Capitolul 12. Grafuri

Grafuri neorientate

Definiţie: Se numeşte **graf neorientat** (G) o pereche ordonată de mulţimi (X,U), unde X este o mulţime finită şi nevidă de elemente, iar U o mulţime de perechi formate cu elemente distincte din mulţimea X.

G=(X,U)
X={1,2,3,4,5,6,7}
U={(1,2),(2,3),(3,7),(7,5),(1,6),(7,1),(2,5),(2,7)}

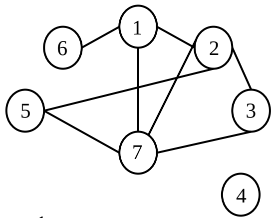


figura1 – reprezentare grafică

Terminologie:
Elementele mulţimii X se numesc **noduri** sau **vârfuri.** Mulţimea X se mai numeşte şi **mulţimea nodurilor** sau **vârfurilor.**
Ordinul grafului reprezintă numărul de noduri ale grafului.
Elementele mulţimii U se numesc **muchii.** Mulţimea U se mai numeşte şi **mulţimea muchiilor.**
Numim **noduri adiacente** orice pereche de noduri care formează o muchie. Fiecare din cele două noduri spunem, că sunt **incidente** cu muchia pe care o formează.

Exemplu:
Nodul **1** este adiacent cu nodurile **2**,**6**,**7**; nodul **5** este adiacent cu nodurile **2**,**7** etc.
Nodurile **3**,**7** sunt incidente cu muchia (3,7). *(fig 1)*

Nodurile vecine ale unui nod sunt toate nodurile adiacente cu el.
Se numesc **muchii incidente** două muchii care au o extremitate comună.

Exemplu:
Muchiile (1,2) şi (2,7) sunt incidente având ca extremitate comună nodul **2**. *(fig 1)*

Definiţie: Gradul unui nod x al grafului G este egal cu numărul muchiilor incidente cu nodul şi se notează cu d(x).

Exemplu:
d(1)=3;d(2)=4;d(3)=2 etc. *(fig 1)*

Terminologie:
Se numeşte **nod terminal** un nod care are gradul egal cu 1 .
Se numeşte **nod izolat** un nod care are gradul 0.

Exemplu:
Nodul **6** este nod terminal.
Nodul **4** este nod izolat. *(fig 1)*

Teoreme:
1. Numărul total de grafuri neorientate cu n noduri este $2^{C_n^2} = 2^{\frac{n*(n-1)}{2}}$
2. Suma gradelor tuturor nodurilor unui graf nerorientat este egală cu dublul numărului de muchii.
$$\sum_{i=1}^{n} d_i = 2*m$$
3. Dacă graful G neorientat are **n** noduri, n>2, atunci **cel puţin 2 noduri au acelaşi grad**.
4. Pentru orice graf neorientat numărul nodurilor de grad impar este **par**.

5. Numărul minim de muchii pe care trebuie să le aibă un graf neorientat cu n noduri, ca să nu existe noduri izolate este $[\frac{n+1}{2}]$.

Grafuri orientate

Definiţie: Se numeşte **graf orientat** sau **digraf** (G) o pereche ordonată de mulţimi (X,U), unde X este o mulţime finită şi nevidă de elemente, iar U o mulţime de perechi ordonate formate cu elemente distincte din mulţimea X.

$G_1=(X_1,U_1)$
$X_1=\{1,2,3,4,5,6,7\}$
$U_1=\{(1,2),(2,3),(3,7),(5,7),(1,6),(7,1),(5,2),(2,7)\}$

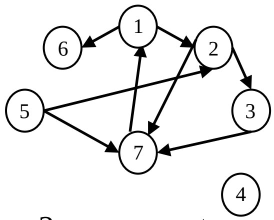


figura 2 – reprezentare grafică

Terminologie:
Elementele mulţimii U se numesc **arce.** Mulţimea U se mai numeşte şi **mulţimea arcelor**.
Numim **vârfuri adiacente** orice pereche de vârfuri care formează un arc. Fiecare din cele două vârfuri spunem, că sunt **incidente** cu arcul pe care îl formează.

Exemplu:
Nodul **1** este adiacent cu nodurile **2**,**6**,**7**; nodul **5** este adiacent cu nodurile **2**,**7** etc.
Nodurile **3**,**7** sunt incidente cu muchia (3,7). *(fig 2)*

Pentru arcul (x,y) spunem că x este **extremitatea iniţială** iar y este **extremitatea finală**.

Exemplu:
Arcul (5,2) are vârful **5** ca extremitate iniţială şi vârful **2** ca extremitate finală. *(fig 2)*

Se numesc **arce incidente** două arce care au o extremitate comună.
Se numeşte **succesor** al vârfului x orice vârf la care ajunge un arc care iese din vârful x.
Mulţimea succesorilor vârfului x este formată din mulţimea vârfurilor la care ajung arce care ies din vârful x şi se notează $\Gamma^+(x)$.
Mulţimea muchiilor ce îl pe x ca extremitate iniţială se notează $\omega^+(x)$.

Exemplu:
Vârfurile **2** şi **7** sunt succesori ai vârfului **5.**
$\Gamma^+(2)=\{3,4\}$
$\omega^+(2)=\{(2,4),(2,3)\}$ *(fig 3)*

Se numeşte **predecesor** al vârfului x orice vârf de la care intră un arc în vârful x.
Mulţimea predecesorilor vârfului x este formată din mulţimea vârfurilor de la care ajung arce care intră în vârful x şi se notează $\Gamma^-(x)$.
Mulţimea muchiilor ce îl pe x ca extremitate finală se notează $\omega^-(x)$.

Exemplu:
Vârfurile **2**,**3** şi **5** sunt predecesori ai vârfului **7.**
$\Gamma^-(6)=\{1\}$
$\omega^-(3)=\{(1,3),(2,3)\}$ *(fig 3)*

Nod sursă al grafului este nodul care are mulţimea succesorilor formată din toate celelalte noduri mai puţin el iar mulţimea predecesorilor săi este vidă.

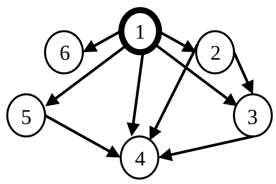


figura 3

Exemplu:
Vârful **1** din graful din figura 3 este vârf sursă.

Nod destinaţie al grafului este nodul care are mulţimea predecesorilor formată din toate celelalte noduri mai puţin el iar mulţimea succesorilor săi este vidă.

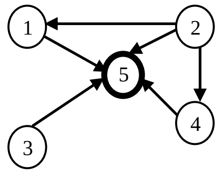


figura 4

Exemplu:
Vârful **5** din graful din figura 4 este vârf destinaţie.

