

Logică și Structuri Discrete

Logică Propozitională

After James Hein - Discrete Structures, Logic and Computability,
Logic in Action (<http://www.logicinaction.org/>) and J.Russell, P. Norvig - Artificial Intelligence
Section 6.2 Propositional Calculus (Subsections Intro, Well-Formed
Formulas, Syntax, Semantics, Equivalence pp. 309-316)
Section 6.2 (Subsections Disjunctive Normal Form, Conjunctive Normal
Form, Constructing CNF/DNF Using Equivalences pp. 320-326) Section 6.3 (Formal Reasoning
Systems, pp. 329-334) Section 6.3 (Subsections Indirect Proof, pp. 338-339)
Section 9.2 (Subsection A Primer of Resolution for Propositions pp.
461-462)

Sintaxa Limbajului

Alfabetul

Simboluri de adevăr: **true, false**

Simboluri pentru conectori logici: **¬, !, ∧ , ∨**

Variabile propoziționale: **p, q, r, etc.**

Simboluri de punctuație: **(și)**

¬ not, negație

∧ and, și, conjuncție

∨ or, sau, disjuncție

! Dacă ... atunci ..., condiție, implicație

Sintaxa Limbajului

Formule bine formate (wff) în logica propozițională

Definiție inductivă (informal)

Un wff este un simbol de adevăr (true, false) sau o variabilă propozițională sau negația (\neg) unui wff sau conjuncția (\wedge) a două wff sau disjuncția (\vee) a două wff sau implicația (\rightarrow) unui wff din alt wff sau un wff între paranteze

Utilizând gramatici

$S ::= \text{true} \mid \text{false} \mid \text{Propoziție} \mid \neg S \mid S \wedge S \mid S \vee S \mid S!S \mid (S)$

Propoziție ::= p | q | r | ...

Sunt acestea wff în logica propozițională ?

$\neg\neg p$, $p \vee q$, $p \wedge (p \vee \neg q)$

Da

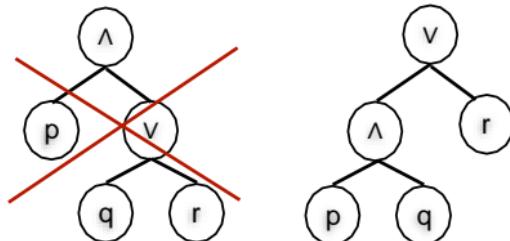
$\neg\wedge q \vee p$, $\neg p \neg q$

Nu

Ordinea de Evaluare a Conectorilor

$p \wedge q \vee r$ este un wff

Dar aceste definiții sunt ambigue
deoarece nu capturează ordinea în
care conectorii sunt evaluati



Pentru a elimina această problemă, stabilim un set de reguli

1. Prioritatea

\neg (cea mai mare - se evaluatează primul)

\wedge

\vee

$!$ (cea mai mică - se evaluatează ultimul)

2. \wedge , \vee , $!$ sunt asociativi la stînga

Cu alte cuvinte, dacă același conector apare succesiv de două sau mai multe ori,
fără paranteze, atunci evaluăm expresia de la stînga la dreapta

Semantica

Înțelesul unei formule este valoarea de adevăr **adevărat (T)** sau **fals (F)**

Înțelesul simbolurilor **true și false** sunt valorile de adevăr **adevărat (T)** respectiv **fals (F)**

Înțelesul conectorilor (pentru wff arbitrară vom utiliza A, B, etc.)

A	B	$A \wedge B$	$A \vee B$	$A \neg B$	$\neg A$
F	F	F	F	T	T
F	T	F	T	T	T
T	F	F	T	F	F
T	T	T	T	T	F

În esență, o stipulare a valorii de adevăr pentru variabilele propoziționale (wff-uri atomice, **adică p, q, r, etc.**) dintr-un wff se numește o **interpretare** a wff-urilor atomice (denumită și atribuire de adevăr, atribuire de valoare, valuation)

Exemplu - În formula $p \wedge q \wedge r$

O interpretare este: p este T, q este F, r este T; o altă interpretare este: p este T, q este T, r este T

Calcularea Valorii de Adevăr

Calculați valoarea de adevăr pentru $\neg p \wedge q \wedge r$ în interpretarea p este T, q este F, r este T

$\neg T \wedge F \wedge T$
 $\neg T \wedge F \wedge T \wedge F \wedge T$
 $F \wedge F \wedge T \wedge F \wedge F$
 $F \wedge F$
T

A	B	$A \wedge B$	$A \vee B$	$A \neg B$	$\neg A$
F	F	F	F	T	T
F	T	F	T	T	T
T	F	F	T	F	F
T	T	T	T	T	F

Câteva Cuvinte Despre!

A	B	$A \rightarrow B$
F	F	T
F	T	T
T	F	F
T	T	T

Citim $A \rightarrow B$ "dacă A atunci B", "A implică B"

A e numit antecedent, premisă sau ipoteză
B e numit consecvent sau concluzie

Mare atenție la semantica acestui conector

Când antecedentul (A) este T, $A \rightarrow B$ este T doar când B este T. $A \rightarrow B$ este tot timpul T când antecedentul (A) este F.
Nu există o relație de cauzalitate între A și B.

"dacă 5 este par atunci Sam este deștept" este T
"dacă 5 este impar atunci București este capitala României" este T

Câteva Cuvinte Despre !

A	B	A ! B
F	F	T
F	T	T
T	F	F
T	T	T

Citim A ! B “dacă A atunci B”, “A implică B”

A e numit antecedent, premisă sau ipoteză B e numit consecvent sau concluzie

Perspectiva inversă

Când ştim că A!B este T atunci

- 1.dacă ştim că A este T atunci B este T
- 2.dacă ştim că A este F atunci nu ştim nimic despre B (adică el poate fi T sau F dar nu ştim exact)

“Dacă plouă (A) atunci sunt nori pe cer (B)” este T “Plouă” (A is T)

Deci, sunt nori pe cer (B is T)

“Dacă plouă (A) atunci sunt nori pe cer

(B)” este T “Nu plouă” este T (A is F)

Deci, pot fi nori pe cer sau pot să nu fie nori pe cer

Câteva Cuvinte Despre !

A	B	$A \neg B$
F	F	T
F	T	T
T	F	F
T	T	T

Citim $A \neg B$ "dacă A atunci B", "A implică B"

A e numit antecedent, premisă sau ipoteză B e numit consecvent sau concluzie

Perspectiva inversă

Când știm că $A \neg B$ este T atunci

- 1.dacă știm că A este T atunci B este T
- 2.dacă știm că A este F atunci nu știm nimic despre B (adică el poate fi T sau F dar nu știm exact)

"Dacă opun rezistență (A) inamicul mă va omorâ (B)" este T "Nu opun rezistență" este T (A este F) Deci, inamicul poate să mă omoare sau poate să nu mă omoare !!!

Obs.

$A \rightarrow B$ este $(A \rightarrow B) \wedge (B \rightarrow A)$

Tabela de Adevăr

Prezintă valorile de adevăr a unui wff în toate posibilele interpretări

Exemplu - tabela de adevăr pentru $\neg(p \vee q) \rightarrow r$

Avem 3 propoziții atomice (p,q,r) deci avem 2^3 interpretări posibile (în general 2^n , unde n este numărul de propoziții atomice distințe)

p	q	r	\neg	(p	\vee	q)	\rightarrow	r
f	f	f	t	t	f	t	f	f
f	f	t	t	f	t	f	t	f
f	t	f	f	t	f	f	t	f
f	t	t	t	t	t	t	t	t
t	f	f	f	t	f	t	f	f
t	f	t	t	f	t	f	t	f
t	t	f	f	t	f	t	t	f
t	t	t	t	t	t	t	t	t

Tabela de Adevăr

Prezintă valorile de adevăr a unui wff în toate posibilele interpretări

Exemplu - tabela de adevăr pentru $\neg(p \vee q) \rightarrow r$

Avem 3 propoziții atomice (p,q,r) deci avem 2^3 interpretări posibile (în general 2^n , unde n este numărul de propoziții atomice distințe)

p	q	r	\neg	(p	v	q)	\rightarrow	r
f	f	f		f		f		f
f	f	t		f		f		t
f	t	f		f		t		f
f	t	t		f		t		t
t	f	f		t		f		f
t	f	t		t		f		t
t	t	f		t		t		f
t	t	t		t		t		t

Tabela de Adevăr

Prezintă valorile de adevăr a unui wff în toate posibilele interpretări

Exemplu - tabela de adevăr pentru $\neg(p \vee q) \wedge r$

Avem 3 propoziții atomice (p,q,r) deci avem 2^3 interpretări posibile (în general 2^n , unde n este numărul de propoziții atomice distințe)

p	q	r	\neg	(p	\vee	q)	\wedge	r
f	f	f		f	f	f		f
f	f	t		f	f	f		t
f	t	f		f	t	t		f
f	t	t		f	t	t		t
t	f	f		t	t	f		f
t	f	t		t	t	f		t
t	t	f		t	t	t		f
t	t	t		t	t	t		t

Tabela de Adevăr

Prezintă valorile de adevăr a unui wff în toate posibilele interpretări

Exemplu - tabela de adevăr pentru $\neg(p \vee q) \rightarrow r$

Avem 3 propoziții atomice (p, q, r) deci avem 2^3 interpretări posibile (în general 2^n , unde n este numărul de propoziții atomice distințe)

p	q	r	\neg	$(p$	\vee	$q)$	\rightarrow	r
f	f	f	t	f	f	f	f	f
f	f	t	t	f	f	f	f	t
f	t	f	f	f	t	t	t	f
f	t	t	f	f	t	t	t	t
t	f	f	f	t	t	f	f	f
t	f	t	f	t	t	f	t	t
t	t	f	f	t	t	t	t	f
t	t	t	f	t	t	t	t	t

Tabela de Adevăr

Prezintă valorile de adevăr a unui wff în toate posibilele interpretări

Exemplu - tabela de adevăr pentru $\neg(p \vee q) \rightarrow r$

Avem 3 propoziții atomice (p, q, r) deci avem 2^3 interpretări posibile (în general 2^n , unde n este numărul de propoziții atomice distințe)

p	q	r	\neg	$(p$	\vee	$q)$	\rightarrow	r
f	f	f	t	f	f	f	f	f
f	f	t	t	f	f	f	t	t
f	t	f	f	f	t	t	t	f
f	t	t	f	f	t	t	t	t
t	f	f	f	t	t	f	t	f
t	f	t	f	t	t	f	t	t
t	t	f	f	t	t	t	t	f
t	t	t	f	t	t	t	t	t

Tabela de Adevăr

role de adevăr a unui wff în toate interpretări

Înțelesul unei formule interpretările este tabela săde adevăr (de multe ori, înțelesul de adevăr)

tabelă de adevăr pentru $\neg(p \vee q) \wedge r$

propoziții atomice (p,q,r) deci avem 2^3 interpretări posibile (în general 2^n , unde n este numărul de propoziții atomice distințe)

p	q	r	\neg	(p	\vee	q)	\wedge	r
f	f	f	t	f	f	f	f	f
f	f	t	t	f	f	f	f	t
f	t	f	f	f	t	t	t	f
f	t	t	f	f	t	t	t	t
t	f	f	f	t	t	f	t	f
t	f	t	f	t	t	f	t	t
t	t	f	f	t	t	t	t	f
t	t	t	f	t	t	t	t	t

Terminologie

Tautologie - o formulă care are valoarea de adevăr

T în toate posibilele interpretări (mai spunem că formula este validă)

Contradicție - o formulă care are valoarea de

adevăr F în toate posibilele interpretări (mai spunem că formula este nerealizabilă (unsatisfiable))

Contingență - o formulă care are valoarea de adevăr F în unele interpretări și T în alte interpretări

O formulă este **realizabilă** (satisfiable) când ea are valoarea de adevăr T în cel puțin o interpretare

Echivalență

În esență, două formule A și B sunt **echivalente** dacă și numai dacă tabelele lor de adevăr au aceleași valori. Scriem $A \equiv B$

