GRIGOR MOLDOVAN IOAN DZIȚAC

SISTEME DISTRIBUITE

MODELE MATEMATICE

CCC Publications
Editura Universității Agora, Oradea
2006

Referent: *Acad. Florin Gheorghe Filip*, Academia Română
Editor: *Prof. univ. dr. ing. Mişu-Jan Manolescu*, Universitatea Agora
Asistent editor şi coperta: *Emma M. Văleanu*, Universitatea Agora, doctorandă la Academia Română

Descrierea CIP a Bibliotecii Naţionale a României

MOLDOVAN, GRIGOR
Sisteme distribuite - Modele matematice / Grigor Moldovan, Ioan Dziţac. - Oradea : Editura Universităţii Agora, 2006
Bibliogr.
ISBN (10) 973-88205-0-2 ; ISBN (13) 978-973-88205-0-0

I. Dziţac, Ioan

004

Titlu: SISTEME DISTRIBUITE - MODELE MATEMATICE

Autori: *Grigor Moldovan & Ioan Dziţac*

Rezumat. Această carte se adresează în special studenţilor de la specializările *Informatică* şi *Matematică- Informatică*, dar poate fi utilă oricui doreşte să abordeze sitemele distribuite dintro perspectivă matematizată. Lucrarea este, în principal, rodul experienţei acumulate de prof. univ. dr. Grigor Moldovan în predarea cursului *Sisteme distribuite* la Facultatea de Matematică şi Informatică a Universităţii Babeş – Bolyai din Cluj Napoca şi prezintă în cea mai mare parte un material complementar faţă de cartea *Sisteme distribuite – Modele informatice*[1], publicată de cei doi autori în aceaaşi editură.

Title: DISTRIBUTED SYSTEMS - MATHEMATICAL MODELS

Authors: *Grigor Moldovan & Ioan Dziţac*

Abstract. We address this book especially for the students from the Informatics and Mathematics-Informatics specializations, but can be used by anyone who wants to approach the distributive systems from a mathematical point of view. In essence, this book is a collection of the experience and course notes accumulated by prof. univ. dr. Grigor Moldovan in teaching of the *Distributed Systems* course at the Mathematics and Informatics Faculty from Babeş – Bolyai University, Cluj Napoca and presents at most a complementary material for the book *Distributed Systems – Information Models*[2], published by the two authors in the same publishing house.

[1] Dziţac, I., Moldovan, G., *Sisteme distribuite – Modele informatice*, Ed. Univ. Agora, 2006, ISBN (10) 973-87960-9-1 ; ISBN (13) 978-973-87960-9-6, 146 p.
[2] Dziţac, I., Moldovan, G., *Distributed systems – Information models*, Ed. Univ. Agora, 2006, ISBN (10) 973-87960-9-1 ; ISBN (13) 978-973-87960-9-6, 146 p.

CONTENTS

AUTORII

Prof. univ. dr. Grigor MOLDOVAN

Domenii de interes:
Sisteme distribuite, Limbaje formale

Conf. univ. dr. Ioan DZIŢAC

Domenii de interes:
Calcul paralel şi distribuit,
Matematică şi informatică economică

Grigor MOLDOVAN

(n. 29.12.1939, Vadu Izei, Maramureş), este doctor în matematică, profesor universitar şi conducător de doctorate în informatică la Universitatea „Babeş-Bolyai" din Cluj - Napoca, Facultatea de Matematică şi Informatică. A obţinut doctoratul în matematică în 1972 sub conducerea ştiinţifică a acad. Tiberiu Popoviciu şi acad. Dimitrie Stancu. Este unul din pionierii informaticii româneşti, predând cursuri de informatică începând încă din anul 1971, scriind articole şi cărţi de informatică şi fiind directorul Centrului de Calcul al Universităţii, de la înfiinţarea sa în 1975, până în anul 1992. A fost mulţi ani, decan iar, apoi rector al Şcolii Superioare de Afaceri.
A publicat 19 cărţi şi peste 70 de articole ştiinţifice.

Ioan DZIŢAC

(n. 14.02.1953, Poieni de sub Munte - Repedea, Maramureş), este doctor în informatică, conferenţiar universitar şi şeful catedrei de informatică economică la Universitatea Agora, Oradea. A obţinut doctoratul în informatică în 2002 sub conducerea ştiinţifică a prof. Grigor Moldovan. A fost directorul Departamentului de Matematică şi Informatică al Universităţii din Oradea, iar în prezent este directorul centrului de cercetare „Tehnologii informatice avansate în management şi inginerie" la Universitatea AGORA.
A publicat 14 cărţi şi a editat 4 volume ale unor conferinţe internaţionale şi 48 de articole. Este fondator şi editor executiv la *International Journal of Computers, Communications and Control* (www.journal.univagora.ro). De asemenea a fondat *International Conference on Computers, Communications and Control* (www.iccc.univagora.ro)

REFERENT ŞTIINŢIFIC

Acad. Florin Gheorghe FILIP

Domenii de interes:
Sisteme suport pentru decizii, Sisteme expert

EDITOR

Prof. univ. dr. ing. M.J. MANOLESCU

Domenii de interes:
Knowledge Management, Knowledge Engineering

Acad. Florin Gheorghe FILIP

(n. 25.07.1947, Bucureşti), este doctor în automatică, profesor universitar şi conducător de doctorate în Ştiinţa Calculatoarelor la Academia Română. A obţinut doctoratul în automatică în 1982.
În 1990 a obţinut titlul de cercetător ştiinţific I la Institutul Naţional de Cercetare- Dezvoltare în Informatică, unde a fost director (1991-1997); director-responsabil cu cooperarea internaţională (1997-2001) şi preşedintele comitetului ştiinţific(1995-2003). În 1998 a obţinut titlul didactic de profesor universitar.
Este membru activ al Academiei Române (1991 - membru corespondent, 1999 - membru titular). Din 2000 până în prezent este vicepreşedinte al Academiei Române.
Este fondator şi redactor şef al revistelor internaţionale *Studies in Informatics and Control* şi *International Journal of Computers, Communications and Control*.
A publicat 6 cărţi, a editat 9 volume şi publicat peste 200 de articole ştiinţifice în reviste prestigioase din ţară şi străinătate.

Mişu- Jan MANOLESCU

(n. 30.09.1958, Craiova), este doctor în inginerie electroenergetică (1995) şi în managementul resurselor umane (2002). A lucrat la Universitatea din Oradea]n perioada 1984-1999, unde a fost cadru didactic, şef de catedră, secretar ştiinţific pe facultate şi directorul departamentului de relaţii internaţionale. Din 2000 este fondatorul şi rectorul Universităţii Agora din Oradea. A publicat peste 10 cărţi, peste 60 de articole ştiinţifice şi este posesorul a 2 brevete invenţie. Este fondator şi director la *International Journal of Computers, Communications and Control* (www.journal.univagora.ro). De asemenea este fondator al *International Conference on Computers, Communications and Control* (www.iccc.univagora.ro)

CUPRINS

CAPITOLUL 1

SISTEME

1.1. Conceptul de sistem în general

Conceptul de sistem ne este cunoscut şi-l folosim frecvent în viaţa cotidiană. Vorbim mereu despre diferite sisteme politice, economice, filozofice şi sociale. Suntem familiarizaţi, de asemenea, cu multe sisteme particulare, precum sistemul monetar, sistemele informatice, sistemele de comunicaţii etc.

Evident, ne punem întrebarea dacă putem da definiţia generală al acestui concept. Iată două definiţii de dicţionar, din anii ’70, pentru noţiunea generală de sistem.

Definiţia 1.1.[3] Sistemul este un (n.a.) *ansamblu de elemente (principii, reguli, forţe, etc.) dependente între ele şi formînd un tot organizat, care pune ordine într-un domeniu de gîndire teoretică, reglementează clasificarea materialului într-un domeniu de ştiinţe ale naturii sau face ca o activitate practică să funcţioneze potrivit scopului urmărit.*

Definiţia 1.2.[4] (gr. *systema*- ansamblu/ total; din *synestemi*- a aşeza împreună). Sistemul este un (n.a.) *ansamblu de elemente aflate în interacţiune şi care formează împreună un tot unitar. Orice sistem acţionează pentru realizarea unei funcţionalităţi, pentru atingerea unui scop; acestea se realizează printr-un proces în care sînt angrenate elementele care determină atît conţinutul, cît şi procesele specifice sistemului. Sistemele sînt organizate în structuri ierarhice ; în acest sens putem spune că un sistem poate fi constituit din subsisteme, fiecare urmărind un scop subordonat scopului general al sistemului; sistemele se încadrează la rîndul lor în suprasisteme, pe care le slujesc. ...Pentru a fi mereu adecvat suprasistemului, sistemul se reglează. Reglarea are loc prin analiza produselor sistemului (outputlui), prin compararea lor cu parametrii obiectivelor propuse (scopului) şi introducerea schimbărilor necesare în sistem.*

Definiţiile de mai sus îşi păstrează actualitatea şi azi, deşi de atunci au apărut foarte multe sisteme noi. Pentru domenii restrânse aceste definiţii sunt mai uşor de dat şi ele au o consistenţă mult mai mare.

În cele ce urmează vom formula o definiţie cât mai generală, acceptabilă şi pentru domeniul informatic pe care-l vom aborda în continuare.

---

[3] Marcu, F., Maneca, C., Dicţionar de neologisme, Ed. Academiei, 1978

[4] Manolache, A. ş.a., Dicţionar de pedagogie, Ed. didactică şi pedagogică, Bucureşti, 1979

Definiţia 1.3. *Prin sistem, în general, înţelegem un ansamblu de obiecte care pot fi lucruri, concepte, fiinţe etc. şi care se găsesc într-o anumită interdependenţă şi interacţiune pentru realizarea unui scop comun.*

Din definiţia 1.3 rezultă că în orice sistem avem două lucruri esenţiale pe care le vom nota corespunzător:

obiecte: O - mulţimea obiectelor;

legături între obiecte: L - mulţimea legăturilor.

Având în vedere aceste notaţii un sistem S convenim să-l notăm astfel:

$$S = (O, L)$$

Reprezentările unui sistem, respectiv modelele pe care le putem alege, se bazează pe noţiuni şi concepte din matematică, precum: teoria grafelor[5], teoria mulţimilor, teoria relaţiilor. Există şi reprezentări fizice pentru anumite sisteme.

Exemplul 1.1. Fie sistemul S ale cărui obiecte formează mulţimea
$O = \{o_1, o_2, o_3, o_4\}$,
iar mulţimea legăturilor este
$L = \{o_1\text{-}o_2, o_1\text{-}o_3, o_1\text{-}o_4, o_2\text{-}o_4, o_3\text{-}o_4, o_4\text{-}o_1\}$;
unde legăturile sunt precizate prin perechi ordonate de obiecte.
Atunci $S = (O,L)$ cu precizările de mai sus este un sistem.

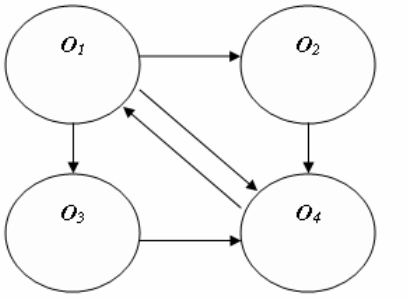


Fig.1.1. Graful sistemului din exemplul 1.1

Dacă luăm în considerare modul cum am precizat legăturile dintre obiectele unui sistem, atunci un sistem nu este altceva decât o *relaţie binară* definită pe mulţimea obiectelor O care formează acel sistem. În acest fel, un sistem este tripletul:

$$(O,O;L) \quad \text{unde } L \subseteq O \times O.$$

Rezultă, deci, că un sistem, din punct de vedere matematic, este o relaţie binară omogenă pe mulţimea obiectelor acelui sistem.

[5] Aici *grafe* este folosit ca plural de la *graf*. Unii autori preferă forma *grafuri* în loc de *grafe*.

Reprezentarea geometrică (care este de fapt un graf!) a sistemului precizat în exemplul de mai sus este prezentat în *Fig. 1.1*.

Oricare ar fi un sistem, întotdeauna există nişte mărimi notate, de exemplu, cu u dintr-un anumit univers U, mărimi care caracterizează sistemul respectiv.

Legăturile dintre obiectele sistemului se pot preciza printr-o relaţie de felul următor:

$$F(u)=0, \quad \text{unde} \quad F : U \to V, \quad V \text{ fiind un alt univers şi } 0 \in V.$$

În practică există foarte multe exemple de sisteme pentru care mărimile ce caracterizează sistemul respectiv pot fi diferenţiate în:

- mărimi de intrare: X
- mărimi de ieşire: Y

Rezultă că avem $u=(x,y),\ x \in X,\ y \in Y,$ deci $F(u)=F((x,y))=G(x,y)=0$, adică $G(x,y)=0,$ (aplicaţie definită pe $X \times Y$).

Avem,

$$G : X \times Y \to V$$

1.2. Sisteme deschise. Sisteme închise

Sistemele pentru care putem face o diferenţiere între mărimile de intrare şi mărimile de ieşire, le numim sisteme cu intrări şi ieşiri, adică ***sisteme deschise***. În această situaţie un sistem deschis se mai notează $(X,Y,G),$ cu $G(x,y)=0.$ Altfel, sistemul se numeşte ***sistem închis***.

În concluzie, avem sisteme:

– *deschise* (cu intrări şi ieşiri);

– *închise* (nu conţin intrări şi ieşiri).

Majoritatea sistemelor sunt deschise. De exemplu, Internetul este un sistem deschis (sistem informatic distribuit).

În cazul sistemelor informatice, termenul de *deschidere* (*openness*) este caracteristica unui sistem care indică, dacă el poate fi extins şi implementat în moduri diferite. Deschiderea pentru un sistem distribuit se referă în primul rând la disponibilitatea de adăugare şi publicarea de noi servicii de partajare a resurselor (interfeţele cărora devin publice). Sistemele distribuite deschise sunt bazate pe asigurarea unui mecanism uniform de comunicare şi publicare a interfeţelor pentru accesul la resursele partajate în mod transparent[6].

Unii autori consideră Universul un sistem închis, dar acest subiect naşte dispute care depăşesc obiectivele acestei lucrări.

Exemplele cele mai des folosite pentru a ilustra conceptul de *sistem închis* sunt reacţiile chimice petrecute într-un mediu etanş şi terrariul (Filip,

[6] Dziţac, I., Modovan, G., *Sisteme distribuite- Modele informatice*, Ed. Univ. Agora, 2006

2004). „Examinarea chiar sumară a acestor exemple arată că, pentru a putea dăinui, un sistem trebuie să interacţioneze, din când în când cu mediul, altfel decade şi se distruge. Din această cauză şi deoarece sistemele perfect închise nu există decât în condiţii de laborator, este util să fie luate în considerare *sistemele relativ închise.* Acestea interacţionează cu mediul în mod limitat şi în condiţii de incertitudine minimă. Altfel spus, sistemele relativ închise pot accepta din partea mediului numai mărimi de intrare cunoscute sau planificate pentru a le prelucra într-un mod bine controlat în vederea transmiterii către mediu a unor mărimi de ieşire, de asemenea bine definite. Sistemele informatice de prelucrare a tranzacţiilor constituie exemplu pentru această categorie. Un *sistem deschis* este caracterizat prin aceea că poate accepta şi mărimi de intrare imprecise şi incerte (pe lângă cele precise şi certe) şi chiar perturbaţii pentru a furniza la ieşire mărimi atât predictibile, cât şi impredictibile. În plus, pentru a putea avea o anumită durată de viaţă, sistemul deschis trebuie să fie adaptabil la evoluţiile petrecute în mediu şi chiar în interiorul său. Ca urmare a acestei proprietăţi, sistemele deschise sunt denumite uneori sisteme *adaptive* (sau *organice).*"[7]

[7] Filip, F.G., *Sisteme suport pentru decizii*, Ed. Tehnică, 2004

CAPITOLUL 2

SISTEME DESCHISE

În cele mai multe situații, pentru un sistem *(X,Y,G),* cu *G(x,y)=0,* ieşirile acestui sistem pot fi explicitate în funcție de intrări, adică:

$$y=f(x); \qquad \text{unde} \quad f: X \to Y \quad \text{sau} \quad f: X \to P(Y)$$

notând cu *P(Y)* mulțimea părților lui *Y.*

Reprezentarea unui sistem deschis se face adesea sub forma din figura 2.1, unde:

x - *intrările* în sistem

y - *ieşirile* din sistem

f - *acțiuni* asupra intrărilor *x.*

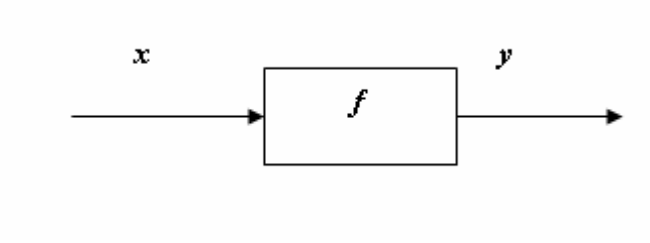


Fig. 2.1. Sistem deschis

Intrările şi ieşirile se reprezintă adesea vectorial, ca în figura 2.2:

$$x=\begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{pmatrix} \qquad y=\begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_n \end{pmatrix} \Rightarrow \qquad \begin{matrix} x_1 \to \\ x_2 \to \\ . \\ . \\ . \\ x_n \to \end{matrix} \boxed{f} \begin{matrix} \to y_1 \\ \to y_2 \\ . \\ . \\ . \\ \to y_n \end{matrix}$$

Fig. 2.2. Reprezentarea vectorială a unui sistem deschis

În general $x_1, x_2, ..., x_n, y_1, y_2, ..., y_m$ pot avea semnificații diferite.

2.1. Clasificări ale sistemelor deschise

Clasificările sistemelor deschise se fac în funcţie de natura mărimilor de intrare, de ieşire şi de natura acţiunilor f asupra mărimilor de intrare. În continuare vom da câteva astfel de clasificări ale sistemelor deschise.

2.1.1. Sisteme deterministe. Sisteme nedeterministe. Sisteme fuzzy

O clasificare a sistemelor cu intrări şi ieşiri, în funcţie de natura acţiunilor, este următoarea:

a) sisteme deterministe: dacă mărimile x,y – sunt de natură deterministă (nu întâmplătoare) şi dacă f – sunt acţiuni bine determinate;

b) sisteme nedeterministe: în caz contrar.

Fie X - o mulţime de elemente, iar $A \subset X$. În această situaţie, de exemplu, pentru $x \in X$, avem $x \in A$ sau $x \notin A$ (ceea ce are o legătură cu logica bivalentă !), iar funcţia caracteristică $f_A: X \to \{0,1\}$ are valorile:

$$f_A(x) = \begin{cases} 1 & daca \quad x \in A \\ 0 & daca \quad x \notin A \end{cases} \quad ;$$

c) sisteme fuzzy: mărimile x, y, f au o anumită imprecizie sau incertitudine în determinarea lor.

Fie, de exemplu, intrările X - o mulţime. În teoria sistemelor fuzzy, un element aparţine acestei mulţimi mai mult sau mai puţin, adică avem o incertitudine privind această apartenenţă; aparţine mulţimii X într-o mică măsură, care este, deci, mai aproape de *0* sau într-o mai mare măsură, care este mai aproape de *1*. Funcţia caracteristică, are orice valoare din intervalul [0,1], adică $f_A: X \to [0,1]$. Analog, pentru mulţimea Y.

2.1.2. Sisteme discrete. Sisteme continue

În funcţie de natura mărimilor de intrare/ieşire sistemele se clasifică în:

a) sisteme discrete: sisteme pentru care mărimile de intrare $x \in X$ sunt mărimi discrete (se pot reprezenta sub forma unor vectori), mărimile de ieşire $y \in Y$ sunt, de asemenea, mărimi discrete. Avem,

$x = (x_1, x_2, \ldots, x_n)$

$y = (y_1, y_2, \ldots, y_m)$

b) sisteme continue: mărimile de intrare şi de ieşire nu pot fi discretizate;

$y=f(x)$ x,y nu se pot discretiza

2.1.3. Sisteme statice. Sisteme dinamice

În funcţie de independenţa/dependenţa de timp a mărimilor de intrare/ieşire sistemele se clasifică în:

a) sisteme statice: mărimile de intrare şi de ieşire nu depind de timp;

b)sisteme dinamice: mărimile de intrare şi de ieşire depind de timp:

$x=x(t), \quad y=y(t)$

2.1.4. Sisteme cibernetice

Cele mai importante sisteme cibernetice sunt cele de natură *economică* şi cele de natură *tehnică*; ele au multe aplicaţii practice importante.

Un sistem cu intrări şi ieşiri, l-am reprezentat sub forma din figura 2.3:

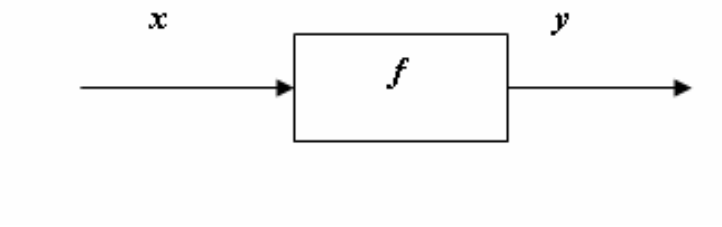


Fig. 2.3. Sistem deschis (cu intrare şi ieşire)

Uneori, o parte din ieşiri se întorc în sistem sub formă de intrări, ca în figura 2.4, în acest caz având loc un aşa-numit fenomen de *feed – back* (adică, presupune o întoarcere a unor ieşiri sub formă de intrări). În acest caz, sistemul se numeşte *sistem cibernetic*.

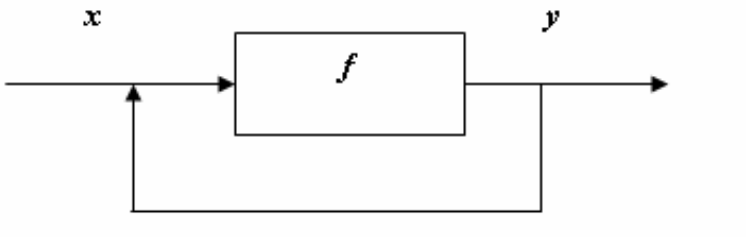


Fig. 2.4. Sistem deschis cu feed-back

Precizare importantă. Într-un sistem cibernetico – economic, fenomenul de feed-back se foloseşte în scopul reglării unor fenomene (de natură economică), adesea de a impune ieşirilor anumite condiţii (restricţii) care să ducă economia pe un drum de progres.

2.1.5. Exemple de sisteme deschise

În exemplele de sisteme deschise (cu intrări şi ieşiri), pe care le prezentăm în continuare, vom preciza natura mărimilor de intrare şi ieşire x, respectiv y şi natura acţiunilor f asupra intrărilor x.

Exemplul 2.1. Sisteme fizice

x,y – mărimi fizice
f - fenomene fizice

Exemplul 2.2. Sisteme chimice

x,y - mărimi chimice
f - reacţii chimice

Exemplul 2.3. Sisteme biologice

x,y - mărimi biologice
f - acţiuni biologice

Exemplul 2.4. Sisteme algebrice

x, y mărimi numerice
f - corespondenţă

În acest caz, relaţia $y=f(x)$ se scrie sub forma $y=A*x$, unde A este matricea coeficienţilor (sistem discret, static), apoi x, y sunt vectori coloană, iar $A*x$ reprezintă produsul dintre matricea coeficienţilor A şi vectorul coloană x ale cărui componente sunt necunoscutele sistemului. Aici y este vectorul termenilor liberi din sistemul algebric considerat.

Exemplul 2.5. Sisteme dinamice

Sistemele dinamice (din teoria ecuaţiilor diferenţiale) sunt cele care se definesc printr-o ecuaţie diferenţială de ordinul întâi, de forma:

$$\frac{dx}{dt} = f(t,x).$$

Putem considera această formă, ca fiind una vectorială şi atunci avem un sistem de ecuaţii diferenţiale de ordinul întâi.

Exemplul 2.6. Sisteme economice

x, y - mărimi economice
f - legislaţie economică

Exemplul 2.7. Sisteme politice

x, y - oamenii unei colectivităţi
f - doctrina politică

Exemplul 2.8. Sisteme filozofice

x, y - oamenii unei colectivităţi
f - doctrina filozofică

2.1.6. Generalizări. Sisteme logice secvenţiale şi automate finite

Definiţia 2.1. Fie sistemul cu intrări şi ieşiri (X,Y,G) , cu $G(x,y)=0$. Mulţimile X,Y pot fi mulţimi de vectori ale căror componente sunt elemente $0,1$ deci, dacă $x \in X$, atunci $x=(x_1, x_2, ..., x_n); x_i \in \{0,1\}, i=\overline{1,n}$ Vom considera în continuare că X şi Y au această particularitate şi adăugăm la acestea încă o mulţime Q - mulţimea stărilor interioare ale unui sistem precum şi nişte funcţii.

Fie funcţiile

$$\lambda: X \times Q \to Q$$
$$\mu: X \times Q \to Y,$$

iar apoi să considerăm *sistemul* $S = (X, Y, Q, \lambda, \mu)$. Sistemul S se numeşte *sistem logic secvenţial*. Funcţiile λ, μ se numesc *funcţii de tranziţie* (analog noţiunii de *automat finit*); λ - funcţia de tranziţie pentru stările sistemului şi μ - funcţia de tranziţie pentru ieşirile din sistem.

Definiţia 2.2. Generalizăm funcţiile λ, μ modificând codomeniile acestora, astfel:

$$\lambda: X \times Q \to P(Q)$$
$$\mu: X \times Q \to P(Y).$$

Sistemul S în acest caz poartă denumirea de *automat Mealy*.

Definiţia 2.3. Fie automatul Mealy $S = (X, Y, Q, \lambda, \mu)$. Dacă

$$|\lambda(x,q)| \le 1$$
$$|\mu(x,q)| \le 1$$

automatul este *determinist*, altfel avem un *automat nedeterminist*.

Definiţia 2.4. Dacă $S=(X, Y, Q, \lambda, \gamma)$, unde

$$\lambda: X \times Q \to P(Q)$$
$$\gamma: Q \to P(Y),$$

iar X, Y, Q au semnificaţiile de mai sus, sistemul se numeşte *automat Moore*.

Definiţia 2.5. Dacă $S=(X, Y, Q, \mu)$, unde $\mu: X \times Q \to P(Y)$,
iar X, Y, Q au semnificaţiile de mai sus, avem un *automat Starke.*

2.1.7. Sisteme de tranziţie

Sistemele de tranziţie sunt sisteme dinamice în care:
- se pot pune în evidenţă *stări* ale sistemului;
- *relaţii* între stările sistemului la momente consecutive de timp;
- trecerea de la o anumită situaţie, adică de la o anumită mulţime de stări în care se găseşte sistemul într-un anumit moment, la o altă situaţie, deci la o altă mulţime de stări ale sistemului în momentul următor, acceptăm că se face în urma unor *acţiuni*.

Definiţia 2.6.. *Un **sistem de tranziţie** este un ansamblu de forma ST = (Q, I, A, T) unde:*

Q - mulţimea stărilor (mulţime finită);
$I \subseteq Q$ *– mulţimea stărilor iniţiale (intrări în sistem);*
A - mulţimea acţiunilor (finită), identificate cu nişte etichete;
$T \subseteq Q \times A \times Q$ *relaţia tranziţiilor*

Observaţie: *(Q, A, Q; T)* este o relaţie ternară ce defineşte relaţia tranziţiilor.

2.1.7.1. Reprezentări ale sistemelor de tranziţie

Avem următoarele reprezentări ale sistemelor de tranziţie:
a) sub formă de graf orientat etichetat; vârfurile sunt etichetate cu stări ale sistemului, iar arcele cu acţiuni.

Fie $p,q \in Q;\ a \in A;\ (p, a, q) \in T$.

Convenim să reprezentăm această tranziţie ca în figura 2.5:

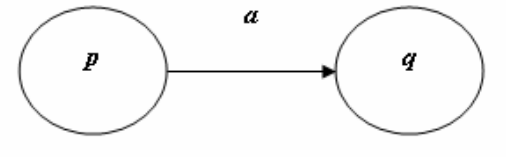


Fig. 2.5. Graful unei tranziţii

Convenim, apoi, să reprezentăm stările iniţiale*: $s \in I$,* ca în figura 2.6:

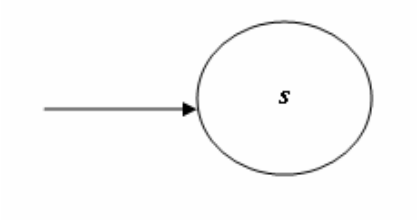


Fig. 2.6. O stare iniţială

Exemplul 2.9.

Fie $ST= (Q, I, A, T)$ $Q = \{s_1,s_2,s_3,s_4\}$
$I = \{s_1,s_2\} \subset Q$; $A=\{a_1, a_2\}$
$T = \{(s_1,a_1,s_3), (s_1,a_2,s_2), (s_2,a_1,s_4), (s_2,a_2,s_4)\}$

Avem:

a) *reprezentarea sub formă de graf* din figura 2.7.

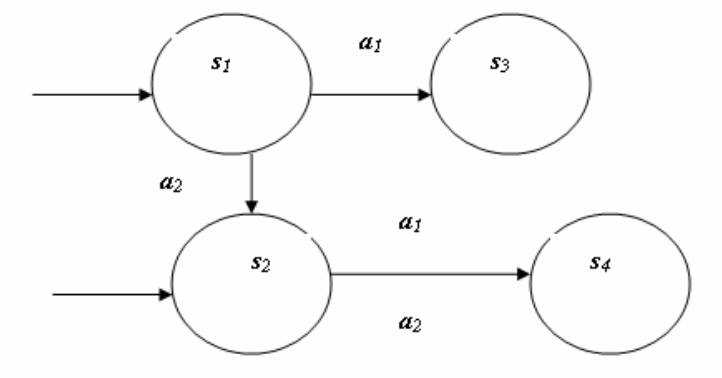


Fig. 2. 7. Graful sistemului de tranziţie *ST*

b) *reprezentarea tabelară*

Fie $Q = \{s_1, s_2, ..., s_m\}$ mulţimea stărilor şi A mulţimea acţiunilor sistemului considerat. În general, două stări, de exemplu, s_i şi s_j pot fi legate prin mai multe arce. Convenim să notăm acţiunea asociată arcului al k-lea, $k \in \{1, 2, ..., r\}$ ce leagă cele două stări s_i , s_j cu $a_{ij}^k \in A$. Atunci o tranziţie din mulţimea tranziţiilor T în această situaţie va fi: $t_{ij} = (s_i, a_{ij}^k, s_j)$, $t_{ij} \in T$, $k \in \{1, 2, ..., r\}$.

Se defineşte tabelul *T* din figura 2.8, cu elementele:
$A_{ij}=\{a_{ij}^1, ..., a_{ij}^r\}$, $a_{ij}^k \in A$, $i,j = 1... m \quad k \in \{1, 2, ..., r\}$.
$I = \{s_1, s_2, ... , s_n\} \quad n \le m,$ stările iniţiale se plasează în tabel la început.

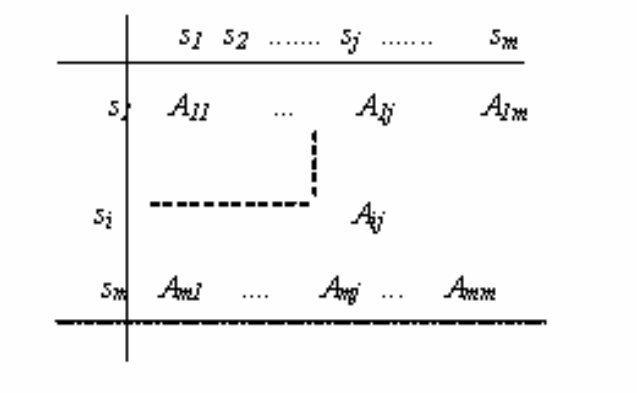


Fig. 2.8. Tabelul *T*

Sistemul de tranziţii *ST*, considerat anterior ca exemplu, se reprezintă sub formă de tabel, astfel:

ST	s_1	s_2	s_3	s_4
s_1	$\varnothing$	$\{a_2\}$	$\{a_1\}$	$\varnothing$
s_2	$\varnothing$	$\varnothing$	$\varnothing$	$\{a_1,a_2\}$
s_3	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
s_4	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

Fig. 2.9. Tabelul sistemul de tranziţii *ST* din exemplul 2.9

2.1.7.2. Comportarea unui sistem de tranziţii (ST). Căi si urme

Vom defini la început noţiunile de cale şi urmă într-un ST.

Definiţia 2.10. *Numim **cale** într-un ST o succesiune de stări şi acţiuni ce sunt etichete ale nodurilor şi arcelor şi care se obţine pornind de la o stare iniţială până la o stare ce este etichetă a unui nod terminal în reprezentarea sub forma de graf a unui ST.*

Exemple: $s_1a_1s_3$, $s_1a_2s_2a_1s_4$ etc.

Fie *X* – un sistem de tranziţie; notăm cu *căi(X)* – mulţimea tuturor căilor dintr-un astfel de sistem de tranziţie.

Observăm că în exemplul de ST considerat mai sus avem *căi(ST)*= $\{s_1a_1s_3,\ s_1a_2s_2a_1s_4\ ,\ s_1a_2s_2a_2s_4,\ s_2a_1s_4\ ,\ s_2a_2s_4\}$.

Definiţia 2.11. *Numim **urmă** a unui sistem de tranziţie, ca fiind orice prefix a unei secvenţe de stări şi acţiuni care reprezintă o cale într-un astfel de sistem.*

Exemple de urme. Fie calea $s_1\ a_1\ s_3$. Avem următoarele urme: $\varepsilon,\ s_1,\ s_1a_1,\ s_1a_1s_3$ unde ε este secvenţa vidă.

Fie *X* – un sistem de tranziţie; notăm cu *urme(X)* – mulţimea tuturor urmelor din sistemul de tranziţie *X*.

Definiţia 2.12. *Lungimea unei căi sau a unei urme este egală cu numărul simbolurilor din care aceasta este formată.*

Comportamentul unui sistem de tranziţii este caracterizat de posibilităţile de traversare şi utilizare a stărilor din graful sistemului de tranziţii. Posibilităţile de traversare a grafului unui sistem de tranziţii, în esenţă, revine la determinarea căilor şi urmelor din acel sistem de tranziţii. Având dat un sistem de tranziţii *X,* putem determina relativ uşor elementele mulţimii *căi(X)*, respectiv *urme(X).*

Iată cum definim, pe baza noţiunilor prezentate până aici, comportamentul unui sistem de tranziţii.

Definiţia 2.13. *Comportamentul unui sistem de tranziţii este definit de unele proprietăţi şi caracteristici ale acestuia ca: cicluri, urme de lungime maximă, numărul stărilor iniţiale, accesibilitatea stărilor, iar apoi relaţia dintre acesta şi alte sisteme.*

2.2. Sisteme distribuite

Cu excepţia Internetului, nici un alt exemplu de sistem cu intrări şi ieşiri, pe care l-am dat până acum nu se referă la domeniul strict al informaticii. Să ne referim acum la câteva dintre cele mai importante exemple de sisteme din domeniul informaticii.

Avem în vedere relaţia care caracterizează un sistem cu intrări şi ieşiri de forma

$$y = f(x)$$

Fie

x - *date de intrare*
y - *date de ieşire*
f - *mijloace de prelucrare a datelor de intrare*

Un astfel de sistem îl numim *sistem de prelucrare a datelor.*

Un sistem în care datele de intrare şi cele de ieşire reprezintă *informaţii* de un anumit interes, atunci acest sistem îl numim ***sistem informaţional***.

Dacă prelucrarea datelor de intrare se face mijloace mecanizate spunem că avem un *sistem de prelucrare mecanizată a datelor.*

Dacă prelucrarea datelor de intrare se face cu ajutorul calculatorului (calculatoarelor) atunci avem un *sistem de prelucrare automata a datelor* sau ***sistem informatic***. În general un sistem informatic cuprinde mai multe *aplicaţii informatice.* O aplicaţie informatică se referă la un domeniu mai restrâns din cadrul unei societăţi comerciale sau întreprindere.

Odată cu realizarea unor reţele de calculatoare şi implementarea unor aplicaţii de o anvergură mai largă pe acestea, s-a ajuns la necesitatea realizării unor sisteme informatice distribuite pe care le vom numi simplu *sisteme distribuite.*

Definiţia 2.14. *Prin sistem distribuit înţelegem un sistem informatic implementat pe o reţea de calculatoare în care componentele soft şi hard situate în calculatoarele din reţea comunică şi îşi coordonează acţiunile numai prin intermediul transmiterii unor* ***mesaje***.

Din definiţie rezultă că avem o mulţime de noduri care pot fi unul sau mai multe *procesoare* şi care au acces fiecare la o memorie proprie şi care sunt legate între ele prin linii (canale) de comunicaţie. Un sistem distribuit este un sistem de programe autonome construit pe o reţea transparentă de calculatoare. Un utilizator al unui sistem distribuit acţionează ca şi cum ar exista un singur procesor virtual. Alocarea lucrărilor pe procesoare şi a fişierelor pe discuri, transferul de fişiere între locul unde ele sunt stocate şi locul unde vor fi

utilizate, alături de orice funcţie din sistem, toate trebuie să fie executate automat de sistemul de operare.

Observaţie. Natura liniilor de comunicaţie poate fi diferită: comunicaţie prin fir (fibră optică), prin unde radio, sateliţi etc.

Utilizarea unor sisteme distribuite în rezolvarea unor probleme practice prezintă unele avantaje, precum:

- performanţe superioare în ceea ce priveşte timpul de execuţie al unor programe ce au în vedere rezolvarea unor aplicaţii concrete;
- utilizarea în comun a unor resurse;
- extensibilitatea care se poate realiza uşor într-un sistem distribuit; un sistem distribuit poate fi modificat relativ uşor prin adăugarea sau îndepărtarea unor componente;
- fiabilitate sporită: într-un sistem distribuit avem această caracteristică datorită faptului că aplicaţiile care se rulează într-un sistem distribuit sunt astfel concepute încât ele să nu sufere din cauza nefuncţionării corecte sau deloc a unor componente, respectiv procese;
- facilităţi de comunicare; un sistem distribuit reprezintă un mijloc eficient şi comod de comunicare a unor informaţii la distanţă. De exemplu aplicaţia Internet, prin intermediul căreia se poate realiza, de exemplu, corespondenţa electronică cu condiţia ca adresantul şi destinatarul să se găsească într-o reţea conectata la Internat.

Se pot desprinde unele consecinţe importante, care rezultă din definiţia de mai sus şi care reprezintă caracteristicile de bază a sistemelor distribuite:

- ***concurenţa**:* prin partajarea resurselor;
- ***inexistenţa unui ceas global**:* programele care trebuie să coopereze îşi coordonează acţiunile prin *schimburi de mesaje* şi nu se poate conta pe sincronizarea ceasurilor;
- ***eşecuri independente**:* fiecare componentă a sistemului poate eşua în mod independent neafectându-le pe celelalte.

Din definiţie şi avantajele enumerate se observă că problema comunicării între noduri este esenţială, de unde rezultă că sunt necesare câteva cunoştinţe în domeniu.

Prin conectarea mai multor noduri (staţii, procesoare) se pot realiza mai multe feluri de arhitecturi de reţele, respectiv topologii de reţele. Există două tipuri de topologii numite ***tipuri de bază***:

- prima, presupune **comunicare punct-la-punct**; totdeauna e vorba de existenţa a două noduri cu o legătură între ele, numite *noduri vecine* :

Fig. 2.10. Topologia reţelei de tip „comunicare punct-la punct”

- a doua, se bazează pe **comunicarea multipunct (magistrală)**; într-un sistem de comunicare multipunct: există o linie de comunicaţie pentru mai multe noduri care se conectează la aceeaşi linie de comunicaţie; într-un anumit moment pe această linie de comunicaţie poate fi activ un singur nod.

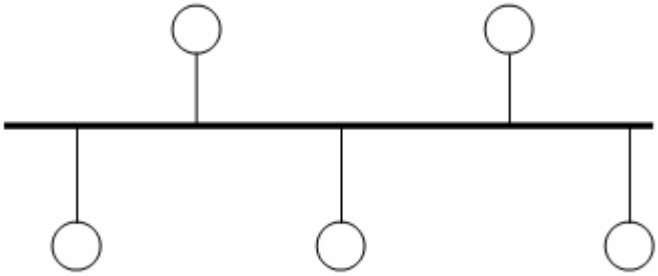


Fig. 2.11. Topologia reţelei de tip magistrală (comunicare multipunct)

2.2.1. Structuri de reţele pentru sisteme distribuite (SD)

În ceea ce priveşte topologiile de reţele ce le folosesc sistemele distribuite (SD) şi care se referă la dispunerea geometrică a nodurilor şi a legăturilor care există, pentru un sistem distribuit, putem pune în evidenţă următoarele structuri:

1. *structuri de tip centralizat* sau *stea*

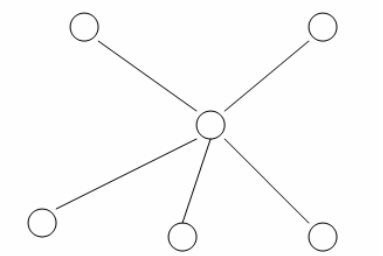


Fig. 2.12. Topologia reţelei SD de tip „stea”

Există un nod central cu funcţii de control în reţeaua de calculatoare asociată sistemului distribuit; celelalte noduri pot comunica între ele numai prin intermediul nodului central.

2. *structura ierarhică* – sub forma unei arborescenţe, cu un nod rădăcină a structurii; celelalte conectate într-o anumita ierarhie, pe nivele. E specifică utilizatorilor dintr-o societate comercială, de regulă organizată într-o anumită ierarhie între diverse sectoare şi compartimente ale societăţii.

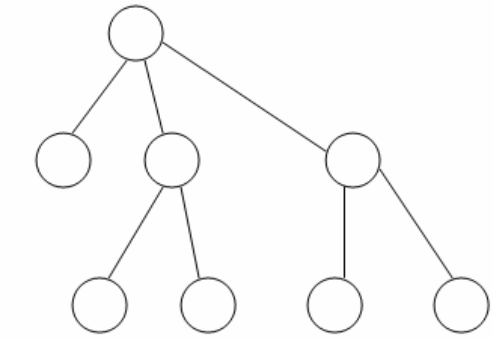


Fig. 2.13. Topologia reţelei SD de tip ierarhic (arbore)

3. *structura de inel sau buclă*, mult folosită în practică, se utilizează atunci când calculatoarele nu sunt la distanţe mari unele de altele.

O astfel de structură se caracterizează prin aceea că fiecare nod are *2* vecini.

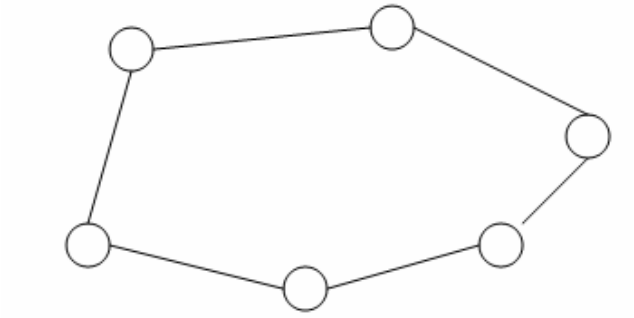


Fig. 2.14. Topologia reţelei SD de tip inel (buclă)

4. *structura de multistea* sau *distribuită* – există mai multe noduri cu funcţii de control (de exemplu: mai multe arhitecturi stea conectate între ele)

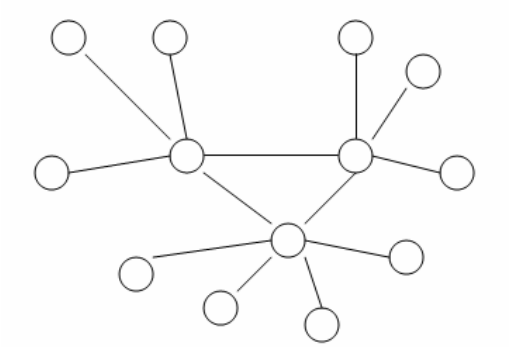


Fig. 2.14. Topologia reţelei SD de tip multistea (distribuită)

5. *structura complet distribuită*
- cea mai generală formă de structură a unei rețele; reprezentarea unei astfel de topologii făcându-se cu un graf complet; între orice două noduri ale sistemului există o linie de comunicație.

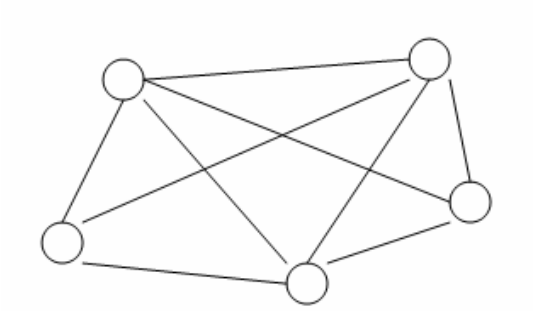


Fig. 2.14. Topologia reţelei SD complet distribuită

Există şi alte structuri de rețele importante, particulare, în practică. Vom prezenta în continuare câteva dintre acestea.

2.2.2. Rutajul în rețele de comunicații

Comunicarea între nodurile unei rețele se face prin intermediul unor mesaje pe care le considerăm, în general, de lungime fixă. Întotdeauna avem un nod sursă, de unde pleacă un mesaj şi un nod destinație, unde trebuie să ajungă mesajul transmis pe o anumită cale de comunicație sau altfel spus pe o anumită **rută**. Problema *rutajului* ne cere să determinăm căile de comunicații între un nod sursă şi un altul destinație. Atunci când se lansează dintr-un nod sursă P_s

un mesaj *m* care trebuie să ajungă la un nod destinaţie P_d , de regulă sub forma *msg(m,d),* rutajul trebuie să indice întâi succesorul lui P_s , adică vecinul acestuia P_v la care trebuie să ajungă mesajul *m.* Apoi, se continuă acest procedeu din aproape în aproape, până se ajunge la destinaţie. Informaţiile relative la rutaj se păstrează de regulă în nişte tabele locale (în fiecare nod). Scopul în determinarea unui rutaj este de a stabili căi de comunicaţii (rute) convenabile din punctul de vedere al unor criterii de optim (de exemplu, cel mai scurt drum, etc.). Aceste rute se pot determina de o manieră centralizată sau distribuit (din aproape în aproape). În general pentru rezolvarea problemei de rutaj se folosesc anumiţi algoritmi în acest scop. Dacă un nod rezolvă numai probleme de rutaj, pe acesta îl numim ***rutăr*** (*ruter*)**.** Pentru structuri particulare de reţele, rutajul poate fi o problemă foarte simplă.

2.3. Structuri de reţele particulare importante

În continuare ne vom ocupa de cele mai importante topologii de reţele, interconectate static pentru a asigura accesul la procesoarele pe care le conţin. Pentru început, vom avea în vedere cele mai simple reţele de comunicaţii între procesoare şi anume ***reţelele liniare*** şi ***reţelele de tip inel***.

Un exemplu de reţea liniară este dat în figura 2.15. *a)*, fiecare procesor comunică, mai puţin primul şi ultimul, cu alte două procesoare. În figura alăturată *b)* avem un exemplu de reţea inel, aici fiecare procesor comunică cu alte două procesoare.

Fig. 2.15 ***a)***

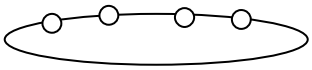


Fig. 2.15 ***b)***

Folosind aceste reţele elementare prezentate aici şi menţionate, de altfel, mai sus, vor fi elemente constituente a altor structuri mai complexe pe care le vom prezenta în continuare.

