**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

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

**Тема: Максимальный поток**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9382 |  | Рыжих Р.В. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2021

**Цель работы.**

Изучить алгоритм Форда-Фалкерсона, то есть алгоритм поиска максимального потока в сети. Написать программу, которая реализует данный алгоритм, используя полученные знания.

**Задание.**

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).

Входные данные:  
*N* - количество ориентированных рёбер графа  
*v0* - исток  
*vn* - сток  
*vi vj ωij* - ребро графа  
*vi vj ωij* - ребро графа  
...

Выходные данные:  
*Pmax* - величина максимального потока  
*vi vj ωij* - ребро графа с фактической величиной протекающего потока  
*vi vj ωij* - ребро графа с фактической величиной протекающего потока  
…

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).

**Sample Input:**

7

a

f

a b 7

a c 6

b d 6

c f 9

d e 3

d f 4

e c 2

**Sample Output:**

12

a b 6

a c 6

b d 6

c f 8

d e 2

d f 4

e c 2

**Вариант 2.**

Поиск в ширину. Обработка совокупности вершин текущего фронта как единого целого, дуги выбираются в порядке уменьшения остаточных пропускных способностей.

**Описание алгоритмов:**

**Алгоритм нахождения пути (Поиск в ширину).**

* В начале во фронте находится только первая вершина, далее фронтом становятся все соседи прошлого фронта. Прошлый фронт становится посещенным, поэтому к нему больше нельзя вернуться.
* При рассмотрении дуг между старым и новым фронтами сохраняется максимальная дуга для каждой вершины нового фронта.
* Поиск завершается, когда алгоритм дойдет до конечной вершины. Далее восстанавливается путь по сохраненным дугам. Восстановление пути однозначно, так как после отбора в каждую вершину входит только одна сохраненная дуга.

**Алгоритм Форда-Фалкерсона (поиск максимального потока).**

* В начале находится первый доступный путь с ненулевой остаточной пропускной способностью.
* Далее по данному пути пропускается поток, соответствующий минимальной из пропускных способностей ребер этого пути. Другими словами, от всех ребер отнимается значение данного потока, а к обратным рёбрам прибавляется это значение.
* Полученное значение прибавляется к переменной, которая считает максимальный поток.
* Алгоритм завершается, когда больше невозможно найти доступный путь.

**Сложность алгоритма Форда-Фалкерсона:**

В худщем случае алгоритм увеличивает поток на каждой итерации на единицу, тогда всего итераций будет F, где F – величина максимального потока. На каждой итерации совершается поиск в ширину, сложность которого равна O(V + E). Итоговая сложность по времени — O(F \* (V + E)).

Данный алгоритм использует исходных граф, поэтому дополнительная память не нужна. Поиск в ширину хранит путь, родителя каждой вершины в обходе, посещенные вершины, поэтому итоговая сложность по памяти равна O(3 \* V) = О(V).

**Функции и структуры данных:**

*Структуры данных:*

*class Finding* – класс для поиска максимального потока

*std::map<char, std::map<char, double>> edges* — ребра с пропускной способностью

*std::map<char, std::map<char, double>> capacity* — ребра с текущим потоком

*std::set<char> visited* — посещенные вершины

*std::vector<char> path* — путь

*Функции:*

*Finding::Read()* — функция для считывания данных.

*void Finding::PrintEdges(const std::map<char, double>& e, const char& ver)* — функция вывода ребер, исходящих из одной вершины.

*void Finding::PrintFront(const std::queue<char>& front)* — фукнция вывода фронта.

*void Finding::PrintCapacity()* - функция вывода графов и их пропускной способности.

*bool Finding::IsNewFront(const std::queue<char>& f, char v)* — функция проверки на новый фронт

*void Finding::FindPath()* - функция поиска пути в ширину. В начале работы функции инициализируются текущая вершина и фронт начальной вершины. Далее начинается цикл while (пока путь не найден):

* рассматриваются все ребра, которые исходят из текущей вершины
* в новый фронт добавляются не посещённые вершины с остаточной пропускной способностью, которая больше 0. Сохраняются только максимальные дуги, входящие в вершины нового фронта.
* Если фронт пуст, то он заменяется новым фронтом. Если и новый фронт пустой, значит пути нет.
* Достается следующая вершина из фронта.

