

Министерство науки и высшего образования РФ
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проектированию
по курсу «Логика и основы алгоритмизации в
инженерных задачах»
на тему «Реализация алгоритма Форда-Беллмана»

24.12.25
хорошо
[подпись]

Выполнил:
Студент группы 24BVB3
Буров А.О.

Принял:
Юрова О.В.

Пенза 2025

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

«__» __ 20__

ЗАДАНИЕ

на курсовое проектирование по курсу

«Поиск и основы оптимизации в инженерных задачах»
Студенту Бурову Александру Олеговичу Группа 24ВВВЗ
Тема проекта Реализация алгоритма Форда-Веллмана

Исходные данные (технические требования) на проектирование

- Разработка алгоритмов и программного обеспечения в соответствии с данными заданием курсового проекта
- Подсчитательная записка должна содержать:
1. Постановку задачи;
 2. теоретическую часть задачи;
 3. Описание алгоритма поставленной задачи;
 4. Пример ручного расчёта задачи и вычислений;
 5. Описание самой программы;
 6. Тесты;
 7. Текст литературы;
 8. Миссия программы;
 9. Результаты 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 програ
- 5 Оформление пояснительной записки
- 6
- 7
- 8

Дата выдачи задания "19" сентября 2025

Дата защиты проекта "

Руководитель

Крова О.В.

Задание получил

"19" сентября

2025 г.

Студент

Буров Александр Олегович

Содержание

Реферат.....	5
Введение.....	6
1. Постановка задачи.....	7
2. Теоретическая часть задания.....	8
3. Описание алгоритма программы.....	10
4. Описание программы.....	12
4.1. Класс Program.....	12
4.2. Класс Edge.....	16
4.3. Основные методы класса Program.....	17
4.4. Диаграмма вариантов использования.....	24
4.5. Диаграмма классов.....	25
5. Тестирование.....	26
5.1. Создание графа (случайная генерация).....	28
5.2. Загрузка графа из файла.....	29
5.3. Поиск кратчайших путей.....	30
5.4. Сохранение графа в файл.....	30
5.5. Просмотр матрицы смежности.....	32
5.6. Результаты тестирования.....	32
Заключение.....	34
Список литературы.....	35
Приложение А.....	36

Реферат

Отчет 53 страниц, 20 рисунков, 2 таблицы, 3 источника.

ГРАФ, ТЕОРИЯ ГРАФОВ, КРАТЧАЙШИЙ ПУТЬ, АЛГОРИТМ ФОРДА-БЕЛЛМАНА

Алгоритм Форда-Беллмана – это алгоритм динамического программирования, который последовательно релаксирует рёбра графа для нахождения кратчайших путей. В работе реализован алгоритм Форда-Беллмана, способный обрабатывать графы с отрицательными весами рёбер и обнаруживать отрицательные циклы. Установлено, что это один из наиболее универсальных алгоритмов для решения задачи поиска кратчайших путей в графах с произвольными весами.

Введение

Граф – математическая структура, состоящая из точек (вершин), соединённых линиями (рёбрами). Используется для моделирования связей между объектами. В настоящее время графы применяются в различных областях: компьютерные сети, транспортные системы, анализ данных и искусственный интеллект.

Графы можно классифицировать по нескольким критериям:

- Неориентированные графы: рёбра не имеют направления, связь между вершинами двусторонняя.
- Ориентированные графы: рёбра имеют направление от одной вершины к другой, образуя односторонние связи.
- Невзвешенные графы: рёбра не имеют числовых значений, важна только факт наличия связи.
- Взвешенные графы: каждому ребру присвоено число — вес, который может означать расстояние, время, стоимость или другую характеристику.

В данной работе рассматриваются взвешенные ориентированные и неориентированные графы, где вес ребра может быть положительным, отрицательным или нулевым. Особенностью реализуемого алгоритма является возможность обработки графов с отрицательными весами, что позволяет решать задачи, где переходы могут быть как выгодными, так и убыточными.

В качестве среды разработки выбрана Microsoft Visual Studio Community 2022, язык программирования — C#. Целью курсовой работы является создание программы на языке C#, реализующей алгоритм Форда-Беллмана для поиска кратчайших путей в графе. Программа позволяет генерировать графы, загружать их из файлов, выполнять анализ на наличие кратчайших путей и обнаруживать отрицательные циклы.

1. Постановка задачи

Программа должна реализовывать алгоритм Форда-Беллмана для поиска кратчайших путей во взвешенном графе, допускающем наличие рёбер с отрицательными весами. Граф может быть как ориентированным, так и неориентированным, и включать в себя петли. Необходимо обеспечить возможность создания графа различными способами: случайная генерация с настраиваемыми параметрами (количество вершин, вероятность образования ребра, диапазон весов), ручной ввод параметров для генерации, а также загрузка из файла в формате матрицы смежности. Программа должна выполнять вычисление кратчайших путей от выбранной пользователем стартовой вершины до всех остальных вершин графа. Критически важной функцией является обнаружение отрицательных циклов, достижимых из стартовой вершины, с выводом соответствующего предупреждения, так как в этом случае кратчайшие пути не существуют. Требуется реализовать возможность сохранения текущего графа в текстовый файл в формате матрицы смежности для последующего использования. Пользовательский интерфейс должен предоставлять визуализацию графа в виде матрицы смежности, отображающей веса всех рёбер. Программа обязана корректно обрабатывать исключительные ситуации, связанные с вводом некорректных данных пользователем, чтением/записью файлов и выполнением алгоритма. Управление программой должно осуществляться через консольное меню, обеспечивающее интуитивно понятную навигацию.

2. Теоретическая часть задания

Алгоритм Форда-Беллмана, также известный как алгоритм Беллмана-Форда (англ. Bellman–Ford algorithm), работает с взвешенными графами, представленными в виде списка рёбер или матрицы смежности. Алгоритм рассматривает граф как набор вершин и взвешенных рёбер, где вес может быть положительным, отрицательным или нулевым. Пример такого графа, содержащий рёбра как с положительными, так и с отрицательными весами, представлен на Рисунке 1.

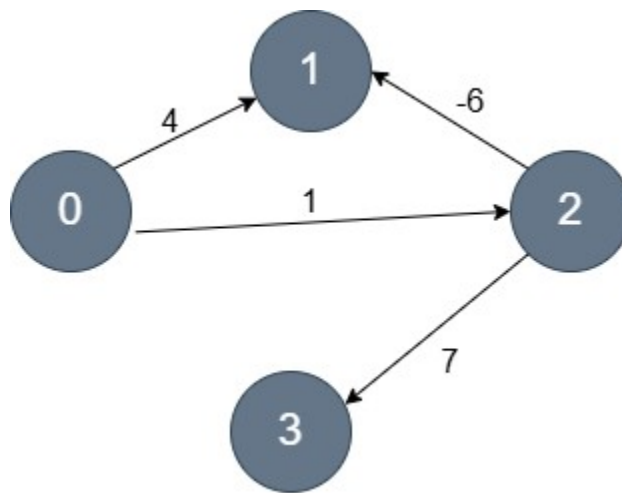


Рисунок 1 - Пример ориентированного взвешенного графа с отрицательными весами

Алгоритм Форда-Беллмана, используемый для поиска кратчайших путей в графах с отрицательными весами, отличается от других алгоритмов (таких как алгоритм Дейкстры) способностью корректно обрабатывать отрицательные веса рёбер и обнаруживать отрицательные циклы. По сравнению с алгоритмом Дейкстры, алгоритм Форда-Беллмана имеет большую временную сложность – $O(V \cdot E)$ вместо $O(E + V \log V)$, однако это компенсируется его универсальностью и возможностью работы с графами произвольных весов.

