M. Caramihai, © 2020

# STRUCTURI DE DATE & ALGORITMI

# CURS 6

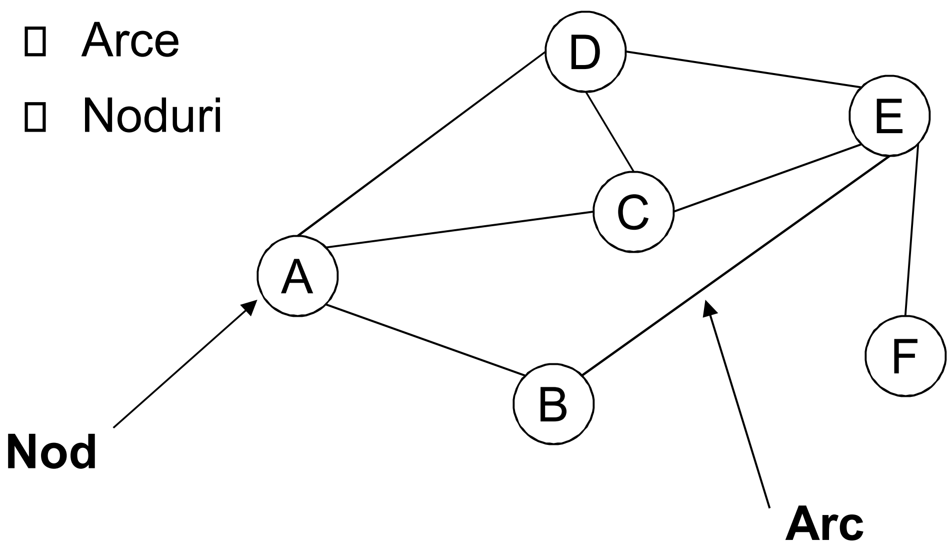
# Grafuri (1)

# Grafuri

- **Definiție**: o structura de date formata din **noduri** si **arce** ce leaga nodurile intre ele

→ mai **simplu**: modelarea unui set de conexiuni

- Instrument foarte util in modelarea problemelor.
- Sunt compuse din:
  - Arce
  - Noduri

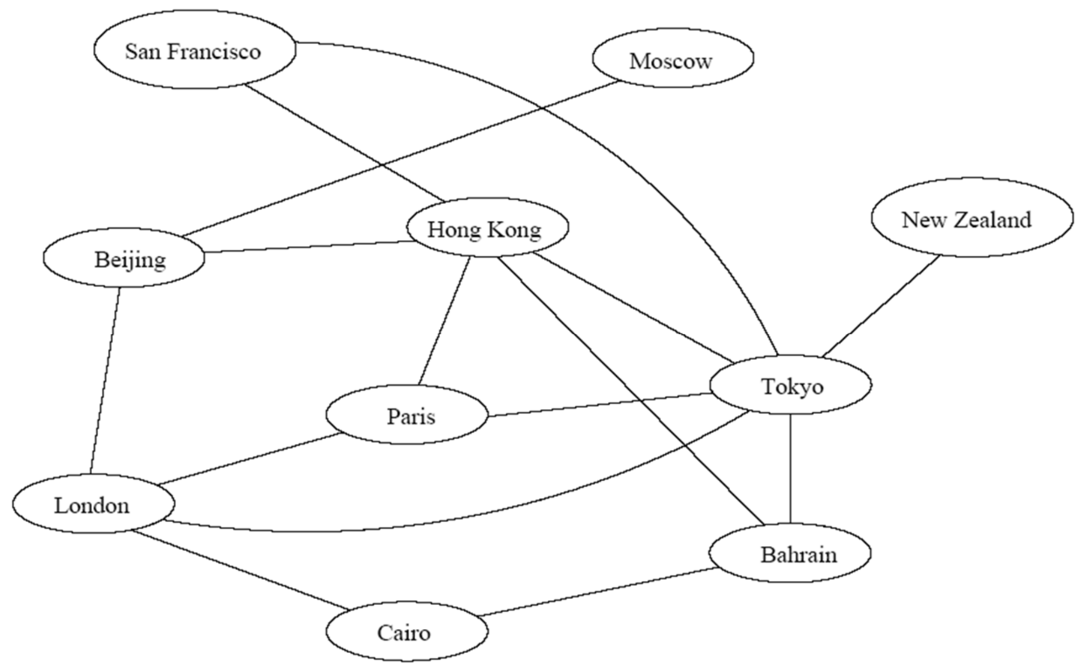



**Nodurile** pot fi considerate localitati.

**Arcele** reprezinta conexiunile (drumurile) de acces.

# Exemplu de graf

Sistem de control al zborului




- Fiecare nod reprezinta un oras
- Fiecare arc reprezinta zborul direct dintre doua orase
- O cerere de zbor direct devine o verificare a existentei unui zbor direct.
- Fiecarui arc i se poate asocia un “cost” (**graf ponderat**): astfel se poate determina zborul cl mai ieftin intre A si B

# Terminologie…

- Un *graf* este un obiect matematic format din două mulţimi: **G = (V, E),** în care **V** este o *mulţime de vârfuri (noduri)*, iar **E** – o *mulţime de muchii* (sau *arce*).

- O muchie de la varful *a* la varful *b* este notata cu perechea ordonata (*a*, *b*), daca graful este ***orientat***, si cu multimea {*a*, *b*}, daca graful este ***neorientat***.

- Doua varfuri unite printr-o muchie se numesc ***adiacente***. Un drum este o succesiune de muchii de forma

  (*a*1, *a*2), (*a*2, *a*3), ..., (*an*-1, *an*)

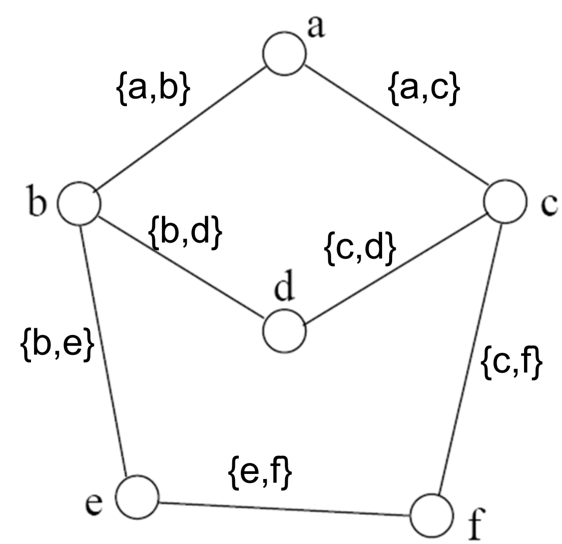
  sau de forma

  {*a*1, *a*2}, {*a*2, *a*3}, ..., {*an*-1, *an*}

dupa cum graful este **orientat** sau **neorientat**.

# Exemplu

- Graf **neorientat**
  - Un graf neorientat este specificat prin perechea (V,E), asa cum a fost definita anterior



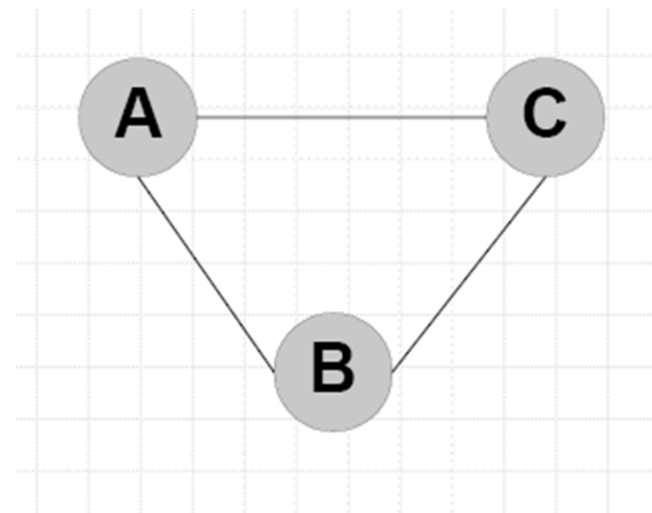

$$V = \{a, b, c, d, e, f\}$$

$$E = \{\{a, b\}, \{a, c\}, \{b, d\}, \{c, d\}, \{b, e\}, \{c, f\}, \{e, f\}\}$$

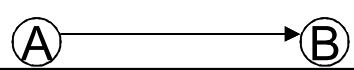
# Terminologie… (cont)

1. Daca *A* si *B* sunt legate printr'un arc, ele vor fi adiacente
   - ***A*** si ***B*** sunt varfurile arcului {***A, B***}
2. Daca un arc *e* este conectat la *v*, atunci se poate spune ca *v* este incident la *e*. Reciproc, arcul *e* este incident la *v*.

   $\{\boldsymbol{v_1}, \boldsymbol{v_2}\} = \{\boldsymbol{v_2}, \boldsymbol{v_1}\}^*$




**Graf orientat**: arcele sunt directionate. Acest lucru presupune ca $\{v_1,v_2\} \neq \{v_2,v_1\}$ . Grafurile orientate sunt desenate cu ajutorul unor sageti:



# Terminologie... (cont)

- Arcul **`(u, v)`** este *incident din* (*pleaca din*) **`u`** si este *incident in* (*intra in*) **`v`**.
- *Gradul unui varf* **`deg(v)`** intr-un graf neorientat, reprezinta numarul muchiilor incidente in acel varf. Un *varf izolat* este un varf avand gradul 0.
- *Gradul interior* (sau *gradul de intrare*) **`indeg(v)`** a unui vârf a unui graf orientat este egal cu numărul arcelor care intră în acel vârf. Un vârf cu gradul interior 0 este un *vârf de ieşire* (*sursă*).
- *Gradul exterior* (sau *gradul de iesire*) **`outdeg(v)`** a unui varf a unui graf orientat este egal cu numarul arcelor care ies din acel varf. Un varf cu gradul exterior 0 este un *varf de intrare* (*puţ*). *Ordinul* unui graf este egal cu numarul de varfuri al grafului: **`n = |V|.`**

*Dimensiunea* unui graf este egală cu numărul de muchii (arce) ale grafului: **`m = |E|.`**

# Terminologie… (cont)

- Drum:
  - Elementar: noduri distincte
  - Ciclu: nod initial ≡ nod final
  - Bucla: ciclu de lungime 1

- **Drum Hamiltonian**: drum elementar ce trece prin toate varfurile grafului
- **Drum Eulerian**: drum elementar ce trece prin toate varfurile grafului o singura data

# Terminologie... (cont)

- Un *drum* de lungime p intre două varfuri a si b este o succesiune de varfuri : $v_0, v, ..., v_p$, cu $v_0 = a$ si $v_p = b$ si $\{v_{i-1}, v_i\} \in E$ pentru $i=1:p$. Drumul contine atat arcele $(v_0,v_1), ..., (v_{p-1},v_p)$ cat si nodurile $v_0,...,v_p$. Daca exista un drum de la **a** la **b**, atunci **b** este *accesibil* din **a** ($a \to b$).
- Un *drum elementar* intr-un graf orientat are toate nodurile de pe el distincte. Un *ciclu* este un drum ($v_0, v_1, \dots, v_p$) în care $v_0 = v_p$. O *bucla* (**a,a**) este un ciclu de lungime 1. Un *ciclu elementar* are toate nodurile distincte (exceptand varful de plecare) si nu contine bucle. Un *graf aciclic* (*padure*) nu contine cicluri.

# Terminologie... (cont)

- Un graf neorientat este *conex*, dacă oricare două vârfuri pot fi unite printr-un drum. Un graf conex aciclic este *arbore liber*.
- *Componentele conexe* ale unui graf neorientat sunt clasele de echivalenţă ale varfurilor prin relaţia "*este accesibil din*". Un graf neorientat este conex, dacă are o singură componentă conexa.
- Un graf orientat este *tare conex* daca, pentru oricare două vârfuri **a** şi **b**, a este accesibil din **b** şi **b** este accesibil din **a**.
- Graful $G'=(V',E') \subseteq G$ este *subgraf* al grafului $G=(V,E)$, dacă $V' \subseteq V$ şi $E' \subseteq E$.
- *Subgraful indus* de multimea de vârfuri $V' \subseteq V$ este graful: $G'=(V',E')$ în care: $E' = \{(u,v) \in E : u,v \in V'\}$
- Graful $G'=(V,E')$ este un *graf partial* (sau *subgraf de acoperire*) al grafului $G=(V,E)$ daca $E' \subseteq E$.

