Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare

GRAFURI

Definiție

- Un graf este o modalitate de a reprezenta **relațiile** care există între **perechi de obiecte**
- Un **graf G** este o pereche de două mulțimi **G = (V, E)**, unde **V** este mulțimea (nevidă) a vârfurilor (nodurilor), iar **E** este mulțimea muchiilor (arcelor)
- O muchie din **E** unește o pereche de două vârfuri din **V** și se notează **(v,w)**

Tipuri de grafuri

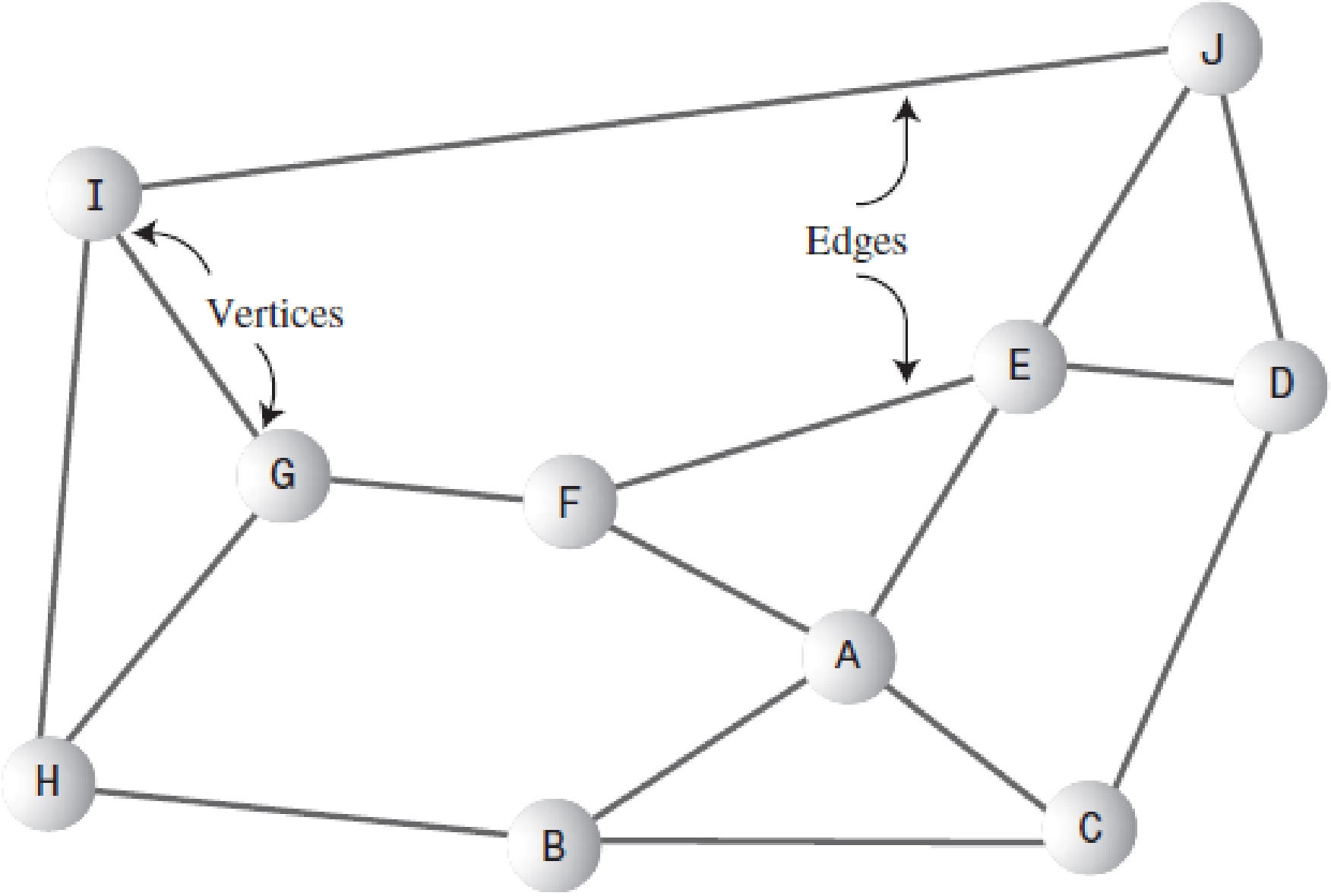
- 1) Graf **neorientat** (*undirected graph*)
- 2) Graf **orientat** (*directed graph, digraph*)
- 3) Graf **neorientat** și **ponderat** (*undirected weighted graph*)
- 4) Graf **orientat** și **ponderat** (*directed weighted graph*)

Observații

- Nodurile unui graf se numerotează începând cu **1** și mulțimea **V** este o submulțime a numerelor naturale
- Termenii “**vârf**” și “**muchie**” provin din analogia unui graf cu un poliedru și se folosesc mai ales pentru **grafuri neorientate**
- Termenii “**nod**” și “**arc**” se folosesc mai ales pentru **grafuri orientate**

Definiții

- Două vârfuri se numesc **adiacente** dacă sunt conectate direct printr-o muchie
- Vârfurile adiacente cu un vârf se numesc **vecinii** vârfului dat
- Un **lanț/drum** reprezintă o secvență de **muchii/arce**


