

Министерство науки и высшего 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 Флойда»

Выполнил:

студент группы 23BVB4

Епинин Дмитрий

Принял:

к.т.н. Юрова О.В.

18.12.24
хорошо
ома

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 Етимишу Д.В.

Группа 23ВВВ4

Тема проекта Реализация алгоритма Рлобда

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 небольшой участке работы алгоритма);
5. Описание самой программы;
6. Тесты;
7. Список литературы;
8. Листинг программы;
9. Результат работы программы.

Объем работы по курсу

1. Расчетная часть

Ручной расчёт 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 курсового

2 Разработка алгоритмов программы

3 Разработка программы

4 Тестирование и завершение разработки программы

5 Оформление пояснительной записки

6

7

8

Дата выдачи задания "17" сентября 2024г.

Дата защиты проекта " "

Руководитель Нрова О.В. ф.и.о.

Задание получил "17" сентября 2024 г.

Студент Епшин Д.В. Епшин

Содержание

Реферат.....	6
Введение.....	7
Постановка задачи.....	8
Теоретическая часть задания.....	9
Описание алгоритма.....	11
Описание программы.....	17
Тестирование.....	27
Ручной расчет программы.....	30
Заключение.....	33
Список литературы.....	34
Приложение А. Листинг программы.....	35

Реферат

Отчет 34 страниц, 17 рисунков, 1 таблица, 1 приложение

**АЛГОРИТМ ФЛОЙДА-УОРШЕЛЛА, ТЕОРИЯ ГРАФОВ, ГРАФ,
КРАТЧАЙШИЕ ПУТИ, МАТРИЦА КРАТЧАЙШИХ РАССТОЯНИЙ, ВЕС
РЕБЕР.**

Цель исследования — разработка программы, предназначенная для нахождения кратчайших путей между всеми парами вершин в графе, как ориентированном, так и неориентированном, используя алгоритм Флойда-Уоршелла.

В работе рассматривается алгоритм Флойда-Уоршелла, который используется для нахождения кратчайших путей между всеми парами вершин в ориентированном графе. Установлено, что с помощью данного алгоритма можно эффективно вычислить расстояния между любыми парами вершин в графе, независимо от его структуры — он может быть как несвязанным, так и связанным.

Введение

Алгоритм Флойда-Уоршелла (англ. Floyd-Warshall algorithm) позволяет найти кратчайшие пути между всеми парами вершин взвешенного графа. Этот алгоритм является одним из наиболее известных и эффективных методов для решения задачи поиска кратчайших путей в графах, как ориентированных, так и неориентированных. Основное преимущество алгоритма Флойда-Уоршелла заключается в его способности обрабатывать графы с отрицательными весами ребер, при условии отсутствия циклов отрицательного веса.

Отличие алгоритма Флойда-Уоршелла от других алгоритмов поиска кратчайших путей, таких как алгоритм Дейкстры, заключается в том, что он находит кратчайшие пути между всеми парами вершин, а не только от одной начальной вершины до всех остальных. Это делает его особенно полезным в задачах, где требуется анализ всех возможных маршрутов в графе.

Алгоритм Флойда-Уоршелла работает путем последовательного обновления матрицы расстояний между вершинами, используя промежуточные вершины для улучшения текущих оценок кратчайших путей. В результате, после завершения работы алгоритма, матрица расстояний содержит кратчайшие пути между всеми парами вершин.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, язык программирования – C. Целью данной курсовой работы является разработка программы на языке C. Именно с его помощью в данном курсовом проекте реализуется алгоритм Флойда-Уоршелла, осуществляющий поиск кратчайших путей между всеми парами вершин в графе.

1 Постановка задачи

Требуется разработать программу, которая выделит компоненты сильной связности орграфа, то есть поиск кратчайших путей, используя алгоритм Флойда-Уоршелла.

Исходных граф в программе должен задаваться матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности орграфа, вид орграфа и все компоненты сильной связности орграфа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройства ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом № 15.

2 Теоретическая часть задания

Граф G (рисунок 1) представляет собой ориентированный взвешенный граф с пятью вершинами (A, B, C, D, E) и множествами ребер, соединяющими эти вершины. Каждое ребро имеет вес, указанный рядом с ним. Ребра из множества A ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется ориентированным графом.

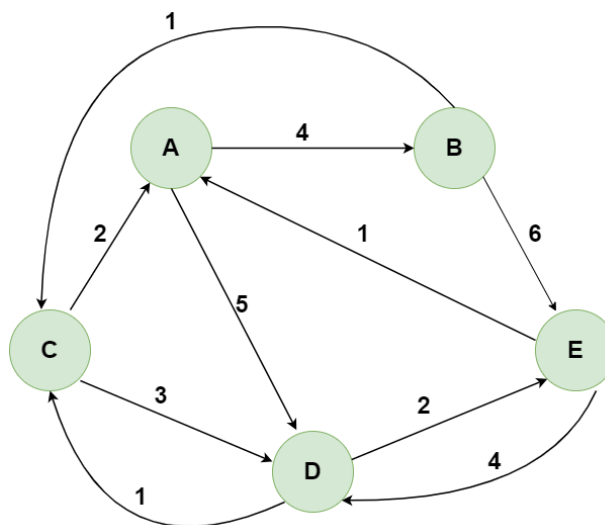


Рисунок 1 – Пример орграфа

При представлении графа смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одного элемента матрицы соответствует весу ребра между вершинами. Если ребра между вершинами нет, то элемент матрицы равен бесконечности (INT_MAX).

Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой, что каждая вершина графа просматривается только один раз, и переход от одной вершины к другой осуществляется по ребрам графа. Остановимся на одном из этих стандартных методов такого перебора – алгоритм Флойда-Уоршелла.

Алгоритм Флойда-Уоршелла используется для нахождения кратчайших путей между всеми парами вершин в взвешенном графе. Он работает с матрицей

смежности, где каждый элемент матрицы представляет собой вес ребра между соответствующими вершинами.

