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. 7383 _____ Маркова А. В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Содержание

Цель работы

Исследовать и реализовывать задачу нахождения максимального потока в сети, применяя алгоритм Форда – Фалкерсона.

Формулировка задачи: 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 – пропускная способность (веса).

Вариант 1с: граф представлен в виде списка смежности, поиск пути задаётся через поиск в ширину.

Входные данные: в первой строке указывается количество ориентированных рёбер графа, затем идут значения начальной и конечной вершин. Далее вводят данные о рёбрах графа и их весе, пропускной способности.

Выходные данные: максимальный поток в сети, а также фактическая величина потока, протекающего через каждую дугу, все рёбра отсортированы в лексикографическом порядке.

Реализация задачи

В данной работе используются главная функция `main()` и следующий класс:

```
class Graph {
private:
    char Parents[N];
    List Given_graph;
    List RealFlow;
    std:: array <bool, N> check;
    int ResidualCapacity(char from, char to);
    bool Bfs(char source, char sink);
public:
    Graph();
    void AddEdge(char from, char to, int capacity);
    int Ford_Fulkerson(char source, char sink);
    void Print();
    ~Graph();
};
```

Параметры, хранящиеся в классе:

- `Parents` – одномерный массив, в котором хранятся значения родителей вершин;
- `Given_graph` – упорядоченный ассоциативный массив типа `map`, благодаря которому заданный граф представлен как список смежности;
- `RealFlow` – упорядоченный ассоциативный массив типа `map`, в котором хранятся значения фактического потока, проходящего через все рёбра графа;

- `check` – массив типа `bool`, в котором хранятся метки, показывающие пройдена данная вершина или нет.

Методы класса:

- `ResidualCapacity` – функция нахождения разницы между величинами потока в заданном и фактическом графах, остаточная пропускная способность;
- `Bfs` – функция, реализующая поиск в ширину, возвращает `true`, если путь до стока существует, в противном случае – `false`;
- `AddEdge` – функция заполнения списка смежности, заданными значениями;
- `Ford_Fulkerson` – алгоритм Форда – Фалкерсона, возвращает максимальное значение потока, а также по ходу работы данной функции заполняются фактические значения потока;
- `Print` – функция вывода результата работы программы на экран.

Параметры, передаваемые в `int ResidualCapacity(char from, char to)`:

- `from` – вершина из которой мы идём;
- `to` – вершина в которую мы идём.

Параметры, передаваемые в `bool Bfs(char source, char sink)`:

- `source` – исток сети, вершина из которой исходят рёбра графа;
- `sink` – сток сети, вершина в которую входят рёбра графа.

Параметры, передаваемые в `void AddEdge(char from, char to, int capacity)`:

- `from` – начало пути;
- `to` – конец пути;
- `capacity` – пропускная способность текущего ребра.

Параметры, передаваемые в `int Ford_Fulkerson(char source, char sink)`:

- `source` – исток сети;
- `sink` – сток сети.

Так же в классе определены деструктор и конструктор по умолчанию.

В функции `main()` считывается количество дуг графа, затем задаются исток и сток, между которыми проходит поток. Далее в цикле начинается считывание рёбер графа и их пропускная способность. Запускается алгоритм Форда – Фалкерсона, который работает до тех пор, пока можно найти путь из истока в сток, используя функцию поиска в ширину `Bfs`. На каждом шаге итерации увеличиваются потоки в рёбрах данного маршрута на минимальную остаточную пропускную способность.

Рассмотрим пример работы программы. На рис. 1 показано в каком виде хранятся данные о графе.

Ключи	a		b		c		d		e		f	g		h
	b	c	d	e	b	e	f	g	d	g	h	f	h	
Значения	6	6	4	2	2	9	4	2	8	7	7	11	4	

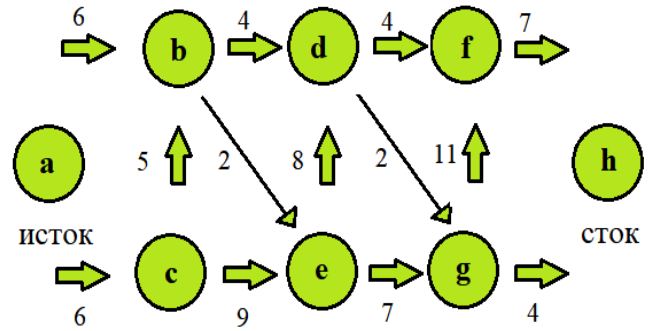

Рисунок 1 – представление графа списком смежности

Программа считывает значения, хранящиеся в текстовом файле, и заполняет поля класса. Сначала алгоритм находит пути из истока к стоку и увеличивает потоки в рёбрах данного маршрута на минимальную остаточную пропускную способность. Как представлено на рис. 2.

