

# Grafuri

SD 2017/2018

Tipul abstract Graf

Tipul abstract Digraf

Implementarea cu matrici de adiacență

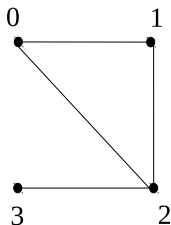
Implementarea cu liste de adiacență înlănțuite

Algoritmi de parcurgere (DFS, BFS)

Determinarea componentelor (tare) conexe

►  $G = (V, E)$

- $V$  mulțime de **vârfuri**
- $E$  mulțime de **muchii**; o **muchie** = o pereche neordonată de vârfuri distincte



$$V = \{0, 1, 2, 3\}$$

$$E = \{\{0, 1\}, \{0, 2\}, \{1, 2\}, \{2, 3\}\}$$

$$u = \{0, 1\} = \{1, 0\}$$



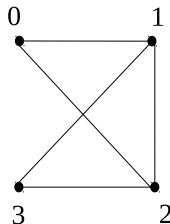
0,1 - **extremitățile** lui  $u$

$u$  este **incidentă** în 0 și 1

0 și 1 sunt **adiacente (vecine)**

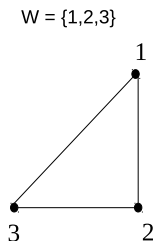
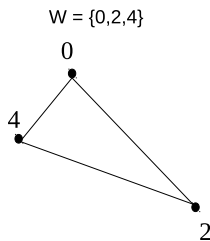
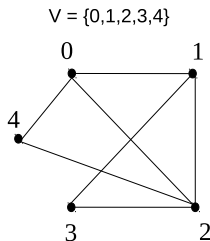
# Grafuri

- ▶ **Mers de la  $u$  la  $v$ :**  $u = i_0, \{i_0, i_1\}, i_1, \dots, \{i_{k-1}, i_k\}, i_k = v$   
3, {3,2}, 2, {2,0}, 0, {0,1}, 1, {1,3}, 3, {3,2}, 2
- ▶ **parcurs:** mers în care oricare două muchii sunt distincte
- ▶ **drum:** mers în care oricare două vârfuri sunt distincte
- ▶ **mers închis:**  $i_0 = i_k$
- ▶ **circuit** = mers închis în care oricare două vârfuri intermediare sunt distincte



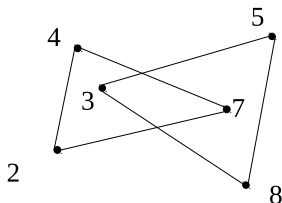
# Subgraf indus

- ▶  $G = (V, E)$  – graf,  $W$  – submulțime a lui  $V$
- ▶ Subgraf indus de  $W$ :  $G'(W, E')$ , unde  
 $E' = \{\{i, j\} \mid \{i, j\} \in E \text{ și } i \in W, j \in W\}$



# Grafuri - Conexitate

- ▶  $i R j$  dacă și numai dacă există drum de la  $i$  la  $j$
- ▶  $R$  este relație de echivalență
- ▶  $V_1, \dots, V_p$  clasele de echivalență
- ▶  $G_i = (V_i, E_i)$  subgraful indus de  $V_i$
- ▶  $G_1, \dots, G_p$  – componente conexe
- ▶ graf conex = graf cu o singură componentă conexă



$$V_1 = \{2, 4, 7\}$$

$$E_1 = \{\{2, 4\}, \{4, 7\}, \{2, 7\}\}$$

$$V_2 = \{3, 5, 8\}$$

$$E_2 = \{\{3, 5\}, \{5, 8\}, \{8, 3\}\}$$

# Tipul de date abstract **Graf**

- ▶ **obiecte:**

- ▶ grafuri  $G = (V, E)$ ,  $V = \{0, 1, \dots, n-1\}$

- ▶ **operații:**

- ▶ **grafVid()**

- ▶ intrare: nimic
    - ▶ ieșire: graful vid  $(\emptyset, \emptyset)$

- ▶ **esteGrafVid()**

- ▶ intrare:  $G = (V, E)$ ,
    - ▶ ieșire: true dacă  $G = (\emptyset, \emptyset)$ , false în caz contrar

