Министерство науки 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 24ВВВ3
Комиссаров А.Г.
Принял:
к.т.н. Юрова О.В.

25.12.25
хорошо

Пенза 2025

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ _____

«___»_____20__

ЗАДАНИЕ

на курсовое проектирование по курсу

Логика и основы алгоритмизации виктимных систем

Студенту Комиссарову Андрею Геннадьевичу Группа 21ВВВ3

Тема_проекта Реализация алгоритма Флойда

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 постановки задачи
4) Пример ручного расчета задачи
5) Описание самой задачи
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-схемы

3. Экспериментальная часть

Тестирование программы
Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

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

Дата выдачи задания "12" сентября 2025

Дата защиты проекта "__" _____

Руководитель Юрова Ольга Викторовна _____

Задание получил "12" сентября 2025 г.

Студент Коломеарев Андрей Геннадьевич _____

Содержание

Реферат

Отчет 45 страниц, 14 рисунков, 2 таблицы, 1 приложение

АЛГОРИТМ ФЛОЙДА, 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, использую алгоритм Флойда.

В работе рассматривается алгоритм Флойда, 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, также известный как алгоритм Флойда-Уоршелла, представляет собой метод нахождения кратчайших путей между всеми парами вершин в графе. Это алгоритм, который применяется для вычисления кратчайших расстояний между каждой парой вершин в графе, используя все возможные промежуточные вершины. В отличие от других алгоритмов поиска кратчайших путей, таких как алгоритм Дейкстры, который решает задачу для одной вершины, алгоритм Флойда решает задачу для всех пар вершин в графе.

Алгоритм Флойда работает по принципу динамического программирования, где на каждом шаге обновляется матрица кратчайших путей. Изначально создается матрица расстояний, в которой элементы представляют собой веса рёбер между вершинами. Если рёбер нет, то устанавливается значение "бесконечность". Диагональные элементы матрицы, отражающие расстояния от вершины до самой себя, равны нулю. Затем алгоритм выполняет несколько итераций, используя каждую вершину как промежуточную. Если через промежуточную вершину путь становится короче, чем прямой путь между двумя вершинами, то обновляется значение в матрице.

Процесс работы алгоритма можно представить следующим образом. Пусть у нас есть граф с несколькими вершинами, связанными рёбрами с определёнными весами. Изначально мы создаём матрицу расстояний, где каждая ячейка представляет расстояние между двумя вершинами, либо бесконечность, если прямого пути нет. После этого на каждом шаге, для каждой вершины, рассматривается возможность использования её как промежуточной для улучшения существующих путей. Таким образом, алгоритм поочередно обновляет матрицу кратчайших путей.

В качестве среду разработки мною была выбрана среда Microsoft Visual Studio 2022, язык программирования – C. Целью данной курсовой работы

является разработка программы на языке С.Именно с его помощью в данном курсовом проекте реализуется алгоритм Флойда, осуществляющий поиск кратчайших путей между всеми парами вершин в графе.

1 Постановка задачи

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

Исходный 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, тип графа и то, как заполняется сама матрица. После обработки этих данных на экран должна выводиться матрица смежности графа, вид графа и все компоненты сильной связности графа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройства ввода – клавиатура и мышь.

2 Теоретическая часть задания

Граф (рисунок 1) представляет собой ориентированный взвешенный граф с семью вершинами (A, B, C, D, E, F, G) 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.

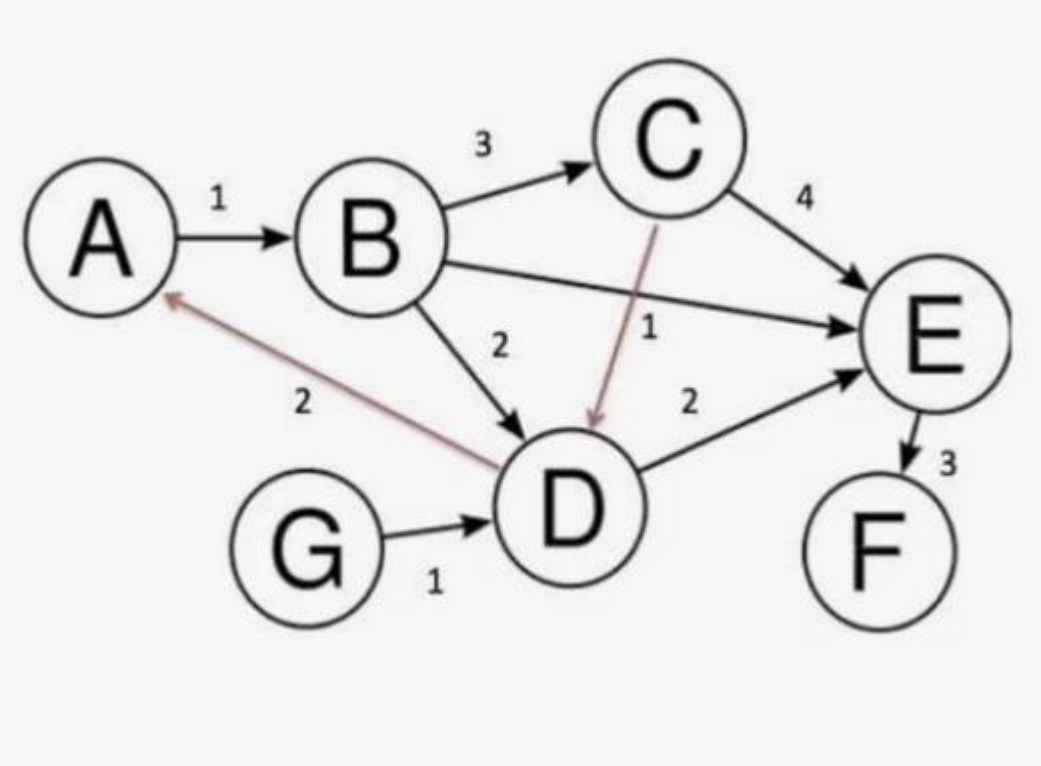

Рисунок 1 – Пример орграфа

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.

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.

Алгоритм Флойда 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 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 проходит через все вершины поочередно, пытаясь использовать каждую вершину как промежуточную между другими вершинами. Если путь от вершины i до вершины j через промежуточную вершину k короче, чем прямой путь от i до j, то обновляется матрица кратчайших расстояний.

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, несвязанный граф — это граф, в котором не существует пути между всеми парами вершин. То есть, если для каких-то двух вершин в графе невозможно пройти, используя рёбра этого графа, то такой граф называется несвязанным. В несвязанном графе могут быть компоненты, которые являются связными подграфами, но между ними не существует рёбер.

3 Описание алгоритма программы

Для программной реализации понадобится объявить функции такие как inputGraph, floydWarshall, randomGraph, saveResults, printGraph, printDistances, printMenu().

Основной цикл программы:

1. **Ввод размера графа**: Программа запрашивает количество вершин и выделяет память для матрицы смежности.
2. **Выбор типа графа**: Пользователь выбирает, будет ли граф ориентированным или неориентированным.
3. **Заполнение графа**: Пользователь может заполнить граф вручную или случайным образом.
4. **Выполнение алгоритма Флойда**: Программа применяет алгоритм Флойда-Уоршелла для нахождения кратчайших путей между всеми вершинами.
5. **Вывод и сохранение результатов**: Матрица кратчайших путей выводится на экран и сохраняется в файл.
6. **Освобождение памяти**: Программа очищает выделенную память.
7. **Повтор или завершение**: Пользователь решает, хочет ли повторить процесс или завершить программу.