# Terminologie… (cont)

- Intr-un *graf bipartit*, multimea varfurilor $\mathbf{V}$ se partiţionează $\mathbf{V} = \mathbf{V_1} \vee \mathbf{V_2}$, $\mathbf{V_1} \wedge \mathbf{V_2} = \varnothing$ astfel încat $(\mathbf{u}, \mathbf{v}) \in \mathbf{E} \Rightarrow \mathbf{u} \in \mathbf{V_1}, \mathbf{v} \in \mathbf{V_2}$ sau $\mathbf{u} \in \mathbf{V_1}, \mathbf{v} \in \mathbf{V_2}$.
- Pentru un graf neorientat cu $\mathbf{m}$ muchii: $\sum_{\mathbf{v} \in \mathbf{V}} \mathbf{deg(v)} = \mathbf{2m}$
- Intr-un graf orientat cu $\mathbf{m}$ arce $\sum_{\mathbf{v} \in \mathbf{V}} \mathbf{in\,deg(v)} = \sum_{\mathbf{v} \in \mathbf{V}} \mathbf{out\,deg(v)} = \mathbf{m}$
- Intr-un graf $\mathbf{G}$, următoarele conditii sunt echivalente:
  - **G** este conex aciclic
  - **G** este *aciclic maximal* (prin adaugarea unei muchii apare un ciclu)
  - **G** este *conex minimal* (prin stergerea unei muchii graful isi pierde conexitatea).

# Graful social (1)

- Un **GS** contine relatii de prietenie (muchii) intr'un grup de n persoane (varfuri)

**Observatie**: o relatie de prietenie este simetrica → 2 varfuri ce nu sunt legate prin muchii sunt dusmani

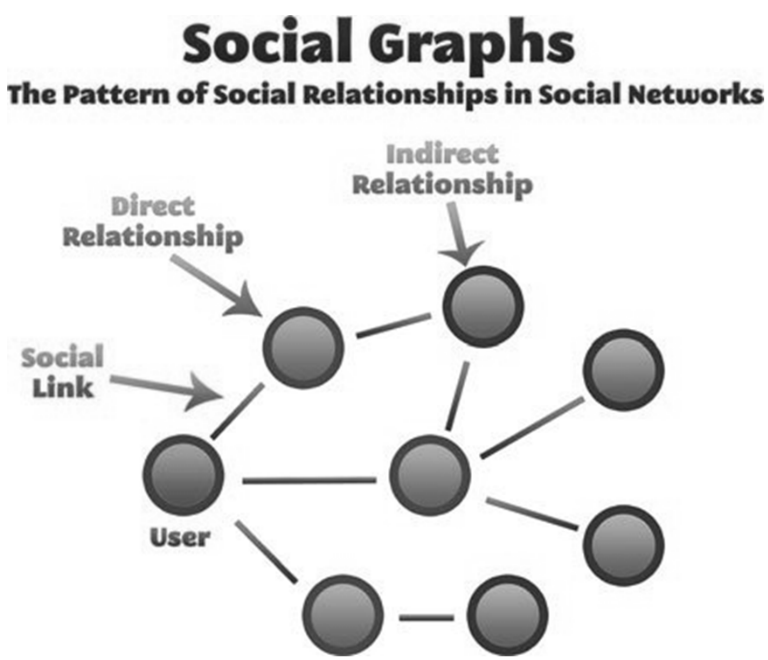
**Probleme**

1. In raport cu graful:
   - Sunt 2 personae conectate
   - Care este ciclul de prieteni?
   - Care este cel mai mare ciclu de prieteni?
2. In raport cu distanta
   - Care este distanta dintre persoana A si persoana B?
   - Care este distanta medie intre 2 personae din retea?
   - Care este diametrul retelei (= distanta cea mai mare dintre 2 noduri?)

# Graful social (2)

**Provocari:**

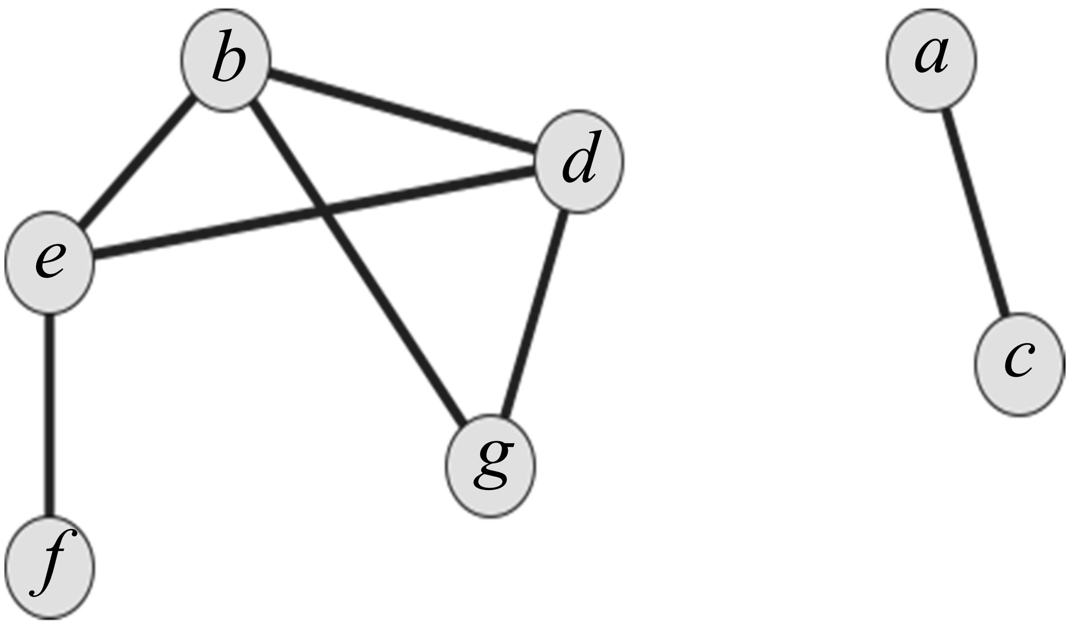
- Analiza de centralitate (i.e. “cea mai importanta persoana din retea”)
- Pozitia / rolul unei persoane in grup
- Dinamica culturala a grupului (prin analiza difuzarii informatiei in cadrul grupului)




Preluat dupa: Social Networks and Recommender Systems: A World of Current and Future Synergies
Kanna Al Falahi Nikolaos Mavridis Y. Atif, 2014

# Reprezentari (1)

Varfurile *a* si *c* sunt ***adiacente***; varfurile *f* si *g* nu sunt adiacente.

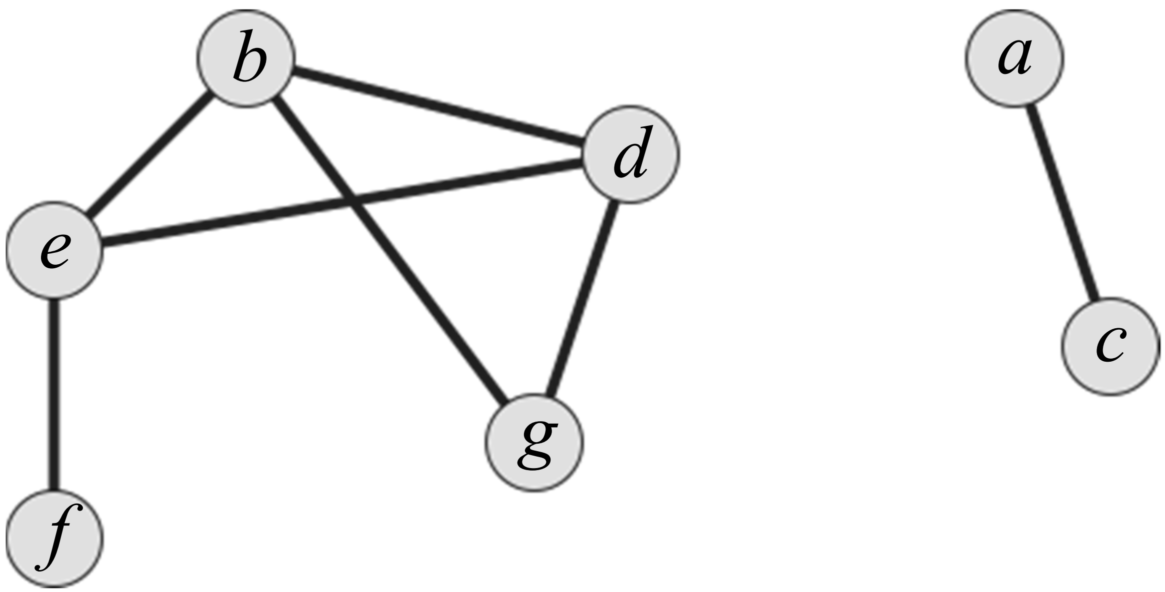



Varful *b* si arcul (*b*, *g*) sunt ***incidente***; Varful *e* si arcul (*d*, *g*) nu sunt incidente. ***Vecinatatea*** varfului *b* este setul $N(b) = \{d, e, g\}$. ***Gradul*** nodului b este 3  ($\deg(b) = 3$).

# Reprezentari (2)

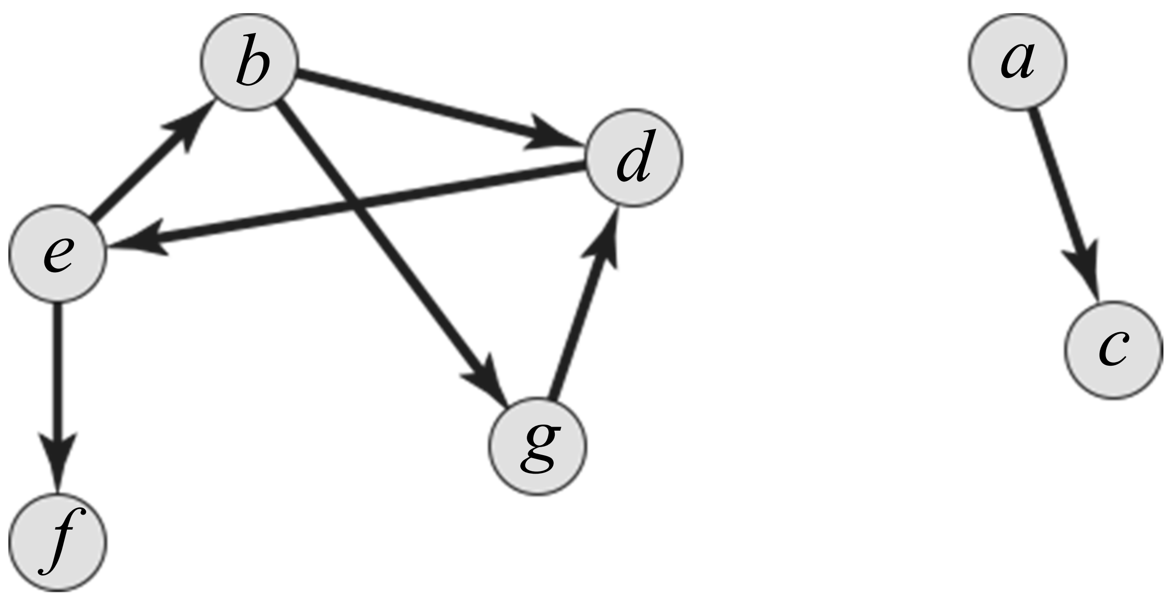
Intr'un graf neorientat, arcele sunt ***perechi neordonate*** ($v$, $w$).

Cu alte cuvinte ($a$, $c$) si ($c$, $a$) sunt exprimari ale unui aceluiasi arc (nu exista determinare a sensului).



# Reprezentari (3)

- Intr’un graf orientat, arcele reprezinta ***perechi ordonate*** (*v*, *w*).
- Cu alte cuvinte, arcele (*v*, *w*) si (*w*, *v*) sunt diferite.
- Arcul (*a*, *c*) este orientat de la a la c.
- Nodul *a* este **originea** lui (*a*, *c*); nodul c este **destinatia** lui (*a*, *c*).

