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

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовому проектированию
По курсу «Логика и основы алгоритмизации
в инженерных задачах»
на тему «Реализация алгоритма Форда-Фалкерсона для нахождения максимального потока»

Выполнил:
Студент группы 24ВВВ3
Плотников И.А.

Приняла:
к.т.н. доцент Юрова О. В.

25.12.25
отлично

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 в инженерных задачах

Студенту Татьянкову Ивану Алексеевичу Группа 24ВВВ3

Тема_проекта Реализация алгоритма Форда-Фалкерсона для нахождения максимального потока.

Исходные данные (технические требования) на проектирование

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

Пояснительная записка должна содержать

1. Постановку задачи.
2. Теоретическую часть задания.
3. Описание алгоритма поставленной задачи
4. Пример ручного расчёта задачи и вычислений (на небольшом участке работы алгоритма).
5. Описание самой программы
6. Тесты
7. Список литературы
8. Листинг программы
9. Результаты работы программы.

Объем работы по курсу

1. Расчетная часть

Ручной расчёт работы программы

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 алгоритма программы
3 Разработка программы
4 Тестирование и завершение разработки
5 Оформление пояснительной записки
6
7
8

Дата выдачи задания "12" сентября 2025

Дата защиты проекта "___" _____

Руководитель Юдова О.К.

Задание получил "12" сентября 2025 г.

Студент Тютников Иван Алексеевич

Содержание

Реферат **5**

Введение **6**

1. Постановка задачи **7**

2. Теоретическая часть задания **9**

2.1. Алгоритм генерации графа **9**

2.2. Алгоритмы поиска пути **11**

2.3. Теория графов в контексте транспортных сетей **13**

3. Описание программы **14**

3.1. Основные структуры и переменные программы **14**

3.2. Функции **14**

3.3. Алгоритм BFS **18**

3.4. Алгоритм Форда-Фалкерсона **20**

3.5. Управление и режимы работы **22**

3.6. Сохранение и загрузка транспортной сети **24**

4. Тестирование **26**

4.1. Тестирование программы **33**

4.2. Результаты тестирования **37**

5. Ручной расчёт задачи **38**

Заключение **41**

Список литературы **43**

Приложение А. Листинг программы **44**

Реферат

Отчёт содержит 60 страниц, 9 рисунков, 2 таблицы, 5 источников.

ГРАФ, ТЕОРИЯ ГРАФОВ, C++, АЛГОРИТМ ФОРДА_ФАЛКЕРСОНА, ТРАНСПОРТНАЯ СЕТЬ, МАКСИМАЛЬНЫЙ ПОТОК

Цель работы — создание многофункционального приложения, предоставляющего пользователю возможность генерации транспортных сетей различной конфигурации и нахождения их максимального потока, а также сохранения и загрузки сетей и результатов работы программы.

В процессе разработки были изучены и реализованы следующие алгоритмы: генерация графа, метод поиска в глубину (DFS), поиск кратчайшего пути алгоритмом BFS. Особое внимание уделено реализации алгоритма Форда-Фалкерсона и созданию интуитивно понятного пользовательского интерфейса.

Введение

Задачи, связанные с нахождением максимального потока в транспортной сети, представляют собой один из краеугольных камней теории графов и дискретной оптимизации. Эти задачи имеют широчайший спектр практических приложений: от очевидных — моделирования транспортных перевозок, управления потоками данных в компьютерных сетях и трубопроводных систем — до более нетривиальных, таких как планирование проектов, задача о паросочетаниях, анализ пропускной способности сетей связи и даже в биоинформатике.

Алгоритм Форда-Фалкерсона, предложенный в 1956 году, является классическим и фундаментальным методом решения данной проблемы. Его изучение и реализация имеют не только теоретическую, но и большую практическую ценность, так как он лежит в основе многих более сложных и специализированных алгоритмов (например, алгоритма Эдмондса-Карпа или Диница). Понимание его принципов, достоинств и ограничений критически важно для любого специалиста в области компьютерных наук, прикладной математики и исследования операций.

Алгоритм Форда-Фалкерсона детально проанализирован в классической литературе по теории графов и алгоритмам (Кормен, Ахо, Тарьян и др.). Существует множество его модификаций, направленных на улучшение асимптотической сложности и устранение недостатков базовой версии. Однако именно базовый алгоритм остается отправной точкой для изучения теории потоков в сетях благодаря наглядности и относительной простоте его идеи.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudioCommunity 2026, язык программирования – C++, Программа обладает всеми средствами необходимыми при разработке и отладке программы.

1. Постановка задачи

Транспортная сеть представляется в виде взвешенного графа, где вершины соответствуют пунктам передачи ресурсов, а рёбра — транспортным маршрутам с ограниченной пропускной способностью. Каждое ребро характеризуется максимальным количеством ресурсов, которые могут быть переданы по данному маршруту в единицу времени. Граф может быть ориентированным (направленные потоки) или неориентированным (двусторонние маршруты). Каждая вершина имеет уникальный идентификатор (номер), а пропускные способности рёбер задаются в виде матрицы смежности.

Для генерации случайной транспортной сети используется алгоритм со следующими параметрами: количество вершин, тип графа, вероятность наличия ребра между вершинами, диапазон значений пропускных способностей. Алгоритм начинается с создания нулевой матрицы, затем для каждой возможной пары вершин (исключая петли) с заданной вероятностью добавляется ребро со случайной пропускной способностью. Для неориентированных графов матрица симметризуется. При генерации ориентированного графа дополнительно регулируется количество взаимных рёбер для повышения реалистичности сети.

Транспортная сеть может быть сгенерирована программой с заданными параметрами или загружена из текстового файла. Файл должен содержать количество вершин, тип графа и матрицу пропускных способностей в читаемом формате с комментариями. После получения сети пользователь выбирает начальную точку (исток, source) и конечную точку (сток, sink) — вершины, между которыми требуется найти максимальный поток. Программа автоматически предлагает варианты выбора, основываясь на анализе исходящих и входящих рёбер вершин, но пользователь может задать вершины вручную.

Для нахождения максимального потока применяется алгоритм Форда-Фалкерсона. Процесс начинается с нулевого потока по всем рёбрам. На каждом шаге алгоритма с помощью поиска в ширину (BFS) находится увеличивающий путь от истока к стоку в остаточной сети. Для найденного пути определяется минимальная остаточная пропускная способность (минимальное значение среди

рёбер пути). Поток вдоль этого пути увеличивается на найденную величину, при этом в остаточной сети обновляются пропускные способности: для прямых рёбер значение уменьшается, для обратных — увеличивается (что позволяет алгоритму "передумывать" и перенаправлять поток). Процесс продолжается до тех пор, пока в остаточной сети существует путь от истока к стоку. Результирующий суммарный поток является максимальным возможным при заданных ограничениях.

В процессе работы алгоритма программа отображает пошаговую информацию: номер итерации, найденный увеличивающий путь, величину потока на этом пути и текущее суммарное значение максимального потока. По завершении вычислений выводится итоговый максимальный поток между выбранными вершинами, а также проводится анализ графа: подсчитывается общее количество рёбер, суммарная пропускная способность всех рёбер, средняя пропускная способность ребра.

Программа должна обеспечивать возможность сохранения текущей транспортной сети в файл (как до, так и после проведения расчётов), а также сохранения результатов расчёта (параметры графа, выбранные вершины, значение максимального потока) в отдельный файл отчёта. После завершения одного расчёта пользователь может запустить новый без перезапуска программы, при этом сохраняется возможность выбора режима работы (генерация новой сети или загрузка из файла).

Управление программой осуществляется через консольный интерфейс с поддержкой русского языка. Интерфейс должен быть интуитивно понятным, обеспечивать визуальную обратную связь и корректную обработку ошибочного ввода. Программа должна быть кроссплатформенной (работать в операционных системах Windows и Linux) и использовать только стандартные библиотеки C++.

Программа должна быть реализована в соответствии с вариантом №28 и включать все указанные функциональные возможности, обеспечивая корректную работу алгоритма Форда-Фалкерсона для различных конфигураций сетей.

2. Теоретическая часть задания

2.1.Алгоритм генерации графа

Транспортная сеть моделируется как взвешенный ориентированный граф. Вершины соответствуют узлам сети (городам, терминалам), а рёбра — транспортным путям с ограниченной пропускной способностью.

Алгоритм вероятностной генерации сети работает следующим образом: выбирается количество вершин n и тип графа (ориентированный/неориентированный). Для каждой возможной пары вершин (i, j), где $i \neq j$, с вероятностью 70% создаётся ребро со случайной пропускной способностью от 1 до 100. Если генерируется неориентированный граф, то для каждой созданной связи (i, j) автоматически создаётся симметричное ребро (j, i) с той же пропускной способностью. Для ориентированных графов выполняется дополнительный шаг: если между парой вершин существуют оба взаимных ребра (i, j) и (j, i), то с вероятностью 50% одно из них удаляется — это имитирует реальные транспортные системы с односторонним движением.

Процесс продолжается до тех пор, пока не будут обработаны все возможные пары вершин. В результате формируется сеть со случайной структурой связей и разнородными пропускными способностями, что соответствует реальным транспортным системам с альтернативными маршрутами и разной пропускной способностью путей.

