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 _____ Бочаров Ф.Д.

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 максимально возможный поток:
   1) На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью Cmin.
   2) Для каждого ребра на найденном пути увеличиваем поток на Cmin, а в противоположном ему — уменьшаем на Cmin.
   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 b e 10 h g 10 a g 17 f d 8	e f 0 f d 0 g h 2 h c 2 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>


struct elem {
	int capacity;
	int flow;
};

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

bool comp_lex(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, std::ostream& output)
{//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);
	}
	output << "Found way: ";
	for (unsigned long int i = 0; i < result.size(); ++i) {
		output << result[result.size() - i - 1];//выводим найденный путь
	}
	output << std::endl << std::endl;
	output << "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]) {
				output << "Capacity " << result[result.size() - i - 1] <<
next.first << ": " << next.second.capacity;
				next.second.capacity -= min;//выводим изменения
				output << " -> " << next.second.capacity << std::endl;
				output << "Flow " << result[result.size() - i - 1] <<
next.first << ": " << next.second.flow;
				next.second.flow += min;//выводим изменения
				output << " -> " << 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]) {
						output << "Capacity " << result[result.size() - i
- 2] << j.first << ": " << j.second.capacity;
						j.second.capacity += min;//выводим изменения
						output << " -> " << j.second.capacity <<
std::endl;
```

```
                                        output << "Flow " << result[result.size() - i - 2]
<< j.first << ": " << j.second.flow;
                                        j.second.flow -= min;//выводим изменения
                                        output << " -> " << j.second.flow << std::endl <<
std::endl;
                                }
                        }
                }
        }

        }
        output << 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, std::ostream&
output) {//и минимального результата
        output << "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_min);//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)) {// 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,
output);//рекурсия
                        if (minim > 0) {//если нашли путь
                                if (minim < result)//если меньше
                                        result = minim;//то обновляем результат
                                return result;
                        }
                }
        }
        return 0;

}


int main() {
        int count;
        char start;
        char finish;
        std::map<char, std::vector<std::pair<char, elem>>> card;//хранение графа
        char first, second;
        int len;
                        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} });
                        }



        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, std::cout))
{//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, std::cout);//обновляем
показатели
                    }
                    std::cout << "No more ways." << std::endl;
                    std::cout << std::endl << "Result: " << std::endl;
                    std::cout << flow << std::endl;
                    for (auto k : card) {
                           std::sort(k.second.begin(), k.second.end(), comp_lex);
                           for (auto i : k.second)
                                  std::cout << k.first << " " << i.first << " " <<
std::max(0, i.second.flow) << std::endl;
                    }



      return 0;
}
```