Для хранения дистанции, то есть динамическое выделение памяти для матрицы расстояний используется переменная malloc. После инициализации графа (матрицы смежности), создаётся ещё одна матрица для хранения кратчайших расстояний между всеми вершинами. Эта матрица создаётся с помощью функции malloc. Для каждой строки (вектора расстояний от одной вершины ко всем остальным) выделяется память с помощью malloc(size * sizeof(int)).

Матрица dist инициализируется значениями из матрицы графа, то есть, сначала в неё копируются значения из исходной матрицы смежности графа (где значения равны весам рёбер или INF, если рёбер нет).

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.

Определение количества вершин в графе: Количество вершин в графе определяется пользователем при запуске программы. После выбора пункта меню для задания размера графа, программа запрашивает у пользователя число, которое и задаёт количество вершин в графе.

Определение типа графа: После задания размера графа пользователю предлагается выбрать тип графа. В меню он может выбрать один из двух вариантов:

1) Ориентированный граф, где рёбра имеют направление.
2) Неориентированный граф, где рёбра симметричны (если существует связь между вершинами A и B, то она автоматически существует и между B и A).В коде представлены следующие функции:

1. printMenu() — отображает меню программы.
2. inputGraph(int** graph, int size, int isDirected) — заполняет граф вручную, принимая матрицу смежности от пользователя.
3. randomGraph(int** graph, int size, int isDirected) — генерирует случайный граф с заданным размером и направленностью.
4. floydWarshall(int** graph, int size) — реализует алгоритм Флойда для нахождения кратчайших путей в графе.

5. saveResults(int** dist, int size, const char* filename) — сохраняет результаты работы алгоритма Флойда в файл.

6. printGraph(int** graph, int size) — выводит граф в виде матрицы смежности.

7. printDistances(int** dist, int size) — выводит матрицу кратчайших расстояний.

Эти функции отвечают за различные части работы программы: создание и обработку графа, выполнение алгоритма Флойда, вывод и сохранение данных.

Псевдокод для программы

Функция main

1. Установить локаль для поддержки русского языка (`setlocale`).

2. Установить флаг `repeat = 1` для выполнения программы в цикле.

3. Пока `repeat == 1`:

   3.1. Объявить переменные:

   - `size = 0` для хранения размера графа.

   - Указатель на матрицу `graph = NULL`.

   - Переменные-флаги:

     - `sizeSet = 0` (размер графа установлен),

     - `graphFilled = 0` (граф заполнен),

     - `floydRun = 0` (алгоритм Флойда выполнен).

   - `isDirected = -1` для хранения типа графа (-1 означает невыбранный тип).

   3.2. Пока `floydRun == 0`:

3.2.1. Вызвать функцию `printMenu()` для отображения пунктов меню.

3.2.2. Вывести: "Ваш выбор: ".

3.2.3. Считать ввод пользователя (`choice`).

- Если ввод некорректен, очистить буфер ввода и попросить повторить ввод.

3.2.4. Выполнить действие в зависимости от значения `choice`:

-Case 1:Установить размер графа:

1. Вывести: "Введите размер графа: ".

2. Считать значение `size`.

- Если `size <= 0`, вывести сообщение об ошибке и попросить повторить ввод.

3. Если `graph` уже существует:

- Освободить выделенную ранее память.

4. Выделить память для матрицы `graph` размером `size x size`.

5. Заполнить матрицу значениями:

- Если `i == j`, установить 0.

- Иначе установить `INF`.

6. Установить `sizeSet = 1` (размер графа задан).

- Case 2: Выбрать тип графа:

1. Если `sizeSet == 0`, вывести: "Сначала задайте размер графа!".

2. Иначе предложить выбор:

- 1 — Ориентированный.

- 2 — Неориентированный.

3. Считать ввод и обновить переменную `isDirected`.

- Case 3: Заполнить граф вручную:

1. Если `sizeSet == 0` или `isDirected == -1`, вывести сообщение о необходимости задать размер и тип графа.

2. Иначе вызвать функцию `inputGraph(graph, size, isDirected)`.

3. Установить `graphFilled = 1` (граф заполнен).

- Case 4: Заполнить граф случайными значениями:

1. Если `sizeSet == 0` или `isDirected == -1`, вывести сообщение о необходимости задать размер и тип графа.

2. Иначе вызвать функцию `randomGraph(graph, size, isDirected)`.

3. Установить `graphFilled = 1` (граф заполнен).

- Case 5: Выполнить алгоритм Флойда:

1. Если `sizeSet == 0`, `graphFilled == 0` или `isDirected == -1`, вывести сообщение о необходимости задать размер и заполнить граф.

2. Иначе вызвать функцию `floydWarshall(graph, size)`.

3. Установить `floydRun = 1` (алгоритм выполнен).

- Case 6: Завершить программу:

1. Установить `repeat = 0` и `floydRun = 1` (выход из обоих циклов).

3.2.5. Освободить память для графа `graph` (если она была выделена).

3.3. Если `repeat == 1`:

- Спросить пользователя, повторить ли выполнение программы.

- Если ответ "нет", установить `repeat = 0`.

4. Завершить выполнение программы.

Функция printMenu

1. Вывести на экран пункты меню:

- "1. Задать размер графа."

- "2. Выбор типа графа."

- "3. Заполнение графа вручную."

- "4. Заполнение графа случайными значениями."

- "5. Выполнить алгоритм Флойда."

- "6. Выход."

Функция inputGraph

1. Вывести сообщение: "Введите матрицу смежности графа (используйте INF для отсутствия связи):".

2. Для каждой вершины `i` от 0 до `size - 1`:

2.1. Для каждой вершины `j` от 0 до `size - 1`:

2.1.1. Считать значение `graph[i][j]`.

- Если ввод некорректный, вывести сообщение об ошибке и повторить ввод.

2.1.2. Если `i != j` и `graph[i][j] == 0`, установить `graph[i][j] = INF`.

3. Если граф неориентированный (`isDirected == 0`):

3.1. Для каждой вершины `i` от 0 до `size - 1`:

3.1.1. Для каждой вершины `j` от `i + 1` до `size - 1`:

- Установить `graph[j][i] = graph[i][j]`.

Функция randomGraph

1. Установить генератор случайных чисел (`srand`).

2. Для каждой вершины `i` от 0 до `size - 1`:

   2.1. Для каждой вершины `j` от 0 до `size - 1`:

      2.1.1. Если `i == j`, установить `graph[i][j] = 0`.

      2.1.2. Иначе случайно выбрать значение (70% вероятности для числа, 30% для `INF`).

3. Если граф неориентированный (`isDirected == 0`):

   3.1. Для каждой вершины `i` от 0 до `size - 1`:

      3.1.1. Для каждой вершины `j` от `i + 1` до `size - 1`:

         - Установить `graph[j][i] = graph[i][j]`.

Функция floydWarshall

1. Создать копию матрицы `graph` в `dist`.

2. Вывести сообщение: "Исходная матрица графа:".

3. Вызвать `printGraph(dist, size)`.