- ▶ **insereazaMuchie()**

- ▶ intrare:  $G = (V, E)$ ,  $i, j \in V$
    - ▶ ieșire:  $G = (V, E \cup \{i, j\})$

- ▶ **insereazaVarf()**

- ▶ intrare:  $G = (V, E)$ ,  $V = \{0, 1, \dots, n-1\}$
    - ▶ ieșire:  $G = (V', E)$ ,  $V' = \{0, 1, \dots, n-1, n\}$

# Tipul de date abstract **Graf**

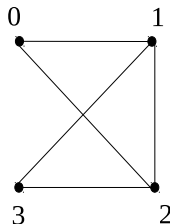
## ► eliminaMuchie()

- intrare:  $G = (V, E)$ ,  $i, j \in V$
- ieșire:  $G = (V, E - \{i, j\})$

## ► eliminaVarf()

- intrare:  $G = (V, E)$ ,  $V = \{0, 1, \dots, n-1\}$ ,  $k$
- ieșire:  $G = (V', E')$ ,  $V' = \{0, 1, \dots, n-2\}$

$$\{i', j'\} \in E' \Leftrightarrow (\exists \{i, j\} \in E) \ i \neq k, j \neq k,$$
$$i' = \text{if } (i < k) \text{ then } i \text{ else } i - 1,$$
$$j' = \text{if } (j < k) \text{ then } j \text{ else } j - 1$$





## ▶ listaDeAdiacenta()

- ▶ intrare:  $G = (V, E)$ ,  $i \in V$
- ▶ ieșire: lista vârfurilor adiacente cu  $i$

## ▶ listaVarfurilorAccesibile()

- ▶ intrare:  $G = (V, E)$ ,  $i \in V$
- ▶ ieșire: lista vârfurilor accesibile din  $i$

Tipul abstract Graf

Tipul abstract Digraf

Implementarea cu matrici de adiacență

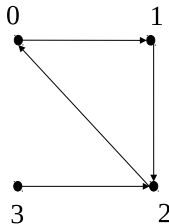
Implementarea cu liste de adiacență înlănțuite

Algoritmi de parcurgere (DFS, BFS)

Determinarea componentelor (tare) conexe

# Digraf (graf orientat)

- ▶  $D = (V, A)$ 
  - ▶  $V$  mulțime de vârfuri
  - ▶  $A$  mulțime de arce; un arc = o pereche ordonată de vârfuri distincte



$$V = \{0, 1, 2, 3\}$$

$$A = \{(0, 1), (2, 0), (1, 2), (3, 2)\}$$

$$a = (0, 1) \neq (1, 0)$$

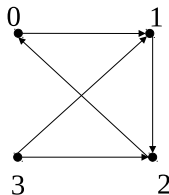


0 – sursa lui  $a$

1 – destinația lui  $a$

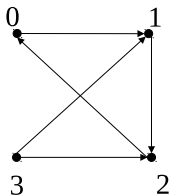
# Digraf

- ▶ **mers**:  $i_0, (i_0, i_1), i_1, \dots, (i_{k-1}, i_k), i_k$   
 $3, (3,2), 2, (2,0), 0, (0,1), 1, (1,2), 2, (2,0), 0$
- ▶ **parcurs**: mers în care oricare două arce sunt distincte
- ▶ **drum**: mers în care oricare două vârfuri sunt distincte
- ▶ **mers închis**:  $i_0 = i_k$
- ▶ **circuit** = mers închis în care oricare două vârfuri intermediare sunt distincte



