf("1. Заполнение графа\n");
        printf("2. Загрузка графа из файла\n");
        printf("3. Сохранение графа в файл\n");
        printf("4. Выполнение алгоритма Дейкстры\n");
        printf("5. Просмотр связей вершин в графе\n");
        printf("0. Выход\n");
        printf("\nВаш выбор: ");

        if (scanf("%d", &choice) != 1) {
            printf("Ошибка ввода. Пожалуйста, введите целое число.\n");
            fflush(stdin); // Очистка буфера ввода
            continue;
        }

        switch (choice) {
        case 1: {
            printf("Выберите тип графа:\n");
            printf("1. Ориентированный\n");
            printf("2. Неориентированный\n");
            printf("\nВаш выбор: ");
            int graphTypeChoice;
            if (scanf("%d", &graphTypeChoice) != 1) {
                printf("Ошибка ввода. Пожалуйста, введите целое число.\n");
                fflush(stdin);
                break;
            }
            vertices = getInputInteger("Введите количество вершин (max 100): ");
            if (vertices <= 0 || vertices > MAX_VERTICES) {
                printf("Ошибка: количество вершин должно быть в пределах от 1 до %d\n",
MAX_VERTICES);
                break;
            }
            if (graphTypeChoice == 1 || graphTypeChoice == 2) {
                if (graphTypeChoice == 1) {
                    fillRandomGraph(graph, vertices, 1);
                }
                else {
                    fillUndirectedGraphRandom(graph, vertices);
                }
                graphFilled = 1;
                printGraph(graph, vertices, 1);
            }
            else {
                printf("Ошибка: неверный выбор типа графа.\n");
            }
            break;
        }
        case 2:
            if (!graphFilled) {
                // Оставьте запрос на имя файла
                char filename_load[MAX_FILENAME_LENGTH];
```

```
            printf("Введите имя файла для загрузки: ");
            scanf("%s", filename_load);
            if (loadGraphFromFile(graph, &vertices, filename_load)) {
                graphFilled = 1;
            }
            else {
                printf("Ошибка при загрузке графа из файла.\n");
            }
        }
        else {
            printf("Граф уже заполнен. Выберите другую опцию.\n");
        }
        break;
    case 3:
        if (graphFilled) {
            saveGraphToFile(graph, vertices, getInputString("Введите имя файла для
сохранения: ", filename, MAX_FILENAME_LENGTH));
        }
        else {
            printf("Граф не был заполнен. Выберите способ заполнения графа перед
сохранением в файл.\n");
        }
        break;
    case 4:
        if (!graphFilled) {
            printf("Граф не был заполнен. Выберите способ заполнения графа перед
выполнением алгоритма Дейкстры.\n");
        }
        else {
            int startVertex = getInputInteger("Введите начальную вершину для алгоритма
Дейкстры: ");
            dijkstra(graph, vertices, startVertex, vertices - 1, &shortestPathLength);
            printf("Рекомендуемый самый короткий путь от вершины %d до вершины %d: ",
startVertex, vertices - 1);
            printShortestPath(graph, startVertex, vertices - 1, shortestPathLength);
            printf("\n");
        }
        break;
    case 5:
        if (graphFilled) {
            printGraph(graph, vertices, 0);
        }
        else {
            printf("Граф не был заполнен. Выберите способ заполнения графа перед
просмотром связей.\n");
        }
        break;
    case 0:
        return 0; // Завершение программы при выборе "0"
    default:
        printf("Неверный выбор. Пожалуйста, попробуйте снова.\n");
        break;
```

```
        }
    }

    return 0;
}

void printGraph(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, int
printWeights) {
    if (printWeights) {
        printf("Веса рёбер графа:\n");
        for (int i = 0; i < vertices; ++i) {
            for (int j = 0; j < vertices; ++j) {
                printf("%4d ", graph[i][j]);
            }
            printf("\n");
        }
    }
    else {
        printf("Связи вершин в графе:\n");
        for (int i = 0; i < vertices; ++i) {
            printf("Вершина %d связана с: ", i);
            int first = 1;
            for (int j = 0; j < vertices; ++j) {
                if (graph[i][j]) {
                    if (!first) {
                        printf(", ");
                    }
                    printf("%d", j);
                    first = 0;
                }
            }
            printf("\n");
        }
    }
}

void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, int start, int end,
int* shortestPathLength) {
    int visited[MAX_VERTICES] = { 0 };
    int distance[MAX_VERTICES];

    for (int i = 0; i < vertices; ++i) {
        distance[i] = INT_MAX;
        visited[i] = 0;
    }

    distance[start] = 0;
    parent[start] = -1;

    for (int i = 0; i < vertices - 1; ++i) {
        int minDistance = INT_MAX;
        int minIndex = -1;
```

