**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

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

**Тема: Алгоритмы поиска максимального потока в графе**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9303 |  | Куршев Е.О. |
| Преподаватель |  | Шевская Н.В. |

Санкт-Петербург

2021

**Цель работы**

Ознакомиться с алгоритмами поиска максимального в ориентированном графе. Реализовать алгоритм Форда-Фалкерсона поиска максимального потока с выбранной индивидуализацией.

**Постановка задачи**

Вар. 6. Поиск не в глубину и не в ширину, а по правилу: каждый раз выполняется переход по дуге, соединяющей вершины, имена которых в алфавите ближе всего друг к другу. Если таких дуг несколько, то выбрать ту, имя конца которой в алфавите ближайшее к началу алфавита.

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

Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа - пропускной способности (веса).

**Основные теоретические положения**

Какая-то теория



**Выполнение работы**

Сначала была написана функция, которая выполняет предобработку графа: сортирует пути в алфавитном порядке.

Далее была написана функция, которая выполняет алгоритм Форда-Фалкерсона по определённому правилу: каждый раз выполняется переход по дуге, соединяющей вершины, имена которых в алфавите ближе всего друг к другу. Если таких дуг несколько, то выбрать ту, имя конца которой в алфавите ближайшее к началу алфавита.

**Оценка сложности алгоритма:**

Пусть E — количество ребер, V — количество вершин. В общем алгоритм оценивается сверху как O(VE2), потому что инициализация графа происходит за O(E), и каждый поиск отрабатывает за O(E), Цикл while в худшем случае будет запущен O(V) раз, откуда получим такую сложность.



**Тестирование**

Было создано 2 кейса тестов:

1. Проверка начальной сортировки графа.
2. Проверка работоспособности алгоритма

**Выводы**

Был успешно реализован алгоритм Форда-Фалкерсона для нахождения максимального потока в графе.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Имя файла: main.cpp

#include "functions.h"

int main() {

char start, end;

char a, b;

int dist;

int n;

way tmp\_way;

std::string filename;

std::cout << "Write filename: ";

std::cin >> filename;

std::ifstream file;

file.open(filename);

std::vector<way> ways;

file >> n;

file >> start;

file >> end;

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

file >> a >> b >> dist;

tmp\_way.a = a;

tmp\_way.b = b;

tmp\_way.ab = dist;

tmp\_way.ba = 0;

tmp\_way.IsVisit = false;

ways.push\_back(tmp\_way);

}

start\_sort(ways);

int l = alg(ways, start, end);

std::cout << l << '\n';

for(auto & way : ways)

std::cout << way.a << ' ' << way.b << ' ' << way.ba << '\n';

return 0;

}

Имя файла: functions.h

#ifndef LAB3\_FUNCTIONS\_H

#define LAB3\_FUNCTIONS\_H

#include <iostream>

#include <vector>

#include <algorithm>

#include <fstream>

struct way{

char a;

char b;

int ab;

int ba;

bool IsVisit;

};

void start\_sort(std::vector<way>& ways){

for(int i = 0; i < ways.size() - 1; i++)

for(int j = 0; j < ways.size() - i - 1; j++)

if(ways[j].a > ways[j + 1].a || (ways[j].a == ways[j + 1].a && ways[j].b > ways[j + 1].b))

std::swap(ways[j], ways[j + 1]);

}

int alg(std::vector<way>& ways, char start, char end){

char cur = '~';

char tmp\_cur = '~';

int dist = 0;

int max = 0;

std::vector<int> l;

std::vector<char> line;

while (cur != start){

for (auto & way : ways)

if(way.a == start && abs(way.a - way.b) < abs(way.a - tmp\_cur) && way.ab > 0){

tmp\_cur = way.b;

dist = way.ab;

way.IsVisit = true;

}

cur = tmp\_cur;

tmp\_cur = '~';

std::cout << "Current peak: " << start << '\n';

std::cout << "Chain: " << start << '\n';

std::cout << "Current peak: " << cur << '\n';

std::cout << "Chain: " << start << ' ' << cur << '\n';

line.push\_back(start);

line.push\_back(cur);

l.push\_back(dist);

while (cur != end && cur != start){

for (auto & way : ways)

if(way.a == cur && abs(way.a - way.b) < abs(way.a - tmp\_cur) && !way.IsVisit && way.ab > 0 && way.b != line[line.size() - 2]){

tmp\_cur = way.b;

dist = way.ab;

}

for (auto & way : ways)

if(way.b == cur && abs(way.b - way.a) < abs(way.b - tmp\_cur) && !way.IsVisit && way.ba > 0 && way.b != line[line.size() - 2]){

tmp\_cur = way.a;

dist = way.ba;

}

if(tmp\_cur == '~'){

cur = line[line.size() - 2];

line.pop\_back();

l.pop\_back();

std::cout << "Current peak: " << cur << '\n';

std::cout << "Chain: ";

for (int i = 0; i < line.size(); i++)

std::cout << line[i] << ' ';

std::cout << '\n';

}

else{

for(int i = 0; i < ways.size(); i++)

if(ways[i].a == cur && ways[i].b == tmp\_cur || ways[i].b == cur && ways[i].a == tmp\_cur)

ways[i].IsVisit = true;

cur = tmp\_cur;

tmp\_cur = '~';

line.push\_back(cur);

l.push\_back(dist);

std::cout << "Current peak: " << cur << '\n';

std::cout << "Chain: ";

for (int i = 0; i < line.size(); i++)

std::cout << line[i] << ' ';

std::cout << '\n';

}

if (cur == start){

for (auto & way : ways)

if(way.a == start && abs(way.a - way.b) < abs(way.a - tmp\_cur) && !way.IsVisit && way.ab > 0){

tmp\_cur = way.b;

dist = way.ab;

way.IsVisit = true;

}

if(tmp\_cur != '~'){

cur = tmp\_cur;

tmp\_cur = '~';

line.push\_back(cur);

l.push\_back(dist);

}

}

}

if (cur == end){

int tmp\_min = \*std::min\_element(l.begin(), l.end());

max += tmp\_min;

while (line.size() != 1){

char a = line[line.size() - 2];

char b = line[line.size() -1];

for (int i = 0; i < ways.size(); i++){

if (ways[i].a == a && ways[i].b == b){

ways[i].ab -= tmp\_min;

ways[i].ba += tmp\_min;

break;

}

else if (ways[i].a == b && ways[i].b == a){

ways[i].ab += tmp\_min;

ways[i].ba -= tmp\_min;

}

ways[i].IsVisit = false;

}

line.pop\_back();

}

l.clear();

line.clear();

std::cout << "Way found! Max flow: " << max << '\n';

}

std::cout << '\n';

}

std::cout << "Way not found!" << '\n';

return max;

}

#endif