4. Для каждой промежуточной вершины `k` от 0 до `size - 1`:

   4.1. Для каждой вершины `i` от 0 до `size - 1`:

      4.1.1. Для каждой вершины `j` от 0 до `size - 1`:

         - Если `dist[i][k] != INF` и `dist[k][j] != INF`:

            - Если `dist[i][k] + dist[k][j] < dist[i][j]`, обновить `dist[i][j]`.

5. Вывести сообщение: "Матрица кратчайших путей:".

6. Вызвать `printDistances(dist, size)`.

7. Вызвать `saveResults(dist, size, "results")`.

8. Освободить память, выделенную для `dist`.

Функция printGraph и printDistances

1. Для каждой строки матрицы:

   1.1. Для каждого элемента:

      - Если элемент равен `INF`, вывести "INF".

      - Иначе вывести значение элемента.

   1.2. Перейти к следующей строке.

Функция saveResults

1. Открыть файл с именем `filename` для записи.

2. Если файл не удалось открыть, вывести сообщение об ошибке и выйти.

3. Записать в файл строку: "Матрица кратчайших расстояний:".

4. Для каждой строки матрицы `dist`:

   4.1. Для каждого элемента:

      - Если элемент равен `INF`, записать "INF".

      - Иначе записать значение элемента.

5. Закрыть файл.

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 функций: printMenu(), inputGraph(), randomGraph(), floydWarshall(), saveResults(), printGra ph(), printDistances()

Программа начинается с отображения главного меню, в котором пользователь может выбрать действия, связанные с графом (рисунок 2). Сначала задаётся размер графа (рисунок 3), затем выбирается его тип — ориентированный или неориентированный (рисунок 4). После этого пользователь может заполнить граф вручную (рисунок 5) или случайным образом (рисунок 6). Когда граф подготовлен, выполняется алгоритм Флойда для нахождения кратчайших путей между вершинами (рисунок 8). Результаты сохраняются в файл и выводятся на экран (рисунок 7). Программа позволяет повторить действия или завершить выполнение. Динамическое выделение памяти используется для хранения графа и матрицы расстояний, что позволяет работать с графами разных размеров. Вид кода, который осуществляет данную реализацию программы:

```
int main() {
    setlocale(LC_ALL, "Russian");
    int repeat = 1; // Флаг для повторения программы

    while (repeat) {
        int size = 0;
        int** graph = NULL;
        int choice;
        char sizeSet = 0, graphFilled = 0, floydRun = 0;
        int isDirected = -1;  // Флаг для ориентации графа (0 - неориентированный, 1 -
ориентированный)
```

```
while (!floydRun) {
    printMenu();
    printf("\nВаш выбор: ");
    if (scanf_s("%d", &choice) != 1) {
        printf("Ошибка ввода! Попробуйте снова.\n");
        while (getchar() != '\n'); // Очистка буфера ввода
        continue;
    }

    switch (choice) {
    case 1: // Задать размер графа
        printf("Введите размер графа: ");
        if (scanf_s("%d", &size) != 1 || size <= 0) {
            printf("Ошибка ввода! Размер графа должен быть положительным числом.\n");
            while (getchar() != '\n'); // Очистка буфера ввода
            break;
        }

        // Освобождаем предыдущую память, если была
        if (graph) {
            for (int i = 0; i < size; i++) {
                free(graph[i]);
            }
            free(graph);
        }

        // Выделяем память для графа
        graph = (int**)malloc(size * sizeof(int*));
        if (!graph) {
            printf("Ошибка выделения памяти!\n");
            exit(1);
        }

        for (int i = 0; i < size; i++) {
            graph[i] = (int*)malloc(size * sizeof(int));
            if (!graph[i]) {
                printf("Ошибка выделения памяти!\n");
                exit(1);
            }
        }

        // Инициализация графа значениями INF
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                graph[i][j] = (i == j) ? 0 : INF;
            }
        }

        printf("Размер графа установлен на %d.\n", size);
        sizeSet = 1; // Устанавливаем флаг
        break;

    case 2: // Выбор типа графа
        if (!sizeSet) {
            printf("Сначала задайте размер графа!\n");
            break;
        }
        printf("Выберите тип графа:\n");
        printf("1. Ориентированный граф\n");
        printf("2. Неориентированный граф\n");
        printf("Ваш выбор: ");
        if (scanf_s("%d", &isDirected) != 1 || (isDirected != 1 && isDirected != 2)) {
```

```
                    printf("Ошибка ввода! Попробуйте снова.\n");
                    break;
                }
                isDirected = (isDirected == 1) ? 1 : 0; // 1 - ориентированный, 0 -
неориентированный
                printf("Тип графа установлен на %s.\n", isDirected ? "ориентированный" :
"неориентированный");
                break;

            case 3: // Заполнение графа вручную
                if (!sizeSet || isDirected == -1) {
                    printf("Сначала задайте размер графа и выберите тип графа!\n");
                    break;
                }
                inputGraph(graph, size, isDirected);
                printf("Граф успешно заполнен вручную.\n");
                graphFilled = 1; // Устанавливаем флаг
                break;

            case 4: // Автоматическое (случайное) заполнение графа
                if (!sizeSet || isDirected == -1) {
                    printf("Сначала задайте размер графа и выберите тип графа!\n");
                    break;
                }
                randomGraph(graph, size, isDirected);
                printf("Граф успешно заполнен случайными значениями.\n");
                graphFilled = 1; // Устанавливаем флаг
                break;

            case 5: // Выполнение алгоритма Флойда
                if (!sizeSet || !graphFilled || isDirected == -1) {
                    printf("Сначала задайте размер графа, выберите тип графа и заполните
его!\n");
                    break;
                }

                floydWarshall(graph, size);
                floydRun = 1; // Устанавливаем флаг, чтобы выйти из вложенного цикла
                break;

            case 6: // Выход
                printf("Программа завершена.\n");
                repeat = 0;
                floydRun = 1; // Чтобы выйти из вложенного цикла
                break;

            default:
                printf("Неверный выбор. Попробуйте снова.\n");
            }
        }
        // Освобождаем память
        if (graph) {
            for (int i = 0; i < size; i++) {
                free(graph[i]);
            }
            free(graph);
        }
        // Функция повторения программы
        if (repeat) {
            printf("\nПовторить?\n1. Да\n2. Нет\nВаш выбор: ");
            if (scanf_s("%d", &repeat) != 1 || (repeat != 1 && repeat != 2)) {
                printf("Ошибка ввода! Программа завершена.\n");
                repeat = 0;
```

```
            }
            if (repeat == 2) {
                printf("Программа завершена.\n");
                repeat = 0;
            }
        }
    }
    return 0;
}
```

