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 Самакаев Д.И.

Преподаватель Размочаева Н.В.

Санкт-Петербург

2019

Вариант 5.

Цель работы.

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

Основные теоретические положения.

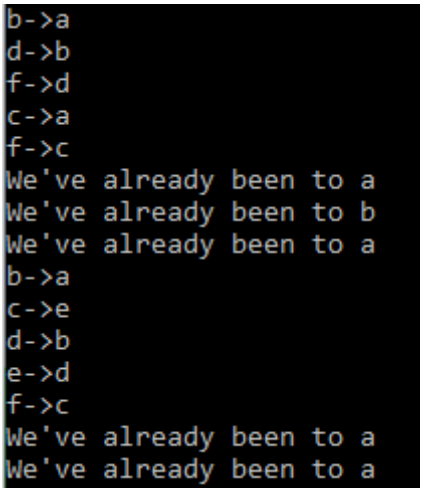
Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа - пропускной способности (веса).В ответе выходные рёбра отсортируйте в лексикографическом порядке по первой вершине, потом по второй (в ответе должны присутствовать все указанные входные рёбра, даже если поток в них равен 0).

Описание алгоритма.

Для решения поставленной задачи был реализован алгоритм Форда-Фалкерсона. Были реализованы функции find, реализующая поиск пути от истока к стоку, update_map, обновляющая вместимости ребёр после нахождения очередного пути и функция choose_direction, позволяющая строить путь в соответствии с заданием 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. Функция find рекурсивная и прекращает свою работу когда не может найти путь от истока к стоку.

Вывод промежуточной информации.

После нахождения пути он выводится, при попытке повторно войти в точку, которую уже посещали, выводится соответствующее сообщение.



Тестирование.

Таблица 1 – Результаты тестирование

Ввод	**Вывод**
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
a g a b 3 b c 2 c d 2 d b 2 d g 4	2 a b 2 b c 2 c d 2 d b 0 d g 2
a e a b 0 b c 1 c d 0 a d 5 d e 1	1 a b 0 a d 1 b c 0 c d 0 d e 1

Вывод.

В ходе работы был построен и анализирован алгоритм Форда-

Фалкерсона на основе решения задачи о нахождении максимального потока в сети. Исходный код программы представлен в приложении 1.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <algorithm>
#include <map>
#include <stack>
#include <set>


struct elem {
	char next;
	size_t capacity;
	size_t flow;
};

char choose_direction(char cur, std::set<char>& open, std::map<char, std::vector<elem>>
my_map) {
	size_t max = 0;
	char dir = 0;
	for (auto i : open) {
		for (auto j : my_map[cur]) {
			if (j.capacity > max && j.next == i) {
				max = j.capacity;
				dir = i;
			}
		}
	}
	return dir;
}

bool cmp(elem a, elem b){
	return a.capacity > b.capacity;
}

void update_map(char end, char start, std::map<char, std::vector<elem>> &my_map, size_t
min_flow, std::map<char, char> path_syms, std::map<char, std::vector<std::pair<char,
size_t>>> &flows) {

	char cur = end;

	while (cur != start) {
		for (auto it = my_map.begin(); it != my_map.end(); ++it) {
			if(it->first == path_syms[cur])
				for (size_t i = 0; i < it->second.size(); ++i) {
					if (it->second[i].next == cur) {
						it->second[i].capacity -= min_flow;
						it->second[i].flow = 0;
					}
				}
		}

		for (auto it = flows.begin(); it != flows.end(); ++it) {
			if (it->first == path_syms[cur])
				for (size_t i = 0; i < it->second.size(); ++i) {
					if (it->second[i].first == cur) {
						it->second[i].second += min_flow;
					}
				}
		}
		my_map[cur].push_back({ path_syms[cur], min_flow, 0 });
```