Такой подход позволяет генерировать сети различной плотности и сложности, обеспечивая тестовые данные для проверки алгоритмов поиска максимального потока. При фиксированном начальном значении генератора случайных чисел алгоритм производит одинаковые сети, что удобно для отладки и воспроизводимости результатов. Ниже приведён пример сгенерированной транспортной сети, представленный матрицей пропускных способностей(Рисунок 1):

Матрица пропускных способностей:

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
[0]	0	20	0	32	0	16	0	43	0	32	0	9	88
[1]	53	0	0	82	76	0	0	0	0	72	0	87	76
[2]	6	80	0	86	0	38	0	0	0	17	0	62	0
[3]	49	0	0	0	0	49	52	0	0	0	11	0	3
[4]	39	0	84	56	0	0	0	0	0	0	0	0	47
[5]	0	55	88	9	0	0	41	86	0	0	72	74	97
[6]	9	0	63	75	42	0	0	20	0	100	0	0	0
[7]	0	18	26	18	0	52	0	0	99	0	0	0	26
[8]	24	0	21	40	54	0	42	0	0	0	89	99	49
[9]	69	0	0	0	96	85	98	19	0	0	0	0	38
[10]	62	91	95	51	5	72	0	76	6	80	0	0	0
[11]	0	84	48	12	10	9	55	4	0	0	99	0	0
[12]	88	92	41	0	0	70	74	6	0	6	61	0	0

Рисунок 1 – Пример транспортной сети

2.2.Алгоритмы поиска пути

BFS — алгоритм поиска в ширину, применяемый для нахождения увеличивающих путей в остаточной сети при реализации алгоритма Форда-Фалкерсона.

Алгоритм реализует следующий процесс:

– Начинается с вершины-истока, которая помещается в очередь и отмечается как посещённая.
– Пока очередь не пуста, извлекается текущая вершина.
– Для каждой соседней вершины, достижимой по ребру с остаточной пропускной способностью больше нуля и ещё не посещённой, выполняется:
– Вершина добавляется в очередь.
– Отмечается как посещённая.
– Запоминается "родительская" вершина для восстановления пути.
– Процесс продолжается до тех пор, пока не будет достигнута вершина-сток или очередь не опустеет (если пути не существует).

Особенности реализации:

– В отличие от классического BFS, учитывается не просто наличие связи, а остаточная пропускная способность ребра.
– Восстановление пути от стока к истоку позволяет определить увеличивающий путь для алгоритма Форда-Фалкерсона.
– Гарантирует нахождение кратчайшего пути по числу рёбер, что важно для эффективности основного алгоритма.

Сложность составляет O(V+E) для одного запуска, где V — количество вершин, E — количество рёбер.

Алгоритм Форда-Фалкерсона — основной алгоритм программы, решающий задачу нахождения максимального потока в транспортной сети. Работает на основе концепции остаточных сетей и увеличивающих путей.

Алгоритм реализует следующий процесс:

1. Инициализация потока нулевыми значениями на всех рёбрах.

2. Пока в остаточной сети существует путь от истока к стоку (найденный с помощью BFS):
3. Находится минимальную остаточную пропускную способность на этом пути.
    3.1. Увеличивает поток вдоль всего пути на найденную величину.
    3.2. Обновляет остаточную сеть:
    3.3. Уменьшает пропускную способность прямых рёбер на величину потока.
    3.4. Увеличивает пропускную способность обратных рёбер на ту же величину.
    3.5. Когда увеличивающих путей больше не остаётся, найденный поток является максимальным.

Ключевые особенности:

– Остаточная сеть — динамически изменяемая структура, где каждый раз заново вычисляются достижимые вершины.
– Увеличивающие пути: На каждом шаге ищется путь, по которому можно увеличить суммарный поток.
– Обратные рёбра: Критический элемент, позволяющий алгоритму корректировать ранее сделанные выборы и гарантировать оптимальность.
– Сходимость: Гарантированно находит максимальный поток для сетей с целочисленными пропускными способностями.

Алгоритм основан на теореме Форда-Фалкерсона, которая утверждает, что поток максимален тогда и только тогда, когда в остаточной сети нет пути от истока к стоку. Это эквивалентно теореме о максимальном потоке и минимальном разрезе.

2.3.Теория графов в контексте транспортных сетей

Транспортная сеть представляется в виде графа, где города или терминалы — это вершины, а дороги или пути — это рёбра с указанной пропускной способностью. Граф может быть ориентированным (одностороннее движение) или неориентированным (двустороннее движение).

В программе граф содержит от 4 до 10 вершин. Между каждой парой вершин ребро создаётся с вероятностью 70%. Если ребро существует, ему присваивается случайная пропускная способность от 1 до 100. Петли (пути из вершины в саму себя) не создаются.

Граф хранится как матрица смежности — таблица, где на пересечении строки i и столбца j записана пропускная способность пути из вершины i в вершину j. Если пути нет, стоит 0. Для неориентированного графа эта таблица симметрична.

Задача о максимальном потоке ставится так: выбраны две вершины — исток (начальная точка) и сток (конечная точка). Нужно найти, какой максимальный объём груза можно перевезти из истока в сток через сеть, учитывая ограничения пропускной способности каждой дороги и правило сохранения груза на перекрёстках.

Решение основывается на теореме: поток максимален тогда, когда в оставшейся после его распределения сети нельзя найти ни одного пути от истока к стоку.

3. Описание программы

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudioCommunity 2026, язык программирования – C++, Программа обладает всеми средствами необходимыми при разработке и отладке программы.

Далее будут приведены описания функций, структур и действия элементов управления, используемых в программе.

3.1.Основные структуры и переменные программы

Программа использует следующие основные структуры данных и переменные:

Переменные для хранения графа:

- int numG — количество вершин графа (от 4 до 10)
- int** graph — двумерный массив (матрица смежности) для хранения пропускных способностей рёбер
- bool isOriented — флаг, определяющий тип графа

Вспомогательные переменные:

- int source, sink — номера вершин-истока и стока
- int max_flow — значение максимального потока
- bool restartProgram — флаг для управления повторным запуском программы
- bool loadedFromFile — флаг, указывающий, был ли граф загружен из файла
- Массивы для алгоритма:
- int** rGraph — остаточный граф для алгоритма Форда-Фалкерсона
- int* parent — массив предков для восстановления пути в BFS
- bool* visited — массив пометок посещённых вершин в BFS

3.2. Функции

П рограмма включает в себя набор ключевых функций, обеспечивающих корректную работу программы, управление интерфейсом и работу с файлами.

int main() — главная функция программы, которая:

- Инициализирует генератор случайных чисел
- Реализует основной цикл с возможностью перезапуска
- Координирует взаимодействие всех модулей программы
- Обрабатывает выбор режима работы (генерация/загрузка графа)

int fordFulkerson(int** graph, int numG, int s, int t) — основная функция алгоритма, которая:

- Создаёт и инициализирует остаточный граф
- Использует BFS для поиска увеличивающих путей
- Находит минимальную пропускную способность на пути
- Обновляет остаточный граф (прямые и обратные рёбра)
- Суммирует потоки и возвращает максимальное значение

bool bfs(int** rGraph, int numG, int s, int t, int parent[]) — функция поиска в ширину, которая:

- Ищет путь от истока к стоку в остаточном графе
- Использует очередь для обхода вершин
- Заполняет массив предков для восстановления пути
- Возвращает true, если путь найден

void printMatrix(int** Matrix, int numG) — функция вывода матрицы пропускных способностей:

- Форматирует вывод матрицы с заголовками строк и столбцов
- Выводит нули как "0" для наглядности
- Обеспечивает читаемое представление графа

bool saveGraphToFile(int** graph, int numG, bool isOriented, const string& filename) — функция сохранения графа:

- Создаёт файл с описанием формата
- Сохраняет параметры графа (количество вершин, тип)
- Записывает матрицу пропускных способностей в читаемом виде

bool loadGraphFromFile(int**& graph, int& numG, bool& isOriented, const string& filename) — функция загрузки графа:

- Открывает и читает файл с графом
- Парсит комментарии для получения параметров

- Загружает матрицу пропускных способностей
- Проверяет корректность загруженных данных

bool saveResultToFile(int source, int sink, int max_flow, int numG, bool isOriented, const string& filename) — функция сохранения результатов:

- Создаёт файл отчёта с результатами расчёта
- Сохраняет параметры графа и выбранные вершины
- Записывает значение максимального потока
- Добавляет дату и время расчёта

int isInteger(const string& message) — функция безопасного ввода целых чисел:

- Выводит переданное сообщение-подсказку
- Проверяет корректность введённого значения
- Зацикливается при ошибочном вводе
- Возвращает корректное целое число

char safeCharInput(const string& message) — функция безопасного ввода символов:

- Очищает буфер ввода перед чтением
- Считывает один символ с клавиатуры
- Очищает оставшийся буфер
- Возвращает введённый символ

bool askToRestart() — функция запроса перезапуска программы:

- Спрашивает пользователя о повторном запуске
- Поддерживает ввод на русском и английском
- Возвращает true при согласии на перезапуск

string intToString(int value) — вспомогательная функция преобразования:

- Конвертирует целое число в строку
- Использует stringstream для совместимости
- Заменяет стандартную функцию to_string

void clearScreen() — функция очистки экрана:

- Использует системные команды для очистки консоли
- Кроссплатформенная реализация (Windows/Linux)

- Обеспечивает чистый интерфейс при перезапуске

Каждая функция выполняет одну конкретную задачу, что обеспечивает модульность, удобство тестирования и сопровождения кода. Функции следуют принципу единой ответственности и имеют чёткие интерфейсы с минимальными зависимостями.