Алгоритм поиска кратчайших путей в программе использует динамическое программирование, с помощью которого программа вычисляет

минимальные расстояния от стартовой вершины до всех остальных. Принцип работы данного алгоритма заключается в последовательном улучшении оценок расстояний на основе информации о весах всех рёбер графа.

3. Описание алгоритма программы

Принцип работы алгоритма Форда-Беллмана:

- 1) Всем вершинам графа присваивается расстояние ∞ (бесконечность), кроме стартовой вершины, которой присваивается расстояние 0.
- 2) Для каждого ребра (u, v) с весом w выполняется операция релаксации: если расстояние до вершины u + вес ребра w < расстояния до вершины v , то расстояние до v обновляется. Этот процесс повторяется $V-1$ раз, где V – количество вершин в графе.
- 3) После $V-1$ итераций выполняется дополнительная итерация релаксации рёбер. Если на этой итерации происходит улучшение расстояния до какой-либо вершины, значит в графе существует отрицательный цикл, достижимый из стартовой вершины.
- 4) Если отрицательный цикл не обнаружен, вычисленные расстояния представляют собой кратчайшие пути от стартовой вершины до всех остальных вершин графа.

Для реализации алгоритма Форда-Беллмана используется метод FindPaths(). Логика данного метода можно описать следующим образом:

ФУНКЦИЯ FindPaths(start):

dist = массив[verticesCount] со значением ∞

dist[start] = 0

ДЛЯ $i = 1$ ДО verticesCount-1:

ДЛЯ КАЖДОГО edge В edges:

ЕСЛИ dist[edge.From] $\neq \infty$ И

dist[edge.From] + edge.Weight < dist[edge.To]:

dist[edge.To] = dist[edge.From] + edge.Weight

hasNegativeCycle = ЛОЖЬ

ДЛЯ КАЖДОГО edge В edges:

ЕСЛИ $\text{dist}[\text{edge.From}] \neq \infty$ И

$\text{dist}[\text{edge.From}] + \text{edge.Weight} < \text{dist}[\text{edge.To}]$:

hasNegativeCycle = ИСТИНА

ПРЕРВАТЬ

ЕСЛИ hasNegativeCycle:

ВЫВЕСТИ "Обнаружен отрицательный цикл!"

ИНАЧЕ:

ДЛЯ КАЖДОЙ вершины i:

ЕСЛИ $\text{dist}[i] = \infty$:

ВЫВЕСТИ "Вершина " + i + " недостижима"

ИНАЧЕ:

ВЫВЕСТИ "До вершины " + i + ": " + $\text{dist}[i]$

КОНЕЦ ФУНКЦИИ

4. Описание программы

Программа была написана в среде разработки VisualStudioCommunity 2022 на языке C# с использованием объектно-ориентированного подхода программирования.

Далее будут приведены описания классов и их методов, используемых в программе. Полный листинг программы приведен в Приложении А.

4.1. Класс Program

Данный класс содержит точку входа в программу – метод Main. В классе Program определены статические поля для хранения данных о графе: список рёбер edges, количество вершин verticesCount, признак ориентированности графа isDirected и имя текущего файла currentFileName.

В методе Main реализовано главное меню программы (Рисунок 2). Главное меню предоставляет 8 вариантов действий: случайная генерация графа, ручной ввод параметров генерации, загрузка из файла, поиск кратчайших путей, сохранение в файл, просмотр матрицы смежности, очистка графа и выход из программы.

```
=== АЛГОРИТМ ФОРДА-БЕЛЛМАНА ===  
Текущий граф: 0 рёбер, 0 вершин  
Тип: Неориентированный  
=====
```

1. Случайная генерация графа
2. Ввести граф вручную (с параметрами)
3. Загрузить из файла
4. Найти кратчайшие пути
5. Сохранить в файл (матрица)
6. Показать граф (матрица смежности)
7. Очистить граф
8. Выход

Выберите:

Рисунок 2 - Главное меню программы

За обработку выбора пользователя отвечает переменная `choice`, которая используется в операторе выбора `switch`.

При выборе случайной генерации графа выполняется блок кода, вызывающий метод `GenerateRandomSimple()`. Данный метод предлагает пользователю ввести количество вершин (Рисунок 3) и выбрать ориентированность графа. После этого программа автоматически генерирует граф с заданными параметрами.

```
Выберите: 1  
Количество вершин:
```

Рисунок 3 - Ввод количества вершин

При выборе ручного ввода параметров вызывается метод `InputManualWithParams()`. Данный метод запрашивает у пользователя все необходимые параметры: количество вершин, ориентированность, вероятность образования ребра, диапазон весов и минимальное количество рёбер (Рисунок 4).

```
Выберите: 2  
  
Количество вершин: 5  
Ориентированный (y/n): y  
Вероятность ребра (0-100%): 50  
Максимальный вес (включая отрицательные): 3  
Минимальное количество рёбер (0 - без ограничений): 2
```

Рисунок 4 - Ввод параметров для генерации графа

При выборе загрузки графа из файла вызывается метод LoadFromFile(). Данный метод запрашивает имя файла и загружает граф в формате матрицы смежности (Рисунок 5).

```
Выберите: 3  
  
Имя файла: matrix.txt
```

Рисунок 5 - Загрузка графа из файла

Если пользователь выбирает поиск кратчайших путей, вызывается метод FindPaths(). Данный метод запрашивает стартовую вершину и выполняет алгоритм Форда-Беллмана для нахождения кратчайших путей до всех остальных вершин (Рисунок 6). Если обнаружен отрицательный цикл, программа выводит соответствующее предупреждение.

```
Выберите: 4  
  
Стартовая вершина (0-4): 3  
  
=== РЕЗУЛЬТАТЫ ===  
Из вершины 3:  
    до 0: недостижима  
    до 1: недостижима  
    до 2: недостижима  
    до 3: 0  
    до 4: недостижима
```

Рисунок 6 – Поиск кратчайших путей

При выборе пользователем просмотра матрицы смежности вызывается метод ShowGraphMatrix(), который выводит матрицу смежности графа, отображая веса всех рёбер и петли (Рисунок 7).

```
Выберите: 6
=== МАТРИЦА СМЕЖНОСТИ ===
Вершин: 5
Рёбер: 8
Тип: Ориентированный

      0   1   2   3   4
-----
0 |   -9  -5  -5  INF  INF
1 |   INF  INF  INF  INF  -6
2 |   INF   7  INF  INF   3
3 |   INF  INF  INF  INF  INF
4 |   INF   5  INF  INF  -8

INF - отсутствие ребра (в т.ч. отсутствие петли)
Число - вес ребра (в т.ч. вес петли)
```

Рисунок 7 – Матрица смежности графа

Для сохранения графа в файл используется метод SaveToFileMatrix(), который создаёт текстовый файл в формате матрицы смежности (Рисунок 8).

```
Выберите: 5

Имя файла: matrix.txt
Граф сохранён в файл matrix.txt (формат матрицы)
Файл можно открыть в любом текстовом редакторе
```

Рисунок 8 – Сохранение графа в файл

4.2. Класс Edge

Данный класс является вложенным классом внутри класса Program и отвечает за представление рёбер графа. Он содержит три публичных поля:

```
public class Edge
{
    public int From;
    public int To;
    public int Weight;
}
```

Поле From хранит индекс начальной вершины ребра, принимая значения от 0 до verticesCount-1. Поле To содержит индекс конечной вершины; для неориентированного графа при сохранении создается два объекта Edge с обратными значениями From и To. Поле Weight определяет вес ребра, который может быть положительным, отрицательным или нулевым, представляя стоимость, расстояние или другую количественную характеристику перехода между вершинами.

