МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»

Тема: Потоки в сети

Студентка гр. 8382 _____ Кулачкова М.К.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Написать программу для нахождения максимального потока в сети с использованием алгоритма Форда-Фалкерсона.

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 ребер графа

v_0 – исток

v_n – сток

$v_i\ v_j\ \omega_{ij}$ – ребро графа

$v_i\ v_j\ \omega_{ij}$ – ребро графа

…

Выходные данные:

P_{max} – величина максимального потока

$v_i\ v_j\ \omega_{ij}$ – ребро графа с фактической величиной протекающего потока

$v_i\ v_j\ \omega_{ij}$ – 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 ввода:

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 вывода:

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

Вариант дополнительного задания.

Вар. 4. Поиск в глубину. Итеративная реализация.

Описание алгоритма.

Программа запускает серию поисков в глубину, в результате каждого из которых по графу пускается увеличивающий поток. Поиск в глубину реализован итеративно при помощи стека:

1. В стек помещается исток, помечается как просмотренный.

   Пока стек не пуст, выполняется следующая последовательность действий:
2. Из стека извлекается верхняя вершина.
3. Если она является стоком, по найденному пути пускается поток. Функция возвращает его величину.
4. Если вершина не является стоком, просматриваются ее соседи. Если соседняя вершина не была посещена или ребро, соединяющее ее с текущей вершиной, имеет ненулевую остаточную пропускную

способность, и если добавление этой вершины в путь не создаст цикла, то вершина кладется в стек и помечается как посещенная.

Если все вершины были просмотрены, но путь не найден, функция возвращает -1.

Поток по найденному пути пускается следующим образом:

1. В пути находится ребро с наименьшей остаточной пропускной способностью *Cmin*.
2. Поток во всех ребрах пути увеличивается на *Cmin*, а остаточная пропускная способность уменьшается на *Cmin*.

Программа пытается найти увеличивающий путь, пока функция обхода в глубину не вернет -1, т.е. пока такой путь существует. Потоки, пускаемые по сети при каждом обходе, суммируются.

Оценка сложности алгоритма.

Обход сети в глубину обладает временной сложностью $O(E+V)$, где E – число ребер, V – число вершин. Обход в глубину запускается не более f раз, где f – величина максимального потока. Итого временная сложность алгоритма составляет $O(f(E+V))$.

Граф представляет собой массив вершин, каждая из которых содержит массив смежных ей ребер, поэтому пространственная сложность алгоритма будет составлять $O(E+V)$.

Описание структур данных.

В работе используются три структуры для хранения рёбер, вершин и самого графа.

Для хранения рёбер реализована структура *Edge*.

```
struct Edge {               //ребро
       char end;            //конец ребра
       int endInd;          //индекс конца ребра
       int cap;             //остаточная пропускная способность ребра
       int factFlow;        //фактический поток в ребре
       Edge(char ver = 0, int w = 0);
};
```

Она состоит из названия и индекса вершины на конце ребра в массиве вершин в графе, остаточной пропускной способности ребра и фактическому потоку в ребре.

Для хранения вершин реализована структура *Vertex*.

```
struct Vertex {             //вершина
       char name;           //имя вершины
       int neighNum; //количество смежных вершин
       bool visited; //флаг просмотренности
       string path; //путь до вершины
       Edge edges[MAX_VERTICES];   //массив инцидентных ребер
       Vertex(char name = 0);
       void addNeighbour(char nName, int cap);   //добавляет нового соседа
       int findEdge(char end); //возвращает индекс ребра с заданным концом
       bool cycle(char ver); //проверяет, не создаст ли добавление новой вершины
цикла
};
```

Она содержит имя вершины, число смежных с ней вершин, массив исходящих рёбер, строку, в которой хранится кратчайший путь из начальной вершины пути в эту, булеву переменную, равную *true*, если вершина была обработана алгоритмом поиска пути, и *false* в противном случае, а также конструктор, метод, добавляющий новое ребро в массив инцидентных рёбер, метод, возвращающий индекс ребра, соединяющего данную вершину с вершиной с заданным именем, а также метод, проверяющий, создаст ли добавление смежной вершины с указанным именем, цикл.