```
        for (int j = 0; j < vertices; ++j) {
            if (!visited[j] && distance[j] < minDistance) {
                minDistance = distance[j];
                minIndex = j;
            }
        }

        visited[minIndex] = 1;

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

            if (!visited[j] && graph[minIndex][j] && distance[minIndex] != INT_MAX
                && distance[minIndex] + graph[minIndex][j] < distance[j]) {
                distance[j] = distance[minIndex] + graph[minIndex][j];
                parent[j] = minIndex;
            }
        }
    }

    *shortestPathLength = distance[end];
}

void fillRandomGraph(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, int
isDirected) {
    srand(time(NULL));  // Инициализация генератора случайных чисел

    // Вероятность установки связи между вершинами (в процентах)
    int connectionProbability = 20; // Например, 30% вероятность

    // Заполняем граф значениями от 0 до 100
    for (int i = 0; i < vertices; ++i) {
        for (int j = 0; j < vertices; ++j) {
            if (i != j && (rand() % 50) < connectionProbability) {
                graph[i][j] = rand() % 101; // Задаем случайный вес связи от 0 до 10
                if (!isDirected) {
                    graph[j][i] = graph[i][j]; // Для неориентированного графа отражаем изменения
                }
            }
            else {
                graph[i][j] = 0;
            }
        }
    }
}

void fillUndirectedGraphRandom(int graph[MAX_VERTICES][MAX_VERTICES], int
vertices)
{
    srand(time(NULL));
    int connectionProbability = 20; // Например, 30% вероятность
```

```
    // Заполнение графа нулями и единицами на главной диагонали
    for (int i = 0; i < vertices; ++i) {
        for (int j = 0; j < vertices; ++j) {
            if (i != j && (rand() % 50) < connectionProbability) {
                if (i == j) {
                    graph[i][j] = 0; // Главная диагональ остается нулевой

                }
                else {
                    graph[i][j] = rand() % 101; // Заполнение ребер 0 или 1
                    graph[j][i] = graph[i][j]; // Отражение изменений для неориентированного графа

                }
            }
        }
    }
// Замена единиц случайными числами
    for (int i = 0; i < vertices; ++i) {
        for (int j = 0; j < vertices; ++j) {
            if (i != j && graph[i][j] == 1) {
            graph[i][j] = rand() % 100; // Заполнение случайным числом вместо единицы
            graph[j][i] = graph[i][j]; // Отражение изменений для неориентированного

            }
        }
    }
}


// Загрузка графа из файла
int loadGraphFromFile(int graph[MAX_VERTICES][MAX_VERTICES], int* vertices, char*
filename) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("Не удалось открыть файл %s\n", filename);
        return 0;
    }

    fscanf(file, "%d", vertices);

    for (int i = 0; i < *vertices; ++i) {
        for (int j = 0; j < *vertices; ++j) {
            fscanf(file, "%d", &graph[i][j]);
        }
    }

    fclose(file);
    return 1;
}

// Функция сохранения графа в файл
```

```
void saveGraphToFile(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, char*
filename) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("Не удалось создать файл %s\n", filename);
        return;
    }

    fprintf(file, "%d\n", vertices);

    for (int i = 0; i < vertices; ++i) {
        for (int j = 0; j < vertices; ++j) {
            fprintf(file, "%d ", graph[i][j]);
        }
        fprintf(file, "\n");
    }

    fclose(file); // Закрываем файл после записи данных
}

// Функция вывода кратчайшего пути


// Функция вывода кратчайшего пути
void printShortestPath(int graph[MAX_VERTICES][MAX_VERTICES], int start, int end, int
shortestPathLength) {
    if (parent[end] == -1) {
        printf("Путь не найден.\n");
        return;
    }

    printf("%d", start);
    int current = end;
    int totalWeight = 0; // Суммарный вес кратчайшего пути

    while (current != -1) {
        printf(" -> %d", current);
        if (parent[current] != -1) {
            totalWeight += graph[parent[current]][current];
        }
        current = parent[current];
    }

    printf(" (суммарный вес: %d)", totalWeight);
}


int getInputInteger(const char* prompt) {
    int value;
    printf("%s", prompt);
    while (1) {
        if (scanf("%d", &value) == 1) {
```

```
            while (getchar() != '\n');  // Очистка буфера ввода
            return value;
        }
        else {
            printf("Ошибка ввода. Пожалуйста, введите целое число: ");
            while (getchar() != '\n');  // Очистка буфера ввода
        }
    }
}

char* getInputString(const char* prompt, char* buffer, size_t bufferSize) {
    printf("%s", prompt);
    while (1) {
        if (scanf("%s", buffer) == 1) {
            while (getchar() != '\n');  // Очистка буфера ввода
            return buffer;
        }
        else {
            printf("Ошибка ввода. Пожалуйста, введите строку: ");
            while (getchar() != '\n');  // Очистка буфера ввода
        }
    }
}
```