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. 8303 _____ Данилов А.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучение алгоритма Форда-Фалкерсона для нахождение 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

v_0 - исток
v_n - сток
v_i v_j ω_{ij} - ребро графа
v_i v_j ω_{ij} - ребро графа
...

Выходные данные:
P_{max} - величина максимального потока
v_i v_j ω_{ij} - ребро графа с фактической величиной протекающего потока
v_i v_j ω_{ij} - 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.

Вар. 5. 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 Форда-Фалкерсона.

Остаточная сеть — это граф $G_f = (V, E_f)$, где E_f - множество ребер с положительной остаточной пропускной способностью. В остаточной сети может быть путь из *u* в *v,* даже если его нет в исходном графе. Это выполняется, когда в исходной сети есть обратный путь *(v, u)* и поток по нему положительный.

Дополняющий путь — это путь в остаточной сети.

Идея алгоритма заключается в том, чтобы запускать поиск в глубину *в остаточной сети* до тех пор, пока поиск в глубину находит путь от истока к стоку.

Вначале алгоритма *остаточная сеть* — это исходный граф. Алгоритм ищет *дополняющий путь* в *остаточной сети,* если путь был найден, то *остаточная сеть* перестраивается, а к максимальному потоку прибавляется величина максимальной пропускной способности *дополняющего пути,* если путь от истока к стоку найден не был, то значит максимальный поток был найден и алгоритм завершает свою работу. Максимальный поток в сети является суммой всех максимальных пропускных способностей *дополняющих путей*.

Описание функций и структур данных.

Структуры данных.

- struct Rib - структура для хранения остаточной пропускной способности основного и вспомогательного ребра.
- map<char, map<char, Rib>> resNet - матрица смежности для хранения остаточной сети.
- set<pair<char, char>> graph - контейнер для хранения списка смежности графа. Используется для упрощения сортировки входных данных
- vector<bool> visited - вектор, в котором отмечаются уже посещенные вершины.
- int source, sink - сток и исток.
- set<pair<int, char>> toVisit - контенйнер для сортировки смежных вершин по остаточной пропускной способности

Методы.

- int dfs(char v, int Cmin) - рекурсивный поиск в глубину с модификацией: каждый раз выполняется переход по дуге, имеющей максимальную остаточную пропускную способность. Функция принимает на вход вершину, из которой ищется путь и текущая минимальная пропускная способность на пути. Возвращает значение, на которое увеличился поток.
- void print() - выводит ребра исходного графа с фактической величиной протекающего потока
- void readGraph() - считывает граф и создает начальную остаточную сеть.
- void maxFlow() - метод, который находит максимальный поток в графе.

Сложность алгоритма.

E – множество ребер графа.

V – множество вершин графа.

F – величина максимальной пропускной способности графа.

По времени.

На каждом шаге мы ищем путь от стока к истоку, поиском в глубину с модификацией: каждый раз выполняется переход по дуге, имеющей максимальную остаточную пропускную способность.

Так как просматривать ребра нужно в порядке уменьшения пропускной способности, для этого в каждой новой вершине все ребра сортируются, на это приходится тратить E * log(E) операций. В остальном это поиск в глубину, поэтому поиск нового дополняющего пути в сети происходит за $O(E * log E * V)$.

В худшем случае, на каждом шаге мы будем находить дополняющий путь с пропускной способностью 1, тогда получим сложность по времени $O(F * E * logE * V)$.

По памяти.

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 $O(V^2)$.

Тестирование.

Input	Output
7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	acf current flow is: 6 A new path increased flow by: 6 abdecf current flow is: 8 A new path increased flow by: 2 abdef current flow is: 12 A new path increased flow by: 4 ab current flow is: 12 New path not found MAX flow is: 12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2
10 a f a b 16 a c 13 c b 4 b c 10 b d 12 c e 14 d c 9 d f 20 e d 7 e f 4	acbdf current flow is: 4 A new path increased flow by: 4 acef current flow is: 8 A new path increased flow by: 4 acedbf current flow is: 13 A new path increased flow by: 5 abdeccf current flow is: 21 A new path increased flow by: 8 abcedf

	current flow is: 23 A new path increased flow by: 2 abcec current flow is: 23 New path not found MAX flow is: 23 a b 10 a c 13 b c 0 b d 12 c b 2 c e 11 d c 0 d f 19 e d 7 e f 4

4 a d a c 1 a b 1 c b 1 b c 1	abcc current flow is: 0 New path not found MAX flow is: 0 a b 0 a c 0 b c 0 c b 0
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	acefh current flow is: 1 A new path increased flow by: 1 adfecgh current flow is: 2 A new path increased flow by: 1 adecfh current flow is: 3 A new path increased flow by: 1 abecdfh current flow is: 4 A new path increased flow by: 1 ab current flow is: 4 New path not found MAX flow is: 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

Вывод.