Для хранения графа реализована структура *Graph*.

```
struct Graph {              //граф
       int vnum;            //количество вершин в графе
       char source;  //исток
       char sink;           //сток
       int maxFlow;  //максимальный поток
       Vertex vertices[MAX_VERTICES];     //массив вершин
       Graph(char s, char t);
       void addEdge(char v1, char v2, int w);    //добавляет ребро в граф
       void addVertex(char name);                        //добавляет вершину в граф
       int find(char verName);     //возвращает индекс вершины с заданным именем
       void sort();  //сортирвка вершин в графе
       int dfs(); //поиск пути в глубину
       int letItFlow(string path); //пускает поток по заданному пути
};
```

Структура включает в себя число вершин в графе, статический массив вершин (число вершин в массиве равно 26, т.к. они обозначаются буквами английского алфавита), исток и сток, размер максимального потока, а также ряд методов, которые будут описаны в следующем пункте.

Описание функций и методов.

- Метод *void Vertex::addNeighbour(char nName, int cap)* добавляет новое ребро в массив рёбер, инцидентных данной вершине. Здесь *nName* – имя вершины на конце ребра, *cap* – пропускная способность ребра.
- Метод *bool Vertex::cycle(char ver)* проверяет, создаст ли добавление в путь вершины *ver*, смежной с данной, цикл.
- Метод *int Graph::find(char verName);* ищет в массиве вершин вершину с заданным именем. Возвращает индекс искомой вершины в графе; если вершины нет – возвращает -1.
- Метод *void Graph::addVertex(char name)* добавляет в граф вершину с заданным именем.
- Метод *void Graph::addEdge(char v1, char v2, int w)* добавляет в граф ребро с заданными характеристиками. Если вершин на концах ребра ещё не было в массиве вершин графа, они туда добавляются. Для начальной вершины вызывается метод *addNeighboor*.
- Метод *void Graph::sort()* выполняет сортировку вершин графа, а также ребер, инцидентных каждой вершине, в алфавитном порядке. Это делается для того, чтобы после нахождения максимального потока ребра выводились в требуемом порядке. Для работы функции были реализованы два компаратора: для вершин и для ребер.
- Метод *int Graph::dfs()* осуществляет поиск в глубину пути из истока в сток. Возвращает поток в этом пути.
- Метод *int Graph::letItFlow(string path)* пускает по пути *path* максимально возможный поток и возвращает его величину.

Тестирование.

№	Ввод	Вывод
1	7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	Maximum flow: 12 Flow through the edges: a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2
2	7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e d 2	Maximum flow: 10 Flow through the edges: a b 4 a c 6 b d 4 c f 6 d e 0 d f 4 e d 0
3	6 a d a b 2 a d 2 a c 3 b c 2 c d 5 b d 10	Maximum flow: 7 Flow through the edges: a b 2 a c 3 a d 2 b c 0 b d 2 c d 3
4	10 a g a b 5 a c 2 a d 3 b c 1 c e 4 d e 5 b f 3 e f 3 e g 3 f g 9	Maximum flow: 9 Flow through the edges: a b 4 a c 2 a d 3 b c 1 b f 3 c e 3 d e 3 e f 3 e g 3 f g 6

Выводы.

