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. 8304 _____ Сергеев А.Д.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы.

Изучить понятие сети и научиться искать максимальный поток сети, используя алгоритм 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 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).

Дополнительное задание (вариант 3): Поиск в глубину. Рекурсивная реализация.

Порядок выполнения работы.

Написание работы производилось на базе операционной системы Windows 10 на языке программирования java в среде программирования IntelliJ IDEA.

Элементы сети представлены классами, находящимися в пакете graphic.

Класс *Net* содержит в себе саму сеть, представленную в виде словаря, ключами в котором являются буквы, а значениями — соответствующие им узлы. Также отдельно хранятся исток и сток.

Класс *Node* представляет из себя узел графа и содержит в себе соответствующую узлу букву, входной и выходной потоки, а также два списка дуг, которые начинаются и заканчиваются в нём соответственно. Оба списка представляют из себя словари, значениями в которых являются рёбра, а ключами — буквы, соответствующие узлам, с которыми эти дуги соединяют текущий узел.

Класс *Ark* изображает дугу графа и содержит в себе узел, из которого эта дуга выходит, узел, в который эта дуга входит, а также максимальный и текущий поток через эту дугу.

Класс *Path* представляет из себя путь по графу, он содержит связный список узлов, которые в этот граф входят. Так как алгоритм в момент построения пути не учитывает направление дуг, а в момент обработки учитывает, класс *Path* работает не с классом *Node*, а с его наследником — классом *DirectedNode*, который

содержит информацию о том, достигается ли данный узел по направлению дуги или против.

Класс *Pathfinder* содержит единственный метод solve, который находит максимальный поток в графе, используя алгоритм Форда-Фалкерсона, а также выполняет проверку правильности решения. Так как в заданиях, представленных на сайте stepik.org, не было соблюдено условие о том, что в исток не входят дуги, а из стока не выходят, проверка для этих узлов не выполняется.

Класс *Filer* осуществляет связь с файловой системой, открывает потоки ввода и вывода в файл.

Описание классов в UML-виде приложено к отчеты в файле UML.png.

Тестирование.

Для двух наборов входных данных (предоставленных на сайте stepik.org) было проведено тестирование алгоритма нахождения максимального потока в сети.

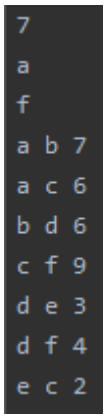

Рисунок 1 – Входные данные первого теста.

```
Inner path built: Path: a- b+ d+ e+ c+ f+
Inner path built: Path: a- b+ d+ f+
Inner path built: Path: a- c+ f+
Inner path built: Path: a- c+ e- d- f+
Innter path loops resolving...
12
a b 6
a c 6
b d 6
c f 8
d e 2
d f 4
e c 2
Operation finished!
```

Рисунок 2 – Вывод программы для первого теста.

```
16
a
e
a b 20
b a 20
a d 10
d a 10
a c 30
c a 30
b c 40
c b 40
c d 10
d c 10
c e 20
e c 20
b e 30
e b 30
d e 10
e d 10
```

Рисунок 3 – Входные данные второго теста.

