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: Потоки в сети

Студентка гр. 9383 _____ Лихашва А.Д.

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.

Жадный алгоритм.

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, соотвествующий минимальной из пропускных способностей ребер этого пути. Другими словами, от всех ребер отнимается значение данного потока, а к обратным ребам 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).

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 непосещенные 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 следущая 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	10 a b 4 a c 6

a b 6 a c 7 b d 4 c f 6 d e 3 d f 5 d m 2 e c 2	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 lab3.h

```
#pragma once

#include <iostream>
#include <map>
#include <string>
#include <set>
#include <vector>
#include <queue>
#include <algorithm>
#include <cmath>


class Finding {
private:
    char start;          // исток
    char end;  //сток
     std::map<char, std::map<char, double>> edges;         // ре-
бра с пропускной способностью
     std::map<char, std::map<char, double>> capacity;     // ре-
бра с текущим потоком
     std::set<char> visited;  // посещенные вершины
     std::vector<char> path;  //путь

public:
    Finding() {};
    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();
};
```

Файл lab3.cpp

```
#include "lab3.h"


void Finding::Read() {   //считывание данных
    std::string str;
    int n;
```

```
    std::cout << "Введите количество ориентированных ребер гра-
фа:\n";
    std::cin >> n;
    std::cout << "Введите исток и сток:\n";
    std::cin >> start >> end;
    char first, second;
    double weight;
    std::cout << "Введите ребра графа и их пропускную способ-
ность:\n";
    for (int i = 0; i < n; i++) {
        std::cin >> first >> second >> weight;
        edges[first][second] = weight;
    }
}


void Finding::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 Finding::PrintFront(const std::queue<char>& front) {  //вы-
вод 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 Finding::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 Finding::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 Finding::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 верши-
на (вершина, предыдущая вершина на пути)

    path.clear();
    visited.clear();

    std::cout << "\nПоиск пути:\n";
    visited.emplace(start);

    while (!IsFound) {  //пока путь не найден
        std::vector<std::pair<char, double>> CurPathes; //ребра
текущей вершины
        std::cout << "\nТекущая вершина: " << current << '\n';
        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));
            PrintEdges(found, current);
        }
        else {
            CurPathes = std::vector<std::pair<char, double>>();
            std::cout << "Нет ребер.\n";
        }

        auto iter_visited = visited.end();
        int n = (int)CurPathes.size();
        for (auto& vert : CurPathes) { //добавление всех непосе-
щенных соседей к фронту
            std::cout << "Проверка пути: " << current << '-' <<
vert.first << '\n';
            //проверка, если не было посещено или если пропуск-
ная 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])) {
                std::cout << "Не было посещено ранее, пропускная
способность больше 0 ==> добавление во фронт\n";
```

```
                if (!IsNewFront(NewFrontier, vert.first)) // до-
бавление во фронт
                    NewFrontier.push(vert.first);
                from[vert.first] = current;
            }
            else
                std::cout << "Было посещено ранее или же вмести-
мость равна 0.\n";
        }
        if (!IsFound) {
            if (frontier.empty() && !(NewFrontier.empty())) { //
переход к следующему фронту, если старый закончился; он стано-
вится посещенным
                auto tmp = NewFrontier;
                while (!tmp.empty()) {
                    char v = tmp.front();
                    if (v == end) {//проверка, если путь найден
                        std::cout << "Путь найден. Текущая вер-
шина конечная.\n";
                        IsFound = true;
                        break;
                    }
                    tmp.pop();
                    visited.emplace(v);
                }
                frontier = NewFrontier;
                while (!NewFrontier.empty())
                    NewFrontier.pop();
            }
            else if (frontier.empty() && NewFrontier.empty()) {
                std::cout << "Путей больше нет.\n";
                break;
            }
            std::cout << "Фронт:\n"; //получаем следующую верши-
ну из фронта
            PrintFront(frontier);
            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());

        std::cout << "Путь: ";
        for (auto& v : path)
            std::cout << v;
        std::cout << '\n';
```

```
    }
}


double Finding::AlgorithmFF() {  //алгоритм Форда-Фалкерсона
    std::cout << "\nАлгоритм Форда-Фалкерсона:\n";
    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 a = start;
        char b;
        std::cout << "Пропускная способность пути:\n  "; //
подсчет минимальной пропускной способности пути
        double MinCapacity = 1000;
        for (int i = 1; i < path.size(); i++) {
            b = path[i];
            double CurCapacity = edges[a][b];
            if (CurCapacity < MinCapacity)
                MinCapacity = CurCapacity;
            std::cout << a << "-(" << CurCapacity << ")-" << b
<< " ";
            a = b;
        }
        std::cout << "\nМинимальная пропускная способность: " <<
MinCapacity << '\n';
        a = start;  //обновление графа
        for (int i = 1; i < path.size(); i++) {
            b = path[i];
            edges[a][b] -= MinCapacity;
            edges[b][a] += MinCapacity;
            a = b;
        }
        flow += MinCapacity; //обновление текущего максимального
потока
        a = start; //обновление пропускной способности
        for (int i = 1; i < path.size(); i++) {
            b = path[i];
            if (capacity.find(a) != capacity.end() &&
capacity[a].find(b) != capacity[a].end())
                capacity[a][b] += MinCapacity;
            else
                capacity[b][a] -= MinCapacity;
            a = b;
        }
    }

    return flow;
}
```

```
int main() {
    setlocale(LC_ALL, "Russian");
    Finding f;
    f = Finding();
    f.Read();
    double res = f.AlgorithmFF();
    std::cout << "\n\n_Результат работы_:\n";
    std::cout << res << '\n';  //максимальный поток
    f.PrintCapacity();
    return 0;
}
```