**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №**2

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Алгоритмы на графах**

|  |  |  |
| --- | --- | --- |
| 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.

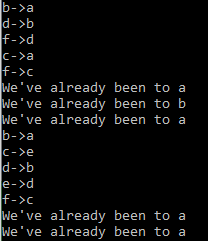
Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа - пропускной способности (веса).В ответе выходные рёбра отсортируйте в лексикографическом порядке по первой вершине, потом по второй (в ответе должны присутствовать все указанные входные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

}