МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Построение и анализ алгоритмов»
Тема: Потоки в сети

Студентка гр. 8383 _____ Ишанина Л. Н.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2020

Цель работы.

Реализовать алгоритм Форда-Фалкерсона, найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро.

Постановка задачи.

Вариант 3. Поиск в глубину. Рекурсивная реализация.

Задание.

Найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро, используя алгоритм Форда-Фалкерсона.

Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа - пропускной способности (веса).

Входные данные:

N – количество ориентированных рёбер графа

V_0 – исток

V_N – сток

$V_iV_jW_{ij}$ – ребро графа

$V_iV_jW_{ij}$ – ребро графа

…

Выходные данные:

P_{max} – величина максимального потока

$V_iV_jW_{ij}$ – ребро графа сфактической величиной протекающего потока

$V_iV_jW_{ij}$ – ребро графа с фактической величиной протекающего потока

…

В ответе выходные рёбра отсортируйте в лексикографическом порядке по первой вершине, потом по второй (в ответе должны присутствовать все указанные входные рёбра, даже если поток в них равен 0).

Пример входных данных

7

a

f

ab 7

a c 6

b d 6

c f 9

d e 3

d f 4

e c 2

Пример выходных данных

12

a b 6

a c 6

b d 6

c f 8

d e 2

d f 4

e c 2

Описание используемого класса.

Класс ориентированного ребра графа(class Path):

Класс ориентированного ребра графа состоит из следующих свойств:

- char nameFrom – имя вершины откуда исходит ребро;

- char nameOut – имя вершины куда входит ребро;
- int bandwidth – пропускная способность;
- int flow – поток;

Класс ориентированного ребра графа содержит следующие методы:

- Path(char nameFrom, char nameOut, int bandwidth) – конструктор, принимающий на вход имя вершины куда идём, имя вершины куда идём и пропускную способность.
- void setFlow(int flow) – функция для переопределения значения потока;
- char getNameFrom() – функция, которая возвращает значение имени вершины откуда исходит ребро;
- char getNameOut() – функция, которая возвращает значение имени вершины куда входит ребро;
- int getBandwidth() – функция, которая возвращает значение пропускной способности;
- int getFlow() – функция, которая возвращает значение потока;

Описание алгоритма.

Решение поставленной задачи осуществляется с помощью рекурсивной функции bool findPath(std::vector<Path>& paths, std::vector<Path*>& local, std::vector<Path*>& local2, char myPoint, char endPoint, int depth), которая принимает на вход: исходный вектор, в котором записаны все ребра с величиной пропускаемого потока; вектор, в который записывается найденный путь; вектор, в который записываются посещенные ребра; вершина, с которой функция будет начинать поиск; вершина, до которой функция ищет путь; и переменная, необходимая для визуального показания рекурсии алгоритма.

Функция начинает работу прежде всего с проверки, является ли текущая вершина искомой, если нет, то происходит отбор всех ребер,

которые исходят из текущей вершины. Нужные ребра записываются во временный граф ребер (localPaths). Далее, если таких ребер несколько, то они сортируются по наименьшему значению разности между пропускной способности и потоком.

Следующий шаг. Начинается в цикле проход по отсортированным путям. Происходит проверка, что ребро не переполнено, т.е. значение потока меньше значения пропускной способности. Затем проверяется, посещалась ли уже вершина, куда входит текущее ребро. Если обе проверки пройдены, то это ребро записывается в вектор просмотренных ребер. После чего, рекурсивно вызывается функция поиска пути, но теперь в качестве текущей вершины будет вершина, в которую входило данное ребро.

Также рассмотрим следующий случай. Если на проверке уже отобранных и отсортированных ребер окажется так, что ребро переполнено, то цикл for продолжится, просто перейдя к следующему ребру, также выполнив все проверки. Но если среди всех доступных для вершины ребер не будет того, которое не переполнено, то функция сделает откат назад, к предыдущей рассматриваемой вершине.

Как только на очередном рекурсивном вызове будет достигнута искомая вершина(сток), то функция возвратит true и рекурсия вернется на шаг назад, затем в вектор ответа (local) будет записано ребро и функция снова вернет true. Получается, что все ребра пути будут записаны в вектор ответа, но в обратном порядке.

Затем, после того как функция поиска пути вернула true, происходит вызов функции поиска минимальной разности между величиной пропускной способности и величиной потока. Далее найденный минимум прибавляется ко всем потокам найденного пути и также к значению максимального потока.

После выхода из функции происходит очищение вектора ответа(в котором записан текущий путь до стока) и вектора пройденных ребер. И цикл в main снова повторяется, снова запускается функция поиска путей с

текущей начальной вершиной. Данный цикл завершается, когда функция поиска путей не сможет найти ни один путь до искомой вершины.

После завершения цикла, происходит сортировка всех ребер в лексикографическом порядке с помощью функции sort() и написанного к ней компаратора.