*double Finding::AlgorithmFF()* - функция, реализующая алгоритм Форда-Фалкерсона. Начинается цикл while (пока есть доступный путь):

* Находится минимальная пропускная способность ребра на данном пути
* От остаточных пропускных способностей ребер отнимается минимальная пропускная способность, а также минимальная пропускная способность прибавляется для обратных ребер.
* Обновляются текущий поток ребер на пути и текущий максимальный поток.

**Тестирование.**

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 7  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 |
| 11  a  h  a b 4  b e 2  a c 2  c e 3  a d 3  d e 4  e g 3  e f 2  f h 3  g h 1  d f 1 | 4  a b 0  a c 1  a d 3  b e 0  c e 1  d e 2  d f 1  e f 2  e g 1  f h 3  g h 1 |
| 8  a  f  a b 6  a c 7  b d 4  c f 6  d e 3  d f 5  d m 2  e c 2 | 10  a b 4  a c 6  b d 4  c f 6  d e 0  d f 4  d m 0  e c 0 |
| 5  a  e  a c 1  a b 1  c b 1  b c 1  c d 1 | 0  a b 0  a c 0  b c 0  c b 0  c d 0 |

**Выводы.**

В результате выполнения работы был изучен алгоритм Форда-Фалкерсона (поиск максимального потока). Основываясь на полученных знаниях, была написана программа, которая реализует данный алгоритм.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Файл main.cpp.

#include <iostream>

#include <map>

#include <string>

#include <set>

#include <vector>

#include <queue>

#include <algorithm>

#define INFO

class MaxFlow {

private:

char start; // исток

char end; //сток

std::map<char, std::map<char, double>> edges; // ребра с остаточной пропускной способностью

std::map<char, std::map<char, double>> capacity; // ребра с текущим потоком

std::set<char> visited; // посещенные вершины

std::vector<char> path; //путь

public:

MaxFlow() {};

void Read();

void PrintEdges(const std::map<char, double>& e, const char& vert);

void PrintFront(const std::queue<char>& f);

void PrintCapacity();

bool IsNewFront(const std::queue<char>& nf, char v);

void FindPath();

double AlgorithmFF();

};

void MaxFlow::Read() { //считывание данных

int n;

#ifdef INFO

std::cout << "Введите количество ориентированных ребер графа:\n";

#endif

std::cin >> n;

#ifdef INFO

std::cout << "Введите исток и сток:\n";

#endif

std::cin >> start >> end;

char first, second;

double weight;

#ifdef INFO

std::cout << "Введите ребра графа и их пропускную способность:\n";

#endif

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

std::cin >> first >> second >> weight;

edges[first][second] = weight;

}

}

void MaxFlow::PrintEdges(const std::map<char, double>& e, const char& ver) { //вывод ребер, исходящих из одной вершины

for (auto& el : e)

std::cout << ver << '-' << el.first << ": " << el.second << '\n';

}

void MaxFlow::PrintFront(const std::queue<char>& front) { //вывод фронта

std::queue<char> tmp = front;

std::vector<char> arr;

while (!tmp.empty()) {

arr.push\_back(tmp.front());

tmp.pop();

}

for (auto& el : arr)

std::cout << el;

std::cout << '\n';

}

void MaxFlow::PrintCapacity() { //вывод графов и их пропускной способности

for (auto& a : capacity)

for (auto& b : a.second)

std::cout << a.first << ' ' << b.first << ' ' << b.second << '\n';

}

bool MaxFlow::IsNewFront(const std::queue<char>& f, char v) { //проверка на новый фронт

auto tmp = f;

while (!tmp.empty()) {

char cur = tmp.front();

tmp.pop();

if (cur == v)

return true;

}

return false;

}

void MaxFlow::FindPath() { //функция поиска пути в ширину

char current = start; // текущая вершина

bool IsFound = false; //конец поиска

std::queue<char> frontier, NewFrontier; // непросмотренные вершины с пропускной способностью

std::map<char, char> from; // значение, откуда взята вершина (вершина, предыдущая вершина на пути)

path.clear();

visited.clear();

#ifdef INFO

std::cout << "\nПоиск пути:\n";

#endif

visited.emplace(start);

while (!IsFound) { //пока путь не найден

std::vector<std::pair<char, double>> CurPathes; //ребра текущей вершины

#ifdef INFO

std::cout << "\nТекущая вершина: " << current << '\n';

#endif

if (edges.find(current) != edges.end()) { //получение ребер

auto found = edges.find(current)->second;

for (auto& el : found)

CurPathes.push\_back(std::make\_pair(el.first, el.second));

#ifdef INFO

PrintEdges(found, current);

