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

Студент гр.8382 Гордиенко А.М.

Преподаватель Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Реализовать алгоритм поиска максимального потока в сети. Реализовать алгоритм Форда-Фалкерсона.

Постановка 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

Ui Vi Wi - ребро графа

Ui Vi Wi - ребро графа

...

Выходные данные:
Pmax - величина максимального потока
Ui Vi Wi - ребро графа с фактической величиной протекающего потока.

Ui Vi Wi - 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).

Индивидуализация.

Вариант 1. Поиск в ширину. Поочередная обработка вершин текущего фронта, перебор вершин в алфавитном порядке.

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

Пример выходных данных.

12

a b 6

a c 6

b d 6

c f 8

d e 2

d f 4

e c 2

Описание алгоритма.

В начале алгоритма подается граф для поиска максимального потока, вершина-исток и вершина-сток. Затем производится поиск в глубину.

На каждом шаге поиска с помощью очереди находиться путь от истока к стоку. Из ребер пути берется ребро с минимальной проводимостью. Из всех ребер пути от истока к стоку вычитается вес взятого ребра. Параллельно с этим к ребрам пути от стока к истоку вес взятого ребра прибавляется. При отсутствии такого ребра, оно достраивается. К величине максимального потока графа прибавляется тот же вес.

Цикл этих действий продолжается, пока поиск не станет невозможным, то есть нельзя будет пустить больший поток. Результатом будет конечное значение величины максимального потока. Фактический поток через ребро определяется разность между первоначальным ребром и его итоговым видом.

Сложность алгоритма.

По памяти.

O(|V| + |E|).

По времени.

O(|V| * |E|).

Описание функций и структур данных.

В качестве хранения графа была использована структура map<char, map<char, int>>. Является контейнером вершин и соответствующего контейнера расстояний между вершинами.

Функция поиска в ширину так и названа BFS(), принимает на вход ссылку на граф, вершины истока и стока, и пройденный путь. Функция возвращает истину, если поток можно провести, иначе - ложь.

Функция вывода текущего состояния потока в графе printCurrentFlows() принимает ссылку на граф, по найденному пути с величиной потока pathFlow.

Функция вывода результата работы алгоритма printResult(). По начальному графу и графу максимального потока выводятся пары вершин, по ребрам между которыми по ребрам идет фактический поток.

Функция, отвечающая за алгоритм Флойда-Фалкерсона, FFA(). Принимает ссылку на поток и вершина истока и стока.

Тестирование.

Входные данные.

8

a

g

a b 4

a c 5

a d 7

b c 3

c g 10

c f 8

d e 3

e g 9

Выходные данные.

Maximal flow = 11

a b 3

a c 5

a d 3

b c 3

c f 0

c g 8

d e 3

e g 3

Выводы.

В ходе лабораторной работы был изучен и реализован алгоритм Форда-Фалкерсона для поиска максимального потока в сети, а также нахождения фактической величины потока, протекающего через каждое ребро.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
#include <iostream>

#include <map>

#include <queue>

using std::cin;

using std::cout;

using std::endl;

using std::map;

using std::queue;

using std::string;

using Graph = map<char, map<char, int>>; // Хранение графа


bool BFS(Graph &graph, char start, char end, map<char, char>& path)
{ // Функция поиска в ширину

    cout <<
"############################################################" <<
endl;

    cout << "Wide search started" << endl;

    map<char,bool> findVertex; // Посещенные вершины

    queue<char> queueVertex; // Очередь вершин

    queueVertex.push(start);

    bool canReachAnother; // Флаг на то, имеет ли вершина
непосещенных соседей

    findVertex[start] = true;

    while (!queueVertex.empty()) // Пока очередь не пустая

    {

        char vertex = queueVertex.front();

        queueVertex.pop();

        cout << "Current vertex " << vertex << " has paths to: " <<
endl;

        canReachAnother = false;

        for (auto const neighbour : graph[vertex]){
//Просматриваются соседи и кладутся в очередь
```