Класс реализован без методов, содержа только поля данных, что соответствует паттерну Data Transfer Object (DTO). Все поля объявлены публичными для упрощения доступа из внешнего кода. Для хранения всех рёбер графа используется универсальная коллекция List<Edge>, обеспечивающая динамическое изменение количества рёбер. В программе допускается существование нескольких рёбер с одинаковыми вершинами From и To, что позволяет моделировать мультиграфы.

4.3. Основные методы класса Program

Класс Program содержит 8 основных методов, реализующих функционал работы с графами и алгоритмом Форда-Беллмана. Рассмотрим ключевые из них.

1) Метод FindPaths()

Данный метод реализует алгоритм Форда-Беллмана для поиска кратчайших путей от выбранной стартовой вершины до всех остальных вершин графа. Он принимает один параметр – номер стартовой вершины. Внутри метода определяются следующие локальные переменные: массив расстояний `dist`, итерационные переменные и флаг обнаружения отрицательного цикла.

В ходе выполнения данного метода сначала происходит инициализация массива расстояний: всем вершинам присваивается значение `int.MaxValue` (бесконечность), кроме стартовой вершины, которой присваивается 0. Затем выполняется $V-1$ итераций релаксации рёбер, где V – количество вершин. На каждой итерации для каждого ребра проверяется возможность улучшения расстояния до конечной вершины через начальную. После $V-1$ итераций выполняется дополнительная проверка на наличие отрицательных циклов: если на этой дополнительной итерации происходит улучшение какого-либо расстояния, значит в графе существует достижимый из стартовой вершины отрицательный цикл. Логика данного метода можно описать следующим образом:

ФУНКЦИЯ `FindPaths(start)`:

`dist` = массив[`verticesCount`] со значением ∞

`dist[start] = 0`

ДЛЯ $i = 1$ ДО `verticesCount-1`:

ДЛЯ КАЖДОГО `edge` В `edges`:

ЕСЛИ `dist[edge.From]  $\neq \infty$` И

`dist[edge.From] + edge.Weight < dist[edge.To]`:

$\text{dist}[\text{edge.To}] = \text{dist}[\text{edge.From}] + \text{edge.Weight}$

hasNegativeCycle = ЛОЖЬ

ДЛЯ КАЖДОГО edge В edges:

ЕСЛИ $\text{dist}[\text{edge.From}] \neq \infty$ И

$\text{dist}[\text{edge.From}] + \text{edge.Weight} < \text{dist}[\text{edge.To}]$:

hasNegativeCycle = ИСТИНА

ПРЕРВАТЬ

ЕСЛИ hasNegativeCycle:

ВЫВЕСТИ "Обнаружен отрицательный цикл!"

ИНАЧЕ:

ДЛЯ КАЖДОЙ вершины i:

ЕСЛИ $\text{dist}[i] = \infty$:

ВЫВЕСТИ "Вершина " + i + " недостижима"

ИНАЧЕ:

ВЫВЕСТИ "До вершины " + i + ": " + $\text{dist}[i]$

КОНЕЦ ФУНКЦИИ

2) Метод GenerateRandomSimple()

Данный метод реализует случайную генерацию графа с заданными параметрами. Он не принимает параметров, но запрашивает у пользователя ввод: количество вершин и ориентированность графа. Внутри метода создается объект класса Random для генерации случайных чисел.

В ходе выполнения данного метода для каждой пары вершин (i, j) с заданной вероятностью (30% по умолчанию) создается ребро со случайным весом в диапазоне от -10 до 10. Для неориентированных графов создается два ребра с обратными направлениями. Также с определенной вероятностью могут создаваться петли (ребра, где начальная и конечная вершины совпадают). Логику данного метода можно описать следующим образом:

ФУНКЦИЯ FindPaths(start):

dist = массив[verticesCount] со значением ∞

dist[start] = 0

ДЛЯ i = 1 ДО verticesCount-1:

ДЛЯ КАЖДОГО edge В edges:

ЕСЛИ dist[edge.From] $\neq \infty$ И

dist[edge.From] + edge.Weight < dist[edge.To]:

dist[edge.To] = dist[edge.From] + edge.Weight

hasNegativeCycle = ЛОЖЬ

ДЛЯ КАЖДОГО edge В edges:

ЕСЛИ dist[edge.From] $\neq \infty$ И

dist[edge.From] + edge.Weight < dist[edge.To]:

hasNegativeCycle = ИСТИНА

ПРЕРВАТЬ

ЕСЛИ hasNegativeCycle:

ВЫВЕСТИ "Обнаружен отрицательный цикл!"

ИНАЧЕ:

ДЛЯ КАЖДОЙ вершины i:

ЕСЛИ dist[i] = ∞ :

ВЫВЕСТИ "Вершина " + i + " недостижима"

ИНАЧЕ:

ВЫВЕСТИ "До вершины " + i + ": " + dist[i]

КОНЕЦ ФУНКЦИИ

3) Метод ShowGraphMatrix()

Данный метод выводит матрицу смежности текущего графа. Он не принимает параметров и работает с глобальными переменными edges и verticesCount. Метод создает двумерный массив строк для представления матрицы.

В ходе выполнения данного метода сначала инициализируется матрица значениями "INF" (бесконечность), что означает отсутствие ребра. Затем для каждого ребра в списке edges записывается его вес в соответствующую ячейку матрицы. После формирования матрицы происходит её вывод в консоль с форматированием: заголовки столбцов, разделительная линия и строки матрицы с выравниванием. Логика данного метода можно описать следующим образом:

ФУНКЦИЯ ShowGraphMatrix():

ЕСЛИ edges.размер() = 0:

ВЫВЕСТИ "Граф пуст!"

ВЕРНУТЬ

matrix = двумерный_массив[verticesCount, verticesCount] со значением "INF"

ДЛЯ КАЖДОГО edge В edges:

matrix[edge.From, edge.To] = строка(edge.Weight)

ВЫВЕСТИ "=== МАТРИЦА СМЕЖНОСТИ ==="

ВЫВЕСТИ " "

ДЛЯ j = 0 ДО verticesCount-1:

ВЫВЕСТИ форматировать(j, ширина=5)

ВЫВЕСТИ перевод_строки()

ВЫВЕСТИ " "

ДЛЯ j = 0 ДО verticesCount-1:

ВЫВЕСТИ "-----"

ВЫВЕСТИ перевод_строки()

ДЛЯ i = 0 ДО verticesCount-1:

ВЫВЕСТИ форматировать(i, ширина=2) + " | "

ДЛЯ j = 0 ДО verticesCount-1:

ВЫВЕСТИ форматировать(matrix[i,j], ширина=5)

ВЫВЕСТИ перевод_строки()

ВЫВЕСТИ "INF - отсутствие ребра"

КОНЕЦ ФУНКЦИИ

4) Метод SaveToFileMatrix()

Данный метод сохраняет текущий граф в текстовый файл в формате матрицы смежности с дополнительной возможностью сохранения результатов поиска кратчайших путей. Он запрашивает у пользователя имя файла и создает файл с указанным именем.

