

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

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по курсу «Логика и основа алгоритмизации в инженерных задачах»
на тему «Разработка алгоритма Форда-Фалкерсона для нахождения
максимального потока»

28.02 25
хорошо
Фед

Выполнил: студент группы 22ВВВ3
Гречихин П. П.

Приняли:

к.э.н. доцент Акифьев И. В.

к.т.н. доцент Юрова О. В.

Пенза 20251

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

«___» _____ 20__

ЗАДАНИЕ

на курсовое проектирование по курсу

Копира и алгоритмы алгоритмизации в имп. задачах
Студенту Брахинову Полку Группа 221303
Тема проекта реализация алгоритма Форда-Фалксона для нахождения максим. потока
в графе.

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и курс. отчета
в соответствии с данными задания
и курсового проекта
Подать полную запись работы

1. Постановка задачи
2. Теоретическая часть
3. Описание алгоритма
4. Выводы из задания
5. Описание программы
6. Тесты
7. Список литературы
8. Итоги
9. Результаты работы.

Объем работы по курсу

1. Расчетная часть

ручной расчет работы программы

2. Графическая часть

схема охлаждения в формате блок-схем

3. Экспериментальная часть

Тестирование программы

Срок выполнения проекта по разделам

- 1 исследование теоретической части
- 2 разработка алгоритмов
- 3 разработка программы
- 4 тестирование, завершение разработки
- 5 оформление письменной записки
- 6
- 7
- 8

Дата выдачи задания "06" сентября

Дата защиты проекта "

Руководитель Корота Ольга Викторовна

Задание получил "02" сентября 2023 г.

Студент Бречихин Павел Павлович

Содержание

Содержание	4
Реферат	5
Введение	6
1. Постановка задачи.	7
2. Теоретическая часть задания.....	8
3. Описание алгоритма поставленной задачи.	10
4. Описание программы.	15
5. Отладка и тестирование.....	17
6. Ручной расчёт	20
Заключение.....	23
Список литературы.	24
Приложение А Листинги программы.....	25

Реферат

Отчёт 26 стр., 20 рисунков, 1 таблица, 1 приложение.

ГРАФ, ОСТАТОЧНАЯ СЕТЬ, АЛГОРИТМ ФОРДА -
ФАЛКЕРСОНА, .ОБХОД В ШИРИНУ (BFS), ПОТОК

Цель алгоритма Форда-Фалкерсона — найти максимальный поток в сети с учетом ограничений на пропускные способности ребер и закона сохранения потока. Этот алгоритм широко применяется в различных областях, где требуется оптимизация потоков.

Введение

В современном мире задачи, связанные с оптимизацией потоков, имеют огромное значение в различных областях, таких как транспортные системы, компьютерные сети, распределение ресурсов и многие другие. Одной из ключевых задач в теории графов является задача нахождения максимального потока в сети. Алгоритм Форда-Фалкерсона, разработанный в 1956 году Лестером Фордом и Делбертом Фалкерсоном, является классическим методом решения этой задачи. Его изучение и применение актуально как для теоретических исследований, так и для практических приложений.

Целью данного курсового проекта является изучение алгоритма Форда-Фалкерсона, его теоретических основ, а также разработка программной реализации данного алгоритма. В рамках проекта планируется рассмотреть основные этапы работы алгоритма, его преимущества и ограничения, а также провести анализ эффективности на различных примерах.

1. Постановка задачи.

Задание для курсового проектирования заключается в создании программного продукта позволяющего находить максимальный поток в ориентированном графе для заданных вершин. При том, одной из главных особенностей программного продукта является визуализация графа по средствам ручного ввода матрицы смежности или её случайная генерация.

Для удовлетворения требований задачи я выбрал язык C++. C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения очень широка. C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#. Синтаксис C++ унаследован от языка C. Также C++ является объектно-ориентированным языком программирования (ООП). Объектно-ориентированное программирование – это методология программирования, основанная на представлении программ в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

C++ доступен для большинства операционных систем, включая Linux, многие модификации Unix, Microsoft Windows, Mac OS X, RISC OS, и многих других.