В ходе лабораторной работы был изучен алгоритм поиска максимального потока в сети - метод Форда-Фалкерсона.

Приложение А.

```
#include <iostream>
#include <algorithm>
#include <map>
#include <set>
#include <vector>
#define PRINT


using namespace std;


// ребро, содержит информацию об остаточной пропускной спобоности и потоке, который можно пустить назад
struct Rib
{
    int C;  // остаточная пропускная способность, изначальноравна пропускной
    int F;  // обратный поток
    Rib() : C(0), F(0) {}
    Rib(int C, int F) : C(C), F(F) {}
};
class FordFulkerson
{
private:
```

```
    map<char, map<char, Rib>> resNet; //остаточная сеть
    set<pair<char, char>> graph; // исходный граф
    vector<bool> visited; //  посещенные вершины
    char source = 0, sink = 0;      // исток и сток

    int dfs(char v, int Cmin);
    void printRes();

public:
    void maxFlow();;
    void input();

};



void FordFulkerson::maxFlow() {
    int flow = 0;
    int answer = 0;
    while(true)
    {
        //обнуляем все посещения
        fill(visited.begin(), visited.end(), false);
        flow = dfs(source, INT32_MAX); // пускаем максимальный поток из
истока

#ifdef PRINT
        cout << "\ncurrent flow is: " << answer+flow<<endl;
//        printRes();
#endif

        // если путь не найден
        if (flow == 0 || flow == INT32_MAX)
        {
#ifdef PRINT
            cout << "New path not found\n";
#endif
```

```
            break;
        } else
#ifdef PRINT
                  cout << "A new path increased flow by: " << flow <<"\n";
#endif

        answer += flow;
    }
    cout <<"MAX flow is:  "<< answer << endl;
    printRes();
}

int FordFulkerson::dfs(char v, int Cmin) // Cmin - минимальная пропускная способность
{
#ifdef PRINT
      cout << char(v + 'a');
#endif
    //если вершина была посещены-> выходим
    if (visited[v])
        return 0;
    visited[v] = true;

    //если веришна - сток -> выходим
    if (v == sink)
        return Cmin;

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

    for (auto u : resNet[v])
        if (!visited[u.first])
            toVisit.insert({max(u.second.C, u.second.F), u.first});

    //обходим вершины из множества в порядке убывания остаточной пропускной способности
```

```
    for (auto u : toVisit)
    {
        // если есть поток, который можем пустить обратно, делаем это с ребром мин веса
        if (resNet[v][u.second].F > 0) {
            auto delta = dfs(u.second, min(Cmin, resNet[v][u.second].F));
            if (delta > 0)
            {
                resNet[u.second][v].C += delta;
                resNet[v][u.second].F -= delta;
                return delta;
            }
        }
        // если остаточная пропускная способнсть положительна,
        // находим ребро с наименьшим весом и пускаем поток по нему
        if (resNet[v][u.second].C > 0)
        {
            auto delta = dfs(u.second, min(Cmin, resNet[v][u.second].C));
            if (delta > 0){
                resNet[u.second][v].F += delta;
                resNet[v][u.second].C -= delta;
                return delta;
            }
        }
    }
    return 0;
}


void FordFulkerson::printRes() {
    for (auto &i : graph)
    {
        cout << char(i.first + 'a') << ' ' << char(i.second + 'a') <<' '<< resNet[i.second][i.first].F;
        cout << endl;
    }
}
```

```
void FordFulkerson::input() {
    int N;
    int capacity;
    char u, v;

    cin >> N;
    cin >> source >> sink;
    source -= 'a';
    sink -= 'a';

    visited.resize(26);

    for (int i = 0; i < N; i++)
    {
        cin >> u >> v >> capacity;
        u -= 'a';
        v -= 'a';
        graph.insert({u,v});
        resNet[u][v].C = capacity;
    }
}



int main() {
    FordFulkerson F;
    F.input();
    cout << endl;
    F.maxFlow();
    return 0;
}
```