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. 9383 _____ Нистратов Д.Г.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Форда-Фалкерсона для нахождения максимального потока в сети.

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).

Задание (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 работы алгоритма.

Инициализация: для всех ребер положим остаточную пропускную способность равной первоначальной.

Шаг 1. Определяем узлы, в которые можно перейти с максимальной остаточной пропускной способностью. Если таких не существует переходим к 3 шагу.

Шаг 2. В множестве узлов находим максимальный и помечаем его как посещенный. Если узел равен финишу переходим к 4 шагу.

Шаг 3. Если путь невозможен переходим к 5 шагу. Если список узлов не пустой, удаляем последний элемент и переходим к 1 шагу.

Шаг 4. Вычисляем максимальный поток по принципу, минимальное значение потока на всех узлах.

Шаг 5. Вывод узлов и максимальной пропускной способности.

Анализ алгоритма.

Путь в остаточной сети в среднем находится за время O(N). В худшем за O(2N). Где N - количество ребер в графе

Сложность алгоритма Форда-Фалкерсона в среднем будет равна O(fN), а в худшем случае O(f2N), где f – максимальный поток в графе.

Описание основных функций.

Хранение ребер реализовано с помощью стандартного контейнера c++ std::map<char, map<char, int>>. Реализация функций алгоритма описана в классе FFsolver.

Список функций:

std::pair<int, EDGE_MAP> FordFulkerson(); - основная функция, алгоритм Форда-Фалкерсона

void print_edges(EDGE_MAP cEdges); - вывод всех граней в терминал

void input(); - считывание потока с терминала

void setInput(std::stringstream& ss); - считывание потока строки.

bool isVisited(char node); - проверка узла на посещение

bool findPath(EDGE_MAP& cEdges); - поиск пути по наиб. Пропускной способности

char findMaxFlow(std::map<char, int> node); - поиск наибольшей пропускной способности.

Тестирование.

Тесты проведены с помощью сторонней библиотеки catch2 и описаны в файле test.cpp.

Проверяются две основные функции алгоритма. В поиске пути осуществляется проверка посещенности узлов, нахождение по заданному графу, а также проверку на н ахождение максимальной пропускной способности. Проверка алгоритма Форда-Фалкерсона описана в файле test.cpp, а также в таблице 1.

Таблица 1. Результаты тестирования

Ввод	Вывод
7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	7 a f a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2
7 1 7 1 2 7 1 3 6 2 4 6 3 7 9 4 5 3 4 7 4 5 3 2	7 1 7 1 2 6 1 3 6 2 4 6 3 7 8 4 5 2 4 7 4 5 3 2
3 a d a b 2 b c 3 c d 2	3 a d a b 2 b c 2 c d 2

Выводы.

При выполнении работы был изучен и реализован алгоритм Форда-Фалкерсона для нахождения максимального потока в сети. Поиск пути для реализации алгоритма был описан по правилу: переход осуществляется по максимальному значение остаточной пропускной способности. Написанный алгоритм выполняется со сложностью O(fN), в среднем случае, и O(2fN), в худшем случае.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД.

Название файла: main.cpp

```
#include "ford_falk.h"

int main(){
    FFsolver solver;
    solver.input();
    auto answer = solver.FordFulkerson();
    std::cout << answer.first << std::endl;
    solver.print_edges(answer.second);
    return 0;
}
```

Название файла: ford_falk.h

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>
#include <map>

#define EDGE_MAP std::map<char, std::map<char, int>>

class FFsolver
{
public:
    FFsolver() {};

    EDGE_MAP edges;
    char start, finish;
    std::vector<char> visited, cameFrom;

    std::pair<int, EDGE_MAP> FordFulkerson();

    void print_edges(EDGE_MAP cEdges);

    void input();

    void setInput(std::stringstream& ss);
```

```
    bool isVisited(char node);

    bool findPath(EDGE_MAP& cEdges);

    char findMaxFlow(std::map<char, int> node);
};
```

Название файла: ford_falk.cpp

```
#include "ford_falk.h"

bool FFsolver::isVisited(char node){
    for (auto name : visited){
        if (name == node){
            return true;
        }
    }
    return false;
}

bool FFsolver::findPath(EDGE_MAP& cEdges){
    visited.push_back(start);
    cameFrom.push_back(start);
    char currNode = start;
    for (;;){
        if (currNode == finish){
            return true;
        }
        currNode = findMaxFlow(cEdges[currNode]);
        if (currNode == ' '){
            if (cameFrom.empty()){
                return false;
            }
            cameFrom.pop_back();
            currNode = cameFrom.back();
        }
        else{
            cameFrom.push_back(currNode);
```