Преимущества C++:

- Большая безопасность
- Возможность писать обобщенный код с помощью шаблонов
- Возможность использовать объектно-ориентированный подход
- Управление ресурсами с помощью RAII
- Упрощение кода за счет перегрузки функций и операторов
- Более простая обработка ошибок за счет исключений

2. Теоретическая часть задания.

Алгоритм Форда-Фалкерсона используется для нахождения максимального потока в сети (графе) от истока (source) к стоку (sink). Он работает с остаточной сетью и ищет увеличивающие пути, по которым можно увеличить поток. Давайте разберем его по шагам.

1. Инициализация:

- a. Начальный поток в сети равен нулю на всех ребрах.
- b. Создается остаточная сеть, которая изначально совпадает с исходным графом.

2. Поиск увеличивающего пути:

- a. Используется поиск в ширину (BFS) или поиск в глубину (DFS) для нахождения пути от истока к стоку в остаточной сети.
- b. Если путь найден, переходим к шагу 3.
- c. Если путь не найден, алгоритм завершается, и текущий поток является максимальным.

3. Определение минимальной остаточной пропускной способности:

- a. Для найденного увеличивающего пути вычисляется минимальная остаточная пропускная способность (минимальное значение на ребрах пути).
- b. Это значение называется величиной увеличивающего потока (pathFlow).

4. Обновление потока и остаточной сети:

- a. Для каждого ребра на увеличивающем пути:
 - i. Уменьшаем пропускную способность прямого ребра на pathFlow.
 - ii. Увеличиваем пропускную способность обратного ребра на pathFlow.

б. Это эквивалентно увеличению потока на `pathFlow` вдоль увеличивающего пути.

5. Повторение:

а. Повторяем шаги 2–4, пока существуют увеличивающие пути в остаточной сети.

6. Завершение:

а. Когда увеличивающих путей больше нет, алгоритм завершается. Текущий поток является максимальным.

3. Описание алгоритма поставленной задачи.

Разобьем одну итерацию проведения потока по выбранному пути на маленькие шаги, чтобы визуально стало понятно, как меняется остаточная сеть после проталкивания очередного потока.

- Наш алгоритм нашел следующий путь из вершины в нашей остаточной сети, которая на момент начала, совпадает с исходным графом.

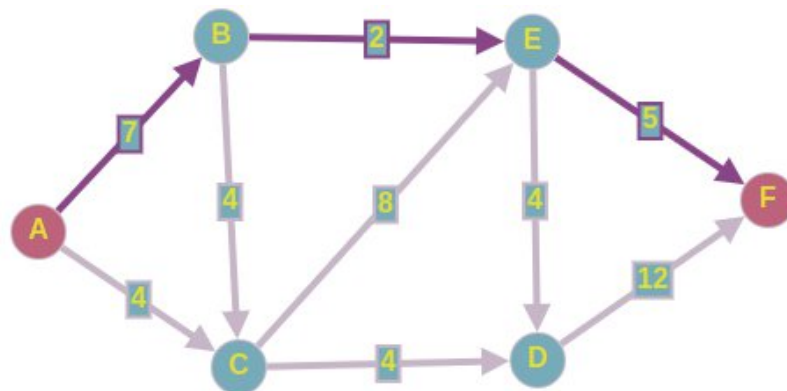


Рисунок 1 - Исходный граф

- Больше 2 ед.мы никак не сможем пустить, пропускаем наш поток по этим рёбрам из истока в сток.