# Digraf - Conexitate

- ▶  $i R j$  dacă și numai dacă există drum de la  $i$  la  $j$  și drum de la  $j$  la  $i$
- ▶  $R$  este relație de echivalență
- ▶  $V_1, \dots, V_p$  clasele de echivalență
- ▶  $G_i = (V_i, A_i)$  subdigraful indus de  $V_i$
- ▶  $G_1, \dots, G_p$  – componente tare conexe
- ▶ digraf tare conex = digraf cu o singură componentă tare conexă



$$V1 = \{0, 1, 2\}$$

$$A1 = \{(0, 1), (1, 2), (2, 0)\}$$

$$V2 = \{3\}$$

$$A2 = \emptyset$$

# Tipul de date abstract **Digraf**

- ▶ **obiecte**: digrafuri  $D = (V, A)$
- ▶ **operații**:
  - ▶ **digrafVid()**
    - ▶ intrare: nimic
    - ▶ ieșire: digraful vid  $(\emptyset, \emptyset)$
  - ▶ **esteDigrafVid()**
    - ▶ intrare:  $D = (V, A)$ ,
    - ▶ ieșire: true dacă  $D = (\emptyset, \emptyset)$ , false în caz contrar
  - ▶ **insereazaArc()**
    - ▶ intrare:  $D = (V, A)$ ,  $i, j \in V$
    - ▶ ieșire:  $D = (V, A \cup (i, j))$
  - ▶ **insereazaVarf()**
    - ▶ intrare:  $D = (V, A)$ ,  $V = \{0, 1, \dots, n-1\}$
    - ▶ ieșire:  $D = (V', A)$ ,  $V' = \{0, 1, \dots, n-1, n\}$

# Tipul de date abstract **Digraf**

## ► eliminaArc()

- intrare:  $D = (V, A)$ ,  $i, j \in V$
- ieşire:  $D = (V, A - (i, j))$

## ► eliminaVarf()

- intrare:  $D = (V, A)$ ,  $V = \{0, 1, \dots, n-1\}$ ,  $k$
- ieşire:  $D = (V', A')$ ,  $V' = \{0, 1, \dots, n-2\}$

$$\begin{aligned}\{i', j'\} \in A' &\Leftrightarrow (\exists \{i, j\} \in A) \ i \neq k, j \neq k, \\ i' &= \text{if } (i < k) \text{ then } i \text{ else } i - 1, \\ j' &= \text{if } (j < k) \text{ then } j \text{ else } j - 1\end{aligned}$$

# Tipul de date abstract **Digraf**

## ▶ listaDeAdiacentaExterioara()

- ▶ intrare:  $D = (V, A)$ ,  $i \in V$
- ▶ ieşire: lista vârfurilor destinate ale arcelor care pleacă din  $i$

## ▶ listaDeAdiacentaInterioara()

- ▶ intrare:  $D = (V, A)$ ,  $i \in V$
- ▶ ieşire: lista vârfurilor sursă ale arcelor care sosesc în  $i$

## ▶ listaVarfurilorAccesibile()

- ▶ intrare:  $D = (V, A)$ ,  $i \in V$
- ▶ ieşire: lista vârfurilor accesibile din  $i$



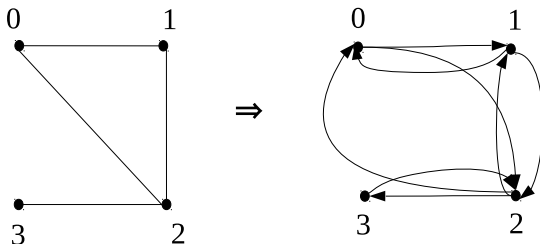
# Reprezentarea grafurilor ca digrafuri

$$G = (V, E) \implies D(G) = (V, A)$$

$$i, j \in E \implies (i, j), (j, i) \in A$$

► topologia este păstrată

- lista de adiacență a lui  $i$  în  $G$  = lista de adiacență exterioară (=interioară) a lui  $i$  în  $D$



Tipul abstract Graf

Tipul abstract Digraf

**Implementarea cu matrici de adiacență**

Implementarea cu liste de adiacență înlănțuite

Algoritmi de parcurgere (DFS, BFS)

Determinarea componentelor (tare) conexe

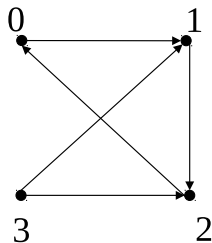
# Implementarea cu matrici de adiacență a digrafurilor

