МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»

Тема: Потоки в сети

Студент гр. 9383 Поплавский И.

Преподаватель Фирсов М.А.

Санкт-Петербург

2021

Цель работы

Изучить алгоритм Форда-Фалкерсона поиска максимального потока в сети, а также реализовать данный алгоритм на языке программирования C++.

Основные теоретические положения.

Сеть – ориентированный взвешенный граф, имеющий один исток и один сток.

Исток – вершина, из которой рёбра только выходят.

Сток – вершина, в которую рёбра только входят.

Поток – абстрактное понятие, показывающее движение по графу.

Величина потока – числовая характеристика движения по графу (сколько всего выходит из стока = сколько всего входит в сток).

Пропускная способность – свойство ребра, показывающее, какая максимальная величина потока может пройти через это ребро.

Максимальный поток (максимальная величина потока) – максимальная величина, которая может быть выпущена из стока, которая может пройти через все рёбра графа, не вызывая переполнения ни в одном ребре.

Фактическая величина потока в ребре – значение, показывающее, сколько величины потока проходит через это ребро.

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

v_0 – исток

v_n – сток

v_i v_j ω_{ij} - ребро графа

v_i v_j ω_{ij} - ребро графа

...

Выходные данные:

Pmax - 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).

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
```

Соответствующие выходные данные

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

Вар. 3. Поиск в глубину. Рекурсивная реализация.

Реализация задачи

Описание алгоритма

Создается граф, заданный матрицей смежности. Изначально все вершины отмечены как не просмотренные, все потоки изначально равны 0.

Для нахождения пути в графе выполняется рекурсивный поиск в глубину.

Если удалось найти путь из истока в сток, то для данного пути выполняется поиск максимального потока.

Для прямых ребер поток увеличивается на найденную величину, для обратных ребер поток уменьшается на найденную величину.

Значение максимального потока, найденного для данного пути, прибавляется к конечному значению максимального потока для всего графа.

Сложность по памяти — линейная от количества ребер O(|E|), т. к. каждой будет соответствовать метка с информацией о величине потока.

Добавляя поток увеличивающего пути к уже имеющемуся потоку, максимальный поток будет получен, когда нельзя будет найти увеличивающий путь. Тем не менее, если величина пропускной способности — иррациональное число, то алгоритм может работать бесконечно. В целых числах таких проблем не возникает и время работы ограничено O(|E|*f), где E — число рёбер в графе, f — максимальный поток в графе, так как каждый увеличивающий путь может быть найден за O(E) и увеличивает поток как минимум на 1.

Описание функций и структур данных

Был использован следующий класс: class Ford_Falkerson

Переменные класса Ford_Falkerson:

- int size – количество ориентированных ребер;
- char source – исток графа;

- char stock – сток графа;
- char from – начальная вершина ребра;
- char to – конечная вершина ребра;
- int cost – величина потока ребра;
- int delta – константа для сортировки в лексикографическом порядке, равна 97 для букв и 49 для цифр;
- vector<vector<int>> graph – двумерный вектор, хранящий вершины графа;
- vector<vector<int>> flows – двумерный вектор, хранящий потоки графа;
- vector<bool>visited – вектор, хранящий информацию о просмотре вершин графа;
- vector<int> way – вектор, хранящий путь.

Методы класса Ford_Falkerson:

- Ford_Falkerson(int digit, char symbol):size(digit), source(symbol), stock(symbol), from(symbol), to(symbol), cost(digit), delta(digit), graph(COUNT, vector<int>(COUNT, 0)), flows(COUNT, vector<int>(COUNT, 0)), visited(COUNT, false), way(COUNT, 0) – конструктор класса. Принимает количество ориентированных рёбер digit и исток symbol. Создается граф, заданный матрицей смежности. Изначально все вершины отмечены как не просмотренные, все потоки изначально равны 0.
- void creation_graph() – считывает построчно ориентированные ребра графа, сортирует их в лексикографическом порядке и заполняет двумерный вектор графа.
- int FF() – функция реализующая алгоритм Форда-Фалкерсона, находит все возможные пути из истока в сток, вызывая функцию get_way, вычисляет их максимальный поток и возвращает его. Если пути от истока в сток нет, то алгоритм прекращает работу.

- bool get_way() –вызывает функцию поиска в глубину DFS, убирает метки просмотренных вершин после нахождения пути и возвращает true, если был найден путь из истока в сток.
- void DFS(int vertex) – функция принимает вершину vertex, от которой рекурсивно начинается поиск в глубину. Рассматривает все рёбра от текущей вершины, если ребро ведёт в вершину, которая не была рассмотрена ранее, то добавляем в путь новую вершину и запускаем алгоритм поиска в глубину от этой нерассмотренной вершины.
- void clear() – вспомогательная функция, очищает путь после нахождения максимального потока.
- void print_graph() – функция выводит список существующих путей с текущим потоком и стоимостью пути.