Алгоритм Флойда – Уоршелла работает следующим образом:

1. Инициализация: Создается матрица расстояний D и матрица предшественников P . Матрица D инициализируется весами ребра графа, а если ребра между вершинами нет, то элемент матрицы устанавливается в бесконечность. Матрица P инициализируется значениями по умолчанию, указывающими на отсутствие указывающих вершин.
2. Обновление матрицы расстояний: алгоритм последовательно обновляет матрицу расстояний, используя промежуточные вершины для улучшения текущих оценок кратчайших путей. Для каждой тройки вершин (i, j, k) выполняется проверка: если путь через промежуточную вершину k короче текущего кратчайшего пути между вершинами i и j , то обновляется значение в матрице D и матрице P .
3. Завершение: После завершения всех итераций матрица D содержит кратчайшие расстояния между всеми парами вершин, а матрица P позволяет восстановить сами кратчайшие пути.

В данном контексте, связным называется подграф, в котором любые две его вершины связаны. Две вершины X_1 и X_2 считаются связанными, если существует направленный путь от X_1 к X_2 и также направленный путь от X_2 к X_1 . В процессе выполнения алгоритма мы анализируем все пары вершин, чтобы проверить наличие ориентированных путей между ними.

Поскольку для каждой пары вершин существует ориентированный путь в обе стороны, данный орграф является сильно связанным. Компонента сильной связности данного орграфа включает все вершины: $\{A, B, C, D, E\}$.

Таким образом, данный орграф является сильно связанным, так как любые две его вершины сильно связаны.

3 Описание алгоритма программы

Для программной реализации понадобится объявить методы такие как `printMatrisa`, `Floyd`, `generateRandomMatrisa`, `inputMatrisa`, `saveFile`, `loadFile`.

Основной цикл программы:

1) Пользователь выбирает действие из меню: создать матрицу графа, загрузить файл или завершить программу.

2) Если выбрано создание матрицы графа, то:

2.1) В зависимости от выбора пользователя, матрица графа создается вручную, генерируется случайным образом при этом выбирается тип графа, ориентированный или неориентированный.

2.2) Выводится созданная матрица графа.

2.3) Выполняется алгоритм Флойда-Уоршелла для нахождения кратчайших путей между всеми парами вершин.

2.4) Результаты сохраняются в файл.

Для хранения дистанции, то есть динамическое выделение памяти для матрицы расстояний используется переменная `int dist`, которая в дальнейшем позволяет добавлять промежуточные вершины и в конце выполнения программы позволяет выводить матрицу кратчайших путей, то есть выполнять алгоритм Флойда. Используется в следующих методах: `printMatrisa` и `Floyd`.

Чтобы пользователь мог определить тип графа, ориентированный или неориентированный, используется переменная `int directed`, которая используется в следующих методах: `int main`, `void generateRandomMatrisa`, `inputMatrisa`. Если матрица ориентированная, то переменная имеет значение 1, если иначе тогда равна 0.

Для того чтобы определить количество вершин в графе, используется переменная `int V`. Изначально значение данной переменной имело значение `int V = 0`, когда пользователь вводит с клавиатуры, значение данной переменной задается пользователем. Благодаря данной переменной пользователь может

вводить количество вершин в графе, то есть определять размерность матрицы смежности. Используется во всех методах в программной реализации.

Для динамического выделения памяти для матрицы графа используется переменная `int graph`, с помощью данной переменной программа выделила место, где с помощью меню выбирается та часть выполнения программы, например, осуществление генерации случайной матрицы смежности или программа предоставляет возможность заполнить ее с помощью клавиатуры пользователем. Используется в методе `int main` и вышеперечисленных методах, таких как `printMatrisa`, `Floyd`, `generateRandomMatrisa`, `inputMatrisa`, `saveFile`, `loadFile`.

Ниже представлен псевдокод методов `Floyd()`, `saveFile()`, `loadFile()` и `int main()`.

Floyd(int graph, int V)**

1. Динамическое выделение памяти для матрицы расстояний `dist`
2. для $i = 0$ пока $i < V$ делать $i = i + 1$
3. `dist[i] = (int*)malloc(V * sizeof(int))`
4. конец цикла
5. для $i = 0$ пока $i < V$ делать $i = i + 1$
6. для $j = 0$ пока $j < V$ делать $j = j + 1$
7. `dist[i][j] = graph[i][j]`
8. конец вложенного цикла
9. конец цикла
10. для $k = 0$ пока $k < V$ делать $k = k + 1$
11. для $i = 0$ пока $i < V$ делать $i = i + 1$
12. для $j = 0$ пока $j < V$ делать $j = j + 1$

13. если $\text{dist}[i][k] \neq \text{INT_MAX}$ и $\text{dist}[k][j] \neq \text{INT_MAX}$ и $\text{dist}[i][k] + \text{dist}[k][j] < \text{dist}[i][j]$

14. $\text{dist}[i][j] = \text{dist}[i][k] + \text{dist}[k][j]$

15. конец условия

16. конец вложенного цикла

17. конец цикла

18. вызвать функцию `printMatrisa(dist, V)`

19. для $i = 0$ пока $i < V$ делать $i = i + 1$

20. `free(dist[i])`

21. конец цикла

22. `free(dist)`

int main()

1. Установлен локаль для русского языка

2. Инициализирован генератор случайных чисел

3. Инициализированы переменные `V, choice, exit, directed = 0`

4. Пока `exit` равно 0:

5. Вывод меню выбора действия

6. Считывание выбора пользователя в переменную `choice`

7. Если `choice == 1`:

8. Вывод запроса на ввод количества вершин

9. Считывание количества вершин в переменную `V`

10. Выделение памяти для матрицы смежности размером $V \times V$

11. Вывод запроса на выбор типа графа

12. Считывание выбора пользователя в переменную directed
13. Если directed равно 1, то установить directed в 1 (ориентированный граф), иначе установить directed в 0 (неориентированный граф)
14. Вывести запрос на выбор способа создания матрицы графа
15. Считать выбор пользователя в переменную choice
16. Если choice равно 1:
17. Генерация случайной матрицы смежности с учетом типа графа (directed)
18. Иначе если choice равно 2:
19. Ввод матрицы смежности вручную с учетом типа графа (directed)
20. Иначе:
21. Вывод сообщения о неверном выборе
22. Продолжить цикл
23. Вывод сгенерированной матрицы смежности
24. Иначе если choice равно 2:
25. Вывод запроса на ввод имени файла для загрузки
26. Считывание имени файла в переменную filename
27. Вывод запроса на ввод количества вершин
28. Считывание количества вершин в переменную V
29. Выделение памяти для матрицы смежности размером $V \times V$
30. Загрузка матрицы смежности из файла
31. Вывести загруженную матрицу смежности
32. Выполнение алгоритма Флойда для нахождения кратчайших путей
33. Освобождение памяти, выделенную для матрицы смежности

34. Продолжить цикл
35. Иначе если choice равно 3:
36. Установить exit == 1 (завершить программу)
37. Иначе:
38. Вывод сообщения о неверном выборе
39. Продолжить цикл
40. Вывод матрицы смежности
41. Если матрица графа существует:
42. Выполнение алгоритма Флойда для нахождения кратчайших путей
43. Вывод запроса на ввод имени файла для сохранения матрицы
44. Считывание имени файла в переменную saveFilename
45. Сохранение матрицы смежности в файл
46. Освобождение памяти, выделенную для матрицы смежности
47. Конец программы

saveFile(int graph, int V, const char* filename)**

1. Открыть файл для записи с именем filename
2. Если файл не удалось открыть
3. Вывести сообщение "Не удалось открыть файл для записи."
4. Завершить функцию
5. Конец условия
6. Для $i = 0$ пока $i < V$ делать $i = i + 1$
7. Для $j = 0$ пока $j < V$ делать $j = j + 1$

8. Записать в файл значение `graph[i][j]`
9. Записать в файл символ новой строки
10. Конец цикла
11. Заккрыть файл
12. Вывести сообщение "Матрица успешно сохранена в файл `filename`."

`loadFile(int graph, int V, const char* filename)`**

1. Открыть файл для чтения с именем `filename`
2. Если файл не удалось открыть
3. Вывести сообщение "Не удалось открыть файл для чтения."
4. Завершить функцию
5. Конец условия
6. Для $i = 0$ пока $i < V$ делать $i = i + 1$
7. Для $j = 0$ пока $j < V$ делать $j = j + 1$
8. Считать из файла значение `graph[i][j]`
9. Конец вложенного цикла
10. Конец цикла
11. Заккрыть файл
12. Вывести сообщение "Матрица успешно загружена из файла `filename`."

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 в терминале Windows (консольное приложение C++).

Данная программа является многомодульной поскольку состоит из нескольких методов: `int main`, `printMatrisa`, `Floyd`, `generateRandomMatrisa`, `inputMatrisa`, `saveFile`, `loadFile`.

Работа программы начинается с выбора пункта в меню – генерация матрицы смежности, либо загрузка файла, либо завершение программы. Если пользователь выбрал ввод количества вершин и создания матрицы графа, то на экране выводится запрос на количество вершин в графе. После этого, пользователю предлагается выбрать вид графа – ориентированный или неориентированный. Затем выводится запрос на ввод матрицы. Пользователь может ввести матрицу смежности вручную, иначе программа сделает генерацию матрицы смежности в автоматическом формате. Далее программа позволяет сохранить результат программы в виде файла, имя файла вводится пользователем вручную. А также имеется возможность выгрузить содержимое файла в консоль. Вид кода, который осуществляет данную реализацию программы:

```
int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));
    int V, choice, exit = 0, directed;
    int** graph = NULL;
    while (!exit) {
        printf("Выберите действие:\n");
        printf("1. Ввести количество вершин и создать матрицу графа\n");
        printf("2. Загрузить матрицу графа из файла\n");
        printf("3. Завершить программу\n");
        scanf("%d", &choice);
        if (choice == 1) {
            printf("Введите количество вершин в графе: ");
            scanf("%d", &V);
            // Динамическое выделение памяти для матрицы графа
```

```

graph = (int**)malloc(V * sizeof(int*));
for (int i = 0; i < V; i++) {
    graph[i] = (int*)malloc(V * sizeof(int));
}
printf("Выберите тип графа:\n");
printf("1. Ориентированный\n");
printf("2. Неориентированный\n");
scanf("%d", &directed);
directed = (directed == 1) ? 1 : 0;
printf("Выберите способ создания матрицы графа:\n");
printf("1. Сгенерировать случайным образом\n");
printf("2. Ввести самостоятельно\n");
scanf("%d", &choice);
if (choice == 1)
    generateRandomMatriza(graph, V, directed);
else if (choice == 2)
    inputMatriza(graph, V, directed);
else {
    printf("Неверный выбор. Попробуйте снова.\n");
    continue;
}
}
else if (choice == 2) {
    char filename[100];
    printf("Введите имя файла для загрузки: ");
    scanf("%s", filename);
    printf("Введите количество вершин в графе: ");
    scanf("%d", &V);
    graph = (int**)malloc(V * sizeof(int*));
    for (int i = 0; i < V; i++) {
        graph[i] = (int*)malloc(V * sizeof(int));
    }
    loadFile(graph, V, filename);
    // Вывод загруженной матрицы графа
    printf("Матрица графа:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (graph[i][j] == INT_MAX)
                printf("%7s", "INF");
            else
                printf("%7d", graph[i][j]);
        }
        printf("\n");
    }
    // Выполнение алгоритма Флойда
    Floyd(graph, V);
    // Освобождение памяти
    for (int i = 0; i < V; i++) {
        free(graph[i]);
    }
    free(graph);
    graph = NULL;
    continue; // Возвращаемся к началу меню
}
else if (choice == 3) {
    exit = 1;
}
else {
    printf("Неверный выбор. Попробуйте снова.\n");
    continue;
}
}

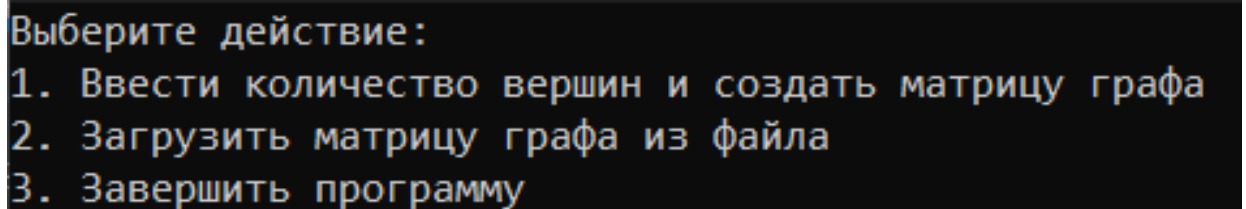
```

```

    }
    printf("Матрица графа:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (graph[i][j] == INT_MAX)
                printf("%7s", "INF");
            else
                printf("%7d", graph[i][j]);
        }
        printf("\n");
    }
    if (graph != NULL) {
        // Выполнение алгоритма Флойда
        Floyd(graph, V);
        char saveFilename[100];
        printf("Введите имя файла для сохранения матрицы: ");
        scanf("%s", saveFilename);
        saveFile(graph, V, saveFilename);
        // Освобождение памяти
        for (int i = 0; i < V; i++) {
            free(graph[i]);
        }
        free(graph);
        graph = NULL;
    }
}
return 0;
}