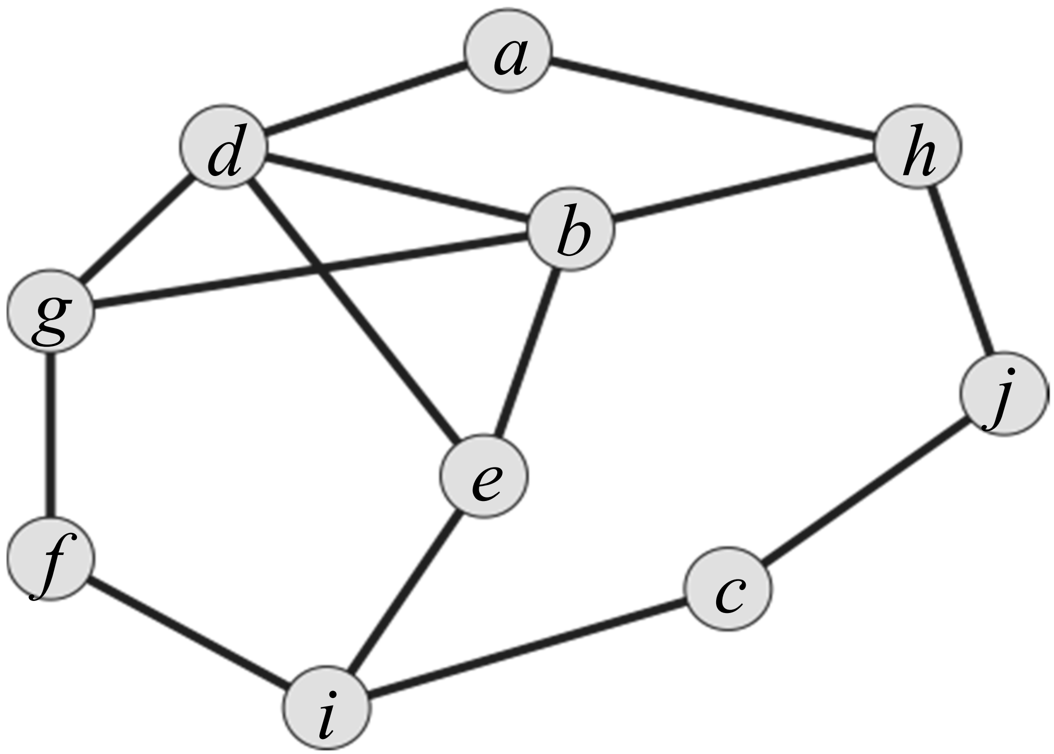


# Reprezentari (4)

Un drum:

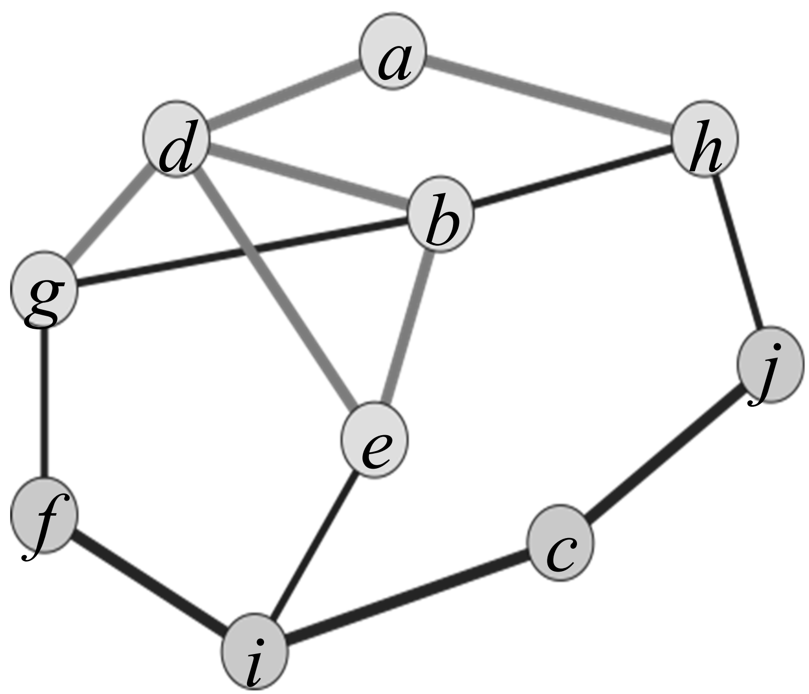
$P = (v_0, v_1, \ldots, v_k)$ a.i., pentru $1 \leq i \leq k$, arcele $(v_{i-1}, v_i) \in E$.

Drumul $P$ este ***simplu*** daca nici un nod nu apare de mai multe ori pe drumul P.



# Reprezentari (5)

$P_1 = (g, d, e, b, d, a, h)$ nu este simplu.



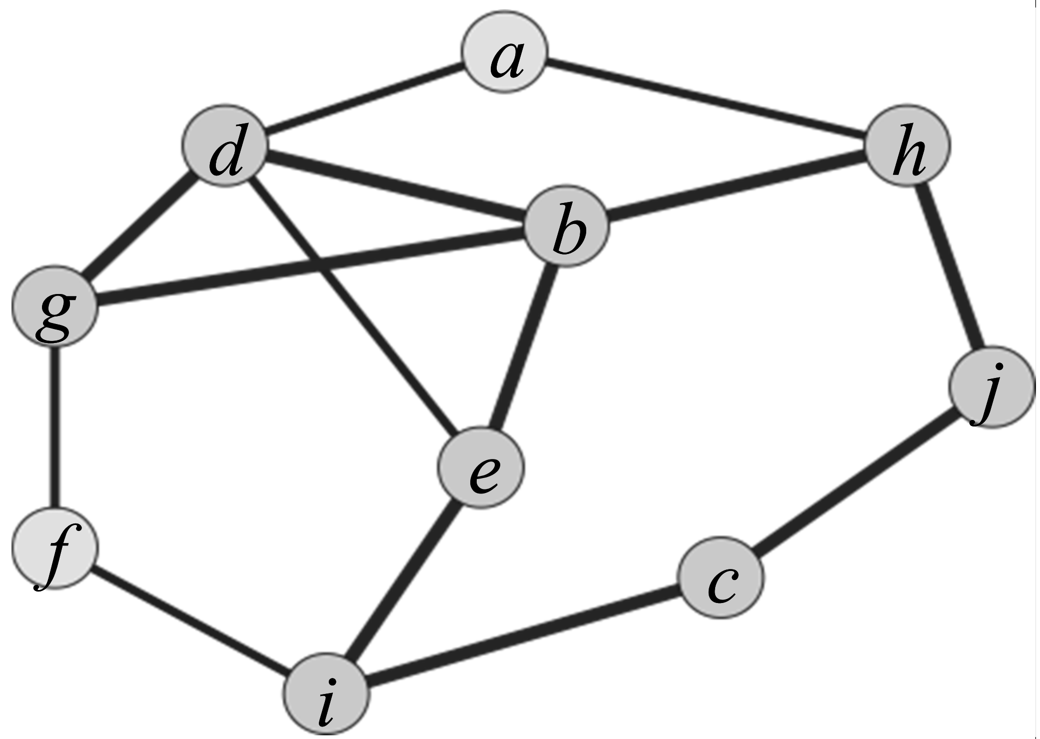

$P_2 = (f, i, c, j)$ este simplu.

# Reprezentari (6)

$C_1 = (a, h, j, c, i, f, g, d)$ este simplu.

Un ***ciclu*** este o secventa de noduri:
$C = (v_0, v_1, \ldots, v_{k-1})$ a.i. pentru $0 \le i < k$, arcele $(v_i, v_{(i+1) \bmod k}) \in E$.

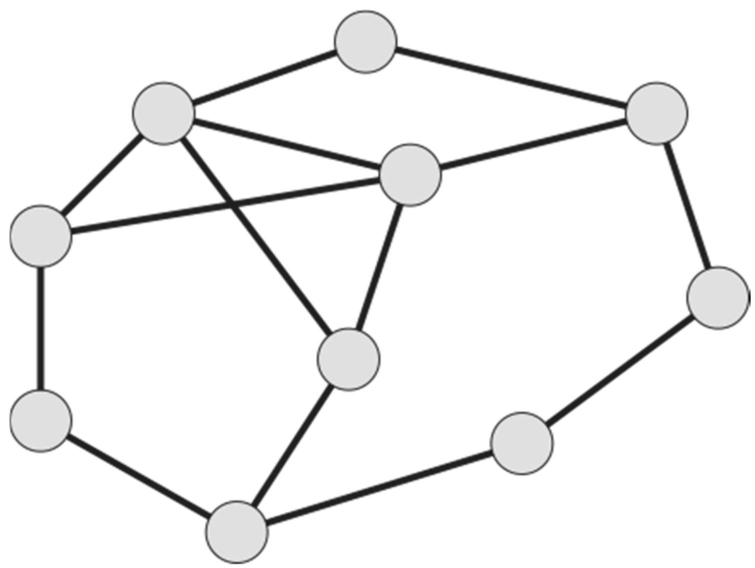
Ciclul $C$ este ***simplu*** daca drumul $(v_0, v_1, \ldots, v_{k-1})$ este simplu.



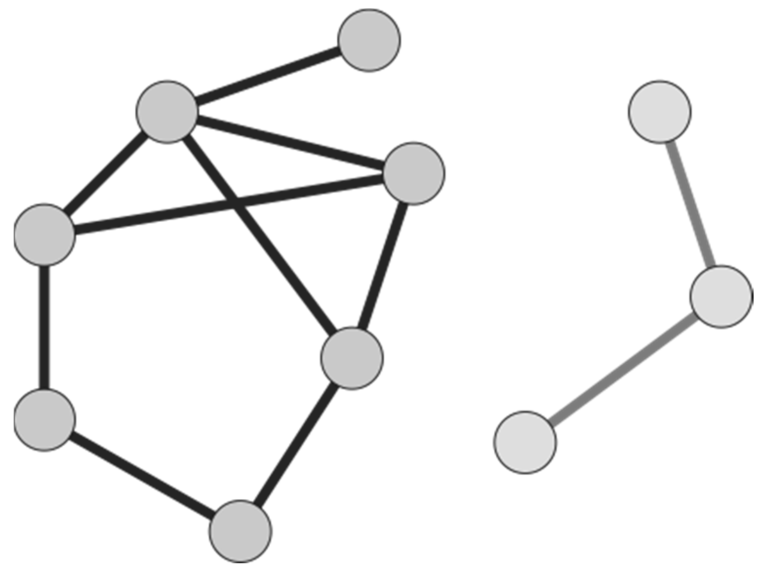

$C_2 = (g, d, b, h, j, c, i, e, b)$ nu este simplu.

# Reprezentari (7)

Un graf *G* este ***conex*** daca exista un drum intre oricare doua noduri din *G*.