В ходе выполнения данного метода сначала проверяется, не пуст ли граф. Затем создается матрица аналогично методу ShowGraphMatrix(), но для сохранения в файл. Формат файла: первая строка содержит тип формата, количество вершин и ориентированность графа, последующие строки – строки матрицы смежности. После сохранения матрицы пользователю предлагается сохранить результаты поиска кратчайших путей для выбранной стартовой вершины. Логика данного метода можно описать следующим образом:

ФУНКЦИЯ SaveToFileMatrix():

ЕСЛИ edges.размер() = 0:

ВЫВЕСТИ "Граф пуст! Сначала создайте граф."

ВЕРНУТЬ

ВЫВЕСТИ "Имя файла: "

filename = ВВОД_СТРОКИ()

matrix = двумерный_массив[verticesCount, verticesCount] со значением "INF"

ДЛЯ КАЖДОГО edge В edges:

matrix[edge.From, edge.To] = строка(edge.Weight)

ОТКРЫТЬ_ФАЙЛ(filename) ДЛЯ_ЗАПИСИ

ЗАПИСАТЬ_В_ФАЙЛ("Matrix " + verticesCount + " " +
(ЕСЛИ isDirected ТО "directed" ИНАЧЕ "undirected"))

ДЛЯ i = 0 ДО verticesCount-1:

row = ""

ДЛЯ j = 0 ДО verticesCount-1:

row = row + matrix[i,j]

ЕСЛИ j < verticesCount-1:

row = row + " "

ЗАПИСАТЬ_В_ФАЙЛ(row)

ВЫВЕСТИ "Для какой вершины сохранить результаты поиска? (0-" +
(verticesCount-1) + "): "

startVertex = ВВОД_ЧИСЛА()

ЕСЛИ startVertex >= 0 И startVertex < verticesCount:

dist = массив[verticesCount] со значением ∞

dist[startVertex] = 0

ДЛЯ k = 1 ДО verticesCount-1:

ДЛЯ КАЖДОГО edge В edges:

ЕСЛИ dist[edge.From] $\neq \infty$ И

dist[edge.From] + edge.Weight < dist[edge.To]:

$\text{dist}[\text{edge.To}] = \text{dist}[\text{edge.From}] + \text{edge.Weight}$

hasNegativeCycle = ЛОЖЬ

ДЛЯ КАЖДОГО edge В edges:

ЕСЛИ $\text{dist}[\text{edge.From}] \neq \infty$ И

$\text{dist}[\text{edge.From}] + \text{edge.Weight} < \text{dist}[\text{edge.To}]$:

hasNegativeCycle = ИСТИНА

ПРЕРВАТЬ

ЗАПИСАТЬ_В_ФАЙЛ("\n=== РЕЗУЛЬТАТЫ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ ===")

ЗАПИСАТЬ_В_ФАЙЛ("Стартовая вершина: " + startVertex)

ЕСЛИ hasNegativeCycle:

ЗАПИСАТЬ_В_ФАЙЛ("Обнаружен отрицательный цикл!")

ИНАЧЕ:

ЗАПИСАТЬ_В_ФАЙЛ("Кратчайшие расстояния:")

ДЛЯ $i = 0$ ДО $\text{verticesCount}-1$:

ЕСЛИ $\text{dist}[i] = \infty$:

ЗАПИСАТЬ_В_ФАЙЛ(" до вершины " + i + ": недостижима")

ИНАЧЕ:

ЗАПИСАТЬ_В_ФАЙЛ(" до вершины " + i + ": " + $\text{dist}[i]$)

ИНАЧЕ:

ЗАПИСАТЬ_В_ФАЙЛ("Неверный номер вершины.")

ЗАКРЫТЬ_ФАЙЛ()

ВЫВЕСТИ "Граф и результаты сохранены в файл " + filename

КОНЕЦ ФУНКЦИИ

4.4. Диаграмма вариантов использования

Диаграмма вариантов использования (диаграмма прецедентов) – это визуальное представление взаимодействия пользователей с системой. Основное ее назначение – показать, что система делает с точки зрения пользователя, а не как она это делает технически. Диаграмма вариантов использования программы представлена на рисунке 9.

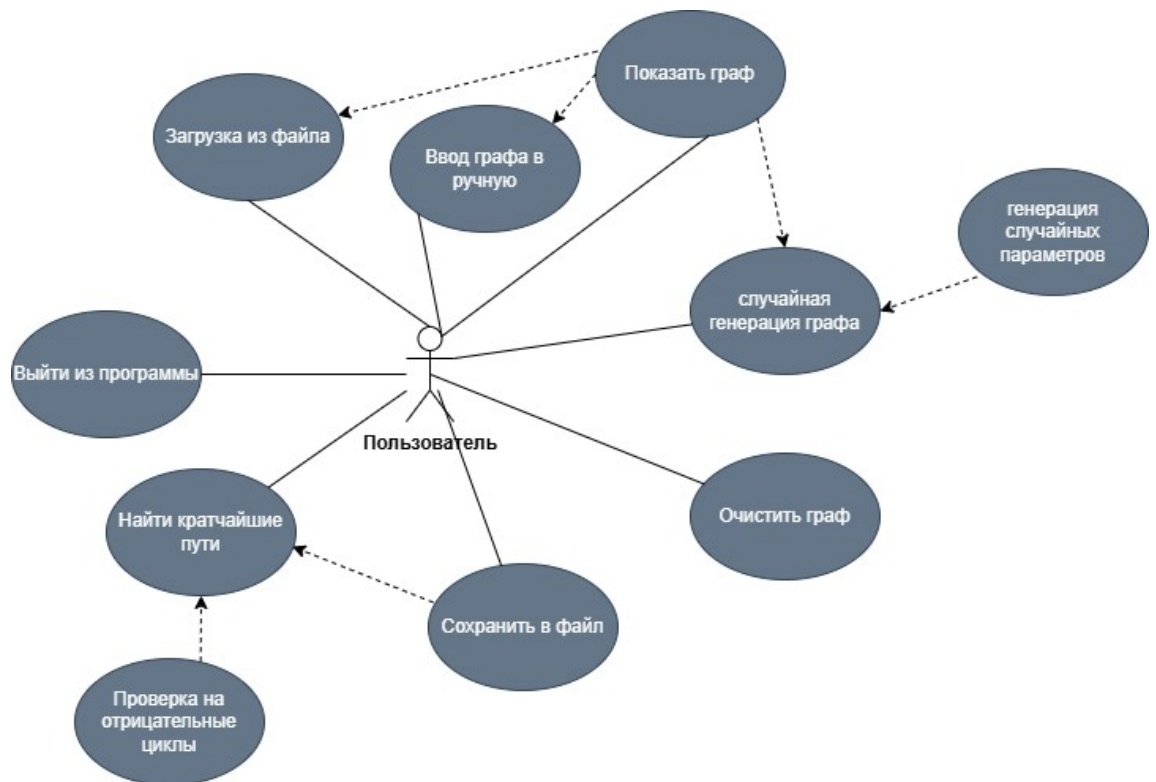


Рисунок 9 – Диаграмма вариантов использования

4.5. Диаграмма классов

Диаграмма классов – структурная диаграмма, которая демонстрирует общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними. На рисунке 10 представлена диаграмма классов программы.