Definiţie: Gradul intern unui nod x al grafului G este egal cu numărul arcelor care intră în nodul x şi se notează cu $d^-(x)$.**Gradul extern** unui nod x al grafului G este egal cu numărul arcelor care ies din nodul x şi se notează cu $d^+(x)$.

Exemplu:
$d^+(1)=1$ $d^-(1)=1$ $d^+(3)=1$ $d^-(3)=0$ *(fig 4)*; $d^+(5)=1$ $d^-(5)=1$ $d^+(4)=0$ $d^-(4)=4$ *(fig 3)*

Terminologie:
Se numeşte **nod terminal** un nod care are suma gradelor egală cu 1.
Se numeşte **nod izolat** un nod care suma gradelor egală cu 0.

Exemplu:
Vârful 3 este terminal în graful din figura 4.
Teoreme:

1. Numărul total de grafuri orientate care se pot forma cu **n** noduri este $4^{C_n^2}=4^{\frac{n*(n-1)}{2}}$.
2. Într-un graf orientat cu **n** vârfuri, suma gradelor interne ale tuturor nodurilor este egală cu suma gradelor exterioare ale tuturor nodurilor şi cu numărul de arce.

Metode de reprezentare:
Considerăm un graf G cu **n** noduri numerotate de la 1 la **n** şi **m** muchii/arce.

1. **Matricea de adiacenţă:** este o matrice cu **n** linii şi **n** coloane ale cărei elemente sunt definite astfel

$$a_{i,j}=\begin{cases}1, \text{ daca nodul i este adiacent cu nodul j}\\0, \text{ daca nodul i nu este adiacent cu nodul j}\end{cases}$$

Observaţie: Într-un graf neorientat numărul de valori 1 din matricea de adiacenţă reprezintă dublul numărului de muchii ale grafului. Într-un graf orientat numărul de valori 1 din matricea de adiacenţă reprezintă numărul de arce ale grafului.

Exemplu:
Pentru graful neorientat din figura 5 matricea de adiacenţă este:

0	0	1	1	0	1
0	0	1	1	0	1
1	1	0	0	0	1
1	1	0	0	1	0
0	0	0	1	0	0
1	1	1	0	0	0

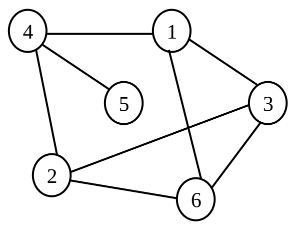


figura 5

Pentru graful orientat din figura 6 matricea de adiacenţă este:

0	0	0	1	0	0
1	0	0	0	1	0
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	1	1	0	0

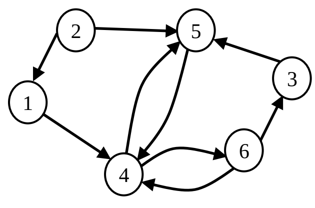


figura 6

2. **Matricea de incidenţă:**
Pentru graful neorientat G este o matrice cu **n** linii şi **m** coloane ale cărei elemente sunt definite astfel:

$$a_{i,j} = \begin{cases} 1, \text{ daca nodul i este incident cu muchia j} \\ 0, \text{ daca nodul i nu este incident cu muchia j} \end{cases}$$

Exemplu:
Pentru graful neorientat din figura 5 matricea de incidenţă este:

1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0
1	0	0	0	0	1	0	1
0	1	0	1	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	1

m1=(1,3) m5=(4,2)
m2=(1,4) m6=(2,3)
m3=(1,6) m7=(2,6)
m4=(4,5) m8=(3,6)

Matricea de incidenţă a unui graf orientat este o matrice cu **n** linii şi **m** coloane ale cărui elemente sunt definite astfel:

$$a_{i,j} = \begin{cases} 1, \text{ daca nodul i este extremitate initiala a arcului j} \\ -1, \text{ daca nodul i este extremitatea finala a arcului j} \\ 0, \text{ daca nodul i nu este extremitate a arcului j} \end{cases}$$

Obs unii autori iau în considerare invers 1și -1
Exemplu:
Pentru graful neorientat din figura 5 matricea de incidenţă este:

1	-1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	0	0	1	0	0	0	-1	0
-1	0	0	0	1	1	-1	0	-1
0	0	-1	-1	-1	0	1	0	0
0	0	0	0	0	-1	0	1	1

m1=(1,4) m6=(4,6)
m2=(2,1) m7=(5,4)
m3=(2,5) m8=(6,3)
m4=(3,5) m9=(6,4)
m5=(4,5)

3. **Lista muchiilor/arcelor:** este formată din **m** elemente care conţin, fiecare, câte o pereche de noduri, x şi y care formează o muchie/arc.
În implementare se pot utiliza fie o matrice de dimensiuni *2Xm* sau *mX2*, un vector de structuri, sau liste liniare alocate dinamic ce memorează extremităţile muchiilor/arcelor.
4. **Lista de adiacenţă:** este formată din listele $L_i(1 \le i \le n)$ care conţin toţi vecinii unui nod x_i dacă graful G este neorientat, respectiv toţi succesorii nodului x_i dacă graful G este orientat.

Exemplu:

Listele de adiacenţă pentru graful din figura 5:
1: 3, 4, 6
2: 3, 4, 6
3: 1, 2, 6
4: 1, 2, 5
5: 4
6: 1, 2, 3

Listele de adiacenţă pentru graful din figura 6:
1: 4
2: 1, 5
3: 5
4: 5, 6
5: 4
6: 3, 4

Grafuri speciale

Definiţie: Graful **G s**e numeşte **graf nul** dacă mulţimea U este vidă, adică graful nu are muchii.
Definiţie: Un graf cu **n** noduri se numeşte **complet** dacă are proprietatea că, oricare ar fi două noduri ale grafului, ele sunt adiacente.

Teoreme:

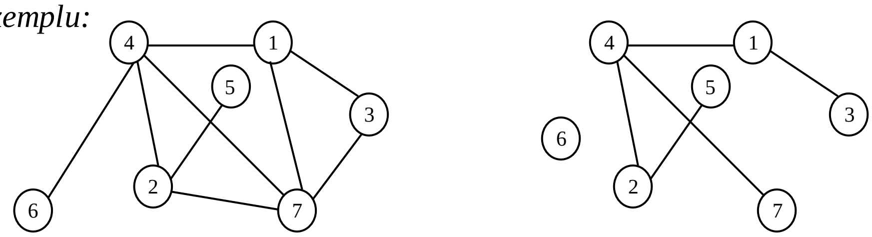
Numărul de muchii al uni **graf neorientat complet** este $\frac{n(n-1)}{2}$.