Получаем следующее:

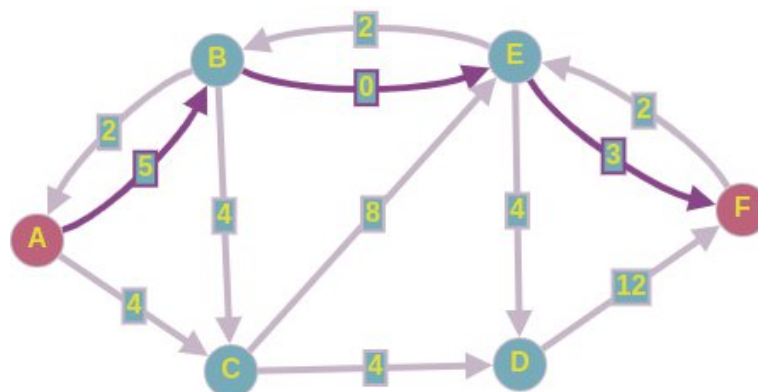


Рисунок 2 - Первый поток

Рёбра с нулевым весом можно удалять. Таким образом, на первой итерации мы смогли увеличить максимальный поток на 2 ед.

Теперь дело за малым, остается всего лишь интегрироваться до тех пор, пока существует путь из A в F.

- На 2-ой итерации мы нашли такой путь в нашей остаточной сети:

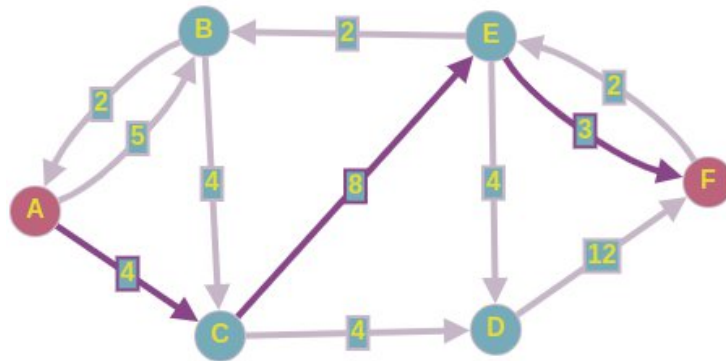


Рисунок 3 - Следующий поток

Пропускаем 3 ед. потока по этому пути, и перестраиваем остаточную сеть. Получаем следующее:

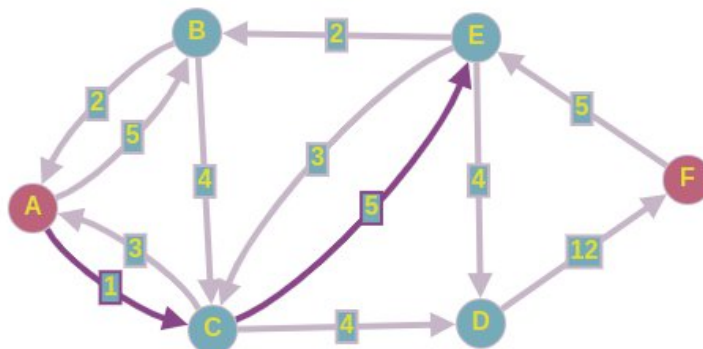


Рисунок 4 – Остаточная сеть

- На 3-ей итерации нашли такой путь в нашей модифицированной остаточной сети:

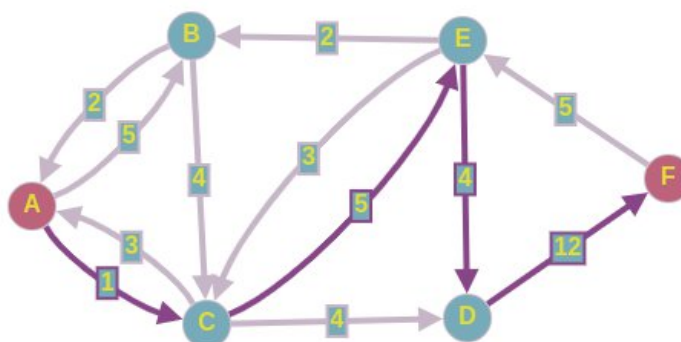


Рисунок 5 - Следующий поток

Пускаем 1 ед. потока по этому пути и перестраиваем нашу остаточную сеть. Получим следующую картину:

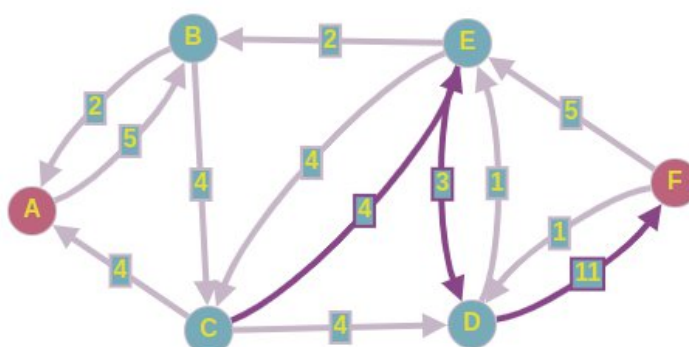


Рисунок 6 - Остаточная сеть

- На 4-ой итерации находим следующий путь в остаточной сети:

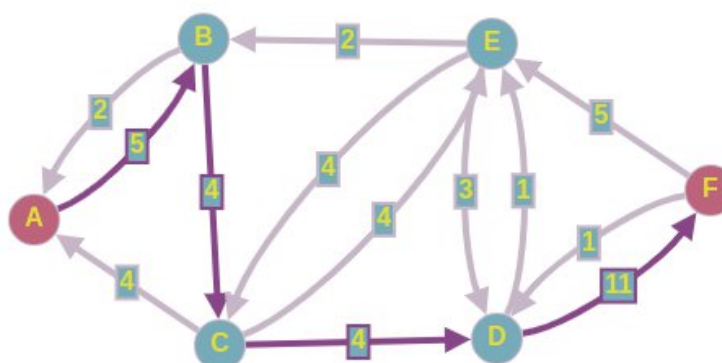


Рисунок 7 - Следующий поток

Пускаем 4 ед. потока по этому пути и перестраиваем нашу остаточную сеть. Получим следующую картину:

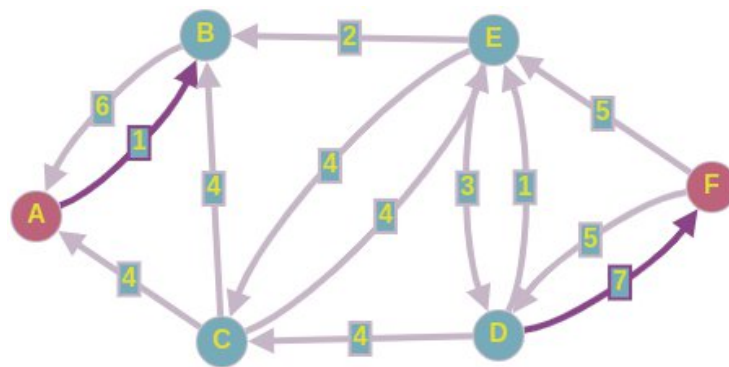


Рисунок 8 - Остаточная сеть

На этом этапе наш алгоритм прекратит выполнение из-за того, что пути из истока в сток не существует. И ответом к поставленной задаче будет служить сумма потоков всех найденных увеличивающихся путей.

Псевдокод алгоритма Форда-Фалкерсона:

Функция BFS(остаточная_сеть, родитель, исток, сток):

V = количество вершин в остаточной_сети
 visited = массив размера V, заполненный false
 очередь = пустая очередь
 очередь.добавить(исток)
 visited[исток] = true
 родитель[исток] = -1

Пока очередь не пуста:

u = очередь.удалить()

Для каждой вершины v от 0 до V-1:

Если visited[v] == false и остаточная_сеть[u][v] > 0:

очередь.добавить(v)

родитель[v] = u

visited[v] = true

Если v == сток:

Вернуть true

Вернуть false

Функция ФордФалкерсон(граф, исток, сток):

V = количество вершин в графе
остаточная_сеть = копия графа
родитель = массив размера V
max_flow = 0

Пока BFS(остаточная_сеть, родитель, исток, сток) == true:
 path_flow = бесконечность

 // Находим минимальную пропускную способность на пути

 Для v = сток; v != исток; v = родитель[v]:

 u = родитель[v]

 path_flow = минимум(path_flow, остаточная_сеть[u][v])

 // Обновляем остаточную сеть

 Для v = сток; v != исток; v = родитель[v]:

 u = родитель[v]

 остаточная_сеть[u][v] -= path_flow

 остаточная_сеть[v][u] += path_flow

 // Увеличиваем общий поток

 max_flow += path_flow

Вернуть max_flow

4. Описание программы.

