Sorin Nădăban Andrea Şandru

ALGORITMICA GRAFURILOR

- sinteze de curs şi aplicaţii -

Editura MIRTON Timişoara
2007

Prefaţă

Grafurile au devenit astăzi foarte răspândite datorită ariei largi de aplicabilitate a acestora, de la aplicaţii atât software cât şi hardware, la diverse aplicaţii în modelarea sistemelor economice, în ştiinţele inginereşti şi în cele sociale.

Această carte constituie suportul pentru cursurile şi laboratoarele susţinute de către autori studenţilor de la Facultatea de Ştiinţe Exacte din cadrul Universităţii "Aurel Vlaicu" Arad.

Cartea prezintă, după o scurtă familiarizare cu limbajul utilizat, algoritmi esenţiali pentru prelucrarea grafurilor. Sunt tratate subiecte precum: parcurgerea unui graf, matricea drumurilor, componente conexe şi tare conexe, drumuri de valoare optimă, arbore de acoperire minim, fluxuri maxime, probleme de afectare şi ordonanţare.

Mulţumim de pe acum tuturor celor care prin sugestii şi observaţii ne vor ajuta la îmbunătăţirea unei eventuale reeditări.

Arad,
23 noiembrie 2007

Autorii

Cuprins

Capitolul 1

Noţiuni introductive

1.1 Reprezentarea grafurilor orientate

Definiţia 1.1.1 *Se numeşte* ***graf orientat*** *perechea* $G = (X, U)$ *formată dintr-o mulţime* $X = \{x_1, x_2, \ldots, x_n\}$ *ale cărei elemente se numesc* ***vârfuri*** *şi o mulţime* $U = \{u_1, u_2, \ldots, u_m\}$ *formată din perechi ordonate de vârfuri numite* ***arce****.*

Exemplul 1.1.2 *Fie graful* $G = (X, U)$ *unde* $X = \{x_1, x_2, x_3, x_4, x_5\}$ *şi* $U = \{(1,2), (1,4), (1,5), (2,4), (3,1), (3,2), (3,5), (4,4), (4,5), (5,3)\}$.

Observaţia 1.1.3 *Remarcăm că s-a făcut convenţia ca arcul* (x_3, x_1) *să fie notat* $(3,1)$ *ş.a.m.d.*

Observaţia 1.1.4 *Un graf orientat* (X, U) *poate fi privit şi ca ansamblul format dintr-o mulţime finită* X *şi o relaţie binară*[1] U *pe* X*.*

Observaţia 1.1.5 *Un graf orientat* (X, U) *este de fapt ansamblul format dintr-o mulţime finită* X *şi o aplicaţie multivocă* $\Gamma : X \to X$*.*

Pentru graful din exemplul 1.1.2 aplicaţia multivocă $\Gamma : X \to X$ este definită prin:

$$\Gamma(x_1) = \{x_2, x_4, x_5\}\ ,\ \Gamma(x_2) = \{x_4\}\ ,\ \Gamma(x_3) = \{x_1, x_2, x_5\}\ ,$$

$$\Gamma(x_4) = \{x_4, x_5\}\ ,\ \Gamma(x_5) = \{x_3\}\ .$$

[1] o submulţime a produsului cartezian $X \times X$.

Observaţia 1.1.6 *Un graf orientat admite o reprezentare sagitală, vârfurile fiind reprezentate prin cercuri, iar arcele prin săgeţi.*

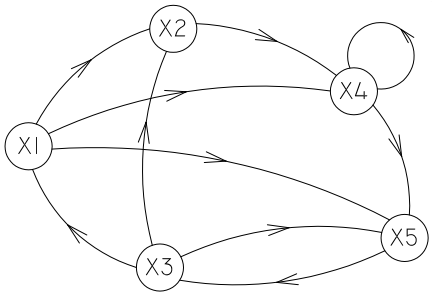


Figura 1.1: Reprezentarea sagitală a grafului din exemplul 1.1.2

Definiţia 1.1.7 *Dacă considerăm arcul $u = (x_i, x_j)$ atunci x_i se numeşte* ***extremitatea iniţială****, iar x_j se numeşte* ***extremitatea finală*** *a arcului u. Vârfurile x_i şi x_j se numesc* ***adiacente****. Un graf se numeşte* ***complet*** *dacă vârfurile sale sunt adiacente două câte două.*
Vârful x_j se numeşte ***succesor*** *al lui x_i, iar vârful x_i se numeşte* ***predecesor*** *al lui x_j.*
Pentru un vârf $x \in X$ vom nota prin $\Gamma^-(x)$ ***mulţimea predecesorilor*** *lui x, adică $\Gamma^-(x) = \{y \in X : (y, x) \in U\}$. Vom nota prin $\Gamma^+(x)$* ***mulţimea succesorilor*** *lui x, adică $\Gamma^+(x) = \{y \in X : (x, y) \in U\}$.*
Numărul predecesorilor lui x se notează $d^-(x)$ şi se numeşte ***semigrad interior*** *al lui x. Numărul succesorilor lui x se notează $d^+(x)$ şi se numeşte* ***semigrad exterior*** *al lui x. Numărul $d(x) := d^-(x) + d^+(x)$ se numeşte* ***gradul*** *vârfului x. Un vârf de grad 0 se numeşte* ***vârf izolat****.*
Un arc de forma (x_i, x_i), adică de la un vârf la el însuşi, se numeşte ***buclă****.*

Observaţia 1.1.8 *Pentru graful din exemplul 1.1.2 remarcăm că (x_4, x_4) este o buclă. Mai notăm că*

$$\Gamma^-(x_1) = \{x_3\}; \quad \Gamma^-(x_2) = \{x_3\}; \quad \Gamma^-(x_3) = \{x_5\};$$

$$\Gamma^-(x_4) = \{x_1, x_2, x_4\}; \quad \Gamma^-(x_5) = \{x_1, x_3, x_4\}.$$

$$\Gamma^+(x_1) = \{x_2, x_4, x_5\}; \quad \Gamma^+(x_2) = \{x_4\} \ \ etc.$$

Avem $d^-(x_1) = 1$, $d^+(x_1) = 3$ *şi* $d(x_1) = 4$ *ş.a.m.d.*
Precizăm că nu avem un graf complet deoarece vârfurile x_2 *şi* x_5 *nu sunt adiacente.*

Definiţia 1.1.9 *Pentru* $x_i, x_j \in X$ *vom nota prin* $m^+(x_i, x_j)$ *numărul arcelor de la* x_i *la* x_j *şi îl numim* ***multiplicitatea pozitivă****. În mod similar,* $m^-(x_i, x_j)$ *reprezintă numărul arcelor de la* x_j *la* x_i *şi se numeşte* ***multiplicitatea negativă****. Vom pune*

$$m(x, y) := m^+(x, y) + m^-(x, y) .$$

Observaţia 1.1.10 *În baza celor de mai sus avem că* $m^+(x, y) \in \{0, 1\}$, $(\forall)x, y \in X$. *În unele cărţi sunt considerate aşa numitele* $p-$***grafuri*** *sau* ***multigrafuri****. Aceasta înseamnă că de la vârful* x *la vârful* y *pot exista mai multe arce, dar numărul lor nu depăşeşte un număr natural* p.

Observaţia 1.1.11 *Un graf este complet dacă şi numai dacă* $m(x, y) \geq 1$ *pentru orice* $x, y \in X$.

Definiţia 1.1.12 *Numim* ***matrice de adiacenţă*** *a grafului* $G = (X, U)$ *matricea* $A = (a_{ij})_{i,j=1}^n$, *unde* $a_{ij} = m^+(x_i, x_j)$.

Observaţia 1.1.13 *Pentru graful din exemplul 1.1.2 matricea de adiacenţă este*

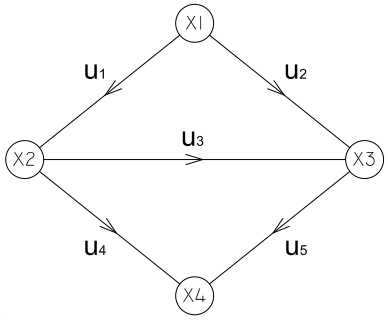
$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Observaţia 1.1.14 *Menţionăm că dacă cunoaştem reprezentarea unui graf cu ajutorul matricei de adiacenţă, atunci se poate scrie cu uşurinţă mulţimea* X *a vârfurilor şi mulţimea* U *a arcelor.*

Observaţia 1.1.15 *O altă posibilitate de reprezentare a unui graf este cu ajutorul matricei "arce-vârfuri". Astfel dacă* $G = (X, U)$ *este un graf cu* n *vârfuri şi* m *arce atunci matricea "arce-vârfuri" este o matrice* $B = (b_{ij})$ *cu* n *linii şi* m *coloane, unde*

$$b_{ij} = \begin{cases} 1, & \text{dacă vârful } x_i \text{ este extremitatea iniţială a arcului } u_j \\ -1, & \text{dacă vârful } x_i \text{ este extremitatea finală a arcului } u_j \\ 0, & \text{dacă vârful } x_i \text{ nu este extremitate a arcului } u_j \end{cases} .$$

Exemplul 1.1.16 *Pentru graful $G = (X, U)$ care admite reprezentarea sagitală*



matricea "arce-vârfuri" este

$$B = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

1.2 Grafuri neorientate

Definiţia 1.2.1 *Se numeşte* ***graf neorientat*** *perechea $G = (X, U)$ formată dintr-o mulţime $X = \{x_1, x_2, \ldots, x_n\}$ ale cărei elemente se numesc* ***vârfuri*** *sau* ***noduri*** *şi o mulţime $U = \{u_1, u_2, \ldots, u_m\}$ formată din perechi de vârfuri neordonate numite* ***muchii****.*

Observaţia 1.2.2 *Cu alte cuvinte, o muchie este o mulţime $\{x, y\}$ unde $x, y \in X$şi $x \neq y$. Pentru o muchie vom folosi notaţia (x, y). Remarcăm că (x, y) şi (y, x) reprezintă aceeaşi muchie şi că într-un graf neorientat sunt interzise buclele. Astfel fiecare muchie este formată din exact două vârfuri distincte.*

Exemplul 1.2.3 *Fie graful $G = (X, U)$, unde $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ şi $U = \{(1, 2), (1, 6), (2, 5), (3, 6), (4, 1), (5, 6), (6, 2)\}$.*

Observaţia 1.2.4 *Un graf neorientat admite o reprezentare grafică, vârfurile fiind reprezentate prin cercuri iar muchiile prin segmente.*

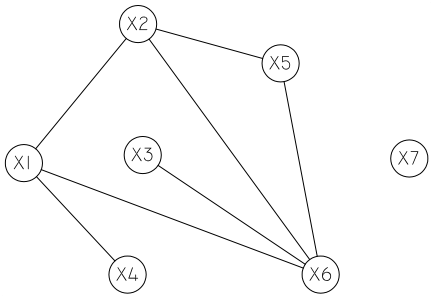


Figura 1.2: Reprezentarea grafică a grafului din exemplul 1.2.3

Observaţia 1.2.5 *Multe definiţii pentru grafuri orientate şi neorientate sunt aceleaşi, deşi anumiţi termeni pot avea semnificaţii diferite în cele două contexte. Remarcăm că dacă avem un graf neorientat acesta poate fi transformat într-un graf orientat înlocuind fiecare muchie (x, y) prin două arce (x, y) şi (y, x). Reciproc, dacă avem un graf orientat versiunea neorientată se obţine eliminînd buclele şi direcţiile. Prin urmare, în continuare atunci când nicio precizare nu este făcută ne referim la grafuri orientate.*

1.3 Operaţii cu grafuri

Definiţia 1.3.1 *Fie $G = (X, U)$ şi $G' = (Y, V)$ două grafuri. Spunem că G şi G' sunt **izomorfe** şi notăm $G \simeq G'$ dacă există o funcţie bijectivă $\varphi : X \to Y$ astfel încât*

$$(x, y) \in U \Leftrightarrow (\varphi(x), \varphi(y)) \in V, \quad (\forall) x, y \in X.$$

*Aplicaţia φ se numeşte **izomorfism**.*

Observaţia 1.3.2 *În general, nu facem deosebire între două grafuri izomorfe. Astfel, vom scrie $G = G'$ în loc de $G \simeq G'$.*

Definiţia 1.3.3 *Fie $G = (X, U)$ şi $G' = (Y, V)$ două grafuri. Se numeşte **reuniunea** grafurilor G şi G' graful*

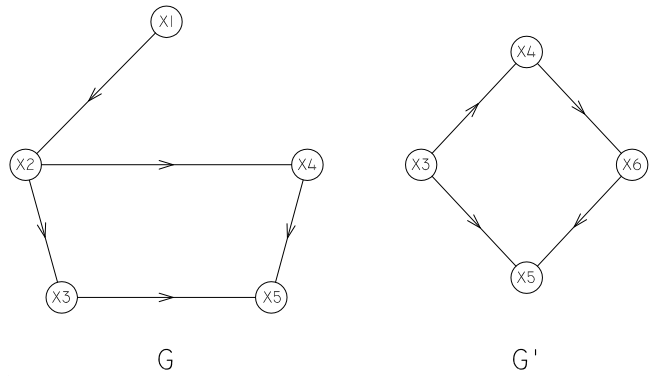
$$G \cup G' := (X \cup Y, U \cup V).$$

Se numeşte ***intersecţia*** *grafurilor G şi G' graful*

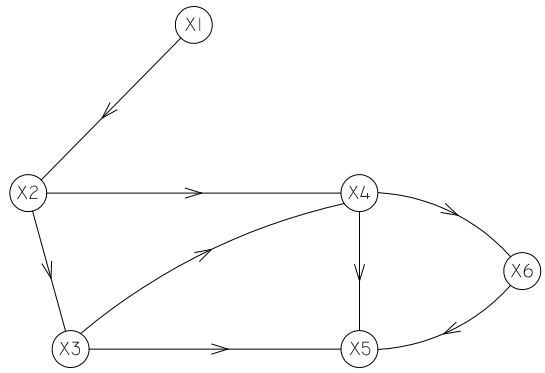
$$G \cap G' := (X \cap Y, U \cap V).$$

Dacă $G \cap G' = \emptyset$ atunci G şi G' se numesc ***disjuncte****.*

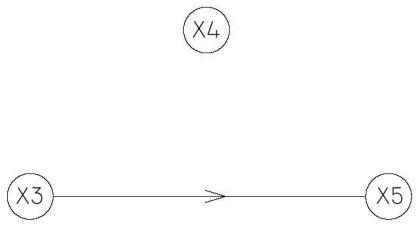
Exemplul 1.3.4 *Considerăm grafurile*



Atunci $G \cup G'$ este



şi $G \cap G'$ este



Definiţia 1.3.5 *Fie $G = (X, U)$ şi $G' = (X, V)$ două grafuri cu aceeaşi mulţime de vârfuri. Se numeşte* ***suma*** *grafurilor G şi G' graful*

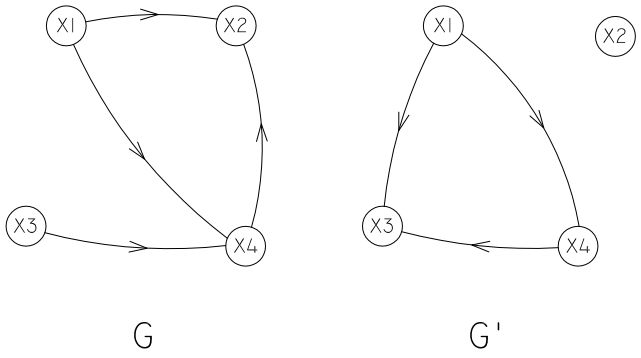
$$G \oplus G' := (X, U \cup V) .$$

Se numeşte ***produsul*** *grafurilor G şi G' graful*

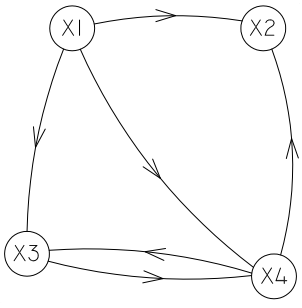
$$G \otimes G' := (X, W) ,$$

unde $W := \{(x, y) \in X \times X \ : \ (\exists) z \in X$ astfel încât $(x, z) \in U, \ (z, y) \in V\}$. Se numeşte ***transpusul*** *grafului $G = (X, U)$ graful $G^t := (X, U^t)$, unde $U^t := \{(x, y) \in X \times X \ : \ (y, x) \in U\}$.*

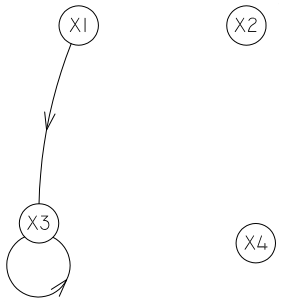
Exemplul 1.3.6 *Se consideră grafurile*



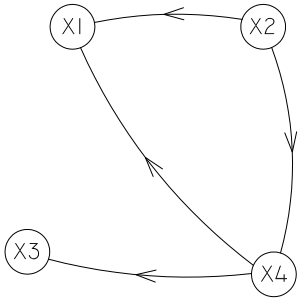
Atunci $G \oplus G'$ este



$G \otimes G'$ este



G^t *este*



Observaţia 1.3.7 *Dacă graful $G = (X, U)$ are matricea de adiacenţă A şi graful $G' = (X, V)$ are matricea de adiacenţă A' atunci*

1. *$G \oplus G'$ are matricea $A + A'$;*
2. *$G \otimes G'$ are matricea $A \cdot A'$;*
3. *G^t are matricea A^t.*

Definiţia 1.3.8 *Fie $G = (X, U)$ şi $G' = (Y, V)$ două grafuri. Dacă $Y \subseteq X$ şi $V \subseteq U$ vom spune că G' este un **subgraf** al lui G şi notăm $G' \subseteq G$.*

Definiţia 1.3.9 *Fie $G = (X, U)$ un graf şi $Y \subseteq X$. Se numeşte **subgraf generat de** Y graful $G[Y] := (Y, U_Y)$ unde U_Y reprezintă mulţimea acelor arce din U ce au ambele extremităţi în Y.*

Observaţia 1.3.10 *Evident $G[Y]$ este un subgraf al lui G dar nu orice subgraf este generat de o submulţime de vârfuri aşa cum arată exemplul următor.*

Exemplul 1.3.11 *Se consideră graful G din exemplul* 1.1.2. *Pentru $Y = \{x_2, x_3, x_4\}$, subgraful generat de Y este $G[Y] = (Y, U_Y)$ unde $U_Y = \{(2,4),(3,2),(4,4)\}$. Dar $G' = (Y, V)$ unde $V = \{(2,4)\}$ este un subgraf al lui G care nu este însă un subgraf generat.*

Definiţia 1.3.12 ***Complementarul*** *unui graf $G = (X, U)$ este graful $\overline{G} = (X, X \times X \setminus U)$.*

1.4 Grafuri valorizate

Definiţia 1.4.1 *Se numeşte* ***graf valorizat*** *un graf $G = (X, U)$ împreună cu o aplicaţie*

$$v : U \to \mathbb{R}_+ ,$$

$$U \ni (x, y) \mapsto v(x, y) \in \mathbb{R}_+ .$$

Numărul real pozitiv $v(x, y)$ se numeşte ***valoarea arcului*** *(x, y).*

Observaţia 1.4.2 *Un graf valorizat poate fi reprezentat sagital ca şi în figura 1.1, trecând însă deasupra fiecărui arc valoarea acestuia. In general însă preferăm ca un graf valorizat să fie reprezentat printr-un tabel de forma:*

arcele	(x_i, x_j)
valorile arcelor	v_{ij}

.

Exemplul 1.4.3 *Se consideră graful valorizat $G = (X, U)$, unde $X = \{x_1, x_2, x_3, x_4, x_5\}$, reprezentat prin tabelul*

ij	*12*	*14*	*15*
v_{ij}	*7*	*8*	*9*

.

Să se determine reprezentarea sagitală a acestui graf.

Soluţie.

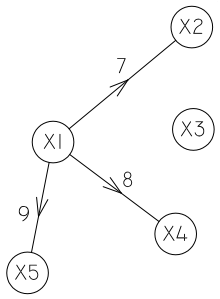


Figura 1.3: Reprezentarea sagitală a unui graf valorizat

1.5 Drumuri, circuite şi lanţuri

Definiţia 1.5.1 *Se numeşte* ***drum de lungime*** p *un şir ordonat de arce* $\mu = (u_1, u_2, \ldots, u_p)$ *cu proprietatea că extremitatea finală a arcului* u_i *coincide cu extremitatea iniţială a arcului* u_{i+1} *,* $(\forall) i = \overline{1, p-1}$*. Lungimea drumului* μ *o vom nota* $l(\mu)$*.*
Extremitatea iniţială a primului arc se numeşte ***extremitatea iniţială*** *a drumului, iar extremitatea finală a ultimului arc se numeşte* ***extremitatea finală*** *a drumului.*
Atunci când extremitatea finală a drumului coincide cu extremitatea lui iniţială spunem că avem un ***drum închis*** *sau* ***circuit****.*

Exemplul 1.5.2 *Exemple de drumuri în graful din exemplul 1.1.2 ar fi:*

$$\mu_1 = \{(1,2),(2,4)\}\ ,\ l(\mu_1) = 2\ .$$

$$\mu_2 = \{(3,2),(2,4),(4,5)\}\ ,\ l(\mu_2) = 3\ .$$

$$\mu_3 = \{(1,2),(2,4),(4,5),(5,3),(3,1)\}\ ,\ l(\mu_3) = 5\ .$$

Convenim ca drumul μ_1 *să fie notat* $(1,2,4)$*, drumul* μ_2 *să fie notat* $(3,2,4,5)$*, iar drumul* μ_3 *să fie scris* $(1,2,4,5,3,1)$*. Precizăm că* μ_3 *este un circuit.*

Definiţia 1.5.3 *Un drum se numeşte* ***simplu****, dacă arcele din care este format drumul sunt distincte.*
Un drum se numeşte ***elementar*** *dacă toate vârfurile din el sunt distincte.*
Un drum care trece o dată şi o singură dată prin fiecare vârf al grafului se numeşte ***drum hamiltonian****. Prin urmare aceste drumuri au lungimea* $n-1$ *(unde* n *reprezintă numărul de vârfuri).*
Prin ***circuit hamiltonian*** *înţelegem acele circuite ce conţin toate vârfurile grafului o dată şi o singură dată, cu excepţia vârfului iniţial care coincide cu cel final. Prin urmare au lungimea* n*.*
Prin ***circuit eulerian*** *al unui graf înţelegem un circuit simplu care foloseşte toate arcele grafului.*
Un graf care conţine un circuit hamiltonian se numeşte ***graf hamiltonian****.*
Un graf care conţine un circuit eulerian se numeşte ***graf eulerian****.*

Exemplul 1.5.4 *In graful din exemplul 1.1.2,* $\mu = (1, 2, 4, 5, 3)$ *este un drum hamiltonian de lungime 4 şi* $\mu_3 = (1, 2, 4, 5, 3, 1)$ *este un circuit hamiltonian de lungime 5. Prin urmare graful din exemplul 1.1.2 este hamiltonian.*

Definiţia 1.5.5 *Fie* $G = (X, U)$ *un graf cu* n *vârfuri.* ***Matricea drumurilor*** *este o matrice* $D = (d_{ij})$ *cu* n *linii şi* n *coloane, unde*

$$d_{ij} = \begin{cases} 1, & \text{dacă există un drum de la } x_i \text{ la } x_j, \\ 0, & \text{în caz contrar.} \end{cases}$$

Observaţia 1.5.6 *Pentru graful din exemplul 1.1.2 matricea drumurilor este*

$$D = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

În capitolul următor vom vedea diverşi algoritmi pentru determinarea acestei matrici.

Definiţia 1.5.7 ***Puterea de atingere*** *a unui vârf* $x \in X$ *în graful* $G = (X, U)$ *reprezintă numărul de vârfuri la care se poate ajunge printr-un drum ce pleacă din* x *şi notează* $p(x)$*.*

Observaţia 1.5.8 *Evident* $p(x) \geq d^+(x),\ (\forall)x \in X$*.*

Definiţia 1.5.9 *Se numeşte* ***lanţ*** *de lungime r un şir ordonat de vârfuri $L = (y_0, y_1, y_2, \ldots, y_r)$ cu proprietatea că oricare două vârfuri consecutive sunt adiacente, adică $(y_i, y_{i+1}) \in U$ sau $(y_{i+1}, y_i) \in U$, pentru orice $i = \overline{0, r-1}$.*
Dacă în plus toate vârfurile din lanţ sunt distincte două câte două, lanţul se numeşte ***elementar****.*

Exemplul 1.5.10 *În graful din exemplul* 1.1.2 *avem lanţul $L = (1, 2, 3, 5)$.*

Într-adevăr, $(1, 2) \in U$, $(3, 2) \in U$, $(3, 5) \in U$.

Observaţia 1.5.11 *Un lanţ poate fi privit şi ca un şir ordonat de arce $L = (u_1, u_2, \ldots, u_r)$ cu proprietatea că arcele u_i şi u_{i+1} au o extremitate comună, pentru orice $i = \overline{1, r-1}$.*
Lanţul din exemplul precedent se poate scrie $L = [(1, 2), (3, 2), (3, 5)]$.
Noţiunea de lanţ este mai mult întâlnită la grafurile neorientate unde înlocuieşte de fapt noţiunea de drum. Noi însă preferăm să o folosim pe cea de drum.
Mai precizăm că în cazul grafurilor neorientate nu este folosită noţiunea de circuit, ea fiind înlocuită cu cea de ***ciclu****. Astfel prin ciclu al unui graf neorientat înţelegem un lanţ $L = [y_0, y_1, \ldots, y_r]$ cu proprietatea că $y_0 = y_r$. Dacă în plus toate muchiile sunt distincte două câte două ciclul se numeşte* ***simplu****. Dacă toate vârfurile ciclului sunt distincte două câte două cu excepţia vârfului iniţial care coincide cu cel final, ciclul se numeşte* ***elementar****. Un ciclu elementar ce conţine toate vârfurile grafului se numeşte* ***ciclu hamiltonian****. Un graf neorientat ce conţine un ciclu hamiltonian se numeşte* ***graf hamiltonian****. Vom numi* ***ciclu eulerian*** *un drum închis într-un graf neorientat care conţine fiecare muchie a grafului exact odată. Un graf neorientat se numeşte* ***eulerian*** *dacă admite cicluri euleriene.*

1.6 Componente conexe şi tare conexe

Definiţia 1.6.1 *Un graf neorientat $G = (X, U)$ se numeşte* ***conex*** *dacă $(\forall) x, y \in X$ există un drum de la x la y.*
Se numeşte ***componentă conexă*** *a unui graf neorientat $G = (X, U)$ un subgraf $G[Y] = (Y, U_Y)$ care este conex şi care este maximal în raport cu incluziunea faţă de această proprietate, adică $(\forall) x \in X \setminus Y$, subgraful generat de $Y \cup \{x\}$ nu este conex.*

Observaţia 1.6.2 *Componentele conexe ale unui graf sunt clasele de echivalenţă ale vârfurilor în raport cu relaţia de "accesibil".*

Definiţia 1.6.3 *Fie $G = (X, U)$ un graf conex.*

1. *Se numeşte **distanţa** dintre vârfurile x şi y numărul de muchii conţinute în cel mai scurt drum care uneşte pe x cu y şi se notează $d(x, y)$;*

2. *Se numeşte **excentricitatea** vârfului x numărul*

$$e(x) := \max_{y \in X} d(x, y);$$

3. *Se numeşte **raza** grafului G numărul*

$$\rho(G) := \min_{x \in X} e(x);$$

4. *Se numeşte **diametrul** grafului G numărul*

$$d(G) := \max_{x \in X} e(x);$$

5. *Se numeşte **centrul** grafului G mulţimea*

$$C(G) := \{y \in X : \ e(y) = \rho(G)\}.$$

Definiţia 1.6.4 *Un graf orientat $G = (X, U)$ se numeşte **tare conex** dacă $(\forall)x, y \in X$ există un drum de la x la y şi un drum de la y la x.*
*Se numeşte **componentă tare conexă** a unui graf orientat $G = (X, U)$ un subgraf $G[Y] = (Y, U_Y)$ care este tare conex şi care este maximal în raport cu incluziunea faţă de această proprietate, adică $(\forall)x \in X \setminus Y$, subgraful generat de $Y \cup \{x\}$ nu este tare conex.*

Observaţia 1.6.5 *Componentele tare conexe ale unui graf orientat sunt clasele de echivalenţă ale vârfurilor în raport cu relaţia de "reciproc accesibile".*

În capitolul următor vom vedea algoritmi pentru determinarea componentelor tare conexe.

1.7 Arbori

Definiţia 1.7.1 *Se numeşte* ***arbore*** *un graf neorientat fără cicluri şi conex.*

Exemplul 1.7.2

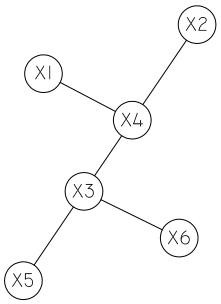


Figura 1.4: Un arbore

Propoziţia 1.7.3 *Fie $G = (X, U)$ un graf neorientat cu n vârfuri ($n \geq 2$). Următoarele afirmaţii sunt echivalente:*

1. *G este un arbore;*
2. *Oricare două vârfuri sunt legate printr-un lanţ unic;*
3. *G este un graf conex minimal, adică G este conex dar $G - (x, y)$ nu este conex pentru orice $(x, y) \in U$;*
4. *G este un graf aciclic maximal, adică G este fără cicluri dar $G + (x, y)$ are cicluri pentru orice două vârfuri neadiacente $x, y \in X$;*
5. *G este un graf conex cu $n - 1$ muchii;*
6. *G este un graf aciclic cu $n - 1$ muchii.*

Definiţia 1.7.4 *Se numeşte* ***frunză*** *orice vârf de grad 1. Se numeşte* ***nod intern*** *un vârf care nu este frunză.*

Observaţia 1.7.5 *Orice arbore cu n vârfuri ($n \geq 2$) are cel puţin două frunze. Dacă îndepărtăm o frunză de la un arbore ceea ce rămâne este încă un arbore.*

Observaţia 1.7.6 *Arborele din exemplul 1.7.2 are drept frunze nodurile* x_1, x_2, x_5, x_6.

Observaţia 1.7.7 *Uneori este convenabil să considerăm un vârf al arborelui ca fiind special. Un astfel de vârf îl numim* ***rădăcină****. Precizăm că orice vârf poate fi ales rădăcină. Alegerea unei rădăcini r pentru un arbore* $G = (X, U)$ *conduce la o relaţie de ordine parţială pe X, punând* $x \leq y$ *dacă x aparţine lanţului ce uneşte pe r cu y. In raport cu această relaţie de ordine parţială r este cel mai mic element şi orice frunză* $x \neq r$ *a lui G este un element maximal. In plus, alegerea unei rădăcini conduce la aşezarea arborelui pe nivele. Astfel:*

1. *se aşează rădăcina pe nivelul* 1*;*
2. *se plasează pe fiecare nivel* $i > 1$ *vârfuri pentru care lungimea lanţurilor care se leagă de rădăcină este* $i - 1$*;*
3. *se trasează muchiile arborelui.*

Vârfurile de pe nivelul următor legate de acelaşi vârf i poartă numele de ***fii*** *(descendenţi direcţi) ai vârfului i, iar vârful i poartă numele de* ***tată*** *(ascendent direct) al acestor vârfuri. Vârfurile care au acelaşi tată poartă denumirea de* ***fraţi****.*

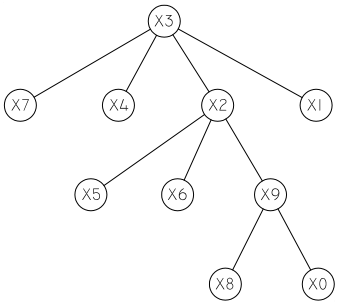


Figura 1.5: Aşezarea unui arbore pe nivele

Pentru arborele de mai sus, nodul x_3 este rădăcină, un nod tată ar putea fi nodul x_2 care are noduri fii pe x_5, x_6, x_9, prin urmare aceştia sunt fraţi.

Observaţia 1.7.8 *Dintre parcurgerile folosite pentru un arbore oarecare amintim:*

1. ***parcurgerea în preordine**: aceasta constă în prelucrarea rădăcinii şi apoi parcurgerea descendenţilor direcţi de la stânga la dreapta. Pentru arborele din figura 1.5 obţinem:* 3 7 4 2 5 6 9 8 0 1.

2. ***parcurgerea în postordine**: aceasta constă în parcurgerea de la stânga la dreapta a descendenţilor direcţi şi apoi prelucrarea rădăcinii. Pentru arborele din figura 1.5 obţinem:* 7 4 5 6 8 0 9 2 1 3.

3. ***parcurgerea pe nivele**: se parcurg în ordine vârfurile unui nivel de la stânga la dreapta, începând de la primul nivel până la ultimul. Pentru arborele anterior obţinem:* 3 7 4 2 1 5 6 9 8 0.

Definiţia 1.7.9 *Un graf neorientat $G = (X, U)$ aciclic se numeşte **pădure**.*

Observaţie 1.7.10 *Remarcăm că o pădure este un graf a cărui componente conexe sunt arbori.*

Exemplul 1.7.11

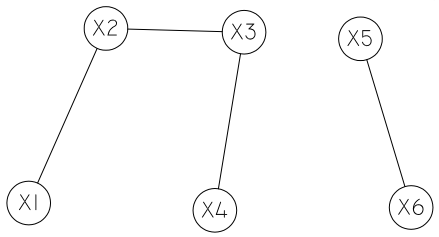


Figura 1.6: O pădure

Definiţia 1.7.12 *Un graf orientat se numeşte **conex** dacă graful neorientat obţinut din acesta (prin suprimarea direcţiilor) este conex.*
*Un graf orientat se numeşte **ramificare** dacă graful neorientat obţinut din acesta este o pădure şi fiecare vârf x are cel mult un arc ce intră în el.*

O ramificare conexă se numeşte ***arborescenţă****. In baza propoziţiei 1.7.3 o arborescenţă cu n vârfuri are $n-1$ arce. Astfel există un singur vârf r cu proprietatea $d^-(r) = \emptyset$. Acest vârf se numeşte* ***rădăcină****. Acele vârfuri x cu proprietatea $d^+(x) = \emptyset$ se numesc* ***frunze****.*

Propoziţia 1.7.13 *Fie $G = (X, U)$ un graf orientat cu n vârfuri. Următoarele afirmaţii sunt echivalente.*

1. *G este o arborescenţă cu rădăcina r;*
2. *G este o ramificare cu $n-1$ arce şi $d^-(r) = \emptyset$;*
3. *G are $n-1$ arce şi fiecare vârf poate fi atins plecând din r;*
4. *Fiecare vârf poate fi atins plecând din r, dar renenţând la un arc distrugem această proprietate.*

1.8 Grafuri bipartite

Definiţia 1.8.1 *Fie $r \geq 2$ un număr natural. Un graf neorientat $G = (X, U)$ se numeşte r-partit dacă:*

1. *X admite o partiţie în r clase, adică există $\{A_i\}_{i=1}^r$, astfel încât:*
   (a) $\bigcup_{i=1}^{r} A_i = X$;
   (b) $A_i \neq \emptyset,\ (\forall) i = \overline{1, r}$;
   (c) $A_i \cap A_j = \emptyset,\ (\forall) i, j = \overline{1, r},\ i \neq j$.
2. *orice muchie $u \in U$ îşi are extremităţile în clase diferite, adică două vârfuri din aceeaşi clasă nu pot fi adiacente.*

În cazul în care $r = 2$ vom folosii termenul de ***graf bipartit****.*

Definiţia 1.8.2 *Un graf r-partit pentru care orice două vârfuri din clase diferite sunt adiacente se numeşte* ***complet****.*

Teorema 1.8.3 *Un graf este bipartit dacă şi numai dacă nu conţine cicluri de lungime impară.*

Exemplul 1.8.4

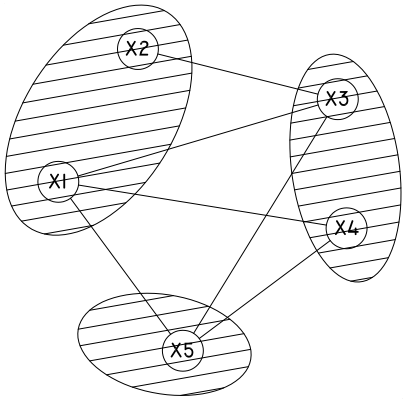


Figura 1.7: Graf 3-partit

Exemplul 1.8.5

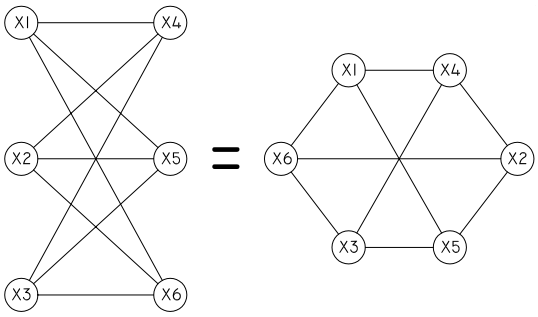


Figura 1.8: Două reprezentări pentru un graf bipartit complet

1.9 Reţele de transport

Definiţia 1.9.1 *Un graf orientat $G = (X, U)$ fără bucle se numeşte* ***reţea de transport*** *dacă satisface următoarele condiţii:*

1. *există un unic $x_0 \in X : \Gamma^-(x_0) = \emptyset$. Vârful x_0 se numeşte* ***vârf sursă*** *sau* ***intrarea reţelei****;*

2. *există un unic* $x_n \in X : \Gamma^+(x_n) = \emptyset$. *Vârful* x_n *se numeşte* ***vârf destinatar*** *sau* ***ieşirea reţelei****;*

3. *G este conex (între oricare două vârfuri există cel puţin un lanţ) şi există drumuri de la* x_0 *la* x_n *în G;*

4. *există o funcţie* $c : X \times X \to \mathbb{R}_+$ *numită* ***capacitatea reţelei****, cu proprietatea că* $c(x, y) = 0$ *dacă* $(x, y) \notin U$. *Dacă* $(x, y) \in U$ *atunci* $c(x, y)$ *va fi numit* ***capacitatea arcului*** (x, y).

O reţea de transport va fi notată $G = (X, U, c)$.

1.9.1 Problema fluxului maxim

Definiţia 1.9.2 *Se numeşte* ***flux*** *o aplicaţie* $f : X \times X \to \mathbb{R}$ *care îndeplineşte condiţiile:*

1. ***"condiţia de conservare"****:*

   (a) *dacă* $x \in X \setminus \{x_0, x_n\}$ *atunci*

   $$\sum_{y \in X} f(x, y) = 0 \ ;$$

   (b)

   $$\sum_{y \in X} f(x_0, y) = \sum_{x \in X} f(x, x_n) = \overline{f} \ ;$$

2. ***"condiţia de simetrie"****:* $f(x, y) = -f(y, x) \ , (\forall) x, y \in X$;

3. ***"condiţia de mărginire"****:* $f(x, y) \leq c(x, y) \ , (\forall) x, y \in X$.

Definiţia 1.9.3 ***Problema fluxului maxim*** *constă în determinarea unui flux pentru care valoarea* $\overline{f}$ *este maximă.*

Observaţia 1.9.4 *Vom rezolva această problemă doar în ipoteza că funcţia capacitate ia doar valori numere raţionale. In capitolul următor vom prezenta algoritmul Ford-Fulkerson care rezolvă această problemă. Ideea acestui algoritm este ca plecând de la un flux să-l mărim până când atinge cea mai mare valoare posibilă.*

Observaţia 1.9.5 *Dacă $x, y \in X$ atunci $f(x,y)$ măsoară intensitatea fluxului de la x la y. Sunt posibile următoarele situaţii:*

1. *$f(x,y) > 0$. În acest caz spunem că avem un **flux efectiv** de la x la y. În această situaţie $(x,y) \in U$, pentru că în caz contrar $c(x,y) = 0$ şi nu ar fi îndeplinită condiţia de mărginire;*
2. *$f(x,y) < 0$. În acest caz spunem că avem un **flux virtul** de la x la y. În această situaţie $f(y,x) > 0$ (din condiţia de simetrie) şi prin urmare $(y,x) \in U$;*
3. *$f(x,y) = 0$. În această situaţie nu există flux de la x la y şi nici de la y la x.*

Observaţia 1.9.6 *Pentru $x \in X$ vom nota cu $f^+(x)$ suma fluxurilor efective (pozitive) pe arcele care ies din x. Vom nota cu $f^-(x)$ suma fluxurilor efective (pozitive) pe arcele care intră în x. Mai precis*

$$f^+(x) = \sum_{(x,y)\in U, f(x,y)>0} f(x,y) \; ;$$

$$f^-(x) = \sum_{(y,x)\in U, f(y,x)>0} f(y,x) \; .$$

Atunci condiţia de conservare a fluxului se poate scrie

$$f^+(x) = f^-(x) \; , (\forall) x \in X \setminus \{x_0, x_n\} \;\; ; \;\; f^+(x_0) = f^-(x_n) = \overline{f} \; .$$

Definiţia 1.9.7 *Un arc $u \in U$ se numeşte **saturat** dacă $f(u) = c(u)$. În caz contrar arcul se numeşte **nesaturat** sau **arc cu capacitate reziduală** sau mai simplu spus **arc rezidual**.*

Observaţia 1.9.8 *Definiţia precedentă se poate extinde la $(x,y) \notin U$. Astfel, $(x,y) \notin U$ se numeşte **arc rezidual** dacă $(y,x) \in U$ şi $f(y,x) > 0$.*

Definiţia 1.9.9 *Pentru $x, y \in X$ definim **capacitatea reziduală** indusă de fluxul f prin*

$$c_f(x,y) = c(x,y) - f(x,y) \; .$$

Observaţia 1.9.10 *In baza definiţiei precedente, $(x,y) \in X \times X$ este arc rezidual în următoarele situaţii:*

1. $(x,y) \in U$ *şi* $c_f(x,y) = c(x,y) - f(x,y) > 0$;
2. $(x,y) \notin U$ *dar* $(y,x) \in U$ *şi*

$$c_f(x,y) = c(x,y) - f(x,y) = 0 + f(y,x) = f(y,x) > 0 \ .$$

Definiţia 1.9.11 *Fie* $G = (X,U,c)$ *o reţea de transport şi* f *un flux. Se numeşte* ***reţea reziduală*** *indusă de* f *în* G *reţeaua de transport* $G_f = (X,U_f,c_f)$, *unde*

$$U_f = \{(x,y) \in X \times X \ : \ c_f(x,y) > 0\} \ .$$

Un drum μ *de la* x_0 *la* x_n *în reţeaua reziduală* G_f *va fi numit* ***drum rezidual****. Dacă există un astfel de drum se defineşte* ***capacitatea reziduală*** *a drumului* μ *ca fiind numărul*

$$c_f(\mu) = \min_{(x,y)\in\mu} c_f(x,y) \ .$$

Teorema 1.9.12 ***(Procedeu de creştere a fluxului)****. Dacă în reţeaua reziduală* G_f *nu există un drum rezidual atunci fluxul* f *este maxim.*
Dacă în reţeaua reziduală G_f *există un drum rezidual* μ *atunci fluxul* f *poate fi mărit. Fie*

$$f_\mu(x,y) := \begin{cases} c_f(\mu) & dacă \quad (x,y) \in \mu \cap U \\ -c_f(\mu) & dacă \quad (x,y) \in \mu \text{ şi } (y,x) \in U \\ 0 & în \ rest \end{cases} \ .$$

Atunci $f' = f + f_\mu$ *este un nou flux şi* $\overline{f'} = \overline{f} + c_f(\mu)$.

1.9.2 Probleme de transport

Un produs este stocat în depozitele $D_1, D_2, \ldots, D_m$ în cantităţile $a_1, a_2, \ldots, a_m$. El este solicitat de beneficiarii $B_1, B_2, \ldots, B_n$ în cantităţile $b_1, b_2, \ldots, b_n$. Costul unitar de transport de la depozitul D_i la beneficiarul B_j este c_{ij}. Aceste date se pun în general într-un tabel de forma

	B_1	B_2	$\ldots$	B_n	Disponibil
D_1	c_{11}	c_{12}	$\ldots$	c_{1n}	a_1
D_2	c_{21}	c_{22}	$\ldots$	c_{2n}	a_2
$\vdots$					$\vdots$
D_m	c_{m1}	c_{m2}	$\ldots$	c_{mn}	a_m
Necesar	b_1	b_2	$\ldots$	b_n	

Se cere să se determine cantităţile x_{ij} ce urmează să fie transportate de la depozitul D_i la beneficiarul B_j astfel încât costul total de transport să fie minim, să ne încadrăm în cantităţile disponibile şi să fie satisfăcut necesarul. Forma standard a unei probleme de transport este:

$$\begin{cases} min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \sum_{j=1}^{n} x_{ij} = a_i, \ (\forall) i = \overline{1,m} \\ \sum_{i=1}^{m} x_{ij} = b_j, \ (\forall) j = \overline{1,n} \\ 0 \leq x_{ij} \leq \alpha_{ij}, \ (\forall) i = \overline{1,m}, \ j = \overline{1,n} \\ \sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j \end{cases} .$$

Observaţia 1.9.13 *Sunt necesare câteva precizări:*

1. *Condiţia* $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$ *se numeşte* ***condiţia de echilibru****. Dacă ea nu este îndeplinită acest lucru poate fi rezolvat prin introducerea unui depozit sau a unui beneficiar fictiv. Astfel, dacă* $\sum_{i=1}^{m} a_i > \sum_{j=1}^{n} b_j$ *se introduce un beneficiar fictiv,* B_f*, cu un necesar* $\sum_{i=1}^{m} a_i - \sum_{j=1}^{n} b_j$ *şi costurile de transport* c_{if} *de la depozitele* D_i *la beneficiarul fictiv* B_f *egale cu zero pentru orice* $i = \overline{1,m}$*. Dacă* $\sum_{i=1}^{m} a_i < \sum_{j=1}^{n} b_j$ *se introduce un depozit fictiv* D_f *cu disponibilul de* $\sum_{j=1}^{n} b_j - \sum_{i=1}^{m} a_i$ *şi vom pune costurile unitare de transport* c_{fj} *de la depozitul fictiv* D_f *la beneficiarii* B_j *egale cu zero, pentru orice* $j = \overline{1,n}$*.*

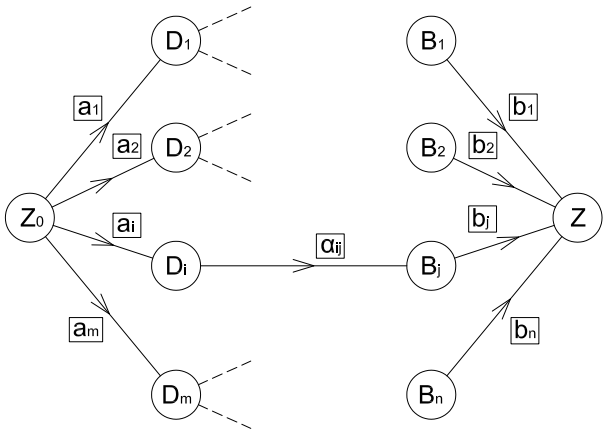
2. *Numărul* α_{ij} *reprezintă capacitatea de transport între depozitul* D_i *şi beneficiarul* B_j*, pentru* $i = \overline{1,m}$ *şi* $j = \overline{1,n}$*.*

Observaţia 1.9.14 *Unei probleme de transport i se poate asocia o reţea de transport cu vârfurile* $D_1, D_2, \ldots, D_m, B_1, B_2, \ldots, B_n, Z_0, Z$*.*

1. *Intrarea reţelei* Z_0 *este legată de vârful* D_i *printr-un arc de capacitate* $a_i, \ (\forall) i = \overline{1,m}$*;*

2. *Vârful* D_i *este legat de vârful* B_j *printr-un arc de capacitate* α_{ij}, $(\forall)i = \overline{1,m},\ j = \overline{1,n}$;

3. *Vârful* B_j *este legat de ieşirea reţelei* Z *printr-un arc de capacitate* $b_j,\ (\forall)j = \overline{1,n}$.



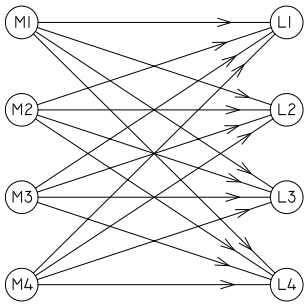
Problema revine la a găsi un flux f care să satureze arcele iniţiale şi terminale şi pentru care cantitatea $\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$ este minimă.

1.9.3 Probleme de afectare

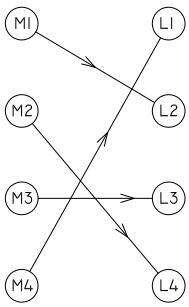
Într-un proces de producţie există n maşini, $M_1, M_2, \ldots, M_n$, pe care pot fi executate lucrările $L_1, L_2, \ldots, L_n$. Timpul de execuţie al lucrării L_i la maşina M_j este t_{ij}. Se pune problema repartizării lucrărilor pe maşini astfel încât timpul total de execuţie să fie minim. Modelul matematic al unei probleme de afectare este:

$$\begin{cases} min \sum_{i=1}^{n}\sum_{j=1}^{n} t_{ij}x_{ij} \\ \sum_{i=1}^{n} x_{ij} = 1,\ (\forall)j = \overline{1,n} \\ \sum_{j=1}^{n} x_{ij} = 1,\ (\forall)i = \overline{1,n} \\ x_{ij} \in \{0,1\},\ (\forall)i = \overline{1,n},\ j = \overline{1,n} \end{cases}.$$

Prin urmare o problemă de afectare este un caz particular de problemă de transport, acela în care $a_i = 1,\ b_j = 1,\ (\forall) i = \overline{1,n},\ j = \overline{1,n}$.
O altă abordare a problemelor de afectare este cu ajutorul teoriei cuplajelor. Această teorie este foarte dezvoltată, dar noi ne vom mărgini la a da câteva noţiuni. Astfel, dacă notăm cu X mulţimea maşinilor şi cu Y mulţimea lucrărilor ($X \cap Y = \emptyset$), atunci putem construi graful $G = (X \cup Y, \Gamma)$ unde Γ este o aplicaţie multivocă de la X la Y. Un astfel de graf este numit **graf simplu**. Precizăm că, în general, cardinalul mulţimii X nu trebuie să fie egal cu cel al mulţimii Y. În cazul nostru aceasta ar reprezenta o problemă puţin mai complicată, căci, într-o astfel de situaţie, o maşină trebuie să execute o submulţime de lucrări din mulţimea lucrărilor. Dacă U reprezintă mulţimea arcelor din graful simplu $(X \cup Y, \Gamma)$ atunci vom numi **cuplaj** o submulţime de arce $V \subset U$ cu proprietatea că oricare două arce distincte din V nu sunt adiacente (nu au comună vreo extremitate).
Spre exemplu, dacă avem 4 maşini ce pot efectua 4 lucrări graful tuturor posibilităţilor are forma



Un cuplaj ar putea fi



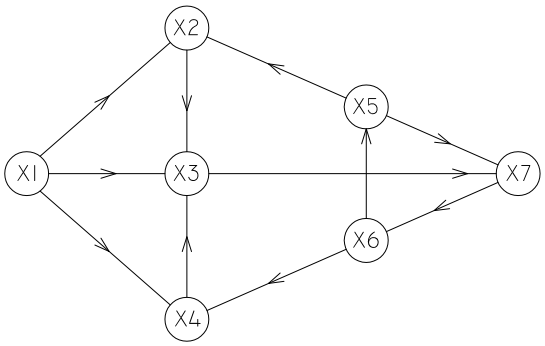
În concluzie, rezolvarea unei probleme de afectare revine la a găsi într-un graf simplu, plecând de la diferite cuplaje posibile pe cel care optimizează expresia

$$\sum_{i=1}^{n}\sum_{j=1}^{n} t_{ij}x_{ij},$$

unde x_{ij} va avea valoarea 0 dacă maşina M_i nu este cuplată cu lucrarea L_j sau valoarea 1 dacă maşina M_i este cuplată cu lucrarea L_j.

1.10 Exerciţii

Exerciţiul 1.10.1 *Se consideră graful orientat $G = (X, U)$ având reprezentarea sagitală*

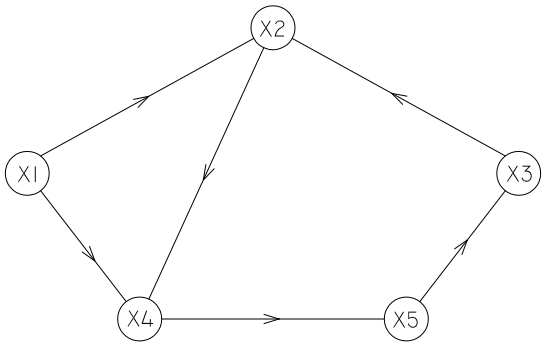


1. *Să se determine matricea de adiacenţă a grafului;*
2. *Să se determine pentru fiecare vârf mulţimea predecesorilor şi mulţimea succesorilor;*
3. *Să se calculeze gradul fiecărui vârf;*
4. *Să se determine matricea "arce-vârfuri".*

Exerciţiul 1.10.2 *Există un graf neorientat cu* 8 *vârfuri pentru care şirul gradelor vârfurilor sale este:*

$$1, 1, 1, 3, 3, 4, 5, 7 \ ?$$

Exerciţiul 1.10.3 *Se consideră graful G din exerciţiul* 1.10.1 *şi graful G' reprezentat sagital prin*



Să se determine

1. $G \cup G'$;
2. $G \cap G'$;
3. G^t;
4. $G \otimes G$;
5. *Subgraful lui G generat de mulţimea de vârfuri $Y = \{x_1, x_3, x_5\}$;*
6. $\overline{G}$.

