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

Студентка гр. 9382 _____ Пя С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2021

Цель работы.

Реализовать программу, занимающуюся нахождением величины потока в ориентированном графе, используя алгоритм Форда-Фалкерсона.

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
$v_i\, v_j\, \omega_{ij}$ - ребро графа
$v_i\, v_j\, \omega_{ij}$ - ребро графа
...

Выходные данные:
P_{max} - величина максимального потока
$v_i\, v_j\, \omega_{ij}$ - ребро графа с фактической величиной протекающего потока
$v_i\, v_j\, \omega_{ij}$ - 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 3.

Поиск в глубину. Рекурсивная реализация.

Описание структуры данных, используемой для представления графа.

Для представления графа используется структура данных – std::map<Vertical, std::list<Edge>> graph. Ключом является вершина, а в значением – список вершин, до которых возможен путь, образуя ребро. Оно описывается структурой Edge, полями которой являются название вершины, поток, который через ребро можно пропустить и поток, который уже был пропущен через нее: Vertical vertical, Weight stream, Weight pastedStream соответственно.

Описание алгоритма.

Сначала считываем количество ребер, затем сами ребра с весом, записывая их в граф, который является полем класса MaxStreamSearching. Далее вызываем метод класса doAlgoritm, который отвечает за поиск величины суммарного потока. Пока поток последнего построенного пути не будет равен нулю (то есть пока пути для прохода по графу существуют), алгоритм будет искать его, суммируя с предыдущими значениями, при этом сбрасывая статус

вершин на не посещенные. Этот метод имеет в себе рекурсивный метод doSearchInDepth, предназначенный для поиска пути от начальной вершины до конечной. Выход из рекурсии осуществляется при равенстве текущей рассматриваемой вершины и конечной или при отсутствии возможных путей к конечной вершине. Метод у текущей вершины рассматривает возможный путь, проверяя не была ли посещена следующая вершина и возможно ли пропустить через нее еще поток. Как только алгоритм нашел минимально возможный поток, в граф записывается сумма этого потока и ранее пропущенного в случае нахождения пути. Минимальное значение выбирается между разностями изначального заданного потока и уже пропущенного потока ребер, из которых состоит текущий путь. Так как в граф записываются ранее пропущенный поток, алгоритм рано или поздно закончится, так как возможного пути не останется. Поиск пути от начальной до конечной вершины осуществляется на базе поиска в глубину.

Оценка сложности по памяти

Каждая вершина имеет список других доступных вершин, поэтому сложность равна O(N^2).

Оценка сложности по времени

Алгоритм заканчивается тогда, когда рекурсивный метод возвращает нуль, значит, на каждом шаге выполнения алгоритм возвращает хотя бы 1. Значит, максимум алгоритм совершит N шагов, где N – максимальный поток в графе. Каждый шаг можно выполнить за O(T), где T – число ребер в графе, тогда сложность по времени ограничена O(NT).

Тестирование алгоритма.

Нумерация	Входные данные	Выходные данные
1	7 a f a b 7	Хотите считать данные из фай самостоятельно?(1/2) 1 Начинается работа алгоритма

	a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	Текущая вершина a, a текущий мин 10000 Отмечаем вершину посещенной Рассматриваем ребра до вершин c b Рассмотрим ребро до вершины c Ребро было включено в путь. Выполня поиск минимального потока Текущая вершина c, a текущий минима Отмечаем вершину посещенной Рассматриваем ребра до вершин f Рассмотрим ребро до вершины f Ребро было включено в путь. Выполня поиск минимального потока Текущая вершина f, a текущий минима Конечная вершина достигнута, алгорит Минимальный поток, найденный в теку a b 0 a c 6 b d 0 c f 6 d e 0 d f 0 e c 0 Текущая вершина a, a текущий мин 10000 Отмечаем вершину посещенной Рассматриваем ребра до вершин c b Рассмотрим ребро до вершины c Вершина была уже посещена, либо чер пропущено максимальная величина потока Рассмотрим ребро до вершины b Ребро было включено в путь. Выполня поиск минимального потока