- ▶ reprezentarea digrafurilor

- ▶  $n$  numărul de vârfuri
- ▶  $m$  numărul de arce (opțional)
- ▶ o matrice ( $a[i,j] \mid 1 \leq i, j \leq n$ )  
 $a[i,j] = \text{if } (i,j) \in A \text{ then } 1 \text{ else } 0$
- ▶ dacă digraful reprezintă un graf, atunci  $a[i,j]$  este simetrică
- ▶ lista de adiacență exterioară a lui  $i \subseteq$  linia  $i$
- ▶ lista de adiacență interioară a lui  $i \subseteq$  coloana  $i$

# Implementarea cu matrici de adiacență

	0	1	2	3
0	0	1	0	0
1	0	0	1	0
2	1	0	0	0
3	0	1	1	0



# Implementarea cu matrici de adiacență

- ▶ operații

- ▶ digrafVid

- $n \leftarrow 0; m \leftarrow 0$

- ▶ insereazaVarf:  $O(n)$

- ▶ insereazaArc:  $O(1)$

- ▶ eliminaArc:  $O(1)$

# Implementarea cu matrici de adiacență

## ► eliminaVarf()

**Procedure** *eliminaVarf*( $a, n, k$ )

**begin**

**for**  $i \leftarrow 0$  **to**  $n - 1$  **do**

**for**  $j \leftarrow 0$  **to**  $n - 1$  **do**

**if**  $(i > k)$  **then**

$a[i - 1, j] \leftarrow a[i, j]$

**if**  $(j > k)$  **then**

$a[i, j - 1] \leftarrow a[i, j]$

$n \leftarrow n - 1$

**end**

**timp de execuție:**  $O(n^2)$

# Implementarea cu matrici de adiacență

## ► listaVarfurilorAccesibile()

**Procedure** *inchReflTranz*(*a*, *n*, *b*) // (Warshall, 1962)

**begin**

**for**  $i \leftarrow 0$  **to**  $n - 1$  **do**

**for**  $j \leftarrow 0$  **to**  $n - 1$  **do**

$b[i, j] \leftarrow a[i, j]$

**if**  $(i = j)$  **then**

$b[i, j] \leftarrow 1$

**for**  $k \leftarrow 0$  **to**  $n - 1$  **do**

**for**  $i \leftarrow 0$  **to**  $n - 1$  **do**

**if**  $(b[i, k] = 1)$  **then**

**for**  $j \leftarrow 0$  **to**  $n - 1$  **do**

**if**  $(b[k, j] = 1)$  **then**

$b[i, j] \leftarrow 1$

**end**

timp de executie:  $O(n^3)$

Tipul abstract Graf

Tipul abstract Digraf

Implementarea cu matrici de adiacență

**Implementarea cu liste de adiacență înlănțuite**

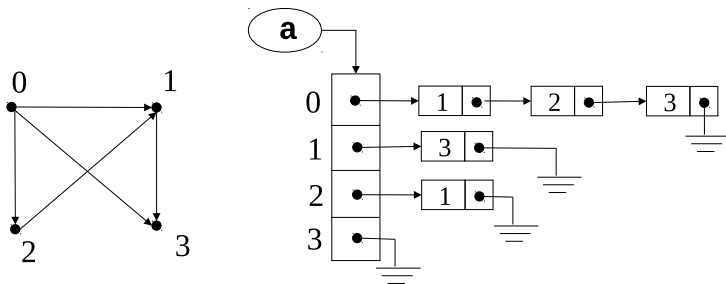
Algoritmi de parcurgere (DFS, BFS)

Determinarea componentelor (tare) conexe



# Implementarea cu liste de adiacență

- ▶ reprezentarea digrafurilor cu liste de adiacență exterioară

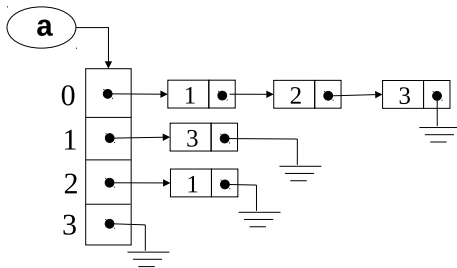


- ▶ un tablou  $a[0..n-1]$  de liste înlănțuite (pointeri)
- ▶  $a[i]$  este lista de adiacență exterioară corespunzătoare lui  $i$