Exerciţiul 1.10.4 *Fie $G = (X, U)$ un graf neorientat. Să se arate că fie G fie complementarul său $\overline{G}$ este conex.*

Exerciţiul 1.10.5 *Să se demonstreze afirmaţiile din observaţia* 1.3.7.

Exerciţiul 1.10.6 *Să se arate că mulţimea componentelor conexe ale unui graf neorientat $G = (X, U)$ formează o partiţie a lui X.*

Exerciţiul 1.10.7 *Să se demonstreze propoziţia* 1.7.3.

Exerciţiul 1.10.8 *Să se demonstreze propoziţia* 1.7.13.

Capitolul 2

Algoritmi pentru grafuri

2.1 Matricea drumurilor

2.1.1 Algoritmul lui Roy-Warshall

Fie $A = (a_{ij})_{i,j=1}^{n}$ matricea de adiacenţă a grafului $G = (X, U)$. Pentru a determina matricea drumurilor D acestei matrici îi aplicăm un şir de n transformări $T_k,\ k = \overline{1,n}$, calculând $A := T_k(A)$.
Etapa k. Pentru $i \neq k$ şi $j \neq k$ elementele a_{ij} se înlocuiesc cu

$$\max(a_{ij}, \min(a_{ik}, a_{kj}))$$

Dacă $i = k$ sau $j = k$ elementele a_{ij} rămân neschimbate.
La sfârşitul algoritmului matricea obţinută $T_n(A)$ este tocmai matricea drumurilor.

Observaţia 2.1.1 *Remarcăm că elementele $a_{ij} = 1$ (pentru $i \neq k$ şi $j \neq k$) rămân invariante la o transformare T_k iar elementele $a_{ij} = 0$ (pentru $i \neq k$ şi $j \neq k$) devin $\min(a_{ik}, a_{kj})$. Altfel spus, în etapa k se înlocuiesc toate elementele 0 care nu se găsesc pe linia sau coloana k prin 1 dacă ambele lor proiecţii pe linia şi coloana k sunt egale cu 1.*

Exemplul 2.1.2 *Utilizând algoritmul lui Roy-Warshall, să se determine matricea drumurilor pentru graful din figura următoare.*

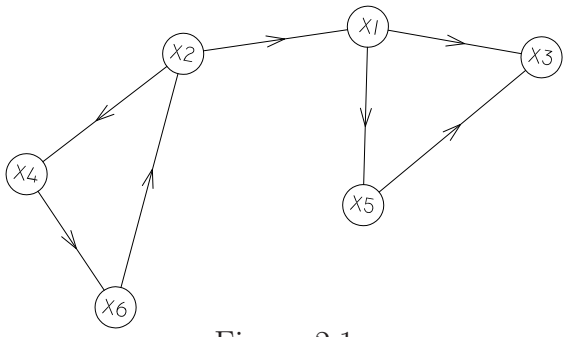


Figura 2.1:

Soluţie.

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \; ; \; A := T_1(A) = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A := T_2(A) = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \; ; \; A := T_3(A) = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$A := T_4(A) = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \; ; \; A := T_5(A) = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$A := T_6(A) = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

2.1.2 Metoda compunerii booleene

Pe mulţimea $\{0,1\}$ se definesc operaţiile boolene de adunare şi înmulţire, notate $\oplus$ şi $\otimes$ prin:

$$a \oplus b = \max\{a,b\}, \quad a \otimes b = \min\{a,b\}, \quad (\forall) a,b \in \{0,1\}.$$

Mai precis avem:

$$0 \oplus 0 = 0; \quad 0 \oplus 1 = 1; \quad 1 \oplus 0 = 1; \quad 1 \oplus 1 = 1\ ;$$

$$0 \otimes 0 = 0; \quad 0 \otimes 1 = 0; \quad 1 \otimes 0 = 0; \quad 1 \otimes 1 = 1\ .$$

Aceste operaţii se pot extinde la matrici cu coeficienţi din mulţimea $\{0,1\}$ obţinând "adunarea booleană" a matricilor care acţionează la fel ca şi în calculul matricial clasic "termen cu termen" şi "înmulţirea booleană" a matricilor care acţionează la fel ca şi în calculul matricial clasic "linie per coloană". Aceste operaţii vor fi notate tot cu $\oplus$ şi $\otimes$, sperând că nu este posibilă nicio confuzie.
Vom defini

$$A^{(k)} := A \otimes A \otimes \cdots \otimes A = (a_{ij}^{(k)})\ .$$

Teorema 2.1.3 *Fie $A = (a_{ij})_{i,j=1}^n$ matricea de adiacenţă a grafului $G = (X,U)$. Atunci*

1. *Matricea $A^{(k)}$ este matricea drumurilor (elementare sau nu) de lungime k, adică, dacă $a_{ij}^{(k)} = 1$ atunci între x_i şi x_j există un drum de lungime k, iar dacă $a_{ij}^{(k)} = 0$ atunci între x_i şi x_j nu există drum de lungime k;*
2. *Matricea drumurilor (elementare sau nu) este*

$$D = A \oplus A^{(2)} \oplus \ldots \oplus A^{(n-1)}.$$

Exemplul 2.1.4 *Utilizând metoda compunerii boolene să se determine matricea drumurilor pentru graful din figura 2.1.*

Soluţie.

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \; ; \; A^{(2)} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$A^{(3)} = A^{(2)} \otimes A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$A^{(4)} = A^{(3)} \times A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$A^{(5)} = A^{(4)} \otimes A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

În final

$$D = A \oplus A^{(2)} \oplus A^{(3)} \oplus A^{(4)} \oplus A^{(5)} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Observaţia 2.1.5 *Pentru a face economie de timp este suficient să observăm că:*

1. $D = A \otimes (A \oplus I)^{(n-2)}$, *unde* I *este matricea unitate;*
2. $A \otimes (A \oplus I)^{(n-2+k)} = A \otimes (A \oplus I)^{(n-2)}, \ (\forall)k \geq 0.$

Prin urmare vom calcula

$$(A \oplus I)^{(2)}, \ (A \oplus I)^{(4)}, \ldots, (A \oplus I)^{(2^r)}$$

unde r *se alege în aşa fel încât* $2^r \geq n - 2$.
În final $D = A \otimes (A \oplus I)^{(2^r)}$.

Exemplul 2.1.6 *Să se determine matricea drumurilor pentru graful din figura 2.1.*

Soluţie. Cum $n = 6$ alegem r astfel încât $2^r \geq n - 2$. Prin urmare $r = 2$.

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \ ; \ A \oplus I = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$(A \oplus I)^{(2)} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \ ; \ (A \oplus I)^{(4)} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$D = A \otimes (A \oplus I)^{(4)} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

2.1.3 Algoritmul lui Chen

Fie $A = (a_{ij})_{i,j=1}^{n}$ matricea de adiacenţă a grafului $G = (X, U)$.
Pentru $k = \overline{1, n}$ efectuăm:
Etapa k. Vom aduna boolean la linia k toate liniile j pentru care $a_{kj} = 1$. Se repetă această etapă până când nu se mai obţin noi elemente egale cu 1 pe linia k.

Exemplul 2.1.7 *Aplicând algoritmul lui Chen să se determine matricea drumurilor pentru graful din figura 2.1.*

Soluţie.

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Etapa 1. La linia 1 adunăm boolean liniile 3 şi 5.

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Pe linia 1 nu s-au obţinut noi elemente egale cu 1 şi prin urmare trecem la etapa 2.

Etapa 2. La linia 2 adunăm boolean liniile 1 şi 4.

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Adunăm la linia 2 liniile 1, 3, 4, 5, 6. Obţinem

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Etapa 3. Cum pe linia 3 nu avem elemente egale cu 1 matricea rămâne neschimbată.
Etapa 4. Adunăm boolean linia 6 la linia 4.

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

La linia 4 adunăm boolean liniile 2 şi 6.

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Etapa 5. Adunăm boolean la linia 5 linia a 3-a. Observăm că matricea A rămâne neschimbată.

Etapa 6. La linia a 6-a adunăm boolean linia a 2-a. Obţinem

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

adică tocmai matricea drumurilor căutată.

2.1.4 Algoritmul lui Kaufmann

Algoritmi prezentaţi în secţiunile precedente ne permit doar să aflăm dacă există sau nu drum între două vârfuri ale grafului, fără însă a şti care sunt aceste drumuri. Cum toate drumurile pot fi descompuse în drumuri elementare, algoritmul pe care îl vom prezenta este dedicat găsirii drumurilor elementare.

Definiţia 2.1.8 *Se numeşte* ***alfabet*** *o mulţime de simboluri sau litere* $\{s_i\}_{i\in I}$*, unde* I *este o mulţime de indici finită sau nu.*

Definiţia 2.1.9 *Se numeşte* ***cuvânt*** *un şir finit de simboluri, notat* $s_1 s_2 \dots s_k$*.*

Definiţia 2.1.10 *Se numeşte* ***înmulţire latină*** *o operaţie definită pe mulţimea cuvintelor unui alfabet, notată* $\otimes_L$ *şi definită prin*

$$s_1 s_2 \dots s_k \otimes_L t_1 t_2 \dots t_n = s_1 s_2 \dots s_k t_1 t_2 \dots t_n,$$

adică produsul a două cuvinte se obţine prin concatenarea lor.
Se numeşte ***adunare latină*** *o operaţie notată* $\oplus_L$ *definită prin*

$$s_1 s_2 \dots s_k \oplus_L t_1 t_2 \dots t_n = \{s_1 s_2 \dots s_k, t_1 t_2 \dots t_n\},$$

adică suma a două cuvinte este mulţimea formată cu cele două cuvinte.

Observaţia 2.1.11 *Aceste operaţii se pot extinde la matrici cu elemente cuvinte. Astfel vom putea vorbi de* ***multiplicarea latină*** *a două matrici, notată* $\otimes_L$*, care funcţionează la fel ca şi în calculul matriceal clasic "linie per coloană" cu precizarea că operaţia de înmulţire latină a cuvintelor este uşor modificată, în sensul că produsul a două cuvinte este* 0 *dacă unul dintre cuvinte este* 0 *sau dacă cele două cuvinte au un simbol comun.*

Observaţia 2.1.12 *În cazul în care avem un graf $G = (X, U)$ alfabetul va fi mulţimea vârfurilor grafului.*

Etapa 1. Se construieşte **matricea latină** $L_1 = (l_{ij})_{i,j=1}^{n}$ asociată grafului, unde:

$$l_{ij} = \begin{cases} x_i x_j, & \text{dacă există arcul } (x_i, x_j) \text{ şi } i \neq j; \\ 0, & \text{dacă nu există arcul } (x_i, x_j) \text{ sau } i = j. \end{cases}$$

Se construieşte matricea latină redusă L_R, obţinută din L_1 prin suprimarea fiecărei litere iniţiale.
Pentru $k = \overline{2, n-1}$ vom efectua:
Etapa k. Se obţin drumurile elementare de lungime k prin multiplicarea latină, calculând

$$L_k = L_{k-1} \otimes_L L_R.$$

Exemplul 2.1.13 *Să se determine drumurile elementare pentru graful $G = (X, U)$ unde $X = \{x_1, x_2, x_3, x_4, x_5\}$ şi*

$$U = \{(1,2), (1,4), (1,5), (2,4), (3,1), (3,2), (3,5), (4,4), (4,5), (5,3)\} .$$

Soluţie. Preferăm ca vârfurile grafului să le notăm $A,\ B,\ C,\ D,\ E$.
Etapa 1.

$L_1 =$

	AB		AD	AE
			BD	
CA	CB			CE
				DE
		EC		

; $L_R =$

	B		D	E
			D	
A	B			E
				E
		C		

Etapa 2.

$L_2 =$

		AEC	ABD	ADE
				BDE
	CAB		CAB, CBD	CAE
		DEC		
ECA	ECB			

Etapa 3.

$L_3 =$

	AECB	ADEC		ABDE
		BDEC		
			CABD	CADE, CBDE
DECA	DECB			
	ECAB		ECAD, ECBD	

Etapa 4.

$L_4 =$

	ADECB	ABDEC	AECBD	
BDCEA				
				CABDE
	DECAB			
		ECABD		

2.2 Determinarea componentelor conexe

2.2.1 Algoritmul de scanare al grafului

Fie $G = (X, U)$ un graf orientat sau neorientat. Acest algoritm determină drumurile de la un vârf specificat x_0 la celelalte vârfuri ca pot fi atinse plecând din x_0. În cazul neorientat el construieşte un arbore maximal ce conţine x_0. În cazul orientat el construieşte o arborescenţă maximală ce conţine x_0 ca rădăcină.

Etapa 0. Fie $R := \{x_0\}$, $Q := \{x_0\}$ şi $A = \emptyset$.

Etapa 1. Dacă $Q = \emptyset$ algoritmul s-a încheiat. În caz contrar alegem $x \in Q$.

Etapa 2. Alegem $y \in X \setminus R$ cu proprietatea că $u = (x, y) \in U$. Dacă nu există un astfel de vârf punem $Q := Q \setminus \{x\}$ şi trecem la etapa 1.

Etapa 3. Punem $R := R \cup \{y\}$, $Q := Q \cup \{y\}$ şi $A := A \cup \{u\}$ şi trecem la etapa 1.

Exemplul 2.2.1 *Fie $G = (X, U)$ un graf neorientat, unde*

$$X = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6\}$$

şi

$$U = \{(x_0, x_1), (x_0, x_5), (x_1, x_3), (x_2, x_6), (x_3, x_5), (x_4, x_5)\} .$$

Să se determine arborele maximal ce conţine x_0.

Soluţie. Fie $R := \{x_0\}$, $Q := \{x_0\}$ şi $A = \emptyset$. Cum $x_0 \in Q$ alegem $x_1 \in X \setminus R$ care are proprietatea că $(x_0, x_1) \in U$. Fie $R := R \cup \{x_1\} = \{x_0, x_1\}$, $Q := Q \cup \{x_1\} = \{x_0, x_1\}$ şi $A := A \cup \{(x_0, x_1)\} = \{(x_0, x_1)\}$.

Alegem $x_0 \in Q$. Cum $(x_0, x_5) \in U$ adăugăm x_5 la Q şi R şi muchia (x_0, x_5) la A. Astfel $Q = \{x_0, x_1, x_5\}$, $R = \{x_0, x_1, x_5\}$ şi $A = \{(x_0, x_1), (x_0, x_5)\}$.

Alegem $x_0 \in Q$. Cum nu există $y \in X \setminus R$ astfel încât $(x_0, y) \in U$ punem $Q = Q \setminus \{x_0\} = \{x_1, x_5\}$.

Pentru $x_1 \in Q$, cum $(x_1, x_3) \in U$, adăugăm x_3 la Q şi R şi muchia (x_1, x_3) la A. Obţinem

$$Q = \{x_1, x_5, x_3\} , \ R = \{x_0, x_1, x_5, x_3\} , \ A = \{(x_0, x_1), (x_0, x_5), (x_1, x_3)\} .$$

Alegem $x_1 \in Q$. Cum nu există $y \in X \setminus R$ astfel încât $(x_1, y) \in U$ punem $Q = Q \setminus \{x_1\} = \{x_5, x_3\}$.

Pentru $x_5 \in Q$, cum $(x_4, x_5) \in U$, adăugăm x_4 la Q şi R şi muchia (x_4, x_5) la A. Obţinem $Q = \{x_5, x_3, x_4\}$, $R = \{x_0, x_1, x_5, x_3, x_4\}$ şi

$$A = \{(x_0, x_1), (x_0, x_5), (x_1, x_3), (x_4, x_5)\} .$$

Alegem $x_5 \in Q$. Cum nu există $y \in X \setminus R$ astfel încât $(x_5, y) \in U$ punem $Q = Q \setminus \{x_5\} = \{x_3, x_4\}$. Pentru $x_3 \in Q$ observăm că nu există $y \in X \setminus R$ astfel încât $(x_3, y) \in U$. Punem $Q = Q \setminus \{x_3\} = \{x_4\}$. Pentru $x_4 \in Q$ observăm că nu există $y \in X \setminus R$ astfel încât $(x_4, y) \in U$. Punem $Q = Q \setminus \{x_4\} = \emptyset$. Algoritmul s-a încheiat şi arborele căutat este (R, A).

Observţia 2.2.2 *Ideea acestui algoritm este ca la orice moment* (R, A) *să fie un arbore (respectiv o arborescenţă ce conţine* x_0 *ca rădăcină, în cazul unui graf orientat).*

Observaţia 2.2.3 *În legătură cu algoritmul prezentat se pune o întrebare firească: în ce ordine se aleg în etapa* 1 *vârfurile* $x \in Q$ *? Două metode sunt frecvent utilizate. Acestea sunt numite* ***parcurgerea în adâncime*** *(Depth-First Search) şi* ***parcurgerea în lăţime*** *(Breadth-First Search). În metoda DFS se alege acel vârf* $x \in Q$ *care a fost ultimul adăugat. În metoda BFS se alege acel vârf* $x \in Q$ *care a fost primul intrat. Mai mult despre parcurgerea unui graf poate fi găsit în ultimul capitol.*

2.2.2 Componente conexe

Algoritmul precedent poate fi aplicat pentru a determina componentele conexe ale unui graf. Astfel alegem un vârf x_0, aplicăm algoritmul şi verificăm dacă $R = X$. În caz afirmativ graful este conex. În caz contrar R este o componentă conexă maximală şi reluăm algoritmul alegând un vârf $x \in X \setminus R$ până când toate vârfurile au fost puse într-o componentă conexă maximală.

Observaţia 2.2.4 *Graful din exemplul* 2.2.1 *are două componente conexe:* $C(x_0) = \{x_0, x_1, x_5, x_3, x_4\},\ C(x_2) = \{x_2, x_6\}$.

2.3 Determinarea componentelor tare conexe

În această secţiune se consideră un graf orientat $G = (X, U)$.

2.3.1 Algoritmul Malgrange

Etapa 0. Pentru fiecare vârf x din X se determină mulţimea predecesorilor lui x şi mulţimea succesorilor lui x.

Etapa 1. Fixăm un vârf $x \in X$ şi determinăm componenta tare conexă $C(x)$ ce conţine vârful x.
(A) Vom calcula prin recurenţă

$$\Gamma^{-2}(x) = \Gamma^{-}(\Gamma^{-}(x))\ ;\ \cdots\ ;\ \Gamma^{-k}(x) = \Gamma^{-}(\Gamma^{-(k-1)}(x))$$

până când în mulţimea

$$\widehat{\Gamma}^{-}(x) = \{x\} \cup \Gamma^{-}(x) \cup \Gamma^{-2}(x) \cup \cdots \cup \Gamma^{-k}(x)\ ,$$

numită **închiderea tranzitivă inversă** a lui $x \in X$, nu se mai adugă vârfuri noi.
(B) Vom calcula prin recurenţă

$$\Gamma^{+2}(x) = \Gamma^{+}(\Gamma^{+}(x))\ ;\ \cdots\ ;\ \Gamma^{+k}(x) = \Gamma^{+}(\Gamma^{+(k-1)}(x))$$

până când în mulţimea

$$\widehat{\Gamma}^{+}(x) = \{x\} \cup \Gamma^{+}(x) \cup \Gamma^{+2}(x) \cup \cdots \cup \Gamma^{+k}(x)\ ,$$

numită **închiderea tranzitivă directă** a lui $x \in X$, nu se mai adugă vârfuri noi.
În final

$$C(x) = \widehat{\Gamma}^{-}(x) \cap \widehat{\Gamma}^{+}(x) \ .$$

Etapa 2. Se reia etapa 1 cu un alt vârf x' din $X \setminus C(x)$.

Procedeul continuă până când se epuizează toate vârfurile grafului.

Exemplul 2.3.1 *Să se determine componentele tare conexe pentru graful din figura de mai jos.*

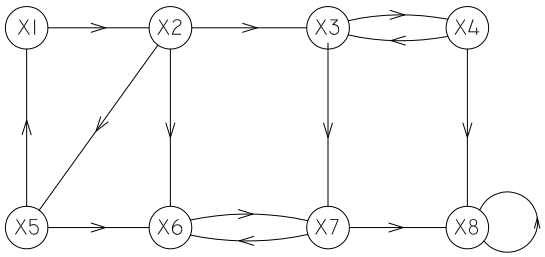


Figura 2.2:

Soluţie. **Etapa 0.** Pentru fiecare vârf vom determina mulţimea predecesorilor şi mulţimea succesorilor.

$$\Gamma^{-}(1) = \{5\} \ ; \ \Gamma^{-}(2) = \{1\} \ ; \ \Gamma^{-}(3) = \{2,4\} \ ; \ \Gamma^{-}(4) = \{3\} \ ;$$

$$\Gamma^{-}(5) = \{2\} \ ; \ \Gamma^{-}(6) = \{2,5,7\} \ ; \ \Gamma^{-}(7) = \{3,6\} \ ; \ \Gamma^{-}(8) = \{4,7,8\} \ .$$

$$\Gamma^{+}(1) = \{2\} \ ; \ \Gamma^{+}(2) = \{3,5,6\} \ ; \ \Gamma^{+}(3) = \{4,7\} \ ; \ \Gamma^{+}(4) = \{3,8\} \ ;$$

$$\Gamma^{+}(5) = \{1,6\} \ ; \ \Gamma^{+}(6) = \{7\} \ ; \ \Gamma^{+}(7) = \{6,8\} \ ; \ \Gamma^{+}(8) = \{8\} \ .$$

Etapa 1. Vom alege mai întâi vârful x_8, pentru că $\Gamma^{+}(8) = \{8\}$. Prin urmare $C(8) = \{8\}$.
Etapa 2. Alegem vârful x_1.
(A)

$$\Gamma^{-2}(1) = \Gamma^{-}(\Gamma^{-}(1)) = \Gamma^{-}(5) = \{2\} \ ;$$

$$\Gamma^{-3}(1) = \Gamma^{-}(\Gamma^{-2}(1)) = \Gamma^{-}(2) = \{1\} \ .$$

Prin urmare

$$\widehat{\Gamma}^-(1) = \{1, 2, 5\} .$$

(B)

$$\Gamma^{+2}(1) = \Gamma^+(\Gamma^+(1)) = \Gamma^+(2) = \{3, 5, 6\} ;$$
$$\Gamma^{+3}(1) = \Gamma^+(\Gamma^{+2}(1)) = \Gamma^+(\{3, 5, 6\}) = \{4, 7, 1, 6\} ;$$
$$\Gamma^{+4}(1) = \Gamma^+(\Gamma^{+3}(1)) = \Gamma^+(\{4, 7, 1, 6\}) = \{3, 8, 6, 2, 7\} .$$

Astfel

$$\widehat{\Gamma}^+(1) = \{1, 2, 3, 4, 5, 6, 7, 8\} \text{ şi } C(1) = \widehat{\Gamma}^-(1) \cap \widehat{\Gamma}^+(1) = \{1, 2, 5\} .$$

Etapa 3. Alegem vârful x_3.

(A)

$$\Gamma^{-2}(3) = \Gamma^-(\Gamma^-(3)) = \Gamma^-(\{2, 4\}) = \{1, 3\} ;$$
$$\Gamma^{-3}(3) = \Gamma^-(\Gamma^{-2}(3)) = \Gamma^-(\{1, 3\}) = \{5, 2, 4\} ;$$
$$\Gamma^{-4}(3) = \Gamma^-(\Gamma^{-3}(3)) = \Gamma^-(\{5, 2, 4\}) = \{2, 1, 3\} .$$

Deci

$$\widehat{\Gamma}^-(3) = \{1, 2, 3, 4, 5\} .$$

(B)

$$\Gamma^{+2}(3) = \Gamma^+(\Gamma^+(3)) = \Gamma^+(\{4, 7\}) = \{3, 8, 6\} ;$$
$$\Gamma^{+3}(3) = \Gamma^+(\Gamma^{+2}(3)) = \Gamma^+(\{3, 8, 6\}) = \{4, 7, 8\} .$$

Astfel

$$\widehat{\Gamma}^+(3) = \{3, 4, 6, 7, 8\} \text{ şi } C(3) = \widehat{\Gamma}^-(3) \cap \widehat{\Gamma}^+(3) = \{3, 4\} .$$

Etapa 4. Alegem vârful x_6.

(A)

$$\Gamma^{-2}(6) = \Gamma^-(\Gamma^-(6)) = \Gamma^-(\{2, 5, 7\}) = \{1, 2, 3, 6\} ;$$
$$\Gamma^{-3}(6) = \Gamma^-(\Gamma^{-2}(6)) = \Gamma^-(\{1, 2, 3, 6\}) = \{5, 1, 2, 4, 7\} ;$$
$$\Gamma^{-4}(6) = \Gamma^-(\Gamma^{-3}(6)) = \Gamma^-(\{5, 1, 2, 4, 7\}) = \{2, 5, 1, 3, 6\} .$$

Deci

$$\widehat{\Gamma}^-(6) = \{1, 2, 3, 4, 5, 6, 7\} .$$

(B)

$$\Gamma^{+2}(6) = \Gamma^+(\Gamma^+(6)) = \Gamma^+(7) = \{6, 8\} ;$$
$$\Gamma^{+3}(6) = \Gamma^+(\Gamma^{+2}(6)) = \Gamma^+(\{6, 8\}) = \{7, 8\} .$$

Astfel

$$\widehat{\Gamma}^+(6) = \{6, 7, 8\} \text{ şi } C(6) = \widehat{\Gamma}^-(6) \cap \widehat{\Gamma}^+(6) = \{6, 7\} .$$

2.3.2 Algoritmul Chen

Etapa 0. Se scrie matricea de adiacenţă. Se fixează un vârf x_k şi se determină componenta tare conexă ce conţine vârful x_k.
Etapa 1. Vom aduna boolean la linia k toate liniile j pentru care $a_{kj} = 1$. Se repetă această etapă până când pe linia lui k nu se mai obţin noi elemente egale cu 1. Fie

$$\widehat{\Gamma}^+(x_k) := \{x_k\} \cup \{x_j \ : \ a_{kj} = 1\} \ .$$

Etapa 2. Vom aduna boolean la coloana k toate coloanele j pentru care $a_{jk} = 1$. Se repetă această etapă până când nu se mai obţin noi elemente egale cu 1 pe coloana k. Fie

$$\widehat{\Gamma}^-(x_k) := \{x_k\} \cup \{x_j \ : \ a_{jk} = 1\} \ .$$

În final

$$C(x_k) = \widehat{\Gamma}^+(x_k) \cap \widehat{\Gamma}^-(x_k) \ .$$

Etapa 3. Se elimină toate liniile şi coloanele corespunzătoare vârfurilor din componenta tare găsită şi se repetă etapele 1 şi 2 fixând un alt vârf.
Procedeul continuă pâna când se epuizează toate vârfurile grafului.

Exemplul 2.3.2 *Aplicând algoritmul Chen să se determine componentele tare conexe pentru graful din figura 2.2.*

Soluţie.

$A =$

	1	2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	0
2	0	0	1	0	1	1	0	0
3	0	0	0	1	0	0	1	0
4	0	0	1	0	0	0	0	1
5	1	0	0	0	0	1	0	0
6	0	0	0	0	0	0	1	0
7	0	0	0	0	0	1	0	1
8	0	0	0	0	0	0	0	1

.

Alegem mai întâi vârful x_8 pentru că observăm că pe linia 8 toate elementele sunt 0 (excepţie ultimul). Prin urmare $\widehat{\Gamma}^+(x_8) = \{x_8\}$ şi astfel $C(x_8) = \{x_8\}$.

Eliminăm linia 8 şi coloana 8.

$$A_1 = \begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 3 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}.$$

Alegem vârful x_1. Adunăm boolean linia a 2-a la linia 1.

$$A_2 = \begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 3 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}.$$

Adunăm boolean la linia 1 liniile $2, 3, 5, 6$. Obţinem

$$A_3 = \begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 3 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}.$$

Avem $\widehat{\Gamma}^+(x_1) = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$. Revenim la matricea A_1 şi adunăm boolean coloana 5 la coloana 1.

$$A_4 = \begin{array}{c||c|c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline\hline 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 2 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ \hline 3 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 5 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline \end{array}.$$

Adunăm boolean coloanele 2 şi 5 la coloana 1.

$$A_5 = \begin{array}{c||c|c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline\hline 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 2 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ \hline 3 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 5 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline \end{array}.$$

Astfel $\widehat{\Gamma}^-(x_1) = \{x_1, x_2, x_5\}$ şi deci $C(x_1) = \widehat{\Gamma}^+(x_1) \cap \widehat{\Gamma}^-(x_1) = \{x_1, x_2, x_5\}$. Revenim la matricea A_1 şi eliminăm liniile $1, 2, 5$ şi coloanele $1, 2, 5$. Obţinem

$$A_6 = \begin{array}{c||c|c|c|c|} & 3 & 4 & 6 & 7 \\ \hline\hline 3 & 0 & 1 & 0 & 1 \\ \hline 4 & 1 & 0 & 0 & 0 \\ \hline 6 & 0 & 0 & 0 & 1 \\ \hline 7 & 0 & 0 & 1 & 0 \\ \hline \end{array}.$$

Alegem vârful x_3. Adunăm boolean la linia 3 liniile 4 şi 7. Obţinem

$$A_7 = \begin{array}{c||c|c|c|c|} & 3 & 4 & 6 & 7 \\ \hline\hline 3 & 1 & 1 & 1 & 1 \\ \hline 4 & 1 & 0 & 0 & 0 \\ \hline 6 & 0 & 0 & 0 & 1 \\ \hline 7 & 0 & 0 & 1 & 0 \\ \hline \end{array}.$$

Astfel $\widehat{\Gamma}^+(x_3) = \{x_3, x_4, x_6, x_7\}$. Revenim la matricea A_6 şi adunăm boolean coloana 4 la coloana 3. Obţinem

$$A_8 = \begin{array}{c||c|c|c|c|} & 3 & 4 & 6 & 7 \\ \hline\hline 3 & 1 & 1 & 0 & 1 \\ \hline 4 & 1 & 0 & 0 & 0 \\ \hline 6 & 0 & 0 & 0 & 1 \\ \hline 7 & 0 & 0 & 1 & 0 \\ \hline \end{array}.$$

Atunci $\widehat{\Gamma}^-(x_3) = \{x_3, x_4\}$ şi deci $C(x_3) = \widehat{\Gamma}^+(x_3) \cap \widehat{\Gamma}^-(x_3) = \{x_3, x_4\}$. Eliminăm din matricea A_6 liniile $3, 4$ şi coloanele $3, 4$.

Obţinem $A_9 =$
$\begin{array}{c||c|c|} & 6 & 7 \\ \hline\hline 6 & 0 & 1 \\ \hline 7 & 1 & 0 \\ \hline \end{array}$.

Alegem vârful x_6. Adunăm boolean linia 7 la linia 6.

Obţinem $A_{10} =$
$\begin{array}{c||c|c|} & 6 & 7 \\ \hline\hline 6 & 1 & 1 \\ \hline 7 & 1 & 0 \\ \hline \end{array}$.

Deci $\widehat{\Gamma}^+(x_6) = \{x_6, x_7\}$. Revenim la matricea A_9 şi adunăm boolean coloana 7 la coloana 6.

Obţinem $A_{11} =$
$\begin{array}{c||c|c|} & 6 & 7 \\ \hline\hline 6 & 1 & 1 \\ \hline 7 & 1 & 0 \\ \hline \end{array}$.

Avem $\widehat{\Gamma}^-(x_6) = \{x_6, x_7\}$. Atunci $C(x_6) = \widehat{\Gamma}^+(x_6) \cap \widehat{\Gamma}^-(x_6) = \{x_6, x_7\}$.

2.3.3 Algoritmul Foulkes

Acest algoritm se bazează pe adunarea şi înmulţirea booleană a matricilor.
Etapa 0. Se scrie matricea de adiacenţă A asociată grafului.
Etapa 1. Se calculează $A \oplus I$, unde I este matricea unitate.
Etapa k. Se calculează $(A \oplus I)^{(k)}$.
Algoritmul se încheie când $(A \oplus I)^{(k)} = (A \oplus I)^{(k-1)}$.
În matricea $(A \oplus I)^{(k)}$ se suprimă liniile $i_1, i_2, \cdots, i_p$ formate doar din cifra 1 şi coloanele corespunzătoare. Vârfurile corespunzătoare acestor linii formează prima componenta tare conexă, adică

$$C(x_{i_1}) = \{x_{i_1}, x_{i_2}, \cdots, x_{i_p}\} .$$

Se reia algoritmul cu matricea rămasă .

Exemplul 2.3.3 *Aplicând algoritmul lui Foulkes să se determine componentele tare conexe pentru graful din figura 2.2.*

Soluţie.

$A =$

	1	2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	0
2	0	0	1	0	1	1	0	0
3	0	0	0	1	0	0	1	0
4	0	0	1	0	0	0	0	1
5	1	0	0	0	0	1	0	0
6	0	0	0	0	0	0	1	0
7	0	0	0	0	0	1	0	1
8	0	0	0	0	0	0	0	1

$A \oplus I =$

	1	2	3	4	5	6	7	8
1	1	1	0	0	0	0	0	0
2	0	1	1	0	1	1	0	0
3	0	0	1	1	0	0	1	0
4	0	0	1	1	0	0	0	1
5	1	0	0	0	1	1	0	0
6	0	0	0	0	0	1	1	0
7	0	0	0	0	0	1	1	1
8	0	0	0	0	0	0	0	1

$(A \oplus I)^2 =$

	1	2	3	4	5	6	7	8
1	1	1	1	0	1	1	0	0
2	1	1	1	1	1	1	1	0
3	0	0	1	1	0	1	1	1
4	0	0	1	1	0	0	1	1
5	1	1	0	0	1	1	1	0
6	0	0	0	0	0	1	1	1
7	0	0	0	0	0	1	1	1
8	0	0	0	0	0	0	0	1

$(A \oplus I)^3 =$

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	0
2	1	1	1	1	1	1	1	1
3	0	0	1	1	0	1	1	1
4	0	0	1	1	0	1	1	1
5	1	1	1	0	1	1	1	1
6	0	0	0	0	0	1	1	1
7	0	0	0	0	0	1	1	1
8	0	0	0	0	0	0	0	1

$(A \oplus I)^4 =$

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	0	0	1	1	0	1	1	1
4	0	0	1	1	0	1	1	1
5	1	1	1	1	1	1	1	1
6	0	0	0	0	0	1	1	1
7	0	0	0	0	0	1	1	1
8	0	0	0	0	0	0	0	1

Observăm că $(A \oplus I)^5$ va fi egal cu $(A \oplus I)^4$. Astfel am găsit prima componentă tare conexă $C(x_1) = \{x_1, x_2, x_5\}$. Eliminăm liniile $1, 2, 5$ şi coloanele $1, 2, 5$.

Obţinem

$B =$

	3	4	6	7	8
3	1	1	1	1	1
4	1	1	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
8	0	0	0	0	1

.

Atunci $B^{(2)} =$

	3	4	6	7	8
3	1	1	1	1	1
4	1	1	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
8	0	0	0	0	1

Observăm că $B^{(2)} = B$. Astfel am găsit a doua componentă tare conexă $C(x_3) = \{x_3, x_4\}$. Eliminăm liniile $3, 4$ şi coloanele $3, 4$.

Matricea rămasă este $C =$

	6	7	8
6	1	1	1
7	1	1	1
8	0	0	1

. Atunci $C^{(2)} =$

	6	7	8
6	1	1	1
7	1	1	1
8	0	0	1

Cum $C^{(2)} = C$ algoritmul s-a încheiat şi am obţinut a treia componentă tare conexă $C(x_6) = \{x_6, x_7\}$. Eliminăm liniile $6, 7$ şi coloanele $6, 7$. Matricea rămasă este $D =$

	8
8	1

Prin urmare, a patra componentă tare conexă este $C(x_8) = \{x_8\}$.

2.4 Determinarea circuitelor euleriene

2.4.1 Introducere

Teorema 2.4.1 (Euler [1736]) *Un graf neorientat conex este eulerian dacă şi numai dacă fiecare vârf are grad par.*

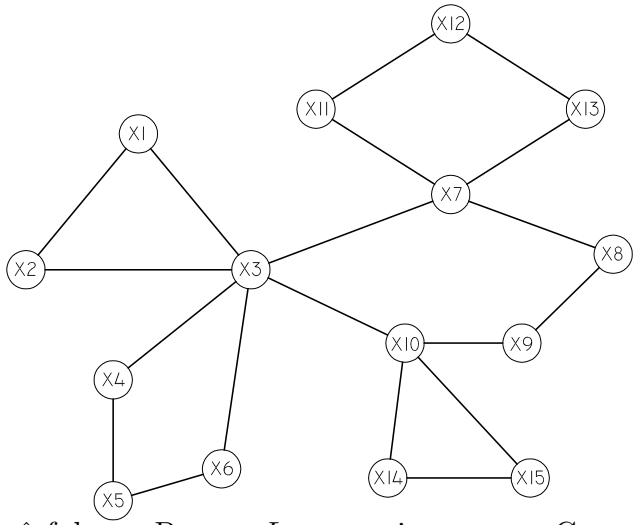
Fie $G = (X, U)$ un graf neorientat conex (verificarea conexităţii se poate face utilizând algoritmul de scanare a grafului). Presupunem că G este eulerian

(verificarea parităţii gradelor fiecărui vârf este imediată). Algoritmul prezentat mai jos va determina un ciclu eulerian în graful G. Ideea acestui algoritm este că plecând dintr-un vârf arbitrar al grafului, spre exemplu x_1, să determinăm un ciclu $L = L(x_1)$. Se alege apoi un alt vârf, spre exemplu x_i, al ciclului L pentru care mai există muchii incidente cu el, neluate încă. Se determină un nou ciclu $L(x_i)$ pe care îl concatenăm cu ciclul L obţinând un ciclu L mai lung. Se reia această etapă atât timp cât mai există muchii care nu au fost incluse în ciclul L.

2.4.2 Algoritmul lui Euler

Alegem un vârf arbitrar al grafului, spre exemplu x_1 şi determinăm $L := L(x_1)$.
Etapa 0. Punem $L := x_1$ şi $x := x_1$.
Etapa 1. Dacă $\Gamma(x) = \emptyset$ trecem la etapa 3. În caz contrar fie $y \in \Gamma(x)$.
Etapa 2. Punem $L := L, y$ şi $x := y$. Punem $U := U \setminus \{(x, y)\}$ şi trecem la etapa 1.
Etapa 3. Pentru fiecare vârf x_i din L determinăm $L := L(x_i)$.
Etapa 4. Intercalăm în L ciclurile L_i.

Exemplul 2.4.2 *Să se determine un ciclu eulerian pentru graful din figura de mai jos.*



Soluţie. Alegem vârful x_1. Punem $L := x_1$ şi $x := x_1$. Cum $(x_1, x_2) \in U$ punem $L := x_1, x_2$ şi $x := x_2$. Punem $U := U \setminus \{(x_1, x_2)\}$. Continuăm să

parcurgem muchiile grafului din x_2. Cum $(x_2, x_3) \in U$ adăugăm x_3 la L şi scădem din U muchia parcursă (x_1, x_3). Avem $L := x_1, x_2, x_3$.
După mai mulţi paşi obţinem $L = x_1, x_2, x_3, x_4, x_5, x_6, x_3, x_1$. Acum L este un ciclu.
Observăm că x_3 este singurul vârf din L care mai are muchii incidente neluate încă.
Determinăm ca şi mai sus $L_3 = L(x_3)$. Obţinem $L_3 = x_3, x_{10}, x_9, x_8, x_7, x_3$. Intercalăm în L ciclul L_3. Obţinem $L = x_1, x_2, L_3, x_4, x_5, x_6, x_3, x_1$ adică $L = x_1, x_2, x_3, x_{10}, x_9, x_8, x_7, x_3, x_4, x_5, x_6, x_3, x_1$. Acum L este un ciclu ce conţine vârfurile x_7 şi x_{10} care au muchii incidente neparcurse încă. Prin urmare vom determina ciclurile $L_7 = x_7, x_{11}, x_{12}, x_{13}, x_7$ şi $L_{10} = x_{10}, x_{14}, x_{15}, x_{10}$ pe care le intercalăm în L şi obţinem

$$L = x_1, x_2, x_3, L_{10}, x_9, x_8, L_7, x_3, x_4, x_5, x_6, x_3, x_1$$

adică

$$L = x_1, x_2, x_3, x_{10}, x_{14}, x_{15}, x_{10}, x_9, x_8, x_7, x_{11}, x_{12}, x_{13}, x_7, x_3, x_4, x_5, x_6, x_3, x_1 \ .$$

Observaţia 2.4.3 *În cazul unui graf orientat algoritmul se păstrează modificând doar etapa* 1. *Astfel aceasta devine:*
Etapa $1'$***.*** *Dacă* $\Gamma^+(x) = \emptyset$ *trecem la etapa* 3. *În caz contrar fie* $y \in \Gamma^+(x)$.

2.5 Drumuri şi circuite hamiltoniene

2.5.1 Algoritmul lui Kaufmann

Am văzut în secţiunea 2.1.4 că algoritmul lui Kaufmann este dedicat găsirii drumurilor elementare. În particular, drumurile elementare de lungime $n-1$ (unde n este numărul de vârfuri ale grafului) sunt tocmai drumurile hamiltoniene. Pentru a găsi şi circuitele hamiltoniene mai trebuie să completăm acest algoritm cu o singură etapă.
Etapa n. Se obţin circuitele hamiltoniene calculând

$$L_n^* = L_{n-1} \otimes_L L_R \ ,$$

operaţia de multiplicare latină fiind puţin modificată, în sensul că acum vom păstra acele cuvinte ce au un simbol comun cu condiţia ca ele să se găsească pe diagonala principală.

Exemplul 2.5.1 *Să se determine drumurile şi circuitele hamiltoniene pentru graful* $G = (X, U)$ *unde* $X = \{x_1, x_2, x_3, x_4, x_5\}$ *şi*

$$U = \{(1,2), (1,4), (1,5), (2,4), (3,1), (3,2), (3,5), (4,4), (4,5), (5,3)\} .$$

Soluţie. Etapele 1,2,3,4 au fost făcute în soluţia exemplului 2.1.13. Precizăm că în etapa 4 au fost găsite tocmai drumurile hamiltoniene.
Etapa 5.

$L_5^* =$

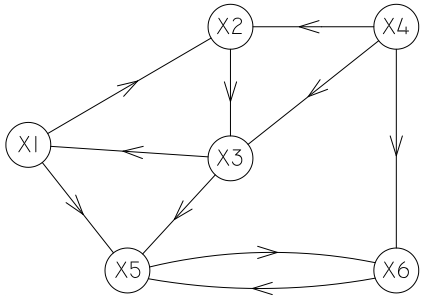
ABDECA				
	BDCEAB			
		CABDEC		
			DECABD	
				ECABDE

2.5.2 Algoritmul lui Foulkes

Etapa 1. Se determină componentele tare conexe ale grafului.
Etapa 2. Trecând de la o componentă tare conexă la alta, utilizând arce din graf, se determină toate drumurile hamiltoniene ale grafului.

Observaţia 2.5.2 *Algoritmul este eficient atunci când componentele tare conexe conţin un număr mic de vârfuri.*

Exemplul 2.5.3 *Aplicând algoritmul lui Foulkes să se determine drumurile hamiltoniene pentru graful din figura de mai jos.*



Soluţie.

$$A \oplus I = \begin{array}{|c|c|c|c|c|c|}\hline 1&1&0&0&1&0\\\hline 0&1&1&0&0&0\\\hline 1&0&1&0&1&0\\\hline 0&1&1&1&0&1\\\hline 0&0&0&0&1&1\\\hline 0&0&0&0&1&1\\\hline\end{array} \quad , \quad (A \oplus I)^{(2)} = \begin{array}{|c|c|c|c|c|c|}\hline 1&1&1&0&1&1\\\hline 1&1&1&0&1&0\\\hline 1&1&1&0&1&1\\\hline 1&1&1&1&1&1\\\hline 0&0&0&0&1&1\\\hline 0&0&0&0&1&1\\\hline\end{array}$$

$$(A \oplus I)^{(3)} = \begin{array}{|c|c|c|c|c|c|}\hline 1&1&1&0&1&1\\\hline 1&1&1&0&1&1\\\hline 1&1&1&0&1&1\\\hline 1&1&1&1&1&1\\\hline 0&0&0&0&1&1\\\hline 0&0&0&0&1&1\\\hline\end{array} \quad , \quad C(x_4) = \{x_4\}$$

$$B = \begin{array}{|c|c|c|c|c|}\hline 1&1&1&1&1\\\hline 1&1&1&1&1\\\hline 1&1&1&1&1\\\hline 0&0&0&1&1\\\hline 0&0&0&1&1\\\hline\end{array} \quad , \quad B^{(2)} = \begin{array}{|c|c|c|c|c|}\hline 1&1&1&1&1\\\hline 1&1&1&1&1\\\hline 1&1&1&1&1\\\hline 0&0&0&1&1\\\hline 0&0&0&1&1\\\hline\end{array}$$

Se găsesc componentele tare conexe $C(x_1) = \{x_1, x_2, x_3\}$ şi $C(x_5) = \{x_5, x_6\}$. Drumul hamiltonian este $x_4, x_2, x_3, x_1, x_5, x_6$.

2.5.3 Algoritmul lui Chen

Acest algoritm se bazează pe următoarele teoreme:

Teorema 2.5.4 (Chen) *Un graf orientat cu n vârfuri, fără circuite conţine un drum hamiltonian dacă şi numai dacă*

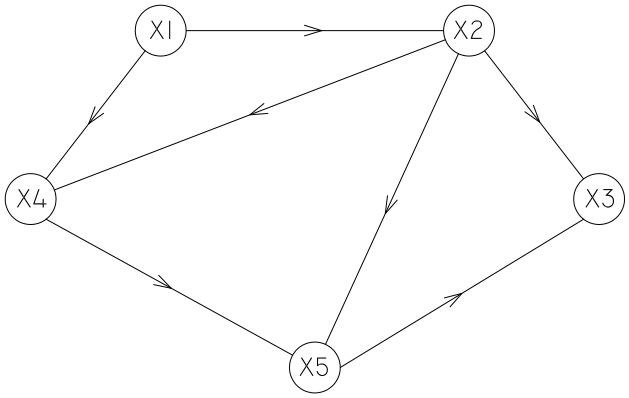
$$\sum_{i=1}^{n} p(x_i) = \frac{n(n-1)}{2} ,$$

unde $p(x_i)$ este puterea de atingere a vârfului x_i.

Teorema 2.5.5 *Dacă într-un graf orientat fără circuite există un drum hamiltonian atunci acesta este unic.*

Etapa 1. Se determină matricea drumurilor D.
Etapa 2. Dacă există un indice $i \in \{1,\ 2,\ ,\ldots,\ n\}$ astfel încât $d_{ii} = 1$ atunci graful are circuite şi nu se poate aplica algoritmul lui Chen.
Etapa 3. Se calculează puterea de atingere pentru fiecare vârf.
Dacă $\sum_{i=1}^{n} p(x_i) \neq \frac{n(n-1)}{2}$ atunci graful nu are drum hamiltonian.
Dacă $\sum_{i=1}^{n} p(x_i) = \frac{n(n-1)}{2}$ se ordonează vârfurile în ordine descrescătoare a puterilor lor de atingere şi se obţine drumul hamiltonian.

Exemplul 2.5.6 *Aplicând algoritmul lui Chen să se determine drumul hamiltonian pentru graful din figura de mai jos.*



Soluţie. **Etapa 1.**

$$A = \begin{array}{|c|c|c|c|c|}\hline 0&1&0&1&0\\\hline 0&0&1&1&1\\\hline 0&0&0&0&0\\\hline 0&0&0&0&1\\\hline 0&0&1&0&0\\\hline\end{array}\ ,\quad A \oplus I = \begin{array}{|c|c|c|c|c|}\hline 1&1&0&1&0\\\hline 0&1&1&1&1\\\hline 0&0&1&0&0\\\hline 0&0&0&1&1\\\hline 0&0&1&0&1\\\hline\end{array}$$

$$(A \oplus I)^{(2)} = \begin{array}{|c|c|c|c|c|}\hline 1&1&1&1&1\\\hline 0&1&1&1&1\\\hline 0&0&1&0&0\\\hline 0&0&1&1&1\\\hline 0&0&1&0&1\\\hline\end{array}\ ,\quad (A \oplus I)^{(4)} = \begin{array}{|c|c|c|c|c|}\hline 1&1&1&1&1\\\hline 0&1&1&1&1\\\hline 0&0&1&0&0\\\hline 0&0&1&1&1\\\hline 0&0&1&0&1\\\hline\end{array}$$

$$D = A \otimes (A \oplus I)^{(4)} = \begin{array}{|c|c|c|c|c|}\hline 0 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

Etapa 2. Observăm că în matricea D toate elementele de pe diagonala principală sunt zero. Prin urmare graful nu are circuite şi algoritmul lui Chen poate fi aplicat.
Etapa 3. Remarcăm că $p(x_1) = 4;\ p(x_2) = 3;\ p(x_3) = 0;\ p(x_4) = 2;$ $p(x_5) = 1$. Atunci $\sum_{i=1}^{5} p(x_i) = 10$. Dar $\frac{n(n-1)}{2} = \frac{5 \cdot 4}{2} = 10$. Algoritmul lui Chen ne spune că există un drum hamiltonian care se obţine ordonând vârfurile în ordine descrescătoare a puterii lor de atingere: $x_1,\ x_2,\ x_4,\ x_5,\ x_3$.

2.6 Drumuri de valoare optimă

Considerăm un graf orientat, valorizat, fără bucle. Suntem interesaţi să găsim un drum de valoare minimă sau maximă între două vârfuri date ale acestui graf. Binenţeles, dacă $\mu = (u_1, u_2, \cdots, u_p)$ atunci

$$v(\mu) = v(u_1) + v(u_2) + \cdots + v(u_p)$$

se numeşte **valoarea drumului** μ.

2.6.1 Algoritmul lui Ford

Acest algoritm determină drumurile de valoare optimă de la un vârf fixat (spre exemplu $x_1 \in X$) la celelalte vârfuri ale grafului.
Etapele algoritmului pentru o problemă de minim:
Etapa 1. Vom marca fiecare vârf al grafului un număr λ_i, care reprezintă valoarea unui drum arbitrar de la x_1 la x_i. Deci $\lambda_1 = 0$ şi $\lambda_i = v(\mu(x_1, x_i))$.
Etapa 2. Pentru $(x_i, x_j) \in U$ calculăm $t_{ij} = \lambda_j - \lambda_i$. Dacă $t_{ij} \leq v_{ij}$ se trece la etapa finală.
Etapa 3. Dacă există $t_{ij} > v_{ij}$ vom pune $\lambda'_j = \lambda_i + v_{ij}$, celelalte marcaje rămânând neschimbate. Revenim la etapa 2.
Etapa finală. Drumul de valoare minimă este format din acele arce pentru care $t_{ij} = v_{ij}$.

Exemplul 2.6.1 *Să se determine drumurile de valoare minimă de la* x_1 *la* $x_i, (i = \overline{2,9})$ *în următorul graf valorizat:*

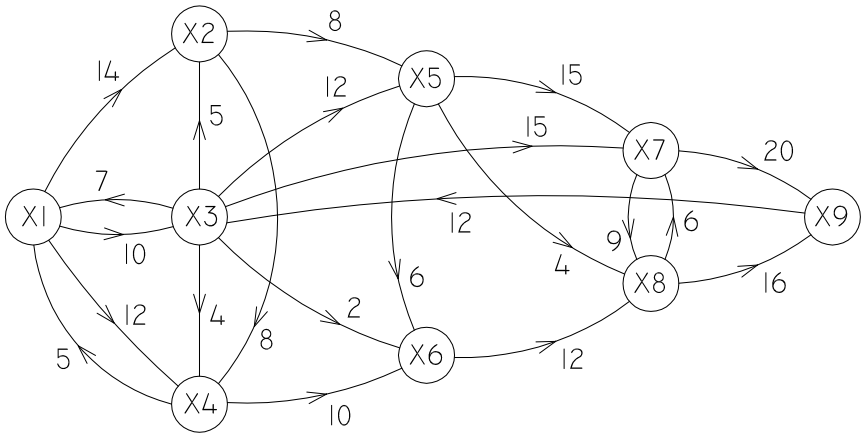


Figura 2.3:

Soluţie.

ij	12	13	14	24	25	31	32	34	35	36	37
v_{ij}	14	10	12	8	8	7	5	4	12	2	15
t_{ij}	14	10	12	-2	8	-10	4	2	12	2	15
t_{ij}	14	10	12	-2	8	-10	4	2	12	2	15
t_{ij}	14	10	12	-2	8	-10	4	2	12	2	15

ij	41	46	56	57	58	68	78	79	87	89	93
v_{ij}	5	10	6	15	4	12	9	20	6	16	12
t_{ij}	-12	0	-10	3	4	14	1	17	-1	16	-32
t_{ij}	-12	0	-10	3	2	12	-1	17	1	18	-32
t_{ij}	-12	0	-10	3	2	12	-1	15	1	16	-30

Tabelul marcajelor este

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
λ_i	0	14	10	12	22	12	25	26	42
λ_i	0	14	10	12	22	12	25	24	42
λ_i	0	14	10	12	22	12	25	24	40

,

unde $\lambda'_8 = \lambda_6 + v_{68} = 12 + 12 = 24$. Revenim la etapa 2.
Apoi $\lambda'_9 = \lambda_8 + v_{89} = 24 + 16 = 40$. In final
$\mu(x_1, x_2) = (1, 2)$, $v(\mu(x_1, x_2)) = 14$.
$\mu(x_1, x_3) = (1, 3)$, $v(\mu(x_1, x_3)) = 10$.
$\mu(x_1, x_4) = (1, 4)$, $v(\mu(x_1, x_4)) = 12$.
$\mu(x_1, x_5) = (1, 2, 5)$ sau $\mu(x_1, x_5) = (1, 3, 5)$ şi $v(\mu(x_1, x_5)) = 22$.
$\mu(x_1, x_6) = (1, 3, 6)$, $v(\mu(x_1, x_6)) = 12$.
$\mu(x_1, x_7) = (1, 3, 7)$, $v(\mu(x_1, x_7)) = 25$.
$\mu(x_1, x_8) = (1, 3, 6, 8)$, $v(\mu(x_1, x_8)) = 24$.
$\mu(x_1, x_9) = (1, 3, 6, 8, 9)$, $v(\mu(x_1, x_9)) = 40$.