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

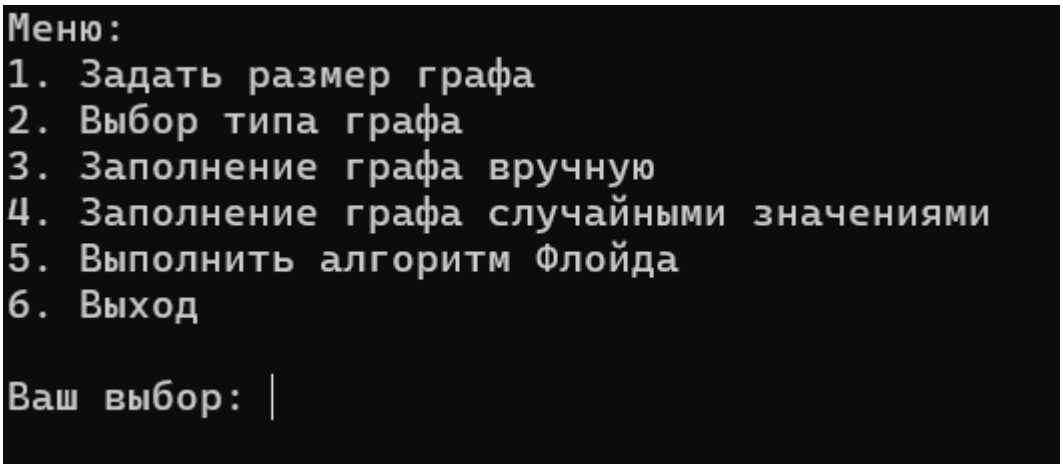

Рисунок 2 – основное меню

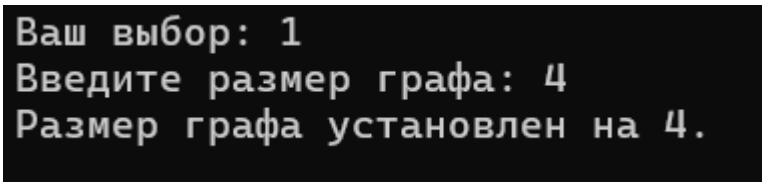

Рисунок 3 – Определение размера графа

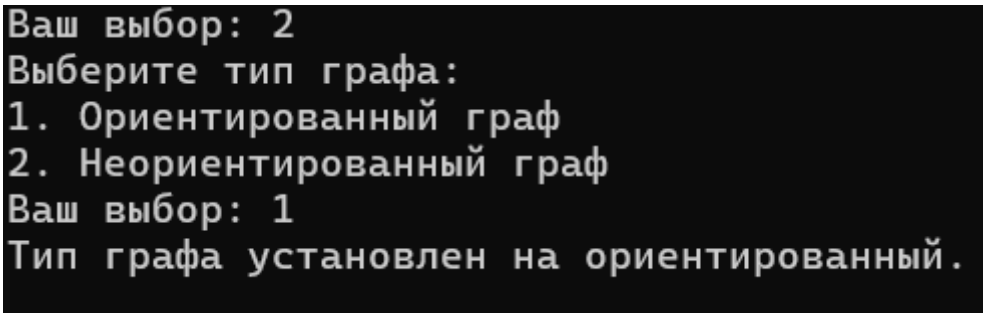

Рисунок 4 – Тип графа

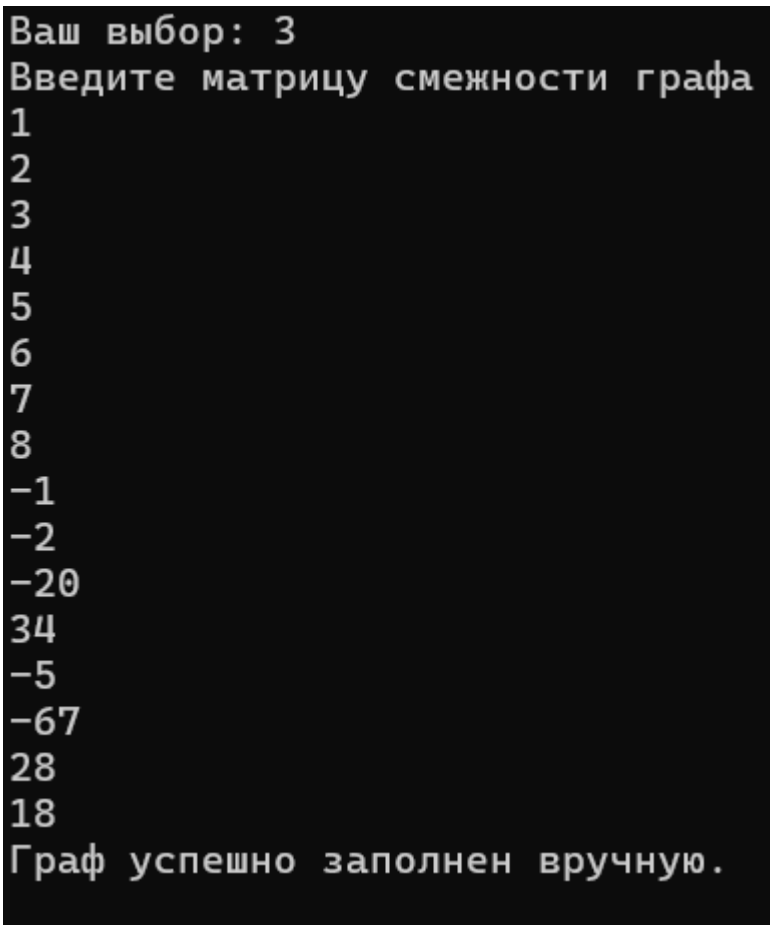

Рисунок 5 – Ввод матрицы с клавиатуры

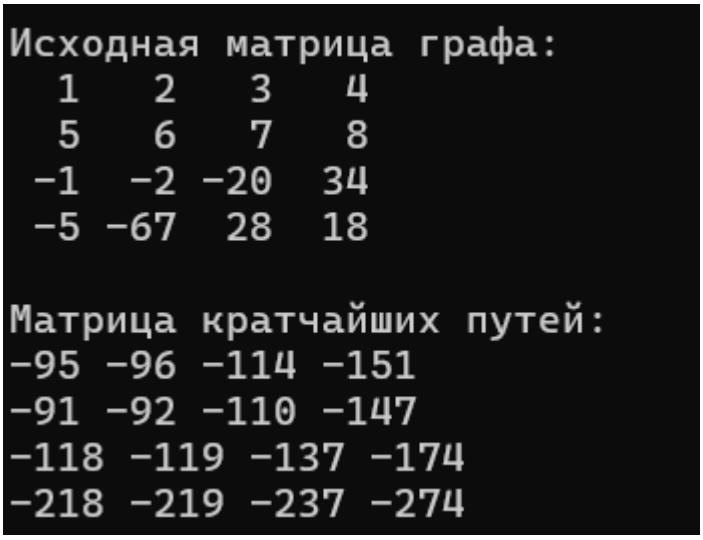

Рисунок 6 – Результат выполнения алгоритма Флойда

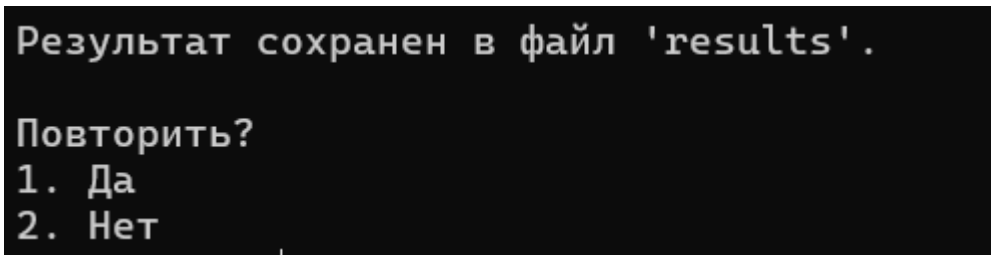

Рисунок 7 – Сохранение результата в файл и выбор повторить ли заново программу

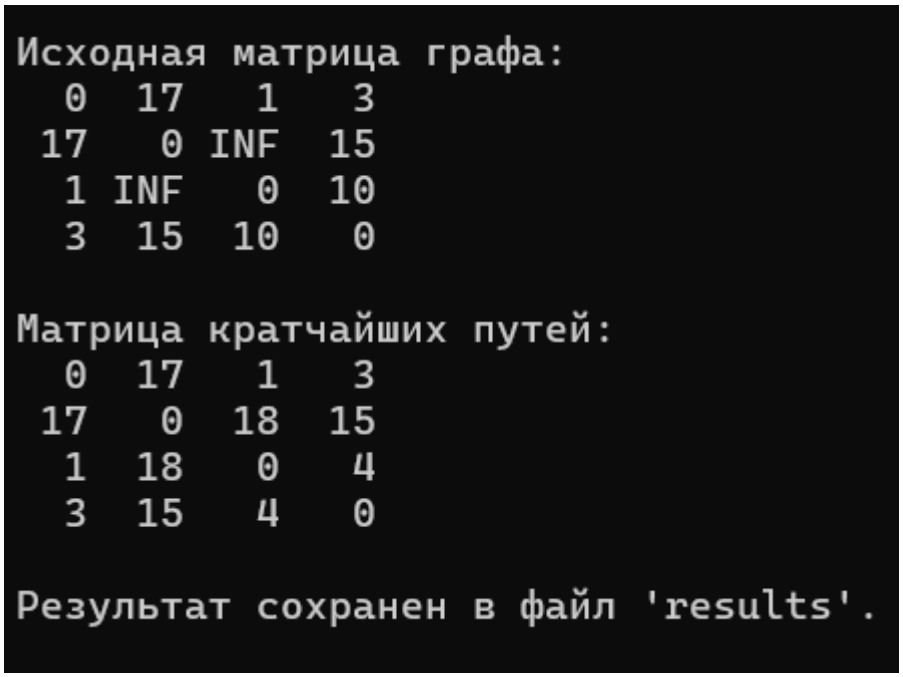

Рисунок 8 – генерация неориентированной матрицы смежности, заполненной случайными значениями

Когда пользователь задаёт размер графа, то выполняется часть функции intMain():

```
case 1: // Задать размер графа
    printf("Введите размер графа: ");
    if (scanf_s("%d", &size) != 1 || size <= 0) {
        printf("Ошибка ввода! Размер графа должен быть положительным числом.\n");
        while (getchar() != '\n'); // Очистка буфера ввода
        break;
    }
    // Освобождаем предыдущую память, если была
    if (graph) {
```

```
        for (int i = 0; i < size; i++) {
            free(graph[i]);
        }
        free(graph);
    }
    // Выделяем память для графа
    graph = (int**)malloc(size * sizeof(int*));
    if (!graph) {
        printf("Ошибка выделения памяти!\n");
        exit(1);
    }
    for (int i = 0; i < size; i++) {
        graph[i] = (int*)malloc(size * sizeof(int));
        if (!graph[i]) {
            printf("Ошибка выделения памяти!\n");
            exit(1);
        }
    }
    // Инициализация графа значениями INF
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            graph[i][j] = (i == j) ? 0 : INF;
        }
    }
    printf("Размер графа установлен на %d.\n", size);
    sizeSet = 1; // Устанавливаем флаг
    break;
```

Когда пользователь задёт вид графа, то выбор ориентированного или неориентированного графа в программе осуществляется через ввод пользователя, который выбирает одну из двух опций: "1" для ориентированного и "2" для неориентированного графа. Программа устанавливает флаг isDirected в 1 для ориентированного графа и в 0 для неориентированного, после чего соответствующим образом обрабатывает связи между вершинами:

```
case 2: // Выбор типа графа
```

```
    if (!sizeSet) {
        printf("Сначала задайте размер графа!\n");
        break;
    }
    printf("Выберите тип графа:\n");
    printf("1. Ориентированный граф\n");
    printf("2. Неориентированный граф\n");
    printf("Ваш выбор: ");
    if (scanf_s("%d", &isDirected) != 1 || (isDirected != 1 && isDirected != 2)) {
        printf("Ошибка ввода! Попробуйте снова.\n");
        break;
    }
    isDirected = (isDirected == 1) ? 1 : 0; // 1 - ориентированный, 0 - неориентированный
    printf("Тип графа установлен на %s.\n", isDirected ? "ориентированный" :
"неориентированный");
```

break;

Если пользователь выбрал четвертый пункт меню, выполняется метод randomGraph, в которой программа генерирует числа случайным образом для заполнения матрицы смежности. Ниже представлено то, как реализуется заполнение матрицы:

```