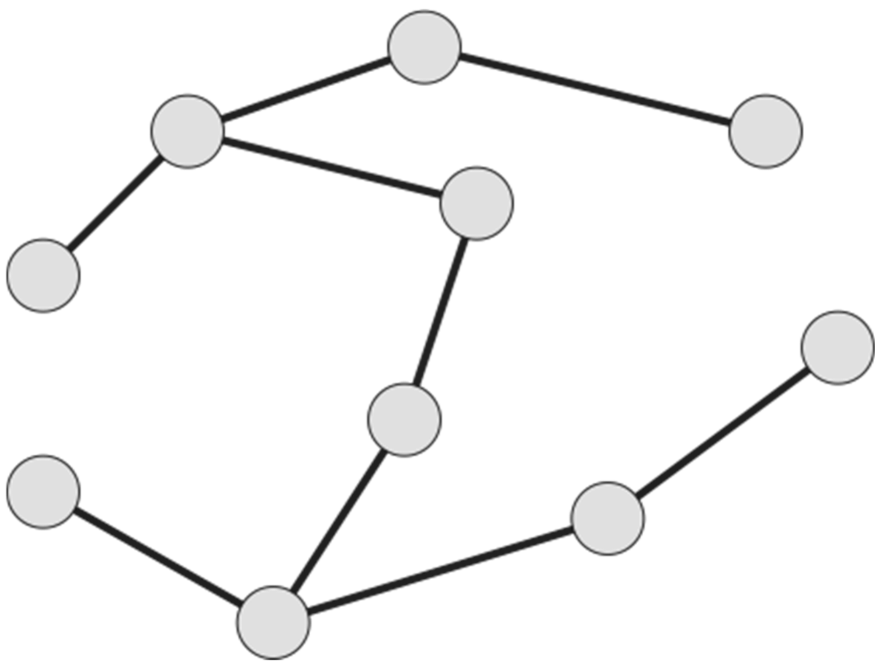


***Componentele conexe*** ale unui graf sunt date de subgrafurile sale conexate maximal.

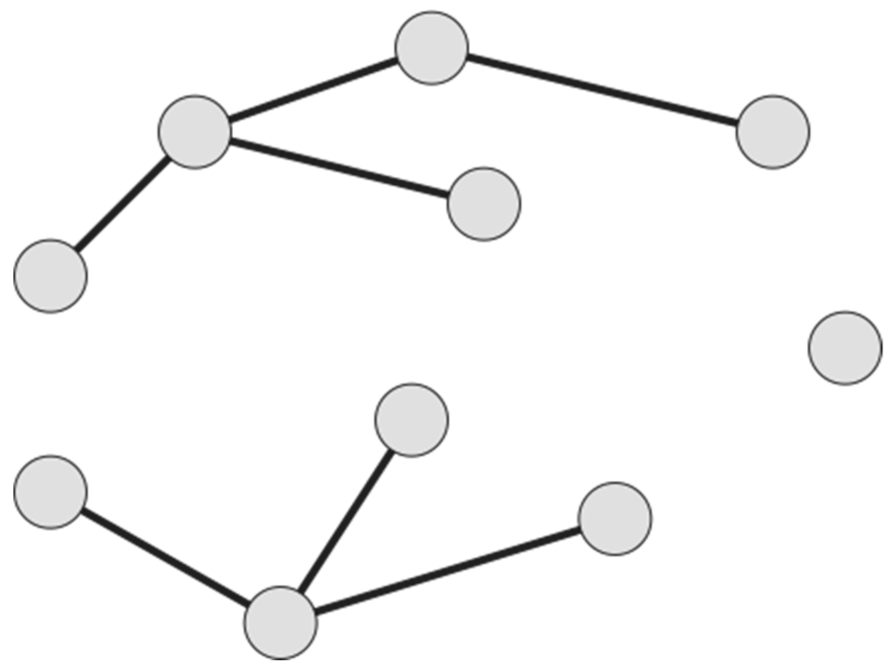


# Reprezentari (8)

Un ***arbore*** este un graf conexat si fara cicli.



O ***padure*** este un graf fara ciclii. Arborii reprezinta elementele conexe ale unei paduri.



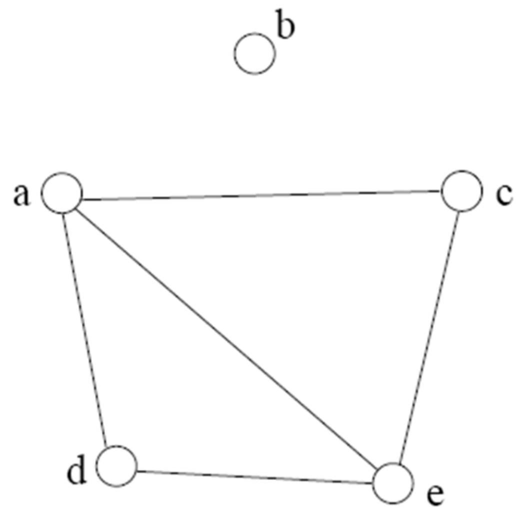
# Reprezentarea grafurilor

- Exista doua metode uzuale de reprezentare a grafurilor. Ambele reprezinta setul de arce / noduri – dar in forma diferite.

1. Matricea de adiacenta: se utilizeaza o matrice 2D pentru reprezentarea grafului

2. Lista de adiacenta: se utilizeaza un vector 1D de liste inlantuite

# Matricea de adiacenta (1)




| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | 0 | 1 | 1 | 1 |
| b | 0 | 0 | 0 | 0 | 0 |
| c | 1 | 0 | 0 | 0 | 1 |
| d | 1 | 0 | 0 | 0 | 1 |
| e | 1 | 0 | 1 | 1 | 0 |

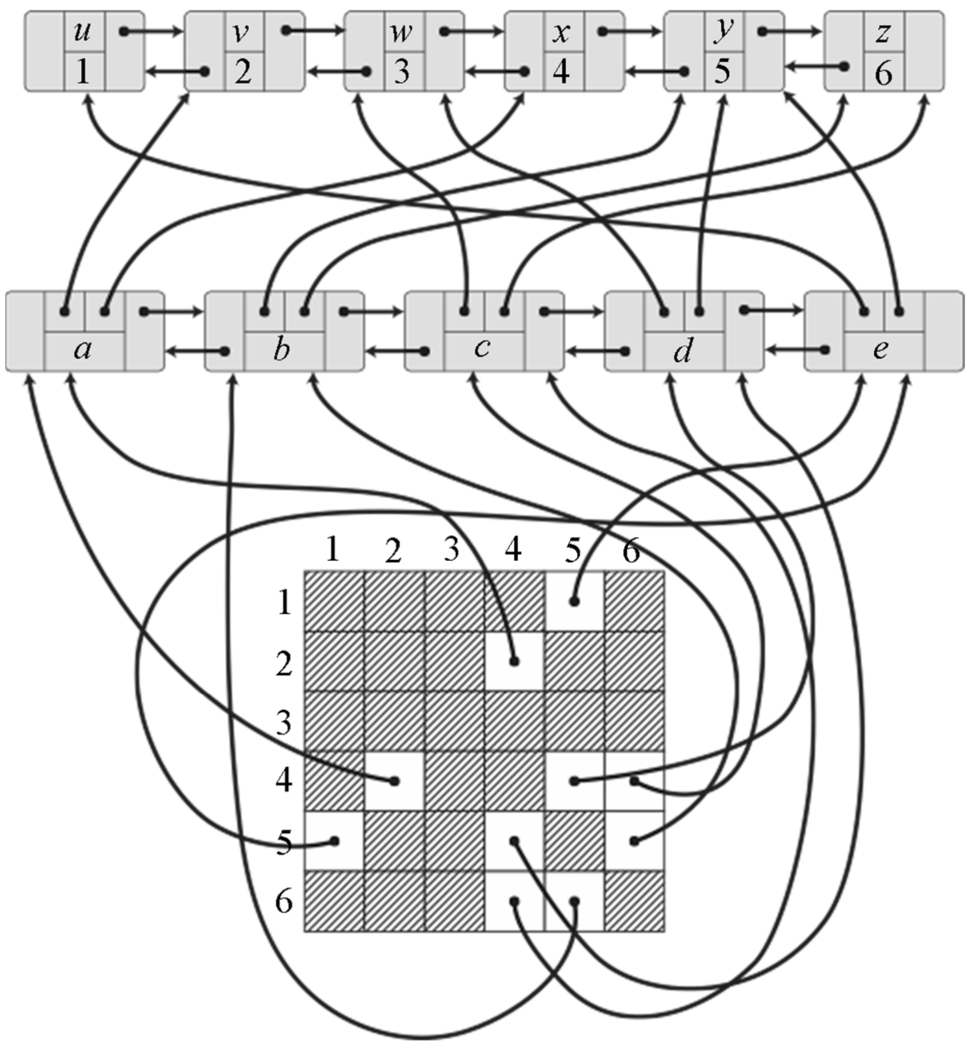
Reprezentare

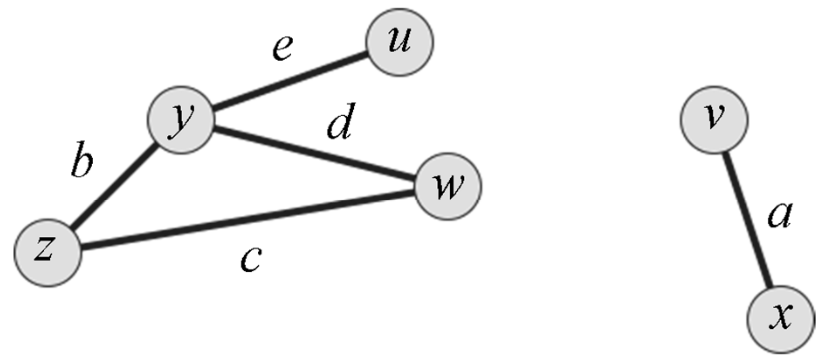
$$MA[i][j] = \begin{cases} 1 & dacă \quad (i,j) \in E \\ 0 & dacă \quad (i,j) \notin E \end{cases}$$

Reprezentarea prin *matrice de adiacenţe* asigură o simplitate a reprezentării, dar utilizează în mod ineficient memoria (consumul de memorie este $O(n^2)$

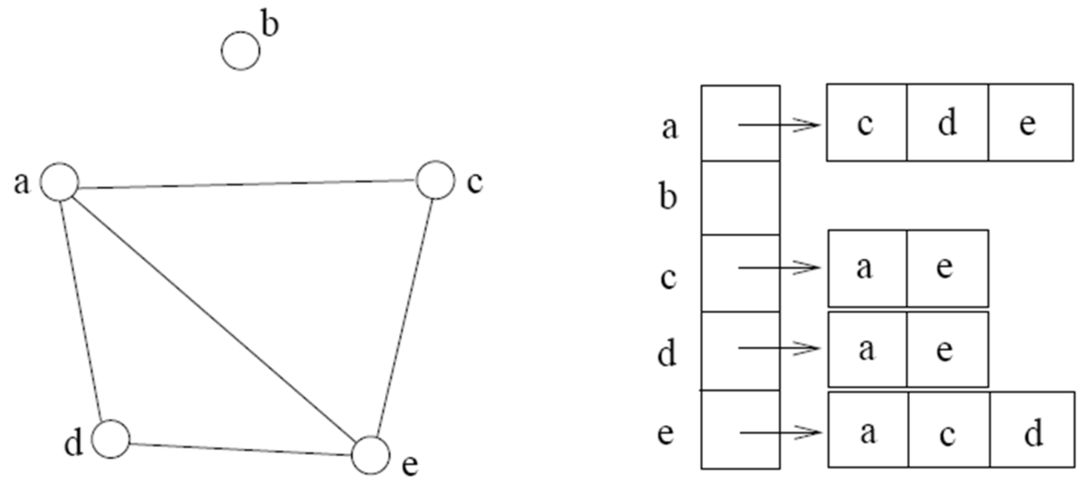
Algoritmii dezvoltaţi au în general complexitate $O(n^2)$ .