Exemplu

$$p \wedge q \equiv q \wedge p$$

p	q	$p \wedge q$	$q \wedge p$
F	F	F	F
F	T	F	F
T	F	F	F
T	T	T	T

$A \equiv B$ dacă și numai dacă $(A \rightarrow B) \wedge (B \rightarrow A)$ este o tautologie

Echivalențe Utile

$\neg\neg A \equiv A$ (legea dublei negații) $A \vee \text{true} \equiv \text{true}$

$A \vee \text{false} \equiv A$
 $A \vee A \equiv A$ (legea idempotenței) $A \vee \neg A \equiv \text{true}$

$A \wedge \text{true} \equiv A$
 $A \wedge \text{false} \equiv \text{false}$
 $A \wedge A \equiv A$ (legea idempotenței) $A \wedge \neg A \equiv \text{false}$

$A \neq \text{true} \equiv \text{true}$
 $A \neq \text{false} \equiv \neg A$ $\text{true} \neq A \equiv A \neq \text{false}$
 $\neg A \equiv \text{true}$ $A \neq A \equiv \text{true}$

$A \neq B \equiv \neg A \vee B$ (legea implicației)
 $A \neq B \equiv \neg B \neq \neg A$ (legea contrapoziției)

$\neg(A \neq B) \equiv A \wedge \neg B$
 $A \neq B \equiv A \wedge \neg B \neq \text{false}$ Legile comutativității
 $A \wedge B \equiv B \wedge A$
 $A \vee B \equiv B \vee A$ Legile asociativității
 $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
 $(A \vee B) \vee C \equiv A \vee (B \vee C)$ Legile distributivității

$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
 $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
Legile absorbției
 $A \wedge (A \vee B) \equiv A$
 $A \vee (A \wedge B) \equiv A$
 $A \wedge (\neg A \vee B) \equiv A \wedge B$ $A \vee (\neg A \wedge B) \equiv A \vee B$ Legile De Morgan
 $\neg(A \wedge B) \equiv \neg A \vee \neg B$

$\neg(A \vee B) \equiv \neg A \wedge \neg B$

De ce?

Putem să le utilizăm pentru a arăta că alte formule sunt echivalente fără să folosim tabele de adevăr

Orice sub-wff a unui wff poate fi înlocuit de un wff echivalent fără a schimba valoarea de adevăr a wff-ului original

Regula Înlocuirii

Exemplu

Arătăm că $A \rightarrow (B \rightarrow C) \equiv B \rightarrow (A \rightarrow C)$

$$\begin{aligned} A \rightarrow (B \rightarrow C) & \equiv A \rightarrow (\neg B \vee C) && \text{(legea implicației)} \\ & \equiv \neg A \vee (\neg B \vee C) && \text{(legea implicației)} \\ C) & \equiv (\neg A \vee \neg B) \vee C && \text{(legea asociativității)} \\ & \equiv (\neg A \vee \neg B) \vee C && \text{(legea comutativității)} \\ & \equiv (\neg B \vee \neg A) \vee C && \text{(legea asociativității)} \\ & \equiv (\neg B \vee \neg A) \vee C && \text{(legea implicației)} \\ & \equiv \neg B \vee (\neg A \vee C) && \text{(legea implicației)} \\ & \equiv B \rightarrow (\neg A \vee C) && \\ & \equiv B \rightarrow (A \rightarrow C) && \end{aligned}$$

Forma Normală Disjunctivă

Un **literal** este o variabilă propozițională sau negată ei

Exemple $p, \neg p$

O **conjuncție fundamentală** este un literal sau o conjuncție de (doi sau mai mulți) **literali**

Exemple

$p, \neg p, \neg p \wedge q$

O **formă normală disjunctivă (DNF)** este o conjuncție fundamentală sau

o **disjuncție de** (două sau mai multe) **conjuncții fundamentale**

Examples

$p, \neg p, \neg p \wedge q, p \vee (\neg p \wedge q), (p \wedge q) \vee (\neg q \wedge p)$

Orice wff are un DNF echivalent

Pași de conversie

1. Eliminăm toate ! utilizând echivalența $A ! B \equiv \neg A \vee B$

2. Mutăm toate negațiile în sub-wff-uri pentru a crea literali, utilizând legile De Morgan $\neg(A \wedge B) \equiv \neg A \vee \neg B$, $\neg(A \vee B) \equiv \neg A \wedge \neg B$. Eliminăm dublele negații utilizând echivalența $\neg\neg A \equiv A$

3. Aplicăm distributivitatea $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$ pentru a obține DNF

Exemplu

$$\begin{aligned} & ((p \wedge q) ! r) \wedge s \\ & \equiv (\neg(p \wedge q) \vee r) \wedge s \\ & \equiv (\neg p \vee \neg q \vee r) \wedge s \\ & \equiv ((\neg p \vee \neg q) \wedge s) \vee (r \wedge s) \\ & \equiv (\neg p \wedge s) \vee (\neg q \wedge s) \vee (r \wedge s) \end{aligned}$$

Forma Normală Disjunctivă Completă

Presupunem că un wff W are n variabile propoziționale

Un DNF pentru W e un DNF **complet** dacă fiecare conjuncție fundamentală are exact n literali, unul pentru fiecare din cele n variabile din W

Exemplu pentru $((p \wedge q) \vee r) \wedge s$ ($n = 4$)

$(\neg p \wedge s) \vee (\neg q \wedge s) \vee (r \wedge s)$ nu e un DNF complet

Pași de conversie

1,2,3 de pe slide-ul anterior

4. Pentru a adăuga o variabilă lipsă (ex. r) la o conjuncție fundamentală C (conservând valoarea ei) scriem $C \equiv C \wedge \text{true} \equiv C \wedge (r \vee \neg r)$. Apoi distribuim \wedge peste \vee pentru a obține o disjuncție de două conjuncții fundamentale.
Repetăm pentru toate conjuncțiile fundamentale “incomplete” până obținem un DNF complet

Exemplu

Exemplu

$p \neq q$ (are 2 variabile/litere)

$$\equiv \neg p \vee q$$

Este un DNF dar nu complet. Introducem q în conjuncția fundamentală $\neg p$

$$\equiv (\neg p \wedge \text{true}) \vee q$$

$$\equiv (\neg p \wedge (q \vee \neg q)) \vee q$$

$$\equiv (\neg p \wedge q) \vee (\neg p \wedge \neg q) \vee q$$

Este un DNF dar nu complet. Introducem p în conjuncția fundamentală q

$$\equiv (\neg p \wedge q) \vee (\neg p \wedge \neg q) \vee (q \wedge \text{true})$$

$$\equiv (\neg p \wedge q) \vee (\neg p \wedge \neg q) \vee (q \wedge (p \vee \neg p))$$

$$\equiv (\neg p \wedge q) \vee (\neg p \wedge \neg q) \vee (q \wedge p) \vee (q \wedge \neg p)$$

$\equiv (\neg p \wedge q) \vee (\neg p \wedge \neg q) \vee (q \wedge p)$ Acesta e un DNF complet :)

Forma Normală Conjunctivă

Un **literal** este o variabilă propozițională sau negată ei

Exemple $p, \neg p$

O **disjuncție fundamentală** (denumită și clauză) este un literal sau o disjuncție de (doi sau mai mulți) literali

Exemple

$p, \neg p, \neg p \vee q$

O **formă normală conjunctivă (CNF)** este o disjuncție fundamentală sau

o conjuncție de (două sau mai multe) disjuncții fundamentale

Exemple

$p, \neg p, \neg p \vee q, p \wedge (\neg p \vee q), (p \vee q) \wedge (\neg q \vee p)$

Forma Normală Conjunctivă Completă

Presupunem că un wff W are n variabile propoziționale distincte

Un CNF pentru W este un CNF *complet* dacă fiecare disjuncție fundamentală are exact n literali, unul pentru fiecare variabilă din W

Pași de conversie

1. Eliminăm toate ! utilizând echivalența $A ! B \equiv \neg A \vee B$
2. Mutăm toate negațiile în sub-wff pentru a crea literali, utilizând legile De Morgan $\neg(A \wedge B) \equiv \neg A \vee \neg B$, $\neg(A \vee B) \equiv \neg A \wedge \neg B$. Eliminăm negațiile duble utilizând echivalența $\neg\neg A \equiv A$
3. Aplicăm distributivitatea $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ pentru a obține CNF
4. Pentru a adăuga o variabilă lipsă (ex. r) la o disjuncție fundamentală D (conservând valoarea ei) scriem $D \equiv D \vee \text{false} \equiv D \vee (r \wedge \neg r)$. Apoi distribuim \vee peste \wedge pentru a obține o conjuncție de două disjuncții fundamentale. Repetăm pentru toate disjuncțiile fundamentale “incomplete” până când obținem un CNF complet

Exemplu

Exemplu

$$\begin{aligned} p \wedge (p \neq q) & \quad (\text{are 2 litere/variabile}) \\ \equiv p \wedge (\neg p \vee q) & \end{aligned}$$

Este un CNF dar nu complet. q lipsește din prima disjuncție fundamentală

$$\begin{aligned} \equiv (p \vee \text{false}) \wedge (\neg p \vee q) \\ \equiv (p \vee (\neg q \wedge q)) \wedge (\neg p \vee q) \\ \equiv (p \vee \neg q) \wedge (p \vee q) \wedge (\neg p \vee q) \end{aligned}$$

Aceasta este un CNF complet :)

Exemplu

Exemplu

$$\begin{aligned} p \wedge (p \neq q) & \quad (\text{are 2 litere/variabile}) \\ \equiv p \wedge (\neg p \vee q) \end{aligned}$$

Este un CNF dar nu complet. q lipsește din prima disjuncție fundamentală.

$$\begin{aligned} & \equiv (p \vee \text{false}) \wedge (\neg p \vee q) \\ & \equiv (p \vee (\neg q \wedge q)) \wedge (\neg p \vee q) \\ & \equiv (p \vee \neg q) \wedge (p \vee q) \wedge (\neg p \vee q) \end{aligned}$$

Aceasta este un CNF complet :)

In acest exemplu și în cele
convenționale, mai în formule echivalente doar
manipulând simboluri pe baza unor reguli. Nu
am folosit tabele de adevară. Putem face
și alte lucruri doar "sintactic" ?DA!

Sistem Formal de Deducție

Observații

- Tabelele de adevăr sunt suficiente pentru a determina adevărul oricărei formule propoziționale. Dar, când formula are multe variabile și mulți conectori, tabela de adevăr este destul de “complicată” :(
- Noi (ca oameni) nu rationăm în termeni de tabele de adevăr

Un sistem formal de deducție are 3 ingrediente

1. Un set de wff pentru a reprezenta afirmațiile de interes
2. Un set de **axiome** (adică formule despre care “știm că sunt adevărate/valide”, de exemplu, arătând cu tabele de adevăr că sunt tautologii)
3. Un set de **reguli de inferență**

Reguli de Inferență

O regulă de inferență mapează un set de wff-uri, numite **premise** sau **antecedenți**, într-un singur wff denumit **concluzie** sau **consecvent**

$$\frac{P_1, P_2, \dots, P_k}{\therefore C}$$

Spunem:

“C este inferat din P_1 și P_2 și ... și P_k ”

“C este consecință directă a lui P_1 și P_2 și ... și P_k ”

∴ e citit ca “deci”, “prin urmare”, “astfel”, etc

O regulă de inferență conservă adevărul logic

Cu alte cuvinte

În orice interpretare în care toate premisele sunt adevărate, concluzia e adevărată

Când toate premisele sunt tautologii, concluzia e o tautologie

Pentru o regulă de inferență $P_1 \wedge P_2 \wedge \dots \wedge P_k \vdash C$ este o tautologie

Unele Reguli de Inferență în Logica Propozițională

$$\frac{A \vdash B, A}{\therefore B}$$

Modus ponens
(MP)

$$\frac{A \vdash B, \neg B}{\therefore \neg A}$$

Modus tollens
(MT)

$$\frac{A, B}{\therefore A \wedge B}$$

Conjunction
(Conj)

$$\frac{\begin{array}{c} A \\ \hline \end{array}}{\therefore A \vee B}$$

Addition
(Add)

$$\frac{\begin{array}{c} A \vee B, \neg A \\ \hline \end{array}}{\therefore B}$$

Disjunctive
syllogism (DS)

$$\frac{\begin{array}{c} A \wedge B \\ \hline \end{array}}{A \wedge B}$$

$$\frac{A \vdash B, B \vdash C}{\therefore A \vdash C}$$

Simplification
(Simp)
Hypothetical
syllogism (HS)

$$\frac{A \vee B, \neg B \vee C}{\therefore A \vee C}$$

Resolution
Rule

Ele conservă adevărul
logic

Să arătăm asta pentru MP

$$\frac{A \rightarrow B, A}{\therefore B}$$

Modus ponens (MP)

$(A \rightarrow B) \wedge A \rightarrow B$ este o tautologie

A	B	$A \rightarrow B$	$(A \rightarrow B) \wedge A$	$(A \rightarrow B) \wedge A \rightarrow B$
F	F	T	F	T
F	T	T	F	T
T	F	F	F	T
T	T	T	T	T

Observați că, atunci când premisele sunt adevărate, concluzia e adevărată

Ce putem face cu ele ?

