Prefaţă

Rapida dezvoltare a algoritmicii grafurilor s-a datorat în primul rând progresului exponenţial cunoscut de dezvoltarea calculatoarelor. Solicitată a participa la procesul de optimizare combinatorie, algoritmica grafurilor şi-a construit un fond de teoreme proprii, pe baza cărora s-au elaborat o mulţime de algoritmi ce formează astăzi instrumentul de bază al acestui domeniu. Aplicaţiile algoritmicii grafurilor în diverse doemnii, de la fundamentarea deciziilor politice la probleme macroeconomice, de la probleme de producţie la studiul reţelelor electrice, îi conferă o importanţă crescută.

Deoarece această lucrare se adresează în primul rând studenţilor din domeniul informatică, ea conţine principalele probleme ale algoritmicii grafurilor: noţiuni de bază, parcurgeri de grafuri, arbori şi arborescenţe, distanţe şi drumuri, probleme euleriene şi hamiltoniene. Expunerea este însoţită de numeroase exemple care facilitează înţelegerea corectă şi completă a algoritmilor prezentaţi. De asemenea, fiecare capitol conţine şi aplicaţii în diverse domenii ale algoritmilor prezentaţi în capitolul respectiv.

Cartea se adresează şi altor categorii de cititori: studenţilor din domeniile care studiază la anumite discipline şi algoritmica grafurilor, elevilor de la clasele de informatică, profesorilor de liceu care predau noţiuni de algoritmica grafurilor, ingineri, economişti etc. Cititorii interesaţi să studieze şi alte capitole de algoritmica grafurilor pot consulta bibliografia prezentată la sfârşitul lucrării.

Cuprins

Capitolul 1

Noţiuni introductive

1.1 Vocabularul de bază în teoria grafurilor

Avertizăm cititorul că terminologia din Teoria Grafurilor nu este complet unitară. Vom păstra, în cea mai mare parte, terminologia consacrată în literatura română de specialitate.

Definiţia 1.1. Se numeşte *graf orientat* un triplet $G = (N, A, g)$ format dintr-o mulţime N de elemente numite *noduri* sau *vârfuri*, dintr-o familie A de elemente numite *arce* şi dintr-o aplicaţie $g : A \to N \times N$ numită *funcţie de incidenţă*, prin care fiecărui element $a \in A$ i se asociază o pereche ordonată $(x, y) \in N \times N$ cu $x \neq y$; dacă eliminăm condiţia $x \neq y$ atunci arcul de forma (x, x) se numeşte *buclă*, iar G se numeşte *graf general orientat*.

În continuare vom presupune că graful orientat G este finit, adică mulţimea nodurilor $N = \{\ldots, x, \ldots\}$ este finită şi familia arcelor $A = (\ldots, a, \ldots)$ este un şir finit. Cardinalul mulţimii N notat $|N| = n$, se numeşte *ordinul grafului* orientat G.

Un graf orientat $G = (N, A, g)$ se reprezintă grafic în modul următor:
i. fiecare nod $x \in N$ se reprezintă printr-un punct sau cerculeţ în plan;
ii. fiecare arc $a \in A,\ a = (x, y)$ se reprezintă printr-o linie care uneşte cele două noduri şi pe care se află o săgeată cu sensul de la x la y.

Exemplul 1.1. Graful din figura 1.1. este de ordinul 8.

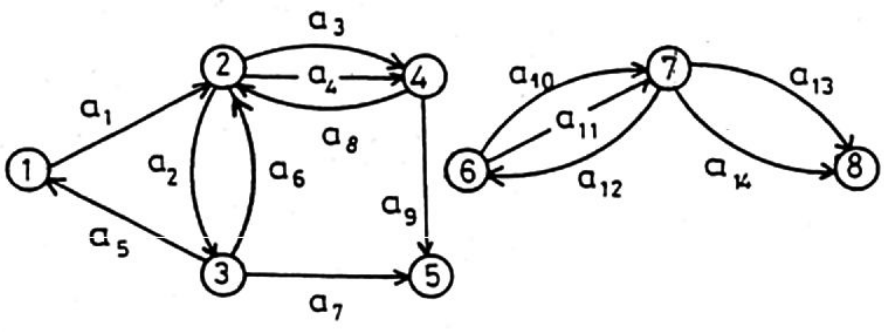


Fig.1.1

Observaţia 1.1. Reprezentarea grafică a unui graf orientat $G = (N, A, g)$ nu este unică. În primul rând nodurile se pot plasa în plan la întâmplare. În al doilea rând nu este obligatoriu ca arcele să fie segmente de dreaptă.
Exemplul 1.2. Cele trei grafuri din figura 1.2. reprezintă acelaşi graf.

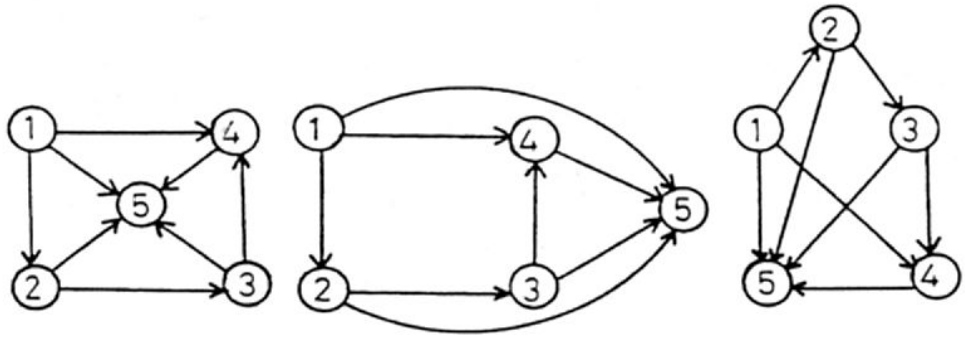


Fig.1.2

Definiţia 1.2. Două grafuri orientate, $G_1 = (N_1, A_1, g_1)$ şi $G_2 = (N_2, A_2, g_2)$ se numesc *izomorfe*, dacă există o bijecţie $\phi : N_1 \to N_2$ cu proprietatea că aplicaţia $\psi : A_1 \to A_2$, definită prin $\psi(x_1, y_1) = (\phi(x_1), \phi(y_1)), (x_1, y_1) \in A_1, (\phi(x_1), \phi(y_1)) = (x_2, y_2) \in A_2$, este o bijecţie.
Exemplul 1.3. Grafurile $G_1 = (N_1, A_1, g_1)$ şi $G_2 = (N_2, A_2, g_2)$ prezentate în figura 1.3 sunt izomorfe.

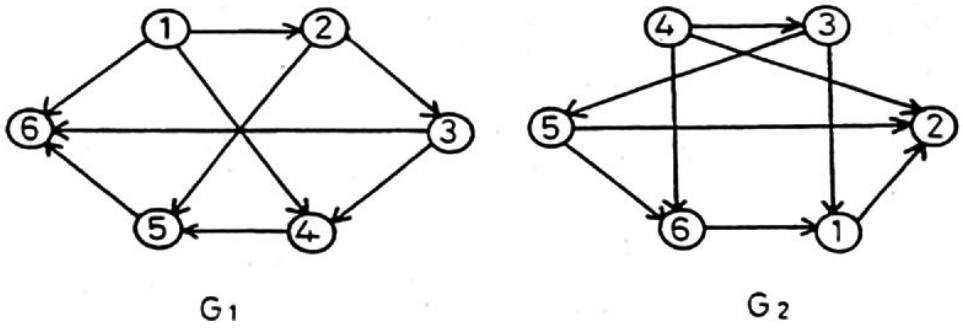


Fig.1.3

Dacă definim bijecţia ϕ prin $\phi(1) = 4, \phi(2) = 3, \phi(3) = 5, \phi(4) = 6, \phi(5) = 1, \phi(6) = 2$, atunci $\psi(1, 2) = (\phi(1), \phi(2)) = (4, 3)$, $\psi(1, 4) = (\phi(1), \phi(4)) = (4, 6)$, $\psi(1, 6) = (\phi(1), \phi(6)) = (4, 2)$ etc. este evident bijecţia $\psi : A_1 \to A_2$.
Definiţia 1.3. Dacă oricare pereche ordonată $(x, y) \in N \times N$ este imaginea a cel mult q, $q > 1$, elemente din A, atunci $G = (N, A, g)$ se numeşte *multigraf orientat.*

Exemplul 1.4. În figura 1.1. avem un multigraf orientat cu $q = 2$.

În paragrafele şi capitolele următoare ne vom ocupa de grafuri orientate cu $q = 1$. În acest caz funcţia g este injectivă şi familia A este o mulţime. Un astfel de graf se numeşte *digraf* şi se notează $G = (N, A)$.

Definiţia 1.4. Se numeşte *graf neorientat* un triplet $G = (N, A, g)$ format dintr-o mulţime N de elemente numite *noduri* sau *vârfuri*, dintr-o familie A de elemente numite *muchii* şi dintr-o aplicaţie $g : A \to \mathcal{P}_2(N)$ numită *funcţie de incidenţă*, prin care fiecărui element $a \in A$ i se asociază o pereche $\{x, y\} \in \mathcal{P}_2(N)$; dacă considerăm $g : A \to \mathcal{P}_{(2)}(N)$, unde: $\mathcal{P}_{(2)}(N) = \mathcal{P}_2(N) \cup \mathcal{P}_1(N)$, atunci aplicaţia g asociază fiecărui element $a \in A$, fie o pereche de noduri $\{x, y\} \in \mathcal{P}_2(N)$ care se notează $[x, y]$ sau $[y, x]$, fie un nod $\{x\} \in \mathcal{P}_1(N)$ care se notează $[x, x]$ şi se numeşte *buclă*, iar G se numeşte *graf general neorientat*.

În continuare vom presupune că graful neorientat G este finit, adică mulţimea nodurilor N este finită şi familia muchiilor A este un şir finit.

Un graf neorientat $G = (N, A, g)$ se reprezintă grafic la fel ca în cazul grafurilor orientate cu deosebirea că o muchie $a = [x, y]$ se reprezintă printr-o linie care uneşte cele două noduri fără săgeata care precizează sensul în cazul arcului. De asemenea, la fel ca în cazul grafurilor orientate se defineţe izomorfismul a două grafuri neorientate.

Definiţia 1.5. Dacă oricare pereche $[x, y] \in \mathcal{P}_2(N)$ este imaginea a cel mult $q, q > 1$, elemente din A atunci $G = (N, A, g)$ se numeţe *multigraf neorientat*.

Exemplul 1.5. Dacă se elimină săgeata de pe fiecare arc al grafului din figura 1.1, atunci fiecare arc (x, y) devine o muchie $[x, y]$ şi graful devine un multigraf neorientat cu $q = 3$.

În paragrafele şi capitolele următoare ne vom ocupa de grafuri neorientate cu $q = 1$. În acest caz funcţia g este injectivă şi familia A este o mulţime. Un graf neorientat cu $q = 1$ se numeşte *graf simplu* şi se notează $G = (N, A)$.

În majoritatea problemelor prezentate în continuare, se presupune că graful este orientat. De aceea, vom prezenta definiţiile pentru grafurile orientate. Definiţiile pentru grafurile neorientate se pot deduce, în cele mai multe cazuri, cu uşurinţă din definiţiile corespunzătoare pentru grafuri orientate. Totuşi, vom face unele precizări referitoare la unele definiţii pentru grafurile neorientate.

Definiţia 1.6. Fie un arc $a = (x, y) \in A$. Nodurile x, y se numesc *extremităţile arcului*, nodul x este *extremitatea iniţială* şi y *extremitatea finală*; nodurile x şi y se numesc *adiacente*.

Evident că, pentru o muchie $a = [x, y] \in A$, nu se pot preciza extremitatea iniţială şi extremitatea finală.

Definiţia 1.7. Fie un arc $a = (x, y) \in A$. Nodul x se numeşte *predecesor* al nodului y şi nodul y se numeşte *succesor* al nodului x. Mulţimea succesorilor nodului x este mulţimea $V^+(x) = \{y|(x, y) \in A\}$ şi mulţimea predecesorilor nodului x este mulţimea $V^-(x) = \{y|(y, x) \in A\}$. Mulţimea $V(x) = V^+(x) \cup V^-(x)$ se numeşte *vecinătatea* nodului x. Dacă $V(x) = \emptyset$, atunci x se numeşte *nod izolat*.

Exemplul 1.6. Pentru 2 - graful din figura 1.1. avem $V^+(4) = \{2, 5\}, V^-(4) = \{2\}, V(4) = \{2, 5\}$.

Definiţia 1.8. Se spune că nodul x este *adiacent* cu submulţimea $N' \subset N$, dacă $x \notin N'$ şi $x \in V(N'), V(N') = \cup\{V(y)|y \in N'\}$.

Observaţia 1.2. Conceptul de digraf se poate defini şi prin perechea $G = (N, \Gamma)$, unde $N = \{\ldots, x, \ldots\}$ este mulţimea nodurilor şi Γ aplicaţia multivocă $\Gamma : N \to \mathcal{P}(N), \Gamma(x) = V^+(x), x \in N$. Cele două definiţii sunt echivalente. Într-adevăr, dacă digraful este dat sub forma $G = (N, A)$, atunci $\Gamma(x) = V^+(x), x \in N$. Reciproc, dacă digraful este dat sub forma $G = (N, \Gamma), \Gamma(x) = V^+(x),\ x \in N$, atunci se determină $E^+(x) = \{(x, y)|y \in V^+(x)\}, x \in N$ şi $A = \cup\{E^+(x)|x \in N\}$. De asemenea, se defineşte $\Gamma^{-1}(x) = V^-(x), x \in N$. Recursiv avem

$$\Gamma^2(x) = \Gamma(\Gamma(x)), \ldots, \Gamma^k(x) = \Gamma\left(\Gamma^{k-1}(x)\right), \ldots$$

şi analog

$$\Gamma^{-2}(x) = \Gamma^{-1}(\Gamma^{-1}(x)), \ldots, \Gamma^{-k}(x) = \Gamma^{-1}\left(\Gamma^{-(k-1)}(x)\right), \ldots$$

Un nod $y \in \Gamma^k(x)$ se numeşte *descendent* al nodului x şi un nod $z \in \Gamma^{-k}(x)$ se numeşte *ascendent* al nodului x.

Exemplul 1.7. Fie digraful din figura 1.4.

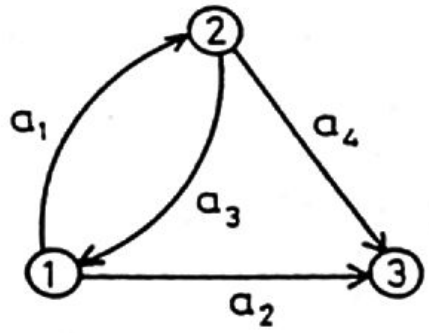


Fig.1.4

Dacă digraful este dat sub forma $G = (N, A)$ cu $N = \{1, 2, 3\}, A = \{a_1, a_2, a_3, a_4\} = \{(1, 2), (1, 3), (2, 1), (2, 3)\}$, atunci $\Gamma(1) = V^+(1) = \{2, 3\}, \Gamma(2) = V^+(2) = \{1, 3\}, \Gamma(3) = V^+(3) = \emptyset$ şi am obţinut digraful sub forma $G = (N, \Gamma)$.

Dacă digraful este dat sub forma $G = (N, \Gamma)$ cu $N = \{1, 2, 3\}, \Gamma(1) = \{2, 3\}, \Gamma(2) = \{1, 3\}, \Gamma(3) = \emptyset$, atunci $E^+(1) = \{(1, 2), (1, 3)\}, E^+(2) = \{(2, 1), (2, 3)\},\ E^+(3) = \emptyset$ şi $A = E^+(1) \cup E^+(2) \cup E^+(3) = \{(1, 2), (1, 3), (2, 1), (2, 3)\} = \{a_1, a_2, a_3, a_4\}$.

Definiţia 1.9. Două arce se numesc *adiacente* dacă au cel puţin o extremitate în comun.

Definiţia 1.10. Fie arcul $a = (x, y) \in A$. Se spune că arcul a este *incident către exterior* la nodul x şi *incident către interior* la nodul y. Mulţimea arcelor incidente către exterior la nodul x este $E^+(x) = \{a_i \mid a_i = (x, y) \in A\}$, mulţimea arcelor incidente către interior la nodul x este $E^-(x) = \{a_j \mid a_j = (y, x) \in A\}$ şi mulţimea arcelor incidente la nodul x este $E(x) = E^+(x) \cup E^-(x)$. Numărul $\rho^+(x) = |E^+(x)|$ se numeşte *semigradul exterior* al nodului x, numărul $\rho^-(x) = |E^-(x)|$ se numeşte *semigradul interior* al nodului x şi numărul $\rho(x) = \rho^+(x) + \rho^-(x)$ se numeşte *gradul nodului* x.

Exemplul 1.8. Se consideră multigraful orientat din figura 1.1. Avem $\rho^+(3) = 3, \rho^-(3) = 1$, deci $\rho(3) = 3 + 1 = 4$.

Dacă $G = (N, A, g)$ este un graf neorientat atunci $\rho(x) = \rho^+(x) = \rho^-(x)$. În cazul în care $G = (N, A, g)$ este multigraf orientat atunci $\rho^+(x) \geq |V^+(x)|, \rho^-(x) \geq |V^-(x)|$ şi în care $G = (N, A)$ este un digraf relaţiile sunt verificate cu egalităţi. Evident că dacă x este nod izolat atunci $\rho(x) = 0$. Un nod x cu $\rho(x) = 1$ se numeşte *perdant*. Dacă toate nodurile lui G au acelaşi grad ρ atunci G se numeşte *graf ρ - regulat*. Un graf 0-regulat se numeşte *graf nul* şi un graf 3-regulat se numeşte *graf trivalent* sau *cubic*.

Definiţia 1.11. Într-un graf orientat $G = (N, A, g)$ se numeşte *lanţ* de la nodul x_1 la nodul x_{k+1}, o secvenţă $L = (x_1, a_1, x_2, \ldots, x_k, a_k, x_{k+1}),\ x_i \in N,\ i = 1, \ldots, k+1, a_i \in A,\ i = 1, \ldots, k$, cu proprietatea că fiecare arc a_i este de forma (x_i, x_{i+1}) sau (x_{i+1}, x_i). Dacă $a_i = (x_i, x_{i+1})$, atunci a_i se numeşte *arc direct* al lanţului şi dacă $a_i = (x_{i+1}, x_i)$, atunci a_i se numeşte *arc invers* al lanţului. Numărul de arce din secvenţă este, prin definiţie, *lungimea lanţului* L. Lanţul L se numeşte *simplu* dacă fiecare arc a_i din secvenţă este utilizat o singură dată şi se numeşte *elementar* dacă fiecare nod x_i din secvenţă este utilizat o singură dată. Dacă $x_{k+1} = x_1$ atunci lanţul simplu L se numeşte *ciclu* şi se notează $\overset{\circ}{L}$.

Un lanţ poate fi reprezentat şi ca o secvenţă de arce $L = (a_1, \ldots, a_k)$ şi în cazul când $G = (N, A)$ este digraf ca o secvenţă de noduri $L = (x_1, \ldots, x_{k+1})$.

Exemplul 1.9. Se consideră graful orientat din figura 1.1. Secvenţa $L = (1, a_1, 2, a_8, 4, a_3, 2, a_6, 3, a_2, 2, a_8, 4) = (a_1, a_8, a_3, a_6, a_2, a_8)$ este un lanţ, dar nu este nici lanţ simplu, nici lanţ elementar. Secvenţa $L = (1, a_1, 2, a_3, 4, a_4, 2, a_2, 3, a_6, 2) = (a_1, a_3, a_4, a_2, a_6)$ este un lanţ simplu, dar nu este lanţ elementar. Secvenţa $L = (1, a_1, 2, a_6, 3, a_7, 5, a_9, 4) = (a_1, a_6, a_7, a_9)$ este un lanţ elementar.

Noţiunea de lanţ se poate defini şi într-un graf neorientat $G = (N, A, g)$ ca o secvenţă $L = (x_1, a_1, x_2, \ldots, x_k, a_k, x_{k+1})$ cu proprietatea că fiecare muchie a_i din secvenţă este de forma $a_i = [x_i, x_{i+1}]$ şi lanţul poate fi reprezentat şi sub forma $L = (a_1, \ldots, a_k)$. Un lanţ $L = (x_1, a_1, x_2, \ldots, x_k, a_k, x_{k+1})$, al grafului orientat $G = (N, A, g)$, în care fiecare arc a_i este arc direct, adică este de forma $a_i = (x_i, x_{i+1})$, se numeşte *lanţ orientat* sau *drum* şi se notează prin D. Un lanţ simplu orientat cu $x_{k+1} = x_1$ se numeşte *ciclu orientat* sau *circuit* şi se notează prin $\overset{\circ}{D}$. Noţiunile de drum şi circuit au sens numai pentru grafuri orientate.

Exemplul 1.10. Se consideră graful orientat din figura 1.1. Secvenţa $D = (1, a_1, 2, a_3, 4, a_8, 2, a_2, 3, a_6, 2, a_3, 4, a_8, 2) = (a_1, a_3, a_8, a_2, a_6, a_3, a_8)$ este un drum, dar nici drum simplu, nici drum elementar. Secvenţa $D = (1, a_1, 2, a_3, 4, a_8, 2, a_2, 3, a_7, 5) = (a_1, a_3, a_8, a_2, a_7)$ este un drum simplu, dar nu este un drum elementar. Secvenţa $D = (3, a_6, 2, a_4, 4, a_9, 5) = (a_6, a_4, a_9)$ este un drum elementar.

În paragrafele următoare vom considera, în general, lanţuri şi drumuri elementare, dar fără a mai specifica de fiecare dată atributul "elementar", ci numai în cazurile când este necesar.

Definiţia 1.12. Se spune că graful orientat $G' = (N', A', g')$ este un *subgraf* al grafului orientat $G = (N, A, g)$ dacă $N' \subseteq N$ şi $A' \subseteq A$. Dacă $N' \subseteq N$ şi $A' = (N' \times N') \cap A$ atunci $G' = (N', A', g')$ se numeşte *subgraf indus* în G de mulţimea de noduri N'. Dacă $N' = N$ şi $A' \subseteq A$ atunci $G' = (N', A', g')$ se numeşte *subgraf parţial* al lui G.

Concepte similare se pot defini în mod analog şi pentru grafuri neorientate.

Fie $G = (N, A)$ un digraf cu mulţimea nodurilor $N = \{1, \ldots, n\}$ şi mulţimea arcelor $A = \{1, \ldots, m\}$. *Matricea de adiacenţă* asociată digrafului G este matricea

$$M = (m_{ij})_{n \times n}$$

unde

$$m_{ij} = \begin{cases} 1 & \text{dacă} \quad (i,j) \in A \\ 0 & \text{dacă} \quad (i,j) \notin A \end{cases}$$

Evident că au loc relaţiile:

$$\rho^+(i) = \sum_{j=1}^{n} m_{ij}, i \in N, \quad \rho^-(j) = \sum_{i=1}^{n} m_{ij}, j \in N.$$

Matricea de incidenţă asociată digrafului G este matricea $\overline{M} = (\overline{m}_{ij})_{n \times m}$ unde

$$\overline{m}_{ij} = \begin{cases} 1 & \text{dacă} \quad j \in E^+(i) \\ -1 & \text{dacă} \quad j \in E^-(i) \\ 0 & \text{dacă} \quad j \notin E(i) \end{cases}$$

În acest caz au loc relaţiile:

$$\rho^+(i) = \sum_{j | \overline{m}_{ij} > 0} \overline{m}_{ij}, i \in N, \quad \rho^-(i) = \sum_{j | \overline{m}_{ij} < 0} |\overline{m}_{ij}|, i \in N.$$

Exemplul 1.11. Pentru digraful din figura 1.4. avem:

$$M = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \overline{M} = \begin{bmatrix} 1 & 1 & -1 & 0 \\ -1 & 0 & 1 & 1 \\ 0 & -1 & 0 & -1 \end{bmatrix}$$

Se remarcă faptul că matricea de incidenţă $\overline{M}$ are pe fiecare coloană un 1, un -1, şi $n - 2$ zerouri.

Dacă $G = (N, A)$ este un graf simplu neorientat atunci

$$m_{ij} = \begin{cases} 1 & \text{dacă} \quad [i,j] \in A, \\ 0 & \text{dacă} \quad [i,j] \notin A \end{cases}$$

şi

$$\overline{m}_{ij} = \begin{cases} 1 & \text{dacă} \quad j \in E(i), \\ 0 & \text{dacă} \quad j \notin E(i) \end{cases}$$

Evident că în cazul unui graf simplu neorientat matricea de adiacentă M este simetrică, adică $m_{ij} = m_{ji}$ pentru $i = 1, \ldots, n$ şi $j = 1, \ldots, n$.

1.2 Clase de grafuri

O clasă de grafuri este alcătuită din grafuri cu proprietăţi particulare.

Definiţia 1.13. Se spune că digraful $G = (N, A)$ este *simetric* dacă oricare ar fi $(x, y) \in A$ implică $(y, x) \in A$.

Cu alte cuvinte, digraful $G = (N, A)$ este simetric dacă orice pereche de noduri adiacente este legată prin arce în ambele sensuri.

Definiţia 1.14. Se spune că digraful $G = (N, A)$ este *antisimetric* dacă oricare ar fi $(x, y) \in A$ implică $(y, x) \notin A$.

Deci, digraful $G = (N, A)$ este antisimetric dacă orice pereche de noduri adiacente este legată cel mult printr-un singur arc.

Exemplul 1.12. Digraful reprezentat în figura 1.5(a) este simetric şi digraful reprezentat în figura 1.5(b) este antisimetric.

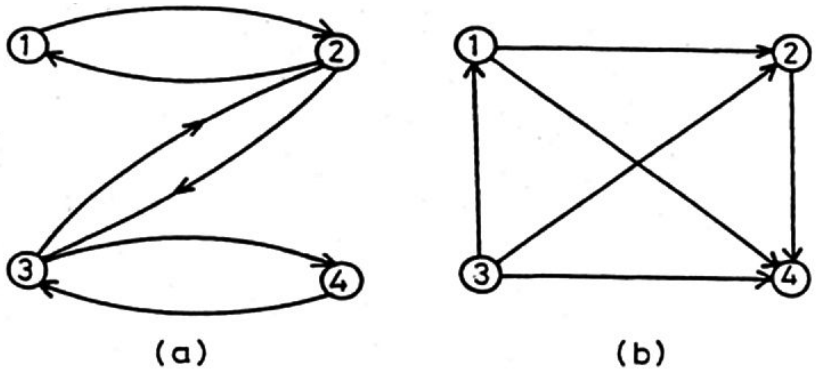


Fig.1.5

Definiţia 1.15. Se spune că digraful $G = (N, A)$ este *pseudosimetric* dacă $\rho^+(x) = \rho^-(x)$ pentru fiecare nod $x \in N$.

Orice digraf simetric este şi pseudosimetric. Reciproca nu este adevărată.

Exemplul 1.13. Digraful reprezentat în figura 1.5(a) este simetric deci este şi pseudosimetric, iar cel reprezentat în figura 1.5(b) este antisimetric, dar nu este pseudosimetric. Digraful reprezentat în figura 1.6(a) este pseudosimetric, dar nu este simetric şi nici antisimetric, iar cel reprezentat în figura 1.6(b) este antisimetric şi pseudosimetric.

Definiţia 1.16. Se spune că digraful $G = (N, A)$ este *nesimetric* dacă există un arc $(x, y) \in A$ pentru care $(y, x) \notin A$.

Cu alte cuvinte, digraful $G = (N, A)$ este nesimetric dacă nu este simetric. Orice graf antisimetric este nesimetric. Reciproca nu este adevărată. Orice graf pseudosimetric care nu este simetric este nesimetric. Reciproca nu este adevărată.

Definiţia 1.17. Se spune că digraful $G = (N, A)$ este *complet* dacă oricare ar fi $(x, y) \notin A$ implică $(y, x) \in A$.

Un graf simplu neorientat $G = (N, A)$ cu n noduri care este complet se notează cu K_n şi are $n(n-1)/2$ muchii.

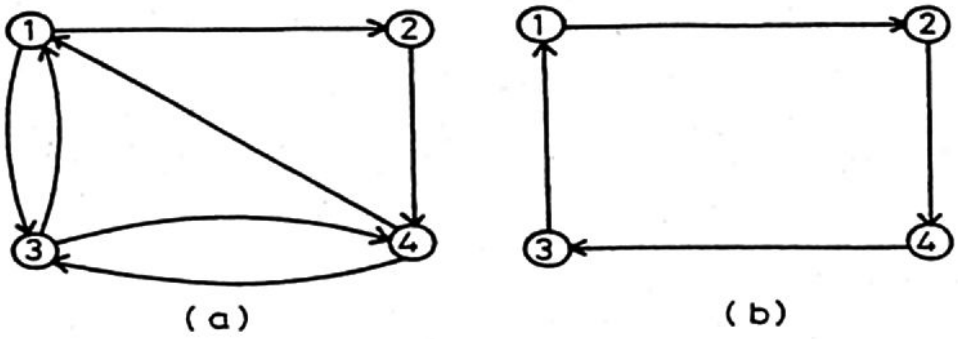


Fig.1.6

Exemplul 1.14. Digraful din figura 1.5(b) este antisimetric deci este nesimetric şi digrafurile reprezentate în figura 1.6 sunt pseudosimetrice care nu sunt simetrice, deci sunt nesimetrice. Digraful din figura 1.7(a) este nesimetric, dar nu este nici antisimetric şi nici pseudosimetric.

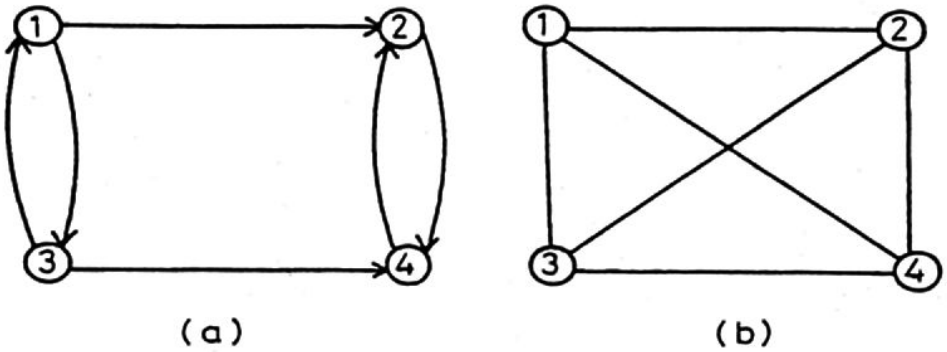


Fig.1.7

Digraful reprezentat în figura 1.5(b) este complet. Graful K_4 este reprezentat în figura 1.7(b).

Definiţia 1.18. Se spune că digraful $G = (N, A)$ este *graf turneu* dacă este antisimetric şi complet.

Denumirea de graf turneu se justifică prin faptul că un astfel de graf poate reprezenta un turneu sportiv în care fiecare jucător (echipă) joacă câte un meci cu toţi (toate) ceilalţi (celelalte) jucători (echipe).

Definiţia 1.19. Se spune că digraful $G = (N, A)$ este *tranzitiv* dacă oricare ar fi $(x, y) \in A$ şi $(y, z) \in A$ implică $(x, z) \in A$.

Exemplul 1.15. Digraful reprezentat în figura 1.5.(b) este un graf turneu şi de asemenea este tranzitiv.

Definiţia 1.20. Se spune că digraful $G = (N, A)$ este o *clică* dacă este simetric şi complet.

Denumirea de clică se justifică prin faptul că un digraf simetric şi complet poate reprezenta o coaliţie sociologică, politică etc. Noţiunea de clică are sens şi pentru grafuri neorientate, astfel graful K_n este o clică.

Definiţia 1.21. Se spune că digraful $G = (N, A)$ este *bipartit* dacă mulţimea nodurilor N admite o partiţie în două submulţimi N_1 şi $N_2 (N_1 \neq \emptyset, N_2 \neq \emptyset, N_1 \cap N_2 = \emptyset, N_1 \cup N_2 = N)$, astfel încât orice arc $(x, y) \in A$ are una dintre extremităţi în N_1, iar cealaltă extremitate în N_2.

Un digraf bipartit se notează $G = (N_1, N_2, A)$ şi se caracterizează prin inexistenţa ciclurilor care conţin un număr impar de arce. Un graf neorientat bipartit şi complet $G = (N_1, N_2, A)$ cu $|N_1| = n_1$ şi $|N_2| = n_2$ se notează K_{n_1,n_2}.

Exemplul 1.16. Digraful reprezentat în figura 1.8(a) este o clică. În figura 1.8.(b) este reprezentat graful $K_{2,3}$.

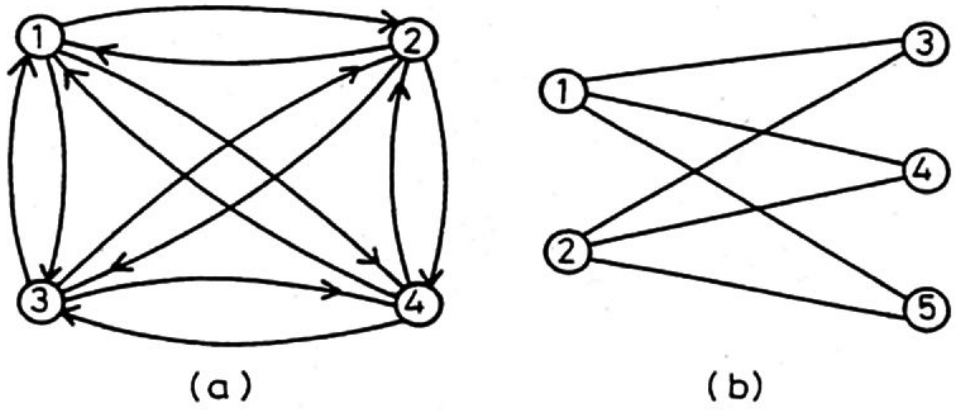


Fig.1.8

Definiţia 1.22. Se spune că digraful $G = (N, A)$ este *planar* dacă este posibil să-l reprezentăm pe un plan astfel încât oricare două arce să se întâlnească eventual numai în extremităţile lor.

Analog se defineşte un graf simplu neorientat planar. Arcele (muchiile) unui graf planar $G = (N, A)$ determină contururi care mărginesc regiuni numite *feţe*. Există o singură regiune din plan fără contur numită *faţă nemărginită*.

Exemplul 1.17. Digraful reprezentat în figura 1.5(b) este planar, deoarece poate fi reprezentat ca în figura 1.9.

Feţele 1,2,3 sunt feţele mărginite, iar faţa 4 este faţa nemărginită. Exemple de grafuri neplanare minimale sunt K_5 şi $K_{3,3}$.

O altă clasă de grafuri este alcătuită din grafuri asociate unui graf dat.

În paragraful 1.1. s-a definit subgraful $G' = (N', A')$ al unui graf dat $G = (N, A)$.

Definiţia 1.23. Se spune că digraful $\overline{G} = (\overline{N}, \overline{A})$ este *complementarul* digrafului $G = (N, A)$ dacă $\overline{N} = N$ şi $\overline{A} = \{(x,\ y) \mid x, y \in N,\ x \neq y,\ (x, y) \notin A\}$.

Analog se defineşte complementarul unui graf simplu neorientat. Operaţia de complementare este involutivă, adică complementarul complementarului este graful iniţial.

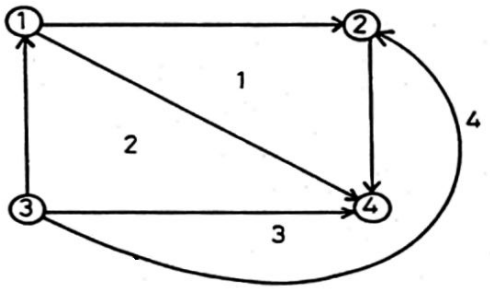


Fig.1.9

Exemplul 1.18. Digraful reprezentat în figura 1.10(b) este complementarul digrafului reprezentat în figura 1.10.(a)

Definiţia 1.24. Se spune că digraful $G^{-1} = (N^{-1}, A^{-1})$ este *inversul* digrafului $G = (N, A)$ dacă $N^{-1} = N$ şi A^{-1} se obţine din A prin inversarea sensului arcelor.

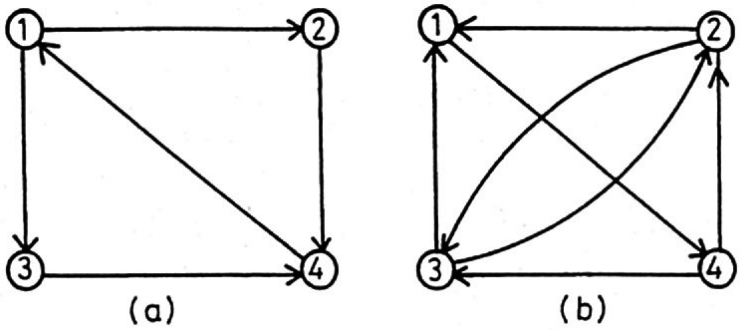


Fig.1.10

Operaţia de inversare este involutivă, adică inversul inversului este graful iniţial ($(G^{-1})^{-1} = G$). Graful autoreciproc ($G^{-1} = G$) este simetric.

Exemplul 1.19. Digraful reprezentat în figura 1.11(b) este inversul digrafului reprezentat în figura 1.11(a).

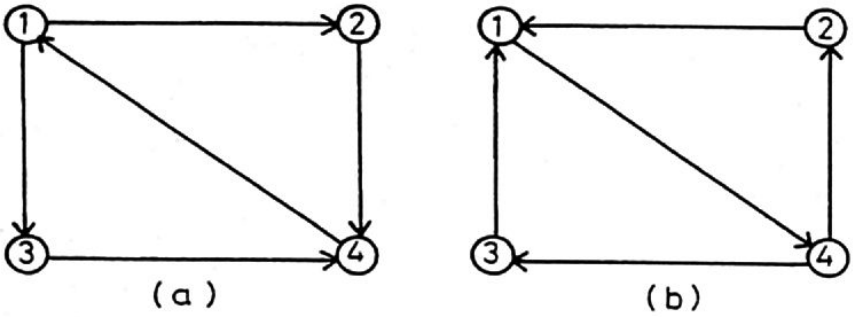


Fig.1.11

1.3 Operaţii cu grafuri

Definiţia 1.25. *Suma cartezianǎ* a două digrafuri $G_1 = (N_1, A_1)$ şi $G_2 = (N_2, A_2)$ este notată $G_1 + G_2$ şi este digraful $G = (N, A)$ definit astfel:

$$N = N_1 \times N_2 = \{x_1x_2 | x_1 \in N_1, x_2 \in N_2\},$$
$$A = \{(x_1x_2, y_1y_2) | x_1, y_1 \in N_1, x_2, y_2 \in N_2,$$
$$x_1 = y_1 \text{ şi } (x_2, y_2) \in A_2 \text{ sau } x_2 = y_2 \text{ şi} (x_1, y_1) \in A_1\}.$$

Definiţia 1.26. *Produsul cartezian* al două digrafuri $G_1 = (N_1, A_1)$ şi $G_2 = (N_2, A_2)$ este notat $G_1 \times G_2$ şi este digraful $G = (N, A)$ definit astfel:

$$N = N_1 \times N_2 = \{x_1x_2 | x_1 \in N_1, x_2 \in N_2\},$$
$$A = \{(x_1x_2, y_1y_2) | x_1, y_1 \in N_1, x_2, y_2 \in N_2,$$
$$(x_1, y_1) \in A_1 \quad \text{şi } (x_2, y_2) \in A_2\}.$$

Suma cartezianǎ şi produsul cartezian a două grafuri simple neorientate se definesc analog ca pentru două digrafuri. De asemenea suma cartezianǎ şi produsul cartezian a p grafuri, $p > 2$, se definesc asemănător ca pentru două grafuri. **Exemplul 1.20.** Digraful reprezentat în figura 1.13(a) este suma cartezianǎ a digrafurilor reprezentate în figura 1.12.

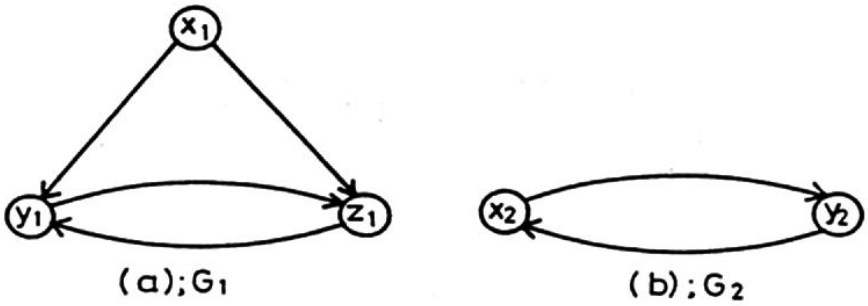


Fig.1.12

Arcul $(x_1x_2, x_1y_2) \in A$ deoarece $x_1 = x_1$ şi $(x_2, y_2) \in A_2$; arcul $(x_1x_2, y_1x_2) \in A$ deoarece $x_2 = x_2$ şi $(x_1, y_1) \in A_1$ etc. Digraful reprezentat în figura 1.13(b) este produsul cartezian al digrafurilor reprezentate în figura 1.12.

Arcul $(x_1x_2, z_1y_2) \in A$ deoarece $(x_1, z_1) \in A_1$ şi $(x_2, y_2) \in A_2$; arcul $(y_1y_2, z_1x_2) \in A$ deoarece $(y_1, z_1) \in A_1$ şi $(y_2, x_2) \in A_2$ etc.

1.4 Structuri de date utilizate în reprezentarea grafurilor

Performanţa unui algoritm utilizat pentru rezolvarea unei probleme din teoria grafurilor depinde nu numai de structura algoritmului utilizat ci şi de modul de reprezentare în calculator a topologiei grafului.

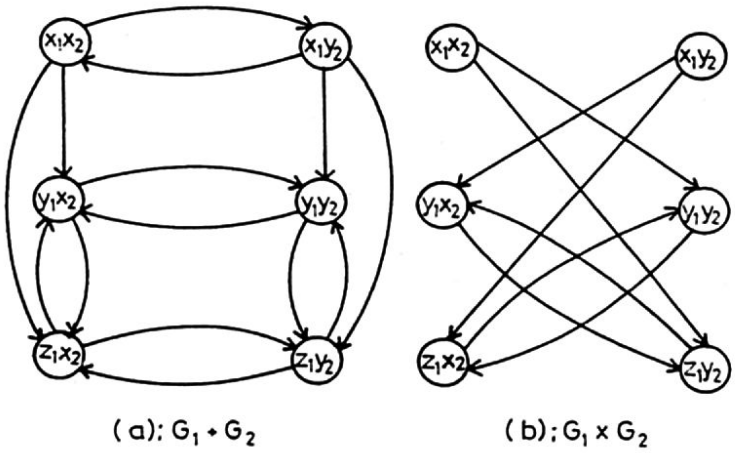


Fig.1.13

Fie un digraf $G = (N, A)$ cu $N = \{1, \ldots, n\}$ şi $A = \{1, \ldots, m\}$. Se consideră funcţia valoare $b : A \to \Re$ şi funcţia capacitate $c : A \to \Re^+$. Funcţia valoare b reprezintă fie funcţia lungime, fie funcţia cost etc. Funcţiile b, c etc. se pot defini şi pe mulţimea nodurilor.

Definiţia 1.27. Un digraf $G = (N, A)$ pe care s-a/s-au definit funcţia b sau/şi funcţia c se numeşte *reţea orientată* şi se notează fie $G = (N, A, b)$, fie $G = (N, A, c)$, respectiv $G = (N, A, b, c)$.

