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 _____ Камзолов Н.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Познакомиться на практике с алгоритмом Форда-Фалкерсона, реализовать данный алгоритм на языке программирования C++.

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

v_0 – исток

v_n– сток

v_i v_i ω_{ij} – ребро графа

v_i v_i ω_{ij} – 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).

Пример входных данных:

7

a f

a b 7

a c 6

b d 6

c f 9

d e 3

d f 4

e c 2

Соответствующие выходные данные

12

a b 6

a c 6

b d 6

c f 8

d e 2

d f 4

e c 2

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.

Основные теоретические положения.

Сеть – ориентированный взвешенный граф, имеющий один исток и один сток. Исток – вершина, из которой рёбра только выходят*.

Сток – вершина, в которую рёбра только входят*.

Поток – абстрактное понятие, показывающее движение по графу.

Величина потока – числовая характеристика движения по графу (сколько всего выходит из истока = сколько всего входит в сток).

Пропускная способность – свойство ребра, показывающее, какая максимальная величина потока может пройти через это ребро.

Максимальный поток (максимальная величина потока) – максимальная величина, которая может быть выпущена из истока, которая может пройти через все рёбра графа, не вызывая переполнения ни в одном ребре.

Фактическая величина потока в ребре – значение, показывающее, сколько величины потока проходит через это ребро.

Алгоритм Форда-Фалкерсона – алгоритм, решающий задачу нахождения максимального потока в сети.

Описание алгоритма.

В первую очередь, строится список смежности, содержащий информацию о графе. Все потоки равны 0, а пропускные способности ребер – произвольному целому числу.

Далее начинается работа самого алгоритма:

1. Запускается поиск пути в графе из истока в сток.
2. Если путь найден, то для него вычисляется максимальный поток.
3. Для всех ребер пути поток увеличивается на величину максимального потока, а пропускная способность уменьшается на эту величину.
4. Значение максимального потока на текущем шаге прибавляется к конечному значению максимального потока всего графа, а затем поиск пути запускается заного
5. Алгоритм останавливается, если на очередном шаге не было найдено пути из истока в сток.

Сложность по памяти линейная – O(E), где E – число ребер.

Сложность по времени O(E*f), где E – число ребер, f – максимальный поток в графе.

Описание функций и структур данных.

struct Edge – структура, которая хранит информацию о ребре графа. Здесь есть информация о пропускной способности и потоке, а также логическая переменная, которая хранит информацию о том, было ли ребро

изначально в графе или его добавили туда для корректной работы алгоритма.

class FordFulkerson – класс, который хранит методы и переменные для работы алгоритма Форда-Фалкерсона:

- std::map<char, std::vector<std::pair<char, Edge>>> graph – хранит информацию о всем графе.
- char start, finish – переменные, которые хранят информацию о вершинах истока и стока.
- std::map<char, bool> visited – хранит информацию, о посещенных вершинах на очередном шаге алгоритма.
- std::map<char, char> parent – хранит информацию о пути из истока в сток на очередном шаге алгоритма для последующего пересчета максимального потока.
- int maxFlow – переменная, которая хранит информацию о максимальном пропускном потоке для всего графа.
- search(char curVertex) – метод, который рекурсивно выполняет поиск и построение пути из истока в сток на каждом шаге алгоритма. Обход графа выполняется по следующему 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.
- fordFulkersonAlgo() – основной метод алгоритма, в котором циклично происходит вызов метода search(), а также находится максимальный поток пути и пересчитываются потоки на ребрах графа каждый шаг алгоритма. Также здесь пересчитывается максимальный пропускной поток для всего графа. Если на очередном шаге алгоритма путь не был найден, то метод заканчивает работу.

Функция flowComparator(std::pair<char, Edge> first, std::pair<char, Edge> second) – нужна для сортировки ребер по величине их максимальной остаточной пропускной способности.

Функция charComparator(std::pair<char, Edge> first, std::pair<char, Edge> second) **–** нужна для сортировки ребер в лексикографическом порядке.

Функция readGraph() – функция, которая нужна для считывания графа из входного потока.

Было написано тестирование для некоторых функций программы:

1. Тестирование функции readGraph() – была протестирована корректная обработка верных входных данных, а также отлавливание неверных входных данных.
2. Тестирование функции search() – было протестировано корректное нахождение пути в графе из истока в сток с разными входными данными. Функция успешно отлавливает ситуации, когда пути из истока в сток нет.
3. Тестирование функции fordFulkersonAlgo() – было протестировано корректное нахождение максимального потока в графе с помощью алгоритма Форда-Фалкерсона.

Тестирование.

Входные данные	Выходные данные
7 a f a b 7 a c 6 b d 6	12 a b 6 a c 6 b d 6 c f 8 d e 2

c f 9 d e 3 d f 4 e c 2	d f 4 e c 2
16 a e a b 20 b a 20 a d 10 d a 10 a c 30 c a 30 b c 40 c b 40 c d 10 d c 10 c e 20 e c 20 b e 30 e b 30 d e 10 e d 10	60 a b 20 a c 30 a d 10 b a 0 b c 20 b e 30 c a 0 c b 30 c d 0 c e 20 d a 0 d c 0 d e 10 e b 0 e c 0 e d 0
9 a d a b 8 b c 10 c d 10	18 a b 8 a g 10 b c 0 b e 8 c d 10

h c 10 e f 8 g h 11 b e 8 a g 10 f d 8	e f 8 f d 8 g h 10 h c 10
5 a d a b 20 a c 1 b c 20 b d 1 c d 20	21 a b 20 a c 1 b c 20 b d 1 c d 20

Выводы.