В конце main происходит вывод ответа, полностью соответствующий всем требованиям. А именно, сначала выводится найденное значение максимального потока, а уже за ним все ребра графа с фактической величиной протекающего потока.

Для удобства весь код оснащен выводами промежуточных значений, а также в коде присутствуют комментарии.

Описание main ( ) :

В функции прописан ввод количества ориентированных рёбер графа, исток, сток и ребра графа. Также в main происходит вызов функций и выводы промежуточных данных на консоль.

Описание дополнительных функций.

Функция void findMin(std::vector<Path*>& local, int* maxFlow, int depth), принимает вектор, в котором хранятся ребра найденного пути, переменную, которая считает максимальный поток и переменную, которая отвечает за демонстрацию рекурсии.

Функция bool comp2(Path* a, Path* b) принимает 2 переменные класса ребер. Данная функция является компаратором для сортировки ребер по наименьшему значению разности между пропускной способностью и потоком.

Функция bool isVisitedPath(std::vector<Path*>& local2, char element, int depth) принимает на вход вектор просмотренных ребер, переменную типа char и переменную, которая отвечает за демонстрацию рекурсии. Функция

выполняет проверку на то, что вершину, куда входит ребро, ещё не посещали.

Сложности.

Сложность алгоритма по времени:

Добавляя поток увеличивающего пути к уже имеющемуся потоку, максимальный поток будет получен, когда нельзя будет найти увеличивающий путь. В целых числах сложность по времени будет равняться O (E * F), E – число ребер в графе, F – максимальный поток. Это можно объяснить тем, что каждый увеличивающий путь может быть найден за O(E) и увеличивает поток как минимум на 1.

Сложность алгоритма по памяти:

Сложность по памяти - O(|E|), где E – количество ребер в графе. Потому что максимальный размер вектора, а, следовательно, размер выделенной памяти, будет зависеть от количества ребер.

Тестирование.

Входные данные:

7
a
f
a b 7
a c 6
b d 6
c f 9
d e 3
d f 4
e c 2

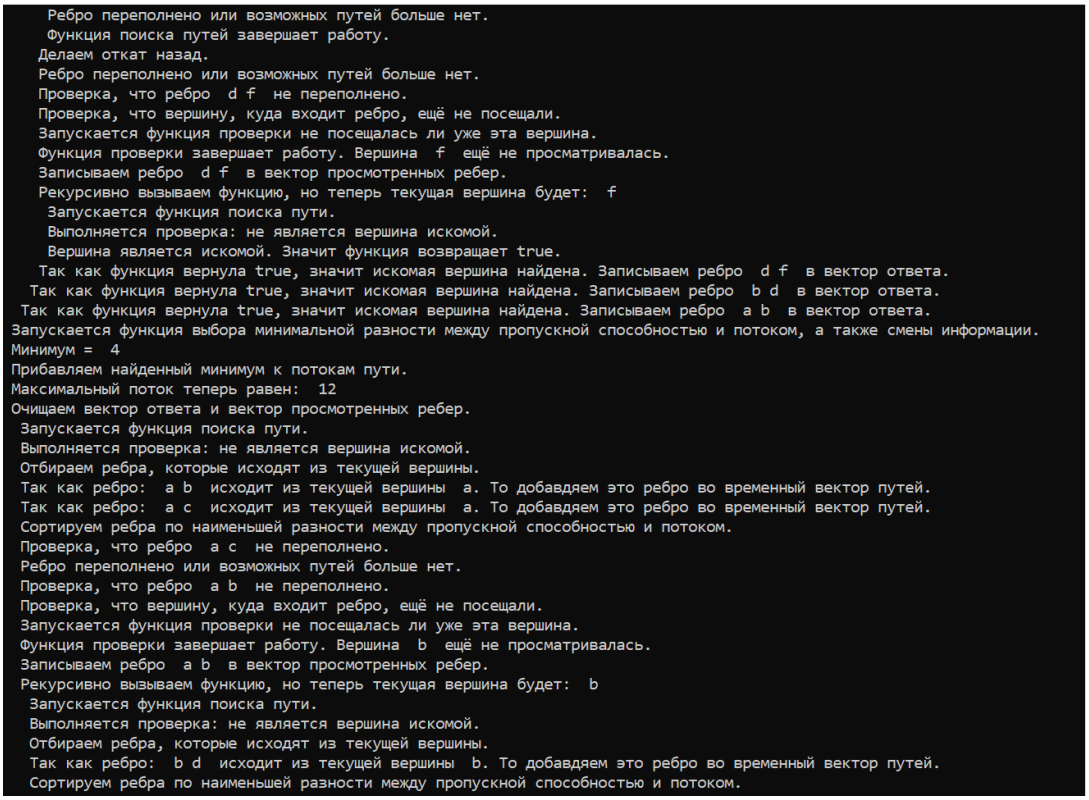
Результат работы программы:

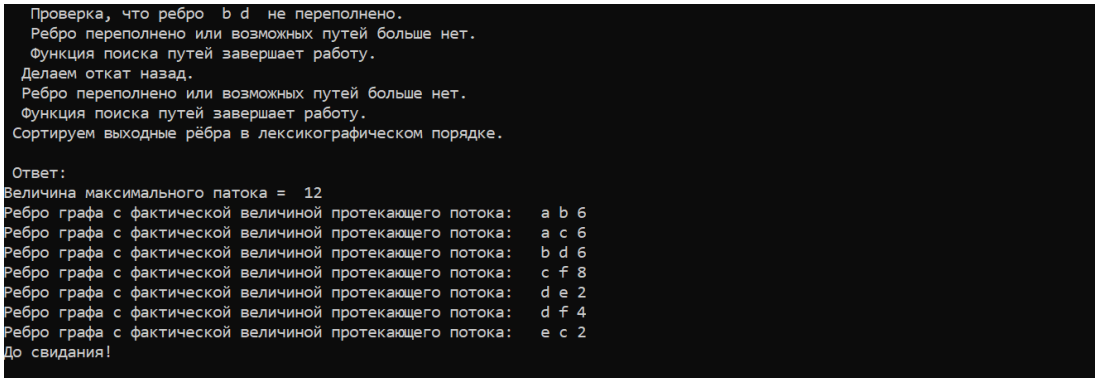
```
Здравствуйте! Введите ,пожалуйста, количество ориентированных рёбер графа, исток, сток и ребра графа.
7
a
f
a b 7
a c 6
b d 6
c f 9
d e 3
d f 4
e c 2
Вызывается функция поиска пути.
 Запускается функция поиска пути.
 Выполняется проверка: не является вершина искомой.
 Отбираем ребра, которые исходят из текущей вершины.
 Так как ребро:  a b  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Так как ребро:  a c  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
 Проверка, что ребро  a c  не переполнено.
 Проверка, что вершину, куда входит ребро, ещё не посещали.
 Запускается функция проверки не посещалась ли уже эта вершина.
 Функция проверки завершает работу. Вершина  c  ещё не просматривалась.
 Записываем ребро  a c  в вектор просмотренных ребер.
 Рекурсивно вызываем функцию, но теперь текущая вершина будет:  c
  Запускается функция поиска пути.
  Выполняется проверка: не является вершина искомой.
  Отбираем ребра, которые исходят из текущей вершины.
  Так как ребро:  c f  исходит из текущей вершины  c. То добавдяем это ребро во временный вектор путей.
  Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
  Проверка, что ребро  c f  не переполнено.
  Проверка, что вершину, куда входит ребро, ещё не посещали.
  Запускается функция проверки не посещалась ли уже эта вершина.
  Функция проверки завершает работу. Вершина  f  ещё не просматривалась.
  Записываем ребро  c f  в вектор просмотренных ребер.
  Рекурсивно вызываем функцию, но теперь текущая вершина будет:  f
   Запускается функция поиска пути.
   Выполняется проверка: не является вершина искомой.
   Вершина является искомой. Значит функция возвращает true.
  Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  c f  в вектор ответа.
 Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  a c  в вектор ответа.
```

```
Запускается функция выбора минимальной разности между пропускной способностью и потоком, а также смены информации.
Минимум =  6
Прибавляем найденный минимум к потокам пути.
Максимальный поток теперь равен:  6
Очищаем вектор ответа и вектор просмотренных ребер.
 Запускается функция поиска пути.
 Выполняется проверка: не является вершина искомой.
 Отбираем ребра, которые исходят из текущей вершины.
 Так как ребро:  a b  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Так как ребро:  a c  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
 Проверка, что ребро  a c  не переполнено.
 Ребро переполнено или возможных путей больше нет.
 Проверка, что ребро  a b  не переполнено.
 Проверка, что вершину, куда входит ребро, ещё не посещали.
 Запускается функция проверки не посещалась ли уже эта вершина.
 Функция проверки завершает работу. Вершина  b  ещё не просматривалась.
 Записываем ребро  a b  в вектор просмотренных ребер.
 Рекурсивно вызываем функцию, но теперь текущая вершина будет:  b
  Запускается функция поиска пути.
  Выполняется проверка: не является вершина искомой.
  Отбираем ребра, которые исходят из текущей вершины.
  Так как ребро:  b d  исходит из текущей вершины  b. То добавдяем это ребро во временный вектор путей.
  Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
  Проверка, что ребро  b d  не переполнено.
  Проверка, что вершину, куда входит ребро, ещё не посещали.
  Запускается функция проверки не посещалась ли уже эта вершина.
  Функция проверки завершает работу. Вершина  d  ещё не просматривалась.
  Записываем ребро  b d  в вектор просмотренных ребер.
  Рекурсивно вызываем функцию, но теперь текущая вершина будет:  d
   Запускается функция поиска пути.
   Выполняется проверка: не является вершина искомой.
   Отбираем ребра, которые исходят из текущей вершины.
   Так как ребро:  d e  исходит из текущей вершины  d. То добавдяем это ребро во временный вектор путей.
   Так как ребро:  d f  исходит из текущей вершины  d. То добавдяем это ребро во временный вектор путей.
   Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
   Проверка, что ребро  d e  не переполнено.
   Проверка, что вершину, куда входит ребро, ещё не посещали.
   Запускается функция проверки не посещалась ли уже эта вершина.
   Функция проверки завершает работу. Вершина  e  ещё не просматривалась.
```

```
  Записываем ребро  d e  в вектор просмотренных ребер.
  Рекурсивно вызываем функцию, но теперь текущая вершина будет:  e
   Запускается функция поиска пути.
   Выполняется проверка: не является вершина искомой.
   Отбираем ребра, которые исходят из текущей вершины.
   Так как ребро:  e c  исходит из текущей вершины  e. То добавдяем это ребро во временный вектор путей.
   Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
   Проверка, что ребро  e c  не переполнено.
   Проверка, что вершину, куда входит ребро, ещё не посещали.
   Запускается функция проверки не посещалась ли уже эта вершина.
   Функция проверки завершает работу. Вершина  c  ещё не просматривалась.
   Записываем ребро  e c  в вектор просмотренных ребер.
   Рекурсивно вызываем функцию, но теперь текущая вершина будет:  c
    Запускается функция поиска пути.
    Выполняется проверка: не является вершина искомой.
    Отбираем ребра, которые исходят из текущей вершины.
    Так как ребро:  c f  исходит из текущей вершины  c. То добавдяем это ребро во временный вектор путей.
    Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
    Проверка, что ребро  c f  не переполнено.
    Проверка, что вершину, куда входит ребро, ещё не посещали.
    Запускается функция проверки не посещалась ли уже эта вершина.
    Функция проверки завершает работу. Вершина  f  ещё не просматривалась.
    Записываем ребро  c f  в вектор просмотренных ребер.
    Рекурсивно вызываем функцию, но теперь текущая вершина будет:  f
     Запускается функция поиска пути.
     Выполняется проверка: не является вершина искомой.
     Вершина является искомой. Значит функция возвращает true.
    Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  c f  в вектор ответа.
   Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  e c  в вектор ответа.
  Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  d e  в вектор ответа.
 Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  b d  в вектор ответа.
Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  a b  в вектор ответа.
Запускается функция выбора минимальной разности между пропускной способностью и потоком, а также смены информации.
Минимум =  2
Прибавляем найденный минимум к потокам пути.
Максимальный поток теперь равен:  8
Очищаем вектор ответа и вектор просмотренных ребер.
 Запускается функция поиска пути.
 Выполняется проверка: не является вершина искомой.
 Отбираем ребра, которые исходят из текущей вершины.
```

```
 Так как ребро:  a b  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Так как ребро:  a c  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
 Проверка, что ребро  a c  не переполнено.
 Ребро переполнено или возможных путей больше нет.
 Проверка, что ребро  a b  не переполнено.
 Проверка, что вершину, куда входит ребро, ещё не посещали.
 Запускается функция проверки не посещалась ли уже эта вершина.
 Функция проверки завершает работу. Вершина  b  ещё не просматривалась.
 Записываем ребро  a b  в вектор просмотренных ребер.
 Рекурсивно вызываем функцию, но теперь текущая вершина будет:  b
  Запускается функция поиска пути.
  Выполняется проверка: не является вершина искомой.
  Отбираем ребра, которые исходят из текущей вершины.
  Так как ребро:  b d  исходит из текущей вершины  b. То добавдяем это ребро во временный вектор путей.
  Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
  Проверка, что ребро  b d  не переполнено.
  Проверка, что вершину, куда входит ребро, ещё не посещали.
  Запускается функция проверки не посещалась ли уже эта вершина.
  Функция проверки завершает работу. Вершина  d  ещё не просматривалась.
  Записываем ребро  b d  в вектор просмотренных ребер.
  Рекурсивно вызываем функцию, но теперь текущая вершина будет:  d
   Запускается функция поиска пути.
   Выполняется проверка: не является вершина искомой.
   Отбираем ребра, которые исходят из текущей вершины.
   Так как ребро:  d e  исходит из текущей вершины  d. То добавдяем это ребро во временный вектор путей.
   Так как ребро:  d f  исходит из текущей вершины  d. То добавдяем это ребро во временный вектор путей.
   Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
   Проверка, что ребро  d e  не переполнено.
   Проверка, что вершину, куда входит ребро, ещё не посещали.
   Запускается функция проверки не посещалась ли уже эта вершина.
   Функция проверки завершает работу. Вершина  e  ещё не просматривалась.
   Записываем ребро  d e  в вектор просмотренных ребер.
   Рекурсивно вызываем функцию, но теперь текущая вершина будет:  e
    Запускается функция поиска пути.
    Выполняется проверка: не является вершина искомой.
    Отбираем ребра, которые исходят из текущей вершины.
    Так как ребро:  e c  исходит из текущей вершины  e. То добавдяем это ребро во временный вектор путей.
    Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
    Проверка, что ребро  e c  не переполнено.
```