Observaţia 2.6.2 *In cazul unei probleme de maxim algoritmul se încheie când $t_{ij} \geq v_{ij}$. Prin urmare, vom schimba marcajul vârfului $x_j \in X$, punând $\lambda'_j = \lambda_i + v_{ij}$, dacă $t_{ij} < v_{ij}$.*

2.6.2 Algoritmul Bellman-Kalaba

Acest algoritm determină drumurile de valoare optimă ce unesc vârfurile grafului cu un vârf fixat (spre exemplu x_n).
Etapele algoritmului pentru o problemă de minim:
Etapa 0. Construim matricea $V = (v_{ij})$ unde

$$v_{ij} = \begin{cases} v(x_i, x_j) & \text{dacă } (x_i, x_j) \in U \text{ şi } i \neq j; \\ 0 & \text{dacă } i = j; \\ \infty & \text{în rest.} \end{cases}$$

Etapa 1. Se pune $a_i^{(1)} = v_{in}$;
Etapa m. Se calculează

$$a_i^{(m)} = \min_j \{a_j^{(m-1)} + v_{ij}\} .$$

Algoritmul se încheie dacă $a_i^{(m)} = a_i^{(m-1)}$. In această situaţie drumurile de valoare optimă sunt formate din acele arce $(x_i, x_j) \in U$ pentru care $a_i^{(m)} = v_{ij} + a_j^{(m-1)}$.
Dacă există i astfel încât $a_i^{(m)} \neq a_i^{(m-1)}$ se trece la etapa m+1.

Exemplul 2.6.3 *Să se determine drumurile de valoare minimă de la x_i, ($i = \overline{1, 8}$), la x_9 în graful valorizat din figura 2.3.*

Soluţie.

V	1	2	3	4	5	6	7	8	9
1	0	14	$\boxed{10}$	12	∞	∞	∞	∞	∞
2	∞	0	∞	8	$\boxed{8}$	∞	∞	∞	∞
3	7	5	0	4	12	$\boxed{2}$	15	∞	∞
4	5	∞	∞	0	∞	$\boxed{10}$	∞	∞	∞
5	∞	∞	∞	∞	0	6	15	$\boxed{4}$	∞
6	∞	∞	∞	∞	∞	0	∞	$\boxed{12}$	∞
7	∞	∞	∞	∞	∞	∞	0	9	$\boxed{20}$
8	∞	∞	∞	∞	∞	∞	6	0	$\boxed{16}$
9	∞	∞	12	∞	∞	∞	∞	∞	0
$a_i^{(1)}$	∞	∞	∞	∞	∞	∞	20	16	0
$a_i^{(2)}$	∞	∞	35	∞	20	28	20	16	0
$a_i^{(3)}$	45	28	30	38	20	28	20	16	0
$a_i^{(4)}$	40	28	30	38	20	28	20	16	0
$a_i^{(5)}$	40	28	30	38	20	28	20	16	0

In concluzie:
$\mu(x_1, x_9) = (1, 3, 6, 8, 9)$, $v(\mu(x_1, x_9)) = 40$;
$\mu(x_2, x_9) = (2, 5, 8, 9)$, $v(\mu(x_2, x_9)) = 28$;
$\mu(x_3, x_9) = (3, 6, 8, 9)$, $v(\mu(x_3, x_9)) = 30$;
$\mu(x_4, x_9) = (4, 6, 8, 9)$, $v(\mu(x_4, x_9)) = 38$;
$\mu(x_5, x_9) = (5, 8, 9)$, $v(\mu(x_5, x_9)) = 20$;
$\mu(x_6, x_9) = (6, 8, 9)$, $v(\mu(x_6, x_9)) = 28$;
$\mu(x_7, x_9) = (7, 9)$, $v(\mu(x_7, x_9)) = 20$;
$\mu(x_8, x_9) = (8, 9)$, $v(\mu(x_8, x_9)) = 16$.

Observaţia 2.6.4 *In cazul unei probleme de maxim matricea V se scrie asemănător cu precizarea că $v_{ij} = -\infty$ dacă $(x_i, x_j) \notin U$. La etapa m în loc de minim se va lucra cu maxim.*

2.6.3 Algoritmul lui Dijkstra

Algoritmul lui Dijkstra determină drumurile de valoare minimă de la un vârf fixat (spre exemplu $x_1 \in X$) la celelalte vârfuri ale grafului precum şi lungimea lor. Mai precis vom determina $d(x)$ şi $P(x)$ pentru fiecare vârf

$x \in X \setminus \{x_1\}$, unde $d(x)$ reprezintă lungimea drumului de valoare minimă de la x_1 la x iar $P(x)$ reprezintă predecesorul lui x, aceasta însemnând că drumul optim de la x_1 la x este $(x_1, P(x), x)$. Dacă vârful x nu poate fi atins plecând din x_1 atunci $d(x) = \infty$ şi $P(x)$ este nedefinit. Algoritmul utilizează o mulţime R formată cu vârfurile pentru care valorile finale corespunzătoare drumurilor optime de la vârful x_1 au fost determinate. Iniţial $R = \emptyset$ şi la fiecare iteraţie se mai adaugă un vârf la R.

Etapa 0. (Iniţializare)

$$d(x_1) = 0;\ d(x) = \infty,\ P(x) = \emptyset,\ (\forall)x \in X \setminus \{x_1\}\ ; R = \emptyset\ .$$

Etapa 1. Se alege un vârf $x \in X \setminus R$ astfel încât

$$d(x) = \min_{y \in X \setminus R} d(y)\ .$$

Etapa 2. $R := R \cup \{x\}$.

Etapa 3. Pentru fiecare $y \in X \setminus R$ cu proprietatea $(x, y) \in U$ executăm: Dacă $d(y) > d(x) + v(x, y)$ atunci punem $d(y) := d(x) + v(x, y)$ şi $P(y) = x$. În caz contrar $d(y)$ şi $P(y)$ rămân neschimbate.

Etapa 4. Dacă $R \neq X$ se trece la etapa 1. Dacă $R = X$ algoritmul s-a încheiat.

Exemplul 2.6.5 *Aplicând algoritmul lui Dijkstra să se determine drumurile de valoare minimă de la x_1 la x_i (pentru $i = \overline{2,6}$) în următorul graf valorizat:*

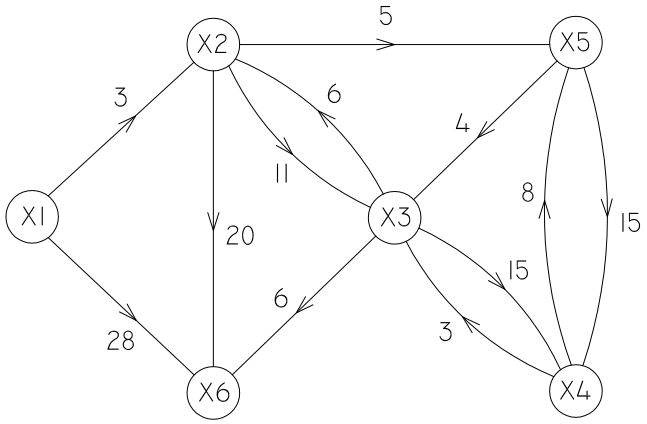


Figura 2.4:

Soluţie.

Iteraţia	0	1	2	3	4	5	6
$R := R \cup \{x\}$	$\emptyset$	$\{x_1\}$	$\{x_2\}$	$\{x_5\}$	$\{x_3\}$	$\{x_6\}$	$\{x_4\}$
$d(x_2)/P(x_2)$	$\infty/-$	$3/x_1$	$3/x_1$	$3/x_1$	$3/x_1$	$3/x_1$	$3/x_1$
$d(x_3)/P(x_3)$	$\infty/-$	$\infty/-$	$14/x_2$	$12/x_5$	$12/x_5$	$12/x_5$	$12/x_5$
$d(x_4)/P(x_4)$	$\infty/-$	$\infty/-$	$\infty/-$	$23/x_5$	$23/x_5$	$23/x_5$	$23/x_5$
$d(x_5)/P(x_5)$	$\infty/-$	$\infty/-$	$8/x_2$	$8/x_2$	$8/x_2$	$8/x_2$	$8/x_2$
$d(x_6)/P(x_6)$	$\infty/-$	$28/x_1$	$23/x_2$	$23/x_2$	$18/x_3$	$18/x_3$	$18/x_3$

Vom explica construcţia tabelului de mai sus.
Iniţializare: $d(x_1) = 0;\ d(x_i) = \infty,\ (\forall) i = \overline{2,6}\ ;\ R = \emptyset$.
Iteraţia 1: Cum $d(x_1) = \min\limits_{y \in X \setminus R} d(y)$ punem $R = R \cup \{x_1\} = \{x_1\}$. Cum $(x_1, x_2), (x_1, x_6) \in U$ vom pune

$$d(x_2) = d(x_1) + v(x_1, x_2) = 0 + 3 = 3;\ P(x_2) = x_1\ ;$$

$$d(x_6) = d(x_1) + v(x_1, x_6) = 0 + 28 = 28;\ P(x_6) = x_1\ .$$

Iteraţia 2: Cum $d(x_2) = \min\limits_{y \in X \setminus R} d(y)$ punem $R = R \cup \{x_2\} = \{x_1, x_2\}$. Cum $28 = d(x_6) > d(x_2) + v(x_2, x_6) = 3 + 20 = 23$, vom pune $d(x_6) = 23$ şi $P(x_6) = x_2$. Apoi

$$d(x_3) = d(x_2) + v(x_2, x_3) = 3 + 11 = 14;\ P(x_3) = x_2\ ;$$

$$d(x_5) = d(x_2) + v(x_2, x_5) = 3 + 5 = 8;\ P(x_5) = x_2\ .$$

Iteraţia 3: Cum $d(x_5) = \min\limits_{y \in X \setminus R} d(y)$ punem $R = R \cup \{x_5\} = \{x_1, x_2, x_5\}$. Cum $14 = d(x_3) > d(x_5) + v(x_5, x_3) = 8 + 4 = 12$, vom pune $d(x_3) = 12$ şi $P(x_3) = x_5$. Apoi

$$d(x_4) = d(x_5) + v(x_5, x_4) = 8 + 15 = 23;\ P(x_4) = x_5\ .$$

Iteraţia 4: Cum $d(x_3) = \min\limits_{y \in X \setminus R} d(y)$ vom adăuga x_3 la R. Deci

$$R = R \cup \{x_3\} = \{x_1, x_2, x_5, x_3\}\ .$$

Cum

$$23 = d(x_4) < d(x_3) + v(x_3, x_4) = 12 + 15 = 27 \ ,$$

$d(x_4)$ şi $P(x_4)$ rămân neschimbate. Dar

$$23 = d(x_6) > d(x_3) + v(x_3, x_6) = 12 + 6 = 18 \ .$$

Prin urmare vom pune $d(x_6) = 18$ şi $P(x_6) = x_3$.
Iteraţia 5: Se adaugă x_6 la R.
Iteraţia 6: Se adaugă x_4 la R.
Interpretarea tabelului:
$\mu(x_1, x_2) = (x_1, x_2);\ v(\mu(x_1, x_2)) = 3$.
$\mu(x_1, x_3) = (x_1, x_2, x_5, x_3);\ v(\mu(x_1, x_3)) = 12$.
$\mu(x_1, x_4) = (x_1, x_2, x_5, x_4);\ v(\mu(x_1, x_4)) = 23$.
$\mu(x_1, x_5) = (x_1, x_2, x_5);\ v(\mu(x_1, x_5)) = 8$.
$\mu(x_1, x_6) = (x_1, x_2, x_5, x_3, x_6);\ v(\mu(x_1, x_6)) = 18$.

2.6.4 Algoritmul Floyd-Warshall

Acest algoritm determină drumurile de valoare minimă dintre toate perechile de vârfuri ale unui graf orientat $G = (X, U)$ valorizat. Funcţia $v : U \to \mathbb{R}$ poate lua şi valori negative dar vom presupune că nu există cicluri de cost negativ.
Etapa 0. Construim matricea $V = (v_{ij})_{i,j=1}^{n}$, unde

$$v_{ij} = \begin{cases} v(x_i, x_j), & \text{dacă } (x_i, x_j) \in U \text{ şi } i \neq j \\ 0, & \text{dacă } i = j \\ \infty, & \text{în rest} \end{cases} \ .$$

Construim matricea $P = (p_{ij})_{i,j=1}^{n}$, unde

$$p_{ij} = \begin{cases} \emptyset, & \text{dacă } i = j \text{ sau } v_{ij} = \infty \\ i, & \text{dacă } i \neq j \text{ şi } v_{ij} < \infty \end{cases} \ .$$

Aplicăm matricilor V şi P un şir de n transformări T_k $(k = \overline{1, n})$ calculând $V := T_k(V);\ P := T_k(P)$.
Etapa k. Dacă $v_{ij} \leq v_{ik} + v_{kj}$ atunci v_{ij} şi p_{ij} rămân neschimbate.
Dacă $v_{ij} > v_{ik} + v_{kj}$ atunci v_{ij} se înlocuieşte cu $v_{ik} + v_{kj}$ iar p_{ij} devine p_{kj}.

La sfârşitul algoritmului se obţine matricea $V = T_n(V) = (v_{ij})$, ale cărei elemente reprezintă valoarea drumului minim de la x_i la x_j, şi matricea $P = T_n(P) = (p_{ij})$ numită **matricea predecesorilor**, unde $p_{ij} = \emptyset$ în cazul în care $i = j$ sau nu există drum de la x_i la x_j iar dacă $p_{ij} = l$ atunci predecesorul lui x_j într-un drum de valoare minimă de la x_i este x_l.

Observaţia 2.6.6 *v_{ik} este proiecţia lui v_{ij} pe coloana k şi v_{kj} este proiecţia lui v_{ij} pe linia k. În fapt v_{ij} devine* $\min\{v_{ij}, v_{ik} + v_{kj}\}$.

Exemplul 2.6.7 *Să se determine drumurile de valoare minimă între toate perechile de vârfuri ale grafului din figura 2.4 precum şi valorile acestora.*

Soluţie.

$$V = \begin{pmatrix} 0 & 3 & \infty & \infty & \infty & 28 \\ \infty & 0 & 11 & \infty & 5 & 20 \\ \infty & 6 & 0 & 15 & \infty & 6 \\ \infty & \infty & 3 & 0 & 8 & \infty \\ \infty & \infty & 4 & 15 & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}, \quad P = \begin{pmatrix} \emptyset & 1 & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & 2 & \emptyset & 2 & 2 \\ \emptyset & 3 & \emptyset & 3 & \emptyset & 3 \\ \emptyset & \emptyset & 4 & \emptyset & 4 & \emptyset \\ \emptyset & \emptyset & 5 & 5 & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

$$V = T_1(V) = \begin{pmatrix} 0 & 3 & \infty & \infty & \infty & 28 \\ \infty & 0 & 11 & \infty & 5 & 20 \\ \infty & 6 & 0 & 15 & \infty & 6 \\ \infty & \infty & 3 & 0 & 8 & \infty \\ \infty & \infty & 4 & 15 & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}, \quad P = \begin{pmatrix} \emptyset & 1 & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & 2 & \emptyset & 2 & 2 \\ \emptyset & 3 & \emptyset & 3 & \emptyset & 3 \\ \emptyset & \emptyset & 4 & \emptyset & 4 & \emptyset \\ \emptyset & \emptyset & 5 & 5 & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

$$V = T_2(V) = \begin{pmatrix} 0 & 3 & 14 & \infty & 8 & 23 \\ \infty & 0 & 11 & \infty & 5 & 20 \\ \infty & 6 & 0 & 15 & 11 & 6 \\ \infty & \infty & 3 & 0 & 8 & \infty \\ \infty & \infty & 4 & 15 & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}, \quad P = \begin{pmatrix} \emptyset & 1 & 2 & \emptyset & 2 & 2 \\ \emptyset & \emptyset & 2 & \emptyset & 2 & 2 \\ \emptyset & 3 & \emptyset & 3 & 2 & 3 \\ \emptyset & \emptyset & 4 & \emptyset & 4 & \emptyset \\ \emptyset & \emptyset & 5 & 5 & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

$$V = T_3(V) = \begin{pmatrix} 0 & 3 & 14 & 29 & 8 & 20 \\ \infty & 0 & 11 & 26 & 5 & 17 \\ \infty & 6 & 0 & 15 & 11 & 6 \\ \infty & 9 & 3 & 0 & 8 & 9 \\ \infty & 10 & 4 & 15 & 0 & 10 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}, \quad P = \begin{pmatrix} \emptyset & 1 & 2 & 3 & 2 & 3 \\ \emptyset & \emptyset & 2 & 3 & 2 & 3 \\ \emptyset & 3 & \emptyset & 3 & 2 & 3 \\ \emptyset & 3 & 4 & \emptyset & 4 & 3 \\ \emptyset & 3 & 5 & 5 & \emptyset & 3 \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

$$V = T_4(V) = \begin{pmatrix} 0 & 3 & 14 & 29 & 8 & 20 \\ \infty & 0 & 11 & 26 & 5 & 17 \\ \infty & 6 & 0 & 15 & 11 & 6 \\ \infty & 9 & 3 & 0 & 8 & 9 \\ \infty & 10 & 4 & 15 & 0 & 10 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}, \quad P = \begin{pmatrix} \emptyset & 1 & 2 & 3 & 2 & 3 \\ \emptyset & \emptyset & 2 & 3 & 2 & 3 \\ \emptyset & 3 & \emptyset & 3 & 2 & 3 \\ \emptyset & 3 & 4 & \emptyset & 4 & 3 \\ \emptyset & 3 & 5 & 5 & \emptyset & 3 \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

$$V = T_5(V) = \begin{pmatrix} 0 & 3 & 12 & 23 & 8 & 18 \\ \infty & 0 & 9 & 20 & 5 & 15 \\ \infty & 6 & 0 & 15 & 11 & 6 \\ \infty & 9 & 3 & 0 & 8 & 9 \\ \infty & 10 & 4 & 15 & 0 & 10 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}, \quad P = \begin{pmatrix} \emptyset & 1 & 5 & 5 & 2 & 3 \\ \emptyset & \emptyset & 5 & 5 & 2 & 3 \\ \emptyset & 3 & \emptyset & 3 & 2 & 3 \\ \emptyset & 3 & 4 & \emptyset & 4 & 3 \\ \emptyset & 3 & 5 & 5 & \emptyset & 3 \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

$$V = T_6(V) = \begin{pmatrix} 0 & 3 & 12 & 23 & 8 & 18 \\ \infty & 0 & 9 & 20 & 5 & 15 \\ \infty & 6 & 0 & 15 & 11 & 6 \\ \infty & 9 & 3 & 0 & 8 & 9 \\ \infty & 10 & 4 & 15 & 0 & 10 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}, \quad P = \begin{pmatrix} \emptyset & 1 & 5 & 5 & 2 & 3 \\ \emptyset & \emptyset & 5 & 5 & 2 & 3 \\ \emptyset & 3 & \emptyset & 3 & 2 & 3 \\ \emptyset & 3 & 4 & \emptyset & 4 & 3 \\ \emptyset & 3 & 5 & 5 & \emptyset & 3 \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

În concluzie: cum $p_{41} = \emptyset$ înseamnă că nu există drum din x_4 în x_1.
Dacă suntem interesaţi de un drum de valoare minimă din x_1 în x_6, cum $v_{16} = 18$ avem că valoarea acestuia este 18. Apoi $p_{16} = 3$ înseamnă că predecesorul lui x_6 este x_3. Cum $p_{13} = 5$ avem că predecesorul lui x_3 este x_5. Dar $p_{15} = 2$ şi deci predecesorul lui x_5 este x_2. Apoi $p_{12} = 1$ ne spune că predecesorul lui x_2 este x_1. Prin urmare drumul este $\mu = (x_1, x_2, x_5, x_3, x_6)$.
Apoi $v_{42} = 9$ ne spune că valoarea minimă a unui drum din x_4 în x_2 este 9. Pentru a găsi acest drum observăm că $p_{42} = 3$, adică predecesorul lui x_2 este x_3. Cum $p_{43} = 4$ avem că predecesorul lui x_3 este x_4. Deci drumul este $\mu = (x_4, x_3, x_2)$.

2.7 Arbore de acoperire minim

Fie $G = (X, U)$ un graf neorientat, conex, valorizat. Căutăm $A \subset U$, A fără cicluri, care conectează toate vârfurile şi a cărei cost total $v(A) := \sum_{u \in A} v(u)$

este minim. A va fi numit **arbore de acoperire minim**.
Ideea unui astfel de algoritm:
- algoritmul foloseşte o mulţime A (iniţial $A = \emptyset$) care la fiecare pas este o submulţime a unui arbore de acoperire minim. La fiecare pas se determină o muchie u care poate fi adăugată la A respectând proprietatea de mai sus, în sensul că $A \cup \{u\}$ este de asemenea o submulţime a unui arbore de acoperire minim. O astfel de muchie u se numeşte **muchie sigură** pentru A.
- în orice moment al execuţiei algoritmului $G_A = (X, A)$ este o pădure şi fiecare din componentele conexe ale lui G_A este un arbore (iniţial, când $A = \emptyset$, pădurea conţine n arbori, câte unul pentru fiecare vârf). La fiecare iteraţie se reduce numărul de arbori cu 1. Când pădurea conţine un singur arbore, algoritmul se încheie.
- orice muchie sigură pentru A uneşte componente distincte ale lui A.
Există mulţi algoritmi pentru determinarea arborelui parţial de cost minim. Dintre care cei mai cunoscuţi sunt algoritmul lui Kruskal şi algoritmul lui Prim. Chiar dacă nu generează toate soluţiile posibile, ei generează într-un timp scurt o singură soluţie optimă. Dacă toate costurile muchiilor sunt diferite între ele atunci soluţia este unică.

2.7.1 Algoritmul lui Kruskal

Etapa 1. Punem $A = \emptyset$. Se formează n arbori, câte unul pentru fiecare vârf.
Etapa 2. Se ordonează muchiile crescător după cost.
Etapa 3. Se alege o muchie (x_i, x_j).
3.1 Dacă vârfurile terminale ale acestei muchii aparţin aceluiaşi arbore se trece la următoarea muchie.
3.2 Dacă vârfurile terminale ale acestei muchii aparţin la arbori diferiţi se adaugă muchia (x_i, x_j) la A şi se unesc vârfurile din cei doi arbori. Se trece la următoarea muchie.

Exemplul 2.7.1 *Să se determine arborele de acoperire minim pentru graful din figura 2.5.*

Soluţie. Punem $A = \emptyset$. Alegem muchia (x_3, x_5). Cum vârfurile terminale ale acestei muchii aparţin la arbori diferiţi se adaugă această muchie la A. Deci $A = \{(x_3, x_5)\}$. După mai mulţi paşi

$$A = \{(x_3, x_5), (x_5, x_8), (x_4, x_6), (x_4, x_8), (x_1, x_2)\}.$$

Alegem muchia (x_5, x_6). Cum vârfurile terminale aparţin aceluiaşi arbore se trece la muchia următoare (x_4, x_7). Vârfurile terminale ale acestei muchii aparţin la arbori diferiţi. Prin urmare se adaugă la A această muchie. Astfel

$$A = \{(x_3, x_5), (x_5, x_8), (x_4, x_6), (x_4, x_8), (x_1, x_2), (x_4, x_7)\}.$$

Alegem muchia (x_3, x_6). Cum vârfurile terminale ale acestei muchii aparţin aceluiaşi arbore se trece la muchia următoare (x_1, x_3) care va fi adăugată la A. Deci

$$A = \{(x_3, x_5), (x_5, x_8), (x_4, x_6), (x_4, x_8), (x_1, x_2), (x_4, x_7), (x_1, x_3)\}.$$

Alegem muchia (x_2, x_4). Vârfurile terminale ale acestei muchii aparţin aceluiaşi arbore. Trecem la următoarea muchie. Alegem muchia (x_7, x_9). Vârfurile terminale ale acestei muchii aparţin la arbori diferiţi. Se adaugă această muchie la A. Obţinem

$$A = \{(x_3, x_5), (x_5, x_8), (x_4, x_6), (x_4, x_8), (x_1, x_2), (x_4, x_7), (x_1, x_3), (x_7, x_9)\}.$$

Am obţinut arborele de acoperire minim.

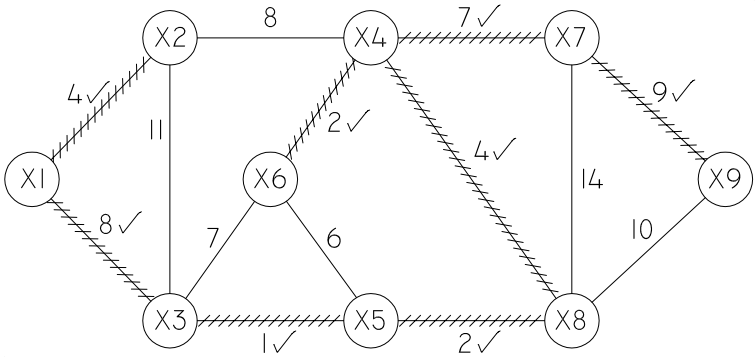


Figura 2.5:

2.7.2 Algoritmul lui Prim

Acest algoritm este asemănător cu algoritmul lui Kruskal. Arborele căutat porneşte dintr-un vârf arbitrar şi creşte până când acoperă toate vâfurile. La fiecare pas se adugă mulţimii A o muchie sigură. In timpul execuţiei algoritmului toate vârfurile care nu sunt în A se află într-o coadă de prioritate Q (iniţial $Q = X$). Algoritmul se încheie când $Q = \emptyset$.
Etapa 0. $A = \emptyset$; $Q = X$.
Etapa 1. $A = \{x_1\}$; $Q = X \setminus \{x_1\}$.
Etapa 2. Se determină $y_0 \in Q$ cu proprietatea că există $x_0 \in A$ astfel încât

$$v(x_0, y_0) = \min_{x \in A, y \in Q} v(x, y) \ .$$

Etapa 3. y_0 se adaugă la A şi se elimină din Q. Dacă $Q = \emptyset$ algoritmul s-a încheiat. In caz contrar se reia etapa 2.

Exemplul 2.7.2 *Aplicând algoritmul lui Prim să se determine arborele de acoperire minim pentru graful din figura 2.5.*

Soluţie. Fie $A = \{x_1\},\ Q = X \setminus \{x_1\}$. Cum

$$\min_{x \in A, y \in Q} v(x, y) = \min\{v(x_1, x_2),\ v(x_1, x_3)\} = v(x_1, x_2) \ ,$$

vom adăuga x_2 la A şi se elimină din Q. Deci $A = \{(x_1, x_2)\}$, $Q = X \setminus \{x_1, x_2\}$. Apoi

$$\min_{x \in A, y \in Q} v(x, y) = \min\{v(x_1, x_3),\ v(x_2, x_3),\ v(x_2, x_4)\} = v(x_1, x_3) \ .$$

Astfel $A = \{(x_1, x_2), (x_1, x_3)\}$ şi $Q = X \setminus \{x_1, x_2, x_3\}$. Observăm că

$$\min_{x \in A, y \in Q} v(x, y) = \min\{v(x_2, x_4),\ v(x_3, x_6),\ v(x_3, x_5)\} = v(x_3, x_5) \ .$$

Prin urmare $A = \{(x_1, x_2), (x_1, x_3), (x_3, x_5)\}$.În final

$$A = \{(x_1, x_2), (x_1, x_3), (x_3, x_5), (x_5, x_8), (x_4, x_8), (x_4, x_6), (x_4, x_7), (x_7, x_9)\} \ .$$

2.8 Algoritmul Ford-Fulkerson

Etapa 0. Fixăm $f(x,y)=0,\ (\forall)x,y\in X$.
Etapa 1. Căutăm un drum rezidual în G_f. Vom marca intrarea reţelei cu $+$. Dacă un vârf x este marcat vom marca cu:

1. $\boxed{+x}$ acele vârfuri $y:\ (x,y)\in U$ şi $c_f(x,y)=c(x,y)-f(x,y)>0$;
2. $\boxed{-x}$ acele vârfuri $y:\ (y,x)\in U$ şi $c_f(x,y)=f(y,x)>0$.

Dacă prin acest procedeu de marcare nu se poate marca ultimul vârf, fluxul obţinut este maxim. În caz contrar se trece la etapa 2.
Etapa 2. Fie μ un drum rezidual găsit. Fie $c_f(\mu)=\min\limits_{(x,y)\in\mu} c_f(x,y)$ capacitatea reziduală a drumului μ.
Fie $f_\mu: X\times X\to\mathbb{R}$

$$f_\mu(x,y)=\begin{cases} c_f(\mu), & \text{dacă } (x,y)\in\mu\cap U\\ -c_f(\mu), & \text{dacă } (x,y)\in\mu \text{ şi } (y,x)\in U\\ 0, & \text{în rest}\end{cases}.$$

Considerăm un nou flux $f'=f+f_\mu$ şi revenim la etapa 1.

Exemplul 2.8.1 *În figura 2.6 este considerată o reţea de transport. Vârful x_0 este intrarea reţelei iar vârful x_{10} este ieşirea reţelei. Capacitatea fiecărui arc este trecută în chenar. Să se determine fluxul maxim aplicând algoritmul Ford-Fulkerson.*

Soluţie. **A)** Presupunem mai întâi că $f(x,y)=0,\ (\forall)x,y\in X$. În această situaţie reţeaua reziduală G_f coincide cu reţeaua iniţială.
B) Căutăm un drum rezidual în G_f. Marcăm intrarea reţelei cu $+$.
Atunci vârful x_1 va putea fi marcat cu $\boxed{+0}$. Plecând de la vârful x_1, vârful x_2 va putea fi marcat cu $\boxed{+1}$. Apoi x_5 va fi marcat cu $\boxed{+2}$, vârful x_8 va fi marcat cu $\boxed{+5}$ şi vârful x_{10} va fi marcat cu $\boxed{+8}$. Obţinem drumul rezidual $\mu_1=(x_0,\ x_1,\ x_2,\ x_5,\ x_8,\ x_{10})$ care are capacitatea reziduală $c_f(\mu_1)=\min\{12,\ 8,\ 14,\ 13,\ 12\}=8$. Prin urmare mărim fluxul cu 8 unităţi pe arcele din care este format acest drum şi îl lăsăm neschimbat în rest.
C) În mod similar găsim drumul rezidual $\mu_2=(x_0,\ x_1,\ x_4,\ x_5,\ x_7,\ x_{10})$ care

are capacitatea reziduală $c_f(\mu_2) = \min\{4,\ 9,\ 8,\ 12,\ 15\} = 4$. Mărim cu 4 unităţi fluxul pe arcele din care este format acest drum.

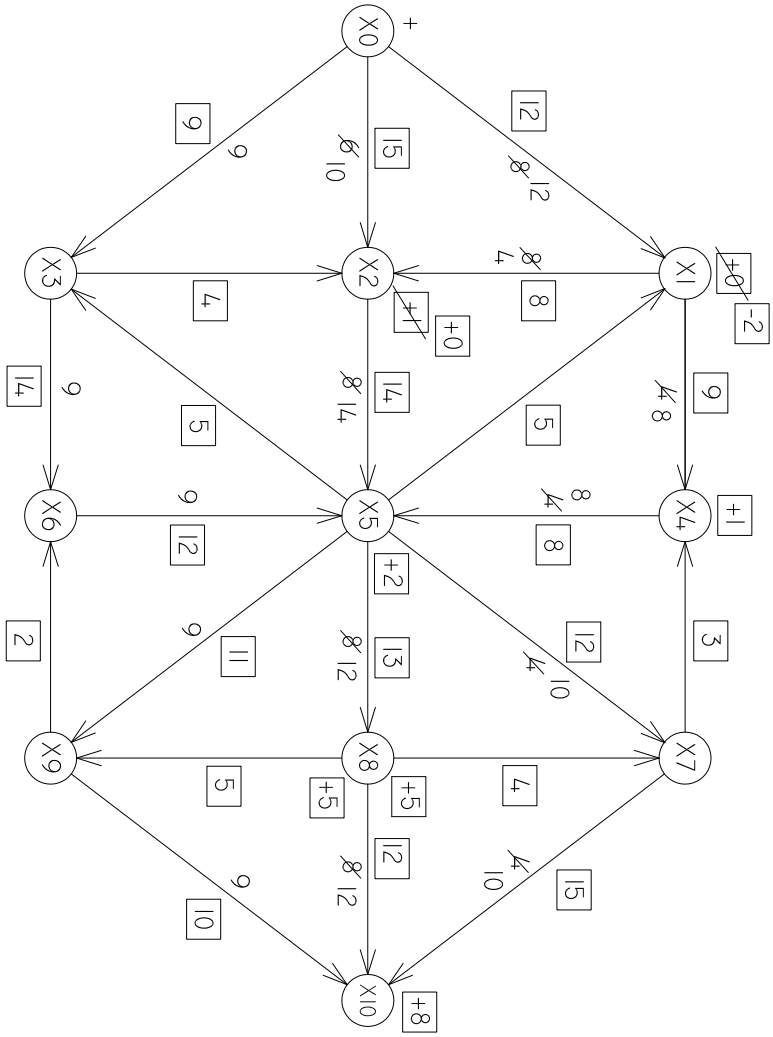


Figura 2.6:

D) Găsim drumul rezidual $\mu_3 = (x_0,\ x_2,\ x_5,\ x_7,\ x_{10})$ care are capacitatea reziduală $c_f(\mu_3) = \min\{15,\ 6,\ 8,\ 11\} = 6$. Mărim cu 6 unităţi fluxul pe arcele din care este format acest drum.
E) Găsim drumul rezidual $\mu_4 = (x_0,\ x_3,\ x_6,\ x_5,\ x_9,\ x_{10})$ având capacitatea reziduală $c_f(\mu_4) = \min\{9,\ 14,\ 12,\ 11,\ 10\} = 9$. Mărim cu 9 unităţi fluxul

pe arcele din care este format acest drum.
F) Observăm acum că dacă intrarea reţelei este marcată cu +, doar vârful x_2 poate fi marcat cu $\boxed{+0}$. Vârfurile x_1 şi x_3 nu pot fi marcate cu $\boxed{+0}$ deoarece arcele (x_0, x_1) şi (x_0, x_3) sunt saturate.
Plecând de la vârful x_2, vârful x_5 nu mai poate fi marcat cu $\boxed{+2}$ deoarece arcul (x_2, x_5) este saturat. În schimb însă vârful x_1 poate fi marcat cu $\boxed{\text{-}2}$ deoarece fluxul pe arcul (x_1, x_2) este strict pozitiv. Vârful x_3 nu poate fi marcat $\boxed{\text{-}2}$ pentru că nu avem flux pe arcul (x_3, x_2).
Plecând de la x_1 vom putea marca x_4, apoi x_5, x_8 şi x_{10}. Obţinem drumul rezidual $\mu_5 = (x_0,\ x_2,\ x_1,\ x_4,\ x_5,\ x_8,\ x_{10})$ care are capacitatea reziduală $c_f(\mu_5) = \min\{9,\ 8,\ 5,\ 4,\ 5,\ 4\} = 4$. Mărim cu 4 unităţi fluxul pe arcele (x_0, x_2), (x_1, x_4), (x_4, x_5), (x_5, x_8), (x_8, x_{10}) şi scădem cu 4 unităţi fluxul pe arcul (x_1, x_2).
G) Reluăm procedeul de marcare. Marcăm intrarea reţelei cu +. Plecând de la x_0 doar x_2 va putea fi marcat cu $\boxed{+0}$. Plecând de la x_2 doar x_1 poate fi marcat cu $\boxed{\text{-}2}$.
Plecând de la x_1 doar x_4 poate fi marcat cu +1 căci arcul (x_1, x_4) nu este saturat. Vârful x_5 nu poate fi marcat plecând de la x_1 căci nu avem flux pe arcul (x_5, x_1). Plecând de la x_4 nu mai putem marca niciun vârf. Vârful x_7 nu poate fi marcat $\boxed{\text{-}4}$ căci nu avem flux pe arcul (x_7, x_4) iar vârful x_5 nu poate fi marcat $\boxed{+4}$ căci arcul (x_4, x_5) este saturat.
Cum prin acest procedeu de marcare nu se poate marca ieşirea reţelei, fluxul găsit este maxim având valoarea $\overline{f} = 12 + 10 + 9 = 31$.
H) În final precizăm că puteam alege şi alte drumuri reziduale care conduceau la alt flux maxim, bineînţeles tot de valoare 31.

2.9 Probleme de afectare

2.9.1 Algoritmul lui Little

Fie $T = (t_{ij})_{i,j=1}^{n}$ matricea timpilor.
Etapa 1. Se determină matricea redusă T_R efectuând operaţiile:
1.1. Se scade din fiecare linie cel mai mic element. Mai precis, pentru $i = \overline{1,n}$, se scade din elementele liniei "i" elementul

$$t_{i\alpha} = \min_{j=\overline{1,n}}\{t_{ij}\};$$

1.2. Se scade din fiecare coloană cel mai mic element. Mai precis, pentru $j = \overline{1,n}$, se scade din elementele coloanei "j" elementul

$$t_{\beta j} = \min_{i=\overline{1,n}} \{t_{ij} - t_{i\alpha}\};$$

Etapa 2. Se determină **constanta de reducere**

$$h = \sum_{i=1}^{n} t_{i\alpha} + \sum_{j=1}^{n} t_{\beta j} \ .$$

Etapa 3. Vom construi un arbore. Fiecărui nod x îi asociem o margine $\omega(x)$. Nodul iniţial este E (mulţimea permutărilor definite pe $\{1, 2, \ldots, n\}$) şi vom pune $\omega(E) = h$.
Etapa 4.
4.1. Pentru fiecare element $t_{ij} = 0$ din T_R se calculează

$$\theta_{ij} = \min_{q \neq j} t_{iq} + \min_{p \neq i} t_{pj} \ .$$

4.2. Se determină

$$\theta_{kl} = \max\{\theta_{ij}\} \ .$$

4.3. Se asociază lui E două noduri: E_{kl} şi $\overline{E}_{kl}$. Vom pune

$$\omega(\overline{E}_{kl}) = \omega(E) + \theta_{kl} \ .$$

4.4. Se construieşte matricea $T(E_{kl})$ obţinută din T_R prin suprimarea liniei k şi a coloanei l. Se aplică acestei matrici etapele 1 şi 2 şi se determină $T_R(E_{kl})$ şi constanta de reducere h_{kl}. Vom pune

$$\omega(E_{kl}) = \omega(E) + h_{kl} \ .$$

Etapa 5. Dacă s-a obţinut o matrice de tipul $(1, 1)$ algoritmul s-a încheiat. În caz contrar ramificarea se continuă din acel nod cu marginea cea mai mică. În cazul mai multor noduri se alege un nod de tip E_{kl}.
Dacă s-a ales un nod de tip E_{kl} se trece la etapa 4.
Dacă s-a ales un nod de tip $\overline{E}_{kl}$ atunci pentru a evita afectarea (k, l) se pune în matricea nodului precedent $t_{kl} = \infty$ şi se trece la etapa 4.

Exemplul 2.9.1 *Să se rezolve problema de afectare (minim) în care matricea timpilor este:*

$T =$

	1	2	3	4	5	6	7
1	10	1	3	1	5	1	1
2	∞	2	5	3	0	∞	8
3	11	8	∞	∞	5	9	5
4	5	11	3	3	6	9	∞
5	∞	7	4	9	∞	4	4
6	11	8	7	∞	∞	3	2
7	9	∞	12	9	5	10	8

Soluţie. Precizăm mai întâi că anumite elemente ale matricei T sunt egale cu ∞, ceea ce înseamnă că pe o anumită maşină nu poate fi executată o anumită lucrare. De exemplu $t_{21} = \infty$ ne spune că maşina M_2 nu poate executa lucrarea L_1.
Scădem din fiecare linie cel mai mic element. Acestea sunt în ordine $1, 0, 5, 3, 4, 2, 5$. Obţinem

$T =$

	1	2	3	4	5	6	7
1	9	0	2	0	4	0	0
2	∞	2	5	3	0	∞	8
3	6	3	∞	∞	0	4	0
4	2	8	0	0	3	6	∞
5	∞	3	0	5	∞	0	0
6	9	6	5	∞	∞	1	0
7	4	∞	7	4	0	5	4

Scădem din fiecare coloană cel mai mic element. Prin urmare din coloana întâi va trebui să scădem 2

$T_R =$

	1	2	3	4	5	6	7
1	7	0	2	0	4	0	0
2	∞	2	5	3	0	∞	8
3	4	3	∞	∞	0	4	0
4	0	8	0	0	3	6	∞
5	∞	3	0	5	∞	0	0
6	7	6	5	∞	∞	1	0
7	2	∞	7	4	0	5	4

Constanta de reducere este

$$h = 1 + 5 + 3 + 4 + 2 + 5 + 2 = 22 \ .$$

Punem $\omega(E) = h = 22$. Pentru $t_{ij} = 0$ vom calcula θ_{ij}. Obţinem $\theta_{12} = 0,\ \theta_{14} = 0,\ \theta_{16} = 0,\ \theta_{17} = 0,\ \theta_{25} = 2,\ \theta_{35} = 0,\ \theta_{37} = 0,\ \theta_{41} = 2,\ \theta_{43} = 0,\ \theta_{44} = 0,\ \theta_{53} = 0,\ \theta_{56} = 0,\ \theta_{57} = 0,\ \theta_{67} = 1,\ \theta_{75} = 2.$ Alegem $\theta_{12} = \max\{\theta_{ij}\}$ şi prin urmare avem afectarea $(1, 2)$ şi

$$\omega(\overline{E}_{12}) = \omega(E) + \theta_{12} = 22 + 2 = 24 \ .$$

Tăiem linia 1 şi coloana 2 şi obţinem

$T(E_{12}) =$

	1	3	4	5	6	7
2	∞	5	3	0	∞	8
3	4	∞	∞	0	4	0
4	0	0	0	3	6	∞
5	∞	0	5	∞	0	0
6	7	5	∞	∞	1	0
7	2	7	4	0	5	4

Scădem din fiecare linie cel mai mic element şi apoi din fiecare coloană cel mai mic element. Observăm că $T_R(E_{12}) = T(E_{12})$ şi $h_{12} = 0$. Deci $\omega(E_{12}) = \omega(E) + h_{12} = 22 + 0 = 22$. Continuăm ramificarea din nodul E_{12}. $\theta_{25} = 3,\ \theta_{35} = 0,\ \theta_{37} = 0,\ \theta_{41} = 2,\ \theta_{42} = 0,\ \theta_{43} = 3,\ \theta_{53} = 0,\ \theta_{56} = 1,\ \theta_{57} = 0,\ \theta_{67} = 1,\ \theta_{75} = 2.$ Fie $\theta_{25} = \max\{\theta_{ij}\}$. Alegem afectarea $(2, 5)$.
Avem $\omega(\overline{E}_{25}) = \omega(E_{12}) + \theta_{25} = 22 + 3 = 25$. Tăiem linia 2 şi coloana 5. Obţinem:

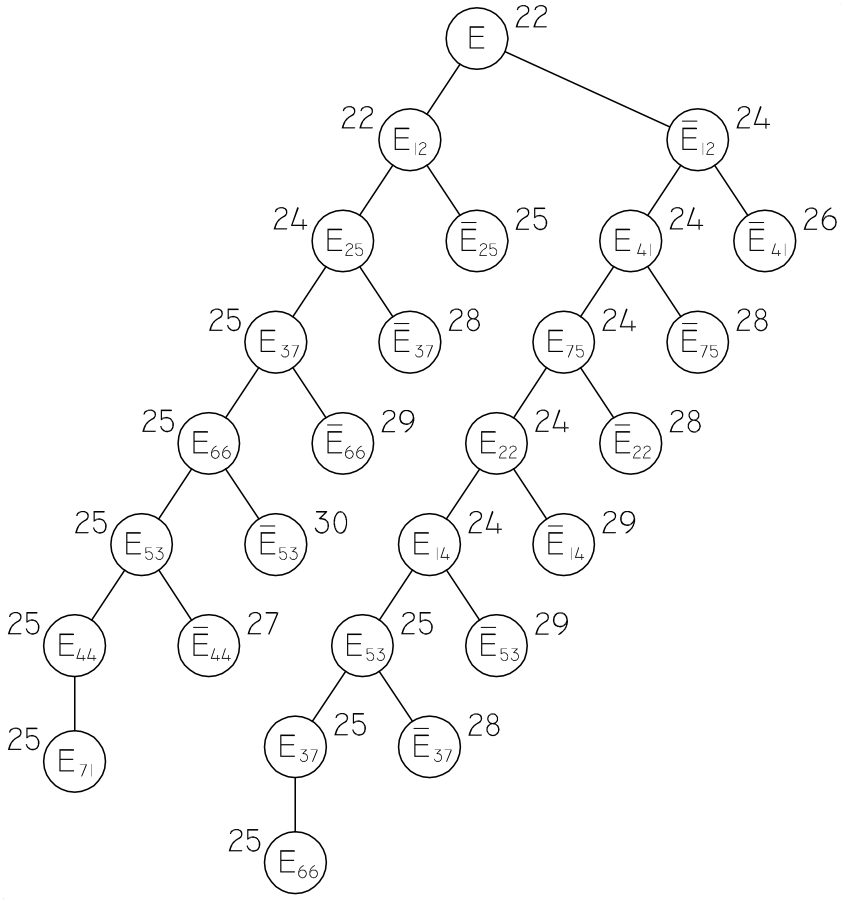
$T(E_{25}) =$

	1	3	4	6	7
3	4	∞	∞	4	0
4	0	0	0	6	∞
5	∞	0	5	0	0
6	7	5	∞	1	0
7	2	7	4	5	4

Scădem 2 din ultima linie. Obţinem $h_{25} = 2$ şi

$$T_R(E_{25}) = \begin{array}{c|c|c|c|c|c} & 1 & 3 & 4 & 6 & 7 \\ \hline 3 & 4 & \infty & \infty & 4 & 0 \\ \hline 4 & 0 & 0 & 0 & 6 & \infty \\ \hline 5 & \infty & 0 & 5 & 0 & 0 \\ \hline 6 & 7 & 5 & \infty & 1 & 0 \\ \hline 7 & 0 & 5 & 2 & 3 & 2 \end{array}$$

$\omega(E_{25}) = \omega(E_{12}) + h_{25} = 22 + 2 = 24$. Continuăm ramificarea din E_{25}.



$\theta_{37} = 4$, $\theta_{41} = 0$, $\theta_{43} = 0$, $\theta_{44} = 2$, $\theta_{53} = 0$, $\theta_{56} = 1$, $\theta_{57} = 0$, $\theta_{67} = 1$, $\theta_{71} = 2$. Observăm că $\theta_{37} = \max\{\theta_{ij}\}$. Alegem afectarea $(3, 7)$.

Avem $\omega(\overline{E}_{37}) = \omega(E_{25}) + \theta_{37} = 24 + 4 = 28$. Tăiem linia 3 şi coloana 7. Obţinem:

$T(E_{37}) =$

	1	3	4	6
4	0	0	0	6
5	∞	0	5	0
6	7	5	∞	1
7	0	5	2	3

Scădem 1 din linia a 6-a. Obţinem $h_{37} = 1$. Deci

$$\omega(E_{37}) = \omega(E_{25}) + h_{37} = 24 + 1 = 25 .$$

Prin urmare trebuie să continuăm din nodul $\overline{E}_{12}$. Atunci pentru a nu alege afectarea $(1, 2)$ revenim la matricea T_R şi elementul $t_{12} = 0$ îl punem $t_{12} = \infty$. Obţinem şi matricea redusă scăzând 2 din coloana a doua.

$T_R =$

	1	2	3	4	5	6	7
1	7	∞	2	0	4	0	0
2	∞	0	5	3	0	∞	8
3	4	1	∞	∞	0	4	0
4	0	6	0	0	3	6	∞
5	∞	1	0	5	∞	0	0
6	7	4	5	∞	∞	1	0
7	2	∞	7	4	0	5	4

$\theta_{14} = 0,\ \theta_{16} = 0,\ \theta_{17} = 0,\ \theta_{22} = 1,\ \theta_{25} = 0,\ \theta_{35} = 0,\ \theta_{37} = 0,\ \theta_{41} = 2,\ \theta_{43} = 0,\ \theta_{44} = 0,\ \theta_{53} = 0,\ \theta_{56} = 0,\ \theta_{57} = 0,\ \theta_{67} = 1,\ \theta_{75} = 2.$ Aleg $\theta_{41} = \max\{\theta_{ij}\}$. Avem afectarea $(4, 1)$.

$$\omega(\overline{E}_{41}) = \omega(\overline{E}_{12}) + \theta_{41} = 24 + 2 = 26 .$$

Tăiem linia 4 şi coloana 1.