3.3.Алгоритм BFS

Поиск увеличивающих путей в остаточной сети осуществляется с помощью алгоритма поиска в ширину (BFS), который гарантирует нахождение кратчайшего пути по количеству рёбер от истока к стоку.

Принцип работы алгоритма:

```
bfs(остаточный_граф, количество_вершин, исток, сток, массив_предков):
    создать массив посещённых вершин, заполнить false
    создать очередь вершин

    пометить исток как посещённый
    добавить исток в очередь
    установить предка истока = -1

    пока очередь не пуста:
        текущая_вершина = извлечь из очереди

        для каждой вершины v от 0 до количества_вершин-1:
            если v не посещена и остаточная_пропускная_способность(текущая_вершина, v) > 0:
                пометить v как посещённую
                установить предка v = текущая_вершина
                добавить v в очередь

                если v == сток:
                    вернуть true (путь найден)

    вернуть false (путь не найден)
```

Ключевые особенности алгоритма в контексте задачи о максимальном потоке:

- Остаточная пропускная способность: В отличие от классического BFS, алгоритм проверяет не просто наличие ребра, а положительную остаточную пропускную способность rGraph[u][v] > 0.
- Поиск кратчайшего пути: BFS находит путь с минимальным количеством рёбер, что ускоряет сходимость алгоритма Форда-Фалкерсона.

– Восстановление пути: Массив предков parent[] позволяет восстановить найденный путь от стока к истоку после завершения поиска.
– Отсутствие весов: BFS работает с ненагруженными рёбрами (учитывается только факт наличия положительной остаточной пропускной способности), что упрощает реализацию.

Визуализация процесса поиска:

Алгоритм последовательно исследует вершины, начиная от истока:

- Уровень 0: исток
- Уровень 1: все вершины, достижимые из истока напрямую
- Уровень 2: все вершины, достижимые из вершин уровня 1
- И так далее, пока не будет достигнут сток или не будут исследованы все достижимые вершины.

Такой подход гарантирует, что первый найденный путь будет кратчайшим по числу промежуточных вершин, что минимизирует количество обновлений остаточного графа на каждой итерации алгоритма Форда-Фалкерсона.

3.4.Алгоритм Форда-Фалкерсона

Нахождение максимального потока в транспортной сети осуществляется с помощью алгоритма Форда-Фалкерсона, который итеративно увеличивает поток вдоль увеличивающих путей в остаточной сети.

Принцип работы алгоритма:

```
fordFulkerson(граф, количество_вершин, исток, сток):
    создать остаточный_граф и скопировать в него исходный граф
    инициализировать максимальный_поток = 0

    пока существует путь от истока к стоку в остаточном_графе (найденный через BFS):
        // Найти минимальную пропускную способность на пути
        путь_поток = бесконечность
        для вершины = сток; вершина != исток; вершина = предок[вершина]:
            предыдущая = предок[вершина]
            путь_поток = min(путь_поток, остаточный_граф[предыдущая][вершина])

        // Обновить остаточный граф вдоль найденного пути
        для вершина = сток; вершина != исток; вершина = предок[вершина]:
            предыдущая = предок[вершина]
            остаточный_граф[предыдущая][вершина] -= путь_поток
            остаточный_граф[вершина][предыдущая] += путь_поток

        // Добавить найденный поток к общему
        максимальный_поток += путь_поток

    вернуть максимальный_поток
```

Ключевые особенности алгоритма:

1. Остаточная сеть: На каждой итерации алгоритм работает с остаточным графом, который представляет собой исходную сеть с учётом уже распределённого потока.
2. Увеличивающие пути: Алгоритм ищет любой путь от истока к стоку в остаточной сети, по которому можно пропустить дополнительный поток.

3. Обратные рёбра: При обновлении остаточного графа создаются обратные рёбра, позволяющие "передумать" и перенаправить ранее распределённый поток, что гарантирует нахождение оптимального решения.
4. Критерий остановки: Алгоритм завершается, когда в остаточной сети больше не существует пути от истока к стоку, что согласно теореме Форда-Фалкерсона означает, что найден максимальный поток.

Визуализация процесса работы, при которой на каждой итерации алгоритм:

- Находит увеличивающий путь с помощью BFS
- Определяет "узкое место" пути — минимальную остаточную пропускную способность
- Увеличивает поток вдоль всего пути на эту величину
- Обновляет остаточный граф (уменьшает прямые рёбра, увеличивает обратные)

Такой итеративный подход гарантирует, что после завершения работы алгоритма будет найден максимально возможный поток между заданными вершинами при соблюдении ограничений пропускной способности всех рёбер сети.

3.5. Управление и режимы работы

Программа поддерживает несколько режимов работы с транспортной сетью:

1. Режим генерации случайного графа:
    1.1. Пользователь задаёт количество вершин (от 4 до 10)
    1.2. Выбирает тип графа (ориентированный или неориентированный)
    1.3. Программа автоматически генерирует сеть со случайными связями и пропускными способностями
2. Режим загрузки из файла:
    2.1. Пользователь указывает имя файла с сохранённой сетью
    2.2. Программа загружает матрицу пропускных способностей и параметры графа
    2.3. Осуществляется проверка корректности данных
3. Автоматический выбор истока и стока:
    3.1. Программа анализирует сгенерированную сеть
    3.2. Выбирает вершину с максимальным количеством исходящих рёбер в качестве истока
    3.3. Выбирает вершину с максимальным количеством входящих рёбер в качестве стока
    3.4. Пользователь может подтвердить или изменить выбор вручную
4. Режим ручного выбора вершин:
    4.1. Пользователь самостоятельно указывает номера вершин-истока и стока
    4.2. Осуществляется проверка корректности ввода
    4.3. Проверяется существование пути между выбранными вершинами
5. Режим расчёта максимального потока:
    5.1. Автоматическое выполнение алгоритма Форда-Фалкерсона
    5.2. Пошаговый вывод информации о найденных увеличивающих путях
    5.3. Отображение текущего значения потока на каждой итерации
    5.4. Вывод итогового максимального потока

Интерактивные возможности:

- Сохранение графа: Возможность сохранения сгенерированной сети в файл как до, так и после проведения расчётов
- Сохранение результатов: Запись параметров графа, выбранных вершин и результата расчёта в отдельный файл отчёта
- Анализ сети: Автоматический подсчёт характеристик графа (количество рёбер, суммарная пропускная способность, средняя пропускная способность)
- Повторный запуск: Возможность выполнения нового расчёта без перезапуска программы

Управление программой осуществляется через консольный интерфейс:

- Ввод числовых параметров через клавиатуру
- Выбор опций с помощью символов 'y'/'n' (да/нет)
- Навигация по меню с помощью цифрового выбора
- Обработка ошибочного ввода с повторными запросами

Во всех режимах отслеживается и выводится информация о параметрах сети, ходе выполнения алгоритма и результатах расчёта. Программа обеспечивает визуальную обратную связь через консольный вывод и поддерживает возможность детального анализа работы алгоритма.

3.6.Сохранение и загрузка транспортной сети

Программа обеспечивает возможность сохранения сгенерированной транспортной сети в файл и загрузки ранее сохранённой сети для дальнейшего анализа. Формат файла является текстовым и включает:

1. Параметры графа:
    1.1.Количество вершин
    1.2.Тип графа (ориентированный/неориентированный)
2. Матрицу пропускных способностей:
    2.1.Полное представление матрицы смежности в читаемом формате
    2.2.Каждая строка матрицы соответствует отдельной строке файла
    2.3.Значения разделены пробелами для удобства обработки
3. Дополнительные возможности:
    3.1.Сохранение графа как перед расчётом, так и после него
    3.2.Сохранение результатов расчёта в отдельный файл отчёта
    3.3.Возможность многократного использования сохранённых сетей

Формат файла графа:

```
# Формат файла графа для алгоритма Форда-Фалкерсона
# Количество вершин: 5
# Тип графа: ориентированный
# Матрица пропускных способностей:
0 45 0 78 0
0 0 32 0 91
19 0 0 0 64
0 67 0 0 22
53 0 88 14 0
```

Формат файла результатов:

```
==========================================
РЕЗУЛЬТАТЫ РАСЧЕТА МАКСИМАЛЬНОГО ПОТОКА
Алгоритм Форда-Фалкерсона
==========================================

ПАРАМЕТРЫ ГРАФА:
Количество вершин: 5
Тип графа: ориентированный
Исток (source): вершина 0
```

```
Сток (sink): вершина 4

РЕЗУЛЬТАТ РАСЧЕТА:
Максимальный поток: 127

==========================================
Дата расчета: May 15 2024 14:30:45
```

Особенности реализации:

- Сохранение графа перед расчётом: Позволяет сохранить исходную сеть для последующего анализа разных алгоритмов
- Сохранение графа после расчёта: Фиксирует состояние сети с учётом распределённого потока
- Файл отчёта: Содержит полную информацию о проведённом расчёте для документации
- Обработка ошибок: Программа корректно реагирует на отсутствие файлов, повреждённые данные и ошибки ввода-вывода
- Совместимость: Текстовый формат обеспечивает возможность обработки файлов в других программах

Таким образом, программа представляет собой комплексное решение для анализа транспортных сетей, объединяющее генерацию сетей, расчёт максимального потока и удобную систему сохранения результатов, что делает её эффективным инструментом для изучения алгоритмов теории графов.