# Implementarea cu liste de adiacență

## ► operații

- `digrafVid`
- `insereazaVarf`:  $O(1)$
- `insereazaArc`:  $O(1)$
- `eliminaVarf`:  $O(n + m)$
- `eliminaArc`:  $O(m)$



Tipul abstract Graf

Tipul abstract Digraf

Implementarea cu matrici de adiacență

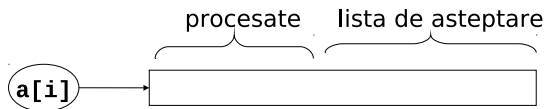
Implementarea cu liste de adiacență înlănțuite

Algoritmi de parcurgere (DFS, BFS)

Determinarea componentelor (tare) conexe

# Digrafuri: explorare sistematică

- ▶ se gestionează două mulțimi
  - ▶  $S$  = mulțimea vârfurilor vizitate deja
  - ▶  $SB \subseteq S$  submulțimea vârfurilor pentru care există șanse să găsim **vecini nevizitați** încă
- ▶ lista de adiacență (exterioară) a lui  $i$  este divizată în două:



# Digrafuri: explorare sistematică

- ▶ pasul curent
  - ▶ citește un vârf  $i$  din  $SB$
  - ▶ extrage un  $j$  din lista de “așteptare” a lui  $i$  (dacă este nevidă)
  - ▶ dacă  $j$  nu este în  $S$ , atunci îl adaugă la  $S$  și la  $SB$
  - ▶ dacă lista de “așteptare” a lui  $i$  este vidă, atunci elimină  $i$  din  $SB$
- ▶ inițial
  - ▶  $S = SB = \{i_0\}$
  - ▶ lista de “așteptare a lui  $i$ ” = lista de adiacenta a lui  $i$
- ▶ terminare  $SB = \emptyset$

# Digrafi: explorare sistematică

**Procedure** *explorare*(*a*, *n*, *i0*)

**begin**

**for**  $i \leftarrow 0$  **to**  $n - 1$  **do**

$p[i] \leftarrow a[i]$

$SB \leftarrow (i0)$

$viziteaza(i0)$ ;  $S \leftarrow (i0)$

**while** ( $SB \neq \emptyset$ ) **do**

$i \leftarrow citeste(SB)$

**if** ( $p[i] = NULL$ ) **then**

$SB \leftarrow SB - \{i\}$

**else**

$j \leftarrow p[i] \rightarrow varf$

$p[i] \leftarrow p[i] \rightarrow succ$

**if** ( $j \notin S$ ) **then**

$SB \leftarrow SB \cup \{j\}$

$viziteaza(j)$

$S \leftarrow S \cup \{j\}$

**end**

# Explorare sistematică: complexitate

## Teorema

*În ipoteza că operațiile peste  $S$  și  $SB$  precum și  $viziteaza()$  se realizează în  $O(1)$ , complexitatea timp, în cazul cel mai nefavorabil, a algoritmului explorare este  $O(n + m)$ .*

# Explorarea DFS (*Depth First Search*)

- ▶  $SB$  este implementată ca stivă

$$SB \leftarrow (i0) \Leftrightarrow SB \leftarrow stivaVida()$$

$$push(SB, i0)$$

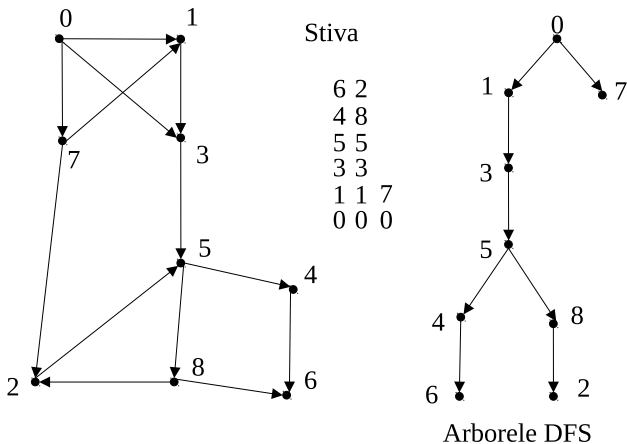
$$i \leftarrow citeste(SB) \Leftrightarrow i \leftarrow top(SB)$$