Структура программы:

1. Основные компоненты:
 - a. Функция `bfs`: реализует поиск в ширину (BFS) для нахождения увеличивающего пути в остаточной сети.
 - b. Функция `fordFulkerson`: реализует основной алгоритм Форда-Фалкерсона.
 - c. Функция `inputGraph`: позволяет пользователю ввести граф с клавиатуры.
 - d. Функция `generateRandomGraph`: генерирует случайный граф.
 - e. Функция `main`: управляет взаимодействием с пользователем и запускает алгоритм.
2. Используемые библиотеки:
 - a. `<iostream>`: для ввода и вывода данных.
 - b. `<vector>`: для хранения графа и остаточной сети.
 - c. `<queue>`: для реализации обхода в ширину.
 - d. `<climits>`: для использования константы `INT_MAX`.
 - e. `<cstdlib>` и `<ctime>`: для генерации случайных чисел.

Подробное описание функций:

1. Функция `bfs`:

- a. Назначение: Поиск увеличивающего пути от истока к стоку в остаточной сети.
- b. Аргументы:
 - `residualGraph`: остаточная сеть (матрица пропускных способностей).
 - `parent`: массив для хранения пути.
 - `source`: исток.
 - `sink`: сток.
- c. Возвращаемое значение: `true`, если путь найден, иначе `false`.
- d. Алгоритм:
 - Используется обход в ширину для обхода графа.
 - Если путь от истока к стоку найден, он сохраняется в массиве `parent`.

2. Функция `fordFulkerson`:

- a. Назначение: Реализация алгоритма Форда-Фалкерсона для нахождения максимального потока.
- b. Аргументы:

- graph: исходный граф (матрица пропускных способностей).
- source: исток.
- sink: сток.
- c. Возвращаемое значение: Значение максимального потока.
- d. Алгоритм:
 - Инициализирует остаточную сеть.
 - Пока существует увеличивающий путь (с помощью bfs):
 - o Находит минимальную пропускную способность на пути.
 - o Обновляет остаточную сеть и увеличивает поток.

3. Функция inputGraph:

- a. Назначение: Ввод графа с клавиатуры.
- b. Возвращаемое значение: Матрица пропускных способностей.
- c. Алгоритм:
 - Запрашивает у пользователя количество вершин.
 - Запрашивает пропускные способности для каждого ребра.

4. Функция generateRandomGraph:

- a. Назначение: Генерация случайного графа.
- b. Аргументы:
 - V: количество вершин.
- c. Возвращаемое значение: Матрица пропускных способностей.
- d. Алгоритм:
 - Заполняет матрицу случайными значениями от 0 до 19.
 - Убеждается, что петли (ребра из вершины в себя) отсутствуют.

отсутствуют.

5. Функция main:

- e. Назначение: Управление программой.
- f. Алгоритм:
 - Предлагает пользователю выбрать способ задания графа (ввод или генерация).
 - В зависимости от выбора вызывает inputGraph или generateRandomGraph.
 - Запрашивает у пользователя номера истока и стока.
 - Запускает алгоритм Форда-Фалкерсона и выводит результат.

5. Отладка и тестирование.

Среды разработки Code::Blocks IDE предоставляет все средства, необходимые при разработке и отладке многомодульной программы. Тестирование проводилось как в процессе разработки, так и после завершения написания программы.