Analog se defineşte o *reţea neorientată*.

În acest paragraf se vor prezenta cele mai uzuale structuri de date utilizate pentru reprezentarea reţelelor. Pentru reprezentarea unei reţele este necesar, în general, să memorăm două tipuri de informaţii:

a) informaţii privind topologia reţelei;

b) informaţii privind funcţia b sau/şi funcţia c.

În acest paragraf se prezintă reprezentările reţelelor orientate. Reprezentările corespunzătoare reţelelor neorientate sunt similare cu reprezentările reţelelor orientate. La sfârşitul acestui paragraf sunt prezentate pe scurt şi reprezentările reţelelor neorientate.

Reprezentarea prin *matricea de adiacenţă* constă în memorarea matricei de adiacenţă

$$M = (m_{ij})_{n \times n}$$

unde

$$m_{ij} = \begin{cases} 1 & \text{dacă} \quad (i,j) \in A \\ 0 & \text{dacă} \quad (i,j) \notin A \end{cases}$$

Dacă este necesar să memorăm funcţia b sau/şi funcţia c, atunci memorăm matricea valoare

$$B = (b_{ij})_{n \times n}$$

unde

$$b_{ij} = \begin{cases} b(i,j) & \text{dacă} \quad (i,j) \in A, \\ 0 & \text{dacă} \quad (i,j) \notin A \end{cases}$$

sau/şi matricea capacitate

$$C = (c_{ij})_{n \times n}$$

unde

$$c_{ij} = \begin{cases} c(i,j) & \text{dacă} \quad (i,j) \in A, \\ 0 & \text{dacă} \quad (i,j) \notin A \end{cases}$$

Reprezentarea prin *matricea de incidenţă* constă în memorarea matricei de incidenţă

$$\overline{M} = (\overline{m}_{ij})_{n \times m}$$

unde

$$\overline{m}_{ij} = \begin{cases} 1 & \text{dacă} \quad j \in E^+(i), \\ -1 & \text{dacă} \quad j \in E^-(i), \\ 0 & \text{dacă} \quad j \notin E(i) \end{cases}$$

Exemplul 1.21. Considerăm reţeaua orientată din figura 1.14.

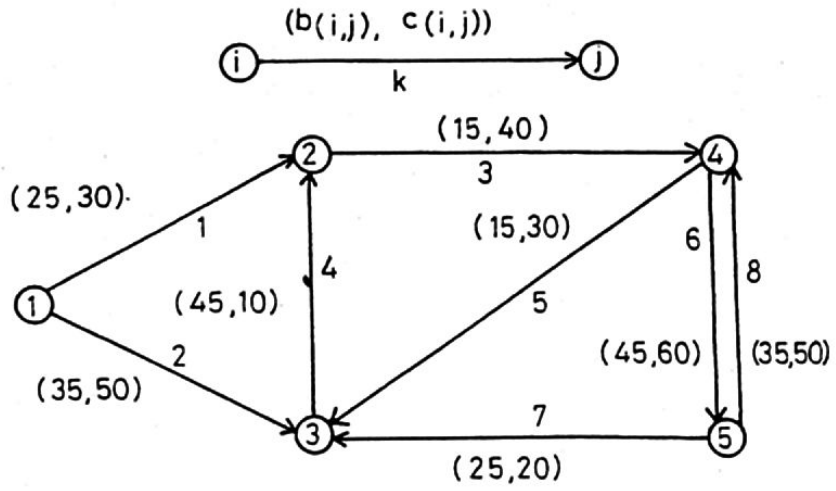


Fig.1.14

Matricele $M, B, C, \overline{M}$ sunt următoarele:

$$M = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 25 & 35 & 0 & 0 \\ 0 & 0 & 0 & 15 & 0 \\ 0 & 45 & 0 & 0 & 0 \\ 0 & 0 & 15 & 0 & 45 \\ 0 & 0 & 25 & 35 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 30 & 50 & 0 & 0 \\ 0 & 0 & 0 & 40 & 0 \\ 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 30 & 0 & 60 \\ 0 & 0 & 20 & 50 & 0 \end{bmatrix}, \overline{M} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 \end{bmatrix}$$

Matricele M, B, C au fiecare n^2 elemente şi numai m elemente nenule. Matricea $\overline{M}$ are nm elemente şi numai $2m$ elemente nenule (pe fiecare coloană câte 2 elemente nenule, un 1 şi un -1). Rezultă că reprezentarea prin matrice asociate reţelei este eficientă numai în cazul reţelelor dense ($m \gg n$). Pentru reţelele rare ($m \ll n^2$) este eficient să utilizăm liste de adiacenţă sau liste de incidenţă.

Lista nodurilor adiacente către exterior nodului i este $V^+(i) = \{j|(i,j) \in A\}$ şi lista arcelor incidente către exterior nodului i este $E^+(i) = \{(i,j)|j \in V^+(i)\}$.

Considerăm că listele $V^+(i)$ şi $E^+(i)$ sunt ordonate, şi avem $|V^+(i)| = |E^+(i)| = \rho^+(i),\ i = 1, \ldots, n$.

Reprezentarea prin *liste de adiacenţă* poate fi utilizată în una din următoarele două variante:

- reprezentarea cu ajutorul tablourilor;
- reprezentarea cu ajutorul listelor simplu înlănţuite.

Reprezentarea cu ajutorul *tablourilor* constă în a defini două tablouri unidimensionale. Primul tablou, notat P, are $n+1$ elemente şi reprezintă o listă de pointeri. Tabloul P se defineşte astfel:$P(1) = 1, P(i+1) = P(i) + \rho^+(i), \quad i = 1, \ldots, n$. Al doilea tablou, notat V, are m elemente şi reprezintă o listă de noduri. Tabloul V conţine nodurile j din lista $V^+(i)$, în ordinea stabilită, între locaţiile $P(i)$ şi $P(i+1) - 1, \quad i = 1, \ldots, n$.

Reprezentarea cu ajutorul tablourilor poate fi realizată şi prin utilizarea unui tablou bidimensional

$$T = \begin{bmatrix} T(1,1) & \ldots & T(1,p) \\ T(2,1) & \ldots & T(2,p) \end{bmatrix}$$

unde $p = n+m$. Elementele $T(1,k), \quad k = 1, \ldots, p$ sunt noduri şi elementele $T(2,k), \quad k = 1, \ldots, p$ sunt pointeri. Prima linie a tabloului T se defineşte astfel:

$$T(1,k) = \begin{cases} k, & \text{pentru} \quad k = 1, \ldots, n, \\ V(k-n), & \text{pentru} \quad k = n+1, \ldots, p \end{cases}$$

Elementul $T(1,k)$ reprezintă:

- nodul k, dacă $1 \leq k \leq n$,
- un nod j din $V^+(i)$, dacă $n + P(i) \leq k \leq n + P(i+1),\ i = 1, \ldots, n$.

A doua linie a tabloului T se defineşte astfel:

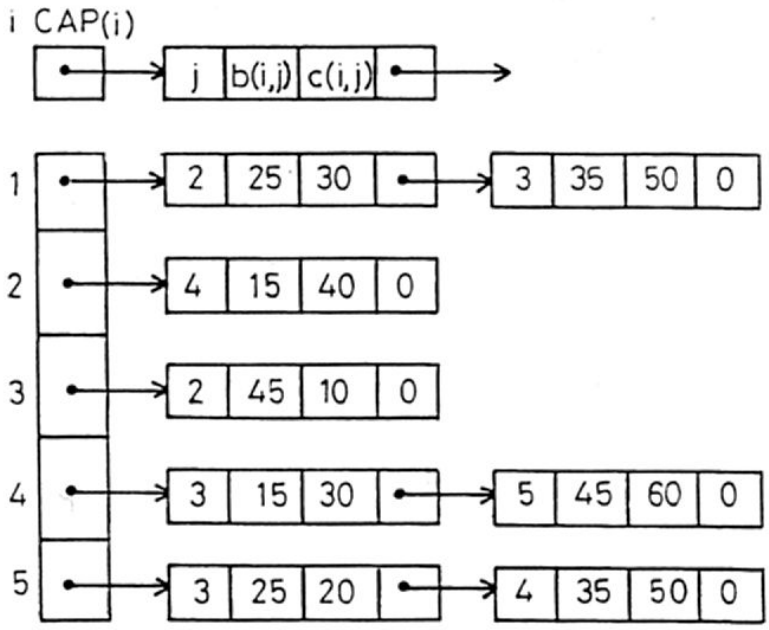
$$T(2,k) = \begin{cases} n + P(k), & \text{dacă} \quad V^+(k) \neq \emptyset, \text{ pentru} \quad k = 1, \ldots, n \\ 0, & \text{dacă} \quad V^+(k) = \emptyset, \text{ pentru} \quad k = 1, \ldots, n \\ k+1, & \text{dacă} \quad T^(1,k) = j \text{ nu este ultimul nod din } V^+(i) \\ & \text{pentru} \quad k = n + P(i), \ldots, n + P(i+1) - 1,\ i = 1, \ldots, n \\ 0, & \text{dacă} \quad T^(1,k) = j \text{ este ultimul nod din } V^+(i) \\ & \text{pentru} \quad k = n + P(i), \ldots, n + P(i+1) - 1,\ i = 1, \ldots, n \end{cases}$$

Elementul $T(2,k) \neq 0$ reprezintă adresa (coloana) din $T(1,k')$, adică $k' = T(2,k)$:
- a primului nod j din $V^+(k)$, dacă $1 \leq k \leq n$,
- a următorului nod j' a lui $j = T(1,k)$ din $V^+(i)$, dacă $n + P(i) \leq k < n + P(i+1) - 1$, $i = 1, \ldots, n$.

Reprezentarea cu ajutorul *listelor simplu înlănţuite* constă din n liste, fiecare listă corespunde la un nod i şi are $\rho^+(i)$ locaţii. Fiecare locaţie corespunde unui arc (i,j) şi conţine mai multe câmpuri: un câmp pentru memorarea nodului j, un câmp pentru memorarea pointerului care indică legătura la următoarea locaţie şi eventual un câmp sau două câmpuri pentru $b(i,j)$ sau/şi $c(i,j)$. Dacă locaţia este ultima din lista înlănţuită atunci prin convenţie în câmpul pointerului punem valoarea 0. Deoarece avem n liste, una pentru fiecare nod i cu $\rho^+(i)$ locaţii, este necesar un tablou unidimensional, notat CAP, cu n elemente ce conţine pointeri care indică prima locaţie din fiecare listă înlănţuită. Dacă $\rho^+(i) = 0$ atunci $CAP(i) = 0$.

Exemplul 1.22. Considerând reţeaua din figura 1.14 avem următoarele reprezentări cu ajutorul tablourilor şi cu ajutorul listelor simplu înlăţuite:
$V^+(1) = \{2,3\}$, $\rho^+(1) = 2, V^+(2) = \{4\}$, $\rho^+(2) = 1$, $V^+(3) = \{2\}$, $\rho^+(3) = 1$, $V^+(4) = \{3,5\}, \rho^+(4) = 2, V^+(5) = \{3,4\}, \rho^+(5) = 2; P = (1,3,4,5,7,9), V = (2,3,4,2,3,5,3,4); n = 5, m = 8, n + m = 5 + 8 = 13.$

$$T = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 2 & 3 & 4 & 2 & 3 & 5 & 3 & 4 \\ 6 & 8 & 9 & 10 & 12 & 7 & 0 & 0 & 0 & 11 & 0 & 13 & 0 \end{bmatrix}$$



Reprezentarea prin liste de adiacenţă revine la descrierea matricei de adiacenţă linie cu linie. În mod similar se poate descrie matricea de incidenţă coloană cu coloană utilizând liste de incidenţă.

Reprezentarea prin *liste de incidenţă* constă în utilizarea tabloului unidimensional P al pointerilor şi a unui tablou bidimensional notat E. Dacă reprezentăm o reţea $G = (N, A, b)$ sau o reţea $G = (N, A, c)$ atunci tabloul E are trei linii şi m coloane şi dacă reprezentăm reţeaua $G = (N, A, b, c)$ atunci tabloul E are patru linii şi m coloane. Fiecare coloană k, corespunde unui arc (i, j) şi anume $E(1, k) = i, E(2, k) = j, E(3, k) = b(i, j)$ sau $c(i, j)$ dacă reţeaua este $G = (N, A, b)$ sau $G = (N, A, c)$ şi $E(1, k) = i, E(2, k) = j, E(3, k) = b(i, j), E(4, k) = c(i, j)$ dacă reţeaua este $G = (N, A, b, c)$. Tabloul E conţine arcele din $E^+(i)$, în ordinea stabilită, şi informaţiile asociate acestor arce, de la coloana $P(i)$ inclusiv până la coloana $P(i + 1) - 1$ inclusiv.
Exemplul 1.23. Considerăm reţeaua reprezentată în figura 1.14 Tablourile P şi E sunt următoarele:

$$P = (1, 3, 4, 5, 7, 9)$$

$$E = \begin{bmatrix} 1 & 1 & 2 & 3 & 4 & 4 & 5 & 5 \\ 2 & 3 & 4 & 2 & 3 & 5 & 3 & 4 \\ 25 & 35 & 15 & 45 & 15 & 45 & 25 & 35 \\ 30 & 50 & 40 & 10 & 30 & 60 & 20 & 50 \end{bmatrix}$$

Un alt avantaj al reprezentării prin liste de adiacenţă sau liste de incidenţă este că permite reprezentarea reţelelor care conţin arce paralele, adică arce care au aceeaşi extremitate iniţială şi aceeaşi extremitate finală.

Reprezentarea prin liste de adiacenţă este avantajoasă când este implementată într-un limbaj care are posibilitatea să lucreze cu liste înlănţuite (PASCAL, C). În acest caz topologia reţelei se poate modifica uşor. Reprezentarea prin liste de incidenţă are avantajul că are nevoie de mai puţină memorie decât reprezentarea prin liste de adiacenţă. Alegerea unei reprezentări depinde de problema de rezolvat şi de limbajul ales.

O reţea neorientată $G = (N, A, b, c)$ se transformă într-o reţea orientată $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{b}, \widetilde{c})$ cu $\widetilde{N} = N, \widetilde{A} = \{(i, j), (j, i) \,|\, [i, j] \in A\}$, funcţiile $\widetilde{b}$ şi $\widetilde{c}$ sunt definite în raport cu problema de rezolvat şi $\widetilde{G}$ se reprezintă prin una din metodele descrise mai sus.

1.5 Noţiuni de complexitatea algoritmilor în teoria grafurilor

În acest paragraf se va nota cu $\lceil r \rceil$ cel mai mic întreg mai mare sau egal cu r pentru $r \in \Re$.

Fie un digraf $G = (N, A)$ cu $|N| = n$ noduri şi $|A| = m$ arce. Se consideră funcţia valoare $b : A \to \Re$ şi funcţia capacitate $c : A \to \Re$. Funcţia valoare b reprezintă fie funcţia lungime, fie funcţia cost etc. Fie $\overline{b} = max\{b(x, y)|(x, y) \in A\}$ şi $\overline{c} = max\{c(x, y)|(x, y) \in A\}$.
Definiţia 1.28. Prin *problemă* se înţelege o funcţie total definită $P : I \to F$, unde I este mulţimea informaţiilor iniţiale (datele de intrare ale problemei) şi F este mulţimea informaţiilor finale (datele de ieşire ale problemei).

Se presupune că mulţimile I şi F sunt nevide şi cel mult numărabile.

Exemplul 1.24. În reţeaua $G = (N, A, b)$ se poate formula problema drumului minim, iar în reţeaua $G = (N, A, c)$ problema fluxului maxim.

Definiţia 1.29. Se numeşte *instanţă* a problemei P, precizarea problemei $P(i)$ pentru o valoare specificată $i \in I$.

Pentru o instanţă $P(i)$ se va folosi şi notaţia p, iar prin abuz de notaţie uneori vom scrie $p \in P$.

Exemplul 1.25. În problema drumului minim în reţeaua $G = (N, A, b)$, pentru a defini o instanţă a acestei probleme este necesar să specificăm topologia digrafului $G = (N, A)$, nodurile sursă şi destinaţie şi funcţia b.

Un *algoritm* este o procedură care rezolvă pas cu pas o instanţă $p, p \in P$.

Definiţia 1.30. Se spune că *algoritmul α rezolvă problema P* dacă algoritmul determină soluţia oricărei instanţe $p \in P$.

Un algoritm α care rezolvă instanţa p va porni de la o codificare a informaţiei iniţiale $i \in I$ şi va furniza o codificare a rezultatului.

Definiţia 1.31. Se numeşte *dimensiunea problemei P* un număr natural notat $d(p), p \in P$, care reprezintă lungimea unei codificări binare a informaţiei iniţiale.

În general, $d(p)$ este un număr natural care reprezintă dimensiunea structurală a informaţiei iniţiale, deoarece lungimea codificării binare a unei informaţii iniţiale va fi mărginită de o funcţie având ca argument dimensiunea sa structurală.

Exemplul 1.26. Să considerăm o problemă care are ca dată de intrare un digraf $G = (N, A)$ reprezentat prin matricea sa de adiacenţă M. Pentru această problemă trebuie $\lceil \log n \rceil$ biţi pentru codul numărului n şi n^2 biţi pentru reprezentarea matricei. Dimensiunea problemei, este $\lceil \log n \rceil + n^2$. Pentru n suficient de mare se poate considera dimensiunea problemei egală cu n^2.

Exemplul 1.27. Să considerăm o problemă care are ca dată de intrare, o reţea $G = (N, A, b)$. Presupunem că pentru reprezentarea digrafului $G = (N, A)$ se folosesc listele de adiacenţă P (lista de pointeri) şi V (lista nodurilor). Dacă notăm cu $\rho^+(i)$ semigradul exterior al nodului $i \in N$, atunci lista P este un tablou unidimensional ce are $n + 1$ elemente cu $P(1) = 1$ şi $P(i+1) = P(i) + \rho^+(i), \quad i = 1, 2, \ldots, n$. Lista V este un tablou unidimensional ce are m elemente şi $V(P(i))$ până la $V(P(i+1) - 1)$ conţin succesorii nodului i. Rezultă $1 \leq P(i) \leq m + 1,\ i = 1, \ldots, n + 1$ şi $1 \leq V(i) \leq n,\ i = 1, \ldots, m$. Deci pentru a descrie această problemă sunt necesari:

$\lceil \log n \rceil$ biţi pentru codificarea lui n;
$\lceil \log m \rceil$ biţi pentru codificarea lui m;
$(n + 1)\lceil \log(m + 1) \rceil$ biţi pentru codificarea elementelor tabloului P;
$m\lceil \log n \rceil$ biţi pentru codificarea elementelor tabloului V;
$m\lceil \log \overline{b} \rceil$ biţi pentru codificarea valorilor numerice $b(x, y), (x, y) \in A$.

Rezultă că dimensiunea problemei este $\lceil \log n \rceil + \lceil \log m \rceil + (n + 1)\lceil \log(m + 1) \rceil + m\lceil \log n \rceil + m\lceil \log \overline{b} \rceil$. Pentru m şi n suficienţi de mari se poate considera că problema are dimensiunea $m\lceil \log n \rceil$.

În continuare se vor prezenta diferite abordări ale complexităţii unui algoritm.

Resursele de calcul asociate execuţiei unui algoritm sunt spaţiul de memorie şi timpul necesar de execuţie. Ne vom ocupa numai de resursa timp, deoarece progresele tehnologice din ultima perioadă conduc la o scădere a importanţei memoriei folosite de algoritm.

Vom nota $T'_\alpha(p)$ timpul de execuţie necesar algoritmului α pentru rezolvarea instanţei $p, p \in P$. Acest timp reprezintă numărul paşilor efectuaţi de algoritm pentru rezolvarea instanţei p. Un pas al unui algoritm α este o operaţie elementară (instrucţiune elementară). În general, operaţiile elementare sunt: operaţii de atribuire, operaţii aritmetice (+, -, $\star$, /), operaţii logice (comparaţii). Se presupune, că execuţia unei instrucţiuni elementare nu depinde de mărimea operanzilor şi de timpul de memorare a rezultatelor. Aceasta înseamnă că resursa timp este studiată independent de sistemul de calcul pe care se face implementarea algoritmului.

În literatura de specialitate există trei tipuri de abordări ale complexităţii unui algoritm α: analiza empirică, analiza cazului mediu şi analiza cazului cel mai defavorabil.

Obiectivul *analizei empirice* constă în studierea comportării în practică a unui algoritm. Se scrie un program pentru algoritmul respectiv şi se testează performanţele programului pe diferite clase de instanţe ale problemei. Această analiză are câteva dezavantaje majore: (1) performanţele unui program depind de limbajul de programare, de calculatorul folosit şi de priceperea programatorului care a scris programul; (2) de obicei această analiză este mare consumatoare de timp şi este scumpă; (3) compararea algoritmilor prin această analiză poate să conducă la rezultate eronate.

Obiectivul *analizei cazului mediu* constă în studierea comportării în medie a unui algoritm, care presupune rezolvarea următoarelor două etape: (a) precizarea unei distribuţii de probabilitate pe mulţimea instanţelor $p, p \in P$; (b) determinarea mediei variabilei aleatoare $T'_\alpha(p), T^m_\alpha(k) = M(\{T'_\alpha(p) \mid p \in P, d(p) = k\})$.
Analiza cazului mediu are, de asemenea, dezavantaje majore: (1) în general, este dificil să se determine o distribuţie de probabilitate corectă pe mulţimea instanţelor $p, p \in P$; (2) această analiză depinde de distribuţia de probabilitate aleasă; (3) în general, determinarea mediei $T^m_\alpha(k)$ se reduce la calculul unor sume finite care sunt evaluate cu mari dificultăţi. Totuşi această analiză este folosită în cazul când algoritmul are performanţe bune pentru majoritatea instanţelor şi pentru un număr mic de instanţe, care apar rar în practică, algoritmul funcţionează prost sau foarte prost.

Analiza *cazului cel mai defavorabil* elimină multe din dezavantajele prezentate mai sus. Această analiză constă în a determina $T_\alpha(k) = sup\{T'_\alpha(p) | p \in P, d(p) = k\}$. Rezultă că analiza cazului cel mai defavorabil: (1) furnizează o margine superioară pentru numărul operaţiilor elementare (instrucţiunilor elementare) necesare algoritmului α pentru rezolvarea oricărei instanţe $p, p \in P$; (2) este independentă de calculatorul pe care se face implementarea algoritmului; (3) face posibilă compararea algoritmilor; (4) are şi avantajul că $T_\alpha(k)$ se determină mai uşor decât $T^m_\alpha(k)$. Totuşi, analiza cazului cel mai defavorabil nu este perfectă. Un dezavantaj major este faptul că $T_\alpha(k)$ poate fi determinată de instanţe "patologice" care apar destul de rar în practică. Dar avantajele acestei analize depăşesc dezavantajele şi de aceea este cea mai folosită metodă pentru a măsura performanţele unui algoritm.

Deoarece determinarea exactă a lui $T_\alpha(k)$ este uneori dificilă, iar pe de altă parte, aceasta ar însemna considerarea unor detalii de implementare, se vor căuta margini superioare şi inferioare pentru $T_\alpha(k)$, se va studia comportarea sa asimptotică.

În cele ce urmează se vor introduce notaţiile O, Ω, Θ şi se vor defini algoritmii polinomiali şi cei exponenţiali.

Fie funcţia $f : \mathcal{N} \to \mathcal{N}$. Se folosesc următoarele notaţii:

$$O(f) = \{g | g : \mathcal{N} \to \mathcal{N},\ \exists\, r_1 \in \Re_+^*,\ \exists k_0 \in \mathcal{N} : g(k) \leq r_1 f(k),\ \forall\, k \geq k_0\};$$

$$\Omega(f) = \{g | g : \mathcal{N} \to \mathcal{N},\ \exists\, r_2 \in \Re_+^*,\ \exists k_0 \in \mathcal{N} : g(k) \geq r_2 f(k),\ \forall\, k \geq k_0\};$$

$$\Theta(f) = \{g \mid g : N \to N,\ g \in O(f) \cap \Omega(f)\}.$$

Se obişnueşte ca în loc de a scrie $g \in O(f)$ să se folosească notaţia $g = O(f)$ (similar pentru Ω şi Θ).

Definiţia 1.32. Se numeşte *complexitatea timp* (sau simplu *complexitate*) a algoritmului α comportarea asimptotică a lui $T_\alpha(k)$.

Definiţia 1.33. Se spune că o problemă algoritmică P are *complexitatea O(f(k))* dacă există un algoritm α care rezolvă problema P şi algoritmul α are complexitatea $T_\alpha(k) = O(f(k))$.

Definiţia 1.34. Se spune că o problemă algoritmică P are *complexitatea* $\Omega(f(k))$ dacă orice algoritm α care rezolvă problema P are complexitatea $T_\alpha(k) = \Omega(f(k))$.

Definiţia 1.35. Se spune că un algoritm α pentru rezolvarea problemei P este *optimal* dacă problema P are complexitatea $\Omega(T_\alpha(k))$.

Demonstrarea optimalităţii unui algoritm α pentru o problemă P este o activitate foarte complicată. Există foarte puţine rezultate de acest fel.

Definiţia 1.36. Se numeşte *algoritm polinomial* un algoritm α cu complexitatea $O(f(k))$ unde $f(k)$ este un polinom în k. Un algoritm care nu este polinomial se numeşte *exponenţial*.

Exemplul 1.28. Presupunem că o operaţie elementară necesită pentru execuţie 10^{-6} secunde, adică $O(1) = 10^{-6}$ secunde. În tabelul de mai jos sunt prezentate complexităţile timp pentru câteva funcţii, unde m înseamnă minute, z înseamnă zile şi s înseamnă secole.

n	n	n^5	2^n	n^n
20	$0,33 \times 10^{-6}$m	$5,33 \times 10^{-2}$m	$1,66 \times 10^{-2}$ m	3×10^{10}s
40	$0,66 \times 10^{-6}$ m	$0,17 \times 10^{1}$ m	$1,27 \times 10$ z	...

Exemplul 1.29. Să considerăm graful $G = (N, A)$ cu $N = \{1, \ldots, n\}, A = \{1, \ldots, m\}$ reprezentat prin matricea sa de adiacenţă M. Unele probleme de teoria grafurilor necesită calcularea matricei M^{n-1}, unde prin M^i notăm, în acest exemplu, puterea booleană de ordinul i a matricei M. Deci problema P constă în calculul matricei M^{n-1}.

Dintr-un exemplu prezentat mai sus rezultă că dimensiunea problemei este n^2.

Dacă avem de efectuat produsul boolean a două matrice boolene (cu elemente 0 şi 1) B şi C pătratice de dimensiune n, atunci pentru calculul matricei produs $D = B \dot{\times} C$ trebuie determinate toate elementele $d_{ij} = b_{i1} \dot{\times} c_{1j} \dot{+} \ldots \dot{+} b_{in} \dot{\times} c_{nj}$, unde $\dot{\times}, \dot{+}$ reprezintă înmulţirea booleană, respectiv adunarea booleană. Convenim să considerăm ca operaţie elementară perechea ordonată $(\dot{\times}, \dot{+})$. Rezultă că, pentru calculul unui element d_{ij} sunt necesare n operaţii elementare. Deci pentru calculul celor n^2 elemente d_{ij} sunt necesare n^3 operaţii elementare. Fie:

$$n - 1 = \sum_{i=0}^{k} a_i\, 2^i, a_k \neq 0, a_i \in \{0, 1\}.$$

În acest caz avem $M^{n-1} = M^{a_0}\dot{\times}(M^2)^{a_1}\dot{\times}\ldots\dot{\times}(M^{2^k})^{a_k}$. Acest produs boolean conţine cel mult $\lceil\log(n-1)\rceil$ înmulţiri boolene, deoarece dacă $a_i = 0$ atunci $(M^{2^i})^{a_i} = U$ (matricea unitate) şi nu se mai efectuează înmulţirea. Considerăm cazul cel mai defavorabil cu $a_i = 1, \quad i = 0,\ldots,k, \quad k = \lceil\log(n-1)\rceil - 1$ ($2^k < n-1 < 2^{k+1}$) şi $k = \lceil\log(n-1)\rceil$ ($n-1 = 2^k$). Algoritmul necesită atunci
1) calculul matricelor booleene:

$$M, M^2 = M\dot{\times}M, \ldots, M^{2^k} = M^{2^{k-1}}\dot{\times}M^{2^{k-1}}$$

ce reprezintă k înmulţiri booleene de matrice booleene;
2) calculul matricelor booleene:

$$M^3 = M\dot{\times}M^2, \quad M^7 = M^3\dot{\times}M^4, \quad M^{15} = M^7\dot{\times}M^8, \ldots$$

ce reprezintă k înmulţiri booleene de matrice booleene.

Deci pentru a calcula M^{n-1} trebuie să efectuăm, în cazul cel mai defavorabil, $2k$ înmulţiri booleene de matrice booleene. Cum pentru înmulţirea booleană a două matrice booleene sunt necesare n^3 operaţii elementare, rezultă că pentru calculul matricei M^{n-1} sunt necesare $2n^3k$ operaţii elementare. Deoarece $k = \lceil\log(n-1)\rceil - 1$ şi dimensiunea problemei este n^2, rezultă că algoritmul pentru calculul matricei M^{n-1} are complexitatea $O(n^{3/2}\log n)$.

Referitor la problemele din teoria grafurilor, în practica curentă, se exprimă complexitatea unui algoritm în funcţie de $m, n, \overline{b}, \overline{c}$. Cum problema este din teoria grafurilor se consideră că algoritmul pentru calculul matricei M^{n-1} are complexitatea $O(n^3\log n)$.

În teoria grafurilor, referitor la complexitatea algoritmilor, avem următoarele noţiuni specifice. Dacă complexitatea algoritmului este $O(f(n, m, \log\overline{b}, \log\overline{c}))$, unde $f(n, m, \log\overline{b}, \log\overline{c})$ este o funcţie polinomială de $n, m, \log\overline{b}, \log\overline{c}$, atunci se spune că algoritmul este *polinomial*. Un algoritm polinomial se spune că este algoritm *tare polinomial* dacă f este o funcţie polinomială numai de n şi m şi algoritm *slab polinomial* altfel. În general, algoritmii tare polinomial sunt preferaţi faţă de algoritmii slab polinomial, deoarece ei pot rezolva probleme cu valori arbitrar de mari pentru funcţiile b şi c. Remarcăm faptul că definiţia algoritmului polinomial din teoria grafurilor se încadrează în definiţia algoritmului polinomial pentru cazul general (Definiţia 1.36). De asemenea, conform definiţiei 1.36, un algoritm din teoria grafurilor este *exponenţial* dacă nu este polinomial. De exemplu, $O(2^n), O(n!), O(m\cdot n\cdot\overline{c})$ sunt complexităţi pentru algoritmi exponenţiali. Se spune că un algoritm este *pseudopolinomial* dacă f este o funcţie polinomială de $m, n, \overline{b}, \overline{c}$. Algoritmii pseudopolinomiali constituie o subclasă importantă a clasei algoritmilor exponenţiali.

Evident, că sunt preferaţi algoritmii polinomiali faţă de algoritmii exponenţiali şi în cadrul algoritmilor exponenţiali sunt preferaţi algoritmii pseudopolinomiali.

În continuare se vor prezenta anumite reguli de calcul pentru O, Ω, Θ. Mai întâi să precizăm că relaţiile O, Ω, Θ pot fi considerate ca relaţii între funcţii: O este relaţia "este dominată asimptotic de", Ω este relaţia "domină asimptotic pe" şi Θ este relaţia "are acelaşi ordin de mărime ca". Se constată uşor că relaţiile O şi Ω sunt relaţii de preordine (reflexive şi tranzitive) şi că relaţia Θ este o relaţie de echivalenţă (reflexivă, simetrică şi tranzitivă). Regulile de calcul pentru O, Ω, Θ sunt următoarele:

1) Reflexivitatea relaţiilor O, Ω, Θ:
   $a) f = O(f), \quad b) f = \Omega(f), \quad c) f = \Theta(f).$
2) Simetria relaţiei Θ:
   $g = \Theta(f)$ implică $f = \Theta(g)$.
3) Tranzitivitatea relaţiilor O, Ω, Θ:
   $a) g = O(f)$ şi $f = O(h)$ implică $g = O(h)$,
   $b) g = \Omega(f)$ şi $f = \Omega(h)$ implică $g = \Omega(h)$,
   $c) g = \Theta(f)$ şi $f = \Theta(h)$ implică $g = \Theta(h)$.
4) Fie $c \in \Re_+^*$:
   $a) g = O(f)$ implică $cg = O(f)$,
   $b) g = \Omega(f)$ implică $cg = \Omega(f)$,
   $c) g = \Theta(f)$ implică $cg = \Theta(f)$.
5) Fie $g_1 \geq g_2$ sau $g_1 \leq g_2$:
   $a) g_1 = O(f_1)$ şi $g_2 = O(f_2)$ implică $g_1 + g_2 = O(\max(f_1, f_2))$,
   $b) g_1 = \Omega(f_1)$ şi $g_2 = \Omega(f_2)$ implică $g_1 + g_2 = \Omega(\min(f_1, f_2))$,
   $c) g_1 = \Theta(f_1)$ şi $g_2 = \Theta(f_2)$ implică $g_1 + g_2 = \Theta(\max(f_1, f_2))$.
6) Fie $g_1 \geq g_2$:
   $a) g_1 = O(f_1)$ şi $g_2 = O(f_2)$ implică $g_1 - g_2 = O(f_1)$,
   $b) g_1 = \Omega(f_1)$ şi $g_2 = \Omega(f_2)$ implică $g_1 - g_2 = \Omega(f_2)$,
   $c) g_1 = \Theta(f_1), g_2 = \Theta(f_2)$ şi $f_1 = \Omega(f_2), f_2 = O(f_1)$ implică $g_1 - g_2 = \Theta(f_1)$.
7) Fie $g_1 \geq g_2$ sau $g_1 \leq g_2$:
   $a) g_1 = O(f_1)$ şi $g_2 = O(f_2)$ implică $g_1 \cdot g_2 = O(f_1 \cdot f_2)$,
   $b) g_1 = \Omega(f_1)$ şi $g_2 = \Omega(f_2)$ implică $g_1 \cdot g_2 = \Omega(f_1 \cdot f_2)$,
   $c) g_1 = \Theta(f_1)$ şi $g_2 = \Theta(f_2)$ implică $g_1 \cdot g_2 = \Theta(f_1 \cdot f_2)$.

Aceste reguli sunt consecinţe evidente ale definiţiilor.

Deseori se pot compara funcţiile f şi g calculând

$$\lim_{n\to\infty} \frac{g(n)}{f(n)}.$$

Se obţin următoarele proprietăţi:

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = c > 0 \quad \text{implică} \quad g = \Theta(f) \tag{1.1}$$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0 \quad \text{implică} \quad g = O(f) \quad \text{şi } g \neq \Theta(f) \tag{1.2}$$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty \quad \text{implică} \quad g = \Omega(f) \quad \text{şi } g \neq \Theta(f) \tag{1.3}$$

Aceste proprietăţi rezultă din definiţia limitei: de exemplu pentru proprietatea (1.1) avem: $\forall \epsilon > 0, \exists n_0 \in \mathcal{N}$ astfel încât $\forall n \geq n_0$ avem $\left|\frac{g(n)}{f(n)} - c\right| < \epsilon$ sau $\forall \epsilon > 0, \ni n_0 \in \mathcal{N}$ astfel încât $\forall n \geq n_0$ avem

$$(c - \epsilon) f(n) < g(n) < (c + \epsilon) f(n).$$

Reciprocile proprietăţilor (1.1), (1.2), (1.3) sunt false. În cazul când nu există limită, trebuie revenit la definiţiile O, Ω, Θ pentru a putea compara g şi f. De exemplu, dacă

$f(n) = 2n, g(n) = n$ pentru $n = 2k + 1$ şi $g(n) = 2n$ pentru $n = 2k$, atunci limita de mai sus nu există, dar $g \in \Theta(f)$ deoarece pentru orice n avem $\frac{1}{2}f(n) \leq g(n) \leq f(n)$.

1.6 Aplicaţii

1.6.1 Reţele de comunicaţii

O reţea de comunicaţie se poate descrie printr-o reţea $G = (N, A, b, c)$. Astfel, pot fi descrise traseele dintre localităţi pe şosele, autostradă sau cale ferată, traseele din localităţi ale autobuzelor, troleibuzelor sau tramvaielor, traseele aeriene sau maritime, reţelele cristalografice, geografice, topografice, cartografice, cadastrale, hidrografice, de irigaţii, de drenaj, de alimentare cu apă, gaze sau energie electrică, de termoficare, telefonice, telecomunicaţii, telegrafice, radio, televiziune, calculatoare. De asemenea repartiţiile de materiale sunt reţele de comunicaţii, fluxuri tehnologice, programele de investiţii, de organizare a muncii, sistemul informaţional. Totodată sunt reţele aparatul circulator al sângelui, sistemul nervos, sistemul limfatic etc.

Exemplul 1.30. În figura 1.15 este reprezentată reţeaua $G = (N, A, b)$ a traseelor pe şosele din judeţul Braşov, unde fiecare nod $x \in N$ reprezintă un oraş din judeţ, o muchie $[x, y] \in A$ reprezintă faptul că oraşele x, y sunt conectate printr-o şosea modernizată şi b reprezintă funcţia distanţă. S-a considerat reţeaua $G = (N, A, b)$ neorientată, deoarece pe o şosea $[x, y]$ se circulă în ambele sensuri. Pe fiecare muchie s-a trecut distanţa în kilometri.

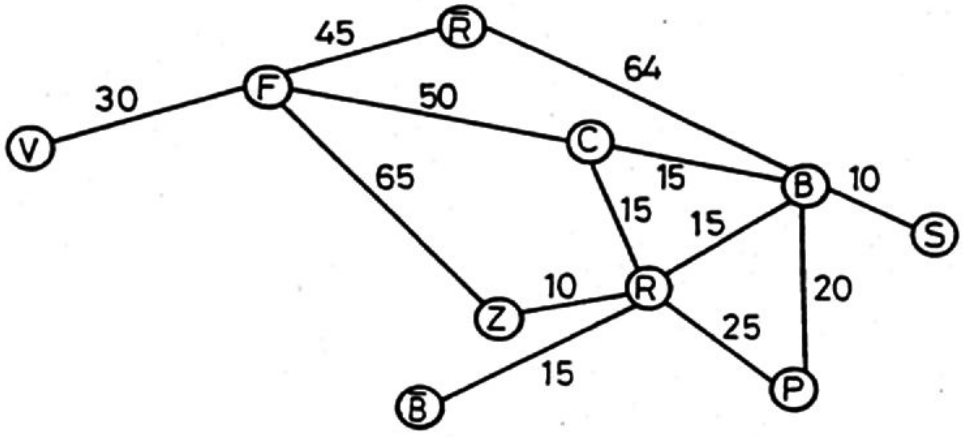


Fig.1.15 B - Braşov; $\overline{B}$ - Bran; C - Codlea; F - Făgăraş; P - Predeal; R - Râşnov; $\overline{R}$ - Rupea; S - Săcele; V - Victoria; Z - Zărneşti

Exemplul 1.31. În digraful din figura 1.16. este schematizată circulaţia sângelui în corpul omenesc.

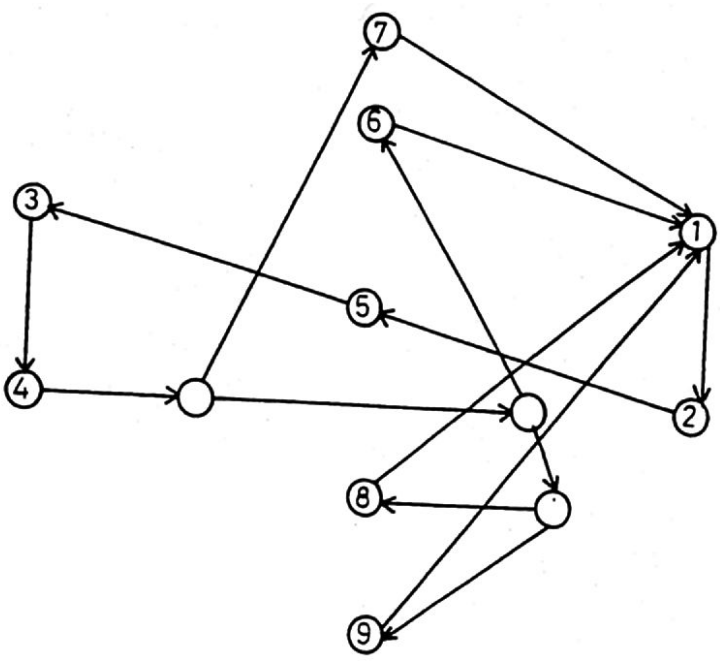


Fig.1.16 1 - auricul drept; 2 - ventricul drept; 3 - auricul stâng;
4 - ventricul stâng; 5 - plămâni; 6 - membre superioare;
7 - cap; 8 - trunchi celiac; 9 - membre inferioare.

Dacă pentru o persoană ar exista arcul $(1,3)$, atunci această persoană ar avea boala lui Roger, iar dacă pentru o persoană se instituie comunicarea directă de la nodul 4 la nodul 2, în digraf ar apărea arcul $(4,2)$, atunci persoana decedează.

Multe alte aplicaţii ale grafurilor vor fi prezentate în capitolele următoare.

1.6.2 Probleme de programare dinamică

Metoda programării dinamice se aplică problemelor de optimizare în care soluţia poate fi privită ca rezultatul unui şir de decizii din mulţimea deciziilor $D = \cup_{i=1}^{n} D_{0i}$.

Fie sistemul S cu mulţimea stărilor X. Presupunem că trecerea sistemului S dintr-o stare în alta are loc numai în urma unei decizii. Un vector $\overline{x} = (x_0, \ldots, x_n), x_i \in X$, $i = 0, \ldots, n$ reprezintă o soluţie şi un vector $\overline{d} = (d_1, \ldots, d_n)$, $d_i \in D$, $i = 1, \ldots, n$ reprezintă o politică a sistemului S. Un vector $\overline{x}_{ij} = (x_i, \ldots, _j)$ este o soluţie parţială şi un vector $\overline{d}_{ij} = (d_{i+1}, \ldots, d_j)$ $0 \leq i < j \leq n$, este o subpolitică a sistemului S. Printre politicile $\overline{d}$ care determină evoluţia sistemului S dintr-o stare iniţială x_0 într-o stare finală x_n poate exista o politică $\overline{d}^* = (d_1^*, \ldots, d_n^*)$ care optimizează funcţia obiectiv dată. Această politică se numeşte *politică optimă* şi soluţia $\overline{x}^* = (x_0^*, \ldots, x_n^*)$ determinată de politica optimă $\overline{d}^*$ se numeşte *soluţie optimă*.

Principiul optimalităţii are următoarea formulare:

Dacă $\overline{x}^* = (x_0^*, \ldots, x_n^*)$ este o soluţie optimă atunci oricare soluţie parţială $\overline{x}_{ij} = \left(x_i^*, \ldots, x_j^*\right)$ a sa este optimă.

Deci programarea dinamică se poate aplica problemelor pentru care optimul total implică optimul parţial.

Analiza de luarea deciziilor începând cu starea iniţială x_0, continuând cu stările $x_1, x_2, \ldots$ şi terminând cu starea finală x_n se numeşte *analiza prospectivă*. Analiza de luarea deciziilor începând cu starea finală x_n, continuând cu stările $x_{n-1}, x_{n-2}, \ldots$ şi terminând cu starea iniţială x_0 se numeşte *analiza retrospectivă*.

