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: Максимальный поток

Студентка гр. 9383 ____________________ Чебесова И. Д.

Преподаватель ____________________ Фирсов М. А.

Санкт-Петербург

2021

Цель работы.

Познакомиться с алгоритмом Форда-Фалкерсона, реализовать алгоритм на одном из языков программирования.

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

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 W_{ij} – ребро графа

V_i V_j W_{ij} – ребро графа

…

Выходные данные:

P_{max} – величина максимального потока

V_i V_j W_{ij} – ребро графа с фактической величиной протекающего потока

V_i V_j W_{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).

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

Основные теоретические положения.

Чтобы говорить об алгоритме необходимо ввести ряд понятий:

Сеть – это такой ориентированный взвешенный граф, что имеет один исток и один сток.

Исток – вершина, из которой ребра выходят, но не входят.

Сток – вершина, в которую ребра входят, но не выходят.

Поток – это понятие, описывающее движение по графу.

Величина потока – числовая характеристика потока.

Пропускная способность ребра – свойство ребра, которое показывает, какая максимальная величина потока может пройти через ребро графа.

Максимальный поток – такая максимальная величина, которая может пройти из истока по всем ребрам графа, не вызывая переполнения ни в одной пропускной способности ребра.

Фактическая величина потока в ребре – значение, которое показывает сколько величины потока проходит через ребро.

Алгоритм Форда-Фалкерсона – алгоритм, который служит для нахождения максимального потока в сети.

Описание алгоритма.

В начале работы алгоритму на вход подается граф для поиска максимального потока, вершина-исток и вершина-сток графа.

После считывания входных данных начинает работу сам алгоритм по следующим принципам:

1. Запускается поиск пути в графе. В виду специфики задания, поиск осуществляется с помощью обхода в ширину.

2. Если путь найден, то происходит вычисление максимального потока – это будет величина минимального ребра этого пути.

3. Для всех ребер найденного пути поток увеличивается на найденную в пункте 2 величину, а пропускная способность на эту величину уменьшается.

4. Полученное значение максимального потока в найденном пути в пункте 2 прибавляется к значению максимального потока всего графа, после чего запускается новый поиск в ширину.

5. Алгоритм осуществляет свою работу пока существует какой-либо пусть из вершины-истока к вершине-стоку.

Для удобства отслеживания процесса работы алгоритма в консоль выводятся промежуточные результаты.

Сложность алгоритма по операциям: O (E*F), E – число ребер в графе, F – максимальный поток

Сложность алгоритма по памяти: O (E), где E – количество ребер

Описание функций и структур данных.

using Graph = std::map<char, std::map<char, int>>;

Структура данных, используемая для хранения направленного ориентированного графа. Представляет собой ассоциативный контейнер хранения вершин и соответствующего ей контейнера вершина-расстояние. Для каждой вершины хранится ассоциативный массив вершин, до которых можно добраться из текущей и вес пути до них.

Например, граф вида:

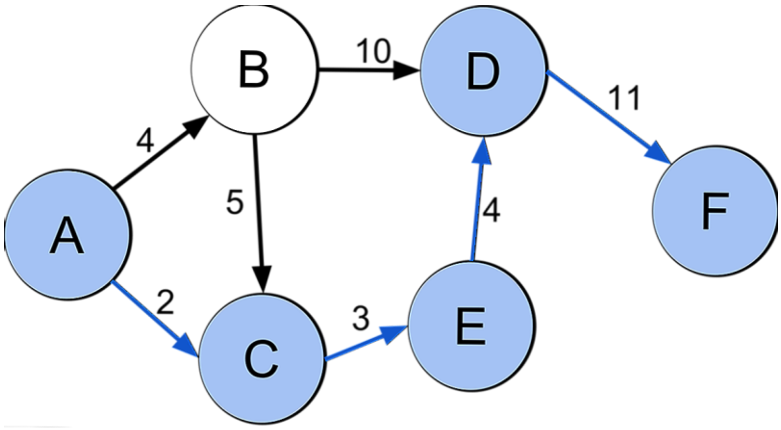

Рисунок 1 – Пример хранения графа

Визуально будет хранить в следующем виде:

A: B 4

C 2

B: C 5

D 10

C: E 3

D: F 11

E: D 4

bool BFS(Graph &graph, char start, char end, std::map<char, char>& path)

Функция осуществляющая поиск в ширину. На вход принимает ссылку на граф *graph*, в котором будет осуществляться поиск, исток и сток вершины *start* и *end* соответственно, ассоциативный массив пар *path*, из которого будет получен путь.