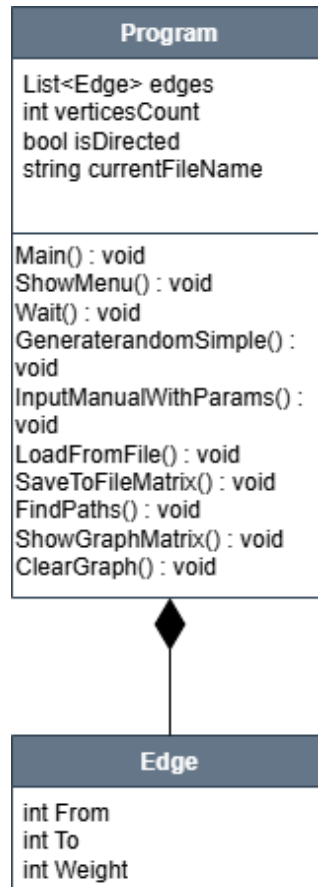


Рисунок 10 – Диаграмма классов

5. Тестирование

Тестирование программы осуществлялось на протяжении всего процесса разработки, а также после её завершения. Каждый новый функциональный модуль или элемент управления проверялся в режиме отладки сразу после реализации с целью выявления и устранения потенциальных ошибок на ранних этапах. В процессе тестирования были обнаружены и исправлены проблемы, связанные с обработкой входных данных, управлением графами и взаимодействием с файловой системой. Процедура проверки работоспособности программы и её соответствия требованиям отражена в таблице 1.

Таблица 1 – План тестирования программы

№	Описание теста	Предусловие	Тестирование	Ожидаемый результат
1	Запуск программы	нет	Запуск программы с помощью VisualStudio	Запуск метода Main класса Program
2	Создание графа (случайная генерация)	Программа запущена, введены параметры	Выбор пункта №1 в главном меню	Инициализация списка edges и переменных графа
3	Загрузка графа из файла	Файл с графом существует	Выбор пункта №3 в главном меню	Успешная загрузка графа из файла
4	Поиск кратчайших путей	Граф успешно создан или загружен	Выбор пункта №4 в главном меню	Корректное выполнение алгоритма Форда-Беллмана

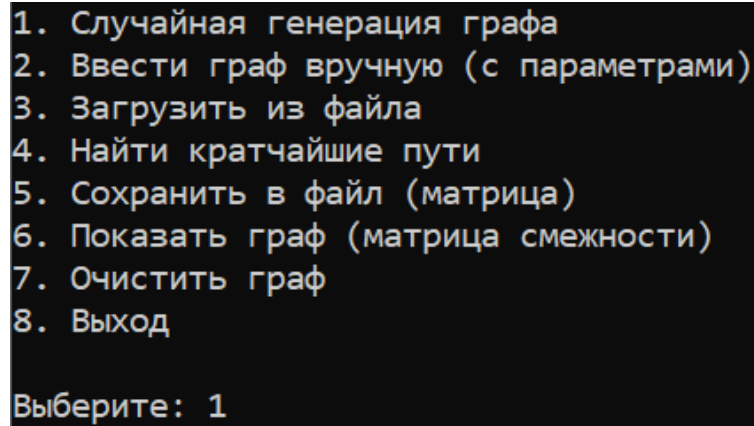
Продолжение таблицы 1

№	Описание теста	Предусловие	Тестирование	Ожидаемый результат
5	Сохранение графа в файл	Граф успешно создан или загружен	Выбор пункта №5 в главном меню	Успешное создание файла с матрицей смежности
6	Просмотр матрицы смежности	Граф успешно создан или загружен	Выбор пункта №6 в главном меню	Корректный вывод матрицы в консоль

Ниже будет приведено несколько тестов.

5.1. Создание графа (случайная генерация)

В главном меню выбираю пункт №1 – «Случайная генерация графа» (Рисунок 11).

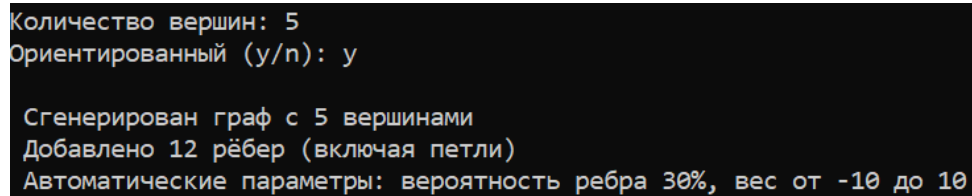


```
1. Случайная генерация графа
2. Ввести граф вручную (с параметрами)
3. Загрузить из файла
4. Найти кратчайшие пути
5. Сохранить в файл (матрица)
6. Показать граф (матрица смежности)
7. Очистить граф
8. Выход

Выберите: 1
```

Рисунок 11 – Выбор пункта «Случайная генерация графа»

Затем ввожу количество вершин (например, 5) и выбираю ориентированность графа (y/n) (Рисунок 12).



```
Количество вершин: 5
Ориентированный (y/n): y

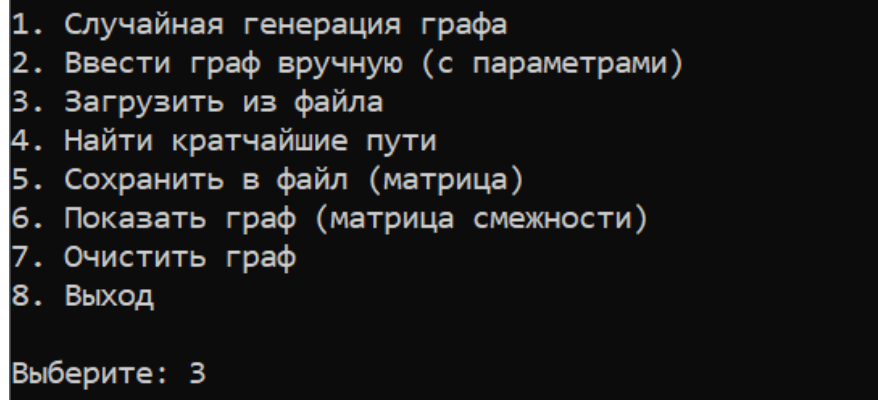
Сгенерирован граф с 5 вершинами
Добавлено 12 рёбер (включая петли)
Автоматические параметры: вероятность ребра 30%, вес от -10 до 10
```

Рисунок 12 – Ввод количества вершин графа

После чего в консоли отображается информация о сгенерированном графе: количество вершин и рёбер. Тест пройден успешно.

5.2. Загрузка графа из файла

Для загрузки графа из файла выбираю пункт №3 — «Загрузить из файла» (Рисунок 13).

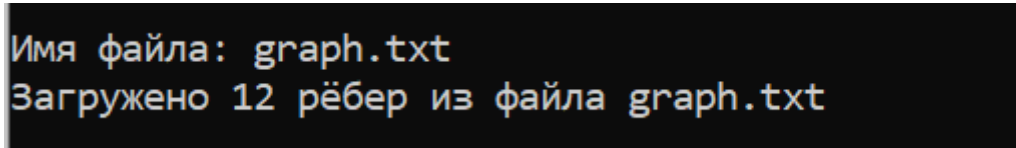


```
1. Случайная генерация графа
2. Ввести граф вручную (с параметрами)
3. Загрузить из файла
4. Найти кратчайшие пути
5. Сохранить в файл (матрица)
6. Показать граф (матрица смежности)
7. Очистить граф
8. Выход

Выберите: 3
```

Рисунок 13 – Выбор пункта «Загрузить из файла»

Ввожу имя существующего файла (например, «graph.txt») с матрицей смежности. Программа загружает граф и выводит количество загруженных рёбер (Рисунок 14).