1 шаг. Путь: abdfh, min.пропускная способность = 4
2 шаг. Путь: abegh, min.пропускная способность = 2
3 шаг. Путь: acegh, min.пропускная способность = 2

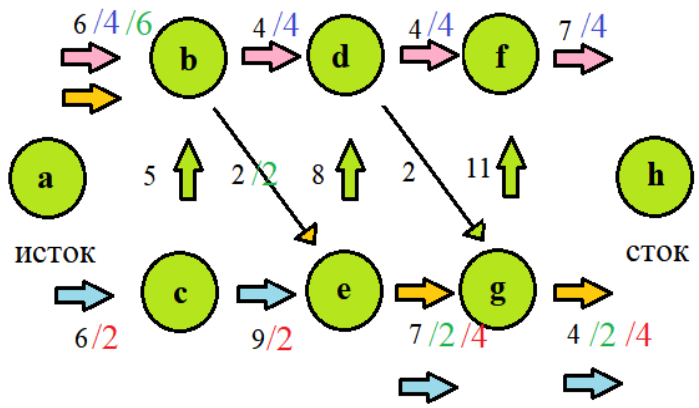

Рисунок 2 – первые три шага алгоритма.

По насыщенным рёбрам больше двигаться нельзя, найдём оставшиеся возможные пути. Следующие шаги алгоритма показаны на рис. 3, 4.

4 шаг. Путь: acegfh

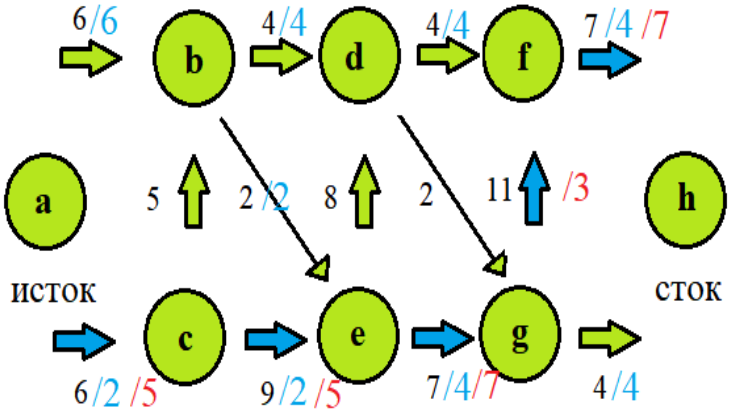

Рисунок 3 – четвёртый шаг алгоритма.

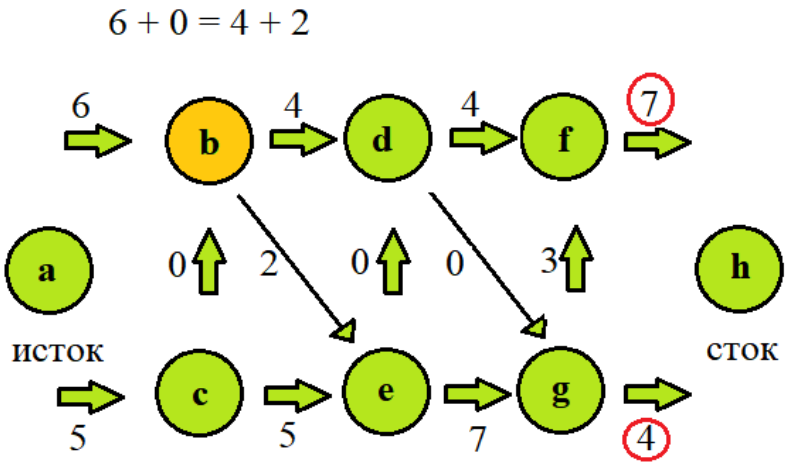

Рисунок 4 – пятый шаг алгоритма.

После того как алгоритм завершает работу, он выводит получившийся граф с фактическим потоком, проходящим через все рёбра. Чтобы проверить правильность решения можно посчитать входящий поток в вершину и выходящий, он должен остаться неизменным. Максимальная пропускная способность равна одиннадцати.

Исходный код программы представлен в приложении Б.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04, с использованием компилятора g++ версии 5.4.0 20160609. В других ОС и компиляторах тестирование не проводилось.

