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

Студент гр. 8304 _____ Масалыкин Д.Р.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

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

Вар. 2. Поиск в ширину. Обработка совокупности вершин текущего фронта как единого целого, дуги выбираются в порядке уменьшения остаточных пропускных способностей. При равенстве остаточных пропускных способностей выбирается та дуга, из начала которой выходит меньше дуг, при этом учитываются только дуги с положительными остаточными пропускными способностями, не ведущие в вершины текущего или прошлых фронтов.

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_i V_j W_{ij} – ребро графа

V_i V_j W_{ij} – ребро графа

…

Выходные данные:

P_{max} – величина максимального потока

V_i V_j W_{ij} – ребро графа с фактической величиной протекающего потока

V_i V_j W_{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
a b 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 алгоритма.

В начале работы алгоритма создается граф. С помощью поиска в ширину производится поиск пути от истока в сток. Если такой путь существует, то мы в этом пути ищем дугу с наименьшей пропускной способностью, затем обновляем граф: для всех дуг, которые попали в путь от истока в сток и для обратных им пересчитывается их пропускная способность. Затем производиться подсчет максимального потока, к этому мы шагу сразу переходим, если путь не был найден. Выводятся фактические потоки для дуг.

Сложность алгоритма по операциям: O (E * F), E – число ребер в графе, F – максимальный поток

Сложность алгоритма по памяти: O (N+E), N – количество вершин, E – количество ребер

Описание функций и структур данных.

```
class FordFalkGraph {
        std::map< char, std::map< char, int > > graph;
}
```

Структура данных, используемая для хранения направленного графа. Представляет собой ассоциативный контейнер хранения вершин и соответствующего ей контейнера вершина-расстояние. Для каждой вершины хранится ассоциативный массив вершин, до которых можно добраться из текущей и вес пути до них.

1. `void maxStream(char source, char estuary);`

   Метод вычисления максимального потока.

2. `int get_min_stream(std::map< char, std::pair< char, int > > way, char source, char stock);`

   Метод нахождения минимальной пропускной способности на пути.

3. `void update_network(std::map< char, std::map< char, int > > &network, std::map< char, std::pair< char, int > > way, char source, char stock);`

   Метод перестройки сети в соответствии с правилом: если дуга входит в путь, то к ней прибавляется обратная(относительно нуля)

величина минимальной пропускной способности в данном пути. Если дуга обратная той, которая находится в этом пути, то к ней прибавляется величина минимальной пропускной способности.

4. std::map< char, std::pair< char, int > > BFS(std::map< char, std::map< char, int > > net, char source, char stock);

Поиск в ширину.

5. UML-диаграмма класса

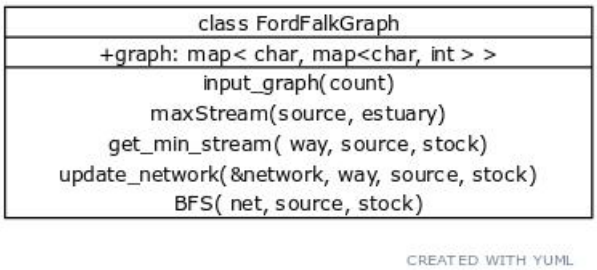

рис. 1. UML-диаграмма класса.

Тестирование и исследование.

Таблица 1 – Тестирование программы.

Input	Output	Время
7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2	3 сек
10 a h a b 5	6 a b 1 a c 4 a d 1	2 сек

a c 4 a d 1 b g 1	b g 1 c e 2	

c e 2 c f 3 d e 6 e h 4 f h 4 g h 8	c f 2 d e 1 e h 3 f h 2 g h 1	
d a b 1000 a c 1000 b c 1 b d 1000 c d 1000	2000 a b 1000 a c 1000 b c 0 b d 1000 c d 1000	1 сек

Пример изображения потока

d

a b 1000

a c 1000

b c 1

b d 1000

c d 1000

2000

a b 1000

a c 1000

b c 0

b d 1000

c d 1000

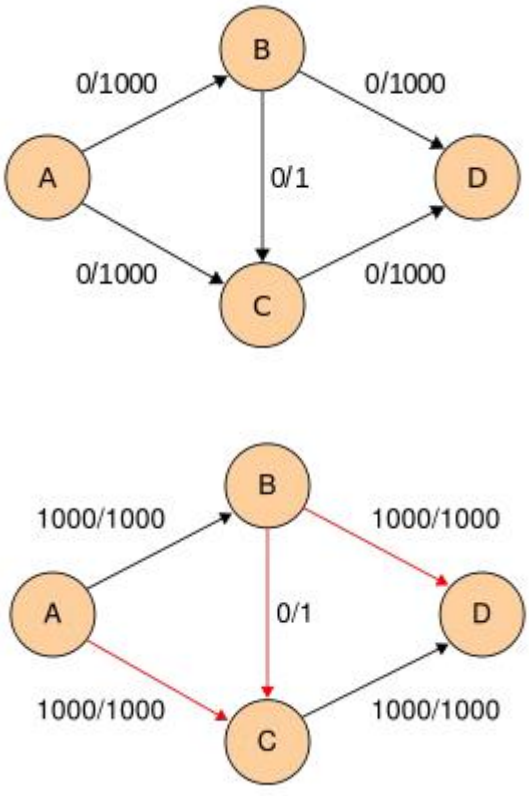

рис. 2.
Иллюстрация работы алгоритма

Выводы.

