Laborator 4:
structuri de date(II)

Întocmit de: Claudia Pârloagă

Îndrumător: Asist. Drd. Gabriel Danciu

I. NOŢIUNI TEORETICE

A. Grafuri

DEFINIŢII:

Un **graf** este o pereche $G =< V, E >$, V este o mulţime de vârfuri, iar $E \subseteq V \times V$ este o mulţime de muchii.

Un **graf orientat sau digraf** este o pereche notată $G =< V, E >$ în care fiecare muchie este o pereche notată $\{u,v\}$ unde arcul dintre nodurile u,v pleacă din u şi ajunge în v. În acest tip de graf, este permisă muchia ce are acelaşi nod ca destinaţie şi sursă.

Un **subgraf** este un graf $G =< V', E' >$ unde $V' \subseteq V$ iar E' este o submulţime a lui E, adică a muchiilor ce unesc nodurile din V'.

Un **graf parţial** este un graf $G =< V_1, E'' >$ în care $E'' \subseteq E$ iar $V_1 = V$.

Două vârfuri unite printr-o muchie se numesc **vârfuri adiacente**.

Convenţii:

- notaţia (a, b) va fi folosită pentru a delimita o muchie dintr-un graf neorientat; a şi b sunt două vârfuri diferite.
- notaţia $\{a, b\}$ va fi folosită pentru a delimita o muchie dintr-un graf orientat.

Un **drum** este o succesiune de muchii.

Lungimea unui drum este egală cu numărul muchiilor care îl constituie.

Un drum simplu va fi un drum în care nici un vârf nu se repetă.

Un **ciclu** este un drum simplu, cu excepţia primului şi a ultimului vârf, care coincid.

Un **ciclu simplu** este un drum de la un nod la el însuşi, drum ce conţine cel puţin alte două noduri intermediare.

Un **graf aciclic** este un graf neorientat fără cicluri.

Un **graf neorientat conex** este un graf în care între oricare două vârfuri există un drum.

Un **graf orientat este tare conex** dacă între oricare două vârfuri i şi j există un drum de la i la j şi un de la j la i.

Pentru un **graf neconex** se vor determina componentele sale conexe.

O componentă conexă este un subgraf conex minimal, adică un subgraf conex în care nici un vârf din subgraf nu este unit cu unul din afară printr-o muchie a grafului iniţial.

Un **arbore** este un *graf neorientat aciclic conex*. Altfel spus, un arbore este un graf neorientat în care există exact un drum între oricare două vârfuri.

Un **arbore liber** este un graf neorientat conex, aciclic.

Un **arbore cu radăcină** este un arbore în care un singur nod este desemnat ca rădăcină.

Un graf parţial care este arbore se numeşte **arbore parţial**.

Un graf aciclic, neorientat se numeşte **pădure**.

Vârfurilor unui graf neorientat li se pot ataşa informaţii numite uneori *valori*, iar muchiilor li se pot ataşa informaţii numite uneori *lungimi* sau *costuri*.