4. Тестирование

В качестве среды разработки была выбрана программа Microsoft Visual Studio версии Community 2022. Данная среда разработки обладает всеми необходимыми инструментами для создания, отладки и тестирования программ. Для отладки использовались стандартные возможности Visual Studio: режим отладки с точками останова, трассировка выполнения, анализ содержимого переменных в режиме реального времени, мониторинг использования оперативной памяти.

Тестирование проводилось на протяжении всего процесса разработки по методологии итеративного тестирования: после реализации каждого ключевого модуля (генерации графа, алгоритма BFS, алгоритма Форда-Фалкерсона, функций работы с файлами) программа запускалась для проверки корректности работы и выявления потенциальных ошибок.

В ходе тестирования были выявлены и устранены следующие категории проблем:

- Проблемы с вводом данных: Изначальная реализация неадекватно реагировала на некорректный ввод (буквы вместо цифр, значения вне диапазона). Для решения была разработана функция isInteger() с цикличным запросом корректных данных.
- Ошибки очистки буфера ввода: При последовательных запросах ввода возникали проблемы с чтением данных из-за остатков в буфере. Добавлена функция safeCharInput() с принудительной очисткой буфера.
- Проблемы с выбором истока и стока: Автоматический выбор мог приводить к выбору несвязанных вершин. Реализована проверка исходящих/входящих рёбер и возможность ручной корректировки.
- Утечки памяти: При многократном перезапуске программы возникали утечки памяти. Добавлено корректное освобождение всех динамически выделенных массивов.

– Обработка файловых операций: Отсутствовала проверка успешности открытия файлов. Добавлены проверки возвращаемых значений функций работы с файлами.

Тестирование также включало проверку граничных условий:

- Минимальное количество вершин (4)
- Максимальное количество вершин (10)
- Графы без пути между выбранными вершинами
- Полностью связанные графы
- Графы с минимальными/максимальными пропускными способностями

Результаты тестирования подтвердили корректность работы всех компонентов программы, соответствие реализованных алгоритмов их теоретическим описаниям и устойчивость системы к различным типам входных данных. Программа демонстрирует стабильную работу в различных средах выполнения (Windows, Linux) при условии поддержки стандарта C++11.

Таблица 1 – Описание поведения программы при тестировании

№	Описание теста	Предусловие	Тестирование	Ожидаемый результат
1	Запуск программы	нет	Запуск программы из командной строки или IDE	Отображение заголовка программы и меню выбора режима работы с графом
2	Генерация случайного графа	Программа запущена	Выбор режима 1 → Ввод количества вершин (например, 5) → Выбор типа графа	Создание матрицы пропускных способностей 5×5, вывод матрицы на экран с корректными значениями от 0 до 100

Продолжение таблицы 1

№	Описание теста	Предусловие	Тестирование	Ожидаемый результат
3	Автоматический выбор истока и стока	Граф сгенерирован	Нажатие Enter при запросе изменения выбора вершин	Автоматическое определение вершин с максимальными исходящими и входящими рёбрами, вывод предложенных вершин
4	Ручной выбор истока и стока	Граф сгенерирован	Выбор 'y' при запросе изменения → Ввод корректных номеров вершин (0-4 для графа из 5 вершин)	Принятие введённых вершин, проверка на корректность и уникальность, установка выбранных вершин как истока и стока
5	Вычисление максимального потока	Граф сгенерирован, вершины выбраны	Наблюдение за пошаговой работой алгоритма	Поиск увеличивающих путей через BFS, пошаговый вывод путей и потоков, вычисление итогового максимального потока

Продолжение таблицы 1

№	Описание теста	Предусловие	Тестирование	Ожидаемый результат
6	Анализ характеристик графа	Расчёт максимального потока завершён	Автоматический анализ после вычисления потока	Вывод количества вершин, рёбер, суммарной и средней пропускной способности
7	Сохранение графа перед расчётом	Граф сгенерирован	Выбор 'y' при запросе сохранения перед расчётом → Ввод имени файла (например, "graph1.txt")	Создание файла с матрицей пропускных способностей и параметрами графа, подтверждение успешного сохранения
8	Загрузка графа из файла	Файл с графом существует	Выбор режима 2 → Ввод имени существующего файла	Корректная загрузка матрицы и параметров графа, вывод матрицы на экран, установка параметров

Продолжение таблицы 1

№	Описание теста	Предусловие	Тестирование	Ожидаемый результат
9	Сохранение графа после расчёта	Расчёт максимального потока завершён	Выбор 'y' при запросе сохранения графа → Ввод имени файла	Сохранение текущего состояния сети в файл, подтверждение успешного сохранения
10	Сохранение результатов расчёта	Расчёт максимального потока завершён	Выбор 'y' при запросе сохранения результатов → Ввод имени файла (например, "result1.txt")	Создание файла отчёта с параметрами графа, выбранными вершинами, максимальным потоком и датой расчёта
11	Перезапуск программы	Любой этап работы программы	Выбор 'y' при запросе о повторном запуске	Очистка экрана, возврат к начальному меню выбора режима, возможность нового расчёта

Продолжение таблицы 1

№	Описание теста	Предусловие	Тестирование	Ожидаемый результат
12	Завершение работы программы	Любой этап работы программы	Выбор 'n' при запросе о повторном запуске	Вывод сообщения о завершении работы, корректное завершение программы
13	Обработка ошибочного ввода	Программа запрашивает ввод	Ввод некорректных данных (буквы вместо цифр, номера вершин вне диапазона)	Вывод сообщения об ошибке, повторный запрос корректных данных, продолжение работы программы
14	Загрузка несуществующего файла	Файл не существует	Выбор режима 2 → Ввод имени несуществующего файла	Вывод сообщения об ошибке загрузки, автоматический переход к генерации случайного графа
15	Тестирование на разных размерах графов	Программа запущена	Последовательный запуск с n=4, 7, 10	Корректная работа алгоритма для всех размеров, пропорциональное увеличение времени расчёта с ростом n

Продолжение таблицы 1

№	Описание теста	Предусловие	Тестирование	Ожидаемый результат
16	Сравнение ориентированных и неориентированных графов	Программа запущена	Генерация графов с одинаковыми параметрами, но разным типом	Правильное формирование матриц (симметричных для неориентированных), корректный расчёт потоков для обоих типов

4.1. Тестирование программы

Для тестирования процесса создания транспортной сети в главном меню был выбран пункт «1 Сгенерировать случайный граф», а затем указано количество вершин графа «5», а затем и тип графа «Ориентированный». Затем было введено «y» и указано название файла в консоли «lol.txt» и выбран сток и исток, предложенный автоматически. (Рисунок 2)

```
=============================================||
||   РЕАЛИЗАЦИЯ АЛГОРИТМА ФОРДА-ФАЛКЕРСОНА    ||
||   Для поиска максимального потока в сети   ||
||=============================================

=== РЕЖИМ РАБОТЫ С ГРАФОМ ===
1 - Сгенерировать случайный граф
2 - Загрузить граф из файла
Выберите режим (1 или 2): 1
Введите количество вершин графа (рекомендуется 4-10): 5

Тип графа:
1 - Ориентированный
2 - Неориентированный
Выберите тип (1 или 2): 1

=== ГЕНЕРАЦИЯ СЛУЧАЙНОЙ СЕТИ ===

Параметры генерации:
- Вероятность наличия ребра: 70%
- Пропускная способность: 1-100
- Петли отсутствуют

Матрица пропускных способностей:
      [0]  [1]  [2]  [3]  [4]
[0]    0    77   63   86   13
[1]    0    0    70   12   0
[2]    0    0    0    94   44
[3]    19   0    0    0    0
[4]    0    70   0   100   0

Хотите сохранить граф в файл перед расчетом? (y/n): y
Введите имя файла для сохранения: lol.txt
Граф успешно сохранен в файл 'lol.txt'

=== ВЫБОР ИСТОКА И СТОКА ===
Автоматически выбрано:
Исток (source): вершина 0
Сток (sink): вершина 4

Хотите изменить выбор? (y/n): n
```

Рисунок 2 – Генерация матрицы пропускных способностей для графа

Результат по окончанию работы самого алгоритма мы можем увидеть. После мы можем сохранить результаты в файл. Выбираем «y» и пишем название файла «lol1.txt» (Рисунок 3).

```
=== ВЫЧИСЛЕНИЕ МАКСИМАЛЬНОГО ПОТОКА ===
Исток: 0, Сток: 4
Количество вершин: 5
Тип графа: ориентированный

Поиск максимального потока...
----------------------------------------
Итерация 1:
  Путь: 0 -> 4
  Поток на пути: 13
  Текущий поток: 13
----------------------------------------
Итерация 2:
  Путь: 0 -> 2 -> 4
  Поток на пути: 44
  Текущий поток: 57
----------------------------------------

Алгоритм завершен. Увеличивающих путей больше нет.

================================================
Максимальный поток из вершины 0 в вершину 4 равен: 57
================================================

=== АНАЛИЗ ГРАФА ===
Количество вершин: 5
Количество рёбер: 11
Суммарная пропускная способность всех рёбер: 648
Средняя пропускная способность ребра: 58.9091

=== СОХРАНЕНИЕ РЕЗУЛЬТАТОВ ===
Хотите сохранить граф в файл? (y/n): n

Хотите сохранить результаты расчета? (y/n): y
Введите имя файла для сохранения результатов: lol1.txt
Результаты успешно сохранены в файл 'lol1.txt'
```