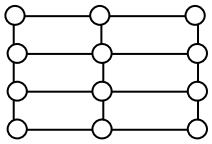
2.3.1. Unele reţele cu structură de masiv (grilă)

Vom deosebi mai multe feluri de reţele pe care le vom prezenta în continuare.

a) reţele bidimensionale

1) ***rectangulare (masiv, grilă)***; cu proprietatea că fiecare nod (procesor) comunică direct cu cel mult alte *4* procesoare. Un exemplu de structură de acest fel avem alăturat în figura 2.16. a) *rectangular (masiv, grilă).*

2) ***cu structură de tor***; cu proprietatea că fiecare nod (procesor) comunică direct cu exact alte *4* procesoare. Un exemplu de structură de acest fel avem alăturat în figura 2.16. b) *tor bidimensional*



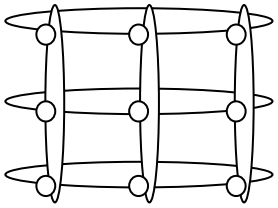


Fig. 2.16. a) Rectangular (masiv, grilă) **b) Tor bidimensional**

b) reţele tridimensionale

1) *masiv tridimensional*; cu proprietatea că fiecare nod (procesor) comunică direct cu cel mult alte *6* procesoare. Un exemplu de structură de acest fel avem în figura 2.17.

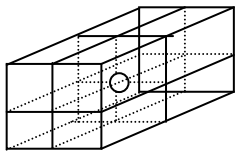


Fig. 2.17. Masiv tridimensional

2) *tor tridimensional*; cu proprietatea că fiecare nod (procesor) comunică direct cu exact alte *6* procesoare incluse în structuri de tip inel.

Se poate construi uşor un exemplu de tor tridimensional, ca în figura 2.18, unde am pus în evidenţă legăturile de inel doar pe două muchii.

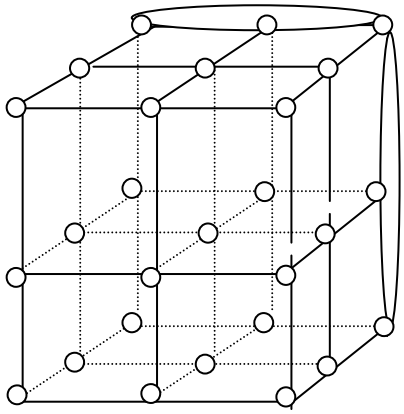


Fig. 2.18. Tor parţial tridimensional

c) cu structură arborescentă

O structură arborescentă este o rețea în care există un singur drum de comunicare între două procesoare. Structurile liniară şi stea sunt cazuri particulare a structurilor arborescente. Vom deosebi în acest caz următoarele trei tipuri de structuri arborescente:

a) statică – fiecare nod conține un procesor. Un exemplu avem în figura 2.19 a), unde este marcat drumul de la procesorul sursă P_s la procesorul destinație P_d.

b) dinamică - procesoarele sunt în nodurile terminale marcate cu cerculeţe, iar celelalte noduri marcate cu pătrate sunt noduri ce reprezintă o legătură dinamică, ele conţin sursa şi destinaţia din subarborele cu această rădăcină. Un exemplu avem în figura 2.19 b), unde este marcat drumul de la procesorul sursă P_s la procesorul destinație P_d.

Fig. 2.19. a)Arborescenţă statică **b) Arborescenţă dinamică**

c) arborescenţă dezvoltată - mai multe legături de comunicare cu cât procesoarele sunt mai apropiate de rădăcină. Un exemplu avem în figura 2.20, unde fiecare nod terminal are o legătură directă cu rădăcina arborescenței.

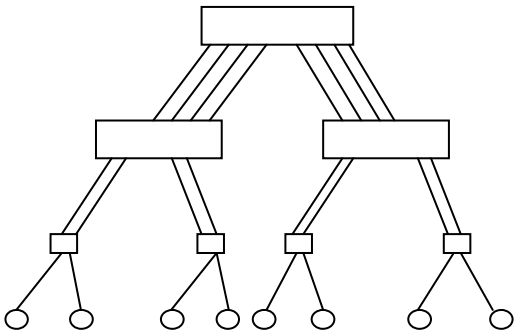


Fig. 2.20. Arborescenţă dezvoltată

2.3.2. Reţele hipercubice

O *reţea hipercubică* este o reţea multidimensională cu exact doua procesoare în fiecare dimensiune; o structura hipercubică se defineşte recursiv. Convenim să notăm un hipercub n-dimensional, adică un *n*-cub, cu H_n, iar nodurile acestuia le etichetăm cu secvenţe de cifre binare *0,1* de lungime *n, n≥1* adică, $w=a_1a_2...a_n;\ a_i\in\{0,1),\ i=1...\ n.$ Construcţia recursivă este următoarea: pornim de la un hipercub *0*-dimensional H_0 care are un singur procesor şi pe care convenim să nu-l etichetăm; un hipercub *1*-dimensional este construit conectând două hipercuburi *0*-dimensionale, etichetând procesoarele, primul cu *0* al doilea cu *1*, vom scrie $H_1=(0H_0,\ 1H_0)$. În general un hipercub n-dimensional este construit conectând corespunzător procesoarele a două hipercuburi *(n-1)*-dimensionale; în faţa etichetelor procesoarelor primului hipercub H_{n-1} vom pune *0,* iar în faţa etichetelor procesoarelor celui de al doilea hipercub H_{n-1} vom pune *1,* adică $H_n=(0H_{n-1},\ 1H_{n-1})$. Aceste etichete *identifică* nodurile (procesoarele) hipercubului (fig. 2.21-2.24).

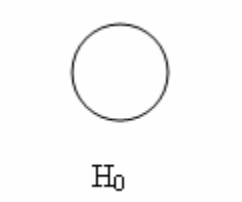


Fig. 2.21. H_0 –hipercub cu 2^0 = 1 nod (procesor)

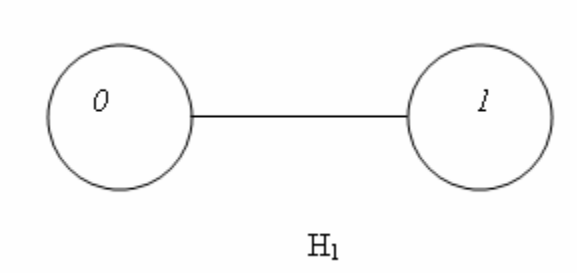


Fig. 2.22. H_1 –hipercub cu 2^1 = 2 noduri

Hipercubul H_2 se obţine recursiv din două hipercuburi unidimensionale H_1 unind nodurile cu aceeaşi etichetă şi punând: *0* în faţa etichetelor nodurilor primului hipercub H_1, iar *1* în faţa etichetelor nodurilor celui de al doilea hipercub H_1, adică: $H_2=(0H_1,\ 1H_1)$ (Fig. 2.23).

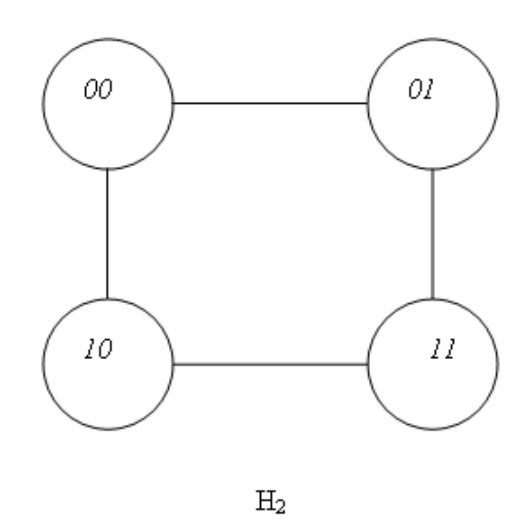


Fig. 2.23. H_2 –hipercub cu $2^2 = 4$ noduri

Hipercubul H_3 se obţine din două hipercuburi bidimensionale procedând la fel ca în cazul precedent.

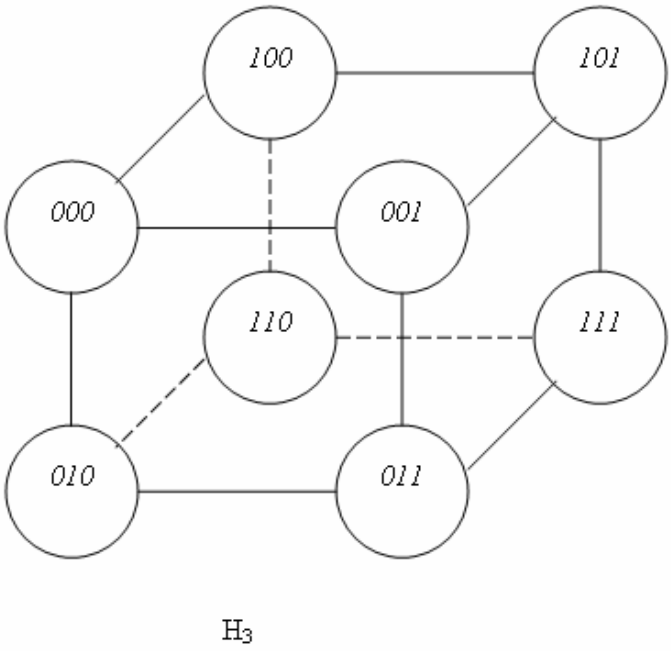


Fig. 2.24. H_3 –hipercub cu $2^3 = 8$ noduri

Definiţia 2.15. (recursivă) *Un hipercub n-dimensional, notat H_n, se obţine conectând două hipercuburi (n-1)-dimensionale H_{n-1}, adică $H_n=(0H_{n-1},1H_{n-1})$, după procedeul explicat în exemplele precedente.*

O structură de hipercub este importantă din punct de vedere practic pentru că ea are anumite proprietăţi deosebite, utile pentru realizarea unei comunicaţii optime într-un sistem distribuit cu o astfel de structură de reţea. Iată câteva din aceste proprietăţi.

Proprietatea 1. *O structură de hipercub n-dimensională H_n posedă 2^n procesoare şi $n\times2^{n-1}$ canale de legătură directă.*

Demonstraţie. Folosim metoda inducţiei matematice pentru demonstraţia acestei proprietăţi.

Fie $n=0$; după cum am văzut deja H_0 are $1=2^0$ procesoare şi 0 canale de legătură.

Dacă presupunem că hipercubul H_k are 2^k procesoare şi $k\times2^{k-1}$ canale de legătură vom demonstra că H_{k+1} are 2^{k+1} procesoare şi $(k+1)\times2^k$ canale de legătură.

Din definiţia unui hipercub $(k+1)$-dimensional acesta se obţine prin conectarea a două hipercuburi k-dimensionale. Folosind presupunerea că H_k posedă 2^k procesoare rezultă că H_{k+1} are $2\times2^k=2^{k+1}$ procesoare. Numărul canalelor de legătură într-un H_{k+1} este dat din dublarea numărului canalelor din H_k plus numărul canalelor care leagă cele 2 hipercuburi H_k şi care este egal cu numărul procesoarelor din H_k. Deci, numărul canalelor de legătură în H_{k+1} este $2\times(k\times2^{k-1})+2^k = (k+1)\times2^k$.

Vom da în continuare alte proprietăţi care se pot demonstra analog cu demonstraţia de mai sus.

Proprietatea 2. *Două procesoare sunt legate în mod direct dacă şi numai dacă reprezentarea binară a etichetelor acestora diferă într-o singură poziţie. Etichetele lui H_n sunt toate secvenţe binare de lungime n.*

Proprietatea 3. *Numărul de vecini ai fiecărui nod din H_n este n. Deci fiecare procesor într-o structură de hipercub H_n are legătură directă cu exact alte n procesoare.*

Proprietatea 4. *Fixând oricare k poziţii de biţi în etichetele procesoarelor dintr-un hipercub n-dimensional: procesoarele care diferă pe cele n-k poziţii rămase, formează un (n-k)- hipercub care conţine 2^{n-k} procesoare.*

Observaţie. Cei k biţi din *etichetele procesoarelor* se pot fixa în 2^k moduri diferite; rezultă că există 2^k cuburi *n-k* dimensionale.

Proprietatea 5. *Diametrul unei structuri de hipercub (cea mai lunga dintre cele mai scurte căi de legătură dintre oricare cupluri de procesoare) este n într-un* H_n.

Din proprietatea *2* rezultă că numărul de biţi diferiţi între două etichete ale nodurilor reprezintă lungimea celei mai scurte căi (adică cu cel mai mic număr de legături - muchii) dintre nodurile respective. Aceasta este chiar *distanţa Hamming* între două astfel de reprezentări *s* şi *d* notată *distHam(s,d)*, egală cu numărul de simboluri binare diferite între acestea şi care se determină printr-o operaţie "SAU exclusiv", notată $\oplus$, între cele două numere binare. Se ştie că: $0\oplus 0=0,\ 0\oplus 1=1,\ 1\oplus 0=1,\ 1\oplus 1=0$. De exemplu, dacă avem H_3 din fig. 2.25 şi considerăm eticheta nodului sursă *s=010,* iar eticheta nodului destinaţie *d=111,* atunci $s\oplus d=101$, deci *distHam(s,d)=2,* adică egală cu numărul de simboluri *1* în secvenţa rezultat. Regula de ***rutaj*** este următoarea: dintr-un nod curent se trece într-un nod vecin care este cel mai aproape de destinatar. Nu este nevoie de un algoritm sofisticat în acest sens. Putem folosi următoarea regulă: se calculează în fiecare nod *s* în care ajungem, $s\oplus d$ şi se trimite mesajul mai departe de-a lungul dimensiunii corespunzătoare bitului cel mai puţin semnificativ diferit de zero, până când mesajul ajunge la destinaţie. De exemplu, dacă *s=010, d=111,* avem $s\oplus d=101$. Astfel, mesajul se transmite nodului *011;* apoi se calculează $011\oplus 111=100$, deci mesajul ajunge la destinaţie *111*.

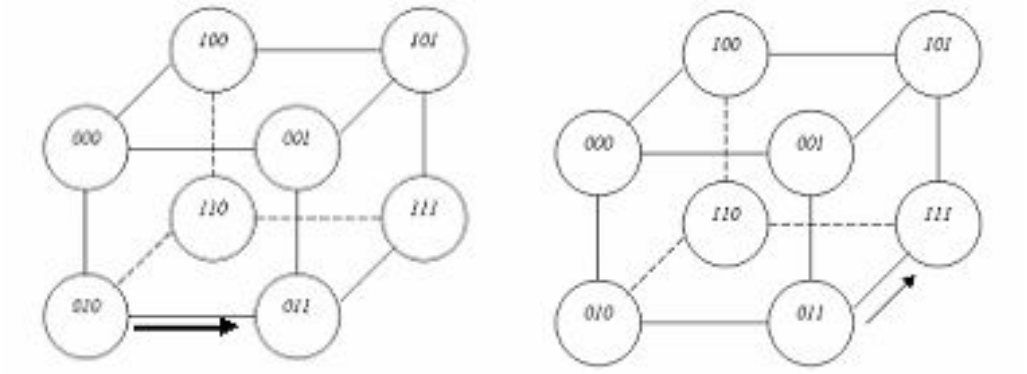


Fig. 2.25. Regula de rutaj

2.3.3. Modelarea unor reţele particulare într-o reţea de hipercub

În cele ce urmează vom pune în evidenţă într-un hipercub câteva structuri de reţele particulare pe care le-am prezentat mai sus. Unele structuri de reţele considerăm că le modelăm, astfel, cu ajutorul unui hipercub. Pentru aceasta trebuie să stabilim corespondenţa biunivocă între nodurile, respectiv arcele, structurii de reţea şi nodurile, respectiv arcele, hipercubului corespunzător. Corespondenţa între noduri revine la stabilirea unei corespondenţe între două secvenţe de cifre binare , respectiv între două numere care identifică noduri din două structuri diferite. Evident că este convenabil ca această corespondenţă, adică codificare, să se determine prin procedee cât mai simple.

2.3.3.1. Modelarea unei reţele cu structură de inel într-un hipercub

O reţea cu structură de inel, formată din 2^n noduri (procesoare), notată I_{2^n} poate fi modelată într-un n-cub, H_n (hipercub n-dimensional) cu 2^n noduri. Pentru aceasta, nodul i al inelului îl punem în corespondenţă biunivocă cu nodul $G(i,n)$ al hipercubului H_n , pentru $i \in \{0,1, \dots ,2^n-1\}$. Adică:

$$\{0,1, \dots ,2^n-1\} \ni i \leftrightarrow G(i,n) \text{ nod din } H_n$$

Funcţia G se defineşte recursiv, astfel:

$G(0,1)=0; \quad G(1,1)=1;$

$$G(i,m+1)=\begin{cases} G(i,m), & daca \quad i < 2^m \\ 2^m + G(2^{m+1}-1-i,m), & daca \ i \geq 2^m \end{cases}$$

Funcţia G se numeşte **codul binar reflectat al lui Gray.** Codurile Gray se construiesc printr-un procedeu extrem de simplu, ca în exemplul ce urmează, ilustrat fig. 2.26.

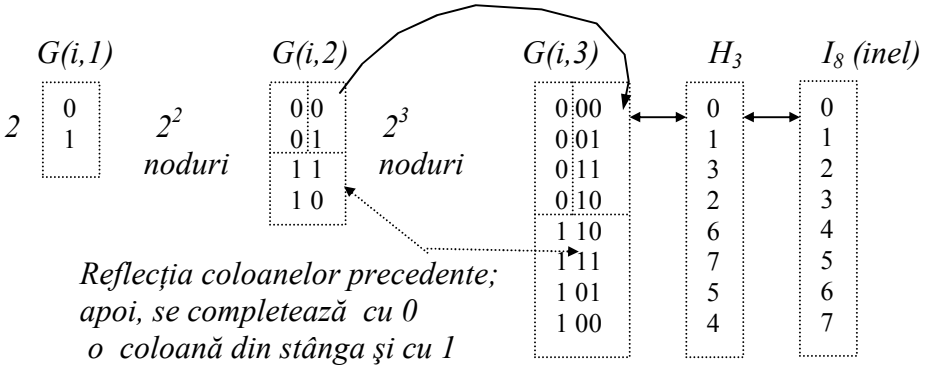


Fig. 2.26. Codul Gray

Procedeul de construire a codurilor Gray este exemplificat pe un caz particular, respectiv un inel cu $2^3=8$ $(n=3)$ noduri pe care să le identificăm cu numere de la 0 la 7. Trebuie să realizăm o corespondenţă biunivocă între un

nod *i* al unui inel cu *8* procesoare şi un nod *G(i,n)* adică *G(i,3)* al hipercubului H_3. Valorile *G(0,3), G(1,3), ... ,G(7,3)* se obţin recursiv pe baza definiţiei, dar se observă un procedeu simplu de stabilire a acestor valori, aşa cum se vede în diagrama din fig. 2.26. Prin săgeata de sus, din diagramă, am indicat faptul că se copiază codul Gray precedent, adică coloanele precedente, pentru a forma codul Gray curent.

Figura 2.27 ilustrează modelarea unei reţele inel cu *8* noduri (procesoare) într-un hipercub *3*-dimensional, H_3. Inelul este indicat prin săgeţi punctate.

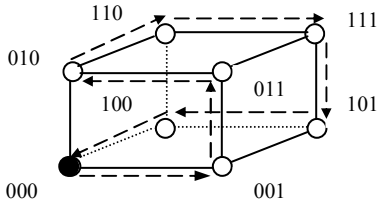


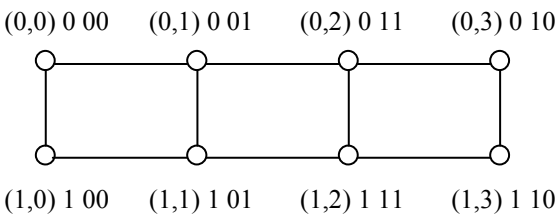
Fig. 2. 27. Modelarea unui inel cu 8 noduri

2.3.3.2. Modelarea unei reţele cu structură de masiv într-un hipercub

Modelarea unei reţele cu structură de masiv (rectangulară) într-un hipercub este o extensie naturală a modelării unui inel într-un hipercub.

Putem modela un masiv de dimensiune $2^r \times 2^s$ într-un hipercub cu 2^{r+s} noduri (procesoare) prin transformarea nodului (procesorului) *(i,j)* în nodul *G(i,r)||G(j,s)*, unde || denotă concatenarea celor două coduri Gray. În acest fel, două procesoare vecine din masiv sunt transformate în hipercub în procesoare a căror etichete diferă exact printr-o poziţie binară, deci vor fi, de asemenea, vecine.

Să considerăm în fig. 2.28 un exemplu de masiv de dimensiune 2×2^2, adică cu $2 \times 4 = 8$ noduri pe care să-l modelăm (reprezentăm) într-un hipercub cu $2^{1+2}=8$ procesoare. Avem *r=1, s=2*. Procesorul *(i,j)* al masivului considerat se transformă în procesorul *G(i,1)||G(j,2)* al hipercubului H_3. De aceea, procesorul *(0,0)* al masivului este transformat în procesorul *000* al hipercubului. Analog, procesorul *(0,1)* al masivului este transformat în procesorul *001* al hipercubului ş.a.m.d.



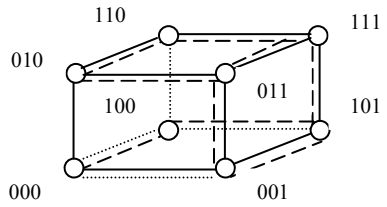


Fig. 2.28. Modelarea unui masiv într-un hipercub

Observăm proprietatea că toate procesoarele de pe o aceeaşi linie a masivului sunt transformate în hipercub în procesoare a căror etichete au cei mai semnificativi (din stânga) *r* biţi identici. Fiecare linie orizontală a masivului conţine 2^s *(s=2)* procesoare şi se transformă într-un subcub H_2 al hipercubului H_3 . Analog fiecare linie verticală (coloană) conţine 2^r *(r=1)* procesoare şi se transformă într-un subcub H_1 al hipercubului.

2.3.3.3. Modelarea arborelui binar într-un hipercub

Un arbore binar conţine o rădăcină la nivel *0*, iar în stânga şi dreapta fiecărui nod de la un nivel *i,* avem câte un arbore binar. Dacă fiecare nod are doi succesori, atunci la nivelul *i* vor fi 2^i noduri, iar arborele binar este *complet* şi se spune că are *adâncimea i+1*. Nodurile care nu au succesori le numim noduri terminale. Să considerăm un arbore de adâncime *d+1* şi care conţine procesoare numai în nodurile terminale. Modelarea acestui arbore binar într-un hipercub cu 2^d procesoare se face astfel:

a) Un nod oarecare al hipercubului devine rădăcină a arborelui binar;

b) Pentru fiecare procesor *i* de adâncime *j* nodul succesor stâng reprezintă procesorul *i* din arbore, iar nodul succesor drept va corespunde nodului din hipercub a cărui etichetă se obţine prin inversarea biţilor, adică *1* în *0* şi *0* în *1,* de pe poziţia *j* (nivel) de la dreapta la stânga din eticheta nodului stâng.

Exemplu. Să considerăm *d=3,* deci un arbore binar complet de adâncime *4* care va avea 2^3 noduri terminale. Figura 2.29 ilustrează această modelare.

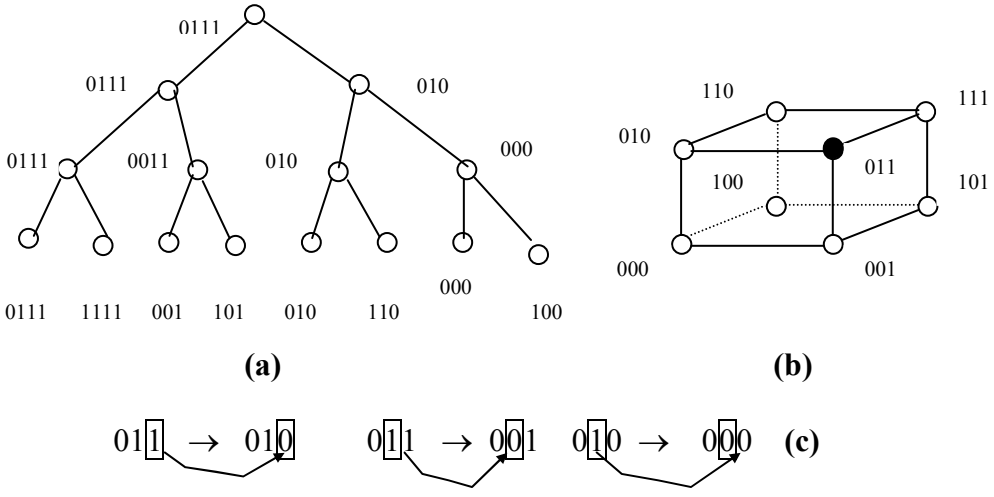
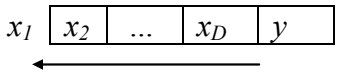


Fig. 2.29. Modelarea unui arbore binar (a) într-un hipercub (b). La (c) se vede cum se obţin etichetele nodurilor de la nivelele *1* şi *2*

În hipercubul H_3 se pot marca muchiile care corespund arcelor spre nodurile din dreapta de la nivelele *1,2,3* pentru a avea modelarea corespunzătoare.

2.3.4. Reţele De Bruijn

Vom defini în cele ce urmează o structură de reţea *de Bruijn* orientată (se pot considera şi reţele *de Bruijn* neorientate). Identificarea nodurilor (procesoarelor) se va face prin secvenţe de simboluri de lungime *D* peste un alfabet *A* de *d* simboluri. Există posibilitatea de a identifica d^D noduri. Vom nota reţeaua în acest caz cu *B(d,D)*. Dacă considerăm un nod identificat de secvenţa $x_1x_2...x_D$, se obţine secvenţa care identifică un nod succesor al acestuia, respectiv secvenţa $x_2...x_D\ y$ obţinută prin decalarea spre stânga cu un simbol a secvenţei date



şi introducând la dreapta toate simbolurile *y* ale alfabetului *A*. Un nod are *d* succesori la care putem transfera mesajele şi *d* noduri predecesoare de la care putem primi un mesaj. O proprietate remarcabilă a unei reţele *de Brujin* orientate este că diametrul său este egal cu *D* şi există un unic drum de lungime exact *D* între oricare două noduri. Să considerăm *A={0,1},* deci d=2 şi
să reprezentăm reţelele *B(2,1), (*adică *D=1), B(2,2)(*adică *D=2), B(2,3) (*adică *D=3).*

Ruta unui mesaj într-o reţea *de Bruijn* între nodul sursă $x_1x_2...x_D$ şi nodul destinaţie $y_1y_2...y_D$ se stabileşte automat astfel: succesorul nodului sursă va fi $x_2x_3...x_Dy_1$ iar acesta va trimite mesajul succesorului acestuia $x_3x_4...x_Dy_1y_2$ ş.a.m.d. Acest rutaj nu foloseşte în mod obligatoriu cel mai scurt drum între cele două noduri. Dacă dorim drumul optim, procesorul iniţial începe prin a căuta cel mai lung sufix al secvenţei $x_1x_2...x_D$ care să fie un prefix al secvenţei $y_1y_2...y_D$. Să presupunem că acest sufix este pentru indicele *j,* avem deci:

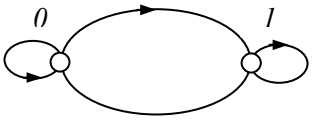
$$x_1x_2...x_D = x_1x_2...x_{j-1}\ y_1y_2...y_{D-j+1}$$

Vom avea, deci, un drum de lungime *j-1* pentru a ajunge la destinatar.

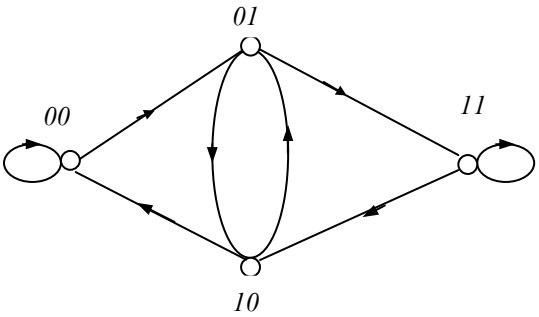
Observaţie. Păstrând etichetele reţelelor *de Bruijn* şi renunţând la orientarea liniilor de comunicaţii obţinem reţele *de Bruijn* neorientate.

În acest caz se constată uşor că sunt adevărate ***proprietăţile***:

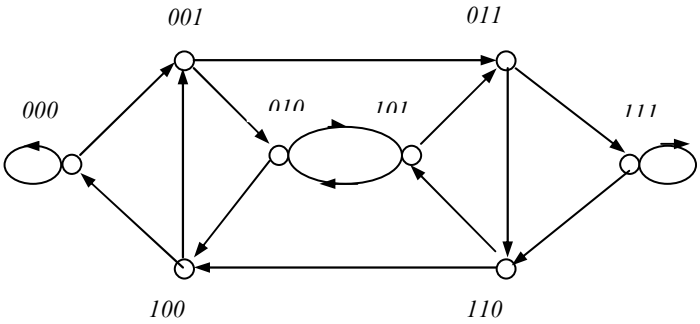
- Diametrul unei reţele *de Bruijn* neorientate *B(d,D)* este *d*, adică între oricare două noduri există o cale de lungime cel mult *d;*
- Fiecare nod, în afară de primul identificat cu *0,* respectiv ultimul identificat cu 2^D-1 are exact patru legături;
- O reţea *de Bruijn* are $2^{D+1}-2$ legături orientate între noduri distincte.



A={0,1}, adică d=2; D=1; respectiv B(2,1)



A={0,1}, adică d=2; D=2; respectiv B(2,2)



A={0,1}, adică d=2; D=1; respectiv B(2,3)

Fig. 2. 30. Reţele De Brujin

CAPITOLUL 3

COMUNICAREA ÎN SISTEMELE DISTRIBUITE

3.1. Codificarea datelor

Conceptul de codificare este cunoscut de foarte mult timp. Problema codificării a apărut din necesități practice legate de transmiterea unor mesaje, în particular secrete, care să poată fi înțelese numai de persoanele cărora le sunt destinate.

Transpunerea informației sub diferite forme numerice, a căpătat o importanță mare odată cu folosirea pe scară largă a calculatoarelor.

Ori de câte ori este vorba de o anumită codificare trebuie să existe o mulțime de ***simboluri primare*** care urmează să fie codificate; fie acestea:

$$S= \{s_0, s_1, \ldots, s_n\}.$$

Mai trebuie să existe, de asemenea, o mulțime de secvențe dintr-un anumit limbaj, care vor reprezenta coduri ale simbolurilor mulțimii *S*. Considerăm alfabetul peste care este definit limbajul respectiv ca fiind

$$A = \{a_0, a_1, \ldots, a_n\}$$

şi pe care îl vom numi ***alfabet al codificării.***

Notăm mulțimea secvențelor de lungime *n*, care se pot forma cu simboluri din mulțimea *A*, adică secvențe de forma $w=a_{i_1} a_{i_2} \ldots a_{i_n}$ astfel:

$$A^n = \{w \mid w=a_{i_1} a_{i_2} \ldots a_{i_n},\ a_{i_j} \in A;\ j=\overline{1,n}\}\ ;$$

Notăm, de asemenea,

$$A^+ = A \cup A^2 \cup \ldots \cup A^n \cup \ldots$$

Definiţia 3.1. *Aplicaţia injectivă*

$$f : S \to A^+$$

unde S, A^+ *au semnificaţiile de mai sus, se numeşte codificare a simbolurilor din S.*

Definiţia 3.2. *Aplicaţia injectivă*

$$g: S \to A^n$$

defineşte o codificare uniformă a simbolurilor din S. În acest caz lungimea codurilor este n (aceeaşi).

Observaţie. Codurile care nu au aceeaşi lungime se numesc *neuniforme.*

Exemple.

1) Fie $B_2=\{0,1\}$, $B_8=\{0,1,...,8\}$, $B_{16}=\{0,1,...,9,A,B,C,D,E,F\}$. Funcţiile

$$f_1 : B_8 \to B_2^3 ; \qquad f_2 : B_{16} \to B_2^4$$

reprezintă exemple de coduri uniforme prin care, în primul caz, unei cifre octale îi punem în corespondenţă o secvenţă de trei cifre binare, respectiv, în al doilea caz, unei cifre hexazecimale îi punem în corespondenţă o secvenţă de patru cifre binare.

2) Alfabetul Morse constituie un exemplu de codificare ne-uniformă pentru litere, cifre şi alte câteva semne.

Problema codificării este strâns legată de problema inversă a acesteia numită *problema decodificării.* Dacă prin aplicaţia f am codificat un simbol $a \in S$ obţinând $f(a)= w \in A^+$, cel care cunoaşte secvenţa w îşi pune întrebarea, care este simbolul a codificat. Acest lucru se poate stabili dacă cunoaştem funcţia f şi dacă aceasta este bijectivă.

Fiind date mulţimile S, A cu semnificaţiile de mai sus şi aplicaţia injectivă $f: S\to A^+$, atunci aplicaţia $g: f(S) \to S$ cu proprietatea $f \circ g = 1$ o numim *decodificare* a codificării date. Funcţia g este *bijectivă*.

3.1.1. Codificarea datelor nenumerice

Orice informaţie, în cele mai multe situaţii, e necesar să fie scrisă; pentru redarea ei se folosesc caractere (simboluri) ce pot fi tipărite sau afişate (se găsesc la orice imprimantă a unui calculator sau sunt prezente pe tastatură). De asemenea, pentru aranjarea textului în pagină cu ajutorul imprimantelor sau a monitoarelor se folosesc caractere care provoacă îndeplinirea anumitor funcţii (acţiuni) speciale.

În general, aceste caractere sunt:

- literele mari şi mici ale alfabetului latin: { *A, ... , Z, a, ..., z* } ;
- cifre: {*0, 1, ..., 9*} ;
- semne speciale {*+, -, *, /, spaţiu*, $, @ , …} ;
- funcţii speciale ale tastelor, respective: *TAB, CR*, … .

Notăm cu *S* mulţimea acestor caractere, adică:

S={ A, B,...,Z, a, b,..., z, 0, 1, 2, ..., 9,+, ... , $, ...}

Cea mai folosită codificare a tuturor acestor caractere este codificarea definită de funcţii injective de felul următor:

$C : S \rightarrow [o, n]$ unde $[0,n]=\{0,1,2, \dots , n\}$

Domeniul valorilor acestei funcţii este mulţimea numerelor întregi zecimale de la *0* la *n* sau întregi hexazecimali de la *0* la *m*, $m_{16}=n_{10}$ (adică *n)*.

Un astfel de cod este **codul ASCII-7** *(**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)*, adică $C:S \rightarrow [0,127]=[0,7F]$. Întregii din mulţimea $[0,127]=\{0,1, \dots ,2^7-1\}$ se reprezintă binar sub forma *aaabbbb, a,b*$\in\{0,1\}$, adică *oh;* $o \in B_8=\{0,1,2,3,4,5,6,7\}$, $h \in B_{16}=\{0,1,2, \dots ,9,A,B,C,D,E,F\}$ şi în acest caz fiecărui simbol (caracter) îi corespund *7* cifre binare; de exemplu $L \rightarrow 76_{10}=4C_{16} = \underbrace{100}_{4}\ \underbrace{1100}_{C} = 1001100.$

Dacă se foloseşte extinderea **ASCII-8** a acestui cod, adică reprezentarea pe *8* biţi, atunci *n=255,* deci avem *256* de semne, $255=2^8-1$, iar $n=m_{16}=FF$; adică $C:S \rightarrow [0,255]=[0,FF]$. Întregii din mulţimea $[0,255]=\{0,1, \dots ,2^8-1\}$ se reprezintă binar sub forma *hh=aaaabbbb,* adică *0aaabbbb (*ASCII-7) +(la care se adaugă) *1aaabbbb* (extindere); $h \in B_{16}=\{0,1,2, \dots ,9,A,B,C,D,E,F\}$, $a,b \in \{0,1\}$ şi în acest caz fiecărui simbol (caracter) îi corespund *8* cifre binare; de exemplu: caracterului *â* îi corespunde $226_{10}=E2_{16}$. Dacă considerăm reprezentarea binară a codului ASCII, avem un cod uniform pe *7* biţi (*n=127),* respectiv pe *8* biţi (*n=255).*

Codul ***ASCII*** îndeplineşte următoarele condiţii:

1. dacă simbolul din *S* este o cifră, literă mică sau mare a alfabetului latin, atunci, codul său este mai mic decât codul pentru spaţiul dintre aceste caractere;

2. codurile literelor mici şi mari ale alfabetului latin, aşa cum îl cunoaştem pe acesta, sunt în ordine crescătoare;

3. pentru cifrele de la *0* la *9,* codurile sunt consecutive şi crescătoare.

Prin intermediul codului ASCII-8 putem folosi la un calculator un număr destul de mare de semne, respectiv *256* de caractere. Nu acesta a fost, însă, primul sistem de codificare a caracterelor la un calculator; a existat altul, numit codul EBCDIC (nu se mai foloseşte!) într-o grilă de *16* pe *16* (*256* semne), deci cod cu secvenţe de *8* biţi care codifică $2^8 = 256$ semne.

Aplicaţiile recente realizate pe calculatoare, au impus necesitatea codificării unui număr mult mai mare decât *256* de semne. Astfel, au apărut codurile ***UNICODE*** pe *16* biţi, de forma *hhhh,* $h \in B_{16}$ ca o extensie a codului *ASCII-8* (în secvenţa hexazecimală *00hh* avem *hh* cod ASCII-8) şi cu care se pot codifica $2^{16} = 65536\ (=256 \times 256)$ caractere diferite, precum şi ***ISO/IEC 10646*** pe *32* de biţi, de forma *hhhhhhhh,* $h \in B_{16}$, în secvenţa hexazecimală *0000hhhh* avem *hhhh* cod UNICODE.

3.1.2. Criptarea datelor

Trimiterea unor informaţii "secrete" imposibil de descifrat, adică sub formă *criptată*, mai ales în domeniul militar a fost şi este o problemă importantă din punct de vedere practic. Să precizăm pe scurt şi într-o formă simplificată problema criptării.

Informaţiile iniţiale reprezintă *mesaje* adică secvenţe de caractere.

Fie un alfabet sursă $A=\{a_1, a_2, \ldots, a_n\}$. Orice mesaj *m,* adică *text sursă* (necriptat) va fi un cuvânt, deci o secvenţă din A^+, $m \in A^+$. Criptarea presupune transformarea acestui text sursă prin intermediul unui *algoritm de criptare* într-un text criptat (numit uneori *cifrat*) peste alfabetul *A*. Un algoritm de criptare foloseşte , de regulă, o cheie de criptare *k* care este ţinută secretă. În acest fel *criptarea* este o funcţie bijectivă

$$C_k: A^+ \to A^+$$
$$A^+ \ni s=s_0s_1 \ldots s_n \to c=c_0c_1 \ldots c_m = C_k(s) \in A^+$$

Inversa acestei funcţii este numită *decriptare.* Există două tipuri de algoritmi bazaţi pe nişte chei care definesc: sisteme cu ***cheie privată*** şi sisteme cu ***cheie publică***.

Sistemele cu ***cheie privată***, numite şi ***simetrice***, utilizează o cheie unică pentru criptare şi decriptare. Acest sistem prezintă dezavantajul că această cheie publică trebuie comunicată tuturor destinatarilor.

Sistemele cu ***cheie publică, asimetrice*** utilizează două chei: o cheie publică care permite să cripteze nişte mesaje şi o *cheie secretă* utilizată pentru decriptare. Cheia publică poate fi difuzată fără probleme, pe când cheia privată/secretă nu este cunoscută decât de un singur destinatar. Algoritmii de criptare şi decriptare folosiţi în acest caz sunt utilizaţi, de asemenea, pentru generarea unor ***semnături digitale,*** care permit autentificarea autorului unui mesaje. În acest caz cheia secretă este utilizată pentru decodarea semnăturii (un mic bloc de date) şi cheia publică pentru validare.

3.2. Comunicaţii în sisteme distribuite

3.2.1. Elementele de bază ale comunicării

Comunicarea presupune realizarea unor legături între nişte noduri ale unui sistem distribuit. În orice comunicaţie deosebim, indiferent de natura ei, următoarele ***elemente de bază***:

a) *transmiţător* - poate fi un dispozitiv sau o persoană care trebuie să transmită un mesaj la o anumită destinaţie;

b) *receptor* - însemnă un dispozitiv sau o persoană capabilă de a recepta, primi, mesaje comunicate de un transmiţător;

c) *mediul de transmisie* – calea, respectiv mijlocul prin intermediul căruia se transmit mesaje de la a) la b); ca mijloace şi căi de transmisie avem: fire electrice, cablu coaxial, fibră optică, unde radio-electronice, satelit;

d) *zgomot* - orice mijloc fizic ce interferează cu mijlocul de comunicaţie reprezintă zgomot în sistemul de comunicaţie.

În general, într-un calculator, datele sunt reprezentate sub formă codificată şi transmiterea lor se face tot în această formă codificată; în principiu ca succesiune de cifre binare *0, 1*. În ceea ce priveşte transmisia datelor într-un sistem de comunicaţie, aceasta se poate face în două moduri:

a) *transmisie paralelă* – mediul foloseşte câte un fir pentru fiecare bit din codul unui caracter şi un fir pentru aşa-numitul impuls de ceas. Această metodă de transmisie asigură o viteză foarte mare, dar nu e uşor de realizat în practică în cazul distanţelor mari (se foloseşte pentru distante mici).

b) *transmisie serială* – mediul constă în acest caz din două fire: unul prin care se transmit datele bit după bit şi celalalt constituie un aşa-numit fir de masă.

Observaţie. Dacă folosim ca mediu de transmisie circuitele telefonice atunci e nevoie să folosim nişte modem-uri (modem = o interfaţă fizică între calculator şi postul telefonic, respectiv, invers).

În transmisia serială se folosesc trei moduri de transmisie: *asincronă*, *sincronă*, *isocronă*.

a) Transmisia asincronă. În acest caz, emiţătorul transmite un caracter ori de câte ori acest caracter este pregătit pentru a fi transmis(fig.3.1).

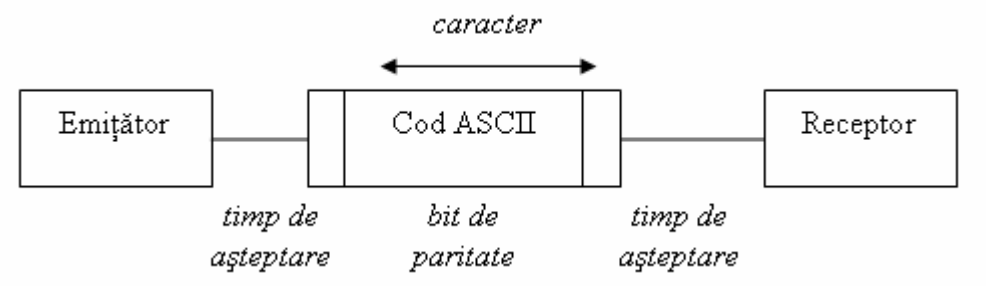


Fig. 3.1. Transmisia asincronă

După un anumit *timp de aşteptare* se lansează transmisia unui caracter care presupune în principiu o structură de felul următor: *codul ASCII + bit de paritate,* adică *codul ASCII* al caracterului şi după un timp de aşteptare acest caracter, prin codul său ajunge la receptor.

Bitul de paritate se foloseşte pentru a asigura un control al transmisiei de caractere. Bitul de paritate poate fi, de exemplu, restul împărţirii la *2* a sumei biţilor din codul ASCII.

b) Transmisie sincronă. O transmisie sincronă presupune transmiterea unui *bloc de caractere* - o succesiune de coduri ASCII ale unor caractere (fig.3.2).

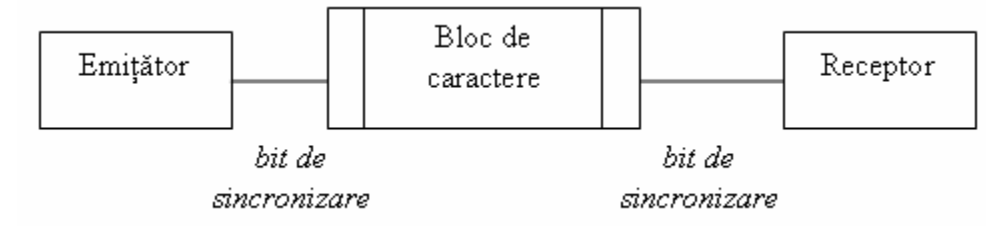


Fig.3.2. Transmisia sincronă

Blocul de caractere are la începutul şi sfârşitul său câte un *bit de sincronizare*.

Observaţie. Cele două moduri de transmisie sunt cunoscute şi sub denumirea de *transmisie mod caracter,* respectiv *transmisie mod mesaj*.

c) Transmisia isocronă. Transmisia isocronă este o combinaţie între transmisia asincronă şi transmisia sincronă, făcută cu scopul reducerii timpului de aşteptare în ce priveşte transmisia pe care o realizăm.

Observaţie. Orice informaţie care se transmite dintr-un nod în altul este o succesiune de biţi. Transmiterea se face la nivel de *semnale* care presupun durată, adică intervale de timp, egale. Transportul semnalelor este supus unor constrângeri relative la frecvenţa posibilă pe un mediu de comunicaţie. De exemplu pe o linie telefonică transmisia se face cu frecvenţe cuprinse între *300* la *3400* hertz (Hz), banda de frecvenţa fiind *3400 – 300=3100 Hz.* Această bandă se mai numeşte şi ***bandă de trecere.*** Un cablu coaxial are o bandă de trecere de *500MHz,* iar o fibră optică de *3.3 GHz.* O bandă de trecere a unei căi de comunicaţii este o caracteristică importantă a acesteia, ea indicând ***capacitatea de transmisie*** a acesteia. Dacă notăm cu *W* banda de trecere (hertz) a unei căi de comunicaţii şi cu *n* numărul de biţi / semnal, atunci capacitatea unei căi de comunicaţii s-a demonstrat că este dată de relaţia *C=2Wn biţi/secundă (bps).* Utilizatorilor le sunt atribuite părţi ale unei benzi de trecere contra unor taxe.

3.2.2. Protocoale

Transmisia datelor impune totdeauna utilizarea unor reguli de transmisie numite ***protocoale***. Acestea trebuie să asigure o succesiune corectă a datelor transmise precum şi integritatea lor. Este vorba de fapt de noţiunea de protocol de transmisie a datelor într-o reţea. Există protocoale pentru fiecare din tipurile de transmisii *a), b)* şi *c)*.

Definiţia 3.3. *Un protocol este mulţimea regulilor care trebuie să fie respectate pentru realizarea unui schimb de informaţii între calculatoare.*

Protocolul pentru transmisia asincronă e foarte simplu; se compune din biţi de *start* şi *stop* ai transmisiei şi structura caracterelor care se transmit. Pentru o astfel de transmisie nu se impune considerarea unor reguli speciale.

Pentru transmisia sincronă se folosesc diferite protocoale de comunicare (ele sunt mai complicate). Aceste protocoale se definesc prin structura şi funcţiile pe care le realizează. În general structura lor se referă la două elemente de bază care o compun:

- nişte *caractere de control* care sunt secvenţe de biţi ce furnizează anumite informaţii despre procesul de comunicare şi comunicaţia în sine;

- *textul* care reprezintă datele ce se transmit.

Funcţiile protocoalelor sincrone:

1. sincronizarea – funcţia prin care se pun de acord transmiţătorul *T* si receptorul *R* asupra fiecărui bit transmis; e nevoie să se cunoască exact unde începe un bit şi unde se termină acesta.

2. formatarea – asigură un anumit format mesajului transmis. De regulă acesta cuprinde trei părţi:

- prima parte se refera la *capul (header-ul) mesajului* care conţine adresa receptorului;
- apoi, *date* privind succesiunea caracterelor în blocul transmis şi informaţii de control;
- *textul* care conţine blocul de caractere ce se transmite şi eventual anumite reguli de verificare folosite în procesul de transmitere de date;

3. utilizarea liniei prin care se caută asigurarea unei eficienţe maxime privind tipul de canal de comunicaţie;

4. punerea de acord în ce priveşte legătura dintre *E* şi *R* care îşi pot inversa rolurile, realizând astfel un dialog asupra corectitudinii transmisiei;

5. verificarea şi corecţia erorilor, dacă acest lucru este posibil.

Orice protocol s-ar realiza pentru o transmisie serială, este nevoie ca el să realizeze aceste funcţii.