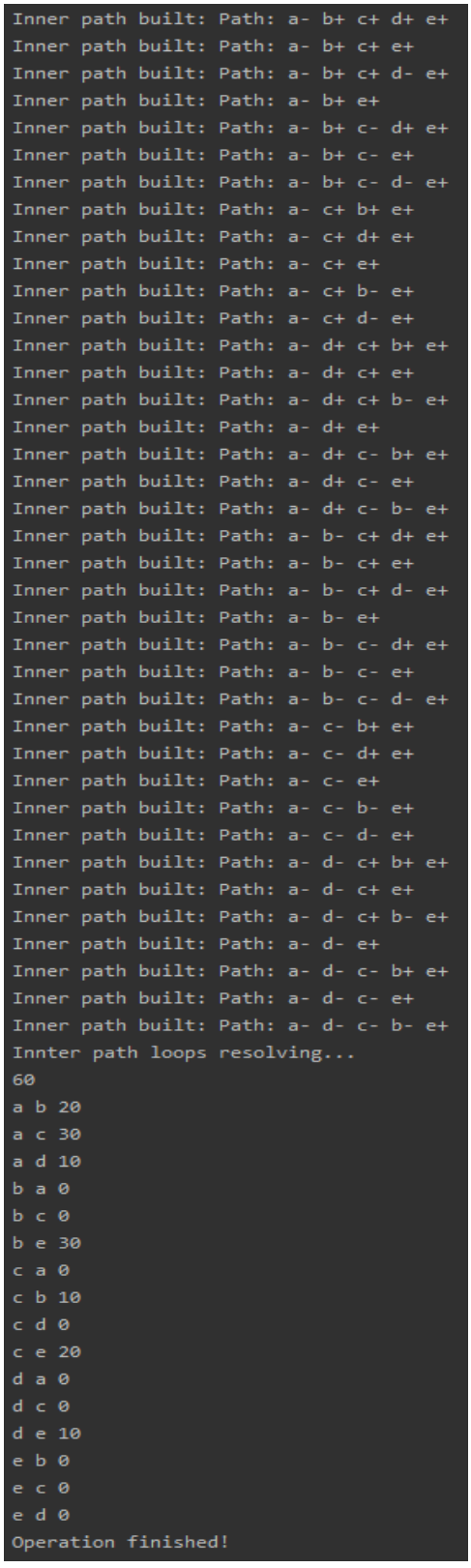

Рисунок 4 – Вывод программы для второго теста.

Вывод.

В результате лабораторной работы был изучен алгоритм Форда-Фалкерсона, а также получены знания о максимальном потоке в сети.

Приложение А

Исходный код программы, файл Net.java

```
package graphic;

import java.util.*;

public class Net {
    private Node source, exit;
    private HashMap<Character, Node> table;

    public Net(char source, char exit) {
        this.table = new HashMap<>();
        this.source = new Node(source);
        this.exit = new Node(exit);

        table.put(source, this.source);
        table.put(exit, this.exit);
    }

    public void putArk(char from, char to, int capacity) {
        Node fromN, toN;

        if (!table.containsKey(from)) {
            fromN = new Node(from);
            table.put(from, fromN);
        } else fromN = table.get(from);

        if (!table.containsKey(to)) {
            toN = new Node(to);
            table.put(to, toN);
```

```
            } else toN = table.get(to);

            Ark between = new Ark(fromN, toN, capacity);
            fromN.putArk(between);
            toN.putArk(between);
        }

        public char getSource() {
            return source.getName();
        }

        public char getExit() {
            return exit.getName();
        }

        public boolean check() {
            boolean passed = true;
            for (Map.Entry<Character, Node> node : table.entrySet())
{
                if ((node.getValue() == source) || (node.getValue()
== exit)) continue;
                passed &= node.getValue().inEqualsOut();
            }
            return passed;
        }

        public int getResultThrough() {
            return source.getOut();
        }

        public Map<Character, Map<Character, Integer>>
getOldStyleGraph() {
```

```
            Map<Character, Map<Character, Integer>> old = new
TreeMap<>();
            for (Map.Entry<Character, Node> node : table.entrySet())
{
                Map<Character, Integer> ark = new TreeMap<>();
                for (char leaf : node.getValue().getAncestors()) {
                    ark.put(leaf, getArk(node.getKey(),
leaf).getFilled());
                }
                old.put(node.getKey(), ark);
            }
            return old;
        }

        public void optimizeArks() {
            for (Map.Entry<Character, Node> node : table.entrySet())
{
                ArrayList<Character> anc =
node.getValue().getAncestors();
                ArrayList<Character> pred =
node.getValue().getPredecessors();
                for (Character a : anc) {
                    if (pred.contains(a)) {
                        Ark in = getArk(node.getKey(), a);
                        Ark out = getArk(a, node.getKey());
                        int delta = Math.min(in.getFilled(),
out.getFilled());
                        in.modifyFilled(-delta);
                        out.modifyFilled(-delta);
                    }
                }
            }
```

```
        }



        Node getNode(char node) {
            return table.get(node);
        }

        Ark getArk(char from, char to) {
            if (!table.containsKey(from) || !table.containsKey(to))
return null;
            return table.get(from).arkTo(to);
        }

        Ark getArkDirect(char from, char to) {
            Ark forth = getArk(from, to);
            Ark back = getArk(to, from);
            return forth == null ? back : forth;
        }
    }
```