În analiza prospectivă, dacă sistemul S se află în starea x_{i-1} şi se ia decizia $d_i \in D_{0i}(x_{i-1})$ atunci S trece în starea x_i, $i = 1, \ldots, n$. Această evoluţie poate fi descrisă prin relaţiile:

$$x_i = t_{0i}(x_{i-1}, d_i),\ d_i \in D_{0i}(x_{i-1}),\ i = 1, \ldots, n.$$

Fie $u_{0i}(x_i, d_i)$ utilitatea trecerii sistemului S de la starea x_{i-1}, la starea x_i, $i = 1, \ldots, n$. utilitatea totală este funcţia $f_{0n}(u_{01}, \ldots, u_{0n})$ şi reprezintă funţia obiectiv. Considerăm cazul aditiv $f_{0n}(u_{01}, \ldots, u_{0n}) = u_{01} + \ldots + u_{0n}$.

Fie X_{0i}, $0 \leq i \leq n$, mulţimea stărilor accesibile la momentul i. prin urmare, avem:

$$X_{0i} = \{y | y \in X,\ \text{există } x \in X_{0i-1} \text{ şi } d \in D_{0i}(x) \text{ astfel încât } y = t_{0i}(x, d)\}.$$

Oricărui proces secvenţial de decizii i se poate asocia un digraf $G = (N, \Gamma)$ unde $N = \cup_{i=0}^{n} X_{0i}$, $\Gamma(x) = \{y | y \in X_{0i},\ y = t_{0i}(x, d),\ d \in D_{0i}(x)\}$ pentru orice $x \in X_{0i-1}$. Digraful poate fi exprimat echivalent prin $G = (N, A)$, unde $(x, y) \in A$ dacă şi numai dacă $y \in \Gamma(x)$. Un drum în digraful G de la starea $x_{i-1} \in X_{0i-1}$ la starea $x_j \in X_{oj}$, $1 \leq i < j \leq n$, determină în mod unic o subpolitică $\overline{d}_{ij} = (d_i, \ldots, d_j)$ şi reciproc. În particular, un drum de la starea iniţială $x_0 \in X_{00}$ la starea finală $x_n \in X_{0n}$ determină în mod unic o politică $\overline{d} = (d_1, \ldots, d_n)$ şi reciproc. Digraful $G = (N, A)$ are o structură particulară deoarece mulţimea nodurilor N este partiţionată în $n+1$ mulţimi $X_{00}, \ldots, X_{0n}$ şi pentru orice arc $(x, y) \in A$, dacă $x \in X_{0i-1}$ atunci $y \in X_{0i}$. Un digraf $G = (N, A)$ cu aceste proprietăţi se numeşte *digraf secvenţial*. Într-un digraf secvenţial $G = (N, A)$ nu există circuite. Definim funcţia $v : A \to \Re$ prin $v(x, y) = u_{0i}(x, d)$, dacă $x \in X_{0i-1}$, $d \in D_{0i}(x)$ pentru orice arc $(x, y) \in A$. Astfel, în cazul funcţiei obiectiv aditive, problema determinării unei politici optime este echivalentă cu problema determinării unui drum de valoare optimă (minimă sau maximă) în digraful $G = (N, A)$.

Exemplul 1.32. Se consideră sistemul S cu mulţimea stărilor $X = \{x_0, x_1, x_2, x_3\}$ şi mulţimea stărilor iniţiale $X_{00} = \{x_0\}$. Mulţimile deciziilor $D_{0i}(x)$ şi transformările $t_{0i}(x, d)$ sunt următoarele:

$D_{01}(x_0) = \{d_1, d_2\}$, $t_{01}(x_0, d_1) = x_1$, $t_{01}(x_0, d_2) = x_2$;
$D_{02}(x_1) = \{d_3, d_4\}$, $t_{02}(x_1, d_3) = x_1$, $t_{02}(x_1, d_4) = x_2$;
$D_{02}(x_2) = \{d_5, d_6\}$, $t_{02}(x_2, d_5) = x_1$, $t_{02}(x_2, d_6) = x_0$;
$D_{03}(x_0) = \{d_1, d_7\}$, $t_{03}(x_0, d_1) = x_1$, $t_{03}(x_0, d_7) = x_0$,
$D_{03}(x_1) = \{d_3, d_4, d_8\}$, $t_{03}(x_1, d_3) = x_1$, $t_{03}(x_1, d_4) = x_2$, $t_{03}(x_1, d_8) = x_0$,
$D_{03}(x_2) = \{d_5\}$, $t_{03}(x_2, d_5) = x_1$.

Rezultă că mulţimile X_{0i} ale stărilor accesibile sunt următoarele: $X_{00} = \{x_0\}$, $X_{01} = \{x_1, x_2\}$, $X_{02} = \{x_0, x_1, x_2\}$, $X_{03} = \{x_0, x_1, x_2\}$. Digraful secvenţial $G = (N, \Gamma)$ este

definit în modul următor: $N = X_{00} \cup X_{01} \cup X_{02} \cup X_{03} = \{x_0, x_1, x_2\}$. Aplicaţia Γ este definită astfel:

Dacă $x \in X_{00}$ atunci $\Gamma(x) = \{x_1, x_2\}$ pentru $x = x_0$

Dacă $x \in X_{01}$ atunci

$$\Gamma(x) = \begin{cases} \{x_1, x_2\} & \text{pentru} \quad x = x_1 \\ \{x_0, x_1\} & \text{pentru} \quad x = x_2 \end{cases}$$

Dacă $x \in X_{02}$ atunci

$$\Gamma(x) = \begin{cases} \{x_0, x_1\} & \text{pentru} \quad x = x_0 \\ \{x_0, x_1, x_2\} & \text{pentru} \quad x = x_1 \\ \{x_1\} & \text{pentru} \quad x = x_2 \end{cases}$$

Reprezentarea grafică a digrafului $G = (N, \Gamma)$ este prezentată în figura 1.17.

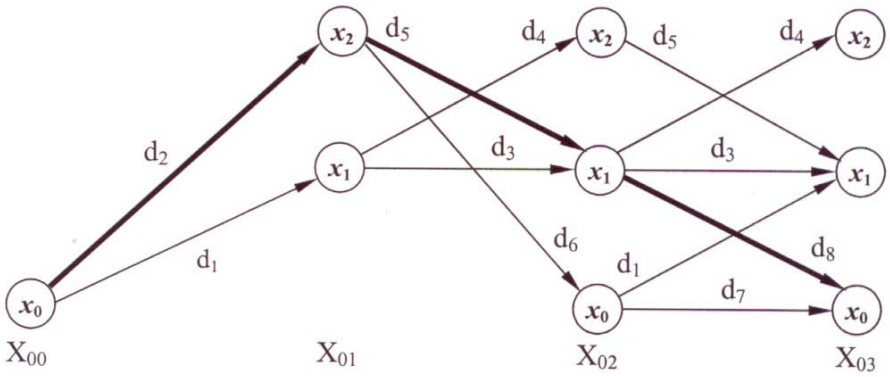


Fig.1.17

Drumul care corespunde stării iniţiale x_0 şi politicii (d_2, d_5, d_8) este trasat cu linie groasă.

Digraful secvenţial $G = (N, \Gamma)$ asociat unui proces secvenţial de decizii definit ca mai sus este un digraf dinamic care pune în evidenţă natura secvenţială în timp a procesului. Se poate asocia procesului de decizii un alt digraf care este un digraf static şi nu mai pune în evidenţă natura secvenţială a procesului. În acest caz digraful static $G' = (N', \Gamma')$ se defineşte în modul următor.

$$N' = X,\ \Gamma'(x) = \{y | y \in X,\ y = t_0(x, d),\ d \in D_0(x)\},\ x \in N'$$

$$t_0(x, d) = t_{0i}(x_{i-1}, d_i) = \ldots = t_{0j}(x_{j_1}, d_j) \text{ dacă}$$
$$d = d_i = \ldots = d_j,\ D_0(x) = \cup_{i=1}^{n} D_{0i}(x),\ x \in N'.$$

Exemplul 1.33. Se consideră sistemul S cu mulţimea stărilor X, mulţimea stărilor iniţiale X_{00}, mulţimea deciziilor $D_{0i}(x)$ şi transformările $t_{0i}(x, d)$ prezentate în exemplul 1.32. Reprezentarea grafică a digrafului $G' = (N', \Gamma')$ este prezentată în figura 1.18.

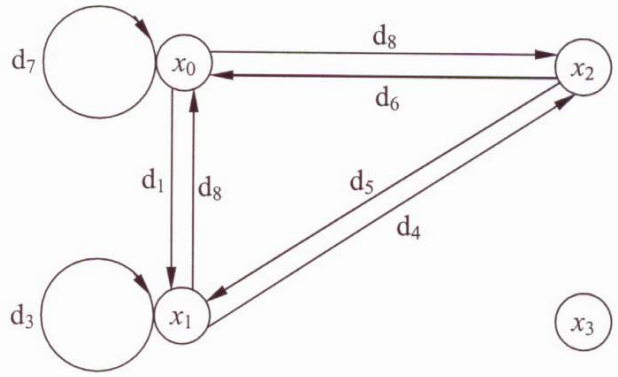


Fig. 1.18

Capitolul 2

Parcurgeri de grafuri

Problema parcurgerii unui digraf $G = (N, A), \quad N = \{1, \ldots, n\}, \quad A = \{1, \ldots, m\}$ are următoarea formulare: să se genereze mulţimea $W \subset N$ a nodurilor y pentru care există drum de la un *nod sursă* dat s la nodul y în digraful G. Dacă există drum, în digraful G, de la nodul sursă s la nodul y atunci se spune că nodul y este *accesibil* din nodul s.

Algoritmii pe care îi vom prezenta pentru rezolvarea problemei parcurgerii unui digraf G sunt metode sistematice de *vizitare* a nodurilor y accesibile din s. Fiecare iteraţie a execuţiei oricărui algoritm de parcurgere stabileşte pentru fiecare nod apartenenţa la una din următoarele trei *stări*:

- *nevizitat*;
- *vizitat şi neanalizat*, adică un nod vizitat ai cărui succesori au fost parţial vizitaţi;
- *vizitat şi analizat*, adică un nod vizitat ai cărui succesori au fost în totalitate vizitaţi.

Dacă nodul x este vizitat şi neanalizat, există arc (x, y) şi nodul y este nevizitat, atunci se poate *vizita* nodul y. În acest caz se spune că arcul (x, y) este arc *admisibil* şi dacă nodul y este vizitat explorând arcul (x, y) se spune că nodul x este *predecesorul parcurgere* al nodului y.

În acest capitol se vor prezenta următorii algoritmi pentru parcurgerea unui digraf: algoritmul generic, algoritmul BF şi algoritmul DF. Aceşti algoritmi utilizează următoarele notaţii comune:

U mulţimea nodurilor nevizitate;
V mulţimea nodurilor vizitate şi neanalizate;
W mulţimea nodurilor vizitate şi analizate;
p tabloul predecesor, care este unidimensional cu n elemente.

2.1 Parcurgerea generică a grafurilor

Se folosesc notaţiile precizate mai sus şi în plus notăm cu *o tabloul ordine* care este unidimensional şi are n elemente.

Algoritmul parcurgerii generice (algoritmul PG) este următorul:

(1) PROGRAM PG;
(2) BEGIN
(3) $U := N - \{s\}$; $V := \{s\}$; $W := \emptyset$;
(4) FOR toţi $y \in N$ DO $p(y) := 0$;
(5) $k := 1$; $o(s) := 1$;
(6) FOR toţi $y \in U$ DO $o(y) := \infty$;
(7) WHILE $V \neq \emptyset$ DO
(8) BEGIN
(9) se selectează un nod x din V;
(10) IF există $\text{arc}(x, y) \in A$ şi $y \in U$
(11) THEN $U := U - \{y\}$; $V := V \cup \{y\}$;
$p(y) := x$; $k := k + 1$; $o(y) := k$
(12) ELSE $V := V - \{x\}$; $W := W \cup \{x\}$;
(13) END;
(14) END.

Teorema 2.1 *Algoritmul PG este convergent şi la terminarea execuţiei determină mulţimea tuturor nodurilor care sunt accesibile din nodul sursă s în digraful $G = (N, A)$.*
Demonstraţie Din liniile (10), (11) şi (12) ale algoritmului rezultă că toate nodurile introduse în V sunt eliminate după ce sunt analizate. Deci după un număr finit de iteraţii se obţine $V = \emptyset$ şi execuţia algoritmului se opreşte. Pentru a arăta că la terminarea execuţiei, algoritmul determină mulţimea tuturor nodurilor care sunt accesibile din nodul sursă s în digraful $G = (N, A)$, trebuie să arătăm că la terminarea execuţiei algoritmului mulţimea W este:

$$W = \{y | y \in N \quad \text{şi există drum de la } s \text{ la } y\}.$$

Din liniile (10), (11) şi (12) ale algoritmului rezultă că în V sunt introduse numai noduri y care sunt accesibile din s şi că după ce un nod $x \in V$ a fost analizat el este eliminat din V şi introdus în W. Deoarece algoritmul se opreşte când $V = \emptyset$ rezultă că W conţine toate nodurile $y \in N$ care sunt accesibile din s şi introduse în V. Să arătăm că W conţine toate nodurile $y \in N$ care sunt accesibile din s. Prin reducere la absurd să presupunem că există un drum $D = (y_1, y_2, \ldots, y_{k-1}, y_k)$ cu $y_1 = s$, $y_k = y$ în G şi $y \notin W$. Rezultă că $y_k \notin V$. Deoarece $y_k \notin V$ şi $(y_{k-1}, y_k) \in A$ deducem că $y_{k-1} \notin V$, astfel y_k ar fi fost introdus în V. Continuând procedeul vom deduce în final că $s = y_1 \notin V$. Aceasta contrazice faptul că în linia (3) a algoritmului iniţializăm $V{:=}\{s\}$. Rezultă că $y \in V$, deci $y \in W$ şi teorema este demonstrată. ■
Teorema 2.2 *Algoritmul PG are complexitatea $O(m)$.*
Demonstraţie Din liniile (10), (11) şi (12) ale algoritmului rezultă că fiecare nod al digrafului G este introdus şi eliminat din V cel mult o dată. Deoarece execuţia algoritmului se termină când $V = \emptyset$ deducem că algoritmul execută cel mult $2n$ iteraţii. Fiecare arc $(x, y) \in A$ este explorat cel mult o dată pentru identificarea arcelor admisibile. Deci complexitatea algoritmului PG este $O(m + n) = O(m)$.■

Un digraf $\widehat{G} = \left(\widehat{N}, \widehat{A}\right)$ se numeşte *arborescenţă* cu rădăcina s dacă este fără cicluri şi $\widehat{\rho}^-(s) = 0$, $\widehat{\rho}^-(x) = 1$ pentru toate nodurile x din $\widehat{N}$ cu $x \neq s$.

Subgraful $G_p = (N_p, G_p)$ cu $N_p = \{y \in N \mid p(y) \neq 0\} \cup \{s\}$, $A_p = \{(p(y), y) \in A \mid y \in N_p - \{s\}\}$ se numeşte *subgraf predecesor* al digrafului $G = (N, A)$. Dacă $N_p = W$ şi G_p este o arborescenţă atunci G_p se numeşte *arborescenţă parcurgere*. Arcele din A_p se numesc *arce arborescenţă*.

Teorema 2.3 *Algoritmul PG determină elementele tabloului p astfel încât subgraful predecesor $G_p = (N_p, A_p)$ este o arborescenţă parcurgere.*

Demonstraţie Din liniile (10), (11) şi (12) ale algoritmului rezultă că $p(y) := x$ numai dacă y este accesibil din s. Evident că la terminarea execuţiei algoritmului avem $N_p = W$. Din modul cum sunt definite N_p şi A_p rezultă că G_p este o arborescenţă. Deci subgraful predecesor G_p este o arborescenţă parcurgere a digrafului $G = (N, A)$.■

Observaţia 2.1. Mulţimea W este în general o submulţime a mulţimii nodurilor N. Pentru parcurgerea întregului digraf $G = (N, A)$ se utilizează algoritmul parcurgerii totale generice (algoritmul PTG). Algoritmul PTG este următorul:

```
(1)   PROGRAM PTG;
(2)    BEGIN;
(3)      U := N − {s}; V := {s}; W := ∅;
(4)      FOR toţi y ∈ N DO p(y) := 0;
(5)      k := 1; o(s) := 1;
(6)      FOR toţi y ∈ U DO o(y) := ∞;
(7)      WHILE W ≠ N DO
(8)      BEGIN
(9)         WHILE V ≠ ∅ DO
(10)         BEGIN
(11)            se selectează un nod x din V;
(12)            IF există arc (x, y) ∈ A şi y ∈ U
(13)               THEN U := U − {y}; V := V ∪ {y}; p(y) := x;
                          k := k + 1; o(y) := k
(14)                ELSE V := V − {x}; W := W ∪ {x};
(15)         END;
(16)            se selectează s ∈ U; U := U − {s}; V := {s};
                          k := k + 1; o(s) := k;
(17)      END;
(18)   END.
```

Este evident că algoritmul PTG are complexitatea tot $O(m)$ şi că vectorul p determină una sau mai multe arborescenţe parcurgere.

Observaţia 2.2. Din modul cum se calculează $o(y) := k$, în linia (11) a algoritmului PG sau în liniile (13) sau (16) a algoritmului PTG, rezultă că y este al k-lea nod vizitat în parcurgerea digrafului $G = (N, A)$.

Observaţia 2.3. Algoritmul PG sau PTG poate fi aplicat şi grafurilor neorientate. În acest caz subgraful predecesor $G_p = (N_p, A_p)$ este tot o arborescenţă sau mai multe arborescenţe.

Observaţia 2.4. Drumul unic de la nodul sursă s la un nod y din arborescenţa parcurgere $G_p = (N_p, A_p)$ este furnizat de următoarea procedură:

```
(1)   PROCEDURA DRUM (G, s, y);
(2)   BEGIN;
(3)      se tipăreşte y;
(4)      WHILE p(y) ≠ 0 DO
(5)      BEGIN
(6)         x := p(y);
(7)         se tipăreşte x;
(8)         y := x
(9)      END;
(10)  END.
```

După ordinea de selectare şi adăugare a nodurilor la mulţimea V se cunosc mai multe moduri de parcurgere a unui digraf $G = (N, A)$. Cele mai uzuale moduri de parcurgere sunt:

- parcurgerea se face "mai întâi în lăţime", în engleză breadth first (BF);
- parcurgerea se face "mai întâi în adâncime", în engleză depth first (DF).

2.2 Parcurgerea BF a grafurilor

Fie digraful $G = (N, A)$ cu nodul sursă s şi $\mathcal{D}_x$ mulţimea drumurilor de la nodul sursă s la nodul $x \in N$. Numărul de arce ce compun un drum $D_x \in \mathcal{D}_x$ defineşte *lungimea* acestui drum pe care o notăm $l(D_x)$. *Distanţa* de la nodul s la nodul x se defineşte în modul următor:

$$d(x) = \begin{cases} min\{l(D_x) \mid D_x \in \mathcal{D}_x\} & \text{dacă} \quad \mathcal{D}_x \neq \emptyset \\ \infty & \text{dacă} \quad \mathcal{D}_x = \emptyset \end{cases}$$

Un drum $\widehat{D}_x \in \mathcal{D}_x$ cu $l\left(\widehat{D}_x\right) = d(x)$ se numeşte *cel mai scurt drum* de la nodul sursă s la nodul x.

Observaţia 2.5. Pentru orice arc $(x, y) \in A$ avem $d(y) \leq d(x) + 1$.

Într-adevăr dacă $\mathcal{D}_x = \emptyset$ atunci $d(x) = \infty$ şi inegalitatea se păstrează. Dacă $\mathcal{D}_x \neq \emptyset$ atunci evident $\mathcal{D}_y \neq \emptyset$. Fie $\widehat{D}_x$ un cel mai scurt drum de la s la x, deci $l\left(\widehat{D}_x\right) = d(x)$. Mulţimea $\mathcal{D}_y$ poate conţine un drum D_y astfel încât $l(D_y) < d(x) + 1$. Conform definiţiei distanţei avem $d(y) \leq l(D_y)$ şi rezultă că $d(y) < d(x) + 1$. Avem egalitate când un drum cel mai scurt $\widehat{D}_y$ de la s la y are lungimea $d(y) = l\left(\widehat{D}_y\right) = d(x) + 1$, de exemplu $\widehat{D}_y = \widehat{D}_x \cup \{(x, y)\}$.

În algoritmul parcurgerii BF (algoritmul PBF) se folosesc aceleaşi notaţii ca în algoritmul PG cu deosebirea că în locul tabloului ordine o se utilizează tabloul lungime l care este unidimensional şi are n elemente. Mulţimea nodurilor vizitate şi neanalizate V este organizată, ca structură de date, ca o coadă.

Algoritmul PBF este următorul:

```
(1)   PROGRAM PBF;
(2)   BEGIN;
(3)      U := N − {s} ; V := {s} ; W := ∅;
(4)      FOR toţi y ∈ N DO p (y) := 0;
(5)      l (s) := 0;
(6)      FOR toţi y ∈ U DO l (y) := ∞;
(7)      WHILE V ≠ ∅ DO
(8)      BEGIN
(9)         se selectează cel mai vechi nod x introdus în V;
(10)        FOR (x, y) ∈ A DO
(11)           IF y ∈ U
(12)              THEN U := U − {y} ; V := V ∪ {y} ; p (y) := x;
                        l (y) := l (x) + 1;
(13)        V := V − {x} ; W := W ∪ {x} ;
(14)     END;
(15)  END.
```

Teorema 2.4 *(1) Algoritmul PBF calculează elementele tabloului l astfel încât* $d(y) \leq l(y)$, $y \in N$;
(2) Dacă la iteraţia k oarecare a algoritmului PBF avem $V = \{x_1, \ldots, x_r\}$ *în această ordine, atunci* $l(x_r) \leq l(x_1) + 1$ *şi* $l(x_i) \leq l(x_{i+1})$, $i = 1, \ldots, r-1$.

Demonstraţie *(1)* Utilizăm inducţia după k numărul de iteraţii ale ciclului WHILE. Iniţial $l(s) := 0$, $l(y) := \infty$ pentru $y \in U$ şi evident $d(y) \leq l(y)$, $y \in N$. Presupunem că la iteraţia k avem $d(y) \leq l(y)$pentru $y \in N$. La iteraţia $k+1$ pot exista cazurile:
(c1) Există arc (x, y) admisibil ($(x, y) \in A$ şi $y \in U$). În acest caz $l(y) = l(x) + 1$ şi $d(y) \leq d(x) + 1 \leq l(x) + 1 = l(y)$ ($l(x)$ pentru $x \in V$ nu se modifică). Deci pentru toate arcele $(x, y) \in A$ şi $y \in U$ avem $d(y) \leq l(y)$. Pentru celelalte noduri y, conform ipotezei inducţiei, avem $d(y) \leq l(y)$, deoarece la iteraţia $k+1$ $l(y)$ nu se mai modifică.
(c2) Nu există arc (x, y) admisibil ($(x, y) \notin A$ sau $y \notin U$). În acest caz la iteraţia $k+1$ nu se modifică nici un element $l(y)$ şi conform ipotezei inducţiei $d(y) \leq l(y)$ pentru $y \in N$.
(2) Utilizăm inducţia după k numărul de iteraţii ale ciclului WHILE. Iniţial $V := \{s\}$. Deci $x_1 = s$, $x_r = s$ şi $l(s) < l(s) + 1$, $l(s) = l(s)$. Presupunem că la iteraţia k avem $l(x_r) \leq l(x_1) + 1$ şi $l(x_i) < l(x_{i+1})$, pentru $i = 1, \ldots, r-1$. La iteraţia $k+1$ pot exista cazurile:
(c1) Există arc (x, y) admisibil ($(x, y) \in A$ şi $y \in U$). În acest caz $V = \{x_1, \ldots, x_r, x_{r+1}\}, x_1 = x, x_{r+1} = y$. Astfel $l(x_{r+1}) = l(y) = l(x) + 1 = l(x_1) + 1$. De asemenea, avem $l(x_r) \leq l(x_1) + 1 = l(x) + 1 = l(y) = l(x_{r+1})$ şi inegalităţile $l(x_i) \leq l(x_{i+1})$, $i = 1, \ldots, r-1$ au rămas nemodificate.
(c2) Nu există arc (x, y) admisibil ($(x, y) \notin A$ sau $y \notin U$). În acest caz $V = \{x_2, \ldots, x_r\}$. Avem $l(x_r) \leq l(x_1) + 1 \leq l(x_2) + 1$ şi inegalităţile $l(x_i) \leq l(x_{i+1})$, $i = 1, \ldots, r-1$ au rămas nemodificate.■

Teorema 2.5. *Algoritmul PBF este convergent şi determină:*
(1) mulţimea tuturor nodurilor care sunt accesibile din nodul sursă s;
(2) elementele tabloului l astfel încât $l(y) = d(y)$ *pentru* $y \in N$.

Demonstraţie Convergenţa şi punctul *(1)* se demonstrează la fel ca Teorema 2.1. Punctul *(2)* se demonstrează prin inducţie după k numărul iteraţiilor ciclului WHILE. Fie mulţimea $N_k = \{y \in N \mid d(y) = k\}$. Pentru $k = 0$ avem $N_0 = \{s\}$ şi deci $d(s) = l(s) = 0$. Presupunem afirmaţia adevărată pentru k. Afirmaţia pentru $k+1$ rezultă cu uşurinţă, deoarece, în conformitate cu Teorema 2.4 punctul *(2)*, un nod $y \in N_{k+1}$ este vizitat plecând de la un nod $x \in N_k$ numai după ce toate nodurile din N_k sunt vizitate. Deci, dacă $y \in N_{k+1}$ şi este vizitat explorând arcul (x, y), $x \in N_k$, atunci, $l(y) = l(x) + 1 = d(x) + 1 = k + 1 = d(y)$. ■

Teorema 2.6. *Algoritmul PBF are complexitatea* $O(m)$.

Demonstraţie Evident că algoritmul PBF are aceeaşi complexitate ca a algoritmului PG, adică $O(m)$. ■

În parcurgerea BF, dacă $N_p = W$ şi subgraful predecesor $G_p = (N_p, A_p)$ este o arborescenţă atunci G_p se numeşte *arborescenţă parcurgere BF.*

Teorema 2.7. *Algoritmul PBF determină elementele tabloului p astfel încât subgraful predecesor* $G_p = (N_p, A_p)$*este o arborescenţă parcurgere BF.*

Demonstraţie Se demonstrează analog cu Teorema 2.3.■

Observaţia 2.6. Algoritmul parcurgerii totale BF (algoritmul PTBF) se obţine din algoritmul PBF analog cum s-a obţinut algoritmul PTG din algoritmul PG.

Observaţia 2.7. Drumul unic de la nodul sursă s la un nod y din arborescenţa parcurgere BF este un cel mai scurt drum de la nodul sursă s la acelaşi nod y din digraful G, conform punctului *(2)* al Teoremei 2.5.

Observaţia 2.8. Algoritmul PBF sau PTBF se poate aplica şi grafurilor neorientate. În acest caz subgraful predecesor $G_p = (N_p, A_p)$ este o arborescenţă sau mai multe arborescenţe.

Observaţia 2.9. Drumul unic de la nodul sursă s la un nod y din arborescenţa parcurgere BF se poate determina cu PROCEDURA DRUM prezentată în Observaţia 2.4.

Exemplu 2.1. Se aplică algoritmul PBF digrafului din figura 2.1.

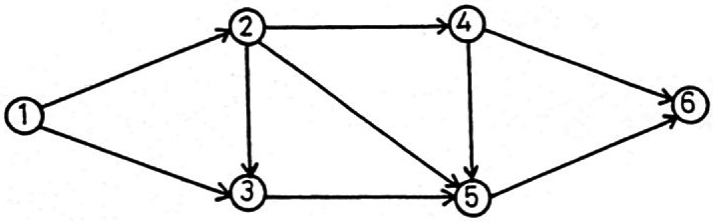


Fig.2.1

Iniţializări: $s = 1$, $U = \{2, 3, 4, 5, 6\}$, $V = \{1\}$, $W = \emptyset$, $p = (0, 0, 0, 0, 0, 0)$, $l = (0, \infty, \infty, \infty, \infty, \infty)$.
Iteraţia 1: $x = 1$, $(1, 2) \in A$, $2 \in U : U = \{3, 4, 5, 6\}$, $V = \{1, 2\}$, $p = (0, 1, 0, 0, 0, 0)$, $l = (0, 1, \infty, \infty, \infty, \infty)$; $(1, 3) \in A$, $3 \in U : U = \{4, 5, 6\}$, $V = \{1, 2, 3\}$, $p = (0, 1, 1, 0, 0, 0)$, $l = (0, 1, 1, \infty, \infty, \infty)$; $V = \{2, 3\}$; $W = \{1\}$.

Iteraţia 2: $x = 2$, $(2, 4) \in A$, $4 \in U : U = \{5, 6\}$, $V = \{2, 3, 4\}$,
$p = (0, 1, 1, 2, 0, 0)$, $l = (0, 1, 1, 2, \infty, \infty)$; $(2, 5) \in A$,
$5 \in U : U = \{6\}$, $V = \{2, 3, 4, 5\}$, $p = (0, 1, 1, 2, 2, 0)$,
$l = (0, 1, 1, 2, 2, \infty)$; $V = \{3, 4, 5\}$; $W = \{1, 2\}$.
Iteraţia 3: $x = 3$, $V = \{4, 5\}$, $W = \{1, 2, 3\}$.
Iteraţia 4: $x = 4$, $(4, 6) \in A$, $6 \in U : U = \emptyset$, $V = \{4, 5, 6\}$,
$p = (0, 1, 1, 2, 2, 4)$, $l = (0, 1, 1, 2, 2, 3)$; $V = \{5, 6\}$; $W = \{1, 2, 3, 4\}$.
Iteraţia 5: $x = 5$, $V = \{6\}$, $W = \{1, 2, 3, 4, 5\}$.
Iteraţia 6: $x = 6$, $V = \emptyset$, $W = \{1, 2, 3, 4, 5, 6\}$.
$N_p = \{1, 2, 3, 4, 5, 6\} = W$.

Arborescenţa parcurgere BF este prezentată în figura 2.2.

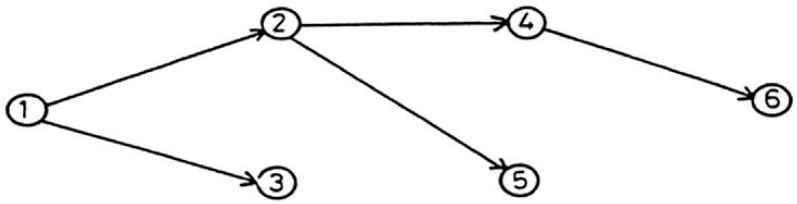


Fig.2.2

Drumul unic de la nodul 1 la nodul 6 se obţine cu PROCEDURA DRUM în modul următor:
$y = 6$ este ultimul nod al drumului;
Iteraţia 1: $x = p(6) = 4$, $y = 4$.
Iteraţia 2: $x = p(4) = 2$, $y = 2$.
Iteraţia 3: $x = p(2) = 1$, $y = 1$.
Drumul este: $1, 2, 4, 6$.

2.3 Parcurgerea DF a grafurilor

În algoritmul parcurgerii DF (algoritmul PDF) se folosesc aceleaşi notaţii ca în algoritmul PG cu deosebirea că în locul tabloului unidimensional *ordine o* se utilizează tablourile *timp* unidimensionale t_1 şi t_2 care au fiecare n elemente. Mulţimea nodurilor vizitate şi neanalizate V este organizată, ca structură de date, ca stivă.

Algoritmul PDF este următorul:

(1) PROGRAM PDF;
(2) BEGIN;
(3) $U := N - \{s\}$; $V := \{s\}$; $W := \emptyset$;
(4) FOR toţi $y \in N$ DO $p(y) := 0$;
(5) $t := 1$; $t_1(s) := 1$; $t_2(s) := \infty$;
(6) FOR toţi $y \in U$ DO $t_1(y) := \infty$; $t_2(y) := \infty$;

(7) WHILE $V \neq \emptyset$ DO
(8) BEGIN
(9) se selectează cel mai nou nod x introdus în V;
(10) IF există arc $(x, y) \in A$ şi $y \in U$
(11) THEN $U := U - \{y\}; V := V \cup \{y\}; p(y) := x;$
$t := t + 1;\ t_1(y) := t$
(12) ELSE $V := V - \{x\}; W := W \cup \{x\}; t := t + 1;\ t_2(x) := t;$
(13) END;
(14) END.

Din liniile (10), (11), (12) ale algoritmului PDF rezultă că elementul $t_1(y)$ reprezintă momentul când y devine nod vizitat şi neanalizat şi elementul $t_2(x)$ reprezintă momentul când x devine nod vizitat şi analizat.

Deoarece algoritmul parcurgerii totale DF (algoritmul PTDF) are multiple aplicaţii, se va studia în continuare acest algoritm.

Algoritmul PTDF este următorul:

(1) PROGRAM PTDF;
(2) BEGIN;
(3) $U := N - \{s\}; V := \{s\}; W := \emptyset;$
(4) FOR toţi $y \in N$ DO $p(y) := 0;$
(5) $t := 1;\ t_1(s) := 1; t_2(s) := \infty;$
(6) FOR toţi $y \in U$ DO $t_1(y) := \infty; t_2(y) := \infty;$
(7) WHILE $W \neq N$ DO
(8) BEGIN
(9) WHILE $V \neq \emptyset$ DO
(10) BEGIN
(11) se selectează cel mai nou nod x introdus în V;
(12) IF există arc $(x, y) \in A$ şi $y \in U$
(13) THEN $U := U - \{y\}; V := V \cup \{y\}; p(y) := x;$
$t := t + 1;\ t_1(y) := t$
(14) ELSE $V := V - \{x\}; W := W \cup \{x\};$
$t := t + 1;\ t_2(x) := t;$
(15) END;
(16) se selectează $s \in U; U := U - \{s\}; V := \{s\};$
$t := t + 1;\ t_1(s) := t;$
(17) END;
(18) END.

Fie mulţimea $S = \{s \mid s \in N,\ s$ selectate în linia (3) şi linia (16)$\}$.

Teorema 2.8. *Algoritmul PTDF este convergent şi determină mulţimile nodurilor accesibile din s, $s \in S$.*

Demonstraţie. Se demonstrează la fel ca Teorema 2.1.■

Teorema 2.9 *Algoritmul PTDF are complexitatea $O(m)$.*

Demonstraţie. Evident că algoritmul PTDF are aceeaşi complexitate ca a algoritmului PTG, adică $O(m)$. ■

Un digraf $\widehat{G} = \left(\widehat{N}, \widehat{A}\right)$se numeşte *pădure* dacă este format din una sau mai multe arborescenţe. Un graf neorientat $\widehat{G} = \left(\widehat{N}, \widehat{A}\right)$se numeşte pădure dacă este format din unul sau mai mulţi arbori.

În parcurgerea totală DF, dacă subgraful predecesor $G_p = (N_p, A_p)$, $N_p = \{y \,|\, p(y) \neq 0 \cup S$, $A_p = \{(p(y), y) \,|\, y \in N_p - S\}$ este o pădure şi $N_p = W$, atunci G_p se numeşte *pădure parcurgere* DF.

Teorema 2.10. *Algoritmul PTDF determină elementele tabloului p astfel încât subgraful predecesor $G_p = (N_p, A_p)$ este o pădure parcurgere DF.*

Demonstraţie. Se demonstrează analog ca Teorema 2.3.■

Teorema 2.11. *În orice parcurgere totală DF a unui digraf $G = (N, A)$ pentru oricare două noduri x şi y, una din următoarele trei condiţii este îndeplinită:*
(c1) intervalele $[t_1(x), t_2(x)]$ şi $[t_1(y), t_2(y)]$ sunt disjuncte;
(c2) intervalul $[t_1(x), t_2(x)]$ include strict intervalul $[t_1(y), t_2(y)]$ şi x este un ascendent al lui y în pădurea parcurgere DF;
(c3) intervalul $[t_1(x), t_2(x)]$ este inclus strict în intervalul $[t_1(y), t_2(y)]$ şi x este un descendent al lui y în pădurea parcurgere DF.

Demonstraţie. Există patru posibilităţi de considerat:
(p1) $t_1(x) < t_1(y)$ şi $t_2(x) < t_1(y)$. Rezultă $t_1(x) < t_2(x) < t_1(y) < t_2(y)$, adică (c1).
(p2) $t_1(x) < t_1(y)$ şi $t_2(x) > t_1(y)$. Rezultă că x este mai vechi decât y în V şi deci x va fi introdus după y în W. Evident că x este ascendent al lui y în pădurea parcurgere DF şi $t_2(x) > t_2(y)$, adică (c2).
(p3) $t_1(x) > t_1(y)$ şi $t_1(x) > t_2(y)$. Rezultă $t_1(y) < t_2(y) < t_1(x) < t_2(x)$, adică (c1).
(p4) $t_1(x) > t_1(y)$ şi $t_1(x) < t_2(y)$. Rezultă că x este mai nou decât y în V şi deci x va fi introdus înaintea lui y în W. Evident că x este descendent al lui y în pădurea parcurgere DF şi $t_2(x) < t_2(y)$, adică (c3).■

Parcurgerea totală DF poate fi utilizată la clasificarea arcelor digrafului $G = (N, A)$ în raport cu pădurea parcurgere DF. În această clasificare există două clase de arce:

- arce arborescenţă;
- arce nonarborescenţă.

Mulţimea *arcelor arborescenţă*, notată P, este $P = A_p$. Aceste arce introduc, în timpul parcurgerii totale DF a digrafului $G = (N, A)$, noduri în mulţimea nodurilor vizitate şi neanalizate V. Mulţimea *arcelor nonarborescenţă*, notată $\overline{P}$, este $\overline{P} = A - A_p$. Aceste arce introduc, în timpul parcurgerii totale DF a digrafului $G = (N, A)$, noduri în mulţimea nodurilor vizitate şi analizate W.

Clasa arcelor nonarborescenţă $\overline{P}$ este alcătuită din trei subclase:

- arce de înaintare;
- arce de revenire;
- arce de traversare.

Mulţimea arcelor de *înaintare*, notată I, este alcătuită din acele arce (x, y) pentru care nodul x este un ascendent al nodului y în pădurea parcurgere DF. Mulţimea *arcelor de revenire*, notată R, este alcătuită din acele arce (x, y) pentru care nodul x este un descendent al nodului y în pădurea parcurgere DF. Mulţimea *arcelor de traversare*,

notată T, este alcătuită din acele arce (x, y) pentru care nodul x nu este nici ascendent nici descendent al nodului y în pădurea parcurgere DF. Aceste trei mulţimi de arce nonarborescenţă sunt disjuncte două câte două şi $\overline{P} = I \cup R \cup T$.

Teorema 2.12. *Într-o parcurgere totală DF a unui digraf $G = (N, A)$, arcul (x, y) este:*

(1) un arc arborescenţă sau de înaintare dacă şi numai dacă $t_1(x) < t_1(y) < t_2(y) < t_2(x)$;

(2) un arc de revenire dacă şi numai dacă $t_1(y) < t_1(x) < t_2(x) < t_2(y)$;

(3) un arc de traversare dacă şi numai dacă $t_1(y) < t_2(y) < t_1(x) < t_2(x)$.

Demonstraţie. Rezultă din Teorema 2.11 şi definiţiile clasificării arcelor.■

Observaţia 2.10. Algoritmul PDF sau PTDF se poate aplica şi grafurilor neorientate. În acest caz subgraful predecesor $G_p = (N_p, A_p)$ este tot o arborescenţă respectiv o pădure. Într-o parcurgere totală DF a unui graf neorientat $G = (N, A)$ avem $I = \emptyset$ şi $T = \emptyset$. Într-adevăr, fie $[x, y] \in A$ şi presupunem fără a restrânge generalitatea că $t_1(x) < t_1(y)$. În acest caz, nodul x este introdus înainte de nodul y în V şi după nodul y în W. Dacă muchia $[x, y]$ este explorată de la x la y, atunci $[x, y]$ devine un arc arbore (x, y). Dacă muchia $[x, y]$ este explorată de la y la x, atunci muchia $[x, y]$ devine un arc de revenire (y, x), deoarece x este nod vizitat la acest timp.

Exemplul 2.2. Se aplică algoritmul PTDF digrafului reprezentat în figura 2.3.

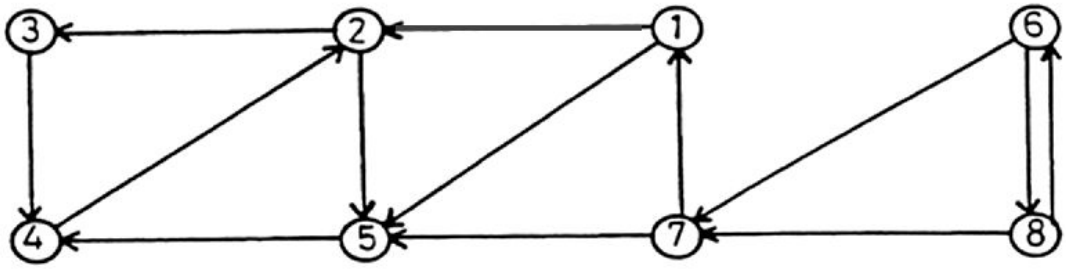


Fig.2.3

Iniţializări: $s = 1, U = \{2, 3, 4, 5, 6, 7, 8\}, V = \{1\}, W = \emptyset$,
$p = (0, 0, 0, 0, 0, 0, 0, 0), t = 1, t_1 = (1, \infty, \infty, \infty, \infty, \infty, \infty, \infty)$,
$t_2 = (\infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty)$.
Iteraţia 1: $x = 1, (1, 2) \in A, 2 \in U : U = \{3, 4, 5, 6, 7, 8\}, V = \{1, 2\}$,
$p = (0, 1, 0, 0, 0, 0, 0, 0), t = 2, t_1 = (1, 2, \infty, \infty, \infty, \infty, \infty, \infty)$.
Iteraţia 2: $x = 2, (2, 3) \in A, 3 \in U : U = \{4, 5, 6, 7, 8\}, V = \{1, 2, 3\}$,
$p = (0, 1, 2, 0, 0, 0, 0, 0), t = 3, t_1 = (1, 2, 3, \infty, \infty, \infty, \infty, \infty)$.
Iteraţia 3: $x = 3, (3, 4) \in A, 4 \in U : U = \{5, 6, 7, 8\}, V = \{1, 2, 3, 4\}$,
$p = (0, 1, 2, 3, 0, 0, 0, 0), t = 4, t_1 = (1, 2, 3, 4, \infty, \infty, \infty, \infty)$.
Iteraţia 4: $x = 4 : V = \{1, 2, 3\}, W = \{4\}, t = 5, t_2 = (\infty, \infty, \infty, 5, \infty, \infty, \infty, \infty)$.
Iteraţia 5: $x = 3 : V = \{1, 2\}, W = \{4, 3\}, t = 6, t_2 = (\infty, \infty, 6, 5, \infty, \infty, \infty, \infty)$.
Iteraţia 6: $x = 2, (2, 5) \in A, 5 \in U : U = \{6, 7, 8\}, V = \{1, 2, 5\}, p = (0, 1, 2, 3, 2, 0, 0, 0), t = 7, t_1 = (1, 2, 3, 4, 7, \infty, \infty, \infty)$.