3.3. Tehnici de comutaţie a canalelor de comunicaţie

În sistemele de comunicaţie, legătura dintre emiţător si receptor se poate face fie direct, când acest lucru este posibil (transmisia e cea mai eficientă în acest caz), fie prin tehnici de comutaţie a canalelor de comunicaţie.

Există trei tehnici de comutaţie de bază şi rutarea traficului într-o reţea de comunicaţii (rutarea presupune realizarea unei succesiuni de noduri din sistemul distribuit de la *E* la *R*).

1. Comutaţia de circuite – în acest caz trebuie să existe un centru de comunicaţii care stabileşte o legătură directă între emiţător şi un receptor, dar este evident că este vorba de o succesiune de noduri prin care trebuie să se treacă de la *E* la *R*; trecerea se face complet, apoi legătura este întreruptă (linia de comutaţie) şi aşteaptă o altă cerere; uneori această comutaţie e folosită şi pentru rutare: odată stabilită o rută traficul între emiţător şi receptor se realizează doar pe acea rută.

2. Comutaţia de mesaje – fiecare mesaj este transmis în sistem şi rutat (dirijat) ca o unitate distinctă la destinaţia prevăzută în mesajul respectiv. O succesiune de mesaje cu aceeaşi sursă (*S*) şi destinaţie (*D*) poate fi transmisă în acest caz pe rute diferite; traseele de la *S* la *D* sunt diferite. Acest lucru e posibil datorită faptului că fiecare centru de comutaţie este capabil să facă o analiză a informaţiilor de control cuprinse în mesaj şi să decidă care e următorul nod la care trebuie să ajungă mesajul în drumul său de la *E* la *R*.

3. Comutaţia de pachete – este o formă derivată a comutaţiei de mesaje şi diferă de aceasta prin faptul că mesajele în acest tip de comutaţie au o aceeaşi lungime; mesajele cu această proprietate le numim ***pachete***; un pachet într-un sistem de comutaţie poate să însemne o parte dintr-un mesaj divizat în mai multe pachete sau un pachet conţine mai multe mesaje. Pachetele sunt analizate individual de către centrele de comutaţie şi sunt transmise pe ruta cea mai convenabilă către destinaţia lor.

Acest mod de comutaţie este preferat în practică, în general din considerente tehnice. Este mult mai uşor să se gestioneze într-un sistem distribuit segmente de lungime fixă ce reprezintă pachete decât blocuri de caractere de lungimi variabile.

Exemplu. În cele ce urmează se dă un exemplu de schemă de comutaţie de pachete într-o reţea numită ARPANET (printre primele reţele de calculatoare din lume); transmisia între două noduri, unul receptor, celălalt emiţător.

Facem următoarele convenţii de notaţii şi reprezentări în schema ce urmează:

HOST – un calculator gazdă de prelucrare a datelor;

IMP- un procesor de interfaţă pentru mesaj;

IMP – transformă mesajele în pachete şi invers, în raport cu situaţia dorită; acesta trebuie să aibă o structură clar definită şi recunoscută în reţea, de exemplu să fie clar spre ce destinaţie sunt dirijate anumite mesaje în reţea şi de unde se primeşte întotdeauna confirmarea primirii.

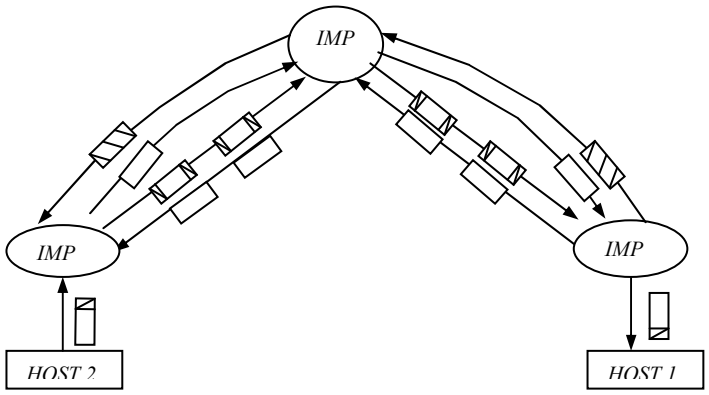


Fig. 3.3. Schema comutaţiei de pachete

Procedura de transmitere a unor mesaje este următoarea (fig.3.3):

- se lansează o cerere de la *HOST 1* către *HOST 2;*
- se produce confirmarea primirii cererii de la *2* la *1;*
- se formează pachetele care reprezintă mesajul solicitat;
- pentru fiecare pachet transmis – câte o confirmare a primirii acestuia la destinaţie;
- IMP al lui *1* face compunerea pachetelor în mesaj.

Un exemplu de structură a unui mesaj transmis este dat în fig. 3.4.

SOH	HEADER	STX	Text	ETB

Fig. 3.4. Structura unui mesaj transmis

HEADER – capul propriu-zis al mesajului;
SOH – marchează startul din capul mesajului;
STX – startul spre secvenţa de ***text*** care conţine informaţiile popriu-zise ale mesajului;
ETB – sfârşitul blocului de transmisie al mesajului.

Pentru confirmarea primirii mesajelor este nevoie ca să folosim anumite câmpuri speciale.

3.4. Transmiterea mesajelor într-o reţea de calculatoare

3.4.1. Modelul client/server în reţelele de calculatoare

Se numeşte *reţea de calculatoare* un sistem de comunicaţii care leagă mai multe noduri gazdă, numite adesea *host-uri,* dispersate geografic şi care permit schimburi de informaţii prin intermediul unor mesaje. Unele noduri dispun de sisteme de interfaţă numite de multe ori *controllere*, cu ajutorul cărora se realizează o conexiune la reţea.

Reţelele pot fi:

- ***reţele locale*** – ***LAN*** *(Local Area Networks),* care acoperă în general distanţe de *1Km,* localizate de regulă într-o singură clădire;
- ***reţele metropolitane*** – ***MAN*** *(Metropolitan Area Networks),* care acoperă în general distanţe în jur de *10 Km;*
- ***reţele întinse*** (inter)naţionale – ***WAN*** *(Wide Area Networks*).

Într-o reţea, nodurile acesteia pot juca roluri diferite; pot fi specializate pentru anumite servicii, ca de exemplu: server de fişiere, server Web sau au o utilitate generală (servicii diferite).

Modelul dominant ce se utilizează în reţelele de calculatoare (aplicaţii distribuite) este *modelul* ***client / server***.

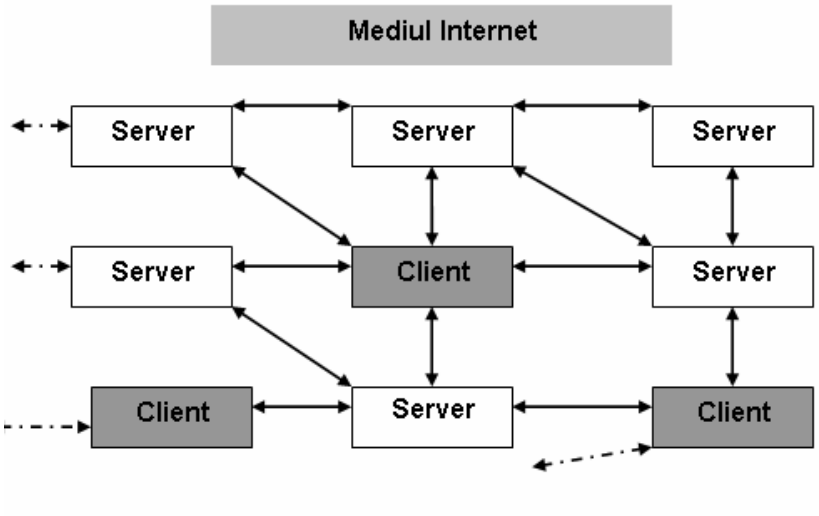


Fig. 3.5. Un model client/server în mediul Internet [DZI06]

Serverul este un program numit *program-server,* care porneşte primul şi realizează:

- deschiderea unui canal de comunicare la o adresă (locaţie) determinată;
- programul rămâne în aşteptare până la sosirea unei cereri de la un client;
- prin prelucrarea cererii o acceptă sau nu şi trimite un răspuns clientului.

Altfel, serverul:

……..

-primeşte(cerere) de la client;

-execută operaţia cerută;

-trimite(răspuns) la client;

……

Clientul este un program numit *programul-client,* care realizează acţiuni similare. Adică,

Clientul:

……

-trimite(răspuns) la client;

-trimite(răspuns) la client;

-trimite(cerere) la server;

-primeşte(răspuns);

……

Reţelele individuale pot fi interconectate formându-se inter-reţele, adică o reţea de reţele. Un exemplu important de inter-reţea, o reţea a tuturor reţelelor, este cea cu numele INTERNET devenită atât de populară în întreaga lume.

Începuturile INTERNET-ului (1969) se leagă de realizarea programului de cercetare: DARPA (Defense Advanced Research Projects Agency) pentru Departamentul Apărării din SUA. Principala caracteristică a Internet-ului este heterogenitatea (calculatoare de tipuri diferite cu sisteme de operare diferite).

În fig. 3.5 prezentăm un model client/server, care apare şi în [Dzi06].

3.4.2. Protocoale de comunicaţii

Analogie. Comunicarea între nodurile unei reţele de calculatoare se realizează pe baza unor reguli foarte bine precizate şi care sunt stabilite prin intermediul unor protocoale. Ele se bazează pe analogia cu alte sisteme, de exemplu, sistemul producător / consumator sau furnizor (serviciu) / client care presupun o interconectare într-un sistem deschis.

Sistemul furnizor / client constă în următoarele: un furnizor, ca urmare a unei cereri primite de la un client, trebuie să trimită un pachet (colet) de dimensiuni mici, clientului, la o anumită destinaţie (vezi fig. 3.6).

De regulă, o astfel de cerere se rezolvă prin intermediul unei ierarhii cu trei niveluri: nivelul furnizor, respectiv, nivelul firmă de transporturi (expediţii) şi apoi, nivelul Societate transportatoare, de exemplu, auto sau feroviară. Această divizare pe niveluri are la bază ca motivaţie ideea de specializare a firmelor în ce priveşte realizarea unor servicii de către acestea în condiţii de eficienţă.

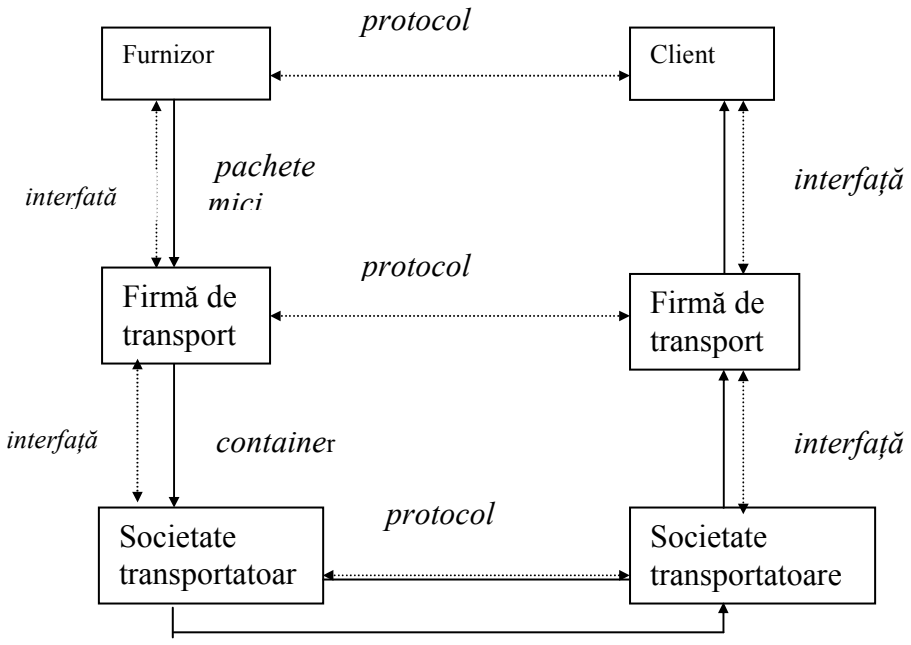


Fig. 3.6. Un sistem de tip furnizor / client

Între client şi furnizor se stabileşte o legătură în baza unui protocol, pentru satisfacerea cererii solicitate. Furnizorul confirmă rezolvarea cererii şi începe demersurile cu o firmă de transporturi (expediţii), firma *1,* cu care are stabilită o interfaţă. Firma *1* preia setul de pachete mici de la furnizor şi împreună cu alte pachete de la alţi furnizori formează un container standard care să fie expediat unei alte firme de transport, firma *2*, care asigură servicii

pentru clientul în cauză. Această relaţie este stabilită în baza unui protocol între cele două firme de transporturi. Transportul acestui colet este asigurat efectiv de societăţi transportatoare (trenuri, camioane, avioane etc). Între firma *1* de transporturi şi o societate de transportatoare există o interfaţă, cum şi între firma *2* de transporturi şi eventual o altă societate transportatoare există o interfaţă de colaborare. Evident, o relaţie de colaborare, stabilită printr-un protocol, trebuie să avem şi între cele două eventuale societăţi transportatoare.

3.4.3. Modelul OSI, model de referinţă. Protocoalele TCP/IP

În ce priveşte comunicarea, protocolul de comunicaţie în reţele de calculatoare are la bază un model de referinţă *OSI (Open Systems Interconnection)* standardizat în *1985,* analog cu cel prezentat anterior. Se are în vedere un calculator sursă care trebuie să trimită mesaje unui alt calculator care reprezentă destinaţia dată a acestor mesaje. Acest model descrie şapte niveluri între care există două feluri de comunicări (relaţii): orizontale - *protocoale* şi verticale - *interfeţe*. Comunicarea pe orizontală între aceleaşi niveluri (straturi) se face prin *protocoale* care stabilesc reguli de schimburi de informaţii reciproce între acestea. Relaţiile verticale între niveluri (straturi) ale aceluiaşi sistem *A* sau *B* sunt prin *interfeţe.* Fiecare nivel dintr-un sistem 0furnizează nivelului superior un anumit număr de primitive. Relaţia de la fiecare nivel între cele două sisteme este independentă de cea a altor niveluri (straturi).

Pentru ca o informaţie să ajungă de la sursa *A* la destinaţia *B* este necesar să parcurgă întâi nivelurile sursei *A*: aplicaţie, prezentare, sesiune, transport, reţea, legătură şi fizic*,* să se folosească eventual un *nod intermediar*, iar apoi în ordine inversă celor prezentate anterior, respectiv să se parcurgă nivelurile nodului destinaţie *B*: fizic, legătură, reţea, transport, sesiune, prezentare şi aplicaţie*,* aşa cum se indică prin săgeţi în desenul din fig. 3.7.

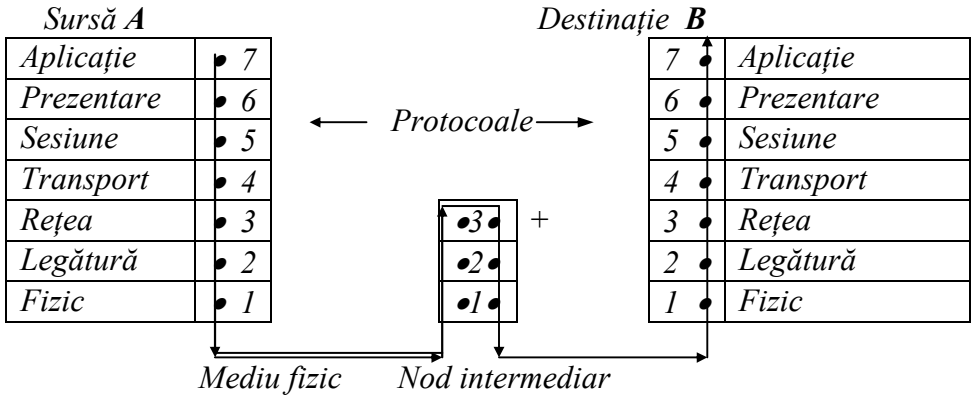


Fig. 3.7. Modelul OSI

Conexiuni între reţele sau părţi de reţele locale pot să fie făcute utilizând:

- un *repetor* (în engleză – repeater) la nivelul *1* din modelul *OSI,* care are singurul scop de a transmite biţii dintr-o reţea în alta;
- o *punte* (în engleză – bridge) la nivelul *2* din modelul *OSI,* care asigură transmisia fizică a pachetelor de biţi în condiţii de siguranţă;
- un *ruter* (în engleză – router) la nivel *3* din modelul *OSI,* adică nivelul reţea, care asigură rutajul în diferite reţele.
- o punte de comandă (în engleză – gateway) care este un dispozitiv ce leagă permanent două reţele între ele oricare ar fi natura acestora. O punte de comandă lucrează în general la nivelul *7* din modelul *OSI*, adică la nivel de aplicaţie.

La nivelul *2* al modelului *OSI,* legăturile se fac pe baza unui protocol numit *protocolul HDLC (High-level Data Link Control)* care defineşte o structură de mesaj, cu un anumit şablon, structură pe care o prezentăm în fig.3.8.

Fanion	*Adresă*	*Comandă*	*Informaţie*	*FCS*	*Fanion*
01111110	*8* biţi	*8* biţi	*n* biţi	*18* biţi	*01111110*

Fig. 3.8. Protocolul HDLC

Semnificaţia fiecărui câmp este:

- *Fanion* – delimitează începutul şi sfârşitul dintr-un şablon;
- *Adresă* – identifică staţia receptoare;
- *Comandă* – indică tipul de şablon şi conţine numărul de pachete (dacă există mai multe);
- *Informaţie* – se compune dintr-o succesiune de *n* biţi, oarecare;
- *FCS (Frame Check Sequence)* – este o ***secvenţă de control*** a şablonului obţinută cu ajutorul unui cod ciclic polinomial ce foloseşte polinomul $G(x)=x^{16}+x^{12}+x^5+1$. În caz de eroare, mesajul se retransmite.

Observaţie. Codurile ciclice polinomiale pentru detectarea erorilor de grup, numite *CRC (Cyclic Redondant Coding),* folosite la transmiterea mesajelor care sunt secvenţe de cifre binare, se bazează pe următoarele considerente. Înainte de transmiterea unui mesaj se adaugă acestuia o secvenţă de control *(FCS)* care se determină în baza unui algoritm. O informaţie de *n* biţi poate fi considerată ca lista coeficienţilor binari ai unui polinom cu *n* termeni de grad *n-1*. De exemplu: $1011 \rightarrow 1\cdot x^3+0\cdot x^2+1\cdot x+1= x^3+x+1$.

În general, la sursă şi destinaţie, se alege un acelaşi polinom *G(x)* numit *polinom generator,* de grad *r,* care va fi folosit la generarea biţilor secvenţei de control.

Algoritmul pentru stabilirea mesajului care se trimite este următorul. Dacă *M(x)* este polinomul corespunzător mesajului original, iar polinomul generator *G(x)* are gradul *r,* atunci:

- se înmulţeşte *M(x)* cu x^r ceea ce înseamnă că adăugăm *r* zerouri la sfârşitul mesajului original;
- se efectuează împărţirea *modulo 2* :

$$\frac{M(x)\cdot x^r}{G(x)} = Q(x) + R(x)$$

- câtul *Q(x)* este ignorat. Restul *R(x),* de grad *r-1,* conţine *r* biţi care sunt coeficienţii acestuia. Se efectuează scăderea *modulo 2,* identică cu operaţia logică *XOR* (SAU exclusiv):

$$M(x)\, x^r - R(x) = T(x)$$

Polinomul *T(x)* este polinomul ciclic corespunzător mesajului care va fi trimis la destinaţie. Se observă că: $T(x)=Q(x)\, G(x)$.

La recepţia mesajului se efectuează:

$$\frac{T(x)}{G(x)}$$

Dacă:

- restul este *0,* atunci nu sunt erori de transmisie;
- restul este $\neq 0$, atunci există erori şi se va retransmite mesajul respectiv.

Protocoalele TCP / IP

Internet-ul rezolvă problema conexiunii între diferite reţele, definind o mulţime de protocoale, precum şi o metodă de interconexiune fizică între reţele. Primul model în ordinea apariţiei a fost *modelul TCM / IP (Transmission Control Protocol / Internet Protocol* – adică – *Protocol de Control al Transmisiei / Protocol Internet)* adoptat ca standard în 1983.

TCP /IP este un ansamblul de programe care rezolvă probleme de comunicare între calculatoare folosind convenţii de interconectare a reţelelor şi dirijare a traficului.

Protocolul *TCP* furnizează un serviciu de transport pentru toate aplicaţiile.

Protocolul *IP* este responsabil cu rutajul de informaţii care traversează reţelele.

Protocolul de comunicaţie *TCP / IP* are la bază modelul de referinţă *OSI* prezentat de mai sus. Din modelul *OSI* nu se utilizează nivelurile *5, 6,* iar nivelurile *1* şi *2* se constituie într-o interfaţă de reţea, în consecinţă modelul de referinţă *TCP / IP* cuprinde patru niveluri aşa cum se observă în fig.3.9.

Niveluri OSI	**Niveluri TCP / IP**
7	*Aplicaţie (ftp, telnet, SMTP ...)*
4	*Transport (TCP, UDP)*
3	*Rutaj (IP)*
1, 2	*Interfaţă reţea (Ethernet ...)*

Fig. 3.9. Protocoalele TCP/IP

UDP (User Datagram Protocol) este un protocol de transport fără conexiune prealabilă.

Un număr important de aplicaţii se bazează pe protocolul *TCP / IP* de nivel înalt ce pot fi considerate ca servicii universale, precum:

- ***Telnet*** – cu ajutorul căruia deschidem o *sesiune* de comunicare la distanţă;
- ***FTP*** *(File Transfer Protocol)*- transfer de fişiere;
- *SMTP (Simple Mail Transfer Protocol)* – program care asigură transfer de mesaje prin poştă electronică între două maşini din Internet;
- *DNS (Domain Name System)* – cuprinde nume de domenii;
- ***HTTP*** *(HyperText Transfer Protocol)* – protocol pentru transfer de hipertext folosit de Web.

Atunci când se transmit date utile într-o reţea Ethernet, în familia de protocoale *TCP/ IP* există încapsulate mai multe *antete* de protocol aşa cum se prezintă acest lucru în figura 3.10.

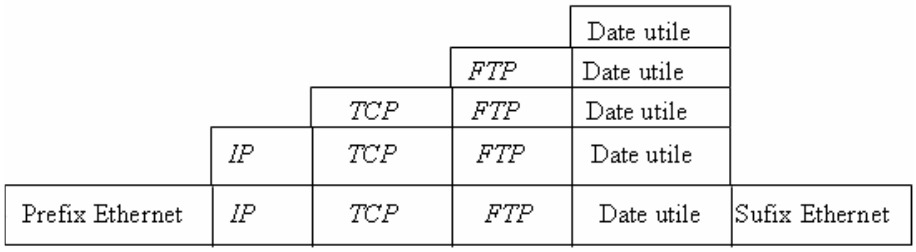


Fig. 3.10. Antete din familia protocoalelor TCP/IP cu reţea Ethernet

3.4.4. Acces în Internet

Din punct de vedere conceptual Internet-ul este o colecţie de grupuri de reţele care formează ***domenii***. Fiecare domeniu are un *nume*, o *adresă simbolică* unică ce are o structură ierarhică, cu cel mult cinci nivele.

Numele de domeniu ce are nivelul cel mai înalt s-a convenit să fie codul unei ţări, dar s-au păstrat din ArpaNet şi alte nume, precum:

- **.com** – organizaţii comerciale;
- **.edu** – organizaţii educaţionale;
- **.gov** – organizaţii guvernamentale din *SUA*;
- **.mil** – organizaţii militare din *SUA*;
- **.org** – alte organizaţii;
- **.net** – pentru administrarea unor resurse de reţea.

Sistemul de Nume de Domenii (DNS)[8] îi ajută pe utilizatori să trimită cu uşurinţă mesaje de poştă electronică (e-mail) şi să navigheze pe Internet. Asemănător cu numerele de telefon, fiecare calculator are o adresă unică pe Internet, numită număr pentru Protocol Internet sau număr IP. Deoarece este dificil să se memoreze aceste numere, a fost creat DNS-ul care permite folosirea numelor de domenii în locul cifrelor, fiind astfel mai uşor de memorat. De exemplu, cu ajutorul DNS-ului, utilizatorii Internet pot găsi o adresă pe Internet doar tastând un nume, cum ar fi "www.internic.net", în loc să tasteze numărul 207.151.159.3.

DNS-ul permite înregistrarea numelor de domenii în cadrul unor registre, cunoscute ca "domenii de nivel superior", sau TLD-uri. La rândul său, fiecare TLD (Top Level Domain), poate să aibă câteva sub-domenii. Astăzi, TLD-urile se pot încadra în două categorii mari: 1) domenii generice de nivel superior (gTLDs) cum ar fi .com, .net, .org şi .info, care sunt deschise pentru înregistrare pentru toţi utilizatorii de Internet din lume şi 2) domenii de nivel superior pe coduri de ţară (ccTLDs), cum ar fi .uk pentru Marea Britanie, sau .ng pentru Nigeria, corespunzător unei ţări, unui teritoriu, sau altei locaţii geografice. În timp ce ambele categorii de domenii de nivel superior funcţionează asemănător din punct de vedere tehnic, regulile şi politicile de înregistrare a numelor de domeniu în gTLD şi în ccTLD pot avea deosebiri semnificative. Pentru informaţii suplimentare în legătură cu structura DNS accesaţi http://www.internic.net/faqs/domain-names.html.

Un exemplu de grupare de reţele pe domenii, reţele utilizate în general de către o singură organizaţie, se dă în fig. 3.11.

[8] http://www.internic.net/faqs/domain-names.html

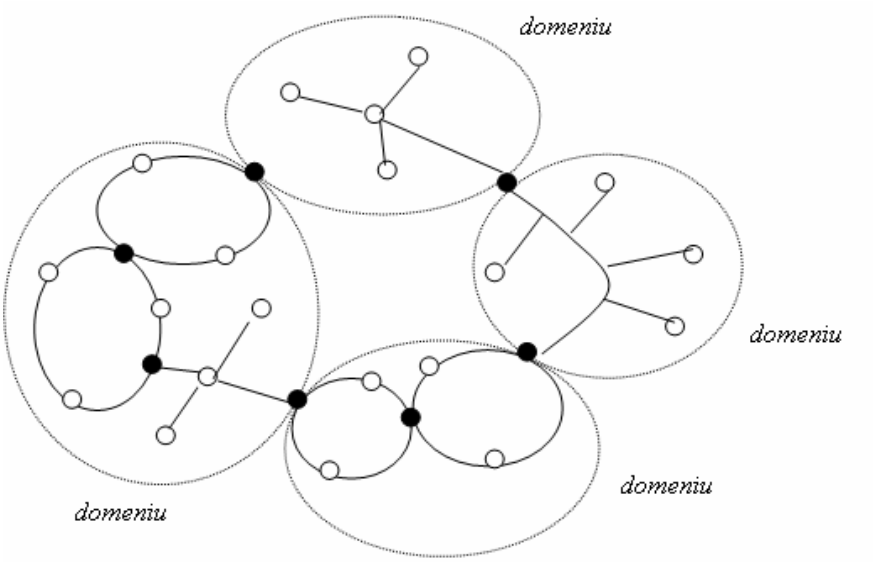


Fig. 3.11. Gruparea reţelelor locale în domenii

Numele de domeniu pentru România este *" ro ",* analog avem şi pentru toate celelalte ţări din lume care sunt conectate la Internet (v. şi [Dzi 06], pp. 116-121).

Apoi, fiecare domeniu controlează alocarea subdomeniilor descendente (v. fig. 3.12). În acest fel o porţiune a spaţiului numelor de domenii din care face parte, de exemplu, Departamentul de informatică *"cs"* de la Universitatea "Babeş Bolyai" din Cluj-Napoca, cu numele de domeniu *" ubbcluj "* este prezentat în continuare. Departamentul de informatică *"cs"*, se identifică precizând calea acestuia, marcată în desenul care urmează, respectiv numele de domenii prin care se trece şi care se separă prin " . ", adică ***cs.ubbcluj.ro*** .

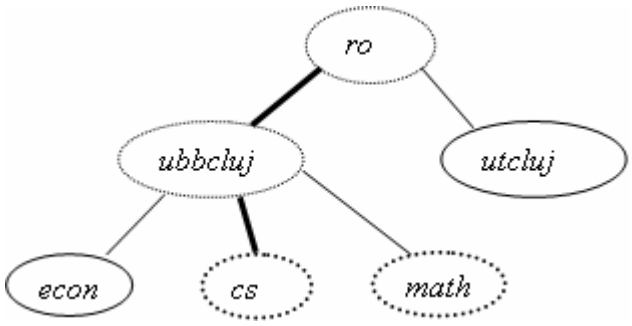


Fig. 3.12. Subdomenii ale unui domeniu

Nodurile terminale ale acestei structuri arborescente pot să conţină unu sau mai multe *calculatoare gazdă*. Fiecare nume din construcţia precedentă este un nume de domeniu. În general un nume poate conţine *64* de caractere şi nu se face distincţie între literele mari şi mici.

Utilizatorii individuali din cadrul unui domeniu care transmit mesaje se identifică printr-o ***adresă simbolică***, adresă de *e-mail* ce are structura *nume_calculator_gazdă@subdom1.subdom2...dom_niv_înalt* . De exemplu: *moldovan@cs.ubbcluj.ro*

Pentru evitarea unor situaţii conflictuale privind numele de domenii s-a ajuns la necesitatea constituirii unui *sistem al numelor de domenii,* numit *DNS (Domain Name System)* şi care este gestionat de către un centru internaţional.

Partea din adresă care defineşte domeniul(numele) şi identificatorul de reţea este atribuit de către o autoritate centrală, respectiv, organizaţia internaţională: *InterNic (Internet Network Information Center)*.

Pentru crearea unui domeniu se trimite un e-mail la adresa Hostmaster@INTERNIC.NET, cerând atribuirea unui *identificator de reţea*. În general aceşti identificatori de reţea sunt gestionaţi de către un administrator de reţea care supervizează reţeaua respectivă.

Utilizatorii individuali conectaţi la Internet se identifică, pe lângă adresa simbolică şi cu o adresă numerică corespunzătoare acelei locaţii. Adresele numerice care identifică o reţea sunt atribuite de aceeaşi autoritate, unde se cunoaşte în acest fel corespondenţa între domenii şi aceste adrese numerice.

Adresele numerice ale calculatoarelor gazdă (host) numite ***adrese Internet pentru IP*** sau mai scurt ***adrese IP,*** sunt şiruri a câte *patru octeţi,* adică patru grupe de câte *opt* cifre binare, separate cu " . ", în total *32* de biţi. Putem spune, în baza acestei convenţii de scriere, că avem, de fapt, patru numere zecimale separate cu ".". Adresa unui calculator gazdă este atribuită de autoritatea locală.

Exemplu: fie adresa *131.107.2.133*, partea *131.107.2* reprezintă identificatorul de reţea (domeniu), iar 133 reprezintă adresa calculatorului gazdă. Adresa se mai poate scrie în binar astfel: 10000011**.**01101011**.**00000001**.**11001000

În activitatea concretă, de fapt, se utilizează **adrese *IP* false**, obţinute din cele reale prin operaţii cu anumite "măşti".

Există trei clase de adrese *IP,* stabilite în funcţie de valoarea primului octet din adresa numerică a oricărei adrese *IP*:

- clasa *A: 1 - 127*
- clasa *B: 128 - 191*
- clasa *C: 192 - 223*

De regulă, o reţea mică, care este din clasa *C,* se identifică prin primii trei octeţi din adresa *IP*, iar ultimul octet identifică un calculator din această reţea. De exemplu o adresă de reţea poate fi: *192.35.91.** Organizaţiile mari obţin pentru reţele *IP* din clasa *B* şi sunt identificate de primii doi octeţi, de exemplu *130.132.*.**

3.4.5. Aplicaţia Web (world wide web – www)

Aplicaţia *Web* reprezintă cea mai spectaculoasă dezvoltare a protocolului TCP/IP şi a fost declanşată de dezvoltarea unui mod de transmitere a informaţiilor prin intermediul documentelor.

„World Wide Web”, prescurtat **WWW** sau simplu **Web**, este un *sistem de distribuţie locală sau globală a informaţiilor hipermedia.*

Din punct de vedere tehnic, spaţiul Web nu trebuie confundat cu Internetul sau cu o reţea, cum se crede greşit uneori. Spaţul Web este doar o aplicaţie distribuită în Internet care pune la dispoziţia utilizatorilor un sistem global şi standardizat de comunicare multimedia.

Iniţial WWW a fost conceput de cercetătorii de la Laboratorul European pentru Particule Fizice de la CERN (Centrul de Cercetări Nucleare de la Geneva), sub conducerea lui *Tim Berners-Lee*, care au propus un sistem **hipertext** (text neliniar, care permite salturi, analog trimiterilor din Biblie), care permitea partajarea eficientă a informaţiilor între membri unui grup de cercetătători care studiau fizica energiilor înalte.

Deci, sistemul Web poate fi folosit şi pe calculatoare dintr-o reţea care nu este conectată la Internet sau chiar pe un singur calculator izolat, dar astăzi se foloseşte mai ales în Internet pentru distribuţia informaţiilor hipermedia.

World Wide Web are facilităţi multimedia şi integrative, o interfaţă grafică pentru utilizator - **GUI (Graphic User Interface)** foarte atrăgătoare din punct de vedere grafic, practică şi simplu de folosit (prietenoasă).

Deci, Web-ul este un *sistem distribuit deschis* utilizat pentru distribuţia locală sau globală a informaţiilor, putând fi extins şi implementat în diferite moduri fără a-i afecta funcţionalitatea. Se utilizează în prezent, în general, în Internet pe baza modelului client/server. Clienţii, adică *navigatoarele Web* sau *browserele Web* (*Internet Explorer, Netscape Navigator, NCSA Mosaic, Mozilla, Opera* ş.a.), au acces la informaţiile hipermedia şi multiprotocol organizate asociativ, aflate pe un server Web (cele mai cunoscute servere Web sunt: *Apache, Netscape Enterprise Server, Sun Web Server, Microsoft Internet Information Server, Stronghold, Jigsaw*).”[DZI06}

HTTP (HyperText Transfer Protocol) este protocolul cu ajutorul căruia se lansează într-o reţea documente create prin intermediul unui standard dat de

limbajul *HTML (HypertText Markup Language)*, care stă la baza acestei aplicaţii.

Web-ul este o colecţie imensă de informaţii structurată pe ***pagini*** şi grupate în ***site***-uri. Fiecare pagină poate conţine legături către alte pagini.

Textele dintr-o pagină care conţin legături spre alte pagini se numesc *HyperTexte* sau *Hyperlegături*.

Documentele *HyperText* conţin: cuvinte, expresii, imagini, care sunt legate în cadrul documentului.

Web este o mulţime de documente hipertext interconectate şi distribuite pe o reţea mondială.

Programele care asistă citirea documentelor hypertext sunt de două feluri: client şi server.

Programele client care se află pe calculatorul cititorului sunt acelea care permit cititorului de documente să navigheze (*browser*) prin el.

Programele server sunt rezidente pe un calculator care conţine documentele la care se poate realiza un acces. Cel mai important exemplu de browser este *Netscape Navigator*.

Crearea de documente hypertext necesită o metodă de stabilire a legăturilor între documente.

În acest scop, fiecare document e identificat de o adresă unică denumită ***adresă URL*** *(Uniform Resource Locator)*, prin intermediul căreia un browser (navigator) să poată contacta serverul potrivit şi să solicite documentul dorit. O adresă URL în forma cea mai generală:

(1)<service>: //(2)<user>: <password>@(3)<host>:<port>/(4) <url-path>

(1) identifică protocolul de acces: *HTTP, FTP, Wais*, etc.

(2) este facultativă, dacă o folosim înseamnă că dorim să realizăm accesul la un document protejat prin intermediul unei parole.

(3) identifică maşina (calculatorul sau server-ul) pe care se găseşte documentul pe care vrem să-l accesăm; *<host>* poate fi precizat printr-o adresă IP, sau alfabetic precizând domenii; exemplu: *129.100.204* sau *cs.ubbcluj.ro*. *<port>*-ul în general este implicit.

(4) este calea de acces la documentul dorit (html).

Exemplu: http://www.cs.ubbcluj.ro/popescu.html.

CAPITOLUL 4

FLUXURI ÎN SISTEME DE COMUNICAȚIE

4.1. Câteva elemente de teoria grafelor

4.1.1. Noțiunea de graf şi multigraf

Unele reprezentări de grafe au fost folosite deja. Sunt multe probleme şi situații practice care conduc la necesitatea folosirea grafelor. Distingem două feluri de grafe: grafe orientate şi grafe neorientate. Le vom prezenta pe scurt în cele ce urmează.

Grafe şi multigrafe orientate.

Fie mulțimea X, $|X| < \infty$, $X \neq \varnothing$. O aplicație $\Gamma: X \to X$ o numim aplicație multivocă dacă oricare ar fi x, $X \ni x \mapsto \Gamma(x) \equiv \Gamma x \subseteq X$.

Definiție. *Fie mulțimea X, $|X| < \infty$, $X \neq \varnothing$ şi aplicația multivocă $\Gamma: X \to X$. Perechea $G = (X,\Gamma)$ se numeşte graf orientat. Dacă $x \in X$, $y \in \Gamma x$ atunci (x,y) se numeşte arc*.

Fie arcul (x,y), x reprezintă *extremitatea inițială* a arcului; y *extremitatea terminală* a arcului. Vom nota mulțimea arcelor grafului G cu $U=\{(x,y) | \forall x \in X, \forall y \in \Gamma x\}$.

Observații. 1) Putem avea $x \equiv y$, adică arcul (x,x) care în acest caz se numeşte *buclă*.

2) Mulțimea vârfurilor X, fiind finită şi dacă are cardinalul $|X|=n$ vârfurile se pot indica sub forma $X= \{x_1, x_2, ..., x_n\}$ sau identifica chiar prin $X=\{1,2,...,n\}$.

Se poate preciza un graf $G=(X,\Gamma)$ şi prin perechea (X,U), deci:

$G=(X,U)$, $U \subset X \times X$.

Avem o echivalență între perechea (X, Γ) şi (X,U), în sensul că putem trece de la Γ la U şi invers. Mulțimea U se defineşte prin intermediul lui Γ. Dacă se dă U, uşor putem determina valorile aplicației Γ şi invers.

Observație. Un graf se poate defini şi ca un triplet (X,A,f), unde:

X este mulțimea vârfurilor (nodurilor), ca mai sus;

A este mulțimea arcelor;

$f: A \to X \times X$ aplicație univocă

Această definiţie este mai cuprinzătoare, ea conduce la noţiunea de ***multigraf orientat***; adică între două vârfuri pot exista mai multe arce. Dacă între oricare două vârfuri există cel mult p arce atunci spunem că avem un *p-graf*. Definiţia grafului orientat dată la început precizează un *2*-graf particular: între două vârfuri x,y pot exista cel mult două arce: $(x,y), (y,x) \in A$.

Un *1*-graf orientat este întotdeauna un graf orientat (nu este valabilă afirmaţia pentru un *2*-graf).

Alte tipuri de multigrafe

a) multigraf cu arce etichetate este ansamblul (X,A,f, φ), unde X este mulţimea vârfurilor, A este mulţimea arcelor, iar

$f : A \to X \times X$, $\varphi : A \to D_1$;

unde arcelor le asociem nişte valori, elemente din D_1, care sunt numere pozitive sau care aparţin unui domeniu oarecare D_1.

b) multigraf cu vârfuri etichetate; este ansamblul (X,A,f, ψ), unde X este mulţimea vârfurilor, A este mulţimea arcelor, iar

$f : A \to X \times X$, $\psi : X \to D_2$,

unde D_2 este domeniul etichetelor vârfurilor.

c) multigraf cu vârfuri şi arce etichetate (X,A,f, φ, ψ) cu semnificaţiile de sus,

$f: A \to X \times X$, $\varphi : A \to D_1$, $\psi : X \to D_2$

Grafe neorientate

Fie mulţimea X, $|X| < +\infty$, $X= \{x_1,x_2,...,x_n\}$, deci $|X|=n$.

În loc de a considera perechi ordonate de vârfuri, considerăm *perechi neordonate* de vârfuri pe care le numim *muchii*.

Notăm o muchie între x_i şi x_j prin $[x_i, x_j]$, iar mulţimea tuturor muchiilor care leagă noduri din X, cu M.

Definiţie. *Fie mulţimea* X, $X= \{x_1,x_2,...,x_n\}$ *şi mulţimea muchiilor* M. *Perechea* (X, M) *se numeşte graf neorientat*.

Analog multigrafelor orientate se pot defini şi multigrafe neorientate, cu muchii şi/sau vârfuri etichetate.

Un multigraf neorientat se poate defini prin tripletul (X,M,f), $f:M \to B$, unde B reprezintă mulţimea tuturor perechilor neordonate din X.

4.1.2. Reprezentarea grafelor

1) Reprezentarea geometrică

a) Fie $G = (X,U)$ un graf orientat.
Dacă $u \in U$; $u=(x,y)$ atunci reprezentarea geometrică a acestuia este dată în fig. 4.1.

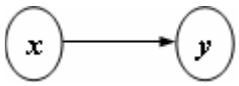


Fig. 4.1. Reprezentarea arcului *(x,y)*

Exemplul 4.1. Fie $G=(X,U)$; $X=\{x_1,x_2,x_3,x_4\}$
$U=\{(x_1,x_2),(x_1,x_3),(x_3,x_2),(x_2,x_4)\}$

Pentru acest exemplu, avem: $G=(X,\Gamma)\equiv(X,U)$
$\Gamma: X \to X$; $\Gamma(x_1)=\{x_2,x_3\} \subset X$, $\Gamma(x_2)=\{x_4\}$, $\Gamma(x_3)=\{x_2\}$, $\Gamma(x_4)=\varnothing$
Reprezentarea geometrică este dată în fig. 4.2.

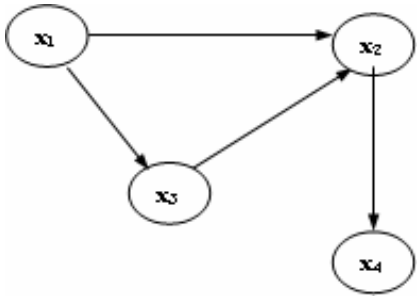


Fig. 4.2. Graful orientat din exemplul 4.1

c) Fie $G=(X,M)$ un graf neorientat. Dacă $m=[x,y] \in M$, reprezentarea geometrică a acestei muchii este dată în fig. 4.3.

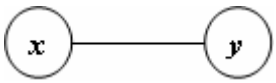


Fig. 4.3. Muchia unui graf neorientat

Exemplul 4.2. Fie $G=(X,M)$; $X=\{x_1,x_2,x_3,x_4\}$
$M=\{[x_1,x_2],[x_1,x_3],[x_3,x_2],[x_2,x_4]\}$
Reprezentarea geometrică a acestui graf neorientat, este dată în figura 4.4.

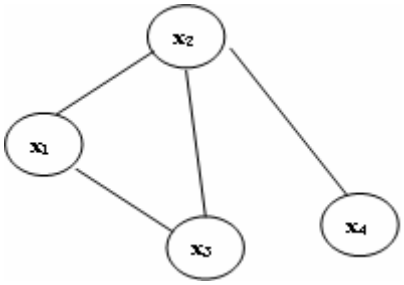


Fig.4.4. Graful neorientat din exemplul 4.2

2. Matrici de adiacenţă

a) Fie $G=(X,U)$ un graf orientat unde:
$X=\{x_1,x_2,...,x_n\}$
Matricea de adiacenţă, care defineşte acest graf, este:

$$A=(a_{ij}) ; \quad a_{ij}=\begin{cases} 1, \text{ daca } (x_i,x_j)\in U \\ 0, \text{ daca } (x_i,x_j)\notin U \end{cases} \quad ; \quad i,j=\overline{1,n} \ .$$

Exemplu. Fie $G=(X,U)$; $X=\{x_1,x_2,x_3,x_4\}$
$U=\{(x_1,x_2),(x_1,x_3),(x_3,x_2),(x_2,x_4)\}$

Matricea de adiacenţă a acestui graf orientat este:

	x_1	x_2	x_3	x_4
x_1	0	1	1	0
x_2	0	0	0	1
x_3	0	1	0	0
x_4	0	0	0	0

b) Fie $G=(X,M)$ un graf neorientat unde: $X=\{x_1,x_2,...,x_n\}$. Matricea de adiacenţă, în acest caz, se defineşte, astfel:

$$A=(a_{ij}) ; \quad a_{ij}=\begin{cases} 1, \text{ daca } [x_i,x_j]\in M \\ 0, \text{ daca } [x_i,x_j]\notin M \end{cases} \quad ; \quad i,j=\overline{1,n} \ .$$

Observaţie. Matricea de adiacenţă a unui graf neorientat este o matrice simetrică.

3. Matricea de incidenţă a unui graf orientat

Fie graful orientat şi fără bucle: $G=(X,U)$, $X=\{x_1,...,x_n\}$, $U=\{u_1,...,u_m\}$.
Definim matricea de incidenţă a acestui graf orientat ca fiind:

$$B=(b_{ij});$$

unde

$$b_{ij}=\begin{cases} 1, \text{ daca } x_i \text{ este extremitatea initiala a arcului } u_j \\ -1, \text{ daca } x_i \text{ este extremitatea terminala a arcului } u_j \\ 0, \text{ altfel} \end{cases}$$

$$i=\overline{1,n}\ ;\ j=\overline{1,m}$$

Exemplu. Fie $G=(X,U)$; $X=\{x_1,x_2,x_3,x_4\}$
$U=\{(x_1,x_2),(x_1,x_3),(x_3,x_2),(x_2,x_4)\}=\{u_1, u_2, u_3, u_4\}$;
$u_1=(x_1,x_2)$, $u_2=(x_1,x_3)$, $u_3=(x_3,x_2)$, $u_4=(x_2,x_4)$

Matricea de incidenţă a acestui graf orientat este:

	$u_1=(x_1,x_2)$	$u_2=(x_1,x_3)$	$u_3=(x_3,x_2)$	$u_4=(x_2,x_4)$
x_1	1	1	0	0
x_2	-1	0	-1	1
x_3	0	-1	1	0
x_4	0	0	0	-1

4.1.3. Drum într-un graf orientat

Fie graful orientat $G=(X,U)$, $|X|>2$. Vom defini, într-un graf orientat, noţiunea de drum. Prin ***drum*** înţelegem o succesiune de arce $(u_1,u_2,..,u_n)$, $n>1$ astfel încât extremitatea terminală a unui astfel de arc, în afară de ultimul arc, să coincidă cu extremitatea iniţială a arcului următor din această succesiune de arce.

Exemplul 4.3. Fie $G=(X,U)$; $X=\{x_1,x_2,x_3,x_4\}$
$U=\{(x_1,x_2),(x_1,x_3),(x_3,x_2),(x_2,x_4)\}$
pe care-l reprezentăm geometric în fig. 4.5..Am marcat în acest graf orientat drumul format cu arcele: $u_1=(x_1,x_3)$, $u_2=(x_3,x_2)$, $u_3=(x_2,x_4)$, deci:
$d=(u_1,u_2,u_3)$.
Drumul se poate preciza şi indicând vârfurile prin care trece el, în exemplul nostru: $d=<x_1,x_3,x_2,x_4>$.

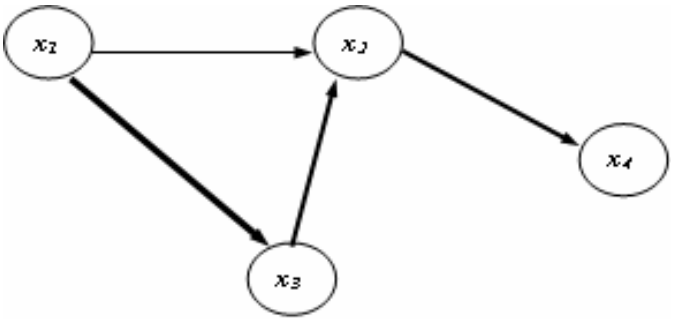


Fig. 4.5. Graful orientat din exemplul 4.3

Lungimea unui drum d, pe care o notăm cu *l(d),* este numărul arcelor pe care le conţine. În exemplul de drum considerat mai sus, avem *l(d) = 3*.