$$SB \leftarrow SB - \{i\} \Leftrightarrow pop(SB)$$

$$SB \leftarrow SB \cup \{j\} \Leftrightarrow push(SB, j)$$



# Explorarea DFS: exemplu

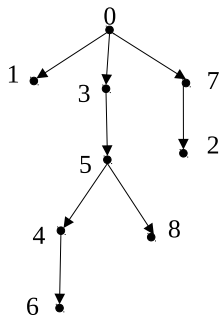
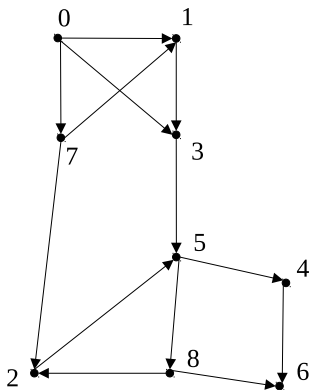


# Explorarea BFS (*Breadth First Search*)

- $SB$  este implementată ca o coadă

$$SB \leftarrow (i0) \Leftrightarrow SB \leftarrow coadaVida();$$
$$insereaza(SB, i0)$$
$$i \leftarrow citeste(SB) \Leftrightarrow citeste(SB, i)$$
$$SB \leftarrow SB - \{i\} \Leftrightarrow elimina(SB)$$
$$SB \leftarrow SB \cup \{j\} \Leftrightarrow insereaza(SB, j)$$

# Explorarea BFS: exemplu



Arborele BFS

Tipul abstract Graf

Tipul abstract Digraf

Implementarea cu matrici de adiacență

Implementarea cu liste de adiacență înlănțuite

Algoritmi de parcurgere (DFS, BFS)

Determinarea componentelor (tare) conexe

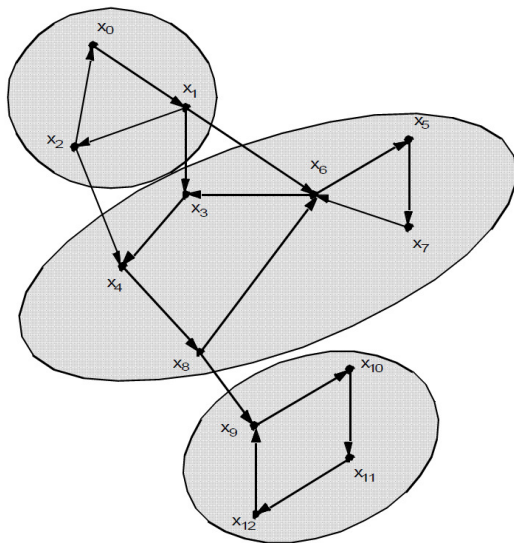
# Determinarea componentelor conexe (grafuri neorientate)

```
Function CompConexeDFS(D)  
begin  
  for  $i \leftarrow 0$  to  $n - 1$  do  
     $culoare[i] \leftarrow 0$   
   $k \leftarrow 0$   
  for  $i \leftarrow 0$  to  $n - 1$  do  
    if ( $culoare[i] = 0$ ) then  
       $k \leftarrow k + 1$   
      DfsRecCompConexe( $i, k$ )  
  return  $k$   
end
```

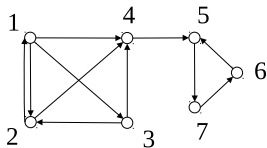
# Determinarea componentelor conexe (grafuri neorientate)

```
Procedure DfsRecCompConexe(i, k)  
begin  
    culoare[i]  $\leftarrow$  k  
    for (fiecare vârf j în listaDeAdiac(i)) do  
        if (culoare[j] = 0) then  
            DfsRecCompConexe(j, k)  
end
```