		Текущая вершина b, а текущий минима Отмечаем вершину посещенной Рассматриваем ребра до вершин d Рассмотрим ребро до вершины d Ребро было включено в путь. Выполня поиск минимального потока Текущая вершина d, а текущий минима Отмечаем вершину посещенной Рассматриваем ребра до вершин f e Рассмотрим ребро до вершины f Ребро было включено в путь. Выполня поиск минимального потока Текущая вершина f, а текущий минима Конечная вершина достигнута, алгорит Минимальный поток, найденный в теку a b 4 a c 6 b d 4 c f 6 d e 0 d f 4 e c 0 Текущая вершина a, а текущий мин 10000 Отмечаем вершину посещенной Рассматриваем ребра до вершин c b Рассмотрим ребро до вершины c Вершина была уже посещена, либо чере пропущено максимальная величина потока Рассмотрим ребро до вершины b Ребро было включено в путь. Выполня поиск минимального потока Текущая вершина b, а текущий минима

		Отмечаем вершину посещенной Рассматриваем ребра до вершин d Рассмотрим ребро до вершины d Ребро было включено в путь. Выполня поиск минимального потока Текущая вершина d, а текущий минима Отмечаем вершину посещенной Рассматриваем ребра до вершин f e Рассмотрим ребро до вершины f Вершина была уже посещена, либо чер пропущено максимальная величина потока Рассмотрим ребро до вершины e Ребро было включено в путь. Выполня поиск минимального потока Текущая вершина e, а текущий минима Отмечаем вершину посещенной Рассматриваем ребра до вершин c Рассмотрим ребро до вершины c Ребро было включено в путь. Выполня поиск минимального потока Текущая вершина c, а текущий минима Отмечаем вершину посещенной Рассматриваем ребра до вершин f Рассмотрим ребро до вершины f Ребро было включено в путь. Выполня поиск минимального потока Текущая вершина f, а текущий минима Конечная вершина достигнута, алгорит Минимальный поток, найденный в теку a b 6 a c 6 b d 6 c f 8

		d e 2 d f 4 e c 2 Текущая вершина a, а текущий мин 10000 Отмечаем вершину посещенной Рассматриваем ребра до вершин с b Рассмотрим ребро до вершины c Вершина была уже посещена, либо чере пропущено максимальная величина потока Рассмотрим ребро до вершины b Ребро было включено в путь. Выполняе поиск минимального потока Текущая вершина b, а текущий минима Отмечаем вершину посещенной Рассматриваем ребра до вершин d Рассмотрим ребро до вершины d Вершина была уже посещена, либо чере пропущено максимальная величина потока Минимальный поток, найденный в теку a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2 Алгоритм завершился Суммарный поток 12 Ребро графа с фактической величино потока a b 6 a c 6

		b d 6 c f 8 d e 2 d f 4 e c 2 Хотите продолжить?(y/n) n
2	8 a h a c 8 a d 2 c d 16 c f 4 d g 6 g f 18 g h 5 f h 5	Суммарный поток 10 Ребро графа с фактической величино потока a c 8 a d 2 c d 4 c f 4 d g 6 f h 5 g f 1 g h 5
3	5 a e a b 8 b c 10 b e 3 a e 4 c e 2	Суммарный поток 9 Ребро графа с фактической величино потока a b 5 a e 4 b c 2 b e 3 c e 2
4	5 b d a c 5 a b 6 c d 3	Суммарный поток 2 Ребро графа с фактической величино потока a b 0 a c 0 a d 0

	b c 2 a d 4	b c 2 c d 2

Выводы.

Был изучен алгоритм Форда-Фалкерсона. Реализована программа для поиска величины потока в графе.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <map>
#include <algorithm>
#include <list>
#include <fstream>

using Vertical = char;
using Weight = int;

struct Edge {//стуктура для ребра
    Vertical vertical;
    Weight stream;//заданный поток
    Weight pastedStream;//пропущенный поток
    Edge(Vertical ver, Weight st, Weight fSt) {
        vertical = ver;
        stream = st;
        pastedStream = fSt;
    }
};

bool lowerSortThings (const Edge& first, const Edge& second) {
    return first.vertical > second.vertical;
}

bool upperSortThings (const Edge& first, const Edge& second) {
    return first.vertical < second.vertical;
}