Iată, câteva *probleme mai importante care se pot pune relativ la drumuri:*

- Găsirea unor drumuri de lungime minimă între două vârfuri ale unui graf, adică lungimea şi drumul efectiv; sunt cunoscuţi algoritmii: Ford, Bellman-Kalaba etc. care rezolvă această problemă.
- Fie graful $G=(X,A,f, \varphi)$. Dacă graful G este cu arce etichetate, atunci considerăm că am asociat fiecărui arc u o valoare numerică reală pozitivă:

$$\varphi(u) \in R_+$$

Relativ la un drum $d = (u_1, ..., u_n)$ putem vorbi de valoarea drumului, notată $v(d)$. Avem,

$$v(d)=\sum_{i=1}^{n}\varphi(u_i).$$

Putem formula următoarea *problemă*: să se determine drumul de valoare minimă dintre două vârfuri date ale unui graf (sunt cunoscuţi algoritmii: Ford, Bellman Kalaba etc. care rezolvă această problemă).

Observaţie. Între matricea de adiacenţă a unui graf şi numărul de drumuri de o lungime dată există o legătură: dacă ridicăm la pătrat matricea de adiacenţă de ordinul n, atunci elementele acesteia ne dau numărul de drumuri de lungime 2 între vârfurile corespunzătoare ($c_{ij}=a_{i1}a_{1j}+a_{i2}a_{2j}+ ... +a_{in}a_{nj} \in A \times A = A^2$ ne dă numărul de drumuri de lungime *2* de la x_i la x_j); analog pentru o putere p oarecare a matricei de adiacenţă A obţinem numărul de drumuri de lungime p.

4.1.4. Lanţ (cale) într-un graf neorientat

Fie graful neorientat $G = (X,M)$, $|X|>2$. Vom defini, într-un graf neorientat, noţiunea de lanţ sau altfel numit, cale. Prin ***lanţ (cale)*** înţelegem o succesiune de muchii $(m_1,m_2,..,m_n)$, $n>1$ astfel încât două muchii consecutive au în comun un vârf (nod). Un lanţ se poate preciza şi prin indicarea nodurilor prin care trece. Lungimea unui lanţ este dată de numărul muchiilor pe care le conţine, iar dacă muchiilor le asociem nişte valori, atunci valoarea unui lanţ este egală cu suma valorilor muchiilor care formează lanţul respectiv.

În exemplul de graf neorientat considerat mai sus: $G = (X,M)$; $X=\{x_1,x_2,x_3,x_4\}$, $M=\{[x_1,x_2], [x_1,x_3], [x_3,x_2], [x_2,x_4]\}$, adică $M=\{m_1, m_2, m_3, m_4\}$ unde $m_1=[x_1,x_2]$, $m_2= [x_1,x_3]$, $m_3=[x_3,x_2]$, $m_4=[x_2,x_4]$ punem în evidenţă, ca un caz particular, lanţul (calea);

$$c=(m_2,m_3,m_4) =<x_1, x_3, x_2, x_4>;$$

lungimea acestui lanţ este $l(c)=3$.

Problemele care se pun relativ la grafe neorientate sunt similare cu cele care se pun relativ la grafe orientate şi se rezolvă analog.

4.1.5. Sisteme de tranziţie neetichetate

Să revenim cu o completare la sistemele de tranziţie pe care le-am prezentat într-un paragraf anterior, acum când avem mai multe cunoştinţe de teoria grafelor.

Am indicat modul în care un sistem de tranziţii $ST = (S,I,A,T)$ se poate prezenta sub forma unui graf orientat cu noduri şi arce etichetate.

Să notăm $SA = \{sa \mid s \in S, a \in A\}$, care reprezintă concatenarea mulţimilor S şi A. Dacă pentru graful $G = (X, U, f, g)$ se renunţă la funcţia $g:U \to P(A)$, atunci graful $H = (X, U, f)$ unde X şi U sunt aceleaşi mulţimi ca mai înainte, iar de data aceasta $f:X \to SA \cup S$, este un graf neetichetat asociat sistemului de tranziţii considerat.

Fie, ca exemplu, $(s,a,t) \in T \subseteq S \times A \times S$, atunci în graful de tranziţii pe care îl numim acum graf de tranziţii etichetat am considerat reprezentarea:

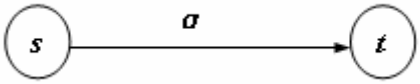


Fig. 4.6. Graf de tranziţii

Acestui arc etichetat cu *a* facem să-i corespundă drumul ce leagă nodurile etichetate cu stările: *s, sa* şi *t*, care conţine, însă, două arce neetichetate (v. fig. 4.7).

Fig. 4.7. Drum între noduri etichetate cu arce neetichetate

Exemplu: Să considerăm sistemul de tranziţii *X,* reprezentat în fig. 4.8 de graful de tranziţii etichetat şi pe care-l vom numi sistem de tranziţii etichetat. Acest sistem de tranziţii are starea iniţială *s* şi nodul terminal etichetat cu *v*.

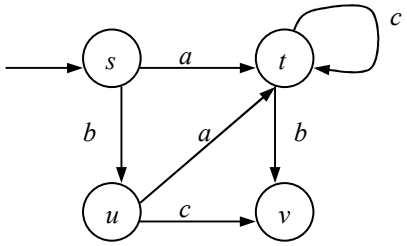


Fig. 4.8. Graful cu noduri şi arce etichetate ale unui sistem de tranziţii

Dacă ţinem seama de procedeul precedent prin care am înlocuit un arc etichetat printr-o pereche de arce neetichetate, graful din *fig. 1* se transformă în graful din *fig. 4.9*. Această transformare este unică.

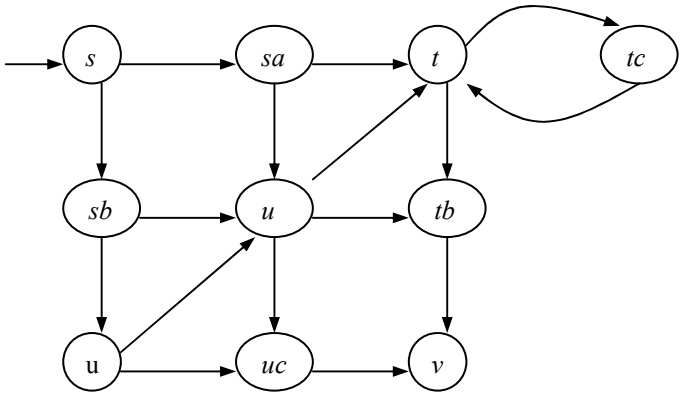


Fig. 4.9. Graful cu noduri etichetate, echivalent cu cel din fig.4.8

Din procedeul de construcţie indicat mai sus rezultă că este adevărată următoarea teoremă:

Teoremă. *Fie E, respectiv N, mulţimea sistemelor de tranziţie etichetate, respectiv neetichetate. Oricare ar fi $e \in E$, un sistem de tranziţie etichetat; există o funcţie $F:E \to N$, astfel încât F(e) este un sistem de tranziţie neetichetat care păstrează comportamentul sistemului de tranziţie dat.*

Pentru a avea o demonstraţie completă a acestei teoreme să facem următoarea observaţie privind determinarea *căilor* din graful asociat. Să avem în vedere exemplul din *fig. 4.9*. La formarea secvenţei, pornind din starea iniţială ce va constitui o *cale*, vom stabili un drum din starea iniţială până la nodul terminal *v* şi compunem apoi succesiv stările prin care trece, fără a repeta o anumită stare. De exemplu, nu vom scrie secvenţa *ssbuucv*, ci în locul acesteia vom considera *sbucv,* care reprezintă o cale în sistemul de tranziţie dat, *X*. În felul acesta sistemul de tranziţie, neetichetat din *fig. 4.9* ne furnizează aceeaşi lungime de căi ca sistemul de tranziţii etichetat din *fig. 4.8*. Evident că şi urmele vor fi aceleaşi. Procedăm analog în cazul general.

4.2. Reţele de comunicaţii

Despre comunicaţii şi căi de comunicaţii am discutat mai sus în multe situaţii. Am folosit şi reprezentări ale acestora. Acum aceste concepte şi altele, le vom raporta la grafe ce au anumite proprietăţi speciale.

Vom defini noţiunea de reţea de comunicaţie la care ne vom referi mereu în continuare. Pentru aceasta, vom considera o mulţime de vârfuri $X=\{x_0,x_1 \ldots,x_n\}$.

Definiţie*: Numim reţea de comunicaţie un 1-graf finit şi fără bucle $G = (X,U) = (X, \Gamma)$ în care fiecărui arc $u \in U$ îi asociem un număr $c(u)>0$ numit capacitatea arcului, iar în acest graf sunt adevărate proprietăţile:*

- *există un singur vârf x_0 astfel încât $\Gamma^1(x_0)= \varnothing$ şi el reprezintă intrarea în reţea;*
- *există un singur vârf x_n astfel încât $\Gamma(x_n)= \varnothing$ şi el reprezintă ieşirea din reţea;*
- *oricare ar fi un alt vârf x_i, diferit de x_0 şi x_n, , există un drum notat $x_0 \supset x_i \supset x_n$ care porneşte din x_0, trece prin x_i şi ajunge în x_n.*

Observaţie. În definiţia de mai sus se poate presupune şi o situaţie mai generală, dar aceasta se reduce la cea avută în vedere mai sus. Putem considera o reţea de comunicaţie ca având mai multe intrări, deci o mulţime de noduri X_0,

$X_0 \subset X$ şi cu mai multe ieşiri pe care să le notăm X_n, $X_n \subset X$. Situaţia aceasta revine la cea precizată în definiţia de mai sus, dacă se consideră o extindere a mulţimii nodurilor prin considerarea a încă două vârfuri x_0' , x_n', deci avem mulţimea vârfurilor $X \cup \{x_0', x_n'\}$, iar virtual să avem arcele $U \cup \{(x_0', x) | \forall x \in X_0\} \cup \{(x, x_n') | \forall x \in X_n\}$.

4.2.1. Flux într-o reţea de comunicaţie

Fie o reţea $G = (X,U)$, $X = \{x_0, x_1, ..., x_n\}$ şi $x_i \in X$ un nod oarecare al reţelei.

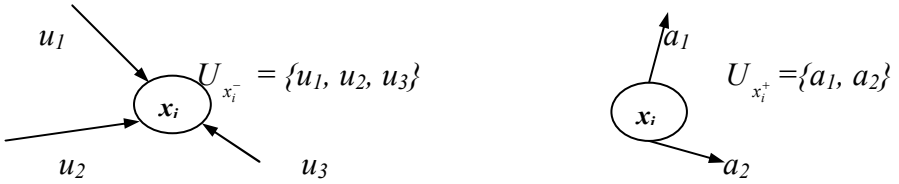


Fig. 4.10

În acest vârf punem în evidenţă mulţimea arcelor incidente spre interiorul vârfului x_i, notată $U_{x_i^-}$

Analog, pentru vârful x_i există în general o mulţime de arce care pornesc din x_I, deci cu extremitatea iniţială în x_i, notată $U_{x_i^+}$

Relativ la o reţea de comunicaţie vom defini noţiunea de flux.

Definiţie. *Fie $H = (X,U)$ o reţea de comunicaţie. Prin definiţie, o funcţie $\varphi: U \to R_+$ (de multe ori în loc de R_+ se consideră Z_+) se numeşte* ***flux*** *al acestei reţele, dacă ea îndeplineşte următoarele proprietăţi:*

1. $\varphi(u) \geq 0, \forall u \in U$;
2. $\sum_{u \in U_{x_i^-}} \varphi(u) = \sum_{u \in U_{x_i^+}} \varphi(u)$ $\quad \forall x_i \in X,\ x_i \neq x_0,\ x_i \neq x_n$
3. $\varphi(u) \leq c(u)$; $c(u)$ –*capacitatea arcului u,* $\forall u \in U$

Din 2) rezultă $\varphi_0 = \sum_{u \in U_{x_0^+}} \varphi(u) = \sum_{u \in U_{x_0^-}} \varphi(u)$

φ_0 se numeşte ***valoarea fluxului*** reţelei de comunicaţii date.

Fluxul poate avea în particular semnificaţia următoare: cantitatea de informaţie care trece printr-o reţea de comunicaţie la un moment dat.

4.2.2. Exemple de reţele de comunicaţie

Reţea de comunicaţie statică

Fie $A_1, A_2, ..., A_m$ noduri în care se găsesc anumite cantităţi de informaţie (pachete) ce spunem că reprezintă o *ofertă* pentru alte noduri.

Fie oferta pentru alte noduri $x_1, x_2, ..., x_m$ ce înseamnă număr de pachete; în nodul A_i avem x_i , $i=\overline{1,m}$ pachete .

Apoi, fie $B_1, B_2, ..., B_n$ noduri în care se formulează cereri de informaţii (pachete) de la nodurile precizate mai sus.

Fie aceste cereri $y_1, y_2, ..., y_n$ ce se exprimă în număr de pachete; în nodul B_i se solicită y_i , $i=\overline{1,n}$ pachete.

Problema care se pune este de a construi o reţea de comunicaţie ce leagă nodurile A_i de nodurile B_j astfel încât să realizăm cererea din oferta existentă. Această problemă se poate transcrie ca o problemă de programare matematică.

Problema* Cerere – Ofertă *de pachete

Între A_i si B_j pot fi transferate cantităţi de informaţie $\le c_{ij}$ (limita capacităţii liniei care leagă A_i de B_j). Urmărim satisfacerea în condiţii optime a tuturor cererilor existente în sistem şi de a se organiza expedierea pachetelor de informaţii din $A_1, A_2, ..., A_m$ spre nodurile $B_1, B_2, ..., B_n$.

Pentru rezolvare construim o reţea de comunicaţie din fig. 4.11.

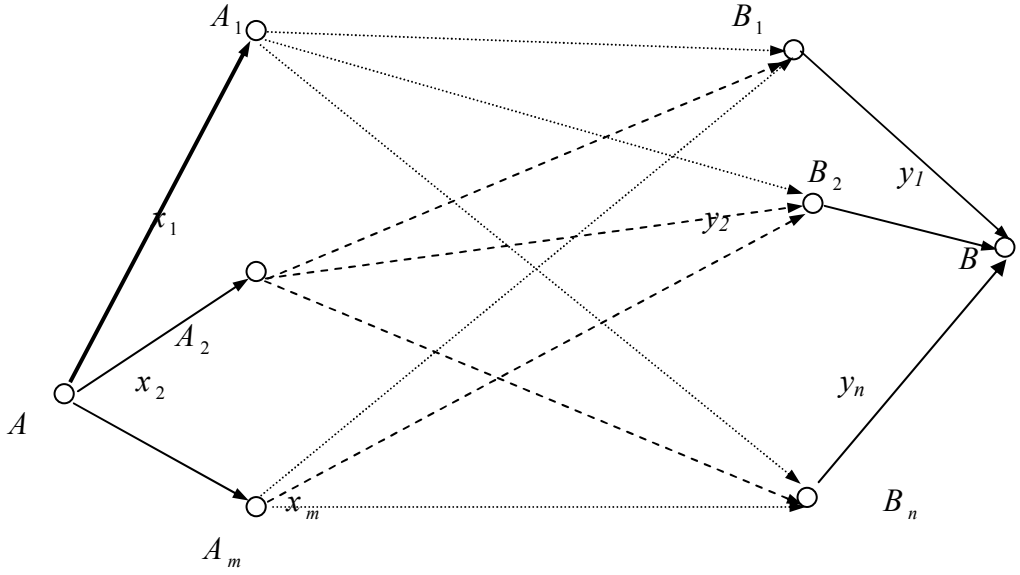


Fig. 4.11. Reţea de comunicaţie

Putem presupune că există legături A_i - B_j, adică arce (A_i, B_j) $\forall i=\overline{1,m}$, $\forall j=\overline{1,n}$ (dacă nu există această legătură considerăm capacitatea arcului respectiv ca fiind 0).

Modelul matematic, asociat problemei, este:

$$[max]R = \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij}$$ unde r_{ij} = număr de pachete care se transportă efectiv din A_i în B_j ;

$$r_{ij} \le c_{ij},\ 0 \le r_{ij},\ i=\overline{1,m};\ j=\overline{1,n};$$

$$\sum_{i=1}^{m} r_{ij} = \xi_j \le y_j \quad j=\overline{1,n};$$

$$\sum_{j=1}^{n} r_{ij} = \eta_i \le x_i \qquad i=\overline{1,m}.$$

Această problemă se poate rezolva ca o problemă de programare matematică, dar şi ca o problema de flux după cum vom vedea acest lucru în continuare, considerând $r_{ij}=\varphi(u_{ij});\ u_{ij}=(A_i,B_j)$.

Reţea de comunicaţie dinamică

Presupunem că din diferite noduri, notate $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ se cere să se transfere către o aceeaşi destinaţie $\bar{y}$ nişte pachete (comutaţie de pachete). Dacă $\bar{x}_i$ şi $\bar{x}_j$ sunt legate printr-un drum vom nota $\bar{x}_i \nearrow \bar{x}_j$. Fie t_{ij} timpul necesar ca un pachet să ajungă din $\bar{x}_i$ în $\bar{x}_j$; c_{ij} numărul de pachete care pot fi transferate pe acest drum într-o unitate de timp. Dacă nu există drum de la $\bar{x}_i$ la $\bar{x}_j$ atunci $c_{ij}=0$. Notăm cu:

c_{ii} - numărul de pachete care pot să staţioneze în nodul (staţia) $\bar{x}_i$

a_i - numărul de pachete care se află iniţial în $\bar{x}_i$.

Problema este: cum trebuie organizat traficul de rutare pentru ca într-un interval de timp $[0, \theta]$, numărul pachetelor ajunse în $\bar{y}$ să fie cel mai mare posibil?

Să discretizăm intervalul de timp $[0, \theta]$, sub forma $\{0, 1, \ldots, \theta\}$. Pentru rezolvarea problemei să definim o reţea de comunicaţie ale cărei vârfuri sunt elementele mulţimii $X = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n, \bar{y}\} \times \{0, 1, \ldots, \theta\}$.

Notăm

$$x_i(t) = (\bar{x}_i, t),\ i=\overline{1,n} \quad y(t)=(\bar{y}, t);\quad t \in \{0, 1, \ldots, \theta\}.$$

Între două vârfuri, pentru t şi t_{ij} care se exprimă în aceleaşi unităţi de măsură, se pot defini arcele:

- $(x_i(t), x_j(t + t_{ij}))$, iar arcului respectiv i se asociază capacitatea c_{ij}
- $(x_i(t), x_i(t + 1))$, acestui arc i se asociază capacitatea c_{ii}.

Considerăm funcţia $\varphi : A \to R_+$ ce reprezintă fluxul în reţeaua de comunicaţie definită. Aici A este mulţimea arcelor, iar

$$\varphi((x_i(t), x_j(t+t_{ij})))$$

este valoarea fluxului arcului considerat şi ea reprezintă numărul pachetelor care pleacă din $\bar{x}_i$ la momentul t pentru a ajunge în $\bar{x}_j$ după t_{ij} unităţi de timp.

Analog,

$$\varphi((x_i(t), x_i(t+1)))$$

este numărul de pachete care staţionează în $\bar{x}_i$ la momentul t.

Organizarea traficului de rutare în acest sistem de comunicaţii revine la a găsi o funcţie φ ale cărei valori pe arce sunt cele mai mari posibile în condiţiile date.

Construim reţeaua de comunicaţii astfel:

- considerăm nodurile $x_i(t) = (\bar{x}_i, t),\ i=\overline{1,n}$ şi nodul $y(t) = (\bar{y}, t)$, pentru $t \in \{0, 1, \ldots, \theta\}$, între care vom considera că avem diferite legături înspre ieşirea din reţea;
- la acestea adăugăm:

  - un nod x_0 care reprezintă intrarea în reţea şi pe acesta îl legăm de toate nodurile $x_i(0)$ prin linii de comunicaţie cărora le asociem valoarea a_i, $i=\overline{1,n}$, adică arcelor $(x_0, x_i(0)),\ i=\overline{1,n}$ le-am asociat aceste valori;

  - un nod z care va reprezenta ieşirea din reţea şi care va fi legat prin linii de comunicaţie, de toate vârfurile $y(t)$, iar $(y(t), z)$ are valoarea ∞, pentru $t \in \{0,1,\ldots,\theta\}$.

Avem următoarea reţea de comunicaţii care este desenată doar parţial:

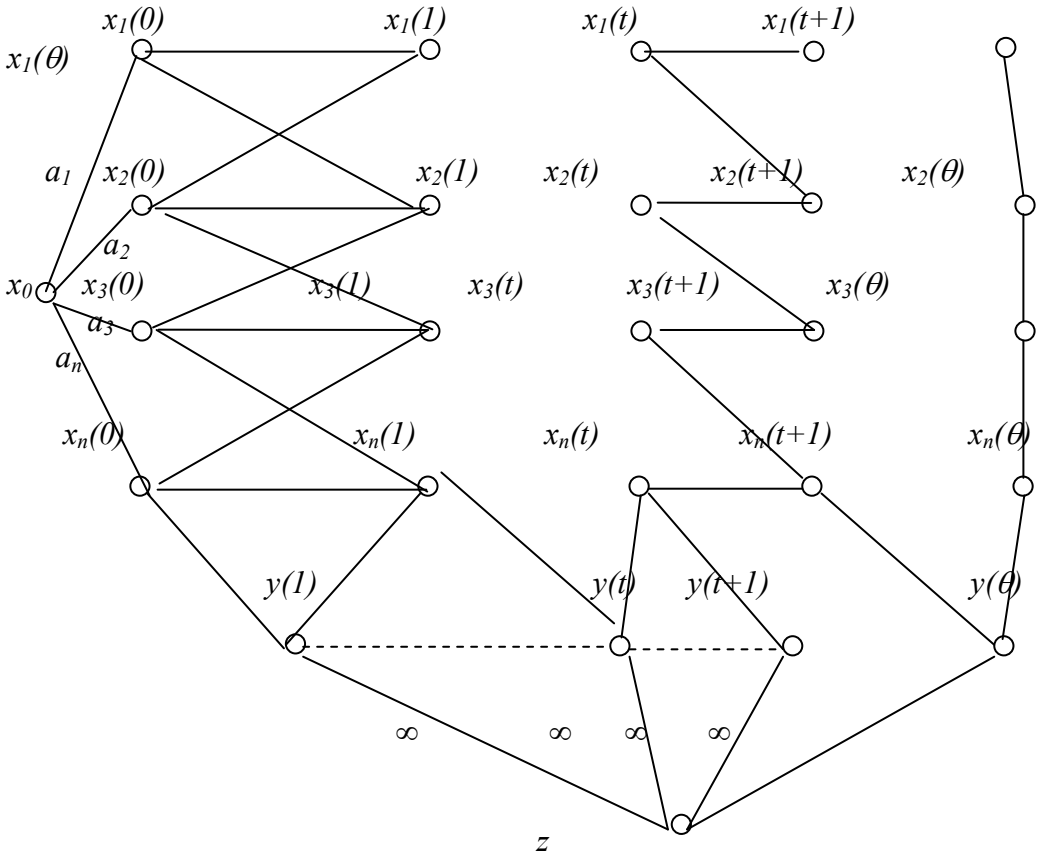


Fig. 4.12. Reţea de comunicaţie (parţială)

Pentru această rețea de comunicații, acum o rețea statică, $G=(X,U)$ cu capacitățile $c:U\to R_+$ să considerăm ca necunoscute valorile fluxului pe arce care leagă două noduri oarecare ale rețelei; notăm cu I numerele naturale care identifică nodurile din mulțimea X. Întregul care identifică ieşirea z din rețea îl notăm (z), $(z)\in I$. Dacă avem nodurile x_i, x_j din X, notăm:

$$\xi^i_j = \varphi(x_i, x_j);\ \ i,j\in I.$$

Să notăm, de asemenea,

$$c^i_j = 0 \begin{cases} (x_i, x_j)\notin U \\ c(x_i, x_j), \quad (x_i, x_j)\in U \quad i,j\in I, \end{cases}$$

unde c^i_j reprezintă capacităţile arcelor acestei reţele de comunicaţie.

Problema fluxului maxim în această reţea de comunicaţii revine la determinarea unei mulţimi de numere ξ^i_j care trebuie să verifice condiţiile :

$$0 \le \xi^i_j \le c^i_j; \qquad i,j \in I$$

$$\sum_i \xi^i_j = \sum_k \xi^j_k, \qquad j \in I$$

şi pentru care se cere

$$[max] \sum_{j \in I} \xi^j_{(z)}.$$

Această problemă de programare matematică se poate rezolva cu metoda simplex, dar datorită particularităţii ei, ea se poate rezolva şi cu algoritmi speciali, care se referă la determinarea unui flux optim într-o reţea.

4.2.3. Flux optim într-o reţea de comunicaţie. Algoritmul lui Ford - Fulkerson

Pentru început să ne reamintim noţiunea de flux într-o reţea de comunicaţie prezentată mai sus.

Fie o reţea $G = (X,U)$ şi $c: U \to R_+$ capacitatea arcelor. Prin flux într-o reţea de comunicaţii înţelegem valorile pe arce ale funcţie $\varphi : U \to R_+$ (sau cu valori în Z_+), astfel încât:

- $0 \le \varphi(u) \le c(u) \ \forall u \in U$;
- dacă $\forall x_i \in X$; $x_i \ne x_0$, $x_i \ne x_n$ iar

$U_{x_i^-}$ - mulţimea arcelor incidente spre interiorul vârfului x_i $(\to x_i)$

$U_{x_i^+}$ - mulţimea arcelor incidente spre exteriorul vârfului x_i $(x_i \to)$

atunci $\sum_{u \in U_{x_i^-}} \varphi(u) = \sum_{u \in U_{x_i^+}} \varphi(u)$.

Valoarea fluxului este

$$\varphi_0 = \sum_{u \in U_{x_0^+}} \varphi(u) = \sum_{u \in U_{x_n^-}} \varphi(u)$$

Această relaţie reprezintă, de asemenea, ***ecuaţia de conservare***.

Tăietură într-o reţea de comunicaţie

Fie o reţea de comunicaţii $G=(X,U)$ cu intrarea x_0 şi ieşirea x_n şi fie următoarea partiţie a mulţimii nodurilor:

$$X = X' \cup X''; \quad X' \cap X'' = \varnothing$$
$$x_0 \in X' \text{ şi } x_n \in X''.$$

Mulţimea $U_{x'x''} = \{(x_i, x_j) \mid (x_i, x_j) \in U, \ x_i \in X', x_j \in X''\}$ tuturor arcelor cu această proprietate se numeşte ***tăietură*** a reţelei. Un exemplu de tăietură avem în figura 4.13.

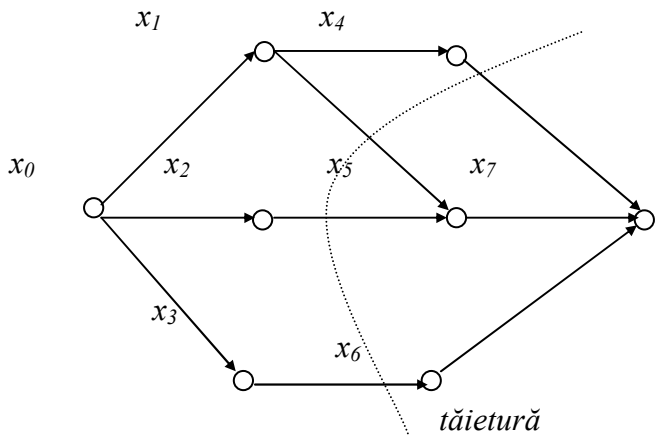


Fig. 4.13. Exemplu de tăietură întro reţea de comunicaţie

Avem $X=\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$, iar

$$X' = \{x_0, x_1, x_2, x_3, x_4\};$$
$$X'' = \{x_5, x_6, x_7\}.$$

Mulţimea de arce :

$$U_{x'x''} = \{ (x_1, x_5), (x_2, x_5), (x_4, x_7), (x_3, x_6) \},$$

marcate în reprezentarea de mai sus prin linia întreruptă, este o tăietură în reţeaua de comunicaţii considerată.

Capacitatea unei tăieturi $U_{x'x''}$, se defineşte şi notează, astfel:

$$C(X', X'') = \sum_{u \in U_{x'x''}} c(u).$$

Ne vom referi în cele ce urmează tot timpul la reţele de comunicaţii $G=(X,U)$ pentru care avem intrarea x_0, iar ieşirea din reţea x_n.

Lema 1. *Toate drumurile ce leagă pe* x_0 *de* x_n *trebuie să conţină un arc al oricărei tăieturi dintr-o reţea de comunicaţie.*

Demonstraţie. Fie un drum oarecare care leagă intrarea de ieşirea din reţea $d = (x_0, x_{i_1}, ..., x_{i_n}, x_n)$. Atunci

$\exists k>0$ astfel încât $x_{i_k} \in X'$, $x_{i_{k+1}} \in X''$,

iar $(x_{i_k}, x_{i_{k+1}}) \in U_{x'x''}$.

Din lema enunţată, rezultă următoarea consecinţă.

Consecinţă. *Dacă toate arcele unei tăieturi ar fi înlăturate din reţea, atunci nu ar exista nici un drum de la* x_0 *la* x_n *şi valoarea fluxului ar fi nulă.*

Deoarece o tăietură întâlneşte toate drumurile de la x_0 la x_n rezultă că valoarea fluxului nu poate fi superioară capacităţii unei tăieturi, oricare ar fi aceasta. Este, astfel, adevărată lema care urmează.

Lema2. *Avem*

$$\varphi_0 = \sum_{u \in U_{x_n^-}} \varphi(u) \le \sum_{u \in U_{x'x''}} c(u), \quad \forall \text{ ar fi o tăietură } U_{x'x''}$$

Pe de altă parte $\sum_{u \in U_{x'x''}} c(u) = C(X', X'')$, din definiţie.

Datorită faptului că folosim numai mulţimi cu număr finit de elemente, rezultă că există un flux şi o tăietură astfel încât $\varphi_0 = C(X', X'')$. De aici rezultă o teoremă importantă pe care o enunţăm şi demonstrăm în continuare, teorema lui Ford Fulkerson.

Teoremă (Ford – Fulkerson) *Valoarea fluxului maxim de la* x_0 *la* x_n *într-o reţea de comunicaţii este egală cu capacitatea acelei tăieturi care, dintre toate tăieturile care separă pe* x_0 *de* x_n*, este de capacitate minimă.*

Adică:

$$\max_{\varphi} \varphi_0 = \min_{U_{X'X''}} C(X', X'')$$

Demonstraţie. Din *Lema 2* rezultă că e suficient să stabilim existenţa unui flux φ şi a unei tăieturi $U_{X'X''}$ pentru care să aibă loc egalitatea $\varphi_0 = C(X', X'')$. Realizăm acest lucru dacă, pornind de la un flux maxim φ_0, definim o tăietura $U_{X'X''}$ astfel încât $\varphi_0 = C(X', X'')$.

Dacă $u=(x_i, x_j)$ este un arc al unei reţele de comunicaţii convenim să notăm valoarea fluxului pe acest arc, astfel $\varphi(u) = \varphi(x_i, x_j)$; analog şi pentru capacitatea arcului respectiv $c(u) = c(x_i, x_j)$. Considerăm, deci, un flux φ pe care îl presupunem ca fiind maxim şi construim mulţimea X' prin următorul procedeu recurent:

- $x_0 \in X'$; $i \geq 0$
- dacă $x_i \in X'$ şi $\varphi(x_i, x_j) < c(x_i, x_j)$ atunci $x_j \in X'$;
- dacă $x_i \in X'$ şi $\varphi(x_k, x_j) > 0$ atunci $x_k \in X'$.

Prin acest procedeu între x_0 şi oricare element din X' vom putea pune în evidenţă un drum. În X'' considerăm toate elementele lui X care nu aparţin lui X'. Procedând astfel, afirmăm că, în mod sigur $x_n \in X''$, deci $x_n \notin X'$.

Demonstrăm faptul că $x_n \notin X'$, prin metoda reducerii la absurd; presupunem contrariul, adică $x_n \in X'$, ceea ce înseamnă că există un lanţ care conţine nodurile $x_0, x_{i_1}, \ldots, x_{i_n}, x_n$ (inclusiv x_n !), format din arce cu orientarea înainte ($\rightarrow$), adică de la x_0 spre x_n, şi arce cu orientarea înapoi, ($\leftarrow$) (lanţul apare din construirea iterativă a mulţimii X').

Notăm cu

$\varepsilon_1 = \min_{\rightarrow} (c(u) - \varphi(u))$ (minim după arcele orientate înainte $\rightarrow$)

$\varepsilon_2 = \min_{\leftarrow} \varphi(u)$; (minim după arcele orientate înapoi $\leftarrow$)

$$\varepsilon = \min\{\varepsilon_1, \varepsilon_2\} > 0$$

Pe acest lanţ, putem îmbunătăţi fluxul, adică putem să-l majoram , în modul următor:

- φ se majorează cu ε pe toate arcele cu orientarea înainte şi
  se micşorează cu ε pe toate arcele cu orientare înapoi.

Procedând în acest fel constatăm că fluxul maxim φ_0 se modifică, crescând cu ε - contradicţie cu ipoteza că φ e flux maxim. Contradicţia apare din presupunerea făcută $x_n \in X'$, deci presupunerea făcută e falsă şi în consecinţă $x_n \in X''$.

Din ceea ce am arătat până aici rezultă că există nişte arce pentru care $\varphi(u)= c(u)\ \forall u \in U_{X'X"}$, unde $u=(x_k, x_l)$, cu $x_k \in X'$, $x_l \in X"$, iar pentru celelalte arce cu orientare înapoi $(\leftarrow)$ care fac parte din $U_{X"X'}$ avem $\varphi(u)=0$.

Rezultă că sunt adevărate relaţiile:

$$\sum_{u \in U_{x' x"}} \varphi(u) \quad - \quad \sum_{u \in U_{x" x'}} \varphi(u) \quad = \quad \sum_{u \in U_{x' x"}} c(u) \; - \; 0 \; = \; C\,(X', X").$$

Astfel am construit o tăietura $U_{x' x"}$ pentru care $\varphi_0 = C\,(X', X")$ c.c.t.d.

Definiţie. *Numim un lanţ de la x_0 la x_n, un lanţ de îmbunătăţire a fluxului în raport cu un flux φ dacă $\varphi(u)<c(u)\ \forall u$ din lanţ cu orientare înainte $(\rightarrow)$ şi $\varphi(u)>0$, $\forall u$ cu orientare înapoi $(\leftarrow)$ ale lanţului.*

Consecinţă. *Un flux φ este maxim dacă şi numai dacă nu există nici un lanţ de îmbunătăţire a fluxului în raport cu fluxul dat φ.*

Demonstraţia se face prin reducere la absurd.

Algoritmul lui Ford - Fulkerson

Din demonstraţia teoremei lui Ford - Fulkerson rezultă un algoritm simplu şi eficient de construire a fluxului maxim (algoritmul Ford - Fulkerson) care constă în 2 etape:

Etapa 1: facem să treacă un flux ce reprezintă soluţia iniţială, prin reţea; ca soluţie iniţială e avantajos să consideram un flux complet.

Etapa 2: verificăm dacă acest flux iniţial e maximal; dacă nu este trecem la determinarea unui alt flux, mai bun.

Definiţie. (flux complet) *Fluxul pentru care toate drumurile ce leagă x_0 de x_n conţin cel puţin un arc u pentru care $\varphi(u)=c(u)$ numit* ***arc saturat****, se numeşte* ***flux complet****.*

Pentru obţinerea unui flux complet este suficient să considerăm toate drumurile de la x_0 la x_n şi să facem să treacă prin fiecare drum un flux de aşa natură încât cel puţin un arc al acestui drum să fie saturat.

Etapa a doua constă în a verifica dacă există sau nu lanţuri de îmbunătăţire a fluxului în raport cu fluxul iniţial. Pentru această verificare se

foloseşte un procedeu de marcare a vârfurilor; acest procedeu de marcare a vârfurilor consta în:

- marcăm pe x_0 cu +;
- un vârf x_i fiind marcat, atunci se marchează cu $+x_i$ toate vârfurile x_j nemarcate, astfel încât să existe $(x_i,x_j) \in U$ nesaturat;
- marcăm cu $-x_i$ toate vârfurile x_k nemarcate pentru care există $(x_k,x_i) \in U$ şi $\varphi(x_k,x_i) \neq 0$;
- trebuie observat că procesul de marcaje menţionat înseamnă o cercetare sistematică a unui lanţ de îmbunătăţire a fluxului de la x_0 la x_n ;
- dacă ajungem să marcăm în acest fel şi pe x_n atunci vom putea modifica fluxul de-a lungul unui lanţ, şi după modificare se reia procesul de marcare a vârfurilor; ajungem la un flux maxim atunci când prin procedeul de marcare, x_n nu mai poate fi marcat; prin acest procedeu se obţine, evident, şi tăietura minimală.

Exemplu. Fie reţeaua de comunicaţii cea pe care o avem reprezentată în figura 4.14. Pentru această reţea dată ne propunem să determinam fluxul maxim.

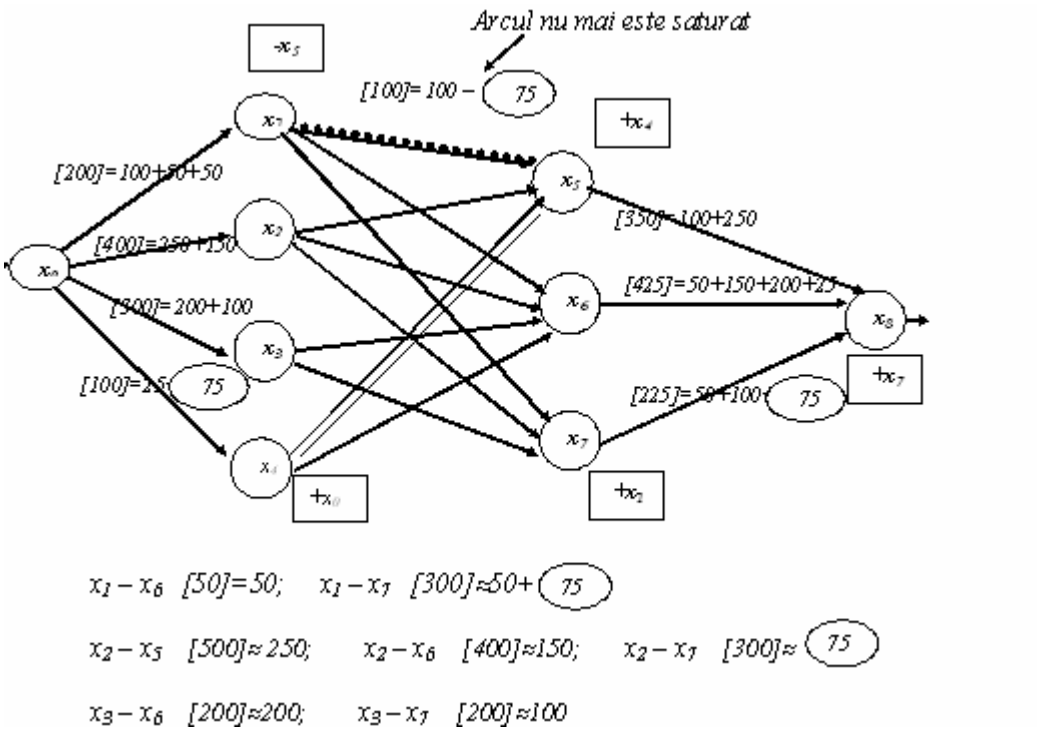


Fig. 4.14. Determinarea fluxului maxim într-o reţea de comunicaţii

Se dau capacităţile arcelor ca valori cuprinse între parantezele [ ]. Pentru ca să nu avem un desen încărcat, aceste capacităţi şi valori ale fluxului le-am precizat în partea de jos a desenului. De exemplu, pentru arcul (x_2, x_5) notat $x_2 - x_5$, care are capacitatea *500* şi pe care am considerat fluxul *250,* precizăm acest lucru sub forma $[500] = 250$.

1) *Soluţia iniţială* este echivalentă cu precizarea unui flux complet.

Se consideră toate drumurile ce leagă x_0 de x_8 şi saturăm câte un arc al acestor drumuri.

Pentru determinarea drumurilor luăm în mod sistematic arcul cel mai din stânga, altul decât cele deja alese; luăm capacitatea minimă a arcelor de pe acel drum ca fiind valoarea fluxului pe arcele respective din drum; arcul cu capacitatea minimă va fi un arc saturat; marcăm acest arc saturat. Procedând în acest fel vom obţine o soluţie iniţială.

În exemplul considerat, pornim din x_0 şi luăm arcul cel mai din stânga, spre nodul x_1 , cu capacitatea *200*, apoi din x_1 luăm arcul cel mai din stânga, spre nodul x_5 cu capacitatea *100* şi în sfârşit de aici luăm arcul spre nodul x_8 cu capacitatea *300.* Minimul dintre cele trei capacităţi *200, 100, 350* este *100; min{200, 100, 350}=100.* Alegem fluxul pe arcele acestui drum ca fiind acest minim *100,* deci arcul (x_1, x_5), în acest fel, va fi saturat; de aceea primele valori de după ≈ (sau =) pe aceste trei arce în reprezentarea de mai sus sunt *100.* Stabilim următorul drum care porneşte din x_0 folosind acelaşi procedeu de alegere din aproape în aproape a arcelor cele mai din stânga şi care nu sunt saturate. Acest drum conţine iniţial arcul (x_0, x_1) pe care mai putem considera un flux de cel mult *200-100=100;* următorul arc nesaturat şi cel mai din stânga este (x_1, x_6) cu capacitatea *50*; apoi arcul (x_6, x_8) cu capacitatea *425.* Determinăm *min { 100, 50, 425}=50* şi în consecinţă fixăm fluxul pe aceste arce *50,* iar arcul (x_1, x_7) devine saturat. Vom avea în această etapă pe arcul (x_0, x_1), $[200] \approx 100+50$, apoi pe arcul (x_1, x_6), $[50]=50$ (arc saturat), iar pe arcul (x_6, x_8), $[425] \approx 50$. Pe arcul (x_0, x_1) putem considera un flux de cel mult *200-(100+50)=50.* În continuare avem drumul (x_0, x_1, x_7, x_8), pe care fixăm un flux de *min{50, 300, 225}=50.* Arcul (x_0, x_1) devine saturat, având $[200] \approx 100+50+50$, apoi pe arcul (x_1, x_7), avem $[300] \approx 50$

2)*Marcarea vârfurilor*

Singurul arc nesaturat, care are extremitatea iniţială în x_0 , este $x_0 - x_4$. Vom începe procesul de marcare a nodurilor. Primul nod pe care-l marcăm cu + este x_0. Al doilea nod pe care-l marcăm cu + este x_4 precizând nodul precedent, scriem alături de x_2 pe $+x_0$. Următorul arc nesaturat pe care-l alegem, este (x_4, x_5); marcăm nodul x_5 cu $+x_4$. Din nodul x_5 nu pornesc arce nesaturate. Arcul (x_5, x_8) care porneşte din x_5 este saturat; revenim pe un arc înspre înapoi, astfel

încât fluxul să fie strict pozitiv, >0, de exemplu, spre x_1; acest nod îl marcăm cu $-x_5$. Continuăm marcarea nodurilor, următorul arc nesaturat fiind (x_1, x_7), deci marcăm nodul x_7 cu $+x_1$. În sfârşit vom marca nodul x_8 cu $+x_7$. Lanţul astfel construit este un lanţ de îmbunătăţire a fluxului. Acest lanţ îl reprezentăm în figura 4.15.

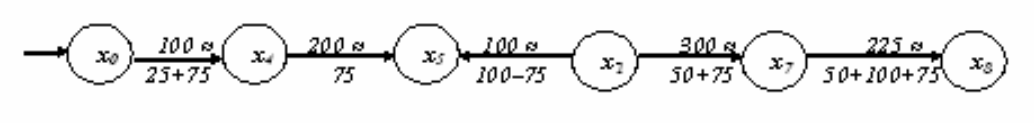


Fig. 4.15. Lanţ de îmbunătăţire a fluxului

Acest lanţ conţine arce cu orientarea înainte şi arce cu orientarea înapoi. Calculăm:

$$\rightarrow \quad \varepsilon_1 = \min_{\rightarrow}(c(u)-\varphi(u)) = \min\{100-25,\ 200-0,\ 300-50,\ 225-150\} = 75$$
$$(x_0,x_4)\quad (x_4,x_5)\quad (x_1,x_7)\quad (x_7,x_8)$$

$$\leftarrow \quad \varepsilon_2 = \min_{\leftarrow}\varphi(u) = \min\{100\} = 100$$

$$\varepsilon = \min\{\varepsilon_1, \varepsilon_2\} = \min\{75,100\} = 75.$$

Astfel, se modifică fluxul pe lanţul considerat, adunând 75 la fluxul arcelor orientate înainte $(\rightarrow)$ şi scăzând 75 la fluxul arcelor înaintate înapoi $(\leftarrow)$. Aceste valori sunt indicate pe lanţul de mai sus. După această modificare a fluxului pe arcele reţelei, arcele (x_0, x_4), (x_7, x_8) devin saturate.

Începem din nou marcarea vârfurilor: din x_0 nu mai porneşte nici un arc nesaturat , rezultă că nu mai putem obţine lanţuri de îmbunătăţire a fluxului, deci am obţinut fluxul maxim $\varphi_0=1000$, egal cu suma fluxurilor arcelor care pornesc din x_0, respectiv suma fluxurilor arcelor care sosesc în x_8.

Observatie: Drumurile le-am construit luând toate arcele cele mai din stânga (se poate alege şi arcul cel mai din dreapta pentru obţinerea aceluiaşi lucru – exerciţiu pentru cititor).

CAPITOLUL 5
NUMERE CATALAN ŞI SISTEME (OBIECTE) DEFINITE BINAR RECURSIV

5.1. Introducere

În lumea ştiinţifică este bine cunoscut faptul că multe fenomene din natură urmează aceleaşi legi cum şi în rezolvarea multor probleme diferite suntem conduşi la un acelaşi model matematic, avem, deci probleme diferite, dar totuşi echivalente în ce priveşte rezolvarea lor. În aceste situaţii este vorba de fapt de acea "*unitate în diversitate*" care guvernează lumea. Aşa, de exemplu, ajungem la bine cunoscutele numere Catalan (denumire dată în onoarea matematicianului belgian *Eugène Charles Catalan* (1814–1894). Vom menţiona în cele ce urmează unele probleme cunoscute din matematică şi informatică, rezolvarea cărora conduce la numerele Catalan . În final vom prezenta un model general pentru obiecte definite binar recursiv.

5.2. Binomul lui Newton şi numerele Catalan

5.2.1. Binomul lui Newton

Fie R mulţimea numerelor reale, N mulţimea numerelor naturale, iar $x, y \in R$.

Binomul lui Newton este dat de relaţia:

(2.1) $$(x + y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}, \quad n \in N,$$

unde $\binom{n}{k}$ reprezintă numărul combinărilor de n obiecte luate câte k.

Binomul lui Newton se scrie la fel şi pentru structuri matematice mult mai generale, dar rămânem în cele ce urmează în cadrul mulţimii numerelor reale.

Folosind binomul lui Newton deducem imediat unele proprietăţi ale combinărilor.

