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: Алгоритм Форда-Фалкерсона

Студентка гр. 7383 _____ Прокопенко Н.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2019

Цель работы

Изучить и реализовать на языке программирования c++ алгоритм Форда-Фалкерсона, который позволяет найти максимальный поток в сети.

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 - 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). Вариант 2c: представить граф в виде списка смежности. Поиск пути через поиск в глубину.

Реализация задачи

В данной работе для решения поставленной цели был написан класс `Graf` и несколько методов, содержащихся в данном классе. А также написана структура.

Параметры, хранящиеся в структуре данных `struct edge`:

• `char beg` – вершина откуда;

• `char end` – вершина куда;

• `int heft` – пропускная способность (вес);

• `int forward`– потоком для рёбер из u в v.

• `int back`– потоком для рёбер из v в u;

• `bool doubl`– данные о двунаправленности ребра.

Конструктор класса создает массив структур и считывает данные в вектор `graph`.

Ниже представлены поля класса:

`vector <edge> graph` — вектор структур.

`vector <char> result` — вектор, хранящий путь.

`vector <char> viewingpoint` — вектор, хранящий просмотренные вершины.

`int maxFlow` — переменная, хранящая значения максимального потока в сети.

`int N` — количество ребер.

`char source` — исток.

`char estuary` — сток.

Далее представлены методы класса:

Метод `void FordFalk()` который идет по списку смежностей, используя следующий алгоритм:

Шаг 1: строится остаточная сеть, в которой изначально поток через каждое ребро равен 0, максимальный поток в сети равен 0.

Шаг 2: ищется путь от истока к стоку через рёбра, которые имеют не нулевой вес (разрешается переход от конца ребра к его началу с уменьшением потока через него), если путь не найден, то переход на шаг 4.

Шаг 3: в найденном пути ищется ребро с минимальным весом, величина этого ребра добавляется к максимальному потоку в графе, его величина вычитается из весов всех рёбер и прибавляется к величине потока, после чего переход на шаг 2.

Шаг 4: выход в `main()`.

Метод `bool Search(char value, int& min)` осуществляет поиск в глубину и поиск минимальную пропускную способность. Возвращает true если при поиске по не просмотренным вершинам достиг стока, false в противном случае.

Метод `bool isViewing(char value)` проверяет была ли вершина просмотрена на данном проходе.

Метод `void print()` выводит в консоль максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро.

В главной функции `main()` создается класс для графа и при инициализации происходит считывание данных. Далее вызывается метод поиска максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро, используя алгоритм Форда-Фалкерсона.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Так же было проведено исследование алгоритма. Для поиска очередного пути от истока к стоку использовался алгоритм поиска в глубину (DFS), сложность которого составляет O (|E| + |V|), где E – множество всех рёбер в графе, V – множество всех вершин. В результате общая сложность алгоритма Форда-Фалкерсона составляет O ((|E|*F), где F – максимальный поток в сети.

Выводы

В ходе выполнения данной лабораторной работы были изучен и реализован на языке программирования c++ алгоритм Форда-Фалкерсона, который ищет максимальный поток через сеть. Для поиска очередного пути от

истока к стоку использовался алгоритм поиска в глубину (DFS), сложность которого составляет O (|E| + |V|), где E – множество всех рёбер в графе, V – множество всех вершин. В результате общая сложность алгоритма Форда-Фалкерсона составляет O ((|E| + |V|)*F), где F – максимальный поток в сети.

Также, стоит отметить особенность данного алгоритма, которая заключается в том, что он гарантированно сходится только для целых пропускных способностей рёбер, в случае вещественных весов алгоритм может работать бесконечно долго, даже не сходясь к правильному результату.

ПРИЛОЖЕНИЕ А.
КОД ПРОГРАММЫ

lab3.cpp

```
#include <iostream>
```

```
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

struct edge{
    char beg;
    char end;
    int heft;
    int forward;
    int back;
    bool doubl;
};

bool compare(edge first, edge second){
    if(first.beg == second.beg)
        return first.end < second.end;
    return first.beg < second.beg;
}

class Graph{
private:
    vector <edge> graph;
    char source;
    char estuary;
    int N, maxFlow;
    vector <char> viewingpoint;
    vector <char> result;
public:
    Graph(): maxFlow(0){
        cin >> N >> source >> estuary;
        for(int i = 0; i < N; i++){
                      edge element;
            cin >> element.beg >> element.end >> element.heft;
            element.forward = element.heft;
            element.back = 0;
            element.doubl = false;
            bool flag = true;
            for(int i = 0; i < graph.size(); i++){
                if(graph[i].beg == element.end && graph[i].end ==
element.beg){
                    graph[i].back += element.forward;
                    flag = false;
                    graph[i].doubl = true;
                    break;
                }
```

```
        }
        if(!flag)
            continue;
        graph.push_back(element);
    }
}
bool isViewing(char value){
    for(size_t i = 0; i < viewingpoint.size(); i++)
        if(viewingpoint[i] == value)
            return true;
    return false;
}
bool Search(char value, int& min){
    if(value == estuary){
        result.push_back(value);
        return true;
    }
    viewingpoint.push_back(value);
    for(size_t i(0); i < graph.size(); i++){
        if(value == graph[i].beg){
            if(isViewing(graph[i].end) || graph[i].forward == 0)
                continue;
            result.push_back(graph[i].beg);
            bool flag = Search(graph[i].end, min);
            if(flag){
               if(graph[i].forward < min)
                   min = graph[i].forward;
               return true;
            }
            result.pop_back();
        }
        if(value == graph[i].end){
            if(isViewing(graph[i].beg) || graph[i].back == 0)
                continue;
            result.push_back(graph[i].end);
            bool flag = Search(graph[i].beg, min);
            if(flag){
               if(graph[i].back < min)
                    min = graph[i].back;
               return true;
            }
            result.pop_back();
        }
    }
    return false;
}
```

```
    void FordFalk(){
        int min = INT_MAX;
        while(Search(source, min)){
            for(int i = 1; i < result.size(); i++){
                for(int j = 0; j < graph.size(); j++){
                    if(graph[j].beg == result[i-1] && graph[j].end ==
result[i]){
                        graph[j].forward -= min;
                        graph[j].back += min;
                    }
                    if(graph[j].end == result[i-1] && graph[j].beg ==
result[i]){
                        graph[j].forward += min;
                        graph[j].back -= min;
                    }
                }
            }
            maxFlow += min;
            viewingpoint.clear();
            result.clear();
            min = INT_MAX;
        }
    }
    void print(){
        sort(graph.begin(), graph.end(), compare);
        cout << maxFlow << endl;
        for(int i = 0; i < graph.size(); i++){
            int value = max(graph[i].heft - graph[i].forward, 0 -
graph[i].back);
             if(graph[i].doubl == true){
                if(value < 0)
                    value = 0;
                cout << graph[i].beg << " " << graph[i].end << " " <<
value << endl;
                swap(graph[i].beg, graph[i].end);
                swap(graph[i].back, graph[i].forward);
                graph[i].doubl = false;
                sort(graph.begin(), graph.end(), compare);
                i--;
            }
            else{
                cout << graph[i].beg << " " << graph[i].end << " " <<
value << endl;
            }
        }
    }
```

```
};

int main(){
    Graph element;
    element.FordFalk();
    element.print();
    return 0;
}
```

ПРИЛОЖЕНИЕ Б.

ТЕСТОВЫЕ СЛУЧАИ

Результаты тестов представлены в табл. 1.

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
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 2 e d 2 e g 5 f h 7 g f 3 g h 4