

Prob 1. Tema: luminare  
în conexitate, ciclice.

Ex 1

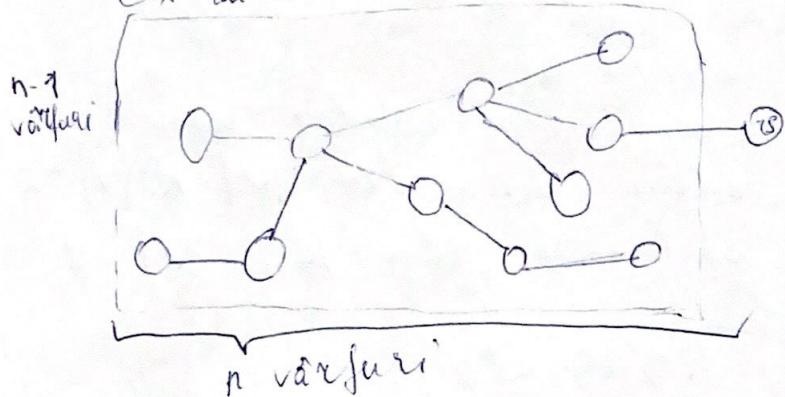


Un arbore este un graf neorientat conex și aciclic.

Demonstratie prin inducție după  $n$ .

• Un arbore cu  $n$  varfuri are  $n-1$  muchii.

Ex de arbore cu  $n-1$  muchii și  $n$  varfuri.



Inductie  $n=1$

Fie  $T$  un arbore cu  $n$  varfuri. Fie  $v$  varf terminal în  $T$  (cf. Lemă 1)  $\Rightarrow$

Lemă 1: Orice arbore  $T$  cu  $n > 1$  are cel puțin o varfuri terminale.

$\Rightarrow T - v$  este un arbore cu  $n-1$  varfuri (Lemă 2)

Lemă 2: Fie  $T$  un arbore cu  $n > 1$  varfuri și un varf terminal în  $T$ . Atunci  $T - v$  este arbore.

• Aplicăm ipoteza de inducție pentru  $T - v$

$$\Rightarrow |E(T-v)| = |V(T-v)| - 1 = (n-1) - 1 = n-2$$

$$\Rightarrow |E(T)| = |E(T-v)| + 1 = n-1$$

—————||—————

Demonstrat

$T$  este conex și aciclic.

Thm: Un graf neorientat conex cu  $n$  noduri și  $n-1$  muchii este aciclic maxim!

baza se obține o nouă muchie graful nu mai este aciclic.

Propozitie:  $G$  este conex.

(1\*)  $|E| \geq |V| - 1$  demonstrat prin inducție.

i)  $G$  este conex cu  $n=1$  sau  $n=2$  varfuri și  $n-1$  muchii.

șă se demonstreze că dacă grafurile satisfac relația (\*) și au mai puțin de  $n$  varfuri, se arată că obținând relația  $|E| \leq |V| - 1$ .

Eliminând o muchie arbitrară din  $T$ , se obține grafuri în  $k \geq 2$  componente conexe. fiecare componentă satisfac relația (\*), deci, astfel,  $T$  nu ar satisface relația (\*\*).

Prin inducție, dacă avem  $n$  muchii, atunci se poate scrie  $|E| = |V| - k \leq |V| - 2$ .

Holonomică muchia eliminată, obținem  $|E| \leq |V| - 1$ .

(2\*) Ipoteză: să se arate că  $(1*) \Rightarrow (2^*)$

$T$  este conex și conține un ciclu cu  $k$  varfuri

$v_1, v_2, \dots, v_k$ .

Fixăm  $T_k = (V_k, E_k)$  subgraful lui  $G$  format din ciclul respectiv.

Obs! ca  $|V_k| = |E_k| = k$ . Dacă  $k < |V|$ , atunci trebuie să existe un varf  $v_{k+1} \in V - V_k$  care să fie adiacent unui varf care face parte din un nou ciclu  $T$  este conex.

Dacă  $T$  are un nou ciclu  $T_{k+1} = (V_{k+1}, E_{k+1})$  ca fiind subgraf.

Aceeași varf  $v_{k+1} = v_k \in V \setminus \{v_{k+1}\}$  și

$E_{k+1} = E_k \cup \{v_i, v_{k+1}\}$ .

Obs!  $|V_{k+1}| = |E_{k+1}| = k+1$ . Dacă  $k+1 < n$  putem continua, definită  $T_{k+2}$ , în același mod,

d.ă. m.d. până obținem  $T_n = (V_n, E_n)$ , unde  $n = |V|$  și  $|E_n| = |V_n| = |V|$ .