Construim demonstrații

O demonstrație - o secvență finită de wff-uri, fiecare wff fiind
a.o axiomă sau
b.inferată din wff-uri anterioare utilizând reguli de inferență

1. W Motiv pentru W_1

¹ Motiv pentru W_2

2. W Motiv pentru W_3

² Motiv pentru W_n

3. W Motiv pentru W_n

Acest wff (adică W_n) e denumit teoremă

³

...

n. W_n

Tehnici de demonstrare

1.Demonstrare condiționată

2.Demonstrare indirectă

Demonstrare Condiționată

De obicei vrem să demonstrăm lucruri precum: dacă
A, B și C sunt adevărate atunci D e adevărat

“D e consecință a lui A, B și C”

sau “Din premisele A, B și C putem concluziona D” sau $A \wedge B \wedge C \vdash D$

O demonstrare condiționată dintr-un set de premise este o secvență finită de wff-uri, fiecare wff fiind:

- a.o axiomă sau
- b.o premisă sau
- c.inferată din wff-uri anterioare prin reguli de inferență

Regula Demonstrării Condiționate (CP)

Vrem o demonstrație pentru $A_1 \wedge A_2 \wedge \dots \wedge A_n \vdash B$. Construim o demonstrație pentru B utilizând ca premise A_1, \dots, A_n

Exemplul 1

Construim o demonstrație pentru $(a \vee b) \wedge (a \vee c) \wedge \neg a \vdash b \wedge c$

1. $a \vee b$ P P P
2. $a \vee c$ 1, 3, DS
3. $\neg a$ 2, 3, DS
4. b 4, 5, Conj
5. c
6. $b \wedge c$
7. QED. 1,2,3,6,CP

$$\frac{A \vee B, \neg A}{\therefore B}$$

Disjunctive syllogism
(DS)

$$\frac{A, B}{\therefore A \wedge B}$$

Conjunction
(Conj)

Exemplul 2

Echipa câștigă sau eu sunt trist. Dacă echipa câștigă atunci eu merg la un film.
Dacă eu sunt trist atunci câinele meu latră. Câinele meu nu latră. Prin urmare
eu merg la un film.

w - echipa câștigă s - eu sunt trist

m - eu merg la un film

b - câinele meu latră

Echipa câștigă sau eu sunt trist. w ∨ s Dacă echipa câștigă atunci eu merg

la un film. w ! m

Dacă eu sunt trist atunci câinele meu latră. s ! b

Câinele meu nu latră. $\neg b$

Prin urmare eu merg la un film. $(w \vee s) \wedge (w ! m) \wedge (s ! b) \wedge \neg b ! m$

Exemplul 2 (cont.)

$$(w \vee s) \wedge (w \neq m) \wedge (s \neq b) \wedge \neg b \neq m$$

$$\frac{A \neq B, A}{\therefore B}$$

- | | | |
|---------------|--------------|---|
| 1. $w \vee s$ | P | |
| 2. $w \neq m$ | P | Modus ponens
(MP) |
| 3. $s \neq b$ | P | |
| 4. $\neg b$ | P | |
| 5. $\neg s$ | 3,4, MT | <u>A \neq B, $\neg B$</u> |
| 6. w | 1,5, DS | $\therefore \neg A$ |
| 7. m | 2,6, MP | Modus tollens (MT) |
| 8. QED | 1,2,3,4,7,CP | |

$$\frac{A \vee B, \neg A}{\therefore B}$$

Disjunctive syllogism
(DS)

Regulile de Inferență Conservă Adevărul !

Data's Problem

Star Trek Next Generation
Episode 11, Series 4, Data's Day

$$\frac{A \neq B, A}{\therefore B}$$

Modus ponens
(MP)

În creierul pozitronic al lui Data (simplificat)

- | | | |
|------------------------|---|--------|
| 1. C P | nunta e anulată (C) | (true) |
| 2. C ! H P | dacă nunta e anulată (C) atunci Keiko e fericită (H) dacă | (true) |
| 3. H ! S P | Keiko e fericită (H) atunci O'Brian e încântat (S) | (true) |
| 4. H 1,2,MP | | |
| 5. S 3,4,MP | | |
| 6. QED 1,2,3,5,CP | O'Brian e încântat :) | |

Regulile de Inferență Conservă Adevărul !

Data's Problem

Star Trek Next Generation
Episode 11, Series 4, Data's Day

$$\frac{A \rightarrow B, A}{\therefore B}$$

Modus ponens
(MP)

În creierul pozitronic al lui Data (simplificat)

- | | | |
|------------------------|---|--------|
| 1. C P | nunta e anulată (C) | (true) |
| 2. C ! H P | dacă nunta e anulată (C) atunci Keiko e fericită (H) dacă | (true) |
| 3. H ! S P | Keiko e fericită (H) atunci O'Brian e încântat (S) | (true) |
| 4. H 1,2,MP | | |
| 5. S 3,4,MP | | |
| 6. QED 1,2,3,5,CP | O'Brian e încântat :) | |

Cum ?!?!??!
O'Brian nu e încântat (S e fals) ??!?!?
Dar Data a demonstrat ! E Data defect ?
Este Logica incorectă ?

Regulile de Inferență Conservă Adevărul !

Logica e corectă, regulile de inferență sunt corecte, demonstrația e corectă dar ... **pornind de la premise FALSE putem demonstra corect lucruri incorecte !**

Deci Data funcționează corect din punctul de vedere al Logicii; problema lui este că nu poate înțelege și anticipa emoțiile umane

A ! B,A

∴ B

Modus ponens
(MP)

În creierul pozitronic al lui Data (simplificat)

1. C P
2. C ! H P
3. H ! S P
4. H 1,2,MP
5. S 3,4,MP
6. QED 1,2,3,5,CP

nunta e anulată (C)	Unele din aceste premise sunt de fapt false	(true)
dacă nunta e anulată (C) atunci Keiko e fericită (H)		(true)
dacă Keiko e fericită (H) atunci O'Brian e încântat (S) (true)		
O'Brian e încântat :)		

Cum ?!?!??!
O'Brian nu e încântat (S e fals) ??!!?!!
Dar Data a demonstrat ! E Data defect ?
Este Logica incorectă ?

Demonstrare Indirectă

Să construim demonstrații nu e tocmai simplu

Alternative

$A \rightarrow B \equiv \neg B \rightarrow \neg A$. Încercăm să demonstrăm contrapositiva lui $A \rightarrow B$. Altfel spus, construim o demonstrație pentru $\neg B \rightarrow \neg A$

$A \rightarrow B \equiv A \wedge \neg B \rightarrow \text{false}$. Încercăm să demonstrăm a doua formulă. Demostrația trebuie să concluzioneze fals. Aceasta este o demonstrație “*reductio ad absurdum*”.

Regula demonstrării indirecte (IP)

Vrem o demonstrație pentru $A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow B$. Construim o demonstrație pentru fals utilizând premisele A_1, \dots, A_n și $\neg B$

Exemplul 3

Aceeași problemă cu filmul și câinele :)
 $(w \vee s) \wedge (w \neq m) \wedge (s \neq b) \wedge \neg b \neq m$

- | | |
|--|---|
| 1. $w \vee s$ | P P P P |
| 2. $w \neq m$ | P for IP |
| 3. $s \neq b$ | 2, 5, MT |
| 4. $\neg b$ | |
| 5. $\neg m$ | |
| 6. $\neg w$ | |
| 7. $\neg s$ | 3, 4, MT |
| 8. $\neg w \wedge \neg s$ | 6, 7, Conj |
| 9. $\neg(w \vee s)$ | 8, (DeMorgan) |
| 10. $(w \vee s) \wedge \neg(w \vee s)$ | 1, 9, Conj |
| 11. false | 10, ($A \wedge \neg A \equiv \text{false}$) |
| 12. QED | 1,2,3,4,5,11,IP |

$$\begin{array}{c} A \neq B, \neg B \\ \hline \therefore \neg A \\ \text{Modus tollens} \\ (\text{MT}) \\ \\ \hline A, B \\ \hline \therefore A \wedge B \\ \text{Conjunction (Conj)} \end{array}$$

Înapoi la Sisteme Formale de Deducție

În exemplele anterioare de demonstrații am folosit diferite reguli de inferență + diferite echivalențe (ex. DeMorgan) ca axiome

Există oare un sistem pentru logica propozițională care să aibă **un set fix de reguli de inferență și axiome și în care:**

1. Toate demonstrațiile să producă teoreme care sunt tautologii (proprietatea de **soundness**)
2. Toate posibilele tautologii să fie demonstrabile ca teoreme (proprietatea de **completeness**)

Da, există :)

Exemplu

Axiome

1. $\alpha \vdash (\beta \vdash \alpha)$
2. $(\alpha \vdash (\beta \vdash \gamma)) \vdash ((\alpha \vdash \beta) \vdash (\alpha \vdash \gamma))$
3. $(\neg \beta \vdash \neg \alpha) \vdash (\alpha \vdash \beta)$

Regula de inferență

$\alpha \vdash \beta, \alpha$

$\therefore \beta$

Modus ponens (MP)

Demonstrație pentru $p \vdash p$

1. $p \rightarrow ((p \rightarrow p) \rightarrow p)$ A1
2. $(p \rightarrow ((p \rightarrow p) \rightarrow p)) \rightarrow ((p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow p))$ A2
3. $(p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow p)$ 1,2,MP
4. $p \rightarrow (p \rightarrow p)$ A1
5. $p \rightarrow p$ 3,4,MP

Regula Rezoluției în Logica Propozițională

$$\begin{array}{c} \underline{\alpha \vee \beta, \neg\beta \vee \gamma} \\ \therefore \alpha \vee \gamma \end{array}$$

Regula de inferență a Rezoluției

$(\alpha \vee \beta) \wedge (\neg\beta \vee \gamma) \vdash (\alpha \vee \gamma)$ este o tautologie

α	β	γ	$\neg\beta$	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$	$(\alpha \vee \beta) \wedge (\neg\beta \vee \gamma)$	$(\alpha \vee \beta) \wedge (\neg\beta \vee \gamma) \vdash \alpha \vee \gamma$
F	F	F	T	F	T	F	F	T
F	F	T	T	F	T	T	F	T
F	T	F	F	T	F	F	F	T
F	T	T	F	T	T	T	T	T
T	F	F	T	T	T	T	T	T
T	F	T	T	T	T	T	T	T
T	T	F	F	T	F	T	F	T
T	T	T	F	T	T	T	T	T

Rescriere În Termeni de Disjuncții Fundamentale

Literalii - variabile propoziționale sau negata lor (ex. $p, \neg q$)

Doi literali se spune că sunt **complementi** dacă unul e negata celuilalt Exemple