J
I
Edges
Vertices
E
D
G
F
A
H
B
C

Arce

- Dacă graful este **orientat, v – originea arcului** (v,w), **w – destinația arcului** (v,w)
- **Arce de ieșire** (outgoing) ale unui **nod v** – arcele orientate care au ca origine acel nod v
- **Arce de intrare** (incoming) ale unui **nod v** – arcele orientate care au ca destinație acel nod v

Grade

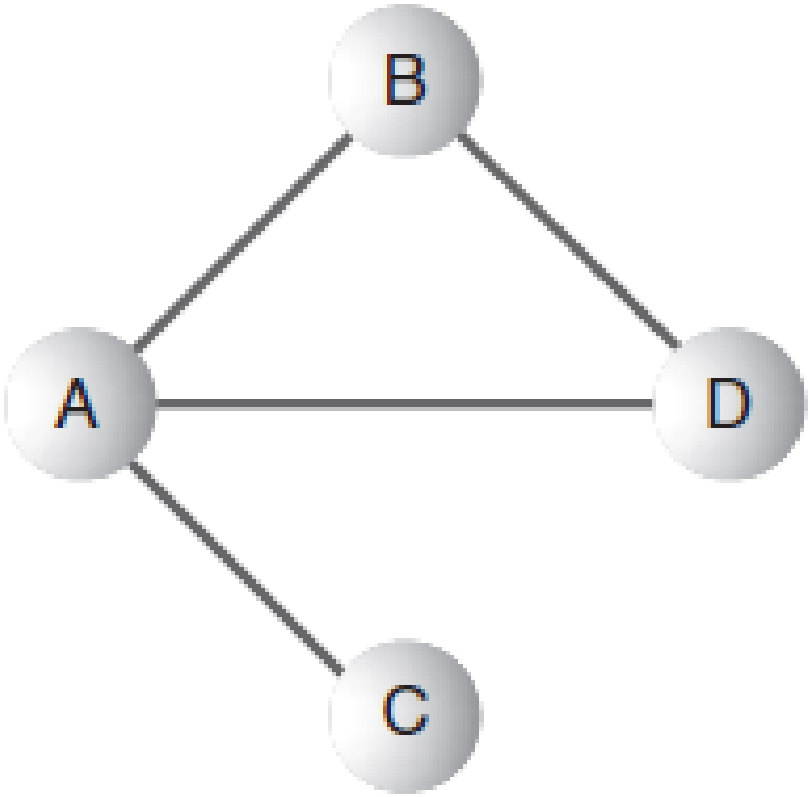
- **Gradul unui nod** v este numărul de arce incidente cu acesta – **d(v)**
- Nod izolat – d(v) = 0
- **Gradul interior al unui nod** dintr-un graf orientat **$d^-(v)$** – numărul arcelor care intră în acel nod
- **Gradul exterior al unui nod** dintr-un graf orientat **$d^+(v)$** – numărul arcelor care ies din acel nod

Ordin și dimensiune

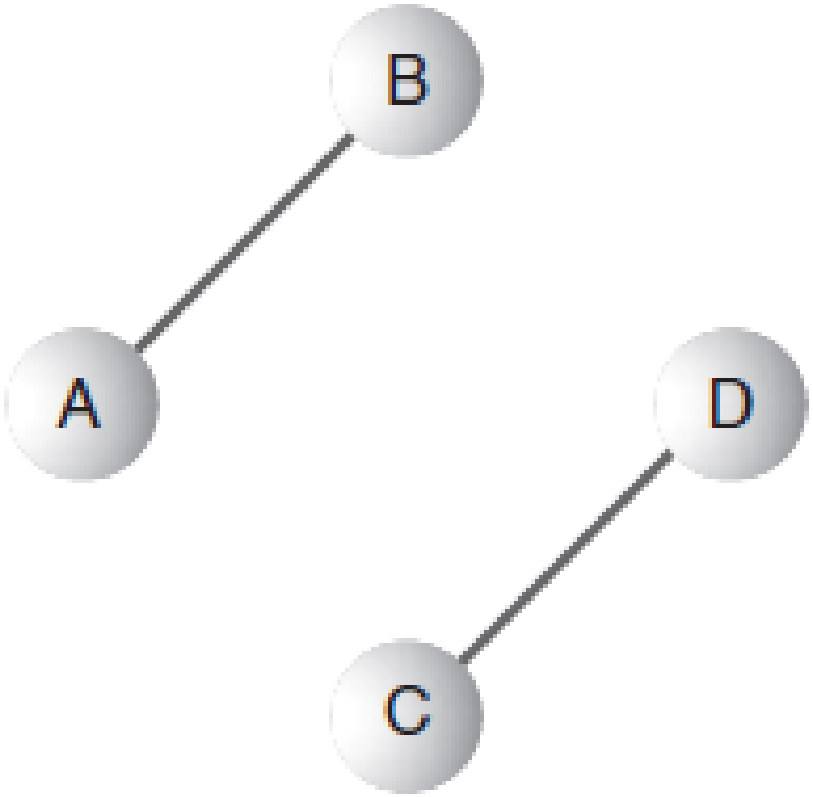
- **Ordinul unui graf** este egal cu numărul de noduri din graf **n = |V|**
- **Dimensiunea unui graf** este egală cu numărul de muchii **m = |E|**