```
            visited.push_back(currNode);
        }
    }
}

char FFsolver::findMaxFlow(std::map<char, int> node){
    char nodeName = ' ';
    int tempFlow = 0;
    if (node.empty()){
        return nodeName;
    }
    for (auto a : node){
        if (a.second > tempFlow && !isVisited(a.first)){
            nodeName = a.first;
            tempFlow = a.second;
        }
    }
    return nodeName;
}

std::pair<int, EDGE_MAP> FFsolver::FordFulkerson(){
    EDGE_MAP cEdges = edges;
    EDGE_MAP dEdges = edges;
    int max_flow = 0;
    int tmp = INT_MAX;
    while (findPath(cEdges)){
        for (int i = 0; i < cameFrom.size() - 1; i++){
            tmp = std::min(tmp, cEdges[cameFrom[i]][cameFrom[i+1]]);
        }
        max_flow += tmp;
        for (int i = 0; i < cameFrom.size() - 1; i++){
            cEdges[cameFrom[i]][cameFrom[i+1]] -= tmp;
            cEdges[cameFrom[i+1]][cameFrom[i]] += tmp;
        }
        visited.clear();
        cameFrom.clear();
    }
    for (auto& kv1 : edges){
```

```
        for (auto& kv2 : kv1.second){
            kv2.second -= cEdges[kv1.first][kv2.first];
            if (kv2.second < 0){
                kv2.second = 0;
            }
        }
    }
    return std::make_pair(max_flow, edges);
}

void FFsolver::print_edges(EDGE_MAP cEdges){
    for (auto& kv1 : cEdges){
        for (auto& kv2 : kv1.second){
            std::cout << kv1.first << " " << kv2.first << " " << kv2.second << std::endl;
        }
    }
}

void FFsolver::input(){
    int n;
    std::cin >> n >> start >> finish;
    char node1, node2;
    int weight;
    for (int i = 0; i < n; i++){
        std::cin >> node1 >> node2 >> weight;
        edges[node1][node2] = weight;
    }
}

void FFsolver::setInput(std::stringstream& ss){
    int n;
    ss >> n >> start >> finish;
    char node1, node2;
    int weight;
    for (int i = 0; i < n; i++){
        ss >> node1 >> node2 >> weight;
        edges[node1][node2] = weight;
    }
```

```
}
```

Название файла: test.cpp

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"
#include <sstream>

#include "../source/ford_falk.h"

TEST_CASE("Поиск пути по наибольшей остаточной пропускной способности" ) {
   std::stringstream ss;
   FFsolver ff;
   SECTION("Путь существует") {
      ss << "3 a e a b 2 b e 3 a c 1";
      ff.setInput(ss);
      REQUIRE(ff.findPath(ff.edges) == true);
   }
   SECTION("Путь существует, наибольшая остаточная пропускная способность приводит в тупик") {
      ss << "3 a e a b 2 b e 3 a c 7";
      ff.setInput(ss);
      REQUIRE(ff.findPath(ff.edges) == true);
   }
   SECTION("Путь не существует") {
      ss << "3 a e a b 2 b c 3 a c 1";
      ff.setInput(ss);
      REQUIRE(ff.findPath(ff.edges) == false);
   }
   SECTION("Путь существует, но все пути посещены") {
      ss << "3 a e a b 2 b e 3 a c 1";
      ff.visited.push_back('e');
      ff.visited.push_back('b');
      ff.setInput(ss);
      REQUIRE(ff.findPath(ff.edges) == false);
   }
}

TEST_CASE("Алгоритм Форда-Фалкерсона"){
   std::stringstream ss;
```

```
    FFsolver ff;
    SECTION("Граф соеденен"){
        ss << "7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2";
        ff.setInput(ss);
        auto a = ff.FordFulkerson();
        ss << "7 a f a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2";
        ff.setInput(ss);
        REQUIRE(a.first == 12);
        REQUIRE(a.second == ff.edges);
    }
    SECTION("Граф задан числами"){
        ss << "7 1 7 1 2 7 1 3 6 2 4 6 3 7 9 4 5 3 4 7 4 5 3 2";
        ff.setInput(ss);
        auto a = ff.FordFulkerson();
        ss << "7 1 7 1 2 6 1 3 6 2 4 6 3 7 8 4 5 2 4 7 4 5 3 2";
        ff.setInput(ss);
        REQUIRE(a.first == 12);
        REQUIRE(a.second == ff.edges);
    }
    SECTION("Ноды в цикле"){
        ss << "3 a d a b 2 b c 3 c d 2";
        ff.setInput(ss);
        auto a = ff.FordFulkerson();
        ss << "3 a d a b 2 b c 2 c d 2";
        ff.setInput(ss);
        REQUIRE(a.first == 2);
        REQUIRE(a.second == ff.edges);
    }
}
```

```
        print(v1,v2, 0)
```