На практике были применены теоретические знания об алгоритме Форда-Фалкерсона. Была успешно написана программа, реализующая данный алгоритм. Был экспериментально применен обход графа, который выполняется 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. Данный обход относительно не эффективен, так как каждый шаг поиска нам нужно находить максимальный элемент в массиве.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include "FordFulkerson.h"

int main() {
    std::map<char, std::vector<std::pair<char, Edge>>> graph;
    char start, finish;
    int size;

    readGraph(graph, size, start, finish, std::cin);
    std::map<char, std::vector<std::pair<char, Edge>>> indicator = {};
    if(graph == indicator) {
        std::cout << "Wrong input!\n";
    }
    FordFulkerson algosRun = FordFulkerson(graph, start, finish, size);
    algosRun.fordFulkersonAlgo();
    algosRun.printAnswer();
}
```

Файл FordFulkerson.h:

```
#ifndef FORDFULKERSON_H
#define FORDFULKERSON_H

#include <iostream>
#include <vector>
#include <queue>
#include <map>
#include <algorithm>
#include <climits>
#include <sstream>


struct Edge {
    int capacity, flow;
    bool onStart;
    bool used = false;
};

inline bool operator==(const Edge& lhs, const Edge& rhs) {
    return (lhs.capacity == rhs.capacity && lhs.flow == rhs.flow &&
lhs.onStart == rhs.onStart);
}

class FordFulkerson {
public:
    std::map<char, std::vector<std::pair<char, Edge>>> graph;
    char start, finish;
    int size;
    std::map<char, bool> visited;
    std::map<char, char> parent;
    int maxFlow = 0;
```

```
    FordFulkerson(
        std::map<char,
        std::vector<std::pair<char, Edge>>> graph,
        char start,
        char finish,
        int size
    ) : graph(graph), start(start), finish(finish), size(size) {}

    void printAnswer();
    int search(char curVertex);
    void fordFulkersonAlgo();

};

bool flowComparator(std::pair<char, Edge> first, std::pair<char, Edge>
second);

bool charComparator(std::pair<char, Edge> first, std::pair<char, Edge>
second);


void readGraph(
    std::map<char, std::vector<std::pair<char, Edge>>>& graph,
    int& size,
    char& start,
    char& finish,
    std::istream& in
);

#endif
```

Файл FordFulkerson.cpp:

```
#include "FordFulkerson.h"

bool flowComparator(std::pair<char, Edge> first, std::pair<char, Edge>
second) {
    return first.second.capacity > second.second.capacity;
}

bool charComparator(std::pair<char, Edge> first, std::pair<char, Edge>
second) {
    return first.first < second.first;
}

void readGraph(
    std::map<char, std::vector<std::pair<char, Edge>>>& graph,
    int& size,
    char& start,
    char& finish,
    std::istream& in
) {
    in >> size;
    in >> start >> finish;
    char tempFrom, tempTo;
    int tempFlow;
```

```
    for(int i = 0; i < size; i++) {
        in >> tempFrom >> tempTo >> tempFlow;
        if(tempFlow < 0) {
            graph = {};
            return;
        }
        graph[tempFrom].push_back({tempTo, {tempFlow, 0, true}});
        graph[tempTo].push_back({tempFrom, {0, tempFlow, false}});
    }
}



int FordFulkerson::search(char curVertex){
    if (curVertex == finish) return true;
    visited[curVertex] = true;
    std::sort(graph[curVertex].begin(), graph[curVertex].end(),
flowComparator);
    for(auto& neighbour : graph[curVertex]) {
        if(!visited[neighbour.first] && neighbour.second.capacity > 0) {
            parent[neighbour.first] = curVertex;
            if(search(neighbour.first)){
                neighbour.second.used = true;
                for(auto& rNeighbour : graph[neighbour.first]) {
                    if(rNeighbour.first == curVertex)
                    {
                        if(rNeighbour.second.flow ==
neighbour.second.capacity){
                            if(rNeighbour.second.capacity ==
neighbour.second.flow)
                                rNeighbour.second.used = true;
                            }
                    }
                }
                return true;
            }
        }
    }
    return false;
}

void FordFulkerson::fordFulkersonAlgo(){
    while(search(start)) {
        int tempFlow = INT_MAX;
        for(char v = finish; v != start; v = parent[v]) {
            char u = parent[v];
            for(size_t i = 0; i < graph[u].size(); i++) {
                if(graph[u][i].first == v) {
                    if(!graph[u][i].second.used) continue;
                    tempFlow = std::min(tempFlow,
graph[u][i].second.capacity);
                }
            }
        }

        for (char v = finish; v != start; v = parent[v]) {
            char u = parent[v];
```

```
            for(size_t i = 0; i < graph[u].size(); i++) {
                if(graph[u][i].first == v && graph[u][i].second.used) {
                    graph[u][i].second.capacity -= tempFlow;
                    graph[u][i].second.flow += tempFlow;
                    graph[u][i].second.used = false;
                }

            }
            for(size_t i = 0; i < graph[v].size(); i++) {
                if(graph[v][i].first == u && graph[u][i].second.used) {
                    graph[v][i].second.capacity += tempFlow;
                    graph[v][i].second.flow -= tempFlow;
                    graph[v][i].second.used = false;
                }
            }
        }
        visited = {};
        parent = {};
        maxFlow += tempFlow;
    }
}

void FordFulkerson::printAnswer() {
    std::cout << maxFlow;
    std::cout << '\n';
    for (auto& from : graph) {
        if(from.second.size() > 0) {
            std::sort(graph[from.first].begin(), graph[from.first].end(),
charComparator);
            for (auto& to : from.second) {
                if(to.second.onStart)
                    std::cout << from.first << ' ' <<to.first << ' ' <<
to.second.flow << '\n';
            }
        }
    }
}
```