```

Ниже можно увидеть оформление начального запроса и дальнейшее действие с ним.

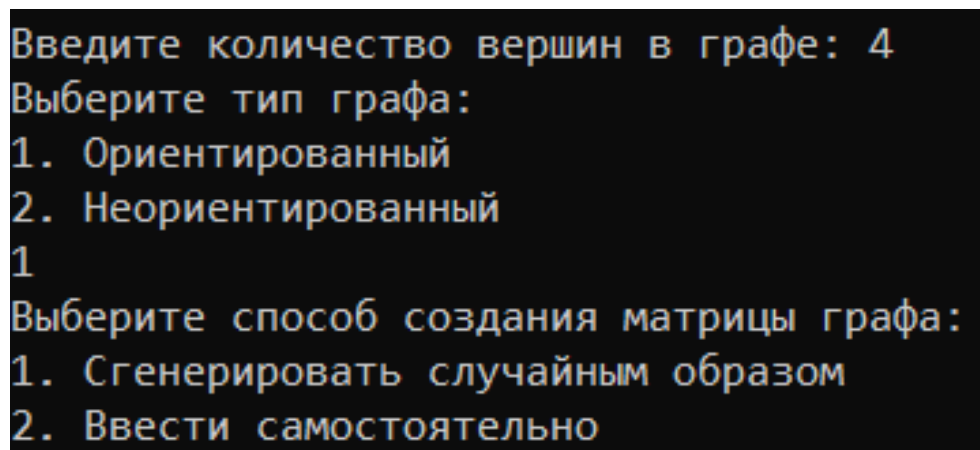


```

Выберите действие:
1. Ввести количество вершин и создать матрицу графа
2. Загрузить матрицу графа из файла
3. Завершить программу

```

Рисунок 2 – Ввод количества вершин в графе



```

Введите количество вершин в графе: 4
Выберите тип графа:
1. Ориентированный
2. Неориентированный
1
Выберите способ создания матрицы графа:
1. Сгенерировать случайным образом
2. Ввести самостоятельно

```

Рисунок 3 – Определение типа графа


```

Выберите тип графа:
1. Ориентированный
2. Неориентированный
1
Выберите способ создания матрицы графа:
1. Сгенерировать случайным образом
2. Ввести самостоятельно
1
Матрица графа:
    10    INF     0     4
    2    13    10    18
    INF    17     3     2
    INF    17   INF   INF
Матрица кратчайших путей:
    10    17     0     2
    2    13     2     4
    19    17     3     2
    19    17    19    21

```

Рисунок 4 – Генерация матрицы случайным образом

```

Выберите способ создания матрицы графа:
1. Сгенерировать случайным образом
2. Ввести самостоятельно
2
Введите матрицу графа (используйте -1 для INF):
Введите ребро (0, 0): -1
Введите ребро (0, 1): -1
Введите ребро (0, 2): 3
Введите ребро (1, 0): 6
Введите ребро (1, 1): 5
Введите ребро (1, 2): 13
Введите ребро (2, 0): 18
Введите ребро (2, 1): -1
Введите ребро (2, 2): 7
Матрица графа:
    INF    INF     3
    6     5    13
    18    INF     7
Матрица кратчайших путей:
    21    INF     3
    6     5     9
    18    INF     7

```

Рисунок 5 – Ввод матрицы с клавиатуры

```

Матрица кратчайших путей:
    21    INF    3
     6     5     9
    18    INF    7
Введите имя файла для сохранения матрицы: File
Матрица успешно сохранена в файл File.