Входные данные:

8
a
h
a b 5
a c 4
a d 1
b g 1
c e 2
c f 3
d e 6
e h 4
f h 4

```
Здравствуйте! Введите ,пожалуйста, количество ориентированных рёбер графа, исток, сток и ребра графа.
8
a
h
a b 5
a c 4
a d 1
b g 1
c e 2
c f 3
d e 6
e h 4
Вызывается функция поиска пути.
 Запускается функция поиска пути.
 Выполняется проверка: не является вершина искомой.
 Отбираем ребра, которые исходят из текущей вершины.
 Так как ребро:  a b  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Так как ребро:  a c  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Так как ребро:  a d  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
 Проверка, что ребро  a d  не переполнено.
 Проверка, что вершину, куда входит ребро, ещё не посещали.
 Запускается функция проверки не посещалась ли уже эта вершина.
 Функция проверки завершает работу. Вершина  d  ещё не просматривалась.
 Записываем ребро  a d  в вектор просмотренных ребер.
 Рекурсивно вызываем функцию, но теперь текущая вершина будет:  d
  Запускается функция поиска пути.
  Выполняется проверка: не является вершина искомой.
  Отбираем ребра, которые исходят из текущей вершины.
  Так как ребро:  d e  исходит из текущей вершины  d. То добавдяем это ребро во временный вектор путей.
  Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
  Проверка, что ребро  d e  не переполнено.
  Проверка, что вершину, куда входит ребро, ещё не посещали.
  Запускается функция проверки не посещалась ли уже эта вершина.
  Функция проверки завершает работу. Вершина  e  ещё не просматривалась.
  Записываем ребро  d e  в вектор просмотренных ребер.
  Рекурсивно вызываем функцию, но теперь текущая вершина будет:  e
   Запускается функция поиска пути.
   Выполняется проверка: не является вершина искомой.
   Отбираем ребра, которые исходят из текущей вершины.
   Так как ребро:  e h  исходит из текущей вершины  e. То добавдяем это ребро во временный вектор путей.
   Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
   Проверка, что ребро  e h  не переполнено.
   Проверка, что вершину, куда входит ребро, ещё не посещали.
   Запускается функция проверки не посещалась ли уже эта вершина.
   Функция проверки завершает работу. Вершина  h  ещё не просматривалась.
   Записываем ребро  e h  в вектор просмотренных ребер.
   Рекурсивно вызываем функцию, но теперь текущая вершина будет:  h
    Запускается функция поиска пути.
    Выполняется проверка: не является вершина искомой.
    Вершина является искомой. Значит функция возвращает true.
   Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  e h  в вектор ответа.
  Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  d e  в вектор ответа.
 Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  a d  в вектор ответа.
Запускается функция выбора минимальной разности между пропускной способностью и потоком, а также смены информации.
Минимум =  1
Прибавляем найденный минимум к потокам пути.
Максимальный поток теперь равен:  1
Очищаем вектор ответа и вектор просмотренных ребер.
 Запускается функция поиска пути.
 Выполняется проверка: не является вершина искомой.
 Отбираем ребра, которые исходят из текущей вершины.
 Так как ребро:  a b  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Так как ребро:  a c  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Так как ребро:  a d  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
 Проверка, что ребро  a d  не переполнено.
 Ребро переполнено или возможных путей больше нет.
 Проверка, что ребро  a c  не переполнено.
 Проверка, что вершину, куда входит ребро, ещё не посещали.
 Запускается функция проверки не посещалась ли уже эта вершина.
 Функция проверки завершает работу. Вершина  c  ещё не просматривалась.
 Записываем ребро  a c  в вектор просмотренных ребер.
 Рекурсивно вызываем функцию, но теперь текущая вершина будет:  c
  Запускается функция поиска пути.
  Выполняется проверка: не является вершина искомой.
  Отбираем ребра, которые исходят из текущей вершины.
  Так как ребро:  c e  исходит из текущей вершины  c. То добавдяем это ребро во временный вектор путей.
```

```
  Так как ребро:  c f  исходит из текущей вершины  c. То добавдяем это ребро во временный вектор путей.
  Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
  Проверка, что ребро  c e  не переполнено.
  Проверка, что вершину, куда входит ребро, ещё не посещали.
  Запускается функция проверки не посещалась ли уже эта вершина.
  Функция проверки завершает работу. Вершина  e  ещё не просматривалась.
  Записываем ребро  c e  в вектор просмотренных ребер.
  Рекурсивно вызываем функцию, но теперь текущая вершина будет:  e
   Запускается функция поиска пути.
   Выполняется проверка: не является вершина искомой.
   Отбираем ребра, которые исходят из текущей вершины.
   Так как ребро:  e h  исходит из текущей вершины  e. То добавдяем это ребро во временный вектор путей.
   Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
   Проверка, что ребро  e h  не переполнено.
   Проверка, что вершину, куда входит ребро, ещё не посещали.
   Запускается функция проверки не посещалась ли уже эта вершина.
   Функция проверки завершает работу. Вершина  h  ещё не просматривалась.