Grafuri conexe și neconexe

- Un graf se numește **conex** dacă există cel puțin un drum de la fiecare nod până la fiecare alt nod
- Un graf este **conex** dacă, pentru orice pereche de noduri (v,w), există cel puțin o cale de la v la w sau de la w la v
- Un graf neconex este alcătuit din mai multe **componente conexe**



a) Connected Graph



b) Non-connected Graph

Observații

- Vârfurile A și B alcătuiesc una din componentele conexe, iar C și D alcătuiesc a doua componentă conexă
- Aceste grafuri sunt **grafuri neorientate**
- Muchiile nu au o **direcție**, ne putem deplasa pe ele în orice sens

Observații

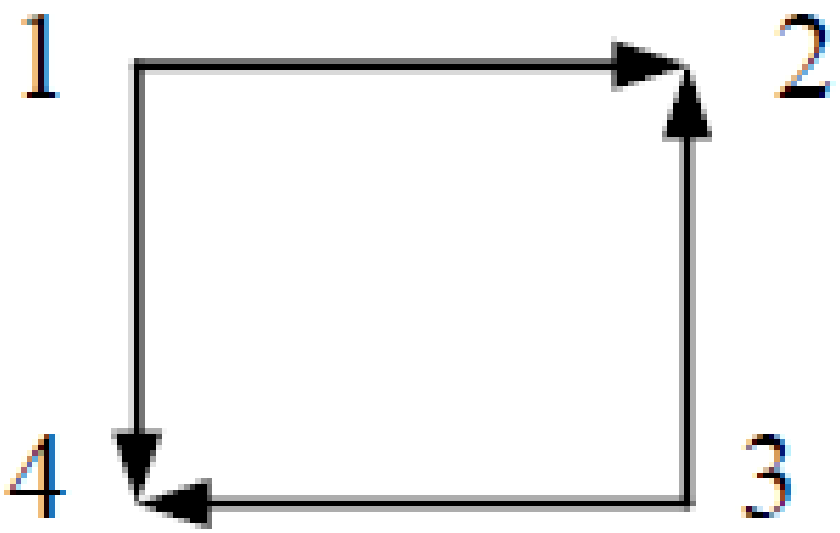
- Un graf în care ne putem deplasa într-o singură direcție de-a lungul unei muchii date se numește **graf orientat**
- Direcția de deplasare permisă este indicată printr-o săgeată plasată pe muchie

Componentă conexă

- O **componentă conexă** a unui graf (V,E) este un subgraf conex (V',E'), unde V' este o submulțime a lui V, iar E' este o submulțime a lui E
- Împărțirea unui **graf neorientat** în componente conexe este **unică**, dar un **graf orientat** poate fi partiționat în mai multe moduri în componente conexe

Exemplu

- Graful (1,2), (1,4), (3,2), (3,4) poate avea componentele conexe:
  - {1,2,4} și {3}
  - {3,2,4} și {1}



Graf orientat tare conex

- Un graf orientat este **tare conex** (**puternic conectat**) dacă, pentru orice pereche de noduri (v,w), există (cel puțin) o cale de la v la w și (cel puțin) o cale de la w la v
- Exemplu de graf tare conex - graf care conține un ciclu ce trece prin toate nodurile:
- (1,2), (2,3), (3,4), (4,1)

Observații

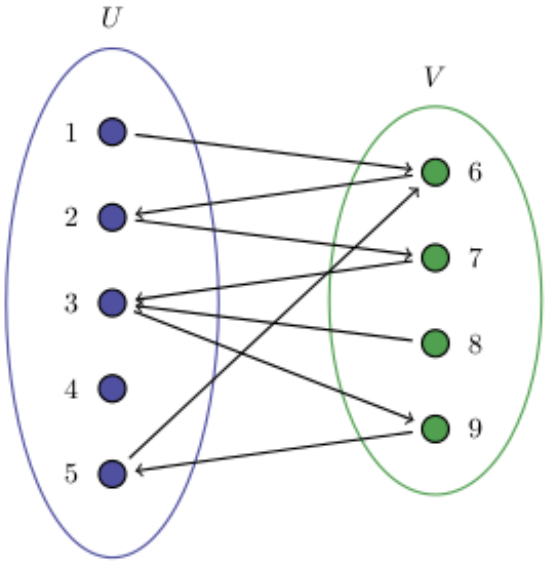
- Într-un graf orientat, arcul (v,w) pleacă din nodul v si intră în nodul w
- Acesta este diferit de arcul (w,v), care pleacă de la w la v
- Într-un graf neorientat poate exista o singură muchie între două vârfuri, notată (v,w) sau (w,v)

Observații

- Două noduri între care există un arc se numesc noduri **vecine** sau **adiacente**
- Într-un graf orientat ne putem referi la **succesorii** și **predecesorii** unui nod, respectiv la arce care **ies** și la arce care **intră** într-un nod