В ходе выполнения лабораторной работы был реализован алгоритм Форда-Фалкерсона(точнее алгоритм Эдмондса — Карпа, так как используется поиск в ширину), который находит максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро.

ПРИЛОЖЕНИЕ А.

Исходный код программы main.cpp

```
#include <iostream>
#include <map>
#include <cmath>
#include <set>
#include <queue>
#include <fstream>


class FordFalkGraph {
public:
    std::map< char, std::map< char, int > > graph;

    void input_graph(int count) {
        char src;
        char dst;
        int stream;
        std::cin >> src >> dst >> stream;
        for(int i = 0; i < count; i++) {
            graph[src][dst] = stream;
            std::cin >> src >> dst >> stream;
        }
    }

    void input_from_file(int count, char* file){
        std::ifstream input(file);
        char src;
```

```
    char dst;
    int stream;
    input >> src >> dst >> stream;
    for(int i = 0; i < count; i++) {
        graph[src][dst] = stream;
        input >> src >> dst >> stream;
    }
    input.close();
}

void maxStream(char source, char estuary){
    std::map< char, std::map< char, int > > network = graph;
    std::map< char, std::pair< char, int > > way = BFS(network, source, estuary);

    while( !way.empty() ){
        update_network(network, way, source, estuary);
        way = BFS(network, source, estuary);
    }

    int max_stream = 0;

    for(auto i : graph[source]){
        max_stream += i.second - network[source][i.first];
    }

    std::cout << max_stream << std::endl;

    for(const auto& neighbour : graph)
        for(auto node : neighbour.second){
            std::cout << neighbour.first << " " << node.first << " ";
            if(node.second - network[neighbour.first][node.first] >= 0)
```

```
                std::cout << node.second - network[neighbour.first][node.first] <<
std::endl;
            else
                std::cout << 0 << std::endl;
        }
    }
    int get_min_stream(std::map< char, std::pair< char, int > > way, char source, char
stock){
        int min_flow = way[stock].second;
        for(char node = stock; node != source; node = way[node].first) {
            if (min_flow > way[node].second)
                min_flow = way[node].second;
        }
        return min_flow;
    }

    //перестройка графа после очередного прохода
    void update_network(std::map< char, std::map< char, int > > &network, std::map<
char, std::pair< char, int > > way, char source, char stock){
        int min_flow = get_min_stream(way, source, stock);
        for(char node = stock; node != source; node = way[node].first){
            if(way[node].second - min_flow == 0)
                network[way[node].first].erase(node);
            else
                network[way[node].first][node] -= min_flow;
            network[node][way[node].first] += min_flow;
        }
    }
    //Поиск в ширину
```

```
    std::map< char, std::pair< char, int > > BFS(std::map< char, std::map< char, int >
> net, char source, char stock){
        std::set<char> visited_nodes;//множество посещенных вершин
        std::queue<char> queue_nodes;//Очередь вершин
        std::map< char, std::pair< char, int > > way;//путь
        std::map< char, std::map< char, int > >::iterator current;//Текущий
обрабатывваемый элемент
        visited_nodes.insert(source);
        queue_nodes.push(source);
        while(queue_nodes.front() != stock && !queue_nodes.empty()){
            current = net.find(queue_nodes.front());
            queue_nodes.pop();
            for(auto elem : current->second)
                if(visited_nodes.find(elem.first) == visited_nodes.end()){
                    queue_nodes.push(elem.first);
                    visited_nodes.insert(elem.first);
                    way[elem.first] = std::pair< char, double >(current->first, elem.second);
                }
        }
        if(visited_nodes.find(stock) == visited_nodes.end())
            return std::map<char,std::pair< char, int > >();
        return way;
    }
};

int main(int argc, char* argv[]) {
    int count;
    char source;
    char stock;
    FordFalkGraph graph;
```

```
    if(argc == 2){
        std::ifstream input(argv[1]);
        input>>count;
        input>>source;
        input>>stock;
        input.close();
    }
    else {
        std::cin >> count;
        std::cin >> source;
        std::cin >> stock;
        graph.input_graph(count);
    }
    graph.maxStream(source, stock);
    return 0;
}
```