Iteraţia 7: $x = 5 : V = \{1, 2\}, W = \{4, 3, 5\}, t = 8, t_2 = (\infty, \infty, 6, 5, 8, \infty, \infty, \infty)$.
Iteraţia 8: $x = 2 : V = \{1\}, W = \{4, 3, 5, 2\}, t = 9, t_2 = (\infty, 9, 6, 5, 8, \infty, \infty, \infty)$.
Iteraţia 9: $x = 1 : V = \emptyset, W = \{4, 3, 5, 2, 1\}, t = 10, t_2 = (10, 9, 6, 5, 8, \infty, \infty, \infty)$.
Actualizări: $s = 6, U = \{7, 8\}, V = \{6\}, t = 11, t_1 = (1, 2, 3, 4, 7, 11, \infty, \infty)$
Iteraţia 10: $x = 6, (6, 7) \in A, 7 \in U : U = \{8\}, V = \{6, 7\}, p = (0, 1, 2, 3, 2, 0, 6, 0)$,
$t = 12, t_1 = (1, 2, 3, 4, 7, 11, 12, \infty)$.
Iteraţia 11: $x = 7 : V = \{6\}, W = \{4, 3, 5, 2, 1, 7\}, t = 13, t_2 = (10, 9, 6, 5, 8, \infty, 13, \infty)$.
Iteraţia 12: $x = 6, (6, 8) \in A, 8 \in U : U = \emptyset, V = \{6, 8\}, p = (0, 1, 2, 3, 2, 0, 6, 6)$,
$t = 14, t_1 = (1, 2, 3, 4, 7, 11, 12, 14)$.
Iteraţia 13: $x = 8 : V = \{6\}, W = \{4, 3, 5, 2, 1, 7, 8\}, t = 15, t_2 = (10, 9, 6, 5, 8, \infty, 13, 15)$.
Iteraţia 14: $x = 6 : V = \emptyset, W = \{4, 3, 5, 2, 1, 7, 8, 6\}, t = 16, t_2 = (10, 9, 6, 5, 8, 16, 13, 15)$.

În figura 2.4 se prezintă pădurea parcurgere DF formată din două arborescenţe parcurgere DF şi se prezintă intervalele $[t_1(x), t_2(x)]$.

Fiecare dreptunghi în care este înscris un nod x acoperă intervalul $[t_1(x), t_2(x)]$. Tabloul predecesor este $p = (0, 1, 2, 3, 2, 0, 6, 6)$.

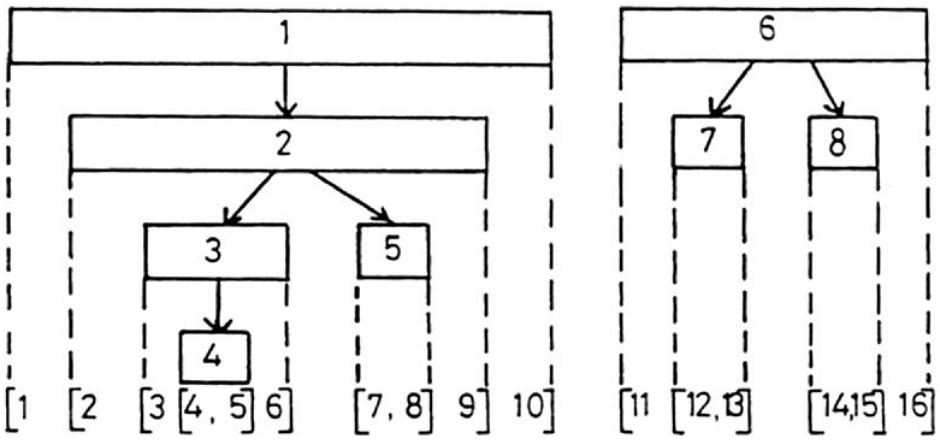


Fig.2.4

În figura 2.5 se prezintă redesenat graful din figura 2.3. Toate arcele arbore şi de înaintare sunt orientate de sus în jos. Pe fiecare arc se specifică clasa la care aparţine.

2.4 Aplicaţii

2.4.1 Sortarea topologică

Teorema 2.13. *Un digraf $G = (N, A)$ este fără circuite dacă şi numai dacă orice parcurgere totală DF a lui G nu produce arce de revenire.*

Demonstraţie. Presupunem că digraful G este fără circuite. Prin reducere la absurd presupunem că o parcurgere totală DF a lui G produce arce de revenire ($R \neq \emptyset$). Fie arcul $(z,\ x) \in R$. În acest caz nodul z este un descendent al nodului x în pădurea parcurgere DF. Deci există un drum D de la x la z în G. Atunci $\overset{\circ}{D} = D \cup \{z, x\}$ este un circuit în G şi aceasta contrazice ipoteza că digraful G este fără circuite.

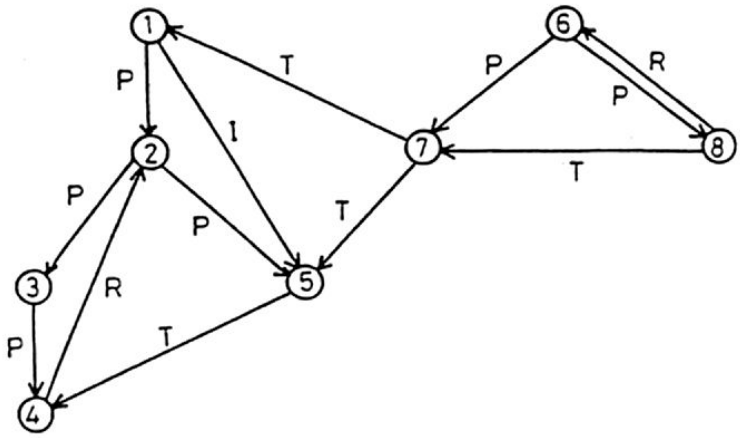


Fig.2.5

Reciproc, presupunem că o parcurgere totală DF a digrafului G nu produce arce de revenire ($R = \emptyset$). Prin reducere la absurd presupunem că G conţine un circuit $\overset{\circ}{D}$. Fie x primul nod vizitat din $\overset{\circ}{D}$ şi fie arcul $(z, x) \in \overset{\circ}{D}$. Deoarece nodurile x, $z \in \overset{\circ}{D}$ rezultă faptul că există un drum D de la x la z. De asemenea x fiind primul nod vizitat din $\overset{\circ}{D}$ rezultă că nodul z devine un descendent al nodului x la pădurea PDF. Deci (z, x) este un arc de revenire ce contrazice ipoteza $R = \emptyset$.■

Sortarea topologică a unui digraf $G = (N, A)$ fără circuite constă într-o ordonare liniară a nodurilor din N astfel încât dacă $(x, y) \in A$ atunci x apare înaintea lui y în ordonare.

Algoritmul sortare topologică (algoritmul ST) se obţine din algoritmul PTDF făcând următoarele două completări:
(1) în partea de iniţializări (liniile (3)-(6)) se iniţializează o listă a nodurilor;
(2) în linia (16) după calculul lui $t_2(x)$, nodul x se introduce la începutul listei.

La terminarea algoritmului ST, lista furnizează sortarea topologică a digrafului $G = (N, A)$ fără circuite şi nodurile x sunt plasate în listă în ordinea des-crescătoare a timpilor $t_2(x)$.

2.4.2 Componentele conexe ale unui graf

Definiţia 2.1 Un digraf $G = (N, A)$ se numeşte *slab conex* sau *conex* dacă pentru oricare două noduri diferite x, y există un lanţ care are aceste două noduri drept extremităţi.

Noţiunea de conexitate are sens şi pentru grafuri neorientate.

Definiţia 2.2. Se numeşte *componentă conexă* a unui digraf $G = (N, A)$ un subgraf $G' = (N', A')$ al lui G, care este conex şi care este maximal în raport cu incluziunea faţă de această proprietate (oricare ar fi $x \in \overline{N}' = N - N'$, subgraful G'_x generat de $N'_x = N' \cup \{x\}$ nu este conex).

O componentă conexă $G' = (N', A')$ a unui digraf $G = (N, A)$ se poate identifica cu mulţimea N' care generează subgraful G'.

Deoarece în problema conexităţii sensul arcelor nu contează se va considera că grafurile sunt neorientate. Dacă $G = (N, A)$ este digraf atunci i se asociază graful neorientat $\widehat{G} = (\widehat{N}, \widehat{A})$, unde $\widehat{N} = N, \widehat{A} = \{[x, y] \mid (x, y) \in A$ şi /sau $(y, x) \in A\}$. Este evident că G şi $\widehat{G}$ au aceleaşi componente conexe.

Algoritmul componentelor conexe (algoritmul CC) este o adaptare a algoritmului PTDF aplicat unui graf neorientat $G = (N, A)$. Nu se calculează tablourile timp t_1, t_2 şi prin p notăm o variabilă scalară a cărei valoare reprezintă numărul componentelor conexe. Algoritmul CC este următorul:

```
(1)   PROGRAM CC;
(2)   BEGIN
(3)     U := N − {s}; V := {s}; W := ∅; p := 1; N' = {s};
(4)     WHILE W ≠ N DO
(5)     BEGIN
(6)       WHILE V ≠ ∅ DO
(7)       BEGIN
(8)         se selectează cel mai nou nod x introdus în V;
(9)         IF există muchie [x, y] ∈ A şi y ∈ U
(10)              THEN U := U − {y}; V := V ∪ {y}; N' := N' ∪ {y}
(11)              ELSE V := V − {x}; W := W ∪ {x};
(12)        END;
(13)        se tipăresc p şi N';
(14)        se selectează s ∈ U; U := U − {s}; V := {s}; p := p + 1;
                          N' := {s};
(15)      END;
(16)  END.
```

La terminarea algoritmului pot exista cazurile:
(c1) se tipăreşte o singură componentă conexă şi în acest caz graful $G = (N, A)$ este conex;
(c2) se tipăresc mai multe componente conexe şi în acest caz graful $G = (N, A)$ nu este conex, obţinându-se toate componentele conexe ale lui G.

Teorema 2.14. *Algoritmul CC determină componentele conexe ale unui graf neorientat* $G = (N, A)$.

Demonstraţie. La terminarea execuţiei ciclului WHILE se determină mulţimea N' a tuturor nodurilor accesibile printr-un lanţ cu aceeaşi extremitate, nodul s. Mulţimea N' generează evident o componentă conexă $G' = (N', A')$. Deoarece la terminarea execuţiei algoritmului avem $W = N$ rezultă că algoritmul CC determină toate componentele conexe ale lui $G = (N, A)$. ■

Teorema 2.15. *Algoritmul CC are complexitatea* $O(m)$.

Demonstraţie. Algoritmul CC are aceeaşi complexitate cu a algoritmului PTDF, adică $O(m)$.■

Exemplul 2.3. Fie digraful din figura 2.6. Pentru a determina componentele conexe ale acestui digraf se transformă într-un graf neorientat reprezentat în figura 2.7 căruia i se aplică algoritmul CC.

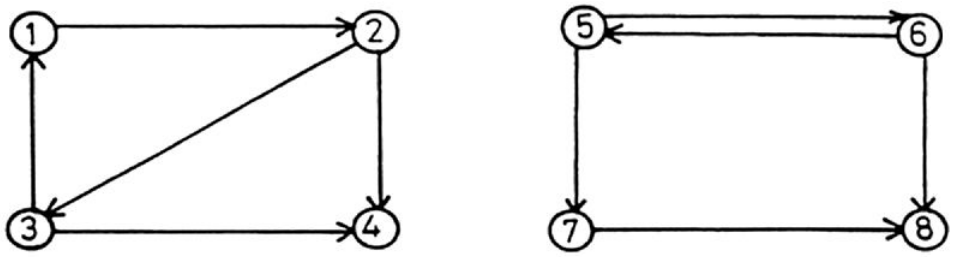


Fig.2.6

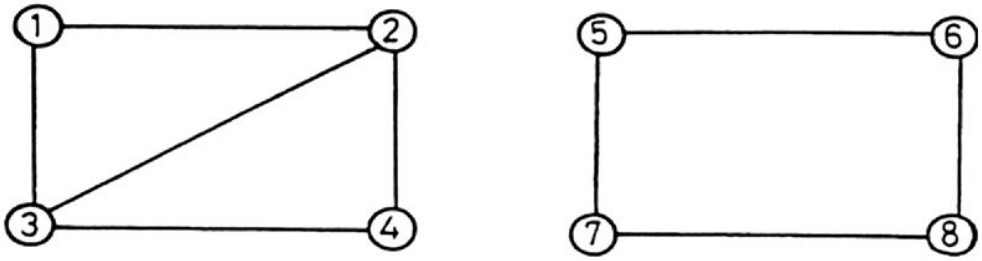


Fig.2.7

Iniţializări: $s = 1, U = \{2,3,4,5,6,7,8\}, V = \{1\}, W = \emptyset, p = 1, N' = \{1\}$.
Iteraţia 1: $x = 1, [1,2] \in A, 2 \in U : U = \{3,4,5,6,7,8\}, V = \{1,2\}, N' = \{1,2\}$.
Iteraţia 2: $x = 2, [2,3] \in A, 3 \in U : U = \{4,5,6,7,8\}, V = \{1,2,3\}, N' = \{1,2,3\}$.
Iteraţia 3: $x = 3, [3,4] \in A, 4 \in U : U = \{5,6,7,8\}, V = \{1,2,3,4\}, N' = \{1,2,3,4\}$.
Iteraţia 4: $x = 4 : V = \{1,2,3\}, W = \{4\}$.
Iteraţia 5: $x = 3 : V = \{1,2\}, W = \{4,3\}$.
Iteraţia 6: $x = 2, V = \{1\}, W = \{4,3,2\}$.
Iteraţia 7: $x = 1 : V = \emptyset, W = \{4,3,2,1\}$.
Se tipăresc $p = 1$ şi $N' = \{1,2,3,4\}$
Actualizări: $s = 5, U = \{6,7,8\}, V = \{5\}, p = 2, N' = \{5\}$.
Iteraţia 8: $x = 5, [5,6] \in A, 6 \in U : U = \{7,8\}, V = \{5,6\}, N' = \{5,6\}$.
Iteraţia 9: $x = 6, [6,8] \in A, 8 \in U : U = \{7\}, V = \{5,6,8\}, N' = \{5,6,8\}$.
Iteraţia 10: $x = 6, [8,7] \in A, 7 \in U : U = \emptyset, V = \{5,6,8,7\}, N' = \{5,6,8,7\}$.
Iteraţia 11: $x = 7 : V = \{5,6,8\}, W = \{4,3,2,1,7\}$.

După încă trei iteraţii se obţine $V = \emptyset, W = \{4,3,2,1,7,8,6,5\}$ se tipăresc $p = 2, N' = \{5,6,8,7\}$ şi execuţia algoritmului se opreşte.

2.4.3 Componentele tare conexe ale unui graf

Definiţia 2.3 Un digraf $G = (N, A)$ se numeşte *tare conex* dacă pentru oricare două noduri x, y există un drum de la x la y şi un drum de la y la x.

Noţiunea de tare conexitate are sens numai pentru grafuri orientate.

Definiţia 2.4. Se numeşte *componentă tare conexă* a unui digraf $G = (N, A)$ un subgraf $G' = (N', A')$ al lui G care este tare conex şi care este maximal în raport cu incluziunea faţă de acestă proprietate (oricare ar fi $x \in \overline{N}' = N - N'$, subgraful G'_x generat de $N'_x = N' \cup \{x\}$ nu este tare conex).

O componentă tare conexă $G' = (N', A')$ a unui digraf $G = (N, A)$ se poate identifica cu mulţimea N' care generează subgraful G'.

Definiţia 2.5. Se numeşte *digraf condensat* asociat digrafului $G = (N, A)$ digraful $\widehat{G} = \left(\widehat{N}, \widehat{A}\right), \widehat{N} = \{N'_1, \ldots, N'_p\}, \widehat{A} = \{\left(N'_i, N'_j\right) \mid N'_i, N'_j \in \widehat{N}$, există $(x, y) \in A$ cu $x \in N'_i, \ , y \in N'_j\}$, unde $N'_i, \ldots, N'_p$ sunt componentele tare conexe ale digrafului $G = (N, A)$.

Algoritmul componentelor tare conexe (algoritmul CTC) este următorul:

```
(1)   PROGRAM CTC;
(2)   BEGIN
(3)      PROCEDURA PTDF(G);
(4)      PROCEDURA INVERSARE(G);
(5)      PROCEDURA PTDF(G^{-1});
(6)      PROCEDURA SEPARARE(G^{-1});
(7)   END.
```

Procedurile din program rezolvă următoarele probleme:
PROCEDURA PTDF(G) aplică algoritmul PTDF digrafului G;
PROCEDURA INVERSARE(G) determină inversul G^{-1} al digrafului G;
PROCEDURA PTDF(G^{-1}) aplică algoritmul PTDF digrafului G^{-1}, unde nodul s este selectat întotdeauna astfel încât $t_2(s) > t_2(x), x \in U^{-1} - \{s\}$ cu t_2 calculat în PROCEDURA PTDF(G);
PROCEDURA SEPARARE(G^{-1}) extrage nodurile fiecărei arborescenţe parcurgere DF din pădurea parcurgere DF obţinută în PROCEDURA PTDF (G^{-1}) pentru separarea componentelor tare conexe.

Se spune că nodul $r(x) \in N$ este *ascendent rădăcină* al nodului x în parcurgerea totală DF a digrafului $G = (N, A)$ dacă $r(x)$ este accesibil din x şi $t_2(r(x)) = \max\{t_2(z) \mid z$ este accesibil din $x\}$.

Teorema 2.16. *În oricare parcurgere totală DF a unui digraf* $G = (N, A)$:
(1) toate nodurile din aceeaşi componentă tare conexă aparţin la aceeaşi arborescenţă a pădurii parcurgere DF;
(2) pentru orice nod $x \in N$, *ascendentul rădăcină* $r(x)$ *este un ascendent al lui* x *în pădurea parcurgere DF şi nodurile* $x, r(x)$ *se găsesc în aceeaşi componentă tare conexă.*

Demonstraţie. Se recomandă cititorului comentariile bibliografice din acest capitol.■

Teorema 2.17. *Algoritmul CTC determină componentele tare conexe ale unui digraf* $G = (N, A)$.

Demonstraţie. Se bazează pe Teorema 2.16.■

Teorema 2.18. *Algoritmul CTC are complexitatea* $O(m)$.

Demonstraţie. Algoritmul CTC are aceeaşi complexitate cu a algoritmului PTDF, adică $O(m)$.■

Exemplul 2.4. Fie digraful din figura 2.8. Pentru a determina componentele tare conexe ale acestui digraf se aplică algoritmul CTC.

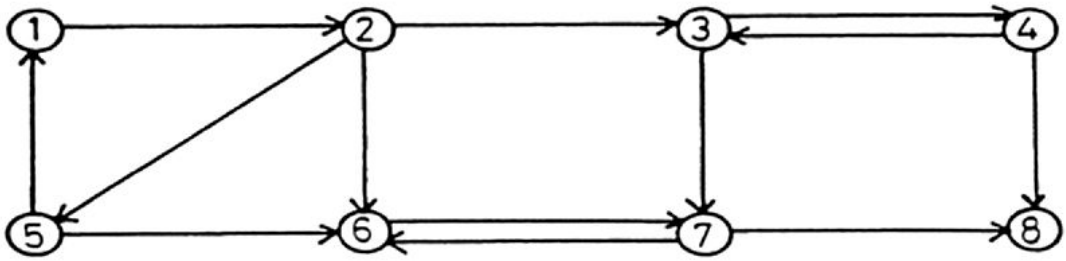


Fig.2.8

PROCEDURA PTDF determină $p = (5, 0, 0, 3, 2, 7, 3, 7), t_1 = (13, 11, 1, 8, 12, 3, 2, 5)$, $t_2 = (14, 16, 10, 9, 15, 4, 7, 6)$. Iniţial $s = 3$ şi apoi $s = 2$.

PROCEDURA INVERSARE determină inversul G^{-1} al digrafului G şi este reprezentat în figura 2.9.

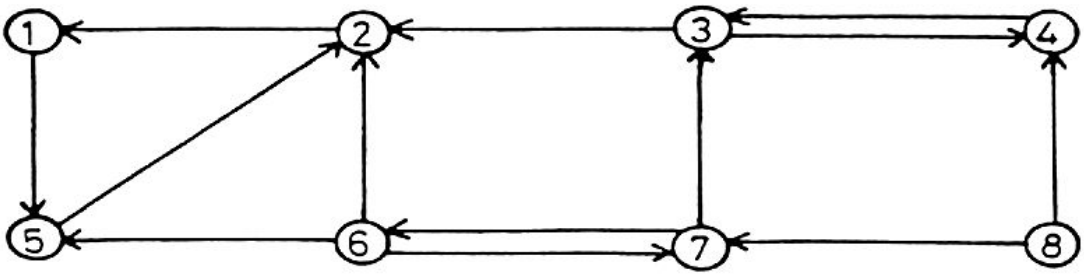


Fig.2.9

PROCEDURA PTDF aplicată digrafului G^{-1} determină: $p^{-1} = (2, 0, 0, 3, 1, 7, 0, 0)$, $t_1^{-1} = (2, 1, 7, 8, 3, 12, 11, 15), t_2^{-1} = (5, 6, 10, 9, 4, 13, 14, 16)$ şi nodurile s au fost selectate astfel înât $t_2(s) > t_2(t),\ , x \in U^{-1} - \{s\}$.

PROCEDURA SEPARARE determină componentele tare conexe reprezentate ca noduri ale digrafului condensat $\widehat{G} = (\widehat{N}, \widehat{A})$, din figura 2.10.

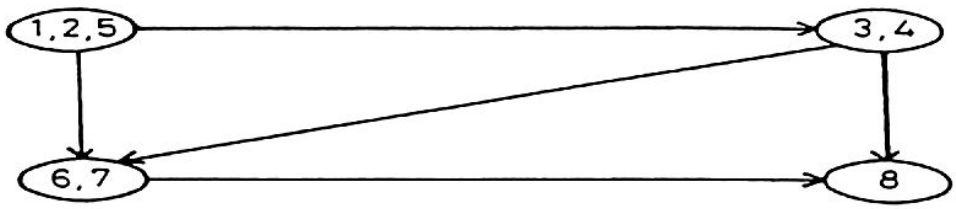


Fig.2.10

Capitolul 3

Arbori şi arborescenţe

3.1 Cicluri şi arbori

În acest paragraf se va lucra cu cicluri şi cicluri elementare.

Fie digraful $G = (N, A)$ cu $N = \{1, \ldots, n\}$ şi $A = \{a_1, \ldots, a_m\}$. Pentru un ciclu $\overset{\circ}{L}$ vom nota cu $\overset{\circ}{L}^+$ mulţimea arcelor directe şi prin $\overset{\circ}{L}^-$ mulţimea arcelor inverse ale ciclului. Unui ciclu $\overset{\circ}{L}$ i se poate asocia un vector $\overset{\circ}{l} = \left(\overset{\circ}{e}_1, \ldots, \overset{\circ}{e}_m\right)$, unde

$$\overset{\circ}{e}_i = \begin{cases} -1 & \text{dacă} \quad a_i \in \overset{\circ}{L}^-, \\ 1 & \text{dacă} \quad a_i \in \overset{\circ}{L}^+, \\ 0 & \text{dacă} \quad a_i \notin \overset{\circ}{L} \end{cases}$$

În acest paragraf un ciclu $\overset{\circ}{L}$ va fi identificat uneori cu vectorul $\overset{\circ}{l}$ pe care îl defineşte, iar orice sumă de cicluri $\overset{\circ}{L}_1 + \ldots + \overset{\circ}{L}_s$ va fi suma lor vectorială $\overset{\circ}{l}_1 + \ldots + \overset{\circ}{l}_s$.

Teorema 3.1. *Orice ciclu $\overset{\circ}{L}$ al digrafului $G = (N, A)$ este o sumă de cicluri elementare fără arce comune.*

Demonstraţie. Când se parcurge ciclul $\overset{\circ}{L}$ se obţin cicluri elementare $\overset{\circ}{L}_i$ $(i = 1, \ldots, s')$, de fiecare dată când se ajunge într-un nod deja întâlnit. Ciclurile elementare nu au arce comune deoarece ciclul $\overset{\circ}{L}$ nu are arce care se repetă. ■

Definiţia 3.1. Se spune că ciclurile $\overset{\circ}{l}_1, \ldots, \overset{\circ}{l}_s$ sunt *independente* dacă egalitatea $r_1 \overset{\circ}{l}_1 + \ldots + r_s \overset{\circ}{l}_s = 0$ cu $r_1, \ldots, r_s$ numere reale implică $r_1 = r_2 = \ldots = r_s = 0$, altfel se spune că ciclurile $\overset{\circ}{l}_1, \ldots, \overset{\circ}{l}_s$ sunt *dependente*.

Definiţia 3.2. Se spune că ciclurile $\overset{\circ}{l}_1, \ldots, \overset{\circ}{l}_s$ formează *o bază* de cicluri dacă sunt cicluri elementare independente şi oricare alt ciclu $\overset{\circ}{l}$ se poate exprima sub forma $\overset{\circ}{l} = r_1 \overset{\circ}{l}_1 + \ldots + r_s \overset{\circ}{l}_s$ cu $r_1, \ldots, r_s$ numere reale.

Numărul s al ciclurilor care formează o bază este *dimensiunea* subspaţiului liniar $S_{\overset{\circ}{l}}$ al lui $\Re^m$ generat de ciclurile digrafului $G = (N, A)$.

Exemplul 3.1. Fie digraful din figura 3.1. Ciclul $\overset{\circ}{L}= (a_1, a_4, a_9, a_8, a_7, a_6)$ este suma ciclurilor elementare fără arce comune $\overset{\circ}{L}_1= (a_1, a_4, a_6)$ şi $\overset{\circ}{L}_2= (a_9, a_8, a_7)$ sau $\overset{\circ}{l}=\overset{\circ}{l}_1 + \overset{\circ}{l}_2$ cu $\overset{\circ}{l}= (1,0,0,1,0,1,1,-1,-1)$, $\overset{\circ}{l}_1= (1,0,0,1,0,1,0,0,0), \overset{\circ}{l}_2= (0,0,0,0,0,0,1,-1,-1)$.

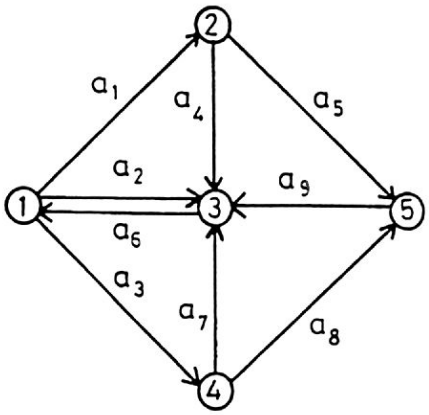


Fig.3.1

Evident că ciclurile $\overset{\circ}{l}, \overset{\circ}{l}_1, \overset{\circ}{l}_2$ sunt dependente.

Definiţia 3.3. Se numeşte *număr ciclomatic* al digrafului $G = (N, A)$ cu n noduri, m arce şi p componente conexe numărul $\nu(G) = m - n + p$.

Teorema 3.2. *Subspaţiul $S_{\overset{\circ}{l}}$ al lui $\Re^m$, generat de ciclurile digrafului $G = (N, A)$, admite o bază formată din $\nu(G)$ cicluri elementare independente.*

Demonstraţie. Vom arăta că digraful G admite $\nu(G) = m - n + p$ cicluri elementare independente. Pentru aceasta vom construi un şir de grafuri parţiale $G_i = (N, A_i)$ ale digrafului $G = (N, A)$ cu $\nu(G_i) = m_i - n + p_i$ cicluri elementare independente. Acest şir se construieşte cu algoritmul următor.

```
(1)  PROGRAM CICLURI;
(2)  BEGIN
(3)      A_0 := ∅;
(4)      FOR i := 1 TO m DO A_i := A_{i-1} ∪ {a_i};
(5)  END.
```

Prin inducţie după i vom arăta că $G_i = (N, A_i)$ admite $\nu(G_i) = m_i - n + p_i$ cicluri elementare independente. Pentru $i = 0$ avem $A_0 = \emptyset$ şi obţinem $m_0 = 0$, deci $\nu(G_0) = 0$. Deoarece $p_0 = n$ rezultă că $\nu(G_0) = m_0 - n + p_0$.

Presupunem afirmaţia adevărată pentru i şi o demonstrăm pentru $i + 1$.

La iteraţia $i + 1$ poate exista unul din cazurile:

(c1) arcul a_{i+1} nu închide un nou ciclu elementar. Deci $\nu(G_{i+1}) = \nu(G_i)$ şi $m_{i+1} = m_i + 1, p_{i+1} = p_i - 1$. Rezultă $\nu(G_{i+1}) = m_i - n + p_i = m_{i+1} - n + p_{i+1}$.

(c2) arcul a_{i+1} închide un nou ciclu elementar $\overset{\circ}{L}_k$. Deoarece arcul a_{i+1} aparţine ciclului elementar $\overset{\circ}{L}_k$ şi nu aparţine ciclurilor elementare independente $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_{k-1}$ obţinute până la această iteraţie, rezultă că ciclurile elementare $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_{k-1}, \overset{\circ}{L}_k$ sunt independente. Rezultă că $\nu(G_{i+1}) = \nu(G_i) + 1,\ m_{i+1} = m_i + 1,\ p_{i+1} = p_i$. Deci $\nu(G_{i+1}) = m_i - n + p_i + 1 = m_{i+1} - n + p_{i+1}$.

Deci afirmaţia este adevărată pentru orice i. La terminarea execuţiei algoritmului avem $G_m = G, m_m = m, p_m = p$ şi $\nu(G) = \nu(G_m) = m_m - n + p_m = m - n + p$.

În continuare vom arăta că G nu admite mai mult de $\nu(G) = m - n + p$ cicluri elementare independente. Fie $\overset{\circ}{L}$ un ciclu elementar diferit de ciclurile $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_\nu$ construite cu algoritmul de mai sus. Din modul cum au fost determinate ciclurile elementare $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_\nu$ rezultă că oricare ar fi un arc a_j al ciclului $\overset{\circ}{L}$ există cel puţin un $\overset{\circ}{L}_i,\ i \in \{1, \ldots, \nu\}$, astfel încât a_j aparţine ciclului $\overset{\circ}{L}_i$. Deci există o relaţie $r_1 \overset{\circ}{l}_1 + \ldots + r_\nu \overset{\circ}{l}_\nu + r \overset{\circ}{l} = 0$ fără ca $r_1 = \ldots = r_\nu = r = 0$. Această relaţie implică faptul că ciclurile $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_\nu, L$ sunt dependente. Am demonstrat că $dim\left(S_{\overset{\circ}{l}}\right) = \nu(G) = m - n + p$.

■

Definiţia 3.4. Se spune că un digraf $G' = (N, A')$ este un *arbore* dacă este un digraf fără cicluri şi conex. Se spune că digraful $G' = (N, A')$ este o *pădure* dacă fiecare componentă conexă a lui G' este un arbore.

Din definiţie rezultă că o pădure este un digraf $G' = (N, A)$ fără cicluri. Noţiunile de arbore şi pădure au sens şi pentru grafuri neorientate.

Exemplul 3.2. Digraful reprezentat în figura 3.2. este o pădure compusă din doi arbori. Orientarea arcelor poate fi ignorată.

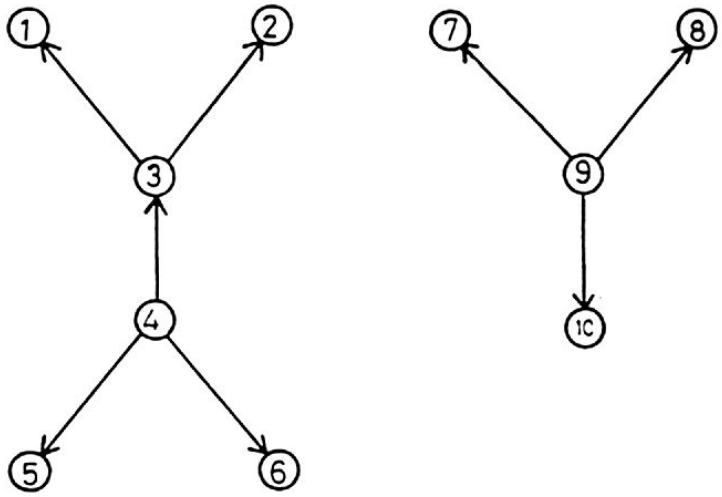


Fig.3.2

Teorema 3.3. *Fie $G' = (N,\ A')$ un digraf cu $n \geq 2$. Proprietăţile următoare sunt echivalente şi caracterizează un arbore:*
(1). G' este fără cicluri şi conex;
(2). G' este fără cicluri şi are $n-1$ arce;
(3). G' este conex şi are $n-1$ arce;
(4). G' este fără cicluri şi dacă se adaugă un singur arc între oricare două noduri atunci graful obţinut $\widetilde{G}' = \left(N, \widetilde{A}'\right)$ conţine un ciclu unic;
(5). G' este conex şi dacă se elimină un arc oarecare, atunci digraful obţinut $\widehat{G}' = \left(N, \widehat{A}'\right)$ nu este conex;
(6). G' are un lanţ unic între oricare două noduri ale lui.
Demonstraţie. (1) $\Rightarrow$ (2). Din (1) rezultă că $\nu(G') = 0$ şi $p' = 1$. Deci $m' - n + 1 = 0$, de unde $m' = n - 1$. Deducem că G' este fără cicluri şi are $n-1$ arce.
(2) $\Rightarrow$ (3). Din (2) rezultă că $\nu(G') = 0$ şi $m' - n + p' = n - 1 - n + p' = 0$, de unde $p' = 1$. Deducem că G' este conex şi are $n-1$ arce.
(3) $\Rightarrow$ (4). Din (3) rezultă $p' = 1$ şi $m' = n-1$. Deci $\nu(G') = m' - n + p' = n - 1 - n + 1 = 0$ şi G' este fără cicluri. Digraful $\widetilde{G}' = (N, \widetilde{A}')$ are $\widetilde{p}' = p' = 1$, $\widetilde{m}' = m' + 1 = n$ şi $\nu\left(\widetilde{G}'\right) = \widetilde{m}' - n + \widetilde{p}' = 1$. Rezultă că $\widetilde{G}'$ are un singur ciclu.
(4) $\Rightarrow$ (5). Din (4) rezultă $\nu(G') = 0$ şi $p' = 1$. Într-adevăr, dacă $p' > 1$ atunci arcul adăugat la A' poate lega două componente conexe şi $\widetilde{G}'$ nu conţine ciclu. Din $\nu(G') = 0$ şi $p' = 1$ rezultă $m' = n - 1$. Digraful $\widehat{G}' = (N, \widehat{A}')$ are $\widehat{m}' = m' - 1 = n - 2$, $\nu(\widehat{G}') = 0, \widehat{p}' = 2$ şi deci $\widehat{G}'$ nu este conex.
(5) $\Rightarrow$ (6). Dacă G' este conex atunci între oricare două noduri există un lanţ. Deoarece $\widehat{G}'$ este neconex rezultă că lanţul este unic.
(6) $\Rightarrow$ (1). Dacă între oricare două noduri ale lui G' există un lanţ unic atunci este evident că G' este conex şi fără cicluri.■
Teorema 3.4. *Un digraf $G = (N, A)$ admite un subgraf parţial $G' = (N, A')$ care este un arbore dacă şi numai dacă G este conex.*
Demonstraţie. Obţinem subgraful parţial $G' = (N, A')$ cu următorul algoritm:

```
(1)  PROGRAM ARBORE1;
(2)  BEGIN
(3)      A_m := A;
(4)      FOR i := m DOWNTO n DO
(5)      BEGIN
(6)          se selectează un arc a din A_i care nu deconexează G_i;
(7)          A_{i-1} := A_i - {a};
(8)      END;
(9)  END.
```

Se obţine $G' = (N, A') = (N, A_{n-1})$. Din modul cum este construit G' rezultă că el este conex şi are $n-1$ arce. Deci G' este arbore.

Reciproca este evidentă. ■

Deoarece arborele G' construit în Teorema 3.4 are aceeaşi mulţime de noduri ca a digrafului G se numeşte *arbore parţial* al lui G.

Observaţia 3.1. Un arbore parţial $G' = (N, A')$ al digrafului $G = (N, A)$ se poate construi şi cu algoritmul următor:

```
(1)  PROGRAM ARBORE2;
(2)  BEGIN
(3)     A_0 := ∅;
(4)     FOR i := 0 TO n − 2 DO
(5)     BEGIN
(6)        se selectează un arc a din Ā_i = A − A_i care nu formează
           ciclu în G_i;
(7)        A_{i+1} := A_i ∪ {a};
(8)     END;
(9)  END.
```

Se obţine $G' = (N, A') = (N, A_{n-1})$. Deoarece G' este fără ciclu şi are $n-1$ arce el este un arbore.

Teorema 3.5. *Dacă $G = (N, A)$ este un digraf conex, $G' = (N, A')$ este un arbore parţial al lui G şi $\overline{A}' = A - A'$, atunci oricare arc $a_i \in \overline{A}'$ determină cu arcele din A' un ciclu unic $\overset{\circ}{L}_k$ şi ciclurile $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_s$ obţinute în acest mod cu toate arcele din $\overline{A}'$ formează o bază de cicluri a digrafului G.*

Demonstraţie. Dacă arcul $a_i \in \overline{A}'$, atunci digraful $\widetilde{G}' = (N, \widetilde{A}')$ cu $\widetilde{A}' = A' \cup \{a_i\}$ conţine un ciclu $\overset{\circ}{L}_k$ conform proprietăţii (4) din Teorema 3.3. Ciclurile $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_s$ obţinute în acest mod sunt independente, deoarece fiecare ciclu $\overset{\circ}{L}_k$ conţine un arc a_i care nu este conţinut de celelalte cicluri. Numărul acestor cicluri este $s = |\overline{A}'| = |A| - |A'| = m - (n-1) = m - n + 1 = m - n + p = \nu(G)$. Deci ciclurile $\overset{\circ}{L}_1, \ldots, \overset{\circ}{L}_s$ formează o bază de cicluri.■

Teorema 3.5 ne furnizează un algoritm pentru a construi o bază de cicluri a unui digraf conex G.

Exemplul 3.3. Fie digraful $G = (N, A)$ reprezentat în figura 3.1. Un arbore parţial $G' = (N, A')$, $A' = \{a_1, a_2, a_3, a_5\}$ al digrafului G este reprezentat în figura 3.3. Avem $\overline{A}' = \{a_4, a_6, a_7, a_8, a_9\}$.

Se obţin ciclurile $\overset{\circ}{L}_1 = (a_1, a_4, a_2)$, $\overset{\circ}{L}_2 = (a_1, a_5, a_9, a_2)$, $\overset{\circ}{L}_3 = (a_1, a_5, a_8, a_3)$, $\overset{\circ}{L}_4 = (a_2, a_6)$, $\overset{\circ}{L}_5 = (a_2, a_7, a_3)$.

3.2 Arbori parţiali minimi

3.2.1 Condiţii de optimalitate

Se consideră un graf neorientat şi conex $G = (N, A)$ cu $N = \{1, \ldots, x, \ldots, n\}$, $A = \{1, \ldots, a, \ldots, m\}$ şi o funcţie $b : A \to \Re$ care asociază fiecărei muchii $a \in A$ un număr real $b(a)$ numit *valoarea* acestei muchii, care poate reprezenta lungime, cost etc.

Deoarece graful $G = (N, A)$ este conex, el admite un arbore parţial $G' = (N, A')$.

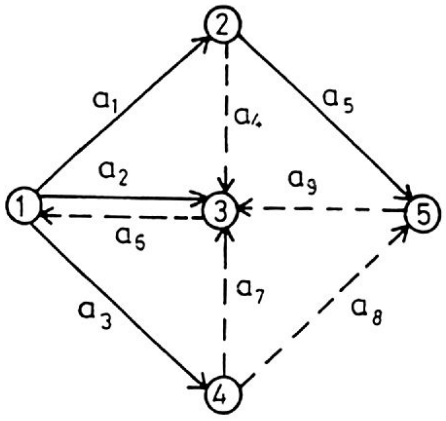


Fig.3.3

Valoarea unui arbore parţial $G' = (N, A')$, notată $b(G')$, este prin definiţie

$$b(G') = \sum_{A'} b(a).$$

Dacă notăm cu $\mathcal{A}_G$ mulţimea arborilor parţiali ai grafului G, atunci problema arborelui parţial minim este următoarea: să se determine $G' \in \mathcal{A}_G$ astfel încât $b(G') = min\{b(G'') \mid G'' \in \mathcal{A}_G\}$.

Dacă eliminăm o muchie arbore oarecare $a = [x, \overline{x}]$ din A' atunci se obţine o partiţie a mulţimii nodurilor $N = X \cup \overline{X},\ x \in X,\ \overline{x} \in \overline{X}$.

Mulţimea de muchii

$$[X, \overline{X}]_a = \{[y, \overline{y}] \mid [y, \overline{y}] \in A,\ y \in X,\ \overline{y} \in \overline{X}\}$$

se numeşte *tăietură* generată de eliminarea muchiei arbore $a = [x, \overline{x}]$ sau *cociclu* generat de eliminarea muchiei arbore $a = [x, \overline{x}]$.

Pentru problema arborelui parţial minim se pot formula condiţii de optimalitate în două moduri: condiţii de optimalitate ale tăieturii şi condiţii de optimalitate ale lanţului.

Teorema 3.6. (Condiţiile de optimalitate ale tăieturii)
Fie $G = (N, A)$ un graf neorientat şi conex. Un arbore parţial $G' = (N, A')$ al grafului G este minim dacă şi numai dacă, pentru fiecare muchie arbore $a = [x, \overline{x}] \in A'$, avem:

$$b[x, \overline{x}] \leq b[y, \overline{y}] \text{ pentru toate muchiile } [y, \overline{y}] \in [X, \overline{X}]_a \tag{3.1}$$

Demonstraţie. Presupunem că arborele parţial $G' = (N, A')$ al grafului $G = (N, A)$ este minim. Prin reducere la absurd presupunem că nu sunt satisfăcute condiţiile (3.1). Atunci există o muchie arbore $a = [x, \overline{x}] \in A'$ şi o muchie nonarbore $[y, \overline{y}] \in [X, \overline{X}]_a$ cu $b[x, \overline{x}] > b[y, \overline{y}]$. Arborele parţial $G'' = (N, A'')$ cu $A'' = A' - \{[x, \overline{x}]\} \cup \{[y, \overline{y}]\}$ are valoarea $b(G'') = b(G') - b[x, \overline{x}] + b[y, \overline{y}]$. Deoarece $b[x, \overline{x}] > b[y, \overline{y}]$ rezultă $b(G'') < b(G')$. Această relaţie contrazice faptul că arborele parţial G' este minim. Deci condiţiile (3.1) sunt satisfăcute.