Graf bipartit

- Un **graf bipartit** este un graf în care mulţimea nodurilor **V** se partiţionează:
- $\mathbf{V = V_1 \cup V_2}$, $\mathbf{V_1 \cap V_2 = \emptyset}$ astfel încât
- $\forall$ **(v, w)** $\in$ **E**, **v** $\in$ $\mathbf{V_1}$ și **w** $\in$ $\mathbf{V_2}$ sau **w** $\in$ $\mathbf{V_1}$ și **v** $\in$ $\mathbf{V_2}$



Grafuri ponderate

- Într-un **graf ponderat**, muchiile au asociate **ponderi**, adică numere care reprezintă distanțe fizice între două vârfuri, sau timpul necesar parcurgerii drumului dintre două vârfuri, sau costul drumului dintre cele două vârfuri

Proprietăți

- **P1**: Fie un **graf neorientat G** cu **m muchii** și mulțimea de vârfuri **V**
- $\sum$**d(i)=2*m**
- **P2**: Fie un **graf orientat G** cu **m arce** și mulțimea de noduri **V**
- $\sum d^{-}(i)=m$
- $\sum d^{+}(i)=m$

Proprietăți

- **P3**: Fie un **graf G** cu **n** vârfuri și **m** muchii
- Dacă G este **neorientat** atunci **$m \leq n(n-1)/2$**
- Dacă G este **orientat** atunci **$m \leq n(n-1)$**
- **P4**: Fie un **graf G** cu **n** vârfuri și **m** muchii
- Dacă G este **conex** atunci **$m \geq n-1$**
- Dacă G este **arbore** atunci **$m = n-1$**
- Dacă G este o **pădure** atunci **$m \leq n-1$**

Probleme asociate grafurilor

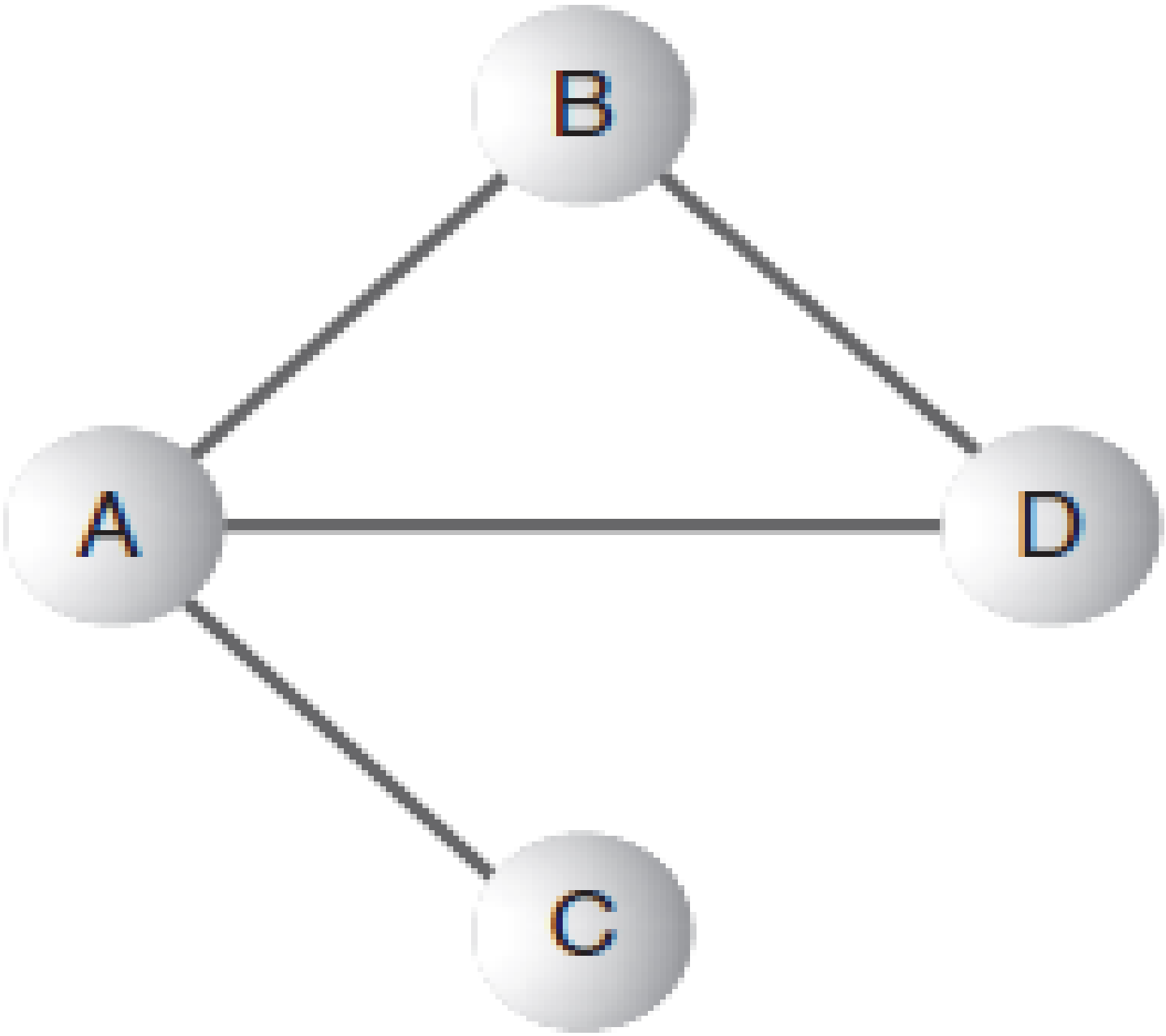
- **Ciclu hamiltonian –** ciclu elementar, în care se vizitează fiecare vârf din graf exact o dată
- **Ciclu eulerian** – un ciclu care vizitează fiecare muchie din graf exact o dată