   Записываем ребро  e h  в вектор просмотренных ребер.
   Рекурсивно вызываем функцию, но теперь текущая вершина будет:  h
    Запускается функция поиска пути.
    Выполняется проверка: не является вершина искомой.
    Вершина является искомой. Значит функция возвращает true.
   Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  e h  в вектор ответа.
  Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  c e  в вектор ответа.
 Так как функция вернула true, значит искомая вершина найдена. Записываем ребро  a c  в вектор ответа.
Запускается функция выбора минимальной разности между пропускной способностью и потоком, а также смены информации.
Минимум =  2
Прибавляем найденный минимум к потокам пути.
Максимальный поток теперь равен:  3
Очищаем вектор ответа и вектор просмотренных ребер.
 Запускается функция поиска пути.
 Выполняется проверка: не является вершина искомой.
 Отбираем ребра, которые исходят из текущей вершины.
 Так как ребро:  a b  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Так как ребро:  a c  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Так как ребро:  a d  исходит из текущей вершины  a. То добавдяем это ребро во временный вектор путей.
 Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
 Проверка, что ребро  a d  не переполнено.
 Ребро переполнено или возможных путей больше нет.
```

```
Проверка, что ребро  a c  не переполнено.
Проверка, что вершину, куда входит ребро, ещё не посещали.
Запускается функция проверки не посещалась ли уже эта вершина.
Функция проверки завершает работу. Вершина  c  ещё не просматривалась.
Записываем ребро  a c  в вектор просмотренных ребер.
Рекурсивно вызываем функцию, но теперь текущая вершина будет:  c
 Запускается функция поиска пути.
 Выполняется проверка: не является вершина искомой.
 Отбираем ребра, которые исходят из текущей вершины.
 Так как ребро:  c e  исходит из текущей вершины  c. То добавдяем это ребро во временный вектор путей.
 Так как ребро:  c f  исходит из текущей вершины  c. То добавдяем это ребро во временный вектор путей.
 Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
 Проверка, что ребро  c e  не переполнено.
 Ребро переполнено или возможных путей больше нет.
 Проверка, что ребро  c f  не переполнено.
 Проверка, что вершину, куда входит ребро, ещё не посещали.
 Запускается функция проверки не посещалась ли уже эта вершина.
 Функция проверки завершает работу. Вершина  f  ещё не просматривалась.
 Записываем ребро  c f  в вектор просмотренных ребер.
 Рекурсивно вызываем функцию, но теперь текущая вершина будет:  f
  Запускается функция поиска пути.
  Выполняется проверка: не является вершина искомой.
  Отбираем ребра, которые исходят из текущей вершины.
  Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
  Функция поиска путей завершает работу.
 Делаем откат назад.
 Ребро переполнено или возможных путей больше нет.
 Функция поиска путей завершает работу.
Делаем откат назад.
Ребро переполнено или возможных путей больше нет.
Проверка, что ребро  a b  не переполнено.
Проверка, что вершину, куда входит ребро, ещё не посещали.
Запускается функция проверки не посещалась ли уже эта вершина.
Функция проверки завершает работу. Вершина  b  ещё не просматривалась.
Записываем ребро  a b  в вектор просмотренных ребер.
Рекурсивно вызываем функцию, но теперь текущая вершина будет:  b
 Запускается функция поиска пути.
 Выполняется проверка: не является вершина искомой.
 Отбираем ребра, которые исходят из текущей вершины.
```

```
  Так как ребро:  b g  исходит из текущей вершины  b. То добавдяем это ребро во временный вектор путей.
  Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
  Проверка, что ребро  b g  не переполнено.
  Проверка, что вершину, куда входит ребро, ещё не посещали.
  Запускается функция проверки не посещалась ли уже эта вершина.
  Функция проверки завершает работу. Вершина  g  ещё не просматривалась.
  Записываем ребро  b g  в вектор просмотренных ребер.
  Рекурсивно вызываем функцию, но теперь текущая вершина будет:  g
   Запускается функция поиска пути.
   Выполняется проверка: не является вершина искомой.
   Отбираем ребра, которые исходят из текущей вершины.
   Сортируем ребра по наименьшей разности между пропускной способностью и потоком.
   Функция поиска путей завершает работу.
  Делаем откат назад.
  Ребро переполнено или возможных путей больше нет.
  Функция поиска путей завершает работу.
 Делаем откат назад.
 Ребро переполнено или возможных путей больше нет.
 Функция поиска путей завершает работу.
Сортируем выходные рёбра в лексикографическом порядке.

 Ответ:
Величина максимального патока =  3
Ребро графа с фактической величиной протекающего потока:   a b 0
Ребро графа с фактической величиной протекающего потока:   a c 2
Ребро графа с фактической величиной протекающего потока:   a d 1
Ребро графа с фактической величиной протекающего потока:   b g 0
Ребро графа с фактической величиной протекающего потока:   c e 2
Ребро графа с фактической величиной протекающего потока:   c f 0
Ребро графа с фактической величиной протекающего потока:   d e 1
Ребро графа с фактической величиной протекающего потока:   e h 3
До свидания!
```