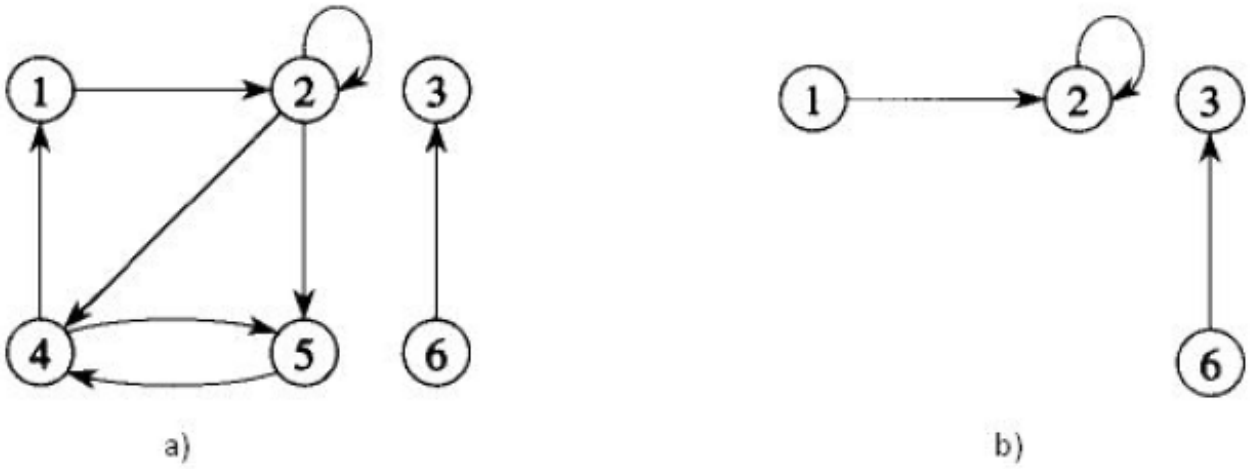

Figura 1. Exemplu de graf orientat. a) Un graf orientat $G = (V, E)$, $cu\ V = \{1,2,3,4,6,7\}$ şi $E = \{(1,2), (2,2), (2,4), (2,5), (4,1), (4,5), (5,4), (6,3)\}$. $(2,2)$ este o buclă ciclică. b) un subgraf al celui din a) format din nodurile $\{1,2,3,6\}$.

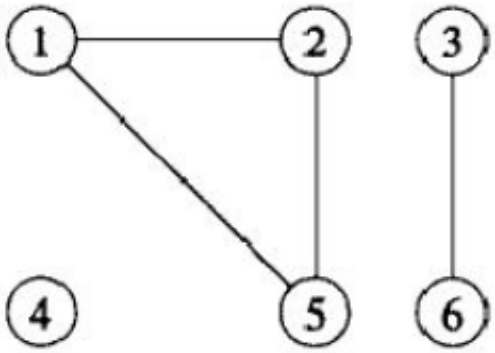

Figura 2. Un graf neorientat în care $V = \{1,2,3,4,5,6\}$ şi $E = \{(1,2), (1,5), (2,5), (3,6)\}$. Nodul 4 este izolat.

B. Reprezentarea grafurilor

1. Matrice de adiacenţă

Matricea de adiacenţă este o matrice A de $|V| \times |V|$ în care $A[i,j] = true$ dacă vârfurile i şi j sunt adiacente, iar $A[i,j] = false$ în caz contrar.

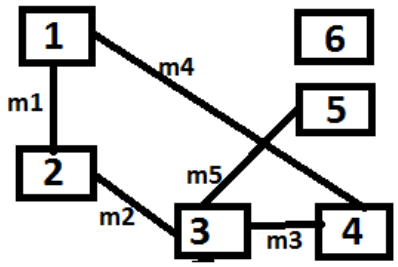

Figura 1: Un graf cu 6 noduri şi 5 muchii

Pentru graful din figura 1 matricea de adiacenţă este: $A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

2. Matrice de incidenţă

Matricea de adiacenţă este o matrice A de $|V| \times |E|$ în care $A[i, j]$ este numărul de câte ori nodul v_i întâlneşte muchia e_j.

Pentru graful din figura 1 matricea de incidenţă este:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

3. *Liste de adiacenţă*

Fiecărui vârf i i se ataşează lista de vârfuri adiacente lui.

4. *Lista de muchii*

Muchiile se scriu sub formă de perechi de noduri.
Pentru graful din figura 1 avem:
$m_1 : (1, 2)$
$m_2 : (2, 3)$
$m_3 : (3, 4)$
$m_4 : (1, 4)$
$m_5 : (3, 5)$

C. Algoritmi *greedy*

Algoritmii greedy (greedy = lacom) sunt, în general, simpli şi sunt folosiţi la probleme de optimizare, de exemplu: să se găsească cel mai scurt drum într-un graf.

1. *Arbori parţiali de cost minim*

Un **graf ponderat** este un graf în care muchiile au ponderi sau costuri pozitive.
Un **arbore parţial** al grafului G este un graf conex ce conţine toate nodurile din G şi este un arbore.
Un **arbore parţial de cost minim** este un arbore parţial în care costul total al ponderilor muchiilor este minim. Un graf poate avea chiar mai mulţi arbori parţiali de cost minim.
Găsirea un arbore parţial de cost minim plecând de la un graf G se poate face astfel: se caută toţi arborii parţiali şi apoi se compară ponderile totale ale acestora.
Prezentăm, în continuare, un algoritm pentru determinarea arborelui parţial de cost minim.