Modalități de reprezentare

- Într-un graf, fiecare vârf poate fi conectat cu un număr arbitrar de alte vârfuri
- Pentru modelarea conectării nodurilor, cele mai frecvente două metode sunt:
  - **Matricea de adiacență**
  - **Liste de adiacență**

Matricea de adiacență

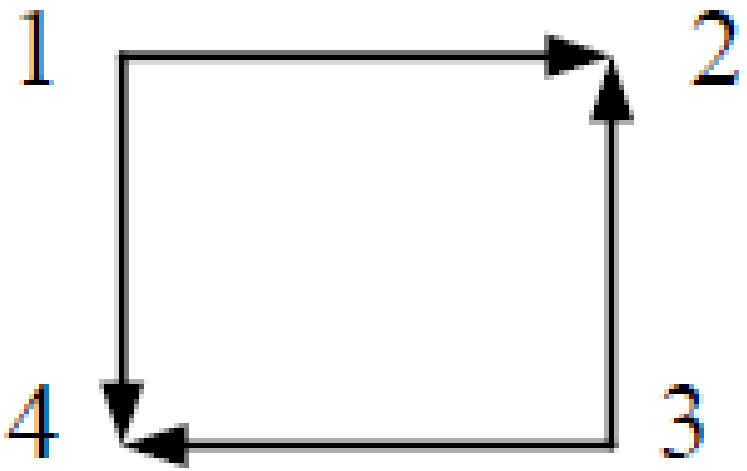
- O matrice de adiacență este un tablou bidimensional, în care elementele indică prezența unei muchii între două vârfuri
- Dacă graful are N vârfuri, matricea de adiacență este un tablou cu N*N elemente


B
A
D
C

	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	0
D	1	1	0	0

Exemplu

	1	2	3	4
1	0	1	0	1
2	0	0	0	0
3	0	1	0	1
4	0	0	0	0



Observații

- Vârfurile sunt trecute atât la capetele liniilor, cât și ale coloanelor din tabel
- O muchie între două vârfuri este indicată printr-o valoare 1 în tabel
- Valoarea 0 înseamnă absența unei muchii
- Legătura dintre un vârf și el însuși este reprezentată prin 0, formând **diagonala principală** a matricei

Observații

- Triunghiul superior al matricei (situat deasupra diagonalei principale) reprezintă imaginea în oglindă a celui inferior
- Ambele triunghiuri conțin aceeași informație – valabil doar la **grafuri neorientate**
- Această redundanță pare ineficientă, dar nu dispunem de o modalitate convenabilă de a crea un tablou triunghiular

Observații

- Dezavantajul matricei de adiacențe apare atunci când numărul de noduri din graf este mult mai mare ca numărul de arce, iar matricea este **rară** (cu peste jumătate din elemente nule)
- În aceste cazuri se preferă reprezentarea prin liste de adiacențe

Definirea structurii de graf

- Definirea structurii de graf printr-o matrice de adiacență alocată dinamic:
- typedef struct {
- int n, m;
- // n = număr de noduri, m = număr de arce
- int ** a; // adresa matrice de adiacență
- } Graf;

Observații

- **Succesorii** unui nod v sunt reprezentați de elementele nenule din **linia** v
- **Predecesorii** unui nod v sunt reprezentați de elementele nenule din **coloana** v
- În general, nu există arce de la un nod la el însuși, deci a[i][i] = 0

Operații uzuale cu grafuri

- 1) **Inițializare graf** cu număr dat de noduri:
  - *initG (Graf & g, int n)*
- 2) **Adăugare muchie** (**arc**) la un graf:
  - *addArc (Graf & g, int x, int y)*
- 3) **Verificare existența unui arc** de la un nod x la un nod y:
  - *int arc(Graf g, int x, int y)*
- 4) **Eliminare arc** dintr-un graf:
  - *delArc (Graf & g, int x, int y)*
- 5) **Eliminare nod** dintr-un graf:
  - *delNod (Graf & g, int x)*

Operații uzuale cu grafuri

```
// funcție de inițializare a grafului
void initG (Graf & g, int n) {
int i;
  g.n = n;
  g.m = 0;
  g.a = (int**) malloc( (n+1)*sizeof(int*));
  // vârfuri numerotate 1...n
  for (i = 1; i <= n; i++)
   g.a[i] = (int*) calloc( (n+1), sizeof(int));
   // linia 0 și coloana 0 sunt nefolosite
}
```