```

Рисунок 6 – Сохранение матрицы в файл

```

Выберите действие:
1. Ввести количество вершин и создать матрицу графа
2. Загрузить матрицу графа из файла
3. Завершить программу
2
Введите имя файла для загрузки: File
Введите количество вершин в графе: 3
Матрица успешно загружена из файла File.
Матрица графа:
    INF    INF    3
     6     5    13
    18    INF    7
Матрица кратчайших путей:
    21    INF    3
     6     5     9
    18    INF    7

```

Рисунок 7 – Загрузка матрицы из файла

Если пользователь выбрал первый пункт меню выполняется метод `generateRandomMatriza`, в которой программа генерирует числа случайным образом для заполнения матрицы смежности. Ниже представлен метод как реализуется заполнение матрицы (рисунок 8):

```

void generateRandomMatriza(int** graph, int V, int directed) {
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (rand() % 100 < 20) {
                graph[i][j] = INT_MAX; // INF
            }
            else {

```

```

        graph[i][j] = rand() % 20;
    }
    if (!directed)
        graph[j][i] = graph[i][j];
    }
}

```

Выберите способ создания матрицы графа:
 1. Сгенерировать случайным образом
 2. Ввести самостоятельно
 1

Матрица графа:

INF	0	11	INF
18	17	10	5
5	17	9	3
18	12	11	4

Матрица кратчайших путей:

15	0	10	5
15	15	10	5
5	5	9	3
16	12	11	4

Рисунок 8 – Генерация матрицы смежности случайным образом

После того как матрица смежности была сгенерирована случайными образом или пользователь ввел ее вручную в автоматическом порядке выполняется метод Floyd. Который выполняет поиск кратчайших путей в матрице смежности. Алгоритм выполняется до тех пор, пока не в матрице смежности не пропадут значения INF, что означает что это число бесконечности. Поиск кратчайших путей происходит путем поиска наименьшего значения в матрице, реализация метода представлена ниже (рисунок 9):

```

void Floyd(int** graph, int V) {
    int** dist = (int**)malloc(V * sizeof(int*));
    for (int i = 0; i < V; i++)
        dist[i] = (int*)malloc(V * sizeof(int));
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            dist[i][j] = graph[i][j];
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][k] +
                    dist[k][j] < dist[i][j])

```

```

        dist[i][j] = dist[i][k] + dist[k][j];
    }
}
printMatrisa(dist, V);
for (int i = 0; i < V; i++) {
    free(dist[i]);
}
free(dist);
}

```

Матрица графа:			
16	5	15	1
INF	5	8	3
INF	17	11	INF
7	4	5	0
Матрица кратчайших путей:			
8	5	6	1
10	5	8	3
27	17	11	20
7	4	5	0

Рисунок 9 – Вывод кратчайшего пути после выполнения алгоритма Флойда

Чтобы пользователь имел возможность видеть, что сгенерировала программа или что он ввел с клавиатуры используется метод `printMatrisa`, ниже представлен код реализации метода (рисунок 10):

```

void printMatrisa(int** dist, int V) {
    printf("Матрица кратчайших путей:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INT_MAX)
                printf("%7s", "INF");
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

```

Матрица графа:				
5	INF	6	16	3
INF	13	12	INF	15
INF	12	6	15	15
17	11	14	1	3
4	0	19	0	7
Матрица кратчайших путей:				
5	3	6	3	3
19	13	12	15	15
19	12	6	15	15
7	3	13	1	3
4	0	10	0	3

Рисунок 10 – Вывод матриц

Одним из функционалом программы является ввод значений с клавиатуры. Это позволяет самостоятельно пользователю определить значения матрицы смежности. Данным метод реализован с помощью `inputMatrisa`, код представлен ниже (рисунок 11):

```
void inputMatrisa(int** graph, int V, int directed) {
    printf("Введите матрицу графа (используйте -1 для INF):\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            printf("Введите ребро (%d, %d): ", i, j);
            int value;
            scanf("%d", &value);
            if (value == -1) {
                graph[i][j] = INT_MAX; // INF
            }
            else {
                graph[i][j] = value;
            }
        }
    }
    if (!directed) {
        for (int i = 0; i < V; i++) {
            for (int j = i + 1; j < V; j++) {
                if (graph[i][j] != INT_MAX) {
                    graph[j][i] = graph[i][j];
                }
            }
        }
    }
}
```



```

Введите матрицу графа (используйте -1 для INF):
Введите ребро (0, 0): -1
Введите ребро (0, 1): -1
Введите ребро (0, 2): 5
Введите ребро (1, 0): 17
Введите ребро (1, 1): 34
Введите ребро (1, 2): 22
Введите ребро (2, 0): -1
Введите ребро (2, 1): 2
Введите ребро (2, 2): 55
Матрица графа:
    INF    INF     5
    17     34    22
    INF     2    55
Матрица кратчайших путей:
    24     7     5
    17    24    22
    19     2    24

```

Рисунок 11 – Ввод матрицы с клавиатуры

Для того чтобы сохранить значения матрицы смежности, которые были сгенерированы случайно или пользователь их ввел вручную с клавиатуры, используется метод `saveFile`, код представлен ниже (рисунок 12):

```

void saveFile(int** graph, int V, const char* filename) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("Не удалось открыть файл для записи.\n");
        return;
    }
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (graph[i][j] == INT_MAX) {
                fprintf(file, "-1 "); // Сохраняем INT_MAX как -1
            }
            else {
                fprintf(file, "%d ", graph[i][j]);
            }
        }
        fprintf(file, "\n");
    }
    fclose(file);
    printf("Матрица успешно сохранена в файл %s.\n", filename);
}

```

```

Матрица кратчайших путей:
    16      8      5      7
    11      3     16      2
    11      3     16      2
     9      1     14      3
Введите имя файла для сохранения матрицы: File1
Матрица успешно сохранена в файл File1.

```

Рисунок 12 – Сохранение результата в файл

И последним из функционала программы является реализация метода вывода файла, то есть вывод содержимого файла в консоль программы.