Функция возвращает *true*, если при поиске была достигнута финальная вершина, *false* – противном случае.

void printCurrentFlows(Graph& flowGraph, int pathFlow, int maxCurrentFlow, std::string& pathStr)

Функция печати текущего состояния графа *flowGraph*, найденного пути *pathStr* с потоком через него размером *pathFlow*, текущего суммарного потока *maxCurrentFlow*.

void printResult(Graph& graph, Graph& flowGraph, int maxFlow)

Функция печати результата работы алгоритма. С помощью начального графа *graph* и графа *graphFlow*, полученного в результате работы алгоритма, печатаются пары вершин с фактической величиной потока через ребра между ними. Также печатается максимальный поток в сети *maxFlow*.

void FF(Graph &graph, char start, char finish)

Функция, осуществляющая алгоритм Форда-Фалкерсона нахождения максимального потока в сети. На вход принимает граф *graph*, в котором будет находиться максимальный поток, исток *start* и сток *finish*.

Тестирование.

Входные 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

Результат работы программы:

Начинаем обход в ширину
Текущая вершина a и ее потомки: b с потоком = 7
c с потоком = 6

Текущая вершина b и ее потомки: d с потоком = 6

Текущая вершина c и ее потомки: f с потоком = 9

Текущая вершина d и ее потомки: e с потоком = 3

Текущая вершина f и ее потомки: нет не посещенных потомков
Текущая вершина e и ее потомки: нет не посещенных потомков

Найден новый путь с потоком = 6: a --> c --> f
Текущее состояние графа:

a b 7
a c 0
b d 6
c a 6
c f 3
d e 3
d f 4
e c 2

f c 6
Текущий максимальный поток графа = 6

Начинаем обход в ширину
Текущая вершина a и ее потомки:
b с потоком = 7

Текущая вершина b и ее потомки:
d с потоком = 6

Текущая вершина d и ее потомки:
e с потоком = 3
f с потоком = 4

Текущая вершина e и ее потомки:
c с потоком = 2

Текущая вершина f и ее потомки:
нет не посещенных потомков

Текущая вершина c и ее потомки:
нет не посещенных потомков

Найден новый путь с потоком = 4: a --> b --> d --> f
Текущее состояние графа:
a b 3
a c 0
b a 4
b d 2
c a 6
c f 3
d b 4
d e 3
d f 0
e c 2
f c 6
f d 4
Текущий максимальный поток графа = 10

Начинаем обход в ширину
Текущая вершина a и ее потомки:
b с потоком = 3

Текущая вершина b и ее потомки:
d с потоком = 2

Текущая вершина d и ее потомки:
e с потоком = 3

Текущая вершина e и ее потомки:
c с потоком = 2

Текущая вершина c и ее потомки:
f с потоком = 3

Текущая вершина f и ее потомки:
нет не посещенных потомков

Найден новый путь с потоком = 2: a --> b --> d --> e --> c --> f
Текущее состояние графа:
a b 1
a c 0
b a 6
b d 0
c a 6
c e 2
c f 1
d b 6
d e 1
d f 0
e c 0
e d 2
f c 8
f d 4
Текущий максимальный поток графа = 12

Начинаем обход в ширину
Текущая вершина a и ее потомки:

b с потоком = 1

Текущая вершина b и ее потомки:
нет не посещенных потомков

Результат работы алгоритма:
Значение максимального потока графа: 12
a b 6
a c 6
b d 6
c f 8
d e 2
d f 4
e c 2

Входные данные:

8
a
h
a b 5
a c 4
a d 1
b g 1
c e 2
c f 3
d e 6
e h 4
f h 4
g h 8

Результат работы программы:

Начинаем обход в ширину
Текущая вершина a и ее потомки:
b с потоком = 5
c с потоком = 4
d с потоком = 1

Текущая вершина b и ее потомки:
g с потоком = 1

Текущая вершина c и ее потомки:
e с потоком = 2
f с потоком = 3

Текущая вершина d и ее потомки:
нет не посещенных потомков

Текущая вершина g и ее потомки:
нет не посещенных потомков

Текущая вершина e и ее потомки:
h с потоком = 4

Текущая вершина f и ее потомки:
нет не посещенных потомков

Текущая вершина h и ее потомки:
нет не посещенных потомков

Найден новый путь с потоком = 2: a --> c --> e --> h
Текущее состояние графа:

a b 5
a c 2
a d 1
b g 1
c a 2
c e 0
c f 3
d e 6
e c 2
e h 2
h e 2

Текущий максимальный поток графа = 2

Начинаем обход в ширину
Текущая вершина a и ее потомки:
b с потоком = 5
c с потоком = 2
d с потоком = 1

Текущая вершина b и ее потомки:
g с потоком = 1

Текущая вершина c и ее потомки:
f с потоком = 3

Текущая вершина d и ее потомки:
e с потоком = 6

Текущая вершина g и ее потомки:
нет не посещенных потомков

Текущая вершина f и ее потомки:
нет не посещенных потомков

Текущая вершина e и ее потомки:
h с потоком = 2

Текущая вершина h и ее потомки:
нет не посещенных потомков

Найден новый путь с потоком = 1: a --> d --> e --> h
Текущее состояние графа:

    a b 5
    a c 2
    a d 0
    b g 1
    c a 2
    c e 0
    c f 3
    d a 1
    d e 5

e c 2
e d 1
e h 1
h e 3
Текущий максимальный поток графа = 3

Начинаем обход в ширину
Текущая вершина a и ее потомки:
b с потоком = 5
c с потоком = 2

Текущая вершина b и ее потомки:
g с потоком = 1

Текущая вершина c и ее потомки:
f с потоком = 3

Текущая вершина g и ее потомки:
нет не посещенных потомков

Текущая вершина f и ее потомки:
нет не посещенных потомков

Результат работы алгоритма:
Значение максимального потока графа: 3
a b 0
a c 2
a d 1
b g 0
c e 2
c f 0
d e 1
e h 3

Выводы.