Decoarece  $T_n$  este subgraf al lui  $T$ , avem  $E_n \subseteq E$  și de acei  $|E| \geq |V|$ . Așa că  $|E| = |V| - 1$

Astfel,  $T$  este ciclic.

Fie  $G = (V, E)$  norientat, conex Exz. □

$V = \{1, 2, \dots, n\}$  nr Max de muchii p/u seturi?

Noul graf  $G' = (V, E')$  ( $E' \subseteq E$ ) conex,  
rezulta  $d_G(1, i) = d_{G'}(1, i)$ , t.i.  $i \leq n$ , unde  
 $d_G(a, b)$  e distanta dintre muchiile  $a$  si  $b$  in  
graful  $G$ .

1. construim un arbore de acoperire min p/u  $G \rightarrow G^1$   
stergem celelalte muchii care nu fac parte  
din calitatea  $1 \text{ m}$

2. Ur maxime de muchii pe care putem sterge:  
 $|E| - (n-1)$ , stim ca un arbore de acoperire  
minim este un graf cu  $n$  noduri are  $n-1$   
muchii. — It deci maximul de stergere.

3. Astfel, nr max de muchii p/u seturi  
este  $|E| - (n-1)$ . ■

Contrafactual, p/u a găsi arborele minim  
putem folosi Algoritmul Kruskal

1:  $A \leftarrow \emptyset$  pentru că arborul minim

2: p/u fiecare varf  $v \in V[G]$  execută:

3: Formuleaza-Multimediu( $v$ )

4: Formuleaza multimediu din  $E$  în ordine crescătoare

5: a cestului  $w = d_G(1, i)$

a unei muchii  $(u, v) \in E$ , în ordere crescătoare

6: p/u fiecare muchie

a cestului execută: dacă  $\text{Gasește-Multimediu}(u) \neq \text{Gasește-Multimediu}(v)$  atunci:

7:  $A \leftarrow A \cup \{(u, v)\}$

8:  $\text{Formuleaza}(u, v)$

9: returnez  $A$ .

Fel Gasește Multimediu( $v$ ) redareață un șlem reprezentator din  
multimediu care îl conține pe  $v$ . Astfel, noi determinăm  
de 2 varfuri  $v$  le apără același arbore format de  
Gasește Multimediu( $v$ ) = Gasește Multimediu( $v$ )  
apoi combină arboreii umpluță

Ex 3  $G_1 = (V_1, E_1)$   $G_2 = (V_2, E_2)$  2 grafuri neorientate conexe  
 cu  $V_1 = V_2 = \{1, 2, \dots, n\}$ ;  $|E_1| = |E_2|$ ;

datorită BFS al  $G_1$  pornind din vîrful 1 este egal

cu el al  $G_2$  din  $V_1$ .

Echivalent și la DFS. (vecinii unui vîrf sunt parcurși în ordine crescătoare.)

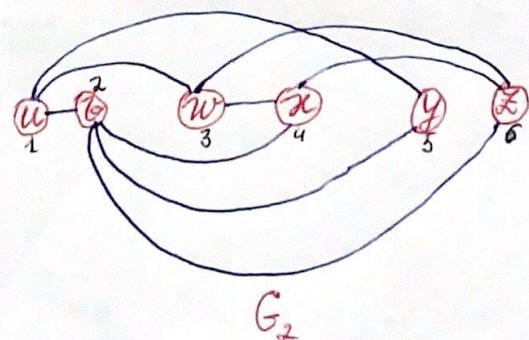
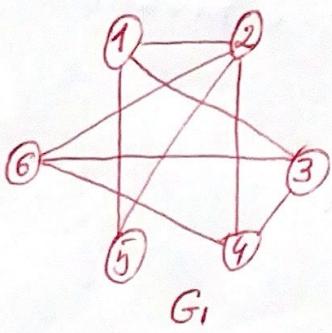
I. Idee: Dacă prin aplicarea algoritmilor BFS și DFS asupra acestor două grafuri, obțin parcurgerile  $L_{G_1}$  și  $L_{G_2}$  (în urma BFS), 2 multimi egale, cu același nr de elemente. Respectiv, în urma DFS,

$$L_{G_1} = L_{G_2}.$$

II. În același ordin de idei, dacă elementele multiorii sunt diferite, dar grafurile au aceeași structură, păcănește de la ideea de izomorfism, că vor fi echivalente.

I — II Fie 2 grafuri  $G_1 = (V_1, E_1)$  și  $G_2 = (V_2, E_2)$  izomorfe dacă există o bijecție  $f: V_1 \rightarrow V_2$ , astfel încât  $(u, v) \in E_1$ , dată și numai dacă  $(f(u), f(v)) \in E_2$ .