Реализовано с помощью метода loadFile (рисунок 13):

```

void loadFile(int** graph, int V, const char* filename) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("Не удалось открыть файл для чтения.\n");
        return;
    }
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            int value;
            fscanf(file, "%d", &value);
            if (value == -1) {
                graph[i][j] = INT_MAX; // Загружаем -1 как INT_MAX
            }
            else {
                graph[i][j] = value;
            }
        }
    }
    fclose(file);
    printf("Матрица успешно загружена из файла %s.\n", filename);
}

```

```

Выберите действие:
1. Ввести количество вершин и создать матрицу графа
2. Загрузить матрицу графа из файла
3. Завершить программу
2
Введите имя файла для загрузки: File
Введите количество вершин в графе: 3
Матрица успешно загружена из файла File.
Матрица графа:
    INF     INF      3
     6      5     13
    18     INF      7
Матрица кратчайших путей:
    21     INF      3
     6      5      9
    18     INF      7

```

Рисунок 13 – Вывод содержимого файла в консоль

5 Тестирование

Среда разработки Microsoft Visual Studio 2019 представляет 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, чтением файла и выполнением метода Флойда.

Ниже продемонстрирован результат тестирования программы при вводе пользователем количества графа, матрицы смежности генерируемую автоматически и вводимую с клавиатуры вручную.

```
Выберите действие:
1. Ввести количество вершин и создать матрицу графа
2. Загрузить матрицу графа из файла
3. Завершить программу
1
Введите количество вершин в графе: 4
Выберите тип графа:
1. Ориентированный
2. Неориентированный
1
Выберите способ создания матрицы графа:
1. Сгенерировать случайным образом
2. Ввести самостоятельно
1
Матрица графа:
    11      5      5      14
    9      10     17      9
    0      16     15     INF
    19      2     12     10
Матрица кратчайших путей:
    5      5      5     14
    9     10     14      9
    0      5      5     14
    11      2     12     10
```

Рисунок 14 – Тестирование при вводе количество вершин в графе = 4, ориентированный граф и автоматическая генерация матрицы смежности

```

Введите количество вершин в графе: 3
Выберите тип графа:
1. Ориентированный
2. Неориентированный
2
Выберите способ создания матрицы графа:
1. Сгенерировать случайным образом
2. Ввести самостоятельно
2
Введите матрицу графа (используйте -1 для INF):
Введите ребро (0, 0): 1
Введите ребро (0, 1): 2
Введите ребро (0, 2): 19
Введите ребро (1, 0): 4
Введите ребро (1, 1): 3
Введите ребро (1, 2): 7
Введите ребро (2, 0): 37
Введите ребро (2, 1): 33
Введите ребро (2, 2): 2
Матрица графа:
    1    2    19
    2    3    7
    19    7    2
Матрица кратчайших путей:
    1    2    9
    2    3    7
    9    7    2

```

Рисунок 15 – Тестирование при вводе количества вершин в графе = 3, неориентированным графом и ручным вводом матрицы смежности

```

Матрица кратчайших путей:
    1    2    9
    2    3    7
    9    7    2
Введите имя файла для сохранения матрицы: File
Матрица успешно сохранена в файл File.
Выберите действие:
1. Ввести количество вершин и создать матрицу графа
2. Загрузить матрицу графа из файла
3. Завершить программу
2
Введите имя файла для загрузки: File
Введите количество вершин в графе: 3
Матрица успешно загружена из файла File.
Матрица графа:
    1    2    19
    2    3    7
    19    7    2
Матрица кратчайших путей:
    1    2    9
    2    3    7
    9    7    2

```

Рисунок 16 – Тестирование на правильную запись файла и чтение содержимого из файла

Таблица 1 – Описание поведения программы при тестировании

№	Описание	Предусловие	Тестирование	Ожидаемый результат
1	Работа меню	Программа запущена	Запускаем программу с помощью Visual Studio 2019	Вывод в консоли меню программы
2	Выбор функции	Программа запущена	Вводим номер функции	Переход к выполнению функции
3	Определение числа вершин графа	Программа запущена	Ввод числа графа с клавиатуры	Переход к выполнению функции
4	Создание неориентированного графа	Ввод 1 в меню	Ввод способа создания матрицы	Создание неориентированного графа
5	Создание ориентированного графа	Ввод 2 в меню	Ввод способа создания матрицы	Создание ориентированного графа
6	Создание матрицы случайным образом	Ввод 1 в меню	Ввод генерации матрицы случайным образом	Вывод матрицы на экран
	Создание матрицы путем ввода с клавиатуры	Ввод 2 в меню	Ввод матрицы ручным способом	Вывод матрицы на экран
7	Вывод матрицы на экран	Наличие матрицы в памяти	Ввод наименования сохранения файла	Вывод матрицы на экран
8	Сохранение в файл результата	Создание любым способом матрицы смежности	Ввод названия файла сохранения	Появление в файле сохранения результатов

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

6 Ручной расчёт программы

Проведем ручной расчет алгоритма Флойд-Уоршелла для заданной матрицы графа. Матрица представляет собой граф с 4 вершинами, где каждая вершина соединена с другими прямо или косвенно через рёбра.

1. Инициализация матрицы расстояний:

12	2	14	0
4	14	15	7
18	1	12	13
14	18	4	8

2. Первая итерация (k=0):

- Проверяем все пары вершин через вершину 0.
- Обновляем путь от вершины 1 до вершины 2: $\text{dist}[1][2] = \text{dist}[1][0] + \text{dist}[0][2] = 4 + 2 = 6$ (не изменяется, так как уже равно 14).
- Обновляем путь от вершины 1 до вершины 3: $\text{dist}[1][3] = \text{dist}[1][0] + \text{dist}[0][3] = 4 + 14 = 18$ (не изменяется, так как уже равно 15).
- Обновляем путь от вершины 2 до вершины 1: $\text{dist}[2][1] = \text{dist}[2][0] + \text{dist}[0][1] = 18 + 12 = 30$ (не изменяется, так как уже равно 1).
- Обновляем путь от вершины 2 до вершины 3: $\text{dist}[2][3] = \text{dist}[2][0] + \text{dist}[0][3] = 18 + 14 = 32$ (не изменяется, так как уже равно 12).
- Обновляем путь от вершины 3 до вершины 1: $\text{dist}[3][1] = \text{dist}[3][0] + \text{dist}[0][1] = 14 + 12 = 26$ (не изменяется, так как уже равно 18).
- Обновляем путь от вершины 3 до вершины 2: $\text{dist}[3][2] = \text{dist}[3][0] + \text{dist}[0][2] = 14 + 2 = 16$ (не изменяется, так как уже равно 4).