void randomGraph(int** graph, int size, int isDirected) {
    srand((unsigned int)time(NULL));
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (i == j) {
                graph[i][j] = 0;
            }
            else {
                graph[i][j] = (rand() % 10 < 7) ? rand() % 20 + 1 : INF;
            }
        }
    }
```

```
// Если граф неориентированный, дублируем связи
if (!isDirected) {
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            if (graph[i][j] != INF) {
                graph[j][i] = graph[i][j];
            }
        }
    }
}
```

После того как матрица смежности была сгенерирована случайными образом или пользователь ввел ее вручную в автоматическом порядке выполняется функция floydWarshall, которая 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:

```
case 5: // Выполнение алгоритма Флойда
    if (!sizeSet || !graphFilled || isDirected == -1) {
        printf("Сначала задайте размер графа, выберите тип графа и заполните его!\n");
        break;
    }
    floydWarshall(graph, size);
    floydRun = 1; // Устанавливаем флаг, чтобы выйти из вложенного цикла
    break;
```

Чтобы пользователь имел возможность видеть, что сгенерировала программа или что он ввел с клавиатуры используются функции printGraph(матрица смежности графа) и printDistances(матрица кратчайших путей). Ниже представлен код реализации метода (рисунок 9):

```
void printGraph(int** graph, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (graph[i][j] == INF) {
                printf("INF ");
            } else {
                printf("%3d ", graph[i][j]);
            }
        }
        printf("\n");
    }
}

void printDistances(int** dist, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (dist[i][j] == INF) {
                printf("INF ");
            } else {
                printf("%3d ", dist[i][j]);
            }
        }
        printf("\n");
    }
}
```

```
Исходная матрица графа:
  0  10  13
 17   0  13
 17  19   0

Матрица кратчайших путей:
  0  10  13
 17   0  13
 17  19   0
```

Рисунок 9 – Вывод матриц

Одним из функционалом программы является ввод значений с клавиатуры (Рисунок 10). Это позволяет самостоятельно пользователю определить значения матрицы смежности. Данным метод реализован с помощью inputGraph, код представлен ниже:

```
void inputGraph(int** graph, int size, int isDirected) {
    printf("Введите матрицу смежности графа (используйте %d для обозначения отсутствия связи):\n", INF);
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
```

```
            if (scanf_s("%d", &graph[i][j]) != 1) {
                printf("Ошибка ввода! Повторите ввод.\n");
                while (getchar() != '\n'); // Очистка буфера ввода
                j--;
            }
            if (i != j && graph[i][j] == 0) {
                graph[i][j] = INF; // Отсутствие пути
            }
        }
    }
    // Если граф неориентированный, дублируем связи
    if (!isDirected) {
        for (int i = 0; i < size; i++) {
            for (int j = i + 1; j < size; j++) {
                if (graph[i][j] != INF) {
                    graph[j][i] = graph[i][j];
                }
            }
        }
    }
}
```