$p, \neg p$ $q, \neg q$

$a_1 \vee a_2 \vee \dots \vee a_i \vee \dots \vee a_n$, $b_1 \vee b_2 \vee \dots \vee b_j \vee \dots \vee b_m$ $a_1 \vee \dots \vee a_{i-1} \vee a_{i+1} \vee \dots \vee a_n \vee b_1 \vee \dots \vee b_{j-1} \vee b_{j+1} \vee \dots \vee b_m$

unde:

$a_1, \dots, a_n, b_1, \dots, b_m$ sunt literali

a_i, b_j sunt complementi (ex. $a_i = p, b_j = \neg p$)

Rezultatul produs se numește rezolvent

$p, \neg p$

#

Un caz important - Obținerea clauzei (disjuncție fundamentală) vide înseamnă
fals

Demonstrarea cu Rezoluție în Logica Propozițională

Pentru a demonstra că W e valid

1. Formăm negata $\neg W$. De exemplu, dacă W are forma $A \wedge B \wedge C \wedge D$ atunci $\neg W$ va fi $\neg A \wedge \neg B \wedge \neg C \wedge \neg D$
2. Transformăm $\neg W$ în forma clauzală (CNF pentru logica propozițională)
3. Luăm clauzele (disjuncții fundamentale) ca premise
4. Aplicăm regula rezoluției pentru a obține clauza vidă (fals). Dacă o găsim înseamnă că avem o contradicție

Din moment ce $\neg W$ e o contradicție, W este validă

Utilă pentru automatizare :)

Exemplu

Aceeași problemă cu filmul și câinele :) $W = (w \vee s) \wedge (w \neq m) \wedge (s \neq b) \wedge \neg b \neq m$

$$\neg W = \neg((w \vee s) \wedge (w \neq m) \wedge (s \neq b) \wedge \neg b \neq m)$$

$$\equiv \neg(\neg((w \vee s) \wedge (w \neq m) \wedge (s \neq b) \wedge \neg b) \vee m)$$

$$\equiv \neg\neg((w \vee s) \wedge (w \neq m) \wedge (s \neq b) \wedge \neg b) \wedge \neg m$$

$\equiv (w \vee s) \wedge (w \neq m) \wedge (s \neq b) \wedge \neg b \wedge \neg m$ (Obs. concluzia initială e negată)

$$\equiv (w \vee s) \wedge (\neg w \vee m) \wedge (\neg s \vee b) \wedge \neg b \wedge \neg m$$

1. $w \vee s$ P
2. $\neg w \vee m$ P
3. $\neg s \vee b$ P
4. $\neg b$ P P
5. $\neg m$ 3,4, Resolution
6. $\neg s$ 6,1, Resolution
7. w 2,5, Resolution
8. $\neg w$ 9. # 7,8, Resolution

QED.

$$p \vee C, \neg p \vee D$$

$$\therefore C \vee D$$

(disjuncțiile redundante din clauze se elimină)

Logică și Structuri Discrete

Logica Predicatelor (de ordinul întâi)

După James Hein - Discrete Structures, Logic and Computability and
J.Russell, P. Norvig - Artificial Intelligence

Section 7.1 First Order Predicate Calculus (Intro, Well-Formed
Formulas, Semantics, Validity - Only the first part)
pp. 351 - pp. 362

Section 7.2 Equivalencies (Normal Forms, English Sentences) pp. 374 -
pp. 379 Section 9.2 (Subsection Clauses and
Clausal Form pp. 455 - pp. 460, Subsection
Substitution and Unification pp. 462 - 467 -
The algorithm and Composition of substitutions sub-subsections are
optional) Subsection Resolution: The general Case
pp. 467 - pp. 470) Subsection Theorem Proving
with Resolution pp. 473 - pp. 474)

De ce altă Logică?

O propoziție (în logica propozițională)
este o afirmație ca **întreg**

Toți specialiștii în știința calculatoarelor dețin un computer.
Socrate nu deține un computer.
Deci, Socrate nu e specialist în știința calculatoarelor.

Pentru formalizare, trebuie să spargem afirmațiile în
părțile lor. **Toți**, **deține** sunt cuvinte importante
pentru a înțelege argumentația aceasta

Predicatul

Informal, un predicat este o afirmație care poate fi falsă ori adevărată **în funcție** de valoarea variabilelor sale

“**x deține** un computer”

Este fals sau adevărat ?

Nu știm deoarece valoarea de adevăr depinde de x

“Mihai **deține** un computer”

Afirmația devine propoziție și are valoarea de adevăr true

Predicatul - Exemple

Formal, predicatul e o relație

(poate fi văzut și ca o proprietate)

Fie $Pc(x)$ însemnând “ x deține un computer”

Pc este un predicat ce descrie proprietatea de a deține un computer

$Pc(Mihai)$ este adevărat

Fie $Q(x,y)$ însemnând “ $x < y$ ”

Q este un predicat pe care îl cunoștem ca relația “mai mic decât” $Q(1,9)$
este adevărat

$Q(8,3)$ este fals

Fie $P(x)$ însemnând “ x este un întreg impar” $P(9)$ este adevărat

$P(8)$ este fals

Cuantificatorul Existențial

Fie $P(x)$ însemnând “ x este un întreg impar”

$P(2) \vee P(3) \vee P(4) \vee P(5)$ este adevărat sau

$$D = \{2, 3, 4, 5\}$$

$\exists x \in D : P(x)$ este adevărat

Considerând afirmația fără să ținem cont de un anume set de numere și fără să ținem cont de ce înseamnă P , putem scrie

$\exists x P(x)$

și îl citim “există un x astfel încât $P(x)$ ”

Observați că nu putem da o valoare de adevăr acestei afirmații

Cuantificatorul Universal

Fie $P(x)$ însemnând “ x este un întreg impar”

$P(1) \wedge P(3) \wedge P(5) \wedge P(7)$ este adevărat sau

$$D = \{1, 3, 5, 7\}$$

$\forall x \in D : P(x)$ este adevărat

Considerând afirmația fără să ținem cont de un anume set de numere și fără să ținem cont de ce înseamnă P , putem scrie

$\forall x P(x)$

și citim “pentru orice $x P(x)$ ”

Observați că nu putem da o valoare de adevăr acestei afirmații

Sintaxa Logicii Predicatelor

Alfabetul

Variabile: x, y, \dots

Constante: a, b, c, \dots

Funcții: f, g, h, \dots

Predicate: P, Q, R, \dots Conectori: $\neg, !, \wedge, \vee$

Cuantificatori: \forall, \exists

Simboluri de punctuație: $(,)$

Sintaxa (cont.)

Formule bine formate (FBF) în logica predicatelor

Un **termen** este o variabilă, o constantă sau o funcție aplicată unor argumente care la rândul lor sunt termeni

Exemple

$x, a, f(x, g(b))$

Un **atom (formulă atomică)** este un predicat aplicat unor argumente care la rândul lor sunt termeni

Exemple

$P(x, a), Q(y, f(x))$

FBF în logica de ordinul întâi (definiție inductivă) Baza: Orice **atom** este un fbf
Inducția: Dacă **W și V** sunt fbf-uri și **x este o variabilă** atunci următoarele expresii sunt fbf-uri

$(W), \neg W, W \wedge V, W \vee V, W ! V, \exists x W, \forall x W$

Exemplu

$\exists x P(x, y) ! Q(x)$

Sintaxa (cont.)

Formule bine formate (FBF) în logica predicatelor

Un **termen** este o variabilă, o constantă sau o funcție aplicată unor argumente care la rândul lor sunt termeni

Exemple

$x, a, f(x, g(b))$

Un **atom (formulă atomică)** este un predicat aplicat unor argumente care la rândul lor sunt termeni

Exemple

$P(x, a), Q(y, f(x))$

FBF în logica de ordinul întâi (definiție inductivă) Baza: Orice atom

Inducția: Dacă W și V sunt fbf-uri și x este o variabilă atunci următoarele expresii sunt fbf-uri

$(W), \neg W, W \wedge V, W \vee V, W ! V, \exists x W, \forall x W$

Exemplu

$\exists x P(x, y) ! Q(x)$

Observați că putem cuantifica numai variabile. În acest caz ne referim la **logica predicatelor de ordinul întâi** (la care ne vom rezuma).

Există și alte logici superioare în care putem cuantifica și alte lucruri

Ordinea de Evaluare

1. Priorități (obs. pentru cuantificatori, aceste convenții pot差别 de la autor la autor)

$\exists x, \forall y, \neg$ (**cea mai mare prioritate - evaluate primele**)

\wedge

\vee

$!$ (**cea mai mică prioritate - evaluată ultima**)

2. Dacă cuantificatorii sau negațiile apar unele lângă altele atunci simbolul cel mai din dreapta este grupat cu cea mai mică fbf din dreapta simbolului

3. $\wedge, \vee, !$ sunt asociative la stânga

Exemple

$\forall x P(x) ! Q(x)$ înseamnă $(\forall x P(x)) ! Q(x)$

$\exists x \neg P(x,y) ! Q(x) \wedge R(y)$ înseamnă $(\exists x (\neg P(x,y))) ! (Q(x) \wedge R(y))$

$\exists x P(x) ! \forall x Q(x) \vee P(x) \wedge R(x)$ înseamnă $(\exists x P(x)) ! ((\forall x Q(x)) \vee (P(x) \wedge R(x)))$

Domeniul Cuantificatorului / Variabile Libere și Legate

Aria de influență a unui cuantificator se numește **domeniu** cuantificatorului

Exemple (culoarea e domeniul)

$\forall x P(x) \rightarrow Q(x)$

$\exists x P(x) \rightarrow \forall x Q(x) \vee P(x) \wedge R(x)$

$\exists x P(x,y) \rightarrow Q(x)$

$\exists x (P(x,y) \rightarrow Q(x))$

O apariție a unei variabile x într-un FBF e **legată** dacă ea e localizată în domeniul unui $\exists x$ sau $\forall x$ sau este variabila cuantificatorului însăși. Altfel, apariția se spune că e **liberă**

Exemple (roșu/portocaliu - legat; albastru - liber)

$\forall x P(x) \rightarrow Q(x)$

$\exists x P(x) \rightarrow \forall x Q(x) \vee P(x) \wedge R(x)$

$\exists x P(x,y) \rightarrow Q(x)$

$\exists x (P(x,y) \rightarrow Q(x))$

Dacă **x** e o variabilă și **t** este un termen atunci **x/t** se numește legare a lui **x** cu **t**

$W(x/t)$ este fbf-ul obținut din W înlocuind toate aparițiile **libere** a lui **x** cu **t**

Exemple

$$W = P(x) \vee \exists x Q(x,y)$$

$$W(x/a) = P(a) \vee \exists x Q(x,y)$$

$$W(x/a)(y/b) = P(a) \vee \exists x Q(x,b)$$

Semantica - Interpretare

O interpretare a unui fbf constă dintr-un set D (ne-vid) denumit **domeniul interpretării**, împreună cu o atribuire ce asociază fiecărui simbol din fbf:

1. Fiecărui simbol de predicat (n-ar) trebuie să îi atribuim o relație (n-ară) peste D. Un predicat fără argumente este o propoziție căreia trebuie să-i atribuim o valoare de adevăr (adevărat sau fals)
2. Fiecărui simbol de funcție (n-ară) trebuie să îi atribuim o funcție (n-ară) peste D
3. Fiecărei variabile libere trebuie să îi atribuim o valoare din D. Toate aparițiile libere a unei variabile x îi atribuim aceeași valoare
4. Fiecărei constante trebuie să îi atribuim o valoare din D. Toate aparițiile aceleleași constante îi atribuim aceeași valoare

Semantica - Exemple de Interpretări

Fie $W = \forall x(P(f(x,x),x) \wedge P(x,y))$

O interpretare

Domeniul interpretării este N (numere naturale)

P este relația de egalitate $y = 0$

$f(a,b) = (a + b) \% 3$

$\forall x \in N (((x+x) \% 3) = x) \wedge (x = 0)$

O altă interpretare

Domeniul interpretării este $D = \{a,b\}$

P este relația de egalitate $y = a$

$f(a, a) = a, f(b, b) = b$

$\forall x \in D ((f(x,x) = x) \wedge (x = a))$

Semantica (cont.)