Numărul de **grafuri orientate complete** care se pot construi cu **n** noduri este egal cu $3^{C_n^2}$

Grafuri derivate dintr-un graf

Definiţie: Fie graful G=(X,U) şi mulţimea V⊆U. Graful G_p=(X,V) se numeşte **graf parţial** al grafului G.

Exemplu:



graful iniţial — graful parţial obţinut prin eliminarea muchiilor (4,6) (2,7) (3,7) (1,7)

figura 7

Definiţie: Fie graful G=(X,U). Graful G_s=(Y,V) se numeşte **subgraf** al grafului G dacă Y⊆U şi muchiile/arcele din mulţimea V sunt toate muchiile/arcele din mulţimea U care au ambele extremităţi în Y.

Exemplu:

graful iniţial — subgraful obţinut prin eliminarea vârfurilor 1 şi 6

figura 8

Definiţie: Un graf G_c se numeşte **complementar** al lui G daca are proprietatea că 2 noduri sunt adiacente în graful G_c dacă şi numai dacă nu sunt adiacente în G.

Exemplu:
Următoarele grafuri sunt complementare:

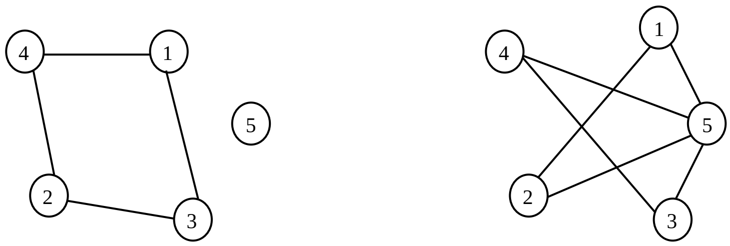


figura 9

Teoreme:

1. Numărul de **grafuri parţiale** ale unui graf cu **m** muchii este egal cu 2^m.
2. Numărul de **subgrafuri** ale unui graf cu **n** noduri este egal cu $2^n - 1$.

Conexitate

Definiţie: Numim **lanţ** o succesiune de noduri care au proprietatea că, oricare ar fi două noduri succesive, ele sunt adiacente.
Definiţie: Numim **ciclu** un lanţ în care toate muchiile/arcele sunt distincte două câte două şi primul nod coincide cu ultimul.
Definiţie: Un graf fără cicluri se numeşte **graf aciclic**.
Definiţie: Numim **drum** o succesiune de noduri care au proprietatea că oricare ar fi două noduri succesive ele sunt legate printr-un arc.
Definiţie: Numim **circuit** un drum în care toate arcele sunt distincte două câte două şi ale cărui extremităţi coincid.

Terminologie:
Lungimea unui lanţ reprezintă numărul de muchii/arce din care este format
Lanţul simplu este lanţul care conţine numai muchii/arce distincte.
Lanţul compus este lanţul care nu este format din muchii/arce distincte.
Lanţul elementar este lanţul care conţine numai noduri distincte
Ciclu elementar este un ciclu în care toate nodurile sunt distincte două câte două.
Lungimea unui drum este dată de numărul de arce care îl compun.
Drumul simplu este drumul care conţine numai arce distincte.
Drumul compus este drumul care nu este format numai din arce distincte.
Drumul elementar este drumul în care nodurile sunt distincte două câte două.
Circuitul elementar este circuitul în care toate nodurile sunt distincte două câte două cu excepţia primului şi a ultimului care coincid.

Exemplu:

L_1=(1,**5**,3,2,**5**) – lanţ
L_2=(6,1,3,2,5,7) – lanţ elementar
C_1=(7,5,**2**,3,4,**2**,7) – ciclu
C_2=(6,3,4,1,7,6) – ciclu elementar

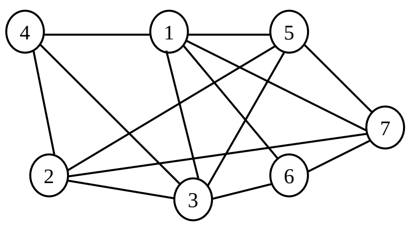


figura 10

L_1=(1,**5**,3,6,**5**,4) – lanţ
L_2=(6,1,5,4,2) – lanţ elementar
C_1=(1,6,**5**,4,2,**5**,1) – ciclu
C_2=(4,5,2,4) – ciclu elementar
D_1=(1,5,2,4,5,6) – drum
D_2=(3,6,2,4,5) – drum elementar
C_3=(**2**,4,**2**,5,**2**) – circuit
C_4=(4,5,3,6,2,4) – circuit elementar

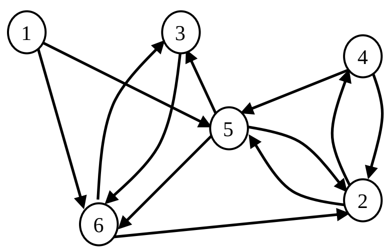


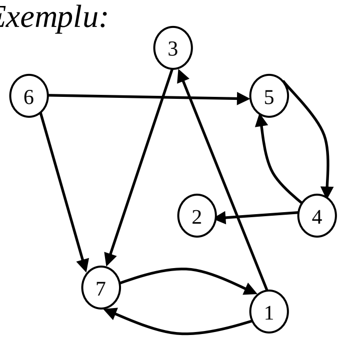
figura 11

Teoreme:
1. Dacă un graf conţine un lanţ între 2 noduri x şi y atunci conţine un lanţ elementar între nodurile x şi y.
2. Dacă un graf conţine un drum între 2 noduri x şi y atunci conţine un drum elementar între nodurile x şi y.
3. Dacă un graf conţine un ciclu atunci conţine şi un ciclu elementar.
4. Dacă un graf conţine un circuit atunci conţine şi un circuit elementar

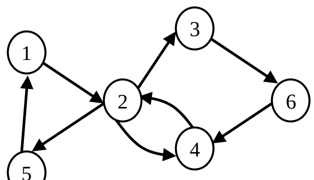
Definiţie: Un graf G se numeşte **graf conex** dacă are proprietatea că, pentru orice pereche de noduri diferite între ele, există un lanţ care să le lege.

Definiţie: Dacă un graf G nu este conex, se numeşte **componentă conexă a grafului** un subgraf conex al său, maximal în raport cu această proprietate (conţine numărul maxim de noduri din G care au proprietatea că sunt legate cu un lanţ).
Un graf conex are o singură componentă conexă.
Definiţie: Un graf orientat G se numeşte **graf tare conex** dacă are proprietatea că pentru orice pereche de noduri diferite între ele, există un drum care să le lege.
Definiţie: Dacă un graf orientat G nu este tare conex, se numeşte **componentă tare conexă** a grafului, un subgraf tare conex al său, maximal în raport cu această proprietate (conţine numărul maxim de noduri din G care au proprietatea că sunt legate printr-un drum).