Presupunem că sunt satisfăcute condiţiile (3.1). Prin reducere la absurd presupunem că arborele parţial $G' = (N, A')$ nu este minim. Atunci există un arbore parţial $G'_0 = (N, A'_0)$ cu $b(G'_0) < b(G')$. Deoarece $A'_0 \neq A'$, există o muchie $a = [x, \overline{x}] \in A'$ şi $a = [x, \overline{x}] \notin A'_0$. Muchia $a = [x, \overline{x}]$ generează tăietura $[X, \overline{X}]_a$ şi un ciclu $\overset{\circ}{L}_a$ cu muchiile din A'_0 (vezi figura 3.4). Atunci există o muchie $[y, \overline{y}] \in \overset{\circ}{L}_a$ astfel încât $[y, \overline{y}] \in [X, \overline{X}]_a$, $[y, \overline{y}] \in A'_0$ şi $[y, \overline{y}] \notin A'$. Deoarece condiţiile (3.1) sunt satisfăcute avem $b[x, \overline{x}] \leq b[y, \overline{y}]$. Arborele parţial $G'' = (N, A'')$ cu $A'' = A' - \{[x, \overline{x}]\} \cup \{[y, \overline{y}]\}$ are valoarea $b(G'') = b(G') - b[x, \overline{x}] + b[y, \overline{y}]$. Rezultă $b(G') \leq b(G'')$ şi G'' are în comun cu G'_0 cu o muchie mai mult decât G'. Iterând acest procedeu se obţine un şir de arbori parţiali $G', G'', \ldots, G'_0$ cu $b(G') \leq b(G'') \leq \ldots \leq b(G'_0)$. Deci obţinem $b(G') \leq b(G'_0)$ care contrazice presupunerea că $b(G'_0) < b(G')$. Rezultă că arborele parţial G' este minim. ■

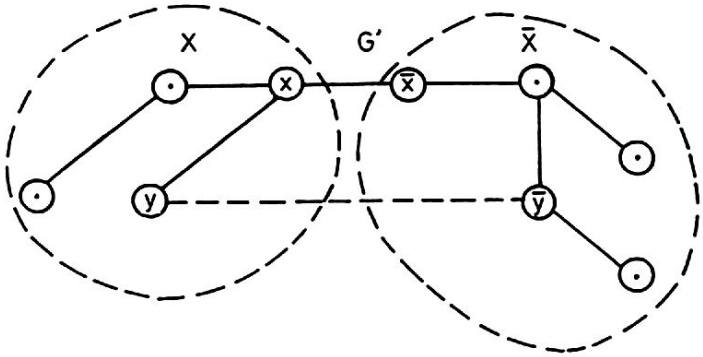


Fig.3.4

Între două noduri $y, \overline{y} \in N$ există un lanţ unic $L'_{y\overline{y}}$ în arborele parţial $G' = (N, A')$ al grafului neorientat şi conex $G = (N, A)$.

Teorema 3.7. (Condiţiile de optimalitate ale lanţului)
Fie $G = (N, A)$ un graf neorientat şi conex. Un arbore parţial $G' = (N, A')$ al grafului $G = (N, A)$ este minim dacă şi numai dacă, pentru fiecare muchie nonarbore $[y, \overline{y}] \in \overline{A}'$, avem

$$b[x, \overline{x}] \leq b[y, \overline{y}] \text{ pentru toate muchiile } [x, \overline{x}] \in L'_{y\overline{y}} \tag{3.2}$$

Demonstraţie. Presupunem că arborele parţial $G' = (N, A')$ al grafului $G = (N, A)$ este minim. Prin reducere la absurd presupunem că nu sunt satisfăcute condiţiile (3.2). Atunci există o muchie nonarbore $[y, \overline{y}] \in \overline{A}'$ şi o muchie arbore $[x, \overline{x}] \in L'_{y\overline{y}}$ cu $b[x, \overline{x}] > b[y, \overline{y}]$. Arborele parţial $G'' = (N, A'')$ cu $A'' = A' - \{[x, \overline{x}]\} \cup \{[y, \overline{y}]\}$ are valoarea $b(G'') = b(G') - b[x, \overline{x}] + b[y, \overline{y}]$. Rezultă că $b(G'') < b(G')$ care contrazice faptul că arborele parţial G' este minim. Deci condiţiile (3.2) sunt satisfăcute.
Presupunem că sunt satisfăcute condiţiile (3.2). Fie o muchie arbore oarecare $a = [x, \overline{x}] \in A'$ şi $[X, \overline{X}]_a$ tăietura generată de muchia a. Se consideră o muchie oarecare nonarbore $[y, \overline{y}] \in [X, \overline{X}]_a$. Atunci există lanţul unic $L'_{y\overline{y}}$ în G' astfel încât $[x, \overline{x}] \in L'_{y\overline{y}}$ (vezi figura 3.4). Condiţiile (3.2) implică faptul că $b[x, \overline{x}] \leq b[y, \overline{y}]$. Deoarece muchiile $[x, \overline{x}]$, $[y, \overline{y}]$ sunt oarecare rezultă că sunt satisfăcute condiţiile (3.1). Conform Teoremei 3.6 arborele parţial G' este minim.■

Observaţia 3.2. Suficienţa Teoremei 3.7 a fost demonstrată utilizând suficienţa Teoremei 3.6. Aceasta stabileşte echivalenţa dintre condiţiile de optimalitate ale tăieturii şi condiţiile de optimalitate ale lanţului.

În continuare se prezintă algoritmi pentru determinarea arborelui parţial minim. Mai întâi se prezintă algoritmul generic. Ceilalţi algoritmi sunt cazuri speciale ale algoritmului generic.

3.2.2 Algoritmul generic

Fie $G = (N, A, b)$ o reţea neorientată şi conexă cu mulţimea nodurilor $N = \{1, \ldots, n\}$ şi mulţimea muchiilor $A = \{1, \ldots, a, \ldots, m\},\ a = [x, \overline{x}]$.

```
(1)   PROGRAM GENERIC;
(2)   BEGIN
(3)      FOR i := 1 TO n DO
(4)      BEGIN
(5)         N_i := {i}; A'_i := ∅;
(6)      END;
(7)      FOR k := 1 TO n − 1 DO
(8)      BEGIN
(9)         se selectează N_i cu N_i ≠ ∅;
(10)        se selectează [y, ȳ] cu b[y, ȳ] := min{b[x, x̄] | x ∈ N_i, x̄ ∉ N_i};
(11)        se determină indicele j pentru care ȳ ∈ N_j;
(12)        N_i := N_i ∪ N_j; N_j := ∅; A'_i = A'_i ∪ A'_j ∪ {[y, ȳ]}; A'_j = ∅;
(13)        IF k = n − 1
(14)           THEN A' := A'_i;
(15)     END;
(16)  END.
```

Teorema 3.8. *Algoritmul generic determină un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$.*

Demonstraţie. Prin inducţie după k, numărul de iteraţii ale ciclului FOR de la linia (7) la linia (15), vom arăta că $G'_i = (N_i, A'_i)$ este conţinut într-un arbore parţial minim al lui G'. Pentru $k = 0$ (înainte de prima iteraţie a ciclului) avem $G'_i = (\{i\}, \emptyset)$ şi afirmaţia este adevărată. Presupunem afirmaţia adevărată pentru execuţia a $k - 1$ iteraţii ale ciclului FOR. Aceasta înseamnă că $G'_i = (N_i, A'_i)$, determinat după execuţia a $k-1$ iteraţii ale ciclului FOR este conţinut într-un arbore parţial minim $G' = (N, A')$ al lui G'.

După execuţia a k iteraţii ale ciclului FOR, referitor la muchia $[y, \overline{y}]$ selectată în linia (10), pot exista cazurile:
(c1) $[y, \overline{y}] \in A'$;
(c2) $[y, \overline{y}] \notin A'$.

În cazul (c1) demonstraţia prin inducţie după k s-a terminat. În cazul (c2), există în G' un lanţ unic $L'_{y\overline{y}}$. Deoarece $y \in N_i$ şi $\overline{y} \notin N_i$, evident că există o muchie $[x, \overline{x}]$ astfel încât $[x, \overline{x}] \in L'_{y\overline{y}'}$ cu $x \in N_i$ şi $\overline{x} \notin N_i$. Conform Teoremei 3.7 avem $b[x, \overline{x}] \leq b[y, \overline{y}]$.

Pe de altă parte, muchia $[y, \overline{y}]$ este selectată în linia (10) astfel încât $b[x, \overline{x}] \geq b[y, \overline{y}]$. Rezultă că $b[x, \overline{x}] = b[y, \overline{y}]$. Arborele parţial $G'' = (N, A'')$ cu $A'' = A' - \{[x, \overline{x}]\} \cup \{[y, \overline{y}]\}$ are valoarea $b(G'') = b(G') - b[x, \overline{x}] + b[y, \overline{y}] = b(G')$. Deci G'' este un arbore parţial minim al lui G şi conţine $G'_i = (N_i, A'_i)$.

Evident că pentru $k = n - 1$ $G'_i = (N_i, A'_i)$ are $N_i = N, |A'_i| = n - 1$ şi $G' = G'_i$ este arborele parţial minim al grafului G. ■

Observaţia 3.3. Nu se poate da complexitatea precisă a algoritmului generic, deoarece ea depinde atât de modul de selectare a mulţimii N_i în linia (9) cât şi de modul de implementare.

În algoritmii care vor fi prezentaţi în continuare se precizează selectarea din liniile (9) şi (10) ale algoritmului generic.

3.2.3 Algoritmul Prim

Fie $G = (N, A, b)$ o reţea neorientată şi conexă cu mulţimea nodurilor $N = \{1, \ldots, n\}$ şi mulţimea muchiilor $A = \{1, \ldots, a, \ldots, m\}$, $a = [x, \overline{x}]$. Graful G este reprezentat prin listele de incidenţă $E(x)$. Algoritmul utilizează două funcţii: $v : N \to \Re$ şi $e : N \to A$ pentru a selecta mai uşor muchia $[y, \overline{y}]$ din linia (10) a algoritmului generic.

```
(1)   PROGRAM PRIM;
(2)   BEGIN
(3)      v(1) := 0; N_1 := ∅; A' := ∅; N̄_1 := N − N_1;
(4)      FOR i := 2 TO n DO
(5)         v(i) := ∞;
(6)      WHILE N_1 ≠ N DO
(7)      BEGIN
(8)         v(y) := min{v(x̄) | x̄ ∈ N̄_1};
(9)         N_1 := N_1 ∪ {y}; N̄_1 := N̄_1 − {y};
(10)        IF y ≠ 1
(11)           THEN A' := A' ∪ {e(y)};
(12)        FOR [y, ȳ] ∈ E(y) ∩ [N_1, N̄_1] DO
(13)           IF v(ȳ) > b[y, ȳ]
(14)              THEN BEGIN v(ȳ) := b[y, ȳ], e(ȳ) := [y, ȳ];END;
(15)     END;
(16)  END.
```

Teorema 3.9. *Algoritmul Prim determină un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$.*

Demonstraţie. Algoritmul Prim s-a obţinut făcând următoarele două modificări în algoritmul generic:

- în linia (9) se selectează întotdeauna N_1;
- selectarea muchiei $[y, \overline{y}]$ în linia (10) se face cu ajutorul funcţiilor v şi e.

Deci, conform Teoremei 3.8 algoritmul Prim este corect.■

Teorema 3.10. *Algoritmul Prim are complexitatea $O(n^2)$.*

Demonstraţie. Ciclul WHILE se execută de n ori. În fiecare iteraţie se execută, în linia (8), $\overline{n}_1$ comparaţii cu $\overline{n}_1 = |\overline{N}_1|$ şi $1 \leq \overline{n}_1 < n$. Ciclul FOR se execută de cel mult $n-1$ ori. Rezultă că algoritmul Prim are complexitatea $O\left(n^2\right)$. ■

Exemplul 3.4. Fie reţeaua $G = (N, A, b)$ reprezentată în figura 3.5(a). Să se determine un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$ cu algoritmul Prim.

Iteraţia 1: $v(1) = 0,\ N_1 = \{1\},\ A' = \emptyset;\ v(2) = 35,\ e(2) = [1,2],\ v(3) = 40,\ e(3) = [1,3.]$

Iteraţia 2: $v(2) = 35,\ N_1 = \{1,2\},\ A' = \{[1,2]\};\ v(3) = 25,\ e(3) = [2,3],\ v(4) = 10,\ e(4) = [2,4.]$

Iteraţia 3: $v(4) = 10,\ N_1 = \{1,2,4\},\ A' = \{[1,2],\ [2,4]\};\ v(3) = 20,\ e(3) = [4,3],\ v(5) = 30,\ e(5) = [4,5.]$

Iteraţia 4: $v(3) = 20,\ N_1 = \{1,2,4,3\},\ A' = \{[1,2],\ [2,4],\ [4,3]\};\ v(5) = 15,\ e(5) = [3,5].$

Iteraţia 5: $v(5) = 15,\ N_1 = \{1,2,4,3,5\},\ A' = \{[1,2],\ [2,4],\ [4,3],\ [3,5]\}; [N_1, \overline{N}_1] = \emptyset.$

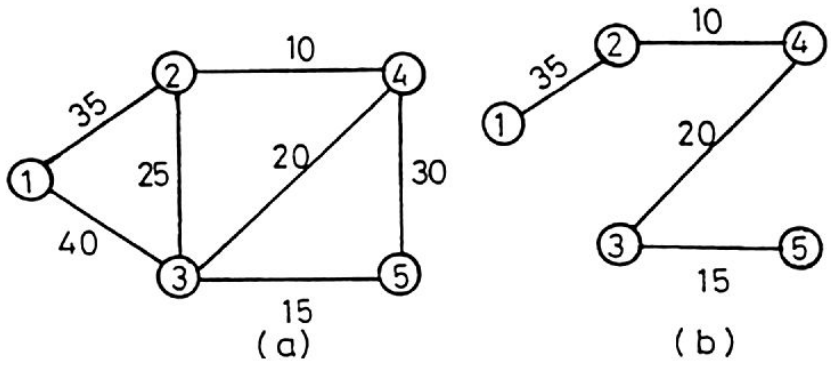


Fig.3.5

Arborele parţial minim $G' = (N, A')$ obţinut este reprezentat în figura 3.5(b).

3.2.4 Algoritmul Kruskal

Fie $G = (N, A, b)$ o reţea neorientată şi conexă cu mulţimea nodurilor $N = \{1, \ldots, n\}$ şi mulţimea muchiilor $A = \{1, \ldots, a, \ldots, m\},\ a = [x, \overline{x}]$.

```
(1)   PROGRAM KRUSKAL;
(2)   BEGIN
(3)      SORTARE (G);
(4)      A' := ∅;
(5)      FOR i := 1 TO m DO
(6)         IF a_i nu formează ciclu cu A';
(7)            THEN A' := A' ∪ {a_i};
(8)      END.
```

```
(1)   PROCEDURA SORTARE (G);
(2)   BEGIN
(3)      se ordonează muchiile din A astfel încât A = {a_1, ..., a_m}
            cu b(a_1) ≤ ... ≤ b(a_m);
(4)   END;
```

Teorema 3.11. *Algoritmul Kruskal determină un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$.*

Demonstraţie. Algoritmul Kruskal s-a obţinut folosind procedura SORTARE şi condiţia din linia (6) pentru selectările din linia (9) şi (10) din algoritmul generic. Deci, conform Teoremei 3.8 algoritmul Kruskal este corect.■

Observaţia 3.4. Complexitatea algoritmului Kruskal depinde de modul de implementare a procedurii din linia (3) şi a condiţiei din linia (6). Astfel complexitatea poate fi $O(max(m \log n,\ n^2))$ sau $O(m \log n)$ etc.

Exemplul 3.5. Fie reţeaua $G = (N, A, b)$ reprezentată în figura 3.5(a). Să se determine un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$ cu algoritmul Kruskal.

Procedura SORTARE ordonează muchiile din A şi se obţine $A = \{[2,4], [3,5], [3,4], [2,3], [4,5], [1,2], [1,3]\}$.
Iteraţia 1: $A' = \{[2,4]\}$
Iteraţia 2: $A' = \{[2,4], [3,5]\}$
Iteraţia 3: $A' = \{[2,4], [3,5], [3,4]\}$
Iteraţia 4: $[2,3]$ formează ciclu cu A'
Iteraţia 5: $[4,5]$ formează ciclu cu A'
Iteraţia 6: $A' = \{[2,4], [3,5], [3,4], [1,2]\}$
Iteraţia 7: $[1,3]$ formează ciclu cu A'.
Arborela parţial minim $G' = (N, A')$ obţinut este prezentat în figura 3.5(b).

3.2.5 Algoritmul Boruvka

Fie $G = (N, A, b)$ o reţea neorientată şi conexă cu mulţimea nodurilor $N = \{1, \ldots, n\}$ şi mulţimea muchiilor $A = \{1, \ldots, a, \ldots, m\},\ a = [x, \overline{x}]$. Funcţia valoare $b : A \to \Re$ are toate valorile muchie diferite.

```
(1)   PROGRAM BORUVKA;
(2)   BEGIN
(3)      FOR i := 1 TO n DO
(4)         N_i := {i};
(5)      A' := ∅; M := {N_1, ..., N_n};
(6)      WHILE |A'| < n − 1 DO
(7)      BEGIN
(8)         FOR N_i ∈ M DO
(9)         BEGIN
(10)              se selectează [y, ȳ] cu b[y, ȳ] := min{b[x, x̄] | x ∈ N_i, x̄ ∉ N_i};
```

(11) se determină indicele j pentru care $\overline{y} \in N_j$;
(12) $A' := A' \cup \{[y, \overline{y}]\}$;
(13) END;
(14) FOR $N_i \in M$ DO BEGIN $N_i := N_i \cup N_j$; $N_j := N_i$; END;
(15) $M := \{\ldots, N_i, \ldots, N_j, \ldots | N_i \cap N_j = \emptyset, \cup N_i = N\}$;
(16) END;
(17) END.

Mulţimile N_j din operaţia $N_i := N_i \cup N_j$ executată în linia (14) sunt cele determinate în linia (11).

Teorema 3.12. *Algoritmul Boruvka determină un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$.*

Demonstraţie. Este evident că algoritmul Boruvka este o variantă a algoritmului generic. Ipoteza că funcţia valoare b are toate valorile muchie diferite asigură că nu se creează cicluri în $G' = (N, A')$ în timpul execuţiei ciclului WHILE. Deci, conform Teoremei 3.8 algoritmul Boruvka este corect.■

Teorema 3.13. *Algoritmul Boruvka are complexitatea $O(m \log n)$.*

Demonstraţie. Deoarece numărul componentelor conexe N_i din M este cel puţin înjumătăţit la fiecare iteraţie, ciclul WHILE este executat de cel mult $\log n$ ori. Operaţiile din liniile (10), (11) au complexitatea $O(m)$. Deci algoritmul Boruvka are complexitatea $O(m \log n)$.■

Observaţia 3.5. Algoritmul Boruvka este cunoscut în literatura de specialitate şi sub numele de algoritmul Sollin.

Exemplul 3.6. Fie reţeaua $G = (N, A, b)$ reprezentată în figura 3.5(a). Să se determine un arbore parţial minim $G' = (N, A')$ al grafului $G = (N, A)$ cu algoritmul Boruvka.

Iniţial $A' = \emptyset,\ M = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$.
Iteraţia 1.
$N_1,\ [y, \overline{y}] = [1, 2],\ N_j = N_2,\ A' = \{[1, 2]\}$;
$N_2,\ [y, \overline{y}] = [2, 4],\ N_j = N_4,\ A' = \{[1, 2], [2, 4]\}$;
$N_3,\ [y, \overline{y}] = [3, 5],\ N_j = N_5,\ A' = \{[1, 2], [2, 4], [3, 5]\}$;
$N_4,\ [y, \overline{y}] = [4, 2],\ N_j = N_2,\ A' = \{[1, 2], [2, 4], [3, 5]\}$;
$N_5,\ [y, \overline{y}] = [5, 3],\ N_j = N_3,\ A' = \{[1, 2], [2, 4], [3, 5]\}$;
$M = \{N_4 = \{1, 2, 4\},\ N_5 = \{3, 5\}\}$;
Iteraţia 2.
$N_4,\ [y, \overline{y}] = [4, 3],\ N_j = N_5,\ A' = \{[1, 2], [2, 4], [3, 5], [4, 3]\}$;
$N_5,\ [y, \overline{y}] = [3, 4],\ N_j = N_4,\ A' = \{[1, 2], [2, 4], [3, 5], [4, 3]\}$;
$M = \{N_5 = \{1, 2, 3, 4, 5\}\}$.

Arborele parţial minim $G' = (N, A')$ obţinut este prezentat în figura 3.5(b).

3.3 Arborescenţe

Definiţia 3.5. Un nod $r \in N$ al unui digraf $G = (N, A)$ se numeşte *nod rădăcină* dacă pentru orice alt nod $x \in N$ există un drum de la r la x.

Observaţia 3.6. Noţiunea de nod rădăcină are sens numai pentru grafuri orientate. Un nod rădăcină nu există întotdeauna.
Exemplul 3.7. Nodul 1 al digrafului reprezentat în figura 3.6 este un nod rădăcină.

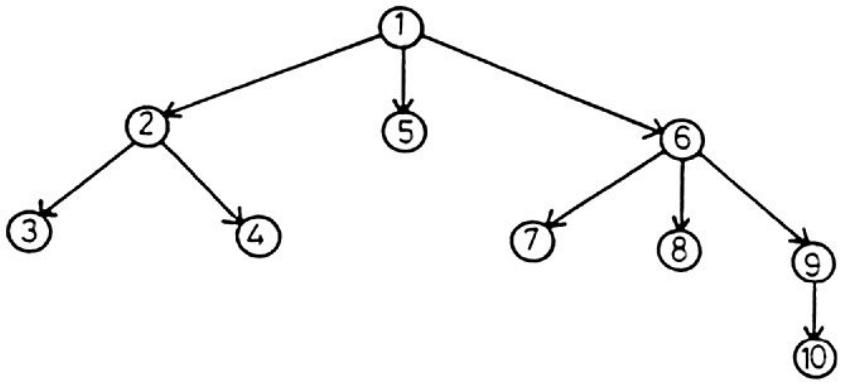


Fig.3.6

Definiţia 3.6. Un digraf $G = (N, A)$ se numeşte *quasi-tare conex* dacă pentru oricare două noduri $x, y \in N$ există un nod $z \in N$ (care poate coincide cu x sau cu y) de la care pleacă un drum care ajunge în x şi un drum care ajunge în y.
Observaţia 3.7. Un digraf tare conex este quasi tare conex, dar reciproca nu este adevărată. Un digraf quasi tare conex este conex, dar reciproca nu este adevărată.
Definiţia 3.7. Un digraf $G' = (N, A')$ se numeşte *arborescenţă* dacă este fără cicluri şi quasi tare conex.
Exemplul 3.8. Digraful reprezentat în figura 3.6 este o arborescenţă.
Teorema 3.14. *Un digraf $G = (N, A)$ admite un nod rădăcină dacă şi numai dacă el este quasi tare conex.*
Demonstraţie. Dacă digraful G admite un nod rădăcină atunci el este quasi tare conex deoarece pentru oricare două noduri $x, y \in N$ există un nod z (de exemplu nodul rădăcină r) de la care pleacă un drum care ajunge în x şi un drum care ajunge în y.

Dacă digraful $G = (N, A)$, $|N| = n$ este quasi tare conex, atunci există un nod z_1 de la care pleacă un drum care ajunge în x_1 şi un drum care ajunge în x_2; există un nod z_2 de la care pleacă un drum care ajunge în z_1 şi un drum care ajunge în x_3. Continuând acest procedeu există un nod z_{n-1} de la care pleacă un drum care ajunge în z_{n-2} şi un drum care ajunge în x_n. Este evident că nodul $r = z_{n-1}$ este nod rădăcină al digrafului $G = (N, A)$. ■
Teorema 3.15. *Fie $G' = (N, A')$ un digraf de ordinul $n \geq 2$. Proprietăţile următoare sunt echivalente şi caracterizează o arborescenţă:*
(1). G' este fără cicluri şi quasi tare conex;
(2). G' este quasi tare conex şi are $n - 1$ arce;
(3). G' este un arbore şi admite un nod rădăcină r;
(4). G' conţine un nod r şi un drum unic de la nodul r la oricare alt nod $x \in N$;
(5). G' este quasi tare conex şi prin eliminarea unui arc oarecare se obţine digraful $\widehat{G}' = \left(N, \widehat{A}'\right)$ care nu este quasi tare conex;

(6). G' este conex şi conţine un nod r astfel încât $\rho^-(r) = 0,\ \rho^-(x) = 1$ pentru oricare nod $x \neq r$;
(7). G' este fără cicluri şi conţine un nod r astfel încât $\rho^-(r) = 0,\ \rho^-(x) = 1$ pentru oricare nod $x \neq r$.
Demonstraţie.
(1) $\Rightarrow$ (2). Dacă (1) este adevărată, atunci G' este conex şi fără cicluri, adică G' este un arbore. Deci G' are $n-1$ arce şi (2) este adevărată.
(2) $\Rightarrow$ (3). Dacă (2) este adevărată, atunci G' este conex şi are $n-1$ arce, adică G' este un arbore. În plus, conform Teoremei 3.14 G' admite un nod rădăcină r şi (3) este adevărată.
(3) $\Rightarrow$ (4). Dacă (3) este adevărată, atunci există un drum de la nodul r la oricare alt nod $x \in N$. Deoarece G' este arbore rezultă că acest drum este unic.
(4) $\Rightarrow$ (5). Dacă (4) este adevărată, atunci nodul r este un nod rădăcină şi conform Teoremei 3.14 G' este quasi tare conex. Fie un arc oarecare $(x, y) \in A'$. Construim digraful $\widehat{G}' = (N, \widehat{A}')$, unde $\widehat{A}' = A' - \{(x, y\})$. Prin reducere la absurd, presupunem că digraful $\widehat{G}'$ este quasi tare conex. Deci în $\widehat{G}'$ există un drum D_1 de la un nod z la nodul x şi un drum D_2 de la nodul z la nodul y. Rezultă că în G' există două drumuri de la z la $y(D_2$ şi $D_3 = D_1 \cup \{(x, y)\})$. Prin urmare există două drumuri de la r la y ce contrazice (4). Deci $\widehat{G}'$ nu este quasi tare conex şi (5) este adevărată.
(5) $\Rightarrow$ (6). Dacă (5) este adevărată, atunci G' este conex şi conform Teoremei 3.14, admite un nod rădăcină r. Deci $\rho^-(r) \geq 0$ şi $\rho^-(x) \geq 1$ pentru oricare nod $x \neq r$. Dacă $\rho^-(r) > 0$, atunci există cel puţin un arc $(y, r) \in A'$. Digraful $\widehat{G}' = (N, \widehat{A}')$ cu $\widehat{A}' = A' - \{(y, r)\}$, admite evident nodul r ca nod rădăcină, adică $\widehat{G}'$ este quasi tare conex ce contrazice (5) şi deci $\rho^-(r) = 0$. Dacă $\rho^-(x) > 1$, atunci există cel puţin două arce $(y, x), (z, x)$ în A'. Aceasta înseamnă că există cel puţin două drumuri distincte de la r la x. Digraful $\widehat{G}' = (N, \widehat{A}')$ cu $\widehat{A}' = A' - \{(y, x)\}$ admite nodul r ca nod rădăcină, adică $\widehat{G}'$ este quasi tare conex ce contrazice (5) şi deci $\rho^-(x) = 1$. Rezultă că (6) este adevărată.
(6) $\Rightarrow$ (7). Dacă (6) este adevărată, atunci numărul de arce al digrafului $G' = (N, A')$ este $m' = \sum_N \rho^-(x) = n - 1$, adică G' este un arbore. Rezultă că G' este fără cicluri şi (7) este adevărată.
(7) $\Rightarrow$ (1). Dacă (7) este adevărată, atunci nodul r este evident un nod rădăcină. Conform Teoremei 3.14 digraful G' este quasi tare conex şi (1) este adevărată.■
Teorema 3.16. *Un digraf $G = (N, A)$ admite un subgraf parţial $G' = (N, A')$ care este o arborescenţă dacă şi numai dacă el este quasi tare conex.*
Demonstraţie. Dacă digraful G admite un subgraf parţial G' care este o arborescenţă atunci el admite un nod rădăcină r şi conform Teoremei 3.14 digraful G este quasi tare conex.
Reciproc, dacă digraful G este quasi tare conex, atunci conform proprietăţii (5) din Teorema 3.15 se poate obţine un subgraf parţial $G' = (N, A')$ care este o arborescenţă cu procedura următoare:

```
(1)  PROCEDURA ARBORESCENTA;
(2)  BEGIN
(3)     A' := A;
(4)     FOR i := 1 TO m DO
(5)        IF prin eliminarea arcului a_i ∈ A' noul digraf rămâne quasi
              tare conex
(6)           THEN A' := A' − {a_i}
(7)  END.
```

3.4 Aplicaţii

Problema arborelui parţial minim apare în aplicaţii în două moduri: direct şi indirect. În aplicaţiile directe se doreşte conectarea unor puncte prin legături care au asociate lungimi sau costuri. Punctele reprezintă entităţi fizice ca utilizatori ai unui sistem sau componente ale unui cip care trebuie conectate între ele sau cu un alt punct ca procesorul principal al unui calculator. În aplicaţiile indirecte problema practică nu apare evident ca o problemă a arborelui parţial minim şi în acest caz este necesar să modelăm problema practică astfel încât să o transformăm într-o problemă a arborelui parţial minim. În continuare se vor prezenta câteva aplicaţii directe şi indirecte ale arborelui parţial minim.

3.4.1 Proiectarea unui sistem fizic

Proiectarea unui sistem fizic constă în proiectarea unei reţele care conectează anumite componente. Sistemul fizic trebuie să nu aibă redundanţe ceea ce conduce la determinarea unui arbore parţial minim. Se prezintă în continuare câteva exemple.
1. Conectarea terminalelor din tablourile electrice astfel încât lungimea totală a firelor folosite să fie minimă.
2. Construirea unei reţele de conducte care să lege un număr de oraşe astfel încât lungimea totală a conductelor să fie minimă.
3. Conectarea telefonică a unor comune dintr-o zonă izolată astfel încât lungimea totală a liniilor telefonice care leagă oricare două comune să fie minimă.
4. Determinarea configuraţiei unei reţele de calculatoare astfel încât costul conec-tării în reţea să fie minim.

Fiecare dintre aceste aplicaţii este o aplicaţie directă a problemei arborelui parţial minim. În continuare se prezintă două aplicaţii indirecte.

3.4.2 Transmiterea optimă a mesajelor

Un serviciu de informaţii are n agenţi într-o anumită ţară. Fiecare agent cunoaşte o parte din ceilalţi agenţi şi poate stabili întâlniri cu oricare din cei pe care îi cunoaşte. Orice mesaj transmis la întâlnirea dintre agentul i şi agentul j poate să ajungă la inamic cu probabilitatea $p(i, j)$. Conducătorul grupului vrea să transmită un mesaj confidenţial tuturor agenţilor astfel încât probabilitatea totală ca mesajul să fie interceptat să fie minimă.

Dacă reprezentăm agenţii prin noduri şi fiecare întâlnire posibilă dintre doi agenţi printr-o muchie, atunci în graful rezultat $G = (N, A)$ dorim să identificăm un arbore parţial $G' = (N, A')$ care minimează probabilitatea interceptării dată de expresia $\left(1 - \prod_{A'}(1 - p(i,j))\right)$. Deci vrem să determinăm un arbore parţial $G' = (N, A')$ care maximizează $\left(\prod_{A'}(1 - p(i,j))\right)$. Putem determina un astfel de arbore dacă definim lungimea arcului (i, j) ca fiind $\log(1 - p(i,j))$ şi rezolvând problema arborelui parţial maxim.

3.4.3 Problema lanţului minimax între oricare două noduri

Într-o reţea $G = (N, A, b)$, definim valoarea unui lanţ $L_{z\overline{z}}$ de la nodul z la nodul $\overline{z}$ ca fiind valoarea maximă a muchiilor din $L_{z\overline{z}}$. Problema lanţului minimax între oricare două noduri constă în a determina între oricare două noduri $z, \overline{z}$ un lanţ de valoare minimă.

Problema lanţului minimax apare în multe situaţii. De exemplu, considerăm o navă spaţială care trebuie să intre în atmosfera Pământului. Nava trebuie să treacă prin zone cu temperaturi diferite pe care le putem reprezenta prin muchii într-o reţea. Pentru a ajunge pe suprafaţa Pământului trebuie aleasă o traiectorie astfel încât temperatura maximă la care urmează să fie expusă nava să fie minimă.

Problema lanţului minimax între oricare două noduri din reţeaua $G = (N, A, b)$ este o problemă a arborelui parţial minim. Fie $G' = (N, A')$ un arbore parţial minim al grafului $G = (N, A)$ şi $L'_{z\overline{z}}$ lanţul unic de la z la $\overline{z}$ în G'. Dacă $[x, \overline{x}]$ este muchia din $L'_{z\overline{z}}$ cu valoarea maximă atunci lanţul $L'_{z\overline{z}}$ are valoarea $b[x, \overline{x}]$. Muchia $a = [x, \overline{x}]$ generează tăietura $\left[X, \overline{X}\right]_a$ şi conform condiţiilor de optimalitate ale tăieturii avem

$$b[x, \overline{x}] \leq b[y, \overline{y}] \text{ pentru toate muchiile } [y, \overline{y}] \in \left[X, \overline{X}\right]_a$$

Oricare alt lanţ $L_{z\overline{z}}$ de la z la $\overline{z}$ conţine o muchie $[y, \overline{y}] \in \left[X, \overline{X}\right]_a$ şi valoarea acestui lanţ este, conform condiţiilor de mai sus, cel puţin $b[x, \overline{x}]$. Rezultă că $L'_{z\overline{z}}$ este un lanţ de valoare minimă de la z la $\overline{z}$. Deci laţul unic dintre oricare două noduri din G' este un lanţ de valoare minimă dintre oricare două noduri din G.

În continuare se prezintă două aplicaţii ale arborilor binari.

Capitolul 4

Distanţe şi drumuri minime

4.1 Principalele probleme de drum minim

Se consideră un digraf $G = (N, A)$ cu $N = \{1, \ldots, n\}$, $A = \{a_1, \ldots, a_m\}$ şi o funcţie valoare $b : A \to \Re$ care asociază fiecărui arc $a \in A$ un număr real $b(a)$ numit *valoarea arcului a*. Această valoare poate fi interpretată ca lungime sau cost etc.

Vom nota cu D_{xyk} un drum în digraful G de la nodul x la nodul y şi cu $\mathcal{D}_{xy} = \{D_{xy1}, \ldots, D_{xyr}\}$ mulţimea acestor drumuri.

Valoarea unui drum $D_{xyk} = (a_1, \ldots, a_q)$, notată $b(D_{xyk})$, este numărul $b(D_{xyk}) = b(a_1) + \ldots b(a_q)$.

Definiţia 4.1. Un drum D_{xyp} cu valoarea $b(D_{xyp}) = \min\{b(D_{xyk}) \mid D_{xyk} \in \mathcal{D}_{xy}\}$ se numeşte *drum de valoare minimă* sau *drum minim*. Valoarea $b(D_{xyp})$ a drumului minim D_{xyp} se numeşte *distanţă* de la nodul x la nodul y şi o notăm $d(x, y)$.

Există două probleme cu dificultăţi privind această definiţie. Prima, poate să nu existe drum de la nodul x la nodul y şi a doua, poate exista drum D_{xyk} cu valoare arbitrar de mică. Prima problemă poate fi rezolvată dacă se defineşte $d(x, y) = \infty$. A doua problemă provine din posibilitatea existenţei circuitelor de valoare negativă în reţeaua $G = (N, A, b)$.

Exemplul 4.1. În reţeaua reprezentată în figura 4.1, putem determina un drum de valoare arbitrar de mică de la nodul 1 la nodul 5 utilizând circuitul de valoare negativă (2, 3, 4, 2), ori de câte ori este necesar.

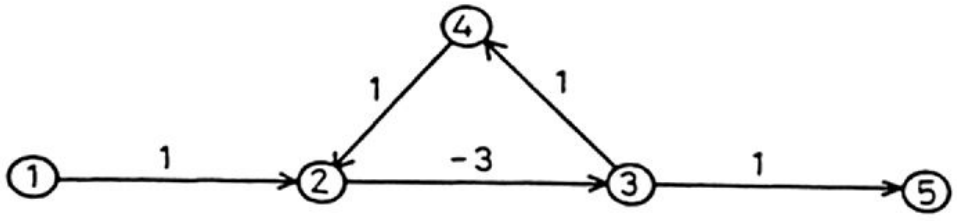


Fig.4.1

Vom presupune că reţeaua $G = (N, A, b)$ nu conţine circuite cu valoare negativă.

Principalele probleme de drum minim care apar în aplicaţii practice sau sunt subprobleme în rezolvarea altor probleme de optimizare în reţele sunt următoarele:
(PDM1) Determinarea unui drum minim D_{stp} de la un nod precizat s la un alt nod precizat t şi a valorii $b(D_{stp})$.
(PDM2) Determinarea unui drum minim D_{syp} de la un nod precizat s la nodul y şi a valorii $b(D_{syp})$, pentru toate nodurile $y \in N,\ y \neq s$.
(PDM3) Determinarea unui drum minim D_{xyp} de la un nod x la nodul y şi a valorii $b(D_{xyp})$, pentru toate nodurile $x, y \in N,\ x \neq y$.

Dacă $G' = (N', A')$ este un graf neorientat şi $b' : A' \to \Re^+$, atunci problemele de lanţ minim din reţeaua neorientată $G' = (N', A')$ pot fi reduse la problemele de drum minim din reţeaua orientată $G = (N, A, b)$ cu $N = N', A = \{(x, y),\ (y, x) \mid [x, y] \in A'\}$ şi $b(x, y) = b(y, x) = b'[x, y]$.

PDM1 se poate rezolva utilizând un algoritm de rezolvare a PDM2 la care se adaugă un test suplimentar de oprire; PDM3 se poate rezolva iterând după s un algoritm de rezolvare a PDM2. Există însă algoritmi eficienţi care rezolvă direct PDM3. De aceea în acest capitol se vor prezenta algoritmi de rezolvare pentru PDM2 şi PDM3.
Observaţia 4.1. Dacă $d(x, y) = \infty$ atunci considerăm că există un drum D_{xyk} cu $b(D_{xyk}) = \infty$.

4.2 Ecuaţiile lui Bellman

Fie reţeaua orientată $G = (N, A, b)$ şi presupunem că G nu conţine circuite cu valoare negativă. Pentru reprezentarea reţelei G se va folosi matricea valoare adiacenţă $B = (b_{ij})$ cu $i, j \in N$, unde

$$b_{ij} = \begin{cases} b(i,j) & \text{dacă} \quad i \neq j \text{ şi } (i,j) \in A; \\ 0 & \text{dacă} \quad i = j; \\ \infty & \text{dacă} \quad i \neq j \text{ şi } (i,j) \notin A. \end{cases}$$

Fără a restrânge generalitatea presupunem că nodul s este nodul 1. Dacă $D_{1jp} = ((1,h), \ldots, (k,i), (i,j))$ este un drum minim de la nodul 1 la nodul j, atunci $D_{1ip} = ((1,h), \ldots, (k,i))$ este un drum minim de la nodul 1 la nodul i, altfel contrazicem faptul că D_{1jp} este drum minim. Astfel distanţele $u_j = d(1, j)$ trebuie să satisfacă ecuaţiile lui Bellman:

$$u_1 = 0,\ u_j = \min\{u_i + b_{ij} \mid i \neq j\},\ j = 2, \ldots, n. \tag{4.1}$$

Fără a restrânge generalitatea, putem presupune că nodul 1 este un nod rădăcină al reţelei orientate G.
Teorema 4.1. *Dacă în reţeaua orientată $G = (N, A, b)$ nodul 1 este nod rădăcină şi G conţine numai circuite $\overset{\circ}{D}$ cu $b\left(\overset{\circ}{D}\right) > 0$ atunci G conţine o arborescenţă parţială $G' = (N, A')$ cu nodul rădăcină 1 şi drumul unic de la 1 la oricare alt nod j din G' este un drum minim D_{1jp} din G.*
Demonstraţie. Dacă în reţeaua orientată $G = (N, A, b)$ nodul 1 este nod rădăcină atunci conform Teoremei 3.14 şi Teoremei 3.16, G admite o arborescenţă parţială

$G' = (N, A')$ cu nodul rădăcină 1. Considerăm un nod j şi determinăm un drum $D_{1jp},\ u_j = b(D_{1jp})$, cu următorul procedeu. Deoarece sunt verificate ecuaţiile (4.1), selectăm un arc (i, j) astfel încât $u_j = u_i + b_{ij}$. În continuare selectăm un arc (k, i) astfel încât $u_i = u_k + b_{ki}$ etc. până când se ajunge la nodul 1. Presupunem prin reducere la absurd că nu se ajunge în nodul 1. Aceasta este posibil dacă procedeul conduce la un circuit $\overset{\circ}{D} = (v, w, \ldots, q, v)$. Avem următoarele ecuaţii:$u_v = u_q + b_{qv} = \ldots = u_v + b_{vw} + \ldots + b_{qv}$. Rezultă că $b(\overset{\circ}{D}) = b_{vw} + \ldots + b_{qv} = u_v - u_v = 0$, care contrazice ipoteza că reţeaua G conţine circuite $\overset{\circ}{D}$ cu $b\left(\overset{\circ}{D}\right) > 0$. Deci procedeul determină un drum D_{1jp} cu $u_j = b(D_{1jp})$. Astfel, aplicând procedeul pentru toate nodurile j, pentru care nu s-a determinat deja un drum D_{1jp}, se obţine arborescenţa parţială $G' = (N, A')$ cu proprietatea din enunţul teoremei. ■

Următoarea teoremă întăreşte rezultatul din Teorema 4.1.

Teorema 4.2. *Dacă în reţeaua orientată $G = (N, A, b)$ nodul 1 este nod rădăcină şi G nu conţine circuite $\overset{\circ}{D}$ cu $b\left(\overset{\circ}{D}\right) < 0$, atunci G conţine o arborescenţă parţială $G' = (N, A')$ cu nodul rădăcină 1 şi drumul unic de la 1 la oricare alt nod j din G' este un drum minim D_{1jp} din G.*

Demonstraţie. Dacă în reţeaua orientată $G = (N, A, b)$ nodul 1 este nod rădăcină, atunci conform Teoremei 3.14 şi Teoremei 3.16, G admite o arborescenţă parţială $G' = (N, A')$ cu nodul rădăcină 1. Considerăm un nod j şi un drum minim $D_{1jp} = ((1, h), \ldots, (k, i), (i, j))$. Atunci avem:

$$d(1, j) = d(1, i) + b(i, j).$$

Deoarece nodul j a fost selectat arbitrar, putem să selectăm un arc (i, j) care verifică condiţia anterioară pentru orice nod $j \neq 1$. În acest mod putem alege $n-1$ arce şi fie A' mulţimea acestor arce. Este evident că în digraful $G' = (N,\ A')$ sunt verificate relaţiile $\rho^{-1} = 0,\ \rho^-(j) = 1,\ \ j = 2, \ldots, n$. Rezultă că $G' = (N, A')$ este o arborescenţă parţială cu nodul rădăcină 1. Drumul unic de la 1 la j din G' este un drum minim D_{1jp} din G, deoarece fiecare arc din A' verifică condiţia de mai sus. ■

4.3 Algoritmi pentru distanţe şi drumuri minime

4.3.1 Algoritmul Dijkstra

Fie reţeaua orientată $G = (N, A, b)$ cu $b : A \to \Re^+$. În acest caz ecuaţiile lui Bellman (4.1) pot fi rezolvate cu algoritmul Dijkstra. Lista nodurilor adiacente către exterior nodului x este $V^+(x) = \{y \mid (x, y) \in A\}$. Algoritmul Dijkstra este următorul:

```
(1)   PROGRAM DIJKSTRA;
(2)   BEGIN
(3)     W := N; d(s) := 0; p(s) := 0;
(4)     FOR y ∈ N − {s} DO
(5)     BEGIN
(6)       d(y) := ∞; p(y) := 0;
(7)     END;
```

```
(8)     WHILE W ≠ ∅ DO
(9)     BEGIN
(10)        se selectează un nod x ∈ W astfel încât d(x) este minimă;
(11)        W := W − {x};
(12)        FOR y ∈ W ∩ V⁺(x) DO
(13)            IF d(x) + b(x, y) < d(y)
(14)                THEN BEGIN d(y) := d(x) + b(x, y); p(y) := x; END;
(15)     END;
(16) END.
```

Teorema 4.3. *Algoritmul Dijkstra determină distanţele $d(s,y)$ şi drumurile minime D_{syp}, $y \in N$, $y \neq s$, în raport cu nodul sursă s din reţeaua orientată $G = (N, A, b)$ cu $b : A \to \Re^+$.*

Demonstraţie. Din modul cum este calculată în algoritm valoarea $d(y)$ rezultă că $d(y)$ reprezintă valoarea unui drum de la nodul s la nodul y din reţeaua G. Deci are loc relaţia $d(s,y) \leq d(y)$ pentru fiecare nod $y \in N$. Trebuie să arătăm că $d(s,y) = d(y)$ pentru fiecare $y \in N$. Pentru aceasta vom demonstra prin inducţie după ordinea în care nodurile sunt eliminate din W că $d(y) \leq d(s,y)$ pentru fiecare nod $y \in N$.