Presupunem că avem o interpretare cu domeniul D pentru un fbf

Dacă **fbf**-ul nu are cuantificatori atunci înțelesul ei este valoarea de adevăr a expresiei obținută din fbf aplicând interpretarea

Dacă **fbf**-ul conține cuantificatori atunci fiecare expresie a unui cuantificator e evaluată astfel:

$\exists x$ W e adevărat dacă W(x/d) e adevărat pentru un $d \in D$. Altfel expresia e falsă

$\forall x$ W e adevărat dacă W(x/d) e adevărat pentru orice $d \in D$. Altfel expresia e falsă

Fie $W = \forall x (P(f(x,x),x) \wedge P(x,y))$

Semantica - Exemple

Interpretarea

Domeniul $D = \{a,b\}$

P e relația de egalitate $y = a$

$f(a,a) = a, f(b,b) = b$

$\forall x \in D ((f(x,x) = x) \wedge (x = a))$

“ $\forall x$ W e adevărat dacă $W(x/d)$ e adevărat pentru orice $d \in D$. Altfel e fals”

$W(x/a) = ((f(a,a) = a) \wedge (a = a)) \quad T \wedge T$

T

$W(x/b) = ((f(b,b) = b) \wedge (b = a)) \quad T \wedge F$

F

Deci, W e falsă în această interpretare

Terminologie

Valid - o formulă care are valoarea de adevăr adevărat în toate interpretările posibile

Nerealizabilă - o formulă care are valoarea de adevăr fals în toate interpretările posibile

Realizabilă - o formulă care are valoarea de adevăr adevărat în cel puțin o interpretare

O interpretare este un **model** pentru o formulă dacă interpretarea face formula adevărată. Altfel, interpretarea se numește **contramodel**

Echivalență

Două formule A și B sunt echivalente dacă și numai dacă ambele au aceeași valoarea de adevăr în raport cu orice interpretare pentru A și B. Scriem $A \equiv B$

Dacă x nu apare în fbf-ul V atunci avem următoarele echivalențe:

Disjunții (2)

$$\forall x (V \vee W(x)) \equiv V \vee \forall x$$

$$W(x)$$

$$\exists x (V \vee W(x)) \equiv V \vee \exists x$$

$$W(x)$$

Conjuncții (3)

$$\forall x (V \wedge W(x)) \equiv V \wedge \forall x$$

$$W(x)$$

$$\exists x (V \wedge W(x)) \equiv V \wedge \exists x$$

$$W(x)$$

Implicații

$$\forall x (V \rightarrow W(x)) \equiv V \rightarrow \forall x W(x)$$

$$\exists x (V \rightarrow W(x)) \equiv V \rightarrow \exists x W(x)$$

$$\forall x (W(x) \rightarrow V) \equiv \exists x W(x) \rightarrow V$$

$$\exists x (W(x) \rightarrow V) \equiv \forall x W(x) \rightarrow V$$

$$\neg(\forall x W) \equiv \exists x \neg W \quad (1)$$

$$\neg(\exists x W) \equiv \forall x \neg W \quad (1)$$

$$\forall x \forall y W \equiv \forall y \forall x W$$

$$\exists x \exists y W \equiv \exists y \exists x W$$

$$\exists x (P(x) \vee Q(x)) \equiv \exists x P(x) \vee \exists x$$

$$Q(x)$$

Reguli de redenumire

$$\exists x W(x) \equiv \exists y W(y) \text{ dacă } y \text{ nu apare în}$$

$$W(x)$$

$$\forall x W(x) \equiv \forall y W(y) \text{ dacă } y \text{ nu apare în}$$

$$W(x) \text{ unde } W(y) \text{ e obținut din } W(x)$$

înlocuind toate aparițiile libere a lui x cu

$$y$$

Forma Normală Prenex

Forma Normală Prenex

Un fbf W este într-o formă normală prenex dacă toți cuantificatorii sunt la stânga expresiei

$Q_1x_1 \dots Q_nx_n M$ unde Q_i este \forall sau \exists , fiecare x_i este distinct, M nu conține \forall sau \exists

Forma Normală Prenex Disjunctivă

$Q_1x_1 \dots Q_nx_n (D_1 \vee D_2 \vee \dots \vee D_m)$, fiecare D_i fiind o conjuncție fundamentală de literali

Forma Normală Prenex Conjunctivă

$Q_1x_1 \dots Q_nx_n (C_1 \wedge C_2 \wedge \dots \wedge C_m)$, fiecare C_i fiind o disjuncție fundamentală de literali

Literal - un fbf atomic sau negata lui Exemple

$P(x)$, $\neg Q(x,y)$

Construirea Formei Prenex CNF/DNF

Pași de conversie și exemplu

Fie $W = \forall x P(x) \vee \exists x Q(x) \neg R(x) \wedge \exists x R(x)$. O transformăm în Prenex CNF

1. Redenumim variabilele din W în aşa fel încât niciun cuantificator nu foloseşte acelaşi nume de variabilă și în aşa fel încât variabilele libere sunt diferite de cele cuantificate

$$\begin{aligned} W &= \forall x P(x) \vee \exists x Q(x) \neg R(x) \wedge \exists x R(x) \\ &\equiv \forall y P(y) \vee \exists z Q(z) \neg R(x) \wedge \exists w R(w) \end{aligned}$$

2. Eliminăm implicațiile pe baza echivalenței $A \rightarrow B \equiv \neg A \vee B$

$$\equiv \neg (\forall y P(y) \vee \exists z Q(z)) \vee (R(x) \wedge \exists w R(w))$$

3. Împingem negațiile în sub-fbf-uri pentru a crea literali, pe baza echivalențelor
 $\neg(\forall x W) \equiv \exists x \neg W$, $\neg(\exists x W) \equiv \forall x \neg W$, $\neg(A \wedge B) \equiv \neg A \vee \neg B$, $\neg(A \vee B) \equiv \neg A \wedge \neg B$, $\neg\neg A \equiv A$

$$\begin{aligned} &\equiv (\neg \forall y P(y) \wedge \neg \exists z Q(z)) \vee (R(x) \wedge \exists w R(w)) \\ &\equiv (\exists y \neg P(y) \wedge \forall z \neg Q(z)) \vee (R(x) \wedge \exists w R(w)) \end{aligned}$$

Construirea Formei Prenex CNF/DNF (cont.)

4. Mutăm cuantificatorii în stânga utilizând echivalențele condiționate

$$\forall x(V \vee W(x)) \equiv V \vee \forall x W(x), \exists x(V \vee W(x)) \equiv V \vee \exists x W(x), \forall x(V \wedge W(x)) \equiv V \wedge$$

$$\forall x W(x)$$

$$\exists x(V \wedge W(x)) \equiv V \wedge \exists x W(x)$$

$$\equiv (\exists y \neg P(y) \wedge \forall z \neg Q(z)) \vee (R(x) \wedge \exists w R(w))$$

$$\equiv \exists y (\neg P(y) \wedge \forall z \neg Q(z)) \vee (R(x) \wedge \exists w R(w))$$

$$\equiv \exists y ((\neg P(y) \wedge \forall z \neg Q(z)) \vee (R(x) \wedge \exists w R(w)))$$

$$\equiv \exists y (\forall z (\neg P(y) \wedge \neg Q(z)) \vee (R(x) \wedge \exists w R(w)))$$

$$\equiv \exists y \forall z ((\neg P(y) \wedge \neg Q(z)) \vee (R(x) \wedge \exists w R(w)))$$

$$\equiv \exists y \forall z \exists w ((\neg P(y) \wedge \neg Q(z)) \vee (R(x) \wedge R(w)))$$

$$\equiv \exists y \forall z \exists w ((\neg P(y) \wedge \neg Q(z)) \vee (R(x) \wedge R(w)))$$

5. Pentru obținerea prenex CNF distribuim \vee peste \wedge (pentru DNF \wedge peste \vee)

$$\equiv \exists y \forall z \exists w (((\neg P(y) \wedge \neg Q(z)) \vee R(x)) \wedge ((\neg P(y) \wedge \neg Q(z)) \vee R(w)))$$

$$\equiv \exists y \forall z \exists w ((\neg P(y) \vee R(x)) \wedge (\neg Q(z) \vee R(x)) \wedge (\neg P(y) \vee R(w)) \wedge (\neg Q(z) \vee R(w)))$$

Demonstrații în Logica de Ordinul Întâi

Există sisteme de deducție sound și complete

Reguli de Inferență Adiționale

Câteva exemple

$\exists x W(x)$	$W(x)$	$\forall x W(x)$	$\forall x$ $W(x)$
$\therefore W(c)$ <small>(dacă c e nouă în demonstrație)</small>	$\therefore \exists x W(x)$ <small>Existential Generalization (I)</small>	$\therefore W(x)$ <small>Universal Instantiation (1)</small>	$\therefore W(c)$ <small>(unde c este orice constantă)</small>
Existential Instantiation		Universal Instantiation	Universal Instantiation (II)

Demonstrații pe baza Rezoluției

Forma Clauzală

O clauză este o disjuncție de zero sau mai mulți literali

(Literal - un fbf atomic sau negata sa)

Exemple $P(x)$

$\neg Q(x, b)$

$\neg P(a) \vee P(b)$

Clauza goală (fără literali) e referită prin # și înseamnă fals

În esență, forma clauzală este forma normală prenex conjunctivă în care toți cuantificatorii sunt universali și nu există variabile libere

Construirea Formei Clauzale

Regula lui Skolem

Fie $\exists x W(x)$ parte dintr-un fbf mai mare:

Dacă $\exists x$ nu este în domeniul unui cuantificator universal, alegem o **constantă nouă** și înlocuim $\exists x W(x)$ cu $W(x/c)$

Dacă $\exists x$ este în domeniul lui $\forall x_1 \forall x_2 \dots \forall x_n$, atunci alegem un simbol de **funcție nouă** f

și înlocuim $\exists x W(x)$ cu $W(x/f(x_1, x_2, \dots, x_n))$

Forma obținută nu este neapărat echivalentă cu fbf original, dar fie sunt ambele realizabile fie ambele sunt nerealizabile

Pași de transformare

1. Construim forma normală conjunctivă prenex pentru W

$\exists y \forall z \exists w ((\neg P(y) \vee R(x)) \wedge (\neg Q(z) \vee R(x)) \wedge (\neg P(y) \vee R(w)) \wedge (\neg Q(z) \vee R(w)))$

2. Înlocuim toate variabilele libere cu constante noi

$\exists y \forall z \exists w ((\neg P(y) \vee R(a)) \wedge (\neg Q(z) \vee R(a)) \wedge (\neg P(y) \vee R(w)) \wedge (\neg Q(z) \vee R(w)))$

3. Utilizăm regula lui Skolem pentru a elimina quantificatorii existențiali

$\forall z \exists w ((\neg P(b) \vee R(a)) \wedge (\neg Q(z) \vee R(a)) \wedge (\neg P(b) \vee R(w)) \wedge (\neg Q(z) \vee R(w)))$

$\forall z ((\neg P(b) \vee R(a)) \wedge (\neg Q(z) \vee R(a)) \wedge (\neg P(b) \vee R(f(z))) \wedge (\neg Q(z) \vee R(f(z))))$

Reprezentarea Formei Clauzale

Obținem o formulă de forma

$$\forall x_1 \forall x_2 \dots \forall x_n (C_1 \wedge C_2 \wedge \dots \wedge C_n)$$

fără variabile libere și fără cuantificatori \exists

Reprezentare

$\{C_1, C_2, \dots, C_n\}$ în timp ce fiecare clauză C_i poate fi reprezentată ca și set de literali

Exemplu

$$\forall z ((\neg P(b) \vee R(a)) \wedge (\neg Q(z) \vee R(a)) \wedge (\neg P(b) \vee R(f(z))) \wedge (\neg Q(z) \vee R(f(z))))$$

$$\{\neg P(b) \vee R(a), \neg Q(z) \vee R(a), \neg P(b) \vee R(f(z)), \neg Q(z) \vee R(f(z))\}$$

$$\{\{\neg P(b), R(a)\}, \{\neg Q(z), R(a)\}, \{\neg P(b), R(f(z))\}, \{\neg Q(z), R(f(z))\}\}$$

