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 алгоримов»

Тема: Потоки в сети

Студент гр. 8304 _____ Матросов Д.В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Реализовать алгоритм Форда-Фалкерсона, найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро.

Задание.

Найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро, используя алгоритм ФордаФалкерсона. 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 Wij – ребро графа Vi Vj Wij – ребро графа … Выходные данные: Pmax – величина максимального потока Vi Vj Wij – ребро графа с фактической величиной протекающего потока Vi Vj Wij – 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).

Ход выполнения работы:

1. Описание СД.

Для жадного алгорима были разработаны следующие СД:

- class Path – класс для хранения путей, также включает в себя иеформацию о локальном потоке.

2. Описание функций и методов:

Для жадного алгорима были разработаны следующие методы:

- void findMin – найти минимальное ребро на пути
- bool SortByAlphabet – сравнивает два потока по лексикографическому признаку
- bool SortByWidght – сравнивает два потока
- bool isVisitedPath – поверяет не вернулись ли мы к истоку

- bool findPath – обходит граф в ширину

Выводы.

В ходе выполнения лабораторной работы был реализован алгоритм Форда-Фалкерсона, который находит максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро.

ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ.

```
#include <iostream>
```

```
#include <vector>
#include <algorithm>

class Path {
public:
    Path(char nameFrom, char nameOut, int bandwidth) :
nameFrom(nameFrom), nameOut(nameOut), width(bandwidth) {}
    void setFlow(int flow_) {
        flow = flow_;
    }
    char GetSourceTop() const {
        return nameFrom;
    }
    char GetDrainTop() const {
        return nameOut;
    }
    int GetWidth() const {
        return width;
    }
    int GetFlow() const {
        return flow;
    }
private:
    char nameFrom;
    char nameOut;
    int width;
    int flow = 0;
};

void findMin(std::vector<Path*>* local, int* maxFlow) {
```

```
    int Min = local->front()->GetWidth();
    for (Path* path : *local) {
        if (Min > (path->GetWidth() - path->GetFlow())) {
            Min = path->GetWidth() - path->GetFlow();
        }
    }
    for (Path* path : *local) {
        path->setFlow(path->GetFlow() + Min);
    }
    *maxFlow = *maxFlow + Min;
}

bool SortByAlphabet(Path a, Path b) {
    if (a.GetSourceTop() != b.GetSourceTop())
        return a.GetSourceTop() < b.GetSourceTop();
    else
        return a.GetDrainTop() < b.GetDrainTop();
}

bool SortByWidght(Path* a, Path* b) {
    return (a->GetWidth() - a->GetFlow()) <= (b->GetWidth() - b->GetFlow());
}

bool isVisitedPath(std::vector<Path*>* local, char element) {
    for (Path* path : *local) {
        if (element == path->GetSourceTop()) {
            return false;
        }
```

```
    }
    return true;
}

bool findPath(std::vector<Path>* paths, std::vector<Path*>* local, std::vector<Path*>* local2, char myPoint, char* endPoint) {
    if (myPoint == *endPoint) {
        return true;
    }

    std::vector<Path*> localPaths;
    localPaths.reserve(0);
    for (auto& path : *paths) {
        if (path.GetSourceTop() == myPoint) {
            char sc = path.GetSourceTop();
            localPaths.emplace_back(&path);
        }
    }

    std::sort(localPaths.begin(), localPaths.end(), SortByWidght);

    for (Path* path : localPaths) {
        if (path->GetFlow() < path->GetWidth()) {
            if (isVisitedPath(local2, path->GetDrainTop())) {
                local2->emplace_back(path);
                if (findPath(paths, local, local2, path->GetDrainTop(), endPoint)) {
                    local->emplace_back(path);
```

```
                    return true;
                }
                else {
                    local2->pop_back();
                }
            }
        }
    }

    return false;

}

int main() {

    char startPoint, endPoint;
    char start, end;
    int count, weight;
    int maxFlow = 0;

    std::vector<Path*> local;
    std::vector<Path*> local2;
    std::vector<Path> paths;

    std::cin >> count;
    std::cin >> startPoint;
    std::cin >> endPoint;

    while (count != 0) {
        std::cin >> start >> end >> weight;
```

```
        paths.emplace_back(Path(start, end, weight));
        count--;
    }
    while (findPath(&paths, &local, &local2, startPoint, &endPoint)) {
        findMin(&local, &maxFlow);
        local.clear();
        local2.clear();
    }
    std::cout << maxFlow << '\n';

    std::sort(paths.begin(),                paths.end(), SortByAlphabet);

    for (Path path : paths) {
        std::cout  <<  path.GetSourceTop()  <<  "  "  << path.GetDrainTop() << " " << path.GetFlow() << "\n";
    }
    return 0;
}
```