

UNIVERSITATEA "OVIDIUS" CONSTANȚA  
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ  
SPECIALIZAREA INFORMATICĂ

RAPORT PRACTICĂ: 25.06-06.07.2018

**Student:**  
Alexandrea Corina

Constanța  
Iunie-Iulie 2018

# Cuprins

1	Introducere	2
2	Activități planificate	3
3	25.06.2018	4
4	26.06.2018	5
5	27.06.2018	6
6	28.06.2018	7
7	29.06.2018	9
8	02.07.2018	10
9	03.07.2018	11
10	04.07.2018	14
11	05.07.2018	16
12	06.07.2018	17
13	Concluzii	18

# Capitolul 1

## Introducere

Acest raport cuprinde descrierea activității desfășurate la practică la calculator în perioada 25.06-06.07.2018 în cadrul Facultății de Matematică și Informatică.

## Capitolul 2

# Activități planificate

- Luni, 25.06.2018  
Aducerea la cunoștință a obiectivelor și cerințelor practicii la calculator.
- Marți, 26.06.2018  
Am ales tema: "Algoritmul de căutare în adâncime într-un graf".
- Miercuri, 27.06.2018  
Am studiat și am practicat Latex: am realizat prima pagina a lucrării și am stabilit activitățile pe care doresc să le parcurg.
- Joi, 28.06.2018  
Am căutat informații referitoare la grafuri și parcurgerea grafurilor.
- Vineri, 29.06.2018  
Am rezolvat două exemple pentru căutarea în adâncime într-un graf.
- Luni, 02.07.2018  
Am căutat pseudocodul pentru tema aleasă.
- Marți, 03.07.2018  
Am scris codul în limbajul java.
- Miercuri, 04.07.2018  
Am verificat funcționalitatea codului.
- Joi, 05.07.2018  
Am completat raportul în latex cu etapele pe care le-am parcurs în zilele precedente.
- Vineri, 06.07.2018  
Prezentarea lucrărilor. Notarea finală a activității.

# Capitolul 3

**25.06.2018**

Am desfășurat următoarele activități:

1. Am studiat obiectivele și cerințele ale de practica la calculator. Am clarificat situațiile incerte.
2. Mi-am creat un cont pe [www.github.com](https://www.github.com) cu scopul de a colabora cu o colega pentru realizarea raportului și un cont pe [www.overleaf.com](https://www.overleaf.com) pentru a scrie în mediu online în latex raportul de practică.

# Capitolul 4

**26.06.2018**

Am ales tema: "Algoritmul de cautare în adancime într-un graf".

## Capitolul 5

**27.06.2018**

Am studiat și am practicat Latex: am realizat prima pagina a lucrării și am stabilit activitățile pe care doresc să le parcurg.

# Capitolul 6

## 28.06.2018

Căutarea sau parcurgerea în adâncime este un algoritm pentru parcurgerea sau căutarea într-o structură de date de tip arbore sau graf. Se începe de la rădăcină și se explorează cât mai mult posibil de-a lungul fiecărei ramuri înainte de a face pași înapoi.

Prin parcurgerea unui graf se înțelege "vizitarea" vârfurilor sale într-o anumită ordine, data de un anumit criteriu.

Există doi algoritmi de parcurgere a grafurilor:

- algoritmul de parcurgere în adâncime BF;
- algoritmul de parcurgere în lățime DF.

### 1. Algoritmul de parcurgere în adâncime BF

În toate cele trei tipuri de parcurgere în adâncime se vizitează mai întâi subarboarele stâng, apoi subarboarele drept. Diferența constă în poziția rădăcinii față de cei doi subarbori. Fie scrise recursiv, fie iterativ, procedurile de parcurgere în adâncime necesită o stivă. Clasificarea tipurilor de parcurgere în adâncime:

- Preordine (rsd): Pentru a parcurge un arbore binar în preordine, se vizitează mai întâi rădăcina, se parcurge în preordine subarboarele stâng, apoi se parcurge în preordine subarboarele drept.
- Inordine (srd): Pentru a parcurge în inordine un arbore binar, se parcurge în inordine subarboarele stâng, se vizitează rădăcina, apoi se parcurge în inordine subarboarele drept.
- Postordine (sdr): Pentru a parcurge în postordine un arbore binar, se parcurge în postordine subarboarele stâng, apoi cel drept, apoi se vizitează rădăcina.