$T(E_{41}) =$

	2	3	4	5	6	7
1	∞	2	0	4	0	0
2	0	5	3	0	∞	8
3	1	∞	∞	0	4	0
5	1	0	5	∞	0	0
6	4	5	∞	∞	1	0
7	∞	7	4	0	5	4

Observăm că $T_R(E_{41}) = T(E_{41})$ şi $h_{41} = 0$. Deci

$$\omega(E_{41}) = \omega(\overline{E}_{12}) + h_{41} = 24 + 0 = 24 .$$

Continuăm ramificarea din nodul E_{41}.
$\theta_{14} = 3,\ \theta_{16} = 0,\ \theta_{17} = 0,\ \theta_{21} = 1,\ \theta_{25} = 0,\ \theta_{35} = 0,\ \theta_{37} = 0,\ \theta_{53} = 2,$
$\theta_{56} = 0,\ \theta_{57} = 0,\ \theta_{67} = 1,\ \theta_{75} = 4.$
Observăm că $\theta_{75} = \max\{\theta_{ij}\}$. Alegem afectarea $(7, 5)$.

$$\omega(\overline{E}_{75}) = \omega(E_{41}) + \theta_{75} = 24 + 4 = 28 .$$

Tăiem linia 7 şi coloana 5. Obţinem

$T(E_{75}) =$

	2	3	4	6	7
1	∞	2	0	0	0
2	0	5	3	∞	8
3	1	∞	∞	4	0
5	1	0	5	0	0
6	4	5	∞	1	0

Observăm că $T_R(E_{75}) = T(E_{75})$ şi $h_{75} = 0$. Deci

$$\omega(E_{75}) = \omega(E_{41}) + h_{75} = 24 + 0 = 24 .$$

Continuăm ramificarea din E_{75}. Avem $\theta_{14} = 3,\ \theta_{16} = 0,\ \theta_{17} = 0,$ $\theta_{22} = 4,\ \theta_{37} = 1,\ \theta_{53} = 2,\ \theta_{56} = 0,\ \theta_{57} = 0,\ \theta_{67} = 1$. Observăm că $\theta_{22} = \max\{\theta_{ij}\}$. Alegem afectarea $(2, 2)$. Avem

$$\omega(\overline{E}_{22}) = \omega(E_{75}) + \theta_{22} = 24 + 4 = 28 .$$

Tăiem linia 2 şi coloana 2. Obţinem

$T(E_{22}) =$

	3	4	6	7
1	2	0	0	0
3	∞	∞	4	0
5	0	5	0	0
6	5	∞	1	0

Observăm că $T_R(E_{22}) = T(E_{22})$ şi $h_{22} = 0$. Deci

$$\omega(E_{22}) = \omega(E_{75}) + h_{22} = 24 + 0 = 24 .$$

Continuăm ramificarea din E_{22}. Avem $\theta_{14} = 5,\ \theta_{16} = 0,\ \theta_{17} = 0,\ \theta_{37} = 4,\ \theta_{53} = 2,\ \theta_{56} = 0,\ \theta_{57} = 0,\ \theta_{67} = 1$. Observăm că $\theta_{14} = \max\{\theta_{ij}\}$. Alegem afectarea $(1, 4)$. Avem

$$\omega(\overline{E}_{14}) = \omega(E_{22}) + \theta_{14} = 24 + 5 = 29 .$$

Tăiem linia 1 şi coloana 4. Obţinem

$$T(E_{14}) = \begin{array}{c||c|c|} & 3 & 6 & 7 \\ \hline\hline 3 & \infty & 4 & 0 \\ \hline 5 & 0 & 0 & 0 \\ \hline 6 & 5 & 1 & 0 \\ \hline \end{array}$$

Observăm că $T_R(E_{14}) = T(E_{14})$ şi $h_{14} = 0$. Deci

$$\omega(E_{14}) = \omega(E_{22}) + h_{14} = 24 + 0 = 24 .$$

Continuăm ramificarea din E_{14}. Avem $\theta_{37} = 4,\ \theta_{53} = 5,\ \theta_{56} = 1,\ \theta_{57} = 0,\ \theta_{67} = 1$. Observăm că $\theta_{53} = \max\{\theta_{ij}\}$. Alegem afectarea $(5, 3)$. Avem

$$\omega(\overline{E}_{53}) = \omega(E_{14}) + \theta_{53} = 24 + 5 = 29 .$$

Tăiem linia 5 şi coloana 3. Obţinem

$$T(E_{53}) = \begin{array}{c||c|c|} & 6 & 7 \\ \hline\hline 3 & 4 & 0 \\ \hline 6 & 1 & 0 \\ \hline \end{array}$$

Scădem 1 din prima coloană. Prin urmare $h_{53} = 1$ şi

$$\omega(E_{53}) = \omega(E_{14}) + h_{53} = 24 + 1 = 25 .$$

Observăm că avem două soluţii căci vom putea continua şi din E_{53} şi din E_{37}. Continuăm mai întâi ramificarea din E_{53}.

$$T_R(E_{53}) = \begin{array}{c||c|c|} & 6 & 7 \\ \hline\hline 3 & 3 & 0 \\ \hline 6 & 0 & 0 \\ \hline \end{array}$$

Avem $\theta_{37} = 3,\ \theta_{66} = 3,\ \theta_{67} = 0$. Aleg $\theta_{37} = \max\{\theta_{ij}\}$. Prin urmare avem afectarea $(3, 7)$ şi

$$\omega(\overline{E}_{37}) = \omega(E_{53}) + \theta_{37} = 25 + 3 = 28 .$$

Tăiem linia 3 şi coloana 7. Obţinem

$$T(E_{37}) = \begin{array}{c|c} & 6 \\ \hline 6 & 0 \end{array}$$

Observăm că $h_{37} = 0$. Deci $\omega(E_{37}) = \omega(E_{53}) + h_{37} = 25 + 0 = 25$. Am ajuns la o matrice $(1, 1)$. Stabilim afectarea $(6, 6)$ şi algoritmul s-a încheiat. Am găsit soluţia $\{(4, 1), (7, 5), (2, 2), (1, 4), (5, 3), (3, 7), (6, 6)\}$.
Revenim acum la nodul E_{37}.

$$T_R(E_{37}) = \begin{array}{c|cccc} & 1 & 3 & 4 & 6 \\ \hline 4 & 0 & 0 & 0 & 6 \\ 5 & \infty & 0 & 5 & 0 \\ 6 & 6 & 4 & \infty & 0 \\ 7 & 0 & 5 & 2 & 3 \end{array}$$

Avem $\theta_{41} = 0,\ \theta_{43} = 0,\ \theta_{44} = 2,\ \theta_{53} = 0,\ \theta_{56} = 0,\ \theta_{66} = 4,\ \theta_{71} = 2$. Observăm că $\theta_{66} = \max\{\theta_{ij}\}$. Avem afectarea $(6, 6)$ şi

$$\omega(\overline{E}_{66}) = \omega(E_{37}) + \theta_{66} = 25 + 4 = 29\ .$$

Tăiem linia 6 şi coloana 6. Obţinem

$$T(E_{66}) = \begin{array}{c|ccc} & 1 & 3 & 4 \\ \hline 4 & 0 & 0 & 0 \\ 5 & \infty & 0 & 5 \\ 7 & 0 & 5 & 2 \end{array}$$

Observăm că $T_R(E_{66}) = T(E_{66})$ şi $h_{66} = 0$. Deci

$$\omega(E_{66}) = \omega(E_{37}) + h_{66} = 25 + 0 = 25\ .$$

Astfel $\theta_{41} = 0,\ \theta_{43} = 0,\ \theta_{44} = 2,\ \theta_{53} = 5,\ \theta_{71} = 2$. Avem că $\theta_{53} = \max\{\theta_{ij}\}$. Aleg afectarea $(5, 3)$.

$$\omega(\overline{E}_{53}) = \omega(E_{66}) + \theta_{53} = 25 + 5 = 30\ .$$

Tăiem linia 5 şi coloana 3. Obţinem

$$T(E_{53}) = \begin{array}{c|cc} & 1 & 4 \\ \hline 4 & 0 & 0 \\ 7 & 0 & 2 \end{array}$$

Avem $T_R(E_{53}) = T(E_{53})$ şi $h_{53} = 0$. Deci

$$\omega(E_{53}) = \omega(E_{66}) + h_{53} = 25 + 0 = 25 .$$

Continuăm ramificarea din E_{53}. Avem $\theta_{41} = 0$, $\theta_{44} = 2$, $\theta_{71} = 2$. Aleg $\theta_{44} = \max\{\theta_{ij}\}$. Avem afectarea $(4, 4)$ şi

$$\omega(\overline{E}_{44}) = \omega(E_{53}) + \theta_{44} = 25 + 2 = 27 .$$

Tăiem linia 4 şi coloana 4. Obţinem

$$T(E_{44}) = \begin{array}{c|c} & 1 \\ \hline 7 & 0 \end{array}$$

Prin urmare $h_{44} = 0$. Deci $\omega(E_{44}) = \omega(E_{53}) + h_{44} = 25 + 0 = 25$. Am ajuns la o matrice $(1, 1)$. Stabilim afectarea $(7, 1)$ şi algoritmul s-a încheiat. Am găsit soluţia $\{(1, 2), (2, 5), (3, 7), (6, 6), (5, 3), (4, 4), (7, 1)\}$.

Observaţia 2.9.2 *Algoritmul se poate aplica şi pentru probleme de maxim. Mai întâi vom trece la minim scăzând elementele fiecărei coloane din maximul lor.*

Exemplul 2.9.3 *Să se rezolve problema de afectare (maxim) în care matricea timpilor este:*

$T =$

	1	*2*	*3*	*4*	*5*
1	*16*	*50*	*20*	*25*	*35*
2	*26*	*20*	*35*	*50*	*5*
3	*30*	*50*	*5*	*0*	*25*
4	*16*	*10*	*45*	*10*	*35*
5	*6*	*35*	*30*	*50*	*5*

Soluţie. Maximul fiecărei coloane este: $30, 50, 45, 50, 35$. Scădem elementele fiecărei coloane din maximul lor. Obţinem:

$T =$

	1	2	3	4	5
1	14	0	25	25	0
2	4	30	10	0	30
3	0	0	40	50	10
4	14	40	0	40	0
5	24	15	15	0	30

Rezolvăm acum problema de minim. Observăm că $T_R = T$ şi constanta de reducere este $h = 0$.
$\theta_{12} = 0,\ \theta_{15} = 0,\ \theta_{24} = 4,\ \theta_{31} = 4,\ \theta_{32} = 0,\ \theta_{43} = 10,\ \theta_{45} = 0,\ \theta_{54} = 15.$
Observăm că $\theta_{54} = \max\{\theta_{15}\}$. Prin urmare alegem afectarea $(5, 4)$. Avem

$$\omega(\overline{E}_{54}) = \omega(E) + \theta_{54} = 0 + 15 = 15\ .$$

Tăiem linia 5 şi coloana 4. Obţinem

$T(E_{54}) =$

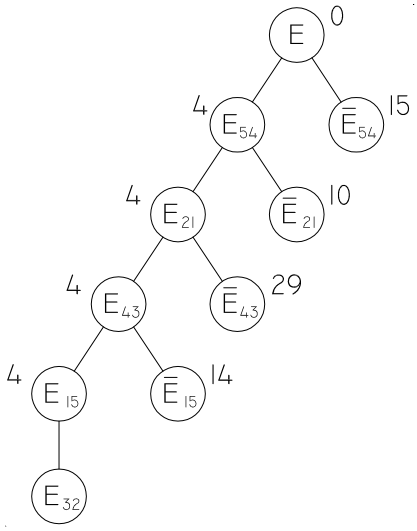
	1	2	3	5
1	14	0	25	0
2	4	30	10	30
3	0	0	40	10
4	14	40	0	0

Scădem 4 din linia a 2-a.

$T(E_{54}) =$

	1	2	3	5
1	14	0	25	0
2	0	26	6	26
3	0	0	40	10
4	14	40	0	0

Prin urmare $h_{54} = 4$ şi $\omega(E_{54}) = \omega(E) + h_{54} = 0 + 4 = 4$.



Continuăm ramificarea din nodul E_{54}. Avem $\theta_{12} = 0,\ \theta_{15} = 0,\ \theta_{21} = 6,\ \theta_{31} = 0,\ \theta_{32} = 0,\ \theta_{43} = 6,\ \theta_{45} = 0$. Alegem $\theta_{21} = \max\{\theta_{ij}\}$. Prin urmare avem afectarea $(2, 1)$ şi

$$\omega(\overline{E}_{21}) = \omega(E_{54}) + \theta_{21} = 4 + 6 = 10 .$$

Tăiem linia 2 şi coloana 1. Obţinem

$$T(E_{21}) = \begin{array}{c|c|c|c} & 2 & 3 & 5 \\ \hline 1 & 0 & 25 & 0 \\ \hline 3 & 0 & 40 & 10 \\ \hline 4 & 40 & 0 & 0 \\ \hline \end{array}$$

Observăm că $T_R(E_{21}) = T(E_{21})$ şi $h_{21} = 0$. Deci

$$\omega(E_{21}) = \omega(E_{54}) + h_{21} = 4 + 0 = 4 .$$

Continuăm ramificarea din nodul E_{21}. Avem $\theta_{12} = 0,\ \theta_{15} = 0,\ \theta_{32} = 10,\ \theta_{43} = 25,\ \theta_{45} = 0$. Observăm că $\theta_{43} = \max\{\theta_{ij}\}$. Prin urmare alegem afectarea $(4, 3)$. Avem

$$\omega(\overline{E}_{43}) = \omega(E_{21}) + \theta_{43} = 4 + 25 = 29 .$$

Tăiem linia 4 şi coloana 3.

$$T(E_{43}) = \begin{array}{c|c|c} & 2 & 5 \\ \hline 1 & 0 & 0 \\ \hline 3 & 0 & 10 \\ \hline \end{array}$$

Observăm că $T_R(E_{43}) = T(E_{43})$ şi $h_{43} = 0$. Deci

$$\omega(E_{43}) = \omega(E_{21}) + h_{43} = 4 + 0 = 4 .$$

Continuăm ramificarea din nodul E_{43}. Avem $\theta_{12} = 0,\ \theta_{15} = 10,\ \theta_{32} = 10$. Alegem $\theta_{15} = \max\{\theta_{ij}\}$. Astfel avem afectarea $(1, 5)$ şi

$$\omega(\overline{E}_{15}) = \omega(E_{43}) + \theta_{15} = 4 + 10 = 14 .$$

Tăiem linia 1 şi coloana 5. Obţinem

$$T(E_{15}) = \begin{array}{c|c} & 2 \\ \hline 3 & 0 \\ \hline \end{array}$$

Observăm că $T_R(E_{15}) = T(E_{15})$ şi $h_{15} = 0$. Deci

$$\omega(E_{15}) = \omega(E_{43}) + h_{15} = 4 + 0 = 4 \ .$$

Am ajuns la o matrice 1×1. Avem afectarea $(3, 2)$ şi algoritmul s-a încheiat. Soluţia optimă este: $\{(5, 4), (2, 1), (4, 3), (1, 5), (3, 2)\}$.

Observaţia 2.9.4 *Algoritmul se aplică şi pentru matrici de tipul (m, n) cu $m < n$ sau $n < m$ adăugând $n - m$ linii sau $m - n$ coloane cu elemente*

$$\max\{t_{ij}\} + 1.$$

Exemplul 2.9.5 *Avem 4 depozite D_1, D_2, D_3, D_4 de la care se pot aproviziona beneficiarii B_1, B_2, B_3, Costurile de transport sunt date în următorul tabel*

	D_1	D_2	D_3	D_4
B_1	*3*	*1*	*5*	*7*
B_2	*1*	*4*	*2*	*6*
B_3	*2*	*3*	*3*	*1*

Să se determine afectarea optimă astfel încât costul total de transport să fie minim.

Soluţie. Adăugăm încă un beneficiar fictiv B_4. Elementele de pe linia lui vor fi $\max\{t_{ij}\} + 1 = 7 + 1 = 8$.

$T =$

	1	2	3	4
1	3	1	5	7
2	1	4	2	6
3	2	3	3	1
4	8	8	8	8

Avem $h = 11$ şi

$T_R =$

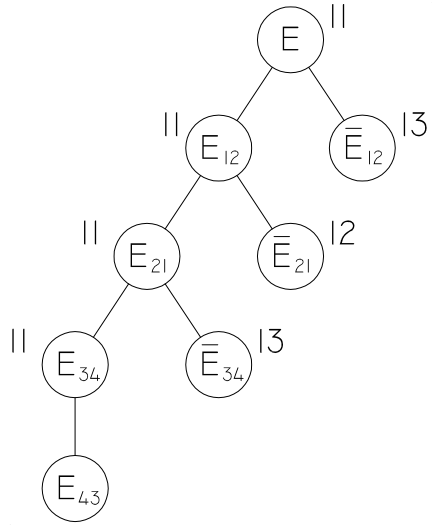
	1	2	3	4
1	2	0	4	6
2	0	3	1	5
3	1	2	2	0
4	0	0	0	0

Avem $\theta_{12} = 2,\ \theta_{21} = 1,\ \theta_{34} = 1,\ \theta_{41} = 0,\ \theta_{42} = 0,\ \theta_{43} = 1,\ \theta_{44} = 0$. Observăm că $\theta_{12} = \max\{\theta_{ij}\}$. Alegem afectarea $(1, 2)$. Avem

$$\omega(\overline{E}_{12}) = \omega(E) + \theta_{12} = 11 + 2 = 13 .$$

Tăiem linia 1 şi coloana 2. Obţinem

$$T(E_{12}) = \begin{array}{c|c|c|c} & 1 & 3 & 4 \\ \hline 2 & 0 & 1 & 5 \\ \hline 3 & 1 & 2 & 0 \\ \hline 4 & 0 & 0 & 0 \end{array}$$



Observăm că $h_{12} = 0$. Deci $\omega(E_{12}) = \omega(E) + h_{12} = 11 + 0 = 11$. Apoi $\theta_{21} = 1,\ \theta_{34} = 1,\ \theta_{41} = 0,\ \theta_{43} = 1,\ \theta_{44} = 0$. Alegem $\theta_{21} = \max\{\theta_{ij}\}$. Rezultă afectarea $(2, 1)$ şi

$$\omega(\overline{E}_{21}) = \omega(E_{12}) + \theta_{21} = 11 + 1 = 12 .$$

Tăiem linia 2 şi coloana 1. Obţinem

$$T_R(E_{21}) = \begin{array}{c|c|c} & 3 & 4 \\ \hline 3 & 2 & 0 \\ \hline 4 & 0 & 0 \end{array}$$

Observăm că $\theta_{34} = 2,\ \theta_{43} = 2,\ \theta_{44} = 0$. Alegem $\theta_{34} = \max\{\theta_{ij}\}$. Deci avem afectarea $(3, 4)$.

$$\omega(\overline{E}_{34}) = \omega(E_{21}) + \theta_{34} = 11 + 2 = 13\ .$$

Tăiem linia 3 şi coloana 4. Obţinem

$$T_R(E_{34}) = \begin{array}{c|c} & 3 \\ \hline 4 & 0 \end{array}$$

Deci $h_{34} = 0$ şi $\omega(E_{34}) = \omega(E_{21}) + h_{34} = 11 + 0 = 11$. Am ajuns la o matrice 1×1. Alegem afectarea $(4, 3)$ şi algoritmul s-a încheiat.

Observaţia 2.9.6 *Algoritmul lui Little poate fi folosit pentru determinarea circuitelor hamiltoniene de valoare optimă. Dacă se doreşte aflarea circuitelor hamiltoniene de valoare minimă se scrie matricea timpilor $T = (t_{ij})$ punând*

$$t_{ij} = \begin{cases} v_{ij}, & \text{dacă } (x_i, x_j) \in U \text{ şi } i \neq j \\ \infty, & \text{în rest} \end{cases}\ .$$

Dacă s-a stabilit afectarea (i, j) în iteraţia următoare se pune $t_{ji} = \infty$ pentru a evita circuitul (x_i, x_j, x_i).

Exemplul 2.9.7 *Să se determine circuitul hamiltonian de valoare minimă în următorul graf valorizat:*

ij	12	13	14	21	23	24	31	32	34	41	42	43
v_{ij}	9	10	4	6	1	6	7	4	1	5	4	6

Soluţie. Avem matricea

$$T = \begin{array}{c|cccc|c} & 1 & 2 & 3 & 4 & \\ \hline 1 & \infty & 9 & 10 & 4 & 4 \\ 2 & 6 & \infty & 1 & 6 & 1 \\ 3 & 7 & 4 & \infty & 1 & 1 \\ 4 & 5 & 4 & 6 & \infty & 4 \end{array}$$

Din fiecare linie vom scădea cantitatea pe care am trecut-o în marginea dreaptă a tabelului.

$$T = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \infty & 5 & 6 & 0 \\ 2 & 5 & \infty & 0 & 5 \\ 3 & 6 & 3 & \infty & 0 \\ 4 & 1 & 0 & 2 & \infty \end{array}$$

Scădem 1 din prima coloană. Deci $h = 4 + 1 + 1 + 4 + 1 = 11$ şi

$$T_R = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \infty & 5 & 6 & 0 \\ 2 & 4 & \infty & 0 & 5 \\ 3 & 5 & 3 & \infty & 0 \\ 4 & 0 & 0 & 2 & \infty \end{array} .$$

Apoi $\theta_{14} = 5$, $\theta_{23} = 6$, $\theta_{34} = 3$, $\theta_{41} = 4$, $\theta_{42} = 3$. Observăm că $\theta_{23} = \max\{\theta_{ij}\}$. Alegem afectarea $(2, 3)$.

$$\omega(\overline{E}_{23}) = \omega(E) + \theta_{23} = 11 + 6 = 17 .$$

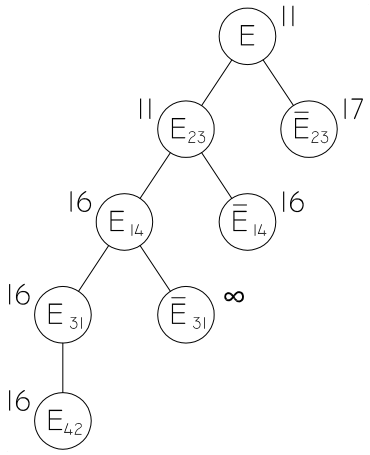
Tăiem linia 2 şi coloana 3. Vom pune $t_{32} = \infty$. Obţinem

$$T(E_{23}) = \begin{array}{c|ccc} & 1 & 2 & 4 \\ \hline 1 & \infty & 5 & 0 \\ 3 & 5 & 3 & 0 \\ 4 & 0 & 0 & \infty \end{array}$$

Observăm că $T_R(E_{23}) = T(E_{23})$ şi $h_{23} = 0$. Deci

$$\omega(E_{23}) = \omega(E) + t_{23} = 11 + 0 = 11 .$$

Continuăm ramificarea din nodul E_{23}.



Observăm că $\theta_{14} = 5,\ \theta_{34} = 5,\ \theta_{41} = 5,\ \theta_{42} = 5$. Alegem $\theta_{14} = \max\{\theta_{ij}\}$. Prin urmare avem afectarea $(1, 4)$ şi

$$\omega(\overline{E}_{14}) = \omega(E_{23}) + \theta_{14} = 11 + 5 = 16 .$$

Tăiem linia 1 şi coloana 4. Vom pune $t_{41} = \infty$.

$$T(E_{14}) = \begin{array}{c|c|c} & 1 & 2 \\ \hline 3 & 5 & \infty \\ \hline 4 & \infty & 0 \end{array}$$

Observăm că $h_{14} = 5$ şi $\omega(E_{14}) = \omega(E_{23}) + t_{14} = 11 + 5 = 16$. Continuăm ramificarea din nodul E_{14}.

$$T_R(E_{14}) = \begin{array}{c|c|c} & 1 & 2 \\ \hline 3 & 0 & \infty \\ \hline 4 & \infty & 0 \end{array} .$$

Observăm că $\theta_{31} = \infty,\ \theta_{42} = \infty$. Alegem $\theta_{31} = \max\{\theta_{ij}\}$. Prin urmare avem afectarea $(3, 1)$ şi

$$\omega(\overline{E}_{31}) = \omega(E_{14}) + \theta_{31} = \infty .$$

Tăiem linia 3 şi coloana 1. Obţinem

$$T(E_{31}) = \begin{array}{c|c} & 2 \\ \hline 4 & 0 \end{array}$$

şi $h_{31} = 0$. Deci $\omega(E_{31}) = 16$. S-a ajuns la o matrice 1×1. Alegem afectarea $(4, 2)$ şi algoritmul s-a încheiat.
Prin urmare afectarea optimă este $\{(2, 3), (1, 4), (3, 1), (4, 2)\}$ şi circuitul hamiltonian de valoare minimă este $(2, 3, 1, 4, 2)$, valoarea acestuia fiind 16.

2.9.2 Algoritmul ungar

Acest algoritm a fost propus de H.W.Kuhn şi a fost denumit de către acesta metoda ungară.
Fie $T = (t_{ij})_{i,j=1}^{n}$ matricea timpilor.
Etapa 0. Se determină matricea redusă T_R ca şi la algoritmul lui Little.

Etapa 1.
1.1. Se încadrează un zero de pe o linie cu cele mai puţine zerouri.
1.2. Se taie celelalte zerouri de pe linia şi coloana zeroului încadrat.
1.3. Repetăm operaţia până când toate zerourile au fost încadrate sau tăiate.
1.4. Dacă se obţine câte un zero încadrat pe fiecare linie şi pe fiecare coloană atunci am obţinut soluţia optimă. În caz contrar se trece la etapa următoare.
Etapa 2.
2.1. Marcăm liniile care nu conţin zerouri încadrate şi coloanele care au zerouri tăiate pe liniile marcate.
2.2. Marcăm liniile cu zero încadrat pe coloanele marcate.
2.3. Tăiem liniile nemarcate şi coloanele marcate.
2.4. Fie θ cel mai mic element netăiat. Adunăm θ la elementle dublu tăite şi îl scădem din elementele netăite lăsând elementele simplu tăiate neschimbate. Cu matricea obţinută se reia etapa 1.

Exemplul 2.9.8 *Să se aplice algoritmul ungar pentru rezolvarea exemplului* 2.9.1.

Soluţie. Aşa cum am văzut în soluţia exemplului 2.9.1. matricea redusă este

$T_R =$

		1	2	3	4	X 5	6	X 7
	1	~~7~~	[~~0~~]	~~2~~	~~0~~	~~4~~	~~0~~	~~0~~
X	2	∞	2	5	3	[0]	∞	8
X	3	4	3	∞	∞	~~0~~	4	[0]
	4	[~~0~~]	~~8~~	~~0~~	~~0~~	~~3~~	~~6~~	~~∞~~
	5	~~∞~~	~~3~~	[~~0~~]	~~5~~	~~∞~~	~~0~~	~~0~~
X	6	7	6	5	∞	∞	1	~~0~~
X	7	2	∞	7	4	~~0~~	5	4

Apoi

		1	2	3	4	X 5	6	7
	1	~~7~~	~~[0]~~	~~2~~	~~0~~	~~5~~	~~0~~	~~1~~
X	2	∞	1	4	2	[0]	∞	8
	3	~~3~~	~~2~~	~~∞~~	~~∞~~	~~0~~	~~3~~	~~[0]~~
	4	~~[0]~~	~~8~~	~~0~~	~~0~~	~~4~~	~~6~~	~~∞~~
	5	~~∞~~	~~3~~	~~[0]~~	~~5~~	~~∞~~	~~0~~	~~1~~
	6	~~6~~	~~5~~	~~4~~	~~∞~~	~~∞~~	~~[0]~~	~~0~~
X	7	1	∞	6	3	~~0~~	4	4

	1	2	3	4	5	6	7
1	7	[0]	2	~~0~~	6	~~0~~	1
2	∞	~~0~~	3	1	[0]	∞	7
3	3	2	∞	∞	1	3	[0]
4	~~0~~	8	~~0~~	[0]	5	6	∞
5	∞	3	[0]	5	∞	~~0~~	1
6	6	5	4	∞	∞	[0]	~~0~~
7	[0]	∞	5	2	~~0~~	3	3

sau

	1	2	3	4	5	6	7
1	7	~~0~~	2	[0]	6	~~0~~	1
2	∞	[0]	3	1	~~0~~	∞	7
3	3	2	∞	∞	1	3	[0]
4	[0]	8	~~0~~	~~0~~	5	6	∞
5	∞	3	[0]	5	∞	~~0~~	1
6	6	5	4	∞	∞	[0]	~~0~~
7	~~0~~	∞	5	2	[0]	3	3

Prin urmare avem afectările optime:
$\{(1,2),(2,5),(3,7),(4,4),(5,3),(6,6),(7,1)\}$ şi
$\{(1,4),(2,2),(3,7),(4,1),(5,3),(6,6),(7,5)\}$.

2.10 Probleme de ordonanţare

Prin proiect înţelegem un ansamblu de activităţi $\{A_i\}_{i=1}^n$, supuse anumitor restricţii în ce priveşte ordinea de execuţie a acestora, a căror realizare permite atingerea unui obiectiv. Trebuie făcută precizarea că fiecare activitate A_i este indivizibilă (nu se poate descompune în subactivităţi), odată începută nu mai poate fi întreruptă şi are o durată de execuţie cunoscută d_i.
O problemă de ordonanţare constă în stabilirea unei succesiuni de efectuare a activităţilor unui proiect, astfel încât relaţiile de precedenţă dintre ele să fie respectate şi timpul total de execuţie a acestuia să fie minim. Relaţia de precedenţă cea mai des întâlnită este aceea în care activitatea A_i precede activitatea A_j (sau altfel spus activitatea A_j succede activitatea A_i) dacă activitatea A_j nu poate să înceapă decât după un interval de timp t_{ij} de la terminarea activităţii A_i. În general $t_{ij} = 0$.

2.10.1 Metoda potenţialelor

Graful activităţilor unui proiect este $G = (X, U)$ unde $X = \{A_i\}_{i=1}^n$ şi $(A_i, A_j) \in U$ dacă activitatea A_i precede activitatea A_j. Acest graf va fi valorizat în cazul în care nu toţi t_{ij} sunt nuli, punând valoarea arcului (A_i, A_j) tocmai valoarea t_{ij}.
În cazul în care există mai multe activităţi care nu sunt condiţionate de nicio activitate a proiectului, vom introduce o activitate iniţială fictivă A_0 cu durata de execuţie $d_0 = 0$ şi care precede aceste activităţi. În mod similar, dacă există mai multe activităţi care nu au nicio activitate succesoare vom introduce o activitate finală fictivă A_{n+1} cu durata de execuţie $d_{n+1} = 0$ fără succesor, dar precedată de aceste activităţi.
Fiecare activitate A_i este caracterizată prin două momente: t_i- momentul de început al activităţii şi $t_i + d_i$- momentul terminării activităţii. Pentru activitatea A_i vom nota cu

t_i^- - momentul cel mai devreme al începerii activităţii
t_i^+ - momentul cel mai devreme al terminării activităţii
T_i^- - momentul cel mai târziu al începerii activităţii
T_i^+ - momentul cel mai târziu al terminării activităţii

Aceste date se pun sub forma

t_i^-	A_i	t_i^+
T_i^-	d_i	T_i^+

Evident $t_i^+ = t_i^- + d_i, T_i^+ = T_i^- + d_i$. Pentru activitatea iniţială se consideră $t_0^- = 0$. Apoi

$$t_j^- = \max_{j|(A_i,A_j)\in U}(t_i^+ + t_{ij}) \; ; \; T_i^+ = \min_{j|(A_i,A_j)\in U}(T_j^- - t_{ij}) \; .$$

Pentru activitatea finală se consideră $T_{n+1}^+ = t_{n+1}^+$. Pentru fiecare activitate se defineşte rezerva de timp a acesteia

$$R_i = T_i^- - t_i^- \; .$$

O activitate A_i pentru care $R_i = 0$ se numeşte activitate critică. Se numeşte drum critic un drum între vârful iniţial şi cel final format din activităţi critice. Momentele timpurii se determină parcurgând graful de la sursă spre destinaţie iar cele târzii se determină parcurgând graful în sens invers, de la destinaţie spre sursă.

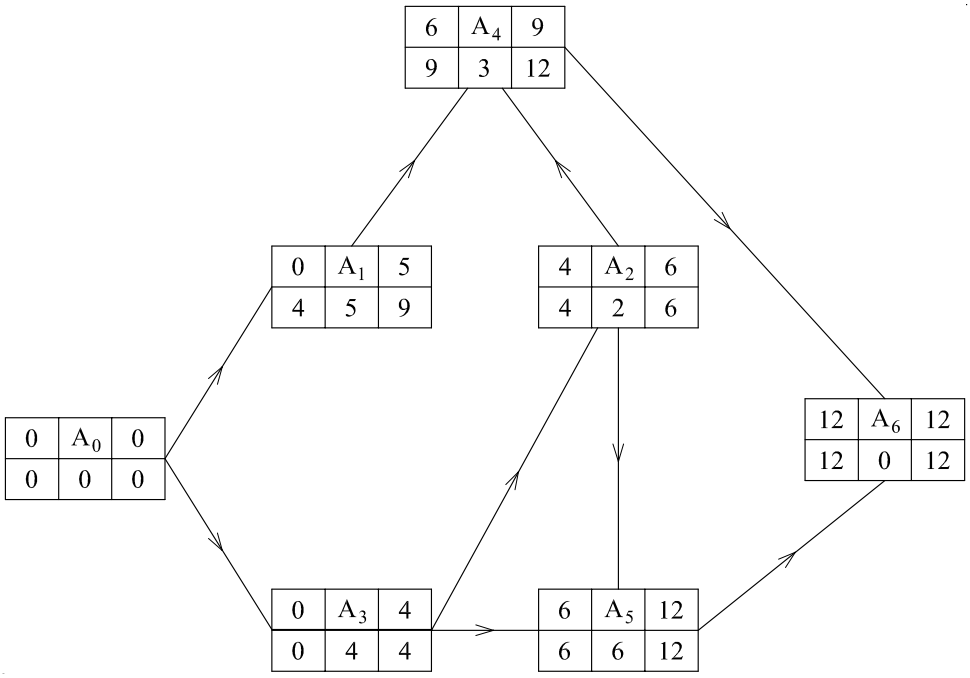
Exemplul 2.10.1 *Să se determine drumul critic pentru proiectul*

Activitatea	*Durata*	*Activităţi precedente*
1	5	–
2	2	3
3	4	–
4	3	1, 2
5	6	2, 3

Soluţie. Cum activităţile A_1 şi A_3 nu au activităţi precedente introducem activitatea iniţială fictivă A_0 care precede aceste activităţi. Cum activităţile A_4 şi A_5 nu au activităţi succesoare introducem activitatea finală fictivă A_6 precedată de aceste activităţi.

Activitatea	Durata	Activităţi precedente
0	0	–
1	5	0
2	2	3
3	4	0
4	3	1, 2
5	6	2, 3
6	0	4, 5

Precizăm că în cazul nostru toţi $t_{ij} = 0$.

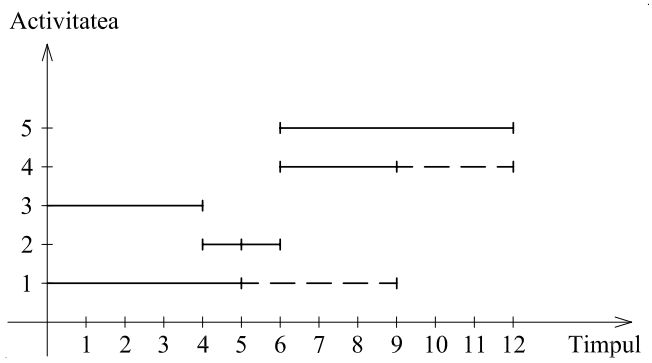


Drumul critic este format de activităţile A_0, A_3, A_2, A_5, A_6.

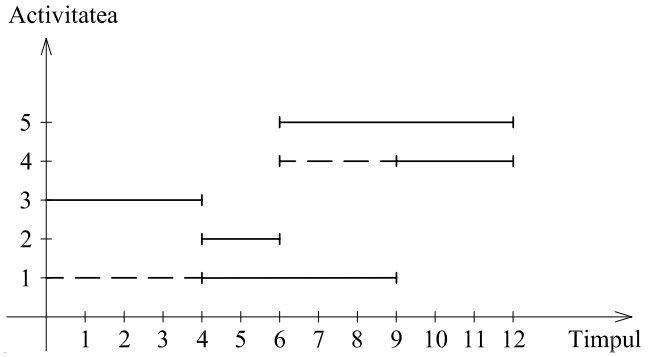
2.10.2 Diagrama Gantt

Un instrument de mare utilitate în analiza drumului critic îl constituie diagrama Gantt, care exprimă la scara timpului, prin linii orizontale duratele activităţilor şi prin linii întrerupte rezervele de timp.

Pentru proiectul din exemplul precedent, diagrama Gantt în care activităţile încep la momentele cele mai devreme este:



Pentru proiectul din exemplul precedent, diagrama Gantt în care activităţile încep la momentele cele mai târzii este:



2.10.3 Algebră de ordonanţare

O altă metodă pentru determinarea momentelor cele mai devreme de începere a activităţilor se obţine prin introducerea pe $\mathbb{R} \cup \{-\infty\}$ a următoarelor operaţii

$$a \oplus b = \max\{a, b\} \; ; \; a \otimes b = a + b \; .$$

Aceste operaţii se extind în mod natural pe mulţimea matricilor cu elemente din $\mathbb{R} \cup \{-\infty\}$.

Etapa 0. Se determină matricea $A = \{a_{ij}\}_{i,j=0}^{n+1}$ unde

$$a_{ij} = \begin{cases} 0, & \text{dacă } i = j; \\ d_i, & \text{dacă } (A_i, A_j) \in U; \\ -\infty, & \text{în rest.} \end{cases}$$

Punem $T_0 = [0, -\infty, -\infty, \ldots, -\infty]$, un vector cu $n+2$ componente. În fapt T_0 este prima coloană din A.

Etapa k. Calculăm $T_k = T_{k-1} \otimes A$. Dacă $(\exists)k \leq n+1$ astfel încât $T_k = T_{k-1}$ atunci algoritmul s-a încheiat iar componentele lui T_k reprezintă momentele cele mai devreme de începere al activităţilor. Dacă $T_{n+2} \neq T_{n+1}$ atunci sistemul de restricţii este incompatibil.

Exemplul 2.10.2 *Utilizând algebra de ordonanţare să se determine momentele cele mai devreme de începere a activităţilor pentru proiectul din exemplul 2.10.1.*

Soluţie. $A =$

	0	1	2	3	4	5	6
0	0	0	$-\infty$	0	$-\infty$	$-\infty$	$-\infty$
1	$-\infty$	0	$-\infty$	$-\infty$	5	$-\infty$	$-\infty$
2	$-\infty$	$-\infty$	0	$-\infty$	2	2	$-\infty$
3	$-\infty$	$-\infty$	4	0	$-\infty$	4	$-\infty$
4	$-\infty$	$-\infty$	$-\infty$	$-\infty$	0	$-\infty$	3
5	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	0	6
6	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	0

$$T_0 = [0, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty]$$

$$T_1 = T_0 \otimes A = [0, 0, -\infty, 0, -\infty, -\infty, -\infty]$$

$$T_2 = T_1 \otimes A = [0, 0, 4, 0, 5, 4, -\infty]$$

$$T_3 = T_2 \otimes A = [0, 0, 4, 0, 6, 6, 10]$$

$$T_4 = T_3 \otimes A = [0, 0, 4, 0, 6, 6, 12]$$

$$T_5 = T_4 \otimes A = [0, 0, 4, 0, 6, 6, 12]$$

În concluzie $t_1^- = 0, t_2^- = 4, t_3^- = 0, t_4^- = 6, t_5^- = 6$.

2.11 Exerciţii

Exerciţiul 2.11.1 *Se consideră graful orientat* $G = (X, U)$ *unde* $X = \{1, 2, 3, 4, 5, 6\}$ *şi* $U = \{(1,2), (1,4), (2,1), (2,3), (2,4), (3,2), (3,4), (3,6), (4,1), (4,3), (4,5), (5,3), (5,4), (5,6), (6,3), (6,5)\}$.
Determinaţi matricea drumurilor aplicând:
a) algoritmul lui Roy-Warshall;
b) metoda compunerii booleene;
c) algoritmul lui Chen.

Exerciţiul 2.11.2 *Să se determine componentele conexe pentru graful neorientat* $G = (X, U)$ *unde* $X = \{x_o, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ *şi* $U = \{(x_0, x_1), (x_0, x_6), (x_0, x_7), (x_1, x_3), (x_2, x_4), (x_2, x_7), (x_3, x_4), (x_5, x_6), (x_5, x_7)\}$.

Exerciţiul 2.11.3 *Să se determine componentele tare conexe pentru graful orientat* $G = (X, U)$ *unde* $X = \{1, 2, 3, 4, 5, 6, 7\}$ *şi* $U = \{(1,2), (1,4), (1,5), (1,7), (2,3), (2,4), (3,1), (3,2), (3,4), (3,5), (4,5), (5,3), (5,6), (6,3), (6,7), (7,1), (7,5)\}$ *aplicând:*
a) Algoritmul lui Malgrange;
b) Algoritmul lui Chen;
c) Algoritmul lui Foulkes.

Exerciţiul 2.11.4 *Să se determine componentele tare conexe pentru graful* $G = (X, U)$ *unde* $X = \{1, 2, 3, 4, 5, 6, 7, 8\}$ *şi* $U = \{(1,2), (1,3), (1,4), (2,3), (2,7), (3,5), (3,6), (3,7), (4,5), (5,8), (5,6), (5,7)\}$ *aplicând:*
a)Algoritmul lui Malgrange;
b) Algoritmul lui Chen;
c) Algoritmul lui Foulkes.

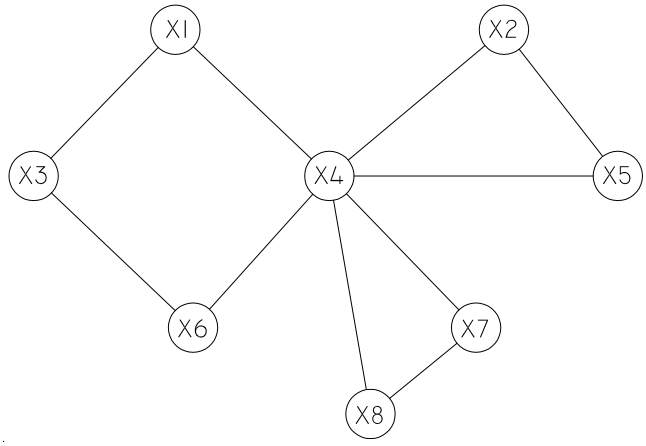
Exerciţiul 2.11.5 *Să se determine componentele tare conexe pentru graful* $G = (X, U)$ *unde* $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ *şi* $U = \{(1,2), (1,3), (1,4), (3,1), (2,3), (2,5), (5,2), (2,8), (3,4), (3,6), (4,5), (4,7), (7,4), (8,6), (6,4), (5,9), (5,7), (4,1), (6,7), (6,8), (7,9), (7,8), (8,7), (8,9)\}$ *aplicând:*
a) Algoritmul lui Malgrange;
b) Algoritmul lui Chen;
c) Algoritmul lui Foulkes.

Exerciţiul 2.11.6 *Să se determine componentele tare conexe pentru graful* $G = (X, U)$ *unde* $X = \{1, 2, 3, 4, 5, 6, 7\}$ *şi* $U = \{(1,4), (2,1), (2,5), (2,3),$

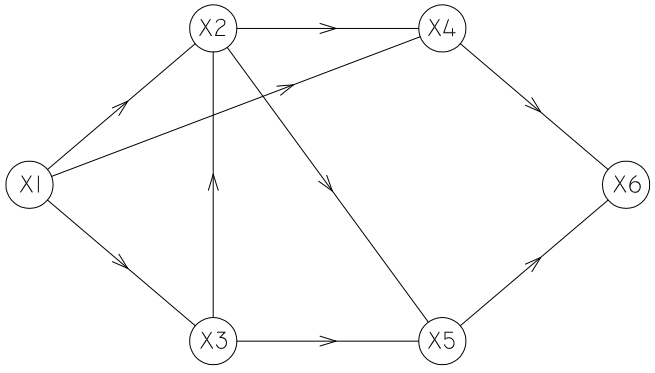
$(3,6),(4,1),(4,5),(4,7),(5,7),(6,3),(6,2),(7,5)\}$ *aplicând:*
a) Algoritmul lui Malgrange;
b) Algoritmul lui Chen;
c) Algoritmul lui Foulkes.

Exerciţiul 2.11.7 *Să se precizeze dacă graful din exerciţiul 2.11.2 este eulerian.*

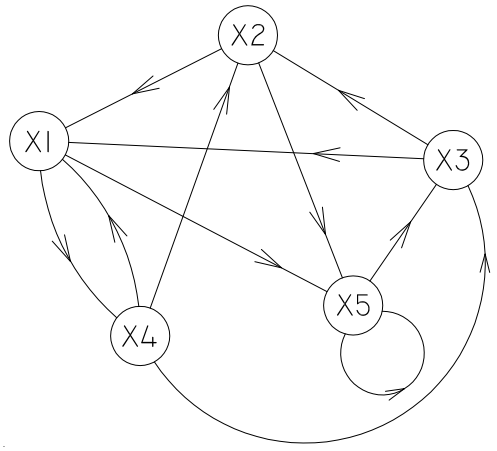
Exerciţiul 2.11.8 *Să se determine un ciclu eulerian pentru graful din figura de mai jos*



Exerciţiul 2.11.9 *Aplicând algoritmul lui Kauffmann să se determine drumurile şi circuitele hamiltoniene pentru graful*
a)



b)



Exerciţiul 2.11.10 *Aplicând algoritmul lui Foulkes să se determine drumurile hamiltoniene pentru grafurile din exerciţiul precedent.*

Exerciţiul 2.11.11 *Aplicând algoritmul lui Chen să se determine drumul hamiltonian pentru grafurile din exerciţiul* 2.11.9.

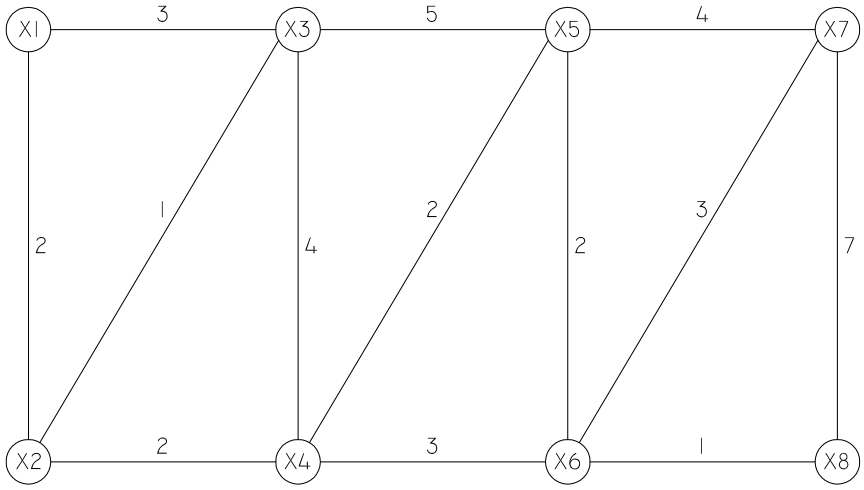
Exerciţiul 2.11.12 *Se consideră graful valorizat*

ij	12	13	14	23	25	28	31	34	36	41	45	47
v_{ij}	8	9	7	4	6	4	6	3	2	5	5	5

ij	52	57	59	64	67	68	74	78	79	86	87	89
v_{ij}	3	10	8	7	6	5	8	7	4	9	8	9

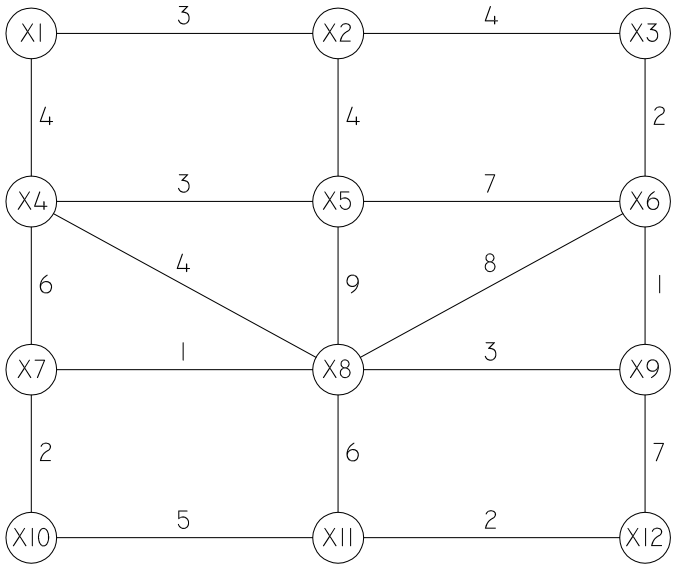
a) Aplicând algoritmul Ford să se determine drumurile de valoare minimă de la x_1 *la* $x_i (i = \overline{2,9})$*;*
b) Aplicând algoritmul Bellman-Kalaba să se determine drumurile de valoare minimă de la x_i *la* $x_9 (i = \overline{1,8})$*;*
c) Aplicând algoritmul lui Dijkstra să se determine drumurile de valoare minimă de la x_1 *la* $x_i (i = \overline{2,9})$*;*
d) Aplicând algoritmul Floyd-Warshall să se determine drumurile de valoare minimă între toate perechile de vârfuri precum şi valorile acestora.

Exerciţiul 2.11.13 *Să se determine arborele de acoperire minim pentru graful*



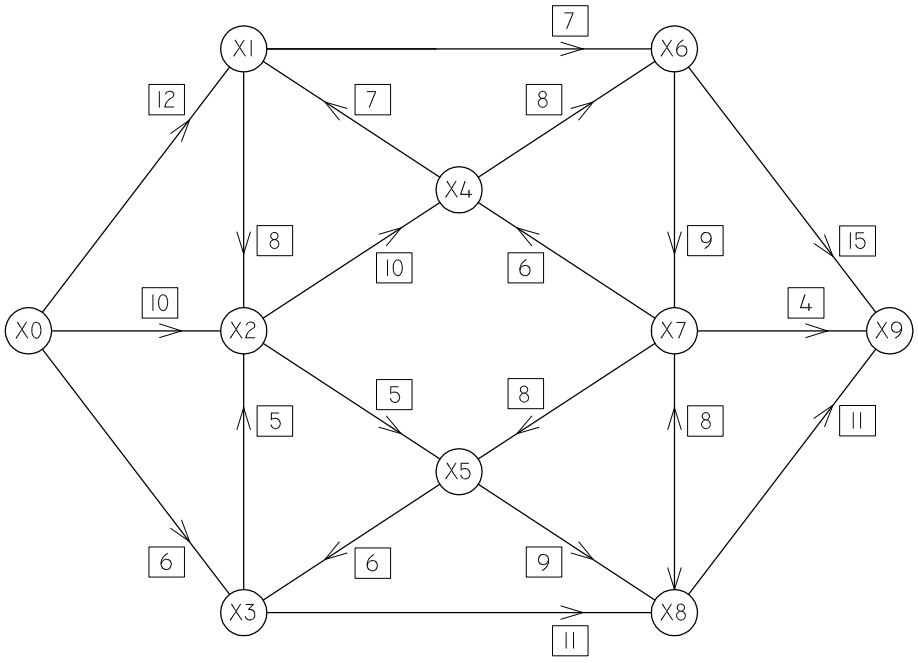
a) Aplicând algoritmul lui Kruskal;
b) Aplicând algoritmul lui Prim.

Exerciţiul 2.11.14 *Să se determine arborele de acoperire minim pentru graful*



a) Aplicând algoritmul lui Kruskal;
b) Aplicând algoritmul lui Prim.

Exerciţiul 2.11.15 *În figura de mai jos este considerată o reţea de transport. Să se determine fluxul maxim aplicând algoritmul Ford-Fulkerson.*



Exerciţiul 2.11.16 *Să se rezolve problema de afectare (minim) în care matricea timpilor este*

$$T = \begin{array}{|c|c|c|c|c|} \hline 4 & 9 & 64 & 169 & 225 \\ \hline 361 & 400 & 1 & 36 & 64 \\ \hline 225 & 256 & 441 & 4 & 16 \\ \hline 484 & 529 & 16 & 81 & 121 \\ \hline 196 & 225 & 500 & 1 & 9 \\ \hline \end{array}$$

1. *Aplicând algoritmul Little;*

2. *Aplicând algoritmul Ungar.*

Exerciţiul 2.11.17 *Să se rezolve problema de afectare (minim) în care matricea timpilor este*

$$T = \begin{array}{|c|c|c|c|c|}\hline 16{,}5 & 13{,}5 & 8 & 5{,}5 & 12 \\ \hline 15 & 16 & 10{,}5 & 7 & 9{,}5 \\ \hline 12 & 15 & 13{,}5 & 11 & 6{,}5 \\ \hline 5{,}5 & 7{,}5 & 13 & 16{,}5 & 11 \\ \hline 10{,}5 & 7{,}5 & 8 & 11{,}5 & 16 \\ \hline \end{array}$$

1. *Aplicând algoritmul Little;*
2. *Aplicând algoritmul Ungar.*

Exerciţiul 2.11.18 *Aplicând algoritmul Little să se determine circuitul hamiltonian de valoare minimă în următorul graf valorizat*

ij	12	13	14	15	21	23	24	25	31	32	34	35
v_{ij}	2	5	6	4	3	7	6	1	5	4	3	6

ij	41	42	43	45	51	52	53	54
v_{ij}	2	5	7	1	6	3	2	5

Exerciţiul 2.11.19 *Se consideră proiectul*

Activitatea	*Durata*	*Activitatea precedentă*
1	4	–
2	6	1
3	4	–
4	12	–
5	10	2, 3
6	24	2, 3
7	7	1
8	10	4, 5, 7
9	3	6, 8

1. *Determinaţi drumul critic aplicând metoda potenţialelor şi construiţi diagramele Gantt.*
2. *Determinaţi momentele cele mai devreme de începere a activităţilor utilizând algebra de ordonanţare.*

Exerciţiul 2.11.20 *Se consideră proiectul*

Activitatea	*Durata*	*Activitatea precedentă*
1	10	–
2	10	1, 8
3	10	2, 4, 7
4	6	1
5	2	1
6	2	5
7	2	6
8	3	–
9	3	2

1. *Determinaţi drumul critic aplicând metoda potenţialelor şi construiţi diagramele Gantt.*

2. *Determinaţi momentele cele mai devreme de începere a activităţilor utilizând algebra de ordonanţare.*

Exerciţiul 2.11.21 *Elaboraţi un algoritm pentru a determina dacă un graf neorientat este bipartit.*

Exerciţiul 2.11.22 *Elaboraţi un algoritm pentru a determina diametrul unui arbore.*

Exerciţiul 2.11.23 *Elaboraţi un algoritm pentru a determina dacă un graf neorientat conţine sau nu un ciclu.*

Exerciţiul 2.11.24 *Un graf orientat $G = (X, U)$ se numeşte semiconex dacă $(\forall)x, y \in X$ avem un drum de la x la y sau de la y la x. Elaboraţi un algoritm pentru a determina dacă un graf orientat este semiconex.*

Exerciţiul 2.11.25 *Fie $G = (X, U)$ un graf orientat. Să se scrie un algoritm pentru determinarea lui G^t.*

Exerciţiul 2.11.26 *Fie $G = (X, U)$ un graf orientat. Să se scrie un algoritm pentru a calcula $G \otimes G$.*

Exerciţiul 2.11.27 *Fie $G = (X, U)$ un graf orientat cu n vârfuri. Să se scrie un algoritm care să determine un vârf $x \in X$ cu proprietatea că $d^+(x) = 0$ şi $d^-(x) = n - 1$.*

Exerciţiul 2.11.28 *Fie $G = (X, U)$ un graf orientat sau neorientat. Să se scrie un algoritm pentru a determina toate vârfurile accesibile dintr-un vârf $x \in X$ precum şi drumurile de lungime minimă de la $x \in X$ la aceste vârfuri accesibile.*

Exerciţiul 2.11.29 *Elaboraţi un algoritm care, pentru un graf care nu este conex, adaugă numărul minim de muchii astfel încât graful să devină conex.*

Exerciţiul 2.11.30 *Elaboraţi un algoritm care, pentru un graf care nu este eulerian, adaugă numărul minim de muchii astfel încât graful să devină eulerian.*

Exerciţiul 2.11.31 *Se consideră graful valorizat*

ij	12	16	23	25	32	34	36	43	45	53	54	56	26
v_{ij}	4	20	10	5	2	22	6	2	8	4	15	12	16

.

a) Aplicând algoritmul Ford să se determine drumurile de valoare minimă de la x_1 la $x_i (i = \overline{2,6})$;
b) Aplicând algoritmul Bellman-Kalaba să se determine drumurile de valoare minimă de la x_i la $x_9 (i = \overline{1,5})$;
c) Aplicând algoritmul lui Dijkstra să se determine drumurile de valoare minimă de la x_1 la $x_i (i = \overline{2,6})$;
d) Aplicând algoritmul Floyd-Warshall să se determine drumurile de valoare minimă între toate perechile de vârfuri precum şi valorile acestora.

