

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

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

Студент гр. 8304

Кириянов Д.И.

Преподаватель

Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

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

Вариант 5.

Поиск не в глубину и не в ширину, а по правилу: каждый раз выполняется переход по дуге, имеющей максимальную остаточную пропускную способность. Если таких дуг несколько, то выбрать ту, которая была обнаружена раньше в текущем поиске пути.

Описание алгоритма.

1. Обнуляем все потоки. Остаточная сеть изначально совпадает с исходной сетью.

2. В остаточной сети находим любой путь из источника в сток. Если такого пути нет, останавливаемся.

3. Пускаем через найденный путь (он называется увеличивающим путём или увеличивающей цепью) максимально возможный поток:

- 1) На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью C_{\min} .
- 2) Для каждого ребра на найденном пути увеличиваем поток на C_{\min} , а в противоположном ему — уменьшаем на C_{\min} .
- 3) Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.

4. Возвращаемся на шаг 2.

Сложность алгоритма $O(|E|f)$, где

E — количество ребер в графе,

f — максимальный поток в графе.

Описание основных структур данных и функций.

```
struct elem {  
    int capacity;  
    int flow;  
};
```

- структура для хранения пропускной способности и потока элемента.

```
std::map<char, std::vector<std::pair<char, elem>>> card;
```

- словарь для хранения графа. Для каждой вершины хранится вектор с парами, в которых хранится вершина, в которую можно перейти, ее пропускная способность и поток.

```
std::map<char, char> prev;
```

- словарь для хранения вершины, из которой мы пришли в текущую.

```
std::map<char, bool> visited;
```

- словарь для проверки на посещение каждой из вершин.

```
bool comp(std::pair<char, elem> i, std::pair<char, elem> j);
```

- компаратор для сортировки по минимальной пропускной способности

```
bool cmp(std::pair<char, elem> i, std::pair<char, elem> j);
```

- компаратор для сортировки в лексикографическом порядке

```
int find(std::map<char, std::vector<std::pair<char, elem>>>& card, char  
current, char finish, std::map<char, char>& prev, std::map<char, bool> visited,  
int result);
```

- функция для поиска пути и ребра с минимальной пропускной способностью.

Возвращает минимальную пропускную способность на пути, если он есть.

Если пути нет, возвращает 0.

```
void net(char start, char finish, std::map<char, char> prev, std::map<char,  
std::vector<std::pair<char, elem>>>& card, int min);
```

- функция для изменения значений пропускной способности и потока для каждой из вершин в соответствии с передаваемой минимальной пропускной

способностью на пути. На найденном пути обновляется каждое ребро, а также обратные ребра, если они имеются.

Тестирование.

Таблица 1 – Результат работы.

Ввод	Вывод
7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	Result: 12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2
9 a d a b 3 b c 7 c d 5 h c 12 e f 9 g h 20 b e 10 a g 17 f d 8	Result: 5 a b 3 a g 2 b c 3 b e 0 c d 5 e f 0 f d 0 g h 2 h c 2
11 a d a b 3 b c 7 c d 5 b a 2 h c 12 e f 9	Result: 5 a b 3 a g 2 b a 0 b c 3 b e 0 c d 5

g h 20	e f 0
b e 10	f d 0
h g 10	g h 2
a g 17	h c 2
f d 8	h g 0

Вывод.

В ходе выполнения данной работы была написана программа, которая находит максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро, используя алгоритм Форда - Фалкерсона.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <queue>
#include <fstream>

std::string input_way = "C:/Users/Danielka/source/repos/piaa_lr3/input"; //пути до файлов
std::string output_way = "C:/Users/Danielka/source/repos/piaa_lr3/output";

struct elem { //структура для хранения
    int capacity; //пропускной способности
    int flow; //и потока элемента
};

bool comp(std::pair<char, elem> i, std::pair<char, elem> j) { //компаратор
    if (i.second.capacity == j.second.capacity) //для сортировки
        return i.first < j.first; //по минимальной пропускной способности
    return i.second.capacity < j.second.capacity;
}

bool cmp(std::pair<char, elem> i, std::pair<char, elem> j) {
    return i.first < j.first; //компаратор для сортировки
} //в лексикографическом порядке

void net(char start, char finish, std::map<char, char> prev, std::map<char, //обновление
значений
    std::vector<std::pair<char, elem>>>& card, int min) { //элементов
    std::vector<char> result;
    char current = finish;
    result.push_back(current);
    while (current != start) { //записываем путь в вектор
        current = prev[current];
        result.push_back(current);
    }
    std::cout << "Found way: "; //отладочный вывод
    for (unsigned long int i = 0; i < result.size(); ++i) {
        std::cout << result[result.size() - i - 1]; //выводим найденный путь
    }
    std::cout << std::endl << std::endl;
    std::cout << "Changes of edges:" << std::endl; //отладочный вывод
    for (unsigned long int i = 0; i < result.size()-1; ++i) { //изменяем значения
элементов
        for (auto& next : card[result[result.size()-i-1]]) { //для обычных путей
            if (next.first == result[result.size() - i - 2]) {
                std::cout << "Capacity " << result[result.size() - i - 1] <<
next.first << ": " << next.second.capacity;
                next.second.capacity -= min; //выводим изменения
                std::cout << " -> " << next.second.capacity << std::endl;
                std::cout << "Flow " << result[result.size() - i - 1] <<
next.first << ": " << next.second.flow;
                next.second.flow += min; //выводим изменения
                std::cout << " -> " << next.second.flow << std::endl <<
std::endl;
                for (auto& j : card[result[result.size() - i - 2]]) { //для
обратных путей
                    if (j.first == result[result.size() - i - 1]) {
                        std::cout << "Capacity " << result[result.size() -
i - 2] << j.first << ": " << j.second.capacity;
                        j.second.capacity += min; //выводим изменения

```