3. Вторая итерация (k=1):

- Проверяем все пары вершин через вершину 1.
- Обновляем путь от вершины 0 до вершины 2: $\text{dist}[0][2] = \text{dist}[0][1] + \text{dist}[1][2] = 12 + 14 = 26$ (не изменяется, так как уже равно 2).

- Обновляем путь от вершины 0 до вершины 3: $\text{dist}[0][3] = \text{dist}[0][1] + \text{dist}[1][3] = 12 + 15 = 27$ (не изменяется, так как уже равно 14).
- Обновляем путь от вершины 2 до вершины 0: $\text{dist}[2][0] = \text{dist}[2][1] + \text{dist}[1][0] = 1 + 12 = 13$ (не изменяется, так как уже равно 18).
- Обновляем путь от вершины 2 до вершины 3: $\text{dist}[2][3] = \text{dist}[2][1] + \text{dist}[1][3] = 1 + 15 = 16$ (не изменяется, так как уже равно 12).
- Обновляем путь от вершины 3 до вершины 0: $\text{dist}[3][0] = \text{dist}[3][1] + \text{dist}[1][0] = 18 + 12 = 30$ (не изменяется, так как уже равно 14).
- Обновляем путь от вершины 3 до вершины 2: $\text{dist}[3][2] = \text{dist}[3][1] + \text{dist}[1][2] = 18 + 14 = 32$ (не изменяется, так как уже равно 4).

4. Третья итерация (k=2):

- Проверяем все пары вершин через вершину 2.
- Обновляем путь от вершины 0 до вершины 1: $\text{dist}[0][1] = \text{dist}[0][2] + \text{dist}[2][1] = 2 + 1 = 3$ (не изменяется, так как уже равно 12).
- Обновляем путь от вершины 0 до вершины 3: $\text{dist}[0][3] = \text{dist}[0][2] + \text{dist}[2][3] = 2 + 12 = 14$ (не изменяется, так как уже равно 14).
- Обновляем путь от вершины 1 до вершины 0: $\text{dist}[1][0] = \text{dist}[1][2] + \text{dist}[2][0] = 14 + 18 = 32$ (не изменяется, так как уже равно 4).
- Обновляем путь от вершины 1 до вершины 3: $\text{dist}[1][3] = \text{dist}[1][2] + \text{dist}[2][3] = 14 + 12 = 26$ (не изменяется, так как уже равно 15).
- Обновляем путь от вершины 3 до вершины 0: $\text{dist}[3][0] = \text{dist}[3][2] + \text{dist}[2][0] = 4 + 18 = 22$ (не изменяется, так как уже равно 14).
- Обновляем путь от вершины 3 до вершины 1: $\text{dist}[3][1] = \text{dist}[3][2] + \text{dist}[2][1] = 4 + 1 = 5$ (не изменяется, так как уже равно 18).

5. Четвертая итерация (k=3):

- Проверяем все пары вершин через вершину 3.
- Обновляем путь от вершины 0 до вершины 1: $\text{dist}[0][1] = \text{dist}[0][3] + \text{dist}[3][1] = 14 + 18 = 32$ (не изменяется, так как уже равно 12).

- Обновляем путь от вершины 0 до вершины 2: $\text{dist}[0][2] = \text{dist}[0][3] + \text{dist}[3][2] = 14 + 4 = 18$ (не изменяется, так как уже равно 2).
- Обновляем путь от вершины 1 до вершины 0: $\text{dist}[1][0] = \text{dist}[1][3] + \text{dist}[3][0] = 15 + 14 = 29$ (не изменяется, так как уже равно 4).
- Обновляем путь от вершины 1 до вершины 2: $\text{dist}[1][2] = \text{dist}[1][3] + \text{dist}[3][2] = 15 + 4 = 19$ (не изменяется, так как уже равно 14).
- Обновляем путь от вершины 2 до вершины 0: $\text{dist}[2][0] = \text{dist}[2][3] + \text{dist}[3][0] = 12 + 14 = 26$ (не изменяется, так как уже равно 18).
- Обновляем путь от вершины 2 до вершины 1: $\text{dist}[2][1] = \text{dist}[2][3] + \text{dist}[3][1] = 12 + 18 = 30$ (не изменяется, так как уже равно 1).

Итоговая матрица расстояний:

6	2	4	0
4	6	8	4
5	1	9	5
9	5	4	8

Таким образом можно сделать вывод, что программа работает верно.

```

Введите количество вершин в графе: 4
Выберите способ создания матрицы графа:
1. Сгенерировать случайным образом
2. Ввести самостоятельно
1
Матрица графа:
    12      2      14      0
    4       14     15      7
    18      1     12     13
    14     18      4      8
Матрица кратчайших путей:
    6       2       4       0
    4       6       8       4
    5       1       9       5
    9       5       4       8

```

Рисунок 17 – Тестирование работы программы

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Флойда для поиска кратчайших путей, в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, освоен принцип работы алгоритма Флойда, чтение и запись содержимого файла в различных видах. Улучшены навыки владения языком программирования Си.

Недостатком разработанной программы является простой графический интерфейс, и его перспектива в развитии как программы, причиной того является достигнутая цель в рамках разработки курсового проекта.

Программа имеет небольшой, но достаточный интерфейс для использования функциональных возможностей от генерации матрицы смежности вплоть до записи и чтения файла и его содержимого.

Список литературы

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с.
2. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978.
3. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006.
4. Уилсон Р. Введение в теорию графов. Пер. с англ. 1977. 208 с.
5. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.
6. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.
7. Скиена Дж., Ревилла Р., Шерифф Б. Алгоритмы: Фундаментальные принципы компьютерных наук. - М.: Вильямс, 2005. - 1024 с.
8. Седжвик Р. Алгоритмы на C++. Часть 5: Графы и строки. - М.: Вильямс, 2011. - 992 с.
9. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. - М.: Вильямс, 2000. - 496 с.
10. Паппас Т. Анализ алгоритмов и доказательство корректности. - М.: Бином, 2012. - 416 с.