Exerciţiul 2.11.32 *Se consideră graful valorizat*

ij	12	14	21	23	24	32	34	36
v_{ij}	3	5	2	4	1	2	1	1

ij	41	43	45	53	54	56	63	65
v_{ij}	1	2	2	1	3	2	2	3

a) Aplicând algoritmul Ford să se determine drumurile de valoare minimă de la x_1 la $x_i (i = \overline{2,6})$;

b) Aplicând algoritmul Bellman-Kalaba să se determine drumurile de valoare minimă de la x_i la $x_9 (i = \overline{1,5})$;
c) Aplicând algoritmul lui Dijkstra să se determine drumurile de valoare minimă de la x_1 la $x_i (i = \overline{2,6})$;
d) Aplicând algoritmul Floyd-Warshall să se determine drumurile de valoare minimă între toate perechile de vârfuri precum şi valorile acestora.

Capitolul 3

Aplicaţii

3.1 Reprezentarea grafurilor

Problema 3.1.1 *Fiind date numărul n de vârfuri şi mulţimea arcelor pentru un graf orientat, se cere să se scrie un program care construieşte matricea de adiacenţă asociată grafului.*

```
#include <iostream.h>
#include <conio.h>
void main( )  {
struct arc{
        int x,y;
}v[20];
int  n,m,i,j,a[20][20];
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
cout<<"Număr de arce "; cin>>m;
for(i=1;i<=m;i++)  {
        cout<<"Extremităţile arcului "<<i<<"  ";
        cin>>v[i].x>>v[i].y;  }
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
              a[i][j]=0;
for(i=1;i<=m;i++)
        a[v[i].x][v[i].y]=1;
cout<<"Matricea de adiacenţă"<<endl;
```

```
for(i=1;i<=n;i++) {
        for(j=1;j<=n;j++)
            cout<<a[i][j]<<" ";
        cout<<endl; }
getch();
}
```

Problema 3.1.2 *Fiind dată matricea de adiacenţă a unui graf orientat cu n vârfuri, se cere să se scrie un program care determină mulţimea arcelor grafului respectiv.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
        int x,y;
}v[20];
int n,m,i,j,k,a[20][20];
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) {
            cout<<"a["<<i<<","<<j<<"]=";
            cin>>a[i][j]; }
k=0;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(a[i][j]==1) {
                k++;
                v[k].x=i;
                v[k].y=j;
            }
cout<<"Graful are "<<n<<" vârfuri"<<endl;
cout<<"Mulţimea arcelor ";
for(i=1;i<=k;i++)
        cout<<"("<<v[i].x<<","<<v[i].y<<") ";
getch();
}
```

Problema 3.1.3 *Pentru un graf orientat se cunosc numărul de vârfuri şi mulţimea arcelor. Să se scrie un program care determină pentru fiecare vârf al grafului mulţimea predecesorilor şi mulţimea succesorilor.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
          int x,y;
}v[20];
int n,m,i,j,a[20][20],e,k,p[20];
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
cout<<"Număr de arce "; cin>>m;
for(i=1;i<=m;i++) {
          cout<<"Extremităţile arcului "<<i<<" ";
          cin>>v[i].x>>v[i].y; }
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
                a[i][j]=0;
for(i=1;i<=m;i++)
          a[v[i].x][v[i].y]=1;
cout<<endl;
for(i=1;i<=n;i++) {
          cout<<"Nodul "<<i<<endl;
           e=0;
           k=0;
           for(j=1;j<=n;j++)
                 if(a[j][i]==1) {
                       e=1;
                       k++;
                       p[k]=j; }
           if(e==0) cout<<"Nu are predecesori"<<endl;
              else {
                 cout<<"Predecesorii sunt ";
                 for(j=1;j<=k;j++)
                       cout<<p[j]<<" ";
                       cout<<endl;
```

```
            }
        e=0;k=0;
         for(j=1;j<=n;j++)
              if(a[i][j]==1)  {
                    e=1;  k++;
                    p[k]=j;  }
        if(e==0)  cout<<"Nu are succesori"<<endl;
            else  {
              cout<<"Succesorii sunt ";
              for(j=1;j<=k;j++)
                    cout<<p[j]<<" ";
                    cout<<endl;  }
           }
    getch();
}
```

Problema 3.1.4 *Fiind dat un graf orientat prin matricea sa de adiacenţă, cu ajutorul unui program, să se determine pentru un vârf x dat gradul său.*

```
#include <iostream.h>
#include <conio.h>
void main( )  {
struct arc{
          int x,y;
}v[20];
int  n,m,i,j,x,a[20][20],d1,d2;
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++)
           for(j=1;j<=n;j++)  {
                cout<<"a["<<i<<","<<j<<"]=";
                cin>>a[i][j];  }
cout<<"Daţi nodul x ";  cin>>x;
d1=d2=0;
for(i=1;i<=n;i++)  {
           if(a[i][x]==1)  d1++;
           if(a[x][i]==1)  d2++;  }
cout<<"Gradul nodului "<<x<<" este "<<d1+d2;
```

```
getch();
}
```

Problema 3.1.5 *Să se determine mulţimea vârfurilor izolate pentru un graf orientat, reprezentat prin matricea sa de adiacenţă.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
        int x,y;
}v[20];
int n,m,i,j,a[20][20],d1,d2,e;
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) {
            cout<<"a["<<i<<","<<j<<"]=";
             cin>>a[i][j]; }
cout<<"Mulţimea vârfurilor izolate este ";
e=0;
for(j=1;j<=n;j++) {
        d1=d2=0;
        for(i=1;i<=n;i++) {
            if(a[i][j]==1)
                d1++;
                if(a[j][i]==1)
                   d2++; }
        if(d1+d2==0) {
             e=1;
            cout<<j<<" ";
        }
}
if(e==0) cout<<"vidă";
getch();
}
```

Problema 3.1.6 *Fiind date numărul n de vârfuri şi mulţimea arcelor pentru un graf orientat, se cere să se scrie un program care construieşte matricea "arce-vârfuri" asociată grafului.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
          int x,y;
}v[20];
int n,m,i,j,a[20][20];
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
cout<<"Număr de arce "; cin>>m;
for(i=1;i<=m;i++) {
           cout<<"Extremităţile arcului "<<i<<" ";
          cin>>v[i].x>>v[i].y; }
for(i=1;i<=n;i++)
          for(j=1;j<=m;j++)
                a[i][j]=0;
for(i=1;i<=m;i++) {
          a[v[i].x][i]=1;
           a[v[i].y][i]=-1; }
cout<<"Matricea arce-vârfuri"<<endl;
for(i=1;i<=n;i++) {
           for(j=1;j<=m;j++)
                cout<<a[i][j]<<" ";
           cout<<endl; }
getch();
}
```

Problema 3.1.7 *Fiind date numărul n de vârfuri şi mulţimea muchiilor pentru un graf neorientat, se cere să se scrie un program care construieşte matricea de adiacenţă asociată grafului.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
           int x,y;
}v[20];
int n,m,i,j,a[20][20];
```

```
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
cout<<"Număr de muchii "; cin>>m;
for(i=1;i<=m;i++) {
        cout<<"Extremităţile muchiei "<<i<<" ";
        cin>>v[i].x>>v[i].y; }
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
                a[i][j]=0;
for(i=1;i<=m;i++) {
        a[v[i].x][v[i].y]=1;
         a[v[i].y][v[i].x]=1; }
cout<<"Matricea de adiacenţă"<<endl;
for(i=1;i<=n;i++) {
        for(j=1;j<=n;j++)
                cout<<a[i][j]<<" ";
         cout<<endl; }
getch();
}
```

Problema 3.1.8 *Fiind dată o matrice binară de dimensiune $n \times n$, cu elemente din mulţimea $\{0, 1\}$, se cere să se determine, cu ajutorul unui program, dacă ea reprezintă matricea de adiacenţă asociată unui graf orientat sau neorientat.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
int n,i,j,k,a[20][20];
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
                do {
                        cout<<"a["<<i<<","<<j<<"]=";
                       cin>>a[i][j];
                }while((a[i][j]!=0)&&(a[i][j]!=1));
k=0;
for(i=1;i<n;i++)
```

```
            for(j=i+1;j<=n;j++)
                  if(a[i][j]!=a[j][i])  k=1;
if(k==0) cout<<"Reprezintă un graf neorientat";
             else  cout<<"Reprezintă un graf orientat";
getch();
}
```

Problema 3.1.9 *Pentru un graf orientat cunoaştem numărul de vârfuri, numărul de arce şi pentru fiecare arc o valoare. Să se scrie un program care determină matricea asociată grafului valorizat dat.*

```
#include <iostream.h>
#include <conio.h>
void main( )  {
struct arc{
             int x,y;
             float v;
}v[20];
int n,m,i,j,a[20][20];
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
cout<<"Număr de arce "; cin>>m;
for(i=1;i<=m;i++)  {
            cout<<"Extremităţile arcului "<<i<<" ";
            cin>>v[i].x>>v[i].y;
            cout<<"Valoarea arcului "<<i<<" ";
            cin>>v[i].v;  }
for(i=1;i<=n;i++)
             for(j=1;j<=n;j++)
                   a[i][j]=0;
for(i=1;i<=m;i++)
            a[v[i].x][v[i].y]=v[i].v;
cout<<"Reprezentarea grafului valorizat"<<endl;
for(i=1;i<=n;i++)  {
             for(j=1;j<=n;j++)
                  cout<<a[i][j]<<" ";
             cout<<endl;  }
getch();
}
```

Problema 3.1.10 *Pentru un graf neorientat cunoaştem numărul de vârfuri, numărul de muchii şi pentru fiecare muchie o valoare. Să se scrie un program care determină matricea asociată grafului valorizat dat.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
          int x,y;
           float v;
}v[20];
int n,m,i,j,a[20][20];
clrscr();
cout<<"Numar de varfuri "; cin>>n;
cout<<"Numar de muchii "; cin>>m;
for(i=1;i<=m;i++) {
          cout<<"Extremitatile muchiei "<<i<<" ";
          cin>>v[i].x>>v[i].y;
          cout<<"Valoarea muchiei "<<i<<" ";
           cin>>v[i].v; }
for(i=1;i<=n;i++)
           for(j=1;j<=n;j++)
                 a[i][j]=0;
for(i=1;i<=m;i++) {
          a[v[i].x][v[i].y]=v[i].v;
          a[v[i].y][v[i].x]=v[i],v; }
cout<<"Reprezentarea grafului valorizat"<<endl;
for(i=1;i<=n;i++) {
           for(j=1;j<=n;j++)
                cout<<a[i][j]<<" ";
           cout<<endl; }
getch();
}
```

Problema 3.1.11 *Un alt mod de reprezentare a elementelor unui graf neorientat este formarea listei sale de adiacenţă. Aceasta este o listă de liste în care informaţia din lista principală este un nod al grafului iar lista secundară este formată din vârfurile adiacente nodului din lista principală. Să se*

scrie un program care permite adăugarea şi ştergerea muchiilor într-un graf neorientat. Memorarea grafului se va realiza cu ajutorul listelor de adiacenţă.

```
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
struct nodlvadiac {
          int nr;
          nodlvadiac *urm;  };
struct nodlladiac {
           int vf;
          nodlvadiac *prim;
           nodlladiac *urm;  };
nodlladiac *cap=NULL;
void insert(int i,int j) {
nodlladiac *p;
nodlvadiac *q;
for(p=cap;p;p=p->urm)
          if(p->vf==i) {
                   q=new nodlvadiac;
                  q->urm=p->prim;
                  q->nr=j;
                  p->prim=q;
                  return; }
p=cap;
cap=new nodlladiac;
cap->urm=p;
cap->vf=i;
cap->prim=NULL;
insert(i,j);
}
int sterge(int i,int j) {
nodlladiac *p;
nodlvadiac *q,*s;
for(p=cap;p;p=p->urm)
          if(p->vf==i) {
                   if(p->prim->nr==j) {
                      q=p->prim->urm;
```

```
                    delete p->prim;
                    p->prim=q;
                    return 0; }
                else for(q=p->prim;s=q->urm;q=q->urm)
                    if(s->nr==j) {
                        q->urm=q->urm->urm;
                        delete s;
                        return 0; }
        return -1; }
return -1;
}
void listare() {
nodlladiac *p;
nodlvadiac *q;
for(p=cap;p;p=p->urm) {
cout<<endl<<p->vf<<":";
        for(q=p->prim;q;q=q->urm) cout<<q->nr<<" "; }
}
void main() {
int i,j;
char c;
clrscr();
do{      cout<<"\n[I]ntroducere muchie || [S]tergere muchie || [E]xit ";
        cin>>c;
        switch(c|32) {
            case 'i':cout<<"Introduceţi capetele muchiei inserate: ";
                cin>>i>>j;
                insert(i,j);
                insert(j,i);
                break;
            case 's':cout<<"Introduceţi capetele muchiei şterse: ";
                cin>>i>>j;
                if(sterge(i,j)==-1) cout<<"\n Nu s-a putut face ştergerea"<<endl;
                sterge(j,i);        }
        listare();
}while(c-'e');
getch();
}
```

Problema 3.1.12 *Se dă un grup format din n persoane, care se cunosc sau nu între ele. Spunem despre o persoană că este celebritate dacă este cunoscută de toţi ceilalţi membri ai grupului, fară ca aceasta să cunoască pe niciun alt membru. Să se determine dacă în grup există o astfel de celebritate.*

Se presupune persoana i ca fiind posibila celebritate. Se iau pe rând persoanele rămase: dacă o persoană j nu cunoaşte pe persoana i atunci persoana j devine posibilă celebritate. Altfel persoana i, posibila celebritate, rămâne în continuare candidată la celebritate. Se trece la următoarea persoană, $j+1$. Când au fost parcurse toate persoanele se verifică încă o dată persoana candidat rămasă. Verificarea se face în acest caz complet. Dacă nu este îndeplinită condiţia, atunci nu există nicio celebritate.

```
#include <stdio.h>
#include <conio.h>
int r[30][30],n,i,j,b;
void main() {
clrscr();
printf("Daţi numărul de persoane "); scanf("%d",&n);
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)  {
              printf("Relaţia dintre %d şi %d ",i,j);
              scanf("%d",&r[i][j]); }
i=1;
for(j=1;j<=n;j++)
        if(r[j][i]==0) i=j;
b=1;
for(j=1;j<=n;j++)
        if((r[j][i]==0) || r[i][j]==1)&& i!=j) b=0;
if(b) printf("Persoana %d e celebritate ",i);
else printf("Nu există celebritate");
getch();
}
```

Problema 3.1.13 *Fiind dat un număr natural n, scrieţi un program care să genereze toate grafurile neorientate cu n vârfuri. Să se afişeze şi numărul de soluţii obţinut.*

```
#include <iostream.h>
#include <conio.h>
```

```
int  st[20],a[20][20],n,nrsol=0;
int  tipar() {
int  i,j,k;
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
                a[i][j]=0;
nrsol++;
k=0;
for(i=1;i<n;i++)
          for(j=i+1;j<=n;j++) {
                k++;
                a[i][j]=st[k];
                a[j][i]=a[i][j]; }
for(i=1;i<=n;i++) {
          for(j=1;j<=n;j++)
                cout<<a[i][j]<<" ";
          cout<<endl; }
cout<<endl;
return n; }
void back(int p) {
int i;
for(i=0;i<=1;i++) {
          st[p]=i;
          if(p==(n*(n-1))/2) tipar();
             else back(p+1); }
}
void  main() {
clrscr();
cout<<"Număr de vârfuri="; cin>>n;
back(1);
cout<<nrsol;
getch();
}
```

Problema 3.1.14 *În fişierul text graf.txt este scrisă o matrice, astfel: pe primul rând două numere naturale separate prin spaţiu, care reprezintă numărul de linii şi numărul de coloane ale matricei, şi, pe următoarele rânduri valori numerice despărţite prin spaţiu, care reprezintă elementele de pe câte*

o linie a matricei. Scrieţi un program care să verifice dacă această matrice poate fi matricea de adiacenţă a unui graf neorientat.

```
#include <fstream.h>
#include <stdio.h>
void main()  {
int a[20][20],n,m,i,j,k,s,ok,ok1,ok2;
fstream f("graf.txt",ios::in);
f>>n>>m;
for(i=1;i<=n;i++)
          for(j=1;j<=m;j++)
             f>>a[i][j];
clrscr();
ok=1;
for(i=1;i<=n;i++)
          for(j=1;j<=m;j++)
             if ((a[i][j]!=0)||(a[i][j]!=1)) ok=0;
ok1=1;
for(j=1;j<=m;j++)  {
          s=0;
          for(i=1;i<=n;i++)
             s=s+a[i][j];
          if (s!=2) ok1=0;  }
ok2=1;
for(j=1;j<m;j++)
          for(k=j+1;k<=m;k++)
             for(i=1;i<=n;i++)
                if(a[i][j]==a[i][k]) ok2=0;
if (ok && ok1 && ok2) cout<<"Matricea de adiacenţă a unui graf neorien-
tat";
          else cout<<"Matricea nu poate fi asociată unui graf neorientat";
f.close();
getch();
}
```

3.2 Parcurgerea unui graf

3.2.1 Introducere

Vom face prezentarea pe cazul unui graf neorientat $G = (X, U)$. În mod asemănător poate fi tratat şi cazul unui graf orientat. Prin parcurgerea unui graf se înţelege examinarea în mod sistematic a nodurilor sale, plecând dintr-un vârf dat x_i, astfel încât fiecare nod, x_j, accesibil din x_i (există lanţ de la x_i la x_j), să fie atins o singură dată. Trecerea de la un nod x_i la altul se face prin explorarea, într-o anumită ordine, a vecinilor lui x_i, adică a vârfurilor cu care nodul x_i curent este adiacent. Această acţiune este numită şi *vizitare* sau *travesare* a vârfurilor grafului, scopul acestei vizitări fiind acela de prelucrare a informaţiei asociată nodurilor.
Există mai multe moduri de parcurgere a grafurilor:

- parcurgerea în lăţime (Breadth First Search);
- parcurgerea în adâncime (Depth First Search);
- parcurgerea prin cuprindere.

Parcurgerile făcute formează arbori parţiali ai grafului iniţial, după numele modului de parcurgere:

- arbori parţiali BFS;
- arbori parţiali DFS;
- arbori parţiali DS.

În continuare vom studia doar primele două metode de parcurgere.

3.2.2 Parcurgerea în lăţime

Se porneşte de la un vârf de start care se vizitează, apoi se vizitează toţi vecinii lui. Pentru fiecare dintre aceste vârfuri, se vizitează vecinii lui care nu au fost vizitaţi. Pentru noile vârfuri, se procedează la fel: se vizitează vecinii acestora care nu au fost vizitaţi. Procedeul continuă până când s-au vizitat toate vârfurile.

Exemplul 3.2.1 *Fie graful din figura următoare.*

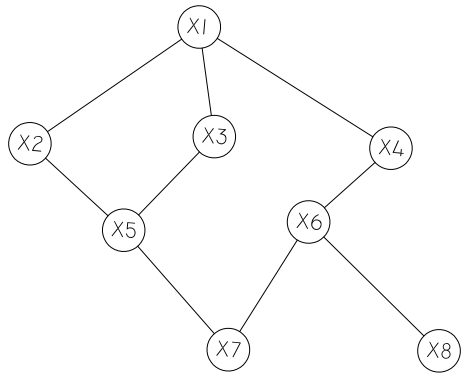


Figura 3.1:

Presupunem că vârful de start este 1. *Ordinea vizitării vârfurilor în parcurgerea BF este* $1, 2, 3, 4, 5, 6, 7, 8$.
Pentru construcţia practică a algoritmului, în vederea alegerii la un moment dat, dintre toţi vecinii unui vârf, pe acela nevizitat încă şi care îndeplineşte condiţia impusă, vom folosi un tablou unidimensional v *cu* n *componente, astfel:*
$(\forall) j \in \{1, 2, \ldots n\}$,

$$v[j] = \begin{cases} 1, & \textit{dacă vârful } j \textit{ a fost vizitat;} \\ 0, & \textit{în caz contrar.} \end{cases}$$

În vectorul c *vom gestiona o coadă în care prelucrarea unui vârf* z *aflat la un capăt al cozii constă în introducerea în celălalt capăt al ei a tuturor vârfurilor* j *vecine cu* z, *nevizitate încă. Iniţial* z *este egal cu vârful dat.*

```
#include <iostream.h>
#include <conio.h>
void main()  {
int a[20][20],c[20],v[20],i,j,k,p,u,n,z,x;
clrscr();
cout<<"Număr de vârfuri ";
cin>>n;
for(i=1;i<=n;i++)
```

```
        a[i][i]=0;
for(i=1;i<n;i++)
        for(j=i+1;j<=n;j++)  {
              cout<<"a["<<i<<","<<j<<"]=";
              cin>>a[i][j];
              a[j][i]=a[i][j];  }
cout<<"Primul nod ";  cin>>x;
for(k=1;k<=n;k++)  {
        c[k]=0;
        v[k]=0;  }
p=u=1;
c[p]=x;
v[x]=1;
while(p<=u)  {
        z=c[p];
        for(k=1;k<=n;k++)
              if((a[z][k]==1)&&(v[k]==0))  {
                 u++;
                 c[u]=k;
                 v[k]=1;  }
        p++;  }
for(k=1;k<p;k++)
        cout<<c[k]<<" ";
getch();
}
```

3.2.3 Parcurgerea în adâncime

Această variantă de parcurgere se caracterizează prin faptul că se merge "în adâncime" ori de câte ori acest lucru este posibil.
Parcurgerea începe cu un vârf iniţial dat x_i şi continuă cu primul dintre vecinii săi nevizitaţi: fie acesta x_j. În continuare se procedează în mod similar cu vârful x_j, trecându-se la primul dintre vecinii lui x_j, nevizitaţi încă. Când acest lucru nu mai este posibil, facem un pas înapoi spre vârful din care am plecat ultima dată şi plecăm, dacă este posibil, spre următorul vârf neluat încă.

Exemplul 3.2.2 *Pentru graful din figura următoare, ordinea de parcurgere DF a nodurilor plecând din nodul* 1 *este:* $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$.

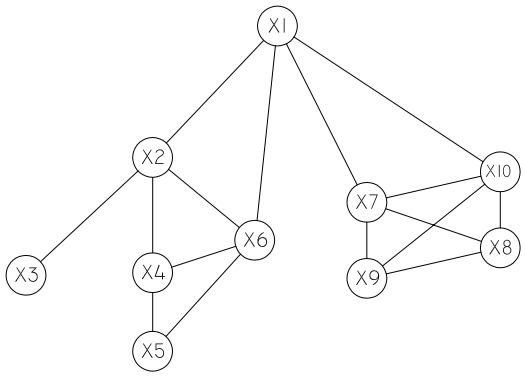


Figura 3.2:

Pentru implementarea algoritmului DF se utilizează vectorul v cu aceeaşi semnificaţie ca şi la algoritmul BF şi se înlocuieşte coada c cu o stivă st care ne permite să plecăm în fiecare moment de la vârful curent spre primul dintre vecinii săi nevizitaţi, acesta din urmă fiind plasat în vârful stivei: cu el se continuă în acelaşi mod. În vectorul urm vom determina în fiecare moment următorul nod ce va fi vizitat după nodul j, (când acesta există). Pentru a-l determina, se parcurge linia j din A începând cu următorul element, până este găsit un vecin al lui j nevizitat încă. Dacă el este găsit, este plasat în vârful stivei, mărind corespunzător şi pointerul de stivă p. Dacă nu se găseşte un asemenea element, stiva coboară (p se decrementează cu 1*) pentru a încerca să continuăm cu următorul element din S.*

```
#include <iostream.h>
#include <conio.h>
void  main()  {
int  a[20][20],m,n,i,k,j,x,y;
int  p,t,st[20],v[20],urm[20];
clrscr();
cout<<"Număr de noduri ";  cin>>n;
cout<<"Număr de muchii ";  cin>>m;
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
```

```
            a[i][j]=0;
for(k=1;k<=m;k++) {
          cout<<"Extremităţi ";
          cin>>x>>y;
          a[x][y]=1;
          a[y][x]=1;  }
for(i=1;i<=n;i++) {
          v[i]=0;
          urm[i]=0;  }
cout<<"Vârf iniţial ";
cin>>x;
cout<<"Parcurgere în adâncime "<<endl;
cout<<x<<" ";
v[x]=1;
p=1;
st[p]=x;
while  (p>0)  {
          t=st[p];
          k=urm[t]+1;
              while  ((k<=n)&& ((a[t][k]==0) || ((a[t][k]==1)&& (v[k]==1))))
                 k++;
          urm[t]=k;
          if  (k==n+1)  p- -;
              else  {
                 cout<<k<<" ";
                 v[k]=1;
                  p++;
                  st[p]=k;
              }
           }
getch();
}
```

3.2.4 Sortarea topologică a unui graf

Elementele unei mulţimi sunt notate cu litere de la 1 la n. Se citeşte un şir de m relaţii de forma xRy cu semnificaţia că elementul x precede elementul y din mulţime. Se cere să se afişeze elementele mulţimii într-o anumită ordine,

în aşa fel încât, pentru orice elemente x, y cu proprietatea că xRy, elementul x să apară afişat înaintea lui y.

Exemplul 3.2.3 *Pentru fişierul de intrare*
5 6
1 2 1 3 1 4 2 4 2 3 4 3
se va afişa
Există posibilitatea sortării topologice
1 2 4 3 5

Redefinim problema în termeni din teoria grafurilor. Dacă elementele mulţimii sunt considerate drept vârfuri şi relaţiile de precedenţă drept arce, se formează în acest mod un graf orientat. Problema are soluţii numai dacă graful nu are circuite (în caz contrar un vârf ar trebui afişat înaintea lui însuşi), ceea ce înseamnă că există cel puţin un vârf în care nu ajunge niciun arc, deci înaintea căruia nu se află niciun alt vârf. Acest vârf va fi afişat primul iar apoi va fi "şters" din graf, reducând problema la $n-1$ vârfuri. Ştergerea se face prin marcarea vârfului găsit ca inexistent şi decrementarea gradelor tuturor vârfurilor care trebuie afişate după el.

```
#include <stdio.h>
#include <mem.h>
#include <conio.h>
int sortare(void);
int solutii(void);
int i,j,a[30][30],l[30][30],m,n,k,ind[20];
void main(void) {
clrscr();
FILE*f;
f=fopen("toposort.in","r");
fscanf(f,"%d%d",&n,&m);
memset(a,0,sizeof(a));
for(i=1;i<=m;i++) {
          fscanf(f,"%d%d",&j,&k);
          a[j][k]=1;  }
fclose(f);
for(i=1;i<=n;i++) ind[i]=0;
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
```

```
            l[i][j]=a[i][j];
for(i=1;i<=n;i++)
        for(k=1;k<=n;k++)
            if(l[i][k]) for(j=1;j<=n;j++)
                if(l[k][j]) l[i][j]=1;
solutii();
}
void solutii(void) {
int di[30];
for(i=1;i<=n;i++) di[i]=0;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(a[i][j]) ++di[j];
for(k=0,i=1;i<=n;i++)
        if((di[i]==0) && (ind[i]==0)) {
            for(j=1;j<=n;j++)
                if(a[i][j]) di[j]–;
            ind[i]=++k; i=1;  }
if(k!=n)
        { puts("Nu există posibilitatea sortării topologice");
        return;}
puts("Există posibilitatea sortării topologice");
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(ind[j]==i){
                printf("%d",j);
                break; }
getch();
}
```

Problema 3.2.4 *Să se verifice dacă un graf neorientat este conex. Se va folosi faptul că un graf este conex, dacă în urma parcurgerii în lăţime s-au vizitat toate nodurile.*

```
#include <iostream.h>
#include <conio.h>
int a[20][20],c[20],v[20],n,i,j,x,z;
int nevizitat() {
```

```
int j,primn;
primn=-1;
j=1;
while((j<=n) && (primn==-1)) {
        if(v[j]==0) primn=j;
        j++; }
return primn;
}
int conex() {
int k,u,p;
cout<<"Primul nod ";  cin>>x;
for(k=1;k<=n;k++) {
        c[k]=0;
        v[k]=0; }
p=u=1;
c[p]=x;
v[x]=1;
while(p<=u) {
        z=c[p];
        for(k=1;k<=n;k++)
           if((a[z][k]==1) && (v[k]==0)) {
              u++;
              c[u]=k;
              v[k]=1; }
           p++; }
if(nevizitat()==-1) return 1;
        else return 0;
}
void main() {
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++)
        a[i][i]=0;
for(i=1;i<n;i++)
           for(j=i+1;j<=n;j++) {
              cout<<"a["<<i<<","j<<"]=";
              cin>>a[i][j];
```

```
                a[j][i]=a[i][j]; }
if(conex()==1)  cout<<"Conex ";
        else cout<<"Nu este conex";
getch();
}
```

Problema 3.2.5 *Se dă matricea de adiacenţă a unui graf neorientat cu n vârfuri, se cere să se afişeze toate componentele conexe precum şi numărul acestora. Se va folosi parcurgerea în lăţime.*

```
#include <iostream.h>
#include <conio.h>
int a[20][20],c[20],v[20],n,i,j,x,z,nc;
int nevizitat() {
int j,primn;
primn=-1;
j=1;
while((j<=n) && (primn==-1)) {
        if(v[j]==0) primn=j;
        j++; }
return primn;
}
void parcurg(int x) {
int k,u,p;
for(k=1;k<=n;k++)
        c[k]=0;
    p=u=1;
    c[p]=x;
    v[x]=1;
    while(p<=u) {
        z=c[p];
        for(k=1;k<=n;k++)
        if((a[z][k]==1) && (v[k]==0)) {
            u++;
            c[u]=k;
            v[k]=1; }
        p++; }
    for(k=1;k<p;k++)
```

```
            cout<<c[k]<<" ";
    cout<<endl; }
void main() {
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++)
            a[i][i]=0;
for(i=1;i<n;i++)
            for(j=i+1;j<=n;j++) {
               cout<<"a["<<i<<","<<j<<"]=";
               cin>>a[i][j];
               a[j][i]=a[i][j]; }
for(i=1;i<=n;i++) v[i]=0;
cout<<"Vârf de plecare "; cin>>x;
nc=0;
while(x!=-1) {
            nc++;
            parcurg(x);
            x=nevizitat(); }
cout<<"Număr de componente conexe "<<nc<<endl;
if(na==1) cout<<"Graf conex";
            else cout<<"Graful nu este conex";
getch();
}
```

Problema 3.2.6 *Fiind dat un graf, ale cărui date se citesc din fişierul "bipartit.in", să se verifice dacă acesta este bipartit. Fişierul este structurat astfel: pe primul rând se găsesc două numere naturale, n şi m, care reprezintă numărul de noduri şi numărul de muchii din graf; pe următoarele m linii găsim perechi de numere naturale ce reprezintă extremităţile unei muchii.*

```
#include <stdio.h>
#include <conio.h>
int i,j,k,n,m,sel,gasit,i1,a[10][10],ind[10];
void intro(void) {
FILE*f;
f=fopen("bipartit.in","r");
```

```
fscanf(f,"%d %d",&n,&m);
for(i=1;i<=m;i++) {
          fscanf(f,"%d%d",&j,&k);
          a[j][k]=a[k][j]=1; }
fclose(f);  }
void main()  {
clrscr();
intro();
sel=1;
for(i1=1;i1<=n;i1++)
          if(ind[i1]==0) {
             ind[i1]=1;
             do{
                gasit=0;
                for(i=i1;i<=n;i++)
                   if(ind[i]==sel)
                      for(j=i1+1;j<=n;j++) {
                         if((a[i][j]==1) && (ind[j]==sel))
                            { printf("Nu este bipartit"); return;}
                if((a[i][j]==1) && (ind[j]==0)) {
                   ind[j]=3-sel;
                   gasit=1;
                   for(k=1;k<=n;k++)
                      if((ind[k]==3-sel) && )a[j][k]==1))
                         { printf("Nu este bipartit"); return;}
                } }
             sel=3-sel;}while(gasit);
          }
puts("\n Prima submulţime: ");
for(i=1;i<=n;i++)
if(ind[i]==1)  printf("%d ",i);
puts("\n A doua submulţime: ");
for(i=1;i<=n;i++)
if(ind[i]==2)  printf("%d ",i);
getch();
}
```

Problema 3.2.7 *Un colecţionar de cărţi rare a descoperit o carte scrisă*

într-o limbă neobişnuită, care foloseşte aceleaşi litere ca şi alfabetul englez. Cartea conţine un index, dar ordinea cuvintelor în index este diferită de cea din alfabetul englez. Colecţionarul a încercat apoi să se folosească de acest index pentru a determina ordinea caracterelor şi a reuşit cu greu să rezolve această problemă. Problema noastră este de a scrie un program care, pe baza unui index sortat dat, să determine ordinea literelor din alfabetul necunoscut. În fişierul alfabet.in se găsesc cel puţin unul şi cel mult 100 de cuvinte, urmate de o linie care conţine caracterul #. Cuvintele dintr-un index conţin numai litere mari din alfabetul englez şi apar în ordinea crescătoare corespunzătoare alfabetului. Rezultatul se va afişa în fişierul "alfabet.out", ordinea literelor se va scrie sub forma unui şir de litere mari, fără spaţii între ele.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
short litere[26],a[26][26],k=0,rez[26];
FILE *f;
void adlalitere(char s[21]) {
unsigned int i;
for(i=0;i<strlen(s);i++)
        litere[s[i]-'A']=1;
}
int esteinlitere(short i) {
return litere[i];
}
int coloanazero(int i) {
for(int j=0;j<26;j++)
         if(a[j][i]) return 0;
return 1;
}
int primalitera() {
int i,j;
for(i=0;i<26;i++)
        if(esteinlitere(i) && coloanazero(i)) {
            rez[k++]=i;
            litere[i]=0;
           for(j=0;j<26;j++)
               a[i][j]=0;
```

```
                return 1;  }
return  0;  }
void  main()  {
char  svechi[21],snou[21];
short  l1,l2,i,j;
f=fopen("alfabet.in","r");
for(i=0;i<26;i++)
            for(j=0;j<26;j++)
                a[i][j]=0;
if(f)  {
            fscanf(f,"%s\n",svechi);
             adlalitere(svechi);  }
if(f)  fscanf(f,"%s\n",snou);
while(strcmp(snou,"#")!=0)  {
            adlalitere(snou);
             i=0;
             l1=strlen(svechi);
             l2=strlen(snou);
            while(svechi[i]==snou[i] && i<l1 && i<l2)  i++;
             if(i<l1 && i<l2)
                a[svechi[i]-'A'][snou[i]-'A']=1;
            strcpy(svechi,snou);
            fscanf(f,"%s\n",snou);  }
fclose(f);
f=fopen("alfabet.out","w");
while(primalitera())  {};
f=fopen("alfabet.out","w");
for(i=0;i<k;i++)
            fprintf(f,"%c",'A'+rez[i]);
fclose(f);
}
```

Problema 3.2.8 *Pentru un graf neorientat se cunoaşte numărul de vârfuri, n, şi pentru cele m muchii extremităţile, se cere să se afişeze toate componentele conexe precum şi numărul acestora. Se va folosi parcurgerea în adâncime.*

```
#include <iostream.h>
#include <conio.h>
```

```
int a[20][20],n,m,vf,p,prim,s[20],viz[20],v[20];
int neviz() {
int j,primn;
primn=-1;
j=1;
while((j<=n)&&(primn==-1))  {
          if(viz[j]==0) primn=j;
          j++; }
return primn;
}
void main() {
int i,j,k,x,y,nc;
clrscr();
cout<<"Număr de noduri ";  cin>>n;
cout<<"Număr de muchii ";  cin>>m;
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
             a[i][j]=0;
for(k=1;k<=m;k++)  {
          cout<<"Extremităţi ";
          cin>>x>>y;
          a[x][y]=1;
          a[y][x]=1;}
prim=1;
for(i=1;i<=n;i++)  {
          viz[i]=0;
          v[i]=0; }
nc=0;
do{
          nc++;
          cout<<"Nodurile componentei conexe "<<nc<<endl;
          s[1]=prim;
          p=1;
          viz[prim]=1;
          cout<<prim<<" ";
          while(p>=1)  {
             j=s[p];
             vf=v[j]+1;
```

```
            while((vf<=n) && ((a[j][vf]==0) || (a[j][vf]==1) && (viz[vf]==1)))
vf++;
            if(vf==n+1)
                p–;
            else {
                cout<<vf<<" ";
                viz[vf]=1;
                p++;
                s[p]=vf; }
          }
           prim=neviz();
          cout<<endl;
}while(prim!=-1);
if(nc==1) cout<<"Graf conex";
          else cout<<"Graful are "<<nc<<" componente conexe ";
getch();
}
```

Problema 3.2.9 *Fiind dat un graf neorientat, prin numărul de noduri, numărul de muchii şi extremităţile fiecărei muchii, se cere să se determine lungimea lanţului minim între două noduri citite de la tastatură, x şi y.*

```
#include <iostream.h>
#include <conio.h>
int m,n,i,pi,ps,x,y,k,a[20][20],c[20],lung[20];
void citire() {
int k,x,y;
cout<<"Număr de noduri=";
cin>>n;
cout<<"Număr de muchii=";
cin>>m;
for(k=1;k<=m;k++) {
          cout<<"Muchia "<<k<<"=";
          cin>>x>>y;
          a[y][x]=a[x][y]=1; }
}
void parcurgere(int ns) {
int lg,pi,ps,k,z,gasit;
for(k=1;k<=20;k++)
```

```
        c[k]=0;
for(k=1;k<=n;k++)
        lung[k]=0; pi=1;
ps=1;
c[pi]=ns;
lung[ns]=-1;
gasit=0;
lg=0;
while((ps<=pi) && (gasit==0)) {
        z=c[ps];
        lg++;
        for(k=1;k<=n;k++)
           if ((a[z][k]==1)&& (lung[k]==0)) {
              pi++;
              c[pi]=k;
              lung[k]=lg;
              if(k==x) {
                 gasit=1;
                 cout<<"Lanţul are lungimea minimă "<<lg; }
           }
        ps++; }
}
void main() {
clrscr();
citire();
cout<<"Nodul de plecare=";
cin>>x;
cout<<"Nodul de sosire=";
cin>>y;
parcurgere(y);
if(lung[x]==0)
        cout<<"Între nodul "<<x<<" şi nodul "<<y<<" nu există lanţ";
getch();
}
```

3.3 Operaţii cu grafuri

Problema 3.3.1 *Fiind date două grafuri orientate pentru care cunoaştem numărul de vârfuri şi mulţimea arcelor, se cere să se scrie un program care determină reuniunea grafurilor date.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc {
        int x,y;
} v[20],w[20],af[20];
int n,m,i,j,n1,m1,g1[20],g2[20],f[20],k,e,k1;
clrscr();
cout<<"Număr de vârfuri în G1 "; cin>>n;
for(i=1;i<=n;i++)  {
        cout<<"Vârful "<<i<<" ";
        cin>>g1[i];  }
cout<<"Număr de arce în G1 "; cin>>m;
for(i=1;i<=m;i++)  {
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>v[i].x>>v[i].y;  }
cout<<"Număr de vârfuri în G2 "; cin>>n1;
for(i=1;i<=n1;i++)  {
        cout<<"Vârful "<<i<<" ";
        cin>>g2[i];  }
cout<<"Număr de arce în G2 "; cin>>m1;
for(i=1;i<=m1;i++)  {
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>w[i].x>>w[i].y;  }
for(i=1;i<=n;i++)
        f[i]=g1[i];
k=n;
for(j=1;j<=n1;j++)  {
        e=0;
        for(i=1;i<=n;i++)
           if(g1[i]==g2[j])  e=1;
           if(e==0) {
```

```
                k++;
                f[k]=g2[j]; }
    }
for(i=1;i<=m;i++)  {
          af[i].x=v[i].x;
          af[i].y=v[i].y; }
k1=m;
for(j=1;j<=m1;j++)  {
          e=0;
          for(i=1;i<=m;i++)
             if((v[i].x==w[j].x)&&(v[i].y==w[j].y))  e=1;
             if(e==0) {
                k1++;
                af[k1].x=w[j].x;
                af[k1].y=w[j].y; }
}
cout<<"Mulţimea vârfurilor este ";
for(i=1;i<=k;i++)
          cout<<f[i]<<" ";
cout<<endl;
cout<<"Arcele noului graf sunt ";
for(i=1;i<=k1;i++)
          cout<<"("<<af[i].x<<","<<af[i].y<<") ";
getch();
}
```

Problema 3.3.2 *Fiind date două grafuri orientate pentru care cunoaştem numărul de vârfuri şi mulţimea arcelor, se cere să se scrie un program care determină intersecţia grafurilor date.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
          int x,y;
}v[20],w[20],af[20];
int n,m,i,j,n1,m1,g1[20],g2[20],f[20],k,e,k1;
```

```
clrscr(); cout<<"Număr de vârfuri în G1 "; cin>>n;
for(i=1;i<=n;i++) {
        cout<<"Vârful "<<i<<" ";
        cin>>g1[i];  }
cout<<"Număr de arce în G1 "; cin>>m;
for(i=1;i<=m;i++)  {
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>v[i].x>>v[i].y;  }
cout<<"Număr de vârfuri în G2 "; cin>>n1;
for(i=1;i<=n1;i++)  {
        cout<<"Vârful "<<i<<" ";
        cin>>g2[i];  }
cout<<"Număr de arce în G2 "; cin>>m1;
for(i=1;i<=m1;i++)  {
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>w[i].x>>w[i].y;  }
k=0;
for(i=1;i<=n;i++)  {
        e=0;
        for(j=1;j<=n1;j++)
           if(g1[i]==g2[j])  e=1;
        if(e==1) {
           k++;
           f[k]=g1[i]; }
}
k1=0;
for(i=1;i<=m;i++)  {
        e=0;
        for(j=1;j<=m1;j++)
           if((v[i].x==w[j].x)&&(v[i].y==w[j].y))  e=1;
        if(e==1) {
           k1++;
           af[k1].x=v[i].x;
           af[k1].y=v[i].y;  }
}
if((k!=0)&&(k1!=0))
        {
           cout<<"Multimea vârfurilor este ";
```

```
            for(i=1;i<=k;i++)
               cout<<f[i]<<" ";
            cout<<endl;
            cout<<"Arcele noului graf sunt ";
            for(i=1;i<=k1;i++)
               cout<<"("<<af[i].x<<","<<af[i].y<<") ";
         }
   else cout<<"Nu există graful";
getch();
}
```

Problema 3.3.3 *Fiind date două grafuri orientate pentru care cunoaştem numărul de vârfuri şi mulţimea arcelor, se cere să se scrie un program care determină suma grafurilor date.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
        int x,y;
}v[20],w[20],af[20];
int n,m,i,j,n1,m1,g1[20],g2[20],f[20],k,e,k1;
clrscr(); cout<<"Număr de vârfuri din cele două grafuri "; cin>>n;
for(i=1;i<=n;i++) {
        cout<<"Vârful "<<i<<" ";
        cin>>g1[i]; }
cout<<"Număr de arce în G1 "; cin>>m;
for(i=1;i<=m;i++) {
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>v[i].x>>v[i].y; }
cout<<"Număr de arce în G2 "; cin>>m1;
for(i=1;i<=m1;i++) {
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>w[i].x>>w[i].y; }
for(i=1;i<=m;i++) {
        af[i].x=v[i].x;
        af[i].y=v[i].y; }
k1=m;
for(j=1;j<=m1;j++) {
```

```
            e=0;
            for(i=1;i<=m;i++)
               if((v[i].x==w[j].x)&&(v[i].y==w[j].y))  e=1;
            if(e==0) {
               k1++;
               af[k1].x=w[j].x;
               af[k1].y=w[j].y;  }
}
cout<<"Mulţimea vârfurilor este ";
for(i=1;i<=n;i++)
            cout<<g1[i]<<" ";
cout<<endl;
if(k!=0) cout<<"Graful nu are arce";
            else  {
               cout<<"Arcele noului graf sunt ";
for(i=1;i<=k1;i++)
                  cout<<"("<<af[i].x<<","<<af[i].y<<") ";
            }
getch();
}
```

Problema 3.3.4 *Fiind date două grafuri orientate pentru care cunoaştem numărul de vârfuri şi mulţimea arcelor, se cere să se scrie un program care determină produsul grafurilor date.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
            int x,y;
}v[20],w[20],af[20];
int n,m,i,j,n1,m1,g1[20],g2[20],f[20],k,e,k1;
clrscr();
cout<<"Număr de vârfuri din cele două grafuri "; cin>>n;
for(i=1;i<=n;i++)  {
            cout<<"Vârful "<<i<<" ";
            cin>>g1[i];  }
cout<<"Număr de arce în G1 "; cin>>m;
for(i=1;i<=m;i++)  {
```

```
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>v[i].x>>v[i].y; }
cout<<"Număr de arce în G2 "; cin>>m1;
for(i=1;i<=m1;i++) {
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>w[i].x>>w[i].y; }
k1=0;
for(i=1;i<=n;i++)
        for(j=1;j<=m1;j++)
           if(v[i].y==w[j].x) {
              k1++;
              af[k1].x=v[i].x;
              af[k1].y=w[j].y; }
cout<<"Mulţimea vârfurilor este ";
for(i=1;i<=n;i++)
        cout<<g1[i]<<" ";
cout<<endl;
cout<<"Arcele noului graf sunt ";
for(i=1;i<=k1;i++)
        cout<<"("<<af[i].x<<","<<af[i].y<<") ";
getch();
}
```

Problema 3.3.5 *Fiind dat un graf orientat pentru care cunoaştem numărul de vârfuri şi mulţimea arcelor, se cere să se scrie un program care determină transpusul grafului dat.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
        int x,y;
}v[20],w[20];
int n,m,i,g1[20];
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++) {
        cout<<"Vârful "<<i<<" ";
```

```
        cin>>g1[i];  }
cout<<"Număr de arce "; cin>>m;
for(i=1;i<=m;i++)  {
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>v[i].x>>v[i].y;  }
for(i=1;i<=m;i++)  {
        w[i].x=v[i].y;
        w[i].y=v[i].x; }
cout<<"Mulţimea vârfurilor este ";
for(i=1;i<=n;i++)
        cout<<g1[i]<<" ";
cout<<endl;
cout<<"Arcele noului graf sunt ";
for(i=1;i<=m;i++)
        cout<<"("<<w[i].x<<","<<w[i].y<<") ";
getch();
}
```

Problema 3.3.6 *Fiind dat un graf orientat pentru care cunoaştem numărul de vârfuri şi mulţimea arcelor, se cere să se scrie un program care determină complementarul grafului dat.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
        int x,y;
}v[20],w[20],c[20];
int n,m,i,j,g1[20],k,l,e;
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++)  {
        cout<<"Vârful "<<i<<" ";
        cin>>g1[i];  }
cout<<"Număr de arce "; cin>>m;
for(i=1;i<=m;i++)  {
        cout<<"Extremităţile arcului "<<i<<" ";
```

```
            cin>>v[i].x>>v[i].y;  }
k=0;
for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)  {
               k++;
               c[k].x=g1[i];
               c[k].y=g1[j];  }
l=0;
for(i=1;i<=k;i++)  {
            e=0;
            for(j=1;j<=m;j++)
               if((c[i].x==v[j].x)&&(c[i].y==v[j].y)) e=1;
            if(e==0) {
               l++;
               w[l].x=c[i].x;
               w[l].y=c[i].y;  }
} cout<<"Mulţimea vârfurilor este ";
for(i=1;i<=n;i++)
            cout<<g1[i]<<" ";
cout<<endl;
cout<<"Arcele noului graf sunt ";
for(i=1;i<=l;i++)
            cout<<"("<<w[i].x<<","<<w[i].y<<") ";
getch();
}
```

Problema 3.3.7 *Fie un graf neorientat G cu n vârfuri a cărei matrice de adiacenţă se citeşte de la tastatură. Mulţimea m de numere întregi reţine vârfurile unui graf $G1$ pentru care se citesc extremităţile muchiilor de la tastatură şi se construieşte vectorul de muchii. Să se verifice dacă $G1$ este subgraf al lui G.*

```
#include <iostream.h>
#include <conio.h>
void main() {
struct muchie {
            int x,y; };
int n1,a[10][10],n,nm,i,j,m[10],mgraf,b,aux,sg,k,ga;
```

```
muchie v[10];
clrscr();
cout<<"Număr de noduri "; cin>>n;
for(i=1;i<=n;i++)  a[i][i]=0;
for(i=1;i<=n-1;i++)
          for(j=i+1;j<=n;j++)  {
             cout<<"a["<<i<<","<<j<<"]=";
             cin>>a[i][j];
             a[j][i]=a[i][j];  }
cout<<"Daţi nodurile subgrafului,-1 pentru sfârşit"<<endl;
n1=0;
cout<<"Nod "; cin>>b;
while(b!=-1) {
          n1++;
          m[n1]=b;
    cout<<"Nod "; cin>>b; }
cout<<"Număr de muchii din subgraf "; cin>>nm;
for(i=1;i<=nm;i++) {
          cout<<"Extremitaţi "; cin>>v[i].x>>v[i].y; }
for(i=1;i<n1;i++)
          for(j=i+1;j<=n1;j++) {
          aux=m[i];m[i]=m[j];m[j]=aux;}
if(m[n1]<=n)  sg=1;
          else  sg=0;
for(i=1;i<=nm;i++)
          if(a[v[i].x][v[i].y]!=1) sg=0;
if(sg==1)
          for(i=1;i<=n-1;i++)
             for(j=i+1;j<=n;j++)
                if((i<=m[n1])&&(j<=m[n1]))  {
                   if(a[i][j]==0)  ga=1;
                      else ga=0;
for(k=1;k<=nm;k++)
          if(((v[k].x==i)&&(v[k].y==j))||((v[k].x==j)&&(v[k].y==i))) ga=1;
if(ga==0) sg=0;  }
if(sg==1) cout<<"Subgraf";
          else cout<<"Nu este subgraf";
getch();
```

}

Problema 3.3.8 *Se citesc de la tastatură m perechi de numere întregi reprezentând extremităţile muchiilor unui graf neorientat G cu n vârfuri şi m muchii şi $m1$ perechi de numere întregi reprezentând extremităţile muchiilor unui graf neorientat $G1$ cu $n1$ vârfuri şi $m1$ muchii. Să se verifice dacă $G1$ este graf parţial al lui G.*

Fie graful $G = (X, U)$ şi mulţimea $V \subseteq U$. Graful $G_p = (X, V)$ se numeşte graf parţial al grafului G.

```
#include <iostream.h>
#include <conio.h>
void main() {
int a[10][10],gp[10][10],n,m,n1,m1,i,j,x,y,ok;
clrscr();
cout<<"Număr de noduri ";cin>>n;
cout<<"Număr de muchii ";cin>>m;
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
             a[i][j]=0;
for(i=1;i<=m;i++) {
          cout<<"x si y "; cin>>x>>y;
          a[x][y]=1; a[y][x]=1; }
cout<<"Număr de noduri din graful G1 "; cin>>n1;
cout<<"Număr de muchii din graful G1 "; cin>>m1;
for(i=1;i<=n1;i++)
          for(j=1;j<=n1;j++)
             gp[i][j]=0;
for(i=1;i<=m1;i++) {
          cout<<"x si y "; cin>>x>>y;
          gp[x][y]=1;
          gp[y][x]=1; }
ok=1;
if(n1!=n) ok=0;
if (ok==1) {
          for(i=1;i<=n-1;i++)
             for(j=i+1;j<=n;j++)
```

```
                if((gp[i][j]==1)&& (a[i][j]==0)) ok=0; }
if (ok==1) cout<<"Graf parţial ";
        else cout<<"Nu e graf parţial ";
getch();
}
```