Substituția

O substituție este un set de legări referite prin

$\Theta = \{x_1/t_1, x_2/t_2, \dots, x_n/t_n\}$ unde x_i sunt variabile distincte și $x_i \neq t_i$ pentru orice i

Fie C un literal/set de literali și Θ o substituție

Aplicarea substituției Θ lui C e referită prin $C\Theta$, iar rezultatul este expresia/expresiile obținute înlocuind / substituind toate aparițiile lui x_i cu termenul t_i

Exemple

$$C = \{P(x,y,f(x))\}, \Theta = \{x/a, y/f(b)\}$$

$$C\Theta = \{P(x,y,f(x))\{x/a, y/f(b)\}\} = \{P(a, f(b), f(a))\}$$

$$C = \{P(x,y), Q(a,y)\}, \Theta = \{x/a, y/f(b)\}$$

$$C\Theta = \{P(x,y), Q(a,y)\}\{x/a, y/f(b)\} = \{P(a,f(b)), Q(a,f(b))\}$$

Unificatorul

O substituție Θ este un **unificator** pentru un set de literali dacă $S\Theta$ are exact un element

Exemple

Avem unificatori pentru:

$\{p(x), q(y)\}$ NU

$\{p(x), \neg p(x)\}$ NU

$\{p(a), p(x)\}$

DA. $\Theta = \{x/a\}$. Să vedem: $\{p(a), p(x)\}\{x/a\} = \{p(a), p(a)\} = \{p(a)\}$

$\{p(x), p(y)\}$

DA, chiar mai mulți. $\{x/y\}, \{y/x\}, \{x/t, y/t\}$ pentru orice t

În esență, cel mai general unificator (cgu) este cel mai general set de legări ce poate fi găsit (există pași de determinare a lui)

Regula de Inferență a Rezoluției

Literali - ffb atomice (adică, predicate) sau negarea lor (ex. P(x), $\neg Q(y)$)

Doi literali se spune că sunt **complementi** dacă unul e negata celuilalt

$$\frac{a_1 \vee a_2 \vee \dots \vee a_i \vee \dots \vee a_n, b_1 \vee b_2 \vee \dots \vee b_j \vee \dots \vee b_m}{(a_1 \vee \dots \vee a_{i-1} \vee a_{i+1} \vee \dots \vee a_n \vee b_1 \vee \dots \vee b_{j-1} \vee b_{j+1} \vee \dots \vee b_m) \quad \Theta}$$

unde:

$a_1, \dots, a_n, b_1, \dots, b_m$ sunt literali

a_i, b_j sunt complementi și Θ este cel mai general unificator pentru ei (ex. $a_i = P(x), b_j = \neg P(y)$ și $\text{UNIFY}(P(x), P(y)) = \Theta$)

Rezultatul este rezolventul

Notă. Pentru simplitate, presupunem că clauzele sunt reprezentate ca seturi și deci, disjuncțiile redundante sunt eliminate. Verificăm dacă cele două clauze au variabile distincte (redenumim dacă e necesar)

Demonstrarea prin Rezoluție

Pentru a demonstra că W este valid

1. Formăm negata $\neg W$. De exemplu, dacă W are forma $A \wedge B \wedge C \rightarrow D$ atunci $\neg W$ va fi $A \wedge B \wedge C \wedge \neg D$
2. Aducem la forma clauzală
3. Luăm clauzele ca premise
4. Aplicăm regula rezoluției pentru a deriva clauza vidă (fals)

Dacă e găsită, W este valid (din moment ce $\neg W$ nu este). Dacă nu putem crea noi rezolvenți, nu e validă. Terminarea procedurii nu e garantată.

Exemplul 1

$P(a,b) \wedge P(c,b) \wedge P(b,d) \wedge P(a,e) \wedge (P(x,z) \wedge P(z,y) \rightarrow G(x,y)) \rightarrow G(a,d)$

$$\begin{aligned}\neg W &= \neg(P(a,b) \wedge P(c,b) \wedge P(b,d) \wedge P(a,e) \wedge (P(x,z) \wedge P(z,y) \rightarrow G(x,y)) \rightarrow G(a,d)) \\ &= \neg(\neg(P(a,b) \wedge P(c,b) \wedge P(b,d) \wedge P(a,e) \wedge (P(x,z) \wedge P(z,y) \rightarrow G(x,y))) \vee G(a,d)) \\ &= P(a,b) \wedge P(c,b) \wedge P(b,d) \wedge P(a,e) \wedge (P(x,z) \wedge P(z,y) \rightarrow G(x,y)) \wedge \neg G(a,d) \\ &= P(a,b) \wedge P(c,b) \wedge P(b,d) \wedge P(a,e) \wedge (\neg(P(x,z) \wedge P(z,y)) \vee G(x,y)) \wedge \neg G(a,d) \\ &= P(a,b) \wedge P(c,b) \wedge P(b,d) \wedge P(a,e) \wedge (\neg P(x,z) \vee \neg P(z,y) \vee G(x,y)) \wedge \neg G(a,d)\end{aligned}$$

- | | |
|---|-------------------|
| 1. $P(a,b)$ | P P P P P |
| 2. $P(c,b)$ | P (negarea |
| 3. $P(b,d)$ | concluziei) 5,6,R |
| 4. $P(a,e)$ | {x/a,y/d} |
| 5. $\neg P(x,z) \vee \neg P(z,y) \vee G(x,y)$ | 1,7,R {z/b} |
| 6. $\neg G(a,d)$ | 3,8,R {} |
| 7. $\neg P(a,z) \vee \neg P(z,d)$ | |
| 8. $\neg P(b,d)$ | |
| 9. # | |

Deci, implicația inițială e validă

Formalizarea Propozițiilor din Limbaj Natural

Socrates nu deține un computer.

$\neg P_c(Socrates)$

$P_c(x)$ "x deține un computer"

Socrates nu este un specialist în știința
calculatoarelor.

$\neg C_s(Socrates)$

$C_s(x)$ "x este un specialist în știința
calculatoarelor"

Toți specialistii în știința
calculatoarelor dețin un computer.

$\forall x (C_s(x) \rightarrow P_c(x))$

Unii politicieni sunt coruși.

$\exists x (P(x) \wedge Q(x))$

$P(x)$ "x este politician"

$Q(x)$ "x este corupt"

Niciun politician nu e corupt.

$\forall x (P(x) \rightarrow \neg Q(x))$

Toți politicienii sunt coruși. Nu

$\forall x (P(x) \rightarrow Q(x))$

toți politicienii sunt coruși.

$\exists x (P(x) \wedge \neg Q(x))$

...

Cuantificatorii universalii cuantifică o implicație.

Cuantificatorii existențiali cuantifică o conjuncție.

Exemplul 2

Jack deține un câine. Orice deținător de câini este un iubitor de animale. Nici un iubitor de animale nu omoară un animal. Fie Jack sau Curiosity au omorât pisica a cărei nume e Tuna. Cine a omorât pisica ?

- A. $\exists x (\text{Dog}(x) \wedge \text{Owns}(\text{Jack}, x))$
- B. $\forall x (\exists y (\text{Dog}(y) \wedge \text{Owns}(x, y)) \wedge \neg \text{AnimalLover}(x))$
- C. $\forall x (\text{AnimalLover}(x) \wedge \forall y (\text{Animal}(y) \wedge \neg \text{Kills}(x, y)))$
- D. $\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})$
- E. $\text{Cat}(\text{Tuna})$
- F. $\forall x (\text{Cat}(x) \wedge \neg \text{Animal}(x))$
- G. $\exists x \text{ Kills}(x, \text{Tuna})$ unde Jack, Tuna, Curiosity sunt constante

$\exists x (\text{Dog}(x) \wedge \text{Owns}(\text{Jack}, x)) \wedge$
 $\forall x (\exists y (\text{Dog}(y) \wedge \text{Owns}(x, y)) \wedge \neg \text{AnimalLover}(x)) \wedge$
 $\forall x (\text{AnimalLover}(x) \wedge \forall y (\text{Animal}(y) \wedge \neg \text{Kills}(x, y))) \wedge$
 $(\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})) \wedge$
 $\text{Cat}(\text{Tuna}) \wedge$
 $\forall x (\text{Cat}(x) \wedge \neg \text{Animal}(x)) !$
 $\exists x \text{ Kills}(x, \text{Tuna})$

Exemplul 2 (cont.)

Negăm și eliminăm ultima implicație

$\exists x (\text{Dog}(x) \wedge \text{Owns}(\text{Jack}, x)) \wedge$
 $\forall x (\exists y (\text{Dog}(y) \wedge \text{Owns}(x, y)) \rightarrow \text{AnimalLover}(x)) \wedge$
 $\forall x (\text{AnimalLover}(x) \rightarrow \forall y (\text{Animal}(y) \rightarrow \neg \text{Kills}(x, y))) \wedge$
 $(\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})) \wedge$
 $\text{Cat}(\text{Tuna}) \wedge$
 $\forall x (\text{Cat}(x) \rightarrow \text{Animal}(x)) \wedge$
 $\neg \exists x \text{ Kills}(x, \text{Tuna})$

Redenumim

$\exists x (\text{Dog}(x) \wedge \text{Owns}(\text{Jack}, x)) \wedge$
 $\forall z (\exists y (\text{Dog}(y) \wedge \text{Owns}(z, y)) \rightarrow \text{AnimalLover}(z)) \wedge$
 $\forall t (\text{AnimalLover}(t) \rightarrow \forall v (\text{Animal}(v) \rightarrow \neg \text{Kills}(t, v))) \wedge$
 $(\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})) \wedge$
 $\text{Cat}(\text{Tuna}) \wedge$
 $\forall r (\text{Cat}(r) \rightarrow \text{Animal}(r)) \wedge$
 $\neg \exists w \text{ Kills}(w, \text{Tuna})$

Exemplul 2 (cont.)

Înlocuim implicațiile

$\exists x (\text{Dog}(x) \wedge \text{Owns}(\text{Jack}, x)) \textcolor{red}{\wedge}$
 $\forall z (\neg \exists y (\text{Dog}(y) \wedge \text{Owns}(z, y)) \vee \text{AnimalLover}(z)) \textcolor{red}{\wedge}$
 $\forall t (\neg \text{AnimalLover}(t) \vee \forall v (\neg \text{Animal}(v) \vee \neg \text{Kills}(t, v))) \textcolor{red}{\wedge}$
 $(\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})) \textcolor{red}{\wedge}$
 $\text{Cat}(\text{Tuna}) \textcolor{red}{\wedge}$
 $\forall r (\neg \text{Cat}(r) \vee \text{Animal}(r)) \textcolor{red}{\wedge}$
 $\neg \exists w \text{ Kills}(w, \text{Tuna})$

Mutăm negațiile

$\exists x (\text{Dog}(x) \wedge \text{Owns}(\text{Jack}, x)) \textcolor{red}{\wedge}$
 $\forall z (\forall y (\neg \text{Dog}(y) \vee \neg \text{Owns}(z, y)) \vee \text{AnimalLover}(z)) \textcolor{red}{\wedge}$
 $\forall t (\neg \text{AnimalLover}(t) \vee \forall v (\neg \text{Animal}(v) \vee \neg \text{Kills}(t, v))) \textcolor{red}{\wedge}$
 $(\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})) \textcolor{red}{\wedge}$
 $\text{Cat}(\text{Tuna}) \textcolor{red}{\wedge}$
 $\forall r (\neg \text{Cat}(r) \vee \text{Animal}(r)) \textcolor{red}{\wedge}$
 $\forall w \neg \text{Kills}(w, \text{Tuna})$

Exemplul 2 (cont.)