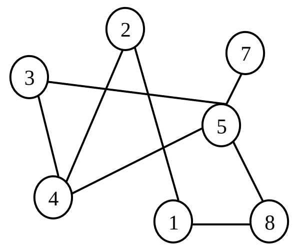
Exemplu:



GT1

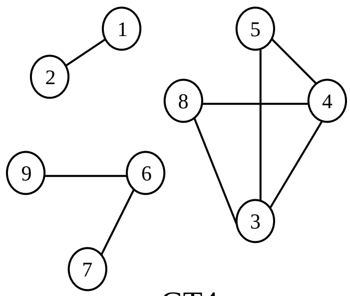


GT2



GT3

Graful GT1 este graf conex dar nu este tare conex; Componentele tare conexe din sunt determinate de submulţimile {2}, {6}, {4,5} şi {1,3,7}
Graful GT2 este tare conex
Graful GT3 este conex
Graful GT4 nu este conex; componentele conexe sunt determinate de submulţimile {1,2}, {3,4,5,8} şi {6,7,9}



GT4

Un graf tare conex are o singură componentă tare conexa (graful însuşi).
Componenta tare conexă din care face parte un nod este dată de mulţimea formată din acel nod reunită cu intersecţia dintre mulţimea predecesorilor acelui nod şi mulţimea succesorilor acelui nod.
Componenta conexă care conţine vârful x_i va fi mulţimea $\{x_i\} \cup (P(x_i) \cap S(x_i))$
Graful componentelor tare conexe ale unui graf care nu este tare conex G se obţine prin reducerea fiecărei componente conexe la un nod.

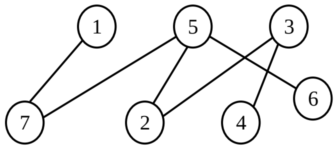
Teoreme:
1. Numărul minim de muchii necesare ca pentru ca un graf neorientat să fie conex este **n-1**.
2. Un graf conex cu **n** noduri şi n-1 muchii este **aciclic** şi maximal cu această proprietate.
3. Dacă un graf neorientat conex are **n** noduri şi **m** muchii, numărul de muchii care trebuie eliminate, pentru a obţine un **graf parţial conex aciclic** este **m-n+1**.
4. Dacă un graf are **n** noduri , **m** muchii şi **p** componente conexe , numărul de muchii care trebuie eliminate pentru a obţine un **graf parţial aciclic (arbore)** este egal cu **m-n+p**.
5. Numărul maxim de muchii dintr-un graf neorientat cu **n** noduri şi **p** componente conexe este $\frac{(n-p)*(n-p+1)}{2}$.

Problema: Pentru a determina numărul maxim de componente conexe/noduri izolate dintr-un graf neorientat cu n noduri si m muchii se determina componenta conexa cu numar minim p de noduri care poate contine cele m muchii. Se determina p minim pentru care p*(p-1)/2>=m. Numărul maxim de noduri izolate va fi n-p, iar numărul maxim de componente conexe va fi n-p+1
Grafuri speciale
Definiţie: Graful G se numeşte **graf bipartit** dacă există 2 mulţimi nevide de noduri A şi B care au următoarele proprietăţi: $A \cup B = X$ şi $A \cap B = \emptyset$ şi orice muchie (arc) din mulţimea U are o extremitate în mulţimea de noduri A şi o altă extremitate în mulţimea de noduri B.

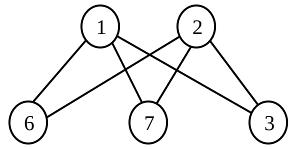
Definiţie: Graful bipartit G se numeşte **graf bipartit complet** dacă pentru orice nod x_i care aparţine lui A şi orice nod x_j care aparţine lui B - există o muchie (un arc) formată din cele 2 noduri care aparţine mulţimii U ($[x_i, x_j] \in U$).

Exemplu:

Graful GT1 este graf bipartit. A={1,3,5} B={2,4,6,7}
Graful GT2 este bipartit complet. A={1,2} B={3,6,7}

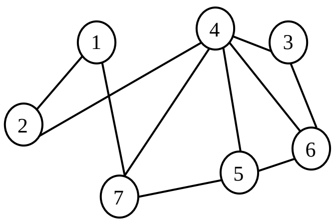


GT1

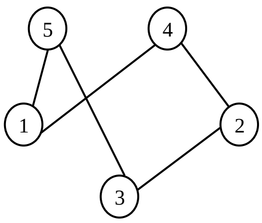


GT2

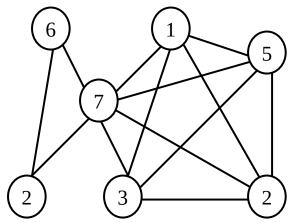
Definiţie:. Numim **lanţ hamiltonian** un lanţ elementar ce conţine toate nodurile grafului.
Definiţie:. Numim **ciclu hamiltonian** un ciclu elementar ce conţine toate nodurile grafului.
Definiţie: Un graf ce conţine un ciclu hamiltonian se numeşte **graf hamiltonian**.
Definiţie: Numim **ciclu eulerian** un ciclu ce conţine toate muchiile grafului.
Definiţie: Un graf ce conţine un ciclu eulerian se numeşte **graf eulerian**.
Definiţie: Un graf orientat în care, între oricare 2 noduri există un singur arc şi numai unul, se numeşte **graf turneu.**
Exemplu:



GT1



GT2

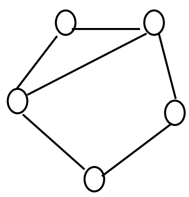


GT3

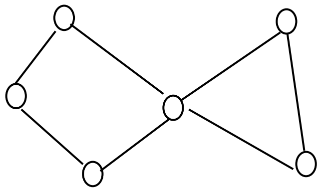
Graful GT1 este hamiltonian dar nu este eulerian.
Graful GT2 este hamiltonian şi eulerian.
Graful GT3 este eulerian dar nu şi hamiltonian.

Teoreme:
1. Un graf cu mai mult de 2 noduri este **hamiltonian** dacă gradul fiecărui nod este $\geq n/2$.
2. Un graf ce nu conţine grafuri izolate este **eulerian** dacă şi numai dacă este conex şi gradele tuturor nodurilor sunt pare.
3. Numărul de cicluri hamiltoniene dintr-un graf complet cu **n** noduri este $\frac{(n-1)!}{2}$.
4. Orice graf turneu conţine un drum elementar care trece prin toate nodurile grafului.
5. Pentru orice graf turneu, există un nod x, astfel încât toate nodurile y $\neq$ x sunt accesibile din x pe un drum care conţine un arc sau două arce.

Exemple:
Graf care este hamiltonian și nu este eulerian



Graf care este eulerian și nu este hamiltonian



Arbori

Definiţie: Un **arbore liber** este un graf neorientat conex şi fără cicluri.
Definiţie: Se numeşte **arbore cu rădăcină** un arbore în care există un nod privilegiat numit nod rădăcină.