Приложение Б

Исходный код программы, файл Node.java

```
package graphic;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;

class Node {
    private char name;
    private int in, out;
    private HashMap<Character, Ark> inArks;
    private HashMap<Character, Ark> outArks;

    Node(char name) {
        this.name = name;
        this.in = this.out = 0;
        this.inArks = new HashMap<>();
        this.outArks = new HashMap<>();
    }

    char getName() {
        return name;
    }

    int getIn() {
        return in;
    }

    int getOut() {
        return out;
```

```
        }

        ArrayList<Character> getNeighbours() {
                        ArrayList<Character>  neighbours  =  new
ArrayList<>(outArks.keySet());
            neighbours.addAll(inArks.keySet());
            neighbours.sort(Comparator.naturalOrder());
            return neighbours;
        }

        ArrayList<Character> getAncestors() {
                         ArrayList<Character>  ancestors  =  new
ArrayList<>(outArks.keySet());
            ancestors.sort(Comparator.naturalOrder());
            return ancestors;
        }

        ArrayList<Character> getPredecessors() {
                       ArrayList<Character>  predecessors  =  new
ArrayList<>(inArks.keySet());
            predecessors.sort(Comparator.naturalOrder());
            return predecessors;
        }

        void putArk(Ark nova) {
                        if  (nova.getFrom().getName()  ==  name)
outArks.put(nova.getTo().getName(), nova);
                    else  if  (nova.getTo().getName()  ==  name)
inArks.put(nova.getFrom().getName(), nova);
        }

        Ark arkTo(char to) {
```

```
        return outArks.get(to);
    }

    void modifyIn(int delta) {
        this.in += delta;
    }

    void modifyOut(int delta) {
        this.out += delta;
    }

    int inArksNum() {
        return inArks.size();
    }

    int outArksNum() {
        return outArks.size();
    }

    boolean inEqualsOut() {
        return in == out;
    }
}
```

Приложение В

Исходный код программы, файл Ark.java

```
package graphic;

class Ark {
    private Node from, to;
    private int capacity, filled;

    Ark(Node from, Node to, int capacity) {
        this.from = from;
        this.to = to;
        this.capacity = capacity;
        this.filled = 0;
    }

    Node getFrom() {
        return from;
    }

    Node getTo() {
        return to;
    }

    void modifyFilled(int delta) {
        this.filled += delta;
    }

    int getCapacity() {
        return capacity - filled;
    }
```

```
    int getFilled() {
        return filled;
    }
}
```