# Matricea de adiacenta (2)



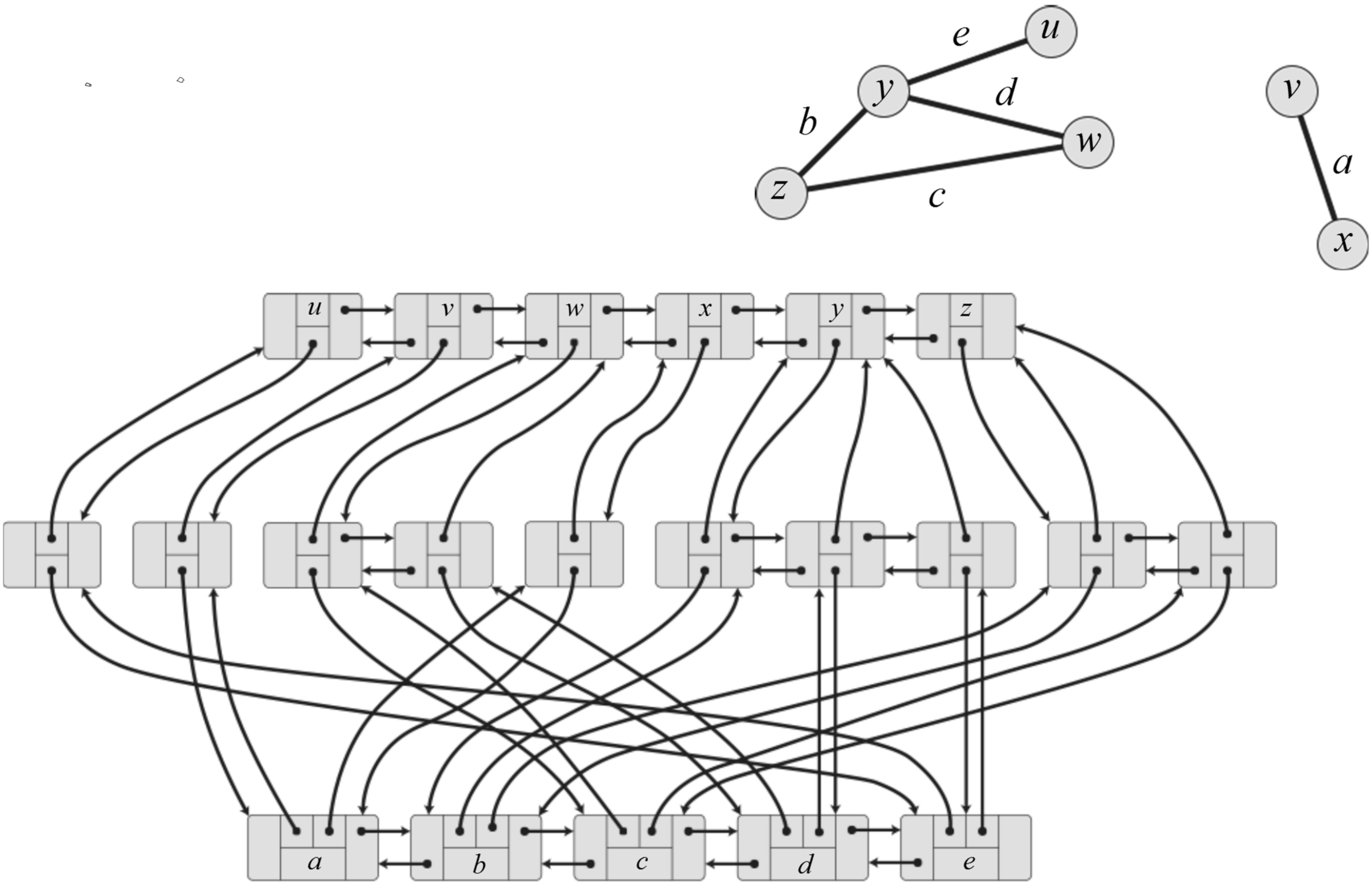
# Lista de adiacenta (1)




Fiecarui varf i se asociază o lista de succesori (sau predecesori).

Graful va fi reprezentat printr-un tablou de pointeri la listele de succesori (sau predecesori) ai fiecarui varf.

# Lista de adiacenta (2)

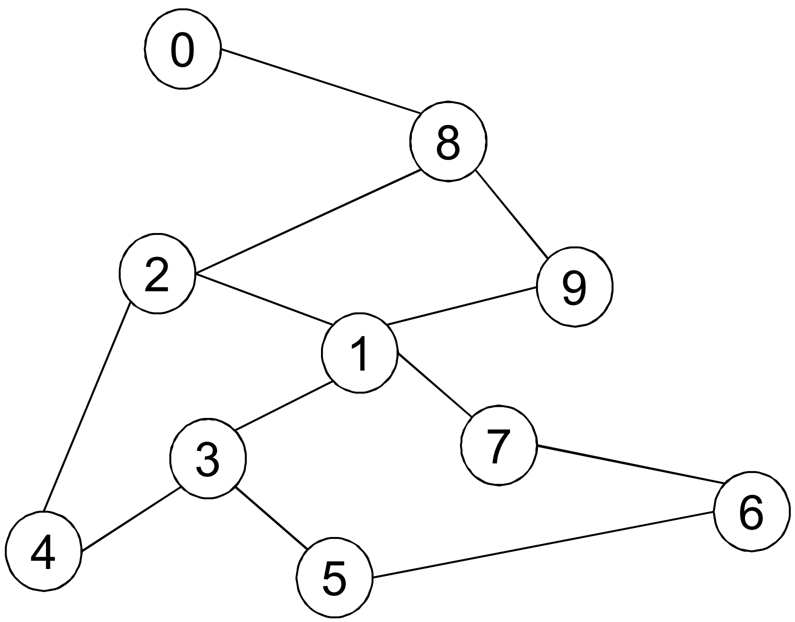


# Analiza temporala

Lista de adiacenta
- Parcurgerea varfurilor: O(n), n = numar de varfuri
- Parcurgerea muchiilor: O(m), m = numar muchii

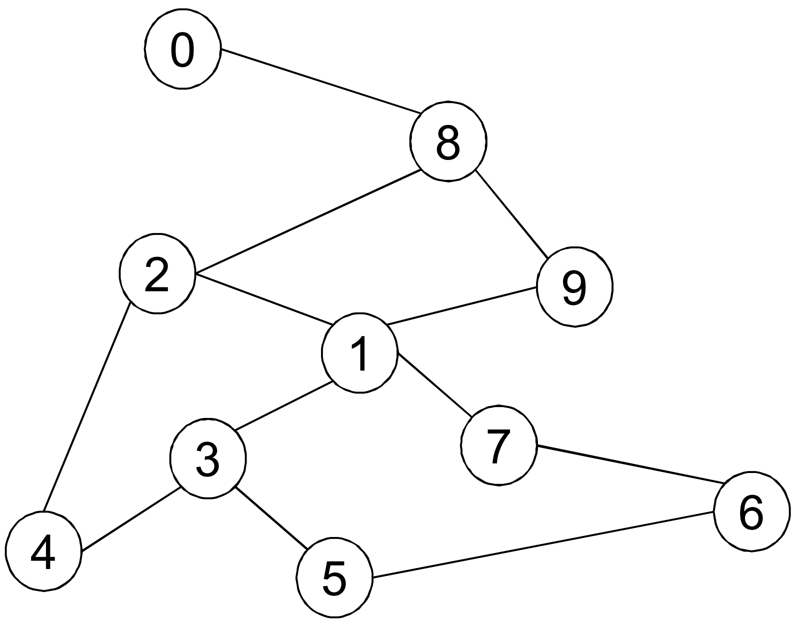
Matrice de adiacenta:
- Parcurgerea varfurilor: O(n), n = numar de varfuri
- Parcurgerea muchiilor: O(m), m = numar muchii

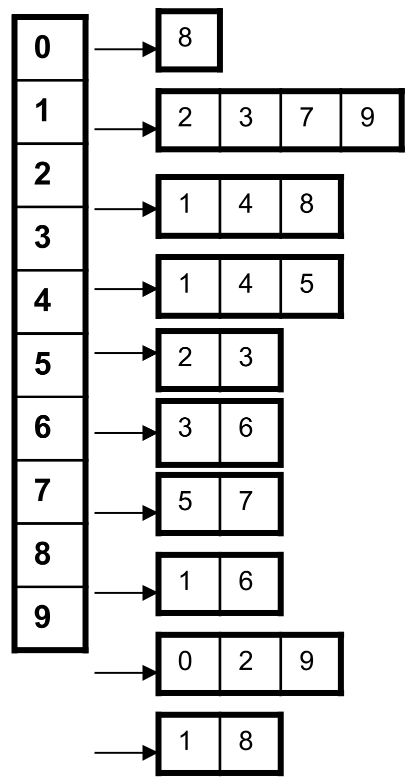
# Exemplu (1)




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **1** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| **2** | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| **3** | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| **7** | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **8** | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **9** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

# Exemplu (2)



# Liste adiacente vs. matrici adiacente

- **Liste adiacente (LA)**
  - Mai compacte decat MA daca graful are mai putine arce
  - Necesita mai mult timp de identificare a existentei unui arc.

- **Matrici adiacente (MA)**
  - Totdeauna necesita un spatiu $n^2$
    - Poate conduce la necesitatea unui spatiu mare de memorie
  - Necesita putin timp de identificare a existentei unui arc

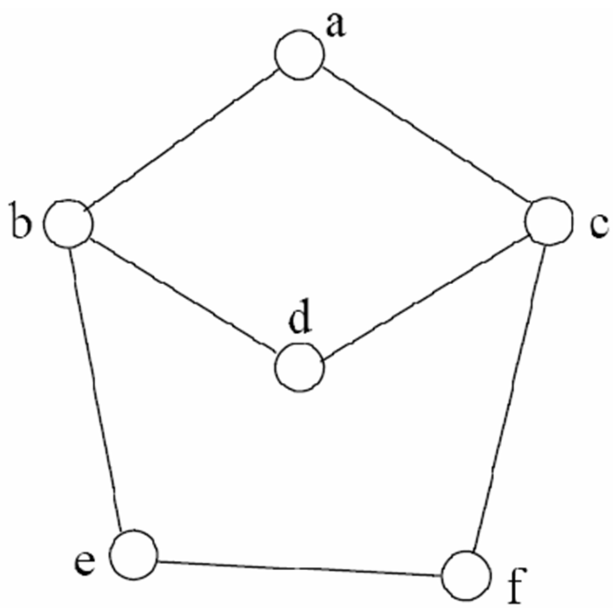
# Tipuri de *paths*

- Un drum este **simplu** daca si numai daca nu intalneste un nod decat o singura data.

- Un drum este **ciclic** daca si numai daca $v_0 = v_k$
  - Inceputul si sfarsitul drumului sunt pe un acelasi nod

- Un drum contine un **ciclu** daca si numai daca un nod apare de doua sau mai multe ori.

# Exemplu




Exista drumuri simple?

Exista cicluri?

Ce reprezinta lungimea unui drum?

1. {a,c,f,e}
2. {a,b,d,c,f,e}
3. {a, c, d, b, d, c, f, e}
4. {a,c,d,b,a}
5. {a,c,f,e,b,d,c,a}

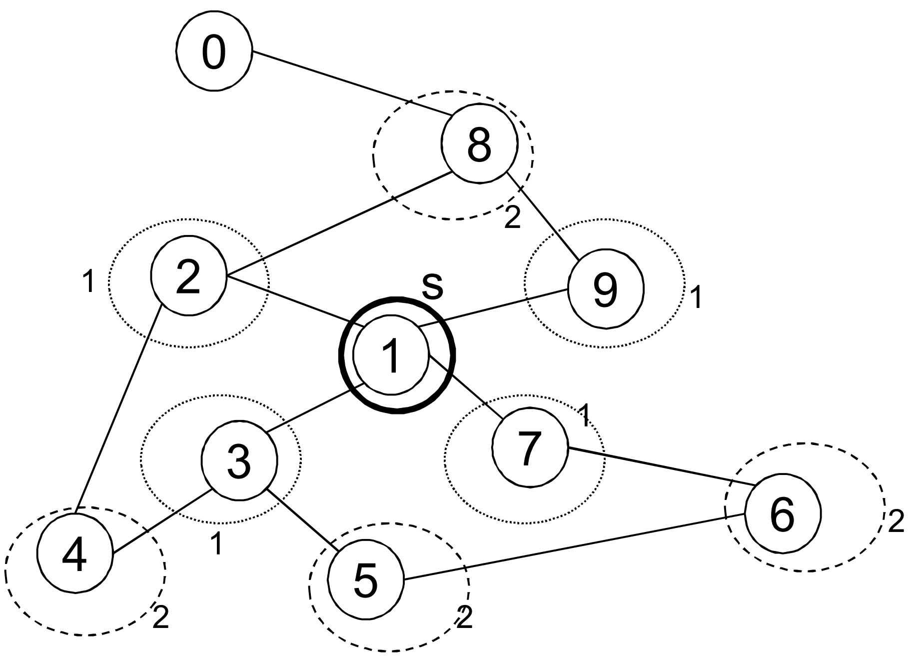
# Explorarea grafurilor

- Un exemplu
  - Fie o reprezentare de tip *graf* si un nod **s** in cadrul acestui graf
  - Sa se gaseasca toate drumurile de la nodul **s** la celelalte noduri

- *Explorarea* (*traversarea* ) unui graf: o metoda sistematica de parcurgere, prin examinarea muchiilor si varfurilor. O traversare eficienta are loc în *timp liniar*. Metode:
  - **Traversarea în latime - Breadth-First Search (BFS)** – permite calculul distantei (in nr muchii) de la sursa la fiecare varf
    - ⇨ Gasirea celui mai scurt drum intr'un graf neponderat
    - ⇨ Pentru orice varf **v**, accesibil din sursa **s,** calea in arbore corespunde celui mai scurt drum de la **s** la **v**.
    - ⇨ Descoperit in '50 de E. F. Moore
  - **Traversarea în adancime - Depth-First Search (DFS)**
    - ⇨ Sortare topologica
    - ⇨ Gasirea componentelor puternic interconectate
    - ⇨ Determina daca un graf este conex

# BFS si problema drumului minim

- Fie un nod sursa **s**; BFS va vizita celelalte noduri ale grafului pe o **distanta crescatoare** pornind din **s**. In acest fel, BFS va descoperi caile ce pornesc din **s** catre alte noduri.
- Ce se intelege prin **distanta**? Numarul de arce de pe un drum ce porneste din **s**.