Problema 3.3.9 *Se citesc de la tastatură m perechi de numere întregi reprezentând extremităţile muchiilor unui graf orientat G cu n vârfuri şi m muchii şi mm perechi de numere întregi reprezentând extremităţile muchiilor unui graf orientat G1 cu nn vârfuri şi mm muchii. Să se verifice dacă G1 este graf generat al lui G.*

```
#include <iostream.h>
#include <conio.h>
void main()  {
struct arc{
    int x,y;
}v[20],w[20];
int n,m,i,j,g[20],gg[20],es,es1,es2,e1,e,nn,mm;
clrscr();
cout<<"Număr de vârfuri pentru graful 1 "; cin>>n;
for(i=1;i<=n;i++)  {
        cout<<"Vârful "<<i<<" ";
        cin>>g[i];  }
cout<<"Număr de arce pentru graful 1 "; cin>>m;
for(i=1;i<=m;i++)  {
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>v[i].x>>v[i].y;  }
cout<<"Număr de vârfuri pentru graful 2 "; cin>>nn;
for(i=1;i<=nn;i++)  {
        cout<<"Vârful "<<i<<" ";
        cin>>gg[i];  }
cout<<"Număr de arce pentru graful 2 "; cin>>mm;
for(i=1;i<=mm;i++)  {
        cout<<"Extremităţile arcului "<<i<<" ";
        cin>>w[i].x>>w[i].y;  }
if(nn<n)  {
        e=1;
```

```
            for(i=1;i<=nn;i++) {
               e1=0;
               for(j=1;j<=n;j++)
                  if(gg[j]==g[i]) e1=1;
               e=e&&e1;
            }
            es=1;
            for(i=1;i<=mm;i++) {
               es1=0;
               for(j=1;j<=nn;j++)
                  if(w[i].x==gg[j]) es1=1;
               es2=0;
               for(j=1;j<=nn;j++)
                  if(w[i].y==gg[j]) es2=1;
               es=es&&es1&&es2; }
            if ((e==1)&&(es==1)) cout<<"Este graf generat";
               else cout<<"Nu este graf generat";
            }
else cout<<"Nu sunt destule vârfuri";
getch();
}
```

Problema 3.3.10 *Pentru un graf neorientat cu n noduri se citeşte matricea de adiacenţă, de la tastatură. Să se scrie un program care obţine un subgraf prin eliminarea tuturor muchiilor care au la extremităţi un nod cu grad par şi nodul x (citit de la tastatură).*

```
#include <fstream.h>
#include <conio.h>
int a[20][20],v[20],n,m,x,i,k,j;
int grad(int i) {
int j,g=0;
for(j=1;j<=n;j++)
         g=g+a[i][j];
return g;
}
void main() {
clrscr();
```

```
cout<<"Număr de vârfuri=";
cin>>n;
for(i=1;i<n;i++)
          for(j=i+1;j<=n;j++)  {
             cout<<"Există muchie între "<<i<<" şi "<<j<<" 0-nu,1-da ";
             cin>>a[i][j];
             a[j][i]=a[i][j];  }
cout<<"x="; cin>>x;
k=0;
for(i=1;i<=n;i++)
          f(grad(i)%2==0) {
             k++;
             v[k]=i;  }
for(i=1;i<=k;i++)
          if(a[v[i]][x]==1)  {
             a[v[i]][x]=0;
             a[x][v[i]]=0;
             m–; }
cout<<"Muchiile grafului parţial sunt"<<endl;
for(i=1;i<=n;i++)
          for(j=1;j<i;j++)
             if(a[i][j]==1) cout<<i<<" "<<j<<endl;
getch();
}
```

Problema 3.3.11 *Să se scrie un program care generează toate grafurile parţiale ale unui graf neorientat pentru care se cunoaşte matricea de adiacenţă.*

```
#include <iostream.h>
#include <conio.h>
typedef int stiva[20];
int n,m,p,k,ev,as,a[20][20],b[20][20],nr;
stiva st;
void citeste()  {
int i,j;
cout<<"Număr de vârfuri ";
cin>>n;
for(i=1;i<=n;i++)
```

```
            for(j=1;j<=n;j++)  {
               cout<<"a["<<i<<","<<j<<"]=";
               cin>>a[i][j];
               if(a[i][j]==1) m++;}
m=m/2;
}
void trans()  {
int i,j,k=1;
for(i=1;i<=n;i++)
            for(j=1;j<i;j++)
               if(a[i][j]==1)  {
                  b[i][k]=1;
                  b[j][k]=1;
                  k++;  }
}
void init()  {
st[k]=0;
}
int succesor() {
if(st[k]<m)  {
               st[k]=st[k]+1;
               return 1;  }
             else return 0;
}
int valid()  {
int i;
if ((k>1) && (st[k]<st[k-1])) return 0;
for (i=1;i<k;i++)
             if (st[k]==st[i]) return 0;
    return 1;
}
int solutie()  {
return k==p;
}
void tipar()  {
int i,j;
cout<<"Graful partial are muchiile ";
for(i=1;i<=p;i++)  {
```

```
            for(j=1;j<=n;j++)
                if(b[i][st[i]]==1) cout<<j<<" ";
            }
cout<<endl;
}
void back()  {
k=1;
init();
while (k>0)  {
            as=1;
            ev=0;
            while (as && !ev){
               as=succesor();
               if (as) ev=valid(); }
            if (as)
               if(solutie()) tipar();
               else {
                    k++;
                   init(); }
             else k–; }
}
void main()  {
            clrscr();
            citeste();
            trans();
            for(p=m;p>=0;p–)
               back();
            getch();
}
```

Problema 3.3.12 *Să se scrie un program care generează toate subgrafurile unui graf neorientat pentru care se cunoaşte matricea de adiacenţă.*

```
#include <iostream.h>
#include <conio.h>
typedef int stiva[100];
int n,p,k,i,j,ev,as,a[10][10],nr;
stiva st;
```

```
void init() {
st[k]=0;
}
int succesor() {
if (st[k]<n) {
          st[k]=st[k]+1;
          return 1; }
else return 0;
}
int valid() {
if ((k>1) && (st[k]<st[k-1])) return 0;
return 1;
}
int solutie() {
return k==p;
}
void tipar() {
int i,j;
cout<<"Nodurile subgrafului: ";
for(i=1;i<=p;i++)
          cout<<st[i]<<" ";
cout<<endl;
cout<<"Muchiile ";
for(i=1;i<=p;i++)
          for(j=i+1;j<=p;j++)
             if(a[st[i]][st[j]]==1)
                cout<<"("<<st[i]<<","<<st[j]<<") ";
cout<<endl;
}
void back() {
k=1;
init();
while (k>0) {
          as=1;
          ev=0;
          while (as && !ev) {
             as=succesor();
             if (as) ev=valid(); }
```

```
            if (as)
               if(solutie()) tipar();
               else  {
                  k++;
                  init(); }
            else k- -; }
}
void main()  {
clrscr();
cout<<"Număr de vârfuri=";
cin>>n;
for(i=1;i<n;i++)
            for(j=i+1;j<=n;j++)  {
               cout<<"Există muchie între "<<i<<" si "<<j<<" 0-nu,1-da ";
               cin>>a[i][j];
               a[j][i]=a[i][j];  }
for(p=n;p>=1;p- -)  {
            back();
            getch();  }
}
```

Problema 3.3.13 *Scrieţi un program care să determine numărul minim de arce care trebuie adăugate la un graf orientat, dat prin matricea sa de adiacenţă, pentru a obţine un graf orientat complet.*

```
#include <iostream.h>
#include <conio.h>
void main()  {
int n,i,j,a[20][20];
clrscr();
cout<<"Număr de vârfuri=";
cin>>n;
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
    if(i!=j)  {
    cout<<"Există arc între "<<i<<" şi "<<j<<" 0-nu,1-da ";
    cin>>a[i][j];  }
```

```
    else a[i][j]=0;
m=0;
for(i=2;i<=n;i++)
    for(j=1;j<i;j++)
    if((a[i][j]==0)&&(a[j][i]==0)) m++;
cout<<"Numărul de arce care trebuie adăugate este "<<m;
getch();
}
```

3.4 Lanţuri şi cicluri

Problema 3.4.1 *Fiind dat un graf orientat pentru care se cunoaşte numărul de vârfuri, numărul de arce şi arcele, se cere să se scrie un program care determină dacă o secvenţă de perechi de numere naturale reprezintă un drum în acel graf.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
struct arc{
        int x,y;
}v[20];
int a[20][20],n,m,i,j,x,y,vb,vb1;
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
           a[i][j]=0;
cout<<"Număr de arce "; cin>>m;
for(i=1;i<=m;i++) {
        cout<<"Extremităţi arc "<<i<<" ";
        cin>>x>>y;
        a[x][y]=1; }
cout<<"Număr de perechi din secvenţă "; cin>>m;
for(i=1;i<=m;i++) {
        cout<<"Perechea "<<i<<" ";
```

```
        cin>>v[i].x>>v[i].y; }
vb=0;
for(i=1;i<m;i++)
        if(v[i].y!=v[i+1].x)  vb=1;
if(vb==0)  {
        vb1=0;
        for(i=1;i<=m;i++)
           if(a[v[i].x][v[i].y]==0)  vb1=1;
        if(vb1==0)  cout<<"Secvenţa dată este drum";
              else cout<<"Nu toate perechile sunt arce";
}
        else cout<<"Secvenţa dată nu este drum";
getch();
}
```

Problema 3.4.2 *Fiind dat un graf orientat pentru care se cunoaşte numărul de vârfuri, numărul de arce şi arcele, se cere să se scrie un program care determină dacă o secvenţă de perechi de numere naturale reprezintă un drum simplu sau nu în acel graf.*

```
#include <iostream.h>
#include <conio.h>
void main( )  {
struct arc{
        int x,y;
}v[20];
int a[20][20],n,m,i,j,x,y,vb,vb1,vb2;
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
           a[i][j]=0;
cout<<"Număr de arce "; cin>>m;
for(i=1;i<=m;i++)  {
        cout<<"Extremităţi arc "<<i<<"  ";
        cin>>x>>y;
        a[x][y]=1;  }
cout<<"Număr de perechi din secvenţă ";  cin>>m;
for(i=1;i<=m;i++)  {
```

```
        cout<<"Perechea "<<i<<" ";
        cin>>v[i].x>>v[i].y; }
vb=0;
for(i=1;i<m;i++)
        if(v[i].y!=v[i+1].x) vb=1;
if(vb==0) {
        vb1=0;
        for(i=1;i<=m;i++)
           if(a[v[i].x][v[i].y]==0) vb1=1;
        if(vb1==0) {
           vb2=0;
           for(i=1;i<m;i++)
              for(j=i+1;j<=m;j++)
                 if((v[i].x==v[j].x)&&(v[i].y==v[j].y)) vb2=1;
           if(vb2==0) cout<<"Secvenţa dată este drum simplu";
              else cout<<"Secvenţa dată este drum"; }
           else cout<<"Nu toate perechile sunt arce"; }
        else cout<<"Secvenţa dată nu este drum";
getch();
}
```

Problema 3.4.3 *Să se verifice dacă o secvenţă de vârfuri dată, reprezintă sau nu un drum elementar într-un graf orientat. Secvenţa de vârfuri este memorată într-un vector cu m componente, graful orientat este dat prin matricea sa de adiacenţă.*

```
#include <iostream.h>
#include <conio.h>
void main()
{
int a[20][20],s[20],i,j,k,n,m,ok;
clrscr();
cout<<"Număr de vârfuri din graf=";
cin>>n;
for(i=1;i<=n;i++)
        a[i][i]=0;
for(i=1;i<=n-1;i++)
        for(j=i+1;j<=n;j++) {
```

```
            cout<<"a["<<i<<","<<j<<"]=";
            cin>>a[i][j];
            a[i][j]=1-a[i][j]; }
cout<<"Număr de vârfuri din secvenţă=";
cin>>m;
for(i=1;i<=m;i++) {
        cout<<"s["<<i<<"]=";
        cin>>s[i]; }
ok=1;
for(i=1;i<=m-1;i++)
        if(a[s[i],s[i+1]]==0) ok=0;
if(ok==0) cout<<"Există noduri între care nu avem arc";
        else {
            for(i=1;i<=m-1;i++)
                for(j=i+1;j<=m;j++)
                    if(s[i]==s[j]) ok=0;
            if (ok==0) cout<<"Secvenţa nu este drum elementar";
                else cout<<"Secvenţa este drum elementar"; }
getch();
}
```

Problema 3.4.4 *Pentru un graf neorientat cu n vârfuri a cărei matrice de adiacenţă se citeşte de la tastatură, să se genereze, folosind metoda backtracking*[1]*, toate lanţurile elementare care au ca extremităţi două vârfuri date,* x_1 *şi* x_2*.*

```
#include <iostream.h>
#include <conio.h>
int a[20][20],st[20],n,i,j,v1,v2;
void tipar(int k)  {
for(i=1;i<=k;i++)
        cout<<st[i]<<" ";
cout<<endl;
}
int valid(int k)  {
int v;
```

[1] Anexa B

```
v=1;
for(i=1;i<=k-1;i++)
          if(st[i]==st[k])  v=0;
if (k>1) if(a[st[k-1]][st[k]]==0) v=0;
if(k>1) if (st[k]<st[k-1]) v=0;
return v; }
void back(int k)  {
int j;
for(j=1;j<=n;j++)  {
          st[k]=j;
             if (valid(k)) if (j==v2) tipar(k);
                else back(k+1);
          }
}
void main()  {
clrscr();
cout<<"Număr de vârfuri="; cin>>n;
for(i=1;i<=n;i++)
          a[i][i]=0;
for(i=1;i<=n-1;i++)
          for(j=i+1;j<=n;j++)  {
             cout<<"a["<<i<<","<<j<<"]=";
             cin>>a[i][j];
             a[j][i]=a[i][j];  }
cout<<"v1="; cin>>v1;
cout<<"v2="; cin>>v2;
st[1]=v1;
back(2);
getch();
}
```

Problema 3.4.5 *Să se scrie un program care să verifice dacă o secvenţă de vârfuri ale unui graf neorientat, memorată într-un vector, formează un lanţ, lanţ simplu sau lanţ elementar, ciclu sau ciclu elementar.*

Pentru testarea unei secvenţe vom face pe rând următoarele verificări:

- dacă oricare două vârfuri consecutive în vector sunt adiacente, atunci secvenţa de vârfuri formează un lanţ. Altfel, ea nu formează un lanţ,

deci nici celelalte tipuri de lanţuri sau cicluri, în acest caz având loc terminarea programului;

- dacă secvenţa nu foloseşte de mai multe ori o muchie, atunci ea este un lanţ simplu. Dacă nu este un lanţ simplu atunci nu poate fi nici lanţ elementar, nici ciclu, în acest caz având loc terminarea programului;

- dacă secvenţa este un lanţ elementar şi, în plus, primul vârf coincide cu ultimul, atunci secvenţa formează un ciclu;

- dacă se formează deja un ciclu se verifică de câte ori a fost luat fiecare vârf. Dacă numărul de apariţii ale tuturor vârfurilor cu excepţia primului vârf şi al ultimului este 1, iar pentru acestea numărul de apariţii este 2, atunci ciclul este elementar.

```
#include <stdio.h>
#include <conio.h>
#include <mem.h>
void main() {
unsigned m,n,i,j,nr[20],a[10][10],l[20],u[10][2],k,ind;
clrscr();
printf("Introduceţi numărul de vârfuri ");scanf("%d",&n);
printf("Introduceţi numărul de muchii ");scanf("%d",&m);
for(i=1;i<=m;i++) {
        printf("Muchia %d: ",i);
        scanf("%d%d",&u[i][0],&u[i][1]); }
memset(a,0,sizeof(a));
for(i=1;i<=m;i++)
        a[u[i][0]][u[i][1]]=a[u[i][1]][u[i][0]]=1;
printf("Introduceţi lungimea secvenţei: ");
scanf("%d",&k);
printf("Introduceţi secvenţa: ");
for(j=1;j<=k;j++)
        scanf("%d",l+j);
for(i=1;i<=k-1;i++)
        if(a[l[i]][l[i+1]]==0) {
           puts("Nu se formează un lanţ"); return; }
puts("Vârfurile formează un lanţ");
```

```
memset(nr,0,sizeof(nr));
for(i=1;i<=k-1;i++)
          for(j=1;j<=m;j++)
             if((u[j][0]==l[i]) && (u[j][1]==l[i+1]) || (u[j][1]==l[i]) &&
                                        (u[j][0]==l[i+1]))
                if(++nr[j]>1)  {
                   puts("Nu este un lanţ simplu"); return; }
puts("Se formează un lanţ simplu");
if(l[1]==l[k])  puts("Se formează un ciclu");
memset(nr,0,sizeof(nr));
ind=1;
for(i=1;i<=k;i++)
          if(++nr[l[i]]>1) {
             puts("Nu se formează lanţ elementar"); ind=0; break; }
if(ind) puts("Se formează un lanţ elementar");
for(i=2;i<=k-1;i++)
          if(nr[l[i]]>1) {puts("Nu se formează ciclu elementar");return;}
if((nr[l[1]]>2) || (l[1]!=l[k])) printf("Nu ");
puts("Se formează ciclu elementar");
getch();
}
```

Problema 3.4.6 *Pentru un graf neorientat cu n vârfuri a cărei matrice de adiacenţă se citeşte de la tastatură, să se genereze toate lanţurile elementare din graf.*

```
#include <iostream.h>
#include <conio.h>
int a[20][20],st[20],n,i,j,v1,v2;
void tipar(int k)  {
for(i=1;i<=k;i++)
          cout<<st[i]<<" ";
cout<<endl;
}
int valid(int k)  {
int v;
v=1;
for(i=1;i<=k-1;i++)
```

```
            if(st[i]==st[k]) v=0;
if (k>1) if(a[st[k-1]][st[k]]==0) v=0;
if(k>1) if (st[k]<st[k-1]) v=0;
return v;
}
void back(int k)  {
int j;
for(j=1;j<=n;j++)  {
            st[k]=j;
            if (valid(k)) if (j==v2) tipar(k);
                else back(k+1);
            }
}
void main()  {
clrscr();
cout<<"Număr de vârfuri="; cin>>n;
for(i=1;i<=n;i++)
            a[i][i]=0;
for(i=1;i<=n-1;i++)
            for(j=i+1;j<=n;j++)  {
                cout<<"a["<<i<<","<<j<<"]=";
                cin>>a[i][j];
                a[j][i]=a[i][j]; }
for(v1=1;v1<=n;v1++)
            for(v2=1;v2<=n;v2++) if(v1!=v2)  {
                st[1]=v1;
                back(2);
            }
getch();
}
```

Problema 3.4.7 *Scrieţi un program care citeşte dintr-un fişier text, "graf.in", lista muchiilor unui graf neorientat şi caută toate lanţurile elementare între două noduri x şi y, care trec printr-un nod z care are gradul minim în graf, presupunem că există maxim un singur vârf cu această proprietate. Etichetele nodurilor între care se caută lanţul se citesc de la tastatură.*

```
#include <fstream.h>
#include <conio.h>
```

```
struct{
        int x,y;
}v[20];
ifstream f;
int x,y,z,st[20],k,m,n,i,j,a[20][20],min,g;
int exista(int z,int k)  {
int ok=0;
for(i=1;i<=k;i++)
        if (st[i]==z) ok=1;
return ok;
}
void tipar(int k)  {
for(i=1;i<=k;i++)
        cout<<st[i]<<" ";
cout<<endl;
}
int valid(int k)  {
int v;
v=1;
for(i=1;i<=k-1;i++)
        if(st[i]==st[k]) v=0;
if (k>1) if(a[st[k-1]][st[k]]==0) v=0;
return v;
}
void back(int k)  {
int j;
for(j=1;j<=n;j++)  {
        st[k]=j;
        if (valid(k)) if ((j==y) && exista(z,k)) tipar(k);
            else back(k+1);
        }
}
void main()  {
clrscr();
f.open("graf.in");
m=0;
n=-1;
while (!f.eof()) {
```

```
            m++;
            f>>v[m].x>>v[m].y;
            if(n<v[m].x) n=v[m].x;
            if(n<v[m].y) n=v[m].y;
};
for(i=1;i<=n;i++)
            a[i][i]=0;
for(i=1;i<=m;i++)  {
            a[v[i].x][v[i].y]=1;
            a[v[i].y][v[i].x]=1; }
cout<<"x="; cin>>x;
cout<<"y="; cin>>y;
min=32000;
z=0;
for(i=1;i<=n;i++)  {
            g=0;
            for(j=1;j<=n;j++)
               g=g+a[i][j];
               if(min>g) {
                    min=g;
                    z=i;
                }
}
if (z==0) cout<<"Nu există nod cu grad minim";
            else {
               st[1]=x;
               back(2);
            }
getch();
}
```

3.5 Arbori

Problema 3.5.1 *Fiind dat un graf neorientat prin matricea sa de adiacenţă, să se scrie un program care verifică dacă acel graf poate reprezenta un arbore.*

```
#include <iostream.h>
#include <conio.h>
```

```
int a[20][20],viz[20],c[20],n,i,j,p,v,con,pr,nod,drum[20][20];
void dfmr(int nod) {
          viz[nod]=1;
          for(int k=1;k<=n;k++)
             if(a[nod][k]==1&&viz[k]==0)
                dfmr(k);
}
void main()  {
clrscr();
cout<<"Număr de noduri=";cin>>n;
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++) {
             cout<<"a["<<i<<","<<j<<"]=";
             cin>>a[i][j];
             a[j][i]=a[i][j]; }
for(i=1;i<=n;i++)
          viz[i]=0;
c[1]=1;
p=1;
u=1;
while(p<=u) {
          v=c[p];
          for(i=1;i<=n;i++)
             if(((a[v][i]==1) || (a[i][v]==1))&&(viz[i]==0)) {
                u++;
                c[u]=i;
                viz[i]=1;  }
          p++; }
con=1;
pr=1;
for(i=1;i<=u;i++)
          pr=pr*viz[i];
if(pr==0) con=0;
if (con==1) {
          for(nod=1;nod<=n;nod++) {
             for(j=1;j<=n;j++)
                viz[j]=0;
             dfmr(nod);
```

```
            viz[nod]=0;
            for(j=1;j<=n;j++)
               drum[nod][j]=viz[j];
         }
         ok1=0;
         for(i=1;i<=n;i++)
            if(a[i][i]==1) ok1=1;
         if (ok1==0) cout<<"Graful este arbore";
            else cout<<"Graful are cicluri";
}
else cout<<"Graful nu este conex";
getch();
}
```

Problema 3.5.2 *Să se construiască un arbore oarecare plecând de la scrierea sa parantezată. Arborele obţinut este stocat folosind legăturile la primul descendent stâng şi la primul frate drept.*

Fiind dat un arbore având vârfurile etichetate cu simboluri formate dintr-un singur caracter, definim scrierea sa parantezată ca fiind scrierea parantezată a rădăcinii sale. Scrierea parantezată a unui vârf constă din afişarea etichetei sale urmată, numai dacă are descendeţi, de scrierea parantezată a descendenţilor cuprinsă între paranteze.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <conio.h>
char n,i,s[128];
struct nod{
         char inf;
         struct nod *fs,*frd;
} *rad;
struct elem{
         char inf,fs,frd;
} t[128];
struct nod*creare(void) {
```

```
struct nod *p;
if(isalnum(s[i])) {
          p=(struct nod*)malloc(sizeof(struct nod));
          p->inf=s[i++];
          p->fs=p->frd=NULL;
          if(s[i]=='(') {
             i++;
             p->fs=creare();}
          p->frd=creare();
          return p;}
else {
          i++;
          return NULL;}
}
void tabel(struct nod*p,int k) {
int v1,v2;
if(p){
          t[k].inf=p->inf;
          if(p->fs) {
             v1=++n;
             t[k].fs=v1;}
          if(p->frd) {
             v2=++n;
             t[k].frd=v2;}
          tabel(p->fs,v1);
          tabel(p->frd,v2);  }
}
void main() {
clrscr();
printf("\n Introduceţi arborele în forma parantezată:\n");
scanf("%s",&s);
i=0;
rad=creare();
n=1;
tabel(rad,1);
printf("\n Număr nod ");
for(i=1;i<=n;i++)
```

```
            printf("%2d",i);
printf("\n Informaţia ");
for(i=1;i<=n;i++)
            printf("%2c",t[i].inf);
printf("\n Fiu stâng ");
for(i=1;i<=n;i++)
            printf("%2d",t[i].fs);
printf("\n Frate drept ");
for(i=1;i<=n;i++)
            printf("%2d",t[i].frd);
getch();
}
```

Problema 3.5.3 *Fiind dat un arbore oarecare, se cere să se scrie un program care permite parcurgerea arborelui în preordine, postordine şi pe orizontală.*

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
#include <alloc.h>
struct nod{
            int inf,n;
            struct nod*leg[10]; };
#define size sizeof(struct nod)
struct nod*s[100], *coada[100];
int ind2,ind3;
struct nod*creare(void);
void pre(struct nod*p);
void post(struct nod*p);
void oriz(struct nod*p);
struct nod*creare(void)  {
int info,nr,i;
struct nod*p;
clrscr();
cout<<"Daţi informaţia "; cin>>info;
p=(struct nod*)malloc(size);
p->inf=info;
cout<<"Daţi numărul de descendenţi pentru "<<info<<" ";
```

```
cin>>p->n;
for(i=0;i<p->n;i++)
        p->leg[i]=creare();
return p;
}
void pre(struct nod*p)  {
int i;
if(p!=NULL)  {
        cout<<p->inf<<" ";
        for(i=0;i<p->n;i++)
           if(p->leg[i]!=NULL)
              pre(p->leg[i]);  }
}
void post(struct nod*p) {
int i;
if(p!=NULL) {
        for(i=0;i¡p->n;i++)
           if(p->leg[i]!=NULL)  post(p->leg[i]);
        cout<<p->inf<<" ";  }
}
void adaug(struct nod*p) {
if(ind2==ind3+1)  cout<<"Coada plină";
        else {
           coada[ind3]=p;
           ind3++; }
}
struct nod*elim(void)  {
if(ind3==ind2)  return 0;
        else return coada[ind2++];
}
void oriz(struct nod*rad)  {
struct nod*p;
int i;
ind2=ind3=0;
adaug(rad);
do{
        p=elim();
        if(p!=NULL) {
```

```
            cout<<p->inf<<" ";
            for(i=0;i<p->n;i++)
               adaug(p->leg[i]);  }
} while(p!=NULL);
}
void main()  {
struct nod*cap;
clrscr();
cout<<"Daţi arborele ";
cap=creare();
cout<<endl<<"Arborele în preordine"<<endl;
pre(cap);
cout<<endl<<"Arborele în postordine"<<endl;
post(cap);
cout<<endl<<"Arborele pe nivele"<<endl;
oriz(cap);
getch();
}
```

3.6 Matricea drumurilor

Problema 3.6.1 *Să se scrie un program care determină matricea drumurilor folosind algoritmul lui Roy-Warshall pentru un graf dat prin matricea de adiacenţă. Matricea de adiacenţă se găseşte în fişierul "graf.in", pe primul rând se găseşte un număr natural n care reprezintă numărul de vârfuri, iar pe următoarele n rânduri liniile matricei de adiacenţă, matricea drumurilor se va afişa în fişierul "drumuri.out".*

```
#include <stdio.h>
FILE *f,*g;
void detdrum(int a[ ][20],int n)  {
int i,j,k;
for(k=1;k<=n;k++)
          for(i=1;i<=n;i++)
             for(j=1;j<=n;j++)
                if((i!=k)&& (j!=k) && (a[i][j]==0)&& (a[i][k]==1)&&
                                    (a[k][j]==1))
```

```
                    a[i][j]=1;
}
void main() {
int i,j,n,a[20][20];
f=fopen("graf.in","r");
g=fopen("drumuri.out","w");
fscanf(f,"%d",&n);
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
             fscanf(f,"%d",&a[i][j]);
detdrum(a,n);
for(i=1;i<=n;i++) {
          for(j=1;j<=n;j++)
             fprintf(g,"%d ",a[i][j]);
          fprintf(g,"\n");  };
fclose(f);
fclose(g);
}
```

Problema 3.6.2 *Fiind dat un graf cu n vârfuri şi m muchii, să se determine componentele sale conexe.*

Vom construi matricea lanţurilor folosind algoritmul lui Roy-Warshall, după care vom determina componentele conexe.

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
void main() {
unsigned m,n,i,j,k,a[10][10],l[10][10],u[10][2],p[10];
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
cout<<"Număr de muchii "; cin>>m;
for(i=1;i<=m;i++) {
          cout<<"Muchia "<<i<<" ";
          cin>>u[i][0]>>u[i][1];  }
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
```

```
            a[i][j]=0;
for(i=1;i<=m;i++)
        a[u[i][0]][u[i][1]]=a[u[i][1]][u[i][0]]=1;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            l[i][j]=a[i][j];
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(l[i][j]==1)
                for(k=1;k<=n;k++)
                    if(l[j][k]) l[i][k]=l[k][i]=1;
k=0;
for(i=1;i<=n;i++)
        p[i]=0;
for(i=1;i<=n;i++)
        if(p[i]==0) {
            p[i]=++k;
            for(j=i+1;j<=n;j++)
                if(l[i][j]==1) p[j]=k;  }
cout<<"Număr de componente conexe "<<k<<" ";
for(i=1;i<=k;i++) {
        cout<<endl<<"Componenta "<<i<<endl;
        for(j=1;j<=n;j++) if(p[j]==i)
            cout<<" "<<j;  }
getch();
}
```

Problema 3.6.3 *Folosind metoda compunerii booleene, să se scrie un program care determină matricea drumurilor pentru un graf orientat dat; se cunoaşte numărul de vârfuri, numărul de arce şi pentru fiecare arc cele două extremităţi.*

```
#include <iostream.h>
#include <conio.h>
int a[20][20],n,u[20][20],dd;
void main()  {
int i,j,n,m,k,x,y,l,d[20][20];
clrscr();
```

```
cout<<"Număr de vârfuri="; cin>>n;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) {
           a[i][j]=0;
           if(i==j) u[i][j]=1;
            else u[i][j]=0; }
cout<<"Număr de arce="; cin>>m;
for(k=1;k<=m;k++) {
        cout<<"Extremităţi arc"<<k<<" ";
        cin>>x>>y;
        a[x][y]=1; }
int b[20][20],c[20][20],e[20][20];
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
           if(a[i][j]>u[i][j]) b[i][j]=a[i][j];
              else b[i][j]=u[i][j];
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
           e[i][j]=b[i][j];
for(k=1;k<=n-2;k++) {
        for(i=1;i<=n;i++)
           for(j=1;j<=n;j++) {
              c[i][j]=0;
              for(int l=1;l<=n;l++) {
                 if(e[i][l]<b[l][j]) dd=e[i][l];
                    else dd=b[l][j];
                 if(c[i][j]>dd) c[i][j]=c[i][j];
                    else c[i][j]=dd; }
           }
        for(i=1;i<=n;i++)
           for(j=1;j<=n;j++)
              e[i][j]=c[i][j]; }
        for(i=1;i<=n;i++)
           for(j=1;j<=n;j++) {
              d[i][j]=0;
              for(l=1;l<=n;l++) {
                 if(a[i][l]<e[l][j]) dd=a[i][l];
                    else dd=e[l][j];
```

```
                    if(d[i][j]>dd)  d[i][j]=d[i][j];
                       else d[i][j]=dd;  }
}
for(i=1;i<=n;i++)  {
          for(j=1;j<=n;j++)
             cout<<d[i][j]<<" ";
          cout<<endl;  }
getch();
}
```

Problema 3.6.4 *Folosind algoritmul lui Chen, să se scrie un program care determină matricea drumurilor pentru un graf orientat dat; se cunoaşte numărul de vârfuri, numărul de arce şi pentru fiecare arc cele două extremităţi.*

```
#include <iostream.h>
#include <conio.h>
void main()  {
int i,j,n,m,ok,k,x,y,v[20],w[20],b[20],a[20][20];
clrscr();
cout<<"Număr de vârfuri=";  cin>>n;
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
             a[i][j]=0;
cout<<"Număr de arce="; cin>>m;
for(k=1;k<=m;k++)  {
          cout<<"Extremităţi arc"<<k<<" ";
          cin>>x>>y;
          a[x][y]=1;  }
k=1;
int ok1=1;
while((k<=n)&&(ok1==1))  {
          for(i=1;i<=n;i++)
             w[i]=a[k][i];
          for(j=1;j<=n;j++)
             if(j!=k)
                if(a[k][j]==1) {
                   for(i=1;i<=n;i++)
```

```
                    v[i]=a[j][i];
                for(i=1;i<=n;i++)
                    if(v[i]>w[i])  b[i]=v[i];
                        else b[i]=w[i];
                for(i=1;i<=n;i++)
                    w[i]=b[i];  };
ok1=0;
for(i=1;i<=n;i++)
        if(w[i]!=a[k][i]) ok1=1;
if(ok1==1)  {
        for(i=1;i<=n;i++)  a[k][i]=w[i];
        k=k;  }
      else  {
        for(i=1;i<=n;i++)
            a[k][i]=w[i];
        k++;
        ok1=1;  }
}
for(i=1;i<=n;i++)  {
        for(j=1;j<=n;j++)
            cout<<a[i][j]<<" ";
        cout<<endl;  }
getch();
}
```

Problema 3.6.5 *Folosind algoritmul lui Kaufmann, să se scrie un program care determină matricea drumurilor pentru un graf orientat dat; se cunoaşte numărul de vârfuri, numărul de arce şi pentru fiecare arc cele două extremităţi.(Observaţie: Graful are maxim zece vârfuri).*

```
#include <iostream.h>
#include <ctype.h>
#include <string.h>
#include <conio.h>
typedef char string[20];
int m,n,i,j,k,t;
char x,y;
string a[10][10],b[10][10],c[10][10],p,l;
```

```
void main() {
clrscr();
cout<<"Număr de vârfuri ";
cin>>n;
cout<<"Număr de arce ";
cin>>m;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) {
            strcpy(a[i][j],"0");
            strcpy(b[i][j],"0"); }
for(i=1;i<=m;i++) {
        cout<<"Extremităţi arc "<<i<<" ";
        cin>>x>>y;
        x=toupper(x);
        y=toupper(y);
        if(x!=y) {
            char *z;
            switch (x) {
                case 'A':z="A";break;
                case 'B':z="B";break;
                case 'C':z="C";break;
                case 'D':z="D";break;
                case 'E':z="E";break;
                case 'F':z="F";break;
                case 'G':z="G";break;
                case 'H':z="H";break;
                case 'I':z="I";break;
                case 'J':z="J";break;
            }
            strcpy(b[(int)x-64][(int)y-64],z);
            switch (y) {
                case 'A':z="A";break;
                case 'B':z="B";break;
                case 'C':z="C";break;
                case 'D':z="D";break;
                case 'E':z="E";break;
                case 'F':z="F";break;
                case 'G':z="G";break;
```

```
                case 'H':z="H";break;
                case 'I':z="I";break;
                case 'J':z="J";break;
            }
            strcpy(a[(int)x-64][(int)y-64],z);
            strcat(b[(int)x-64][(int)y-64],z);
            }
}
for(t=2;t<=n-1;t++) {
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++) {
                strcpy(c[i][j],"0");
                for(k=1;k<=n;k++) {
                    int i1,j1,e=0;
                    for (i1=0;i1<strlen(b[i][k]);i1++)
                        for(j1=0;j1<strlen(a[k][j]);j1++)
                            if(b[i][k][i1]==a[k][j][j1]) e=1;
                    if((strcmp(b[i][k],"0")==0) || (strcmp(a[k][j],"0")==0)||(e==1))
                        strcpy(l,"0");
                    else
                        {
                            strcpy(l,"");
                            strcpy(l,b[i][k]);
                            strcat(l,a[k][j]);
                        }
                        if(strcmp(l,"0")!=0){
                            if (strcmp(c[i][j],"0")==0) strcpy(c[i][j],"");
                            if (strcmp(c[i][j],"")!=0) strcat(c[i][j],",");
                            strcat(c[i][j],l);
                        }
                } }
                for(i=1;i<=n;i++)
                    for(j=1;j<=n;j++)
                        strcpy(b[i][j],c[i][j]);
}
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) {
            if(strcmp(c[i][j],"0")!=0)
```

```
                cout<<c[i][j]<<endl;  }
getch();
}
```

Problema 3.6.6 *Se dă un grup de persoane. O persoană influenţează altă persoană dacă între ele există o relaţie. Să se determine cea mai influentă persoană. Se presupune că două persoane nu se pot influenţa reciproc. Datele problemei se găsesc în fişierul "infl.in", acesta este structurat astfel: pe primul rând se găsesc două numere naturale n şi m care reprezintă numărul de persoane, respectiv numărul de influenţe; pe următoarele m rânduri avem câte o pereche de numere ce reprezintă persoana care influenţează pe altă persoană.*

Rezolvare problemei constă din determinarea vârfului de grad maxim din graful realizat conform influenţelor, după stabilirea tuturor influenţelor indirecte. Aceasta revine la a calcula matricea drumurilor şi a găsi apoi linia cu cei mai mulţi de 1.

```
#include<fstream.h>
#include<mem.h>
#include <conio.h>
void main() {
int i,j,k,n,m,p,a[20][20];
clrscr();
ifstream f("infl.in");
f>>n>>m;
memset(a,0,n*sizeof(a[0]));
for(i=1;i<=m;i++) {
          f>>j>>k;
          a[j][k]=1;}
for(i=1;i<=n;i++)
          for(k=1;k<=n;k++)
             if(a[i][k])
                for(j=1;j<=n;j++)
                   if(a[k][j])
                      a[i][j]=1;
p=m=0;
for(i=1;i<=n;i++) {
```

```
            for(k=0,j=1;j<=n;j++)
               k+=a[i][j];
            if(k>m) {m=k; p=i;}
}
cout<<"Persoana cea mai influentă este "<<p;
getch();
}
```

3.7 Componente conexe şi tare conexe

Problema 3.7.1 *Fiind dat un graf neorientat prin matricea sa de adiacenţă, se cere să se scrie un program care determină toate componentele conexe din graf (se va folosi metoda de programare backtracking).*

```
#include <iostream.h>
#include <conio.h>
typedef int stiva[100];
int a[20][20],n,k,as,ev,x,y,v[20],este=0;
stiva st;
void citeste() {
int i,j;
cout<<"Număr de noduri=";
cin>>n;
for(i=1;i<n;i++)
            for(j=i+1;j<=n;j++) {
            cout<<"a["<<i<<","<<j<<"]=";
            cin>>a[i][j];
            a[j][i]=a[i][j];  }
}
void init() {
st[k]=0;
}
int succesor() {
if(st[k]<n)  {
            st[k]=st[k]+1;
            return 1; }
else return 0;
```

```
}
int valid()  {
int i;
if(k>1)
        if(a[st[k-1]][st[k]]==0) return 0;
for(i=1;i<k;i++)
        if(st[k]==st[i]) return 0;
return 1;
}
int solutie() {
return st[k]==y;
}
void tipar() {
este=1;
}
void back() {
k=2;
init();
while (k>0)  {
        as=1;
        ev=0;
        while (as && !ev) {
            as=succesor();
            if (as) ev=valid(); }
        if (as)
            if(solutie()) tipar();
            else  {
                k++;
                init(); }
        else k–; }
}
void compo()  {
for(x=1;x<=n;x++)
        if(v[x]==0)  {
            st[1]=x;
            v[x]=1;
            cout<<x<<" ";
            for(y=1;y<=n;y++)
```

```
                if(x!=y)  {
                   este=0;
                back();
                if (este)  {
                   v[y]=1;
                   cout<<y<<" "; }
                }
             cout<<endl; }
}
void main() {
          clrscr();
          citeste();
          compo();
          getch();
}
```

Problema 3.7.2 *Fiind dat un graf orientat prin matricea sa de adiacenţă, se cere să se scrie un program care determină toate componentele tare conexe din graf (se va folosi metoda de programare backtracking).*

```
#include <iostream.h>
#include <conio.h>
typedef int stiva[100];
int a[20][20],n,k,as,ev,x,y,v[20],este=0,este1,este2;
stiva st;
void citeste() {
int i,j;
cout<<"Număr de vârfuri=";
cin>>n;
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++) {
             cout<<"a["<<i<<","<<j<<"]=";
             cin>>a[i][j]; }
}
void init() {
st[k]=0;
}
int succesor() {
if(st[k]<n) {
```

```
            st[k]=st[k]+1;
            return 1; }
else return 0;
}
int valid() {
int i;
if(k>1)
            if(a[st[k-1]][st[k]]==0) return 0;
for(i=1;i<k;i++)
            if(st[k]==st[i]) return 0;
return 1;
}
int solutie() {
return st[k]==y;
}
void tipar() {
este=1;
}
void back() {
k=2;
init();
while (k>0)  {
            as=1;
            ev=0;
            while (as && !ev) {
                as=succesor();
                if (as) ev=valid(); }
            if (as)
                if(solutie()) tipar();
                else  {
                    k++;
                    init(); }
            else k–; }
}
void compo() {
int i,j;
for(i=1;i<=n;i++)
            if(v[i]==0) {
```

```
            v[i]=1;
            cout<<i<<" ";
            for(j=1;j<=n;y++)
               if(j!=i) {
                  x=i; y=j; st[1]=x; este=0; back(); este1=este;
                  x=j; y=i; st[1]=x; este=0; back(); este2=este;
                  if (este1 && este2) {
                     v[j]=1;
                     cout<<j<<" "; }
               }
            cout<<endl; }
}
void main() {
         clrscr();
         citeste();
         compo();
         getch();
}
```

Problema 3.7.3 *Fiind dat un graf orientat prin matricea sa de adiacenţă, se cere să se scrie un program care determină toate componentele tare conexe din graf (se va folosi matricea drumurilor).*

```
#include <iostream.h>
#include <conio.h>
typedef int stiva[100];
int a[20][20],n,v[20],b[20][20],ap[20][20];
stiva st;
void citeste() {
int i,j;
cout<<"Număr de vârfuri=";
cin>>n;
for(i=1;i<=n;i++)
         for(j=1;j<=n;j++) {
            cout<<"a["<<i<<","<<j<<"]=";
            cin>>a[i][j]; }
}
void predecesor() {
```

```
int i,j;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            ap[i][j]=a[j][i];
}
void transs() {
int i,j,k;
for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(a[i][j]==0 && i!=k && j!=k)
                    a[i][j]=a[i][k]*a[k][j];
}
void transp() {
int i,j,k;
for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(ap[i][j]==0 && i!=k && j!=k)
                    ap[i][j]=ap[i][k]*ap[k][j];
}
void intersectie() {
int i,j;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            b[i][j]=a[i][j]*ap[i][j];
}
void compo() {
int i,j;
for(i=1;i<=n;i++)
        if(v[i]==0) {
            v[i]=1;
            cout<<endl<<"Componenta conţine "<<i<<" ";
            for(j=1;j<=n;j++)
                if(b[i][j]==1 && i!=j) {
                    cout<<j<<" ";
                    v[j]=1; }
```

```
        }
}
void main() {
        clrscr();
        citeste();
        predecesor();
        transs();
        transp();
        intersectie();
        compo();
        getch();
}
```

Problema 3.7.4 *Fiind dat un graf neorientat prin numărul de noduri, numărul de muchii şi extremităţile fiecărei muchii, se cere să se scrie un program care determină componentele conexe ale grafului care conţin un nod dat x (se va folosi algoritmul de scanare).*

```
#include <iostream.h>
#include <conio.h>
int a[20][20],n,m,i,j,x,y,r[20],c[20],v[20];
int z,p,u,nar,k;
struct{
        int x,y;
}ar[20];
void main()  {
clrscr();
cout<<"Număr de noduri ";
cin>>n;
cout<<"Număr de muchii ";
cin¿¿m;
for(i=1;i<=m;i++)  {
        cout<<"Extremităţi muchia "<<i<<":";
        cin>>x>>y;
        a[x][y]=1;
        a[y][x]=1;  }
cout<<"Primul nod ";
cin>>x;
for(k=1;k<=n;k++) {
```

```
            c[k]=0;
            r[k]=0;
            v[k]=0;  }
p=u=1;
c[p]=x;
r[p]=x;
v[x]=1;
nar=0;
while(p<=u) {
            z=c[p];
            for(k=1;k<=n;k++)
               if((a[z][k]==1)&&(v[k]==0)) {
                  u++;
                  nar++;
                  ar[nar].x=z;
                  ar[nar].y=k;
                  c[u]=k;
                  r[u]=k;
                  v[k]=1; }
            p++;  }
cout<<"Mulţimea R ";
for(i=1;i<=u;i++)
            cout<<r[i]<<" ";
cout<<endl;
cout<<"Mulţimea A ";
for(i=1;i<=nar;i++)
            cout<<"("<<ar[i].x<<","<<ar[i].y<<") ";
getch();
}
```

Problema 3.7.5 *Fiind dat un graf orientat prin numărul de vârfuri, numărul de arce şi extremităţile arcelor, se cere să se scrie un program care determină toate componentele tare conexe ale grafului, folosind algoritmul lui Malgrange.*

```
#include <iostream.h>
#include <conio.h>
int a[20][20],n,m,i,j,x,y,l,k,ok,v1[20],v1c[20],v2[20],v2c[20],vi[20],vic[20];
```

```
int vb,l1,k1,m1,v11[20],v22[20],v111[20],v222[20],kk,e,p;
int viz[20],vb1;
void main()  {
struct arc{
          int x,y;
}v[20];
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
for(i=1;i<=n;i++)
          viz[i]=0;
cout<<"Număr de arce "; cin>>m;
for(i=1;i<=m;i++)  {
          cout<<"Extremităţi arc "<<i<<" ";
          cin>>x>>y;
          a[x][y]=1;  }
vb1=0;
do  {
          int ee=1;
          i=1;
          vb1=0;
          x=0;
          while((i<=n)&&(ee==1))
             if(viz[i]==0)  {
                ee=0;
                x=i;
                vb1=1; }
             else  i++;
          if(x!=0) {
             k=0;
             for(i=1;i<=n;i++)
                if((a[i][x]==1)&&(viz[i]==0))  {
                   k++;
                   v1[k]=i;
                   v11[k]=i;
                   v1c[k]=i; }
             k1=k;
             l=0;
             for(i=1;i<=n;i++)
```

```
if((a[x][i]==1)&&(viz[i]==0))  {
   l++;
   v2[l]=i;
   v2c[l]=i;
   v22[l]=i;  }
l1=l;
m=0;
for(i=1;i<=k;i++)  {
   ok=0;
   for(j=1;j<=m;j++)
      if(v1[i]==v2[j]) ok=1;
      if(ok)  {
         m++;
         vi[m]=v1[i];  }
}
m1=m;
for(i=1;i<=m1;i++)
   vic[i]=vi[i];
vb=1;
while (vb==1) {
   vb=0;
   k=0;
   for(i=1;i<=k1;i++)
      for(j=1;j<=n;j++)
         if((a[j][v11[i]]==1)&&(viz[j]==0))  {
            k++;
            v1[k]=j;
            v111[k]=j;  }
   kk=k;
   for(i=1;i<=k1;i++)  {
      e=0;
      for (p=1;p<=kk;p++)
         if(v1[p]==v1c[i]) e=1;
         if(e==0) {
            k++;
            v1[k]=v1c[i];  }
   }
   l=0;
```

```
for(i=1;i<=l1;i++)
   for(j=1;j<=n;j++)
      if((a[v22[i]][j]==1)&&(viz[j]==0)) {
         l++;
         v2[l]=j;
         v222[l]=j; }
kk=l;
for(i=1;i<=l1;i++) {
   e=0;
   for(p=1;p<=kk;p++)
   if(v2[p]==v2c[i]) e=1;
   if(e==0) {
      l++;
      v2[l]=v2c[i]; }
}
m=0;
for(i=1;i<=k;i++) {
   ok=0;
   for(j=1;j<=l;j++)
      if(v1[i]==v2[j]) ok=1;
      if(ok) {
         m++;
         vi[m]=v1[i]; }
}
int aux;
if(m1!=0)
  for(i=1;i<m1;i++)
   for(j=i+1;j<=m1;j++)
      if(vic[i]>vic[j]) {
         aux=vic[i];
         vic[i]=vic[j];
         vic[j]=aux; }
if(m!=0)
  for(i=1;i<m;i++)
   for(j=i+1;j<=m;j++)
      if(vi[i]>vi[j]) {
         aux=vi[i];
         vi[i]=vi[j];
```

```
                    vi[j]=aux;  }
                if (m1!=m) vb=0;
                  else {
                    int vb1=0;
                    for(i=1;i<=m;i++)
                        if(vic[i]!=vi[i])  vb1=1;
                        if (vb1==1) vb=0;
                          else vb=2; }
                if(vb==0) {
                    k1=k;
                    for(i=1;i<=k;i++) {
                        v1c[i]=v1[i];
                        v11[i]=v111[i]; }
                l1=l;
                for(i=1;i<=l;i++) {
                    v2c[i]=v2[i];
                    v22[i]=v222[i]; }
                m1=m;
                for(i=1;i<=m;i++)
                    vic[i]=vi[i];
                    vb=1; }
            }
            cout<<" ";
            for(i=1;i<=m;i++)  {
                cout<<vi[i]<<" ";
                viz[vi[i]]=1; } }
        }while (vb1);
getch();
}
```

Problema 3.7.6 *Folosind algortimul lui Foulkes, să se determine pentru un graf orientat componetele sale tare conexe. Graful se găseşte în fişierul "foulkes.in"; pe primul rând se găsesc două numere naturale, primul, n, reprezintă numărul de vârfuri ale grafului, iar al doilea număr, m, stabileşte numărul de arce din graf. Pe următorul rând se găsesc m perechi de numere naturale ce reprezintă extremităţile arcelor.*

Pentru următorul fişier de intrare:
7 10

1 2
2 3
4 2
3 5
5 4
4 3
7 1
6 1
6 2
1 6
Se vor afişa cele 3 componente tari conexe de mai jos:
7
1 6
2 3 4 5

```
#include <stdio.h>
#include <stdlib.h>
FILE *f;
int a[20][20],b[20][20],c[20][20],ind[20],i,j,k,n,m,detect,egal;
define false 0
define true !false
void init(void) {
f=fopen("foulkes.in","r");puts("");
fscanf(f,"%d%d",&n,&m);
for(i=1;i<=m;i++) {
          fscanf(f,"%d%d",&j,&k);
          a[j][k]=1;}
for(i=1;i<=n;i++)
          a[i][i]=1;
}
void putere(void) {
int i,j,k;
do{
          for(i=1;i<=n;i++)
             for(j=1;j<=n;j++) {
                c[i][j]=0;
                for(k=1;k<=n;k++)
                   c[i][j]|=b[i][k]& b[k][j];
```

```
            }
        egal=true;
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(c[i][j]!=b[i][j]) {
                    egal=false;
                    b[i][j]=c[i][j];}
}while(!egal);
}
void elim(void) {
detect=false;
for(i=1;i<=n;i++)
        if(ind[i]!=2) {
            egal=true;
            for(j=1;j<=n;j++)
                if(ind[j]!=2)
                    if(c[i][j]==0) {
                        egal=false;
                        break;}
            if(egal) {
                ind[i]=1;
                detect=true;}  }
if(!detect) {
        puts("Graful nu este conex");
        exit(1); }
for(j=1;j<=n;j++)
        if(ind[j]==1) {
            egal=true;
            for(i=1;i<=n;i++)
                if(ind[i]==0)
                    if(c[i][j]==1) egal=false;
if(egal) {
        printf("%d",j);
        ind[j]=2;}  }
puts("");
}
void main() {
int i,j;
```

```
clrscr();
init();
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
             b[i][j]=a[i][j];
do {
          detect=false;
          for(i=1;i<=n;i++)
             if(ind[i]!=2) {
                detect=true;
                putere();
                elim();}
}while(!detect);
getch();
}
```

Problema 3.7.7 *Să se determine componentele tare conexe pentru un graf orientat a cărei matrice de adiacenţă se găseşte în fişierul "in.txt", numărul de vârfuri se citeşte de la tastatură. Se va implementa algoritmul lui Chen.*

```
#include <iostream.h>
#include <fstream.h>
#define max 50
int a[max][max],gp[max],gm[max],v[max];
int n,i,j,k,ok;
int nevizitat()  {
int i;
for (i=1;i<=n;i++)
          if (v[i]==0) return i;
return -1;
}
void main()  {
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
ifstream f;
f.open("in.txt");
for (i=1; i<=n; i++)
          for (j=1; j<=n; j++)
```

```
            f>>a[i][j];
f.close();
for (i=1; i<=n; i++) v[i]=0;
do{
          k=nevizitat();
          if (k!=-1){
             for (i=1;i<=n;i++)  gp[i]=a[k][i];
             do {
                ok=0;
                for (j=1;j<=n;j++)
                   if(gp[j]==1)
                      for (i=1;i<=n;i++)
                         if ((a[j][i]==10&& gp[i]==0)) {
                            ok=1;
                            gp[i]=1; }
             }  while (ok==1);
              for (i=1;i<=n;i++) gm[i]=a[i][k];
             do {
                ok=0;
                for(j=1;j<=n;j++)
                   if (gm[j]==1)
                      for(i=1;i<=n;i++)
                         if((a[i][j]==1)&&(gm[i]==0))  {
                            ok=1;
                            gm[i]=1; }
             }while (ok==1);
for (i=1;i<=n;i++)
                if ((gp[i]==1)&&(gm[i]==1)) {
                   cout<<i<<",";
                   v[i]=1;  }
             if (k==nevizitat()) {
                v[k]=1;
                cout<<k;  }
             cout<<endl;
          }  }while (k!=-1);
getch();
}
```

Problema 3.7.8 *Să se determine componentele tare conexe ale unui graf, folosind un algoritm bazat pe algoritmul Roy-Warshall.*