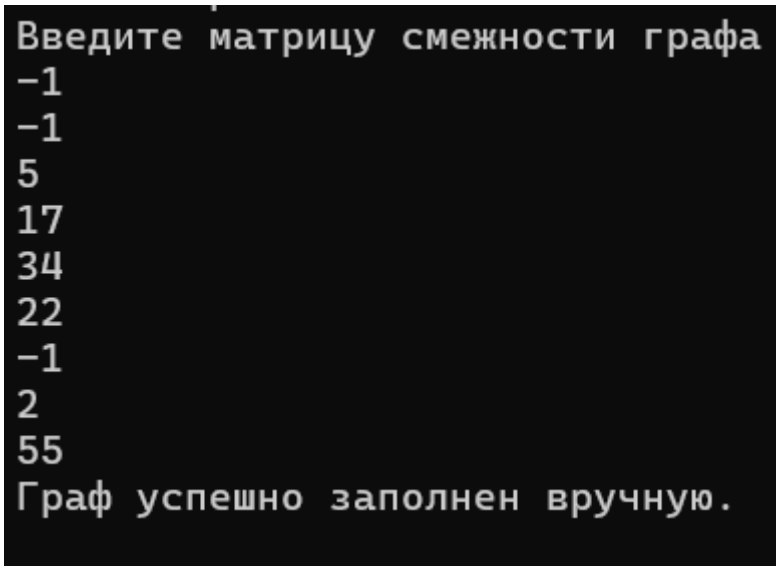

Рисунок 10 – Ввод матрицы с клавиатуры

Для того чтобы сохранить значения матрицы смежности, которые были сгенерированы случайно или пользователь их ввел вручную с клавиатуры, используется метод saveResults, код представлен ниже (рисунок 11):

```
void saveResults(int** dist, int size, const char* filename) {
    FILE* file;
    if (fopen_s(&file, filename, "w") != 0) {
        printf("Ошибка открытия файла для записи.\n");
        return;
    }

    fprintf(file, "Матрица кратчайших расстояний:\n");
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (dist[i][j] == INF) {
                fprintf(file, "INF ");
            }
            else {
                fprintf(file, "%3d ", dist[i][j]);
            }
        }
        fprintf(file, "\n");
    }

    fclose(file);
        }
```

```
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 11 – Сохранение результата в файл

5 Тестирование

Таблица 1 – Описание поведения программы при тестировании

№	Описание	Предусловие	Тестирование	Ожидаемый результат
1	Работа меню	Программа запущена	Запускаем программу с помощью Visual Studio 2022	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 ориентированного графа	Ввод 1 в меню	Ввод способа создания матрицы	Создание ориентированного графа
5	Создание неориентированного графа	Ввод 2 в меню	Ввод способа создания матрицы	Создание неориентированного графа
6	Создание матрицы случайным образом	Ввод 4 в меню	Ввод генерации матрицы случайным образом	Вывод матрицы на экран
7	Создание матрицы путем ввода с клавиатуры	Ввод 3 в меню	Ввод матрицы ручным способом	Вывод матрицы на экран
8	Вывод матрицы на экран	Наличие матрицы в памяти	Ввод 5 в меню	Вывод матрицы на экран
9	Сохранение в файл результата	Создание любым способом матрицы смежности	Автоматическое сохранение результатов в файл	Появление в файле сохранения результатов

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

Среда разработки Microsoft Visual Studio 2022 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 (рисунок 12, рисунок 13).

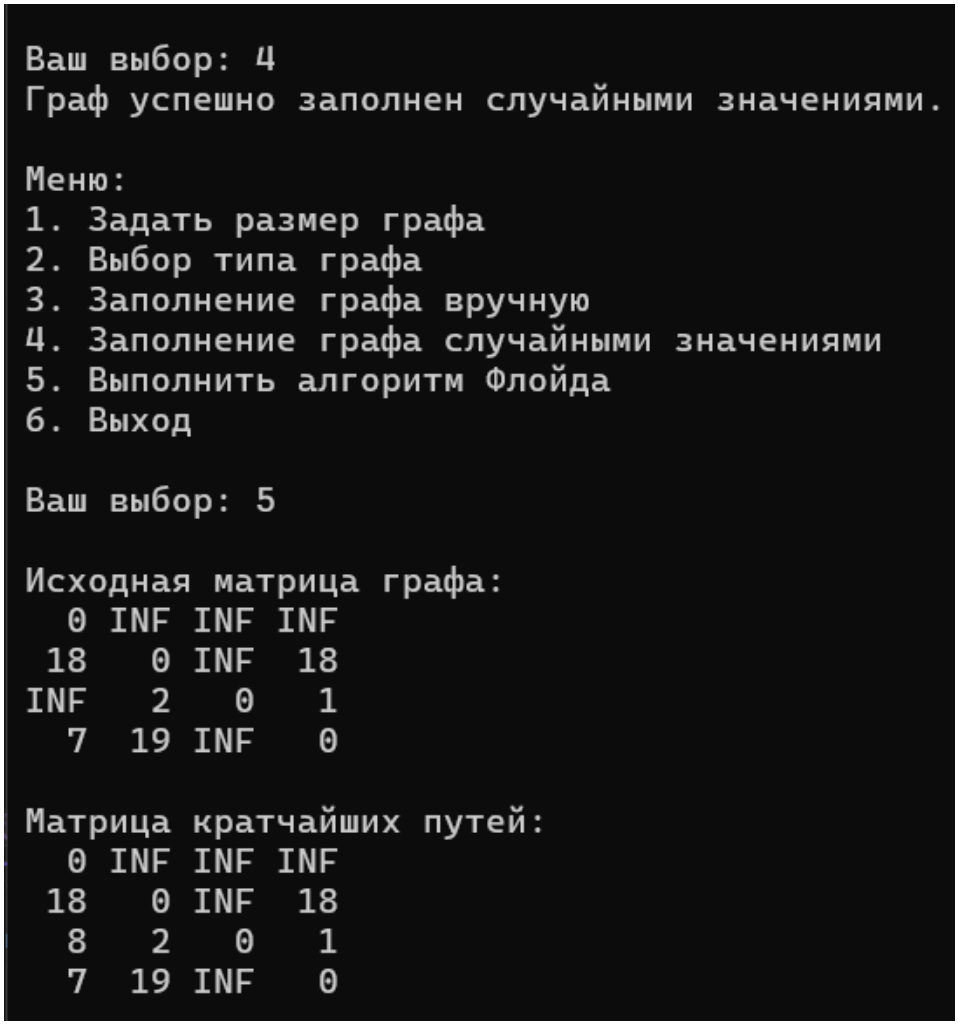

Рисунок 12 – Тестирование при вводе количество вершин в графе = 3, ориентированный граф и автоматическая генерация матрицы смежности

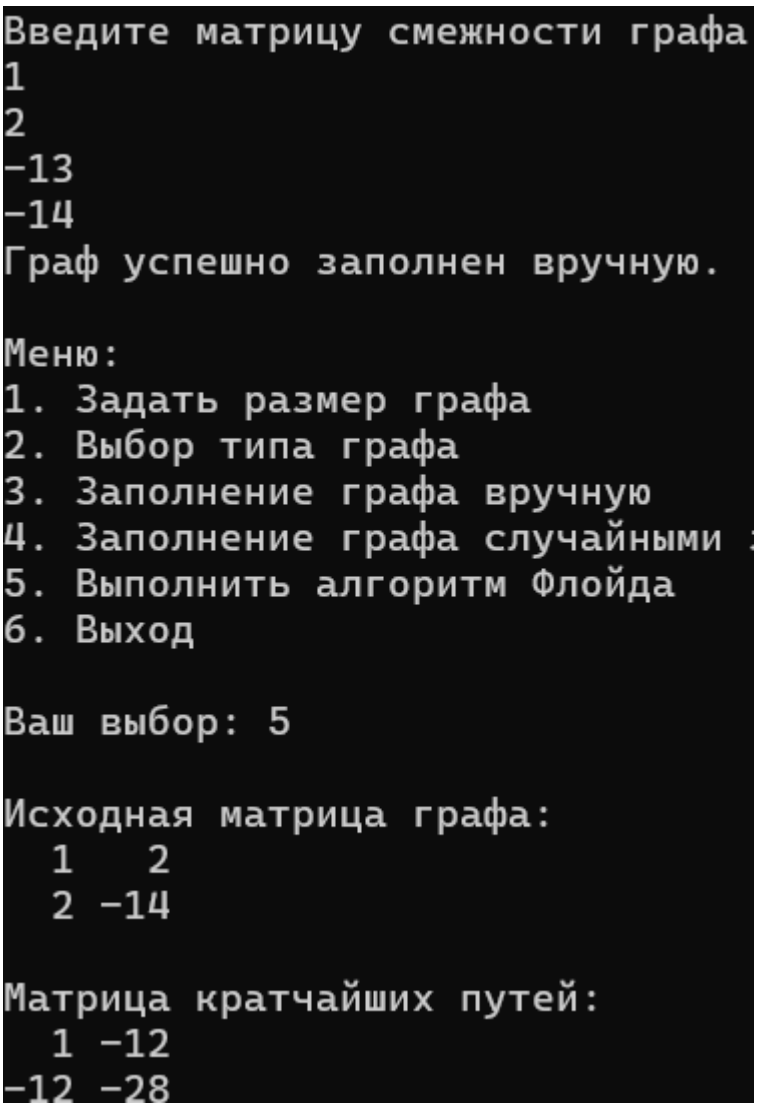

Рисунок 13 – Тестирование при вводе количества вершин в графе = 2, неориентированным графом и ручным вводом матрицы смежности

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

Таблица 2 – Итоги тестирования

№	Описание теста	Полученный результат
1	Запуск программы	Верно
2	Создание графа	Верно
3	Просмотр результатов	Верно
4	Анализ корректности графа	Верно
5	Визуализация графа	Верно
6	Сохранение результатов	Верно

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:

0 7 INF

13 0 11

13 14 0

Для первой вершины. Путь от первой вершины ко второй составляет 7, к третьей - INF. Следовательно, они и будут являться кратчайшими

Для второй вершины. Следовательно, D[0][2]=min(D[3][1],dist[0][1]+dist[1][2]=7+11=18. Это и есть кратчайший путь

Для третьей вершины никакие улучшения через промежуточные вершины (1 и 2) не приводят к сокращению пути:
Рассмотрим использование промежуточной вершины 1:

Путь от 3-й вершины к 1-й через вершину 1:
D[3][1]=min(D[3][1],D[3][0]+D[0][1])=min(13,13+7)=13

Тут путь через вершину 1 не улучшает результат, потому что прямой путь от 3-й вершины к 1-й уже равен 13.

Путь от 3-й вершины ко 2-й через вершину 1:
D[3][2]=min(D[3][2],D[3][0]+D[0][2])=min(14,13+∞)=14

Путь через вершину 1 не дает улучшения, потому что прямого пути от 3-й вершины к 2-й уже достаточно.

Рассмотрим использование промежуточной вершины 2:

Путь от 3-й вершины к 1-й через вершину 2:

D[3][1]=min(D[3][1],D[3][1]+D[1][1])=min(13,14+13)=13

Путь от 3-й вершины ко 2-й через вершину 2:

D[3][2]=min(D[3][2],D[3][1]+D[1][2])=min(14,14+11)=14

Путь через вершину 2 не улучшает результат.

Итоговая матрица расстояний:

0	7	18
13	0	11
13	14	0

Исходные данные были введены в программу. Оценивая результаты работы приложения, можно сделать вывод, что программа работает верно.

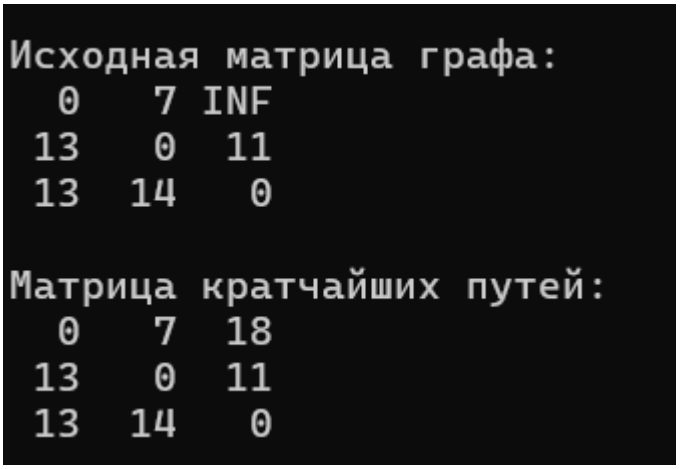

Рисунок 14 – Тестирование работы программы

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, который можно в будущем совершенствовать.

Программа имеет небольшой, но достаточный интерфейс для использования функциональных возможностей от генерации матрицы смежности вплоть до сохранения результатов в файл.

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 анг. 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. 2005. - 1024 с. 7. Скиена Дж., Ревилла Р., Шерифф Б. Алгоритмы: Фундаментальные принципы компьютерных наук. - М.: Вильямс

8. Паппас Т. Анализ алгоритмов и доказательство корректности. - М.: Бином, 2012. - 416 с.

