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

Студентка гр. 8383 Сырцова Е.А.

Преподаватель Фирсов М.А.

Санкт-Петербург

2020

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

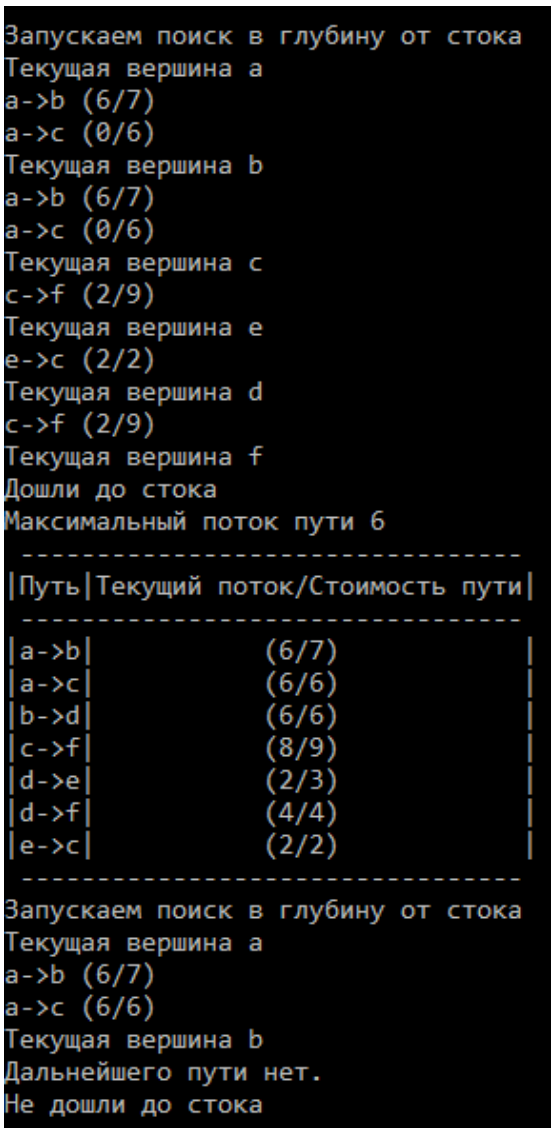
g h 11 b e 8 a g 10 f d 8	g h 10 h c 10
5 a d a b 1000 a c 1000 b c 1 b d 1000 c d 1000	2000 a b 1000 a c 1000 b c 0 b d 1000 c d 1000
1 a c c a 5	0 c a 0
3 a c a b 7 a c 6 b c 4	10 a b 4 a c 6 b c 4

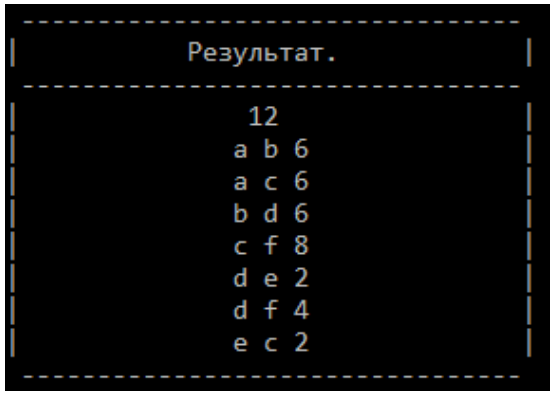
Пример вывода промежуточных данных

```
7
a
f
a b 7
a c 6
b d 6
c f 9
d e 3
d f 4
e c 2
 ---------------------------------
|Путь|Текущий поток/Стоимость пути|
 ---------------------------------
|a->b|           (0/7)            |
|a->c|           (0/6)            |
|b->d|           (0/6)            |
|c->f|           (0/9)            |
|d->e|           (0/3)            |
|d->f|           (0/4)            |
|e->c|           (0/2)            |
 ---------------------------------
Строим путь

Запускаем поиск в глубину от стока
Текущая вершина a
a->b (0/7)
a->c (0/6)
Текущая вершина b
b->d (0/6)
Текущая вершина d
d->e (0/3)
d->f (0/4)
Текущая вершина e
e->c (0/2)
Текущая вершина c
c->f (0/9)
Текущая вершина f
Дошли до стока
Максимальный поток пути 2
```

```
 ---------------------------------
|Путь|Текущий поток/Стоимость пути|
 ---------------------------------
|a->b|           (2/7)            |
|a->c|           (0/6)            |
|b->d|           (2/6)            |
|c->f|           (2/9)            |
|d->e|           (2/3)            |
|d->f|           (0/4)            |
|e->c|           (2/2)            |
 ---------------------------------
Запускаем поиск в глубину от стока
Текущая вершина a
a->b (2/7)
a->c (0/6)
Текущая вершина b
b->d (2/6)
Текущая вершина d
d->e (2/3)
d->f (0/4)
Текущая вершина e
d->e (2/3)
d->f (0/4)
Текущая вершина f
Текущая вершина c
Дошли до стока
Максимальный поток пути 4
 ---------------------------------
|Путь|Текущий поток/Стоимость пути|
 ---------------------------------
|a->b|           (6/7)            |
|a->c|           (0/6)            |
|b->d|           (6/6)            |
|c->f|           (2/9)            |
|d->e|           (2/3)            |
|d->f|           (4/4)            |
|e->c|           (2/2)            |
 ---------------------------------
```