Dacă în binomul lui Newton punem $x=1, y=1$ se obţine

(2.2) $$\sum_{k=0}^{n} \binom{n}{k} = 2^n;$$

Dacă în binomul lui Newton punem $x=-1, y=1$ obţinem

(2.3) $$\sum_{k=0}^{n}\binom{n}{k}(-1)^k=0.$$

Punând $y=1$ în binomul lui Newton, avem:

(2.4) $$(1+x)^n=\sum_{k=0}^{n}\binom{n}{k}x^k,$$

iar dacă derivăm membrii stâng şi drept obţinem:

$$n(1+x)^{n-1}=\sum_{k=1}^{n}k\binom{n}{k}x^{k-1} \quad \text{sau} \quad nx(1+x)^{n-1}=\sum_{k=0}^{n}k\binom{n}{k}x^k.$$

Aici dacă punem $x = 1$ rezultă :

(2.5) $$\sum_{k=0}^{n}k\binom{n}{k}=n2^{n-1}.$$

Dacă integrăm cei doi membrii ai relaţiei *(4)* obţinem:

$$\frac{1}{n+1}(1+x)^{n+1}=\sum_{k=0}^{n}\frac{1}{k+1}\binom{n}{k}x^{k+1}$$

şi dacă punem aici $x = 1$ avem:

(2.6) $$\sum_{k=0}^{n}\frac{1}{k+1}\binom{n}{k}=\frac{1}{n+1}2^{n+1}.$$

Din binomul lui Newton dacă punem $y = 1 – x$ rezultă relaţia:

(2.7) $$\sum_{k=0}^{n}\binom{n}{k}x^k(1-x)^{n-k}=1,$$

care reprezintă o proprietate interesantă a şirului de polinoame fundamentale $p_{n.k}(x)=\binom{n}{k}x^k(1-x)^{n-k}$ şi care au fost folosite la definirea unor remarcabili operatori lineari şi pozitivi de tip Bernstein din teoria aproximării funcţiilor.

Se ştie că dacă în identitatea $(1+x)^m(1+x)^n=(1+x)^{m+n}$ folosim binomul lui Newton şi identificăm coeficienţii lui x^s din cei doi membrii, obţinem formula lui Vandermonde.

5.2.2. Formula lui Vandermonde. Şirul numerelor lui Catalan

(2.8) $$\sum_{k=0}^{s}\binom{m}{k}\binom{n}{s-k}=\binom{m+n}{s}.$$

Dacă convenim să notăm $[p]_i=p(p-1)...(p-i+1)$ atunci:

$$\binom{n}{k} = \frac{n(n-1)\ \ldots\ (n-k+1)}{k!} = \frac{[p]_k}{k!}.$$

Formula lui Vandermonde cu aceste notaţii devine:

(2.9) $$[m+n]_s = \sum_{k=0}^{s} \binom{s}{k} [m]_k [n]_{s-k}.$$

Să punem în formula lui Vandermonde $m=n$, $s=n$ şi să ţinem seama de proprietatea: $\binom{n}{k} = \binom{n}{n-k}$; obţinem identitatea:

(2.10) $$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}.$$

Această identitate ne sugerează o definiţie a şirului numerelor Catalan.

Definiţie. *Numerele*

(2.11) $$c_n = \frac{1}{n+1} \binom{2n}{n},\ n \geq 0\ ;\ c_0 = 1$$

formează şirul numerelor Catalan.

Pentru numerele Catalan se cunosc multe proprietăţi interesante. Menţionăm întâi relaţia:

(2.12) $$(n+2)c_{n+1} = (4n+2)c_n,\ n \geq 0,$$

pe care o verifică numerele Catalan definite mai sus.

Observaţie: Şirul numerelor Catalan poate fi definit şi prin relaţia de recurenţă *(2.12)*. Folosind această relaţie de recurenţă ca definiţie a numerelor Catalan şi înlocuind aici pe rând $n = 0, 1, ..., k - 1$, apoi făcând produsul membrilor stâng şi drept ai acestor relaţii, obţinem numărul Catalan c_k sub forma dată de relaţia *(2.11)*. Din cele arătate până aici rezultă că este adevărată lema care urmează:

***Lemă**: Fie* $c_n = \frac{1}{n+1}\binom{2n}{n}$, *$n \geq 0$; $c_0 = 1$. Sunt echivalente afirmaţiile:*

1. Numerele c_n sunt numere Catalan;
2. Numerele Catalan verifică relaţia (2.12).

5.2.3. Convoluţii şi numere Catalan

O altă proprietate interesantă a numerelor Catalan este cea dată de relaţia:

(3.13) $$c_n = \sum_{k=0}^{n-1} c_k\, c_{n-k+1}\ , \quad n \geq 1\ ;\ c_0 = 1.$$

Relaţia *(3.13)* o numim *relaţie de convoluţie.*

Observaţie*:* Relaţia *(3.13)* este folosită de unii autori pentru a defini numerele Catalan.

Fie $f : N \to N$ şi $f(n) = c_n$. Relaţia *(3.13)* se scrie sub forma unei ecuaţii funcţionale astfel:

(3.14) $$f(n) = \sum_{k=0}^{n-1} f(k)\, f(n-k)\ , \ n \geq 1;\ f(0) = 1.$$

Determinarea numerelor Catalan pornind de la definiţia dată de *(3.13)* se face ceva mai complicat comparativ cu cea dată de *(2.12)*. O metodă de determinare a numerelor Catalan se găseşte în cartea lui Knuth; ea se bazează pe folosirea funcţiilor generatoare ai căror coeficienţi c_n sunt chiar aceste numere. Fie

$$B(z) = c_0 + c_1 z + c_2 z^2 + ... + c_n z^n + ...$$

Dacă ţinem seama de *(3.13)* constatăm uşor că $2B^2(z) - B(z) + 1 = 0$, deci

$$B(z) = \frac{1}{2z}(1 - \sqrt{1-4z})$$

Dezvoltând în serie binomială funcţia $(1 - 4z)^{1/2}$ şi identificând coeficienţii dezvoltărilor în serie din membrul stâng şi membrul drept, obţinem numerele Catalan, adică

$$c_n = \frac{1}{n+1}\binom{2n}{n}$$

sau că soluţia ecuaţiei funcţionale *(3.14)* este: $f(n) = \frac{1}{n+1}\binom{2n}{n}$.

Din cele arătate până aici avem:

Lemă. *Fie* $c_n = \frac{1}{n+1}\binom{2n}{n}$; $n \geq 0$.

Afirmaţiile care urmează sunt echivalente:

1. *Numerele c_n sunt numere Catalan:*
2. *Dacă are loc relaţia (3.13) atunci în această relaţie c_n sunt numere Catalan.*

Punând la un loc rezultatele din cele două leme rezultă următoarea proprietate:

Teorema 1. *Fie* $c_n = \frac{1}{n+1}\binom{2n}{n}$, $n \geq 0$. *Următoarele afirmaţii sunt echivalente:*

1. *Numerele c_n sunt numere Catalan.*
2. *Numerele Catalan verifică relaţia (2.12).*
3. *Dacă are loc relaţia (3.13) atunci c_n din această relaţie sunt numere Catalan.*

5.2.4. Principiul multiplicării şi principiul însumării

Principiul multiplicării

Dacă o activitate *A* se realizează în doi paşi, adică ca o succesiune a celor două activităţi A_1 respectiv A_2 şi dacă n_1 este numărul de moduri posibile pentru realizarea activităţii A_1, iar n_2 este numărul de moduri posibile pentru realizarea activităţii A_2 atunci activitatea A ($A = A_1A_2$) poate fi realizată în n_1n_2 moduri posibile diferite.

Acest principiu al multiplicării îl ilustrăm sub forma unui arbore în figura 5.1.

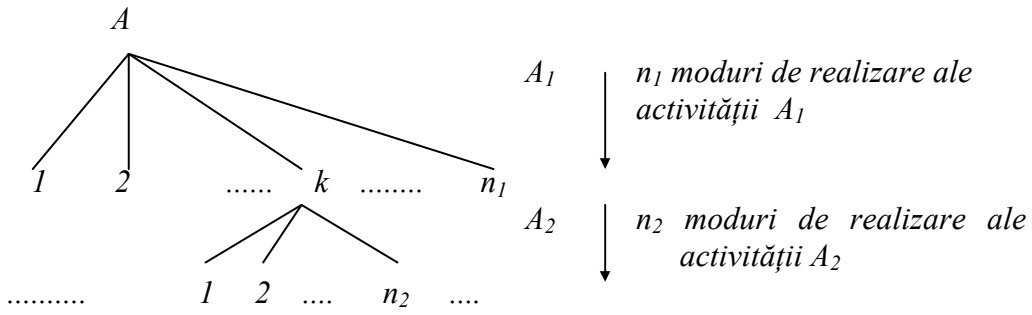


Fig. 5.1. Principiul multiplicării

Numărul total de moduri de realizare ale activităţii $A = A_1A_2$ este n_1n_2 şi coincide cu numărul total de drumuri de la rădăcină la nodurile terminale ale acestei arborescenţe.

Acest principiu se generalizează pentru situaţia când activitatea A se realizează în t paşi succesivi, adică $A = A_1A_2....A_t$, iar pentru $k \in \{1, 2, ... , t\}$ activitatea A_k se poate realiza în n_k moduri diferite. Numărul total de moduri diferite de realizare ale activităţii A se poate demonstra prin inducţie matematică că este $n_1.n_2 . ... .n_t$.

În limbajul teoriei mulţimilor acest principiu are legătură cu proprietatea : $|A_1 \times A_2 \times ... \times A_t| = |A_1|.|A_2| ... |A_t|$, unde $A_1, A_2, ..., A_t$ sunt mulţimi, iar prin $|A|$ am notat cardinalul mulţimii A.

Principiul însumării

Dacă A_1 şi A_2 sunt activităţi distincte şi realizarea unei activităţi A este echivalentă cu realizarea activităţii "A_1 sau A_2" iar activitatea A_1 se poate realiza în n_1 moduri diferite, apoi activitatea A_2 se poate realiza în n_2 moduri diferite, atunci activitatea A adică "A_1 sau A_2" se poate realiza în $n_1 + n_2$ moduri diferite.

Acest principiu al însumării îl ilustrăm în figura 5.2.

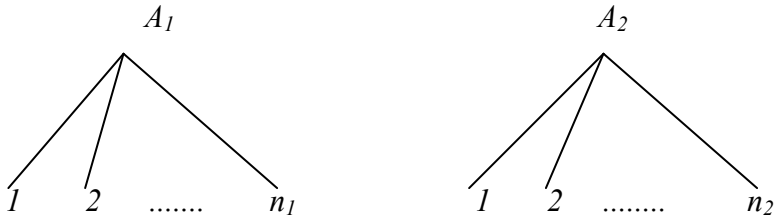


Fig. 5.2. Principiul însumării

Numărul total de moduri de realizare ale activităţii A_1 *sau* A_2 este $n_1 + n_2$.

Dacă $A_1,...A_m$ sunt activităţi distincte, ca în cazul precedent, principiul însumării se generalizează pentru activitatea A_1 *sau* A_2 *sau ... sau* A_n. Dacă pentru $k \in \{1, 2, ..., m\}$ activitatea A_k se poate realiza în n_k moduri diferite atunci activitatea A_1 *sau* A_2 *sau ... sau* A_m se poate realiza în $n_1 + n_2 + ... + n_m$ moduri diferite.

Acest principiu este legat în teoria mulţimilor de proprietatea $|A_1 \cup A_2 \cup ... \cup A_m| = |A_1| + |A_2| + ... + |A_m|$, care are loc dacă mulţimile A_i, $i \in \{1, 2, ..., n\}$ sunt două câte două disjuncte, adică $A_i \cap A_j = 0$, dacă $i \neq j$.

5.2.5. Aplicaţii

5.2.5.1. Partiţionarea poligoanelor convexe

Să considerăm fixat un poligon convex cu $n + 2$ laturi şi ne propunem ca să-l partiţionăm în triunghiuri ale căror laturi nu se intersectează, respectiv n triunghiuri. Această partiţionare o numim *triangulaţie*. Acest lucru se poate realiza în mai multe feluri. Pentru $n=3$ avem *5* situaţii posibile (fig. 5.3.).

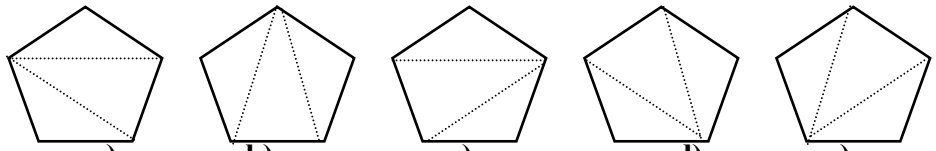


Fig. 5.3. Triangulaţiile unui poligon convex cu 3+2 laturi

În general, se demonstrează că numărul acestor situaţii posibile de partiţionare, adică numărul de triangulaţii a unui poligon convex cu $n + 2$ laturi în n triunghiuri, ducând $n - 1$ linii care unesc vârfuri opuse fără ca acestea să se intersecteze este numărul Catalan c_n.

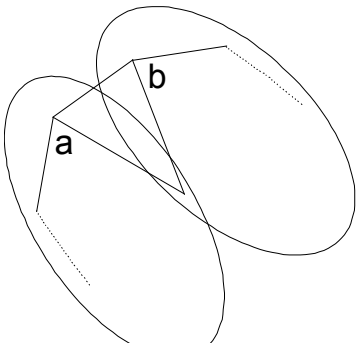


Fig. 5.4.

Pentru demonstraţie folosim principiul mutliplicării şi principiul însumării.
Fie un poligon convex cu $n + 2$ laturi în care considerăm latura "ab" a acestui poligon (fig. 5.4.). Cu latura "ab" alegem un triunghi oarecare obţinut prin unirea vârfurilor a, respectiv b cu un alt vârf al poligonului. Fie la stânga acestui triunghi, de exemplu, poligonul convex pentru care sunt posibile c_k triangulaţii; la dreapta avem poligonul convex cu $n - k + 1$ laturi. Dacă ţinem seama de principiul multiplicării, atunci numărul poziţionărilor posibile în sensul precizat mai sus este $c_k c_{n-k+1}$. Acum făcând uz de principiul însumării, adică selectarea unui triunghi, se poate face în n feluri, deci făcându-l pe k să ia valori de la 0 la $n - 1$, putem obţine numărul c_n de posibilităţi de partiţionare.

Avem: $c_n = c_0 c_{n-1} + c_1 c_{n-2} + \ldots + c_{n-1} c_0$.

Ţinând seama de teorema de mai sus, rezultă că numărul triangulaţiilor posibile ale unui poligon convex cu $n + 2$ laturi este numărul Catalan c_n. Catalan a ajuns la aceste numere ocupându-se tocmai de rezolvarea problemei prezentate aici.

5.2.5.2. Numărul arborilor binari

Arborii binari se definesc astfel (recursiv): *Un arbore binar este fie vid, fie conţine o rădăcină şi un arbore binar în stânga rădăcinii şi un arbore binar în dreapta rădăcinii.*

Un arbore binar este o arborescenţă binară, deci fiecare nod are cel mult doi succesori. Nodurile care nu au succesori se numesc noduri terminale, celelalte noduri ale unui arbore binar se numesc noduri neterminale. Ne punem problema determinării numărului de arbori binari cu n noduri neterminale. Dacă luăm un nod din cele n noduri neterminale şi considerăm nodul rădăcină *1*, iar în stânga acestuia considerăm arbori binari cu k noduri, iar în dreapta arbori binari cu $n - k - 1$ noduri, atunci avem pentru numărul arborilor binari cu n noduri neterminale c_n relaţia:

Fig. 5.5.

$$c_n = c_0c_{n-1} + c_1c_{n-2} + \ldots + c_{n-1}c_0.$$

Prin definiţie $c_0 = 1$, apoi $c_1 = 1$, $c_2 = 2$,
iar dacă $n = 3$ avem următorii arbori binari din fig. 5.5, adică $c_3 = 5$.

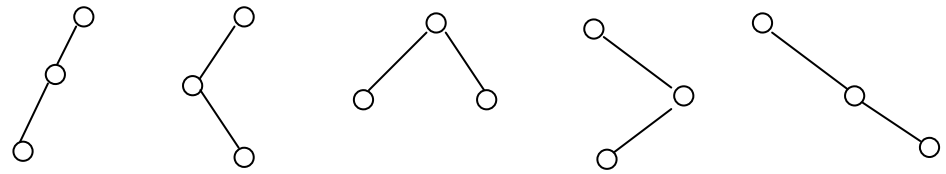


Fig. 5.5. Arbori binari cu 3 noduri terminale

În general, aşa cum rezultă din cele arătate la 3, avem: $c_n = \frac{1}{n+1}\binom{2n}{n}$.

Observaţie: Recursivitatea din definiţia arborilor binari am aplicat-o de n ori selectând de fiecare dată un nou nod din cele n noduri date şi aşezându-l în stânga sau dreapta unui nod rădăcină.

5.2.5.3. Numărul "rutelor bune" într-o grilă

Să considerăm un sistem de axe rectangulare şi să ducem paralele la aceste axe prin punctele *(k, 0)* respectiv *(0, k), k ∈ {0, ... , n}*; obţinem o *grilă de tip n X n.* Fie acum punctele de pe diagonala principală *(0, 0)*, respectiv *(n, n)*, diametral opuse. O astfel de grilă cu *n=5* avem în figura 5.6. Parcurgând segmente orizontale şi verticale, de la un punct la un altul vecin din grilă, întotdeauna putem construi o *rută* între *(0,0)* şi *(n, n)*. Numărul segmentelor orizontale şi verticale dintr-o rută între *(0, 0)* şi *(n, n)* este *2n*.

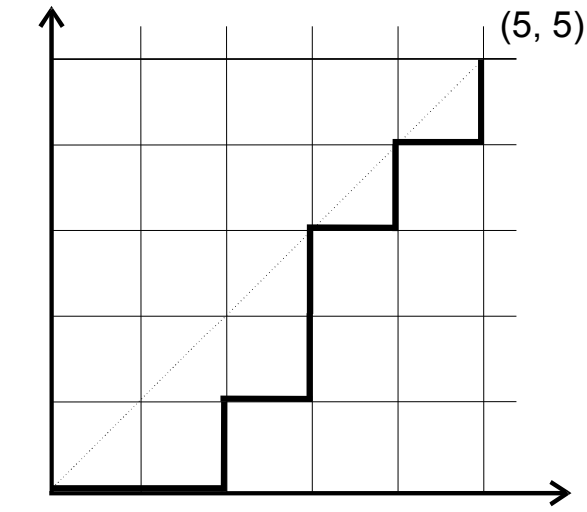


Fig. 5.6. Grilă de tip 5 X 5

Deci, numărul tuturor rutelor între *(0, 0)* şi *(n, n)* este $\binom{2n}{n}$.

Vom numi o rută între *(0, 0)* şi *(n, n)* ca fiind o *rută bună*, dacă parcurgerea se face prin mişcări spre dreapta şi în sus dar fără a depăşi linia diagonală, ea poate fi doar atinsă. Un punct *(a, b)* al unei rute bune are proprietatea $b \geq a$ şi evident există într-o astfel de rută un punct *(k, k)*, $0 < k \leq n$. In grila din figura alăturată am îngroşat o rută bună.

O rută între *(0, 0)* şi *(n, n)* care nu este o rută bună o numim *rută rea*. Într-o rută rea întotdeauna avem cel puţin o depăşire a liniei diagonale.

Pentru numărul rutelor bune B_n şi numărul rutelor rele R_n între *(0, 0)* şi *(n, n)* dintr-o grilă de tip $n \times n$ avem relaţia $B_n + R_n = \binom{2n}{n}$

În grila de tip 5×5 din fig. 5.7. a) am îngroşat o rută rea.

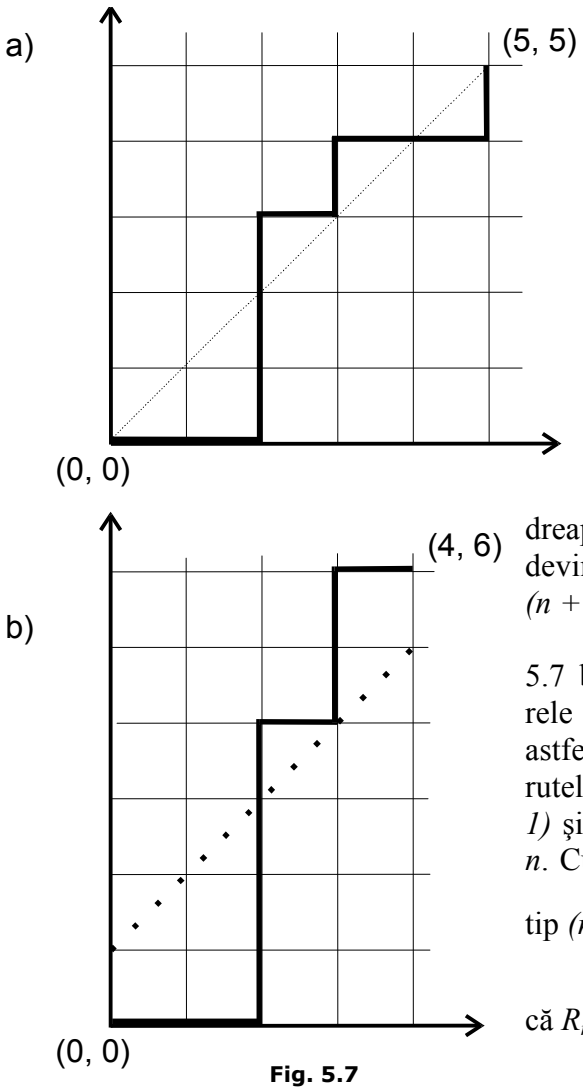


Fig. 5.7

Asociem fiecărei grile de tip $n \times n$ o grilă de tip $(n - 1) \times (n + 1)$ şi facem ca oricărei rute rele în grila de tip $n \times n$ să-i corespundă o rută între $(0, 0)$ şi $(n - 1, n + 1)$ din grila de tip $(n - 1) \times (n + 1)$. Cum o rută rea întotdeauna depăşeşte în cel puţin un punct linia diagonală, atunci în continuare de la prima mişcare spre dreapta, se vor înlocui toate mişcările spre dreapta prin mişcări în sus, iar mişcările în sus prin mişcări la dreapta; în felul acesta o rută rea devine o rută în reţeaua de tip $(n - 1) \times (n + 1)$.

În grila de tip 4×6 din figura 5.7 b) am îngroşat ruta asociată rutei rele din figura 5.7 a). Se stabileşte, astfel, o corespondenţă biunivocă între rutele din reţeaua de tip $(n - 1) \times (n + 1)$ şi rutele rele din reţeaua de tip $n \times n$. Cum numărul rutelor din reţeaua de tip $(n - 1)\times(n + 1)$ este $\binom{2n}{n-1}$ rezultă că $R_n = \binom{2n}{n-1}$.

Fig. 5.8

Deci, $B_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}$, adică $B_n = c_n$ care reprezintă *numărul Catalan*.

Pentru determinarea numărului rutelor bune între $(0, 0)$ şi (n, n) dintr-o reţea pătratică se poate face şi un alt raţionament. Ţinem seama de definiţia unei rute bune care ne asigură că pe aceasta există punctul (k, k), $0 < k \le n$. Fixăm punctul (k, k) şi atunci numărul rutelor bune între $(0, 0)$ şi (k, k) este c_{k-1}, iar numărul rutelor bune între (k, k) şi (n, n) este c_{n-k}. Numărul rutelor bune care conţin punctul (k, k) în baza principiului multiplicării este $c_{k-1}c_{n-k}$. Rutele bune

care ating prima dată diagonala în *(k, k)* sunt distincte de rutele bune care ating prima dată diagonala în *(k', k'), k'≠ k* şi atunci în baza principiului însumării avem:

$$c_n = c_0c_{n-1} + c_1c_{n-2} + \ldots + c_{n-1}c_0 = \sum_{k=0}^{n-1} c_{n-k-1}.$$

5.2.5.4. Grupări de factori (perechi de paranteze)

Să considerăm produsul $a_0*a_1*\ldots*a_n$, în care avem $n+1$ factori, iar operatorul "$*$" apare de n ori. Efectuarea acestui produs fără a schimba ordinea factorilor se va face recursiv pornind de la efectuarea produsului a doi factori, apoi a altor doi factori ş.a.m.d. Aceasta revine la considerarea diferitelor moduri de grupări posibile de câte doi factori. Astfel, dacă $n = 2$, deci avem de efectuat produsul $a_0*a_1*a_2$, atunci grupările posibile sunt: $(a_0*a_1)*a_2$, $a_0*(a_1*a_2)$. Dacă $n = 3$, adică avem produsul $a_0*a_1*a_2*a_3$ atunci cele cinci grupări posibile sunt: $((a_0*a_1)*(a_2*a_3))$, $(a_0*(a_1*(a_2*a_3)))$, $(((a_0*a_1)*a_2)*a_3)$, $((a_0*(a_1*a_2))*a_3)$, $(a_0*((a_1*a_2)*a_3))$. Notăm cu c_n numărul grupărilor posibile de acest fel a celor $n + 1$ factori.

Să observăm că din cei n operanzi "$*$" oricare din ei poate fi efectuat ultimul. Dacă alegem cel de-al $(k + 1)$-lea operand "$*$" ca fiind ultimul care se efectuează, atunci la stânga acestuia sunt k operanzi "$*$" care determină un număr de c_k grupări de factori, iar la dreapta $(n-k-1)$ operanzi "$*$" care determină c_{n-k-1} grupări de factori şi numărul total de grupări de factori va fi c_kc_{n-k-1}. In acest fel deducem că numărul total de grupări de factori din produsul celor $n + 1$ factori va fi $c_n= c_0c_{n-1}+c_1c_{n-2}+\ldots+c_{n-1}c_0$. Rezultă că numărul de grupări posibile pentru efectuarea produsului a n factori în ordinea dată este numărul Catalan c_n.

5.2.5.5. Parcurgeri în tablouri triunghiulare

Să considerăm un tablou triunghiular $A = (a_{ij}),\ i \le j,\ i, j \in \{0, \ldots, n\}$. Ne propunem să parcurgem elemente ale acestui tablou mergând numai la dreapta sau în jos, pornind de la elementul a_{00} şi ajungând la a_{nn}. Numim o astfel de parcurgere *traseu corect*. Fie $n = 4$, avem tabloul din fig. 5.8. În acest tablou este marcat un traseu corect. Ne propunem să determinăm numărul tuturor traseelor corecte de la a_{00} la a_{nn}. Uşor constatăm că o astfel de parcurgere este echivalentă cu o rută bună într-o grilă de tip $n\times n$. Un traseu corect conţine $2n$ elemente ale tabloului A.

Deci, numărul traseelor corecte între a_{00} şi a_{nn} este numărul Catalan c_n.

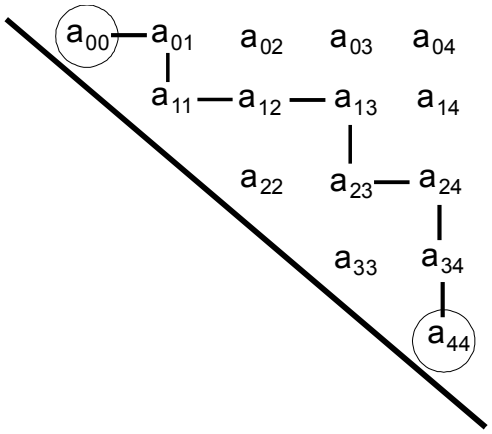


Fig. 5.8. Tablou triunghiular pentru n = 4

Notăm elementele dintr-un traseu corect prin $b_1, b_2,..., b_{2n}$. Evident $b_1 = a_{00}$ iar $b_{2n} = a_{nn}$.

Un element b_k, $k \in \{1, ..., 2n\}$ este un element a_{ij} pe care îl obţinem după o variaţie convenabilă a primului şi a celui de-al doilea indice. Obţinerea tuturor secvenţelor $b_1, ..., b_k$ prin variaţii convenabile ale indicilor elementelor a_{ij} este o problemă importantă.

5.3. Obiecte definite binar recursiv

5.3.1. Obiecte de asamblare binară

Presupunem că la construirea sau formarea unor obiecte, folosim elemente atomice de acelaşi fel şi de mai multe ori. Notăm cu ε lipsa unui element atomic. Construcţia acestor obiecte cu ajutorul unor elemente atomice se realizează după anumite procedee bine precizate. Există multe exemple de obiecte care se construiesc prin asamblare repetată, pornind de la un element atomic şi aplicând o anumită regulă de un număr finit de ori pentru a ajunge la obiectul dorit. Obiectul vid îl notăm $\emptyset$ şi el nu conţine elemente atomice. Dintre aceste proceduri de formare ale unor obiecte vom alege, pentru cele ce urmează, aşa numita *asamblare binară* care se bazează pe concatenarea a două obiecte, definită printr-o procedură concretă.

Fără a folosi simboluri speciale , convenim să notăm concatenarea la stânga a obiectului X cu elementul atomic r prin Xr, iar concatenarea la dreapta a elementului atomic r cu obiectul Y prin rY; în urma acestor concatenări obţinem obiecte noi.

Definiţie. *(Asamblare binară) Dacă r, $r \neq \varepsilon$, este un element atomic, iar X şi Y sunt obiecte formate prin concatenare din elemente atomice de acelaşi fel cu r, atunci XrY, unde am aşezat întâi pe X în stânga lui r, iar apoi pe Y în dreapta lui r, ţinând seama de un anumit mod de definire a acestor concatenări de entităţi, se numeşte* ***asamblare binară****; în urma acestei operaţii de asamblare binară se obţine un nou obiect. Operaţia inversă o numim* ***dezasamblare binară****.*

Definim acum, recursiv, obiectele de asamblare binară, pe care le vom numi mai scurt, *obiecte*. Definiţiile recursive se folosesc mult în informatică .

Definiţie. *(Obiecte de asamblare binară) Notăm cu r , $r \neq \varepsilon$, un element atomic.*

Spunem că:

a) Ø este obiect;

b) Un element atomic r, este obiect;

c) Dacă X şi Y sunt obiecte, atunci prin asamblarea binară XrY obţinem un nou obiect Z, Z := XrY;

d) Repetând de un număr finit de ori a),b),c) obţinem prin această construcţie un ***obiect de asamblare binară****.*

Observaţie. Prin operaţia de dezasamblare binară un obiect dat, considerat ca o entitate (un întreg), se poate descompune recursiv până când în componenţa obiectului dat avem numai elemente atomice.

Pentru (dez)asamblarea binară pot fi date diferite modalităţi de *reprezentare*. Astfel, putem avea reprezentări geometrice , reprezentări prin tablouri etc.

Numărul *n, n>0,* al elementelor atomice dintr-un obiect obţinut prin asamblare binară îl presupunem cunoscut. Vom considera în cele ce urmează elementele atomice definite pentru *n=1,* iar acestea, în fiecare problemă dată, pot fi precizate şi identificate diferit, conform unor reguli bine stabilite.

Notăm cu $A=\{r_1, r_2, ...,r_n\}$ mulţimea elementelor atomice (de acelaşi fel) din care se formează obiectul *O*. În cele ce urmează vom prezenta cinci aplicaţii de construire sau descompunere ale unor obiecte, printr-un procedeu de (dez)asamblare binară recursivă, formate dintr-un număr dat de elemente atomice.

5.3.2. Exemple

Se dau în continuare câteva exemplificări a modului de formare a unor obiecte concrete, prin (dez)asamblare binară, pornind de la elemente atomice nevide, definite pentru *n=1* şi convenabil alese. O parte din aceste exemple sunt cunoscute.

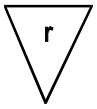
În aceste exemple se vor indica reprezentările elementelor atomice *r,* $r \neq \varepsilon$, şi modalităţile de asamblare binară *X r Y* a acestora, cu scopul de a obţine obiecte nevide.

5.3.2.1. Triangulația

Considerăm fixat un poligon convex cu $n + 2$ laturi şi ne propunem să-l partiționăm în n triunghiuri ale căror laturi, ce unesc vârfuri ale poligonului dat, nu se intersectează. Această partiționare o numim *triangulație*. Acest lucru se poate realiza în mai multe feluri. Fiecare triangulație se face pe baza unei dezasamblări binare recursive. Un triunghi al triangulației poligonului dat, ales ca element atomic, formează alte două poligoane, de o parte a triunghiului vom avea poligonul convex *X,* iar de cealaltă parte, poligonul convex *Y.* Concatenarea a două triunghiuri, de tipul celor menționate, este definită prin aceea că ele au în comun o latură.

Avem:

a) ($n=1$) element atomic r :: =



b) ($n>1$) (dez)asamblarea $X\ r\ Y$ :: =

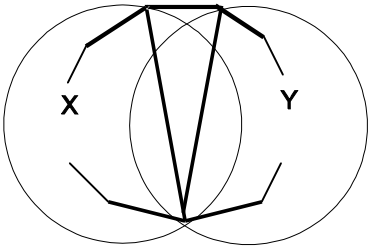


Fig. 5.9. Triangulaţiile unui poligon convex

5.3.2.2. Arbori binari

Ne propunem să construim prin asamblare binară arborii binari cu n, $n>0$, noduri (arbori binari nevizi). Dacă luăm un nod din cele n noduri şi considerăm nodul rădăcină *1*, atunci în stânga acestuia considerăm arbori binari cu k noduri, iar în dreapta, arbori binari cu $n - k - 1$ noduri, pentru $0 \le k < n$ (fig. 5.10).

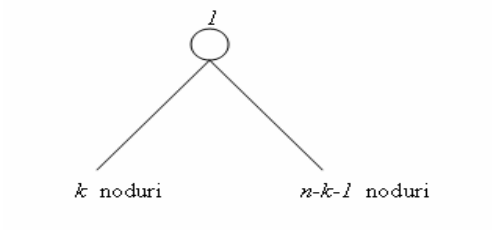


Fig. 5.10

Avem:

a) *(n=1)* element atomic *r* :: = " o " care reprezintă un nod

b) *(n>1)* asamblarea *X r Y* ::=

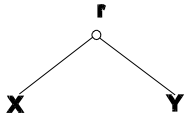


Fig. 5.11

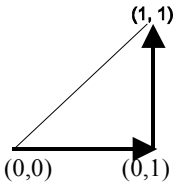
5.3.2.3. Rute bune într-o grilă

Să considerăm un sistem de axe rectangulare şi să ducem paralele la aceste axe prin punctele *(k, 0)* respectiv *(0, k), k ∈ {0, ... , n}*, *n>0*; obţinem o *grilă de tip* $n \times n$. Fie acum punctele de pe diagonala principală *(0, 0),* respectiv *(n,n), n>0,* diametral opuse. Considerând segmente orizontale şi verticale care pleacă de la un punct la un alt punct vecin din grilă, întotdeauna putem construi o *rută* între *(0,0)* şi *(n, n), n>0*. Numărul segmentelor orizontale şi verticale dintr-o rută între *(0, 0)* şi *(n, n)* este *2n*.

Vom numi o rută între *(0, 0)* şi *(n, n)* ca fiind o *rută bună*, dacă parcurgerea se face prin mişcări spre dreapta şi în sus, dar fără a depăşi linia diagoanală; ea poate fi doar atinsă. Dacă considerăm puncte (a, b) din grilă, ale unei rute bune, adică puncte cu proprietatea $b \geq a$, evident, există într-o astfel de rută un punct (k, k), $0 < k \leq n$.

Avem:

a) *(n=1)* element atomic *r* :: = ruta bună



b) *(n>1)* asamblarea *X r Y* :: = legarea în *(k,k), 0<k≤n,* a două rute bune la stânga, între *(0,0)* şi *(k,k)*, respectiv, la dreapta, între *(k,k)* şi *(n,n), n>0*;

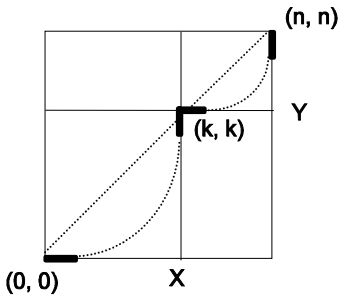


Fig. 5.12

5.3.2. 4. Grupări de factori (perechi de paranteze)

Să considerăm produsul $a_0*a_1*...*a_n$, în care avem $n+1$ factori, iar operatorul "$*$" apare de n ori, $n>0$. Efectuarea acestui produs, fără a schimba ordinea factorilor, se va face recursiv pornind de la efectuarea produsului a doi factori, apoi a altor doi factori ş.a.m.d. Aceasta revine la considerarea diferitelor moduri de grupări posibile de câte doi factori.

Astfel, dacă $n = 3$, adică avem produsul $a_0*a_1*a_2*a_3$ atunci cele cinci grupări posibile sunt: $((a_0*a_1)*(a_2*a_3))$, $(a_0*(a_1*(a_2*a_3)))$, $(((a_0*a_1)*a_2)*a_3)$, $((a_0*(a_1*a_2))*a_3)$, $(a_0*((a_1*a_2)*a_3))$.

În general, oricare din cei n operatori "*" ai produsului de n factori considerat, poate fi ultimul care se efectuează. Alegând pe al k-lea, $k>0$, operator "*", la stânga acestuia, produsul notat X, mai are $k-1$ operatori "*" , iar la dreapta, $n-k$ operatori "*" ai produsului de factori notat Y.

Astfel avem,

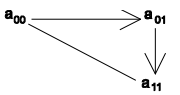
a) ($n=1$) element atomic $r ::= (a*b)$, produsul a doi factori

b) ($n>1$) asamblarea $X\ r\ Y ::$ $\begin{cases} ((X*b), daca\ Y=\phi) \vee ((a*Y), daca\ X=\phi) \\ (X*Y),\ altfel \end{cases}$

5.3.2.5. Trasee corecte în tablouri triunghiulare

Să considerăm un tablou triunghiular $A= (a_{ij}),\ i \le j;\ i, j \in \{0,1, ..., n\}$, $n>0$. Ne propunem să parcurgem elemente ale acestui tablou mergând numai la dreapta sau în jos, pornind de la elementul a_{00} şi ajungând la a_{nn}. Numim o astfel de parcurgere *traseu corect*. Uşor constatăm că o astfel de parcurgere este echivalentă cu o rută bună într-o grilă de tip $n \times n$. Un traseu corect conţine $2n+1$ elemente ale tabloului A. Construcţia acestor trasee corecte se poate face printr-o asamblare binară, ca în figura 5.13.

a) *(n=1)* element atomic *r* :: =



b) *(n>1)* asamblare *X r Y* :: =

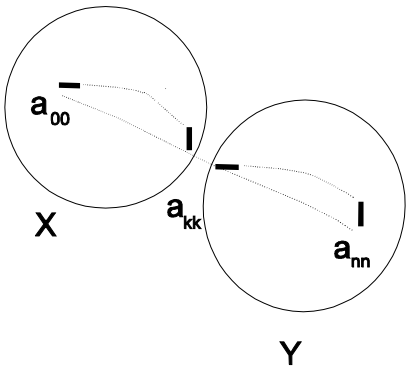


Fig. 5.13

5.4. Numărul obiectelor de asamblare binară

Vom considera obiectele *O* definite binar recursiv, formate din *n* elemente atomice, $A= \{r_1, r_2,...,r_n\}$; acestea se obţin folosind de *n* ori recursia în asamblarea binară. Putem determina numărul tuturor obiectelor formate din *n* elemente atomice şi care se obţin printr-un procedeu binar recursiv.

Teoremă. *Numărul tuturor obiectelor ce se pot forma din n elemente atomice printr-un procedeu recursiv de asamblare binară, este numărul Catalan* c_n.

Demonstraţie. Fie mulţimea $A= \{r_1, r_2,...,r_n\}$ a elementelor atomice din care se formează prin asamblare binară un obiect *O*. Notăm cu c_n numărul tuturor obiectelor de (dez)asamblare binara. Dacă alegem un element atomic r_p , atunci obiectul *O* obţinut prin asamblarea binară $X\ r_p\ Y$, are la stânga lui r_p, obiectul *X* care poate să conţină, de exemplu, *k-1* elemente atomice; atunci *Y* va conţine evident *n-k* elemente atomice, pentru $0<k\le n$. Notăm cu c_{k-1} numărul posibilităţilor de asamblare binară pentru construirea obiectului *X* şi cu c_{n-k} numărul posibilităţilor de asamblare binară pentru construirea obiectului *Y*. Avem o activitate care se realizează în doi paşi, întâi construirea lui X, apoi a lui Y. Conform principiului multiplicării, numărul total de posibilităţi în asamblarea $X\ r_p\ Y$ este $c_{k-1}\ c_{n-k}$. Cum alegerea elementelor atomice din *A* este arbitrară, pentru $p \in \{1, ..., n\}$, prin folosirea principiului însumării, obţinem relaţia de convoluţie

$$c_n = c_0c_{n-1} + c_1c_{n-2} + \ldots + c_{n-1}c_0$$

Numerele care au această proprietate se numesc *numere Catalan* şi avem $c_n = \frac{1}{n+1}\binom{2n}{n}$.

Observaţie. Teorema enunţată şi demonstrată aici are caracter general, dar ea în acelaşi timp are o importanţă practică deosebită. Acest lucru rezultă din aplicaţiile date mai sus şi cu siguranţă că se pot găsi şi alte aplicaţii informatice importante.

5.5. Construcţia obiectelor de asamblare binară

Ne propunem să dăm un algoritm de construire a obiectelor de asamblare binară formate din *n* elemente atomice. Astfel de construcţii se pot face în două moduri:

a) pornind de la un element atomic şi adăugând (prin concatenare) din aproape, în aproape, câte un nou element atomic, până când ajungem la obiectul format din cele *n* elemente atomice. Acest procedeu defineşte *compunerea* obiectelor de asamblare binară.

b) pornind de la numărul n de elemente atomice date şi *descompunând* în două clase de obiecte distincte *X* şi *Y,* ţinând seama de definiţia de asamblare binară *XrY*.

O construcţie iterativă, care porneşte de la cele *n* elemente atomice date, se poate face pe baza algoritmului pe care îl vom prezenta în continuare. Convenim ca să descriem acest algoritm în limbaj pseudocod.

ConstrObiect(n) *//n, n>0, – numărul elementelor atomice*
/ Pentru construirea unui obiect de asamblare binară concret, Z:=XrY, vom apela procedurile de concatenare specifice: "rY", respectiv "Xr";*
•Notăm cu O_k mulţimea obiectelor de asamblare binară formate din k elemente atomice;
•Vom avea: $O_k:=\{O_k^i \mid i=\overline{1,c_k}\}$; c_k – numărul Catalan;
•Notăm cu r^k elementul atomic iniţial (baza, rădăcina) dintr-un obiect de asamblare binară cu k elemente atomice;
•Avem: $O_0:=\{O_0^1\}:=\{Ø\}:=Ø$; $O_1:=\{O_1^1\}:=\{r^1\}$; $rØ:=Ør:=r$
**/*
{ $O_0^1:=Ø$; $O_1^1:=r^1$;
$c_0:=1$;

for *k:=1;n* ***do***

{ *i:=0; // i - contor al obiectelor de asamblare binară*

for *j:=0;k-1* ***do***

/ Avem pentru k fixat,* $O_k:=O_j\, r^k\, O_{k-1-j}$ *adică*

$\{O_k^i \mid i=\overline{1,c_k}\}=\{O_j^s \mid s=\overline{1,c_j}\}\, r^k\, \{O_{k-1-j}^d \mid d=\overline{1,c_{k-1-j}}\}$

**/*

for *d:=1;*c_{k-1-j} ***do***

for *s:=1;*c_j ***do***

{ *i:=i+1;*

$O_k^i := O_j^s\ r^k\, O_{k-1-j}^d$

/ Apelăm procedurile de concatenare: “* $r^k\, O_{k-1-j}^d$ *”, “* $O_j^s\, r^k$ *“*

pentru obţinerea obiectului O_k^i **/*

}

$c_k:=i;$

}

}

CAPITOLUL 6
TIMPUL ŞI STĂRILE GLOBALE

6.1. Importanţa timpul în sisteme distribuite. Introducere

În acest capitol vom prezenta câteva aspecte privind problema timpului în sisteme distribuite [COU01[9], RAY91[10]etc.]. În primul rând trebuie să menţionăm că timpul este de o necesitate practică indiscutabilă. De exemplu, noi cerem adesea calculatoarelor din lumea întreagă să cronometreze tranzacţiile comerciale electronice care se fac la un moment dat, dar, pot ele oare face acest lucru cu exactitate? Timpul este, de asemenea, un important suport teoretic în înţelegerea desfăşurării unei construcţii distribuite. Putem afirma cu certitudine că timpul este o problemă importantă în orice sistem distribuit. Fiecare calculator poate avea propriul său ceas fizic de măsurare a timpului, dar ceasurile de regulă mai şi deviază şi nu le putem sincroniza perfect. Noi trebuie să căutăm algoritmi pentru a aproxima sincronizarea ceasurilor fizice şi apoi să explicăm ceasurile logice, ţinând cont de ceasurile vector, care sunt un mijloc de ordonare a evenimentelor fără a şti exact când au avut loc.

Absenţa timpului fizic global face dificilă găsirea stării programelor distribuite ce se execută. De obicei trebuie să ştim în ce stare este procesul *A* când procesul *B* este într-o stare cunoscută, dar noi nu ne putem baza pe ceasurile fizice pentru a şti ce e adevărat într-un acelaşi moment.

Multe aplicaţii soft depind de evidenţierea timpului şi a marcajelor. Un exemplu îl constituie comanda *make*, în înţeles de *marcaj (sau talie)*, care construieşte (formează) fişierele obiect şi executabilele dintr-un set de fişiere sursă. Când un program rulează, *make* compară marcajele fişierelor sursă *foo.c* cu marcajul corespunzător fişierului obiect *foo.o*. Fişierul sursă *foo.c* va fi recompilat doar dacă *make(foo.c) > make(foo.o)*. Acelaşi criteriu se aplică dacă *make(foo.c) < make(foo.o)* însă, *aHeader.h*, unul dintre *'header'* inclus în *foo.c* satisface *make(aHeader.h) > make(foo.o)*.

Fiecare sistem are propriul său ceas intern şi, cu toate că ceasurile pot funcţiona corect, în general nu se sincronizează. În timp, vor apărea mici devieri în diferite sisteme raportându-se timpi diferiţi. Astfel, dacă fişierele **.c, *.h* şi **.o* sunt distribuite pe mai multe maşini, comanda *make* nu va funcţiona întotdeauna corect.

---

[9] Coulouris, G., Dollimore, J., Kindberg, T., *Distributed Systems, Third Edition,* Addison-Wesley, Pearson Education Ltd., 2001

[10] Raynal, M., La communication et le temps dans les réseaux et les systèmes réparttis. Editions Eyrolles, 1991

Acest capitol introduce concepte şi algoritmi fundamentali referitori la monitorizarea sistemelor distribuite în timpul desfăşurării execuţiei şi la cronometrarea evenimentelor care au loc.

Timpul e o importantă şi interesantă problemă în sistemele distribuite, din mai multe motive. În primul rând timpul este entitatea pe care vrem să o măsurăm cu exactitate. Ca să ştim la ce oră un anumit eveniment a avut loc, la un anumit calculator, e necesară sincronizarea ceasului său cu o sursă externă de timp autorizată. De exemplu, o tranzacţie comercială electronică implică evenimente la calculatorul comerciantului şi al băncii. E important, pentru scopurile contabile, ca acele evenimente să fie marcate exact.

În al doilea rând, au fost dezvoltaţi algoritmi ce depind de sincronizarea ceasului în numeroase probleme de distribuţie (Liskov, 1993)[11]. Aceştia includ menţinerea consistenţei datelor distribuite, verificarea autenticităţii cererii trimise unui server şi eliminarea procesului de actualizare a datelor deja existente.

Einstein a demonstrat, în "Teoria relativităţii"[12] consecinţele care decurg din observaţia că viteza luminii este constantă pentru toţi observatorii, indiferent de viteza lor relativă. El a demonstrat, pornind de la această presupunere, pe lângă multe alte lucruri, că două evenimente care se presupun a fi simultane faţă de un punct de referinţă nu sunt neapărat simultane faţă de alte puncte de referinţă, care se mişcă relativ spre ele. De exemplu, un observator de pe pământ şi un observator care zboară în afara pământului într-o navă spaţială nu vor fi de acord cu intervalul de timp dintre evenimente, cu cât viteza lor relativă creşte.

Mai mult, ordinea relativă a celor două evenimente poate fi inversată din punctul de vedere a doi observatori diferiţi. Dar aceasta nu se poate întâmpla dacă un eveniment a cauzat apariţia celuilalt. În acest caz efectul fizic urmează cauza fizică pentru toţi observatorii, cu toate că timpul trecut dintre cauză şi efect poate varia.