Se va determina matricea drumurilor şi apoi vom determina perechile de forma (i, j) (j, i) cu $i! = j$ cu valoarea 1 în matricea drumurilor.

```
#include <stdio.h>
#include <conio.h>
int a[20][20],i,j,k,n,m,sel[20];
void init(void) {
FILE *f=fopen("tareconx.in","r");
fscanf(f,"%d%d",&n,&m);
for(i=1;i<=m;i++) {
          fscanf(f,"%d%d",&j,&k);
          a[j][k]=1;}
fclose(f);  }
void main() {
clrscr();
init();
for(k=1;k<=n;k++)
          for(i=1;i<=n;i++)
             for(j=1;j<=n;j++)
                if(a[i][j]==0)
                   a[i][j]=a[i][k] & a[k][j];
k=1;
for(i=1;i<=n;i++)
          if(!sel[i]) {
             printf("Componenta tare conexă %d: %d ",k,i);
             for(j=1;j<=n;j++)
                if((j!=i) && (a[i][j]!=0 && (a[j][i]!=0)) {
                   printf("%d",j);
                   sel[j]=1;  }
             k++;
          printf("\n");  }
getch();
}
```

3.8 Determinarea circuitelor euleriene

Problema 3.8.1 *Fiind dat un graf neorientat cu n noduri şi m muchii, se cere să se scrie un program care determină un ciclu eulerian pentru un nod dat x. Se va implementa algoritmul lui Euler. Vom presupune că graful dat este eulerian.*

```
#include <iostream.h>
#include <conio.h>
int a[10][10],l[20],l1[20],n,i,j,m;
int gasit, muchie,x,y,k,x1,k1,ii,p,jj;
void main()  {
clrscr();
cout<<"Număr de vârfuri ";
cin>>n;
cout<<"Număr de muchii ";
cin>>m;
for(i=1;i<=m;i++)  {
          cout<<"Extremităţi muchia "<<i<<" ";
          cin>>x>>y;
          a[x][y]=1;
          a[y][x]=1;  }
k=1;
cout<<"Vârf de început ";
cin>>x;
l[k]=x;
x1=x;
do {
          gasit=0;
          i=1;
          while((i<=n) && (gasit==0))
             if(a[x][i]==1)  {
                gasit=1;
                k++;
                l[k]=i;
                a[x][i]=0;
                a[i][x]=0;
                x=i; }
```

```
            else i++;
}while(x!=x1);
do {
        muchie=0;
        for(i=1;i<=k;i++)  {
            for(j=1;j<=n;j++)
                if(a[l[i]][j]==1) {
                    muchie=1;
                    x=l[i];
                    p=i;
                    x1=x;
                    k1=0;
                    do {
                        gasit=0;
                        i=1;
                        while((i<=n) && (gasit==0))
                            if(a[x][i]==1)  {
                                gasit=1;
                                k1++;
                                l1[k1]=i;
                                a[x][i]=0;
                                a[i][x]=0;
                                x=i; }
                            else i++;
                    }while(x!=x1);
                    for(jj=1;jj<=k1;jj++) {
                        for(ii=k;ii>p;ii–)
                            l[ii+1]=l[ii];
                        k++; }
                    for(ii=1;ii<=k1;ii++)
                        l[p+ii]=l1[ii];  }
            }
}while(muchie==0);
for(i=1;i<=k;i++)
        cout<<l[i]<<" ";
getch();
}
```

Problema 3.8.2 *Fiind dat un graf neorientat prin matricea sa de adiacenţă, se cere să se scrie un program care verifică dacă graful respectiv este eulerian sau nu. Dacă răspunsul este afirmativ se va determina un ciclu eulerian.*

```
#include <iostream.h>
#include <conio.h>
typedef stiva[20];
int n,a[20][20],viz[20],vf,k,m,g[20],p=1,c[20],c1[20];
stiva st;
void init(int i) {
vf=1;
st[vf]=1;
viz[i]=1;
}
int estevida() {
return vf==0;
}
void adaug(int i) {
vf++;
st[vf]=i;
viz[i]=1;
}
void elimin() {
vf- -;
}
void prelucrare() {
int i=1;
k=st[vf];
while(i<=n && (a[i][k]==0 || (a[i][k]==1 && viz[i]==1)))
          i++;
if(i==n+1) elimin();
          else {
             p++;
             adaug(i); }
}
int conex() {
k=1;
init(k);
```

```
while (!estevida())
          prelucrare();
return (p==n);
}
void grad() {
for(int i=1;i<=n;i++)
          for(int j=1;j<=n;j++)
             if (a[i][j]==1) {
                g[i]++;
                m++; }
m=m/2;
}
int izolat() {
for(int i=1;i<=n;i++)
          if(g[i]==0) return 1;
return 0;
}
int gradpar() {
for(int i=1;i<=n;i++)
          if(g[i]% 2==1) return 0;
return 1;
}
void ciclu() {
int i,j,k=1,p,q,gasit;
c[1]=1;
do for(j=1,gasit=0;j<=n && !gasit;j++)
          if (a[c[k]][j]==1) {
             k=k+1;
             c[k]=j;
             a[c[k-1]][j]=0;
             a[j][c[k-1]]=0;
             g[j]- -;
             g[c[k-1]]- -;
             gasit=1; }
while(c[k]!=1);
while(k-1<m) {
          for(i=1,q=0;i<=k-1 && q==0;i++)
             if(g[c[i]]>0) {
```

```
                c1[1]=c[i];
                q=i; }
        p=1;
        do for(j=1,gasit=0;j<=n && !gasit;j++)
        if(a[c1[p]][j]==1) {
            p=p+1;
            c1[p]=j;
            a[c1[p-1]][j]=0;
            a[j][c1[p-1]]=0;
            g[j]- -;
            g[c1[p-1]]- -;
            gasit=1; }while(c1[p]!=c1[1]);
        for(j=k;j>=q;j- -)
            c[j+p-1]=c[j];
        for(j=1;j<=p-1;j++)
            c[j+q]=c1[j+1];
        k=k+p-1; }
}
void main() {
int eulerian,m,x,y,i;
clrscr();
cout<<"Număr de noduri ";
cin>>n;
cout<<"Număr de muchii ";
cin>>m;
for(i=1;i<=m;i++) {
        cout<<"Extremităţi muchie "<<i<<" ";
        cin>>x>>y;
        a[x][y]=1;
        a[y][x]=1;
}
grad();
eulerian=!(izolat()) && gradpar() && conex();
if(!eulerian)  cout<<"Graful nu este eulerian ";
        else {
            cout<<"Graful este eulerian"<<endl;
            ciclu();
            cout<<"Ciclul eulerian este ";
```

```
            for(int i=1;i<=m+1;i++)
               cout<<c[i]<<" ";
}
getch();
}
```

3.9 Drumuri şi circuite hamiltoniene

Problema 3.9.1 *Pentru un graf orientat dat prin arcele sale, se cere să se implementeze algoritmul lui Kaufmann-Malgrange de determinare a drumurilor hamiltoniene. Pentru graful dat cunoaştem numărul de vârfuri, numărul de arce, pentru fiecare arc cele două extremităţi, acestea se consideră litere.*

```
#include <iostream.h>
#include <ctype.h>
#include <string.h>
#include <conio.h>
typedef char string[20];
int m,n,i,j,k,t;
char x,y;
string a[10][10],b[10][10],c[10][10],p,l;
void main() {
clrscr();
cout<<"Număr de vârfuri "; cin>>n;
cout<<"Număr de arce "; cin>>m;
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
             strcpy(a[i][j],"0");
for(i=1;i<=m;i++) {
          cout<<"Extremităţi arc "<<i<<" ";
          cin>>x>>y;
          x=toupper(x);
          y=toupper(y);
          char *z;
          switch (y) {
             case 'A':z="A";break;
```

```
                case 'B':z="B";break;
                case 'C':z="C";break;
                case 'D':z="D";break;
                case 'E':z="E";break;
                case 'F':z="F";break;
                case 'G':z="G";break;
                case 'H':z="H";break;
                case 'I':z="I";break;
                case 'J':z="J";break;
            }
            strcpy(a[(int)x-64][(int)y-64],z);  }
for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                strcpy(b[i][j],a[i][j]);
for(t=2;t<=n-1;t++)  {
            for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)  {
                    strcpy(c[i][j],"0");
                    for(k=1;k<=n;k++)  {
                        int i1,j1,e=0;
                        for (i1=0;i1<strlen(a[i][k]);i1++)
                            for(j1=0;j1<strlen(b[k][j]);j1++)
                                if(a[i][k][i1]==b[k][j][j1]) e=1;
                                if((strcmp(a[i][k],"0")==0) ||
                                        (strcmp(b[k][j],"0")==0)||(e==1))
                                    strcpy(l,"0");
                                else {
                                    strcpy(l,"");
                                    strcpy(l,a[i][k]);
                                    strcat(l,b[k][j]);  }
                                if (strrchr(l,(char)(i+64))) strcpy(l,"0");
                                if(strcmp(l,"0")!=0) {
                                    if (strcmp(c[i][j],"0")==0) strcpy(c[i][j],"");
                                    if (strcmp(c[i][j],"")!=0) strcat(c[i][j],",");
                                    strcat(c[i][j],l);  }
                     }  }
                    for(i=1;i<=n;i++)
                        for(j=1;j<=n;j++)
```

```
                    strcpy(b[i][j],c[i][j]);  }
          for(i=1;i<=n;i++)
             for(j=1;j<=n;j++)  {
                strcpy(c[i][j],"0");
                for(k=1;k<=n;k++)  {
                   int i1,j1,e=0;
                   for (i1=0;i1<strlen(a[i][k]);i1++)
                      for(j1=0;j1<strlen(b[k][j]);j1++)
                         if(a[i][k][i1]==b[k][j][j1]) e=1;
                   if((strcmp(a[i][k],"0")==0) ||
                                 (strcmp(b[k][j],"0")==0)||(e==1))
                         strcpy(l,"0");
                   else  {
                      strcpy(l,"");
                      strcpy(l,a[i][k]);
                      strcat(l,b[k][j]); }
                   if(strcmp(l,"0")!=0) {
                      if (strcmp(c[i][j],"0")==0) strcpy(c[i][j],"");
                      if (strcmp(c[i][j],"")!=0) strcat(c[i][j],",");
                      strcat(c[i][j],l); }
} }
for(i=1;i<=n;i++)  {
          if(strcmp(c[i][i],"0")!=0) {
             switch(i) {
                case 1:cout<<"A";break;
                case 2:cout<<"B";break;
                case 3:cout<<"C";break;
                case 4:cout<<"D";break;
                case 5:cout<<"E";break;
                case 6:cout<<"F";break;
                case 7:cout<<"G";break;
                case 8:cout<<"H";break;
                case 9:cout<<"I";break;
                case 10:cout<<"J";break;  }
          cout<<c[i][i]<<endl;  }  }
getch();
}
```

Problema 3.9.2 *Folosind algortimul lui Foulkes, să se determine pentru un graf orientat toate drumurile hamiltoniene. Graful se găseşte în fişierul "foulkes.in"; pe primul rând se găsesc două numere naturale, primul, n, reprezintă numărul de vârfuri ale grafului, iar al doilea număr, m, stabileşte numărul de arce din graf. Pe următorul rând se găsesc m perechi de numere naturale ce reprezintă extremităţile arcelor.*

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
FILE *f;
int a[10][10], b[10][10], c[10][10],ind[10],i,j,k,n,m,detect,egal; int ad[10],tc;
struct {
        int v,t;  }dh[20],aux;
#define false 0
#define true !false
typedef int stiva[100];
int ev,as;
stiva st;
void init( ) {
        st[k]=0;
}
int succesor( ) {
        if (st[k]<n)  {
           st[k]=st[k]+1;
           return 1;
        }
else return 0;
}
int valid( ) {
    if ((k>1) && (dh[st[k]].t<dh[st[k-1]].t))
        return 0;
    if ((k>1) && (a[dh[st[k-1]].v][dh[st[k]].v]==0))
        return 0;
    for(i=1;i<=k-1;i++)
        if(dh[st[i]].v==dh[st[k]].v)
           return 0;
```

```
        return 1;
}
int solutie( )  {
        return k==n;
}
void tipar( )  {
        int i;
        for(i=1;i<=n;i++)
           cout<<dh[st[i]].v<<" ";
        cout<<endl;
}
void back( ) {
        k=1;
        init();
        while (k>0)  {
           as=1;
           ev=0;
           while (as && !ev) {
              as=succesor();
              if (as) ev=valid();
           }
           if (as)
              if(solutie()) tipar( );
              else  {
                 k++;
                 init();
              }
           else k- -;
        } }
void init1(void)  {
f=fopen("fulkes.in","r");
puts("");
fscanf(f,"%d%d",&n,&m);
for(i=1;i<=m;i++)´  {
        fscanf(f,"%d%d",&j,&k);
        a[j][k]=1;  }
for(i=1;i<=n;i++)
```

```
            a[i][i]=1;
}
void putere(void) {
int i,k,j;
do {
            for(i=1;i<=n;i++)
               for(j=1;j<=n;j++) {
                  c[i][j]=0;
                  for(k=1;k<=n;k++)
                     c[i][j]|=b[i][k]&b[k][j]; }
            egal=true;
            for(i=1;i<=n;i++)
               for(j=1;j<=n;j++)
                  if(c[i][j]!=b[i][j])
                     { egal=false;
                     b[i][j]=c[i][j];
                     }
}while(!egal);
}
void elim(int tc) {
detect=false;
for(i=1;i<=n;i++)
            if(ind[i]!=2)  {
               egal=true;
               for(j=1;j<=n;j++)
                  if(ind[j]!=2)
                     if(c[i][j]==0) {
                        egal=false;
                        break; }
               if(egal) {
                  ind[i]=1;
                  detect=true; }
            }
            if(!detect) {
               printf("Graful nu este conex"); exit(1); }
            for(j=1;j<=n;j++)
               if(ind[j]==1) {
                  egal=true;
```

```
                for(i=1;i<=n;i++)
                    if(ind[i]==0)
                        if(c[i][j]==1)
                            egal=false;
                if(egal) {
                    ind[j]=2;
                    ad[j]=tc; }
            }
puts("");
}
void main() {
int i,j;
clrscr();
init1();
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            b[i][j]=a[i][j];
tc=0;
do {
        detect=false;
        for(i=1;i<=n;i++)
            if(ind[i]!=2) {
                detect=true;
                putere();
                tc++;
                elim(tc); }
}while(!detect);
for(i=1;i<=n;i++) {
        dh[i].v=i;
        dh[i].t=ad[i]; }
for(i=1;i<n;i++)
        for(j=i+1;j<=n;j++)
            if(dh[i].t>dh[j].t) {
                aux=dh[i];
                dh[i]=dh[j];
                dh[j]=aux; }
back();
```

```
getch();
}
```

Problema 3.9.3 *Scrieţi un program care implementează algoritmul lui Chen de determinare a drumurilor hamiltoniene dintr-un graf orientat pentru care se cunoaşte numărul de vârfuri, numărul de arce şi pentru fiecare arc cele două extremităţi.*

```
#include <iostream.h>
#include <conio.h>
void main() {
int i,j,n,m,ok,k,x,y,s,e,v[20],w[20],b[20],a[20][20];
struct putere{
        int p,v;
}c[20],aux;
clrscr();
cout<<"Număr de vârfuri=";
cin>>n;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            a[i][j]=0;
cout<<"Număr de arce=";
cin>>m;
for(k=1;k<=m;k++) {
        cout<<"Extremităţi arc "<<k<<" ";
        cin>>x>>y;
        a[x][y]=1; }
k=1;
int ok1=1;
while((k<=n)&&(ok1==1)) {
        for(i=1;i<=n;i++)
            w[i]=a[k][i];
        for(j=1;j<=n;j++)
            if(j!=k)
                if(a[k][j]==1) {
                    for(i=1;i<=n;i++)
                        v[i]=a[j][i];
                    for(i=1;i<=n;i++)
```

```
                        if(v[i]>w[i]) b[i]=v[i];
                          else b[i]=w[i];
                    for(i=1;i<=n;i++)
                        w[i]=b[i]; };
ok1=0;
for(i=1;i<=n;i++)
        if(w[i]!=a[k][i]) ok1=1;
if(ok1==1) {
        for(i=1;i<=n;i++)
            a[k][i]=w[i];
            k=k; }
        else {
            for(i=1;i<=n;i++)
                a[k][i]=w[i];
            k++;
            ok1=1; }
}
e=0;
for(i=1;i<=n;i++)
        if(a[i][i]==1) e=1;
if(e==1) cout<<"Graful nu are circuite"<<endl;
        else cout<<"Graful are circuite"<<endl;
for(i=1;i<=n;i++) {
        c[i].p=0;
        c[i].v=i;
        for(j=1;j<=n;j++)
            if(a[i][j]==1) c[i].p++; }
s=0;
for(i=1;i<=n;i++)
        s=s+c[i].p;
if(s==(n*(n-1))/2) {
        for(i=1;i<n;i++)
            for(j=i+1;j<=n;j++)
                if(c[i].p<c[j].p) {
                    aux=c[i];
                    c[i]=c[j];
                    c[j]=aux; }
        cout<<"Drumul hamiltonian este:";
```

```
        for(i=1;i<=n;i++)
            cout<<c[i].v<<" "; }
        else cout<<"Graful nu are drum hamiltonian";
        getch();
}
```

Problema 3.9.4 *Se dă un graf neorientat şi o secvenţă de elemente. Să se verifice dacă secvenţa respectivă este ciclu hamiltonian.*

```
#include <iostream.h>
#include <conio.h>
void main() {
int a[20][20],s[20],n,m,i,j,k,ok;
clrscr();
cout<<"Număr de vârfuri=";
cin>>n;
for(i=1;i<n;i++)
        for(j=i+1;j<=n;j++) {
            cout<<"a["<<i<<","<<j<<"]=";
            cin>>a[i][j];
            a[j][i]=a[i][j];}
cout<<"Număr de elemente din secvenţă "; cin>>k;
for(i=1;i<=k;i++) {
        cout<<"s["<<i<<"]=";
        cin>>s[i];}
ok=1;
if(n+1!=k) ok=0;
if(ok) {
        if(s[1]!=s[k]) ok=0;
        if(ok) {
            for(i=1;i<=k-2;i++)
                for(j=i+1;j<=k-1;j++)
                    if(s[i]==s[j]) ok=0;
                    if(ok) {
                        for(i=1;i<=k-1;i++)
                            if(a[s[i]][s[i+1]]==0) ok=0;
                            if(!ok)
                                cout<<"Există noduri între care nu avem muchie";
```

```
                    else  cout<<"Secvenţa dată este ciclu hamiltonian";}
                else cout<<"Nodurile nu sunt distincte";   }
        else  cout<<"Extremităţile nu coincid";  }
else  cout<<"Insuficiente noduri";
getch();
}
```

3.10 Drumuri de valoare optimă

Problema 3.10.1 *Pentru un graf neorientat dat să se determine, folosind algoritmul lui Ford, drumurile de valoare minimă de la un vârf fixat la celelalte vârfuri ale grafului. Datele problemei se găsesc în fişierul "ford.in", fişierul fiind structurat astfel:*

- *pe primul rând numărul de vârfuri n şi numărul de muchii m din graf, separate prin spaţii;*
- *pe fiecare din următoarele m rânduri, câte un triplet de numere întregi, reprezentând extremităţile muchiei şi costul asociat ei;*
- *pe ultimul rând din fişier un număr întreg care reprezintă vârful iniţial al drumului.*

În implementarea algoritmului se va urmări extinderea unei mulţimi de vârfuri, iniţializată în acest caz cu toate vârfurile care au ca predecesor direct vârful de start.
Pentru fiecare vârf din mulţime vom stoca:

- $dist[i]$-distanţa de la vârful x_1 la varful i;
- $pred[i]$-predecesorul vârfului i în cadrul drumului de la x_1 la x_2;
- $atins[i]$-memorează dacă vârful i a fost vizitat.

Se vor testa toate muchiile care au capătul iniţial x în mulţimea de vârfuri selectate:

- dacă muchia are şi celelalt capăt, y, în mulţime, se verifică dacă distanţa până la y este mai mare decât suma dintre distanţa până la x şi costul muchiei. În caz afirmativ se actualizează distanţa până la y, se stochează x ca predecesor al lui y şi se reia ciclul. În caz negativ se trece la următoarea muchie.

- dacă celălalt capăt al muchiei, y, nu aparţine mulţimii vârfurilor selectate, se prelungeşte drumul generat cu vârful y marcându-l pe acesta "atins", al lui y, după care se reia ciclul.

Tipărirea drumurilor se va face folosind vectorul *pred* şi un vector auxiliar *drum*, pentru a evita afişarea în ordine inversă a vârfurilor componente. Numerotarea vârfurilor începe cu 0.

```
#include<stdio.h>
#include<conio.h>
int u[20][3],x0,dist[20],pred[20],atins[20],drum[20],m,n,i,j,k;
void main() {
FILE *f;
int w,x,y;
clrscr();
f=fopen("ford.in","r");
fscanf(f,"%d %d",&n,&m);
for(i=1;i<=m;i++)
fscanf(f,"%d %d %d",&u[i][1],&u[i][2],&u[i][0]);
fscanf(f,"%d",&x0);
fcolse(f);
for(i=1;i<=n;i++) atins[i]=0;
dist[x0]=0;
atins[x0]=1;
for(j=0,i=1;i<=m;i++)
if(u[i][1]==x0)

          drum[++j]=u[i][2];
          dist[drum[j]]=u[i][0];
          pred[drum[j]]=x0;
          atins[drum[j]]=1; }
i=1;
x=u[i][1];
y=u[i][2];
w=u[i][0];
while(i<=m) {
          while(i<=m) {
             if(atins[x]==1)
```

```
                if(atins[y]==1)
                   if(dist[y]<=dist[x]+w)i++;
                   else {
                      dist[y]=dist[x]+w;
                      pred[y]=x;
                      i=1;}
                else {
                   dist[y]=dist[x]+w;
                   pred[y]=x;
                   atins[y]=1;
                   i=1;}
             else i++;
          x=u[i][1];
          y=u[i][2];
          w=u[i][0];
} }
for(i=1;i<=n;i+)
          if(atins[i]==1) {
             printf("\n Costul până la vârful %d este:
                      %d\n Traseul:\n %d",i,dist[i],x0);
             k=0;
             j=i;
             while(pred[j]!=x0) drum[++k]=j=pred[j];
             for(;k;) printf("%d ",drum[k- -]);
             printf("%d",i"); }
getch();
}
```

Problema 3.10.2 *Fiind dat un graf valorizat să se determine drumurile de valoare minimă dintre două vârfuri date. Datele problemei se găsesc în fişierul "graf.in", pe primul rând se găsesc două numere naturale, n şi m, ce reprezintă numărul de vârfuri şi de arce din graful respectiv, pe următoarele m rânduri se găsesc triplete de numere, extremităţile unui arc şi costul asociat arcului. Pe ultimul rând din fişier se găsesc două numere naturale ce reprezintă cele două extremităţi ale drumurilor căutate. Pentru implementarea problemei se va folosi algoritmul Bellman-Kalaba.*

```
#include <stdlib.h>
#include <stdio.h>
```

```
#include <conio.h>
int v[20][20],l[20][20],v1[20],v2[20],x[20],p,s,m,n,s1;
void introd(void)  {
int i,j,r;
FILE *f;
f=fopen("graf.in","r");
fscanf(f,"%d %d",&n,&m);
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++) {
             v[i][j]=0;
             l[i][j]=32000; }
for(i=1;i<=m;i++) {
          fscanf(f,"%d %d %d",&p,&s,&r);
          v[p][s]=1;
          l[p][s]=r;}
fscanf(f,"%d %d",&p,&s);
fclose(f);
}
void prod( ) {
int i,j,min,b;
for(i=1;i<=n-1;i++)
          v1[i]=l[i][n];
v1[n]=0;
do {
          for(i=1;i<=n-1;i++) {
             min=32000;
             for(j=1;j<=n;j++)
                if(v1[j]+l[i][j]<min)
                   min=v1[j]+l[i][j];
             v2[i]=min; }
          v2[n]=0;
          for(i=1,b=0;i<=n && b==0;i++)
             if(v2[i]!=v1[i]) {
                b=1;
                for(j=1;j<n;j++)
                   v1[j]=v2[j];}
}while (b);
}
```

```
void tiparire(int k) {
int i;
printf("Drum minim de la %d la %d ",p,s);
for(i=1;i<k;i++)
          printf("%d ",x[i]);
    printf("\n");
}
void drummin(int k)  {
int i;
if(x[k-1]==s) {
          for(i=2,s1=0;i<=k-1;i++)
             s1+=l[x[i-1]][x[i]];
          if(s1==v2[1])
             tiparire(k); }
          else for(i=1;i<=n;i++)  {
             x[k]=i;
             if(v2[i]<v2[x[k-1]] && v[x[k-1]][i]==1)
                drummin(k+1);  }
}
void main() {
clrscr();
introd();
prod();
printf("\n Costul este %d \n",v2[1]);
x[p]=1;
drummin(2);
getch();
}
```

Problema 3.10.3 *Pentru un graf orientat dat să se determine, folosind algoritmul lui Dijkstra, drumurile de valoare minimă de la un vârf fixat la celelalte vârfuri ale grafului. Datele problemei se găsesc în fişierul "dijkstra.in", fişierul fiind structurat astfel:*

- *pe primul rând numărul de vârfuri n ;*
- *pe fiecare din următoarele n rânduri, câte n numere întregi, reprezentând valorile asociate fiecărui arc din graful respectiv;*

- *pe ultimul rând din fişier un număr întreg care reprezintă vârful iniţial al drumului.*

```
#include <stdio.h>
#include <conio.h>
#include <mem.h>
int a[20][20],d[20],i,j,k,n,x,y,min,imin;
char c[20],varf[20];
void main( ) {
FILE *f;
f=fopen("dijkstra.in","r");
if (f==NULL) {
          printf("Eroare la deschidere ");
          return; }
fscanf(f,"%d",&n);
for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
             fscanf(f,"%d",&a[i][j]);
    fscanf(f,"%d",&x);
memset(d,0,sizeof(d));
memset(varf,0,sizeof(varf));
for(i=1;i<=n;i++) {
          c[i]=1;
          if(a[x][i]) varf[i]=x;
}
c[x]=0;
for(i=1;i<=n;i++)
          if(c[i]) d[i]=a[x][i];
for(y=1;y<=n;y++)
          if(x!=y) {
             for(k=1;k<=n-2;k++) {
                min=10000;
                for(i=1;i<=n;i++)
                   if(c[i]&& d[i]>0 && d[i]<min) {
                      min=d[i];
                      imin=i; }
                if (imin==y) break;
                c[imin]=0;
```

```
                for(i=1;i<=n;i++)
                    if(c[i] && d[imin]!=0 && a[imin][i]!=0)
                        if(d[i]==0 || d[i]>d[imin]+a[imin][i]) {
                            d[i]=d[imin]+a[imin][i];
                            varf[i]=imin;
                        }
                    }
                    printf("\n Distanţa minimă între %d şi %d este:
                                %d \n",x,y,d[y]);
                    c[i=0]=y;
                    while(varf[i]!=x && varf[c[i]])
                        c[++i]=varf[c[i-1]];
                    if(c[i]!=x) c[++i]=x;
                    for(;i>=0;)
                        printf("%d ",c[i- -]);
            }
getch();
}
```

Problema 3.10.4 *Pentru un graf orientat dat să se determine, folosind algoritmul lui Floyd-Warshall, drumurile de valoare minimă dintre oricare două vârfuri ale grafului. Se citesc de la tastatură numărul de vârfuri, numărul de arce şi pentru fiecare arc cele două extremităţi şi costul asociat ei.*

```
#include <iostream.h>
#include <conio.h>
int a[20][20],t[20][20],c[20][20],drum[20][20],m,n,i,j,k;
void floyd( ) {
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) {
            t[i][j]=c[i][j];
            drum[i][j]=0; }
for(i=1;i<=n;i++)
        t[i][i]=0;
for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
```

```
                for(j=1;j<=n;j++)
                    if(t[i][k]+t[k][j]<t[i][j]) {
                        t[i][j]=t[i][k]+t[k][j];
                        drum[i][j]=k; }
}
void traseu(int i,int j) {
int k=drum[i][j];
if(k!=0) {
            traseu(i,k);
            cout<<k<<" ";
            traseu(k,j); }
}
void main() {
int cost;
clrscr();
cout<<"Introduceţi numărul de vârfuri ";
cin>>n;
cout<<"Introduceţi numărul de arce ";
cin>>m;
for(i=1;i<=n;i++)
            for(j=1;j<=n;j++) {
                a[i][j]=0;
                c[i][j]=9999; }
for(i=1;i<=m;i++) {
            cout<<"Extremităţi arc "<<i<<": ";
            cin>>j>>k;
            a[j][k]=1;
            cout<<"Costul arcului ";
            cin>>cost;
            c[j][k]=cost; }
floyd();
for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(i!=j)
                    if(t[i][j]==9999)
                        cout<<endl<<"Nu există drum între "<<i<<" şi "<<j;
                    else {
```

```
                cout<<endl<<"Costul drumului minim între "<<i<<" şi
"<<j<<" este "<<t[i][j];
                cout<<endl<<"Traseul "<<i<<" ";
                traseu(i,j);
                cout<<j<<" "; }
getch();
}
```

Problema 3.10.5 *Să se determine circuitul hamiltonian de cost minim, pentru un graf orientat dat. Datele problemei se găsesc în fişierul "graf.in", fişierul fiind structurat astfel:*

- *pe primul rând numărul de vârfuri n şi numărul de muchii m din graf, separate prin spaţii;*
- *pe fiecare din următoarele m rânduri, câte un triplet de numere întregi, reprezentând extremităţile muchiei şi costul asociat ei;*

Se va folosi o funcţie recursivă de generare a circuitelor. Dacă se găseşte un circuit, se verifică dacă este hamiltonian. Dacă da, se determină costul său. În cazul în care costul său este mai mic decât costul minim obţinut până acum, se stochează traseul şi costul aferent. După generarea tuturor circuitelor posibile, se afişează circuitul de cost minim obţinut. La început vom iniţializa costul minim cu o valoare suficient de mare. Dacă valoarea de minim rămâne neschimbată înseamnă că nu s-au găsit circuite hamiltoniene.

```
#include <mem.h>
#include <stdio.h>
#include <conio.h>
int n,m,a[20][20],u[50][2],i,j,k,sel[20],c[20],cmin[20],l=0,cost[50],min,cos;
void introducere(void);
void circuit(int i);
void main() {
clrscr();
min=10000;
introducere();
c[0]=1;
circuit(1);
if(min==10000) {
```

```
            puts("Nu există circuit hamiltonian");
            return; }
printf("\n Minimul este %d\n",min);
for(i=0;i<=n;i++)
            printf("%d",cmin[i]);
}
void introducere() {
FILE*f;
f=fopen("graf.in","r");
fscanf(f,"%d %d",&n,&m);
for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                a[i][j]=0;
for(i=1;i<=m;i++)  {
            fscanf(f,"%d %d %d",&u[i][0],&u[i][1], &cost[i]);
            a[u[i][0]][u[i][1]]=cost[i]; }
}
void circuit(int i) {
int j;
for(j=c[0];j<=n;j++)
            if((sel[j]==0) && (a[i][j])) {
                l++;
                c[l]=j;
                sel[j]=1;
                if((c[0]==c[l])&& (l==n)) {
                    cos=0;
                    for(k=0;k<l;k++)
                        cos+=a[k][k+1];
                    if(cos<min) {
                        min=cos;
                        memcpy(cmin,c,sizeof(c));
                } }
                else circuit(j);
            l- -;
            sel[j]=0; }
getch();
}
```

Problema 3.10.6 *O societate comercială prezintă un nou tip de telefon celular în mai multe oraşe din ţară, în cadrul unei expoziţii. Deplasarea între aceste oraşe se poate face cu trenul, fie direct, fie trecând prin alte oraşe, unde se schimbă trenul. Ştiind costurile călătoriilor directe între două oraşe, se cere să se determine traseul cel mai ieftin care permite unui agent comercial să se deplaseze de la sediu în toate oraşele din ţară, întorcându-se apoi la sediu, dar fără a trece de două ori prin acelaşi oraş.*

```
#include<iostream.h>
#include<conio.h>
int k,n,i,j,cont,c[20][20],x[20],y[20];
char nume[20][20]; int  costcurent, costminim;
int potcontinua()  {
if(c[x[k]][x[k-1]]==10000) return 0;
if(k==n)
    if(c[x[n]][x[1]]==10000) return 0;
for(int i=1;i<k;i++)
    if(x[i]==x[k]) return 0;
return 1;
}
void main() {
clrscr();
cout<<"Circuit hamiltonian de cost minim"<<endl;
cout<<"Număr de oraşe ";
cin>>n;
for(i=1;i<=n;i++) {
    cout<<"Nume oraş "<<i<<" ";
    cin>>nume[i]; }
for(i=1;i<=n-1;i++)
    for(j=i+1;j<=n;j++) {
    cout<<"Cost drum de la "<<nume[i];
    cout<<" la "<<nume[j]<<" 0-infinit :";
    cin>>c[i][j];
    if(c[i][j]==0)
    c[i][j]=10000;
    c[j][i]=c[i][j]; }
x[1]=1;
k=2;
```

```
x[k]=1;
costminim=10000;
while(k>1) {
    cont=0;
    while((x[k]<n) && (!cont)) {
    x[k]++;
    cont=potcontinua(); }
    if(cont)
    if(k==n) {
    costcurent=0;
    for(i=1;i<n;i++)
    costcurent+=c[x[i]][x[i+1]];
    costcurent+=c[x[n]][x[1]];
    if(costcurent<costminim) {
    costminim=costcurent;
    for(i=1;i<=n;i++)
    y[i]=x[i]; }
    }
    else x[++k]=1;
    else - -k;
}
cout<<"Circuit de cost minim "<<endl;
for(i=1;i<n;i++)
    cout<<nume[y[i]]<<" ";
cout<<nume[y[1]]<<" ";
cout<<"Costul este "<<costminim;
getch();
}
```

3.11 Arbore parţial de cost minim

Problema 3.11.1 *Să se determine, folosind algoritmul lui Kruskal, arborele parţial de cost minim asociat unui graf. Se cunosc: numărul de vârfuri, numărul de muchii şi pentru fiecare muchie extremităţile şi costul asociat ei. Datele se citesc de la tastatură.*

```
#include <iostream.h>
#include <conio.h>
struct muchie {
        int x,y,c; }  v[20];
int a1,a2,m,n,i,j,cost,k,l,nr,t[20];
muchie aux;
void main() {
clrscr();
cout<<"Număr de vârfuri ";
cin>>n;
cout<<"Număr de muchii ";
cin>>m;
for(i=1;i<=m;i++) {
        cout<<"Extremităţi muchie "<<i<<":";
        cin>>v[i].x>>v[i].y;
        cout<<"Cost muchie ";
        cin>>v[i].c; }
cout<<"Arborele de cost minim este "<<endl;
for(i=1;i<=m-1;i++)
        for(j=i+1;j<=m;j++)
           if(v[i].c>v[j].c) {
              aux=v[i];
              v[i]=v[j];
              v[j]=aux; }
for(i=1;i<=n;i++)
        t[i]=i;
cost=0;
i=1;
nr=0;
while(nr<n-1) {
        if(t[v[i].x]!=t[v[i].y]) {
           nr++;
           cost=cost+v[i].c;
           cout<<v[i].x<<" "<<v[i].y<<endl;
           k=t[v[i].x];
           l=t[v[i].y];
           for(j=1;j<=n;j++)
              if(t[j]==k) t[j]=l; }
```

```
            i++;
}
cout<<"Cost "<<cost;
getch();
}
```

Problema 3.11.2 *Să se determine, folosind algoritmul lui Prim, arborele parţial de cost minim asociat unui graf. Se cunosc: numărul de vârfuri, numărul de muchii şi pentru fiecare muchie extremităţile şi costul asociat ei. Datele se citesc de la tastatură.*

```
#include <iostream.h>
#include <conio.h>
void main( ) {
int a[20][20],s[20],t[20],c[20],n,cost,i,j,k,n1,n2,start,costm;
clrscr();
cout<<"Număr de vârfuri";
cin>>n;
cout<<" Daţi 3200 dacă nu există muchie"<<endl;
for(i=1;i<=n;i++)
        a[i][i]=0;
for(i=1;i<n;i++)
        for(j=i+1;j<=n;j++) {
            cout<<"Cost între "<<i<<" şi "<<j<<" ";
            cin>>a[i][j];
            a[j][i]=a[i][j]; }
cout<<"Arborele parţial de cost minim"<<endl;
for(i=1;i<=n;i++)
        s[i]=t[i]=c[i]=0;
start=1;
s[start]=1;
for(k=1;k<=n-1;k++) {
        costm=32000;
        n1=-1;
        n2=-1;
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if((s[i]==1) && (s[j]==0))
```

```
                    if(a[i][j]<costm) {
                       costm=a[i][j];
                       n1=i;
                       n2=j; }
                    s[n2]=1;
                    t[n2]=n1;
                    c[n2]=a[n1][n2];
}
for(i=2;i<=n;i++)
          cout<<t[i]<<" "<<i<<endl;
cost=0;
for(i=1;i<=n;i++)
          cost+=c[i];
cout<<"Cost minim "<<cost;
getch();
}
```

Problema 3.11.3 *Să se scrie un program care determină toţi arborii parţiali de cost minim ai unui graf. Datele se găsesc în fişierul "graf.txt".*

Pentru rezolvarea problemei vom verifica mai întâi dacă graful iniţial este conex, în caz negativ neexistând niciun arbore parţial. Apoi vom genera toate submulţimile de muchii ale grafului iniţial care pot forma un arbore şi vom calcula costul arborelui găsit. Dacă acesta este mai mic decât costul minim curent se va actualiza numărul de arbori parţiali de cost minim la 0 iar costul minim va lua valoarea costului curent. Dacă costul curent este egal cu costul minim se incrementează numărul de arbori şi se stochează arborele găsit.

```
#include <stdio.h>
#include <conio.h>
int m,n,i,j,k,cost,costmin=32767;
int a[50][50],c[50][50],l[50][50],u[50][2];
int x[50],arbori[50][50],sel[50],n1,m1,nrarb=0;
void introd(void) {
FILE*f;
int cost;
f=fopen("graf.txt","r");
```

```
fscanf(f,"%d %d",&n,&m);
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            a[i][j]=c[i][j]=0;
for(i=1;i<=m;i++){
        fscanf(f,"%d %d %d",&j,&k,&cost);
        u[i][0]=j;
        u[i][1]=k;
        a[j][k]=a[k][j]=1;
        c[j][k]=c[k][j]=cost;
}
fclose(f);
}
void sub(int mult[50]) {
int i=1;
while (mult[i]==1) mult[i++]=0;
mult[i]=1;
}
void lanturi(int a[][50],int l[][50]){
int i,j,k;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        l[i][j]=a[i][j];
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if (l[i][j]==1)
                for(k=1;k<=n;k++)
                    if (l[k][j]==1)
                        l[i][k]=l[k][i]=1;
int conex(int l[][50]){
int i,j;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if (l[i][j]==0) return 0;
return 1;
}
void main(void){
int i,k,j;
```

```
introd( );
lanturi(a,l);
if (!conex(l)) {
        printf("Graful nu este conex");
        return; }
for(k=1;k<=(1<<m);k++){
        sub(sel);
        m1=0;
        for(i=1;i<=n;i++)
           for(j=1;j<=n;j++)
              a[i][j]=0;
        for(i=1;i<=m;i++)
           if(sel[i]==1){
              a[u[i][0]][u[i][1]]=a[u[i][1]][u[i][0]]=1;
              x[u[i][0]]=x[u[i][1]]=1;
              m1++;
           }
        n1=0;
        for(i=1;i<=n;i++)
           if (x[i]==1) n1++;
        lanturi(a,l);
        if((n1-1==m1)&&(conex(l)==1)){
           cost=0;
           for(i=1;i<=m;i++)
              if(sel[i]==1) cost+=c[u[i][0]][u[i][1]];
           if(cost<costmin){
              nrarb=0;
              costmin=cost; }
        if(cost==costmin){
           nrarb++;
           for(i=1;i<=m;i++)
              arbori[nrarb][i]=sel[i];}
        } }
printf("Costul minim este %d sunt %d arbori",costmin,nrarb);
for(i=1;i<=nrarb;i++){
        for(j=1;j<=m;j++)
           if (arbori[i][j]==1)
```

```
                printf("%d %d",u[j][0],u[j][1]);
printf("\n");  }
getch();
}
```

Problema 3.11.4 *Să se determine dacă un graf neorientat este sau nu ciclic.*

Problema are o rezolvare bazată pe algoritmul lui Kruskal de obţinere a arborelui parţial de cost minim dintr-un graf.
Se citesc muchiile grafului şi se repartizează în componente conexe, în funcţie de capetele lor. Dacă la un moment dat s-a citit o muchie care are ambele capete în aceeaşi componentă conexă, înseamnă că s-a întâlnit un ciclu deoarece între capetele muchiei există deja un lanţ (altfel nu ar fi făcut parte din aceeaşi componentă conexă).
Se observă că nu este nevoie de stocarea muchiilor şi nici a matricii de adiacenţă, vectorul care conţine pe poziţia i componenta conexă din care face parte vârful i fiind suficient. Acest vector se iniţializează cu 0, după care fiecare muchie va fi tratată distinct, în funcţie de componentele conexe în care se află capetele sale:

- dacă ambele capete ale muchiei au componenta 0 atunci se va incrementa numărul de componente nr, iar cele două vârfuri vor fi în noua componentă (cu indicele nr);
- dacă exact unul din capete are componenta 0, el va fi adăugat la componenta conexă a celuilalt vârf;
- dacă ambele capete se află în aceeaşi componentă, am obţinut un ciclu;
- dacă ambele capete se află în componente conexe diferite atunci muchia citită uneşte cele două componente, deci se va obţine o singură componentă conexă.

```
#include <stdio.h>
#include <conio.h>
void main(){
int i,j,k,l,m,n,nr=0,con[100];
FILE *f=fopen("aciclic.in","r");
fscanf(f,"%d %d",&n,&m);
for(i=1;i<=n;i++)
```

```
        con[i]=0;
for(i=0;i<m;i++){
        fscanf(f,"%d %d",&j,&k);
        if(con[j])
           if(con[k]){
              if(con[j]==con[k]){
                 printf("Ciclic, muchia [%d,%d]",j,k);
                 return;}
              k=con[k];
              j=con[j];
              for(l=1;l<=n;l++)
                 if(con[l]==k) con[l]=j;
           }
           else con[k]=con[j];
        else
           if(con[k]) con[j]=con[k];
           else con[j]=con[k]=++nr;
}
puts("Graful nu are cicluri");
getch();
}
```

Problema 3.11.5 *Edilii unui judeţ vor să refacă o reţea de drumuri care să asigure legătura între oraşul reşedinţă de judeţ şi cele n localităţi ale judeţului precum şi între cele n localităţi. Se cunosc distanţele între oricare două localităţi din judeţ. Să se afişeze perechile de localităţi între care se vor reface drumurile astfel încât să existe drum între oricare două localităţi, iar lungimea totală a drumului refăcut să fie minimă. (Se va folosi algoritmul lui Prim).*

```
#include <iostream.h>
#include <conio.h>
void main() {
int a[20][20],s[20],t[20],c[20],n,cost,i,j,k,n1,n2,start,costm;
clrscr();
cout<<"Număr de oraşe ";
cin>>n;
for(i=1;i<=n;i++)
```

```
            a[i][i]=0;
for(i=1;i<n;i++)
                for(j=i+1;j<=n;j++){
                    cout<<"Cost între "<<i<<" şi "<<j;
                    cin>>a[i][j];
                    a[j][i]=a[i][j]; }
for(i=1;i<=n;i++)
            s[i]=t[i]=c[i]=0;
start=1;
s[start]=1;
for(k=1;k<=n-1;k++)  {
            costm=32767;
            n1=-1;
            n2=-1;
            for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                    if((s[i]==1)&&(s[j]==0))
                        if(a[i][j]<costm)  {
                            costm=a[i][j];
                            n1=i;
                            n2=j;  }
            s[n2]=1;
            t[n2]=n1;
            c[n2]=a[n1][n2];  }
for(i=2;i<=n;i++)
            cout<<t[i]<<"  "<<i<<endl;
cost=0;
for(i=1;i<=n;i++)
            cost+=c[i];
cout<<"Cost  minim  "<<cost;
gecth();
}
```

3.12 Problema fluxului maxim

Problema 3.12.1 *Să se scrie un program care implementează algoritmul Ford-Fulkerson.*

Programul are definite următoarele funcţii:

- *intr*, pentru citirea datelor din fişierul "flux.in"; pe primul rând se găsesc două numere naturale care reprezintă numărul de vârfuri respectiv numărul de arce, iar pe următorul rând numere naturale, câte trei pentru fiecare arc (extremităţile arcului şi costul asociat arcului respectiv);
- *matrice*, determinarea matricii de adiacenţă;
- *verif*, verifică dacă graful este sau nu reţea, face determinarea gradelor şi iniţializarea fluxului cu 0;
- *ver*, verifică dacă fluxul ales este valid pentru arcul dat;
- *flux*, generează lanţurile nesaturate;
- *prel*, calculează valoarea e şi actualizează fluxul arcelor cu această valoare;
- *detcap*, funcţia de determinare a fluxului maxim.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int m,n;
typedef int mat[20][20];
typedef int vect[20];
typedef enum bool{false,true}bool;
mat c,a,l,f,f1;
vect di,de,l1;
int u[100][2],ei,ef;
bool gasit,sel[20];
int a1,b1,min;
void intr(void);
void matrice(void);
void verif(void);
bool ver(int,int);
void flux(int i1);
void prel(int);
```

```
void detcap(void);
void intr() {
FILE*f;int i,j;
f=fopen("flux.in","r");
fscanf(f,"%d %d",&n,&m);
for(i=1;i<=m;i++){
        fscanf(f,"%d %d",u[i],u[i]+1,&c[u[i][0],u[i][1]]);
        fscanf(f,"%d",&j); c[u[i][0][u[i][1]]=j; }
fclose(f);
}
void matrice(void) {
int i,j,k;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
           a[i][j]=0;
for(i=1;i<=m;i++)
        a[u[i][0]][u[i][1]]=1;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
           l[i][j]=a[i][j];
for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
           if(l[i][k])
              for(j=1;j<=n;j++)
                 if(l[i][j]<l[k][j]) l[i][j]=1;
}
void verif() {
int i,j;
for(i=1;i<=n;i++)
        di[i]=de[i]=0;
for(i=1;i<=m;i++) {
        de[u[i][0]]++;
        di[u[i][1]]++;}
a1=b1=0;
for(i=1;i<=n;i++)
        if(de[i]==0) b1=i;
for(i=1;i<=n;i++)
        if(di[i]==0) {
```

```
            a1=i;
            break;}
if(a1==0 || b1==0) {
        printf("Nu este reţea");
        exit(1); }
for(i=1;i<=n;i++)
        printf("Vârful %d are gradul intern %d şi gradul extern:
                %d\n",i,di[i],de[i]);
printf("Graful este reţea \n");
printf("Vârf iniţial %d vârf final %d \n",a1,b1);
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            f[i][j]=0;
}
void prel(int nr) {
int i,i1,j1;
min=255;
for(i=1;i<nr;i++) {
        i1=l1[i];
        j1=l1[i+1];
        if(a[j1][i1] && f[j1][i1]<min)
            min=f[j1][i1];
    if(a[i1][j1] && c[i1][j1]-f[i1][j1]<min)
        min=c[i1][j1]-f[i1][j1]; }
for(i=1;i<nr;i++) {
        i1=l1[i];
        j1=l1[i+1];
        if(a[i1][j1]==1)
            f[i1][j1]+=min;
        else f[i1][j1]-=min; }
}
bool ver(int i,int j) {
if(a[l1[i]][j]+a[j][l1[i]]==0) return false;
if(a[l1[i]][j]==1)
        return c[l1[i]][j]>f[l1[i]][j];
else return f[l1[i]][j]>0;
}
void flux(int i) {
```

```
int j;
for(j=1;!gasit && j<=n;j++)
        if(ver(i-1,j) && sel[j]==false) {
            l1[i]=j;
            sel[j]=true;
            if(j==b1) {
                prel(i);
                gasit=true; }
            else if(i<n) flux(i+1);
        sel[j]=false; }
}
void detcap(void) {
int i,flux1;
flux1=0;
l1[1]=a1;
do {
        gasit=false;
        for(i=1;i<=n;i++)
            sel[i]=false;
        sel[a1]=true;
        flux(2);
        if(gasit) flux1+=min;
}while(gasit);
for(i=1;i<=m;i++)
printf("Arcul (%d%d) capacitatea: %d fluxul: %d \n",
u[i][0],u[i][1],c[u[i][0]][u[i][1]],f[u[i][0]][u[i][1]]);
printf("Fluxul este: %d \n",flux1);
}
void main() {
clrscr();
intr();
matrice();
verif();
detcap();
getch();
}
```

3.13 Probleme de afectare

Problema 3.13.1 *Într-un proces de producţie există n maşini pe care pot fi executate n lucrări. Timpii de execuţie ai fiecărei lucrări pe oricare din cele n maşini se memorează într-o matrice de dimensiune $n \times n$. Să se scrie un program care va repartiza lucrările pe maşini astfel încât timpul total de execuţie să fie minim. Pentru rezolvarea problemei se va implementa algoritmul ungar.*

```
#include <iostream.h>
#include <conio.h>
int a[20][20],d[20],v[20],s,li0,i0,c0,j,k,n,R,c,i,u0,j0;
void calcul( ) {
d[1]=1;
v[1]=1;
s=a[1][1];
for(k=2;k<=n;k++) {
        li0=a[1][k]+a[k][d[1]]-a[1][d[1]];
        i0=1;
        c0=d[i0];
        j=d[i0];
        for(i=1;i<k;i++) {
           j=d[i];
           R=a[i][k]+a[k][j]-a[i][j];
           if(R<li0) {
              li0=R;
              i0=i;
              j=d[i];
              c0=j;
              j0=j; }
           for(c=1;c<k;c++)
              if(c-j) {
                 int u = v[c];
                 R=a[i][k]+a[k][c]-a[i][j]-a[u][c]+a[u][j];
                 if(R<li0) {
                    li0=R;
                    c0=c;
                    i0=i;
```

```
                    u0=u;
                    j0=j; }
                }
            }
            if(li0>= a[k][k]) {
                s+=a[k][k];
                d[k]=k;
                v[k]=k; }
            else {
                s+=li0;
                if(c0==j0) {
                    d[i0]=k;
                    d[k]=c0;
                    v[c0]=k;
                    v[k]=i0; }
                else {
                    d[i0]=k;
                    v[k]=i0;
                    d[u0]=j0;
                    v[j0]=u0;
                    d[k]=c0;
                    v[c0]=k; }
        }
}
cout<<" S= "<<s<<endl;
for(i=1;i<=n;i++)
        cout<<i<<"->"<<d[i]<<" cost "<<a[i][d[i]]<<" "<<endl;
}
void main( ) {
clrscr();
cout<<"Număr de maşini şi lucrări ";
cin>>n;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) {
            cout<<"a["<<i<<","<<j<<"]=";
            cin>>a[i][j]; }
calcul();
```

```
getch();
}
```

3.14 Probleme de ordonanţare

Problema 3.14.1 *Fiind dat un graf orientat cu n vârfuri şi m arce, fiecare arc având asociat un cost, se cere, să se scrie un program care verifică dacă graful respectiv poate fi un graf de activităţi, care sunt evenimentele, drumurile şi activităţile critice din graf. Datele se găsesc în fişierul "grafact.in". Fişierul este structurat astfel: pe primul rând se găsesc două numere naturale care reprezintă numărul de vârfuri respectiv numărul de arce, iar pe următorul rând numere naturale, câte trei pentru fiecare arc (extremităţile arcului şi costul asociat arcului respectiv).*

```
#include<conio.h>
#include<stdio.h>
int i,n,m,u[30[2],a[30][30],l[30][30],di[30],de[30];
int ei,ef,cost[30],ti[30],tf[30],nr,ev[20],t[20][20];
void matricea(void);
void matricel(void);
void grade(void);
void introgract(void);
verif(void);
void adaugare(void);
void determinare(void);
void introgract() {
FILE*f;
memset(t,0,sizeof(t));
f=fopen("grafact.in","r");
fscanf(f,"%d %d",&n,&m);
for(i=1;i<=m;i++) {
          fscanf(f,"%d %d %d",&u[i][0],&u[i][1],&cost[i]);
          t[u[i][0]][u[i][1]]=cost[i]; }
fclose(f);
}
void determinare() {
int j,k,x[30],l1;
```

```
memset(ti,0,sizeof(ti));
for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(a[j][i]==1 && ti[i]<ti[j]+t[j][i])
                    ti[i]=ti[j]+t[j][i];
puts("Timpii iniţiali ");
for(k=1;k<=n;k++)
        printf("%d ",ti[k]);
for(k=1;k<=n;k++)
        tf[i]=ti[ef];
for(k=1;k<=n;k++)
        for(i=1;i<=n;i++) {
            tf[i]=ti[ef];
            for(j=1;j<=n;j++)
                if(a[i][j]==1 && tf[i]>tf[j]-t[i][j])
                    tf[i]=tf[j]-t[i][j]; }
puts("\n Timpii finali ");
for(k=1;k<=n;k++)
        printf("%d ",tf[k]);
putchar('\n');
for(i=1;i<=n;i++)
        if(ti[i]==tf[i]) ev[i]=1;
            else ev[i]=0;
puts("Evenimente critice ");
for(k=1;k<=n;k++)
        if(ev[k]) printf("%d ",k);
putchar('\n');
memset(x,0,sizeof(x));
puts("Drumuri critice ");
x[1]=ei;
k=2;
while(k>1)
        if(x[k]<n) {
            x[k]++;
            if(ev[x[k]] && a[x[k-1]][x[k]]
             && ti[x[k-1]]+t[x[k-1]][x[k]]==ti[x[k]]) {
            if(x[k]==ef) {
```

```
                for(l1=1;l1<=k;l1++)
                    printf("%d ",x[l1]);
                putchar('\n'); }
            x[++k]=0; } }
            else k- -;
}
verif() {
int i,j;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(l[i][j]==1 && l[j][i]==1 && i!=j) return 0;
return 1;
}
void adaugare() {
for(i=1;i<=n;i++)
        if(di[i]==0) {
            ei=i;
            break; }
for(i=ei+1;i<=n;i++)
        if(di[i]==0) {
            u[++m][0]=ei;
            u[m][1]=i; }
for(i=1;i<=n;i++)
        if(de[i]==0) {
            ef=i;
            break; }
for(i=ef+1;i<=n;i++)
        if(de[i]==0) {
            u[++m][0]=i;
            u[m][1]=ef; }
}
void grade() {
int i;
for(i=1;i<=m;i++) {
        di[u[i][1]]++;
        de[u[i][0]]++; }
}
void matricea() {
```

```
int i;
memset(a,0,sizeof(a));memset(l,0,sizeof(l));
for(i=1;i<=m;i++)
        a[u[i][0]][u[i][1]]=l[u[i][0]][u[i][1]]=1;
}
void matricel() {
int i,j,k;
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
           if(l[i][j]==1)
              for(k=1;k<=n;k++)
                 if(l[j][k]==1) l[i][k]=1;
}
void main() {
clrscr();
introdgract();
grade();
matricea();
matricel();
if(!verif()) {
        puts("Graful nu este graf de activităţi");
        return; }
adaugare();
determinare();
getch();
}
```

Problema 3.14.2 *Se dau n evenimente. Între aceste evenimente existând m activităţi, pentru fiecare activitate se cunoaşte costul. Să se determine activităţile critice. Datele se citesc de la tastatură.*

```
#include<iostream.h>
#include<conio.h>
float l[20][20],t[20],tb[20];
int n,i,j;
void citire() {
float c; int i,m,x,y;
clrscr( );
```

```
cout<<"Număr de activităţi "; cin>>m;
for(i=1;i<=m;i++) {
          cout<<"Arc asociat activităţii "<<i<<" şi durata ei ";
          cin>>x>>y>>c;
          l[x][y]=c; }
}
void calct(int i) {
int j;
float max;
if(i<2) t[1]=0;
else {
          max=0;
          for(j=1;j<=n;j++)
             if(l[j][i]>=0) {
                if(t[j]<0) calct(j);
                if(max<l[j][i]+t[j]) max=l[j][i]+t[j]; }
          t[i]=max; }
}
void calctb(int i) {
int j;
float min;
if(i==n) tb[i]=t[i];
       else {
          min=10000;
          for(j=1;j<=n;j++)
             if(l[i][j]>=0) {
                if(tb[j]<0) calctb(j);
                if(min>tb[j]-l[i][j]) min=tb[j]-l[i][j]; }
          tb[i]=min;
       }
}
void main() {
cout<<"Număr de evenimente ";
cin>>n;
for(i=1;i<=n;i++) {
          t[i]=-1;
          tb[i]=-1;
          for(j=1;j<=n;j++)
```

```
            l[i][j]=-1; }
citire();
calct(n);
calctb(1);
cout<<"Evenimente critice ";
for(i=1;i<=n;i++)
          if(t[i]==tb[i]) cout<<i<<" ";
cout<<endl;
getch();
}
```

Observaţia 3.14.3 *Rezolvarea problemelor de ordonanţare este posibilă cu ajutorul mediului Microsoft Excel.*

Exemplul 3.14.4 *Fie un proiect care este alcătuit din următoarele activităţi.*

Activitatea	*Activităţi precedente*	*Durata de realizare*
1	-	*5*
2	*3*	*2*
3	-	*4*
4	*1,2*	*3*
5	*2,3*	*6*

Foaia de calcul care conţine acest model este prezentată mai jos.