Рисунок 3 – Результат работы программы

Проверим введенные ранее файлы и посмотрим и содержимое.(Рисунок 4-5).

```
# Формат файла графа для алгоритма Форда-Фалкерсона
# Количество вершин: 5
# Тип графа: ориентированный
# Матрица пропускных способностей:
0 77 63 86 13
0 0 70 12 0
0 0 0 94 44
19 0 0 0 0
0 70 0 100 0
```

Рисунок 4 – Файл lol.txt

```
╞==========================================
РЕЗУЛЬТАТЫ РАСЧЕТА МАКСИМАЛЬНОГО ПОТОКА
Алгоритм Форда-Фалкерсона
==========================================

ПАРАМЕТРЫ ГРАФА:
Количество вершин: 5
Тип графа: ориентированный
Исток (source): вершина 0
Сток (sink): вершина 4

РЕЗУЛЬТАТ РАСЧЕТА:
Максимальный поток: 57

==========================================
Дата расчета: Dec 24 2025 23:14:18
```

Рисунок 5 – Файл lol1.txt

Проверим наши файлы на совместимость с программой, запустив её заново(рисунок 6)

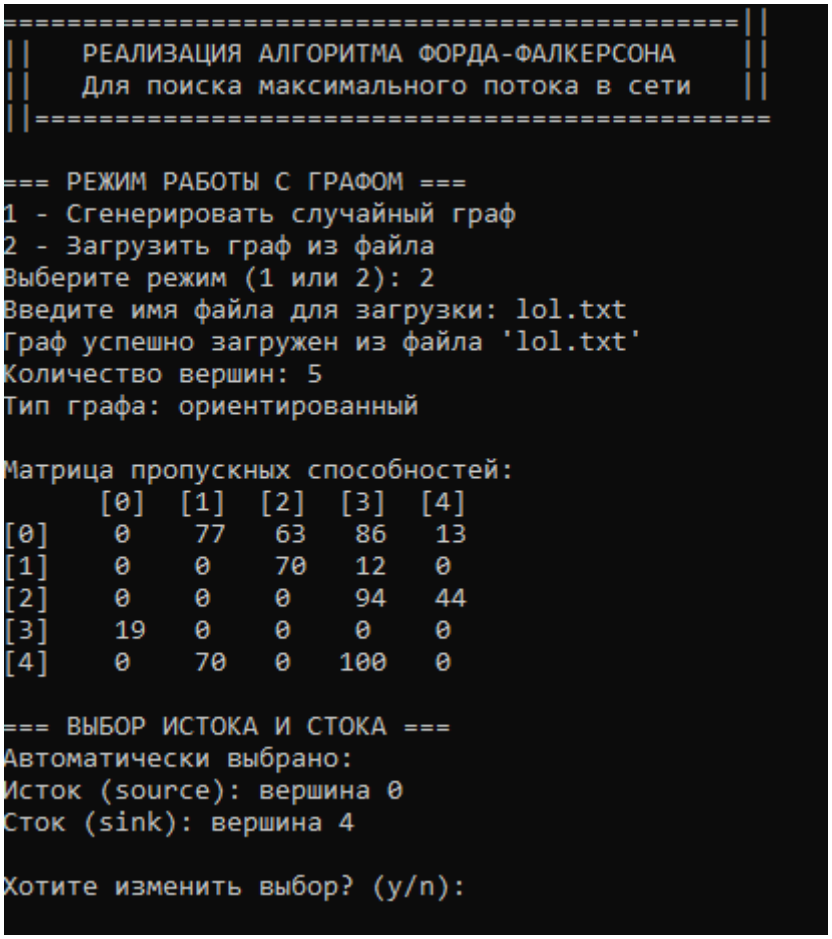

Рисунок 5 – Результат загрузки файла в программу

Тест пройден успешно: функции сохранения и загрузки работают корректно, данные сохраняются и восстанавливаются без изменений, включая пройденный путь.

4.2.Результаты тестирования

Результаты тестирования всех основных функций программы представлены в Таблице 2.

Таблица 2 – Результаты поведения программы при тестировании

№	Описание теста	Полученный результат
1	Запуск программы	Верно
2	Создание транспортной сети	Верно
3	Сохранение сети в файл	Верно
4	Работа алгоритма	Верно
5	Сохранение результатов	Верно
6	Загрузка сети из файла	Верно

Все тесты выполнены успешно, программа работает корректно и соответствует предъявленным требованиям. Интерфейс интуитивно понятен, все режимы функционируют как ожидалось, обработка ошибок осуществляется корректно.

5. Ручной расчёт задачи

Проведем проверку программы посредством ручного анализа работы на примере графа из 5 вершин (Рисунок 6). Рассмотрим небольшой лабиринт с отмеченным вершинами: истоком 1 и стоком 5. Цель – найти максимальный поток из вершины 1 в вершину 5. Программа строит такой граф и выполняет поиск с помощью алгоритмов BFS и Форда-Фалкерсона.

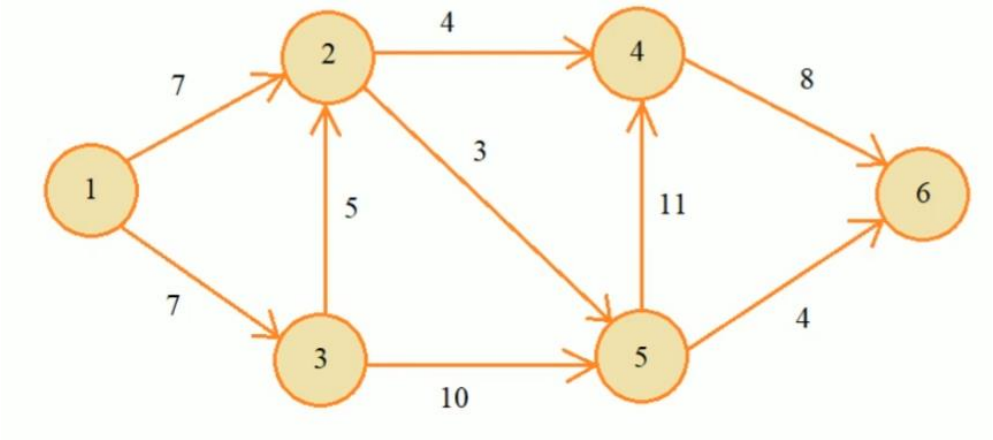

Рисунок 6 – Граф из 6 вершин

Для нахождения максимального потока необходимо рассмотреть маршруты, по которым поток будет формироваться. Рассмотрим первый маршрут 1-2-4-6 (Рисунок 7)

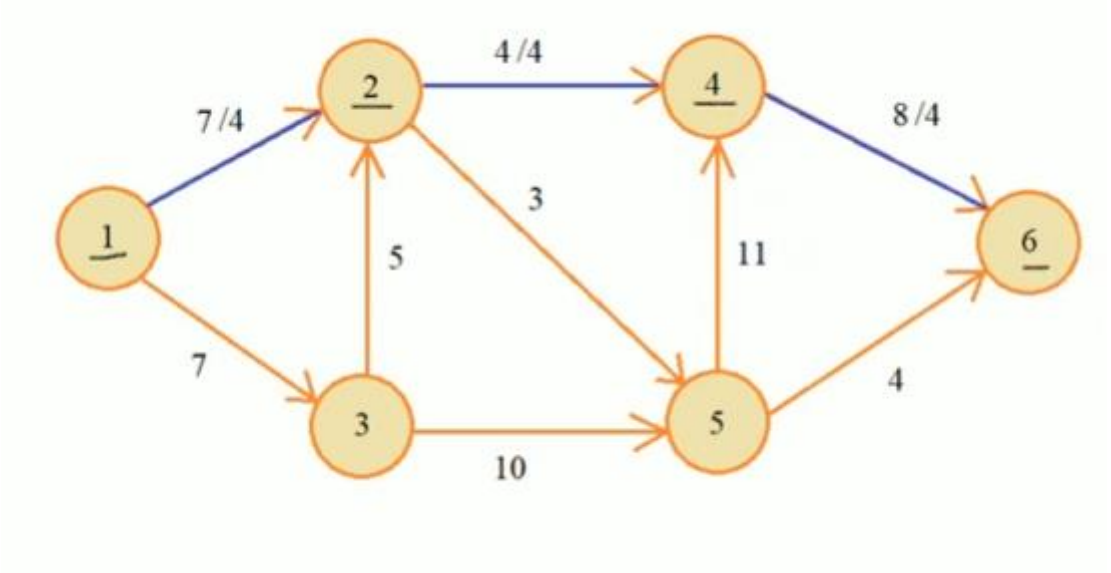

Рисунок 7 – Первый маршрут

Далее мы смотрим максимальный поток, который мы можем пропустить через наш маршрут. Поток не может быть больше самого узкого участка пути.

Максимальный поток по данному маршруту равен 4. Записываем его у каждого пройденного ребра.

