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

Вариант 5

Студент гр. 8303 _____ Дирксен А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучение алгоритмов поиска максимального потока в графе.

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
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).

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
```

Индивидуализация.

Поиск не в глубину и не в ширину, а по правилу: каждый раз выполняется переход по дуге, имеющей максимальную остаточную пропускную способность. Если таких дуг несколько, то выбрать ту, которая была обнаружена раньше в текущем поиске пути.

Описание алгоритма.

В программе используется алгоритм Форда-Фалкерсона. Алгоритм подразумевает запуск поиска в глубину до тех пор, пока возможно найти путь от истока к стоку.

На каждом шаге находится путь от истока к стоку, при этом смежные вершины выбираются в порядке уменьшения остаточной пропускной способности C. В пути выбирается ребро с наименьшей C (далее Cmin). Для каждого ребра пути C уменьшается на Cmin, строится обратное ребро и его пропускной способности прибавляется Cmin.

Время работы алгоритма ограничено O(V*f*E*logE), где E — число рёбер в графе, V — число вершин, f — максимальный поток в графе, так как для каждой вершины сортируются смежные за O(E*logE), в таком случае каждый увеличивающий путь в худшем случае находится за O(V*E*logE) и увеличивает поток как минимум на 1.

Для работы алгоритма хранится граф в виде матрицы смежности ($O(V^2)$), остаточная сеть (также $O(V^2)$) и вектор посещенных вершин (O(V)). в итоге получаем сложность по памяти $O(V^2)$.

Описание структур данных.

struct Edge {
   int resC;
   int revF; } - структура для хранения ребер.
resC — остаточная пропускная способность ребра
revF — обратный поток

map<char, map<char, Edge>> net – остаточная сеть графа.
Используется для хранения информации о графе в виде матрицы смежности. Для каждой вершины хранится карта «смежная вершина-ребро».

vector <bool> visited — контейнер для того, чтобы отмечать посещенные вершины в поиске в глубину.

set<pair<char, char>> graph — контейнер для хранения списка смежности графа. Используется для упрощения сортировки выходных данных.

set<pair<int, char>> toVisit — контейнер для сортировки смежных вершин по остаточной пропускной способности.

priority_queue<pair<char, double>, vector<pair<char, double>>, decltype(&compare)> priorityQueue(&compare) — очередь с приоритетом. Хранит пары вершины — кратчайший путь (включая эвристику). Добавлен отдельный компаратор, который ставит в приоритет вершины с минимальным значением кратчайшего пути.

map<char, pair<string, double>> ways — структура для хранения минимальных путей до каждой вершины.

Описание функций.

void readGraph()

Функция чтения графа. Заполняет контейнеры graph и net.

int dfs (char v, int delta)

v — вершина с которой начинаем поиск.

delta — текущая минимальная остаточная пропускная способность.

Рекурсивная функция поиска в глубину в графе. Возвращает значение, на которое увеличился поток.

void print()

Функция вывода текущего потока в консоль.

void findFordFulkerson()

Функция поиска максимального потока в графе.

Тестирование.

```
11
a
f
a b 7
a c 3
a d 5
c b 4
c f 5
b f 6
b d 3
b e 4
d b 7
d e 8
e f 10
15
a b 7
a c 3
a d 5
b d 0
b e 1
b f 6
c b 0
c f 3
d b 0
d e 5
e f 6
```

```
11
a
h
a b 3
b e 1
a c 1
c e 2
a d 2
d e 4
e g 3
e f 2
f h 3
g h 1
d f 1
4
a b 1
a c 1
a d 2
b e 1
c e 1
d e 1
d f 1
e f 2
e g 1
f h 3
g h 1
```