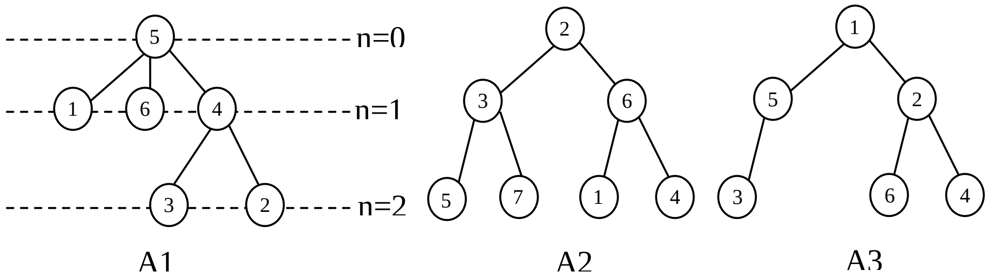


figura 12

Definiţie: Se numeşte **arborescenţă** sau **structură arborescentă** un arbore cu rădăcină în care s-a stabilit nodul rădăcină.
Definiţie: Un **arbore poziţional** este un arbore cu rădăcină în care este precizată poziţia fiecărui fiu.
Definiţie: Se numeşte **arbore binar strict** un arbore care are proprietatea că fiecare nod, cu excepţia nodurilor terminale, are exact 2 descendenţi.
Definiţie: Se numeşte **arbore binar echilibrat** un arbore binar care are proprietatea că diferenţa dintre înălţimile celor doi subarbori ai oricărui nod este cel mult 1.
Definiţie: Se numeşte **arbore binar complet** un arbore binar strict care are toate nodurile terminale pe acelaşi nivel.

Exemplu:
Arborele A2 din figura 12 este arbore binar strict şi complet.
Arborii A2 şi A3 din aceeaşi figură sunt echilibraţi.

Terminologie:
Nodul rădăcină mai este numit şi vârf.
Într-un nod intră o singură muchie care îl leagă de un alt nod numit **părinte** sau **predecesor.**
Dintr-un nod pot să iasă niciuna, una sau mai multe muchii care îl leagă de alte noduri numite **fii** sau **succesor**.
Nodurile fără succesori se numesc **frunze** sau **noduri terminale.**
Două noduri care descind din acelaşi nod tată se numesc noduri **frate.**
Ordinul unui nod este dat de numărul de descendenţi direcţi.
Nodurile sunt organizate pe **niveluri**. Rădăcina se găseşte pe nivelul 0, descendenţii ei pe nivelul 1, descendenţii acestora pe nivelul 2 etc.
Înălţimea unui arbore este dată de numărul maxim dintre nivelurile nodurilor terminale(este egală cu lungimea celui mai lung lanţ de la rădăcină până la un vârf terminal).

Exemplu:
Rădăcinile arborilor A1, A2, A3 sunt în ordine 5, 2, 1.
Predecesorul nodului 4 din A3 este 2.
Succesorii lui 5 din A1 sunt 1, 6, 4.
Nodurile 5, 7, 1 , 4 sunt frunze în A2.

Ordinul nodului 2 în A3 este 2.
Toţi arborii din figura 12 au înălţimea 2.

Teoreme:
1. Orice arbore cu n noduri are (n-1) muchii.
2. Un arbore binar strict care are **n** noduri terminale are în total (2*n-1) noduri.
3. Un arbore binar complet care are **n** noduri terminale are în total (2*n-1) noduri.

Vectorul de taţi
Este o modalitate de reprezentare a arborilor cu ajutorul unui tablou unidimensional definit după cum urmează:

$$t_i = \begin{cases} 0, i \quad este \quad radacina \\ x, x \quad predecesorul \quad lui \quad i(tată) \end{cases}$$

Nodurile frunză sunt cele care nu se găsesc în vectorul de taţi. Nodul rădăcină este nodul care nu are tată (t[r]=0)

Exemplu:
Vectorii de taţi ai arborilor din figura 12 sunt în ordine:
T1=(5,4,4,5,0,5)
T2=(6,0,2,6,3,2,3)
T3=(0,1,5,2,1,2)

Vectorii S si D
Arborii binari pot fi reprezentaţi şi prin vectorii S şi d în care s[i]=fiul stâng al nodului i, sau 0 dacă i nu are fiu stâng, iar d[i]= fiul drept al nodului i, sau 0 dacă nu are fiu drept. Rădăcina va fi nodul care nu este fiu stâng şi nici drept al nici unui alt nod deci nodul care nu se găseşte în cei doi vectori.
Pentru arborele A2 din figura 12 vom avea vectorii S:(0,3,5,0,0,1,0) şi D:(0,6,7,0,0,4,0). Nodul rădăcină este nodul care nu se regăseşte în vectorii S și D(în exemplu nodul 2)

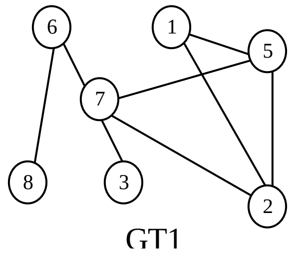
12. 2 Algoritmi de prelucrare a grafurilor

Parcurgerea în lăţime BF
Algoritmul se poate aplica atât grafurilor neorientate caz în care se obţine componenta conexă a nodului de plecare cât şi grafurilor orientate caz în care se obţine lista nodurilor accesibile prin drum din vârful de plecare.
Algoritmul utilizează matricea de adiacenţă, o coadă şi un vector *viz* de dimensiune egala cu numărul de noduri ale grafului prelucrat iniţializat cu 0. *Fiecare* nod ce va fi adăugat în coadă este marcat în vectorul *viz* cu valoarea 1. Se porneşte parcurgerea de la orice nod al grafului G, nod ce este adăugat în coada iniţial vidă. Se adaugă în coadă toţi vecinii/succesorii primului vârf. Pentru fiecare vârf adăugat se repetă procedeul: se adaugă în coadă vecinii/succesorii nevizitaţi. Parcurgerea se opreşte atunci când sau prelucrat toate vârfurile ce au fost adăugate în coadă.
În cazul grafurilor neorientate dacă graful este conex se obţine o coadă ce memoreaza toate nodurile din graf.