Рассмотрим следующий маршрут 1-3-5-6(Рисунок 8).

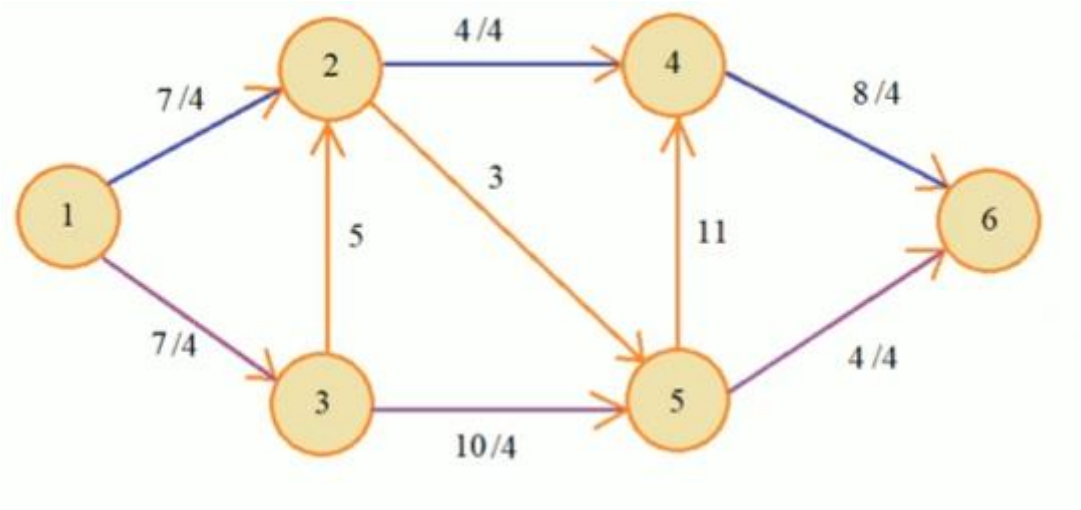

Рисунок 7 – Второй маршрут

Максимальный поток на данном пути равен 4. Записываем его у каждого пройденного ребра.

Смотрим какой поток мы можем пропустить через сеть ещё. Таким маршрутом является 1-3-2-5-4-6(Рисунок 8)

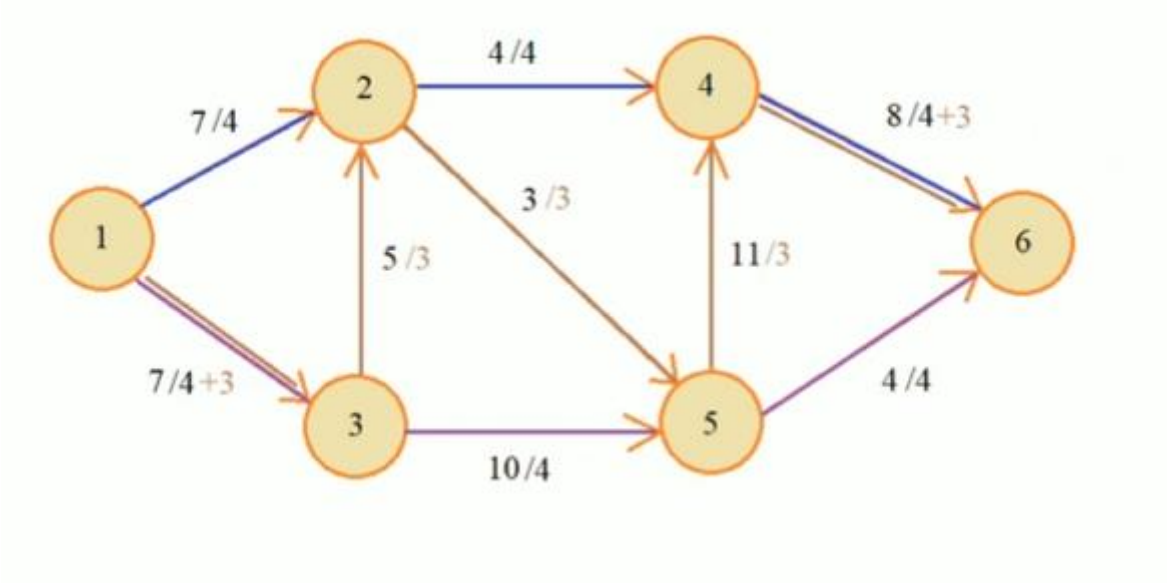

Рисунок 8 – Третий маршрут

По данному маршруту мы можем пустить поток равный 3. Записываем его у каждого пройденного ребра.

Некоторое время считали, что этот поток максимален, однако на самом деле у нас есть 1 ненасыщенный маршрут. Мы не можем идти против направления, однако рассчитывать максимальный поток мы можем. Выбираем маршрут 1-2-3-5-4-6(Рисунок9).

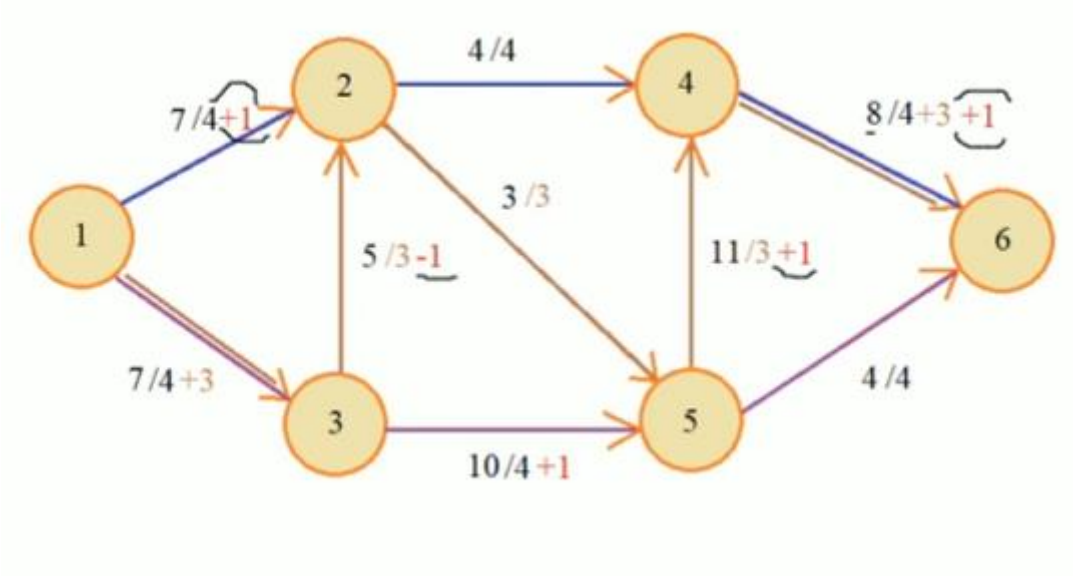

Рисунок 9 – Четвертый маршрут

По данному маршруту максимальный поток равен 1. Если мы двигаемся по стрелке – поток прибавляется, если двигаемся против вычетается. Мы сделали перераспределение потоков и оптимизировали их. Суммарный поток составил 12. Алгоритм завершен

Заключение

При выполнении данной курсовой работы были усовершенствованы навыки разработки программ на языке C++ и применены знания теории графов для решения классической задачи поиска максимального потока в транспортных сетях. Приобретён практический опыт реализации ключевых алгоритмов теории графов: алгоритма Форда-Фалкерсона для нахождения максимального потока и алгоритма поиска в ширину (BFS) для обнаружения увеличивающих путей. Также были развиты навыки работы с файловыми операциями, обработкой пользовательского ввода и созданием консольных приложений с интерактивным интерфейсом.

В рамках курсовой работы была разработана программа для анализа транспортных сетей и расчёта максимального потока. В ней реализованы следующие возможности: генерация случайных транспортных сетей с настраиваемыми параметрами (количество вершин, тип графа, плотность связей); загрузка и сохранение графов в текстовых файлах с сохранением структуры данных; автоматический и ручной выбор вершин-истока и стока; пошаговый расчёт максимального потока с визуализацией процесса работы алгоритма; анализ характеристик графа (количество рёбер, суммарная и средняя пропускная способность); сохранение результатов расчёта в формате отчёта.

Проведённое тестирование показало, что реализованный алгоритм Форда-Фалкерсона с использованием BFS для поиска увеличивающих путей эффективно решает задачу нахождения максимального потока в сетях различной сложности. Алгоритм демонстрирует корректную работу как для небольших, так и для более сложных конфигураций. Особенностью реализации является использование обратных рёбер, что позволяет алгоритму корректировать ранее распределённый поток и гарантирует нахождение оптимального решения согласно теореме Форда-Фалкерсона.

Программа обладает полноценным функциональным набором, достаточным для использования в учебных целях как демонстрационный

инструмент изучения алгоритмов теории графов. Разработанный проект наглядно иллюстрирует применение алгоритмической теории графов к практической задаче оптимизации транспортных потоков и подтверждает корректность реализации рассмотренных алгоритмов. Программа может быть использована в образовательном процессе для изучения основных понятий теории графов, алгоритмов поиска максимального потока и методов анализа транспортных сетей.

Перспективой развития программы может стать добавление визуализации графов в графическом интерфейсе, реализация дополнительных алгоритмов поиска потока (например, алгоритма Диница или Эдмондса-Карпа), а также расширение функционала для анализа более сложных характеристик сетей (связность, центральность вершин, минимальные разрезы).