Algoritmul lui Kruskal

Algoritmul lui Kruskal găseşte arborele parţial de cost minim pentru un graf conex ponderat. Adică, găseşte submulţimea muchiilor care formează un arbore ce include toate vârfurile şi care este minimizat din punct de vedere al costului. Arborele parţial de cost minim poate fi construit astfel: se alege mai întâi muchia de cost minim, iar apoi se adaugă repetat muchia de cost minim nealeasăb anterior şi care nu formează cu precedentele un ciclu.
Dacă graful nu este conex, algoritmul va găsi o pădure parţială de cost minim(un arbore parţial de cost minim pentru foecare componentă conexă).
Algoritmul lui Kruskal este un exemplu de algoritm greedy.

Cum funcţionează:

Fie un graf conex $G =< V, E >$,

- creează o mulţime S care conţine toare muchiile din graf ordonate crescător după cost

- creează o pădure F unde fiecare vârf din graf este un arbore separat
- cât timp S este nevidă:
  - se selecteză o muchie de cost minim din S
  - daca acea muchie nu produce un ciclu în arbore, atunci adaugă muchia în pădure, combinând cei doi arbori într-unul singur
  - altfel, mergi la următoare muchie

La fiecare pas, avem un graf parţial ce reprezintă o pădure de componente conexe, obţinută din pădurea precedentă şi uniunea a două componente. Fiecare componentă conexă este la rândul ei un arbore parţial de cost minim pentru vârfurile pe care le conectează.

La sfârşitul algoritmului, pădurea are doar o componentă conexă care reprezintă un arbore parţial de cost minim al grafului.

2. Interclasarea optimă a şirurilor ordonate

Date de intrare: n şiruri(caz particular: se pot considera două şiruri S_1, S_2) S_1, S_2, ...,S_n ordonate crescător;

Date de ieşire: un şir S ordonat crescător ce conţine elementele din cele n şiruri.

Mod de realizare: se fac interclasări succesive de câte două şiruri

Problemă: determinarea ordinii optime în care trebuie efectuate interclasările aşa încât numărul total de deplasări să fie minim.

Exemplu:

Se dau şirurile: S_1, S_2, S_3 cu lungimile $q_1 = 30, S_2 = 20, S_3 = 10$.

Strategii de interclasare:

- S_1 cu S_2 iar rezultatul cu S_3 numărul total de deplasări este $(30 + 20) + (50 + 10) = 110$
- S_3 cu S_2 iar rezultatul cu S_1 numărul total de deplasări este $(10 + 20) + (30 + 30) = 90$
- S_1 cu S_3 iar rezultatul cu S_2 numărul total de deplasări este $(10 + 30) + (40 + 20) = 100$

Rezultă că numărul minim de deplasări este 90.

Cum funcţionează:

- se ataşează fiecărei strategii de interclasare un arbore binar în care valoarea viecărui vârf este dată de lungimea şirului pe care îl reprezintă;
- pentru un astfel de arbore oarecare A se defineşte *lungimea externă ponderată*: $L(A) = \sum_{i=1}^{n} a_i q_i$, a_i =adâncimea vârfului i;
- se vor interclasa mereu cele mai scurte şiruri disponibile la momentul curent;
- soluţia optimă este arborele pentru care lungimea ponderată este minimă.

Algoritmul greedy de interclasare optimă:

- se foloseşte un min-heap;
- fiecare element al min-heap-ului este o pereche (q, i), i =numărul unui vârf din arborele strategiei de interclasare iar q =lungimea şirului;
- proprietatea de min-heap se referă la q;
- un vârf i al arborelui va fi memorat astfel:
  - $L[i]$ =lungimea şirului reprezentat de vârf
  - $ST[i]$ =lungimea şirului corespondent fiului stâng
  - $DR[i]$ =lungimea şirului corespondent fiului drept

```
PROCEDURE INTEROPT(Q[1..n])
1.  ▷ construieste arborele strategiei greedy de interclasare
2.  ▷ a sirurilor de lungimi Q[i] = q_i, 1 ≤ i ≤ n
3.  H ← min − heap vid
4.  for i ← 1 to n do
5.      (Q[i], i) ⇒ H   ▷ inserează in min − heap
6.      LU[i] ← Q[i]
7.      ST[i] ← 0
8.      DR[i] ← 0
9.  for i ← n + 1 to 2n − 1 do
10.     (s, j) ⇐= H   ▷ extrage rădăcina lui H
11.     (r, k) ⇐= H   ▷ extrage rădăcina lui H
12.     ST[i] ← j
13.     DR[i] ← k
14.     LU[i] ← s + r
```