### 2. Algoritmul de parcurgere în lățime DF

Parcurgerea în lățime începe cu vârful inițial, denumit și vârful de start. Se vizitează mai întâi vârful de start. Se vizitează în ordine toți vecinii



nevizitați ai vârfului de start. Apoi se vizitează în ordine toți vecinii nevizitați ai vecinilor vârfului de start și așa mai departe, până la epuizarea tuturor vârfurilor accesibile din vârful de start.

# Capitolul 7

29.06.2018

Am rezolvat două exemple pentru cautarea în adâncime într-un graf. Exemplu:  
Se da graful:

Care este parcurgerea în adâncime a grafului?

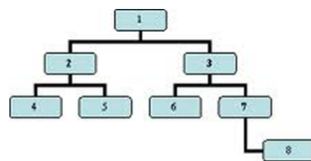


Figura 7.1: Graf neorientat

Parcurgerea în preordine a arborelui din Figura 7.1 este: 1 2 4 5 3 6 7 8.

Parcurgerea în inordine a arborelui din Figura 7.1 este: 4 2 5 1 6 3 7 8.

Parcurgerea în postordine a arborelui din Figura 7.1 este: 4 5 2 6 8 7 3 1.

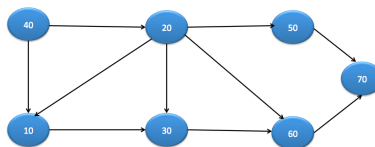


Figura 7.2: Graf orientat

Parcurgerea în adâncime a grafului din Figura 7.2 este: 40, 20, 50, 70, 60, 30, 10.

# Capitolul 8

02.07.2018

Am căutat pseudocodul pentru tema aleasa.

Intrare: Un graf  $G$  și un nod  $v$  din  $G$

Ieșire: Toate nodurile accesibile din  $v$  etichetate pe măsură ce sunt descoperite

O implementare recursivă a lui DFS:

1. procedure DFS( $G, v$ ):
2. etichetează  $v$  ca descoperit
3. for all muchie de la  $v$  la  $w$  in  $G.\text{adjacentEdges}(v)$  do
4.     nodul  $w$  nu este etichetat ca descoperit then
5.     apelează recursiv DFS( $G, w$ )

Ordinea în care nodurile sunt descoperite prin acest algoritm se numește ordine lexicografică.

# Capitolul 9

## 03.07.2018

Am scris codul în limbajul java în NetBeans:

```
package org.arpit.java2blog;
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

public class DepthFirstSearchExampleNeighbourList
{
    static class Node
    {
        int data;
        boolean visited;
        List<Node >neighbours;

        Node(int data)
        {
            this.data=data;
            this.neighbours=new ArrayList<>();
        }
        public void addneighbours(Node neighbourNode){
            this.neighbours.add(neighbourNode);
        }
        public List <Node >getNeighbours() {
            return neighbours;
        }
        public void setNeighbours(List<Node>neighbours) {
            this.neighbours = neighbours;
        }
    }

    // Recursive DFS
```

```

public void dfs(Node node)
{
    System.out.print(node.data + " ");
    List <Node >neighbours=node.getNeighbours();
    node.visited=true;
    for (int i = 0; i <neighbours.size(); i++) {
        Node n=neighbours.get(i);
        if(n!=null && !n.visited)
        {
            dfs(n);
        }
    }
}

// Iterative DFS using stack
public void dfsUsingStack(Node node)
{
    Stack <Node >stack=new Stack<Node >();
    stack.add(node);
    node.visited=true;
    while (!stack.isEmpty())
    {
        Node element=stack.pop();
        System.out.print(element.data + " ");
        List <Node >neighbours=element.getNeighbours();
        for (int i = 0; i <neighbours.size(); i++) {
            Node n=neighbours.get(i);
            if(n!=null && !n.visited)
            {
                stack.add(n);
                n.visited=true;
            }
        }
    }
}

public static void main(String arg[])
{
    Node node40 =new Node(40);
    Node node10 =new Node(10);
    Node node20 =new Node(20);
    Node node30 =new Node(30);
    Node node60 =new Node(60);

    hspace*1cmNode node50 =new Node(50);
    Node node70 =new Node(70);
    node40.addneighbours(node10);