```
            if (neighbour.second > 0 &&
!(findVertex[neighbour.first])) {
                queueVertex.push(neighbour.first);
                findVertex[neighbour.first] = true;
                path[neighbour.first] = vertex;
                cout << "\t->" << neighbour.first << " with possible
flow =  " << neighbour.second << endl;
                canReachAnother = true;
                if (neighbour.first == end) {
                    queueVertex = *(new queue<char>());
                    break;
                }
            }
        }
        if (!canReachAnother){
            cout << "\t>>haven't visited neighbour";
        }
        cout << endl;
    }
    return findVertex[end]; // Была ли достигнута финишная вершина
графа
}


void printCurrentFlows(Graph& flowGraph, int pathFlow, int
maxCurrentFlow, string& pathStr){  //Функция печати текущего
состояния графа, найденного нового пути и найденного нового потока
    cout << "\nNew flow path found = " << pathFlow << ": " + pathStr
<< endl;
    cout << "Flow graph: " << endl;
    for (auto const& vertex: flowGraph) {
        for (auto const neighbour: flowGraph[vertex.first]) {
            cout << "\t" << vertex.first << " " << neighbour.first
<< " " << neighbour.second << endl;
        }
```

```
    }
    cout << "Current flow - " << maxCurrentFlow << endl;
}
void printResult(Graph& graph, Graph& flowGraph, int maxFlow){
//Функция печати результата
    cout <<
"############################################################" <<
endl;
    cout << "\nResult of algorithm: " <<endl;
    int flow;
    cout << "Maximal flow = "<< maxFlow << endl;
//Печать максимального потока
    for (auto const& vertex: graph)
        for (auto const neighbour: graph[vertex.first]) {
            if (neighbour.second -
flowGraph[vertex.first][neighbour.first] < 0)
                flow = 0;
            else
                flow = neighbour.second -
flowGraph[vertex.first][neighbour.first];
            cout << vertex.first << " " << neighbour.first << " " <<
flow << endl;   // Печать потока через ребро
        }
}
void FFA(Graph &graph, char start, char finish) { // Функция,
реализующая алгоритм Форда-Фалкерсона
    Graph flowGraph = graph; // Граф с потоками
    char fromVertex, toVertex;
    map<char,char> path; // Пары, составляющие путь
    string pathStr;
    int maxFlow = 0;
    while (BFS(flowGraph, start, finish, path)) // Пока возможен
поиск в ширину
    {
```

```
        int pathFlow = INT_MAX;
        pathStr = finish;
        for (toVertex = finish; toVertex != start; toVertex =
path[toVertex])           //Восстанавливается путь от финиша к
началу
        {
            fromVertex = path[toVertex];
            pathFlow = std::min(pathFlow,
flowGraph[fromVertex][toVertex]);           //Находится поток пути
        }
        for (toVertex = finish; toVertex != start; toVertex =
path[toVertex])           //Восстанавливается путь от финиша к
началу
        {
            fromVertex = path[toVertex];
            flowGraph[fromVertex][toVertex] -= pathFlow;
//Изменяется граф с потоком
            flowGraph[toVertex][fromVertex] += pathFlow;
            pathStr = string(1, fromVertex) + " --> " + pathStr;
        }
        maxFlow += pathFlow;
//Изменяется число максимального потока
        printCurrentFlows(flowGraph, pathFlow, maxFlow, pathStr);
    }
    printResult(graph, flowGraph, maxFlow);
}
int main() {
    Graph graph;
    char start, finish, vertexFrom, vertexTo;
    int pathLength, countVertex;
    cout << "Your input" << endl;
    cin >> countVertex;
    cin >> start;
    cin >> finish;
```

```
    for (int i=0; i < countVertex; i++) {
//Считывание вершин графа
        cin >> vertexFrom;
        cin >> vertexTo;
        cin >> pathLength;
        graph[vertexFrom][vertexTo] = pathLength;
    }
    FFA(graph, start, finish); // Запуск алгоритма Форда-Фалкерсона
    return 0;
}
```