Exemple

Fie s=nod 1

Noduri la distanta 1?
2, 3, 7, 9

Noduri la distanta 2?
8, 6, 5, 4

Noduri la distanta 3?
0

# Algoritmul BFS

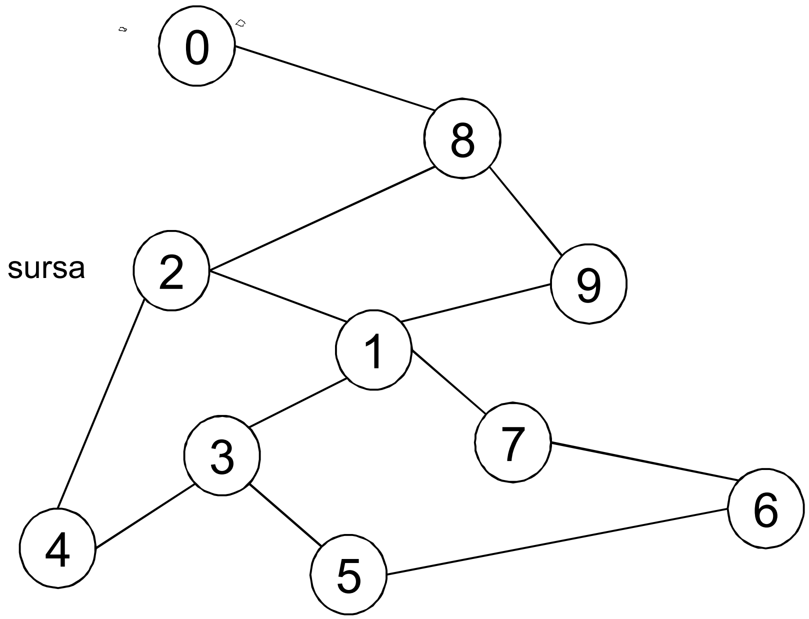
**Algorithm** $BFS(s)$

**Input:** $s$ is the source vertex

**Output:** Mark all vertices that can be visited from $s$.

**for** each vertex $v$<br>
  **do** $flag[v]$ := false;<br>
$Q$ = empty queue;<br>
$flag[s]$ := true;<br>
*enqueue*$(Q, s)$;<br>
**while** $Q$ is not empty<br>
  **do** $v$ := *dequeue*$(Q)$;<br>
    **for** each $w$ adjacent to $v$<br>
      **do if** $flag[w]$ = false<br>
        **then** $flag[w]$ := true;<br>
          *enqueue*$(Q, w)$

# Exemplu (1)




Lista de adiacente

| | |
|---|---|
| 0 | 8 |
| 1 | 3 7 9 2 |
| 2 | 8 1 4 |
| 3 | 4 5 1 |
| 4 | 2 3 |
| 5 | 3 6 |
| 6 | 7 5 |
| 7 | 1 6 |
| 8 | 2 0 9 |
| 9 | 1 8 |

Tabla de vizitare (T/F)

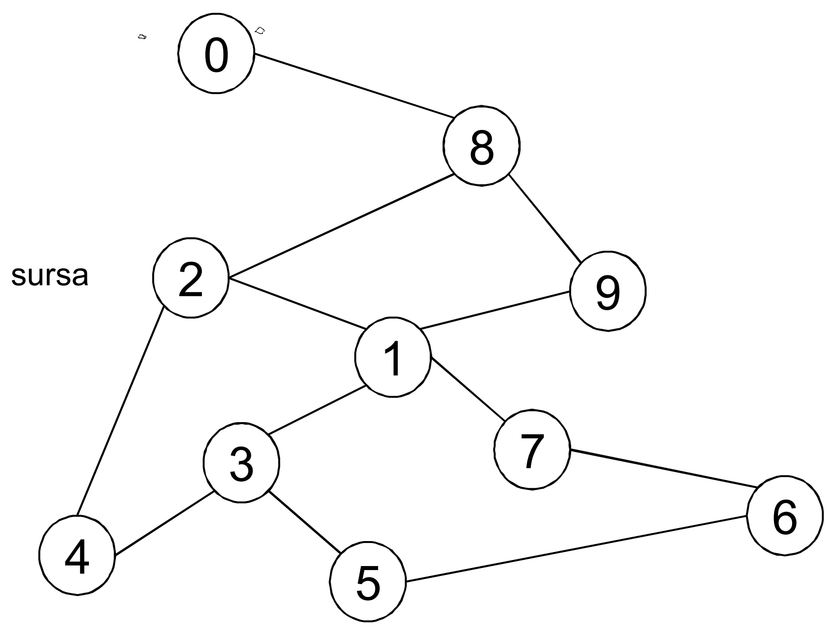
| | |
|---|---|
| 0 | F |
| 1 | F |
| 2 | F |
| 3 | F |
| 4 | F |
| 5 | F |
| 6 | F |
| 7 | F |
| 8 | F |
| 9 | F |

**Q =** { }

Initializare **Q** vid

Initializare tabla (totul False)

# Exemplu (2)




Lista de adiacente

| | |
|---|---|
| 0 | 8 |
| 1 | 3 7 9 2 |
| 2 | 8 1 4 |
| 3 | 4 5 1 |
| 4 | 2 3 |
| 5 | 3 6 |
| 6 | 7 5 |
| 7 | 1 6 |
| 8 | 2 0 9 |
| 9 | 1 8 |

Tabla de vizitare (T/F)

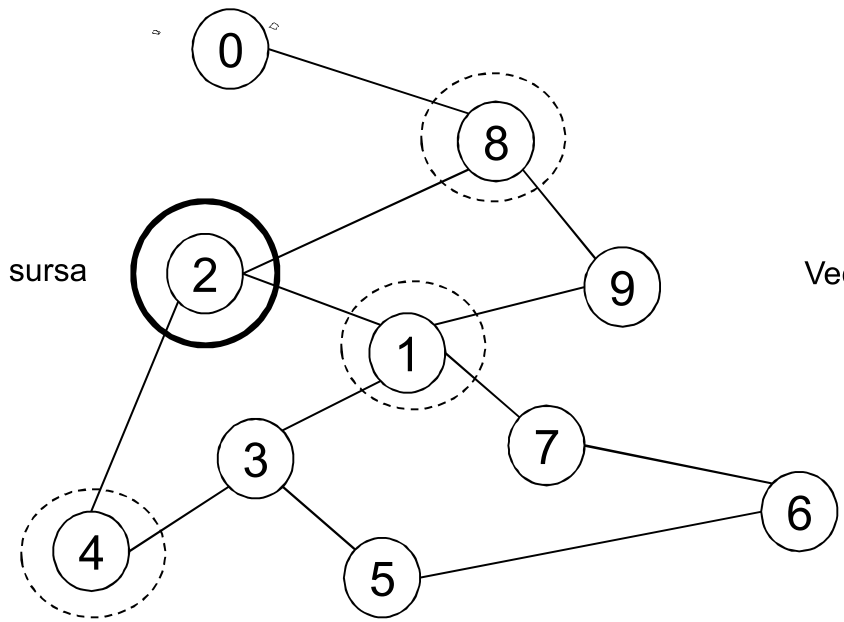
| | |
|---|---|
| 0 | F |
| 1 | F |
| 2 | T |
| 3 | F |
| 4 | F |
| 5 | F |
| 6 | F |
| 7 | F |
| 8 | F |
| 9 | F |

**Q =** { 2 }

Se pune sursa 2 in coada.

*Flag* faptul ca 2 a fost vizitat.

# Exemplu (3)




Lista de adiacente

Vecini →

| | |
|---|---|
| 0 | 8 |
| 1 | 3 7 9 2 |
| 2 | 8 1 4 |
| 3 | 4 5 1 |
| 4 | 2 3 |
| 5 | 3 6 |
| 6 | 7 5 |
| 7 | 1 6 |
| 8 | 2 0 9 |
| 9 | 1 8 |

Tabla de vizitare (T/F)

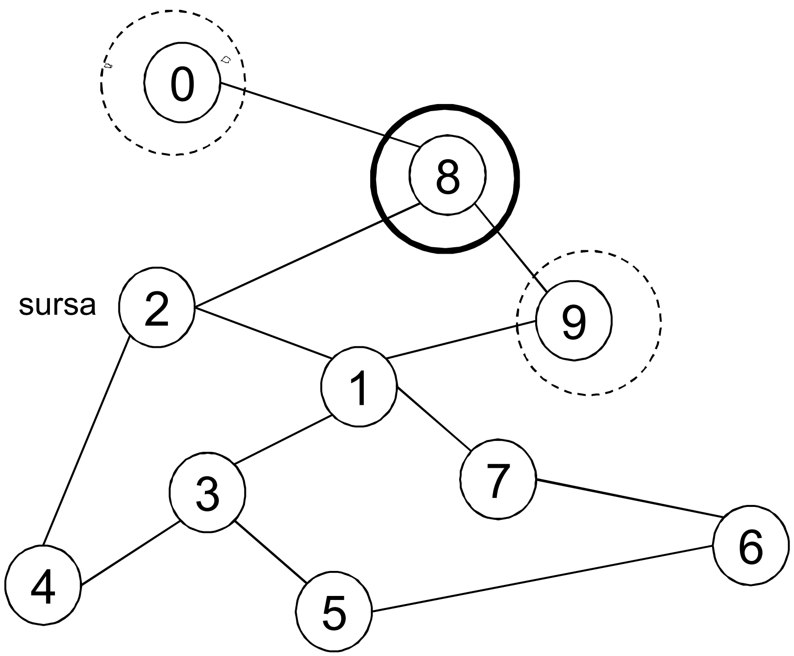
| | |
|---|---|
| 0 | F |
| 1 | T |
| 2 | T |
| 3 | F |
| 4 | T |
| 5 | F |
| 6 | F |
| 7 | F |
| 8 | T |
| 9 | F |

**Q =** {2} → { 8, 1, 4 }

Marcare vecini vizitati.

Dequeue 2.
Plasarea tuturor vecinilor lui 2 nevizitati in coada

# Exemplu (4)




Lista de adiacente

| | | | | |
|---|---|---|---|---|
| 0 | 8 | | | |
| 1 | 3 | 7 | 9 | 2 |
| 2 | 8 | 1 | 4 | |
| 3 | 4 | 5 | 1 | |
| 4 | 2 | 3 | | |
| 5 | 3 | 6 | | |
| 6 | 7 | 5 | | |
| 7 | 1 | 6 | | |
| 8 | 2 | 0 | 9 | |
| 9 | 1 | 8 | | |

Vecini →

Tabla de vizitare (T/F)

| | |
|---|---|
| 0 | T |
| 1 | T |
| 2 | T |
| 3 | F |
| 4 | T |
| 5 | F |
| 6 | F |
| 7 | F |
| 8 | T |
| 9 | T |

**Q =** { 8, 1, 4 } → { 1, 4, 0, 9 }

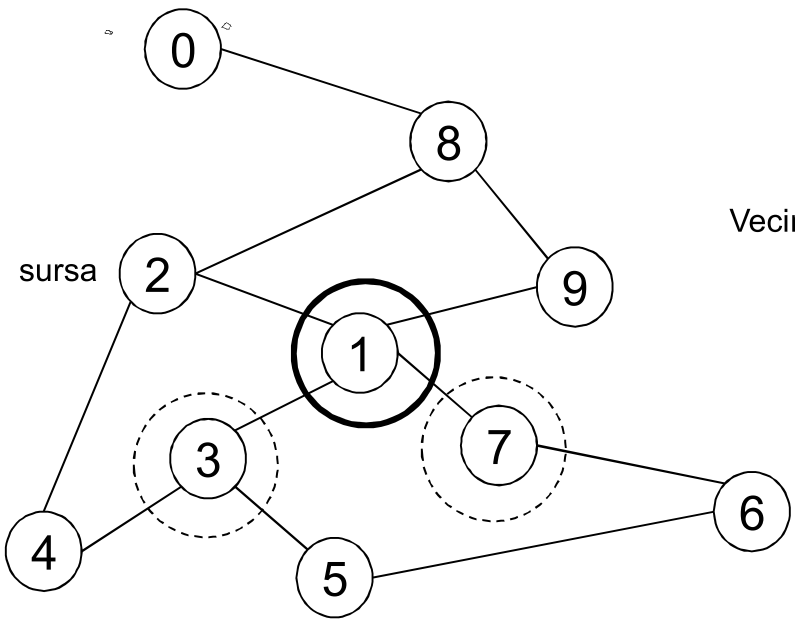
Dequeue 8.