Приложение А.
Листинг программы.

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

#define INF INT_MAX // Для обозначения бесконечности

void printMenu();
void inputGraph(int** graph, int size, int isDirected);
void randomGraph(int** graph, int size, int isDirected);
void floydWarshall(int** graph, int size);
void saveResults(int** dist, int size, const char* filename);
void printGraph(int** graph, int size);
void printDistances(int** dist, int size);

int main() {
    setlocale(LC_ALL, "Russian");
    int repeat = 1; // Флаг для повторения программы

    while (repeat) {
        int size = 0;
        int** graph = NULL;
        int choice;
        char sizeSet = 0, graphFilled = 0, floydRun = 0;
        int isDirected = -1;  // Флаг для ориентации графа (0 - неориентированный, 1 - ориентированный)

        while (!floydRun) {
            printMenu();
            printf("\nВаш выбор: ");
            if (scanf_s("%d", &choice) != 1) {
                printf("Ошибка ввода! Попробуйте снова.\n");
                while (getchar() != '\n'); // Очистка буфера ввода
```

```
            continue;
        }

        switch (choice) {
        case 1: // Задать размер графа
            printf("Введите размер графа: ");
            if (scanf_s("%d", &size) != 1 || size <= 0) {
                printf("Ошибка ввода! Размер графа должен быть положительным
числом.\n");
                while (getchar() != '\n'); // Очистка буфера ввода
                break;
            }

            // Освобождаем предыдущую память, если была
            if (graph) {
                for (int i = 0; i < size; i++) {
                    free(graph[i]);
                }
                free(graph);
            }

            // Выделяем память для графа
            graph = (int**)malloc(size * sizeof(int*));
            if (!graph) {
                printf("Ошибка выделения памяти!\n");
                exit(1);
            }

            for (int i = 0; i < size; i++) {
                graph[i] = (int*)malloc(size * sizeof(int));
                if (!graph[i]) {
                    printf("Ошибка выделения памяти!\n");
                    exit(1);
                }
            }
```

```
                // Инициализация графа значениями INF
                for (int i = 0; i < size; i++) {
                    for (int j = 0; j < size; j++) {
                        graph[i][j] = (i == j) ? 0 : INF;
                    }
                }

                printf("Размер графа установлен на %d.\n", size);
                sizeSet = 1; // Устанавливаем флаг
                break;

            case 2: // Выбор типа графа
                if (!sizeSet) {
                    printf("Сначала задайте размер графа!\n");
                    break;
                }
                printf("Выберите тип графа:\n");
                printf("1. Ориентированный граф\n");
                printf("2. Неориентированный граф\n");
                printf("Ваш выбор: ");
                if (scanf_s("%d", &isDirected) != 1 || (isDirected != 1 && isDirected != 2))
{
                    printf("Ошибка ввода! Попробуйте снова.\n");
                    break;
                }
                isDirected = (isDirected == 1) ? 1 : 0; // 1 - ориентированный, 0 -
неориентированный
                printf("Тип графа установлен на %s.\n", isDirected ? "ориентированный" :
"неориентированный");
                break;

            case 3: // Заполнение графа вручную
                if (!sizeSet || isDirected == -1) {
                    printf("Сначала задайте размер графа и выберите тип графа!\n");
                    break;
                }
                inputGraph(graph, size, isDirected);
```

```
                printf("Граф успешно заполнен вручную.\n");
                graphFilled = 1; // Устанавливаем флаг
                break;

            case 4: // Автоматическое (случайное) заполнение графа
                if (!sizeSet || isDirected == -1) {
                    printf("Сначала задайте размер графа и выберите тип графа!\n");
                    break;
                }
                randomGraph(graph, size, isDirected);
                printf("Граф успешно заполнен случайными значениями.\n");
                graphFilled = 1; // Устанавливаем флаг
                break;

            case 5: // Выполнение алгоритма Флойда
                if (!sizeSet || !graphFilled || isDirected == -1) {
                    printf("Сначала задайте размер графа, выберите тип графа и заполните
его!\n");
                    break;
                }

                floydWarshall(graph, size);
                floydRun = 1; // Устанавливаем флаг, чтобы выйти из вложенного цикла