```

```

        node40.addneighbours(node20);
        node10.addneighbours(node30);
        node20.addneighbours(node10);
        node20.addneighbours(node30);
        node20.addneighbours(node60);
        node20.addneighbours(node50);
        node30.addneighbours(node60);
        node60.addneighbours(node70);
        node50.addneighbours(node70);
        DepthFirstSearchExampleNeighbourList dfsExample = new DepthFirst-
SearchExampleNeighbourList();
        System.out.println("The DFS traversal of the graph using stack ");
        dfsExample.dfsUsingStack(node40);
        System.out.println();
        // Resetting the visited flag for nodes
        node40.visited=false;
        node10.visited=false;
        node20.visited=false;
        node30.visited=false;
        node60.visited=false;
        node50.visited=false;
        node70.visited=false;
        System.out.println("The DFS traversal of the graph using recursion ");
        dfsExample.dfs(node40);
    }
}

```

# Capitolul 10

04.07.2018

Am verificat functionalitatea codului.  
Codul implementat in NetBeans:

```
1 package org.arpit.java2blog;
2 import java.util.ArrayList;
3 import java.util.List;
4 import java.util.Stack;
5
6 public class DepthFirstSearchExampleNeighbourList
7 {
8     static class Node
9     {
10         int data;
11         boolean visited;
12         List<Node> neighbours;
13
14         Node(int data)
15         {
16             this.data = data;
17             this.neighbours = new ArrayList<>();
18         }
19
20         public void addNeighbours(Node neighbourNode)
21         {
22             this.neighbours.add(neighbourNode);
23         }
24
25         public List<Node> getNeighbours() {
26             return neighbours;
27         }
28
29         public void setNeighbours(List<Node> neighbours) {
30             this.neighbours = neighbours;
31         }
32     }
33 }
```

(a) 1

```
34 // Recursive DFS
35 public void dfs(Node node)
36 {
37     System.out.print(node.data + " ");
38     List<Node> neighbours = node.getNeighbours();
39     node.visited = true;
40     for (int i = 0; i < neighbours.size(); i++) {
41         Node n = neighbours.get(i);
42         if (n != null && !n.visited)
43             dfs(n);
44     }
45 }
46
47 // Iterative DFS using stack
48 public void dfsUsingStack(Node node)
49 {
50     Stack<Node> stack = new Stack<Node>();
51     stack.add(node);
52     node.visited = true;
53     while (!stack.isEmpty())
54     {
55         Node element = stack.pop();
56         System.out.print(element.data + " ");
57     }
58 }
```