- void print_result() – функция выводит результат работы программы, т.е. отсортированные в лексикографическом порядке входные ребра с фактической величиной протекающего потока.

Тестирование

Входные данные	Выходные 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	12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2
9 a d a b 8 b c 10 c d 10 h c 10 e f 8	18 a b 8 a g 10 b c 0 b e 8 c d 10 e f 8 f d 8

g h 11 b e 8 a g 10 f d 8	g h 10 h c 10
5 a d a b 1000 a c 1000 b c 1 b d 1000 c d 1000	2000 a b 1000 a c 1000 b c 0 b d 1000 c d 1000
1 a c c a 5	0 c a 0
3 a c a b 7 a c 6 b c 4	10 a b 4 a c 6 b c 4

Вывод

В результате работы была написана полностью рабочая программа, решающая поставленную задачу при использовании изученных теоретических материалов. Программа было протестирована, результаты тестов удовлетворительны.

ПРИЛОЖЕНИЕ

ИСХОДНЫЙ КОД ПОРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>

#define COUNT 26

class Ford_Falkerson {
	int size;                                                    //количество
ориентированных ребер
	char source;                                            //исток
	char stock;                                                  //сток
	char from;                                                   //начальная вершина
ребра
	char to;                                                     //конечная вершина
ребра
	int cost;                                                    //величина потока
ребра
	int delta = 0;                                               //для сортировки в
лексикографическом порядке
	std::vector<std::vector<int>> graph;     //граф
	std::vector<std::vector<int>> flows;     //поток
	std::vector<bool>visited;                       //посещенные
	std::vector<int> way;                                        //путь

public:
	Ford_Falkerson(int digit, char symbol) :size(digit), source(symbol),
stock(symbol), from(symbol), to(symbol), cost(digit),
		delta(digit), graph(COUNT, std::vector<int>(COUNT, 0)), flows(COUNT,
std::vector<int>(COUNT, 0)),
		visited(COUNT, false), way(COUNT, 0) {}

	void creation_graph() {//создание графа
		std::cin >> size >> source >> stock;
		if (isalpha(source)) {
			delta = 97;//ASCII код a
		}
		else {
			delta = 49;//ASCII код 1
		}
		for (int i = 0; i < size; i++)//считываем ребра графа
		{
			std::cin >> from >> to >> cost;
			graph[from - delta][to - delta] = cost;
		}
	}

	void clear() {//очищает путь
		for (size_t i = 0; i < COUNT; i++) {
			way[i] = 0;
		}
	}

	void DFS(int vertex) {//поиск в глубину
		visited[vertex - delta] = true;
		for (size_t i = 0; i < visited.size(); i++) {
			//перебираем все исходящие из рассматриваемой вершины рёбра
			if (!visited[i] && (graph[vertex - delta][i] - flows[vertex -
delta][i] > 0 && graph[vertex - delta][i] != 0 || flows[vertex - delta][i] < 0 &&
graph[i][vertex - delta] != 0)) {
```

```
                        //если ребро ведёт в вершину, которая не была рассмотрена
ранее, то запускаем алгоритм от этой нерассмотренной вершины
                        way[i] = vertex;
                        DFS(i + delta);
                    }
                }
        }

        bool get_way() {
                DFS(source);//поиск в глубину от истока
                for (size_t i = 0; i < visited.size(); i++) {
                        visited[i] = false;//убираем просмотренные вершины
                }
                return (way[stock - delta] != 0);//дошел ли путь до стока
        }

        int alg_ff() {//алгоритм форда фалкерсона
                int max_flow = 0;
                while (get_way()) {
                        int tmp = std::numeric_limits<int>::max();
                        for (int v = stock - delta; 0 <= way[v] - delta; v = way[v] - delta)
{
                                tmp = std::min(tmp, graph[way[v] - delta][v] - flows[way[v] -
delta][v]);//находим максимальный поток
                        }
                        for (int v = stock - delta; 0 <= way[v] - delta; v = way[v] - delta)
{
                                flows[way[v] - delta][v] += tmp;//увеличиваем поток для
обратных путей
                                flows[v][way[v] - delta] -= tmp;//убавляем поток для текущих
путей
                        }
                        max_flow += tmp;
                        clear();//очищаем путь
                }
                return max_flow;
        }

        void print_result() {//выводим рёбра графа с фактическим потоком
                for (int i = 0; i < COUNT; i++) {
                        for (int j = 0; j < COUNT; j++) {
                                if (flows[i][j] != 0 && flows[i][j] < 0) {
                                        flows[i][j] = 0;
                                }
                                if (graph[i][j] > 0) {
                                        std::cout << (char)(i + delta) << " " << (char)(j +
delta) << " " << flows[i][j] << std::endl;
                                }
                        }
                }
        }
};
```