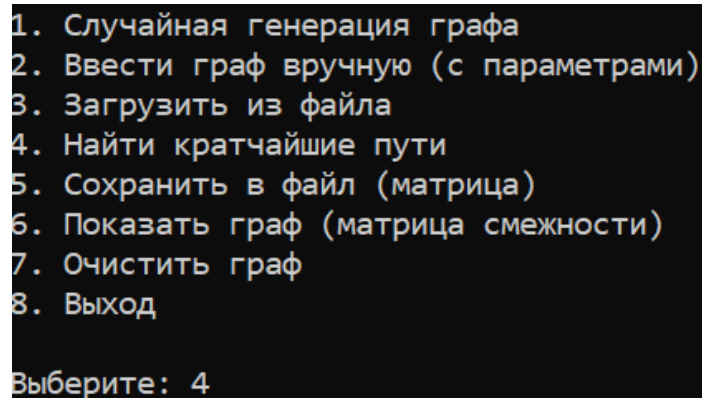
```
Имя файла: graph.txt
Загружено 12 рёбер из файла graph.txt
```

Рисунок 14 – Загрузка графа из файла

Наблюдаемый результат соответствует ожидаемому — граф успешно загружен. Тест пройден успешно.

5.3. Поиск кратчайших путей

Для анализа графа выбираю пункт №4 — «Найти кратчайшие пути» (Рисунок 15).

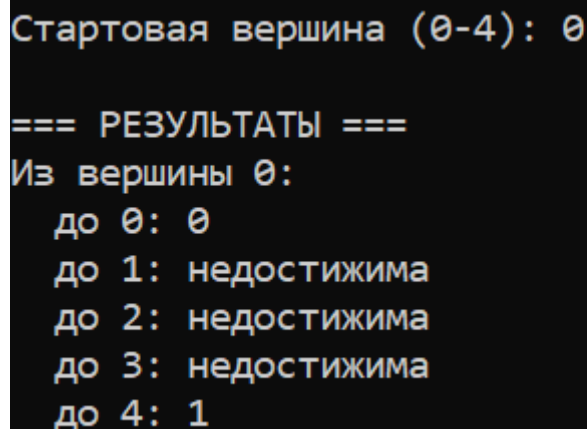


```
1. Случайная генерация графа
2. Ввести граф вручную (с параметрами)
3. Загрузить из файла
4. Найти кратчайшие пути
5. Сохранить в файл (матрица)
6. Показать граф (матрица смежности)
7. Очистить граф
8. Выход

Выберите: 4
```

Рисунок 15 – Выбор пункта «Найти кратчайшие пути»

Программа запрашивает номер стартовой вершины (например, 0). После выполнения алгоритма выводятся расстояния до всех вершин или предупреждение об обнаружении отрицательного цикла (Рисунок 16).



```
Стартовая вершина (0-4): 0

=== РЕЗУЛЬТАТЫ ===
Из вершины 0:
  до 0: 0
  до 1: недостижима
  до 2: недостижима
  до 3: недостижима
  до 4: 1
```

Рисунок 16 – Вывод расстояния до всех вершин

Мы можем наблюдать корректное выполнение алгоритма Форда-Беллмана с правильным расчётом кратчайших расстояний. Тест пройден успешно.

5.4. Сохранение графа в файл

После создания или загрузки графа выбираю пункт №5 — «Сохранить в файл (матрица)» (Рисунок 17).

```
1. Случайная генерация графа
2. Ввести граф вручную (с параметрами)
3. Загрузить из файла
4. Найти кратчайшие пути
5. Сохранить в файл (матрица)
6. Показать граф (матрица смежности)
7. Очистить граф
8. Выход

Выберите: 5
```

Рисунок 17 – Выбор пункта «Сохранить в файл (матрица)»

Ввожу имя файла для сохранения (например, «saved_graph.txt»). Программа создаёт файл с матрицей смежности в указанном формате (Рисунок 18).

```
Имя файла: saved_graph.txt
Граф сохранён в файл saved_graph.txt (формат матрицы)
Файл можно открыть в любом текстовом редакторе
```

Рисунок 18 – Сохранение файла с матрицей смежности

Для проверки успешного сохранения открываю созданный файл (Рисунок 19).

```
saved_graph.txt – Блокнот
Файл Правка Формат Вид Справка
Matrix 5 directed
5 INF INF INF 1
INF INF INF INF INF
-6 INF 5 INF INF
INF INF -1 INF INF
INF INF INF INF INF

=== РЕЗУЛЬТАТЫ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ ===
Стартовая вершина: 0
Кратчайшие расстояния:
  до вершины 0: 0
  до вершины 1: недостижима
  до вершины 2: недостижима
  до вершины 3: недостижима
  до вершины 4: 1
```

Рисунок 19 – Содержимое файла в папке проекта

Все данные графа сохранены корректно. Тест пройден успешно.

5.5. Просмотр матрицы смежности

Для просмотра текущего графа выбираю пункт №6 — «Показать граф (матрица смежности)» (Рисунок 20).

```
Выберите: 6
=== МАТРИЦА СМЕЖНОСТИ ===
Вершин: 5
Рёбер: 5
Тип: Ориентированный

      0   1   2   3   4
-----
0 |    5  INF  INF  INF  1
1 |   INF  INF  INF  INF  INF
2 |   -6  INF   5  INF  INF
3 |   INF  INF  -1  INF  INF
4 |   INF  INF  INF  INF  INF

INF - отсутствие ребра (в т.ч. отсутствие петли)
Число - вес ребра (в т.ч. вес петли)
```

Рисунок 20 – Выбор пункта «Показать граф (матрица смежности)»

Программа выводит матрицу смежности с весами рёбер, где «INF» обозначает отсутствие ребра. Отображённая матрица полностью соответствует текущему графу. Тест пройден успешно.

5.6. Результаты тестирования

Результаты тестирования программы приведены в таблице 2.

Таблица 2 – Результаты тестирования

№	Описание теста	Полученный результат
1	Запуск программы	Верно
2	Создание графа	Верно

Продолжение таблицы 2

№	Описание теста	Полученный результат
3	Загрузка графа из файла	Верно
4	Поиск кратчайших путей	Верно
5	Сохранение графа в файл	Верно
6	Просмотр матрицы смежности	Верно

Все тесты выполнены успешно, программа работает корректно и соответствует требованиям, указанным в постановке задачи.

Заключение

При выполнении данной курсовой работы были улучшены навыки разработки программ и усвоены принципы работы со взвешенными графами. Приобретены практические навыки по реализации алгоритма Форда-Беллмана. Были закреплены умения написания программ объектно-ориентированной направленности, а также навыки программирования на языке C#.

В рамках курсовой работы была написана программа — консольная реализация алгоритма Форда-Беллмана. В ней можно создать граф (случайно или с заданием параметров), загрузить из файла, найти кратчайшие пути от указанной вершины, а также сохранить результаты и просмотреть граф в виде матрицы смежности.

Программа имеет базовый, но достаточный функционал для изучения и применения алгоритма Форда-Беллмана на практике.