Primul nod eliminat este s şi evident că $d(s) = 0 = d(s,s)$. Presupunem că $d(y) \leq d(s,y)$ pentru toate nodurile y care au fost eliminate din W înaintea nodului z. Fie $D_{szp} = (x_0, x_1, \ldots, x_{k-1}, x_k)$ un drum minim de la $x_0 = s$ la $x_k = z$. Atunci pentru $h = 1, \ldots, k$ avem

$$d(s, x_h) = \sum_{i=1}^{h} b(x_{i-1}, x_i)$$

Fie j indicele maxim astfel încât x_j a fost eliminat din W înaintea lui z. Deoarece $x_{j+1} \in W$, $x_{j+1} \in V^+(x_j)$ rezultă că după eliminarea lui x_j din W avem $d(x_{j+1}) \leq d(x_j) + b(x_j, x_{j+1})$. Această inegalitate se păstrează şi când este eliminat nodul z din W, deoarece $d(x_j)$ rămâne nemodificată ($x_j \notin W$) şi $d(x_{j+1})$ poate numai să descrească. De asemenea, din ipoteza inducţiei avem $d(x_j) = d(s, x_j)$. Astfel

$$d(x_{j+1}) \leq d(x_j) + b(x_j, x_{j+1}) = d(s, x_j) + b(x_j, x_{j+1}) = d(s, x_{j+1}) \leq d(s, z) \quad (4.2)$$

Pot exista următoarele două cazuri:
(c_1) $j = k - 1$. În acest caz $x_{j+1} = x_k = z$. Din relaţia (4.2) rezultă $d(z) \leq d(s,z)$;
(c_2) $j < k - 1$. În acest caz $x_{j+1} \neq x_k = z$. Dacă $d(s,z) < d(z)$, atunci din relaţia (4.2) rezultă că $d(x_{j+1}) < d(z)$. Conform liniei (10) din algoritm deducem că x_{j+1} este eliminat din W înaintea lui z. Aceasta contrazice modul de alegere a indicelui j. Astfel şi în acest caz avem $d(z) \leq d(s,z)$.

Am demonstrat că $d(y) = d(s,y)$ pentru fiecare $y \in N$, $y \neq s$. Orice drum minim D_{syp} de la nodul s la nodul y se determină cu elementele tabloului predecesor p. ■

Teorema 4.4. *Algoritmul Dijkstra are complexitatea $O(n^2)$.*

Demonstraţie. Pentru selectarea nodului $x \in W$ astfel încât $d(x)$ este minimă sunt necesare cel mult $n-1$ comparaţii. Ciclul FOR se execută de cel mult $n-1$ ori. Ciclul WHILE se execută de n ori. Deci algoritmul are complexitatea $O(n^2)$. ■

Observaţia 4.2. Algoritmul Dijkstra poate să conducă la rezultate eronate dacă reţeaua $G = (N, A, b)$ conţine şi arce cu valoare negativă, chiar dacă nu conţine circuite cu valoare negativă. În acest caz estimarea din relaţia (4.2) nu mai este valabilă întotdeauna. De exemplu, dacă se aplică algoritmul Dijkstra reţelei reprezentate în figura 4.2 se determină drumul minim $D_{13p} = (1, 3)$ cu $b(D_{13p}) = 1$.

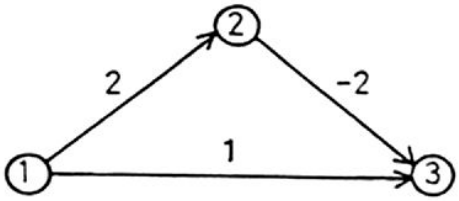


Fig.4.2

În realitate $D_{13p} = (1, 2, 3)$ cu $b(D_{13p}) = 0$.

Algoritmul lui Dijkstra poate fi implementat în mai multe moduri în funcţie de structurile de date utilizate. Astfel, fiecare implementare poate avea o altă complexitate. Aceste aspecte vor fi prezentate în ultimul paragraf al acestui capitol.

Exemplul 4.2. Se consideră reţeaua reprezentată în figura 4.3. Să se aplice algoritmul Dijkstra acestei retele.

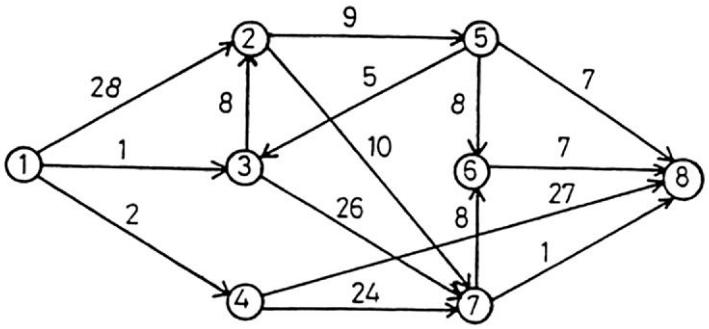


Fig.4.3

Iniţializări: $W = \{1, 2, 3, 4, 5, 6, 7, 8\},\ d = (0, \infty, \infty, \infty, \infty, \infty, \infty, \infty),\ p = (0, 0, 0, 0, 0, 0, 0, 0)$;
Iteraţia 1: $x = 1, W = \{2, 3, 4, 5, 6, 7, 8\},\ d = (0, 28, 1, 2, \infty, \infty, \infty, \infty),\ p = (0, 1, 1, 1, 0, 0, 0, 0)$;
Iteraţia 2: $x = 3, W = \{2, 4, 5, 6, 7, 8\},\ d = (0, 9, 1, 2, \infty, \infty, 27, \infty),\ p = (0, 3, 1, 1, 0, 0, 3, 0)$;
Iteraţia 3: $x = 4, W = \{2, 5, 6, 7, 8\},\ d = (0, 9, 1, 2, \infty, \infty, 26, 29),\ p = (0, 3, 1, 1, 0, 0, 4, 4)$;
Iteraţia 4: $x = 2, W = \{5, 6, 7, 8\},\ d = (0, 9, 1, 2, 18, \infty, 19, 29),\ p = (0, 3, 1, 1, 2, 0, 2, 4)$;
Iteraţia 5: $x = 5, W = \{6, 7, 8\},\ d = (0, 9, 1, 2, 18, 26, 19, 25),\ p = (0, 3, 1, 1, 2, 5, 2, 5)$;
Iteraţia 6: $x = 7, W = \{6, 8\},\ d = (0, 9, 1, 2, 18, 26, 19, 20),\ p = (0, 3, 1, 1, 2, 5, 2, 7)$;
Iteraţia 7: $x = 8, W = \{6\},\ d = (0, 9, 1, 2, 18, 26, 19, 20),\ p = (0, 3, 1, 1, 2, 5, 2, 7)$;
Iteraţia 8: $x = 6, W = \emptyset, d = (0, 9, 1, 2, 18, 26, 19, 20),\ p = (0, 3, 1, 1, 2, 5, 2, 7)$.

Să determinăm D_{18p}. Avem $p(8) = 7, p(7) = 2, p(2) = 3, p(3) = 1$ şi $D_{18p} = (1, 3, 2, 7, 8)$ cu $b(D_{18p}) = d(1, 8) = d(8) = 20$.

Arborescenţa parţială $G' = (N, A')$ este reprezentată în figura 4.4.

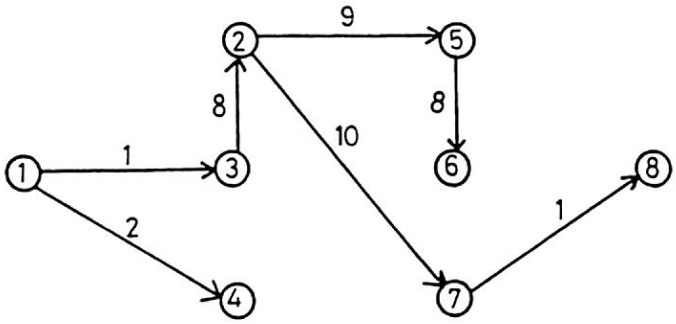


Fig.4.4

4.3.2 Algoritmul Bellman-Ford

Fie reţeaua orientată $G = (N, A, b)$ cu $b : A \to \Re$ şi G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$. În acest caz ecuaţiile lui Bellman (4.1.) pot fi rezolvate cu algoritmul Bellman-Ford. Lista nodurilor adiacente către interior nodului y este $V^-(y) = \{x \mid (x, y) \in A\}$. Algoritmul Bellman-Ford este următorul:

```
(1)   PROGRAM BELLMAN-FORD;
(2)   BEGIN
(3)     d(s) := 0; p(s) := 0;
(4)     FOR y ∈ N − {s} DO
(5)     BEGIN
(6)       d(y) := ∞; p(y) := 0;
(7)     END;
(8)     REPEAT
(9)       FOR y ∈ N DO
(10)        d'(y) := d(y);
(11)      FOR y ∈ N DO
(12)        IF V⁻(y) ≠ ∅
(13)          THEN BEGIN
(14)                 se selectează x ∈ V⁻(y) astfel încât d'(x) + b(x, y)
                     este minimă;
(15)                 IF d'(x) + b(x, y) < d'(y)
(16)                   THEN BEGIN d(y) := d'(x) + b(x, y);
                                  p(y) := x; END;
(17)               END;
(18)    UNTIL d(y) = d'(y) pentru toate nodurile y ∈ N;
(19)  END.
```

Teorema 4.5. *Algoritmul Bellman-Ford determină distanţele $d(s,y)$ şi drumurile minime D_{syp}, $y \in N$, $y \neq s$, în raport cu nodul sursă s din reţeaua orientată $G = (N, A, b)$ cu $b : A \to \Re$ şi G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$.*

Demonstraţie. Mai întâi precizăm că liniile de la (12) la (17) sunt echivalente, referitor la valorile $d(y)$ cu

$$d(y) = \min\{d'(y), \min\{d'(x) + b(x,y) \mid x \in V^-(y)\}\}.$$

Dacă notăm cu $d_0(y)$ valorile definite în liniile (3), (6) şi prin $d_{k+1}(y)$ valorile calculate în timpul iteraţiei $k+1$ a ciclului REPEAT, atunci conform celor precizate mai sus avem

$$d_{k+1}(y) = \min\{d_k(y), \min\{d_k(x) + b(x,y) \mid x \in V^-(y)\}\}.$$

Prin $\ell(D_{syi})$ notăm numărul de arce ale drumului D_{syi} de la nodul s la nodul y şi prin $\mathcal{D}_{sy}$ mulţimea acestor drumuri. Vom arăta prin inducţie după k faptul că

$$d_k(y) = \min\{b\,(D_{syi}) \mid D_{syi} \in \mathcal{D}_{sy},\ \ \ell D_{syi} \leq k\}.$$

Pentru $k = 0$ afirmaţia este evident adevărată. Presupunem că afirmaţia este adevărată pentru k. Să arătăm că

$$d_{k+1}(y) = \min\{b\left(D'_{syi}\right) \mid D'_{syi} \in \mathcal{D}_{sy},\ \ \ell D_{syi} \leq k+1\}.$$

Avem $\min\{b\left(D'_{syi}\right) \mid D'_{syi} \in \mathcal{D}_{sy},\ \ \ell D_{syi} \leq k+1\} =$

$= \min\{\min\{b\left(D'_{syi}\right) \mid D'_{syi} \in \mathcal{D}_{sy},\ \ \ell D_{syi} \leq k\},$

$\min\{b\left(D'_{syi}\right) \mid D'_{syi} \in \mathcal{D}_{sy},\ \ \ell D_{syi} = k+1\}\} =$

$= \min\{d_k(y), \min\{d_k(x) + b(x,y) \mid x \in V^-(y)\}\} = d_{k+1}(y).$

Dacă G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$, atunci un drum minim de la s la y poate să conţină cel mult $n-1$ arce. Deci condiţia din linia (18) este satisfăcută după cel mult n iteraţii ale ciclului REPEAT şi $d(y) = d(s,y)$ pentru toţi $y \in N$, $y \neq s$. Un drum minim D_{syp} de la nodul s la nodul y se determină cu elementele tabloului predecesor p.■

Teorema 4.6. *Algoritmul Bellman-Ford are complexitatea $O(mn)$.*

Demonstraţie. În Teorema 4.5 am arătat că ciclul REPEAT se execută de cel mult n ori. O iteraţie a ciclului REPEAT are complexitatea $O(m)$. Deci algoritmul are complexitatea $O(mn)$.■

Observaţia 4.3. Dacă eliminăm liniile (9),(10),(18), înlocuim în linia (8) REPEAT prin FOR $k = 1$ TO n DO şi valorile $d'(x), d'(y)$ prin $d(x)$ respectiv $d(y)$, atunci se obţine posibilitatea testării existenţei unui circuit de valoare negativă în reţeaua $G = (N, A, b)$. Într-adevăr, dacă după execuţia algoritmului există un arc (x,y) astfel încât $d(x) + b(x,y) < d(y)$, atunci înseamnă că $d(x)$ descreşte nelimitat, lucru posibil numai dacă un drum D_{sxi} conţine un circuit $\overset{\circ}{D}$ cu $b(\overset{\circ}{D}) < 0$.

Exemplul 4.3. Se consideră reţeaua reprezentată în figura 4.5.

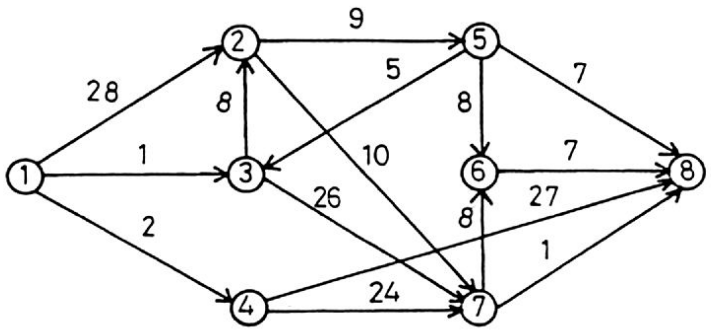


Fig.4.5

Avem $V^-(1) = \emptyset,\ V^-(2) = \{1,3\},\ V^-(3) = \{1,5\},\ V^-(4) = \{1\},\ V^-(5) = \{2\},\ V^-(6) = \{5,7\},\ V^-(7) = \{2,3,4\},\ V^-(8) = \{4,5,6,7\}$.
Iniţializări: $d = (0,\infty,\infty,\infty,\infty,\infty,\infty,\infty),\ p = (0,0,0,0,0,0,0,0)$.
Iteraţia 1: $d' = (0,\infty,\infty,\infty,\infty,\infty,\infty,\infty)$;
$y = 1 : d(1) = 0,\ p(1) = 0$;
$y = 2 : x = 1,\ d(2) = 28,\ p(2) = 1$;
$y = 3 : x = 1,\ d(3) = 1,\ p(3) = 1$;
$y = 4 : x = 1,\ d(4) = 2,\ p(4) = 1$;
$y = 5 : x = 2,\ d(5) = \infty,\ p(5) = 0$;
$y = 6 : x = 5,\ d(6) = \infty,\ p(6) = 0$;
$y = 7 : x = 2,\ d(7) = \infty,\ p(7) = 0$;
$y = 8 : x = 4,\ d(8) = \infty,\ p(8) = 0$;
S-a obţinut $d = (0,28,1,2,\infty,\infty,\infty,\infty),\ p = (0,1,1,1,0,0,0,0)$.
Iteraţia 2: $d' = (0,28,1,2,\infty,\infty,\infty,\infty)$,
$d = (0,9,1,2,37,\infty,26,29),\ p = (0,3,1,1,2,0,4,4)$.
Iteraţia 3: $d' = (0,9,1,2,37,\infty,26,29)$,
$d = (0,9,1,2,28,34,19,27),\ p = (0,3,1,1,2,7,2,7)$.
Iteraţia 4: $d' = (0,9,1,2,18,34,19,27)$,
$d = (0,9,1,2,18,26,19,20),\ p = (0,3,1,1,2,5,2,7)$.
Iteraţia 5: $d' = (0,9,1,2,18,26,19,20)$,
$d = (0,9,1,2,18,26,19,20),\ p = (0,3,1,1,2,5,2,7)$.

4.3.3 Algoritmul Floyd-Warshall

Fie reţeaua orientată $G = (N,A,b)$. Se pune problema rezolvării PDM3, adică determinarea distanţei $d(i,j)$ şi a unui drum minim D_{ijp} între oricare două noduri $i,j \in N, i \neq j$. Dacă $b : A \to \Re^+$, atunci PDM3 se poate rezolva iterând după $s \in N$ algoritmul Dijkstra şi se obţine un algoritm cu complexitatea $O(n^3)$. Dacă $b : A \to \Re$ şi G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$, atunci PDM3 se poate rezolva iterând după $s \in N$ algoritmul Bellman-Ford şi se obţine un algoritm cu complexitatea $O(mn^2)$. Algoritmul Floyd-Warshall rezolvă PDM3 în cazul $b : A \to \Re$ şi G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$. Acest algoritm are complexitatea $O(n^3)$. Evident că

algoritmul Floyd-Warshall este mai avantajos în rezolvarea PDM3 decât iterarea după nodul s a algoritmului Dijkstra sau a algoritmului Bellman-Ford.

Considerăm reţeaua orientată $G = (N, A, b)$ reprezentată prin matricea valoare adiacentă $B = (b_{ij}), i, j \in N$ cu

$$b_{ij} = \begin{cases} b(i,j) & \text{dacă} \quad i \neq j \text{ şi } (i,j) \in A; \\ 0 & \text{dacă} \quad i = j; \\ \infty & \text{dacă} \quad i \neq j \text{ şi } (i,j) \notin A. \end{cases}$$

Algoritmul Floyd-Warshall determină matricea distanţelor $D = (d_{ij}),\ i, j \in N$ şi matricea predecesor $P = (p_{ij}),\ i, j \in N$.

```
(1)   PROGRAM FLOYD-WARSHALL;
(2)   BEGIN
(3)      FOR i := 1 TO n DO
(4)         FOR j := 1 TO n DO
(5)         BEGIN
(6)            d_ij := b_ij;
(7)            IF i ≠ j AND d_ij < ∞
(8)               THEN p_ij := i
(9)               ELSE p_ij := 0
(10)        END;
(11)     FOR k := 1 TO n DO
(12)        FOR i := 1 TO n DO
(13)           FOR j := 1 TO n DO
(14)              IF d_ik + d_kj < d_ij
(15)                 THEN BEGIN d_ij := d_ik + d_kj; p_ij := p_kj; END;
(16)  END.
```

Un drum minim D_{ijp} de la nodul i la nodul j se determină cu algoritmul următor:

```
(1)   PROGRAM DRUM;
(2)   BEGIN
(3)      k := n; x_k := j;
(4)      WHILE x_k ≠ i DO
(5)      BEGIN
(6)         x_{k-1} := p_{i x_k};
(7)         k := k - 1;
(8)      END;
(9)   END.
```

Drumul minim este $D_{ijp} = (x_k, x_{k+1}, \ldots, x_{n-1}, x_n) = (i, x_{k+1}, \ldots, x_{n-1}, j)$.

Teorema 4.7. *Algoritmul Floyd-Warshall determină matricea distanţelor D şi matricea predecesor P ale reţelei orientate $G = (N, A, b)$ cu $b : A \to \Re$ şi cu proprietatea că G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$.*

Demonstraţie. Fie D_0, P_0 matricele definite în liniile (3) la (10) şi D_k, P_k matricele calculate în liniile (11) la (15) la iteraţia k. Prin inducţie după k arătăm că $D_k = \left(d_{ij}^k\right)$ este matricea valorilor drumurilor minime de la i la j având nodurile interioare din $\{1, \ldots, k\}$. Pentru $k = 0$ avem $D_0 = B$ şi afirmaţia este evident adevărată. Presupunem afirmaţia adevărată pentru $k-1$. Liniile (14),(15) la iteraţia k sunt echivalente cu $d_{ij}^k = \min\{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\}$. Din ipoteza inductivă şi principiul optimalităţii lui Bellman rezultă că $D_k = \left(d_{ij}^k\right)$ este matricea valorilor drumurilor minime de la i la j având nodurile interioare din $\{1, \ldots, k\}$. Deoarece G nu conţine circuite $\overset{\circ}{D}$ cu valoarea $b(\overset{\circ}{D}) < 0$ şi în conformitate cu cele precizate mai sus rezultă că D_n este matricea distanţelor. De asemenea, din modul cum se determină p_{ij} rezultă că P_n este matricea predecesor cu ajutorul căreia se determină drumurile minime D_{ijp}. ■

Teorema 4.8. *Algoritmul Floyd-Warshall are complexitatea* $O\left(n^3\right)$.

Demonstraţie. Evident.■

Observaţia 4.4. Dacă se defineşte $b_{ii} = \infty,\ i \in N$, atunci elementul $d_{ii} < \infty$ reprezintă valoarea unui circuit minim ce trece prin i. Dacă $b_{ii} = 0,\ i \in N$ şi se renunţă la ipoteza restrictivă că oricare circuit $\overset{\circ}{D}$ are valoarea $b(\overset{\circ}{D}) \geq 0$, atunci se obţine posibilitatea testării existenţei circuitelor de valoare negativă în reţeaua orientată $G = (N, A, b)$. Într-adevăr, dacă $d_{ii} < 0$ atunci reţeaua conţine un circuit de valoare negativă care trece prin nodul i.

Exemplul 4.4. Se consideră reţeaua reprezentată în figura 4.6.

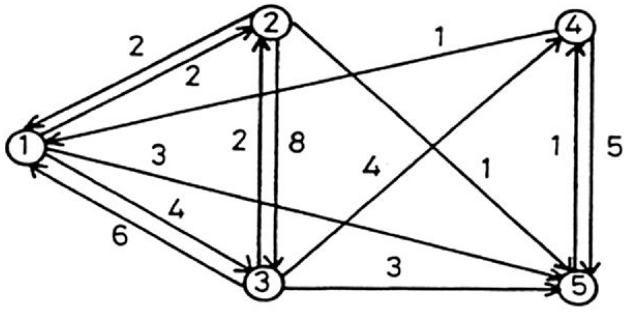


Fig.4.6

$$D_0 = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 8 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \infty & \infty & 0 & 5 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}, \quad P_0 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 2 & 0 & 2 \\ 3 & 3 & 0 & 3 & 3 \\ 4 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}, \quad P_1 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 2 \\ 3 & 3 & 0 & 3 & 3 \\ 4 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 2 \\ 2 & 3 & 0 & 3 & 3 \\ 4 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}, \quad P_3 = \begin{bmatrix} 0 & 1 & 1 & 3 & 1 \\ 2 & 0 & 1 & 3 & 2 \\ 2 & 3 & 0 & 3 & 3 \\ 4 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ 2 & 4 & 6 & 1 & 0 \end{bmatrix}, \quad P_4 = \begin{bmatrix} 0 & 1 & 1 & 3 & 1 \\ 2 & 0 & 1 & 3 & 2 \\ 2 & 3 & 0 & 3 & 3 \\ 4 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 5 & 0 \end{bmatrix}$$

$$D_5 = \begin{bmatrix} 0 & 2 & 4 & 4 & 3 \\ 2 & 0 & 6 & 2 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ 2 & 4 & 6 & 1 & 0 \end{bmatrix}, \quad P_5 = \begin{bmatrix} 0 & 1 & 1 & 5 & 1 \\ 2 & 0 & 1 & 5 & 2 \\ 2 & 3 & 0 & 3 & 3 \\ 4 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 5 & 0 \end{bmatrix}$$

Să determinăm drumul minim D_{52p} : $x_5 = 2,\ x_4 = p_{52} = 1,\ x_3 = p_{51} = 4,\ x_2 = p_{54} = 5$, deci $D_{52p} = (5, 4, 1, 2)$ cu $b(D_{52p}) = b(5, 4) + b(4, 1) + b(1, 2) = 1 + 1 + 2 = 4 = d(5, 2)$.

4.4 Aplicaţii

4.4.1 Reţele de comunicaţii

O reţea $G = (N, A, b)$ poate reprezenta o reţea de comunicaţie cu nodurile N şi rutele directe între noduri formând mulţimea arcelor A. Dacă $b(a)$ reprezintă lungimea arcului a, atunci PDM1, PDM2, PDM3 definite în paragraful 4.1 reprezintă probleme naturale care se pun în astfel de reţele: determinarea drumului/drumurilor cel/celor mai scurt/scurte. Un exemplu concret a fost prezentat în paragraful 1.6.

O problemă specială constă în determinarea drumului cel mai sigur de la nodul s la nodul t. Dacă $p(i, j)$ este o *probabilitate* de funcţionare a arcului $(i, j) \in A$ atunci, presupunând că arcele funcţionează independent unele de altele, probabilitatea de funcţionare a drumului D este: $p(D) = \prod_D p(i, j)$. Considerând $b(i, j) = -logp(i, j)$, problema drumului minim de la s la t semnifică determinarea drumului cel mai sigur de la s la t.

4.4.2 Problema rucsacului

Problema rucsacului este un model clasic în literatura cercetărilor operaţionale.

Un excursionist trebuie să decidă ce obiecte să includă în rucsacul său în vederea unei călătorii. El are de ales între p obiecte, obiectul i are greutatea g_i (în kilograme) şi o utilitate u_i adusă de introducerea obiectului i în rucsac. Obiectivul excursionistului este să maximizeze utilitatea călătoriei astfel încât greutatea obiectelor introduse în rucsac să nu depăşească g kilograme. Această problemă a rucsacului are următoarea formulare ca problemă de programare în numere întregi:

$$\max \sum_{i=1}^{p} u_i x_i$$

$$\sum_{i=1}^{p} g_i x_i \leq g$$

$$x_i \in \{0,1\} \text{ pentru } i = 1, \ldots, p$$

Această problemă se poate rezolva utilizând metode ale programării dinamice. În continuare vom formula problema rucsacului ca o problemă de drum optim într-o reţea. Această aplicaţie pune în evidenţă legătura dintre modelele de programare dinamică discretă şi problemele de drum optim într-o reţea.

Asociem problemei rucsacului o problemă de drum optim într-o reţea $G = (N, A, b)$ care se defineşte în modul următor. Mulţimea nodurilor este $N = N_0 \cup N_1 \cup \ldots \cup N_p \cup N_{p+1}$, unde $N_0 = \{s\},\ N_i = \{i(0), \ldots, i(g)\},\ i = 1, \ldots, p,\ N_{p+1} = \{t\}$. Submulţimile de noduri $N_0, N_1, \ldots, N_p, N_{p+1}$ reprezintă straturi ale mulţimii nodurilor: N_0 stratul corespunzător nodului sursă s, $N_1, \ldots, N_p$ straturile corespunzătoare obiectelor $1, \ldots, p$ şi N_{p+1} stratul corespunzător nodului stoc t. Nodul $i(k)$ are semnificaţia că obiectele $1, \ldots, i$ au consumat k unităţi din capacitatea rucsacului. Nodul $i(k)$ are cel mult două arce incidente către exterior, corespunzătoare următoarelor două decizii:
(1) nu se include obiectul $i + 1$ în rucsac;
(2) se include obiectul $i + 1$ în rucsac, dacă $k + g_{i+1} \leq g$.
Arcul corespunzător primei decizii este $(i(k), (i+1)(k))$ cu $b\,(i(k), (i+1)(k)) = 0$ şi arcul corespunzător celei de a doua decizii (cu condiţia $k + g_{i+1} \leq g$) este $(i(k), (i+1)(k+g_{i+1})$ cu $b\,(i(k), (i+1)(k+g_{i+1})) = u_{i+1}$. Nodul sursă s are două arce incidente către exterior: $(s, 1(0))$ cu $b(s, 1(0)) = 0$ şi $(s, 1(g_1))$ cu $b(s, 1(g_1)) = u_1$ corespunzătoare celor două decizii de a nu include sau de a include obiectul 1 în rucsac. Se introduc şi arcele $(p(k), t)$ cu $b(p(k), t) = 0,\ k = 0, \ldots, g$.

Fiecare soluţie admisibilă a problemei rucsacului defineşte un drum de la nodul sursă s la nodul stoc t; soluţia admisibilă şi drumul au aceeaşi utilitate. Invers, fiecare drum de la nodul sursă s la nodul stoc t defineşte o soluţie admisibilă a problemei rucsacului cu aceeaşi utilitate.

Exemplul 4.5. Ilustrăm formularea prezentată mai sus pentru o problemă a rucsacului cu patru obiecte care au greutăţile şi utilităţile indicate în tabelul de mai jos.

i	1	2	3	4
u_i	40	15	20	10
g_i	4	2	3	1

Figura 4.7 arată reţeaua $G = (N, A, b)$ asociată problemei rucsacului presupunând că rucsacul are capacitatea de $g = 6$.

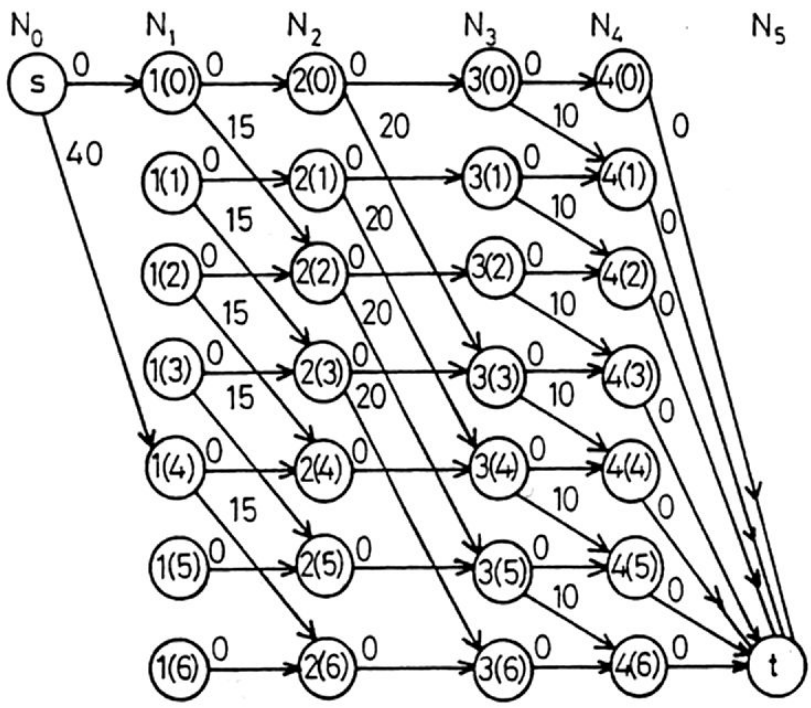


Fig.4.7

Drumul $D = (s, 1(0), 2(2), 3(5), 4(5), t)$ implică soluţia care include obiectele 2 şi 3 în rucsac şi exclude obiectele 1 şi 4.

Corespondenţa dintre problema rucsacului şi reţeaua asociată $G = (N, A, b)$ arată că dacă în reţeaua G se determină un drum de utilitate maximă (drum maxim) de la nodul sursă s la nodul stoc t se rezolvă problema rucsacului. Putem transforma problema drumului maxim într-o problemă de drum minim prin definirea costurilor arcelor egale cu valorile negative ale utilităţilor arcelor. Deoarece reţeaua $G = (N, A, b)$ este o reţea secvenţială (arcele sunt de la stratul N_i la stratul N_{i+1}, $i = 0, 1, \ldots, p,$) ea nu conţine circuite şi deci în G nu există circuite $\overset{\circ}{D}$ cu $b(\overset{\circ}{D}) < 0$. Astfel problema drumului minim în reţeaua G poate fi rezolvată cu algoritmul Bellman-Ford, eventual uşor modificat.

4.4.3 Programarea proiectelor

Pentru executarea unui proiect complex (de exemplu, construcţia unui baraj, a unui centru comercial sau a unui avion), diferitele activităţi trebuie să fie bine coordonate pentru a evita pierderea de timp şi bani. Problemele practice sunt complexe datorită restricţiilor de utilizare concurentă a resurselor (oameni, utilaje etc.) de către diversele activităţi. Ne limităm la cazul simplu unde avem restricţii pe secvenţa cronologică a activităţilor: există unele activităţi care nu pot începe înaintea terminării altor activităţi.

Se cere să se determine un plan de organizare a proiectului astfel încât timpul total de execuţie să fie minim. Două metode foarte similare pentru rezolvarea acestei probleme, numite *Critical Path Method (CPM)* şi *Project Evaluation and Review Technique (PERT)* au fost dezvoltate între anii 1956 şi 1958 de două grupuri diferite. CPM a fost introdus de E.I. du Pont de la Nemours & Company pentru programarea proiectelor de construcţii şi PERT a fost introdus de Remington Rand de la U.S. Navy pentru programarea cercetării şi dezvoltării activităţilor din cadrul programului rachetei Polaris. CPM - PERT este bazată pe determinarea drumurilor maxime într-o reţea orientată fără circuite. Vom utiliza o formulare în care activităţile proiectului sunt reprezentate prin noduri; alternativ, se pot reprezenta prin arce.
Asociem proiectului o reţea orientată $G = (N, A, b)$ în modul următor. Mulţimea nodurilor $N = \{1, \ldots, n\}$ este mulţimea activităţilor proiectului. Mulţimea arcelor este $A = \{(i, j) \mid i, j \in N,$ activitatea j urmează imediat după (nu există activităţi intermediare) activitatea $i\}$. Dacă τ_i este timpul de execuţie al activităţii i, atunci valoarea arcului (i, j) este $b(i, j) = \tau_i$. Observăm că G este fără circuite, deoarece altfel activităţile dintr-un circuit niciodată nu pot să înceapă. În acest caz G conţine cel puţin un nod y cu $\rho^-(y) = 0$ şi cel puţin un nod x cu $\rho^+(x) = 0$. Construim reţeaua extinsă $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{b})$ unde $\widetilde{N} = \widetilde{N}_1 \cup \widetilde{N}_2 \cup \widetilde{N}_3, \widetilde{N}_1 = \{s\}, \widetilde{N}_2 = N, \widetilde{N}_3 = \{t\}, \widetilde{A} = \widetilde{A}_1 \cup \widetilde{A}_2 \cup \widetilde{A}_3, \widetilde{A}_1 = \{(s, y) \mid y \in \widetilde{N}_2, \rho^-(y) = 0\}, \widetilde{A}_2 = A, \widetilde{A}_3 = \{(x, t) \mid x \in \widetilde{N}_2, \rho^+(x) = 0\}, \widetilde{b}(s, y) = 0, (s, y) \in \widetilde{A}_1, \widetilde{b}(x, y) = b(x, y), (x, y) \in \widetilde{A}_2, \widetilde{b}(x, t) = \tau_x, (x, t) \in \widetilde{A}_3$. Nodul sursă s este nod rădăcină al reţelei $\widetilde{G}$ şi considerăm $\widetilde{G}$ sortată topologic.

Notăm cu $\widetilde{d}(i)$ timpul cel mai devreme posibil la care poate să înceapă activitatea i. Deoarece toate activităţile predecesoare activităţii i au fost terminate, obţinem următorul sistem de ecuaţii:

$$\widetilde{d}(s) = 0, \widetilde{d}(j) = \max\{\widetilde{d}(i) + \widetilde{b}(i, j) \mid (i, j) \in \widetilde{A}\}.$$

Acest sistem de ecuaţii este similar sistemului ecuaţiilor lui Bellman şi descrie drumurile maxime din $\widetilde{G}$. La fel ca în cazul sistemului ecuaţiilor lui Bellman sistemul de mai sus are soluţie unică şi poate fi rezolvat recursiv deoarece $\widetilde{G}$ este sortată topologic. Timpul minim de execuţie al proiectului este $T = \widetilde{d}(t)$ valoarea maximă a drumului de la s la t. Dacă proiectul este terminat la timpul T, timpul cel mai târziu $T(i)$ la care putem începe activitatea i este dat recursiv de

$$T(t) = T,\ T(i) = \min\{T(j) - \widetilde{b}(i, j) \mid (i, j) \in \widetilde{A}\}.$$

Astfel, $T(t) - T(i)$ este valoarea drumului maxim de la i la t. Evident, considerăm $T(s) = 0$. Rezerva de timp a activităţii i este $r(i) = T(i) - \widetilde{d}(i)$. Toate activităţile i având rezerva $r(i) = 0$ se numesc *activităţi critice*, deoarece ele trebuie să înceapă la timpul $T(i) = \widetilde{d}(i)$, astfel orice întârziere a lor conduce la întârzierea execuţiei proiectului. Observăm că fiecare drum maxim de la s la t conţine numai activităţi critice; pentru acest motiv fiecare astfel de drum este numit *drum critic*. În general există mai multe drumuri critice.

Exemplul 4.6. Considerăm simplificat construcţia unei case. Lista activităţilor, a timpilor necesari acestor activităţi şi activităţile predecesoare fiecărei activităţi sunt prezentate în tabelul de mai jos.

Nod	Activitate	Timp	Activitate predecesoare
1	Pregătirea şantierului de lucru	3	-
2	Furnizarea materialelor de construcţii	2	-
3	Săparea şanţurilor pentru fundaţie	2	1,2
4	Construirea fundaţiei	2	3
5	Construirea zidurilor	7	4
6	Construirea suporturilor acoperişului	3	5
7	Acoperirea acoperişului	1	6
8	Instalaţii exterioare casei	3	4
9	Tencuire exterioară	2	7,8
10	Punerea ferestrelor	1	7,8
11	Punerea tavanelor (plafoanelor)	3	5
12	Pregătirea grădinii	4	9,10
13	Instalaţii exterioare casei	5	11
14	Izolarea pereţilor	3	10,13
15	Zugrăvirea pereţilor	3	14
16	Mutarea	5	15

Reţeaua orientată $\widetilde{G} = (\widetilde{N}, \widetilde{A}, \widetilde{b})$ este reprezentată în figura 4.8.

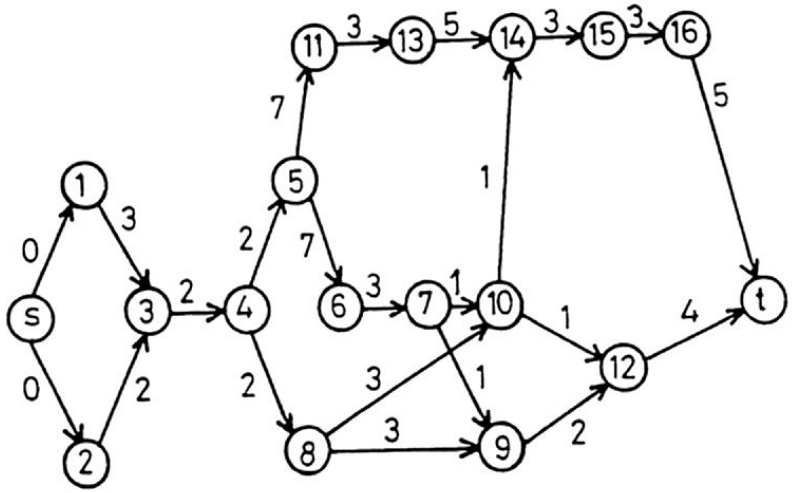


Fig.4.8

Utilizând sistemul de ecuaţii de mai sus calculăm consecutiv $\widetilde{d}(s) = 0, \widetilde{d}(1) = 0, \widetilde{d}(2) = 0, \widetilde{d}(3) = 3, \widetilde{d}(4) = 5, \widetilde{d}(5) = 7, \widetilde{d}(8) = 7, \widetilde{d}(6) = 14, \widetilde{d}(11) = 14, \widetilde{d}(13) = 17, \widetilde{d}(7) = 17, \widetilde{d}(9) = 18, \widetilde{d}(10) = 18, \widetilde{d}(12) = 20, \widetilde{d}(14) = 22, \widetilde{d}(15) = 25, \widetilde{d}(16) = 28, \widetilde{d}(t) = 33$. Analog calculăm $T(t) = 33, r(t) = 0; T(16) = 28, r(16) = 0; T(15) = 25, r(15) = 0; T(12) = 29, r(12) = 9; T(14) = 22, r(14) = 0; T(9) = 27, r(9) = 9; T(10) = 21, r(10) = 3; T(7) = 20, r(7) = 3; T(13) = 17, r(13) = 0; T(6) = 17, r(6) = 3; T(11) = 14, r(11) = 0; T(5) = 7, r(5) = 0; T(8) = 18, r(8) = 11; T(4) = 5, r(4) = 0; T(3) = 3, r(3) = 0; T(1) = 0, r(1) = 0; T(2) = 1, r(2) = 1; T(s) = 0, r(s) = 0$. Astfel,

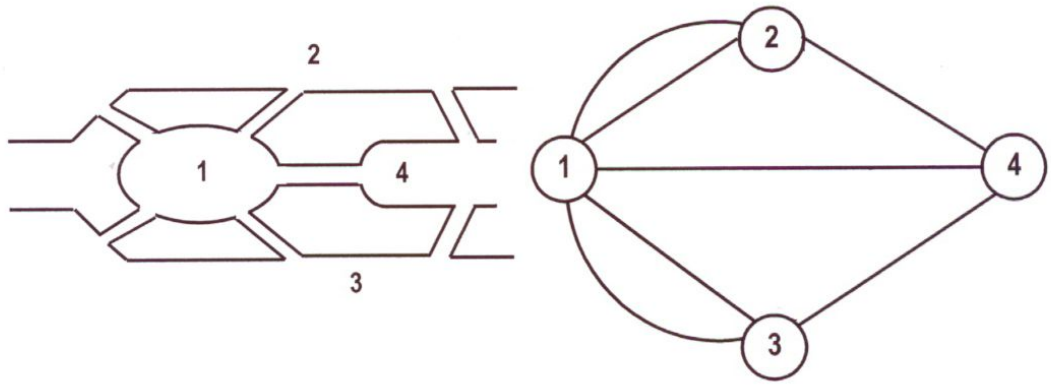
activităţile critice sunt s, 1,3,4,5,11,13,14,15,16,t şi ele formează (în această ordine) drumul critic (care este, în acest caz, unic).

Capitolul 5

Probleme euleriene şi hamiltoniene

5.1 Probleme euleriene

Definiţia 5.1. Într-un graf neorientat $G = (N, A)$ se numeşte *lanţ* (respectiv *ciclu*) *eulerian*, un lanţ (respectiv ciclu) simplu care trece prin toate muchiile grafului G.
Observaţia 5.1. Noţiunea de lanţ (respectiv ciclu) eulerian are sens şi pentru multigrafuri neorientate. Aceste noţiuni au fost introduse de matematicianul elveţian Euler care a publicat în 1736 rezolvarea problemei podurilor din Königsberg (azi Kaliningrad). Acest articol este considerat originea teoriei grafurilor. Oraşul Königsberg, care pe timpul lui Euler era în Prusia de Est, este traversat de râul Pregel, care curge de o parte şi de alta a insulei Kneiphof şi are şapte poduri, ca în figura 5.1(a).