Exemplu:
Presupunem nod de start pentru GT1 vârful 1

Paşii parcurgerii	***Noduri adăugate***
Vizităm nodul de start 1	1
Vizităm vecinii lui 1	2,5
Pentru 2 şi 5 vizităm adiacenţii	

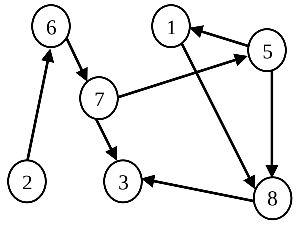
nevizitaţi încă • Vecinii lui 2 sunt 5 zi 7, nevizitat 7 • Vecinii lui 5 sunt 1,2,7, existenţi deja în coadă	7 nu se adaugă nimic
Se vizitează adiacenţii lui 7 care sunt 2,5,3,6	3,6
Se vizitează adiacenţii lui 3	nu se adaugă nimic
Se vizitează vecinii lui 6	8
Se vizitează 8	nu se adaugă nimic
Se finalizează parcurgerea	



GT1

Presupunem nod de start pentru graful GT2 vârful 7

Paşii parcurgerii	***Noduri adăugate***
Vizităm nodul de start 7	7
Vizităm succesorii lui 7	3,5
Paşii parcurgerii	***Noduri adăugate***
Pentru 3 şi 5 vizităm succesorii nevizitaţi încă • 3 nu are succesori • Succesorii lui 5 sunt 1 şi 8	 nu se adaugă nimic 1,8
Se vizitează succesorul 8 al lui 1	nu se adaugă nimic
8 îl are ca succesor pe 3 vizitat anterior	nu se adaugă nimic
Se finalizează parcurgerea	



GT2

Să urmărim „evoluţia” cozii pentru graful GT1 de mai sus; capetele se identifică prin p şi u(poziţia primului respectiv ultimului element) :

p u
1

	p	u
1	2	5

		p	u
1	2	5	7

			p u
1	2	5	7

				P	u
1	2	5	7	3	6

					p u
1	2	5	7	3	6

						p u
1	2	3	7	3	6	8

Algoritmul BF:
Matricea de adiacenţă *a* şi numărul de vârfuri n sunt declarate ca variabile globale.

Implementare C++	Implementare Pascal

```
void BFS(int S, int c[ ]) /* S = nodul
start. c – coada */
{
  int i, x, y,p,u,viz[MAX_N];/*MAX_N
constantă – numărul maxim de noduri*/
  p = u = 0; // initializare coada
  for(i=0;i<=n;i++)
     viz[i]=0; //nici un nod vizitat
  c[p] = S; /* se introduce nodul start in
coada */
  viz[S]=1;/*se marcheaza ca vizitat
nodul S*/
   while(p<=u) /* cat timp coada este
nevida*/
     {x = c[p++]; /* extrag un element din
coada*/
       for(y = 1; y<=n; y++)
         // se cauta adiacentii/succesorii
          if(a[x][y] && !viz[y])
              /* daca am muchia (x,y) si y nu
a fost vizitat*/
            {viz[y] =1; //vizitez y
              c[++u] = y; /* se introduce y in
coada*/
            }
        }
}
```

```
procedure BFS(S:integer; var c:vector);
var i, x, y, p, u;
      viz:vector;
begin
 p:=1;
 u:=1;
 for i:=1 to n do
   viz[i]:=0;
c[p]=S;
viz[S]:=1;
while p<=u do
 begin
   x=c[p];
   inc(p);
   for y:=1 to n do
      if  (a[x, y]=1) and (viz[y]=0)
        then
          begin
             viz[y]=1;
             inc(u);
             c[u]=y;
          end;
  end;
end;
```

Dacă se doreşte aflarea componentelor conexe vectorul *viz* se iniţializează cu 0 în afara funcţiei BFS. Funcţia BFS se apelează pentru fiecare vârf rămas nevizitat după parcurgerile anterioare.
Pentru a obţine în cazul grafurilor orientate componentele conexe se modifică condiţia din instrucţiunea IF astfel:

- (a[x][y] || a[y][x]) && !viz[y] //implementare C++
- ((a[x, y]=1) or (a[y,x]=1)) and (viz[y]=0) {implementare Pascal}

Algoritmul Roy-Warshall

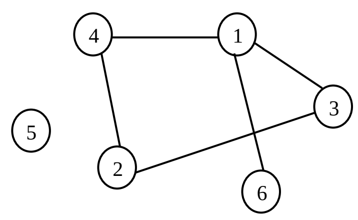
Este un algoritm ce prelucrează matricea de adiacenţă a grafurilor transformând matricea iniţială în matricea lanţurilor/drumurilor în urma unor prelucrări succesive conform următorului principiu: există lanţ/drum de la *i* la *j* dacă există (*i*, *j*) muchie în graf sau există *k* aşa încât există lanţ/drum de la *i* la *k* şi de la *k* la *j*. Un element a_{ij} devine 1 dacă există *k* aşa încât $a_{ik}=1$ şi $a_{kj}=1$.

Exemplu:

Pentru graful GT1 matricea de adiacenţă evolueaza de la forma la

0	0	1	1	0	1
0	0	1	1	0	0
1	1	0	0	0	0
1	1	0	0	0	0
0	0	0	0	0	0
1	0	0	0	0	0

1	1	1	1	0	1
1	1	1	1	0	1
1	1	1	1	0	1
1	1	1	1	0	1
0	0	0	0	0	0
1	1	1	1	0	1



GT1

Pentru graful GT2 matricea de adiacenţă evoluează în patru etape astfel:

0	1	1	0
0	0	1	1
0	0	0	0
1	1	0	0

...

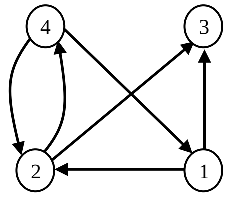
0	1	1	0
0	0	1	1
0	0	0	0
1	1	1	0

...

0	1	1	1
0	0	1	1
0	0	0	0
1	1	1	1

...

1	1	1	1
1	1	1	1
0	0	0	0
1	1	1	1



GT2

Matricea de adiacenţă *a* şi numărul de vârfuri *n* sunt declarate ca variabile globale.

Implementare C++

```
void roy_warshall(int md[][100])/*md matricea lanturilor drumurilor*/
{ int i,j,k;
  for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
      md[i][j]=a[i][j];
/*se copie elementele din a in md; se poate prelucra si direct a dar se pierde matricea de adiacenta*/
  for(k=1;k<=n;k++)
    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
        if(md[i][j]= =0 && i!=k && j!=k)
          md[i][j]=md[i][k]*md[k][j];
}
```

Implementare Pascal

```
procedure roy_warshall (var md:matrice);
var i:integer;
    j:integer;
    k:integer;
begin
 for i:=1 to n do
  for j:=1 to n do
    md[i,j]:=a[i,j];

for k:=1 to n do
  for i:=1 to n do
    for j:=1 to n do
      if((md[i,j]=0) and (i<>k) and (j<>k))
        then
          md[i,j]:=md[i,k]*md[k,j];
end;
```

Algoritmul se poate utiliza şi pentru determinarea componentelor conexe/tare conexe. În cazul grafurilor neorientate dacă $md_{lj}=1$ atunci *j* este în componenta conexă ce îl conţine pe *l*. Pentru grafurile orientate dacă $md_{lj}=md_{jl}=1$ atunci *j* este în aceeaşi componentă tare conexă cu *l*.