class MaxStreamSearching {//класс для нахождения величины потока
private:
    std::map<Vertical, std::list<Edge>> graph;//стуктура для графа
    std::map<Vertical, bool> isVisited;//стуктура для отмечания посещенных
вершин
    Vertical initDestVer, finDestVer;//начальная и конечная вершины
public:
    MaxStreamSearching() {};
    void readData(std::istream& fin, bool isCin) {//ввод данных и
формирование списков
        int N = 0;
        if (isCin)
            std::cout << "Введите количество ребер\n";
        fin >> N;
        if (isCin)
            std::cout << "Введите исток и сток\n";
        fin >> initDestVer >> finDestVer;
        std::map<Vertical, std::list<Edge>>::iterator it;
        if (isCin)
            std::cout << "Введите ребра и вес\n";
        for (auto i = 0; i < N; i++) {
            Vertical initVer, finVer;
            Weight weight;
            fin >> initVer >> finVer >> weight;
            it = graph.find(initVer);
```

```
            if (it != graph.end()) {//если вершина уже записана в граф
                it->second.emplace_back(finVer, weight, 0);
            } else {
                isVisited.insert(std::pair<Vertical, bool>(initVer, false));
                std::list<Edge> listOfVer;
                listOfVer.emplace_back(finVer, weight, 0);
                graph.insert(std::pair<Vertical, std::list<Edge>>(initVer,
listOfVer));
            }
        }
        for (auto &key: graph) graph[key.first].sort(lowerSortThings);
    }

    Weight doSearchInDepth(Vertical ver, Weight Cmin) {
        std::cout << "Текущая вершина " << ver << ", а текущий минимальный
поток " << Cmin << "\n";
        if (ver == finDestVer) {
            std::cout << "Конечная вершина достигнута, алгоритм
завершается\n";
            return Cmin;
        }
        isVisited[ver] = true;
        std::cout << "Отмечаем вершину посещенной\n";
        std::cout << "Рассматриваем ребра до вершин ";
        for (auto& edge : graph.find(ver)->second) {
            std::cout << edge.vertical << " ";
        }
        std::cout << "\n";
        for (auto& edge : graph.find(ver)->second) {
            std::cout << "Рассмотрим ребро до вершины " << edge.vertical <<
"\n";
            if (!isVisited[edge.vertical] && edge.pastedStream < edge.stream)
{
                std::cout << "Ребро было включено в путь. Выполняется
рекурсивный поиск минимального потока\n";
                Weight curStream = doSearchInDepth(edge.vertical,
std::min(Cmin, edge.stream - edge.pastedStream));
                if (curStream > 0) {
                    edge.pastedStream += curStream;
                    return curStream;
                }
            } else std::cout << "Вершина была уже посещена, либо через ребро
уже было пропущено максимальная величина потока\n";
        }
        return 0;
    }

    void doAlgoritm() {
        Weight sum = 0, answer = -1;
        while (answer != 0) {
            answer = doSearchInDepth(initDestVer, 10000);
            std::cout << "Минимальный поток, найденный в текущем пути " <<
answer << "\n";
            writeData(0);
            for (auto& i : isVisited) {
                i.second = false;
            }
            sum += answer;
        }
        std::cout << "Алгоритм завершился\n";
        writeData(sum);
```

```
    }
    void writeData(int sum) {
        if (sum != 0) {
            std::cout << "Суммарный поток " << sum << "\n";
            std::cout << "Ребро графа с фактической величиной протекающего
потока\n";
        }
        for (auto i : graph) {
            i.second.sort(upperSortThings);
            for (auto j : i.second) {
                std::cout << i.first << " " << j.vertical << " " <<
j.pastedStream << "\n";
            }
        }
    }
};

int main() {
    char answ = 'y';
    while (answ == 'y') {
        std::cout << "Хотите считать данные из файла или ввести
самостоятельно?(1/2)\n";
        std::cin >> answ;
        auto answer = new MaxStreamSearching();
        if (answ == '2') {
            answer->readData(std::cin, true);
        } else {
            std::ifstream fin("test1.txt");
            answer->readData(fin, false);
            fin.close();
        }
        std::cout << "Начинается работа алгоритма\n";
        answer->doAlgoritm();
        std::cout << "Хотите продолжить?(y/n)\n";
        std::cin >> answ;
    }
    return 0;
}
```