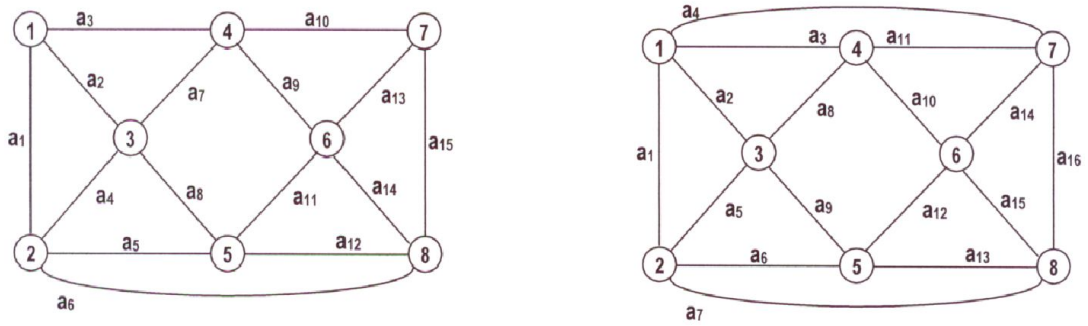


(a) (b)

Fig.5.1

Un pieton primbându-se, va putea să traverseze numai o singură dată fiecare pod înainte de a se întoarce la locul de pornire? Această problemă a pasionat locuitorii oraşului Königsberg, până în anul 1736 când Euler a arătat că nu este posibil. Modelul lui Euler pentru problemă a fost un graf neorientat cu patru vârfuri, câte un vârf pentru fiecare regiune de uscat şi cu şapte muchii, câte o muchie pentru fiecare pod. Acest graf este reprezentat în figura 5.1(b). În terminologia teoriei grafurilor, problema constă în a determina un ciclu eulerian.

Exemplul 5.1. Graful neorientat din figura 5.2(a) conţine lanţul eulerian $L = (a_1, a_4, a_2, a_3, a_7, a_8, a_5, a_6, a_{12}, a_{11}, a_9, a_{10}, a_{13}, a_{14}, a_{15}$, de la nodul 1 la nodul 7, dar nu conţine nici un ciclu eulerian.



a) (b)

Fig.5.2

Graful neorientat din figura 5.2(b) conţine ciclul eulerian $\overset{\circ}{L}= (a_1, a_5, a_2, a_3, a_8, a_9, a_6, a_7, a_{13}, a_{12}, a_{10}, a_{11}, a_{14}, a_{15}, a_{16}, a_4$ de la nodul 1 la nodul 1.

În continuare expresia turneu eulerian neorientat desemnează fie un lanţ eulerian, fie un ciclu eulerian.

Teorema 5.1. *Un graf neorientat $G = (N, A)$, conţine un lanţ (respectiv un ciclu) eulerian dacă şi numai dacă G este conex şi numărul nodurilor cu grad impar este 2 (respectiv 0).*

Demonstraţie. Dacă $G = (N, A)$, conţine un turneu eulerian neorientat atunci evident că G este conex şi numărul nodurilor cu grad impar este fie 2, fie 0, după cum turneul este fie lanţ, fie ciclu, deoarece dacă se ajunge într-un nod y pe muchia $[x, y]$ se poate întotdeauna pleca din acest nod pe o altă muchie $[y, z]$ cu excepţia primului nod şi a ultimului nod în cazul unui lanţ eulerian. Reciproca se demonstrează prin inducţie după numărul m al muchiilor. Pentru grafurile neorientate cu $m = 2$ afirmaţia este evident adevărată. Presupunem afirmaţia adevărată pentru grafurile neorientate cu $|A| < m$ şi vom arăta că afirmaţia este adevărată şi pentru grafurile neorientate cu $A = m > 2$. Dacă G conţine două noduri cu grade impare, le notăm cu x_1 şi x_n. Construim un turneu eulerian neorientat T plecând de la un nod x_i ($x_i = x_1$ dacă există două noduri cu grade impare) şi se parcurg muchii neutilizate. Când se ajunge într-un nod $x_j \neq x_i$ $(x_j \neq x_n)$ se poate pleca din x_j, deoarece este evident că s-au utilizat un număr impar de muchii incidente în x_j. După un număr k de muchii parcurse o singură dată, nu se mai poate pleca din nodul curent x_j deoarece toate muchiile incidente la x_j

au fost utilizate. În acest caz există două posibilităţi:
p_1) $x_j = x_n$, când G conţine 2 noduri cu grade impare;
p_2) $x_j = x_i$, când G conţine 0 noduri cu grade impare.

Turneul neorientat construit până la acest pas îl notăm cu T' şi are k muchii. Dacă avem $k = m$ atunci evident $T' \equiv G$ şi $T \equiv T'$. Dacă avem $k < m$ atunci prin eliminarea din G a tuturor muchiilor din T' se obţin un subgraf $\overline{G}' = \left(N, \overline{A}'\right)$ care are evident toate nodurile cu grade pare.

Fie $\overline{C}'_1, \ldots \overline{C}'_p$ componentele conexe ale subgrafului $\overline{G}'$. Prin ipoteza inducţiei componentele conexe admit ciclurile euleriene $\overset{\circ}{\overline{L}}'_1, \ldots, \overset{\circ}{\overline{L}}'_p$. Deoarece G este conex turneul eulerian neorientat T' trebuie să treacă prin cel puţin un nod din fiecare componentă conexă a lui $\overline{G}'$, să spunem, prin nodurile $y_i \in \overline{C}'_i,\ i = 1, \ldots, p$, în această ordine. Atunci turneul eulerian neorientat T este următorul:

$$T = L_{x_i y_1} \cup \overset{\circ}{\overline{L}}'_1 \cup L_{y_1 y_2} \cup \ldots \cup \overset{\circ}{\overline{L}}'_p \cup L_{y_p x_j}, \text{unde } L_{x_i y_1} \cup \ldots \cup L_{y_p x_j} = T'.$$

Dacă $x_j = x_n$ atunci T este un lanţ eulerian, iar dacă $x_j = x_i$ atunci T este un ciclu eulerian. ■

Observaţia 5.2. Teorema 5.1 este valabilă şi pentru multigrafuri neorientate.

În continuare se prezintă un algoritm pentru determinarea unui ciclu eulerian. În acest algoritm se utilizează următoarele notaţii:

W reprezintă lista nodurilor vizitate în ordinea în care au fost vizitate;
s reprezintă nodul sursă din care se pleacă iniţial;
x reprezintă nodul curent vizitat;
A' reprezintă mulţimea muchiilor deja parcurse şi $\overline{A}' = A - A'$;
$\overline{V}'(x)$ reprezintă lista nodurilor din $\overline{G}' = \left(N, \overline{A}'\right)$ adiacente cu x.

```
(1) PROGRAM CICLUE;
(2) BEGIN
(3)     W := {s}; x := s; A' := ∅; Ā' := A;
(4)     WHILE |V̄'(s)| > 0 DO
(5)     BEGIN
(6)         IF |V̄'(s)| > 1
(7)             THEN se selectează y din V̄'(x) astfel încât [x, y], prin eliminare,
                    nu măreşte numărul componentelor conexe ale lui Ḡ' = (N, Ā')
(8)             ELSE se selectează unicul nod y din V̄'(x);
(9)         V̄'(x) := V̄'(x) − {y}; V̄'(y) := V̄'(y) − {x};
(10)        A' := A' ∪ {[x, y]} Ā' := Ā' − {[x, y]}; ;
(11)        x := y;
(12)        se adaugă x la urma listei W;
(13)    END;
(14) END.
```

Teorema 5.2 *Algoritmul CICLUE determină un ciclu eulerian* $\overset{\circ}{L}$ *al unui graf neorientat* $G = (N, A)$ *conex şi gradul fiecărui nod este un număr par.*
Demonstraţie. Mai întâi arătăm că este posibil întotdeauna determinarea, cu ajutorul instrucţiunii IF din liniile (6), (7) şi (8), a unui nod y care urmează a fi vizitat. Deoarece $G = (N, A)$ este conex şi gradul fiecărui nod este număr par rezultă că iniţial $\left|\overline{V}'(x)\right| > 1$ pentru fiecare nod x din N. La fiecare iteraţie a instrucţiunii WHILE, în linia (9), se elimină y din $\overline{V}'(x)$ şi x din $\overline{V}'(y)$. Deci după un număr de iteaţii avem $\left|\overline{V}'(x)\right| = 1$. În acest caz, conform liniei (8), este selectat unicul nod y din $\overline{V}'(x)$. În cazul $\left|\overline{V}'(x)\right| > 1$ considerăm subcazurile $x = s$ şi $x \neq s$. Dacă $x = s$ atunci nici o muchie $[s, y]$, prin eliminare, nu măreşte numărul componentelor conexe ale lui $\overline{G}' = \left(N, \overline{A}'\right)$, deoarece pentru oricare revenire în nodul s, nodurile din $\overline{G}'$ au grad par. Dacă $x \neq s$ atunci, evident, cel mult o muchie $[x, y]$, prin eliminare, măreşte numărul componentelor conexe ale lui $\overline{G}'$ şi există nod y care verifică condiţia din linia (7).

Algoritmul se opreşte când $\left|\overline{V}'(s)\right| = 0$. Evident că în conformitate cu cele precizate mai sus, în acest caz, s-a determinat un ciclu eulerian $\overset{\circ}{L} = W$. ■
Teorema 5.3 *Algoritmul CICLUE are complexitatea* $O\left(m^2\right)$.
Demonstraţie. Instrucţiunea WHILE se execută de m ori, odată pentru fiecare muchie a ciclului eulerian, adică a grafului $G = (N, A)$. Pentru căutarea unui nod y în linia (7) astfel încât prin eliminarea muchiei $[x, y]$ nu se măreşte numărul componentelor conexe ale lui $\overline{G}' = \left(N, \overline{A}'\right)$ sunt necesare cel mult m operaţii. Deci algoritmul are complexitatea $O\left(m^2\right)$. ■
Observaţia 5.3. Dacă graful neorientat $G = (N, A)$ îndeplineşte condiţiile de existenţă a unui lanţ eulerian de la nodul x_1 la nodul x_n, atunci acest lanţ se poate determina în modul următor. Se construieşte graful neorientat $\tilde{G} = \left(\tilde{N}, \tilde{A}\right)$ cu $\tilde{N} = N$, $\tilde{A} = A \cup \{[x_1, x_n]\}$. Evident că graful neorientat $\tilde{G}$ îndeplineşte condiţiile de existenţă a unui ciclu eulerian. Dacă $\overset{\circ}{\tilde{L}}$ este ciclu eulerian al grafului neorientat $\tilde{G}$ atunci $L = \overset{\circ}{\tilde{L}} - \{[x_1, x_n]\}$ este lanţul eulerian, de la nodul x_1 la nodul x_n, al grafului neorientat G.

De asemenea, remarcăm faptul că există un algoritm pentru determinarea unui ciclu eulerian cu complexitatea $O(m)$, dar cu un text mult mai lung şi mai puţin clar decât algoritmul CICLUE.
Exemplul 5.2. Să se determine cu algoritmul prezentat mai sus un ciclul eulerian al grafului neorientat din figura 5.2(b).

Sunt indeplinite condiţiile din Teorema 5.1 şi anume G este conex şi pentru fiecare nod x din N avem $\rho(x) = 4$ este număr par. În adevăr avem: $\overline{V}'(1) = (2, 3, 4, 7)$, $\overline{V}'(2) = (1, 3, 5, 8)$, $\overline{V}'(3) = (1, 2, 4, 5)$, $\overline{V}'(4) = (1, 3, 6, 7)$,
$\mathrm{V}'(5) = (2, 3, 6, 8)$, $\overline{V}'(6) = (4, 5, 7, 8)$, $\overline{V}'(7) = (1, 4, 6, 8)$, $\overline{V}'(8) = (2, 5, 6, 7)$.
Iniţializări: $W := \{1\}$, $x = 1$, $A' := \emptyset, \overline{A}' = A$;
Iteraţia 1: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(1)\right| > 1$
$y = 2$, $\overline{V}'(1) = (3, 4, 7), \overline{V}'(2) = (3, 5, 8), A' := A' \cup \{[1, 2]\}, \overline{A}' := \overline{A}' - \{[1, 2]\}$,
$x = 2$, $W = (1, 2)$

Iteraţia 2: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(2)\right| > 1$
$y = 3$, $\overline{V}'(2) = (5, 8), \overline{V}'(3) = (1, 4, 5), A' := A' \cup \{[2, 3]\}, \overline{A}' := \overline{A}' - \{[2, 3]\}$,
$x = 3$, $W = (1, 2, 3)$
Iteraţia 3: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(3)\right| > 1$
$y = 4$, $\overline{V}'(3) = (1, 5), \overline{V}'(4) = (1, 6, 7), A' := A' \cup \{[3, 4]\}, \overline{A}' := \overline{A}' - \{[3, 4]\}$,
$x = 4$, $W = (1, 2, 3, 4)$
Iteraţia 4: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(4)\right| > 1$
$y = 6$, $\overline{V}'(4) = (1, 7), \overline{V}'(6) = (5, 7, 8), A' := A' \cup \{[4, 6]\}, \overline{A}' := \overline{A}' - \{[4, 6]\}$,
$x = 6$, $W = (1, 2, 3, 4, 6)$
Iteraţia 5: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(6)\right| > 1$
$y = 5$, $\overline{V}'(6) = (7, 8), \overline{V}'(5) = (2, 3, 8), A' := A' \cup \{[6, 5]\}, \overline{A}' := \overline{A}' - \{[6, 5]\}$,
$x = 5$, $W = (1, 2, 3, 4, 6, 5)$
Iteraţia 6: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(5)\right| > 1$
$y = 2$, $\overline{V}'(5) = (3, 8), \overline{V}'(2) = (8), A' := A' \cup \{[5, 2]\}, \overline{A}' := \overline{A}' - \{[5, 2]\}$,
$x = 2$, $W = (1, 2, 3, 4, 6, 5, 2)$
Iteraţia 7: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(2)\right| = 1$
$y = 8$, $\overline{V}'(2) = \emptyset, \overline{V}'(8) = (5, 6, 7), A' := A' \cup \{[2, 8]\}, \overline{A}' := \overline{A}' - \{[2, 8]\}$,
$x = 8$, $W = (1, 2, 3, 4, 6, 5, 2, 8)$
Iteraţia 8: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(8)\right| > 1$
$y = 5$, $\overline{V}'(8) = (6, 7), \overline{V}'(5) = (3), A' := A' \cup \{[8, 5]\}, \overline{A}' := \overline{A}' - \{[8, 5]\}$,
$x = 5$, $W = (1, 2, 3, 4, 6, 5, 2, 8, 5)$
Iteraţia 9: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(5)\right| = 1$
$y = 3$, $\overline{V}'(5) = \emptyset, \overline{V}'(3) = (1), A' := A' \cup \{[5, 3]\}, \overline{A}' := \overline{A}' - \{[5, 3]\}$,
$x = 3$, $W = (1, 2, 3, 4, 6, 5, 2, 8, 5, 3)$
Iteraţia 10: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(3)\right| = 1$
$y = 1$, $\overline{V}'(3) = \emptyset, \overline{V}'(1) = (4, 7), A' := A' \cup \{[3, 1]\}, \overline{A}' := \overline{A}' - \{[3, 1]\}$,
$x = 1$, $W = (1, 2, 3, 4, 6, 5, 2, 8, 5, 3, 1)$
Iteraţia 11: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(1)\right| > 1$
$y = 4$, $\overline{V}'(1) = (7), \overline{V}'(4) = (7), A' := A' \cup \{[1, 4]\}, \overline{A}' := \overline{A}' - \{[1, 4]\}$,
$x = 4$, $W = (1, 2, 3, 4, 6, 5, 2, 8, 5, 3, 1, 4)$
Iteraţia 12: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(4)\right| = 1$
$y = 7$, $\overline{V}'(4) = \emptyset, \overline{V}'(7) = (1, 6, 8), A' := A' \cup \{[4, 7]\}, \overline{A}' := \overline{A}' - \{[4, 7]\}$,
$x = 7$, $W = (1, 2, 3, 4, 6, 5, 2, 8, 5, 3, 1, 4, 7)$
Iteraţia 13: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(7)\right| > 1$
Muchia $[7, 1]$ măreşte numărul componentelorconexe ale lui $\overline{G}' = \left(N, \overline{A}'\right)$.
$y = 6$, $\overline{V}'(7) = (1, 8), \overline{V}'(6) = (8), A' := A' \cup \{[7, 6]\}, \overline{A}' := \overline{A}' - \{[7, 6]\}$,
$x = 6$, $W = (1, 2, 3, 4, 6, 5, 2, 8, 5, 3, 1, 4, 7, 6)$
Iteraţia 14: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(6)\right| = 1$
$y = 8$, $\overline{V}'(6) = \emptyset, \overline{V}'(8) = (7), A' := A' \cup \{[6, 8]\}, \overline{A}' := \overline{A}' - \{[6, 8]\}$,
$x = 8$, $W = (1, 2, 3, 4, 6, 5, 2, 8, 5, 3, 1, 4, 7, 6, 8)$

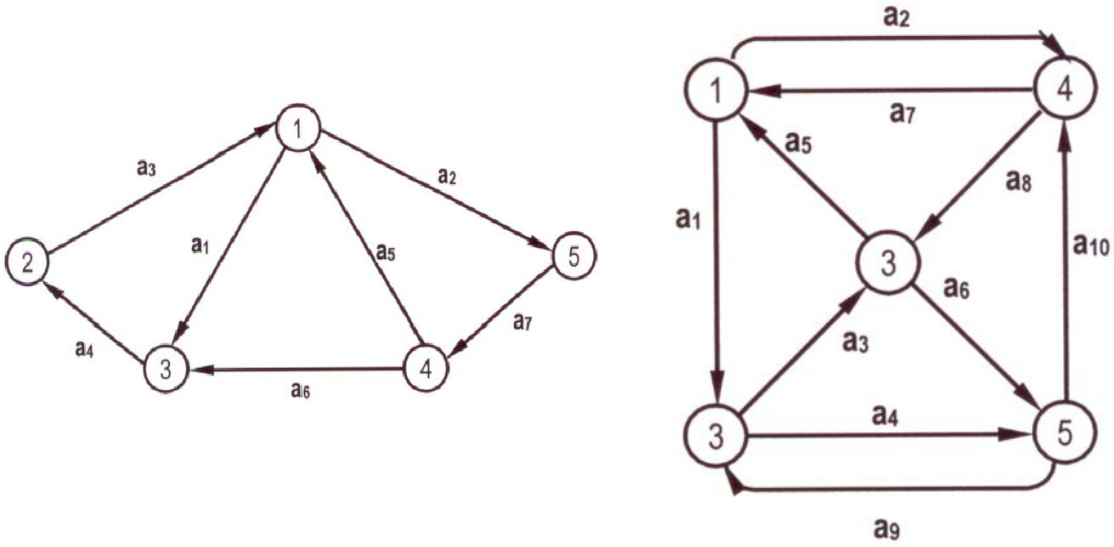
Iteraţia 15: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(8)\right| = 1$
$y = 7$, $\overline{V}'(8) = \emptyset, \overline{V}'(7) = (1), A' := A' \cup \{[8,7]\}, \overline{A}' := \overline{A}' - \{[8,7]\}$,
$x = 7$, $W = (1,2,3,4,6,5,2,8,5,3,1,4,7,6,8,7)$
Iteraţia 16: $\left|\overline{V}'(1)\right| > 0$, $\left|\overline{V}'(7)\right| = 1$
$y = 1$, $\overline{V}'(7) = \emptyset, \overline{V}'(1) = \emptyset, A' := A' \cup \{[7,1]\}, \overline{A}' := \overline{A}' - \{[7,1]\}$,
$x = 1$, $W = (1,2,3,4,6,5,2,8,5,3,1,4,7,6,8,7,1)$
Deoarece $\overline{V}'(1) = 0$ STOP.

Ciclul eulerian este dat de lista ordonată de noduri W sau $A' = (a_1, a_5, a_8, a_{10}, a_{12}, a_6, a_7, a_{13}, a_9, a_2, a_3, a_{11}, a_{14}, a_{15}, a_{16}, a_4)$.

Definiţia 5.2. Într-un graf orientat (diagraf) $G = (N, A)$ se numeşte *drum* (respectiv *circuit*) *eulerian* un drum (respectiv circuit) simplu care trece prin toate arcele digrafului G.

Definiţia 5.3. Un graf neorientat (respectiv orientat) se numeşte *eulerian* dacă conţine un ciclu (respectiv circuit) eulerian.

Exemplul 5.3. Digraful din figura 5.3(a) conţine drumul eulerian $D = (a_6, a_4, a_3, a_2, a_7, a_5, a_1)$, de la nodul 4 la nodul 3, dar nu conţine circuit Eulerian.



(a) (b)

Fig.5.3

Digraful din figura 5.3(b) conţine circuitul eulerian $\overset{\circ}{D} = (a_5, a_2, a_8, a_6, a_9, a_4, a_{10}, a_7, a_1, a_3)$ de la nodul 3 la nodul 3.

Teorema 5.4. *Un digraf $G = (N, A)$ conţine un drum eulerian de la nodul x_1 la nodul x_n dacă şi numai dacă este conex şi semigradele nodurilor sale verifică condiţiile: $\rho^+(x_1) = \rho^-(x_1) + 1$, $\rho^+(x_n) = \rho^-(x_n) - 1$, $\rho^+(x) = \rho^-(x)$ pentru $x \neq x_1$ şi $x \neq x_n$. Un digraf $G = (N, A)$ conţine un circuit eulerian dacă şi numaid acă este conex şi $\rho^+(x) = \rho^-(x)$ pentru fiecare nod x din N.*

Demonstraţie. Se face analog ca demonstraţia Teoremei 5.1.■

Exemplul 5.4. Digraful din figura 5.3(a) verifică condiţiile din Teorema 5.4 pentru existenţa drumului eulerian şi diagraful din figura 5.3(b) verifică condiţiile din această teoremă pentru existenţa circuitului eulerian.

Pentru determinarea unui circuit eulerian al unui digraf $G = (N, A)$ se poate adapta algoritmul CICLUE. Se va prezenta un algoritm specific cirucitelor euleriene. În acest algoritm se utilizează algoritmul parcugerii totale DF(APTDF) pentru determinarea unei arborescenţe parţiale $G' = (N, A')$ cu nodul rădăcină r a digrafului $G = (N, A)$.

În algoritm se utilizează următoarele notaţii:
A' desemnează mulţimea arcelor arborescenţei parţiale $G' = (N, A')$;
b desemnează un tablou unidimensional cu m elemente care au valori 0,1;
V_x^- desemnează lista nodurilor adiacente către interiorul nodului x şi este ordonată în raport cu valorile tabloului b;
i desemnează un tablou unidimensional cu n elemente care sunt utilizate ca index;
W desemnează lista nodurilor care la termianrea execuţiei algoritmului, formează circuitul eulerian.

```
(1) PROGRAM CIRCUITE;
(2) BEGIN
(3)     APTDF(G, r, A')
(4)     FOR (x, y) ∈ A DO
(5)         IF (x, y) ∈ A'
(6)             THEN b(x, y) := 1
(7)             ELSE b(x, y) := 0;
(8)         FOR x ∈ N DO
(9)         BEGIN
(10)            V_x^- := ∅; ρ^-(x) := 0; i(x) := 0;
(11)        END;
(12)        FOR (x, y) ∈ A DO
(13)            IF b(x, y) = 0
(14)                THEN se adaugă x la începutul listei V_y^-; ρ^-(y) := ρ^-(y) + 1;
(15)                ELSE se adaugă x la urma liste V_y^-; ρ^-(y) := ρ^-(y) + 1;
(16)        W := ∅; x := r;
(17)        WHILE i(x) ≤ ρ^-(x) DO
(18)        BEGIN
(19)            se adaugă x la începutul listei W;
(20)            i(x) := i(x) + 1;
(21)            IF i(x) ≤ ρ^-(x)
(22)                THEN x := V_x^-(i(x));
(23)        END;
(24) END.
```

Teorema 5.5. *Dacă $G = (N, A)$ este un digraf conex, $\rho^+(x) = \rho^-(x)$ pentru fiecare nod $x \in N$ şi $G' = (N, A')$ este o arborescenţă parţială cu nodul rădăcină r, atunci un circuit eulerian poate fi construit în sens invers cu regulile următoare:*

(1) arcul iniţial este orice arc incident către interior nodului r;

(2) arcele ulterioare sunt selectate dintre cele incidente către interior nodului curent x şi astfel încât:

(2.1) fiecare arc este utilizat o singură dată,

(2.2) nu este selectat arcul (v, x) din A' dacă există arc (u, x) din $\overline{A}'$ neutilizat;

(3) procesul se opreşte când nodul curent x nu are arce incidente către interior neutilizate.

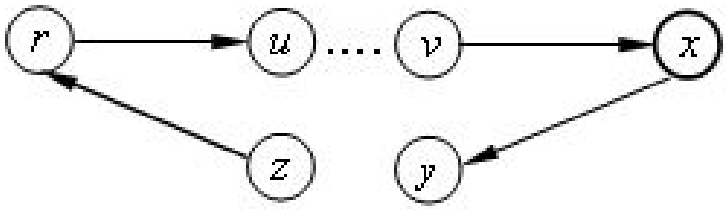
Demonstraţie. Regulile din teoremă sunt exprimate în algoritm în nodul următor:

• regula (1) este exprimată prin atribuirea $x := r$ din linia (16);

• regula (2) este exprimată prin incrementările din linia (20) şi construcţiile listelor V_y^- cu instrucţiunea FOR din liniile (12) la (15);

• regula (3) este exprimată prin condiţia din linia (17) şi instrucţiunea IF din liniile (21), (22).

Pentru a fi mai clară continuarea demonstraţiei folosim figura de mai jos:



Deoarece $\rho^+(x) = \rho^-(x)$ pentru fiecare nod x din N, drumul determinat cu regulile de mai sus se termină numai în nodul r, deci este un circuit $\overset{\circ}{D}$. Presupunem prin reducere la absurd că $\overset{\circ}{D}$ nu este un circuit eulerian. Deci există un arc $(x, y) \in A$ şi $(x, y) \notin \overset{\circ}{D}$. Cum $\rho^+(x) = \rho^-(x)$ rezultă că există un arc (v, x) astfel încât $(v, x) \notin \overset{\circ}{D}$. Putem presupune că $(v, x) \in A'$ deoarece un arc incident către interior la nodul x poate fi neutilizat din cauza regulii (2.2). Dacă se continuă acest procedeu vom ajunge la faptul că există arc $(z, r) \notin \overset{\circ}{D}$. Dar aceasta contrazice regula (3).

Teorema 5.6. *Algoritmul CIRCUITE are compelxitatea $O(m)$.*

Demonstraţie. Algoritmul parcurgerii totale DF(APTDF) are complexitatea $O(m)$. Instrucţiunile FOR de la linia (4) la linia (7) şi respectiv de la linia (12) la linia (15) se execută fiecare de m ori. Instrucţiunea FOR de la linia (8) la linia (11) se execută de n ori. Instrucţiunea WHILE se execută de m ori. Deci algoritmul are complexitatea $O(m)$. ■

În algoritm, circuitul eulerian se cosntruieşte după ce s-a determinat o arborescenţă $G' = (N, A')$ cu rădăcina r. Este posibilă construcţia inversă, anume dat un cirucit eulerian al digrafului $G = (N, A)$ să construim o arborescenţă $G' = (N, A')$ cu rădăcina r. Această construcţie se face cu următoarea regulă:

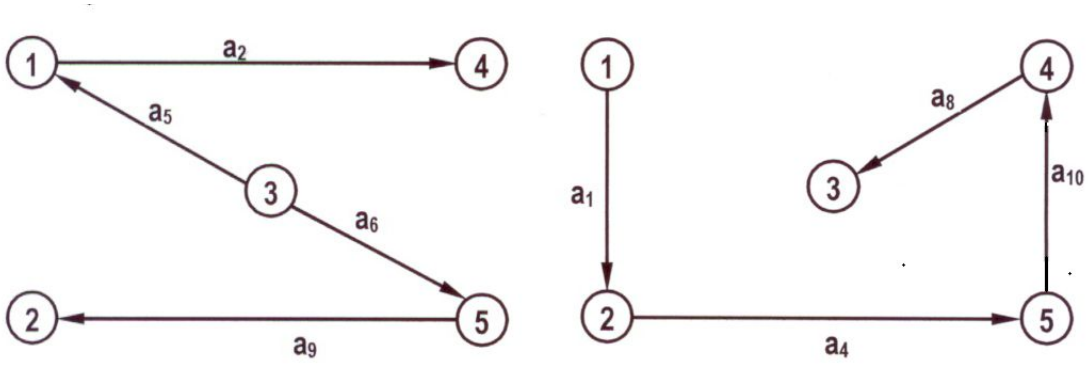
(a) se pleacă de la un nod oarecare r şi în parcurgerea directă a circuitului eulerian, pentru fiecare nod $x \neq r$, selectăm primul arc parcurs incident către interior la nodul x.

Teorema 5.7. *Subgraful $G' = (N, A')$, al unui digraf eulerian $G = (N, A)$, construit conform regulii (a) de mai sus este o arborescenţă parţială cu nodul rădăcină r.*
Demonstraţie. Conform regulii (a) avem $\rho^-(r) = 0$, $\rho^-(r) = 1$ pentru oricare nod $x \neq r$. Evident că G' este şi conex, deci G' este o arborescenţă cu rădăcina r. ■
Observaţia 5.4. Dat un circuit eulerian al diagrafului $G = (N, A)$ se poate construi o arborescenă parţială inversă (există un drum unic de la oricare nod terminal la nodul rădăcină) cu nodul rădăcină r. Această construcţie se face cu următoarea regulă:

(b) se pleacă de la un nod oarecare r şi în parcurgerea inversă a circuitului eulerian, pentru fieare nod $x \neq r$, selectăm primul arc parcurs incident către exterior la nodul x.
Observaţia 5.5. Dacă un diagraf $G = (N, A)$ îndeplineşte condiţiile de existenţă a unui drum eulerian de la nodul x_1, la nodul x_n, atunci acest drum se poate determina în modul următor. Se construieşte digraful $\tilde{G} = \left(\tilde{N}, \tilde{A}\right)$ cu $\tilde{N} = N$, $\tilde{A} = A \cup \{(x_n, x_1)\}$. Evident că digraful $\tilde{G}$ îndeplineşte condiţiile de existenţă a unui circuit eulerian. Dacă $\overset{\circ}{\tilde{D}}$ este circuitul eulerian al digrafului $\tilde{G}$ atunci $D = \overset{\circ}{\tilde{D}} - \{(x_n, x_1)\}$ este drumul eulerian, de la nodul x_1 la nodul x_n, al digrafului G.
Exemplul 5.5. Fie digraful din figura 5.3(b). Acest digraf este eulerian. Aplică algoritmul de mai sus pentru determinarea unui circuit eulerian. Se consideră $r = 3$ şi cu APTDF se determină arborescenţa cu arcele corespunzătoare elementelor $b(1, 4) = 1$, $b(3, 1) = 1$, $b(3, 5) = 1$, $b(5, 2) = 1$. Cu instrucţiunea FOR de la linia (12) la linia (15) se obţin listele ordonate $V_1^- = (4, 3)$, $V_2^- = (1, 5)$, $V_3^- = (4, 2)$, $V_4^- = (5, 1)$, $V_5^- = (2, 3)$. Evident că $\rho^-(1) = \rho^-(2) = \rho^-(3) = \rho^-(4) = \rho^-(5) = 2$. Iteraţiile instrucţiunii WHILE sunt prezentate în tabelul de mai jos.

Iteraţia	x	$i(1)$	$i(2)$	$i(3)$	$i(4)$	$i(5)$	W
0	3	0	0	0	0	0	$\emptyset$
1	4	0	0	1	0	0	(3)
2	5	0	0	1	1	0	(4,3)
3	2	0	0	1	1	1	(5,4,3)
4	1	0	1	1	1	1	(2,5,4,3)
5	4	1	1	1	1	1	(1,2,5,4,3)
6	1	1	1	1	2	1	(4,1,2,5,4,3)
7	3	2	1	1	2	1	(1,4,1,2,5,4,3)
8	2	2	1	2	2	1	(3,1,4,1,2,5,4,3)
9	5	2	2	2	2	1	(2,3,1,4,1,2,5,4,3)
10	3	2	2	2	2	2	(5,2,3,1,4,1,2,5,4,3)
11	-	2	2	3	2	2	(3,5,2,3,1,4,1,2,5,4,3)

În continuare prezentăm arborescenţele construite cu cele două reguli. Se consideră $r = 3$. Arborescenţa construită cu regula (a) este prezentată în figura 5.4(a) şi arborescenţa inversă cosntruită cu regula (b) este prezentată în figura 5.4(b).



(a) (b)

Fig.5.4

5.2 Probleme hamiltoniene

Definiţia 5.4. Într-un graf neorientat $G = (N, A)$ se numeşte *lanţ* (respectiv *ciclu*) *hamiltonian* un lanţ (respectiv ciclu) elementar care trece prin toate nodurile grafului G.

Observaţia 5.6. Noţiunea de ciclu hamiltonian provine de la matematicianul irlandez W. Hamilton, care în 1859 a propus un joc, numit jocul icosian, constând din găsirea unui ciclu elementar, care să unească toate vârfurile unui dodecaedru, marcate cu numele unor capitale din toată lumea, trecând pe muchiile dodecaedrului. Dodecaedrul este unul din cele cinci tipuri de poliedre regulate (tetraedrul, hexaedrul (cubul), octaedrul, dodecaedrul şi icosaedrul) şi are 30 de muchii, 20 de vârfuri şi 12 feţe. Reprezentarea plană a grafului dodecaedrului este arătată în figura 5.5.

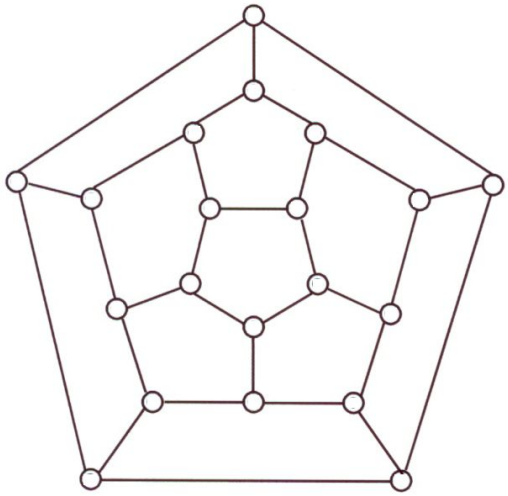
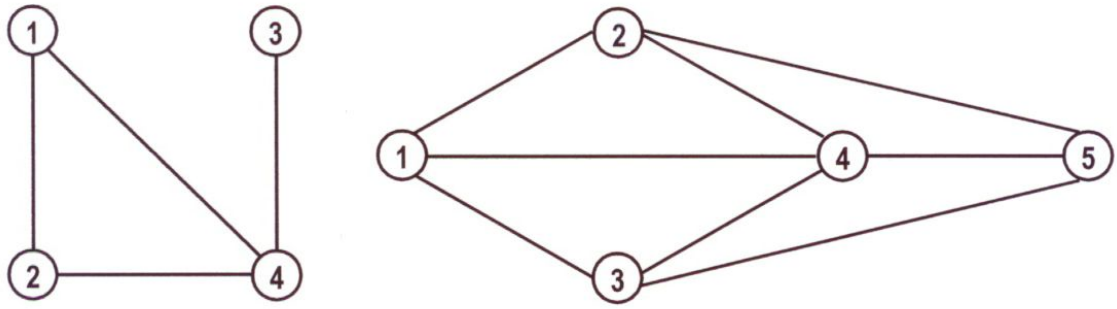


Fig.5.5

Exemplul 5.6. Graful neorientat din figura 5.6(a) conţine lanţul hamiltonian $L = (1, 2, 4, 3)$ de la nodul 1 la nodul 3. Graful neorientat din figura 5.6(b) conţine ciclul hamiltonian $\overset{\circ}{L}= (1, 3, 4, 5, 2, 1)$.



(a) (b)

Fig.5.6

Teorema 5.8. *Fie K_n graful neorientat complet cu $n \geq 3$. Acest graf conţine:*
1. $(n-1)!/2$ cicluri hamiltoniene distincte;
2. $2^{k-1}(n-k-1)!$ cicluri hamiltoniene distincte care trec prin k muchii neadiacente fixate.

Demonstraţie. 1. Deoarece K_n este complet orice succesiune de noduri este un lanţ. Deci un ciclu hamiltonian este orice succesiune de noduri în care apar toate nodurile lui K_n o singură dată, la care se adaugă primul nod din această succesiune. Prin urmare, ciclurile hamiltoniene distincte sunt permutările circulare a celor n noduri (permutările a n puncte pe cerc). Numărul permutărilor liniare ale unei mulţimi cu n elemente este $n!$. Dintr-o permutare circulară se pot obţine $2n$ permutări liniare (prin "ruperea" cercului pe arcul dintre punctul 1 şi punctul 2 sau ... punctul n şi punctul 1 şi apoi "întinderea" sa astfel ca unul din aceste două puncte poate fi plasat fie în stânga, fie în dreapta). Deci numărul ciclurilor hamiltoniene distincte este $n!/2n = (n-1)!/2$.

2. Să notăm cele k muchii fixate prin $[x_{2i-1}, x_{2i}]$, $i = 1, \ldots, k$. În ciclurile hamiltonmiene care trec prin cele k muchii neadiacente fixate nodul x_{2i-1} fie precede, fie succede nodul x_{2i}, $i = 1, \ldots, k$. Fie graful complet K_{n-k} cu $n - 2k$ noduri efective ale lui K_n care nu sunt extremităţile celor k muchii şi k noduri fictive corespunzătoare celor k muchii. Fiecare ciclu hamiltonian $\overset{\circ}{L}_{n-k}$ al lui K_{n-k} generează, prin înlocuirea celor k noduri fictive cu muchii, 2^k cicluri hamiltoniene ale lui K_n care trec prin cele k muchii fixate. Invers, din 2^k cicluri hamiltoniene ale lui K_n care trec prin cele k muchii fixate se obţine un acelaşi ciclu hamiltonian al lui K_{n-k} prin înlocuirea celor k muchii cu câte un nod fictiv. Aceasta rezultă din faptul că înlocuirea celor k noduri fictive cu muchii înseamnă alegerea celor k muchii ale lui K_n ca fiind un element dintre cele 2^k ale mulţimii $\{[x_1, x_2], [x_2, x_1]\} \times \ldots \times \{[x_{2k-1}, x_{2k}], [x_{2k}, x_{2k-1}]\}$, apoi se înlocuieşte fiecare nod fictiv al lui $\overset{\circ}{L}_{n-k}$ cu muchia corespunzătoare. Conform punctului (1) graful K_{n-k} are $(n-k-1)!/2$ cicluri hamiltoniene distincte. Deci K_n are $2^{k-1}(r-k-1)!$ cicluri hamiltoniene distincte care trec prin k muchii fixate.

Teorema 5.9. *Fie $G = (N, A)$ un graf neorientat cu $n \geq 3$. Dacă pentru oricare două noduri x şi y neadiacente are loc $\rho(x) + \rho(y) \geq n$ atunci G conţine ciclu hamiltonian.*
Demonstraţie. Să presupunem, prin reducere la absurd, că G nu conţine un ciclu hamiltonian. Să adăugăm muchii între nodurile neadiacente ale grafului G, atât timp cât graful obţinut nu conţine cicluri hamiltoniene. Prin aceasta gradele nodurilor cresc şi condiţia din enunţul teoremei rămâne în continuare verificată. Prin urmare putem presupune că G este un graf saturat cu proprietatea din enunţul teoremei, adică o muchie adăugată între oricare două noduri neadiacente crează un ciclu hamiltonian. Dacă $[x, y]$ nu este muchie a lui G, prin adăugarea acestei muchii la graful lui G se crează un ciclu hamiltonian. Deci G conţine un lanţ hamiltonian $L = (z_1, \ldots, z_n)$ cu $z_1 = x$ şi $z_n = y$. Să notăm cu $z_{i_1}, \ldots, z_{i_r}$ nodurile adiacente cu x, unde $2 = i_1 < i_2 < \ldots < i_r < n$. Nodul y nu poate fi adiacent cu z_{i_k-1}, $k = 1, \ldots, r$, deoarece în caz contrar G conţine ciclul hamiltonian $\overset{\circ}{L} = (z_1, z_2, \ldots, z_{i_k-1}, z_n, z_{n-1}, \ldots, z_{i_k}, z_n)$ care este prezentat în figura 5.7. Rezultă că $\rho(x) = r$, $\rho(y) \leq n - 1 - r$ şi se obţine că $\rho(x) + \rho(y) \leq n - 1 < n$.

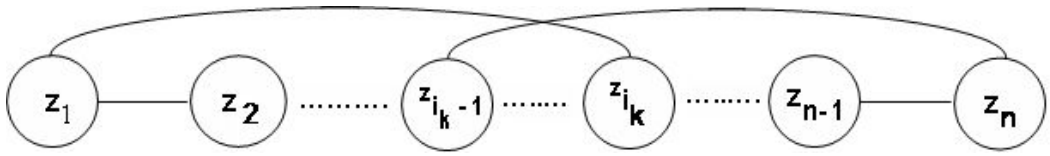


Fig.5.7

Inegalitatea $\rho(x) + \rho(y) < n$ contrazice condiţia din enunţul teoremei. Deci presupunerea făcută este falsă şi rezultă că graful neorientat G conţine ciclu hamiltonian. ■

Teorema 5.10. *Fie $G = (N, A)$ un graf neorientat cu $n \geq 3$ şi x, y două noduri neadiacente pentru care*

$$\rho(x) + \rho(y) \geq n.$$

Graful $G = (N, A)$ conţine ciclu hamiltonian dacă şi numai dacă graful neorientat $\tilde{G} = \left(N, \tilde{A}\right)$, $\tilde{A} = A \cup \{[x, y]\}$ conţine ciclu hamiltonian.

Demonstraţie. Dacă G conţine ciclu hamiltonian $\overset{\circ}{L}$ atunci evident că $\overset{\circ}{L}$ este ciclu hamiltonian şi în $\tilde{G}$.

Dacă $\tilde{G}$ conţine ciclu hamiltonian $\overset{\circ}{\tilde{L}}$ atunci pot exista următoarele două cazuri:

(c_1) muchia $[x, y]$ nu aparţine ciclului hamiltonian $\overset{\circ}{\tilde{G}}$, în acest caz $\overset{\circ}{\tilde{L}}$ este ciclu hamiltonian în G;

(c_2) muchia $[x, y]$ aparţine ciclului hamiltonian $\overset{\circ}{\tilde{L}}$; în acest caz se arată analog ca în teorema 5.9 că $\rho(x) + \rho(y) < n$ şi deci acest caz nu este posibil. ■

Teorema 5.11. *Fie $G = (N, A)$ un graf neorientat cu $n \geq 3$ şi gradele nodurilor $x_1, \ldots, x_n$ verifică inegalităţile $\rho_1 \leq \ldots \leq \rho_n$. Graful G conţine ciclu hamiltonian, dacă este satisfăcută oricare din următoarele trei condiţii:*

(1) $\rho_1 \geq n/2$*;*

(2) $\rho_k \leq k < n/2$ *implică* $\rho_{n-k} \geq n-k$ *;*

(3) $\rho_p \leq p$ *şi* $\rho_q \leq q$ *implică* $\rho_p + \rho_q \geq n$*, pentru orice* p *şi* q*.*

Demonstraţie. (1) Din $\rho_1 \leq \ldots \leq \rho_n$ şi $\rho_1 \geq n/2$ rezultă că pentru oricare x_i şi x_j obţinem $\rho_i + \rho_j \geq n$ şi conform Teoremei 5.9 G conţine ciclu hamiltonian.