Список литературы

1. Microsoft Docs. Документация по C# и .NET [Электронный ресурс]. — URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 8.12.2025 – 13.12.2025).
2. Wikipedia. Алгоритм Форда — Беллмана [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/Алгоритм_Форда_—_Беллмана (дата обращения: 10.12.2025 – 18.12.2025).
3. GeeksforGeeks. Bellman–Ford Algorithm [Электронный ресурс]. — URL: <https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/> (дата обращения: 10.12.2025 – 18.12.2025).

Приложение А

Program.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;

class Program
{
    static List<Edge> edges = new List<Edge>();
    static int verticesCount = 0;
    static bool isDirected = false;
    static string currentFileName = "";

    class Edge
    {
        public int From;
        public int To;
        public int Weight;
    }

    static void Main()
    {
        while (true)
        {
            ShowMenu();
        }
    }

    static void ShowMenu()
```

```

{

    Console.WriteLine("=== АЛГОРИТМ ФОРДА-БЕЛЛМАНА ===");
    Console.WriteLine($"Текущий граф: {edges.Count} рёбер,
{verticesCount} вершин");
    Console.WriteLine($"Тип: {(isDirected ?
"Ориентированный" : "Неориентированный"}}");
    if (!string.IsNullOrEmpty(currentFileName))
        Console.WriteLine($"Файл: {currentFileName}");
    Console.WriteLine("=====");
};

    Console.WriteLine("1. Случайная генерация графа ");
    Console.WriteLine("2. Ввести граф вручную (с
параметрами)");
    Console.WriteLine("3. Загрузить из файла");
    Console.WriteLine("4. Найти кратчайшие пути");
    Console.WriteLine("5. Сохранить в файл (матрица)");
    Console.WriteLine("6. Показать граф (матрица
смежности)");
    Console.WriteLine("7. Очистить граф");
    Console.WriteLine("8. Выход");
    Console.WriteLine("\nВыберите: ");

    string choice = Console.ReadLine();

    switch (choice)
    {
        case "1": GenerateRandomSimple(); Wait(); break;
        case "2": InputManualWithParams(); Wait(); break;
        case "3": LoadFromFile(); Wait(); break;
        case "4": FindPaths(); Wait(); break;
        case "5": SaveToFileMatrix(); Wait(); break;
        case "6": ShowGraphMatrix(); Wait(); break;
        case "7": ClearGraph(); Wait(); break;
    }
}

```

```

        case "8": Environment.Exit(0); break;
    }
}

static void Wait()
{
    Console.Write("\nНажмите любую клавишу для возврата в
меню...");
    Console.ReadKey();
}

static void GenerateRandomSimple()
{
    Console.Write("\nКоличество вершин: ");
    verticesCount = int.Parse(Console.ReadLine());

    Console.Write("Ориентированный (y/n): ");
    isDirected = Console.ReadLine().ToLower() == "y";

    Random rnd = new Random();
    edges.Clear();

    int probability = 30;
    int maxWeight = 10;

    int generatedEdges = 0;

    for (int i = 0; i < verticesCount; i++)
    {
        for (int j = 0; j < verticesCount; j++)
        {
            if (i == j) continue;

```

```

        if (rnd.Next(100) >= probability) continue;

        int weight = rnd.Next(-maxWeight, maxWeight + 1);
        if (weight == 0) weight = 1;

        edges.Add(new Edge { From = i, To = j, Weight =
weight });

        generatedEdges++;

        if (!isDirected)
        {
            edges.Add(new Edge { From = j, To = i, Weight =
weight });

            generatedEdges++;
        }
    }

    if (rnd.Next(100) < probability)
    {
        int weight = rnd.Next(-maxWeight, maxWeight + 1);
        if (weight == 0) weight = 1;
        edges.Add(new Edge { From = i, To = i, Weight =
weight });

        generatedEdges++;
    }
}

Console.WriteLine($"\\n Сгенерирован граф с
{verticesCount} вершинами");

Console.WriteLine($" Добавлено {generatedEdges} рёбер
(включая петли)");

```

```

        Console.WriteLine($" Автоматические параметры:
вероятность ребра {probability}%, вес от {-maxWeight} до
{maxWeight}");
        currentFileName = "";
    }

static void InputManualWithParams()
{
    Console.Write("\nКоличество вершин: ");
    verticesCount = int.Parse(Console.ReadLine());

    Console.Write("Ориентированный (y/n): ");
    isDirected = Console.ReadLine().ToLower() == "y";

    Console.Write("Вероятность ребра (0-100%): ");
    int probability = int.Parse(Console.ReadLine());

    Console.Write("Максимальный вес (включая
отрицательные): ");
    int maxWeight = int.Parse(Console.ReadLine());

    Console.Write("Минимальное количество рёбер (0 - без
ограничений): ");
    int minEdges = int.Parse(Console.ReadLine());

    Random rnd = new Random();
    edges.Clear();

    int generatedEdges = 0;

    for (int i = 0; i < verticesCount; i++)
    {

```

```

for (int j = 0; j < verticesCount; j++)
{
    if (i == j) continue;

    if (rnd.Next(100) >= probability) continue;

    int weight = rnd.Next(-maxWeight, maxWeight + 1);
    if (weight == 0) weight = 1;

    edges.Add(new Edge { From = i, To = j, Weight =
weight });

    generatedEdges++;

    if (!isDirected)
    {
        edges.Add(new Edge { From = j, To = i, Weight =
weight });

        generatedEdges++;
    }
}

if (rnd.Next(100) < probability)
{
    int weight = rnd.Next(-maxWeight, maxWeight + 1);
    if (weight == 0) weight = 1;
    edges.Add(new Edge { From = i, To = i, Weight =
weight });

    generatedEdges++;
}
}

```

```

        Console.WriteLine($"\\n✓ Граф сгенерирован с
параметрами:");
        Console.WriteLine($"    Вершин: {verticesCount}");
        Console.WriteLine($"    Рёбер: {generatedEdges}");
        Console.WriteLine($"    Тип: {(isDirected ?
"Ориентированный" : "Неориентированный"}}");
        Console.WriteLine($"    Вероятность ребра: {probability}
%");
        Console.WriteLine($"    Диапазон весов: [{-maxWeight},
{maxWeight}]");

        currentFileName = "";
    }

    static void LoadFromFile()
    {
        Console.Write("\\nИмя файла: ");
        string filename = Console.ReadLine();

        if (!File.Exists(filename))
        {
            Console.WriteLine("Файл не найден!");
            return;
        }

        try
        {
            edges.Clear();
            string[] lines = File.ReadAllLines(filename);

            if (lines.Length == 0)
            {
                Console.WriteLine("Файл пуст!");
            }
        }
    }
}

```

```

        return;
    }

    bool isMatrixFormat =
lines[0].StartsWith("Matrix");

    if (isMatrixFormat)
    {
        LoadMatrixFormat(lines);
    }
    else
    {
        Console.WriteLine("Ошибка: неподдерживаемый
формат файла. Ожидался формат 'Matrix'.");
        return;
    }

    currentFileName = filename;
    Console.WriteLine($"Загружено {edges.Count} рёбер
из файла {filename}");
}
catch (Exception ex)
{
    Console.WriteLine($"Ошибка: {ex.Message}");
}

static void LoadMatrixFormat(string[] lines)
{

    string[] firstLine = lines[0].Split(' ');
    verticesCount = int.Parse(firstLine[1]);

```