#endif

}

else {

CurPathes = std::vector<std::pair<char, double>>();

#ifdef INFO

std::cout << "Нет ребер.\n";

#endif

}

auto iter\_visited = visited.end();

int n = (int)CurPathes.size();

for (auto& vert : CurPathes) { //добавление всех непосещенных соседей к фронту

#ifdef INFO

std::cout << "Проверка пути: " << current << '-' << vert.first << '\n';

#endif

//проверка, если не было посещено или если пропускная способность больше 0

if ((vert.second > 0) && (visited.find(vert.first) == visited.end()) &&

(!IsNewFront(NewFrontier, vert.first) || vert.second > edges[from[vert.first]][vert.first])) {

#ifdef INFO

std::cout << "Не было посещено ранее, пропускная способность больше 0 ==> добавление во фронт\n";

#endif

if (!IsNewFront(NewFrontier, vert.first)) // добавление во фронт

NewFrontier.push(vert.first);

from[vert.first] = current;

}

else

{

#ifdef INFO

std::cout << "Было посещено ранее или же вместимость равна 0.\n";

#endif

}

}

if (!IsFound) {

if (frontier.empty() && !(NewFrontier.empty())) { //переход к следующему фронту, если старый закончился; он становится посещенным

auto tmp = NewFrontier;

while (!tmp.empty()) {

char v = tmp.front();

if (v == end) {//проверка, если путь найден

#ifdef INFO

std::cout << "Путь найден. Текущая вершина конечная.\n";

#endif

IsFound = true;

break;

}

tmp.pop();

visited.emplace(v);

}

frontier = NewFrontier;

while (!NewFrontier.empty())

NewFrontier.pop();

}

else if (frontier.empty() && NewFrontier.empty()) {

#ifdef INFO

std::cout << "Путей больше нет.\n";

#endif

break;

}

#ifdef INFO

std::cout << "Текущий фронт:\n"; //получаем следующую вершину из фронта

PrintFront(frontier);

if (!NewFrontier.empty())

{

std::cout << "Новый фронт:\n";

PrintFront(NewFrontier);

}

#endif

current = frontier.front();

frontier.pop();

}

}

if (IsFound) { //получение пути

char get = end;

while (get != start) {

path.push\_back(get);

get = from[get];

}

path.push\_back(start);

std::reverse(path.begin(), path.end());

#ifdef INFO

std::cout << "Путь: ";

for (auto& v : path)

std::cout << v;

std::cout << '\n';

#endif

}

}

double MaxFlow::AlgorithmFF() { //алгоритм Форда-Фалкерсона

#ifdef INFO

std::cout << "\nАлгоритм Форда-Фалкерсона:\n";

#endif

double flow = 0; //максимальный поток

for (auto& v1 : edges) //пропускная способность обнуляется

for (auto& v2 : v1.second)

capacity[v1.first][v2.first] = 0;

while (FindPath(), !path.empty()) { //если путь существует

char first = start;

char second;

#ifdef INFO

std::cout << "Пропускная способность пути:\n "; //подсчет минимальной пропускной способности пути

#endif

double MinCapacity = 1000;

for (int i = 1; i < path.size(); i++) {

second = path[i];

double CurCapacity = edges[first][second];

if (CurCapacity < MinCapacity)

MinCapacity = CurCapacity;

#ifdef INFO

std::cout << first << "-(" << CurCapacity << ")-" << second << ' ';

#endif

first = second;

}

#ifdef INFO

std::cout << "\nМинимальная пропускная способность: " << MinCapacity << '\n';

#endif

first = start; //обновление графа

for (int i = 1; i < path.size(); i++) {

second = path[i];

edges[first][second] -= MinCapacity;

edges[second][first] += MinCapacity;

first = second;

}

flow += MinCapacity; //обновление текущего максимального потока

first = start; //обновление пропускной способности

for (int i = 1; i < path.size(); i++) {

second = path[i];

if (capacity.find(first) != capacity.end() && capacity[first].find(second) != capacity[first].end())

capacity[first][second] += MinCapacity;

else

capacity[second][first] -= MinCapacity;

first = second;

}

}

return flow;

}

int main() {

setlocale(LC\_ALL, "Russian");

MaxFlow f;

//f = MaxFlow();

f.Read();

double res = f.AlgorithmFF();

#ifdef INFO

std::cout << "\n\nОтвет:\n";

#endif

std::cout << res << '\n'; //максимальный поток

f.PrintCapacity();

return 0;

}