Программа может быть скомпилирована с помощью команды:

g++ <имя файла>.cpp

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно заметить, что тестовые случаи не выявили неправильного поведения программы, что говорит о том, что по результатам тестирование было показано, поставленная задача была выполнена.

Исследование

Поскольку на каждой итерации поток как минимум увеличивается на один, а поиск пути в графе происходит за O(|E|) операций, то сложность алгоритма составляет O(|F|E|), где F – максимальный поток в сети. Данная оценка требует знать величину максимального потока, но так как он не может превышать сумму пропускных способностей истока и сумму пропускных способностей стока, то можно заменить F на максимальную из этих двух сумм. Тогда O(|M|E|).

Выводы

В ходе лабораторной работы был изучен алгоритм поиска максимального потока в сети, используя алгоритм Форда – Фалкерсона. Был написан код на языке программирования C++, который применял этот метод для поставленной задачи. Сложность реализованного алгоритма составляет O(|M|E|).

ПРИЛОЖЕНИЕ А

Тестовые случаи

Ввод	Вывод	Верно?
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	Да
13 a h a b 6 a c 6 b d 4 b e 2 c b 2 c e 9 d f 4 d g 2 e d 8 e g 7 f h 7 g f 11 g h 4	11 a b 6 a c 5 b d 4 b e 2 c b 0 c e 5 d f 4 d g 0 e d 0 e g 7 f h 7 g f 3 g h 4	Да

ПРИЛОЖЕНИЕ Б

Код программы

```
#include <iostream>
#include <climits>
#include <cstring>
#include <queue>
#include <map>

#define N 'z'

using List = std:: map <char, std:: map <char, int>>;

class Graph {

private:
    char Parents[N];         //массив предков (родителей вершин)
    List Given_graph;        //список смежности заданного графа
    List RealFlow;           //список смежности фактического графа
   std:: array <bool, N> check;

    int ResidualCapacity(char from, char to) {
         //if(Given_graph[from][to] != 0 || Given_graph[from][to]!=0)
        //std:: cout << from << " -> " << to << ": " <<
Given_graph[from][to] << ' ' << RealFlow[from][to] << std:: endl;
        return (Given_graph[from][to] - RealFlow[from][to]);
    }

    bool Bfs(char source, char sink) {     //поиск в ширину
        check.fill (false);
        check[source] = true;
         Parents[source] = -1;              //у истока нет родителей
        std:: queue <char> queue;          //объявили очередь вершин
        queue.push(source);
        while (!queue.empty()) {
            char from = queue.front();
            for (char to = 0; to < N; to++) {
                if (!check[to] && ResidualCapacity(from, to) > 0) {
                    queue.push(to);
                    check[to] = true;
                    Parents[to] = from;
```

```
                    if (to == sink) {
                        queue.pop();
                        return (true);
                         }
                }
            }
        queue.pop();
        }
        return (false);
    }

public:
    Graph() {}

    void AddEdge(char from, char to, int capacity) {
        Given_graph[from][to] = capacity;
         RealFlow[to][from]= 0;
    }

    int Ford_Fulkerson(char source, char sink) {
        int maxFlow = 0;
        while (Bfs(source, sink)) {
            int minFlow = INT_MAX;
            char to = sink;
            for(int i = sink; 0 <= Parents[i]; i = Parents[i])
                minFlow = std:: min(minFlow,
ResidualCapacity(Parents[i], i));
              for(int i = sink; 0 <= Parents[i]; i = Parents[i]) {
                    RealFlow[Parents[i]][i] += minFlow;
                    RealFlow[i][Parents[i]] -= minFlow;
              }
        maxFlow += minFlow;
        memset(Parents, -1, N * sizeof(char));
        check.fill (false);
        }
        return (maxFlow);
    }

    void Print() {
        for (char i = 0; i < N; i++)
            for (char j = 0; j < N; j++)
                if (Given_graph[i][j])
```

```
                    std:: cout << char(i) << " " << char(j) << " "
<< std:: max(RealFlow[i][j], 0) << std:: endl;
    }

   ~Graph() {}
};

int main() {
    int count, capacity;
    char source, sink, from, to;
    std:: cin >> count >> source >> sink;
   Graph graph;
    for (int i = 0; i < count; i++) {
        std:: cin >> from >> to >> capacity;
        graph.AddEdge(from, to, capacity);
    }
    std:: cout << graph.Ford_Fulkerson(source, sink) << std::endl;
    graph.Print();
    return 0;
}
```