Тест на крайние значения:

Входные данные:

0

a

b

Вывод:

```
0
a
b
Введено нулевое количество ориентированных ребер графа!
```

Выводы.

В ходе выполнения лабораторной работы был реализован алгоритм Форда-Фалкерсона, который находит максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>      // std::setw

class Path {
private:
    char nameFrom;
    char nameOut;
    int bandwidth;//пропускная способность
    int flow = 0;//поток

public:
    Path(char nameFrom, char nameOut, int bandwidth) : nameFrom(nameFrom), nameOut(nameOut),
bandwidth(bandwidth) {}

    void setFlow(int flow) {
        Path::flow = flow;
    }

    char getNameFrom() const {
        return nameFrom;
    }

    char getNameOut() const {
        return nameOut;
    }

    int getBandwidth() const {
        return bandwidth;
    }

    int getFlow() const {
        return flow;
    }


};

void findMin(std::vector<Path*>& local, int* maxFlow, long int depth) {//функция для выбора
минимальной разности пропускной спомобности и потока
    std::cout << std::setw(depth + 1) << ' ' << "Запускается функция выбора минимальной
разности между пропускной способностью и потоком, а также смены информации." << std::endl;
    int Min = local->front()->getBandwidth();   //front() Возвращает ссылку на первый элемент
в векторном контейнере
    for (Path* path : *local) {
        if (Min > (path->getBandwidth() - path->getFlow())) {
            Min = path->getBandwidth() - path->getFlow();
        }
    }
    std::cout << std::setw(depth + 1) << ' ' << "Минимум =  " << Min << std::endl;
    std::cout << std::setw(depth + 1) << ' ' << "Прибавляем найденный минимум к потокам пути."
<< std::endl;
    for (Path* path : *local) {
        path->setFlow(path->getFlow() + Min);//теперь прибавляем найденный минимум к потоку
    }

    *maxFlow = *maxFlow + Min;//увеличиваем значение максимального потока
    std::cout << std::setw(depth + 1) << ' ' << "Максимальный поток теперь равен:  " <<
*maxFlow << std::endl;
}

bool comp(Path a, Path b) {
```

```
    if (a.getNameFrom() != b.getNameFrom()) {

        return a.getNameFrom() < b.getNameFrom();
    }
    else {
        return a.getNameOut() < b.getNameOut();
    }
}

bool comp2(Path* a, Path* b) {//компаратор используется для выбора, куда идти. Т.е. например,
есть 2 пути, один 10/40 второй 15/30. Надо идти во второй, т к 30 - 15 < 40 - 10

    return (a->getBandwidth() - a->getFlow()) <= (b->getBandwidth() - b->getFlow());
}

bool isVisitedPath(std::vector<Path*>& local, char element, long int depth) {
    std::cout << std::setw(depth + 1) << ' ' << "Запускается функция проверки не посещалась ли
уже эта вершина." << std::endl;
    for (Path* path : *local) {
        if (element == path->getNameFrom()) {
            std::cout << std::setw(depth + 1) << ' ' << "Функция проверки завершает работу.
Вершина  " << element << "  уже просматривалась. А значит по этому ребру не пойдем." <<
std::endl;
            return false;
        }
    }
    std::cout << std::setw(depth + 1) << ' ' << "Функция проверки завершает работу. Вершина  "
<< element << "  ещё не просматривалась." << std::endl;
    return true;
}

bool findPath(std::vector<Path>& paths, std::vector<Path*>& local, std::vector<Path*>& local2,
char myPoint, char* endPoint, long int depth) {
    depth++;

    std::cout << std::setw(depth + 1) << ' ' << "Запускается функция поиска пути." <<
std::endl;

    std::cout << std::setw(depth + 1) << ' ' << "Выполняется проверка: не является вершина
искомой." << std::endl;
    if (myPoint == *endPoint) {
        std::cout << std::setw(depth + 1) << ' ' << "Вершина является искомой. Значит функция
возвращает true." << std::endl;
        return true;
    }

    std::vector<Path*> localPaths;
    localPaths.reserve(0);