Extragem Cuantificatorii

$$\exists x \forall z \forall y \forall t \forall v \forall r \forall w ($$

$$(\text{Dog}(x) \wedge \text{Owns}(\text{Jack}, x)) \wedge$$

$$(\neg\text{Dog}(y) \vee \neg\text{Owns}(z, y) \vee \text{AnimalLover}(z)) \wedge$$

$$(\neg\text{AnimalLover}(t) \vee \neg\text{Animal}(v) \vee \neg\text{Kills}(t, v)) \wedge$$

$$(\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})) \wedge$$

$$\text{Cat}(\text{Tuna}) \wedge$$

$$(\neg\text{Cat}(r) \vee \text{Animal}(r)) \wedge$$

$$\neg\text{Kills}(w, \text{Tuna})$$

)

Eliminăm cuantificatorii \exists

$$\forall z \forall y \forall t \forall v \forall r \forall w ($$

$$(\text{Dog}(\text{SomeDog}) \wedge \text{Owns}(\text{Jack}, \text{SomeDog})) \wedge$$

$$(\neg\text{Dog}(y) \vee \neg\text{Owns}(z, y) \vee \text{AnimalLover}(z)) \wedge$$

$$(\neg\text{AnimalLover}(t) \vee \neg\text{Animal}(v) \vee \neg\text{Kills}(t, v)) \wedge$$

$$(\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})) \wedge$$

$$\text{Cat}(\text{Tuna}) \wedge$$

$$(\neg\text{Cat}(r) \vee \text{Animal}(r)) \wedge$$

$$\neg\text{Kills}(w, \text{Tuna})$$

)

Exemplul 2 (cont.)

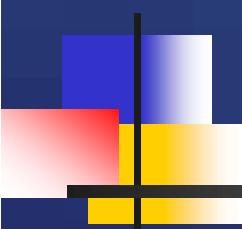
Reținem doar cauzele

Dog(SomeDog) \wedge
Owns(Jack, SomeDog) \wedge
 $(\neg \text{Dog}(y) \vee \neg \text{Owns}(z, y) \vee \text{AnimalLover}(z)) \wedge$
 $(\neg \text{AnimalLover}(t) \vee \neg \text{Animal}(v) \vee \neg \text{Kills}(t, v)) \wedge$
 $(\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})) \wedge$
Cat(Tuna) \wedge
 $(\neg \text{Cat}(r) \vee \text{Animal}(r)) \wedge$
 $\neg \text{Kills}(w, \text{Tuna})$

Exemplul 2 (cont.)

1. Dog(SomeDog)	P P P P P P P P
2. Owns(Jack, SomeDog)	5, 8, Res {w/Curiosity}
3. \neg Dog(y) v \neg Owns(z, y) v AnimalLover(z)	6,7, Res {r/Tuna}
4. \neg AnimalLover(t) v \neg Animal(v) v \neg Kills(t,v)	10,4, Res {v/Tuna}
5. Kills(Jack,Tuna) v Kills(Curiosity,Tuna)	1,3, Res {y/SomeDog}
6. Cat(Tuna)	12,2,Res {z/Jack}
7. \neg Cat(r) v Animal(r)	11,13,Res {t/Jack}
8. \neg Kills(w,Tuna)	9,14,Res {}
9. Kills(Jack,Tuna)	
10. Animal(Tuna)	
11. \neg AnimalLover(t) v \neg Kills(t,Tuna)	
12. \neg Owns(z, SomeDog) v AnimalLover(z)	
13. AnimalLover(Jack)	
14. \neg Kills(Jack,Tuna)	
15. #	

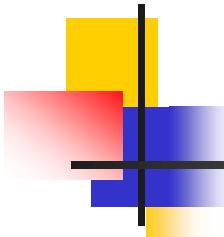
Deci, implicația inițială e validă și Curiosity (w/Curiosity) a omorât-o pe Tuna.



Logică și structuri discrete

**Universitatea Politehnica Timișoara
Anul universitar 2018-2019**

Sl.dr.ing.Mirella Amelia MIOC

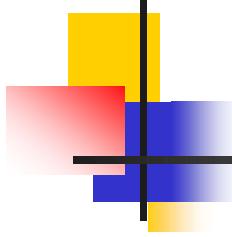


Curs 12

Grafuri

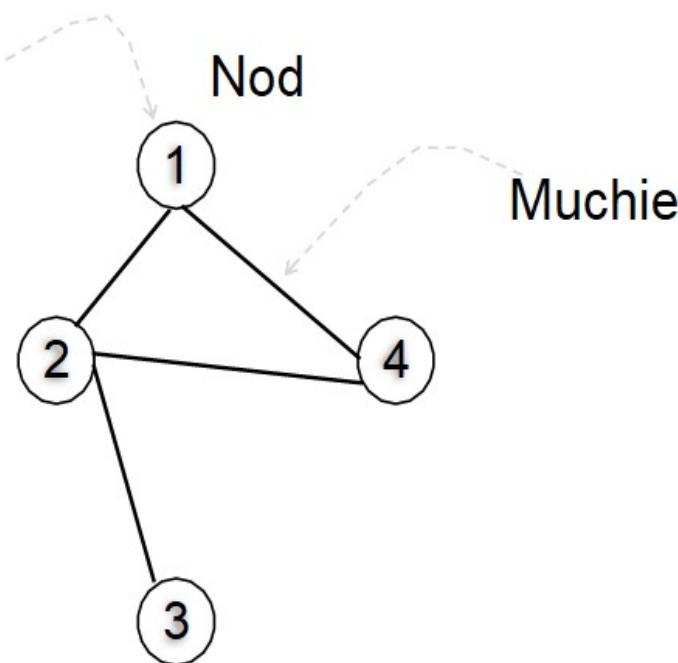
Noțiuni Generale

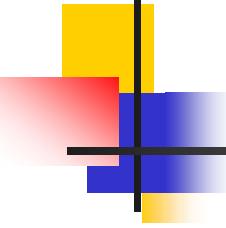
- Tipuri de grafuri
- Traversare



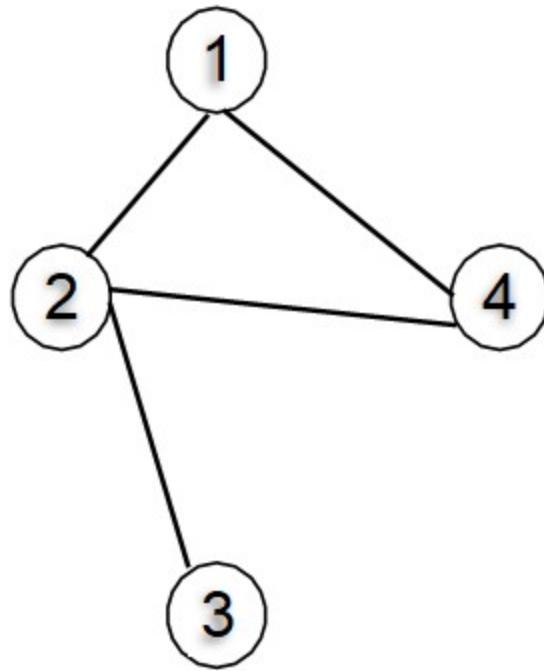
Graf

- Un graf este un set de obiecte interconectate
- Obiectele sunt noduri
- Conexiunile sunt muchii

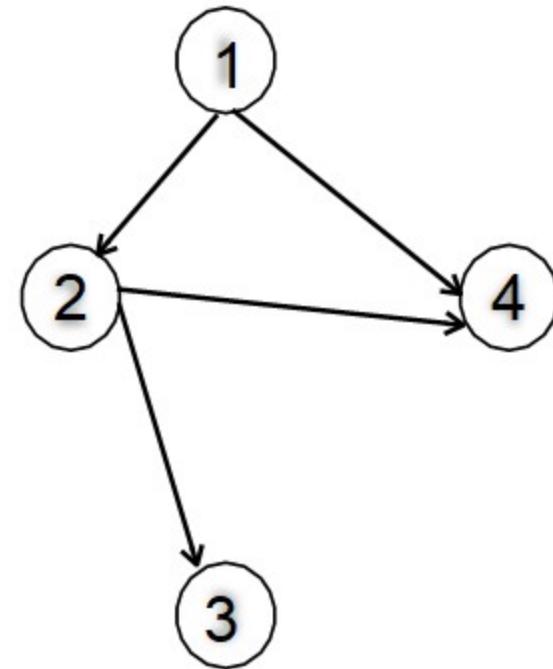




Grafuri neorientate și grafuri orientate



Graf neorientat
muchia este o pereche
neordonată



Graf orientat
muchia este o
pereche ordonată

Reprezentare graf neorientat

$G = (V, E)$ (adică o pereche ordonată ori 2-tuplu)

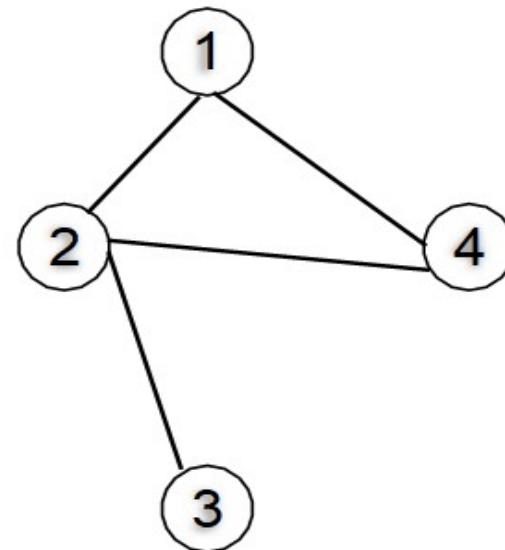
$V = \{1, 2, 3, 4\}$ (adică un set de noduri)

$E = \{\{1,2\}, \{2,3\}, \{2,4\}, \{1,4\}\}$

(adică setul de muchii, fiecare muchie fiind un set)

Notă: Alternativ, putem avea

$E = \{(1,2), (2,3), (2,4), (1,4)\}$ considerând că perechile nu sunt ordonate



Reprezentare graf orientat

$$G = (V, E)$$

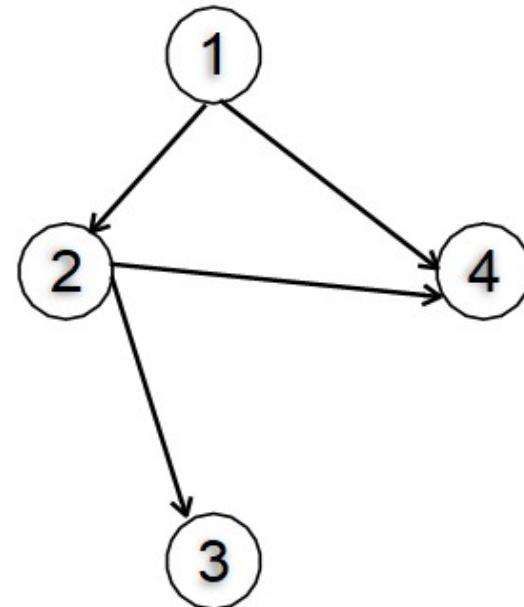
$$V = \{1, 2, 3, 4\}$$

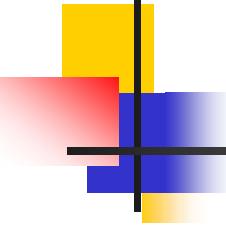
$$E = \{(1,2), (2,3), (2,4), (1,4)\}$$

(adică setul de muchii, fiecare muchie fiind o pereche ordonată)

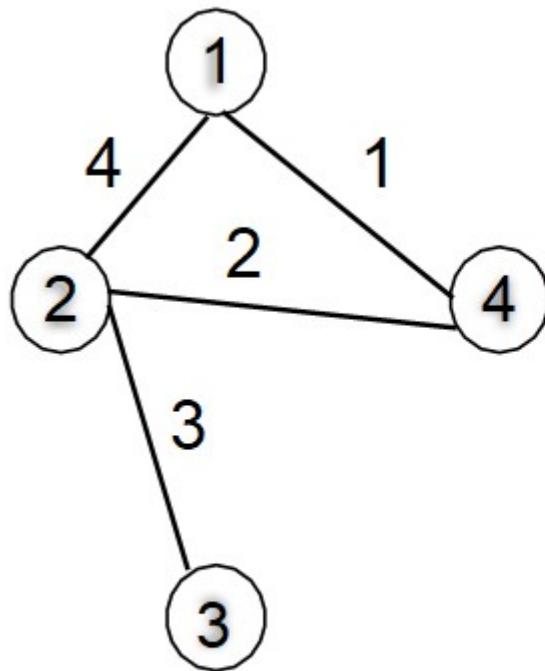
Observați că E este o relație binară peste V (adică $E \subset V \times V$).