.

Список литературы

1. Керниган Б. У., Язык программирования Си / Б. У. Керниган, Д. М. Ритчи; пер. с англ. – Москва: 3-е издание, 1985. – 272 с.
2. Роберт Мартин, Чистый код: создание, анализ и рефакторинг / Роберт Мартин; СПб.: Питер, 2023 г. – 464 с. – ISBN 978-5-4461-0960-9
3. Прата, С. Язык программирования C++. Лекции и упражнения / С. Прата; пер. с англ. – 6-е изд. – Санкт-Петербург: Питер, 2019. – 1248 с. – ISBN 978-5-4461-1153-8.
4. Тюгашев, А. А. Языки программирования: учебное пособие / А. А. Тюгашев. – Санкт-Петербург: Лань, 2018. – 324 с. – ISBN 978-5-8114-2941-7.
5. Аблязов, Р. Р. Программирование на ассемблере на платформе x86-64 / Р. Р. Аблязов. – Санкт-Петербург: БХВ-Петербург, 2016. – 480 с. – ISBN 978-5-9775-3676-1.

Приложение А. Листинг программы

```
#include <iostream>
#include <cstring>
#include <queue>
#include <climits>
#include <locale>
#include <ctime>
#include <cstdlib>
#include <vector>
#include <iomanip>
#include <sstream>
#include <fstream>

using namespace std;

void clearScreen();
int isInteger(const string& message);
bool bfs(int** rGraph, int numG, int s, int t, int parent[]);
int fordFulkerson(int** graph, int numG, int s, int t);
void printMatrix(int** Matrix, int numG);
string intToString(int value);
bool askToRestart();
char safeCharInput(const string& message);
bool saveGraphToFile(int** graph, int numG, bool isOriented, const
string& filename);
bool loadGraphFromFile(int**& graph, int& numG, bool& isOriented,
const string& filename);
bool saveResultToFile(int source, int sink, int max_flow, int numG,
bool isOriented, const string& filename);

int main() {
    setlocale(LC_ALL, "Rus");

    bool restartProgram = true;

    while (restartProgram) {
        clearScreen();
        srand(static_cast<unsigned int>(time(NULL)));
```

```
            cout                                                    <<
"=============================================||\n";
            cout << "||   РЕАЛИЗАЦИЯ АЛГОРИТМА ФОРДА-ФАЛКЕРСОНА    ||\n";
            cout << "||   Для поиска максимального потока в сети   ||\n";
            cout                                                    <<
"||=============================================\n\n";

            int** graph = nullptr;
            int numG = 0;
            bool isOriented = false;
            bool loadedFromFile = false;


            cout << "=== РЕЖИМ РАБОТЫ С ГРАФОМ ===\n";
            cout << "1 - Сгенерировать случайный граф\n";
            cout << "2 - Загрузить граф из файла\n";
            int modeChoice = isInteger("Выберите режим (1 или 2): ");
            while (modeChoice != 1 && modeChoice != 2) {
                cout << "Ошибка! Введите 1 или 2\n";
                modeChoice = isInteger("Выберите режим (1 или 2): ");
            }

            if (modeChoice == 2) {

                string filename;
                cout << "Введите имя файла для загрузки: ";
                cin >> filename;
                cin.ignore(numeric_limits<streamsize>::max(), '\n');

                graph = nullptr;
                if    (loadGraphFromFile(graph,    numG,    isOriented,
filename)) {
                    cout  <<  "Граф  успешно  загружен  из  файла  '"  <<
filename << "'\n";
                    loadedFromFile = true;
                    cout << "Количество вершин: " << numG << "\n";
                    cout   <<   "Тип   графа:   "   <<   (isOriented   ?
"ориентированный" : "неориентированный") << "\n\n";
                }
                else {
```

```
                    cout << "Ошибка загрузки файла. Будет сгенерирован
случайный граф.\n";
                    loadedFromFile = false;
                    modeChoice = 1;
                }
            }

            if (modeChoice == 1) {

                numG = isInteger("Введите количество вершин графа
(рекомендуется 4-10): ");

                while (numG <= 0) {
                    cout << "Ошибка! Количество вершин должно быть
положительным\n";
                    numG = isInteger("Введите количество вершин графа:
");
                }
                while (numG == 1) {
                    cout << "Ошибка! Для построения сети необходимо более
1 вершины\n";
                    numG = isInteger("Введите количество вершин графа:
");
                }

                if (!loadedFromFile) {
                    cout << "\nТип графа:\n";
                    cout << "1 — Ориентированный\n";
                    cout << "2 — Неориентированный\n";
                    int orientChoice = isInteger("Выберите тип (1 или 2):
");
                    while (orientChoice != 1 && orientChoice != 2) {
                        cout << "Ошибка! Введите 1 или 2\n";
                        orientChoice = isInteger("Выберите тип (1 или 2):
");
                    }
                    isOriented = (orientChoice == 1);
                }
```

```
                graph = new int* [numG];
                for (int i = 0; i < numG; i++) {
                    graph[i] = new int[numG];
                }

                cout << "\n=== ГЕНЕРАЦИЯ СЛУЧАЙНОЙ СЕТИ ===\n\n";
                cout << "Параметры генерации:\n";
                cout << "- Вероятность наличия ребра: 70%\n";
                cout << "- Пропускная способность: 1-100\n";
                cout << "- Петли отсутствуют\n\n";


                for (int i = 0; i < numG; i++) {
                    for (int j = 0; j < numG; j++) {
                        if (i == j) {
                            graph[i][j] = 0;
                        }
                        else {
                            int hasEdge = (rand() % 100) < 70;
                            if (hasEdge) {
                                graph[i][j] = rand() % 100 + 1;
                            }
                            else {
                                graph[i][j] = 0;
                            }

                            if (!isOriented && i < j) {
                                graph[j][i] = graph[i][j];
                            }
                        }
                    }
                }

                if (isOriented) {
                    for (int i = 0; i < numG; i++) {
                        for (int j = i + 1; j < numG; j++) {
                            if (graph[i][j] > 0 && graph[j][i] > 0 &&
(rand() % 100) < 50) {
                                if (rand() % 2 == 0) {
                                    graph[i][j] = 0;
```

```
                        }
                        else {
                            graph[j][i] = 0;
                        }
                    }
                }
            }
        }

        cout << "Матрица пропускных способностей:\n";
        printMatrix(graph, numG);


        if (modeChoice == 1) {
            cout << "\nХотите сохранить граф в файл перед расчетом?
(y/n): ";
            char saveBeforeChoice;
            cin >> saveBeforeChoice;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');

            if (saveBeforeChoice == 'y' || saveBeforeChoice == 'Y')
{
                string filename;
                cout << "Введите имя файла для сохранения: ";
                getline(cin, filename);
                if    (saveGraphToFile(graph,    numG,    isOriented,
filename)) {
                    cout  <<  "Граф  успешно  сохранен  в  файл  '"  <<
filename << "'\n";
                }
                else {
                    cout << "Ошибка сохранения файла.\n";
                }
            }
        }

        int source, sink;

        cout << "\n=== ВЫБОР ИСТОКА И СТОКА ===\n";
```

```
source = 0;
sink = numG - 1;

bool sourceHasOutgoing = false;
bool sinkHasIncoming = false;

for (int j = 0; j < numG; j++) {
    if (graph[source][j] > 0) sourceHasOutgoing = true;
    if (graph[j][sink] > 0) sinkHasIncoming = true;
}


if (!sourceHasOutgoing || !sinkHasIncoming) {
    for (int i = 0; i < numG; i++) {
        for (int j = 0; j < numG; j++) {
            if (graph[i][j] > 0) {
                source = i;
                break;
            }
        }
        if (source != 0) break;
    }

    for (int i = numG - 1; i >= 0; i--) {
        for (int j = 0; j < numG; j++) {
            if (graph[j][i] > 0) {
                sink = static_cast<int>(i);
                break;
            }
        }
        if (sink != numG - 1) break;
    }
}

cout << "Автоматически выбрано:\n";
cout << "Исток (source): вершина " << source << endl;
cout << "Сток (sink): вершина " << sink << endl;
```

```
            cout << "\nХотите изменить выбор? (y/n): ";
            char changeChoice = safeCharInput("");

            if (changeChoice == 'y' || changeChoice == 'Y') {
                string prompt = "\nВведите номер вершины-истока (0-" +
intToString(numG - 1) + "): ";
                source = isInteger(prompt);
                while (source < 0 || source >= numG) {
                    cout << "Ошибка! Вершина должна быть от 0 до " <<
numG - 1 << "\n";
                    source = isInteger("Введите номер вершины-истока:
");
                }

                sink = isInteger("Введите номер вершины-стока (0-" +
intToString(numG - 1) + "): ");
                while (sink < 0 || sink >= numG || sink == source) {
                    if (sink == source) {
                        cout << "Ошибка! Сток не может совпадать с
истоком.\n";
                    }
                    else {
                        cout << "Ошибка! Вершина должна быть от 0 до "
<< numG - 1 << "\n";
                    }
                    sink = isInteger("Введите номер вершины-стока: ");
                }
            }

            cout << "\n=== ВЫЧИСЛЕНИЕ МАКСИМАЛЬНОГО ПОТОКА ===\n";
            cout << "Исток: " << source << ", Сток: " << sink << "\n";
            cout << "Количество вершин: " << numG << "\n";
            cout << "Тип графа: " << (isOriented ? "ориентированный" :
"неориентированный") << "\n\n";

            int max_flow = fordFulkerson(graph, numG, source, sink);

            cout                                                    <<
"\n================================================\n";
```

```
            cout << "Максимальный поток из вершины " << source
                << " в вершину " << sink << " равен: " << max_flow <<
endl;
            cout                                                    <<
"=================================================\n";


            cout << "\n=== АНАЛИЗ ГРАФА ===\n";

            int edgeCount = 0;
            int totalCapacity = 0;
            for (int i = 0; i < numG; i++) {
                for (int j = 0; j < numG; j++) {
                    if (graph[i][j] > 0) {
                        edgeCount++;
                        totalCapacity += graph[i][j];
                    }
                }
            }

            cout << "Количество вершин: " << numG << endl;
            cout << "Количество рёбер: " << edgeCount << endl;
            cout << "Суммарная пропускная способность всех рёбер: " <<
totalCapacity << endl;
            if (edgeCount > 0) {
                double avgCapacity = static_cast<double>(totalCapacity)
/ static_cast<double>(edgeCount);
                cout << "Средняя пропускная способность ребра: " <<
avgCapacity << endl;
            }


            cout << "\n=== СОХРАНЕНИЕ РЕЗУЛЬТАТОВ ===\n";


            cout << "Хотите сохранить граф в файл? (y/n): ";
            char saveGraphChoice;
            cin >> saveGraphChoice;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