```
			cur = path_syms[cur];
		}
}

size_t find(char start, char finish, std::map<char, std::vector<elem>> &my_map, size_t
&result_flow, std::map<char, std::vector<std::pair<char, size_t>>>& flows) {

		std::map<char, char> path_syms;

		size_t min_flow = std::numeric_limits<size_t>::max();

		std::set<char> closed_set;
		std::set<char> open_set = { start };

		char cur = start;
		char buff;

		while (!open_set.empty()) {

			closed_set.insert(cur);
			open_set.erase(cur);

			if (cur == finish) {
				update_map(finish, start, my_map, min_flow, path_syms, flows);
				result_flow += min_flow;
				find(start, finish, my_map, result_flow, flows);
				return result_flow;
			}


			for (auto direction : my_map[cur]) {

				if (direction.capacity == 0)
					continue;

				if (closed_set.find(direction.next) != closed_set.end()) {
					std::cout << "We've already been to " << direction.next <<
std::endl;
				}
				else open_set.insert(direction.next);
			}

			if (cur == start && open_set.empty())
				return result_flow;

			buff = choose_direction(cur, open_set, my_map);
			for (auto i : my_map[cur]) {
				if (i.next == buff) {
					min_flow = std::min(min_flow, i.capacity);
					path_syms[buff] = cur;
				}
			}
			cur = buff;
		}

}

void console_input() {

		std::string out_file_name = "out.txt";

		char start, end;
		std::cin >> start >> end;
		char a, b;
		size_t c = 0;
```

```
	std::map<char, std::vector<elem>> my_map;
	std::map<char, std::vector<std::pair<char, size_t>>> flows;

	while (std::cin >> a >> b >> c) {
		my_map[a].push_back({ b,c,0 });
		flows[a].push_back({ b,0 });
		std::sort(my_map[a].begin(), my_map[a].end(), cmp);
	}

	size_t result = 0;

	std::cout << find(start, end, my_map, result, flows) << std::endl;

	for (auto it = flows.begin(); it != flows.end(); ++it) {
		for (size_t i = 0; i < it->second.size(); ++i) {
			std::cout << it->first << " " << it->second[i].first << " " << it-
>second[i].second << " " << std::endl;
		}
	}

	std::ofstream out_file;
	out_file.open(out_file_name);

	if (!out_file.is_open()) {
		std::cout << "Error! Output file isn't open" << std::endl;
	}

	out_file << find(start, end, my_map, result, flows) << std::endl;

	for (auto it = flows.begin(); it != flows.end(); ++it) {
		for (size_t i = 0; i < it->second.size(); ++i) {
			out_file << it->first << " " << it->second[i].first << " " << it-
>second[i].second << "\n";
		}
	}
}

void file_input(char*& argv) {
	std::ifstream file;
	std::string testfile = argv;

	std::string out_file_name = "out.txt";

	char start, end;
	char a, b;
	size_t c = 0;

	std::map<char, std::vector<elem>> my_map;
	std::map<char, std::vector<std::pair<char, size_t>>> flows;

	file.open(testfile);

	if (!file.is_open()) {
		std::cout << "Error! File isn't open" << std::endl;
		return;
	}

	file >> start >> end;

	while (!file.eof()) {
		file >> a >> b >> c;
		my_map[a].push_back({ b,c,0 });
		flows[a].push_back({b,0});
		std::sort(my_map[a].begin(), my_map[a].end(), cmp);
```

```
	}

	size_t result = 0;

	std::cout << find(start, end, my_map, result, flows) << std::endl;

	for (auto it = flows.begin(); it != flows.end(); ++it){
		for (size_t i = 0; i < it->second.size(); ++i) {
			std::cout << it->first << " " << it->second[i].first << " " << it-
>second[i].second << " " << std::endl;
		}
	}

	std::ofstream out_file;
	out_file.open(out_file_name);

	if (!out_file.is_open()) {
		std::cout << "Error! Output file isn't open" << std::endl;
	}

	out_file << result << std::endl;

	for (auto it = flows.begin(); it != flows.end(); ++it) {
		for (size_t i = 0; i < it->second.size(); ++i) {
			out_file << it->first << " " << it->second[i].first << " " << it-
>second[i].second << "\n";
		}
	}

}

int main(size_t argc, char** argv) {
	if (argc == 1)
		console_input();
	else if (argc == 2)
		file_input(argv[1]);

	return 0;
}
```