Ввод графа вручную:

```
Выберите способ задания графа:
1. Ввести граф с клавиатуры
2. Сгенерировать случайный граф
1
Введите количество вершин: 4
Введите матрицу пропускных способностей (размер 4x4):
0 10 0 5
0 0 4 2
0 0 0 8
0 0 0 0
Граф:
0 10 0 5
0 0 4 2
0 0 0 8
0 0 0 0
Введите номер вершины-истока: 0
Введите номер вершины-стока: 3
Максимальный поток: 7
```

Рисунок 9 - результат работы программы

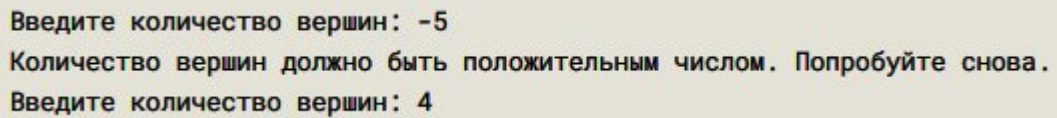
Генерация случайного графа:

```
Выберите способ задания графа:
1. Ввести граф с клавиатуры
2. Сгенерировать случайный граф
2
Введите количество вершин для случайного графа: 5
Граф:
0 12 0 7 0
0 0 9 0 3
0 0 0 0 5
0 0 0 0 0
0 0 0 0 0
Введите номер вершины-истока: 0
Введите номер вершины-стока: 4
Максимальный поток: 10
```

Рисунок 10 - результат работы программы

Результаты тестирования:

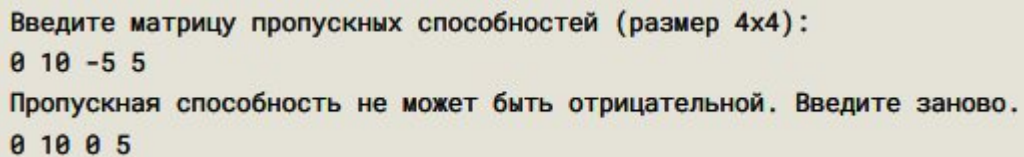
Некорректный ввод количества вершин:



Введите количество вершин: -5
Количество вершин должно быть положительным числом. Попробуйте снова.
Введите количество вершин: 4

Рисунок 11 - Сообщение об ошибке

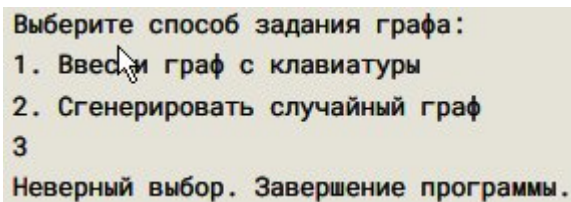
Некорректный ввод пропускной способности:



Введите матрицу пропускных способностей (размер 4x4):
0 10 -5 5
Пропускная способность не может быть отрицательной. Введите заново.
0 10 0 5

Рисунок 12 - Сообщение об ошибке

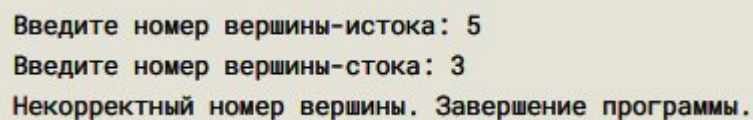
Некорректный выбор:



Выберите способ задания графа:
1. Ввести граф с клавиатуры
2. Сгенерировать случайный граф
3
Неверный выбор. Завершение программы.

Рисунок 13 - Сообщение об ошибке

Некорректный ввод истока и стока:



Введите номер вершины-истока: 5
Введите номер вершины-стока: 3
Некорректный номер вершины. Завершение программы.

Рисунок 14 - Сообщение об ошибке

Таблица 1 - Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Проверка запуска программы	Вывод информации о курсовом проекте	Верно
Проверка на неправильный ввод числа вершин генерируемого графа	Программа должна оповестить пользователя об ошибке и прекратить работу алгоритма	Верно
Проверка на неправильный выбор	Программа должна оповестить пользователя об неверном выборе которого нет в списке	Верно
Проверка на неправильный ввод истока и стока	Программа должна оповестить пользователя об ошибке	Верно

6. Ручной расчёт

Для проверки работы алгоритма Форда-Фалкерсона выполним ручной расчет на графе с 4 вершинами. Рассмотрим пример графа и пошагово пройдемся по алгоритму.

Рассмотрим граф с 4 вершинами (0, 1, 2, 3), где:

- Исток (source): вершина 0.
- Сток (sink): вершина 3.