```
            if (saveGraphChoice == 'y' || saveGraphChoice == 'Y') {
                string filename;
                cout << "Введите имя файла для сохранения графа: ";
                getline(cin, filename);
                if (saveGraphToFile(graph, numG, isOriented, filename))
{
                    cout << "Граф успешно сохранен в файл '" << filename
<< "'\n";
                }
                else {
                    cout << "Ошибка сохранения графа.\n";
                }
            }


            cout << "\nХотите сохранить результаты расчета? (y/n): ";
            char saveResultChoice;
            cin >> saveResultChoice;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');

            if (saveResultChoice == 'y' || saveResultChoice == 'Y') {
                string filename;
                cout << "Введите имя файла для сохранения результатов:
";
                getline(cin, filename);
                if  (saveResultToFile(source,  sink,  max_flow,  numG,
isOriented, filename)) {
                    cout << "Результаты успешно сохранены в файл '" <<
filename << "'\n";
                }
                else {
                    cout << "Ошибка сохранения результатов.\n";
                }
            }


            for (int i = 0; i < numG; i++) {
                delete[] graph[i];
            }
            delete[] graph;
```

```
            restartProgram = askToRestart();
        }

        cout << "\nПрограмма завершена. До свидания!\n";
        return 0;
    }

    bool saveGraphToFile(int** graph, int numG, bool isOriented, const
string& filename) {
        ofstream file(filename);
        if (!file.is_open()) {
            return false;
        }

        file << "# Формат файла графа для алгоритма Форда-Фалкерсона\n";
        file << "# Количество вершин: " << numG << endl;
        file << "# Тип графа: " << (isOriented ? "ориентированный" :
"неориентированный") << endl;
        file << "# Матрица пропускных способностей:\n";

        for (int i = 0; i < numG; i++) {
            for (int j = 0; j < numG; j++) {
                file << graph[i][j];
                if (j < numG - 1) file << " ";
            }
            file << endl;
        }

        file.close();
        return true;
    }

    bool loadGraphFromFile(int**& graph, int& numG, bool& isOriented,
const string& filename) {
        ifstream file(filename);
        if (!file.is_open()) {
            return false;
        }
```

```
        string line;
        vector<vector<int>> matrix;
        numG = 0;
        isOriented = true;


        while (getline(file, line)) {
            if (line.empty() || line[0] == '#') {
                if (line.find("Тип графа:") != string::npos) {
                    if (line.find("неориентированный") != string::npos)
{
                        isOriented = false;
                    }
                }
                continue;
            }

            stringstream ss(line);
            vector<int> row;
            int value;

            while (ss >> value) {
                row.push_back(value);
            }

            if (!row.empty()) {
                matrix.push_back(row);
                if (numG == 0) {
                    numG = static_cast<int>(row.size());
                }
            }
        }

        file.close();

        if (static_cast<int>(matrix.size()) != numG) {
            return false;
        }

        graph = new int* [numG];
```

```
        for (int i = 0; i < numG; i++) {
            graph[i] = new int[numG];
            for (int j = 0; j < numG; j++) {
                graph[i][j] = matrix[i][j];
            }
        }

        return true;
    }

    bool saveResultToFile(int source, int sink, int max_flow, int numG,
bool isOriented, const string& filename) {
        ofstream file(filename);
        if (!file.is_open()) {
            return false;
        }

        file << "=============================================\n";
        file << "РЕЗУЛЬТАТЫ РАСЧЕТА МАКСИМАЛЬНОГО ПОТОКА\n";
        file << "Алгоритм Форда-Фалкерсона\n";
        file << "=============================================\n\n";

        file << "ПАРАМЕТРЫ ГРАФА:\n";
        file << "Количество вершин: " << numG << endl;
        file << "Тип графа: " << (isOriented ? "ориентированный" :
"неориентированный") << endl;
        file << "Исток (source): вершина " << source << endl;
        file << "Сток (sink): вершина " << sink << endl;

        file << "\nРЕЗУЛЬТАТ РАСЧЕТА:\n";
        file << "Максимальный поток: " << max_flow << endl;

        file << "\n=============================================\n";
        file << "Дата расчета: " << __DATE__ << " " << __TIME__ << endl;

        file.close();
        return true;
    }

    void clearScreen() {
```

```
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

char safeCharInput(const string& message) {
    char choice;
    cout << message;

    cin >> choice;


    while (cin.get() != '\n') {
        continue;
    }

    return choice;
}

int isInteger(const string& message) {
    int value;
    while (true) {
        cout << message;
        if (!(cin >> value)) {
            cout << "Ошибка: введено не число.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
        if (cin.peek() != '\n') {
            cout << "Ошибка: введено не целое число.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
        return value;
    }
}
```

```
string intToString(int value) {
    stringstream ss;
    ss << value;
    return ss.str();
}

bool bfs(int** rGraph, int numG, int s, int t, int parent[]) {
    bool* visited = new bool[numG];
    memset(visited, 0, sizeof(bool) * static_cast<size_t>(numG));

    queue<int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (int v = 0; v < numG; v++) {
            if (!visited[v] && rGraph[u][v] > 0) {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }

    bool result = visited[t];
    delete[] visited;
    return result;
}

int fordFulkerson(int** graph, int numG, int s, int t) {
    int u, v;



    int** rGraph = new int* [numG];
    for (int i = 0; i < numG; i++) {
```

```
            rGraph[i] = new int[numG];
            for (int j = 0; j < numG; j++) {
                rGraph[i][j] = graph[i][j];
            }
        }

        int* parent = new int[numG];
        int max_flow = 0;
        int path_flow;
        int iteration = 1;

        cout << "Поиск максимального потока...\n";
        cout << "-----------------------------------\n";


        while (bfs(rGraph, numG, s, t, parent)) {

            path_flow = INT_MAX;
            for (v = t; v != s; v = parent[v]) {
                u = parent[v];
                path_flow = (path_flow < rGraph[u][v]) ? path_flow :
rGraph[u][v];
            }


            for (v = t; v != s; v = parent[v]) {
                u = parent[v];
                rGraph[u][v] -= path_flow;
                rGraph[v][u] += path_flow;
            }


            max_flow += path_flow;


            cout << "Итерация " << iteration++ << ":\n";
            cout << "  Путь: ";
            vector<int> path;
            for (v = t; v != -1; v = parent[v]) {
                path.push_back(v);
```

```
            }
            for (int i = static_cast<int>(path.size()) - 1; i >= 0; i--
) {
                cout << path[i];
                if (i > 0) cout << " -> ";
            }
            cout << "\n  Поток на пути: " << path_flow << endl;
            cout << "  Текущий поток: " << max_flow << endl;
            cout << "----------------------------------------\n";
        }

        cout << "\nАлгоритм завершен. Увеличивающих путей больше нет.\n";


        for (int i = 0; i < numG; i++) {
            delete[] rGraph[i];
        }
        delete[] rGraph;
        delete[] parent;

        return max_flow;
    }

    void printMatrix(int** Matrix, int numG) {

        cout << "    ";
        for (int j = 0; j < numG; j++) {
            cout << setw(3) << "[" << j << "]";
        }
        cout << "\n";


        for (int i = 0; i < numG; i++) {
            cout << "[" << setw(1) << i << "] ";
            for (int j = 0; j < numG; j++) {
                if (Matrix[i][j] == 0) {
                    cout << setw(5) << " 0 ";
                }
                else {
                    cout << setw(5) << Matrix[i][j];
```

```
                }
            }
            cout << "\n";
        }
    }

    bool askToRestart() {
        cout                                                    <<
"\n=================================================\n";
        cout << "Хотите начать заново? (1 - да, 0 - нет): ";

        int choice;
        cin >> choice;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        return (choice == 1);
    }
```