Была реализована программа, определяющая величину максимального потока в графе с использованием алгоритма Форда-Фалкерсона. Поиск улучшающих путей в графе осуществляется итеративным обходом в глубину. Программа выводит величину максимального потока, а также список ребер графа с величиной потока в них.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ
Файл main.cpp

```
#include <iostream>
#include <string>

#define MAX_VERTICES 26

using namespace std;

struct Edge {        //ребро
	char end;            //конец ребра
	int endInd;          //индекс конца ребра
	int cap;             //остаточная пропускная способность ребра
	int factFlow; //фактический поток в ребре
	Edge(char ver = 0, int w = 0);
};

Edge::Edge(char ver, int w) {
	end = ver;
	cap = w;
	endInd = -1;
	factFlow = 0;
}

struct Vertex {             //вершина
	char name;           //имя вершины
	int neighNum; //количество смежных вершин
	bool visited; //флаг просмотренности
	string path; //путь до вершины
	Edge edges[MAX_VERTICES];   //массив инцидентных ребер
	Vertex(char name = 0);
	void addNeighbour(char nName, int cap);   //добавляет нового соседа
	int findEdge(char end); //возвращает индекс ребра с заданным концом
	bool cycle(char ver); //проверяет, не создаст ли добавление новой вершины цикла
};

Vertex::Vertex(char name) {
	this->name = name;
	neighNum = 0;
	path = "";
	visited = false;
}

//добавление нового соседа
void Vertex::addNeighbour(char nName, int cap) {
	edges[neighNum] = Edge(nName, cap);
	neighNum++;
}

//поиск ребра с заданным концом
int Vertex::findEdge(char end) {
	for (int i = 0; i < neighNum; i++) {
		if (edges[i].end == end) return i;
	}
	return -1;
}

//проверка на цикл
```

```
bool Vertex::cycle(char ver) {
	if (path.find(ver) == string::npos) return false;
	return true;
}

struct Graph {			//граф
	int vnum;			//количество вершин в графе
	char source;  //исток
	char sink;			//сток
	int maxFlow;  //максимальный поток
	Vertex vertices[MAX_VERTICES];	//массив вершин
	Graph(char s, char t);
	void addEdge(char v1, char v2, int w);	//добавляет ребро в граф
	void addVertex(char name);			//добавляет вершину в граф
	int find(char verName);	//возвращает индекс вершины с заданным именем
	void sort();  //сортирвка вершин в графе
	int dfs(); //поиск пути в глубину
	int letItFlow(string path); //пускает поток по заданному пути
};

Graph::Graph(char s, char t) {
	vnum = 0;
	source = s;
	sink = t;
	maxFlow = 0;
}

//добавление ребра в граф
void Graph::addEdge(char v1, char v2, int w) {
	int ind1 = find(v1);
	if (ind1 < 0) {
		ind1 = vnum;
		addVertex(v1);
	}
	int ind2 = find(v2);
	if (ind2 < 0) {
		ind2 = vnum;
		addVertex(v2);
	}
	vertices[ind1].addNeighbour(v2, w);
}

//добавление вершины в граф
void Graph::addVertex(char name) {
	if (find(name) >= 0) return;
	vertices[vnum] = Vertex(name);
	vnum++;
}

//поиск вершины с заданным именем в массиве вершин графа
int Graph::find(char verName) {
	for (int i = 0; i < vnum; i++) {
		if (vertices[i].name == verName)
			return i;
	}
	return -1;
}

//компаратор для сортировки вершин графа в алфавитном порядке
int alphVerCmp(const void * a, const void * b) {
	return (((Vertex*)a)->name - ((Vertex*)b)->name);
}

//компаратор для сортировки ребер в алфавитном порядке
```

```
int alphEdgeCmp(const void * a, const void * b) {
	return (((Edge*)a)->end - ((Edge*)b)->end);
}

//сортировка вершин в графе
void Graph::sort() {
	qsort(vertices, vnum, sizeof(Vertex), alphVerCmp);