(2) Să presupunem, prin reducere la absurd, că G nu conţine un ciclu hamiltonian. Analog ca în demonstraţia Teoremei 5.9, putem presupune că G este saturat, adică o muchie adăugată între oricare două noduri neadiacente crează un ciclu hamiltonian. Considerăm două noduri neadiacente x_i, x_j, astfel încât $i < j$ şi suma $i+j$ este maximă. Din maximalitatea lui $i+j$ deducem că:

- nodul x_i este adiacent cu nodurile $x_{j+1}, \ldots, x_n$;
- nodul x_j este adiacent cu nodurile $x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$;

Acestea implică:

$$\rho_i \geq n - j \tag{5.1}$$

$$\rho_j \geq n - i - 1 \tag{5.2}$$

Deoarece graful G este saturat rezultă că graful $\tilde{G} = \left(N, \tilde{A}\right)$, $\tilde{A} = A \cup \{[x_i, x_j]\}$ conţine un ciclu hamiltonian şi folosind raţionamentul din demonstraţia Teoremei 5.9, se obţine:

$$\rho_i + \rho_j \leq n - 1 \tag{5.3}$$

Din (5.2) şi (5.3) rezultă $\rho_i \leq n - 1 - \rho_j \leq n - 1 - (n - i - 1) = i$.

Considerăm $k = \rho_i$ şi din $\rho_i \leq i$ rezultă $k \leq i$. Din $\rho_1 \leq \ldots \leq \rho_n$ şi $k \leq i < j$ se obţine:

$$\rho_k \leq \rho_i \leq \rho_j. \tag{5.4}$$

Din (5.3) şi (5.4) rezultă $2k = 2\,\rho_i \leq n - 1 < n$ sau $k < n/2$. Deci $\rho_k \leq k < n/2$. Conform ipotezei şi (5.3) se obţine $\rho_{n-k} \geq n - k = n - \rho_i \geq \rho_j + 1$ şi

$$\rho_{n-k} \geq \rho_j + 1. \tag{5.5}$$

Dacă $n - k \leq j$ atunci $\rho_{n-k} \leq \rho_j$ care contrazice (5.5).

Dacă $n - k > j$ atunci $\rho_i < n - j$ care contrazice (5.1).

Deci presupunerea făcută este falsă.

(3) Demonstrăm că în acest caz este verificată condiţia (2). Fie un nod x_k cu $\rho_k \leq k < n/2$. Pot exista cazurile:

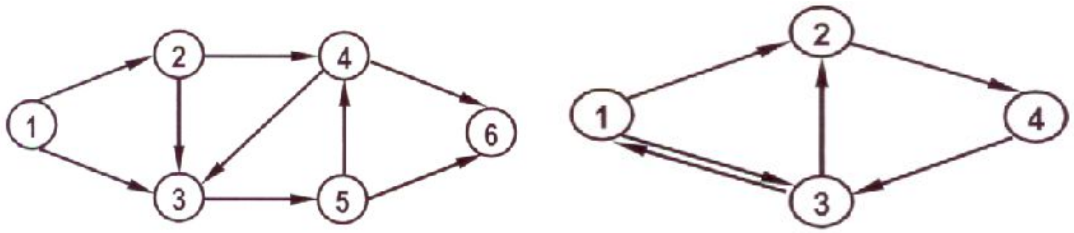
(c_1) $\rho_{n-k} \geq n - k$; în acest caz este verificată condiţia (2);

(c_2) $\rho_{n-k} < n - k$; în acest caz, din ipoteză se obţine $\rho_k + \rho_{n-k} \geq n$, deci $\rho_{n-k} \geq n - \rho_k \geq n - k$ şi acest caz nu este posibil. ■

Definiţia 5.5. Într-un digraf $G = (N, A)$ se numeşte *drum* (respectiv *circuit*) *hamiltonian* un drum (respectiv un circuit) elementar care trece prin toate nodurile diagrafului G.

Definiţia 5.6. Un graf neorientat (respectiv orientat) $G = (N, A)$ se numeşte *hamiltonian* dacă conţine un ciclu (respectiv circuit) hamiltonian.

Exemplul 5.7. Digraful din figura 5.8(a) conţine drumul hamiltonian $D = (1, 2, 3, 5, 4, 6)$ dar nu conţine nici un circuit hamiltonian. Digraful din figura 5.8(b) conţine circuitul hamiltonian $\overset{\circ}{D}= (1, 2, 4, 3, 1)$.



(a) (b)

Fig.5.8

Teorema 5.12. *Fie $G = (N, A)$ un digraf tare conex cu n noduri. Dacă pentru oricare două noduri neadiacente x şi y există inegalitatea $\rho(x)+\rho(y) \geq 2n-1$, atunci G conţine circuit hamiltonian.*
Demonstraţie. O demonstraţie a acestei teoreme poate fi găsită în bibliografia indicată. ■

Teorema 5.13. *Fie $G = (N, A)$ un digraf tare conex cu n noduri. Dacă G este complet atunci G conţine circuit hamiltonian.*
Demonstraţie. Deoarece digraful $G = (N, A)$ este complet rezultă că nu există noduri neadiacente şi conform Teoremei 5.12 conţine circuit hamiltonian. ■

Teorema 5.14. *Fie $G = (N, A)$ un diagraf tare conex cu n noduri. Dacă pentru orice nod x există inegalitatea $\rho(x) \geq n$, atunci G conţine circuit hamiltonian.*
Demonstraţie. Dacă pentru oricare nod x avem $\rho(x) \geq n$, atunci pentru oricare două noduri x, y avem $\rho(x) + \rho(y) \geq 2n > 2n - 1$ şi conform Teoremei 5.12 G conţine cirucit hamiltonian. ■

Teorema 5.15. *Fie $G = (N, A)$ un digraf cu n noduri. Dacă pentru oricare două noduri neadiacente x_i şi x_j există inegalitatea $\rho(x_i)+\rho(x_j) \geq 2n-3$, atunci G conţine drum hamiltonian.*
Demonstraţie. Fie $N = \{x_1, \ldots, x_n\}$ şi $\tilde{G} = \left(\tilde{N}, \tilde{A}\right)$, $\tilde{N} = N \cup \{x_{n+1}\}$, $\tilde{A} = A \cup \{(x_i, x_{n+1}), (x_{n+1}, x_i)$
$| \, i = 1, \ldots, n\}$. Evident că $\tilde{G}$ este un diagraf tare conex şi oricare două noduri neadiacente x_i şi x_j din $\tilde{G}$ sunt neadiacente şi în G. Deci $\tilde{\rho}(x_i)+\tilde{\rho}(x_j) = \rho(x_i)+2+\rho(x_j)+2 \geq 2n - 3 + 4 = 2(n + 1) - 1$. Din Teorema 5.12 rezultă că $\tilde{G}$ conţine circuit hamiltonian, care evident induce în G un drum hamiltonian.■

Teorema 5.16. *Fie $G = (N, A)$ un digraf cu n noduri. Dacă G este complet, atunci G conţine drum hamiltonian.*
Demonstraţie. Deoarece G este complet rezultă că nu există noduri neadiacente şi conform Teoremei 5.15 G conţine drum hamiltonian. ■

Teorema 5.17. *Fie $G = (N, A)$ un digraf cu n noduri. Dacă pentru orice nod x_i există inegalitatea $\rho(x_i) \geq n - 1$, atunci G conţine un drum hamiltonian.*

Demonstraţie. Dacă pentru orice nod x_i există inegalitatea $\rho(x_i) \geq n-1$, atunci pentru oricare două noduri x_i, X_J avem $\rho(x_i)+\rho(x_j) \geq 2n-2 > 2n-3$ şi conform Teoremei 5.15 G conţine drum hamiltonian. ■

Problemele hamiltoniene la fel ca multe alte probleme combinatorii aparţin clasei problemelor NP- complete (nedeterminist polinomial complete). Problemele din această clasă au aceeaşi complexitate algoritmică şi nu se cunoaşte nici un algoritm polinomial de rezolvare a oricăreia din problemele acestei clase. În plus, în momentul când se va descoperi un algoritm polinomial de rezolvare a oricăreia din problemele clasei NP- complete, va exista algoritm polinomial de rezolvare pentru toate celelalte probleme. Pentru rezolvarea unei probleme hamiltoniene se poate utiliza metoda backtracking sau metoda branch and bound. În continuare, pentru rezolvarea unei probleme hamiltoniene, se prezintă un algoritm care nu utilizează nici metoda backtracking, nici metoda branch and bound.

Să considerăm problema determinării tuturor drumurilor hamiltoniene într-un digraf $G=(N,A)$ cu $N=(1,\ldots,n)$. Algoritmul rezolvării acestei probleme este următorul:

(1) PROGRAM DRUMH;
(2) BEGIN
(3) FOR $i := 1$ TO n DO
(4) FOR $j := 1$ TO n DO
(5) IF $(i,j) \in A$
(6) THEN BEGIN $s_{ij}^{(1)} := \{j\}$; $s_{ij}^{(2)} := \{ij\}$; $q_{ij}^{(2)} := 1$; END
(7) ELSE BEGIN $s_{ij}^{(1)} := \emptyset$; $s_{ij}^{(2)} := \emptyset$; $q_{ij}^{(2)} := 0$; END;
(8) FOR $r := 2$ TO $n-1$ DO
(9) FOR $i := 1$ TO n DO
(10) FOR $j := 1$ TO n DO
(11) BEGIN
(12) $s_{ij}^{(r+1)} := \emptyset$;
(13) FOR $k := 1$ TO n DO
(14) BEGIN
(15) $s_{ik}^{(r+1)} := \emptyset$;
(16) IF $s_{ik}^{(r)} \neq \emptyset$ AND $s_{kj}^{(1)} \neq \emptyset$
(17) THEN BEGIN
(18) FOR $\ell := 1$ TO $q_{ik}^{(r)}$ DO
(19) IF $s_{ik\ell}^{(r)} \cap s_{kj1}^{(1)} = \emptyset$
(20) THEN BEGIN $s_{ik\ell}^{(r+1)} := s_{ik\ell}^{(r)} s_{kj1}^{(1)}$;
$s_{ik}^{(r+1)} := s_{ik}^{(r+1)} \cup \{s_{ik\ell}^{(r+1)}\}$;END;
(21) $s_{ij}^{(r+1)} := s_{ij}^{(r+1)} \cup s_{ik}^{(r+1)}$;
(22) END;
(23) END;
(24) $q_{ij}^{(r+1)} := \left| s_{ij}^{(r+1)} \right|$;
(25) END;
(26) END;

În liniile (3) la (7) se definesc matricele cu elemente mulţimi de secvenţe de noduri:

$$S^{(1)} = \left(s_{ij}^{(1)}\right)_{n\times n}, \qquad s_{ij}^{(1)} := \begin{cases} \{j\}, & \text{dacă} \quad (i,j) \in A, \\ \emptyset, & \text{dacă} \quad (i,j) \notin A, \end{cases}$$

$$S^{(2)} = \left(s_{ij}^{(2)}\right)_{n\times n}, \qquad s_{ij}^{(2)} := \begin{cases} \{ij\}, & \text{dacă} \quad (i,j) \in A, \\ \emptyset, & \text{dacă} \quad (i,j) \notin A, \end{cases}$$

şi matricea $Q^{(2)} = \left(q_{ij}^{(2)}\right)_{n\times n}$ cu $q_{ij}^{(2)}$ egal cu numărul de elemente ale lui $s_{ij}^{(2)}$.

În liniile (8) la (23) se calculează $n-2$ produse de concatenare a două matrice. Aceste produse sunt:

$$S^{(r+1)} = S^{(r)} * S^{(1)}, \text{ pentru } r = 2, \ldots, n-1.$$

Matricea $S^{(r+1)}$ este de forma:

$S^{(r+1)} = \left(s_{ij}^{(r+1)}\right)_{n\times n}$, $\qquad s_{ij}^{(r+1)} := \begin{cases} \{s_{ij1}^{(r+1)}, \ldots, s_{ijt}^{(r+1)}\}, & t = q_{ij}^{r+1} \\ \emptyset \end{cases}$

unde: $s_{ij}^{(r+1)} = \cup_{k=1}^{n} s_{ik}^{(r+1)}$, $\quad s_{ik}^{(r+1)} = \cup_{\ell=1}^{p} \{s_{ik\ell}^{(r+1)}\}$, $p = q_{ik}^{(r)}$,

$$s_{ik\ell}^{(r+1)} := \begin{cases} s_{ik\ell}^{(r)}\, s_{kj1}^{(1)}, & \text{dacă} \quad s_{ik\ell}^{(r)} \cap s_{kj1}^{(1)} = \emptyset \\ \emptyset, & \text{în caz contrar.} \end{cases}$$

Deci $s_{ij}^{(r+1)} = \cup_{k=1}^{n} \cup_{\ell=1}^{p} \{s_{ik\ell}^{(r)}\, s_{kj1}^{(1)}\}$, $p = q_{ik}^{(r)}$, $r = 2, \ldots, n-1$, $i = 1, \ldots, n$, j=1,. . . ,n.

Expresia $s_{ik\ell}^{(r)}\, s_{kj1}^{(1)}$ reprezintă concatenarea secvenţei $s_{ik\ell}^{(r)}$ cu secvenţa $s_{kj1}^{(1)}$. În linia (24) se calcuelază elementele $q_{ij}^{(r+1)}$ ale matricei $Q^{(r+1)}$.

La terminarea algoritmului se obţine matricea

$$S^n = \left(s_{ij}^{(n)}\right)_{n\times n}.$$

Dacă $s_{ij}^{(n)} = \emptyset$,, atunci nu există drum hamiltonian între nodurile i şi j. Dacă $s_{ij}^{(n)} = \{s_{ij1}^{(n)}, \ldots, s_{ijt'}^{(n)}\}$, atunci există t' drumuri hamiltoniene între nodurile i şi j. Aceste drumuri sunt date de secvenţele de noduri $s_{ij1}^{(n)}, \ldots, s_{ijt'}^{(n)}$.

Obsevaţia 5.7. Dacă s-au obţinut toate drumurile hamiltoniene din digraful $G = (N, A)$, atunci se pot determina toate circuitele hamiltoniene din G în modul următor: calculăm matricea $S^{(n+1)}$, în care acceptăm ca în fiecare secvenţă de noduri primul nod să coincidă cu ultimul nod; elementele de pe diagonala principală reprezintă circuitele hamiltoniene din G. Algoritmul prezentat mai sus se poate aplica şi grafurilor neorientate pentru determinarea lanţurilor şi ciclurilor hamiltoniene.

Exemplul 5.8. Fie diagraful reprezentat în figura 5.9. Să se determine cu ajutorul algoritmului prezentat mai sus toate drumurile şi circuitele hamiltoniene din acest digraf.

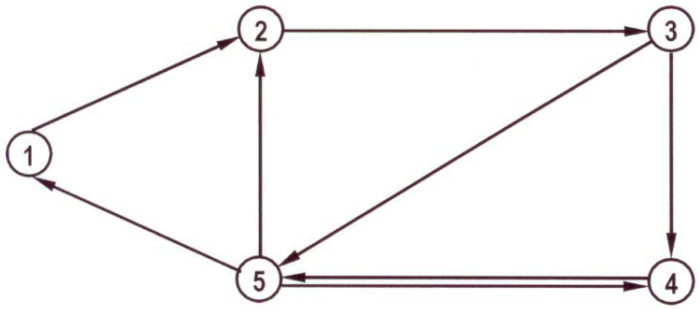


Fig.5.9

$$S^{(1)} = \begin{bmatrix} \emptyset & \{2\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{3\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{4\} & \{5\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{5\} \\ \{1\} & \{2\} & \emptyset & \{4\} & \emptyset \end{bmatrix}, \quad S^{(2)} = \begin{bmatrix} \emptyset & \{12\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{23\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{34\} & \{35\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{45\} \\ \{51\} & \{52\} & \emptyset & \{54\} & \emptyset \end{bmatrix},$$

$$\mathrm{S}^{(3)} = \begin{bmatrix} \emptyset & \emptyset & \{123\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{234\} & \{235\} \\ \{351\} & \{352\} & \emptyset & \{354\} & \{345\} \\ \{451\} & \{452\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \{512\} & \{523\} & \emptyset & \emptyset \end{bmatrix},$$

$$\mathrm{S}^{(4)} = \begin{bmatrix} \emptyset & \emptyset & \emptyset & \{1234\} & \{1235\} \\ \{2351\} & \emptyset & \emptyset & \{2354\} & \{2345\} \\ \{3451\} & \left\{ \begin{matrix} 3512 \\ 3452 \end{matrix} \right\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \{4512\} & \{4523\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \{5123\} & \{5124\} & \emptyset \end{bmatrix},$$

$$\mathrm{S}^{(5)} = \begin{bmatrix} \emptyset & \emptyset & \emptyset & \{12354\} & \{12345\} \\ \{23451\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \{34512\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{45123\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{51234\} & \emptyset \end{bmatrix},$$

$$\mathrm{S}^{(6)} = \begin{bmatrix} \{123451\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \{234512\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{345123\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{451234\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{512345\} \end{bmatrix}$$

Digraful din figura 5.9 conţine şase drumuri hamiltoniene şi un circuit hamiltonian.

5.3 Aplicaţii

5.3.1 Problema poştaşului

Un poştaş trebuie să distribuie poşta într-o reţea de străzi şi să se întoarcă la oficiul poştal cât mai rapid posibil.

Această problemă se poate modela ca o reţea neorientată $G = (N, A, b)$ unde mulţimea nodurilor N reprezintă intersecţia străzilor, mulţimea muchiilor A reprezintă străzile şi funcţia valoare $b : A \to \mathcal{R}$ reprezintă fie funcţia lungime, fie funcţia timp.

Ţinând cont de precizările de mai sus, problema poştaşului are următoarea formulare: să se determine în reţeaua neorientată $G = (N, a, b)$ un ciclu care trece prin fiecare muchie cel puţin o dată şi are valoarea minimă.

Dacă fiecare nod al grafului G are gradul par, atunci orice ciclu eulerian rezolvă problema poştaşului, altfel anumite muchii trebuie să fie reparcurse. Evident că un ciclu al poştaşului în graful G corespunde la un ciclu eulerian într-un multigraf neorientat $G^* = (N, A^*)$, cu $A^* = A \oplus A''$, unde $A'' \subset A$ şi $\oplus$ reprezintă operaţia de adăugare. Deci A^* nu este o mulţime ci o familie de muchii.

Un cuplaj în graful neorientat $G' = (N', M')$ este o submulţime de muchii M'' cu proprietatea că oricare două muchii din M'' nu sunt adiacente. Un cuplaj M'' al grafului G' se numeşte *perfect* dacă pentru fiecare nod x din N' există $[x, y] \in M''$.

Fie submulţimea de noduri $N' = \{x \,|\, x \in N, \rho(x) \text{ impar}\}$, distanţele $d'(x, y)$ şi lanţurile minime $L'(x, y)$ în reţeaua $G = (N, A, b)$ pentru $x, y \in N'$. Se construiesc graful neorientat complet $G' = (N', M')$ şi reţeaua $G' = (N', M', b')$ cu $b'[x, y] = d'[x, y,], [x, y] \in M'$. În reţeaua $G' = (N', M', b')$ se determină un cuplaj perfect M'' cu valoarea minimă. La fiecare muchie $[x, y] \in M''$ corespunde un lanţ minim $L'(x, y)$ în G. Un astfel de lanţ îl vom nota în continuare prin $L''(x, y)$ şi fie $\mathcal{L}'' = \{L''(x, y) \,|\, [x, y] \in M''\}$. Se determină mulţimea $A'' = \{[x, y] \,|\, [x, y] \in L''(x, y),\ L''(x, y) \in \mathcal{L}''\}$ şi se construieşte multigraful neorientat $G^* = (N, A^*)$ cu $A^* = A \oplus A''$.

Algoritmul PP, care rezolvă problema poştaşului, este prezentat în PROGRAM PP ce conţine şase proceduri.

```
(1) PROGRAM PP;
(2) BEGIN
(3)     DLM(G, N', D'_1, L');
(4)     REC(N', D'_1, M');
(5)     CPM(G', D'_1, M'');
(6)     LMC(L', M'', L'');
(7)     MUN(G, L'', G*);
(8)     CIE(G*, C*);
(9) END.
```

Cele şase proceduri determină următoarele:
DLM determină submulţimea de noduri N', distanţele $d'(x, y) \in \mathcal{D}'_1$ şi lanţurile minime $L'(x, y) \in \mathcal{L}'$ în reţeaua $G = (N, A, b)$ pentru $x, y \in N'$;
REC determină reţeaua completă $G' = (N', M', b')$;

CPM determină cuplajul perfect minim M'' din reţeaua G';
LMC determină lanţurile minime $L''(x,y) \in \mathcal{L}''$ corespunzătoare muchiilor din cuplajul perfect minim M'';
MUN determină multigraful neorientat $G^* = (N, A^*)$, $A^* = A \oplus A''$;
CIE determină un ciclu eulerian C^* în multigraful G^*.

Corectitudinea algoritmului PP este evidentă.

Teorema 5.18. *Algoritmul PP are complexitatea* $O(n^3)$.

Demonstraţie. Fiecare procedură utilizată în algoritm are complexitatea cel mult $O(n^3)$. Deci algoritmul PP are complexitatea $O(n^3)$. ■

Exemplul 5.8. Fie reţeaua neorientată din figura 5.10.

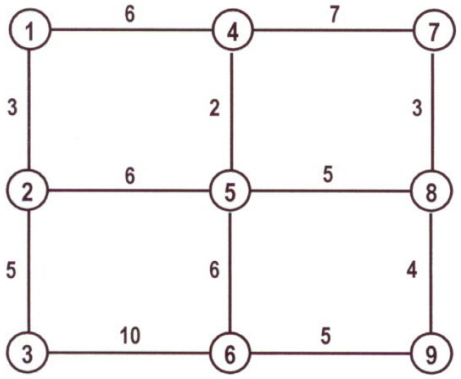


Fig.5.10

Procedura DLM determină: $N' = \{2,4,6,8\}$, $\mathcal{D}'_1 = \{d'(2,4) = 8,\ d'(2,6) = 12,\ d'(2,8) = 11,\ d'(4,6) = 8,\ d'(4,8) = 7,\ d'(6,8) = 9\}$, $\mathcal{L}' = \{L'(2,4) = (2,5,4),\ L'(2,6) = (2,5,6),\ L'(2,8) = (2,5,8),\ L'(4,6) = (4,5,6),\ L'(4,8) = (4,5,8),\ L'(6,8) = (6,9,8)\}$. Procedura REC determină reţeaua reţeaua completă $G' = (N', M', b')$ reprezentată în figura 5.11.

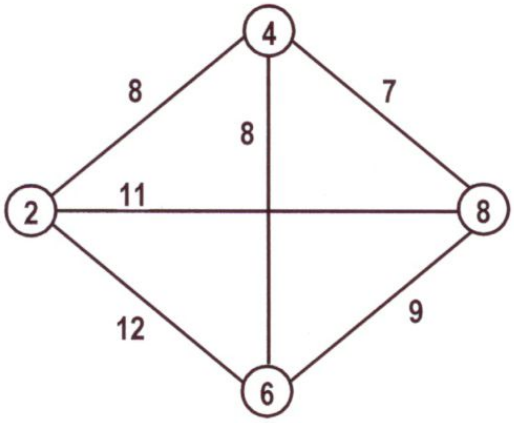


Fig.5.11

Procedura CPM determină cuplajul perfect minim $M'' = \{[2,4],[6,8]\}$ în G'.

Procedura LMC determină $\mathcal{L}'' = \{L''(2,4) = (2,5,4),\ L''(6,8) = (6,9,8)\}$.

Procedura MUN construieşte multigraful neorientat $G^* = (N,\ A^*)$ reprezentat în figura 5.12.

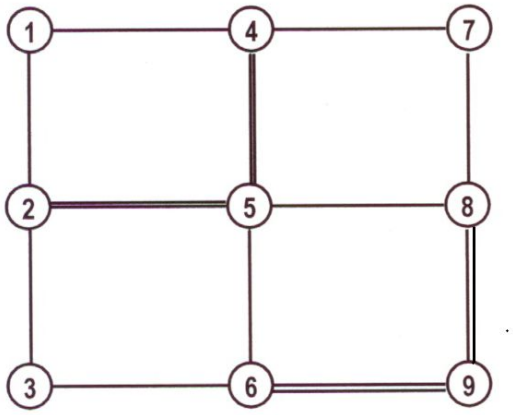


Fig.5.12

Procedura CIE determină circuitul eulerian $C^* = (1,4,7,8,5,4,5,2,5,6,9,8,9,6,3,2,1)$ în multigraful neorientat $G^* = (N, A^*)$.

Remarcăm faptul că pentru rezolvarea procedurii CPM sunt necesare şi alte noţiuni şi rezultate care nu au fost prezentate în această lucrare. De aceea, în acest subparagraf considerăm că un cuplaj perfect minim M'' din reţeaua G' este dată de intrare. Cititorii interesaţi pot consulta bibliografia indicată.

5.3.2 Problema comis-voiajorului

Un comis voiajor (voiajor commercial) trebuie să treacă prin n oraşe, după care trebuie să se întoarcă în oraşul din care a plecat iniţial. Se cere să se găsească ruta cea mai scurtă, sau cea mai ieftină etc. Această problemă se poate reprezenta pe o reţea neorientată $G = (N, A, b)$ cu mulţimea nodurilor N reprezentând cele n oraşe, mulţimea muchiilor A reprezentând legăturile dintre oraşe şi funcţia valoare b reprezentând fie lungimea, fie costul etc. Astfel, problema comis voiajorului revine în determinarea unui ciclu care să treacă prin fiecare nod al reţelei şi să fie cu valoare minimă.

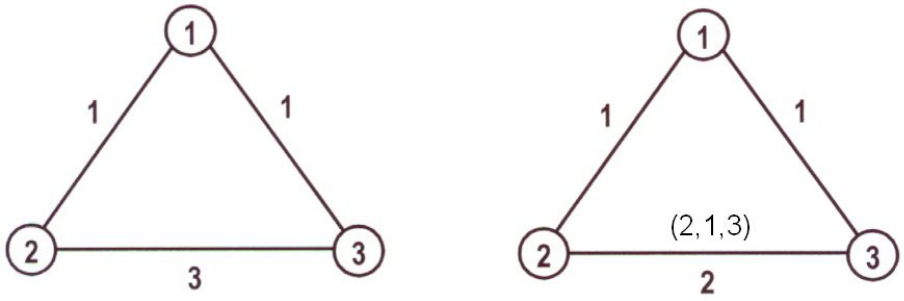
Există mai multe formulări pentru problema comis voiajorului. O primă formulare cere să se treacă prin fiecare nod o singură dată şi o a doua formulare cere să se treacă prin fiecare nod cel puţin o dată.

Dacă pentru fiecare pereche de noduri x şi y a reţelei $G = (N, A, b)$ este verificată condiţia $b[x,y] \leq b[x,z] + b[z,y]$ pentru oricare nod $z \neq x$, $z \neq y$, atunci se spune că este verificată inegalitatea triunghiului în G.

Pentru reţelele în care inegalitatea triunghiului este verificată problema comis voiajorului are prima formulare. În caz contrar problema comis voiajorului are a doua formulare.

Problema comis voiajorului într-o reţea oarecare $G = (N, A, b)$ se poate converti într-o problemă a ciclului hamiltonian minim într-o reţea $G' = (N', A', b')$, unde $N' = N$, $A' = \{[x, y] \,|\, x, y \in N'\}$, $b'[x, y] = d(x, y)$, $x, y \in N$. Rezultă că reţeaua G' are ca support un graf complet cu valorile muchiilor egale cu distanţele calculate în reţeaua G. Evident că în reţeaua G' este verificată inegalitatea triunghiului şi că fiecare muchie corespunde la un lanţ minim format din una sau mai multe muchii din G. În construcţia lui G' este uzual să se eticheteze fiecare muchie cu lanţul corespunzător din G, dacă lanţul este format din cel puţin două muchii.

Exemplul 5.9. Reţelei $G = (N, a, b)$ din figura 5.13(a) îi corespunde reţeaua $G' = (N', A', b')$ din figura 5.13(b).



(a) (b)

Fig.5.13

Teorema 5.19. *O soluţie a problemei comis voiajorului în reţeaua $G = (N, a, b)$ corespunde unui ciclu hamiltonian minim în reţeaua $G' = (N', A', b')$.*

Demonstraţie. Presupunem că C este o soluţie pentru problema comis voiajorului în reţeaua G care nu este echivalentă cu un ciclu hamiltonian în reţeaua G'. Fie C' ciclul echivalent în G'. Notăm că ciclul C' trebuie să urmeze aceeaşi secvenţă de muchii în G' ca ciclul C în G ţinând seama că fiecare muchie $[x, y]$ din G' corespunde la un lanţ minim $L^*_{xy} = (x, \ldots, y)$ în G. Deoarece $b'[x, y] = b\left(L^*_{xy}\right)$ rezultă că $b(C') = b(C)$. Ciclul C' este un ciclu care trece prin fiecare nod x cel puţin o dată.

Presupunem că ciclul C' vizitează un anumit nod y a doua oară. Fie nodurile x şi z vizitate imediat înaintea şi respective imediat după vizitarea nodului y. Putem înlocui sublanţul (x, y, z) din C' prin muchia $[x, z]$ fără a afecta faptul că C' este echivalentul soluţiei problemei comis voiajorului în G, deoarece $b'[x, z] = d(x, z) = b(x, y) + b(y, z)$. În acest mod se poate elimina oricare vizitare multiplă pentru orice nod y din G'. Astfel C' devine un ciclu hamiltonian C''. Deoarece $b(C'') = b(C') = b(C)$ rezultă că C'' este un ciclu hamiltonian minim. Deci contrazicem ipoteza. ■

Conform acestei teoreme, în continuare considerăm problema ciclului hamiltonian minim într-o reţea neorientată şi completă $G = (N, A, b)$ în care este verificată inegalitatea triunghiului.

Până în prezent nu se cunoaşte un algoritm polinomial care să determine un ciclu hamiltonian. O asemenea problemă, pentru a cărei rezolvare nu se cunosc algoritmi polinomiali iar algoritmii cunoscuţi sunt exponenţiali se numeşte problemă NP - completă. Problema determinării unui ciclu hamiltonian este NP - completă.

Pentru determinarea unui ciclu hamiltonian minim se vor prezenta algoritmi de aproximare. Fie C^* un ciclu hamiltonian minim şi C un ciclu Hamiltonian cu $b(C^*) \leq b(C)$. Se cere unui algoritm de aproximare să satisfacă condiţia:

$$1 \leq b(C)/b(C^*) \leq r.$$

Cu cât r este mai mic cu atât algoritmul este mai performant.

Algoritmul pe care îl pezentăm mai jos se va numi algoritmul dublu arbore (DA).

(1) PROGRAM DA;
(2) BEGIN
(3) APM(G, G');
(4) PDF(G', o');
(5) CHA (o$', C$);
(6) END.

Procedurile din algoritm determină următoarele:
APM determină un arbore parţial minim G' al grafului G;
PDF determină tabloul ordine o' în parcurgerea DF a arborelui G';
CHA determină ciclul hamiltonian C cu $b(C^*) \leq b(C)$ şi $C = (x_1, \ldots, x_n, x_1)$ dacă $o'(x_i) = i$, $i = 1, \ldots, n$.

Teorema 5.20. *Pentru algoritmul DA avem $r < 2$.*

Demonstraţie. Mai întâi observăm că $b(G') < b(C^*)$, deoarece un arbore parţial G'' se poate obţine prin eliminarea oricărei muchii din ciclul hamiltonian minim C^*. Parcurgerea DF a lui G' generează un ciclu C' care traversează fiecare muchie a lui G' de două ori. Deci $b(C') = 2b(G') < 2b(C^*)$. Ciclul C determinat de algoritm urmează ciclul C' exceptând faptul că C continuă direct la următorul nod nevizitat fără revizitarea unor noduri. Deci numărul de muchii ale ciclului C este mai mic decât numărul de muchii ale ciclului C'. Din această remarcă şi din faptul că în G este verificată inegalitatea triunghiului avem că $b(C) < b(C')$. Rezultă că $b(C) < 2b(C^*)$ sau $b(C)/b(C^*) < 2$. ■

Teorema 5.21. *Algoritmul DA are complexitatea $O(n^2)$.*

Demonstraţie. Complexitatea procedurii APM este $O(n^2)$ dacă se utilizează algoritmul Prim. Procedura PDF are complexitatea $O(m') = O(n)$. Complexitatea procedurii CHA este $O(n)$. Deci algoritmul are complexitatea $O(n^2)$. ■

Exemplul 5.10. Fie reţeaua neorientată şi completă $G = (N, A, b)$ reprezentată în figura 5.14(a). Se poate constata că este verificată inegalitatea triunghiului.
Procedura APM determină arboreal parţial minim G' din figura 5.14(b).
Procedura PDF determină tabloul $o' = (1, 2, 3, 4, 5, 6)$.
Procedura CHA determină ciclul hamiltonian $C = (1, 2, 3, 4, 5, 6, 1)$. Remarcăm că ciclul C' din demonstraţia Teoremei 5.19 este $C' = (1, 2, 3, 2, 4, 2, 1, 5, 1, 6, 1)$ şi $b(C) = 2 + 1 + 4 + 4 + 2 + 1 = 14$. Ciclul hamiltonian minim este $C^* = (1, 5, 2, 3, 4, 6, 1)$ cu $b(C^*) = 12$. Dacă se elimină muchia $[3, 4]$ se obţine un arbore parţial G'' cu $b(G'') = b(G')$.

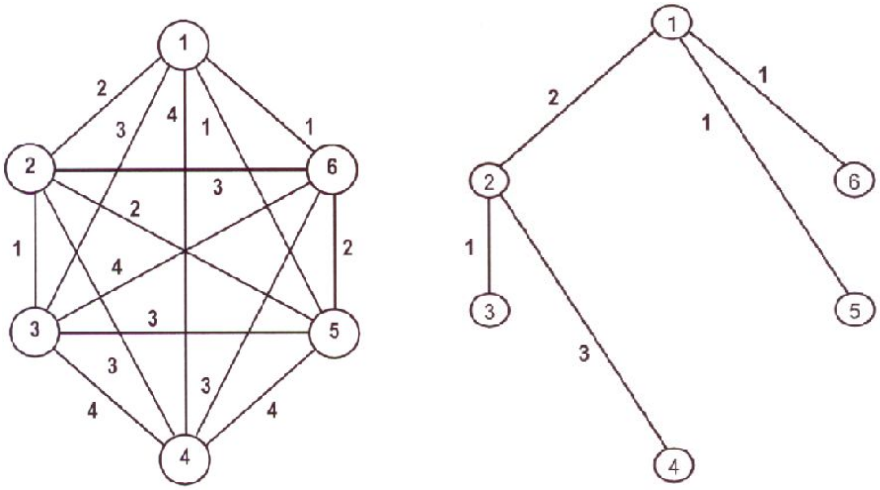


Fig.5.14

Înainte de a prezenta al doilea algoritm de aproximare vom face câteva precizări. Ca în algoritmul anterior notăm cu $G' = (N, A')$ arborele parţial minim al grafului G. Fie mulţimea de noduri $N'' = \{x \,|\, x \in N,\ \rho'(x) \text{ impar în } G'\}$ şi $G'' = (N'',\ A'')$ subgraful indus de N'' în G. Notăm cu M'' cuplajul perfect minim în G''. Evident că graful $\tilde{G}' = \left(N,\ \tilde{A}'\right)$ cu $\tilde{A}' = A' \oplus M''$ este un graf eulerian. Dacă $\tilde{C}'$ este un ciclu eulerian în $\tilde{G}'$ şi $\tilde{o}'$ este tabloul ordine în parcurgerea acestui ciclu, atunci se poate construi ciclul hamiltonian $C = (x_1, \ldots, x_n, x_1)$, unde $o'(x_i) = i$, $i = 1, \ldots, n$.

Algoritmul arborelui şi cuplajului (AC) este prezentat mai jos .

(1) PROGRAM AC;
(2) BEGIN
(3) APM(G, G');
(4) SGR(G, G', G'');
(5) CPM(G'', M'');
(6) GRE$\left(G', M'', \tilde{G}'\right)$;
(7) CIE$\left(\tilde{G}', \tilde{C}', \tilde{o}'\right)$;
(8) CHA $\left(\tilde{C}', \tilde{o}', C\right)$;
(9) END.

Procedurile din program au semnificaţiile următoare:
APM determină arborele parţial minim $G' = (N, A')$ al lui $G = (N, A, b)$;
SGR determină subgraful $G'' = (N'',\ A'')$ al lui G;
CPM deetrmină cuplajul perfect minim M'' în G'';
GRE determină graful eulerian $\tilde{G}'$;
CIE determină ciclul eulerian $\tilde{C}'$ în $\tilde{G}'$ şi tabloul ordine $\tilde{o}'$;
CHA determină ciclul hamiltonian C din $\tilde{C}'$.

Teorema 5.22. *Pentru algoritmul AC avem* $r < 3/2$.
Demonstraţie. Analog ca în Teorema 5.20 avem că $b(G') < b(C^*)$ şi $b(C) < b(\tilde{C}')$. Dar $b(\tilde{C}') = b(G') + b(M'')$ şi se obţine $b(C) < b(C^*) + b(M'')$. Din ciclul C^* se poate construi un ciclu C'' care trece numai prin nodurile din N''. Datorită faptului că C'' are mai puţine muchii decât C^* şi din inegalitatea triunghiului avem $b(C'') < b(C^*)$. Deoarece $|N''| = 2k$ se pot construi două cuplaje perfecte M_1'' şi M_2'' din C'' prin alternarea muchiilor. Este evident că $b(M'') \leq \min\{b(M_1''), b(M_2'')\} \leq \frac{1}{2}b(C^*)$. Rezultă $b(C) < b(C^*) + b(M'') < b(C^*) + \frac{1}{2}b(C^*) = \frac{3}{2}b(C^*)$. Prin urmare $b(C)/b(C^*) < \frac{3}{2}$. ■
Teorema 5.23. *Algoritmul AC are complexitatea* $O(n^3)$.
Demonstraţie. Fiecare procedură utilizată de algoritm are complexitatea cel mult $O(n^3)$. Deci algoritmul AC are complexitatea $O(n^3)$. ■
Exemplul 5.11. Fie reţeaua neorientată şi completă $G = (N, A, b)$ reprezentată în figura 5.14(a).
Procedura APM determină arborele parţial minim G' din figura 5.14(b);
Procedura SGR determină subgraful $G'' = G$, deoarece $N'' = N$;
Procedura CPM determină cuplajul perfect minim $M'' = \{[1,5],[2,3],[4,6]\}$;
Procedura GRE determină graful eulerian $\tilde{G}'$ din figura 5.15;

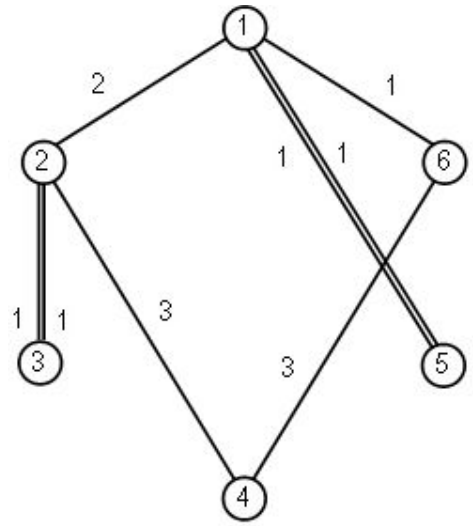


Fig.5.15

Procedura CIE determină ciclul eulerian $\tilde{C}' = (1,5,1,2,3,2,4,6,1)$ şi tabloul ordine $\tilde{o}' = (1,5,2,3,4,6)$;

Procedura CHA determină ciclul hamiltonian $C = (1,5,2,3,4,6,1)$ cu $b(C) = b(1,5) + b(5,2) + b(2,3) + b(3,4) + b(4,6) + b(6,1) = 1 + 2 + 1 + 4 + 3 + 1 = 12$. În acest caz $C = C^*$.

Remarcăm faptul că pentru rezolvarea procedurii CPM sunt necesare şi alte noţiuni şi rezultate care nu au fost prezentate în această lucrare. De aceea, în acest subparagraf considerăm că un cuplaj perfect minim M'' din reţeaua G'' este dată de intrare. Cititorii interesaţi pot consulta bibliografia indicată.

Bibliografie

Ahuja, R. K., Magnanti, T. L. and Orlin, J. B. (1993): *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, Englewood, Cliffs, N. J.

Bang-Jensen, J. and Gutin, G. (2001):*Digraph: Theory, Algorithms and Applications.* Springer-Verlag, London.

Bazaraa, M. S., Jarvis J. J. and Sherali, H. D. (2005): *Linear Programming and Network Flows (third edition).* Wiley, New York.

Berge, C. (1973): *Graphs and Hypergraphs.* North Holland, Amsterdam.

Bertsekas, D. P. (1991): *Liniar Network Optimization: Algorithms and Codes.* TheMIT Press Cambridge, Massachusetts.

Chen, W. K. (1990): *Theory of Nets: Flows in Networks.* Wiley, New York.

Christofides, N. (1975): *Graph Theory: An Algorithmic Approach.* Academic Press, New York.

Ciupală L. 2007 *Algoritmi fundamentali din teoria grafurilor. Aplicaţii.* Editura Universităţii Transilvania Braşov.

Ciurea, E (2001): *Introducere în algoritmica grafurilor.* Editura Tehnică Bucureşti.

Ciurea, E., Ciupală, L. (2006): *Algoritmi. Introducere în algoritmica fluxurilor în reţele.* Editura Matrix Rom, Bucureşti.

Croitoru, C. (1992): *Tehnici de bază în optimizarea combinatorie.* Editura Universităţii 'Al. I. Cuza', Iaşi.

Evans, J. R. and Minieka, E. (1992): *Optimization Algorithms for Networks and Graphs.* Marcel Dekker, Inc., New York.

Ford, L. R. and Fulkerson, D. R. (1962): *Flows in Networks.* Princeton University Press, Princeton, N. J.

Gibbons, A. (it Algorithmic Graphs, Networks and Algorithms). Springer, Berlin, 1999.

Glover, F., Klingman, D. and Philips, N. V. (1992): *Network Models in Optimization and their Applications in Practice.* Wiley, New York.

Godran, M. and Minoux, N. *Graphs and Algorithms.* Wiley, New York, 1984.

Gross, J. and Yellen, J. (1999): *Graph Theory and its Applications*, CRC Press, New York.

Jungnickel, D. (1999): *Graphs, Networks and Algorithms.* Springer, Berlin.

Rühe, G. (1991): *Algorithmic Aspects of Flows in Networks.* Kluwer Academic Publishers, Dordrecht.

Tarjan, R. E. (1983): *Data Structures and Network Algorithms.* SIAM, Philadelphia.

Toadere, T. (2002): *Grafe. Teorie, algoritmi şi aplicaţii.* Editura Albatros, Cluj-Napoca.

Tomescu, I. (1981): *Probleme de combinatorică şi teoria grafurilor.* Editura Didactică şi Pedagogică, Bucureşti.

Tutte, W. T. (1984): *Graph Theory.* Cambridge University Press, Cambridge.