Приложение Г

Исходный код программы, файл Path.java

```
package graphic;

import java.io.PrintStream;
import java.util.ArrayList;
import java.util.LinkedList;

public class Path {
    private Net base;
    private LinkedList<DirectedNode> roadMap;

    public Path(Net base, char beginning) {
        this.base = base;
        this.roadMap = new LinkedList<>();
        pushNode(new DirectedNode(beginning, false));
    }

    public boolean pushNode(DirectedNode node) {
        for (DirectedNode nd : roadMap) {
            if (nd.getNode() == node.getNode()) return false;
        }
                  if  ((node.getNode()  ==  base.getSource())  &&
(node.isDirection())) return false;
                if  ((node.getNode()  ==  base.getExit())  &&  (!
node.isDirection())) return false;
        roadMap.push(node);
        return true;
    }

    public DirectedNode popNode() {
```

```
            return roadMap.pop();
        }

        public DirectedNode getEnd() {
            return roadMap.isEmpty() ? null : roadMap.peek();
        }

        public boolean isDeadEnd() {
            return roadMap.isEmpty();
        }

        public ArrayList<DirectedNode> getTopNeighbours() {
            if (roadMap.isEmpty()) return null;
            ArrayList<DirectedNode> result = new ArrayList<>();
                                ArrayList<Character>   ancestors   =
base.getNode(roadMap.peek().getNode()).getAncestors();
                               ArrayList<Character>   predecessors   =
base.getNode(roadMap.peek().getNode()).getPredecessors();
            for (Character anc : ancestors) {
                result.add(new DirectedNode(anc, true));
            }
            for (Character pred : predecessors) {
                result.add(new DirectedNode(pred, false));
            }
            return result;
        }

        public int findLastInNeighbours(DirectedNode node) {
            ArrayList<DirectedNode> neighbours = getTopNeighbours();
            for (int i = 0; i < neighbours.size(); i++) {
                if (neighbours.get(i).equals(node)) return i;
            }
```

```
            return -1;
        }



        public int getMinCapacity() {
            if (roadMap.size() < 2) return -1;
            int minCapacity;
            if (roadMap.get(roadMap.size()-2).isDirection())
                minCapacity = base.getArk(roadMap.get(roadMap.size()
-         1).getNode(),         roadMap.get(roadMap.size()         -
2).getNode()).getCapacity();
            else
                minCapacity = base.getArk(roadMap.get(roadMap.size()
-         2).getNode(),         roadMap.get(roadMap.size()         -
1).getNode()).getFilled();
            int capacity;
            Ark ark;
            for (int i = roadMap.size() - 3; i >= 0; i--) {
                if (roadMap.get(i).isDirection()) {
                       ark = base.getArk(roadMap.get(i+1).getNode(),
roadMap.get(i).getNode());
                    capacity = ark.getCapacity();
                } else {
                         ark = base.getArk(roadMap.get(i).getNode(),
roadMap.get(i+1).getNode());
                    capacity = ark.getFilled();
                }
                if (capacity < minCapacity) {
                    minCapacity = capacity;
                }
            }
```

```
            return minCapacity;
        }

        public void modifyFilled(Arked runner) {
            if (roadMap.size() < 2) return;
            Ark ark;
            for (int i = roadMap.size() - 2; i >= 0; i--) {
                if (roadMap.get(i).isDirection()) {
                       ark = base.getArk(roadMap.get(i+1).getNode(),
roadMap.get(i).getNode());
                    ark.modifyFilled(runner.modify());
                    ark.getFrom().modifyOut(runner.modify());
                    ark.getTo().modifyIn(runner.modify());
                } else {
                         ark = base.getArk(roadMap.get(i).getNode(),
roadMap.get(i+1).getNode());
                    ark.modifyFilled(-runner.modify());
                    ark.getFrom().modifyOut(-runner.modify());
                    ark.getTo().modifyIn(-runner.modify());
                }
            }
        }

        public interface Arked {
            int modify();
        }



        @Override
        public String toString() {
            StringBuilder b = new StringBuilder();
```

```
            for (int i = roadMap.size() - 1; i >= 0; i--) {
                b.append(roadMap.get(i));
                b.append(' ');
            }
            return "Path: " + b.toString();
        }



        public static class DirectedNode {
            private char node;
            private boolean direction;

            public DirectedNode(char node, boolean direction) {
                this.node = node;
                this.direction = direction;
            }

            public char getNode() {
                return node;
            }

            public boolean isDirection() {
                return direction;
            }

            public boolean equals(DirectedNode other) {
                 return (this.node == other.node) && (this.direction
== other.direction);
            }

            @Override
```

```
            public String toString() {
                  return String.valueOf(node) + (direction ? '+' :
'-');
            }
         }
     }
```