Noţiunea de timp fizic este, de asemenea, o problemă în sistemele distribuite. Aceasta nu se datorează efectelor relativităţii speciale, care sunt neglijabile sau inexistente pentru calculatoarele obişnuite (doar dacă este vorba de un calculator al unei nave spaţiale!). Problema se bazează pe o limitare similară a abilităţii noastre de a marca evenimente, la noduri diferite, suficient de exact pentru a şti ordinea în care au avut loc evenimentele în orice pereche de evenimente sau dacă au avut loc simultan. Aici nu există timp absolut global la care să putem apela. Şi totuşi câteodată e nevoie să observăm sistemele distribuite şi să stabilim cu siguranţă dacă stările au avut loc în acelaşi timp. De exemplu, în sistemele orientate obiect trebuie să putem stabili dacă referinţele

[11] Barbara Liskov: Practical Uses of Synchronized Clocks in Distributed Systems. *Distributed Computing* 6(4): 211-219 (1993)

[12] Albert Einstein, Teoria relativităţii pe înţelesul tuturor; Ed. Humanitas, 2005

la un obiect anume mai există, dacă obiectul a devenit nefolosibil (caz în care putem elibera memoria). Pentru a stabili acestea, se cer observaţii ale stării proceselor (pentru a afla dacă conţin referinţe) şi observaţii ale canalelor de comunicare dintre procese, pentru cazul în care mesajele ce conţin referinţe sunt în trecere.

În prima jumătate a acestui capitol examinăm metode prin care ceasurile calculatoarelor pot fi aproximativ sincronizate, folosind transmiterea de mesaje. Continuăm cu introducerea ceasurilor logice, inclusiv ceasurile vector care sunt folosite pentru a defini o ordine a evenimentelor fără a ţine cont de timpul fizic la care au avut loc.

6.2. Ceasuri, evenimente şi stările procesului

Presupunem că ne este cunoscut un model al interacţiunii dintre procesele interioare ale unui sistem distribuit. Trebuie să modificăm acel model, astfel încât să ne ajute să înţelegem cum se caracterizează evoluţia sistemului în timpul execuţiilor, şi cum să marcăm evenimentele din timpul execuţiilor sistemului, care îl interesează pe utilizator. Începem prin a considera cum ordonăm şi cum marcăm evenimentele care au loc într-un singur proces.

Luăm un sistem distribuit care conţine o colecţie ξ de N procese p_i, $i=1,2,...,N$. Fiecare proces se execută pe un singur procesor, şi procesoarele nu partajează memoria. Fiecare proces p_i din ξ are o stare s_i care, în general, se modifică în timpul execuţie. Starea proceselor include valori ale tuturor variabilelor, mai puţin pe ale ei. Starea sa poate include valorile oricărui obiect din mediul local al sistemului de operare pe care-l afectează, cum ar fi fişierele. Presupunem că procesele nu pot comunica între ele în nici un fel decât prin trimiterea mesajelor prin reţea.

Când un proces oarecare p_i se execută, iau naştere o serie de ***acţiuni***, fiecare dintre ele e o operaţie de *trimitere/recepţionare* de mesaje, sau o operaţie care modifică (transformă) starea procesului p_i, - adică, schimbă una sau mai multe valori din s_i. De exemplu, dacă procesele din ξ sunt angajate într-o aplicaţie de comerţ electronic atunci una dintre acţiuni poate fi: "clientul trimite o cerere printr-un mesaj" sau "serverul comerciantului înregistrează tranzacţiile efectuate".

Definim un ***eveniment*** ca fiind apariţia unei singure acţiuni a cărei proces returnează la sfârşitul execuţiei – o acţiune de comunicare sau o acţiune de transformare a stării. Secvenţa de evenimente din singurul proces p_i, poate fi aşezată într-o singură ordine totală şi care o vom nota cu simbolul de relaţie $\rightarrow_i$ între evenimente. Fie e şi e' două evenimente din cadrul unui proces p_i. Avem relaţia $e \rightarrow_i e'$ dacă şi numai dacă evenimentul e are loc înaintea lui e' în cazul procesul p_i. Această ordine e bine definită, indiferent dacă procesul are mai

multe "thread-uri" (fire de execuţie), din moment ce am presupus că procesul se execută pe un singur procesor.

Acum putem defini *istoricul* procesului p_i, ca fiind o serie de evenimente care au loc în interiorul său, ordonate conform relaţiei $\rightarrow_i$:

$$istoric(p_i) = <e_i^0, e_i^1, e_i^2, \ldots>$$

6.2.1. Ceasurile

Măsurarea timpului e surprinzător de complexă. Am văzut cum ordonăm evenimentele unui proces, dar nu şi cum le marcăm – adică, să le atribuim o dată şi o oră. Fiecare calculator are propriul său ceas fizic. Aceste ceasuri sunt aparate electronice care contorizează oscilaţiile ce au loc într-un cristal la o frecvenţă definită, şi care împart această numărătoare, dar şi stochează rezultatul într-un registru. Ceasurile pot fi programate să genereze întreruperi la un interval regulat de timp.

Sistemul de operare citeşte o valoare a ceasului hardware $H_i(t)$ şi adaugă o valoare β pentru a stabili ceasul software $C_i(t) = \alpha H_i(t) + \beta$ care aproximează timpul fizic real t al procesului p_i. Cu alte cuvinte, când timpul real e o structură absolută de referinţe atunci este t, iar $C_i(t)$ este citirea ceasului software. De exemplu, $C_i(t)$ poate fi valoarea de nanosecunde reprezentată pe *64* biţi, care a trecut la timpul t faţă de timpul de referinţă ales. În general, ceasul nu e perfect exact şi, de aceea, $C_i(t)$ va diferi de t. Dacă ceasul C_i se manifestă suficient de bine (trebuie să examinăm pe scurt noţiunea de ceas exact), putem folosi valoarea sa ca să marcăm orice eveniment al procesului p_i. De reţinut că, evenimentelor succesive le vor corespunde marcaje diferite, dacă rezoluţia ceasului – intervalul de timp dintre modificările valorii ceasului – e mai mică decât intervalul de timp dintre evenimentele succesive. Intervalul de timp la care au loc evenimentele depinde de factori precum lungimea ciclului de instrucţiuni a procesorului.

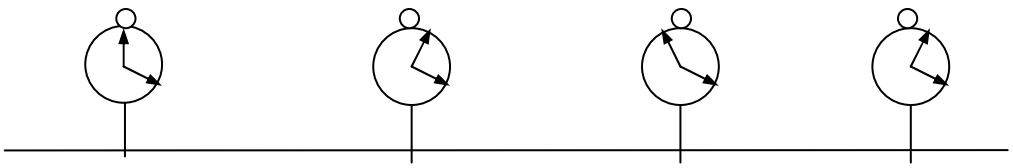


Fig 6.1. Neidenticitatea dintre ceasurile calculatoarelor într-un sistem distribuit

6.2.2. Ceasuri neidentice şi ceasuri curente

Ceasurile calculatoarelor, ca oricare alte ceasuri, tind să nu fie într-o perfectă armonie (fig. 6.1). Diferenţa instantanee dintre citirile oricăror două ceasuri se numeşte *neidenticitate*. De asemenea, ceasurile pe bază de cristal folosite de calculatoare sunt, ca oricare alte ceasuri, subiecte de discuţie pentru *ceasurile curente*, adică contorizează timpul la intervale diferite, şi deci diferă. Este posibil ca aceeaşi frecvenţă a unui ceas să varieze în funcţie de temperatură. Studiul existent încearcă să compenseze această variaţie, dar nu se poate elimina. Diferenţa dată de intervalul de oscilaţie dintre două ceasuri poate fi extrem de mică, însă diferenţa acumulată după mai multe oscilaţii duce la o diferenţă observabilă înregistrată în registre de către două ceasuri, indiferent cât de exact au fost iniţializate cu aceeaşi valoare. Intervalul *ceasurilor neidentice* e schimbarea adresei (diferită de cea citită) dintre ceas şi referinţa perfect nominală a ceasului per unitatea de timp măsurată de referinţa ceasului. Pentru ceasurile pe bază de cristal-guartz e vorba de 10^{-6} secunde/secundă – cu o diferenţă de 1 secundă la fiecare *1.000.000* secunde, sau *11.6* zile. Intervalul neidentic al ceasurilor de quartz de mare precizie este de 10^{-7} sau 10^{-8}.

6.2.3. Coordonarea timpului universal

Ceasurile calculatoarelor pot fi sincronizate cu o sursă externă de mare precizie. Cele mai exacte ceasuri fizice folosesc oscilatoare atomice, a căror interval neidentic este o parte din 10^{13}. Ieşirea acestor ceasuri atomice este folosită ca cea standard pentru timpul real trecut, cunoscut ca *Timpul Atomic International*.

Timpul Solar e măsurat privind Soarele.

"1 secundă solară" = $\frac{1}{24 \times 60 \times 60} \times 1$ zi solară (de la prânz la prânz)

deoarece zilele sunt de diferite lungimi (datorită intervalelor de timp) se face media dintre mai multe zile şi astfel apare 'secunda solară'.

Secundele şi anii şi alte unităţi de măsurare a timpului pe care le folosim au la bază timpul astronomic. Iniţial au fost definite în funcţie de rotaţia Pământului faţă de axă şi de rotaţia faţă de Soare. Oricum perioada de rotaţie a Pământului faţă de axă devine tot mai mare, în primul rând din cauza conflictului de flux; efectelor atmosferice şi curenţilor transmişi în nucleul Pământului, care pe termen scurt duc la creşteri şi descreşteri ale perioadei. Deci timpul astronomic şi cel atomic tind să nu mai ţină pasul.

Coordonarea Timpului Universal – prescurtat ca UTC (din echivalentul său în franceză) – e un nivel internaţional pentru memorarea timpului. Se

bazează pe timpul atomic, dar aşa numita 'secundă săltăreaţă' e inserată – sau, mai rar, ştearsă – ocazional pentru a ţine pasul cu timpul astronomic. Semnalele UTC sunt sincronizate şi emise regular din staţiile radio şi de pe sateliţii care acoperă cea mai mare parte a lumii. De exemplu, în SUA, postul de radio *WWV* emite semnale de timp pe mai multe unde scurte. Sursele de pe sateliţi includ *Sistemul Global de Poziţie (GPS – **G**lobal **P**ositioning **S**ystem).*

Receptorii sunt comerciali disponibili. Comparat cu UTC-ul 'perfect', semnalele recepţionate de staţiile radio au o precizie de *0.1-10* milisecunde, depinzând de staţia folosită. Semnalele recepţionate prin *GPS* au o precizie de o microsecundă. Calculatoarele cu receptori ataşaţi îşi pot sincroniza propriul lor ceas cu aceste semnale de timp. De asemenea, calculatoarele pot recepţiona timpul cu o precizie de câteva milisecunde pe o linie telefonică.

6.3. Sincronizarea ceasurilor fizice

Pentru a şti la ce oră din zi au loc evenimentele proceselor într-un sistem distribuit ξ - de exemplu, pentru evidenţa contabilă – e necesar a sincroniza ceasurile proceselor p_i cu o sursă autoritară de timp. Aceasta e *sincronizarea externă*. Dacă ceasurile C_i sunt sincronizate unul cu celălalt la un grad de precizie cunoscut, atunci putem măsura intervalul de timp dintre două evenimente, ce au loc la calculatoare diferite apelând la propriul lor ceas local – chiar dacă ele nu se sincronizează cu o sursă externă de timp. Aceasta e *sincronizarea internă.* În continuare, vom defini aceste două moduri de sincronizare pe un interval de timp real *I*:

- *Sincronizarea externă*: pentru o sincronizare delimitată de $D > 0$, când avem o sursă *S* de timp UTC, $|S(t) - C_i(t)| < D$ unde $i = 1,2, \ldots N$ şi pentru toţi timpii *t* reali din *I*. Adică ceasurile C_i sunt *exacte* în limita lui *D*.
- *Sincronizarea internă*: pentru o sincronizare delimitată de $D > 0$, $|C_i(t) - C_j(t)| < D$ unde $i, j = 1, 2, \ldots N$ şi pentru toţi timpii *t* reali din *I*. Adică ceasurile C_i sunt *acceptate* în limita lui *D*.

Ceasurile sincronizate intern nu sunt neapărat sincronizate extern, din moment ce ele devin colectate dintr-o sursă externă de timp. Oricum, din definiţie reiese că dacă un sistem ξ e sincronizat extern în limita lui *D*, atunci acelaşi sistem va fi sincronizat intern în limita lui *2D*.

Putem preciza noţiunea de *corectitudine* pentru ceasuri. E normal să definim că ceasul hardware *H* e corect dacă devierea sa se încadrează în limita cunoscută $\rho > 0$ (valoare care ar putea fi 10^{-6} secunde/secundă). Asta înseamnă că pentru intervalul dintre timpul *t* şi *t'* ($t' > t$) avem:

$$(1 - \rho)(t' - t) \le H(t') - H(t) \le (1 + \rho)(t' - t)$$

Această condiţie interzice salturile în valorile ceasurilor harware (în timpul operaţiunii normale). Câteodată noi recomandăm ceasurilor software să se supună condiţiei de mai sus. Însă condiţia de *monotonie* va avea de suferit. Monotonia e condiţia prin care un ceas avansează:

$$t' > t \Rightarrow C(t') > C(t)$$

De exemplu, facilitatea *make* a UNIX-ului e o opţiune care compilează doar acele fişiere care au fost modificate după ultima compilare. Datele modificate ce corespund fiecărei perechi de fişiere sursă şi fişiere obiect sunt comparate pentru a determina această condiţie. Dacă un calculator a cărui ceas merge prea repede îşi setează ceasul înapoi după compilarea fişierului sursă, dar înainte ca fişierul să fi fost modificat, atunci fişierul sursă poate apărea ca fiind modificat înainte de compilare. Din greşeală, *make* nu va recompila fişierele sursă.

Putem demonstra monotonia în ciuda faptului că ceasul merge prea repede. Trebuie doar să modifice intervalul la care au loc actualizările cu timpul dat de aplicaţie. Aceasta poate fi realizată în programe fără schimbarea intervalului la care ceasul hardware ticăie – numim asta $C_i(t) = \alpha H_i(t) + \beta$, α, β aleatoare.

În continuare vom descrie algoritmi pentru sincronizarea externă şi pentru sincronizarea internă.

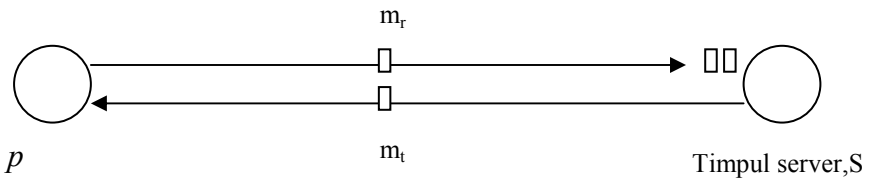


Fig. 6.2. Sincronizarea ceasului folosind timpul server

6.3.1. Sincronizarea într-un sistem distribuit sincron

Începem cu cazul cel mai simplu posibil de sincronizare internă între două procese ale unui sistem distribuit sincron. Într-un sistem sincron limitele pentru intervalele curente ale ceasurilor sunt cunoscute, de asemenea, întârzierea maximă în transmiterea mesajului şi timpul de execuţie pentru fiecare pas al procesului.

Un proces trimite timpul t al ceasului local la alţii printr-un mesaj m. In principiu procesul receptor îşi poate seta propriul său ceas după timpul $t+T_{trans}$, unde T_{trans} este timpul necesar transmiterii mesajului m între cele doua procese. Atunci cele două ceasuri vor fi în concordanţă (din moment ce scopul e sincronizarea internă, nu contează dacă trimiterea ceasului procesului e exactă).

Din păcate, T_{trans} este discutabil şi necunoscut. În general concurează şi alte procese din resurse cu procesele care se sincronizează la nodurile respective, apoi şi alte mesaje concurează cu mesajul *m* din reţea. Întotdeauna există un timp minim *min* de transmisie care este obţinut dacă nu se execută alte procese şi nu există trafic în reţea; *min* poate fi măsurat sau estimat.

Într-un sistem sincron conform definiţiei, există şi un timp *max* pentru transmiterea oricărui mesaj. Notăm nesiguranţa timpului de transmitere cu *u,* deci $u = (max-min)$. Dacă receptorii îşi fixează ceasul ca fiind $t+min$ atunci ceasul neidentic poate fi cel mult *u*, din moment ce mesajul a avut timpul *max* pentru a ajunge. Analog, dacă receptorii îşi fixează ceasul ca fiind $t+max$, atunci diferenţa poate fi cel mult *u*. Dacă totuşi ceasul este setat la jumătate, $t+(max+\min)/2$ atunci diferenţa e cel mult *u/2*. În general, într-un sistem sincron saltul optim care poate fi atins de un ceas neidentic când se sincronizează *N* ceasuri este $u(1-1/N)$.

Majoritatea sistemelor distribuite întâlnite în practică sunt asincronice: factorii care conduc spre întârzierile mesajului nu sunt mărginite în efect, şi nu există un salt maxim *max* în întârzierile transmiterii mesajului. Acesta e specific Internetului. Pentru un sistem asincron, putem spune doar că $T_{trans} = min+ x,$ unde $x \geq 0$. Valoarea lui *x* nu e cunoscută pentru cazul particular, deşi o distribuţie de valori poate fi măsurată pentru o situaţie particulară.

6.3.2. Metoda lui Cristian pentru sincronizarea ceasurilor

Cristian(1989)[13] a sugerat folosirea unui server de timp, conectat la un aparat care recepţionează semnale de la o sursa UTC, pentru a sincroniza calculatoarele din exterior. La cerere, procesul *p* al serverului setează timpul în conformitate cu ceasul său ca în figura *6.2*. Cristian a observat că, din moment ce nu există o limită superioară a întârzierilor transmiterii mesajului într-un sistem asincron, timpii necesarii pentru schimbul de mesaje dintre perechile de procese sunt adesea rezonabili de scurţi – o mică fracţiune de secundă. El descrie un algoritm bazat pe o metodă *probabilistică*: metoda realizează sincronizarea doar dacă timpii necesari observaţi dintre client şi server sunt suficient de puţini în comparaţie cu precizia cerută.

Un proces *p* cere timpul într-un mesaj m_r şi primeşte valoarea timpului *t* într-un alt mesaj m_t (*t* este inserat în m_t la ultimul punct posibil înaintea transmisiei de la procesul *S* al calculatorul). Procesul *p* înregistrează timpul necesar total $T_{turretur}$ trimiterii cererii m_r şi primeşte răspunsul m_t. Acest timp se

[13] Flaviu CRISTIAN (1951-1999) a început studiile universitare (1970-1971), la Universitatea Babeş-Bolyai din Cluj-Napoca, Facultatea de Matematică şi Informatică pe care le-a continuat, apoi, la Universitatea din Grenoble, Franţa. A lucrat o perioadă (1979-1991) ca cercetător în Anglia, iar din 1991 se dedică carierei de profesor la University of California, San Diego, SUA, unde s-a remarcat ca un valoros specialist în informatică.

poate măsura cu o precizie rezonabilă dacă intervalul ceasurilor neidentice e mic. De exemplu, timpul necesar ar trebui să fie de ordinul a *1-10* milisecunde într-o reţea locală. O simplă estimare a timpului la care procesul p ar trebui să-şi seteze ceasul este $t+T_{turretur}/2$, care presupune că timpul trecut este egal împărţit înainte şi după ce S îl pune pe t în m_t. Aceasta, e în mod normal, o presupunere rezonabilă, mai puţin dacă cele două mesaje sunt transmise spre reţele diferite. Dacă valoarea minimă a timpului de transmitere *min* este cunoscută sau poate fi estimată, atunci putem determina precizia acestui rezultat aşa cum urmează el.

Cel mai apropiat punct în care S ar fi putut pune timpul într-un mesaj m_t a fost *min* după ce p a fost trimis la m_r. Cel mai îndepărtat punct în care ar fi putut face asta a fost *min* înainte ca m_t să ajungă la p. Timpul ceasului lui S când mesajul de răspuns ajunge e cuprins între $[t+min,\ t+T_{turretur}-min]$ lungimea acestui interval e $T_{turretur} - 2\ min$, deci precizia e de $\pm (T_{turretur}/2-min)$.

Observaţii relative la algoritmul lui Cristian. După cum s-a descris, metoda lui Cristian suferă din cauza problemei asociate cu implementarea tuturor serviciilor de către un singur server. astfel că singurul timp al serverului poate da eroare şi o sincronizare imposibilă temporal. De exemplu, un client poate multiplica cererile sale tuturor serverelor şi folosi doar primul răspuns obţinut.

De reţinut că, un timp eronat al serverului care a răspuns cu o valoare neadevărată a timpului, sau un timp intrus al serverului care a răspuns în mod deliberat cu un timp incorect poate împiedica un dezastrul în sistemul calculatorului. Rezolvarea unor astfel de probleme au constituit un scop al muncii prestate de Cristian[1989, CRI91[14]], care presupune că sursele semnalelor externe de timp sunt verificate. Cristian şi Fetzer[15] au descris o familie de protocoale pentru sincronizarea ceasurilor interne, fiecare din ele tolerând anumite greşeli. La început Srikanth şi Toueg[16] au descris un algoritm care e optim cu privire la exactitatea ceasurilor sincronizate, în timp ce tolerează câteva greşeli. Dolev *et al.[1986]*[17] a arătat că dacă f e numărul de ceasuri eronate dintr-un total de N, atunci trebuie să avem $N>3f$ ceasuri corecte, ceasuri care sunt în stare de funcţionare.

---

[14] Cristian, F., Understanding Fault-Tolerant Distributed Systems, Communications of the ACM, Vol. 34, No 2, 1991.

[15] F. Cristian and C. Fetzer. Fault-tolerant internal clock synchronization. In *Proc. of the Thirteenth Symposium on Reliable Distributed Systems*, pages 22--31, Dana Point, Ca., Oct 1994.

[16] .K. Srikanth and Sam Toueg, Optimal clock synchronization**,** *Journal of the ACM*, 1987

[17] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, William E. Weihl, Reaching approximate agreement in the presence of faults. J. ACM 33(3): 499-516 (1986)

6.3.4. Protocolul timpului în reţea

Metoda lui Cristian este folosită în principal în reţelele locale (intranet). Protocolul timpului în reţea (NTP – Network Time Protocol) [Mills, 1995][18] defineşte o arhitectură pentru un serviciu de timp şi un protocol pentru distribuţia de informaţii referitoare la timp în Internet.

Pretenţiile de design ale NTP-ului sunt următoarele:

- *De a furniza un serviciu ce permite clienţilor din Internet să se sincronizeze exact conform* UTC: în ciuda numărului mare şi variabil de întârzieri a mesajelor întâlnite în comunicarea în Internet. NTP utilizează tehnici statistice pentru filtrarea datelor referitoare la timp şi face deosebire între calitatea datelor referitoare la timp din diferite servere.

- *De a furniza un serviciu sigur care poate supravieţui unei lungi perioade de pierdere a conectivităţii*: Există servere redundante şi căi redundante între servere. Serverele se pot reconfigura astfel încât să continue furnizarea de servicii dacă unul dintre ele nu poate fi localizat.

- *De a permite clienţilor să se resincronizeze suficient de frecvent pentru a compensa ratele de înaintare întâlnite la majoritatea calculatoarelor.* Serviciul este conceput pentru a fi folosit de un număr mare de clienţi şi servere.

- *De a furniza protecţie împotriva interferenţelor cu serviciile de timp, indiferent dacă sunt rău intenţionate sau accidentale:* Serverul de timp foloseşte tehnici de autentificare pentru a verifica originea datelor de timp din surse sigure. De asemenea, validează adresele returnate ale mesajelor trimise la el.

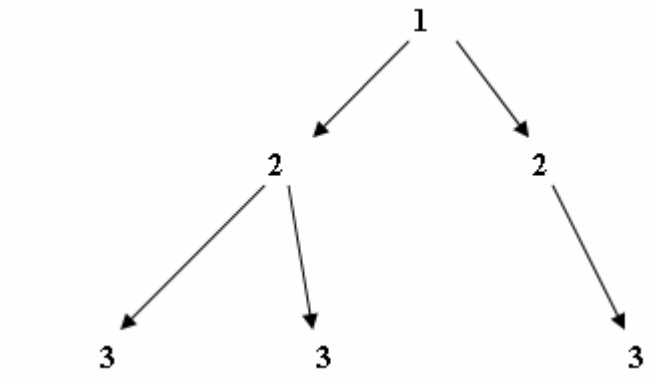


Fig. 6.3 Un exemplu de sincronizare pe nivele într-o implementare NTP (Săgeţile marchează controlul sincronizării iar numerele nivelul)

[18] Mills, D.L. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Trans. Networks 3, 3* (June 1995), 245-254

Serviciul NTP e furnizat de o reţea de servere situate în Internet. *Serverele primare* sunt conectate direct la o sursă de timp precum ceasul radio care primeşte UTC; *serverele secundare* sunt sincronizate, în cele din urmă, cu serverele primare. Serverele sunt conectate într-o ierarhie logică numită *sincronizare pe nivele* (a se vedea figura. 6.3), a căror nivele se numesc uneori şi *straturi*. Serverele primare ocupă stratul *1*: ele sunt la rădăcină. Serverele de pe stratul *2* sunt servere secundare care sunt sincronizate direct cu serverele primare; serverele de pe stratul *3* sunt sincronizate cu serverele de pe stratul 2, şi aşa mai departe. Serverele cele mai de jos (noduri terminale) se execută pe staţiile de lucru ale utilizatorilor.

Ceasurile ce aparţin serverelor de pe stratul cu număr mare sunt obligatoriu mai puţin exacte decât acelea de pe stratul cu număr mic, pentru că erorile sunt introduse la fiecare nivel de sincronizare. NTP ţine cont de întârzierea totală a mesajului de la rădăcină, luând în considerare calitatea timpului de păstrare a datelor trecute de către un server anume.

Sincronizarea pe nivele poate reconfigura serverele astfel încât ele să devină de negăsit sau să aibă erori. Dacă, de exemplu, o sursă a serverului primar UTC dă eroare, atunci poate deveni un server secundar pe stratul *2*. Dacă o sursă normală a unui server secundar sincronizat dă eroare sau devine de negăsit, atunci se poate sincroniza cu alt server.

Serverele NTP se sincronizează unul cu celalalt, în unul sau mai multe moduri: *multiplicare, apelează-procedura* şi *simetric.* Modul *multiplicare* e folosit pentru reţelele LAN cu o viteză mare. Unul sau mai multe servere periodic multiplica timpul de rulare a serverelor a celorlalte calculatoare conectate prin LAN, care setează propriul lor ceas cu o întârziere mică. Acest mod poate obţine o precizie relativ mică, însă pentru multe scopuri e considerată suficientă.

Modul *apelează-pocedura* (folosind TCP-ul) e similar cu operaţia din algoritmul lui Cristian, descris mai sus. În acest mod, un server acceptă cereri de la alte calculatoare, care procesează răspunzând cu citirea ceasului curent. Acest mod e potrivit când se cer precizii mai mari decât se pot obţine prin *modul de multiplicare* - sau când *modul multiplicare* nu e suportat de partea hard a calculatorului. De exemplu, fişierele de pe acelaşi server sau din reţeaua locală învecinată LAN, care trebuie să cronometreze informaţia exactă pentru fişierele accesate, pot contacta un server local prin *modul apelează-procedura*.

În sfârşit, *modul simetric* este folosit de servere care stochează informaţia de timp din LAN şi de nivelele mari (straturile mici) ale sincronizării pe nivele, unde se ating cele mai mari precizii. O pereche de servere ce operează în modul simetric schimbă mesaje suportând informaţia de timp. Cronometrarea datei e reţinută ca o parte a unei asocieri între servere, care e păstrată pentru a îmbunătăţi precizia sincronizărilor după un timp.

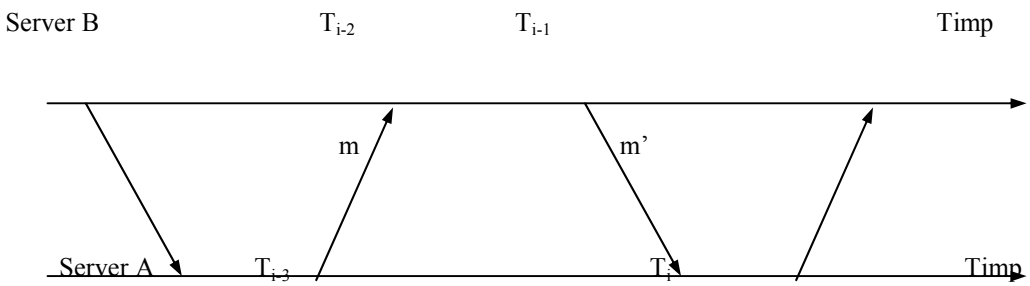


Fig. 6.4 Mesajele schimbate într-o pereche de port-uri NTP

În *modul apelează-proedura* şi *modul simetric* procesele schimbă perechi de mesaje. Fiecare mesaj suportă marcarea evenimentelor recente: timpii locali când mesajul NTP anterior din pereche a fost trimis şi primit şi timpul local când mesajul curent a fost transmis. In corpul mesajului NTP se notează timpul local când primeşte mesajul. Cei patru timpi T_{i-3}, T_{i-2}, T_{i-1}, T_i sunt arătaţi în figura *6.4* pentru mesajele *m* şi *m'* trimise între serverele *A* şi *B*. De reţinut că în modul simetric, spre deosebire de algoritmul lui Cristian descris mai sus, poate exista o întârziere ne-neglijabilă între sosirea unui mesaj şi expedierea următorului. De asemenea, mesajele se pot pierde, însă cei trei timpi curenţi ai fiecărui mesaj sunt valizi.

Pentru fiecare pereche de mesaje trimise între două servere de NTP se calculează o *adresă* o_i, care e o estimare a actualei adrese dintre două ceasuri şi o *întârziere* d_i, care e timpul total de transmisie pentru două mesaje. Dacă adresa corectă a ceasului serverului *B* relativ la *A* este *o*, şi dacă timpii actuali de transmitere a mesajelor *m* şi *m'* sunt *t* respectiv *t'*, atunci avem:

$$T_{i-2} = T_{i-3} + t + o \quad \text{şi} \quad T_i = T_{i-1} + t' - o$$

Rezultă că:

$$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$$

De asemenea:

$$o = o_i + (t' - t) / 2 \text{ unde } o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i) / 2$$

Folosind faptul că $t, t' \geq 0$ se poate arăta că $o_i = d_i / 2 \leq o \leq o_i + d_i / 2$. Astfel o_i e o estimare a adresei şi d_i e o măsurare a preciziei acestei estimări.

Serverele NTP cer un algoritm de filtrare a informaţiei pentru perechile succesive $< o_i , d_i >$, care estimează adresa *o* şi calculează calitatea acestei estimări ca o cantitate statistică numită *dispersia filtru*. O risipire filtru relativ mare reprezintă o informaţie relativ neserioasă. Cele mai recente opt perechi $< o_i , d_i >$ sunt reţinute. Ca şi cu algoritmul lui Cristian, valoarea lui o_j ce corespunde valorii minime d_j e aleasă să estimeze adresa *o*.

Valoarea adresei derivate din comunicarea cu o singură sursă, oricum, nu e folosită neapărat de ea pentru controlul ceasului local. In general, un server NTP se angajează în schimbul de mesaje cu câteva dintre perechile sale. NTP

aplica un algoritm de selecţie a perechilor. Acesta examinează valorile obţinute din schimburile cu fiecare dintre câteva perechi, căutând valori relativ neserioase. Ieşirea din acest algoritm poate face ca un server să schimbe perechile care sunt mai întâi folosite pentru sincronizări.

Perechile de pe stratul cu număr mai mic sunt mai favorizate decât cele de pe stratul cu număr mai mare, pentru că sunt 'mai aproape' de sursele primare de timp. De asemenea, acele perechi cu o mai mică *sincronizare dispersivă* sunt relativ favorizate. Aceasta e suma dispersiilor filtrate măsurată între server şi rădăcina sincronizării pe nivele.

6.4. Timpul logic şi ceasuri logice

6.4.1. Noţiuni generale

Un sistem distribuit constă din procese secvenţiale, care comunică între ele cu ajutorul mesajelor. Ştim că durata transmisiei unui mesaj este imprevizibilă, şi că procesele nu au acces la un ceas global, sau la ceasuri locale perfect sincronizate. Din cele afirmate reiese că nici unul dintre procese nu are o viziune asupra stării globale. Aceasta este cauza multor probleme şi fenomene care caracterizează sistemele distribuite şi care nu au o soluţie trivială.

Comportamentul fiecărui proces din sistem este guvernat de un algoritm, care determină secvenţa acţiunilor şi reacţiilor locale ale procesului. Execuţia unei aplicaţii sau a unui program distribuit pe un astfel de sistem produce un ansamblu de evenimente. Mulţimea evenimentelor o notăm cu *E*. Execuţiile acţiunilor sunt ***evenimente atomice.*** Un eveniment aparţine unei din cele trei tipuri de evenimente:

- emisia unui mesaj;
- recepţia unui mesaj;
- evenimente interne (mesaje interne).

Un astfel de proces partajat abstract pote fi înţeles cu ajutorul diagramei de timp (fig. 6.5). Timpul este reprezentat, ca de obicei, pe orizontală de la stânga la dreapta. Mesajele sunt reprezentate ca săgeţi de la stânga la dreapta, iar evenimentele sunt reprezentate prin puncte.

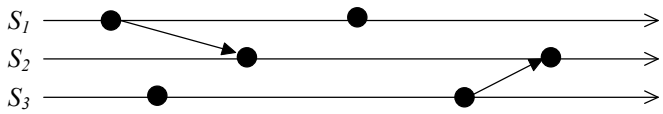


Fig. 6.5. Diagrama de timp

Pentru a fi executat, un program are nevoie de spaţiu de memorie şi timp procesor. Dar timpul nu poate fi restricţionat ca resursă, din acest punct de vedere. Timpul stabileşte dependenţe cauzale între evenimentele produse de execuţia unui program. Într-un sistem distribuit cu n staţii conectate prin canale de comunicaţie, avem următoarele ipoteze:

- evenimentele de pe fiecare staţie sunt total ordonate;
- dacă se consideră un mesaj oarecare *m*, evenimentul: emisia lui *m* precede evenimentul: recepţionarea lui *m*.

Închiderea tranzitivă a acestei precedenţe a evenimentelor defineşte o relaţie de dependenţă cauzală "$\rightarrow$". Această relaţie este o relaţie de ordine parţială.

Definiţia 1. *Pe mulţimea E a evenimentelor introducem o relaţie tranzitivă, numită relaţie de cauzalitate şi notată prin: $\rightarrow$, astfel încât, a, b $\in$ E, a $\rightarrow$ b dacă:*

- *a şi b sunt cauzate de acelaşi proces şi a este predesorul imediat al lui b*
- *b recepţionează un mesaj trimis de a*
- *există un eveniment c astfel încât a $\rightarrow$ c şi c $\rightarrow$ b*

Relaţia de cauzalitate este o relaţie de ordonare parţială. Două elemente, care nu sunt în această relaţie, ***sunt evenimente independente***.

Definiţia 2. *Două elemente a şi b din E sunt în relaţia || (independenţă), dacă $\neg$(a$\rightarrow$ b) şi $\neg$(b $\rightarrow$ a). Deci, oricare ar fi două evenimente a şi b avem: a $\rightarrow$ b sau b$\rightarrow$a sau a || b.*

Pe diagrama de timp, două evenimente sunt în relaţia $\rightarrow$ dacă există drum (săgeţi) între cele două puncte care simbolizează evenimentele respective.

Fie două evenimente, *a* şi *b*, pentru care avea *a* $\rightarrow$ *b*. Mulţimea tuturor evenimentelor x pentru care, pentru un eveniment b dat, avem $x\rightarrow b$, este numită *conul* de cauzalitate al lui *b* (*cone (b),* fig.6.6). Două evenimente *x* şi *y* pentru care nu avem nici $x \rightarrow y$ şi nici $y \rightarrow x$ spunem că sunt independente sau concurente, cu notaţia *x* || *y*.

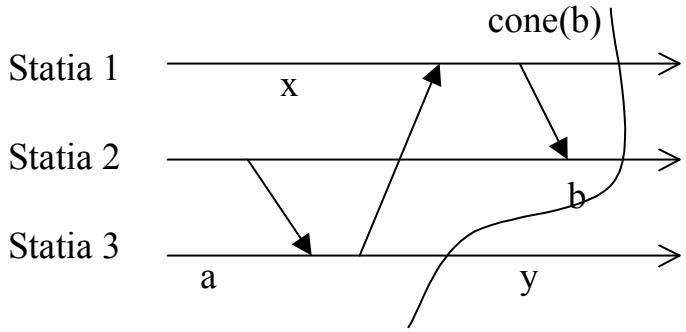


Fig. 6.6. Relaţii între evenimente

În continuare, vom prezenta trei mecanisme care permit asocierea dintre o dată (orară) şi un eveniment. Aceste date trebuie să se bazeze pe timpul logic global pentru a putea compara evenimente produse de staţii diferite, şi trebuie să fie consistente (trebuie respectată monotonia): dacă $a \rightarrow b$, atunci data asociată lui *b* trebuie să fie, în acord cu respectarea timpului logic global, după data asociată evenimentului *a*.

Pentru a asigura monotonia discutată, avem nevoie de nişte structuri de date şi protocoale de manipulare a acestora.

Structuri de date de reprezentare a timpului logic

Fiecare staţie este dotată cu o variabilă locală care îi permite:

- pe de o parte, să-şi măsoare propriul progres; acest lucru este realizat cu ajutorul ceasului logic local (actualizat de regula R1);
- pe de altă parte, să aibă o bună reprezentare a timpului logic global (actualizată de regula R2); această reprezentare permite ştampilarea evenimentelor, fiind de fapt un "view" local asupra timpului global.

Protocolul

Acesta asigură faptul că timpul logic local şi imaginea locală asupra timpului global pe fiecare staţie sunt gestionate consistent cu relaţia de cauzalitate "$\rightarrow$".

Acest lucru este realizat cu ajutorul următoarelor **reguli**:

R1: Înainte de producerea unui eveniment (trimitere, receptionare sau intern), o staţie trebuie să-şi incrementeze ceasul logic local (deoarece progresează).

R2: Pentru ca data de recepţionare a unui eveniment să fie după data evenimentului de trimitere corespunzător, fiecare mesaj *m* împachetează în el şi valoarea timpului logic global văzut de transmiţător în momentul transmiterii

mesajului. Acest lucru permite celui care recepţionează mesajul să-şi actualizeze "părerea" despre timpul logic global. După aceea, execută R1 şi poate ştampila evenimentul recepţionat.

Vom prezenta trei mecanisme de ştampilare a evenimentelor, pentru fiecare făcând o prezentare a modului cum este respectată monotonia (implementarea regulilor R1 şi R2), şi câteva proprietăţi asociate.

6.4.2. Timpul liniar

Mecanismul de ştampilare

Domeniul timpului este o mulţime de întregi. Fiecare staţie S_i are asociată o variabilă h_i care păstrează valorile crescătoare. Timpul logic local al staţiei S_i şi viziunea locală asupra timpului logic global sunt mixate şi sunt reprezentate de o singură variabilă, notată cu h_i.

Regulile R1 şi R2, care definesc consistenţa protocolului, sunt următoarele:

R1: înainte de producerea unui eveniment (trimitere, recepţionare sau intern) se execută:

$h_i := h_i + d \qquad (d > 0)$

la fiecare execuţie a regulii, d poate să aibă o altă valoare

R2: când primeşte un mesaj ştampilat (m, h), staţia S_i execută:

$h_i := max\ (h_i, h)$

după care execută regula R1 înainte de a trimite mesajul m.

Proprietăţi

Se poate construi o relaţie de ordine totală pe o mulţime de evenimente, numită "t-before", consistentă cu relaţia de cauzalitate "$\rightarrow$". Ştampila de timp a unui eveniment este compusă din data şi identitatea staţiei care l-a produs. Deci, dacă luăm în considerare două evenimente x şi y, ştampilate respectiv cu (h, i) şi (k, j), ordinea totală este definită de:

$$x \text{ t-before } y \Leftrightarrow (h < k \text{ or } (h = k \text{ and } i < j))$$

Relaţia de ordine totală astfel definită asigură proprietăţile de "liveness" ale algoritmilor distribuiţi.

Dacă avem în vedere faptul că valoarea de incrementare d este întotdeauna 1, avem următoarea proprietate:

Fie *e* un eveniment ştampilat cu *h*. Atunci, *h-1* reprezintă durata logică minimă, măsurată în unităţi de evenimente, necesară înaintea producerii evenimentului *e*; se numeşte înălţimea conului de cauzalitate asociată evenimentului *e*, mărimea (*height (e)*). Cu alte cuvinte, *h-1* evenimente s-au produs secvenţial înaintea evenimentului *e*.

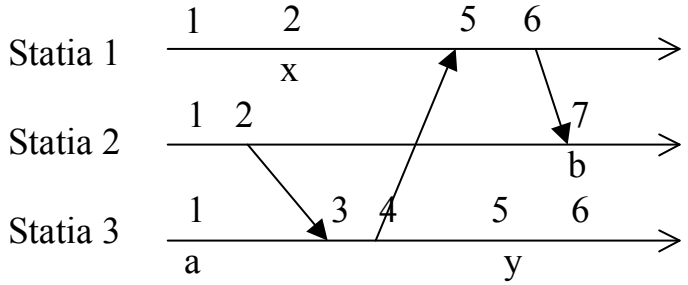


Fig. 6.7. Progresul de timp liniar

6.4.3. Timpul vectorial

Ceasuri vectoriale

În cadrul acestui mecanism, timpul logic global este reprezentat de un vector n-dimensional. Fiecare staţie S_i este dotată cu un astfel de vector, notat cu $vt_i[1..n]$.

Ideea încapsulării unui astfel de vector pe staţia S_i este următoarea:

- vt_i *[i]* descrie progresul timpului logic al staţiei S_i, considerată singura egală cu timpul logic local S_i. Variabila reţine valorile crescătoare generate local (o variabilă locală poate fi incrementată de regula R1)
- $vt_i[j]$ reprezintă cunoştinţele staţiei S_i despre progresul timpului local al staţiei S_j. Este o imagine locală a valorii $vt_j[j]$, şi este actualizată de regula R2.
- Întregul vector vt_i constituie imaginea locală a staţiei S_i asupra timpului logic global folosit la ştampilarea evenimentelor.

Regulile R1 şi R2 sunt următoarele:

R1: înainte de producerea unui eveniment se execută:

$vt_i[i] := vt_i[i] + d \quad (d > 0)$

R2: fiecare mesaj m împachetează vectorul de ceas *vh* al staţiei care l-a trimis. La recepţionarea unui astfel de mesaj *(m, vh)*, staţia S_i îşi actualizează cunoştinţele asupra progresului timpului local:

$1 \le k \le n: vt_i[k] := max(vt_i[k], vh[k]),$

după care execută regula R1.

Data asociată unui eveniment este valoarea vectorului de ceas *vh* al staţiei care îl trimite în momentul producerii evenimentului. În figura 6.8 este prezentat un exemplu al progresului vectorilor de ceas, cu o valoare a incrementului $d = 1$.

Proprietăţi

Dimensiunea vectorilor de ceas nu poate fi mai mică decât n. Să definim următoarele relaţii între vectorii de ceas:

$vh \le vk \Leftrightarrow$ *oricare ar fi* $x : vh[x] \le vk[x]$
$vh < vk \Leftrightarrow vh \le vk$ *şi există* $x : vh[x] < vk[x]$
$vh \| vk \Leftrightarrow$ *not* $(vh < vk)$ *and not* $(vk < vh)$

Dacă luăm în considerare relaţia de ordine parţială $\rightarrow$ pe o multime de evenimente produse de execuţia proceselor într-un sistem distribuit, ştampilate cu vectori de ceas, avem următoarea proprietate: Fie două evenimente *x* şi *y* ştampilate cu *vh*, respectiv *vk*. Atunci:

$x \rightarrow y \Leftrightarrow vh < vk$
$x \| y \Leftrightarrow vh \| vk$

Cu alte cuvinte, avem un izomorfism între o mulţime de evenimente parţial ordonate, produse de calculele dintr-un sistem distribuit, şi ştampilele lor de timp. Dacă considerăm că staţiile sunt numerotate, testul de independenţă poate fi simplificat. Deci, dacă *x* şi *y* sunt ştampilate cu *(vh, i)*, respectiv *(vk, j)*, avem:

$x \rightarrow y \Leftrightarrow vh[i] < vk[i]$
$x \| y \Leftrightarrow vh[i] > vk[i]$ *and* $vh[j] < vk[j]$

Ceasurile vectoriale au o mare varietate de aplicare. Sunt folosite pentru implementarea depanării distribuite, sisteme distribuite cu memorie partajată şi definirea punctelor de întrerupere globale.

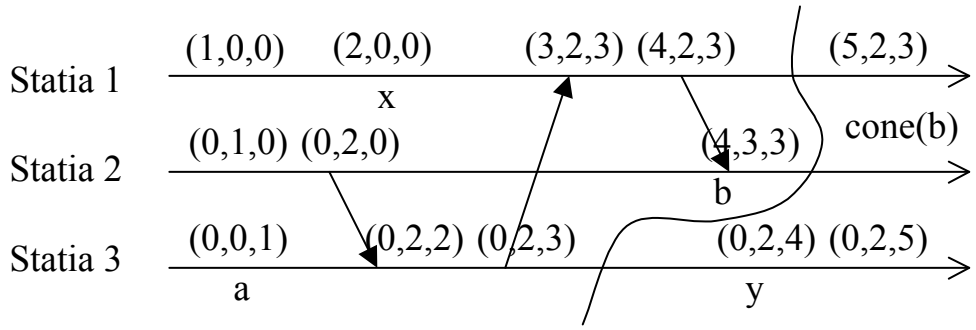


Fig. 6.8. Un exemplu al progresului vectorului de ceas

Dacă în regula R1 considerăm că *d = 1*, atunci avem următorul rezultat: $vt_i[i]$ contorizează numărul de evenimente produse de staţia S_i.
Deci, dacă considerăm evenimentul *e* ştampilat cu *vh* avem:

vh[j] = numărul de evenimente produse de staţia S_j, care îl preced cauzal pe *e (height (e))*.

$\sum vh[j] - 1$ = numărul total de evenimente care îl preced cauzal pe *e*.
Definim acest număr ca fiind lăţimea conului de cauzalitate al evenimentului *e* (*weight of cone (e)*).

În exemplul prezentat în figura 6.8, ştampila *(4, 3, 3)* asociată evenimentului *b* indică faptul că *4* evenimente locate pe staţia S_1 îl preced pe *b*, şi lăţimea conului de cauzalitate asociată lui *b* este *9*. Această lăţime a conului de cauzalitate este numărul minim de evenimente care trebuie să apară înaintea lui e.

O măsură a concurenţei pentru calculele distribuite

O astfel de masură simplă poate fi definită în următorul mod:
Fie *e* un eveniment. Măsura concurenţei asociată evenimentului *e* este:

$$cm(e) = \frac{n * height(e) - weight\_of\_cone(e)}{(n-1) * height(e)}$$

Această măsură pretinde că necesarul de calcul pentru producerea evenimentului *e* este maxim concurent dacă *cm (e) = 0*; în opoziţie, calculul este în întregime secvenţial dacă *cm (e) = 1*;

6.4.4. Timp matriceal

Ceasuri matriceale

În acest caz, timpul logic global este reprezentat de o matrice de dimensiune $n \times n$. Deci, fiecare staţie S_i este dotată cu o astfel de matrice, notată cu $mt_i[1..n, 1..n]$, ale cărei intrări au următoarele înţelesuri:

- $mt_i[i,i]$ este timpul logic local al staţiei S_i, incrementat de progresul de calcul făcut de staţia S_i.
- $mt_i[k,l]$ reprezintă imaginea (cunoştinţele) staţiei S_i despre cunoştinţele staţiei S_k relativ la timpul logic local al staţiei S_l. Întreaga matrice mt_i constituie imaginea locală a staţiei S_i despre timpul logic global.