Матрица пропускных способностей:

$$C = \begin{bmatrix} 0 & 10 & 0 & 5 \\ 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Рисунок 15 - Исходная матрица

1. Описание ребер:

- 0 -> 1: пропускная способность 10.
- 0 -> 3: пропускная способность 5.
- 1 -> 2: пропускная способность 4.
- 1 -> 3: пропускная способность 2.
- 2 -> 3: пропускная способность 8.

$$R = \begin{bmatrix} 0 & 10 & 0 & 5 \\ 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Рисунок 16 - Остаточная сеть

2. Поиск увеличивающего пути:

- Пропускные способности на пути:
 - о 0 -> 1: 10.
 - о 1 -> 3: 2.
- Минимальная пропускная способность: $\min(10, 2) = 2$.

$$R = \begin{bmatrix} 0 & 8 & 0 & 5 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 8 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

Рисунок 17 - измененная остаточная сеть

3. Повторяем:

- Ищем следующий увеличивающий путь.
- Найденный путь: 0 -> 3.

4. Определение минимальной остаточной пропускной способности

- Пропускные способности на пути:
 - о 0 -> 1: 8.
 - о 1 -> 2: 4.
 - о 2 -> 3: 8.
- Минимальная пропускная способность: $\min(8, 4, 8) = 4$.

5. Обновление потока и остаточной сети

- Увеличиваем поток на 4 вдоль пути 0 -> 1 -> 2 -> 3.
- Обновляем остаточную сеть:
 - о Ребро 0 -> 1: $8 - 4 = 4$.
 - о Ребро 1 -> 2: $4 - 4 = 0$.
 - о Ребро 2 -> 3: $8 - 4 = 4$.
 - о Добавляем обратные ребра:
 - 2 -> 1: $0 + 4 = 4$.
 - 3 -> 2: $0 + 4 = 4$.

$$R = \begin{bmatrix} 0 & 8 & 0 & 5 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 8 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

Рисунок 18 - Окончательный вариант остаточной сети

6. Итоговый поток:

- 0 -> 1 -> 3: поток 2.
- 0 -> 3: поток 5.
- 0 -> 1 -> 2 -> 3: поток 4.

Максимальный поток: 11.

Проверка с помощью программы:

```
Введите количество вершин: 4
Введите матрицу пропускных способностей (размер 4x4):
0 10 0 5
0 0 4 2
0 0 0 8
0 0 0 0
Введите номер вершины-истока: 0
Введите номер вершины-стока: 3
```

Рисунок 19 - Ввод данных

```
Максимальный поток: 11
```

Рисунок 20 - Вывод программы

Заключение

Алгоритм Форда-Фалкерсона — это итеративный метод, который на каждом шаге ищет путь для увеличения потока и обновляет остаточную сеть. Он прост в реализации, но может быть медленным для графов с большими пропускными способностями. Для улучшения производительности используются более продвинутые алгоритмы, такие как Эдмондс-Карп (BFS вместо DFS) или Диница.

Список литературы.

1. Электронный ресурс: <https://msdn.microsoft.com> Дата обращения: 05.11.2017.
2. Электронный ресурс: <https://ru.wikipedia.org/wiki/> (Дата обращения: 05.11.2017).
3. Электронный ресурс: <http://www.cyberforum.ru/> Дата обращения: 05.11.2017.
4. Дискретная математика для программистов, Р. Хаггарты ISBN 5-94836-016-4, 0-201-73047-2, 2005.
5. Discrete Mathematics with Combinatorics, Джеймс Андерсон — 2-е изд., 2004. — ISBN 978-5-9775-0445-4.