Algoritmul Roy-Floyd

Fie $f: U \to \Re$, unde *U* este mulţimea muchiilor/arcelor unui graf numită funcţie de cost sau pondere. Valorile funcţiei de cost pot reprezenta costurile de deplasare între două punce dacă graful memorează o reţea stradală, timpul de transfer al pachetelor de date într-o reţea de calculatoare între două noduri ale reţelei sau costurile de producţie pentru a trece un produs dintr-o fază de prelucrare în alta. Un graf *G* ce are asociată o astfel de funcţie se numeşte graf ponderat.

Se defineşte matricea costurilor astfel:

$$mc_{ij} = \begin{cases} 0, daca \quad i = j \\ f((i,j)), daca \quad (i,j) \in U \\ \pm\infty, daca \quad (i,j) \notin U \end{cases}$$

Algoritmul *Roy-Floyd* utilizat pentru a afla costul lanţurilor/drumurilor minime/ maxime într-un graf sau lungimea lor dacă $f(m)=1, \forall m \in U$. Este asemenător cu *Roy-Warshall* folosind următoarea idee: dacă există *k* aşa încât drumul minim/maxim de la un vârf *i* la un vârf *j* are un cost mai mare/mai mic decât decât suma costurilor drumurilor de la *i* la *k* însumat cu costul drumului de la *k* la *j* atunci $mc_{ij}=mc_{ik}+mc_{kj}$.

În implementare valoarea $\pm\infty$ se înlocuieşte cu o constantă predefinită(ex: ±MAXINT, ±MAXLONG) sau definită în program.

Matricea costurilor *mc* şi numărul de vârfuri *n* sunt declarate ca variabile globale. Uzual graful se memorează prin intermediul unui vector de înregistrări ce memorează extremităţile muchiilor şi ponderea lor matricea *mc* construindu-se pe baza informaţiilor din vector.

Implementare C++

```
void roy_floyd( )
{ int i,j,k;
  for(k=1;k<=n;k++)
    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
        if(md[i][j]< = md[i][k]+md[k][j])
          md[i][j]=md[i][k]+md[k][j];
}
```

Implementare Pascal

```
procedure roy_floyd;
var i:integer;
    j:integer;
    k:integer;
begin
for k:=1 to n do
  for i:=1 to n do
    for j:=1 to n do
      if md[i,j]<=md[i,k]+md[k,j]
        then
          md[i,j]:=md[i,k]+md[k,j];
end;
```

Algoritmul Dijkstra

Algoritmul utilizează un tablou *d*, *d[i]* fiind lungimea celui mai scurt drum de la nodul sursă la nodul i la un moment dat şi un vector *viz* ce reţine informaţii despre mulţimea nodurilor vizitate. Iniţial, *d[i]=mc[sursa,i]*, *viz[i]*=0 cu excepţia *viz[sursă]=1*. Algoritmul se execută în *n-1*paşi. La fiecare pas se calculează distanţa minimă de la sursă până la un nod nevizitat reţinând valoarea nodului respectiv. Fie *x* nodul astfel determinat. Nodul x este marcat prin *viz[x]=1* şi pe masură ce se găsesc noduri *y* astfel incat d*[y]>d[x]+mc[x,y]*, d*[y]* este redus. La terminarea algoritmului, *d[i]* va contine lungimea drumului minim de la origine la *i*.

Matricea costurilor *mc* definită astfel: $mc_{ij} = \begin{cases} f((i,j)), daca \ \ (i,j) \in U \\ 0, in \ \ rest \end{cases}$ şi numărul de vârfuri *n* sunt declarate ca variabile globale.

Implementare C++

```
void dijkstra(int S)
{
  int d[MAX_N], i, j;
  int viz[MAX_N]; /* viz[i] = daca a fost
vizitat sau nu*/
for(i=1;i<=n;i++)
 {viz[i]=0;
  d[i]=mc[S][i]?mc[S][i]:inf;}
viz[S] = 1;
int min, pmin = 0;
for (  i = 1; i <= n-1; i++ )
 {min = inf;//constanta predefinita
  for (j = 1; j <= n; j++)
     //extrag minimul din d[]
     if ( d[j] < min && !viz[j] )
        {min = d[j];
         pmin = j;
        }
   viz[pmin] = 1;
   for(j = 1; j<=n; j++)
     /* actualizez drumul pana la vecinii
minimului*/
     if(mc[pmin][j])
        /* daca am muchie (pmin,j), de
```

Implementare Pascal

```
procedure dijkstra(S:integer);
var d,viz:vector;
    i,j,min,pmin:integer;
begin
 for i:=1 to n do
   begin
     viz[i]:=0;
     if mc[S,i]<>0
       then
         d[i]:=mc[S,i]
       else
         d[i]:=inf;
   end;
viz[S]:=1;
for i:=1 to n-1 do
 begin
   min:=inf;
   for j:=1 to n do
     if ((d[j]<min) and (viz[j]=0))
       then
         begin
           min:=d[j];
           pmin:=j;
         end;
```

```
cost mc[pmin][j]*/
        {if ( d[ j ] > d[pmin] + mc[pmin][j] )
            /* daca pot ajunge in j pe un drum mai bun*/
            d[ j ] = d[pmin] + mc[pmin][j];
        }
  }
cout<<"Dijkstra : Distantele de la noul sursa ”<<S<<” la varfurile grafului, in ordine, sunt: ";
for(i=1;i<=n;i++) cout<<d[i];
}
```

```
  viz[pmin]:=1;
  for j:=1 to n do
    if mc[pmin,j]<>0
      then
        if d[j]>d[pmin]+mc[pmin,j]
          then
            d[j]:=d[pmin]+mc[pmin,j];
 end;
write(‘Dijkstra : Distantele de la noul sursa ‘,S,’ la varfurile grafului, in ordine, sunt:’);
for i:=1 to n do
  write(d[i],’ ‘);
end;
```