# Componente tare conexe (digrafuri)

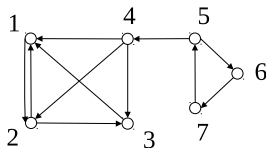


# Componente tare conexe: exemplu



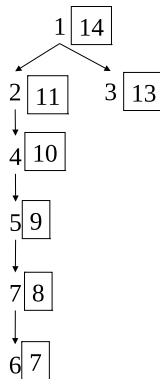
D

$1 \rightarrow (2, 3, 4)$   
 $2 \rightarrow (1, 4)$   
 $3 \rightarrow (2, 4)$   
 $4 \rightarrow (5)$   
 $5 \rightarrow (7)$   
 $6 \rightarrow (5)$   
 $7 \rightarrow (6)$



$D^T$

1	4	5
↓		↓
2		6
↓		↓
3		7





# Determinarea componentelor tare conexe

```
Procedure DfsCompTareConexe(D)  
begin  
  for  $i \leftarrow 0$  to  $n - 1$  do  
     $culoare[i] \leftarrow 0$   
     $tata[i] \leftarrow -1$   
     $timp \leftarrow 0$   
    for  $i \leftarrow 0$  to  $n - 1$  do  
      if ( $culoare[i] = 0$ ) then  
        DfsRecCompTareConexe( $i$ )  
end
```

# Determinarea componentelor tare conexe

```
Procedure DfsRecCompTareConexe(i)  
begin  
    timp  $\leftarrow$  timp + 1  
    culoare[i]  $\leftarrow$  1  
    for (fiecare vârf j in listaDeAdiac(i)) do  
        if (culoare[j] = 0) then  
            tata[j]  $\leftarrow$  i  
            DfsRecCompTareConexe(j)  
    timp  $\leftarrow$  timp + 1  
    timpFinal[i]  $\leftarrow$  timp  
end
```

# Determinarea componentelor tare conexe

Notăție:  $D^T = (V, A^T)$ ,  $(i, j) \in A \Leftrightarrow (j, i) \in A^T$

**Procedure** *CompTareConexe*( $D$ )

**begin**

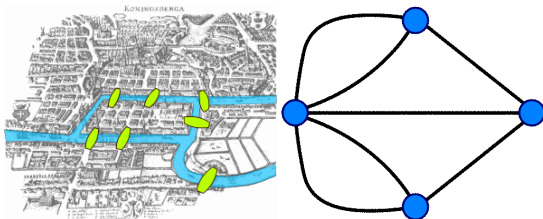
1. *DFSCompTareConexe*( $D$ )
2. calculează  $D^T$
3. *DFSCompTareConexe*( $D^T$ ) dar considerând în bucla *for* principală vârfurile în ordinea descrescătoare a timpilor finali de vizitare *timpFinal*[ $i$ ]
4. returnează fiecare arbore calculat la pasul 3 ca fiind o componentă tare conexă separată

**end**

# Determinarea componentelor tare conexe: complexitate

- ▶  $DFSCompTareConexe(D)$ :  $O(n + m)$
- ▶ calculează  $D^T$ :  $O(m)$
- ▶  $DFSCompTareConexe(D^T)$ :  $O(n + m)$
- ▶ Total:  $O(n + m)$

**Problema celor șapte poduri din Königsberg** (1736): realizarea unui drum prin oraș pornind dintr-o zonă pentru a traversa cele 7 poduri o singură dată?



Zonele: vârfuri, podurile: muchii

Este posibil să alegem un vârf, să parcurgem muchiile și să ne întoarcem în vârful ales, acoperind toate muchiile o dată?

- ▶ Algoritmică, probleme de drum, rețele de calculatoare (rutare), genomică (rețele de aliniere, asamblarea genomului), *multi-relational data mining*, cercetări operaționale (planificare), inteligență artificială (satisfacerea restricțiilor), etc.
- ▶ Motorul de căutare Google: algoritmul *PageRank* - pentru a determina cât de importantă este o anumită pagină
- ▶ Sistem informațional geografic (*GIS*): *Google Maps*, *Bing Maps*
- ▶ Rețele sociale