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 <iostream>
#include <ctime>
#define N 40

class Square {
private:
	int  **coloring; //раскраска по номеру квадрата
	int  *abscissa;  //массив координат x
	int  *ordinate;  //массив координат y
	int  *length;    //массив длин сторон квадратов
	int  count;      //возможное кол-во квадратов
	int  size;       //размер текущего квадрата
	int  num;        //порядковый номер квадрата
	bool f;          //послдений возможный квадарт

	void insert_square(int x, int y, int n, int side);
	void remove_square(int x, int y, int side);
	bool place_to_insert(int &x, int &y);
	void multiple_of_three(int side);
	int  find_max_size(int x, int y);
	void multiple_of_five(int side);
	void insert_the_second_square();
	void insert_the_third_square();
	void even_square(int side);
	void print_square();

public:
	void output_of_the_result(int amount);
	int  insert_the_first_square();
	int  backtracking(int deep);
	Square(int size);
	~Square();
};

void Square::insert_square(int x, int y, int n, int side) { //заполнение квадрата
(координаты, цвет и сторона)
	for (int i = x; i < side + x; i++)
		for (int j = y; j < side + y; j++)
			coloring[i][j] = n;
}

void Square::remove_square(int x, int y, int side) {//удаление квадрата (координаты и
сторона)
	for (int i = x; i < side + x; i++)
		for (int j = y; j < side + y; j++)
			coloring[i][j] = 0;
}

bool Square::place_to_insert(int &x, int &y) { //поиск места для вставки нового
квадрата, функция ищет самую верхнюю и левую пустую клетку поля
	for (int i = 0; i < size; i++)
		for (int j = 0; j < size; j++)
			if (!coloring[i][j]) {//пустая клетка
				x = i;
				y = j;
```

```
                        return true;
                    }
        return false;
}

void Square::multiple_of_three(int side) { //разбиение квадрата, сторона которого
кратна трём
        abscissa[1] = abscissa[2] = abscissa[3] = ordinate[3] = ordinate[5] =
ordinate[4] = side;
        abscissa[5] = ordinate[2] = side * 1 / 2;
        abscissa[4] = ordinate[1] = 0;
        for (int i = 1; i < 6; i++)
                length[i] = size * 1 / 3;
}

int Square::find_max_size(int x, int y) {//находит максимальное значение стороны
квадрата, который возможно поместить на поле
        int  max_size;
        bool allowed = true;
        for (max_size = 1; allowed && max_size <= size - x && max_size <= size - y;
max_size++) //проверка на пересечение границ квадрата
                for (int i = 0; i < max_size; i++)
                        for (int j = 0; j < max_size; j++)
                                if (coloring[x + i][y + j]) { //проверка на пересечение с
уже существующим квадратом
                                        allowed = false;
                                        max_size--;
                                }
        max_size--;
        return max_size;
}

void Square::multiple_of_five(int side) {//разбиение квадрата, сторона которого
кратна пяти
        abscissa[1] = abscissa[2] = abscissa[7] = ordinate[4] = ordinate[5] =
ordinate[7] = side;
        abscissa[5] = abscissa[6] = ordinate[2] = ordinate[3] = size * 2 / 5;
        length[2] = length[3] = length[5] = length[6] = side * 1 / 3;
        length[1] = length[4] = length[7] = side * 2 / 3;
        abscissa[3] = ordinate[6] = size * 4 / 5;
        abscissa[4] = ordinate[1] = 0;
}

void Square::insert_the_second_square() {//вставляет второй квадрат слева от 1 на
поле так, чтобы его сторона была максимально возможной
        abscissa[num] = length[0];
        ordinate[num] = 0;
        length[num] = find_max_size(abscissa[num], ordinate[num]);
        insert_square(abscissa[num], ordinate[num], num + 1, length[num]);
        num++;
}

void Square::insert_the_third_square() {//вставляет третий квадрат снизу от 1 на поле
так, чтобы его сторона была максимально возможной
        abscissa[num] = 0;
        ordinate[num] = length[0];
        length[num] = find_max_size(abscissa[num], ordinate[num]);
        insert_square(abscissa[num], ordinate[num], num + 1, length[num]);
        num++;
}
```