```
4
a
d
a c 1
a b 1
c b 1
b c 1
0
a b 0
a c 0
b c 0
c b 0
```

```
7
a
f
a b 7
b d 6
d e 3
e c 2
a c 6
c f 9
d f 4
12
a b 6
a c 6
b d 6
c f 8
d e 2
d f 4
e c 2
```

Вывод.

В ходе выполнения лабораторной работы были изучен и запрограммирован алгоритм Форда-Фалкерсона для поиска максимального потока в графе.

Приложения А. Исходный код

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <climits>
#include <set>
#include <map>
#define DBG
using namespace std;

//структура ребра, хранит остаточную пропускную способность и поток,
который можно пустить обратно
struct Edge {
    int resC;
    int revF;

    Edge() : resC(0), revF(0) {}
    Edge(int c, int f) : resC(c), revF(f) {}
};

//остаточная сеть
map<char, map<char, Edge>> net;
//сет ребер
set<pair<char, char>> graph;
//вектор посещенных вершин
vector<bool> visited;
char source, sink;

//функция чтение графа из консоли. Заполняет граф и остаточную сеть
void readGraph() {
    int n;
    char u, v;
    int c;

    cin >> n;
    cin >> source >> sink;

    visited.resize(128);

    for (size_t _ = 0; _ < n; _++) {
        cin >> u >> v >> c;
        graph.insert({u, v});
        net[u][v].resC = c;
        if (net.find(v) != net.end() && net[v].find(u) == net[v].end()){
            net[v][u].resC = 0;
        }
    }
}

//рекурсивный поиск в глубину
int dfs(char v, int delta) {
    //если вершина уже была посещена, выходим из нее
    if (visited[v])
        return 0;
    visited[v] = true;
```

```
    //если текущая вершина - сток, выходим из нее
    if (v == sink)
        return delta;

    //множество смежных вершин, сортированное по остаточной пропускной
способности
    set<pair<int, char>> toVisit;

    for (auto u : net[v]) {
        if (!visited[u.first])
            toVisit.insert({max(u.second.resC, u.second.revF),
u.first});
    }
    //обходим вершина из множества в порядке убывания остаточной
пропускной способности

    for (auto u = toVisit.rbegin(); u != toVisit.rend(); u++) {
        //если есть поток который можно пустить обратно,
        //находим минимальный вес ребра в пути и делаем это
        if (net[v][u->second].revF > 0) {
            int newDelta = dfs(u->second, min(delta, net[v][u-
>second].revF));
            if (newDelta > 0) {
                net[u->second][v].resC += newDelta;
                net[v][u->second].revF -= newDelta;
                return newDelta;
            }
        }
        //если остаточная пропускная способность больше нуля,
        //находим минимальный вес ребра в пути и пускаем поток по этому
ребру
        if (net[v][u->second].resC > 0) {
            int newDelta = dfs(u->second, min(delta, net[v][u-
>second].resC));
            if (newDelta > 0) {
                net[u->second][v].revF += newDelta;
                net[v][u->second].resC -= newDelta;
                return newDelta;
            }
        }
    }
    return 0;
}

//функция вывода найденного потока
void print() {
    for (auto &i : graph) {
        cout << i.first << ' ' << i.second << ' ' << net[i.second]
[i.first].revF;
#ifdef DBG
        cout << "(residual capacity " << net[i.first][i.second].resC
        << ", max capacity " << net[i.second][i.first].revF +
net[i.first][i.second].resC << ')';
#endif
        cout << endl;
```

```
    }
}

//запуск поиска алгоритмом Форда-Фалкерсона
void findFordFulkerson() {
    int flow = 0;
    int ans = 0;
    while (true) {
        //обнуляем вектор посещенных вершин
        fill(visited.begin(), visited.end(), false);
        //запускаем поиск в глубину
        flow = dfs(source, INT_MAX);
        //если путь не найден - выходим
#ifdef DBG
        cout << "==================\n";
        cout << "current flow = " << ans+flow << endl;
        print();
#endif
        if (flow == 0 || flow == INT_MAX){
#ifdef DBG
            cout << "<!New path not found!>\n";
#endif
            break;
        }
        //обновляем максимальный поток
        ans += flow;
    }
    cout << ans << endl;
    print();
}

int main() {
    readGraph();
    findFordFulkerson();
    return 0;
}

/*
11
a
f
a b 7
a c 3
a d 5
c b 4
c f 5
b f 6
b d 3
b e 4
d b 7
d e 8
e f 10

 */
```