```

        isDirected = firstLine[2] == "directed";

        for (int i = 1; i <= verticesCount; i++)
        {
            string[] row = lines[i].Split(' ',
StringSplitOptions.RemoveEmptyEntries);

            for (int j = 0; j < verticesCount; j++)
            {
                if (row[j] != "INF" && row[j] != "0")
                {
                    int weight = int.Parse(row[j]);
                    if (weight != 0)
                    {
                        edges.Add(new Edge { From = i - 1, To = j,
Weight = weight });
                    }
                }
            }
        }
    }

    static void SaveToFileMatrix()
    {
        if (edges.Count == 0)
        {
            Console.WriteLine("\nГраф пуст! Сначала создайте
граф.");
            return;
        }
    }

```

```

        Console.Write("\nИмя файла: ");
        string filename = Console.ReadLine();

        try
        {
            string[,] matrix = new string[verticesCount,
verticesCount];

            for (int i = 0; i < verticesCount; i++)
            {
                for (int j = 0; j < verticesCount; j++)
                {
                    matrix[i, j] = "INF";
                }
            }

            foreach (var edge in edges)
            {
                matrix[edge.From, edge.To] =
edge.Weight.ToString();
            }

            using (StreamWriter writer = new
StreamWriter(filename))
            {
                writer.WriteLine($"Matrix {verticesCount}
{(isDirected ? "directed" : "undirected")}");

                for (int i = 0; i < verticesCount; i++)

```

```

        {
            StringBuilder row = new StringBuilder();
            for (int j = 0; j < verticesCount; j++)
            {
                row.Append(matrix[i, j]);
                if (j < verticesCount - 1) row.Append(" ");
            }
            writer.WriteLine(row.ToString());
        }

        writer.WriteLine("\n=== РЕЗУЛЬТАТЫ ПОИСКА
КРАТЧАЙШИХ ПУТЕЙ ===");

        Console.Write("\nДля какой вершины сохранить
результаты поиска? (0-" + (verticesCount - 1) + "): ");
        int startVertex = int.Parse(Console.ReadLine());

        if (startVertex >= 0 && startVertex <
verticesCount)
        {

            int[] dist = new int[verticesCount];
            for (int i = 0; i < verticesCount; i++)
                dist[i] = int.MaxValue;
            dist[startVertex] = 0;

            for (int i = 0; i < verticesCount - 1; i++)
            {
                foreach (var edge in edges)
                {

```

```

        if (dist[edge.From] != int.MaxValue &&
            dist[edge.From] + edge.Weight <
dist[edge.To])
        {
            dist[edge.To] = dist[edge.From] +
edge.Weight;
        }
    }
}

```

```

bool hasNegativeCycle = false;
foreach (var edge in edges)
{
    if (dist[edge.From] != int.MaxValue &&
        dist[edge.From] + edge.Weight <
dist[edge.To])
    {
        hasNegativeCycle = true;
        break;
    }
}

```

```

writer.WriteLine($"Стартовая вершина:
{startVertex}");

```

```

if (hasNegativeCycle)
{
    writer.WriteLine("Обнаружен отрицательный
цикл!");
}
else
{

```

```

        writer.WriteLine("Кратчайшие
расстояния:");

        for (int i = 0; i < verticesCount; i++)
        {
            if (dist[i] == int.MaxValue)
                writer.WriteLine($" до вершины {i}:
недостижима");
            else
                writer.WriteLine($" до вершины {i}:
{dist[i]}");
        }
    }
    else
    {
        writer.WriteLine("Неверный номер вершины.");
    }
}

currentFileName = filename;
Console.WriteLine($"Граф и результаты сохранены в
файл {filename} (формат матрицы)");
Console.WriteLine($"Файл можно открыть в любом
текстовом редакторе");
}
catch (Exception ex)
{
    Console.WriteLine($"Ошибка: {ex.Message}");
}
}

static void FindPaths()
{

```

```

        if (edges.Count == 0)
        {
            Console.WriteLine("\nГраф пуст! Сначала создайте
граф.");
            return;
        }

        Console.Write("\nСтартовая вершина (0- " +
(verticesCount - 1) + "): ");
        int start = int.Parse(Console.ReadLine());

        if (start < 0 || start >= verticesCount)
        {
            Console.WriteLine("Неверный номер вершины!");
            return;
        }

        int[] dist = new int[verticesCount];
        for (int i = 0; i < verticesCount; i++)
            dist[i] = int.MaxValue;
        dist[start] = 0;

        for (int i = 0; i < verticesCount - 1; i++)
        {
            foreach (var edge in edges)
            {
                if (dist[edge.From] != int.MaxValue &&
dist[edge.From] + edge.Weight < dist[edge.To])
                {
                    dist[edge.To] = dist[edge.From] + edge.Weight;
                }
            }
        }
    }
}

```

```

    }

    bool hasNegativeCycle = false;
    foreach (var edge in edges)
    {
        if (dist[edge.From] != int.MaxValue &&
            dist[edge.From] + edge.Weight < dist[edge.To])
        {
            hasNegativeCycle = true;
            break;
        }
    }

    Console.WriteLine("\n=== РЕЗУЛЬТАТЫ ===");

    if (hasNegativeCycle)
    {
        Console.WriteLine("ВНИМАНИЕ: Обнаружен отрицательный цикл!");
        Console.WriteLine("Кратчайшие пути не существуют (расстояния не определены).");
        return;
    }

    Console.WriteLine($"Из вершины {start}:");
    for (int i = 0; i < verticesCount; i++)
    {
        if (dist[i] == int.MaxValue)
            Console.WriteLine($" до {i}: недостижима");
        else
            Console.WriteLine($" до {i}: {dist[i]}");
    }

```

```

    }

    static void ShowGraphMatrix()
    {
        if (edges.Count == 0)
        {
            Console.WriteLine("\nГраф пуст!");
            return;
        }

        Console.WriteLine($"\\n=== МАТРИЦА СМЕЖНОСТИ ===");
        Console.WriteLine($"Вершин: {verticesCount}");
        Console.WriteLine($"Рёбер: {edges.Count}");
        Console.WriteLine($"Тип:  {(isDirected ?
"Ориентированный" : "Неориентированный")})");
        Console.WriteLine();

        string[,] matrix = new string[verticesCount,
verticesCount];

        for (int i = 0; i < verticesCount; i++)
        {
            for (int j = 0; j < verticesCount; j++)
            {
                matrix[i, j] = "INF";
            }
        }

        foreach (var edge in edges)
        {

```

```

matrix[edge.From, edge.To] =
edge.Weight.ToString();
    }

    Console.Write("    ");
    for (int j = 0; j < verticesCount; j++)
    {
        Console.Write($"{j,5}");
    }
    Console.WriteLine();

    Console.Write("    ");
    for (int j = 0; j < verticesCount; j++)
    {
        Console.Write("-----");
    }
    Console.WriteLine();

    for (int i = 0; i < verticesCount; i++)
    {
        Console.Write($"{i,2} | ");
        for (int j = 0; j < verticesCount; j++)
        {
            Console.Write($"{matrix[i, j],5}");
        }
        Console.WriteLine();
    }

    Console.WriteLine("\nINF - отсутствие ребра (в т.ч.
отсутствие петли)");
    Console.WriteLine("Число - вес ребра (в т.ч. вес
петли)");

```

```
}

static void ClearGraph()
{
    edges.Clear();
    verticesCount = 0;
    currentFileName = "";
    Console.WriteLine("\nГраф очищен!");
}
}
```