Exemplu:
Pentru graful GT1 alăturat evoluţia vectorilor *d* şi *viz* este următoarea (considerăm 1 nod sursă):

d	0	4	∞	∞	∞	3	∞	∞	5
viz	1	0	0	0	0	0	0	0	0

pmin=6;min=3

d	0	4	∞	∞	∞	3	8	∞	5
v	1	0	0	0	0	1	0	0	0

pmin=2;min=4

d	0	4	11	13	∞	3	8	∞	5
v	1	1	0	0	0	1	0	0	0

pmin=9;min=5

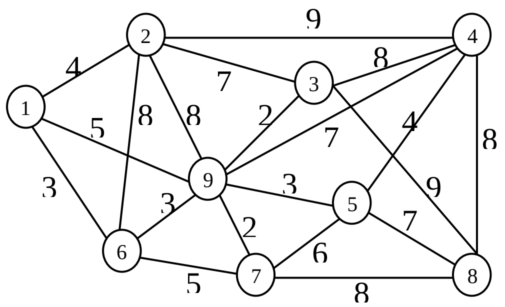
d	0	4	7	12	8	3	7	∞	5
v	1	1	0	0	0	1	0	0	1

pmin=3;min=7

d	0	4	7	12	8	3	7	16	5
v	1	1	1	0	0	1	0	0	1

pmin=7;min=7

d	0	4	7	12	8	3	7	15	5
v	1	1	1	0	0	1	1	0	1


GT1

La următorii paşi *pmin* va lua valorile 5,4,8 dar nu se mai produc modificări asupra vectorului *d*. Configuraţia finală a lui *d* este cea obţinută după prelucrarea cu *pmin*=7.

Algoritmul lui Prim
Este un algoritm care returneaza un arbore de acoperire <u>minim</u> pentru un graf ponderat. Un arbore de acoperire de cost minim este un subgraf alcătuit din toate nodurile grafului iniţial dar nu din toate arcele, ci doar din atâtea arce cât să nu apară cicluri iar costul total al muchiilor alese este minim.
Algoritmul poate fi stilizat astfel:

- Iniţial, toate nodurile grafului se consideră nevizitate
- Se porneşte de la ***<u>un nod oarecare</u>*** al grafului care se marchează ca vizitat
- În permanenţă vor fi menţinute două mulţimi:
    - Mulţimea V a nodurilor vizitate (iniţial, V va contine doar nodul de start)
    - Mulţimea X\V a nodurilor nevizitate (X este mulţimea tuturor nodurilor)
- La fiecare pas se alege acel nod din mulţimea X\V care este legat ***<u>prin arc de cost minim</u>*** de ***<u>oricare</u>*** din nodurile din mulţimea V
- Nodul ales va fi scos din multimea X\V şi trecut în mulţimea V
- Algoritmul continua pana cand V = X(sau la alegerea a n-1 muchii, unde n este cardinalul lui X)

În implementare se utilizează matricea costurilor definită după modelul următor: $mc_{ij} = \begin{cases} f((i,j)), daca \;\; (i,j) \in U \\ 0, in \;\;\; rest \end{cases}$ unde *f* este funcţia de cost asociată grafului şi doi vectori: *s* ce reţine vârfurile selectate şi *t* ce memorează vectorul de taţi al arborelui parţial de cost minim. Matricea şi cele două tablouri sunt utilizate în implementare ca variabile globale.

Implementare C++

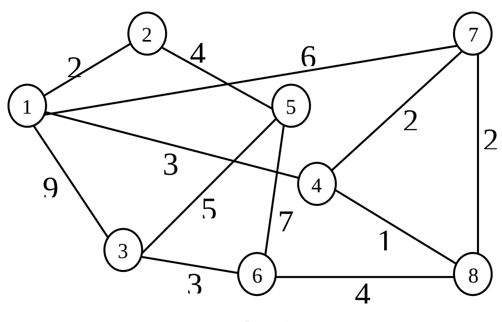
```
void prim( )
{
  int rad,i,j,min;
cout<<"radacina:"
cin>>rad;
/*se poate introduce orice nod*/
for(i=1;i<=n;i++)
 s[i]=rad;
s[rad]=0;
for(i=1;i<=n;i++)
 t[i]=0;
/*se initializeaza vectorii t si s*/
cost=0; //costul total;variabila globala
for(k=1;k<=n-1;k++)
//se aleg n-1 muchii
{min=inf;
  for(i=1;i<=n;i++)
    if (s[i])
      if(mc[s[i]][i]<min && mc[s[i]][i])
         {min=mc[s[i]][i];
          j=i;
         }/*se calculeaza in min costul
minim al unei muchii cu un capat ales; j
extremitatea nemarcata a muchiei
alese*/
  t[j]=s[j];/*se marcheaza predecesorul
lui j*/
  cost+=mc[i][s[j]];//actualizare cost
  s[j]=0;//se marcheaza varful ales
  for(i=1;i<=n;i++)
    if (s[i])
      if (!mc[i][s[i]] || mc[i][s[i]]>mc[i][j])
        if (mc[i][j])
          s[i]=j;/*se actualizeaza s pentru
alegerile viitoare*/
  }
}
```

Implementare Pascal

```
procedure prim;
var rad,i,j,min:integer;
begin
write('radacina:');
readln(rad);
   for i:=1 to n do
   s[i]:=rad;
s[rad]:=0;
for i:=1 to n do
    t[i]:=0;
cost:=0;
for k:=1 to n-1 do
 begin
  min:=inf;
  for i:=1 to n do
    if (s[i]<>0)
     then
      if (mc[s[i],i]<min)and(mc[s[i],i]<>0)
        then
          begin
            min:=mc[s[i],i];
            j:=i;
          end;
  t[j]:=s[j];
  cost:=cost+mc[j,s[j]];
  s[j]:=0;
  for i:=1 to n do
    if (s[i]<>0)
     then
       if (mc[i,s[i]]=0)or(mc[i,s[i]]>mc[i,j])
         then
           if mc[i,j]<>0
             then
               s[i]:=j;
 end;
end;
```

Exemplu:

Pentru graful alăturat evoluţia tablourilor *s* şi *t* considerând rădăcină nodul 1 este următoarea:



GT1

s	0	1	1	1	1	1	1	1
t	0	0	0	0	0	0	0	0
min=2;j=2; se alege muchia (1,2)								

s	0	0	1	1	2	1	1	1
t	0	1	0	0	0	0	0	0

min=3;j=4; se alege muchia (1,4)

s	0	0	1	0	2	1	4	4
t	0	1	0	1	0	0	0	0

min=1;j=8; se alege muchia (4,8)

s	0	0	1	0	2	8	4	0
t	0	1	0	1	0	0	0	4

min=2;j=7; se alege muchia (4,7)

s	0	0	1	0	2	8	0	0
t	0	1	0	1	0	0	4	4

min=4;j=5; se alege muchia (2,5)

s	0	0	5	0	0	8	0	0
t	0	1	0	1	2	0	4	4

min=4;j=6; se alege muchia (6,8)

s	0	0	6	0	0	0	0	0
t	0	1	0	1	2	8	4	4

min=3;j=3; se alege muchia (3,6)

s	0	0	0	0	0	0	0	0
t	0	1	6	1	2	8	4	4