-- Se pun toti vecini nevizitati ai lui 8 in coada.

-- 2 nu mai este pus in coada, el a fost deja vizitat!

Se marcheaza noii vecini vizitati.

# Exemplu (5)




Lista de adiacente

Vecini →

| | |
|---|---|
| 0 | 8 |
| 1 | 3 7 9 2 |
| 2 | 8 1 4 |
| 3 | 4 5 1 |
| 4 | 2 3 |
| 5 | 3 6 |
| 6 | 7 5 |
| 7 | 1 6 |
| 8 | 2 0 9 |
| 9 | 1 8 |

Tabla de vizitare (T/F)

| | |
|---|---|
| 0 | T |
| 1 | T |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | F |
| 6 | F |
| 7 | T |
| 8 | T |
| 9 | T |

**Q =** { 1, 4, 0, 9 } → { 4, 0, 9, 3, 7 }

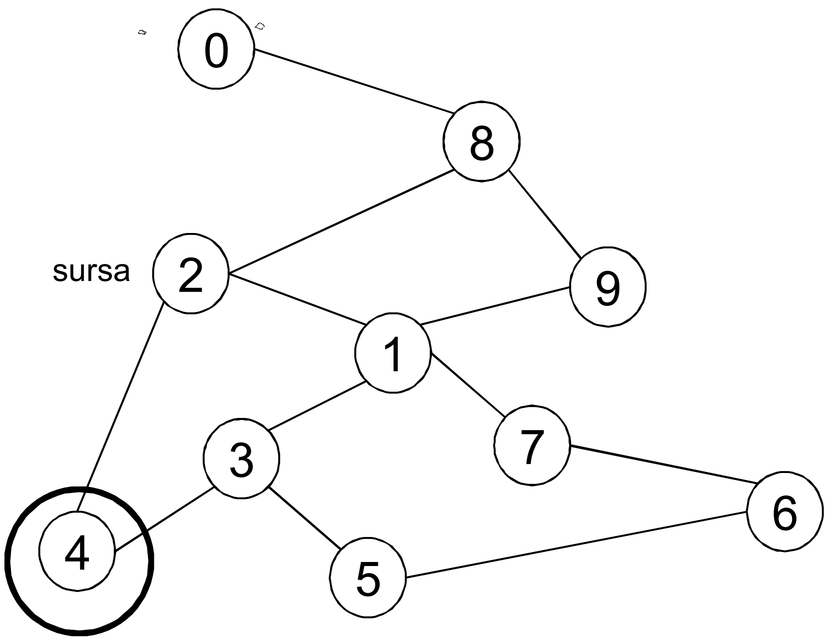
Se marcheaza noii vecini vizitati.

Dequeue 1.

-- Se pun toti vecini nevizitati ai lui 8 in coada.

-- Numai nodurile 3 si 7 nu au fost vizitate acum.

# Exemplu (6)




Lista de adiacente

Vecini →

| | | | | |
|---|---|---|---|---|
| 0 | 8 | | | |
| 1 | 3 | 7 | 9 | 2 |
| 2 | 8 | 1 | 4 | |
| 3 | 4 | 5 | 1 | |
| 4 | 2 | 3 | | |
| 5 | 3 | 6 | | |
| 6 | 7 | 5 | | |
| 7 | 1 | 6 | | |
| 8 | 2 | 0 | 9 | |
| 9 | 1 | 8 | | |

Tabla de vizitare (T/F)

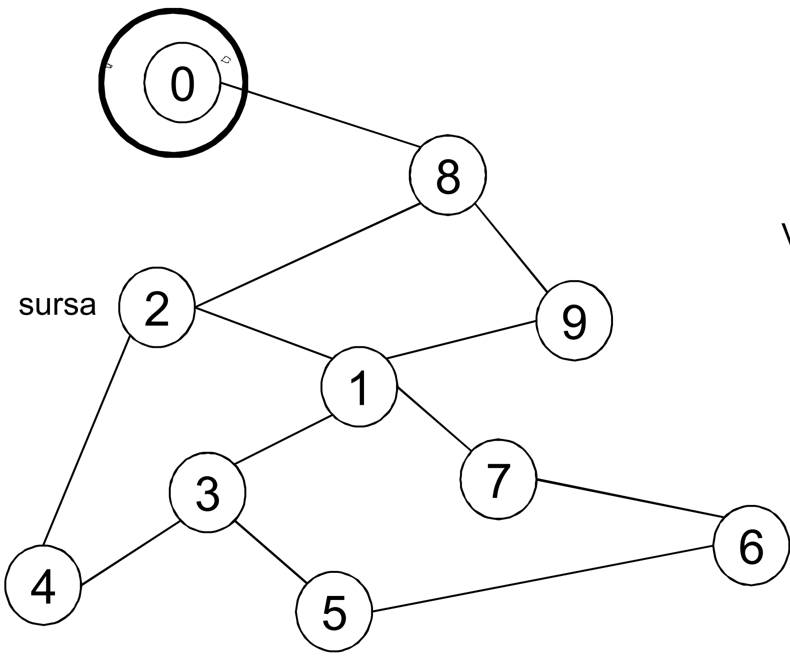
| | |
|---|---|
| 0 | T |
| 1 | T |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | F |
| 6 | F |
| 7 | T |
| 8 | T |
| 9 | T |

**Q =** { 4, 0, 9, 3, 7 } → { 0, 9, 3, 7 }

Dequeue 4.

-- 4 nu are vecini nevizitati!

# Exemplu (7)




Lista de adiacente

Vecini →

| | |
|---|---|
| 0 | 8 |
| 1 | 3 7 9 2 |
| 2 | 8 1 4 |
| 3 | 4 5 1 |
| 4 | 2 3 |
| 5 | 3 6 |
| 6 | 7 5 |
| 7 | 1 6 |
| 8 | 2 0 9 |
| 9 | 1 8 |

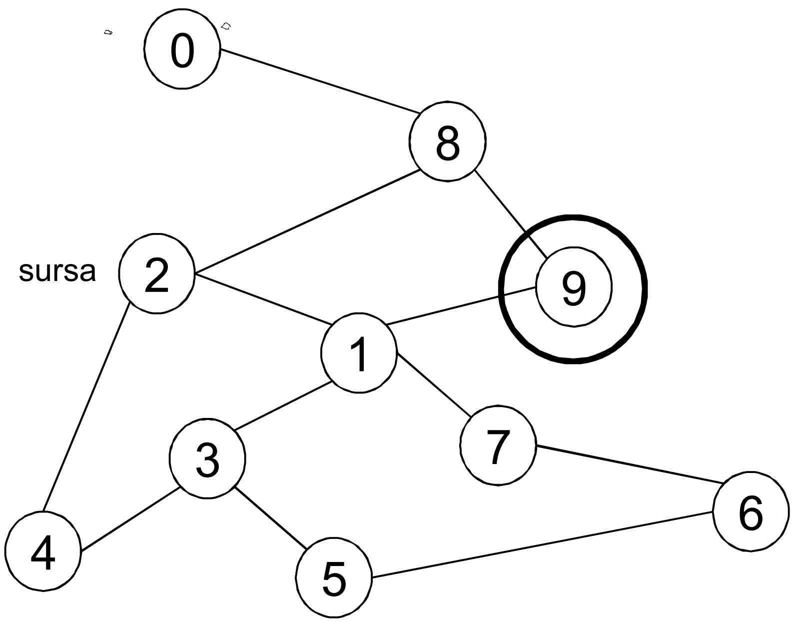
Tabla de vizitare (T/F)

| | |
|---|---|
| 0 | T |
| 1 | T |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | F |
| 6 | F |
| 7 | T |
| 8 | T |
| 9 | T |

**Q =** { 0, 9, 3, 7 } → { 9, 3, 7 }

Dequeue 0.
-- 0 nu are vecini nevizitati!

# Exemplu (8)




Lista de adiacente

| | |
|---|---|
| 0 | 8 |
| 1 | 3 7 9 2 |
| 2 | 8 1 4 |
| 3 | 4 5 1 |
| 4 | 2 3 |
| 5 | 3 6 |
| 6 | 7 5 |
| 7 | 1 6 |
| 8 | 2 0 9 |
| 9 | 1 8 |

Vecini → 9

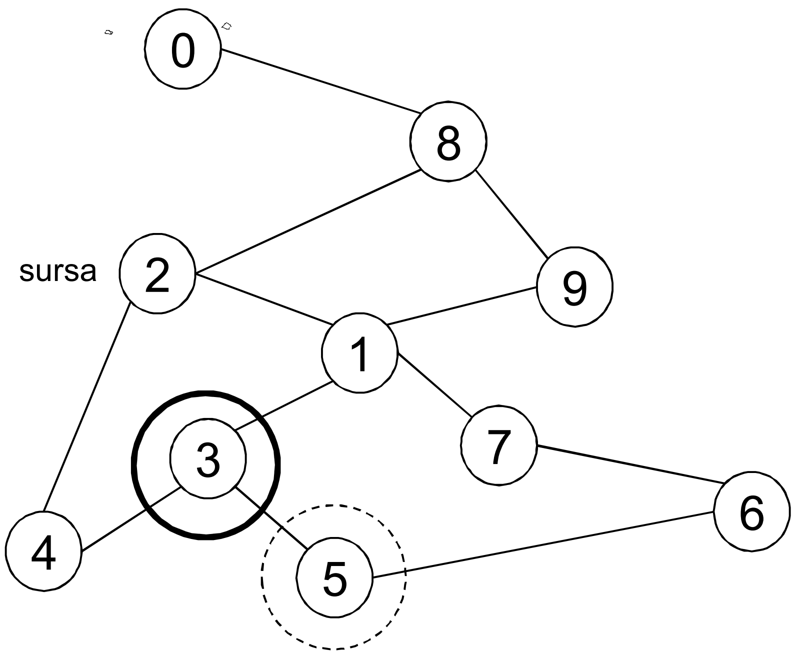
Tabla de vizitare (T/F)

| | |
|---|---|
| 0 | T |
| 1 | T |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | F |
| 6 | F |
| 7 | T |
| 8 | T |
| 9 | T |

**Q =** { 9, 3, 7 } → { 3, 7 }

Dequeue 9.
-- 9 nu are vecini nevizitati!

