**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 |  | Ястребов И.М. |
| 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:
4. На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью Cmin.
5. Для каждого ребра на найденном пути увеличиваем поток на {\displaystyle c\_{\min }}Cmin, а в противоположном ему — уменьшаем на {\displaystyle c\_{\min }}Cmin.
6. 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.

typedef struct elem { //Поток и пропускная способность

int capacity;

int flow;

}elem;

- структура для хранения потока и пропускной способности.

std::map<char, std::vector<std::pair<char, elem>>> desk; //Храним граф в виде словаря

- словарь, в котором хранится граф.

std::map<char, char> path;

- словарь для хранения пути.

std::map<char, bool> visited;

- словарь для учета посещенности вершины.

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

- компаратор для сортировки по минимальной пропускной способности

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

- компаратор для сортировки в лексикографическом порядке

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

- изменение потоков и пропускных способсностей на ребрах пути и противоположных им ребрах.

**Тестирование.**

Таблица 1 – Результат работы.

|  |  |
| --- | --- |
| **Ввод** | **Вывод** |
| 6  a  f  a b 7  b d 6  c f 9  d e 3  d f 5  e c 2 | Result:  6  a b 6  b d 6  c f 2  d e 2  d f 4  e c 2 |
| 9  a d  a b 3  b c 7  c d 5  b a 3  e f 9  b e 10  h g 10  a f 14  f d 8 | Result:  11  a b 3  a f 8  b a 0  b c 3  b e 0  c d 3  e f 0  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 <string>

#include <iostream>

#include <vector>

#include <map>

#include <algorithm>

#include <queue>

#include <fstream>

std::string inPath = "./input.txt";

std::string outPath = "./output.txt"; //Входной и выходной файлы

typedef struct elem { //Поток и пропускная способность

int capacity;

int flow;

}elem;

bool compCapacity(std::pair<char, elem>, std::pair<char, elem>); //Компаратор для сортировки по пропускной способности

bool compLexic(std::pair<char, elem>, std::pair<char, elem>); //Компаратор для сортировки в лексикографическом порядке

void modifyCapacities(char start, char finish, std::map<char, char> path, std::map<char,

std::vector<std::pair<char, elem>>>& desk, int min, std::ostream& output);

//Модифицируем пропускные способности

int findPath(std::map<char, std::vector<std::pair<char, elem>>>& desk, char current, char finish,

std::map<char, char>& path, std::map<char, bool> visited, int result, std::ostream& output);

//Поиск пути в транспортной сети

bool compCapacity(std::pair<char, elem> first, std::pair<char, elem> second) {

if (first.second.capacity == second.second.capacity)

return first.first < second.first;

return first.second.capacity < second.second.capacity;

}

bool compLexic(std::pair<char, elem> first, std::pair<char, elem> second) {

return first.first < second.first;

}

void modifyCapacities(char start, char finish, std::map<char, char> path, std::map<char,

std::vector<std::pair<char, elem>>>& desk, int min, std::ostream& output) {

std::vector<char> result;

char current = finish;

result.push\_back(current);

while (current != start) {//Храним путь в векторе

current = path[current];

result.push\_back(current);

}

output << "Found path: ";//Вывод промежуточных данных

for (size\_t i = 0; i < result.size(); ++i) {

output << result[result.size() - i - 1];//выводим найденный путь

}

output << std::endl << std::endl;

output << "Changes done:" << std::endl; // Вывод промежуточных данных

for (size\_t i = 0; i < result.size() - 1; ++i) {

for (auto& next : desk[result[result.size() - i - 1]]) { //Изменяем пропускные способности forward-пути

if (next.first == result[result.size() - i - 2]) {

output << "Capacity " << result[result.size() - i - 1] << next.first << ": " << next.second.capacity;

next.second.capacity -= min; //Вывод промежуточных данных

output << " changed to " << next.second.capacity << std::endl;

output << "Flow " << result[result.size() - i - 1] << next.first << ": " << next.second.flow;

next.second.flow += min; //Вывод промежуточных данных

output << " changed to " << next.second.flow << std::endl << std::endl;

for (auto& edge : desk[result[result.size() - i - 2]]) { //Изменяем пропускные способности reverse-пути

if (edge.first == result[result.size() - i - 1]) {

output << "Capacity " << result[result.size() - i - 2] << edge.first << ": " << edge.second.capacity;

edge.second.capacity += min;//Вывод промежуточных данных

output << " changed to " << edge.second.capacity << std::endl;

output << "Flow " << result[result.size() - i - 2] << edge.first << ": " << edge.second.flow;

edge.second.flow -= min;//Вывод промежуточных данных

output << " changed to " << edge.second.flow << std::endl << std::endl;