(b) 2

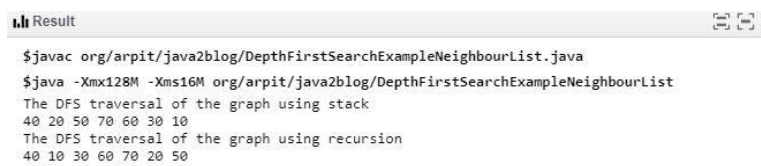
```
87 node20.addNeighbours(node10);
88 node20.addNeighbours(node40);
89 node20.addNeighbours(node50);
90 node30.addNeighbours(node40);
91 node40.addNeighbours(node70);
92 node50.addNeighbours(node70);
93
94 DepthFirstSearchExampleNeighbourList dfsExample = new
95     DepthFirstSearchExampleNeighbourList();
96
97 System.out.println("The DFS traversal of the graph using stack ");
98 dfsExample.dfsUsingStack(node40);
99
100 System.out.println();
101
102 // Marking the visited flag for nodes
103 node40.visited = false;
104 node10.visited = false;
105 node20.visited = false;
106 node30.visited = false;
107 node50.visited = false;
108 node70.visited = false;
109
110
111 System.out.println("The DFS traversal of the graph using recursion ");
112 dfsExample.dfs(node40);
113 }
```

(c) 3

```
87 node20.addNeighbours(node10);
88 node20.addNeighbours(node40);
89 node20.addNeighbours(node50);
90 node30.addNeighbours(node40);
91 node40.addNeighbours(node70);
92 node50.addNeighbours(node70);
93
94 DepthFirstSearchExampleNeighbourList dfsExample = new
95     DepthFirstSearchExampleNeighbourList();
96
97 System.out.println("The DFS traversal of the graph using stack ");
98 dfsExample.dfsUsingStack(node40);
99
100 System.out.println();
101
102 // Marking the visited flag for nodes
103 node40.visited = false;
104 node10.visited = false;
105 node20.visited = false;
106 node30.visited = false;
107 node50.visited = false;
108 node70.visited = false;
109
110
111 System.out.println("The DFS traversal of the graph using recursion ");
112 dfsExample.dfs(node40);
113 }
```

(d) 4

Rezultatul exemplului din Figura 7.2.:



The screenshot shows a terminal window with a title bar that says "Result". The terminal contains the following text:

```
$javac org/arpit/java2blog/DepthFirstSearchExampleNeighbourList.java
$java -Xmx128M -Xms16M org/arpit/java2blog/DepthFirstSearchExampleNeighbourList
The DFS traversal of the graph using stack
40 20 50 70 60 30 10
The DFS traversal of the graph using recursion
40 10 30 60 70 20 50
```

Figura 10.1: Exemplul din Figura 7.2.



# Capitolul 11

**05.07.2018**

Am completat raportul în latex cu etapele pe care le-am parcurs în zilele precedente.

## Capitolul 12

**06.07.2018**

Prezentarea proiectului. Notarea finală a activității.

## Capitolul 13

# Concluzii

Am învățat să lucrez cu Latex și Git. În latex am învățat să scriu cu diacritice cu ajutorul pachetelor “`\usepackage{ucs}`” și “`\usepackage[english,romanian]{babel}`”, să realizez un cuprins “`\tableofcontents`”, să folosesc diferite liste “`\begin{itemize}`”... “`\end{itemize}`” sau “`\begin{enumerate}`”... “`\end{enumerate}`”, să inserez poze cu pachetul “`\usepackage{graphicx}`” și instrucțiunea “`\includegraphics[]{}{}`” și să realizez o bibliografie cu BibTex.

Mi-am creat un cont pe Github și am realizat un depozit numit “practica2018”, unde am introdus fișierele care conțin raportul de practica cu extensia “tex” și “pdf”.

Raportul de practică poate fi găsit și pe <https://github.com/korinaa/practica2018>.

Pentru crearea raportului am utilizat cărțile: [4] și [3] și surse online: [2] și [1].

# Bibliografie

- [1] *Bibliography management with bibtex.*
- [2] Parcurgerea în adâncime în java, 2017.
- [3] Ludmila Malahova Alexandru Colesnicov. *Latex prin exemple.* 2001.
- [4] Andrei Corlat Sergiu Corlat. *Grafuri.Noțiuni.Algoritmi.Implementări.* 2012.