Operații uzuale cu grafuri

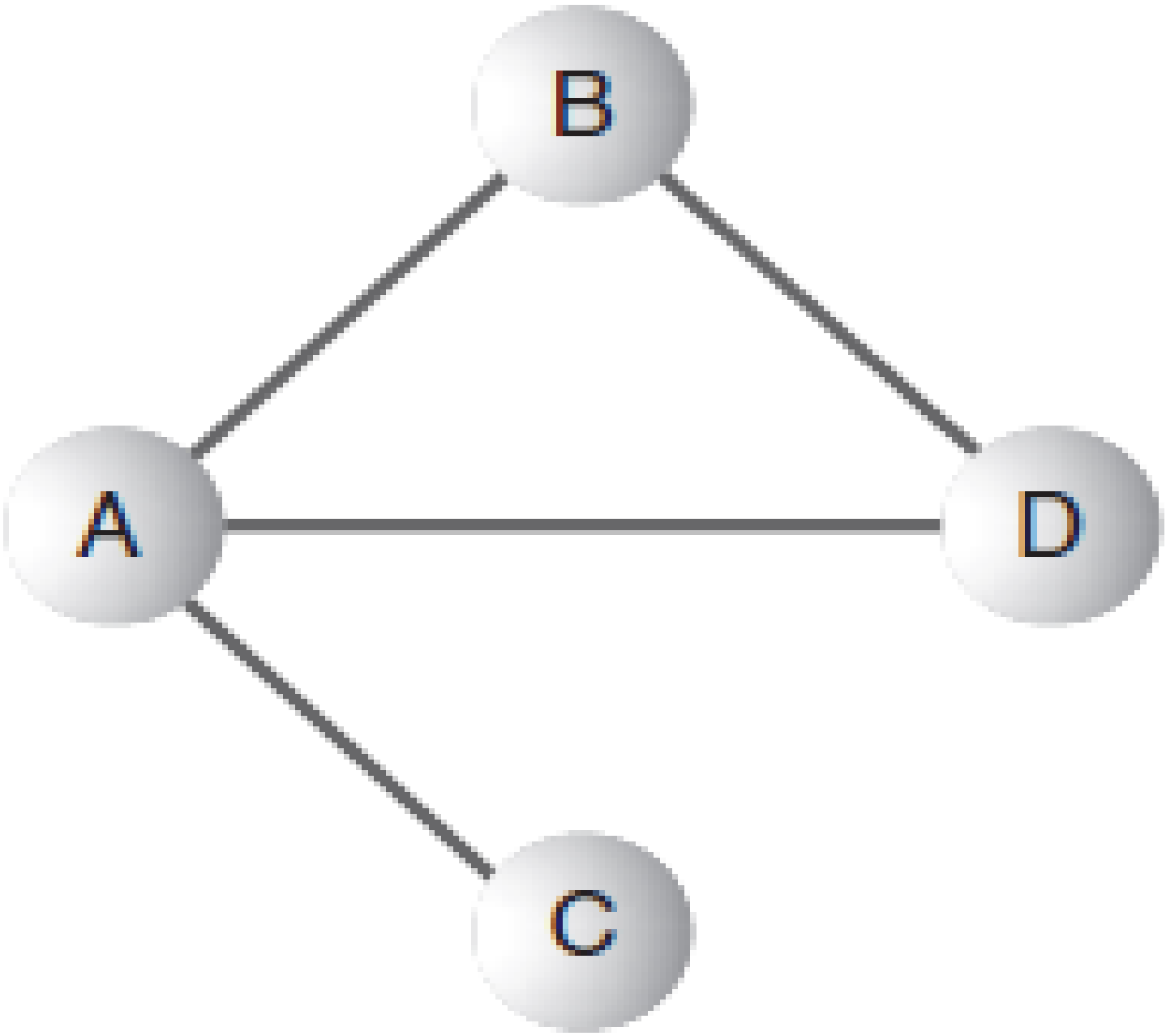
```
// funcție de adăugare a arcului (x,y) la graful g
void addArc (Graf & g, int x, int y) {
g.a[x][y]=1;
g.m++;
}

// funcție care întoarce arcul (x,y) din graful g
int arc (Graf g, int x, int y) {
return g.a[x][y];
}
```

```
// funcție de eliminare a arcului (x,y) din graful g
void delArc (Graf & g, int x, int y) {
        g.a[x][y] = 0;
        g.m--;
}
/*Eliminarea unui nod din graf ar trebui să modifice și
dimensiunile matricei, dar se elimină doar arcele ce
pleacă și vin din/în acel nod*/
// funcție de eliminare a nodului x din graful g
void delNod (Graf & g, int x) {
int i;
  for (i = 1; i <= g.n; i++) {
    delArc(g,x,i);
    delArc(g,i,x);
  }
```

Liste de adiacență

- Cealaltă soluție de reprezentare a muchiilor este utilizând o **listă de adiacență**
- O listă de adiacență reprezintă un tablou de liste (sau o listă de liste)
- Fiecare listă individuală conține vârfurile adiacente unui vârf dat