Приложение А

Листинги программы

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <cstdlib>
#include <ctime>
#include <windows.h>

using namespace std;

// Функция для поиска увеличивающего пути с помощью BFS
bool bfs(vector<vector<int>>& residualGraph, vector<int>& parent, int
source, int sink) {
    int V = residualGraph.size();
    vector<bool> visited(V, false);
    queue<int> q;
    q.push(source);
    visited[source] = true;
    parent[source] = -1;

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

        for (int v = 0; v < V; v++) {
            if (!visited[v] && residualGraph[u][v] > 0) {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
                if (v == sink) {
                    return true;
                }
            }
        }
    }
}
```

```

    }

    return false;
}

// Реализация алгоритма Форда-Фалкерсона
int fordFulkerson(vector<vector<int>>& graph, int source, int sink) {
    int V = graph.size();
    vector<vector<int>> residualGraph = graph; // Остаточная сеть
    vector<int> parent(V); // Массив для хранения пути
    int maxFlow = 0;

    // Пока существует увеличивающий путь
    while (bfs(residualGraph, parent, source, sink)) {
        int pathFlow = INT_MAX;

        // Находим минимальный поток в пути
        for (int v = sink; v != source; v = parent[v]) {
            int u = parent[v];
            pathFlow = min(pathFlow, residualGraph[u][v]);
        }

        // Обновляем остаточную сеть
        for (int v = sink; v != source; v = parent[v]) {
            int u = parent[v];
            residualGraph[u][v] -= pathFlow;
            residualGraph[v][u] += pathFlow;
        }

        // Добавляем поток к общему потоку
        maxFlow += pathFlow;
    }

    return maxFlow;
}

// Функция для ввода графа с клавиатуры
vector<vector<int>> inputGraph() {

```

```

int V;
cout << "Введите количество вершин: ";
cin >> V;

// Защита от некорректного ввода количества вершин
if (V <= 0) {
    cout << "Количество вершин должно быть положительным
числом. Попробуйте снова." << endl;
    return inputGraph(); // Рекурсивный вызов для повторного ввода
}

vector<vector<int>> graph(V, vector<int>(V, 0));

cout << "Введите матрицу пропускных способностей (размер " <<
V << "x" << V << "):" << endl;
for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
        cin >> graph[i][j];
        // Защита от отрицательных пропускных способностей
        if (graph[i][j] < 0) {
            cout << "Пропускная способность не может быть
отрицательной. Введите заново." << endl;
            j--; // Повторный ввод для текущего элемента
        }
    }
}

return graph;
}

// Функция для генерации случайного графа
vector<vector<int>> generateRandomGraph(int V) {
    vector<vector<int>> graph(V, vector<int>(V, 0));

    srand(time(0)); // Инициализация генератора случайных чисел

    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {

```

```

        if (i != j) {
            graph[i][j] = rand() % 20; // Случайные значения от 0 до 19
        }
    }
}

return graph;
}

int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    int choice;
    cout << "Выберите способ задания графа:" << endl;
    cout << "1. Ввести граф с клавиатуры" << endl;
    cout << "2. Сгенерировать случайный граф" << endl;
    cin >> choice;

    // Защита от некорректного выбора
    if (choice != 1 && choice != 2) {
        cout << "Неверный выбор. Завершение программы." << endl;
        return 1;
    }

    vector<vector<int>> graph;
    int V;

    if (choice == 1) {
        graph = inputGraph();
    } else if (choice == 2) {
        cout << "Введите количество вершин для случайного графа: ";
        cin >> V;
        // Защита от некорректного ввода количества вершин
        if (V <= 0) {
            cout << "Количество вершин должно быть положительным
числом. Завершение программы." << endl;
            return 1;
        }
    }
}

```

```

    }
    graph = generateRandomGraph(V);
}

// Вывод графа
cout << "Граф:" << endl;
for (const auto& row : graph) {
    for (int capacity : row) {
        cout << capacity << " ";
    }
    cout << endl;
}

int source, sink;
cout << "Введите номер вершины-источка: ";
cin >> source;
cout << "Введите номер вершины-стока: ";
cin >> sink;

// Защита от некорректного ввода источника и стока
if (source < 0 || source >= graph.size() || sink < 0 || sink >=
graph.size()) {
    cout << "Некорректный номер вершины. Завершение
программы." << endl;
    return 1;
}

cout << "Максимальный поток: " << fordFulkerson(graph, source,
sink) << endl;

return 0;
}

```