De fapt, linia $mt_i[i, .]$ nu este nimic altceva decât vectorul de ceas $vt_i[.]$, deci această linie moşteneşte proprietăţile sistemului cu ceasuri vectoriale.

Regulile R1 şi R2 sunt similare cu modelul precedent, pentru fiecare staţie S_i:

R1: înainte de producerea unui eveniment se execută:

$mt_i[i, i] := mt_i[i, i] + d \quad (d > 0)$

R2: fiecare mesaj m împachetează o matrice de tip mt. La recepţionarea unui astfel de mesaj, (m, mt), de la staţia S_j, staţia S_i execută:

$1 \le k \le n: mt_i[i, k] := max (mt_i[i, k], mt[j, k])$

$1 \le k, l \le n: mt_i[k, l] := max (mt_i[k, l], mt[k, l]),$

după care execută R1

Proprietăţi

În plus faţă de proprietăţile sistemului de ceasuri vectoriale, pentru $mt_i[i, .]$ avem încă una:

$min (mt_i[k, i]) \ge t \Rightarrow$ staţia S_i ştie că fiecare altă staţie îi cunoaşte progresul

k

până la timpul t.

Această proprietate poate permite unei staţii să nu mai trimită o informaţie cu un timp local $\le t$, sau să şteargă informaţiile vechi.

CAPITOLUL 7

ZONĂRI ÎN SISTEME CU SERVICII DISTRIBUITE

O zonare a unui sistem distribuit constă în partiţionarea acelui sistem, în sisteme mai mici, în aşa fel încât fiecare din componentele sale să aibă un anumit grad de autonomie în ce priveşte folosirea unor resurse şi realizarea unor activităţi.

Un sistem distribuit îl modelăm, în general, cu ajutorul unui graf conex *G, G=(X, U) ; X={1, 2, .. . , n}*.

Ne punem problema de a partiţiona mulţimea vârfurilor *X* pentru ca să realizăm o zonare, deci să descompunem graful *G* într-un anumit număr de subgrafe, considerate fiecare o zonă, respectiv $G_i = (X_i, U_i)$, $i = \overline{1,M}$, astfel încât:

$$X = \bigcup_{i=1}^{M} X_i, \quad X_i \cap X_j = \varnothing, \ \forall i \neq j, \quad i = \overline{1,M}, j = \overline{1,M}$$

Această partiţionare se poate face după diverse criterii, având în vedere un sistem de servicii distribuite.

Majoritatea aplicaţiilor pe un sistem distribuit sunt de aşa natură încât nodurile sunt solicitate inegal. Acest lucru duce la crearea unei aglomeraţii în realizarea unor servicii în unele noduri şi o solicitare mai mică a altor noduri dintr-o reţea de calculatoare. O soluţie a descongestionării resurselor şi realizarea unei creşteri a performanţelor de prelucrare globală este descompunerea reţelei de servicii în subreţele numite zone. Se realizează şi o distribuire a unor sarcini la nivelul zonelor care au o anumită autonomie în îndeplinirea acestora. Este util să grupăm nodurile sistemului distribuit astfel încât între aceste grupări să existe o anumită echitate, solicitarea să fie dacă se poate aceeaşi.

Vom lua în considerare şi alte criterii pentru realizarea unei astfel de zonări a grafului ce defineşte un sistem distribuit. Pentru ca să avem o explicaţie mai clară vom considera un exemplu de reţea definită de un graf neorientat şi conex.

Să nu uităm că pentru realizarea unei zonări, după nişte criterii bine precizate, trebuie să ne referim la o aplicaţie distribuită care a rulat mai mult timp şi s-au putut determina anumite valori asociate nodurilor reţelei. Să notăm cu h_i ponderea de solicitare a nodului i din sistem în realizarea unor sarcini, ceea ce înseamnă că:

$$\sum_{i=1}^{M} h_i = 1$$

Ponderea h_i asociată nodului i este egală cu a câta parte din întreg reprezintă solicitarea acestui nod, respectiv procentul de solicitare. De exemplu, ponderii $h=0,45$ îi corespunde procentul de *45%* de solicitare a unui nod.

Iată un exemplu de reţea de servicii distribuite ce conţine şi ponderile asociate vârfurilor)fig. 7.1).

Fie $G = (X, U)$; $X = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$

$U = \{[1,2], [1,3], [2,4], [2,5], [2,9], [3,4], [3,6], [3,7], [4,5], [4,8], [4,7], [5,9], [5,8], [6,7], [7,8], [8,9]\}$

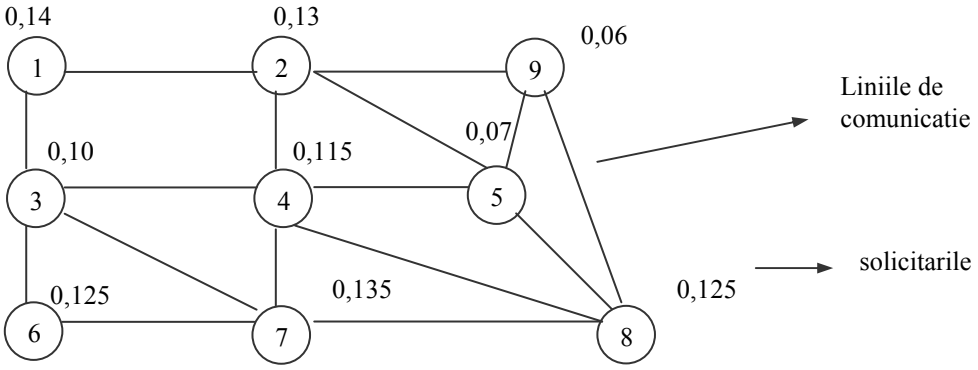


Fig. 7.1. Un exemplu de reţea de servicii distribuite cu ponderile asociate vârfurilor

Se observă că avem suma ponderilor asociate vârfurilor egală cu *1,* adică:

$$\sum_{i=1}^{9} h_i = 1 .$$

Presupunem că vrem să facem o zonare a acestui graf formată din două subgrafe: G_1 şi G_2 cu respectarea unor criterii date.

Vom avea în vedere criteriile de zonare pe care le prezentăm în continuare.

7.1. Criteriul echităţii

Considerăm exemplul dat mai sus pentru a explica criteriul echităţii pe care îl vom prezenta în continuare. Vom partiţiona în două grupe mulţimea nodurilor în aşa fel încât gradul de solicitare în cele două zone să fie, dacă se poate, acelaşi. Admitem o toleranta α, valoare cuprinsă între *0* şi *1*.

Valoarea asociată (solicitarea) nodului *i* o notăm cu h_i şi avem:

$$\sum_{i=1}^{9} h_i = 1$$

Considerăm următoarele două zone care respectă criteriul echităţii cel mai bine posibil în situaţia dată:

$G_1 = (X_1, U_1)$, $G_2 = (X_2, U_2)$; $X_1 \cup X_2 = X$
$X_1 = \{1, 2, 3, 4\}$, $h_1 + h_2 + h_3 + h_4 = 0{,}485$ (48,5%)
$X_2 = \{5, 6, 7, 8, 9\}$, $h_5 + h_6 + h_7 + h_8 + h_9 = 0{,}515$ (51,5%)

Să considerăm acum un graf oarecare $G=(X, U)$, $X=\{1, 2, .. . , n\}$ şi fie *M* numărul de zone dorit. Atunci, $G_i = (X_i, U_i)$, $i = \overline{1,M}$ – vor fi subgrafele ce definesc cele *M* zone a sistemului dat. O echitate perfectă avem atunci când fiecare subreţea are a *1/M –a* parte din solicitarea (cererea) totală. În general acest lucru nu se poate realiza. De aceea, considerăm că avem o echitate în ceea ce priveşte serviciile într-o asemenea partiţie, dacă suma valorilor asociate vârfurilor din fiecare subsistem e foarte aproape de *1/M*.

Notam: $\overline{h} = \frac{1}{M}$. Admitem o anumită *deviere* de la $\overline{h}$, respectiv o *toleranţă* α, $0<\alpha<1$.

Dacă respectăm criteriul echităţii şi dacă mulţimea vârfurilor sistemului distribuit $G = (X, U)$ este $X = \{ 1, 2, ..., n \}$ atunci avem:

$$| \sum_{j \in X_i} h_j - \overline{h} | \le \alpha \cdot \overline{h}$$

Un graf G_i care reprezintă o zona din sistemul distribuit considerat e acceptat conform criteriului echităţii dacă se verifică această inegalitate. Sigur că se poate da un algoritm pentru construirea de subreţele acceptabile din punctul de vedere al echităţii. Pentru aceasta este suficient să alegem toate grăpările posibile de noduri şi să verificăm dacă are loc inegalitatea de mai sus.

Evident că pot fi posibile mai multe grupări de noduri care să verifice criteriul echităţii impus de inegalitatea de mai sus în care apare toleranţa α.

7.2. Criteriul contiguităţii

O zona este contiguă dacă există posibilitatea să ajungem de la un nod la altul al subreţelei fără a trece prin alte noduri din afara zonei determinate de subsistem. Cu alte cuvinte, între oricare două vârfuri ale subsistemului trebuie să existe un lanţ al subgrafului respectiv. Evident că acest drum s-ar putea să nu fie cel mai scurt din întreg sistemul.

Acest criteriu asigură independenţa şi o eventuală autonomie a zonei respective care e contiguă, relativă la gestiunea unor date.

Pentru găsirea vârfurilor unui sistem distribuit ce pot face parte dintr-o zona contiguă se poate folosi matricea de adiacenţă în care se urmăresc elementele *1* vecine.

7.3. Criteriul compactităţii

O interpretare intuitivă a acestui criteriu este că între două vârfuri de la marginea unei zone nu trebuie să existe o distanta prea mare; în practică putem realiza compactitatea prin impunerea unor restricţii privind lungimea celui mai scurt drum între oricare două vârfuri candidate să aparţină unei zone; putem măsura această compactitate, de exemplu, prin asemănarea zonelor cu un pătrat sau un cerc sau prin stabilirea unui diametru a zonei. Pornind de la un tablou al distanţelor între vârfurile unui sistem distribuit putem construi un altul (matricea de excludere) cu elemente *1* şi *0* după cum distanţa dintre două vârfuri e mai mică sau mai mare decât o anumită valoare β pe care o alegem şi care se numeşte *distanţă de excludere*. Două noduri pot face parte dintr-o zonă dacă distanţa dintre ele e mai mică decât valoarea β prestabilită.

Fie E o matrice pătratică $n \times n$ unde

$$E=(E_{ik}),\ apoi\ \ E_{ik}=\begin{cases} 1,\ daca\ \ d(i,k) \le \beta \\ 0,\ \ in\ caz\ contrar \end{cases},$$

iar $d(i,k)$ este cea mai mică distanţă de la i la k unde $1 \le i, k \le n$. Zona G_i este compactă dacă pentru oricare două noduri a şi b ale zonei considerate, avem $E_{ab}=1$.

7.4. Enclave

O *enclavă* este alcătuită din unul sau mai multe noduri care nu pot fi legate între ele, situaţie la care ajungem dacă am ţinut cont pe rând de precedentele criterii. Trebuie construită zonarea în aşa fel încât să nu se ajungă la situaţii în care exista enclave. Se încearcă excluderea enclavelor prin reluarea procedurii de zonare dacă acest lucru este posibil. Putem avea şi eşec în acest proces de zonare. Ieşirea este posibilă prin modificarea valorilor distanţei de excludere β alegând pentru aceasta valori mai mari.

7.5. Criterii suplimentare

Pentru probleme concrete, în afara criteriilor enunţate mai sus pot fi considerate şi alte criterii suplimentare specifice. Criteriile suplimentare sunt în general specifice problemei de rezolvat.

7.6. Modele matematice al problemei de zonare a unui sistem cu servicii distribuite

În concluzie, acum putem formula modelul matematic al problemei de zonare a unui sistem cu servicii distribuite.

Se dă un sistem distribuit, modelat de un graf $G = (X, U)$, $X = \{1, 2, \ldots, n\}$. Să se descompună acest sistem în M zone $G_i = (X_i, U_i)$, $X_i \subset X$, $i = \overline{1,M}$, astfel încât mulţimea nodurilor să fie partiţionată, adică:

$$\bigcup_{i=1}^{M} X_i = X;\ X_i \cap X_j = \varnothing,\quad i \neq j;\ i, j = \overline{1,M}$$

Aceste zone trebuie să aibă caracteristicile, sau altfel spus să îndeplinească criteriile de partiţionare, prezentate mai sus şi pe care le reluăm în continuare.

7.6.1. Echitatea

Dându-se solicitările h_j ($\sum_{f=1}^{n} h_j = 1$) pentru fiecare nod j, când $j \in X=\{1,2, \ldots ,n\}$ avem echitate, dacă pentru fiecare zonă cu nodurile X_j se verifică relaţia:

$$|\sum_{i\in X_j} h_i - \frac{1}{M}| \le \alpha \cdot \frac{1}{M}, \quad j = \overline{1,M}$$

unde α, $0 < \alpha < 1$ este devierea maximă admisă.

7.6.2. Contiguitatea

Zona determinată de subgraful G_i este contiguă dacă şi numai dacă G_i sunt conexe (există un lanţ între oricare două noduri ale acestuia).

7.6.3. Compactitatea

Zona X_j este compactă dacă cea mai scurtă distanţă dintre oricare două noduri $a, b \in X_j$ este mai mică sau egală cu β - o constantă numită distanţă de excludere - anticipat precizată.

$$E_{i,k} = \begin{cases} 1, \text{ daca } d(i,k) \le \beta \\ 0, \text{ altfel} \end{cases}$$

Dacă se obţin enclave, reluăm procedeul pentru a considera alte grupări de vârfuri din sistemul distribuit pentru care să fie respectate criteriile *1, 2, 3* menţionate mai sus, eventual cu o altă valoare pentru β, până când ajung la o situaţie în care nu avem enclave.

7.6.4. Problema zonării ca o problemă de programare booleeană

Problema zonării prezentată mai sus poate fi formulată şi în alte variante, de exemplu, ca o problemă de programare booleeană.

Să presupunem că există S zone fezabile, care respectă criteriile enumerate mai sus. Fie z_j o variabilă binară, $z_j = 1$ dacă zona j este selectată şi $z_j = 0$ în caz contrar.

Se cere

$$min\ max_{j=\overline{1,S}}\ c_j z_j$$

în condiţiile

$$\sum_{j=1}^{S} a_{ij}\, z_j = 1, \quad i = \overline{1,n} \quad ; \qquad \sum_{j=1}^{S} z_j = M$$

unde

$$a_{ij} = \begin{cases} 1, \text{ daca nodul } i \in X_j \\ 0, \text{ in caz contrar} \end{cases}$$

şi

$$c_j = \left| \sum_{i \in X_j} h_i - \frac{1}{M} \right| /(\alpha/M)$$

este deviaţia relativă zonei $G_j(X_j, U_j)$ faţă de solicitarea de $1/M$.

CAPITOLUL 8

SISTEME DE AŞTEPTARE DISTRIBUITE

În afara modelelor deterministe, discutate până acum, exista şi modele probabilistice (stohastice) care sunt legate strâns de teoria aşteptării.

8.1. Elemente de teoria aşteptării

Teoria aşteptării apare la începutul secolului trecut ca instrument de rezolvare a unor probleme legat de încărcarea rațională a liniilor de comunicații prin telefoane. Aceeaşi problemă revine în actualitate odată cu apariția rețelelor de calculatoare care se dezvoltă atât de mult.

Ulterior teoria aşteptării se dezvoltă de sine-stătător.

8.1.1. Elementele unui sistem de aşteptare

Avem de-a face cu un sistem de aşteptare dacă sunt precizate următoarele trei elemente:

1. ***unități*** ale sistemului, în general considerate unități de prelucrare de date, de exemplu mesaje transmise de la o stație la alta în vederea prelucrării mesajului respectiv de stația receptoare;
2. ***stații,*** care sunt mijloace de tratare a unităților care aşteaptă să fie prelucrate (de exemplu calculatoare);
3. ***serviciu***, în sensul că stațiile din cadrul sistemului de aşteptare asigură anumite servicii de prelucrare a unităților ce sosesc la stațiile ce constituie un sistem de aşteptare (*s.a*)

Într-un sistem de aşteptare se poate ca stațiile să nu aibă o capacitate suficient de mare pentru a face serviciile unităților care sosesc prompt, deci imediat, şi în acest caz se formează o ***coadă de aşteptare*** de lungime nedeterminată. Se pune problema studierii unui astfel de sistem, luând în considerare anumite caracteristici ale lui. Ne interesează comportamentul unui sistem de aşteptare ținând cont că unele evenimente dintr-un sistem de aşteptare sunt probabilistice.

Iată câteva probleme ce se studiază în legătură cu un sistem de aşteptare:

- numărul mediu de unități din sistemul de aşteptare;
- numărul mediu de unități care vin într-o unitate de timp în sistemul de aşteptare;

- timpul mediu de servire;
- timpul mediu de aşteptare în coadă;
- probabilitatea ca durata de aşteptare (servire) să depăşească o valoare dată.

8.1.2. Elemente de bază

Un sistem de aşteptare, pentru a putea fi studiat, trebuie modelat. Iată elementele de bază ale unui model de aşteptare:

1) ***Timpul de servire*** într-un sistem de aşteptare – îl considerăm ca fiind de natură probabilistică, adică dat de o variabilă aleatore ce urmează o anumită lege de repartiţie. Sistemul de aşteptare pe care îl luăm în considerare în cadrul sistemului distribuit, urmează legea exponenţială negativă.

Variabilele aleatoare X pot fi:

- *variabile aleatoare de tip discret,* date sub formă de repartiţie sau altfel spus, distribuţie:

$$X\begin{pmatrix} x_k \\ p_k \end{pmatrix}_{k\in I}$$

unde $I \subseteq N$, apoi p_k este probabilitatea cu care variabila aleatoare X ia valoarea x_k. Evident, avem $\sum p_k = 1$.

O variabilă aleatoare X urmează legea de repartiţie Poisson dacă are distribuţia

$$X\begin{pmatrix} k \\ p_k(\lambda) \end{pmatrix}_{k=0,1,\ldots}; \quad p_k(\lambda) = \frac{\lambda^k}{k!}e^{-\lambda}, \ \lambda>0$$

Putem stabili uşor că valoarea medie a variabilei aleatoare X, care urmează legea de repartiţie Poisson, este

$$M(X) = \sum_{k=0}^{\infty} k\, p_k(\lambda) = \sum_{k=0}^{\infty} k \frac{\lambda^k}{k!}e^{-\lambda} = \lambda$$

- *variabile aleatoare de tip continuu*

În acest caz este necesar să definim funcţia de repartiţie asociată unei variabile aleatoare X, discrete sau continue. Prin definiţie ***funcţia de repartiţie*** asociată lui X este o funcţie $F_X : R \to [0,1]$ sau $F : R \to [0,1]$ cu valorile date de probabilitatea ca valorile variabilei aleatoare X să fie mai mici decât $x \in R$.

$$F(x) = P(X < x)$$

Repartiţia unei variabile aleatoare de tip continuu se dă sub forma

$$X\begin{pmatrix} x \\ f(x) \end{pmatrix}_{x\in R}$$

unde f este ***densitatea de probabilitate***, iar funcţia de repartiţie a aceleaşi variabile aleatoare este dată prin relaţia:

$$F(x)= \int_{-\infty}^{x} f(t)dt .$$

Să menţionăm proprietăţile:

- $0\le F(x)\le 1;$ crescătoare;
- $f(x) \ge 0$ şi $\int_{-\infty}^{\infty} f(t)dt = 1$.

Dacă X urmează legea de repartiţie exponenţială negativă, atunci densitatea de probabilitate este

$$f(x)=\begin{cases} \lambda\, e^{-\lambda x}, & x>0 \\ 0, & x\le 0 \end{cases} \quad , \quad \lambda>0$$

iar valoarea medie este

$$M(x) = \int_{-\infty}^{\infty} xf(x)\,dx = \frac{1}{\lambda} .$$

2) ***Venirile*** – le considerăm ca fiind întâmplătoare, deci reprezintă o variabilă X aleatoare cu o anumită repartiţie. Venirile sunt de tipul variabilelor aleatoare discrete şi, de exemplu, le considerăm ca urmând legea de repartiţie Poisson cu,

$$p_k = p_k(\lambda) = \frac{\lambda^k}{k!} e^{-\lambda} , \ \lambda>0; \ M(X)=\lambda .$$

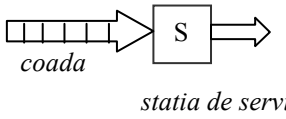
Rezultă că numărul mediu de unităţi $M(X)$ ce sosesc în sistem este λ.

În acest caz, însă, intervalul dintre două veniri succesive îl considerăm o variabilă continuă ce urmează legea de repartiţie exponenţială negativă, cu densitatea de probabilitate *f(x)* dată mai sus.

3) ***Factorul de servire*** – raportul dintre numărul mediu de unităţi ce sosesc în sistem şi numărul mediu de unităţi ce pleacă din sistem într-o unitate de timp aleasă; îl notăm cu ρ. Problema de studiu care prezintă interes pentru noi este atunci când $\rho \geq 1$, caz în care se formează o coadă de aşteptare (dacă $\rho < 1$, avem o servire promptă, nu mai are loc aşteptarea).

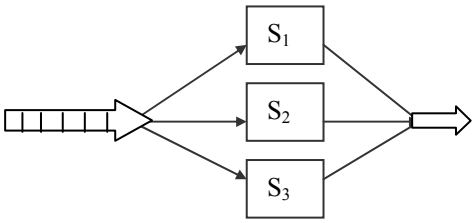
4) ***Staţiile de serviciu***. Se deosebesc trei situaţii pe care le reprezentăm în desenele care urmează:

a) Sistemul are o singură staţie

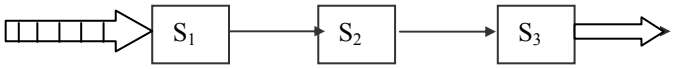


statia de servire

b) Sistem cu mai multe staţii care lucrează în paralel



c) Sistem cu mai multe staţii care lucrează în serie



5) ***Disciplina aşteptării***. Aceasta precizează modul după care se dispun unităţile în şirul de aşteptare şi ordinea în care sunt servite.

Disciplina de aşteptare poate fi de tip *FIFO* (primul venit, primul servit); putem considera şi alte reguli.

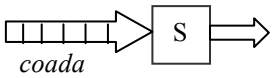
6) ***Modul de servire a unităţilor în sistem***: se deosebesc următoarele cazuri:

- *serviciu imediat*: o unitate este servită o singură dată, după care ea părăseşte definitiv sistemul;

- *serviciu în bloc*: toate unităţile ce aşteaptă să fie servite până la un anumit număr maxim dat intră în staţie pentru a fi servite; evident că servirea se face în mod succesiv.

8.2. Modelul de aşteptare cu o singură staţie de servire

Pentru modelul de aşteptare cu o singură staţie reluăm reprezentarea dată mai sus.



statia de servire

Pentru studiul unui astfel de sistem facem următoarele ipoteze importante:

1. Sosirile în sistem sunt infinite, întâmplătoare, independente unele de altele şi numărul de veniri într-o unitate de timp este o variabilă aleatoare ce urmează o repartiţie exponenţială negativă de parametru λ.

2. Serviciile sunt independente între ele şi independente de veniri; durata serviciului e o variabilă aleatoare ce urmează o repartiţie exponenţială negativă de parametru μ; o unitate, o data servită, părăseşte definitiv sistemul.

3. Există o singură staţie de serviciu în sistem;

4. Disciplina sistemului este de tip *FIFO*.

În aceste ipoteze vom deduce un sistem de ecuaţii diferenţiale ce modelează un sistem de aşteptare de această natură.

Considerăm sistemul la momentul iniţial $t = 0$. Evaluăm probabilitatea ca în sistem, într-un interval de timp Δt, $\Delta t > 0$ să avem o venire. Notam această probabilitate cu $A(\Delta t)$, adică avem $A(\Delta t) = P(X < \Delta t)$. Evident că probabilitatea ca în intervalul Δt să nu aibă loc nici o venire este egală cu $1 - A(\Delta t)$.

Ţinem seama că venirile sunt poisson-iene, iar timpul dintre două veniri succesive în sistem este o variabilă aleatoare cu repartiţia exponenţială negativă de parametru λ, cu densitatea de probabilitate precizată mai sus, deci avem:

$$A(\Delta t) = \lambda \int_0^{\Delta t} e^{-\lambda x} dx$$

Acest rezultat se bazează pe faptul că:

$$A(\Delta t) = P(X < \Delta t) = \int_{-\infty}^{\Delta t} f(x)dx = \int_0^{\Delta t} f(x)dx$$

Acum avem

$$A(\Delta t) = \lambda \int_0^{\Delta t} e^{-\lambda x} dx = \lambda(-\frac{1}{\lambda}) e^{-\lambda x} \Big|_0^{\Delta t} = 1 - e^{-\lambda \Delta t}$$

Deoarece

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots,$$

avem

$$A(\Delta t) = 1 - (1 + (-\lambda\Delta t) + \frac{(\lambda\Delta t)^2}{2!} - \dots) = \lambda\Delta t + \Delta t \cdot h_1(\Delta t),$$

unde $h_1(\Delta t) \to 0$, când $\Delta t \to 0$.

Din relaţia

$$A(\Delta t) = \lambda\Delta t + \Delta t \cdot h_1(\Delta t),$$

rezultă că o venire în sistem o putem considera proporţională cu λ.

Analog se determină probabilitatea ca într-un interval de timp Δt, $\Delta t > 0$ să avem o plecare din sistem – o notăm $B(\Delta t)$. Evident că probabilitatea ca în intervalul Δt să nu aibă loc nici o plecare este $1 - A(\Delta t)$.

Avem,

$$B(\Delta t) = \mu\Delta t + \Delta t h_2(\Delta t),$$

unde $h_2(\Delta t) \to 0$, când $\Delta t \to 0$.

Considerăm acum, în cele ce urmează, o situaţie generală referitoare la un sistem de aşteptare; studiem ce se întâmplă într-un interval de timp Δt (de la momentul t la $t + \Delta t$, ştiind că la momentul t în sistem sunt n unităţi.

Convenim să notăm cu:

$P_n(t)$ este probabilitatea ca la momentul t în sistem să fie n unităţi;
$P_n(t + \Delta t)$ este probabilitatea ca la momentul $t + \Delta t$ în sistem să fie n unităţi.

Studiem toate situaţiile posibile ca la momentul $t + \Delta t$ în sistem să avem n unităţi, ţinând cont că în sistem pot (vor) avea loc atât veniri cât şi plecări, iar acestea sunt independente.

Vom deosebi patru astfel de situaţii:

1) la momentul t în sistem sunt n unităţi, cu probabilitatea $P_n(t)$, deci la momentul $t + \Delta t$ nu pot fi decât tot n unităţi; în concluzie avem *0* – veniri, probabilitatea de apariţie a evenimentului fiind $1 - A(\Delta t)$ şi *0* – plecări, probabilitatea evenimentului fiind $1 - B(\Delta t)$;

Cum cele 3 evenimente sunt independente, rezultă că probabilitatea ca la momentul $t + \Delta t$ să avem în sistem n unităţi şi în intervalul Δt să nu fi avut loc veniri sau plecări este $P_n(t)[1 - A(\Delta t)]\ [1 - B(\Delta t)]$.

2) dacă la momentul t în sistem sunt $n + 1$ unităţi, lucru care se întâmplă cu probabilitatea $P_{n+1}(t)$, atunci la momentul $t + \Delta t$ ca să avem în sistem n unităţi trebuie să avem *0* – veniri cu probabilitatea $1 - A(\Delta t)$ şi *1* – plecare, eveniment a cărui probabilitate este $B(\Delta t)$.

Cum cele 3 evenimente sunt independente, rezultă că probabilitatea ca la momentul $t + \Delta t$ să avem în sistem n unităţi şi în intervalul Δt să nu fi avut loc veniri dar să fi avut loc o plecare este $P_{n+1}(t)[1 - A(\Delta t)]\ B(\Delta t)$.

3) dacă la momentul t în sistem sunt $n - 1$ unităţi, lucru care se întâmplă cu probabilitatea $P_{n-1}(t)$, atunci la momentul $t + \Delta t$ ca să avea n unităţi, înseamnă că avem *1* – venire cu probabilitatea $A(\Delta t)$ şi *0* – plecări, eveniment a cărui probabilitate este $1\text{-}B(\Delta t)$.

Cum cele 3 evenimente sunt independente, rezultă că probabilitatea ca la momentul $t + \Delta t$ să avem în sistem n unităţi şi în intervalul Δt să fi avut loc *1* venire şi să nu fi avut loc plecări este $P_{n-1}(t)\ A(\Delta t)[1 - B(\Delta t)]$.

4) la momentul t în sistem sunt n unităţi, lucru care se întâmplă cu probabilitatea $P_n(t)$, atunci la momentul $t + \Delta t$ rămân tot n unităţi dacă avem *1* – venire cu probabilitatea $A(\Delta t)$ şi *1* – plecare, eveniment a cărui probabilitate este $B(\Delta t)$.

Cum cele 3 evenimente sunt independente, rezultă că probabilitatea ca la momentul $t + \Delta t$ să avem în sistem n unităţi şi în intervalul Δt să fi avut loc *o* venire şi o plecare, este $P_n(t)\ A(\Delta t)B(\Delta t)$.

Cele patru situaţii definesc patru evenimente care sunt incompatibile şi atunci probabilitatea ca la momentul $t + \Delta t$ în sistem să fie n unităţi este suma probabilităţilor evenimentelor corespunzătoare evenimentelor din celor patru situaţii prezentate mai sus. În consecinţă, avem

$$P_n(t + \Delta t) = P_n(t)[1 - A(\Delta t)]\ [1 - B(\Delta t)] + P_{n+1}(t)[1 - A(\Delta t)]\ B(\Delta t) + P_{n-1}(t)\ A(\Delta t)[1 - B(\Delta t)] + P_n(t)\ A(\Delta t)\ B(\Delta t)$$

Apoi, să ţinem cont de expresiile lui $A(\Delta t)$ şi $B(\Delta t)$ şi să observăm că termenii care conţin pe Δt la puteri mai mari sau egale cu 2 îi putem neglija, deoarece tind la 0. În acest fel vom neglija termenii ce conţin produsul $A(\Delta t)\,B(\Delta t)$.

Putem scrie

$$P_n(t+\Delta t) - P_n(t) \approx - P_n(t)[A(\Delta t)+B(\Delta t)] + P_{n-1}(t)\,A(\Delta t) + P_{n+1}(t)B(\Delta t)$$

$$\approx -(\lambda+\mu)\Delta t\, P_n(t) + \lambda\Delta t\, P_{n-1}(t) + \mu\Delta t P_{n+1}(t)$$

De aici dacă împărţim cu Δt, $\Delta t > 0$,obţinem

$$\frac{P_n(t+\Delta t) - P_n(t)}{\Delta t} = -(\lambda+\mu)\, P_n(t) + \lambda\, P_{n-1}(t) + \mu P_{n+1}(t)\ ;\ n>0$$

iar dacă facem ca $\Delta t \to 0$ atunci vom obţine ecuaţia diferenţială

$$\frac{d\,P_n(t)}{d\,t} = \lambda\, P_{n-1}(t) - (\lambda+\mu)\, P_n(t) + \mu\, P_{n+1}(t)\ ;\ n > 0$$

unde λ este numărul mediu de unităţi ce sosesc în sistem/unitate de timp, iar μ este numărul mediu de unităţi ce pleacă din sistem/unitate de timp.

Cazul n = 0. Acest caz necesită o analiză separată.

Avem doar două situaţii distincte care iau în calcul *0* unităţi în sistem

a) La momentul t în sistem sunt *0* unităţi, lucru care se întâmplă cu probabilitatea $P_0(t)$, iar la momentul $t+\Delta t$ vor fi tot *0* unităţi în sistem, ceea ce înseamnă că la acest moment de timp nu avem nici o venire iar despre plecări nu putem vorbi. În concluzie, rezultă că probabilitatea ca la momentul $t + \Delta t$ să avem în sistem *0* unităţi şi în intervalul Δt să nu fi avut loc veniri este este dată de expresia $P_0(t)[1-A(\Delta t)]$.

b) La momentul t în sistem avem o unitate lucru care se întâmplă cu probabilitatea $P_1(t)$. Ca să rămână la momentul $t+\Delta t$, *0* unităţi în sistem înseamnă că trebuie să avem o plecare şi nici o venire. Probabilitatea ca să se întâmple acest lucru este $P_1(t)\, B(\Delta t)\, [1-A(\Delta t)]$.

Cum cele două evenimente sunt incompatibile, probabilitatea ca la momentul $t+\Delta t$ să fie în sistem *0* unităţi este dată de relaţia:

$$P_0(t+\Delta t) = P_0(t)[1-A(\Delta t)] + P_1(t)\, B(\Delta t)\, [1-A(\Delta t)].$$

De aici,

$$P_0(t+\Delta t) - P_0(t) = - P_0(t)\, A(\Delta t) + P_1(t)\, B(\Delta t) - P_1(t)\, B(\Delta t)\, A(\Delta t)],$$

iar dacă neglijăm ultimul termen şi ţinem seama de expresiile probabilităţilor $A(\Delta t)$, $B(\Delta t)$ rezultă

$$P_0(t+\Delta t) - P_0(t) = - \lambda P_0(t)\, \Delta t + \mu P_1(t)\, \Delta t.$$

Acum, împărţind cu Δt şi făcând ca $\Delta t \to 0$, rezultă următoarea ecuaţie diferenţială:

$$\frac{d P_0(t)}{dt} = - \lambda P_0(t) + \mu P_1(t).$$

În final pentru modelul de sistem de aşteptare cu o singură staţie avem următorul sistemul de ecuaţii diferentiale:

$$\begin{cases} \dfrac{d P_n(t)}{d t} = \lambda P_{n-1}(t) - (\lambda + \mu) P_n(t) + \mu P_{n+1}(t)\,; n > 0 \\ \dfrac{d P_0(t)}{dt} = - \lambda P_0(t) + \mu P_1(t) \quad n=0. \end{cases}$$

numite ***ecuaţiile lui Chapman – Kolmogorov***.

Evident, suntem interesaţi să găsim o soluţie a acestui sistem de ecuaţii diferenţiale, pentru valori concrete date lui n.

Ipoteza echilibrului static. În practică se face ipoteza că într-un interval de timp mic nu se produce o modificare esenţială în sistem, adică

$$\frac{d P_n(t)}{dt} = 0, \quad n \geq 0$$

şi atunci notăm

$$P_n(t) = p_n\,; \quad n \geq 0,$$

iar sistemul de ecuaţii diferenţiale, de mai sus, devine un sistem de ecuaţii algebrice:

$$\begin{cases} - \lambda p_0 + \mu p_1 = 0 \\ \lambda p_{n-1} - (\lambda + \mu) p_n + \mu p_{n+1} = 0, \quad n>0. \end{cases}$$

Din prima ecuaţie rezultă

$$p_1 = \frac{\lambda}{\mu} p_0,$$

iar dacă notăm $\rho = \frac{\lambda}{\mu}$ atunci avem:

$$p_1 = \rho\, p_0.$$

Din a doua ecuaţie pentru valori consecutive date lui *n* avem:

$$n=1; \quad \mu p_2 = (\lambda + \mu)\rho p_0 = \frac{\lambda^2}{\mu} p_0$$

$$p_2 = \rho^2 p_0.$$

În general, prin inducţie , putem arăta că avem:

$$p_n = \rho^n p_0 \ ; n \geq 1.$$

Observăm că p_0 se poate determina din proprietatea: dacă $\rho < 1$ atunci

$$\sum_{n=0}^{\infty} \rho^n = \frac{1}{1-\rho}; \text{ apoi, } \sum_{n=0}^{\infty} p_n = 1, \quad \text{deci} \quad p_0 = \left(\frac{1}{1-\rho}\right)^{-1} = 1 - \rho$$

şi deci,

$$p_n = \rho^n (1- \rho), \quad n \geq 0.$$

Se poate determina uşor valoarea maximă a lui p_n , calculând derivata de ordinul întâi în raport cu ρ şi egalând-o cu *0,* obţinem $\rho = n/(n+1)$, de unde

$$max_\rho = [n/(n+1)]^n \, [1/(n+1)].$$

Caracteristici numerice ale modelului

1) *Numărul mediu de unităţi în sistemul de aşteptare*

Numărul mediu de unităţi în sistemul de aşteptare pe care-l notăm cu $\bar{n}$, se determină considerând o variabilă aleatoare cu distribuţia

$$X\begin{pmatrix} 0 & 1 & . & . & . & n & . & . & . \\ p_0 & p_1 & . & . & . & p_n & . & . & . \end{pmatrix}$$

unde numărul p_n este cel determinat anterior; ştiind că la un moment dat în sistem se pot găsi n unităţi, cu probabilitatea $p_n = \rho^n (1-\rho)$, $n \geq 0$. Valoarea medie $\bar{n}$ a acestei variabile aleatoare este:

$$\bar{n} = (1-\rho)\sum_{n=0}^{\infty} n\rho^n ;$$

dar cum,

$$\rho\sum_{n=1}^{\infty} n\,\rho^{n-1} = \rho\frac{d}{d\rho}\frac{1}{1-\rho} = \frac{\rho}{(1-\rho)^2}$$

obţinem că numărul mediu de unităţi în sistemul de aşteptare, este:

$$\bar{n} = \sum_{n=0}^{\infty} np_n = \frac{\rho}{1-\rho}$$

2) Numărul mediu de unităţi în coada de aşteptare

Numărul de unităţi în coada de aşteptare este dat de variabila aleatoare

$$X\begin{pmatrix} 0 & 1 & . & . & . & n & . & . & . \\ p_1 & p_2 & . & . & . & p_{n+1} & . & . & . \end{pmatrix}$$

Probabilitatea ca în coada de aşteptare să fie *0* unităţi, este p_1; aceasta pentru că o unitate este la servire ş.a.m.d. În această situaţie numărul mediu de unităţi în coada de aşteptare, pe care-l notăm cu $\bar{a}$, este:

$$\bar{a} = \sum_{n=0}^{\infty} np_{n+1} = (1-\rho)\sum_{n=0}^{\infty} n\rho^{n+1} = \frac{\rho^2}{1-\rho} \ .$$

3) Timpul mediu de aşteptare în sistem şi în coada de aşteptare a unei unităţi

Timpul mediu de aşteptare în sistem a unei unităţi pe care-l notăm cu w se referă la timpul scurs din momentul în care o unitate se ataşează cozii de aşteptare şi până când staţia de servire devine liberă. După cum ştim λ reprezintă numărul mediu de unităţi ce sosesc în sistem/unitate de timp, deci este adevărată relaţia:

$$\lambda w = \bar{n}$$

de unde

$$w = \frac{\bar{n}}{\lambda} = \frac{\rho}{\lambda(1-\rho)}$$

Apoi, timpul mediu de aşteptare în coada de aşteptare a unei unităţi, pe care-l notăm cu w^* este dat de relaţia

$$w^* = \frac{\bar{a}}{\lambda} = \frac{\rho^2}{\lambda(1-\rho)}; \qquad w - w^* = \frac{1}{\mu}$$

Diferenţa dintre cele două valori de timp era de aşteptat să fie $1/\mu$ deoarece aceasta este tocmai timpul mediu de servire a unei unităţi, care l-am determinat mai sus în alt mod.

Observaţie. Un studiu analog se poate face pentru alte situaţii, ca de exemplu:

- sistemul de aşteptare are o capacitate limitată şi anume N unităţi, deci în coada de aşteptare pot fi cel mult $N-1$ unităţi, iar una se găseşte în staţia de servire. Aceasta înseamnă că în sistemul de ecuaţii diferenţiale stabilit mai sus $n = \overline{0, N-1}$.

  Cum

$$1 = \sum_{n=0}^{N} p_n = p_0(1 + \rho + \ldots + \rho^N); \quad p_0 = \frac{1-\rho}{1-\rho^{N+1}}$$

apoi,

$$p_n = \frac{1-\rho}{1-\rho^{N+1}} \rho^n$$

Numărul mediu de unităţi în sistem, se poate arăta la fel ca mai sus, că este

$$\bar{n} = \frac{\rho(1-(N+1)\rho^N + N\rho^{N+1})}{(1-\rho)(1-\rho^{N+1})}.$$

O relaţie asemănătoare putem obţine pentru numărul mediu de unităţi ce se găsesc în coada de aşteptare.

- Mai mult, modelul studiat care presupune că numărul venirilor în sistem este infinit, poate fi modificat astfel în cât să considerăm numărul venirilor ca fiind foarte mare, dar totuşi finit. Se pot stabili şi în acest caz ecuaţiile lui *Chapman – Kolmogorov,* urmând o cale asemănătoare cu cea de mai sus.

8.3. Reţele de cozi de aşteptare (model Jackson)

Fie un sistem distribuit format dintr-o reţea de calculatoare $C = \{c_1, c_2, \ldots, c_n\}$ şi luăm în considerare aceleaşi ipoteze simplificatoare ca pentru cazul unui sistem cu o singură staţie; presupunem date probabilităţile p_{ij} ca o unitate prelucrată pe c_1 să fie dirijată apoi spre c_j, adică probabilităţile de trecere de la c_i la c_j . Rezultă că se cunoaşte matricea de rutaj a reţelei:

$$P = \begin{pmatrix} p_{11} & p_{12} & \cdot & \cdot & \cdot & p_{1n} \\ p_{21} & p_{22} & \cdot & \cdot & \cdot & p_{2n} \\ & \cdot & & & & \\ & \cdot & & & & \\ & \cdot & & & & \\ p_{n1} & p_{n2} & \cdot & \cdot & \cdot & p_{nn} \end{pmatrix}$$

Notăm $k = (k_1, k_2, \ldots, k_n)$ – starea sistemului de aşteptare unde k_i reprezintă numărul de unităţi din sistemul de aşteptare care sunt în curs de prelucrare sau aşteaptă să fie prelucrate de calculatorul c_i.

Teorema (Jackson) *Probabilitatea ca sistemul de aşteptare să se găsească în starea k este:*

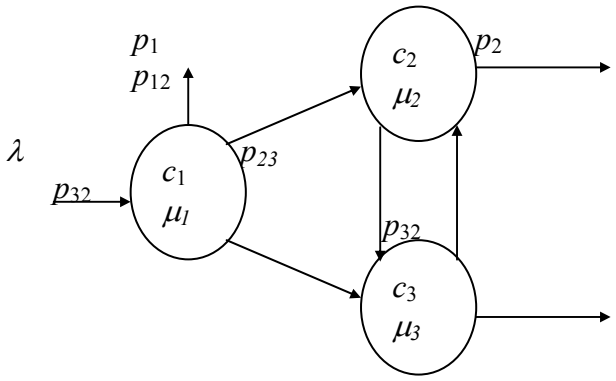
$$P(k) = \prod_{i=1}^{n} p_i(k_i)$$

unde $P_1(k)$ este probabilitatea ca în sistemul de aşteptare al calculatorului c_i să existe k_i unităţi, pentru $\forall i \in \{1, 2, \ldots, n\}$.

În regimul staţionar pe care-l considerăm aici aceste probabilităţi nu depind de timp.

Pentru o situaţie simplificată de reţea cu trei noduri ca cea din figura de mai jos [MOL89c, MOL92b], avem următoarea matrice de rutaj:

$$P=\begin{pmatrix} 0 & p_{12} & p_{13} \\ 0 & 0 & p_{23} \\ 0 & p_{32} & 0 \end{pmatrix}$$



Ipoteze:

- unităţile sosesc la calculatorul c_1 sub forma unui flux poisson-ian de parametru λ. O unitate servită la c_1 părăseşte reţeaua, cu probabilitatea p_1, este dirijată spre c_2 cu probabilitatea p_{12}, sau spre c_3 cu probabilitatea p_{13}. O unitate servită la c_2 părăseşte reţeaua cu probabilitatea p_2 sau este dirijată spre c_3 cu probabilitatea p_{23}. O unitate servită la c_3 părăseşte reţeaua cu probabilitatea p_3 sau este dirijată spre c_2 cu probabilitatea p_{32}. Timpul de servire a unei unităţi la staţia c_i este o variabilă aleatoare cu o repartiţie de tip exponenţial cu parametrul μ_i, $i=\overline{1,3}$

Notând

$$P(k_1,k_2,k_3)=P(k), \qquad k=(k_1,k_2,k_3)$$

ecuaţiile lui Chapman – Kolmogorov conduc, pentru cazul regimului staţionar considerat, la următorul sistem de ecuaţii pentru probabilităţile de stare.

$$(\lambda+\mu_1+\mu_2+\mu_3)P(k_1,k_2,k_3)=\lambda P(k_1-1,k_2,k_3)+\mu_1 p_1 P(k_1+1,k_2,k_3)+\mu_2 p_2 P(k_1,k_2+1,k_3)+\mu_3 p_3 P(k_1,k_2,k_3+1)+\mu_1 p_{12} P(k_1+1,k_2-1,k_3)+\mu_2 p_{23} P(k_1,k_2+1,k_3-1)+\mu_3 p_{32} P(k_1,k_2-1,k_3+1)+\mu_1 p_{13} P(k_1+1,k_2,k_3-1)$$

$k_i=1,2,\ldots;\ i=1,2,3$

$$(\lambda+\mu_2+\mu_3)P(0,k_2,k_3)=\mu_1 p_1 P(1,k_2,k_3)+\mu_2 p_2 P(0,k_2+1,k_3)+\mu_3 p_3 P(0,k_2,k_3+1)+\mu_1 p_{12} P(1,k_2-1,k_3)+\mu_2 p_{23} P(0,k_2+1,k_3-1)+\mu_3 p_{32} P(0,k_2-1,k_3+1)+\mu_1 p_{13} P(1,k_2,k_3-1)$$

$k_i=1,2,\ldots;\ i=1,2,3$

$(\lambda+\mu_1+\mu_3)P(k_1, 0, k_3) = \lambda P(k_1-1, k_2, k_3) +\mu_1 p_1 P(k_1+1, 0, k_3)+ \mu_2 p_2 P(k_1,1, k_3)+ \mu_3 p_3 P(k_1, 0, k_3+1)+\mu_2 p_{23} P(k_1,1, k_3-1) + \mu_1 p_{13} P(k_1+1, 0, k_3-1)$

$k_i=1,2,...; i=1,2,3$

$(\lambda+\mu_1+\mu_2)P(k_1, k_2, 0) = \lambda P(k_1-1, k_2, 0) +\mu_1 p_1 P(k_1+1, k_2, 0)+ \mu_2 p_2 P(k_1, k_2+1, 0)+ \mu_3 p_3 P(k_1, k_2,1)+ \mu_1 p_{12} P(k_1+1, k_2-1, 0)+ \mu_3 p_{32} P(k_1, k_2-1,1)$ $k_i=1,2,...; i=1,2,3$

$$P(0,0,0)= \mu_1 p_1 P(1, 0, 0)+ \mu_2 p_2 P(0, 1, 0)+ \mu_3 p_3 P(0, 0,1)$$

Pentru determinarea soluţiei sistemului de mai sus sub formă de produs

$$P(k_1, k_2,k_3)=P(k_1) P(k_2) P(k_3)$$

vom considera fiecare din cele trei staţii separat. Astfel, pentru prima staţie avem sistemul de aşteptare cu o singură intrare cu parametrul intrării $\lambda_1=\lambda$ şi parametrul servirii μ_1.

Atunci:

$$P_1(k_1)= \rho_1^{k_1} (1-\rho_1), \quad k_1=0,1,...$$

unde

$$\rho_1=\frac{\lambda}{\mu_1}$$

este intensitatea traficului primei staţii.

Pentru a doua staţie avem cu o intrare de parametru

$$\lambda_2 = \lambda \frac{p_{12} + p_{13} p_{32}}{1 - p_{23} p_{32}}$$

şi parametrul servirilor μ_2.