}

}

}

}

}

output << std::endl;

}

int findPath(std::map<char, std::vector<std::pair<char, elem>>>& desk, char current, char finish,

std::map<char, char>& path, std::map<char, bool> visited, int result, std::ostream& output) {

output << "Visiting: " << current << std::endl; //Вывод промежуточных данных

if (current == finish)//Если путь не найден, результат не меняется

return result;

std::sort(desk[current].begin(), desk[current].end(), compCapacity);//Сортируем возможные перемещения по

//возрастанию пропускной способности

visited[current] = true;//Помечаем текущую вершину как посещённую

for (auto& next : desk[current]) {

if ((!visited[next.first]) && (next.second.capacity > 0)) {// Если пропускная способность больше нуля и вершина не посещена

result = next.second.capacity;

path[next.first] = current; //Обновляем путь

int MinPath = findPath(desk, next.first, finish, path, visited, result, output); //Рекурсивная часть алгоритма

if (MinPath > 0) { //Если путь найден

if (MinPath < result) //И он меньше

result = MinPath;//то обновляем результат

return result;

}

}

}

return 0;

}

int main() {

std::cout << "Choose input format" << std::endl << std::endl

<< "1 - console" << std::endl

<< "2 - file" << std::endl;

int cnt(0);

char start = '0';

char finish = '0';

std::map<char, std::vector<std::pair<char, elem>>> desk; //Храним граф в виде словаря

char first, second;

int len(0);

int mode(0);

std::cin >> mode;

if (mode == 1) {//Считывание с консоли

std::cin >> cnt;

std::cin >> start >> finish;

for (int i = 0; i < cnt; ++i) {

std::cin >> first >> second >> len;

desk[first].push\_back({ second, {len, 0} });

}

}

else if (mode == 2) { //Считывание из файла

std::ifstream file;

file.open(inPath);

if (!file.is\_open()) {

std::cout << "Can't open file!" << std::endl;

return 0;

}

file >> cnt;

file >> start >> finish;

for (int i = 0; i < cnt; ++i) {

file >> first >> second >> len;

desk[first].push\_back({ second, {len, 0} });

}

}

std::map<char, char> path; //Храним путь в графе

path[start] = start;

std::map<char, bool> visited;//Посещена ли вершина - чтобы не ходить кругами

int test = 0;

int flow = 0;

mode = 0;

std::cout << std::endl << "Choose output format" << std::endl << std::endl

<< "1 - console" << std::endl//Выбор варианта вывода

<< "2 - file" << std::endl;

std::cin >> mode;

if (mode == 1) {//Консоль

std::cout << std::endl;

while (test = findPath(desk, start, finish, path, visited, 0, std::cout)) { //Пока путь есть

std::cout << std::endl << "Minimal capacity: " << test << std::endl;//Вывод промежуточных данных

flow += test;//Увеличиваем значение потока

modifyCapacities(start, finish, path, desk, test, std::cout); //Обновляем пропускные способности

}

std::cout << "Done" << std::endl;

std::cout << std::endl << "Result: " << std::endl;

std::cout << flow << std::endl;

for (auto k : desk) {

std::sort(k.second.begin(), k.second.end(), compLexic);

for (auto i : k.second)

std::cout << k.first << " " << i.first << " " << std::max(0, i.second.flow) << std::endl;

}

}

else if (mode == 2) { //Вывод в файл

std::ofstream file;

file.open(outPath);

if (!file.is\_open()) {

std::cout << "Can't open file!\n";

return 0;

}

while (test = findPath(desk, start, finish, path, visited, 0, file)) {//Пока путь есть

file << std::endl << "Minimal capacity: " << test << std::endl;//Вывод промежуточных данных

flow += test;//Увеличиваем значение потока

modifyCapacities(start, finish, path, desk, test, file);//Обновляем пропускные способности

}

file << "Done" << std::endl;

file << std::endl << "Result: " << std::endl;

file << flow << std::endl;

for (auto k : desk) {

std::sort(k.second.begin(), k.second.end(), compLexic);

for (auto i : k.second)

file << k.first << " " << i.first << " " << std::max(0, i.second.flow) << std::endl;

}

}

return 0;

}