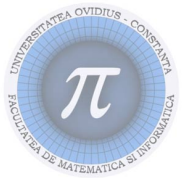


Universitatea "Ovidius" Constanţa
Facultatea de Matematică şi Informatică
Specializarea Informatică

Raport practică: 25.06-06.07.2018

Student:
Tudorică Anca-Maria

Constanţa
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 activitatii desfasurate la practica la calculator in perioada 25.06-06.07.2018 in cadrul Facultatii de Matematica si Informatica.

Capitolul 2

Activități planificate

- Luni, 25.06.2018
  Aducerea la cunoștință a obiectivelor și cerințelor practicii de producție
- Marți, 26.06.2018
  Am ales tema: "Algoritmul de cautare in adancime intr-un graf".
- Miercuri, 27.06.2018
  Am studiat și am practicat Latex: am realizat prima pagina a lucrarii si am stabilit activitatile pe care doresc sa le parcurg.
- Joi, 28.06.2018
  Am cautat informatii referitoare la grafuri și parcurgerea grafurilor.
- Vineri, 29.06.2018
  Am rezolvat un exemplu pentru cautarea in adancime intr-un graf.
- Luni, 02.07.2018
  Am cautat pseudocodul pentru tema aleasa.
- Marți, 03.07.2018
  Am scris codul in limbajul java.
- Miercuri, 04.07.2018
  Am verificat functionalitatea codului.
- Joi, 05.07.2018
  Am completat raportul în latex cu etapele pe care le-am parcurs in zilele precedente.
- Vineri, 06.07.2018
  Prezentarea lucrărilor Notarea finală a activității

Capitolul 3

25.06.2018

Am desfăţurat următoarele activităţi:

- Am identificat sursele pentru MikTeX, Git, SmartGit și BitBucket.
  - Am identificat sursele pentru MikTeX, Git, SmartGit și BitBucket.
  - Am instalat, configurat pe calculatorul de lucru aplicațiile necesare:
    * MikTeX
    * SmartGit
    * Bitbucket
  - Am instalat, configurat pe calculatorul de lucru aplicațiile necesare:
    * MikTeX
    * SmartGit
    * Bitbucket

Studierea obiectivelor și cerințelor față de practica de producție. Clarificarea situațiilor incerte.

Capitolul 4

26.06.2018

Am ales tema: "Algoritmul de cautare in adancime intr-un graf".

Capitolul 5

27.06.2018

Am studiat și am practicat Latex: am realizat prima pagina a lucrarii si am stabilit activitatile pe care doresc sa 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 întelege "vizitarea" vârfurilor sale într-o anumita ordine, data de un anumit criteriu.

Exista doi algoritmi de parcurgere a grafurilor:

- algoritmul de parcurgere în latime BF;
- algoritmul de parcurgere în adîncime DF.

1. Algoritmul de parcurgere în adâncime BF
   În toate cele trei tipuri de parcurgere în adâncime se vizitează mai întâi subarborele stâng, apoi subarborele drept. Diferenţa constă în poziţia rădăcinii faţă 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 subarborele stâng, apoi se parcurge în preordine subarborele drept.
   - Inordine (srd): Pentru a parcurge în inordine un arbore binar, se parcurge în inordine subarborele stâng, se vizitează radăcina, apoi se parcurge în inordine subarborele drept.
   - Postordine (sdr): Pentru a parcurge în postordine un arbore binar, se parcurge în postordine subarborele stâng, apoi cel drept, apoi se vizitează rădăcina.
2. Algoritmul de parcurgere în adâncime DF
   Parcurgerea în lăţime începe cu vârful iniţial, denumit şi vârful de start. Se vizitează mai întâi vârful de start. Se vizitează în ordine toţi vecinii

nevizitaţi ai vârfului de start. Apoi se vizitează în ordine toţi vecinii nevizitaţi ai vecinilor vârfului de start şi aşa mai departe, până la epuizarea tuturor vârfurilor accesibile din vârful de start.

Capitolul 7

29.06.2018

Am rezolvat un exemplu pentru cautarea in adâncime intr-un graf. Exemplu: Se da graful:
Care este parcurgerea in adancime a grafului?

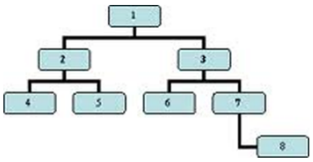


Figura 7.1: Graf neorinetat

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.

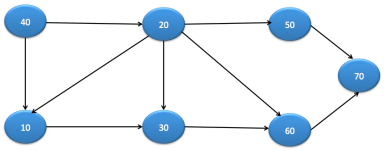


Figura 7.2: Graf orinetat

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

Capitolul 8

02.07.2018

Am cautat 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.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 lexigrafică.

Capitolul 9

03.07.2018

Am scris codul in 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
```

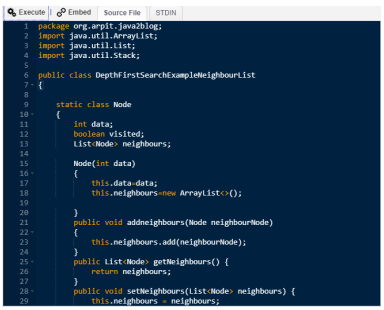
```
public void dfs(Node node)
{
      System.out.print(node.data + " ");
      List <Node >neighbours=node.getNeighbours();
      node.visited=true;
      for (int i = 0; i <neighbours.size(); i++) {
       Node n=neighbours.get(i);
       if(n!=null && !n.visited)
       {
           dfs(n);
     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

// 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

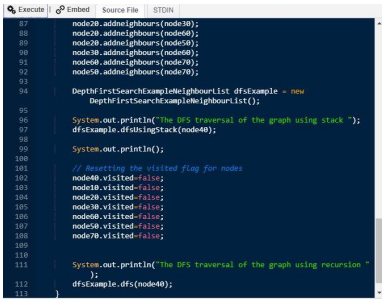
04.07.2018

Am verificat functionalitatea codului.
Codul implementat in NetBeans:

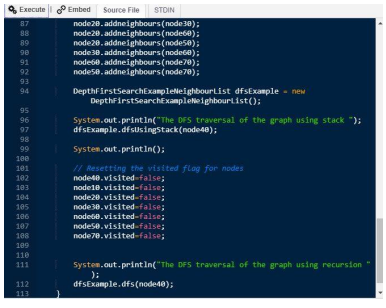


(a) 1

(b) 2



(c) 3



(d) 4

Rezultatul exemplului din Figura 7.2.:

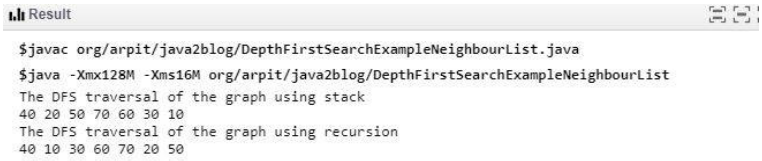


Figura 10.1: Exemplul din Figura 7.2.

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 invățat să lucrez cu Latex și Git. În latex am învățat să scriu cu diacritice cu ajutorul pachetului "usepackageucs" si "usepackage[english,romanian]babel", sa realizez un cuprins "tableofcontents", sa folosesc diferite liste "beginitemize" si sa inserez poze cu pachetul "usepackagegraphicx".

Pentru crearea raportului am utilizat cartile: [4] și [3] si surse online: [2] si [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.