B
A
D
C

Vertex	List Containing Adjacent Vertices
A	B—>C—>D
B	A—>D
C	A
D	A—>B

Observații

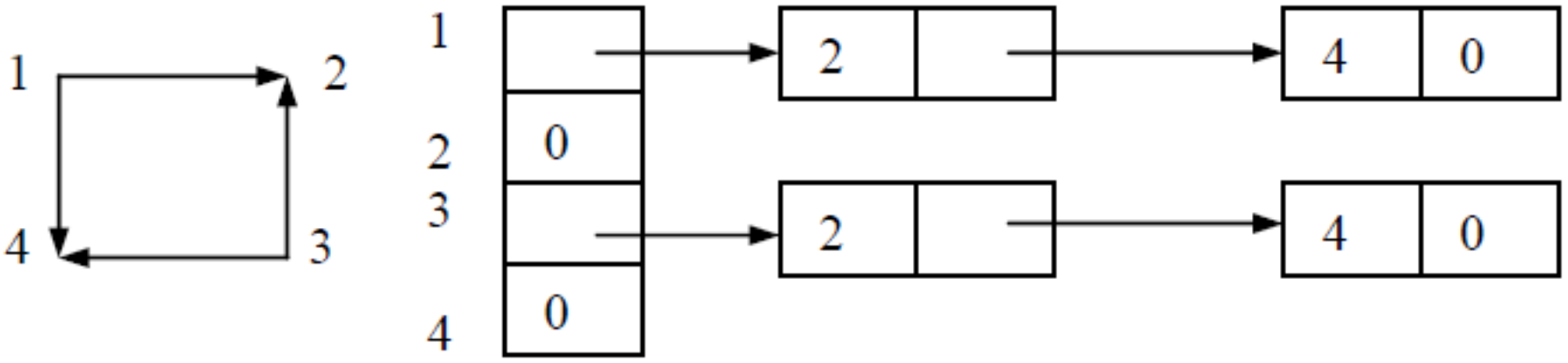
- Fiecare legătură din listă reprezintă un vârf din graf
- În exemplu, vârfurile sunt ordonate alfabetic, deși această ordine nu este necesară
- Nu trebuie confundat conținutul listelor de adiacență cu drumurile din graf

Observații

- Lista tuturor arcelor din graf este împărțită în mai multe subliste, câte una pentru fiecare nod din graf
- Listele de noduri vecine pot avea lungimi diferite și se preferă implementarea lor prin liste înlănțuite

Exemplu

- Reprezentarea grafului orientat (1,2),(1,4),(3,2),(3,4) printr-un tablou de pointeri la liste de adiacență



Observații

- Ordinea nodurilor într-o listă de adiacență nu este importantă și de aceea se poate adăuga mereu la începutul listei de noduri vecine

Definirea structurii de graf

```
typedef struct nod {
    int val;                    // număr nod
    struct nod * leg;
    // adresa listei de succesori pentru un nod
} * pnod ;                      // pnod este un tip pointer
typedef struct {
    int n ;                     // număr de noduri în graf
    pnod * v;
    // tablou de pointeri la liste de succesori
} Graf;
```

Operații uzuale cu grafuri

```
// funcție de inițializare a grafului
void initG (Graf & g, int n) {
g.n = n;       // număr de noduri
g.v = (pnod*) calloc(n + 1, sizeof(pnod));
// inițializare pointeri cu 0
}
```

Operații uzuale cu grafuri

```
// funcție de adăugare a arcului (x,y)
void addArc (Graf & g, int x, int y) {
pnod nou = (pnod) malloc (sizeof(nod));
   nou→val = y;
   nou→leg = g.v[x];
   g.v[x] = nou;
   // se adaugă la începutul listei de adiacență
}
```

Operații uzuale cu grafuri

```
// funcție care testează dacă există arcul (x,y) în graful g
int arc (Graf g, int x, int y) {
  pnod p;
  for (p = g.v[x]; p != NULL; p = p→leg)
     if ( y == p→val) return 1;
  return 0;
}
```

Observații

- Reprezentarea unui graf prin liste de adiacență pentru fiecare vârf asigură cel mai bun timp de explorare a grafurilor (timp proporțional cu suma dintre numărul de vârfuri și numărul de muchii din graf), iar explorarea apare ca operație în mai mulți algoritmi pe grafuri