Приложение Д

Исходный код программы, файл Pathfinder.java

```
import classes.graphic.Net;
import classes.graphic.Path;

import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Map;
import java.util.Scanner;

public class Pathfinder {
    private Net net;

    public Pathfinder(Scanner sc) {
        int arks = Integer.parseInt(sc.next());

        char first = sc.next().charAt(0);
        char last = sc.next().charAt(0);

        this.net = new Net(first, last);

        char source;
        char target;
        int capacity;
        while (sc.hasNextLine()) {
            source = sc.next().charAt(0);
            target = sc.next().charAt(0);
            capacity = Integer.parseInt(sc.next());
            net.putArk(source, target, capacity);
        }
    }
```

```
        public void solve(PrintStream ps) {
            Path path = new Path(net, net.getSource());
            int lastVisitedPos = -1;
            while (!path.isDeadEnd()) {
                if (path.getEnd().getNode() == net.getExit()) {
                                ps.println("Inner path built: " +
path.toString());
                    operate(path);
                }
                        ArrayList<Path.DirectedNode> neighbours =
path.getTopNeighbours();
                  if ((lastVisitedPos + 1 == neighbours.size()) ||
(path.getEnd().getNode() == net.getExit())) {
                    Path.DirectedNode lastVisited = path.popNode();
                    if (path.isDeadEnd()) break;
                                                lastVisitedPos =
path.findLastInNeighbours(lastVisited);
                } else {
                         Path.DirectedNode next = neighbours.get(+
+lastVisitedPos);
                    if (path.pushNode(next)) lastVisitedPos = -1;
                }
            }

            ps.println("Innter path loops resolving...");
            net.optimizeArks();

            if (net.check()) {
                ps.println(net.getResultThrough());
                  Map<Character, Map<Character, Integer>> oldGraph =
net.getOldStyleGraph();
```

```
                    for (Map.Entry<Character, Map<Character, Integer>>
node: oldGraph.entrySet()) {
                                for (Map.Entry<Character, Integer> ark :
node.getValue().entrySet()) {
                                        ps.println(node.getKey() + " " +
ark.getKey() + " " + ark.getValue());
                                }
                            }
                        } else ps.println("Operation failure, aborting.");
                    }

                    private void operate(Path path) {
                        int minCapacity = path.getMinCapacity();
                        path.modifyFilled(() -> minCapacity);
                    }
                }
```

Приложение Е

Исходный код программы, файл Filer.java

```
import java.io.*;
import java.nio.file.Paths;

public class Filer {
    private static final String afterPath = "\\src\\io\\";
    public static final String inputFile = "input.txt";
    public static final String outputFile = "output.txt";

    public static InputStream readFromFile(String fileName) {
                                        String absolute =
Paths.get("").toAbsolutePath().toString() + afterPath + fileName;
        FileInputStream fist;
        try {
            fist = new FileInputStream(absolute);
        } catch (FileNotFoundException ffe) {
            fist = null;
        }
        return fist;
    }

    public static PrintStream writeToFile(String fileName) {
                                        String absolute =
Paths.get("").toAbsolutePath().toString() + afterPath + fileName;
        PrintStream fist;
        try {
            fist = new PrintStream(absolute);
        } catch (FileNotFoundException ffe) {
            fist = null;
        }
```

```
            return fist;
        }
    }
```

Приложение Ж

Исходный код программы, файл Main.java

```
import java.io.PrintStream;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        System.out.print("Press I to input manually or enter the
file name (for default file use D): ");
        Scanner sc = new Scanner(System.in);
        String inp = sc.next();

        Scanner src;
        if (inp.charAt(0) == 'D') {
                                                    src    =    new
Scanner(Filer.readFromFile(Filer.inputFile));
        } else if (inp.charAt(0) == 'I') {
            src = sc;
        } else {
            src = new Scanner(Filer.readFromFile(inp));
        }

        System.out.print("Press O to input into console or enter
the file name (for default file use D): ");
        inp = sc.next();
        PrintStream ps;
        if (inp.charAt(0) == 'D') {
            ps = Filer.writeToFile(Filer.outputFile);
        } else if (inp.charAt(0) == 'O') {
            ps = System.out;
        } else {
```

```
            ps = Filer.writeToFile(inp);
        }

        Pathfinder pf = new Pathfinder(src);
        pf.solve(ps);

        src.close();
        sc.close();
        ps.flush();
        System.out.println("Operation finished!");
    }
}
```