```

std::cout << " -> " << j.second.capacity <<
std::endl;
2] << j.first << ": " << j.second.flow;
j.second.flow -= min; //выводим изменения
std::cout << " -> " << j.second.flow << std::endl
<< std::endl;
    }
    }
}
std::cout << std::endl;
}

int find(std::map<char, std::vector<std::pair<char, elem>>>& card, char current, char
finish, //поиск пути
std::map<char, char>& prev, std::map<char, bool> visited, int result) { //и
минимального результата
std::cout << "Visiting: " << current << std::endl; //отладочный вывод
if (current == finish) //если дошли до конца
return result; //возвращаем результат
std::sort(card[current].begin(), card[current].end(), comp); //сортируем по
возрастанию пропускной способности
visited[current] = true; //считаем, что посетили текущую вершину
for (auto& next : card[current]) {
    if ((!visited[next.first]) && (next.second.capacity > 0)) { //если не посещена
и пропускная способность>0
        result = next.second.capacity; //первое ребро - начальный результат
        prev[next.first] = current; //обновляем путь
        int minim = find(card, next.first, finish, prev, visited,
result); //рекурсия
        if (minim > 0) { //если нашли путь
            if (minim < result) //если меньше
                result = minim; //то обновляем результат
            return result;
        }
    }
}
return 0;
}

int main() {
std::cout << "How do you want to input?" << std::endl << std::endl
<< "Press 1 to input from console." << std::endl //выбор как считать
<< "Press 2 to input from file." << std::endl;
int count = 0;
char start = '0';
char finish = '0';
std::map<char, std::vector<std::pair<char, elem>>> card; //хранение графа
char first, second;
int len=0;
int menu = 0;
while (menu != 1 && menu != 2) { //пока не введется нужная цифра
    std::cin >> menu;
    if (menu == 1) { //считывание с консоли
        std::cin >> count;
        std::cin >> start >> finish;
        for (int i = 0; i < count; ++i) {
            std::cin >> first >> second >> len;
            card[first].push_back({ second, {len, 0} });
        }
    }
}
}

```

```

    }
}
else if (menu == 2) { //считывание из файла
    std::ifstream file;
    file.open(input_way);
    if (!file.is_open()) {
        std::cout << "Can't open file!" << std::endl;
        return 0;
    }
    file >> count;
    file >> start >> finish;
    for (int i = 0; i < count; ++i) {
        file >> first >> second >> len;
        card[first].push_back({ second, {len, 0} });
    }
}
else { //если неверно введена цифра, выводится сообщение
    std::cout << std::endl << "Wrong choice! Try again!" << std::endl;
} //а цикл продолжается
}
std::map<char, char> prev; //для сохранения пути
prev[start] = start;
std::map<char, bool> visited; //проверка на посещение
int test = 0;
int flow = 0;
std::cout << std::endl;
while (test = find(card, start, finish, prev, visited, 0)) { //пока есть путь
    std::cout << std::endl << "Minimal capacity: " << test <<
std::endl; //найденное минимальное ребро
    flow += test; //максимальное значение
    net(start, finish, prev, card, test); //обновляем показатели
}
std::cout << "No more ways." << std::endl;
menu = 0;
std::cout << std::endl << "How do you want to output?" << std::endl << std::endl
<< "Press 1 to output by console." << std::endl //выбор как вывести
<< "Press 2 to output into file." << std::endl;
while (menu != 1 && menu != 2) { //пока не введется нужная цифра
    std::cin >> menu;
    if (menu == 1) { //вывод на консоль
        std::cout << std::endl << "Result: " << std::endl;
        std::cout << flow << std::endl;
        for (auto k : card) {
            std::sort(k.second.begin(), k.second.end(), cmp);
            for (auto i : k.second)
                std::cout << k.first << " " << i.first << " " <<
std::max(0, i.second.flow) << std::endl;
        }
    }
    else if (menu == 2) { //вывод в файл
        std::ofstream file;
        file.open(output_way);
        if (!file.is_open()) {
            std::cout << "Can't open file!\n";
        }
        file << "Result: " << std::endl;
        file << flow << std::endl;
        for (auto k : card) {
            std::sort(k.second.begin(), k.second.end(), cmp);
            for (auto i : k.second)
                file << k.first << " " << i.first << " " << std::max(0,
i.second.flow) << std::endl;
        }
    }
}
}

```



```
        else { //если неверно введена цифра, выводится сообщение
                std::cout << std::endl << "Wrong choice! Try again!" << std::endl;
        } //а цикл продолжается
    }
    return 0;
}
```