	for (int i = 0; i < vnum; i++) {
		qsort(vertices[i].edges, vertices[i].neighNum, sizeof(Edge), alphEdgeCmp);
		for (int j = 0; j < vertices[i].neighNum; j++) {
			vertices[i].edges[j].endInd = find(vertices[i].edges[j].end);
		}
	}
}

//поиск путей в глубину
int Graph::dfs() {
	int st[MAX_VERTICES];
	int ind = 0;
	st[ind] = find(source);
	vertices[find(source)].visited = true;
	vertices[find(source)].path = vertices[find(source)].name;
	while (ind >= 0) {
		int currVer = st[ind];
		ind--;
		if (vertices[currVer].name == sink) {
			return letItFlow(vertices[currVer].path); //если путь найден, пускаем
по нему поток и возвращаем величину потока
		}
		for (int i = 0; i < vertices[currVer].neighNum; i++) {
			if ((!vertices[vertices[currVer].edges[i].endInd].visited ||
vertices[currVer].edges[i].cap > 0) &&

	!vertices[currVer].cycle(vertices[vertices[currVer].edges[i].endInd].name)) {
				vertices[vertices[currVer].edges[i].endInd].path =
vertices[currVer].path + vertices[vertices[currVer].edges[i].endInd].name;
				st[++ind] = vertices[currVer].edges[i].endInd;
				vertices[vertices[currVer].edges[i].endInd].visited = true;
			}
		}
	}
	return -1;	//если путь не найден, возвращаем -1
}

//пуск потока по пути
int Graph::letItFlow(string path) {
	cout << "Found path: " << path << endl;
	int Cmin = 10000;
	cout << "\tCapacity of the edges: ";
	for (int i = 0; i < path.length() - 1; i++) {
		if (vertices[find(path[i])].findEdge(path[i + 1]) < 0) continue;
		cout << vertices[find(path[i])].edges[vertices[find(path[i])].findEdge(path[i
+ 1])].cap << " ";
		if (vertices[find(path[i])].edges[vertices[find(path[i])].findEdge(path[i +
1])].cap < Cmin) {
			Cmin =
vertices[find(path[i])].edges[vertices[find(path[i])].findEdge(path[i + 1])].cap;
		}
	}
	cout << endl;
	cout << "\tFlow in the path: " << Cmin << endl;
	cout << "\tNew capacity of the edges: ";
	for (int i = 0; i < path.length() - 1; i++) {
		if (vertices[find(path[i])].findEdge(path[i + 1]) < 0) continue;
```

```
            vertices[find(path[i])].edges[vertices[find(path[i])].findEdge(path[i +
1])].cap -= Cmin;
            cout << vertices[find(path[i])].edges[vertices[find(path[i])].findEdge(path[i
+ 1])].cap << " ";
            vertices[find(path[i])].edges[vertices[find(path[i])].findEdge(path[i +
1])].factFlow += Cmin;
      }
      cout << endl;
      return Cmin;
}

int main() {
      int n;
      char s, t;
      cout << "Enter the number of edges: ";
      cin >> n;
      cout << "Enter source vertex: ";
      cin >> s;
      cout << "Enter target vertex: ";
      cin >> t;
      Graph gr(s, t);
      char v1, v2;
      int w;
      cout << "Enter edges: " << endl;
      for (int i = 0; i < n; i++) {
            cin >> v1 >> v2 >> w;
            gr.addEdge(v1, v2, w);
      }
      gr.sort();    //сортировка вершин и ребер графа в алфавитном порядке,
                        //чтобы потом было удобно их выводить
      int c = gr.dfs();    //находим путь, по которому пускаем поток
      while (c >= 0) {
            gr.maxFlow += c;
            c = gr.dfs();
      }
      cout << "Maximum flow: " << gr.maxFlow << endl;
      cout << "Flow through the edges: " << endl;
      for (int i = 0; i < gr.vnum; i++) {
            for (int j = 0; j < gr.vertices[i].neighNum; j++) {
                  cout << gr.vertices[i].name << " " << gr.vertices[i].edges[j].end << "
" << gr.vertices[i].edges[j].factFlow << endl;
            }
      }
      return 0;
}
```