Atunci

$$\boldsymbol{P_2(k_2)= \rho_2^{k_2} (1-\rho_2), \quad k_2=0,1,...}$$

unde

$$\rho_2 = \frac{\lambda (p_{12} + p_{13} p_{32})}{\mu_2 (1 - p_{23} p_{32})}$$

este intensitatea traficului celei de a doua staţie.

În fine, pentru cea de a treia sta sistemul de aşteptare tot cu o intrare de parametru

$$\lambda_3 = \lambda \frac{p3 + p_{12} p_{23}}{1 - p_{32} p_{23}}$$

şi parametrul intrărilor μ_3 . Atunci

$$P_3(k_3) = \rho_3^{k_3} (1-\rho_3), \quad k_2 = 0,1,..$$

unde

$$\rho_3 = \frac{\lambda (p_{13} + p_{12} p_{23})}{\mu_3 (1 - p_{32} p_{23})}$$

este intensitatea traficului celei de a treia staţii.

După înlocuiri obţinem

$$P(k_1, k_2, k_3) =$$

$$\frac{\lambda^{k_1+k_2+k_3} (\mu_1 - \lambda)[\mu_2 (1 - p_{23} p_{32}) - \lambda (p_{12} + p_{13} p_{32})]}{\mu_1^{k_1+1} \mu_2^{k_2+1} \mu_3^{k_3+1} (1 - p_{23} p_{32})^2}$$

$$[\mu_3 (1 - p_{32} p_{23}) - \lambda (p_{13} + p_{12} p_{23})]$$

Relativ la un astfel de sistem de aşteptare, distribuit, se pot determina caracteristici numerice.

Numărul mediu de lucrări la calculatorul c_1 va fi

$$n = \frac{\rho_i}{1 - \rho_i}, \quad i = \overline{1,3}$$

Dacă disciplina servirii este *FIFO* atunci timpul mediu de aşteptare al unei unităţi la calculatorul c_i va fi:

$$T_i = \frac{1}{\mu_i(1-\rho_i)}, \quad i = \overline{1,3}$$

Numărul mediu de unităţi în reţea va fi:

$$a = \frac{\rho_1}{1-\rho_1} + \frac{\rho_2}{1-\rho_2} + \frac{\rho_3}{1-\rho_3}$$

Observaţie. Se poate face o generalizare naturală a reţelei particulare considerate în acest paragraf. Se pot lua în locul calculatoarelor c_2 şi c_3 o mulţime de oricâte calculatoare legate două câte două la fel ca în exemplul studiat aici. Se obţin relaţii asemănătoare cu cele găsite aici.

ANEXĂ[19]
MINIDICŢIONAR DE CALCUL PARALEL ŞI DISTRIBUIT

Surse: [DZI01], [DZI03] ş.a.

Pentru a satisface nevoia tot mai mare de reducere a timpului de rezolvare a unei probleme de mari dimensiuni, a fost găsită soluţia de a pune mai multe să lucreze simultan la rezolvarea sa, adică prelucrarea paralelă.

Calculul paralel/ distribuit/ concurent, paralelismul în general, cuprinde trei domenii interdependente:

- arhitectura sistemelor,
- algoritmica,
- limbaje de programare paralelă/ distribuită/ concurentă.

În lucrarea [DZI01] sunt prezentate cele trei domenii şi interdependenţa dintre ele, precum şi un minidicţionar de noţiuni şi concepte de paralelism.

În acest minidicţionar sunt explicaţi sumar principalii termeni şi sintagmele ce exprimă concepte utilizate în limbajul şi metalimbajul calcului paralel şi distribuit, astfel ca şi cititorul neiniţiat să poată înţelege manualul de faţă fără a consulta alte materiale. Ordinea de prezentare a terminologiei nu este cea alfabetică, ci am încercat o anumită ordonare logică, aşa cum am crezut că ar trebui să parcurgă un începător minidicţionarul nostru pentru a se familiariza cât mai rapid cu problematica extrem de complexă a calculului paralel şi distribuit.

Pentru explicaţii suplimentare necesare aprofundării am utilizat următoarele notaţii pentru trimiteri:

- *paranteze drepte* pentru indicarea sursei din bibliografie: ex. [GRI00] indică lucrarea prefixată astfel în bibliografie;
- *paranteze rotunde* pentru a indica numărul de ordine din tabelul minidicţionarului: de exemplu, (1) se referă la expresia din tabel cu numărul curent 1, adică la *paralelism*.

[19] Această anexă apare şi în lucrarea [DZI06], în ideea ca cele două lucrări să poată fi mai inteligibile şi într-o utilizare independentă.

Expresia în română // engleză	**Descriere:**
1. Paralelism //parallelism [PFA95] [GRI00]	**"Paralelism" (P)**: = termen generic pentru desemnarea unui ansamblu de tehnici şi procedee de creştere a performanţelor unui sistem informatic prin exploatarea simultană a mai multor resurse similare sau nu (în special elemente de procesare sau procesoare interconectate în diverse moduri în acelaşi sistem fizic sau computere cuplate într-o reţea eterogenă etc.); **Sinonime:** concurenţă, simultaneitate, suprapunere; **Paralelismul** se manifestă la nivel de: **-arhitectură de procesoare elementare şi unităţi centrale** ale căror redundanţe permit procesări simultane prin tehnica **pipelining** **-programare** care permite exprimarea concurenţei (colaborării între procese); -**limbaje** care oferă instrumente de exprimare a proceselor simultane; -**compilatoare**, care extrag paralelismul din programe pentru a-l adapta arhitecturilor; -**algoritmii** de calcul adaptaţi arhitecturilor paralele**.** **Conceptele de paralelism** sunt numeroase şi diverse: (2)sursă de paralelism; (7)extragerea paralelismului; (10)granularitatea paralelismului; (11)nivele de paralelism.; (12)arhitecturi paralele.
2. Sursele paralelismului	Sursele p. sunt:(3)paralelism de control (concurent); (4)paralelismul datelor; (5)paralelismul de flux; (6)paralelism spaţial.
3. Paralelism de control (concurent) // concurrency parallelism	Paralelismul care autorizează **simultaneitatea** prelucrării proceselor, ţinând cont de resursele acestora; Termenii **paralel** şi **concurent** provoacă încă multe discuţii în literatura de specialitate (de cele mai multe ori se utilizează ca sinonime, dar depinde şi de contextul de utilizare): După **Ben-Ari** [BEN90] **calculul concurent** înseamnă definirea unui număr de activităţi de calcul ce

	se pot executa simultan, pe un număr infinit de procesoare. Conform acestei definiţii **Dan Grigoraş** [GRI00] consideră că cei doi termeni sunt echivalenţi, dar totuşi **concurent** ar avea un grad mai mare de generalitate în anumite contexte decât paralel; **S.A.Wiliams** [Wil90] spune că există multe similitudini între programarea paralelă şi programarea concurentă asociată cu sistemele de operare şi sistemele în timp real, neexistând o delimitare clară între **paralel** şi **concurent,** prin urmare se folosesc după preferinţă (adică sunt sinonime), considerând că **paralel** este totuşi mai general, punct de vedere se manifestă şi în utilizarea expresiei **paralelism concurent,** care ar fi un pleonasm, dacă concurenţa ar fi mai generală decât paralelismul; De fapt cei doi termeni fac parte din metalimbajul asociat CP şi au definiţii imprecise, o definire matematică precisă a celor două noţiuni ar clarifica lucrurile; evident, se poate accepta şi un punct de vedere de compromis, adică cele două sfere noţionale au o mare parte comună, nefiind totuşi una inclusă în cealaltă.
4. Paralelismul datelor //data parallelism	**P.** în care aceeaşi prelucrare este aplicată unor date diferite, fie printr-un **pipeline** de instrucţiuni vectoriale, fie printr-o arhitectură paralelă de tip **SIMD.**

5. Paralelismul de flux
// data flow parallelism
// flux parallelism
	care efectuează o aceeaşi prelucrare asupra unui flux de date, de exemplu căutare **multicriteriu** într-o bază de date, printr-un operator **pipeline** din care fiecare secţiune triază după unul din criterii cu viteza fluxului de date ce provine de pe disc

6. Paralelism spaţial // spatial parallelism	**P.** care efectuează o descompunere a datelor reprezentative în spaţiu, în regiuni prelucrate de procesoare diferite (calcul numeric, prelucrare de imagini)
7. Extragerea paralelismului	Paralelism implicit (8); paralelism explicit (9)
8. Paralelism implicit sau extras // implicit	**P.** în care compilatorul extrage posibilităţile de paralelism ale programului şi le adaptează arhitecturii, cum ar fi paralizarea executării buclelor de programe când iteraţiile succesive sunt independente, sau

parallelism	reordonarea instrucţiunilor pentru utilizarea optimă a arhitecturilor RISC superscalare.
9. Paralelism explicit // explicit parallelism	**P.** definit de către programatorul-utilizator

10. Granularitatea paralelismului, aplicaţiei, sistemului ;
// granularity
// grain size
	În paralelismul de concurenţă [Mor98+] avem următoarele grade de granularitate:
	-inferior (la nivel de programe);
	-mediu (la nivel de procese);
	-ridicat (la nivel de *threads*, adică procese simple).
	După Schwatz [Qui88] granularitatea se mai referă şi la numărul de procesoare dintr-un sistem paralel:
	- brută (zeci şi sute de procesoare);
	- fină (mii şi zeci de mii de procesoare).
	În [Gri00] se defineşte *granularitatea aplicaţiei* ca valoarea minimă a granularităţii pentru toate activităţile paralele componente (proces, thread)= dimensiunea minimă, exprimată în numărul de instrucţiuni, dintr-o unitate secvenţială. Valoarea minimă sub care performanţa sistemului paralel scade semnificativ este numită *granularitatea sistemului.*

11. Nivele de paralelism	**Nivel scăzut** // low level parallelism**:=** p. la nivelul procesorului elementar (arhitectură cu instrucţiuni pipeline, superscalară, sistolică); **Nivel înalt** // high level parallelism**:=** care pune în funcţiune mai multe procesoare elementare
12. Arhitectura paralelismului	**P. centralizat (puternic cuplat)** // centralized (highly coupled) parallelism := permiţând o vedere şi un control global prin partajarea unei resurse comune între procesoare, ca în multiprocesoarele cu memorie comună; **P. distribuit // distributed parallelism**:= unde procesoarele independente nu dialoghează decât prin mesaje transmise prin reţea, ca în cazul unui hipercub; **p. masiv** // massive parallelism**:=** mai multe procesoare universale sau specifice (arhitecturi celulare şi reţele neuronale).

13. Paralelizarea unui program/ algoritm	Adaptarea unui program / algoritm conceput pentru un calculator **monoprocesor** pentru a fi executat pe calculator **multiprocesor**, adică izolarea unor subcomponente care pot fi executate simultan.
14. Clasificarea Flynn	Flynn clasifică arhitecturile în funcţie de modul de prelucrare a seturilor de instrucţiuni şi a seturilor de date în: **SISD** (Single Instruction stream-Single Data stream); **SIMD** (Single Instruction stream-Multiple Data stream); **MISD** (Multiple Instruction stream-Single Data stream); **MIMD** (Multiple Instruction stream-Multiple Data stream);
15. Arhitectură masiv paralelă // large parallel architecture	Arhitectura sistemului central implicând un mare număr de procesoare (de ordinul sutelor) elementare, interconectate în diverse structuri topologice (magistrală, inel, matrice, arborescentă, plasă/grilă, hipercub etc.) care lucrează în colaborare la rezolvarea uneia sau mai multor probleme;
16. Arhitectură multiprocesor // multiprocessor architecture	Arhitectură a sistemului central / unităţii centrale care include mai multe procesoare elementare de prelucrare;
17. Arhitectură pipeline // pipeline architecture	Arhitectură a operatorului sau procesorului care permite un paralelism înlănţuit, de tip conductă / bandă rulantă; operatorul este împărţit în sectoare traversate succesiv de date, după frecvenţa ceasului, eliberând secţiunea pentru operaţia următoare, astfel încât o nouă operaţie poate fi iniţializată la fiecare perioadă de ceas; Un pipeline de instrucţiuni permite procesoarelor RISC prelucrarea unui debit de instrucţiuni într-o bătaie de ceas, atunci când o instrucţiune necesită cinci operaţii: căutarea instrucţiunii în cache, decodarea, căutarea operanzilor, execuţia, pregătirea rezultatului; **Pipeline (canal de prelucrare paralelă):=** o „linie de asamblare" care creşte substanţial viteza de citire, executare şi scriere a instrucţiunilor. Utilizată de mult în UNIX, structura pipeline a fost inclusă în Intel 80486 şi permite prelucrarea unei instrucţiuni în fiecare perioadă

	de ceas. Microprocesorul Intel Pentium conţine două astfel de structuri, una pentru date şi alta pentru instrucţiuni; **Pipelining := prelucrare paralelă.**
18. Procesor CISC (procesor cu set complex de instrucţiuni) // Complex Instruction Set Computer (CISC)	Arhitectură de procesor în care a existat tendinţa de a mări setul de instrucţiuni cablate, evoluţia lor orientându-se spre execuţia dinamică, care se bazează pe trei concepte: predicţia salturilor, analiza dependenţelor de date şi execuţia speculativă. **Ex.:** microprocesoarele Intel Pentium Pro.[Gri00]
19. Procesor RISC (procesor cu set redus de instrucţiuni) // Reduced Instruction Set Computer (RISC) [Gri00]	O alternativă la CISC; Cele mai populare arhitecturi de procesoare pentru CP în care obţinerea performanţelor este urmărită prin utilizarea unui set redus de instrucţiuni cablate ce permite prelucrarea pipeline a instrucţiunilor de executat; Obiectivul arhitecturilor RISC ideale este realizarea unei instrucţiuni la fiecare bătaie de ceas, cea ce implică străduania de alimentare cu instrucţiuni şi date a memoriilor cache, evitarea aşteptărilor în pipeline-rile de instrucţiuni în caz de racordări, evitarea timpilor de schimbare de context în cazul apelurilor de proceduri; Premergătoarele RISC-ului au fost IBM 801, SPARC, MIPS.
20. Arhitectură superscalară // superscalar architecture [GRI00]	extensie RISC în care mai multe instrucţiuni de tip diferit sunt decodificate simultan, apoi lansate în paralel, fiecare către operatorul corespunzător: de exemplu, o operaţie în virgulă fixă, o operaţie în virgulă mobilă şi o operaţie cu memoria; Compilatorul este cel care se ocupă cu reorganizarea instrucţiunilor pentru obţinerea celei mai bune utilizări pipeline şi a unui maxim **paralelism superscalar**, garantând coerenţa programului.
21. Hipercub // hypercub	Calculator paralel cu cuplare slabă şi memorie distribuită la un număr de procesoare ce uneori depăşeşte 65 000 (de ex, CM 1).

22. Multiprocesor // multiprocessor	Denumire dată unui calculator sau unui sistem de calcul care are mai multe procesoare de prelucrare
22. Multiprocesor simetric // symetric multiprocessor	Multiprocesor în care gestiunea sarcinilor sistem este partajată de către toate procesoarele, spre deosebire de multiprocesorul de tipul master/slave, unde master-ului (stăpânului) îi revine gestiunea sistemului de exploatare, sclavii fiind puşi doar să execute , Ex. **Sequent Symmetry**
23.Multiprogra-mare//multipro-gramming	**Concept de natură hardware** prin care mai multe programe stocate în memoria unui calculator sunt executate cu întreruperi, cu partajarea timpului:
24. Multitasking	Executarea a mai multor programe în acelaşi timp pe un sistem de calcul Execuţia unui program presupune utilizarea mai multor resurse sistem: în afara efectuării calculelor pe procesorul sistemului, un program poate transfera date într-un fişier implicând o "comunicaţie" cu discurile fixe, poate trimite date la imprimantă pentru a fi tipărite, sau poate aştepta reacţia umană furnizată prin intermediul mouse-ului sau tastaturii. Toate aceste operaţii sunt realizate la viteza la care perifericul respectiv poate funcţiona fizic, viteza net inferioara celei la care funcţionează procesorul. **Execuţia programelor prin întrepătrundere (multitasking)** este o tehnica ce permite utilizarea la maximum a procesorului, prin planificarea spre execuţie a altor programe pe perioadele de inactivitate ale acestuia. Pentru a evita situaţiile în care un program acaparează integral procesorul (prin evitarea dialogului cu periferia), tehnica "multitasking" prevede ca după o cuantă maxima de timp, un astfel de program sa fie suspendat permiţând astfel şi execuţia altor programe. Un alt avantaj al unui astfel de mecanism constă în robusteţea conferită sistemului. Astfel, indiferent dacă programul utilizator ce se executa la un moment dat e corect sau nu, prin "prelevarea forţată" a sa (suspendarea

	din execuţie) controlul este redat sistemului de operare, permiţând astfel terminarea sau "uciderea" acelui program. Un sistem de operare care funcţionează pe principiul "multitasking"-ului (de ex. UNIX) creează iluzia execuţiei simultane a mai multor programe - o execuţie în acest caz este văzută ca fiind un *proces* distinct. Bazându-se pe această iluzie, un astfel de sistem de operare prevede mecanisme de "comunicaţie" şi "sincronizare" între procese, şi deci dezvoltarea aplicaţiilor "concurente". Atâta timp însa cât este utilizat un singur procesor, nu se pot aştepta creşteri spectaculoase de performanţă. O creştere există, dar ea se datorează utilizării mai eficiente a acestui unic procesor, şi nu creşterii puterii de calcul a sistemului în ansamblu. Pe sistemele real paralele (distribuite sau nu), procesele ce compun un program sunt executate efectiv în acelaşi timp. În plus, particularităţile constructive ale sistemului paralel pot reduce mulţimea mecanismelor disponibile (de exemplu absenţa memoriei partajate în mediile distribuite), impunând o modelare a mecanismelor absente. Existenţa mai multor procesoare oferă o creştere categorica a performanţelor, dar induce şi probleme complexe în controlul execuţiilor programelor. O astfel de problemă constă în decizia plasării proceselor pe procesoarele fizice disponibile. Aceasta decizie poate aparţine programatorului, dacă limbajul utilizat permite acest lucru, fiind vorba astfel de un p**aralelism explicit.**
25. Aplicaţie multifir // multithreaded application	Un program care poate rula în acelaşi timp două sau trei fire (//threads), porţiuni independente de program; Avantajul divizării în fire (threads) constă în posibilitatea oferită sistemului de operare de a decide care fir are prioritate maximă de execuţie.
26. Multiprelucrare // multiprocessing	**Concept de natură software**, un mod de exploatare a unui calculator în care mai multe sarcini sunt executate simultan de către mai multe procesoare printr-o planificare, când unul aşteptă încă date de intrare de care are nevoie pentru continuarea execuţiei, unitatea

	centrală se ocupă de un alt program; astfel că se creează impresia de paralelism în execuţie, adică în acelaşi interval de timp sunt terminate mai multe programe.
27. Sincronizare // synchronization	Mecanism utilizat în execuţii paralele ce permite introducerea de întârzieri în execuţia unui program, necesare unor constrângeri impuse de algoritm sau de structura intimă a procesorului; Sincronizarea se petrece în primul rând la operaţiile indivizibile la nivelul procesorului, execuţia cărora trebuie terminată înaintea oricărei alte utilizări a procesorului (în acest caz nu se permit intercalări în execuţie, deci e nevoie de sincronizare); Există două tehnici importante de sincronizare: **mecanismul** `fork-join` şi utilizarea **mecanismelor de control:** `cobegin-coend, parbegin-parend;` Primul este un mecanism mai puternic (permite crearea dinamică a proceselor, pe când al doilea introduce o structură de control cu o singură intrare şi o singură ieşire);

28. Mecanismul fork-join	Mecanism utilizat în limbajele de programare paralelă (concurentă) pentru specificarea execuţiei concurente (paralele), adecvat pentru aplicaţii în care nici o funcţie majoră a aplicaţiei nu necesită rezultate de la o altă funcţie a acesteia, adică sunt independente şi pot fi evaluate simultan;
	variante ale sale se utilizează în Unix pentru specificarea proceselor concurente;
	instrucţiunea `fork Q,` apărută într-un proces P, declanşează execuţie paralelă a procesului Q În acelaşi timp ce se execută şi procesul P în care apare fork-ul;
	instrucţiunea `join Q` apărută într-un proces P, recombină două procese într-unul singur ca în următorul exemplu:
	`program P begin....fork Q...join Q...`
	`program Q begin...end; end.`
	pentru exemplul de mai sus execuţia lui Q este începută când se ajunge la `fork Q` din P, moment din care P şi Q se execută simultan fie până când P execută `join Q,`

	fie până când Q se termină. Dacă Q nu este terminat şi P execută `join Q`, el va aştepta până când Q se va termina şi numai apoi îşi va continua execuţia. Dacă însă Q se termină înainte ca p să execute `join Q`, atunci execuţia acestei comenzi nu va avea nici un efect asupra lui P, acesta continuându-şi execuţia cu instrucţiunile ce urmează după `join Q`**.**
29. Mecanismele de control: cobegin-coend/ parbegin-parend	Sunt mecanisme pentru specificarea execuţiei concurente/ paralele a instrucţiunilor din lista cuprinsă între început şi sfârşit; de exemplu în programul `program P begin...cobegin Q, R coend S...end`. instrucţiunile Q şi R vor începe să fie executate simultan când se ajunge la `cobegin`, iar R va fi pornită doar după ce ambele instrucţiuni sunt terminate (dacă R se termină mai repede, se aşteptă şi după terminarea lui Q, adică după terminarea tuturor instrucţiunilor din listă); `parbegin-parend` are acelaşi efect.
30. Sistem multiuser //multiuser system	Sistem cu mai mulţi utilizatori; sisteme de operare multiuser: UNIX, Linux etc.
31. Conceptul de program stocat // stored program concept	Este un concept utilizat în programarea serială în care un program se stochează în memorie împreună cu datele, urmând ca execuţia instrucţiunilor din program să nu se facă neapărat în ordinea secvenţială de scriere a programului ci să fie executată acea instrucţiune care are disponibile datele necesare execuţiei.
32. Transputer // TRANSistor and comPUTER	Numit şi **calculator într-un cip**, este un microcircuit VLSI cu memorie adiţională, microprocesor, posibilităţi de comunicaţie şi interconectare; Construit pe baza unei arhitecturi RISC (INMOS, 1985), este strâns legat de limbajul Occam şi constituie elementul de bază al sistemelor multiprocesor masiv paralele

33. Limitare de tip von Neumann // von Neumann bottleneck	Limitarea vitezei de prelucrare constatată de însuşi John von Neumann în arhitecturile care-i poartă numele, conform căreia un calculator de acest tip pierde mai mult timp cu căutarea datelor în memorie decât cu prelucrarea propriu –zisă.
34. Limbajul Occam	Descendent direct din CSP, **Occam** (numele filozofului englez din secolul XIV de la care a rămas *principiul simplităţii* în structura entităţilor, *briciul lui Occam*); un limbaj simplu, cu mecanisme puternice de implementare a algoritmilor paraleli pe sisteme bazate pe transputere şi sisteme distribuite.
35. Limbajul TIMP / TIMişoara Parallel (TIMP)	Limbaj concurent destinat calculatoarelor vectoriale, elaborat de un colectiv de la Universitatea Politehnica din Timişoara
36. Limbajul PVM / Parallel Virtual Machine	Limbaj destinat calculatoarelor paralele virtuale configurate dintr-o mulţime de calculatoare conectate într-o reţea distribuită. Un pachet de programe destinat programării calculatoarelor paralele virtuale.
37. Sistem distribuit de prelucrare // distributed processing sistem	Un sistem de calcul proiectat pentru mai mulţi utilizatori, în maniera client / calculator, care oferă fiecărui utilizator un calculator complet funcţional: În domeniul PC prelucrarea distribuită are forma reţelelor locale (LAN), în care PC-urile unui departament sunt legate prin conexiuni de mare viteză, resursele scumpe putând avea o utilizare în comun; se deosebeşte de sistemele cu mai mulţi utilizatori.
38. Sistem centralizat de prelucrare // centralized processing sistem	Un sistem de calcul proiectat pentru mai mulţi utilizatori, în maniera **mainframe** (un calculator de mare capacitate necesar unei organizaţii departamentale, cu mai multe terminale şi cu o bază de date comună şi partajarea unor resurse scumpe în comun); deosebirea esenţială dintre sistemele centralizate şi cele distribuite este că puterea de calcul se realizează în primele printr-un calculator de mare capacitate, iar în cele distribuie prin mai multe calculatoare de capacităţi mai mici.

39. Scalabilitatea (sistemului paralel) / scalability (of parallel system) [GRI00], p. 68	Se referă la posibilitatea de a asigura creşterea eficienţei prin creşterea numărului de procesoare , în ipoteza că programul prezintă un potenţial suficient de mare de paralelism şi o granularitate suficient de mare; Dacă se obţine o creştere liniară a accelerării, se spune că sistemul este *scalabil liniar*.
40. Procese asincrone // asynchronous processes	Procese concurente care nu fac schimb de date direct, după perioade prestabilite (ca la procesele sincrone), sau prin mesaje la cerere (ca la procesele sincronizabile), ci fac schimb de date prin intermediul memoriei comune în orice moment al desfăşurării proceselor
41. Sistem de calcul în timp real // real time computing system	Este un *sistem de calcul online* (sistem în care datele de intrare sunt introduse direct de la locul de generare în sistemul de calcul, iar datele de ieşire sunt transmise direct la locul de utilizare), ataşat unui fenomen în desfăşurare reală, astfel încât intervalul de timp scurs de la începerea preluării datelor de intrare şi până la obţinerea rezultatelor, să fie suficient de scurt pentru ca desfăşurarea ulterioară a fenomenului să mai poată fi influenţată: Elementul esenţial ce caracterizează un *sistem de calcul în timp real* este *timpul de răspuns* al aplicaţiei, timp care de multe ori nu poate fi suficient de micşorat decât prin mijloace sau procedee de calcul paralel
42. UNIX	Sistem de operare multitasking şi multiuser utilizabil atât pe un PC cât şi pentru reţele de PC-uri; are multe variante specializate: **Linux** pentru configurarea unui calculator paralel virtual, care poate fi programat în **limbaj PVM**, **Parix**, pentru sistemele bazate pe **transputere**; **Dynix,** pentru sistemul **Sequent Symmetry** ş.a.
43. / Grand Challenge	Categorie de probleme de mari dimensiuni, cu caracter ştiinţific sau ingineresc, de o asemenea complexitate încât nici un cercetător sau institut de cercetări nu le poate rezolva prin eforturi individuale; astfel de probleme sunt ţinte ale calculului paralel şi distribuit.

44. Algoritmi paraleli **// parallel algorithms**	Există mai multe modalităţi de obţinere a algoritmilor paraleli: -prin multiplicare şi izolare, adică fiecare procesor execută în mod independent acelaşi program, fiind izolat de restul procesoarelor (acelaşi program, date locale diferite, fără comunicare între procesoare); -prin paralelism spaţial, fiecare procesor executând acelaşi program dar asupra unor date locale rezultate în urma unei divizări a domeniului datelor (acelaşi program, date locale diferite, comunicare pentru datele de la frontiera domeniului); -prin paralelism algoritmic, fiecare procesor fiind responsabil de o anumită parte din program, toate datele trecând prin fiecare procesor (secvenţe de program diferite, date globale, comunicare prin intermediul unei memorii comune); algoritmii paraleli pot fi numerici (se execută calcule cu numere), sau nenumerici (se execută operaţii de căutare, sortare ş.a.).
45. Internet / *The Net*	Cuvântul „**internet**” provine din concatenarea prescurtărilor a două cuvinte englezeşti, ***inter****connected* (*interconectat*) şi ***net****work* (*reţea*) şi desemnează o reţea de mari dimensiuni formată prin interconectarea mai multor reţele autonome eterogene. Astfel, substantivul comun „**internet”** (cu minusculă) desemnează în general o reuniune de reţele, văzută ca o reţea unitară, împreună cu informaţia şi serviciile care sunt oferite utilizatorilor prin intermediul acestei reţele (Web, E-Mail, FTP etc.). Cea mai mare dintre reţele de tip internet este numită **Internet** (nume propriu, scris cu majusculă), adică super-reţeaua mondială unică de computere, interconectate prin protocolul IP Atât **Internetul** sau **The NET (Reţeaua),** cum i se mai spune în lume, cât şi alte reţele mai mici de tip *internet* sunt exemple de *sisteme informatice distribuite*.

46. Intranet/	Un **intranet** este o reţea închisă sau o sub-reţea dintr-un internet sau chiar din Internet care este administrată autonom şi pentru care exista un sistem de securitate local. Un *intranet* poate format din mai multe reţele de tip *Local Area Network* (LAN), legate între ele prin anumite sisteme de comutare. Un intranet poate fi conectat la Internet printr-un *router* , care permite utilizatorilor din intranet să utilizeze servicii ca Web, FTP sau EMAIL. De asemenea permite utilizatorilor din exterior (din Internet) să acceseze servicii pe care le pune eventual la dispoziţie intranetul. Pentru a se proteja de diferite atacuri malitioase, sunt utilizate soft-uri de tip *firewall*, care previn utilizatorul că anumite mesaje neautorizate încearcă să intre sau să plece. Un firewall este implementat să filtreze anumite mesaje conform unor criterii, de exemplu el permite trecerea doar a mesajelor legate de poşta electronică.
47. Calcul mobil/ *Nomadic computing*	În lumea sistemelor informatice distribuite un rol deosebit îl au în prezent *dispozitivele miniaturizate* şi *reţelele wireless*. De exemplu, cu ajutorul unui laptop sau chiar a telefonului mobil, printr-o conexiune de tip wireless ne putem conecta aproape de pretutindeni la intranetul „home” şi putem utiliza resursele de acolo (de pe calculatorul de acasă sau de la serviciu). Putem vorbi astfel de un *calcul mobil* (*nomadic computing*).

48. Computere-omniprezente/ *Ubiquitous computing*	*Ubiquitous computing* promovează ideea aparent opusă**,** „*computere omniprezente*”, adică să existe computere conectate la internet în locuri în care există indivizi obligaţi să stea un timp mai lung sau mai scurt (imobilizaţi acasă sau în spitale, în staţiuni turistice, în gări şi aeroporturi etc.), pe care indivizii le pot accesa pentru comunica sau pentru a accesa anumite informaţii din exterior. De exemplu, de la calculatorul de acasă conectat la Internet, putem accesa diverse informaţii de la serviciu sau putem citi presa din Bucureşti sau Londra, sau putem coresponda prin email sau online cu orice persoană din lume care dispune de aceleaşi facilităţi. In afară de laptopuri şi de telefoanele mobile performante, amintim imprimantele inteligente, ceasurile inteligente, PDA (personal digital assistant), camere video digitale etc., care contribuie la dezvoltarea tot mai expansivă a calculului nomadic.
49. Sistem puternic cuplat/ *Tightly coupled system* **50. Sistem slab cuplat/** *Loosely coupled system*	Din punct de vedere al structurii hardware şi a tipului de conexiune, sistemele care cuprind mai multe procesoare pot fi: • *puternic cuplate* (conectate la nivel de memorie, ex. multiprocesoare); • *slab cuplate* (conectate la nivel de reţea de calculatoare, ex. multicalculatoare). *Sistemele puternic cuplate* sunt sisteme în care mai multe procesoare partajează aceeaşi memorie internă şi folosesc acelaşi ceas intern. De exemplu sistemele din clasa MIMD sunt sisteme puternic cuplate (multiprocesoarele şi calculatoarele paralele masive). *Sistemele slab cuplate* sunt sisteme în care fiecare procesor are propria memorie şi propriul ceas intern.

51. Cluster („ciorchine") Se mai utilizează uneori denumirea **Farm („fermă")** pentru desemnarea aceluiaşi concept	**Clusterul** este un tip de sistem distribuit ce permite calculul paralel, format fizic dintr-o rețea de cel puțin două procesoare, numite **stații de lucru** (care pot fi calculatoare complete, PC-uri, supercalculatoare, calculatoarele vectoriale, multiprocesoare, MPP), care pot fi folosite şi de sine stătător, interconectate într-o rețea, fiind utilizat ca o resursă de calcul integrată şi singulară. În practică se utilizează două tipuri de clustere: **clustere dedicate** (formate din procesoare omogene) şi **clustere de întreprindere** (formate din procesoare neomogene). Un cluster are, în mod iluzoriu, pentru utilizator **o imagine de sistem unic -SSI (Single System Image).** Aceasta este impresia utilizatorului, că are acces la un sistem unic cu resurse multiplicate, cu control unic asigurat prin intermediul unei singure interfețe. Sistemul este simetric, în sensul că un serviciu poate fi solicitat de pe orice nod, iar accesul la resurse este transparent. Astfel, clusterul pare la fel de uşor de folosit ca un PC. Elementele clusterului sunt văzute din afară ca fiind anonime şi interschimbabile.
52. Bază de date distribuită //distributed database	Este o colecție de date distribuită în mai multe locații fizice, controlată de un sistem de management al bazei de date în care dispozitivele de stocare nu sunt ataşate în totalitate de o singură unitate centrală de prelucrare obişnuită. Datele pot fi stocate în mai multe calculatoare plasate în aceeaşi locație fizică sau într-o rețea. O bază de date distribuită este distribuită în partiții/fragmente separate. Fiecare partiție/fragment a unei baze de date distribuită poate fi reprodusă (adică eşecuri redundante, cum ar fi o matrice de hard-disk-uri). În afară de replicarea şi fragmentarea bazelor de date distribuite, există multe alte modele de tehnologii ale bazelor de date distribuite. De exemplu, autonomia locală, şi tehnologii sincrone şi asincrone ale bazelor de date distrbuite. Aceste implementări ale tehnologiilor poate, şi în mod sigur, depinde de nevoile beneficiarului şi sensibilitatea/confidențialitatea datelor care vor fi stocate în baza de date.

BIBLIOGRAFIE

[AHI88] Ahituv, N., and Brman O., *Operations Management of Distributed Service Networks. A Practical Quantitative Approach,* Plenum Press, New York, 1988

[AND01] Andreica, A., Bota, F., *Informare şi comunicare în reţele de calculatoare*, Editura Fundaţiei pentru Studii Europene, Cluj-Napoca, 2001

[ATT98] Attiya, H., Welch, J., *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, McGraw-Hill Publishing Company, London, 1998.

[BÂR96] Bârsan, C., Fortis, F., Domocos, V., Iurian, S., Zaharia, M., *Aspecte ale elaborării algoritmilor distribuiţi (culegere de texte),* Disco, Proiect TEMPUS DISCO: "Development in Romania of short-term higher education in computing, centered on distributed processing and its applications", TEMPUS S_JEP-07101-94, 1996.

[BEN90] Ben-Ari, M., *Principles of concurrent and distributed programming*, Prentice Hall, 1990.

[BOI97] Boian, F. M., *Programarea distribuită în Internet. Metode şi aplicaţii*, Editura Albastră, Cluj-Napoca, 1997.

[BRO97] Brookshear, J.G., *Introducere în Informatică*, Ed. Teora,1998. Titlul original: Computer Science – An Overview, Fiith Edition, Addision Wesley Longman, Inc, 1997

[COC88] Coculescu, L., Cristea, V., Finta, I., Patriciu, V., Pilat, F.: *Proiectarea sistemelor teleinformatice*, Ed. Militară, Bucureşti, 1988

[COU01] Coulouris, G., Dollimore, J., Kindberg, T., *Distributed Systems*, Third Edition, Addison-Wesley, Pearson Education Ltd., 2001

[CRE92] Cristea, V., Ţăpuş, N., Moisa, T., Damian, V., *Reţele de calculatoare*, TEORA, Bucureşti, 1992.

[CRI91] Cristian, F., *Understanding Fault-Tolerant Distributed Systems*, Communications of the ACM, Vol. 34, No 2, 1991.

[DAV83] Davies, D. W., Barber, D. L. A., Price, W. L., Solomonides, C. M., *Teleinformatica. Reţele de calculatoare şi protocoalele lor*, Ed. Tehnică, Bucureşti, 1983.

[DUM88] Dumitrescu, V., Paiu, O., Vlăduţ, T., Ştefănescu, V., *Iniţiere în informatica distribuită,* Editura Tehnică, Bucureşti, 1988.

[DZI01] Dziţac, I. *Calcul paralel,* Ed. Univ. din Oradea, 2001

[DZI02] Dziţac, I., *Procedee de calcul paralel şi distribuit în rezolvarea unor ecuaţii operatoriale*, Teză de doctorat, Universitatea "Babeş-Bolyai", Cluj-Napoca, 2002.

[DZI03] Dziţac, I., Laslo, E., *Programarea paralela utilizând PVM*, Ed. Univ. din Oradea, 2003

[DZI06] Dziţac, I., Modovan, G., *Sisteme distribuite: Modele informatice*, Ed. Univ. Agora, 2006

[FIL04] Filip, F.G., *Sisteme suport pentru decizii*, Ed. Tehnică, 2004

[GÂR84] Gârbacea, L., Moldovan, G., *Distributed data bases*, University of Cluj-Napoca, Faculty of Mathemtics, Research Seminaries, Preprint 5, 1984.

[GRI00] Grigoraş, D., *Calculul paralel- De la sisteme la programarea aplicaţiilor,* Ed. Agora, 2000.

[GUR86] Guran, M., Filip, F. G., *Sisteme ierarhizate în timp real cu prelucrare distribuită a datelor*, Editura Tehnică, Bucureşti, 1986.

[JAI88] Jain, R., Ramakrishnan, K. K., *Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals, and Methodology*, Proceedings of the IEEE Computer Networking Symposium, Washington, D. C., April, 1988.

[JAI90] Jain, R., *Congestion Control in Computer Networks: Issues and Trends*, IEEE Network Magazine, May, 1990.

[JAI92] Jain, R., *Myths About Congestion Management in High-Speed Networks*, IFIP TC6 4th Conference on Information Networks and Data Communication, Helsinki, March, 1992.

[JUR01] Jurcă, I.: *Programarea reţelelor de calculatoare*, Editura de Vest, Timişoara, 2001

[KLE85] Kleinrock, L., *Queuing Systems Theory*, John Wiley & Sons, New York, 1975.

[LAM78] Lamport, L.: Time, *Clocks, and Ordering of Events in a Distributed System*, Comm. ACM, 1978, 21,7

[LEA76] Lee, A. M., *Teoria aşteptării cu aplicaţii*, Editura Tehnică, Bucureşti, 1976.

[MAA76] Mahmoud, S. A., Riordon, J. S., *Optimal Allocation of Resources in Distributed Information Networks,* ACM Transactions Database Systems, March, 1976.

[MCQ74] McQuillan, J. M., *Adaptive routing algorithms for distributed computer networks*, Bolt, Beranek and Newman, 1974.

[MIH73] Mihoc, G., Ciucu, G., Muja, A., *Modele matematice ale aşteptării*, Editura Academiei, Bucureşti, 1973.

[MOL84] Moldovan, G., *Reorganisation of a Distributed Data Base*, University of Cluj-Napoca, Faculty of Mathematics, Research Seminries, Preprint 5, 1984.

[MOL85a] Moldovan, G., *Bazele informaticii II*, Lito. Univ. "Babeş-Bolyai", Cluj-Napoca, 1985.

[MOL85b] Moldovan, G., *O problemă privind bazele de date distribuite*, A VIII-a consfătuire de lucru şi schimb de experienţă a lucrătorilor din unităţile de informatică ale M.E.Î., Centrul de Calcul Coordonator, Institutul de Construcţii Bucureşti, Centrul de Calcul, Lucrări prezentate, Constanţa, 1985.

[MOL86] Moldovan, G., *O problemă de optimizare într-un sistem de baze de date distribuite*, Universitatea din Cluj-Napoca, Facultatea de Matematică, Centrul de Calcul Electronic, Simpozion "Informatica şi aplicaţiile sale", Zilele Academice Clujene, Cluj-Napoca, 1986.

[MOL87a] Moldovan, G., Damian, S., *Asupra unei probleme de optimizare pentru baze de date distribuite*, Lucrările sesiunii ştiinţifice a Centrului de Calcul al Universităţii Bucureşti, Universitatea Bucureşti, Bucureşti, 1987.

[MOL87b] Moldovan, G., Damian, S., *Sur la redistribution des bases des données dans un réseau d'ordinateurs*, Babeş-Bolyai University, Faculty of Mathematics and Physics, Seminar on Computer Science, Nr. 9, 1987.

[MOL87c] Moldovan, G., Damian, S., *On some Generalizations of an Optimization Problem for Distributed Data Bases*, Studia Universitatis Babeş-Bolyai, Mathematica, XXXII, Nr. 3, Cluj-Napoca, 1987.

[MOL87d] Moldovan, G., Damian, S., *Baze de date relaţionale distribuite*, Rezumatele comunicărilor celui de-al IV-lea colocviu de sisteme, modele, informatică şi cibernetică, Academia de Studii Economice, Bucureşti, 1987.

[MOL87e] Moldovan, G., *O problemă de optimizare a redistribuirii bazelor de date*, A VIII-a consfătuire de lucru şi schimb de experienţă a lucrătorilor din unităţile de informatică ale M.E.Î., Centrul de Calcul Coordonator, Institutul de Construcţii Bucureşti, Centrul de Calcul, Lucrări prezentate, Constanţa, 1987.

[MOL88a] Moldovan, G., Damian, S., *On an Optimization Problem for Distributed Data Bases*, Analele Universităţii Bucureşti, Matematică-Informatică, XXXVII, Nr. 2, Bucureşti, 1988.

[MOL89a] Moldovan, G., Damian, S., *A Local Optimization Problem for Data Bases Redistribution in a Computer Net*, Studia Universitatis Babeş-Bolyai, Mathematica, XXXIV, Nr. 3, Cluj-Napoca, 1989.

[MOL89b] Moldovan, G., Damian, S., *O problemă de optimizare locală pentru redistribuirea bazelor de date într-o reţea de calculatoare*, Universitatea Al. I. Cuza, Lucrările celui de-al VII-lea colocviu de informatică INFO-IAŞI, Iaşi, 1989.

[MOL89c] Moldovan, G., Râp, I., *Asupra unei probleme de aşteptare în reţele de calculatoare*, Lucrările Conferinţei de Matematică Aplicată şi Mecanică, 21-23 octombrie 1988, Cluj-Napoca, 1989.

[MOL92a] Moldovan, G. (director de contract), *Probleme de optimizare în sisteme de baze de date distribuite*, Contract de cercetare nr. 531/80/1989 – beneficiari: C. N. Ş. T. Bucureşti (1989) şi Ministerul Învăţământului Bucureşti (1990-1993), Universitatea Babeş-Bolyai, Centrul de Calcul Electronic, Cluj-Napoca, 1992.

[MOL92b] Moldovan, G., Râp,I., *Files d'attente dans des systèmes distribuées des services,* Studia Univ. "Babeş-Bolyai",Series Math., 37(3), 1992, 69 - 74;

[MOL97a] Moldovan, G., *Sisteme distribuite. Reţele de calculatoare (note de curs)*, Universitatea Babeş-Bolyai, Facultatea de Matematică şi Informatică, Cluj-Napoca, 1997.

[MOL97b] Moldovan, G., Moldovan, M., Moldovan, S., *Sisteme şi posibilităţi de modelare a lor*, Volum: Lucrările Sesiunii de Comunicări Ştiinţifice a Universităţii "Bogdan Vodă" Baia Mare, martie 1997, Ed. Risoprint, Cluj-Napoca, 1997 (ISBN- 973-9298-IS-4) p.121-135.

[MOL98] Moldovan, G., *Catalan numbers and binary recursive defined objects,* Univ. "Babeş-Bolyai", Cluj – Napoca, Fac. of Math. and Computer Science, Research Seminars, Preprint no. 2, 1998, 189-200;

[MOL05] Moldovan, G., *Binary recursively defined objects*, Analele Ştiinţifice ale Universităţii "Al. I. Cuza" din Iasi, (Serie nouă) Informatică, Tom XV, 2005

[MOL06] Moldovan, G., Valeanu, M., *Redistributing databases in a computer network*, Analele Univ. Bucureşti, Ser. Math.-Info., 56, 2006, 171-174

[MOR77] Morgen, H. L., Levin, K. D., *Optimal Program and Data Location in Computer Networks*, Comm. ACM, May, 1977.

[PĂU93] Păunescu, F., Goleşteanu, D. P., *Sisteme cu prelucrare distribuită şi aplicaţiile lor*, Editura Tehnică, Bucureşti, 1993.

[PĂT98] Pătruţ, B.: *Internet pentru începători*, Ed. Teora, 1998

[PFA95] B. Pfaffenberger, B., Wall, D., *Que's Computer & Internet Dictionary, 1995* (traducere în limba română de N.D. Pora, Ed. Teora, 1999).

[QUE82] Queille, J.-P., *Le système CESAR: description, specification et analyse des applications réparties*, Thèse, USM Grenoble, 1982.

[RAY88] Raynal, M*., Networks and distributed computations: concepts, tools and algorithms*, The MIT Press and North Oxford Academic, 1988

[RAY91] Raynal, M., *La communication et le temps dans les réseaux et les systèmes réparttis*, Editions Eyrolles, 1991

[SIU94] Siu, K.-Y., Tzeng, H.-Y., *Congestion Control for Multicast Service in ATM Networks*, Technical Report No. 94-03-01, Department. of Electrical & Computer Engineering, University of California, Irving, 1994.

[TAN81] Tanenbaum, A. S., *Network Protocols*, ACM, Computing Surveys, Vol. 13, No. 4, December, 1981.

TAN97] Tanenbaum, A. S., *Reţele de calculatoare*, Computer Press AGORA, Tîrgu Mureş, 1997 (traduere din engleză: Computer Networks, Pretince-Hall, 1981).
[TOA02] Toadere, T., *Grafe teorie, algoritmi şi aplicaţii*, Editura Albastră, Cluj-Napoca, 2002

[WIJ88] Wijbrands, R. J., *Queuing Network Models and Performance Analysis of Computer Systems*, Technische Universiteit, Eindhoven, 1988.

[Wil90] Williams, S. A., *Programming Models for Parallel Systems,* John Wiley & Sons, Chicester, 1990.

[WIS00] Wi, S., Choi, Y., *A Delay Constrained Distributed Multicast Routing Algorithm*, Department of Computer Engineering, Seoul National University, Internet, 2000.

[ZAN98] Zanella, P., Ligier, Y.: *Architecture et technologie des ordinateurs,* 3e édition, DUNOD, Paris, 1998.

Webgrafie:

1. http://www.networkcomputing.com/
2. http://boinc.berkeley.edu/
3. http://www.networkcomputing.co.uk/
4. http://www.distributed.net/
5. http://dsonline.computer.org/portal/site/dsonline/index.jsp
6. http://distributedcomputing.info/
7. www.informatica.com/solutions/resource_center/glossary/default.htm
8. http://www.gridcomputing.com/
9. http://www-1.ibm.com/grid/
10. http://www.oracle.com/technologies/grid/index.html

11. http://www.gridcomputingplanet.com/
12. http://gridcafe.web.cern.ch/gridcafe/
13. http://www.realitygrid.org/information.html
14. http://www.springerlink.com/content/1573-7543/
15. http://www.ieeetfcc.org/
16. http://www.trygve.com/furbeowulf.html
17. http://library.thinkquest.org/C007645/english/0-welcome.htm
18. http://www.internet.com/
19. http://www.legi-internet.ro/
20. http://www.intranetjournal.com/
21. http://www.ici.ro/ici//revista/ria2003_4/art3.html
22. http://www-unix.mcs.anl.gov/mpi/
23. http://www.csm.ornl.gov/pvm/
24. http://www.linux.org
25. http://info.tech.pub.ro/~fionescu/CP/CP.html
26. http://www.linux-ha.org/ClusterResourceManager
27. http://www.csm.ornl.gov/pvm/
28. http://www.globus.org/
29. http://www.globus.org/toolkit/
30. http://detective.internet2.edu/applet/index.html
31. http://www.epm.ornl.gov/pvm/pvm_home.html
32. http://www.netlib.org/pvm3
33. http://www.ICANN.org
34. http://www.gac.icann.org/docs/index.htm
35. http://gac.icann.org/web/about/gac-outreach_Romanian.htm
36. http://www.domreg.ro/domain.html