```
void Square::even_square(int side) {//разбиение квадрата, сторона которого кратна двум
	abscissa[1] = abscissa[3] = ordinate[2] = ordinate[3] = size * 1 / 2;
	abscissa[2] = ordinate[1] = 0;
	for (int i = 0; i < 4; i++)
		length[i] = size * 1 / 2;
}

void Square::print_square() { //вспомогательная функция, которая выводит получившийся квадрат
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++)
			std::cout << coloring[i][j];
		std::cout << std::endl;
	}
	std::cout << std::endl;
}

void Square::output_of_the_result(int amount) {//выводит результат работы программы на консоль
	std::cout <<"Минимальное число квадратов для разбиения "<< amount << std::endl;//количество квадратов
	for (int i = 0; i < amount; i++)
		std::cout <<i+1<< " квадрат с координатами " << abscissa[i] + 1 << " и " << ordinate[i] + 1 << " и стороной " << length[i] << std::endl;
}

int Square::insert_the_first_square() {//функция вставки самого первого квадрата
	abscissa[num] = ordinate[num] = 0;
	if (size % 2 == 0) {				//если сторона квадрата кратна 2-м
		std::cout << "Квадрат с четной стороной. Частный случай. Делим квадрат на 4 части." << std::endl;
		even_square(size * 1 / 2);//квадраты со стороной 1/2 от длины
		return 4;
	}
	if (size % 3 == 0) {				//если сторона квадрата кратна 3-м
		std::cout << "Квадрат со стороной кратной 3. Частный случай. Делим квадрат на 6 частей." << std::endl;
		length[num] = size * 2 / 3;
		multiple_of_three(length[num]); //рисуем 6 квадратов со сторонами 2/3 и 5 по 1/3 от длины
		return 6;
	}
	if (size % 5 == 0) {				//если сторона квадрата кратна 5-ти
		std::cout << "Квадрат со стороной кратной 5. Частный случай. Делим квадрат на 8 частей." << std::endl;
		length[num] = size * 3 / 5;
		multiple_of_five(length[num]); //рисуем 8 квадратов со сторонами 3/5 3 по 2/5 и 4 по 1/5 от длины
		return 8;
	}
	else {							//все остальные случаи
		std::cout << "Общий случай. Используем бэктрекинг." << std::endl;
		length[num] = size * 1 / 2 + 1;
		insert_square(abscissa[num], ordinate[num], num + 1, length[num]);//вставляем квадрат со стороной больше половины
		num++;
		insert_the_second_square();//второй квадрат с максимальной длиной стороны
		insert_the_third_square();//и третий
		return backtracking(4);
	}
```

```
}

int Square::backtracking(int deep) {//перебирает всевозможные варианты кадрирования неразбитого участка поля
	if (f && deep > num) //bool f-последний квадрат 		//если текущее разбиение больше предыдущего
		return deep;
	int min_result = size * size;
	int temporary_length;//временная длина
	int temporary_result;//временный результат
	int temporary_x;//временные координаты
	int temporary_y;
	if (!place_to_insert(temporary_x, temporary_y)) { //если нет места для вставки
		if (!f || (f && deep - 1 < num))
			num = deep - 1;//номер последнего квадрата
		f = true;//последний квадрат
		return num;//количество квадратов
	}
	for (temporary_length = find_max_size(temporary_x, temporary_y); temporary_length > 0; temporary_length--) {//длина-максимальная для вставки
		insert_square(temporary_x, temporary_y, deep, temporary_length);//вставляем новый квадрат
		temporary_result = backtracking(deep + 1);//рекурсия-вставляем квадраты дальше
		min_result = min_result < temporary_result ? min_result : temporary_result;//если промежуточный результат меньше, то заменяем
		if (temporary_result <= num) { //если текущий результат не превосходит номер текущего квадрата (получили меньший результат)
			length[deep - 1] = temporary_length;//сохраняем в массивы данные
			abscissa[deep - 1] = temporary_x;
			ordinate[deep - 1] = temporary_y;
		}
		remove_square(temporary_x, temporary_y, temporary_length);//удаляем квадрат
	}
	return min_result;
}

Square::Square(int size) : size(size) {//конструктор
	coloring = new int*[size];//массив квадрата заполнен 0
	for (int i = 0; i < size; i++) {
		coloring[i] = new int[size];
		for (int j = 0; j < size; j++)
			coloring[i][j] = 0;
	}
	abscissa = new int[N];
	ordinate = new int[N];
	length = new int[N];
	count = N;
	f = false;
	num = 0;
}

Square::~Square() {
	delete[] abscissa;
	delete[] ordinate;
	delete[] length;
	for (int i = 0; i < size; i++)
		delete[] coloring[i];
	delete[] coloring;
}
```

```
int main() {
	system("chcp 1251");  //русский язык
	system("cls");        //очистить консоль
	int size, count;
	clock_t time;
	time = clock();
	std::cout << "Введите размер стола:" << std::endl;
	while (std::cin >> size && size > 0) {
		Square table(size);//создаем квадрат с именем стол заданного размера
		count = table.insert_the_first_square();
		table.output_of_the_result(count);
		time = clock() - time;
		std::cout << "Время работы программы в секундах  " << time /
CLOCKS_PER_SEC << std::endl;
		std::cout << std::endl << "Введите размер стола:" << std::endl;
		time = clock();
	}
	std::cout << "Неверный ввод" << std::endl;
}
```