    std::cout << std::setw(depth + 1) << ' ' << "Отбираем ребра, которые исходят из текущей
вершины." << std::endl;
    for (auto& path : paths) {//отбираем какие ребра исходят из текущей вершины
        if (path.getNameFrom() == myPoint) {
            std::cout << std::setw(depth + 1) << ' ' << "Так как ребро:  " <<
path.getNameFrom() << ' ' << path.getNameOut() << "  исходит из текущей вершины  " << myPoint
<< ". То добавдяем это ребро во временный вектор путей." << std::endl;

            localPaths.emplace_back(&path);
        }
    }

    std::cout << std::setw(depth + 1) << ' ' << "Сортируем ребра по наименьшей разности между
пропускной способностью и потоком." << std::endl;
    std::sort(localPaths.begin(), localPaths.end(), comp2);//выбираем куда идти прежде всего

    for (Path* path : localPaths) {
        std::cout << std::setw(depth + 1) << ' ' << "Проверка, что ребро  " << path-
>getNameFrom() << ' ' << path->getNameOut() << "  не переполнено." << std::endl;
```

```
        if (path->getFlow() < path->getBandwidth()) {//проверка, что поток пути меньше
пропускной способности
            std::cout << std::setw(depth + 1) << ' ' << "Проверка, что вершину, куда входит
ребро, ещё не посещали." << std::endl;
            if (isVisitedPath(local2, path->getNameOut(), depth)) {//если мы ещё не посещали
вершину
                std::cout << std::setw(depth + 1) << ' ' << "Записываем ребро  " << path-
>getNameFrom() << ' ' << path->getNameOut() <<  "  в вектор просмотренных ребер." <<
std::endl;
                local2.emplace_back(path);
                std::cout << std::setw(depth + 1) << ' ' << "Рекурсивно вызываем функцию, но
теперь текущая вершина будет:  " << path->getNameOut() << std::endl;
                if (findPath(paths, local, local2, path->getNameOut(), endPoint, depth))
{//рекурсивно вызываем функцию с непросмотренной вершиной
                    std::cout << std::setw(depth + 1) << ' ' << "Так как функция вернула true,
значит искомая вершина найдена. Записываем ребро  " << path->getNameFrom() << ' ' << path-
>getNameOut() << "  в вектор ответа." << std::endl;
                    depth--;
                    local.emplace_back(path);
                    return true;
                }
                else {
                    //делаем откат назад
                    std::cout << std::setw(depth + 1) << ' ' << "Делаем откат назад." <<
std::endl;
                    local2.pop_back();
                }
            }
        }
        std::cout << std::setw(depth + 1) << ' ' << "Ребро переполнено или возможных путей
больше нет." << std::endl;
    }
    std::cout << std::setw(depth + 1) << ' ' << "Функция поиска путей завершает работу." <<
std::endl;

    return false;

}

int main() {
    setlocale(LC_ALL, "Russian");
    char startPoint, endPoint;//исток и сток
    char start, end;
    int weight;
    int count;
    int flag = 0;
    int flag2 = 0;
    long int depth = 0;
    std::vector<Path*> local;
    //local.reserve(0);

    std::vector<Path*> local2;
    //local.reserve(0);

    std::vector<Path> paths;
    //paths.reserve(0);

    int maxFlow = 0;
    std::cout << "Здравствуйте! Введите ,пожалуйста, количество ориентированных рёбер графа,
исток, сток и ребра графа." << std::endl;
    std::cin >> count;
    std::cin >> startPoint;
    std::cin >> endPoint;
    if (count != 0)
    {
        flag2 = 1;
        while (count != 0) {
            std::cin >> start >> end >> weight;
```

```
            paths.emplace_back(Path(start, end, weight));
            count--;
        }
    }
    else
        std::cout << "Введено нулевое количество ориентированных ребер графа!" << std::endl;

    if (flag2)
    {
        std::cout << "Вызывается функция поиска пути." << std::endl;
        while (findPath(paths, local, local2, startPoint, &endPoint, depth)) {//как только
нашли путь до стока, вызываем функцию поиска минимума и изминения потока
            findMin(local, &maxFlow, depth);


            std::cout << std::setw(depth + 1) << ' ' << "Очищаем вектор ответа и вектор
просмотренных ребер." << std::endl;
            local.clear();
            local2.clear();
        }


            std::cout << std::setw(depth + 1) << ' ' << "Сортируем выходные рёбра в
лексикографическом порядке. " << std::endl;

            std::sort(paths.begin(), paths.end(), comp);//сортировка вершин в
лексикографическом порядке
            std::cout << std::endl;
            std::cout << "Ответ:  " << std::endl;
            std::cout << "Величина максимального патока =  " << maxFlow << std::endl;

            for (Path path : paths) {
                std::cout << "Ребро графа с фактической величиной протекающего потока:   " <<
path.getNameFrom() << " " << path.getNameOut() << " " << path.getFlow() << std::endl;
            }

            std::cout << "До свидания!" << std::endl;

    }

    return 0;
}
```