Figura 2: Algoritmul de interclasare optimă

II. PREZENTAREA LUCRĂRII DE LABORATOR

A. Algoritmul lui Kruskal

Codul de mai jos exemplifică un mod de implementare al algoritmului lui Kruskal:

- clasa Muchie: o muchie este caracterizată de cele 2 noduri care o formează şi costul asociat;

```
1  package kruskal;
2  public class Muchie {
3      private int x, y, cost;
4
5      public Muchie(int x, int y, int cost) {
6          this.x = x;
7          this.y = y;
8          this.cost = cost;
9      }
10
11     public int getX() {
12         return x;
13     }
14
15     public int getY() {
16         return y;
17     }
18
19     public int getCost() {
20         return cost;
21     }
22
23     public String toString()
24     {
25         return "("+x+","+y+","+cost+")";
26     }
27
28 }
```

- clasa AlgKruskal: se implementează algoritmul

```
1  package kruskal;
2  import java.util.*;
3  import java.io.*;
4  public class AlgKruskal {
5      public static void main(String[] args) throws IOException
6      {
7          Scanner sc=new Scanner(new FileReader("D:\\lucru\\algoritmica\\src\\graf"));
8          //numarul de varfuri
9          int n=sc.nextInt();
10
11         //numarul de muchii
12         //o muchie are cele 2 varfuri ce o unesc si costul asociat
13         int m=sc.nextInt();
14
15         //se citesc muchiile
16         Muchie[] muchii=new Muchie[m];
17         for(int i=0;i<m;i++)
18         {
19             int x=sc.nextInt();
20             int y=sc.nextInt();
21             int cost=sc.nextInt();
22             muchii[i]=new Muchie(x, y, cost);
23         }
24         sc.close();
25
26         //se sorteaza muchiile in functie de cost
27         for(int i=0;i<m-1;i++)
28             for(int j=i+1;j<m;j++)
29                 if(muchii[i].getCost()>muchii[j].getCost())
30                 {
31                     Muchie aux=muchii[i];
32                     muchii[i]=muchii[j];
33                     muchii[j]=aux;
34                 }
35
36         int[] comp=new int[n];
37         for(int i=0;i<n;i++)
38             comp[i]=i;
39
40         int sol=0, i=0;
41
42         // pentru fiecare muchie:
43         //se afiseaza nodurile ce o formeaza si costul asociat muchiei
44         while(sol<n-1 && i<m)
45         {
46             int x=muchii[i].getX()-1;
47             int y=muchii[i].getY()-1;
48             if(comp[x]!=comp[y])
49             {
```

```
50                System.out.println(muchii[i]);
51                sol++;
52                int z=comp[x], t=comp[y];
53
54                for(int j=0;j<n;j++)
55                    if(comp[j]==z)
56                        comp[j]=t;
57            }
58            i++;
59        }
60    }
61 }
```

- fişierul graf.txt ce conţine graful asupra căruia se aplică algoritmul

```
1 5 8
2 1 2 1
3 1 3 3
4 1 5 12
5 2 3 1
6 2 5 4
7 3 4 2
8 3 5 7
9 4 5 10
```

III. TEMĂ

Pentru fiecare din următoarele probleme se va scrie pseudocodul şi apoi codul în Java corespunzător.

Problema 1:

Un graf orientat $G = (V, E)$ este simplu conex dacă $u \rightsquigarrow v$ implică un drum simplu de la u la v pentru toate nodurile $u, v \in V$.

Scrieţi un algoritm care determină dacă un graf orientat este simplu conex.

Problema 2:

Un graf orientat $G = (V, E)$ se numeşte semiconex dacă pentru oricare 2 noduri $u, v \in V$ avem drum de la $u \rightsquigarrow v$ sau de la $v \rightsquigarrow u$.

Scrieţi un algoritm care determină dacă G este sau nu semiconex.

Problema 3:

Se dă un graf neorientat G.

Să se scrie un algoritm eficient care determină un arbore parţial al lui G al cărui vârf cu cea mai mare valoare este minimul peste toţi arborii parţiali ai lui G.