                break;

            case 6: // Выход
                printf("Программа завершена.\n");
                repeat = 0;
                floydRun = 1; // Чтобы выйти из вложенного цикла
                break;

            default:
                printf("Неверный выбор. Попробуйте снова.\n");
            }
        }
```

```
        // Освобождаем память
        if (graph) {
            for (int i = 0; i < size; i++) {
                free(graph[i]);
            }
            free(graph);
        }

        // Функция повторения программы
        if (repeat) {
            printf("\nПовторить?\n1. Да\n2. Нет\nВаш выбор: ");
            if (scanf_s("%d", &repeat) != 1 || (repeat != 1 && repeat != 2)) {
                printf("Ошибка ввода! Программа завершена.\n");
                repeat = 0;
            }
            if (repeat == 2) {
                printf("Программа завершена.\n");
                repeat = 0;
            }
        }
    }

    return 0;
}

void printMenu() {
    printf("\nМеню:\n");
    printf("1. Задать размер графа\n");
    printf("2. Выбор типа графа\n");
    printf("3. Заполнение графа вручную\n");
    printf("4. Заполнение графа случайными значениями\n");
    printf("5. Выполнить алгоритм Флойда\n");
    printf("6. Выход\n");
}

void inputGraph(int** graph, int size, int isDirected) {
```

```
    printf("Введите матрицу смежности графа (используйте %d для обозначения отсутствия связи):\n", INF);
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (scanf_s("%d", &graph[i][j]) != 1) {
                printf("Ошибка ввода! Повторите ввод.\n");
                while (getchar() != '\n'); // Очистка буфера ввода
                j--;
            }
            if (i != j && graph[i][j] == 0) {
                graph[i][j] = INF; // Отсутствие пути
            }
        }
    }

    // Если граф неориентированный, дублируем связи
    if (!isDirected) {
        for (int i = 0; i < size; i++) {
            for (int j = i + 1; j < size; j++) {
                if (graph[i][j] != INF) {
                    graph[j][i] = graph[i][j];
                }
            }
        }
    }
}

void randomGraph(int** graph, int size, int isDirected) {
    srand((unsigned int)time(NULL));
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (i == j) {
                graph[i][j] = 0;
            }
            else {
                graph[i][j] = (rand() % 10 < 7) ? rand() % 20 + 1 : INF;
```

```
            }
        }
    }

    // Если граф неориентированный, дублируем связи
    if (!isDirected) {
        for (int i = 0; i < size; i++) {
            for (int j = i + 1; j < size; j++) {
                if (graph[i][j] != INF) {
                    graph[j][i] = graph[i][j];
                }
            }
        }
    }
}

void floydWarshall(int** graph, int size) {
    int** dist = (int**)malloc(size * sizeof(int*));
    for (int i = 0; i < size; i++) {
        dist[i] = (int*)malloc(size * sizeof(int));
        for (int j = 0; j < size; j++) {
            dist[i][j] = graph[i][j];
        }
    }

    printf("\nИсходная матрица графа:\n");
    printGraph(graph, size);

    // Алгоритм Флойда
    for (int k = 0; k < size; k++) {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] <
dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
```

```
            }
        }
    }

    printf("\nМатрица кратчайших путей:\n");
    printDistances(dist, size);

    saveResults(dist, size, "results");
    printf("\nРезультат сохранен в файл 'results'.\n");

    for (int i = 0; i < size; i++) {
        free(dist[i]);
    }
    free(dist);
}

void printGraph(int** graph, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (graph[i][j] == INF) {
                printf("INF ");
            }
            else {
                printf("%3d ", graph[i][j]);
            }
        }
        printf("\n");
    }
}

void printDistances(int** dist, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (dist[i][j] == INF) {
                printf("INF ");
            }
```

```
            else {
                printf("%3d ", dist[i][j]);
            }
        }
        printf("\n");
    }
}

void saveResults(int** dist, int size, const char* filename) {
    FILE* file;
    if (fopen_s(&file, filename, "w") != 0) {
        printf("Ошибка открытия файла для записи.\n");
        return;
    }

    fprintf(file, "Матрица кратчайших расстояний:\n");
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (dist[i][j] == INF) {
                fprintf(file, "INF ");
            }
            else {
                fprintf(file, "%3d ", dist[i][j]);
            }
        }
        fprintf(file, "\n");
    }

    fclose(file);
}
```