# Exemplu (9)




Lista de adiacente

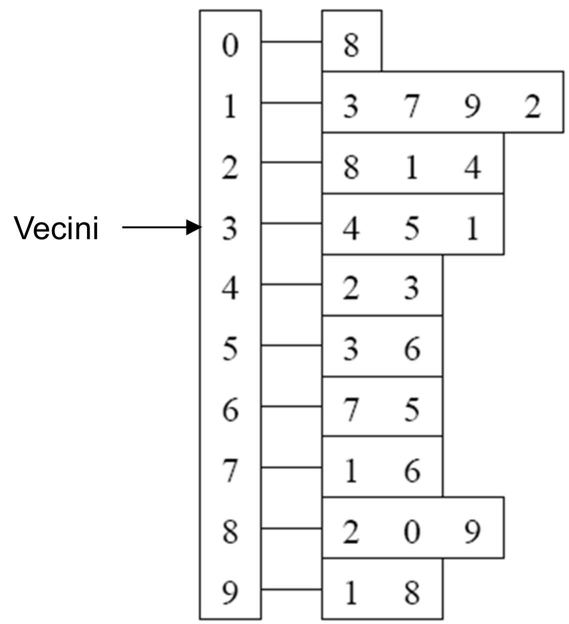



Tabla de vizitare (T/F)

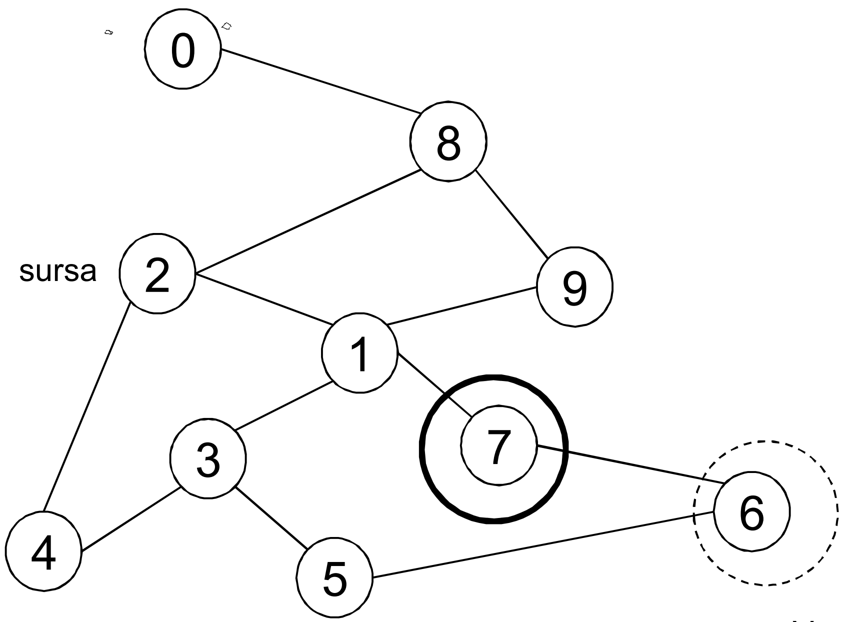
| | |
|---|---|
| 0 | T |
| 1 | T |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | F |
| 7 | T |
| 8 | T |
| 9 | T |

**Q =** { 3, 7 } → { 7, 5 }

Dequeue 3.
-- Vecinul 5 este pus in coada.

Se marcheza
nodul 5 vizitat

# Exemplu (10)




Lista de adiacente

Vecini →

| | |
|---|---|
| 0 | 8 |
| 1 | 3 7 9 2 |
| 2 | 8 1 4 |
| 3 | 4 5 1 |
| 4 | 2 3 |
| 5 | 3 6 |
| 6 | 7 5 |
| 7 | 1 6 |
| 8 | 2 0 9 |
| 9 | 1 8 |

Tabla de vizitare (T/F)

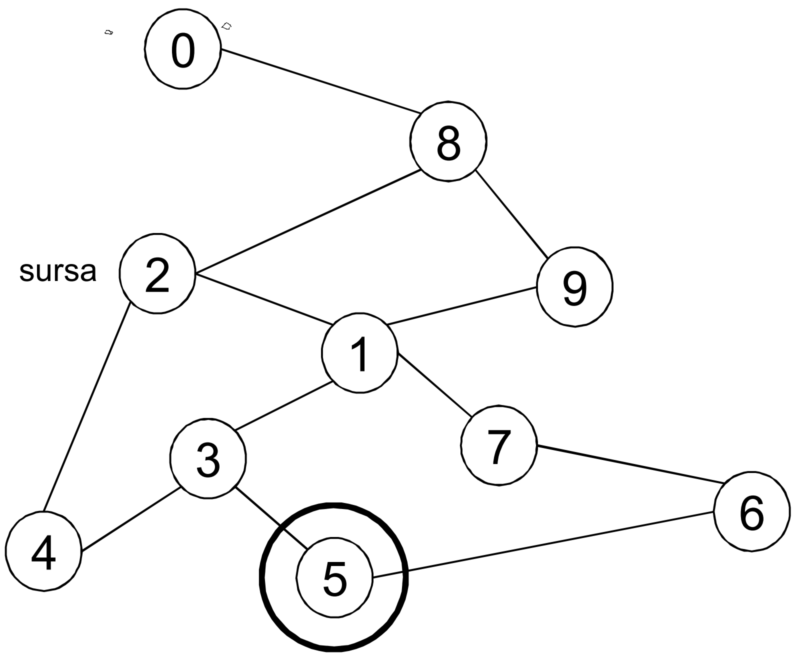
| | |
|---|---|
| 0 | T |
| 1 | T |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |

**Q =** { 7, 5 } → { 5, 6 }

Dequeue 7.
-- Vecinul 6 este pus in coada.

Se marcheaza nodul 6 vizitat.

# Exemplu (11)




Lista de adiacente

Vecini → 5

| | |
|---|---|
| 0 | 8 |
| 1 | 3 7 9 2 |
| 2 | 8 1 4 |
| 3 | 4 5 1 |
| 4 | 2 3 |
| 5 | 3 6 |
| 6 | 7 5 |
| 7 | 1 6 |
| 8 | 2 0 9 |
| 9 | 1 8 |

Tabla de vizitare (T/F)

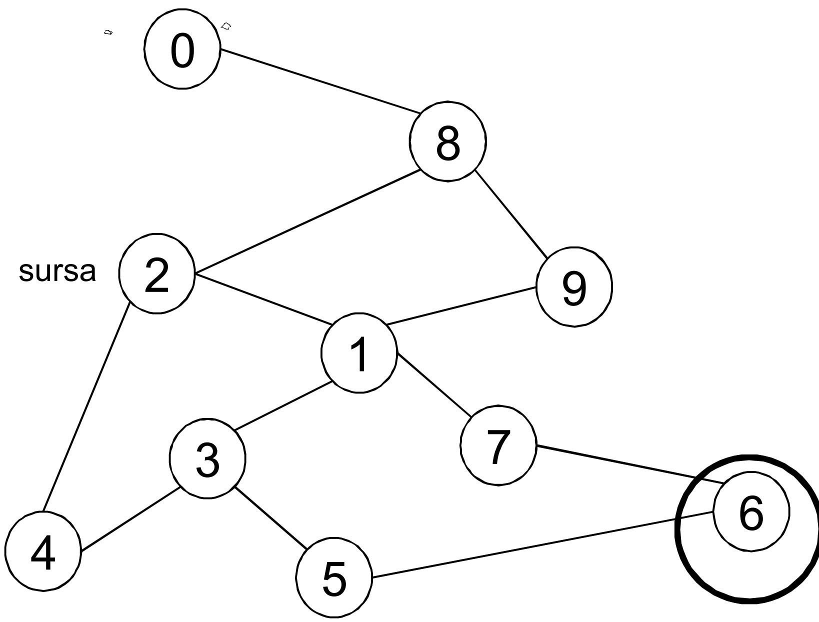
| | |
|---|---|
| 0 | T |
| 1 | T |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |

**Q =** { 5, 6} → { 6 }

Dequeue 5.

-- nu exista vecini nevizitati ai lui 5.

# Exemplu (12)




Vecini →

Lista de adiacente

| | |
|---|---|
| 0 | 8 |
| 1 | 3 7 9 2 |
| 2 | 8 1 4 |
| 3 | 4 5 1 |
| 4 | 2 3 |
| 5 | 3 6 |
| 6 | 7 5 |
| 7 | 1 6 |
| 8 | 2 0 9 |
| 9 | 1 8 |

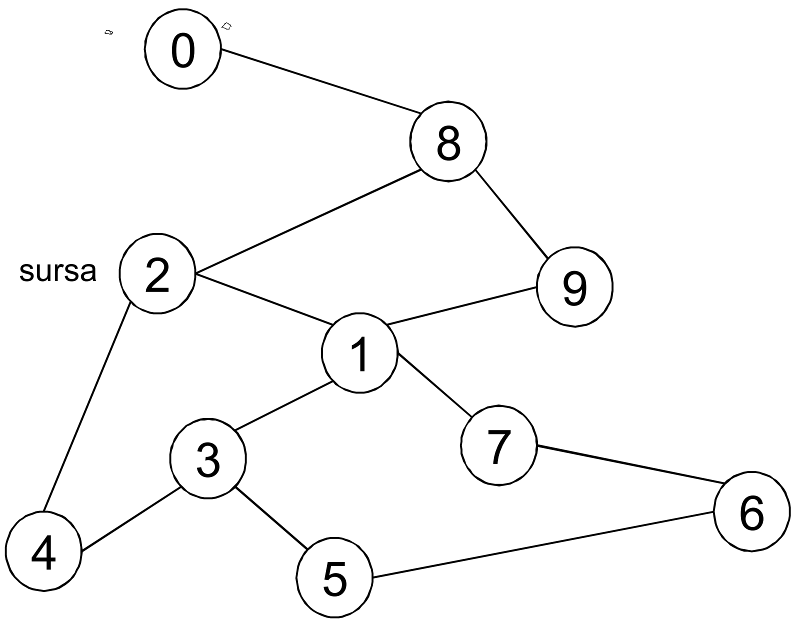
Tabla de vizitare (T/F)

| | |
|---|---|
| 0 | T |
| 1 | T |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |

**Q =** { 6 } → { }

Dequeue 6.
-- nu exista vecini nevizitati ai lui 6.

# Exemplu (13)




Lista de adiacente

| | |
|---|---|
| 0 | 8 |
| 1 | 3 7 9 2 |
| 2 | 8 1 4 |
| 3 | 4 5 1 |
| 4 | 2 3 |
| 5 | 3 6 |
| 6 | 7 5 |
| 7 | 1 6 |
| 8 | 2 0 9 |
| 9 | 1 8 |

Tabla de vizitare (T/F)

| | |
|---|---|
| 0 | T |
| 1 | T |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |

**Q =** { }

**STOP!!! Q este gol!!!**

Ce se observa?

Exista un drum de la sursa nod 2 la toate drumurile din graf.

# Sortare topologica

- Intr'un graf orientat, prezenta unui arc (**u,v**) poate fi privita ca o relatie de precedenta (i.e. **u** precede pe **v**)

- Sortarea topologica a unui graf orientat aciclic reprezinta o ordonare liniara de varfuri in care **u** precede **v**

- Un algoritm de sortare topologica (d.e. AST) porneste dintr'un varf ce nu este precedat de un altul (i.e. are grad interior nul)

# Alocarea de timp (1)

- In cazul listei de adiacenta
  - n = numar de noduri   m = numar de arce

**Algorithm** *BFS*($s$)
**Input:** $s$ is the source vertex
**Output:** Mark all vertices that can be visited from $s$.

```
for each vertex v
    do flag[v] := false;
Q = empty queue;
flag[s] := true;
enqueue(Q, s);
while Q is not empty
    do v := dequeue(Q);
        for each w adjacent to v
            do if flag[w] = false
                then flag[w] := true;
                    enqueue(Q, w)
```

$$\boxed{O(n + m)}$$

# Alocarea de timp (2)

- Fiind dat un graf cu m arce, care este gradul maxim?

$$\Sigma_{\text{nod } v} \ \deg(v) = 2m$$

- Timpul total de rulare pentru bucla *while* este:

$$O(\Sigma_{\text{nod } v} (\deg(v) + 1)) = O(n+m)$$

# Alocarea de timp (3)

- In cazul matricei de adiacenta
  - n = numar de noduri   m = numar de arce

**Algorithm** *BFS*($s$)
**Input:** $s$ is the source vertex
**Output:** Mark all vertices that can be visited from $s$.

```
for each vertex v
    do flag[v] := false;
Q = empty queue;
flag[s] := true;
enqueue(Q, s);
while Q is not empty
    do v := dequeue(Q);
        for each w adjacent to v
            do if flag[w] = false
                then flag[w] := true;
                    enqueue(Q, w)
```

$\boxed{O(n^2)}$

**Gasirea tuturor nodurilor adiacente ale lui v necesita checking pentru toate elementele din linie. Asta presupune un timp linear O(n).**

**Prin insumarea tuturor iteratiilor timpul total va fi O($n^2$).**

**Astfel, cu matricea de adiacenta, BFS este O($n^2$) independent de numarul de arce m. Cu lista de adiacenta, BFS este O(n+m); daca m=O($n^2$) → O(n+m)=O($n^2$).**

# Aplicatii ale BFS

- O utilizare frecventa consta in gasirea componentelor conectate ale unui graf.
- Drumul cel mai scurt dintre 2 noduri
- Retele sociale: identificarea distantei **k** intre 2 persoane (**k** = numarul de nivele)
- Navigare GPS: identificarea vecinatatilor
- Identificarea variantei corecte (cea mai apropiata) intr'un *spelling*: d.e. **readed → reader**