В ходе выполнения лабораторной работы был изучен и реализован алгоритм Форда-Фалкерсона, который находит максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро. Также был реализован алгоритм обхода в ширину в качестве индивидуализации данной лабораторной работы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include <iostream>
#include <map>
#include <climits>
#include <queue>

using Graph = std::map<char, std::map<char, int>>;

bool BFS (Graph& graph, char start, char end, std::map<char,
char>& path)
{
    std::cout << "--------------------------------------------------
--------\n";
    std::cout << "Начинаем обход в ширину\n";

    std::map<char,bool> visited;
    visited[start] = true;

    std::queue<char> queueVertex;
    queueVertex.push(start);

    while (!queueVertex.empty())
    {
        char vertex = queueVertex.front();
        queueVertex.pop();
        std::cout << "Текущая вершина " << vertex << " и ее
потомки: \n";
        bool hasNeighbor = false;

        for (auto const neighbor : graph[vertex])
        {
            if (neighbor.second > 0 &&
!(visited[neighbor.first]))
```

```
                {
                    queueVertex.push(neighbor.first);
                    visited[neighbor.first] = true;
                    path[neighbor.first] = vertex;
                    std::cout << neighbor.first << " с потоком = " <<
neighbor.second << "\n";
                    hasNeighbor = true;
                }
            }
            if (!hasNeighbor)
            {
                std::cout << "нет не посещенных потомков\n";
            }
            std::cout << "\n";
        }
        return visited[end];
    }


    void printCurrentFlows(Graph& flowGraph, int pathFlow, int
maxCurrentFlow, std::string& pathStr)
    {
        std::cout << "\nНайден новый путь с потоком = " << pathFlow
<< ": " + pathStr << "\n";
        std::cout << "Текущее состояние графа:\n";
        for (auto const& vertex: flowGraph)
        {
            for (auto const neighbor: flowGraph[vertex.first])
            {
                std::cout << "\t" << vertex.first << " " <<
neighbor.first << " " << neighbor.second << "\n";
            }
        }
        std::cout << "Текущий максимальный поток графа = " <<
maxCurrentFlow << "\n";
    }
```

```
void printResult(Graph& graph, Graph& flowGraph, int maxFlow)
{
    std::cout << "--------------------------------------------------------
------\n";
    std::cout << "Результат работы алгоритма:\n";
    std::cout << "Значение максимального потока графа: " <<
maxFlow << "\n";

    int flow = 0;
    for (auto const& vertex: graph)
        for (auto const neighbor : graph[vertex.first])
        {
            if (neighbor.second -
flowGraph[vertex.first][neighbor.first] < 0)
            {
                flow = 0;
            }
            else
            {
                flow = neighbor.second -
flowGraph[vertex.first][neighbor.first];
            }
            std::cout << vertex.first << " " << neighbor.first <<
" " << flow << "\n";
        }
}


void FF(Graph& graph, char start, char finish)
{
    Graph flowGraph = graph;
    char fromVertex = 0;
    char toVertex = 0;
    std::map<char,char> path;
    int maxFlow = 0;
    std::string pathStr;
```

```
        while (BFS(flowGraph, start, finish, path))
        {
            int pathFlow = INT_MAX;
            pathStr = finish;

            for (toVertex = finish; toVertex != start; toVertex = 
path[toVertex])
            {
                fromVertex = path[toVertex];
                pathFlow = std::min(pathFlow, 
flowGraph[fromVertex][toVertex]);
            }

            for (toVertex = finish; toVertex != start; toVertex = 
path[toVertex])
            {
                fromVertex = path[toVertex];
                flowGraph[fromVertex][toVertex] -= pathFlow;
                flowGraph[toVertex][fromVertex] += pathFlow;
                pathStr = std::string(1, fromVertex) + " --> " + 
pathStr;
            }
            maxFlow += pathFlow;
            printCurrentFlows(flowGraph, pathFlow, maxFlow, pathStr);
        }
        printResult(graph, flowGraph, maxFlow);
    }


    int main()
    {
        Graph graph;
        char start = 0;
        char finish = 0;
        char vertexFrom = 0;
        char vertexTo = 0;
```

```
    int pathLength = 0;
    int countVertex = 0;

    std::cin >> countVertex >> start >> finish;

    for (auto i = 0; i < countVertex; ++i)
    {
        std::cin >> vertexFrom >> vertexTo >> pathLength;
        graph[vertexFrom][vertexTo] = pathLength;
    }

    FF(graph, start, finish);
    return 0;
}
```