În alte cuvinte, putem redefinde vârfurile lui  $G_1$  pentru ca acestea să fie vârfuri din  $G_2$  cu multimi de vârfuri  $V_1 = \{1, 2, 3, 4, 5, 6\}$  și  $V_2 = \{u, v, w, x, y, z\}$ . Funcția din  $V_1$  în  $V_2$ , dată de  $f(1) = u$ ,  $f(2) = v$ ,  $f(3) = w$ ,  $f(4) = x$ ,  $f(5) = y$ ,  $f(6) = z$  este fapt bijecțivă cercată.



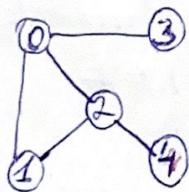
## Algoritmul DFS

$\text{DFS}(G)$

- 1: pentru fiecare  $vârf u \in V[G]$  execută:
- 2:  $\text{culoare}[u] \leftarrow \text{Alb}$
- 3:  $\pi[u] \leftarrow \text{NIL}$
- 4:  $\text{temp} \leftarrow 0$
- 5: pentru fiecare  $vârf u \in V[G]$  execută:
  - dacă  $\text{culoare}[u] = \text{Alb}$  atunci:
    - 6:  $\text{DFS-vizită}(u)$

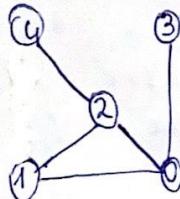
$\text{DFS-vizită}(u)$

- 1:  $\text{culoare}[u] \leftarrow \text{GRAI}$  *vîrstă* altă  $u$  tocmai a fost deschisă.
- 2:  $d[u] \leftarrow \text{temp} \leftarrow \text{temp} + 1$
- 3: pentru fiecare  $v \in \text{Adj}[u]$  execută: *explorare muchie  $(u, v)$*
- 4: daca  $\text{culoare}[v] = \text{Alb}$  atunci:
  - 5:  $\pi[v] \leftarrow u$
  - 6:  $\text{DFS-vizită}(v)$
- 7:  $\text{culoare}[u] \leftarrow \text{NEGRU}$  *vîrstă*  $u$ -negru, este fermintat.
- 8:  $f[u] \leftarrow \text{temp} \leftarrow \text{temp} + 1$ .



$\text{DFS}_1$

|   |
|---|
| 0 |
| 1 |
| 2 |
| 4 |
| 3 |



$\text{DFS}_2$

|   |
|---|
| 0 |
| 1 |
| 2 |
| 4 |
| 3 |

$$\text{DFS}_1(0) = \overline{\text{DFS}_2}$$

$$\text{DFS}_2(1) = 1$$

$$\text{DFS}_2(2) = 9$$

$$\text{DFS}_4(4) = 4$$

$$\text{DFS}_1(3) = 3$$

Funcție din  $\text{DFS}_1$  în  $\text{DFS}_2$ , este bijectivă.  
Deși e conform cu teorema izomorfismului,  
avem 2 grafuri echivalente.

## Algoritmul BFS:

(3)

1: BFS

2: creare coada Q

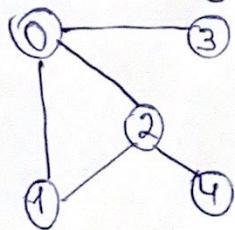
3: mărește și vizită și adaugă în Q

4: while Q are elemente:

5: sterg primul elem din Q și

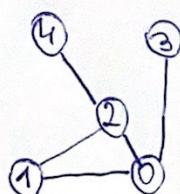
6: mărește /adăgă toate nodurile vizitate (vezi la u)

Același graf:



BFS<sub>1</sub>

0  
1  
2  
3  
4



BFS<sub>2</sub>

0  
1  
2  
3  
4

$\nexists$  et din BFS<sub>1</sub> în BFS<sub>2</sub>

nu este bijectivă.

Deci, conform Thm izomorfism  
avem 2 grafuri echivalente.

$$\text{BFS}_1(0) = 0$$

$$\text{BFS}_1(1) = 1$$

$$\text{BFS}_1(2) = 2$$

$$\text{BFS}_1(3) = 3$$

$$\text{BFS}_1(4) = 4$$

Ex 1.  
Ba.