	A	B	C	D	E	F	G	H
1	Ativitate	Durata	DI	DT	TI	TT	Abatere	Critica
2	0	0	0	0	-5	-5	-5	NU
3	1	5	0	5	-5	0	-5	NU
4	2	2	4	6	4	6	0	DA
5	3	4	0	4	0	4	0	DA
6	4	3	6	9	9	12	3	NU
7	5	6	6	12	6	12	0	DA
8	6	0	12	12	12	12	0	DA
9								
10		Lung. max.		12				

Datele şi formulele introduse sunt cele rezultate prin dezvoltarea grafului ataşat proiectului. Formulele utilizate în foaia de calcul sunt:

Celula	Formula	Se copiază în
C2	=0	-
C3	=MAX(D2)	-
C4	=MAX(D5)	-
C5	=MAX(D2)	-
C6	=MAX(D3,D4)	-
C7	=MAX(D4,D5)	-
C8	=MAX(D6,D7)	-
D2	=C2+B2	D3:D8
E2	=F2-B2	E3:E8
F2	=MIN(E3,E5)	-
F3	=MIN(E5)	-
F4	=MIN(E6,E7)	-
F5	=MIN(E4,E7)	-
F6	=MIN(E8)	-
F7	=MIN(E8)	-
F8	=D8	-
G2	=E2-C2	G3:G8
H2	=IF(G2=0,"DA","NU")	H3:H8
D10	=MAX(D2:D8)	-

În graficul Gantt activităţile sunt afişate pe axa verticală, iar pe axa orizontală este reprezentat timpul. Graficul indică cel mai devreme termen de începere a fiecărei activităţi şi durata activităţii.
În continuare vom ilustra modul de construire a graficelor Gantt asociate unei probleme.

1. Se selectează datele care vor fi reprezentate în grafic: activităţile, durata activităţilor şi cel mai devreme termen de începere a activităţilor.

2. Se creează un grafic de tip *Staked Bar*.

3. Se selectează seria DI.

4. Se apasă butonul din dreapta al mouse-ului şi se selectează comanda *Format Series*. Se selectează butonul *Series Order* şi se stabileşte pentru afişarea seriilor ordinea DI, Durata.

5. Se selectează butonul *Patterns*, şi în secţiunile *Border* şi *Area* se selectează opţiunile *None*. Deci barele ataşate termenelor de începere ale activităţilor vor fi transparente, iar barele care reprezintă durata activităţilor vor apărea în prelungirea lor.

6. Se selectează seria Durata, se apasă butonul din dreapta al mouse-ului şi se selectează comanda *Format Series*. Se selectează butonul *Data Labels*, opţiunea *Show Value*. Astfel în dreptul fiecărei bare va fi afişată durata activităţii.

7. Se selectează axa Y, se apasă butonul din dreapta al mouse-ului şi se selectează comanda *Format Axis*. Se selectează butonul *Scale*, opţiunile *Categories in reverse order* şi *Value (Y) axis crosses at maximum category*. Astfel activităţile vor fi afişate începând din partea de sus a axei Y.

3.15 Aplicaţii propuse

Problema 3.15.1 *Se citesc de la tastatură două numere întregi, m şi n, apoi m perechi de numere întregi reprezentând extremităţile muchiilor unui graf neorientat cu m muchii şi n vârfuri. Să se construiască matricea de adiacenţă, apoi să se scrie gradele varfurilor în fişierul "graf.txt".*

Problema 3.15.2 *Se citesc de la tastatură m perechi de numere întregi (x, y) reprezentând extremităţile arcelor unui graf orientat cu n noduri şi m arce. Să se stabilească dacă în graful astfel definit există noduri izolate .*

Problema 3.15.3 *Se citeşte de la tastatură matricea de adiacenţă a unui graf orientat cu n noduri. Să se scrie arcele grafului în fişierul "arce.txt".*

Problema 3.15.4 *Se citeşte de la tastatură matricea de adiacenţă a unui graf orientat cu n noduri. Să se afişeze pe ecran nodurile cu proprietatea că numărul arcelor care ies din nod este egal cu numărul arcelor care intră în nod.*

Problema 3.15.5 *Se defineşte un arc al unui graf orientat ca o înregistrare cu trei câmpuri: nodul din care iese arcul, nodul în care intră arcul şi un cost asociat arcului. Definim un graf orientat ca un vector de arce. Fiind dat*

vectorul de arce al unui graf orientat cu n noduri şi m arce, să se construiască şi să se afişeze matricea de adiacenţă, apoi să se determine costul mediu al grafului (media aritmetică a costurilor arcelor).

Problema 3.15.6 *Scrieţi un program care citeşte din fişierul text "graf.txt" informaţii despre un graf orientat (de pe prima linie numărul de noduri, apoi matricea de adiacenţă) şi de la tastatură o mulţime A de numere care reprezintă etichetele unor noduri din graf şi afişează mulţimea arcelor ce au o extremitate într-un nod din mulţimea A şi o extremitate în mulţimea $X \setminus A$ (X este mulţimea nodurilor grafului).*

Problema 3.15.7 *În fişierul text "graf.txt" este scrisă o matrice, astfel: pe primul rând două numere naturale separate prin spaţiu, care reprezintă numărul de linii şi numărul de coloane ale matricei şi pe următoarle rânduri valori numerice despărţite prin spaţiu, care reprezintă elementele de pe câte o linie a matricei. Scrieţi un program care să verifice dacă această matrice poate fi matricea de adiacenţă asociată unui graf orientat. În caz afirmativ, să se determine câte noduri care au gradul intern egal cu gradul extern există.*

Problema 3.15.8 *Se citeşte de la tastatură o matrice, de dimensiune $n \times n$, unde n este număr natural, cu elemente numere întregi. Să se scrie un program care verifică dacă matricea respectivă poate reprezenta matricea de adiacenţă asociată unui graf. Dacă da, să se stabilească dacă graful respectiv este neorientat sau orientat.*

Problema 3.15.9 *Cunoscându-se că într-un grup de n persoane, codificate prin numere între 1 şi n, fiecare persoană are o listă (dată) de alte persoane pe care le va informa deîndată ce află un anumit mesaj, să se determine dacă există persoane care vor primi de cel puţin două ori acelaşi mesaj.*

Problema 3.15.10 *În secolul XXI, locul clasicelor plăcuţe indicatoare care ne ajută să ne orientăm în intersecţii, este luat de panouri electronice luminoase. Primarul unui oraş a achiziţionat trei tipuri de astfel de panouri, de culoare roşie, albastră şi verde. El doreşte să plaseze câte unul în fiecare intersecţie, dar în aşa fel încât pe fiecare stradă, delimitată la capete de două intersecţii, să nu se întâlnească acelaşi tip de panou de două ori. Găsiţi o soluţie posibilă de aranjare a panourilor în intersecţii conform dorinţei primarului, sau un răspuns negativ dacă aşa ceva nu e posibil. Presupunem că s-a achiziţionat un număr suficient de panouri de fiecare tip. Se dau:*

numărul n al intersecţiilor, numărul m al străzilor, precum şi m perechi de numere întregi, fiecare pereche reprezentând cele două intersecţii care delimitează strada.

Problema 3.15.11 *Reţeaua de străzi a unui oraş se reprezintă printr-un graf orientat, având ca noduri intersecţii de minimum* 3 *străzi. O piaţă este o intersecţie de minimum* 4 *străzi. Stabiliţi numărul de pieţe, care sunt acestea şi care este piaţa centrală (în care se întâlnesc cele mai multe străzi). Se ştie că aceasta este unică. Stabiliţi care sunt pieţele de la care se poate ajunge la piaţa centrală.*

Problema 3.15.12 *Un graf neorientat este reprezentat prin liste de adiacenţă. Să se scrie un program care determină numărul de muchii.*

Problema 3.15.13 *Scrieţi un program care citeşte din două fişiere text, respectiv din "graf1.txt" un graf orientat, reprezentat prin lista muchiilor, şi din fişierul "graf2.txt" un graf orientat, reprezentat prin lista vecinilor, şi care verifică dacă cele două grafuri sunt identice.*

Problema 3.15.14 *Din fişierele "graf1c.in" şi "graf2c.in" se citesc informaţii despre matricele de adiacenţă a două grafuri neorientate: de pe prima linie numărul de noduri, apoi matricea de adiacenţă, să se verifice dacă unul dintre grafuri este graf complementar celuilalt graf.*

Problema 3.15.15 *Să se coloreze în toate modurile posibile muchiile unui graf neorientat cu n vârfuri şi m muchii, folosind un număr de c culori disponibile, aşa încât oricare două muchii incidente să fie colorate diferit.*

Problema 3.15.16 *Într-un grup de n persoane s-au stabilit două tipuri de relaţii: de prietenie şi de vecinătate. Scrieţi un program care să citească matricele de adiacenţă ale celor două grafuri din fişierul text "pv.in" (pe primul rând, numărul de persoane, şi pe următoarele rânduri, în ordine, liniile fiecărei matrice de adiacenţă) şi care să afişeze persoanele care sunt şi prietene şi vecini.*

Problema 3.15.17 *Într-un grup de n persoane s-au stabilit două tipuri de relaţii: de prietenie şi de vecinătate. Scrieţi un program care să citească matricele de adiacenţă ale celor două grafuri din fişierul text "pv.in" (pe primul rând, numărul de persoane, şi pe următoarele rânduri, în ordine, liniile fiecărei matrice de adiacenţă) şi care să afişeze cel mai mare număr de persoane care se găsesc într-un grup de vecini prieteni.*

Problema 3.15.18 *Să se scrie un program care verifică dacă un graf neorientat dat prin matricea sa de adiacenţă este un graf complet.*

Problema 3.15.19 *Din fişierele "graf1p.in" şi "graf2p.in" se citesc informaţii despre matricele de adiacenţă a două grafuri orientate: de pe prima linie numărul de noduri, apoi matricea de adiacenţă, să se verifice dacă unul dintre grafuri este graf parţial al celuilalt. În caz afirmativ, se afişează care este graful şi care este graful parţial.*

Problema 3.15.20 *Pentru un graf neorientat se cunoaşte matricea de adiacenţă, să se scrie un program care generează un graf parţial al său obţinut prin eliminarea muchiilor care au la extremităţi un nod care are gradul minim şi un nod care are gradul maxim în graf. Graful obţinut se va afişa sub forma unui vector de muchii.*

Problema 3.15.21 *Pentru un graf orientat se cunoaşte matricea de adiacenţă, să se scrie un program care generează subgraful care se obţine prin eliminarea nodului care are cei mai mulţi vecini. Matricea de adiacenţă a grafului obţinut se va afişa în fişierul "subg.out".*

Problema 3.15.22 *Din fişierul text "graf.txt" se citesc muchiile unui graf neorientat, fiecare muchie se găseşte pe câte un rând din fişier. Să se scrie un program care caută şi afişează cel mai lung lanţ elementar care este format din noduri care au etichete numere consecutive, ordonate crescător.*

Problema 3.15.23 *Harta unui arhipelag este codificată printr-o matrice binară pătrată unde suprafeţele terestre sunt marcate cu* 1 *şi cele marine cu* 0*. Să se determine câte insule fac parte din arhipelag. O insulă este compusă din unul sau mai multe elemente ale matricei marcate cu* 1 *care se învecinează pe direcţiile* N, S, E, V*.*

Problema 3.15.24 *Scrieţi un program care citeşte lista muchiilor unui graf neorientat şi afişează toate ciclurile elementare care trec printr-un nod cu gradul minim.*

Problema 3.15.25 *Scrieţi un program care citeşte din fişierul "g.in" lista arcelor unui graf orientat, se presupune că pe fiecare rând al fişierului se găseşte câte o pereche de numere naturale care reprezintă extremităţile unui arc, şi care caută toate ciclurile elementare care există în graf şi le afişează. Dacă nu există niciun ciclu elementar, se afişează un mesaj.*

Problema 3.15.26 *Fiind dat un graf orientat, să se scrie un program care verifică dacă graful respectiv este un graf turneu. (Un graf orientat în care, între oricare două noduri există un singur arc şi numai unul, se numeşte graf turneu.)*

Problema 3.15.27 *Fiind dat un graf orientat prin lista arcelor sale, se cere să se scrie un program care determină toate drumurile elementare din graf. Drumurile obţinute se vor afişa în fişierul "drum.out", fiecare drum se va afişa pe un rând separat. Dacă nu se va găsi niciun drum, în fişier se va afişa un mesaj corespunzător.*

Problema 3.15.28 *Fiind dat un graf orientat prin lista arcelor sale, se cere să se scrie un program care determină toate drumurile elementare cu lungimea cea mai mică dintre două vârfuri, x şi y. Etichetele vârfurilor se citesc de la tastatură.*

Problema 3.15.29 *Speologii care au cercetat o peşteră au constatat existenţa a n încăperi în interiorul acesteia. Prin tehnici specifice meseriei lor, au demonstrat existenţa unor cursuri de apă între unele din aceste încăperi. Fiecare din aceste "pârâuri subterane" izvorăşte dintr-o încăpere şi se varsă într-alta având un anumit sens de curgere. Două încăperi se numesc comunicante între ele dacă plecând dintr-una şi mergând numai de-a lungul unor cursuri de apă, putem ajunge în cealaltă. Să se identifice o porţiune din harta peşterii care conţine un număr maxim de încăperi comunicante între ele.*

Problema 3.15.30 *Într-un grup de n persoane se precizează perechi de persoane care se consideră prietene. Folosind principiul "prietenul prietenului meu îmi este prieten" să se determine grupurile care sunt formate dintr-un număr k de persoane între care se pot stabili relaţii de prietenie, directe sau indirecte, k se citeşte de la tastatură.*

Problema 3.15.31 *Într-un grup de n persoane se precizează perechi de persoane care se consideră prietene. Folosind principiul "prietenul prietenului meu îmi este prieten" să se determine grupurile cu un număr maxim de persoane între care se pot stabili relaţii de prietenie, directe sau indirecte.*

Problema 3.15.32 *Fie un graf cu n vârfuri, a cărui matrice de adiacenţă se citeşte din fişierul "gr.txt". Fişierul conţine pe primul rând valoarea lui*

n, apoi, pe fiecare din următoarele n rânduri, elementele unei linii a matricei separate prin spaţii. Fiind dat un nod x introdus de la tastatură, să se determine numărul drumurilor care încep în nodul x, precum şi numărul drumurilor care se termină în x.

Problema 3.15.33 *Într-o zonă de munte, există n cabane, între unele cabane existând trasee de legătură. Să se determine, dacă există, o ordine de vizitare a cabanelor, astfel încât să se parcurgă o singură dată toate traseele de legătură din zonă revenind la aceeaşi cabană de la care s-a pornit.*

Problema 3.15.34 *Se citeşte matricea de adiacenţă a unui graf neorientat cu n vârfuri dintr-un fişier text. Fişierul conţine pe primul rând valoarea lui n, iar pe fiecare din următoarele n rânduri elementele unei linii a matricei separate prin spaţii. Să se genereze prin metoda backtracking toate ciclurile elementare din graf.*

Problema 3.15.35 *Într-un grup sunt n studenţi, pe care-i numerotăm* $1, 2, \ldots, n$. *Fiecare student cunoaşte o parte dintre ceilalţi elevi. Relaţia de cunoştinţă nu este neapărat reciprocă. Unul dintre studenţi are un CD, pe care toţi membrii grupului vor să-l aibă. CD-ul circulă printre membrii grupului în felul următor: fiecare student după ce l-a primit de la altcineva îl dă mai departe, dar numai unui student pe care îl cunoaşte. Determinaţi o modalitate (dacă există) prin care CD-ul să circule pe la fiecare student exact o singură dată, transmiterea lui făcându-se numai către o cunoştinţă, iar în final CD-ul să ajungă din nou la proprietarul său.*

Problema 3.15.36 *Pe o arenă hipică ce urmează să organizeze un concurs de călărie, s-au montat n obstacole, numerotate* $1, 2, \ldots, n$. *Traseul pe care îl vor parcurge călăreţii în timpul concursului trebuie să ţină cont de următoarele condiţii:*

- *se poate începe cu orice obstacol şi trebuie sărite toate obstacolele, fiecare o singură dată;*
- *distanţa între oricare două obstacole poate fi parcursă numai într-un singur sens, pe care călăreţii îl cunosc la intrarea în concurs.*

Scrieţi un program cu ajutorul căruia călăreţii să-şi stabilească traseul.

Problema 3.15.37 *Se citeşte un graf neorientat din fişierul "graf.txt". Fişierul conţine: pe primul rând numărul de vârfuri n şi numărul de muchii m din graf, separate prin spaţii; pe fiecare din următoarle m rânduri, câte o pereche de numere întregi separate prin spaţii, reprezentând extremităţile unei muchii. Să se verifice dacă graful dat este hamiltonian, tipărindu-se un mesaj corespunzător. Se va folosi teorema potrivit căreia un graf este hamiltonian dacă gradul oricărui vârf x este mai mare sau egal cu $n/2$.*

Problema 3.15.38 *O peşteră are n încăperi, fiecare la o înălţime $h(i)$. Încăperile i şi j sunt legate prin culoarul $i - j$. În încăperea s se află un izvor. Care încăperi sunt inundate?*

Problema 3.15.39 *Să se scrie un program care citeşte dintr-un fişier text un graf precizat prin matricea sa de adiacenţă şi afişează pe ecran cele mai depărtate două noduri, precum şi distanţa dintre ele. Distanţa dintre două noduri este considerată cea minimă.*

Problema 3.15.40 *Harta rutieră a unei ţări este precizată prin autostrăzile care leagă perechi de oraşe $a - b$. Afişaţi toate oraşele la care se poate ajunge dintr-un oraş x, folosind exact n autostrăzi.*

Problema 3.15.41 *Fiind date n oraşe, dorim să construim o reţea telefonică, conexă, cu vârfurile în aceste oraşe. Se cunosc costurile $c(u)$ de construcţie ale liniei pentru orice muchie $u = [x, y]$ care conectează oraşele x şi y. Se cere să se optimizeze costul total de construcţie al reţelei. (Se foloseşte algoritmul lui Prim).*

Problema 3.15.42 *Pentru construirea unei reţele interne de comunicaţie între secţiile unei intreprinderi, s-a întocmit un proiect în care au fost trecute toate legăturile ce se pot realiza între secţiile intreprinderii. În vederea definitivării proiectului şi întocmirii necesarului de materiale etc., se cere să se determine un sistem de legături ce trebuie construit, astfel încât orice secţie să fie racordată la această reţea de comunicaţie, iar cheltuielile de construcţie să fie minime.*

Problema 3.15.43 *n oraşe dintr-o ţară sunt legate între ele prin şosele, specificate prin triplete de forma (i, j, d), cu semnificaţia: oraşul i este legat direct cu oraşul j printr-o şosea de lungime d. Nu toate oraşele sunt legate direct între ele, dar există comunicaţii între toate oraşele. Stabiliţi care oraş poate fi ales capitală, ştiind că acesta satisface proprietatea că suma distanţelor drumurilor la celelalte oraşe este minimă.*

Problema 3.15.44 *Din fişierul "date.txt" se citeşte un graf neorientat, precizat prin numărul de noduri, numărul de muchii şi muchiile sale. Se doreşte afişarea lungimii drumurilor minime între oricare două noduri i şi j. De asemenea, se va calcula şi lungimea "măsurată în noduri" între oricare două noduri. Se va folosi algoritmul Floyd-Warshall.*

Problema 3.15.45 *Cele n pieţe din Veneţia sunt legate printr-un sistem de canale. În piaţa San Marco cineva uită robinetul deschis şi se produce o inundaţie care se extinde în pieţele situate la distanţa <= r de San Marco. Stabiliţi numărul de pieţe inundate. Pieţele sunt identificate prin numerele de la* 1 *la n, iar canalele prin perechi de pieţe* (i, j)*. Graful este reprezentat prin liste de adiacenţă.*

Problema 3.15.46 *Harta unei ţări este formată din mai multe regiuni. O parte dintre regiuni au frontiera comună. Scrieţi un program care afişează regiunea care se învecinează cu cele mai multe dintre regiuni.*

Problema 3.15.47 *Între persoanele dintr-o instituţie, identificate prin numere, s-au stabilit de-a lungul timpului relaţii de rudenie, precizate prin perechi a b cu semnificaţia "a este rudă cu b". Stabiliţi numărul de clanuri existente şi daţi componenţa acestora, ştiind că relaţia de rudenie este simetrică şi tranzitivă.*

Problema 3.15.48 *Într-un munte există n cavităţi, numerotate de la* 1 *la n. Între unele dintre acestea există comunicaţii prin tuneluri, precizate sub forma* $i - j$*. Unele cavităţi comunică şi cu mediul exterior prin tuneluri de forma* $i - 0$ *sau* $0 - i$*. Determinaţi câte cavităţi nu sunt accesibile.*

Problema 3.15.49 *Într-un oraş există n intersecţii, legate între ele prin străzi cu sens unic. Stabiliţi grupele de intersecţii care nu comunică între ele. Câte asemenea grupuri există?*

Problema 3.15.50 *Între n oraşe, date prin coordonatele* (x_i, y_i)*, există legături de comunicaţie de forma* $j - k$*. Stabiliţi numărul de grupuri de oraşe unite prin legături şi determinaţi legăturile necesare pentru ca toate oraşele să comunice între ele, astfel încât suma legăturilor adăugate să fie minimă.*

Problema 3.15.51 *Într-un oraş există n obiective turistice; fiecare obiectiv i se află situat la cota h_i. Între unele dintre aceste obiective există trasee. Un traseu este pricizat prin capetele traseului, i şi j, şi lungimea traseului, l_{ij}. Datele se termină prin* 000*. Un biciclist doreşte să se plimbe între obiectivele a şi b, dar nefiind prea antrenat, alege traseul minim, în care nu se depăşeşte panta p la urcare sau coborâre. Se ştie că panta între două obiective i şi j se calculează ca: $(h_j - h_i)/l_{ij}$. Ajutaţi biciclistul să-şi alcătuiască traseul sau comunicaţi-i că nu-l poate parcurge. Se va folosi algoritmul lui Dijkstra.*

Problema 3.15.52 *Fiind dat un graf orientat prin matricea sa de adiacenţă, se cere să se scrie un program care va parcurge graful cu metoda BF.*

Problema 3.15.53 *Fiind dat un graf orientat prin matricea sa de adiacenţă, se cere să se scrie un program care va parcurge graful cu metoda DF.*

Problema 3.15.54 *Scrieţi un program care afişează soluţia aranjării la masa rotundă a regelui Arthur a celor $2 \times n$ cavaleri, ştiind că fiecare dintre cavaleri are $n - 1$ duşmani şi că la masă niciun cavaler nu trebuie să stea lângă duşmanul lui.*

Problema 3.15.55 *O persoană trebuie să se deplaseze cu autoturismul cât mai repede între două intersecţii din oraş. Traficul între două intersecţii nu este întotdeauna în ambele sensuri. Cunoscând timpul mediu de deplasare între două intersecţii, să se determine care este traseul pe care trebuie să-l aleagă pentru a ajunge de la intersecţia A la intersecţia B cât mai repede.*

Anexa A

Limbajul C/C++

A.1 Vocabularul limbajului

Setul de caractere al limbajului cuprinde: literele mari şi mici ale alfabetului englez; cifrele sistemului de numeraţie zecimal; caracterele speciale $(+,\ -,\ \star,\ /,\ \backslash,\ )$
Identificatorii sunt succesiuni de litere, cifre sau caracterul _, primul caracter fiind obligatoriu o literă sau caracterul _. Limbajul $C{+}{+}$ este case-sensitiv (literele mici se consideră distincte de cele mari).
Cuvintele cheie sunt identificatori cu semnificaţie specială, care nu pot fi folosiţi în alt context decât cel precizat în definirea limbajului. Cuvintele rezervate din $C++$ sunt următoarele:

auto	break	case	char	bool	catch	const
continue	default	do	class	delete	double	else
enum	extern	friend	inline	float	for	goto
if	new	operator	int	long	register	return
private	protected	short	signed	sizeof	static	public
template	struct	switch	typedef	union	this	throw
unsigned	void	volatile	while	try	virtual	wchar_t

Separatori şi comentarii . Separatorii în $C++$ sunt: blancul, sfârşitul de linie şi comentariul. Instrucţiunile şi declaraţiile sunt separate prin ";". Comentariile sunt texte încadrate între: $/*$ şi $*\backslash$ sau precedate de //.

A.2 Tipuri de date standard

Tipurile întregi sunt următoarele:

NUME TIP	DIMENSIUNI ÎN BIŢI	DOMENIU
unsigned char	8	0..255
char	8	-128..127
unsigned int	16	0..65535
short int	16	-32768..32767
int	16	-32768..32767
unsigned long	32	0..4294967295
long	32	-2147483648..2147483647

Tipurile reale de bază sunt prezentate în continuare:

NUME TIP	DIMENSIUNI ÎN BIŢI	DOMENIU ÎN VAL. ABSOLUTĂ
float	32	între 3.4×10^{-38} şi 3.4×10^{38}
double	64	între 1.7×10^{-308} şi 1.7×10^{308}
long double	80	între 3.4×10^{-4932} şi 1.1×10^{4932}

A.3 Constante

Constantele sunt date care nu se modifică pe parcursul execuţie unui program. Dacă au asociat un identificator, atunci se numesc **constante simbolice**. Dacă nu au asociat nici un identificator, atunci ele se reprezintă prin valoarea lor. Există constante întregi, constante caracter, constante reale, constante şir de caractere.
Pentru a defini constante simbolice vom folosi construcţia:
const [*tip_constanta*] *nume_constanta* = *valoare*;

A.4 Declararea variabilelor

Declararea variabilelor se realizează prin specificarea tipului, a identificatorilor şi eventual a valorilor cu care se iniţializează. O declaraţie de variabile simple de acelaşi tip are forma:
nume_tip lista_de_identificatori;

Prin *lista_de_identificatori* se înţelege o succesiune de nume de variabile separate prin virgulă.

A.5 Expresii

Prin expresie înţelegem o succesiune de operatori şi operanzi care respectă anumite reguli. Un operand poate fi: o constantă, o constantă simbolică, numele unei variabile, referirea la elementul unei structuri, apelul unei funcţii, expresie inclusă între paranteze rotunde. Operatorii pot fi unari sau binari. O expresie are o valoare şi un tip şi pot fi folosite parantezele pentru a impune o anumită ordine a operaţiilor.
Regula conversiilor implicite. Această regulă se aplică la evaluarea expresiilor. Ea acţionează atunci când un operator binar se aplică la doi operanzi de tipuri diferite: operatorul de tip "inferior" se converteşte spre tipul "superior".
Operatorii care pot fi utilizaţi în $C/C++$ sunt:
Operatorii aritmetici sunt: - operatorii unari $+$, $-$; operatorii binari multiplicativi $*$, $/$, $\%$; operatorii binari aditivi $+$, $-$.
Operatorii relaţionali sunt:$<$, $<=$, $>=$, $>$.
Operatorii de egalitate sunt: $==$ (pentru egalitate) şi $!=$ (pentru diferit).
Operatorii logici sunt: ! (negaţia), && (şi logic), || (sau logic).
Operatorii logici pe biţi sunt: $\sim$ (complement faţă de 1, operator unitar), $\ll$ (deplasare stânga), $\gg$ (deplasare dreapta), & (şi logic pe biţi), $\hat{}$ (sau exclusiv logic pe biţi), | (sau logic pe biţi).
Operatorul de atribuire este $=$. El se utilizează în expresii de forma:
v=expresie
unde v este o variabilă sau o adresă.
Operatorul de atribuire se mai poate utiliza şi precedat de un operator binar aritmetic sau logic pe biţi.
Operatorii de incrementare/decrementare (cu o unitate) sunt $++$, respectiv $--$. Aceşti operatori pot fi folosiţi prefixat sau postfixat.
Operatorul de forţare a tipului sau de conversie explicită Conversia valorii unui operand spre un anumit tip se poate face folosind construcţia: *(tip)operand.*
Operatorul dimensiune este folosit pentru determinarea dimensiunii în octeţi a unei date sau al uni tip. El poate fi utilizat sub formele: *sizeof data* sau *sizeof(tip)* unde data poate fi o adresă sau o variabilă, iar tip poate fi

un identificator de tip predefint, definit de utilizator sau o construcţie care defineşte un tip.
Operatorii condiţionali permit construirea de expresii a căror valoare depinde de valoarea unei condiţii. O expresie condiţională are formatul:
$e1?e2 : e3$
Această expresie se evaluează astfel:
- se determină valoarea expresiei $e1$.
- dacă $e1$ are o valoare diferită de zero, atunci valoarea şi tipul expresiei condiţionale coincid cu valoarea şi tipul expresiei $e2$. Altfel valoarea şi tipul expresiei condiţionale coincid cu valoarea şi tipul expresiei $e3$.
Operatorul adresă este unar şi se notează prin &. El este utilizat pentru a determina adresa de început a zonei de memorie alocată unei date.
Operatorul virgulă leagă două expresii în una singură conform formatului: *expresie1,expresie2*. Construcţia anterioară este o expresie care are valoarea şi tipul ultimei expresii.

A.6 Tablouri

Un tablou reprezintă un tip structurat care ocupă o zonă de memorie continuă şi conţine elemente de acelaşi tip. Un tablou se declară astfel:

$$tip\ \ nume\_tablou[c1][c2]\ldots[ck];$$

Unde *tip* este un tip predefinit, ce corespunde tipului componentelor. $c1, c2, \ldots, ck$ sunt limitele superioare ale indicilor tabloului. Indicii oricărui tablou încep cu 0 şi se termină cu limita superioară minus 1. Numele unui tablou reprezintă adresa primului său element.
Referirea la o componentă a tabloului se face prin construcţia:

$$nume\_tablou[i1][i2]\ldots[ik]$$

A.7 Funcţii

Un program conţine una sau mai multe funcţii. Dintre acestea una este obligatorie şi se numeşte funcţia principală.
Orice funcţie are un nume. Numele funcţiei principale este *main*. Celelalte au nume definit de utilizator. Forma generală a unei funcţii este:

tip_functie nume_functie(*lista_parametrilor_formali*)
{
declarari
instructiuni
}
Prima linie din formatul de mai sus reprezintă antetul funcţiei, iar partea inclusă între acolade, împreună cu acoladele, formează corpul funcţiei.
lista_parametri_formali este o listă formată din declarările parametrilor formali, fiecare parametru fiind precedat de tipul său.
Există două categorii de funcţii. Prima conţine funcţii care returnează o valoare la revenire. Tipul acestei valori se defineşte prin *tip_functie* din antetul funcţiei. Cealaltă categorie nu returnează nicio valoare la revenire. Pentru aceste funcţii se va folosi cuvântul **void** în calitate de tip. El semnifică lipsa unei valori returnate la revenirea din funcţie.
O funcţie poate avea sau nu parametri. În cazul lipsei parametrilor antetul funcţiei se reduce la:
tip_functie nume_functie() sau
tip_functie nume_functie(*void*)
Valoarea returnată de funcţie se specifică în corpul funcţiei prin construcţia:
return expresie;
unde valoarea expresiei trebuie să fie de acelaşi tip cu cel al funcţiei.
Variabilele utilizate în funcţii pot fi globale, respectiv locale. Cele globale sunt declarate în afara oricărei funcţii şi pot fi utilizate în orice funcţie, iar cele locale sunt declarate în interiorul unui bloc de instrucţiuni şi pot fi utilizate numai în acestea.

A.8 Apelul şi prototipul funcţiilor

O funcţie poate fi apelată folosind o construcţie de forma:
nume_functie(*lista_parametrilor_efectivi*)
La apel se atribuie parametrilor formali valorile parametrilor efectivi şi apoi executarea se continuă cu prima instrucţiune din corpul funcţiei apelate. La revenirea din funcţie se revine la funcţia din care s-a făcut apelul şi executarea continuă cu instrucţiunea următoare apelului.
Dacă funcţia nu are parametri formali atunci apelul funcţiei se realizează prin construcţia:
nume_functie( )

Transferul prin intermediul parametrilor se poate realiza în două moduri: prin valoare, respectiv prin adresă.
În general funcţiile sunt declarate înainte de folosirea lor. Dacă se doreşte apelarea lor înainte de definire, atunci definiţia funcţiei este înlocuită printr-un prototip al ei. Prototipul reprezintă o informaţie pentru compilator cu privire la: tipul valorii returnate de funcţie şi existenţa şi tipurile parametrilor funcţiei. Prototipul unei funcţii este asemănătoare cu antetul său doar că în acest caz la sfârşit se pune ";".

A.9 Preprocesare. Includeri de fişiere. Substituiri

Un program sursă în $C/C++$ poate fi prelucrat înainte de a fi compilat. O astfel de prelucrare se numeşte preprocesare. Ea este realizată automat înaintea compilării. Preprocesarea constă, în principiu, în substituţii. Preprocesarea asigură includeri de fişiere cu texte sursă, definiţii şi apeluri de macrouri, compilare condiţionată.
Preprocesarea asigură prelucrarea unor informaţii aflate pe linii speciale care au ca prim caracter, caracterul "#".
Un fişier cu text sursă poate fi inclus cu ajutorul construcţiei:
$\#include$ "$nume\_fisier$"
sau
$\#include$ $< nume\_fisier >$
Formatul cu paranteze unghiulare se utilizează la includerea fişierelor standard.
Construcţia $\#define$ se poate folosi pentru a substitui prin nume o succesiune de caractere. În acest scop se utilizează formatul:
$\#define$ $nume$ $succesiune\_de\_caractere$
Cu ajutorul acestei construcţii preprocesarea substituie peste tot în textul sursă *nume* cu *succesiune_de_caractere*, exceptând cazul când *nume* apare într-un şir de caractere sau într-un comentariu.
Biblioteci Limbajul conţine multe funcţii pentru prelucrarea datelor; acestea sunt grupate în biblioteci. Dintre cele mai importante biblioteci, precizăm:
- stdio.h şi io.h, pentru citire/scriere;
- stdlib.h şi math.h, pentru prelucrări numerice;

- ctype.h, pentru prelucrarea sau verificarea caracterelor;
- mem.h şi string.h, pentru şiruri de caractere şi zone de memorie;
- alloc.h, malloc.h şi stdlib.h, pentru alocarea memoriei;
- conio.h, pentru interfaţa cu consola.

A.10 Structuri şi tipuri definite de utilizator

Pentru a utiliza date cu componente de tipuri diferite se folosesc structuri. Unul dintre formatele cele mai utilizate este următorul:
struct $NUMES\{$
lista_de_declarari
$\}$ *nume*1, *nume*2, ..., *numen*;
unde $NUMES$, *nume*1, *nume*2, ..., *numen* sunt identificatori ce pot lipsi, dar nu toţi o dată. Dacă $NUMES$ este prezent, atunci el defineşte un tip nou, introdus prin declaraţia de structură respectivă.
*nume*1, *nume*2, ..., *numen* sunt date de tipul $NUMES$.
lista_de_declarari este alcătuită din una sau mai multe declarări prin care se precizează câmpurile structurii precedate de tipul acestora.
O variabilă de tip $NUMES$ poate fi declarată şi ulterior, folosind formatul:
$NUMES$ *nume_variabila*;
Accesul la elementele unei structuri se realizează prin construcţia:
variabila_de_tip_struct.element
Dacă dorim să asociem unui tip un anumit cuvânt cheie putem realiza acest lucru prin construcţia:
typedef tip nume_tip;
unde *tip* este de un tip predefinit, fie un tip utilizator, iar *nume_tip* este identificatorul asociat acestui tip definit.
După ce s-a definit un nou tip, numele respectiv poate fi utilizat pentru a declara variabile de acel tip.

A.11 Citiri/scrieri

- Funcţiile **getch** şi **getche** permit citirea direct de la tastatură a unui caracter. Prototipurile celor două funcţii sunt în fişierul $<$ *conio.h* $>$.

- Funcţia **putch** afişează un caracter pe ecran. Prototipul acestei funcţii se găseşte în fişierul $<$ *conio.h* $>$.

- Funcţiile **gets** şi **puts** permit citirea, respectiv afişarea şirurilor de caractere. Funcţiile gets şi puts au prototipurile în fişierul $< stdio.h >$.

- Funcţia **printf** poate fi folosită pentru a afişa date pe ecran cu un anumit format. Prototipul acestei funcţii se găseşte în fişierul $< stdio.h >$.

- Funcţia **scanf** este utilizată pentru a introduce date de la tastatură cu un anumit format.

- Metodele şi manipulatorii claselor **iostream**. Transferul datelor între calculator şi mediul extern este asimilat cu un "curent" (stream) de date şi este modelat cu ajutorul unor entităţi specifice numite streamuri. Sistemul standard de comunicare, consola, este format din două astfel de entităţi ale clasei iostream: cin (tastatura) şi cout (monitorul). Dintre metodele şi instrumentele principale folosite pentru citirea/ scrierea datelor amintim: $\gg$ - pentru citirea datelor; $\ll$ - pentru scrierea datelor; *get* - pentru citirea unui caracter sau a unui şir de caractere şi *put* - pentru scrierea unui caracter.

Utilizarea obiectelor, *cin*, *cout*, a metodelor şi manipulatorilor specifici necesită includerea în program a unuia dintre fişierele *iostream.h* sau $fstream.h$. Constanta $C++$ *endl* desemnează sfârşitul de linie.

A.12 Instrucţiuni

1. Instrucţiunea vidă. Se reduce la caracterul ";" şi nu are efect.
2. Instrucţiunea expresie. Se obţine scriind caracterul ";" după o expresie.
3. Instrucţiunea compusă. Este o succesiune de declaraţii urmate de instrucţiuni, incluse între acolade. Declaraţiile sau instrucţiunile pot lipsi. Formatul instrucţiunii:
{
declaratii
instructiuni
}
4. Instruţiunea if**.** Această instrucţiune ne permite ramificarea în funcţie de valoarea unei expresii.
Format 1:

$if(expresie)$ $instructiune1$;
Efect:
P1. Se evaluează expresia din paranteză.
P2. Dacă valoarea expresiei este diferită de zero, atunci se execută $instructiune1$; altfel se trece la următoarea instrucţiune.
Format 2:
$if(expresie)$ $instructiune1$;
$else$ $instructiune2$;
Efect:
P1. Se evaluează expresia din paranteză.
P2. Dacă valoarea expresiei este diferită de zero, atunci se execută $instructiune1$; altfel se execută $instructiune2$.
P3. Se trece la instrucţiunea următoare.
5. Instruţiunea *while* Are următorul format:
$while(expresie)$ $instructiune$;
Efect:
P1.Se evaluează expresia din paranteză.
P2. Dacă valoarea expresiei este diferită de zero, se execută instrucţiunea şi se trece la pasul 1.
P3. Dacă valoarea expresiei este 0, se trece la următoarea instrucţiune.
6. Instrucţiunea $do-while$ Are formatul:
do $instructiune$ $while(expresie)$;
Efect:
P1. Se execută $instructiune$.
P2. Se evaluează $expresie$.
P3. Dacă valoarea expresiei este diferită de zero, atunci se reia pasul 1; altfel se trece la instrucţiunea următoare.
7. Instrucţiunea for Formatul instrucţiunii este:
$for(e1;e2;e3)$ $instructiune$;
unde $e1$, $e2$, $e3$ sunt expresii; $e1$ reprezintă partea de iniţializare, $e3$ parte de reiniţializare, iar $e2$ condiţia de continuare a executării instrucţiunii.
Efectul instrucţiunii:
P1. Se execută iniţializarea definită de $e1$.
P2. Se evaluează $e2$ şi, dacă valoarea este diferită de 0, atunci se execută $instructiune$, altfel se termină executarea instrucţiunii for.
P3. Se execută secvenţa de reiniţializare dată de $e3$, după care se reia etapa precedentă.
8. Instrucţiunea *switch* Permite realizarea structurii selective. Formatul

instrucţiunii este:
$switch(expresie)\{$
$case\ c_1:\ sir_1\_instr;\ |break;|$
$case\ c_2:\ sir_2\_instr;\ |break;|$
...
$case\ c_n:\ sir_n\_instr;\ |break;|$
$|default:\ sir\_instr;|$
$\}$
Paşii de execuţie ai instrucţiunii sunt:
P1. Se evaluează expresia din paranteză.
P2. Se compară pe rând valoarea expresiei cu valorile $c_1,\ c_2,\ \ldots, c_n$.
P3. Dacă valoarea expresiei coincide cu valoarea c_k, atunci se execută secvenţa de instrucţiuni definită prin sir_k_instr; dacă nu coincide cu niciuna, atunci se execută sir_instr, dacă există.
9. Instrucţiunea *break* Formatul instrucţiunii este:
break;
Această instrucţiune se utilizează pentru a ieşi dintr-o instrucţiune *switch* sau pentru a ieşi dintr-o instrucţiune ciclică.
10. Instrucţiunea *continue* Formatul instrucţiunii este:
continue;
Efect:

- în cadrul ciclurilor *while* şi $do-while$, se realizează direct evaluarea expresiei care decide asupra continuării ciclului;
- în ciclul for, ea realizează saltul la pasul de reiniţializare.

A.13 Pointeri

Prin pointeri înţelegem o variabilă care are valori adrese de memorie.
Un pointer se declară ca orice variabilă, cu deosebirea că numele ei este precedat de caracterul asterisc, adică vom folosi construcţia:
$tip\ \ *nume\_pointer$;
Valoarea aflată la adresa dintr-un pointer se poate utiliza prin construcţia:
$*nume\_pointer$
Pentru a putea fi folosiţi, variabilelor de acest tip trebuie să li se aloce zone de memorie, acest lucru se realizează în C standard cu ajutorul funcţiei *malloc*, iar în $C++$ cu ajutorul operatorului *new*:

$void\ *malloc(unsigned\ n)$;
Eliberarea zonei de memorie alocate prin *malloc* se poate realiza folosind funcţia $free$ sau operatorul *delete* în $C++$:
$void\ free(void\ *p)$;

A.14 Fişiere text

În limbajul C standard, fiecărui fişier fizic i se asociază o structură de tip $FILE$. Acest tip este definit în fişierul *stdio.h*. De asemenea, toate funcţiile de lucru cu fişiere au prototipurile în fişierul *stdio.h*.
Fişierele text sunt accesate secvenţial, de la primul caracter până la marcajul de sfârşit de fişier (EOF).
1. Deschiderea unui fişier - se realizează cu ajutorul funcţiei $fopen$, cu sintaxa:
$FILE\ *fopen(const\ char\ *nume\_fisier, const\ char\ *mod\_deschidere)$;
Modurile de deschidere pot fi: r (deschidere numai pentru citire), w (deschidere numai pentru scriere), a (deschidere numai pentru adăugare), b (deschidere numai în mod binar) şi t (deschidere în mod text, opţiune implicită). Dacă dorim să combinăm două sau mai multe moduri de deschidere, putem utiliza +.
2. Închiderea unui fişier - se realizează cu ajutorul funcţiei $fclose$, cu sintaxa:
$int\ fclose(FILE\ *stream)$
care returnează 0 în caz de succes.
3. Funcţia de verificare a sfârşitului de fişier - este $feof$, are sintaxa:
$int\ foef(FILE\ *stream)$
şi returnează 0 dacă poziţia pe care ne aflăm în fişier nu este la sfârşitul acestuia şi o valoare diferită de zero dacă poziţia actuală indică sfârşitul de fişier.
4. Funcţii de citire/scriere - pot fi, în funcţie de tipul datelor citite/scrise, de mai multe tipuri.

- Funcţii de citire/scriere pentru caractere:
  Pentru citirea unui caracter vom folosi:
  $int\ fgetc(FILE\ *stream)$;
  $int\ getc(FILE\ *stream)$;
  Dacă citirea a avut succes, se returnează valoarea caracterului citit, altfel se returnează EOF.

Pentru scrierea unui caracter vom folosi:
$int\ \ fputc(int\ c, FILE\ *stream)$;
$int\ \ putc(int\ c, FILE\ *stream)$;
Dacă scrierea a avut succes, ambele întorc caracterul care a fost scris; altfel se returnează EOF.

- Funcţii de citire/scriere pentru şiruri de caractere:
  Funcţia $fgets$ citeşte un şir de caractere dintr-un fişier şi are sintaxa:
  $int\ \ fgets(char\ *s, int\ n, FILE * stream)$
  unde s reprezintă $buffer$-ul în care se stochează şirul citit şi n indică numărul maxim de caractere care se vor citi.
  Funcţia $fputs$ permite scrierea unui şir de caractere într-un fişier, are sintaxa:
  $int\ \ fputs(conts\ char * s, FILE\ *f)$
  În caz de eroare, se întoarce valoarea EOF.

- Funcţii de citire/scriere cu format:
  Funcţia $fscanf$ permite citirea variabilelor dintr-un fişier de tip stream, sintaxa este:
  $int\ \ fscanf(FILE\ *stream, const\ char\ *format[adr\_var1, \ldots])$
  Funcţia citeşte o secvenţă de câmpuri de intrare caracter cu caracter, formează fiecare câmp conform formatului specificat corespunzător, iar câmpul format este transmis la adresa variabilei specificate.
  Funcţia $fprintf$ permite scrierea cu format, sintaxa:
  $int\ \ fprintf(FILE\ *stream, const\ char\ *format[argument, \ldots])$
  Acceptă o serie de argumente de tip expresie pe care le formatează conform specificării din şirul format; scrie datele în fişierul cerut.

Utilizarea entităţilor de tip *stream* din $C++$ în lucrul cu fişiere, presupune includerea fişierului $fstream.h$ în program, declararea unor obiecte de tip $ifstream$ pentru fişiere text utilizate pentru citire, respectiv $ofstream$ pentru fişierele utilizate pentru scriere şi accesarea metodelor şi manipulatorilor specifici.

Anexa B

Metoda BACKTRACKING

Această metodă se foloseşte în rezolvarea problemelor ce îndeplinesc simultan următoarele condiţii:

- soluţia lor poate fi pusă sub formă de vector $S = x_1, x_2, \ldots, x_n$, cu x_1 aparţinând mulţimii A_1, x_2 aparţinând mulţimii A_2,..., x_n aparţinând mulţimii A_n;
- mulţimile $A_1, A_2, \ldots, A_n$ sunt mulţimi finite, iar elementele lor se consideră că se află într-o relaţie de ordine binară bine stabilită;
- nu se dispune de o altă soluţie mai rapidă.

Observaţii:

- nu la toate problemele n este cunoscut de la început;
- $x_1, x_2, \ldots, x_n$ pot fi la rândul lor vectori;
- în multe probleme, mulţimile $A_1, A_2, \ldots, A_n$ coincid.

La întâlnirea unei astfel de probleme, dacă nu cunoaştem această metodă, suntem tentaţi să generăm toate elementele produsului cartezian $A_1 \times A_2 \times \ldots \times A_n$ şi fiecare element să fie testat dacă este soluţie. Rezolvând problema în acest mod, timpul de calcul este atât de mare, încât poate fi considerat infinit.
Metoda backtracking constă în următoarele:

- se alege primul element, x_1, ce aparţine lui A_1;

- presupunând generate elementle $x_1, x_2, \ldots, x_k$ aparţinând mulţimilor $A_1, A_2, \ldots, A_k$, se alege (dacă există) x_{k+1}, primul element disponibil din mulţimea A_{k+1} care îndeplineşte anumite condiţii de continuare, apărând astfel două posibilităţi:

  1. elementul există, se testează dacă nu s-a ajuns la o soluţie, în caz afirmativ aceasta se tipăreşte, în caz contrar se consideră generate $x_1, x_2, \ldots, x_l, x_{k+1}$;
  2. elementul nu există, situaţie în care se consideră generate elementele $x_1, x_2, \ldots, x_{k-1}$, reluându-se căutarea de la elementul următor lui x_k în mulţimea A_k;

- algoritmul se încheie când au fost luate în consideraţie toate elementele mulţimii A_1.

Bibliografie

[1] T.H. Cormen, C.E. Leiserson, R.R. Rivest, *Introducere în algoritmi*, Editura Computer Libris Agora, Cluj Napoca, 2000.

[2] G. Desbazeille, *Exercices et problèmes de recherche opérationnelle*, Ed. Dunod, Paris, 1976.

[3] R. Diestel, *Graph Theory*, Springer-Verlag, New York, Graduate Texts in Mathematics, vol. 173, 2000.

[4] C.Giumale,*Introducere în analiza algoritmilor: teorie şi aplicaţie*, Editura Polirom, Iaşi, 2004.

[5] T. Ionescu, *Grafuri: aplicaţii*, Editura Didactică şi Pedagogică, Bucureşti, 1973.

[6] B. Korte, J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, Springer, 2000.

[7] D. Opriş, Gh. Silberberg, *Optimizări liniare, discrete, convexe: aplicaţii*, Editura Mirton, Timişoara, 1999.

[8] T.Sorin, *Tehnici de programare şi structuri de date*, Editura L&S InfoMat, Bucureşti, 1995.

[9] P. Stavre, *Matematici speciale cu aplicaţii în economie*, Editura Scrisul Românesc, Craiova, 1982.

[10] D.Stoilescu, *Culegere de C\C++*, Editura Radial, Galaţi, 1998.

[11] I. Tomescu, *Probleme de combinatorică şi teoria grafurilor*, Editura Didactică şi Pedagogică, Bucureşti, 1981.