Observații

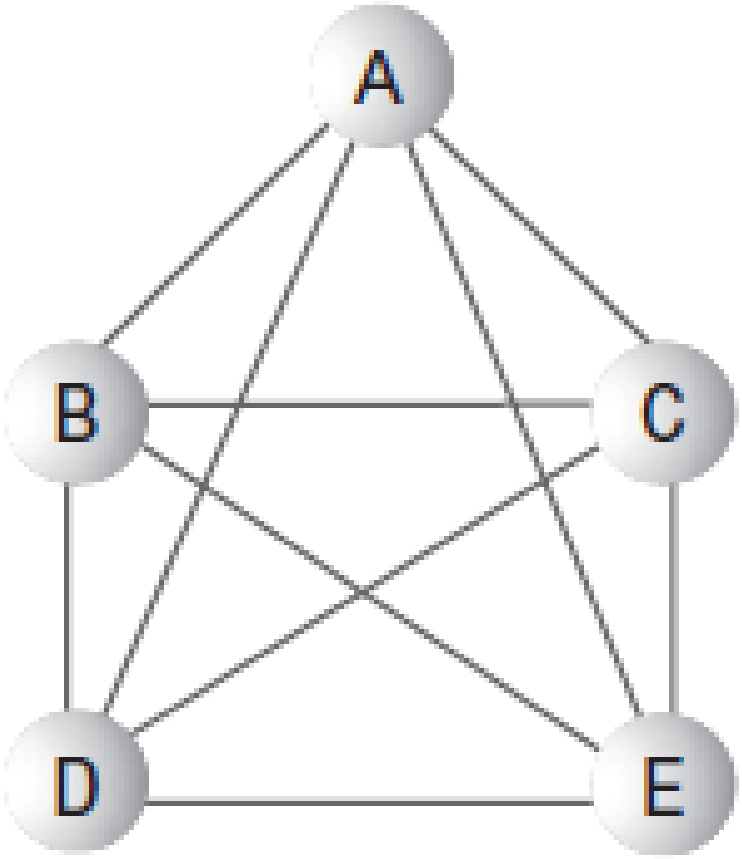
- Pentru un graf neorientat, fiecare muchie (**x,y**) este memorată de două ori: **y** în lista de adiacență a lui **x** și **x** în lista de adiacență a lui **y**
- Pentru un graf orientat, listele de adiacență sunt de obicei liste de succesori, dar pentru unele aplicații ne interesează predecesorii unui nod
- Lipsa de simetrie poate fi un dezavantaj al listelor de adiacență pentru reprezentarea grafurilor orientate

Arbore liber

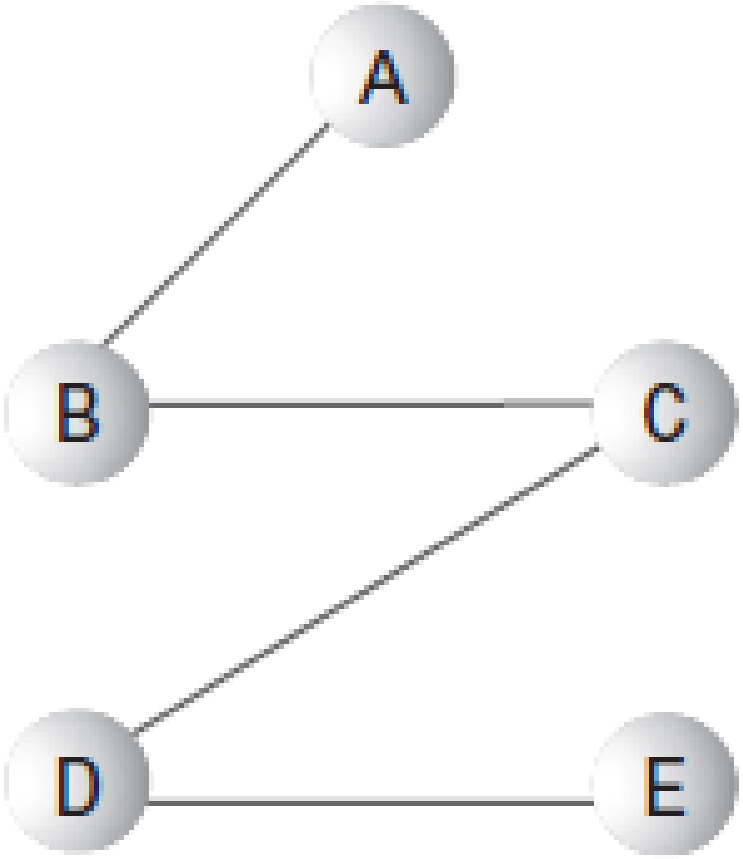
- Un **arbore liber** este un graf neorientat aciclic
- Într-un arbore liber nu există un nod special rădăcină
- Într-un arbore, fiecare vârf are un singur părinte (predecesor), deci se poate reprezenta arborele printr-un tablou de noduri părinte

Arbore minim de acoperire

- Un **arbore minim de acoperire** reprezintă un arbore cu **numărul minim de muchii** necesare pentru conectarea tuturor vârfurilor



a) Extra Edges



b) Minimum Number of Edges

Observații

- În prima figură, sunt cinci vârfuri conectate prin mai multe muchii
- Aceleași vârfuri sunt conectate în a doua figură printr-un număr minim de muchii
- Acesta reprezintă un **arbore minim de acoperire**

Observații

- Pentru o mulțime dată de vârfuri, este posibilă construcția mai multor arbori minimi de acoperire
- Arborele din a doua figură conține muchiile AB, BC, CD și DE
- Un alt arbore este cel care conține muchiile AC, CE, ED și DB

Arbore minim de acoperire

- Pentru un graf conex neorientat **G = (V, E)**, se numeşte **arbore minim de acoperire** al lui **G**, un **subgraf G’ = (V, E’)**, care conţine toate vârfurile grafului **G** şi o submulţime **minimă** de muchii **E’** $\subseteq$ **E**, cu proprietatea că uneşte toate vârfurile şi nu conţine cicluri
- Cum **G’** este conex şi aciclic, el este arbore

Observații

- Numărul **E’** de muchii dintr-un arbore minim de acoperire este cu o unitate mai mic decât numărul **V** de vârfuri
- **E’ = V - 1**
- Când se execută o **parcurgere în adâncime** și se **memorează muchiile traversate**, se obține un **arbore minim de acoperire**