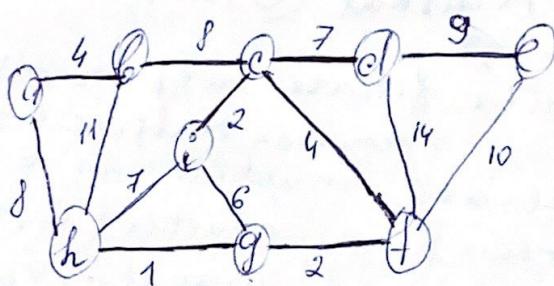
din exemplu avem:

$$L_1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

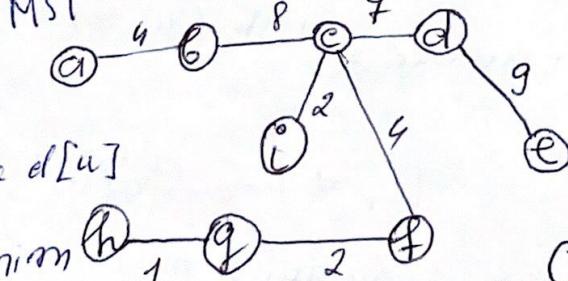
$$L_2 = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$L_1 = L_2$$

Ordinea calcul  $\rightarrow$  greedy  
Pf fiecare  $u \in L_2$  se calculeaza distanta  $d[u]$   
Se alege  $u$  sa fiecare  $u$  sa  
reapar in  $L_2$  cu  $d[u]$  minim



MST



C: 37

## AAM Kruskal ( $G, \omega$ )

- 1:  $H \leftarrow \emptyset$
- 2: pentru fiecare vârf  $v \in V[G]$  execută
  - 3: Formează Multime ( $v$ )
  - 4: sorteză muchiile din  $E$  în ordine crescătoare a costului lor
  - 5: pentru fiecare muchie  $(u, v) \in E$ , în ordine crescătoare a costului execută
    - 6: clasa  $\text{Coresponde} \text{Multime}(u) \neq \text{Coresponde} \text{Multime}(v)$  atunci:
      - 7:  $H \leftarrow H \cup \{(u, v)\}$
      - 8: Uneste  $(u, v)$
  - 9: returnarea  $H$ .

P<sub>p</sub> că  $L_1 \neq L_2$  costuri graf  $G$ .  $L_1 = \{a_1, a_2, \dots, a_n\}$  multimea de costuri

I de  $n=k$  (au același nr de elemente)  $L_2 = \{b_1, b_2, \dots, b_k\}$

afiind sorteate,  $a_1 < a_2 < \dots < a_n$  și  $b_1 < b_2 < \dots < b_k$   $\Rightarrow a_1 = b_1, a_2 = b_2, \dots, a_n = b_k$ .  
apărtinând aceluiași graf  $G$   $n = k$   $\alpha \times$

II de  $n \neq k$  ~~costuri~~  $|L_1| < |L_2|$

afiind sorteate, și putin  
un element evident  $\sum(L_1) < \sum(L_2)$  sumă  $L_2$   
se indică faptul că ~~nu~~ ~~nu~~ ~~nu~~ ~~nu~~ ~~nu~~ ~~nu~~ asociat  $L_2$   
nu este capabil pentru graful  $G$ .

Ex 5

Să ne imaginăm un graf cu  $n$  nodele de interese ale turistului. Costul muchiilor va fi calculat după formula euclidiană dată:  $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

1. Voi folosi algoritmul Dijkstra, deoarece el admite eiluri, astfel, pot calcula traseul cu întoarcere din locul că unde m-am poenit.

2. Ordinea de calcul  $\rightarrow$  greedy.  
Pentru fiecare  $u$  se calculează distanța estimată  $d[u]$ .

Voi alege la fiecare pas varful cu  $d[u]$  minim

3. Voi avea o funcție de complex  $O(n+m)$ , care parcurge graful în adâncime și calculează preventiv distanțele posibile dintre noduri după formula

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad \text{Calculare-Dist( )}$$

bFS ( $G, start$ )

```

1: initialize stiva S
2: initialize lista Viz
3: push nod_start pe S
4: while S not empty:
   nod_current = S.top()
   if nod_current is not in Viz:
      Viz.add(nod_current)
      d[nod_current] = Calculare-Dist()
5: Grevizitat = false
6: for each nod N, adjacent nod_current:
   if N not in Viz:
      S.push(N)
      Grevizitat = true
      break
7: if Grevizitat == false:
      S.pop(nod_current)
  
```

de modificat  
în ceea ce urmăruim  
graf min

loop.

```

1: calculateDist( p1 , p2 ) {
2:     dx = p2.x - p1.x ;
3:     dy = p2.y - p1.y ;
4:     return sqrt( dx * dx + dy * dy );
5: }

```

pe veet ole distante deja caleculat.

```

1: dijkstra() {
2:     for u ∈ V
3:         d[u] = ∞, tata[u] = 0
4:     d[s] = 0
5:     Q = V
6:     while Q ≠ ∅ ← of nriz
7:         u = extract-min(Q, d)
8:         for uv ∈ E → succ lui u
9:             if d[v] > d[u] + w(uv)
10:                d[v] = d[u] + w[uv]
11:                tata[v] = u
12:    return d, tata .

```

algoritmul are u v