Приложение А.

Листинг программы.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>
#include <queue>
#include <vector>
#include <limits.h>
#include <ctime>
#include <limits.h>

void printMatrisa(int** dist, int V);
void Floyd(int** graph, int V);
void generateRandomMatrisa(int** graph, int V, int directed);
void inputMatrisa(int** graph, int V, int directed);
void saveFile(int** graph, int V, const char* filename);
void loadFile(int** graph, int V, const char* filename);

int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));
    int V, choice, exit = 0, directed;
    int** graph = NULL;

    while (!exit) {
        printf("Выберите действие:\n");
        printf("1. Ввести количество вершин и создать матрицу графа\n");
        printf("2. Загрузить матрицу графа из файла\n");
        printf("3. Завершить программу\n");
        scanf("%d", &choice);
        if (choice == 1) {
            printf("Введите количество вершин в графе: ");
            scanf("%d", &V);
            graph = (int**)malloc(V * sizeof(int*));
            for (int i = 0; i < V; i++) {
                graph[i] = (int*)malloc(V * sizeof(int));
            }
            printf("Выберите тип графа:\n");
            printf("1. Ориентированный\n");
            printf("2. Неориентированный\n");
            scanf("%d", &directed);
            directed = (directed == 1) ? 1 : 0;
            printf("Выберите способ создания матрицы графа:\n");
            printf("1. Сгенерировать случайным образом\n");
            printf("2. Ввести самостоятельно\n");
            scanf("%d", &choice);
            if (choice == 1)
                generateRandomMatrisa(graph, V, directed);
            else if (choice == 2)
                inputMatrisa(graph, V, directed);
            else {
                printf("Неверный выбор. Попробуйте снова.\n");
            }
        }
    }
}
```

```

        continue;
    }
}
else if (choice == 2) {
    char filename[100];
    printf("Введите имя файла для загрузки: ");
    scanf("%s", filename);
    printf("Введите количество вершин в графе: ");
    scanf("%d", &V);
    graph = (int**)malloc(V * sizeof(int*));
    for (int i = 0; i < V; i++) {
        graph[i] = (int*)malloc(V * sizeof(int));
    }
    loadFile(graph, V, filename);
    printf("Матрица графа:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (graph[i][j] == INT_MAX)
                printf("%7s", "INF");
            else
                printf("%7d", graph[i][j]);
        }
        printf("\n");
    }
    Floyd(graph, V);
    for (int i = 0; i < V; i++) {
        free(graph[i]);
    }
    free(graph);
    graph = NULL;
    continue;
}
else if (choice == 3) {
    exit = 1;
}
else {
    printf("Неверный выбор. Попробуйте снова.\n");
    continue;
}
printf("Матрица графа:\n");
for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
        if (graph[i][j] == INT_MAX)
            printf("%7s", "INF");
        else
            printf("%7d", graph[i][j]);
    }
    printf("\n");
}
if (graph != NULL) {
    Floyd(graph, V);
    char saveFilename[100];
    printf("Введите имя файла для сохранения матрицы: ");
    scanf("%s", saveFilename);
    saveFile(graph, V, saveFilename);
    // Освобождение памяти
    for (int i = 0; i < V; i++) {
        free(graph[i]);
    }
    free(graph);
}

```

```

        graph = NULL;
    }
}
return 0;
}

void generateRandomMatriza(int** graph, int V, int directed) {
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (rand() % 100 < 20) {
                graph[i][j] = INT_MAX;
            }
            else {
                graph[i][j] = rand() % 20;
            }
            if (!directed) {
                graph[j][i] = graph[i][j];
            }
        }
    }
}

void inputMatriza(int** graph, int V, int directed) {
    printf("Введите матрицу графа (используйте -1 для INF):\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            printf("Введите ребро (%d, %d): ", i, j);
            int value;
            scanf("%d", &value);
            if (value == -1) {
                graph[i][j] = INT_MAX;
            }
            else {
                graph[i][j] = value;
            }
        }
    }
    if (!directed) {
        for (int i = 0; i < V; i++) {
            for (int j = i + 1; j < V; j++) {
                if (graph[i][j] != INT_MAX) {
                    graph[j][i] = graph[i][j];
                }
            }
        }
    }
}

void saveFile(int** graph, int V, const char* filename) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("Не удалось открыть файл для записи.\n");
        return;
    }
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (graph[i][j] == INT_MAX) {
                fprintf(file, "-1 ");
            }
            else {
                fprintf(file, "%d ", graph[i][j]);
            }
        }
    }
}

```

```

        fprintf(file, "\n");
    }
    fclose(file);
    printf("Матрица успешно сохранена в файл %s.\n", filename);
}

void loadFile(int** graph, int V, const char* filename) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("Не удалось открыть файл для чтения.\n");
        return;
    }
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            int value;
            fscanf(file, "%d", &value);
            if (value == -1) {
                graph[i][j] = INT_MAX;
            }
            else {
                graph[i][j] = value;
            }
        }
    }
    fclose(file);
    printf("Матрица успешно загружена из файла %s.\n", filename);
}

void Floyd(int** graph, int V) {
    int** dist = (int**)malloc(V * sizeof(int*));
    for (int i = 0; i < V; i++)
        dist[i] = (int*)malloc(V * sizeof(int));
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            dist[i][j] = graph[i][j];
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][k] +
dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    printMatrisa(dist, V);
    for (int i = 0; i < V; i++) {
        free(dist[i]);
    }
    free(dist);
}

void printMatrisa(int** dist, int V) {
    printf("Матрица кратчайших путей:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INT_MAX)
                printf("%7s", "INF");
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

```