Deci orice relație binară poate fi analizată / văzută ca un graf orientat !

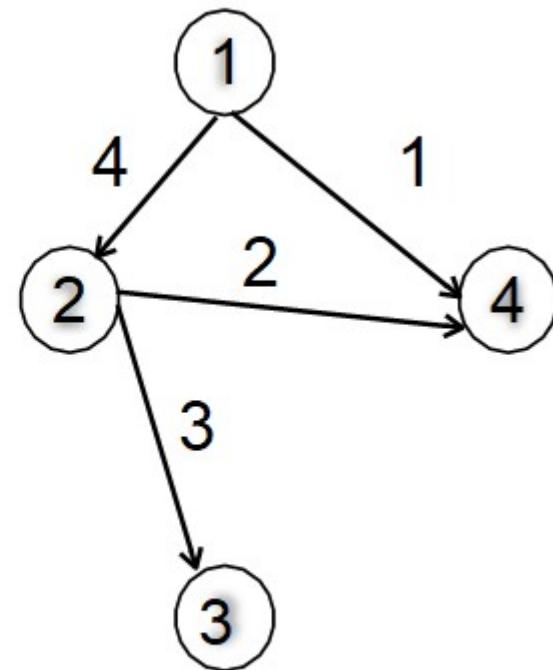




Graf ponderat (numar asociat muchiei)



Graf ponderat neorientat



Graf ponderat orientat

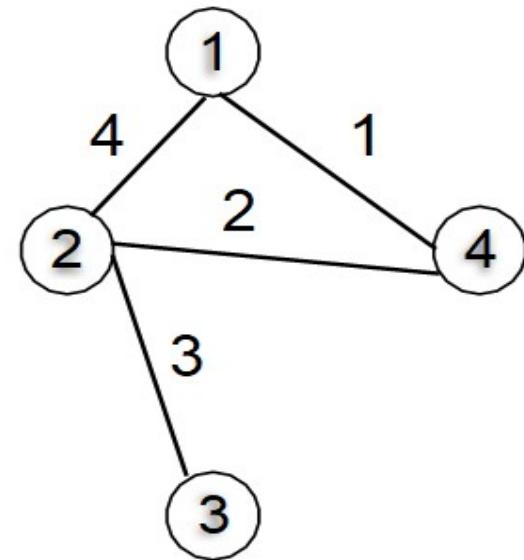
Reprezentare graf ponderat neorientat

$$G = (V, E)$$

$$V = \{1, 2, 3, 4\}$$

$$E = \{(\{1,2\}, 4), (\{2,3\}, 3), (\{2,4\}, 2), (\{1,4\}, 1)\}$$

Fiecare muchie între a și b e reprezentată ca o pereche ordonată
 $(\{a,b\}, \text{cost})$



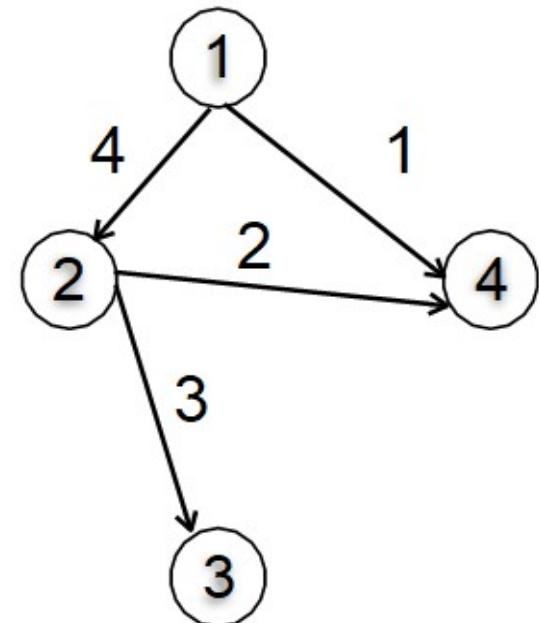
Reprezentare graf ponderat orientat

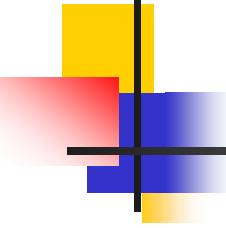
$$G = (V, E)$$

$$V = \{1, 2, 3, 4\}$$

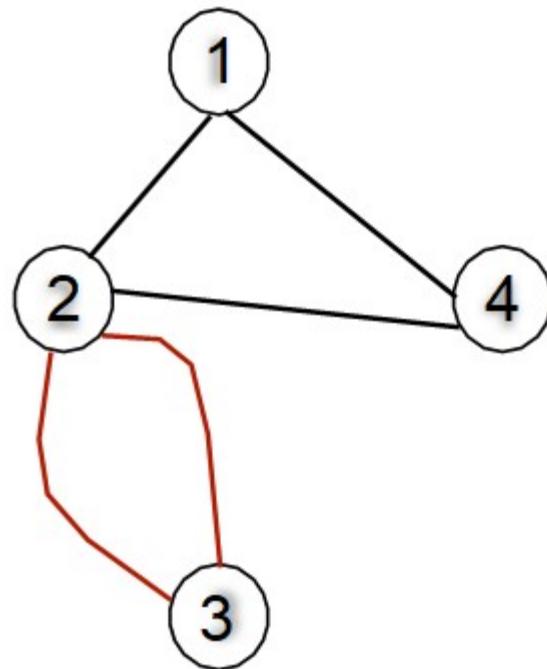
$$E = \{(1,2,4), (2,3,3), (2,4,2), (1,4,1)\}$$

Fiecare muchie între a și b e reprezentată ca un triplet ordonat (a,b,cost)

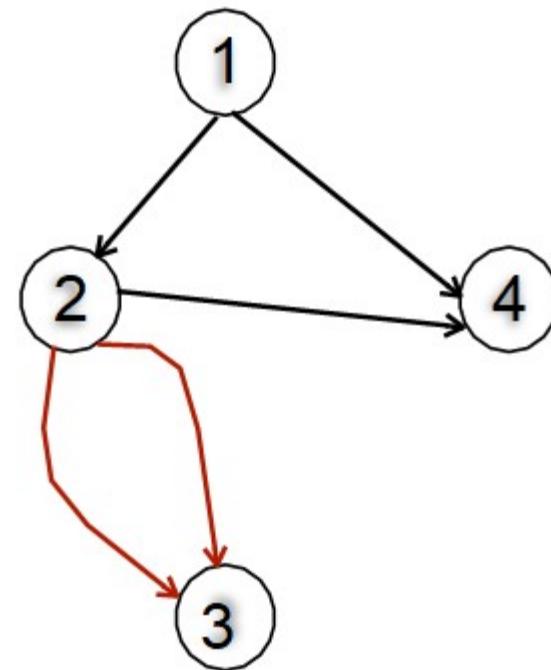




Multigraf (mai multe muchii între aceleasi două noduri)



Multigraf
neorientat



Multigraf orientat
(muchiile au acelasi sens)

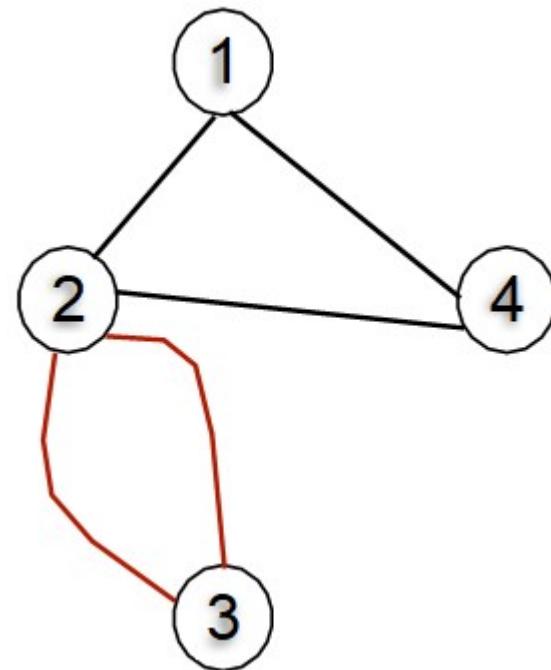
Reprezentare multigraf neorientat

$$G = (V, E)$$

$$V = \{1, 2, 3, 4\}$$

$$E = [\{1,2\}, \{2,3\}, \{2,3\}, \{2,4\}, \{1,4\}]$$

E este un multiset / multimultime



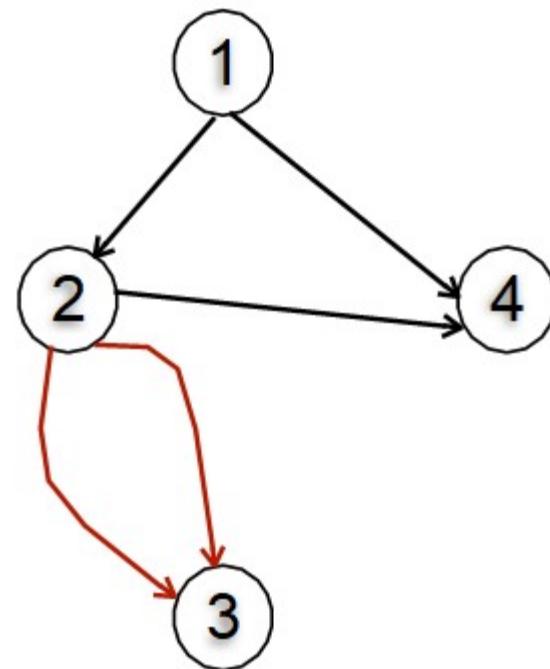
Reprezentare multigraf orientat

$$G = (V, E)$$

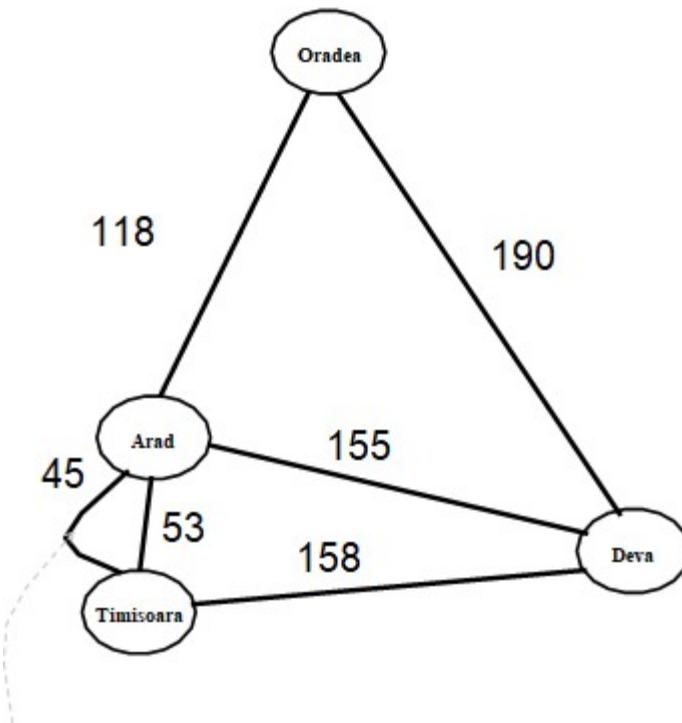
$$V = \{1, 2, 3, 4\}$$

$$E = [(1,2), (2,3), (2,3), (2,4), (1,4)]$$

E este un multiset / multimulțime



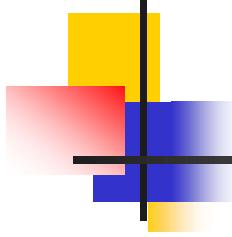
Exemplu utilizare grafuri



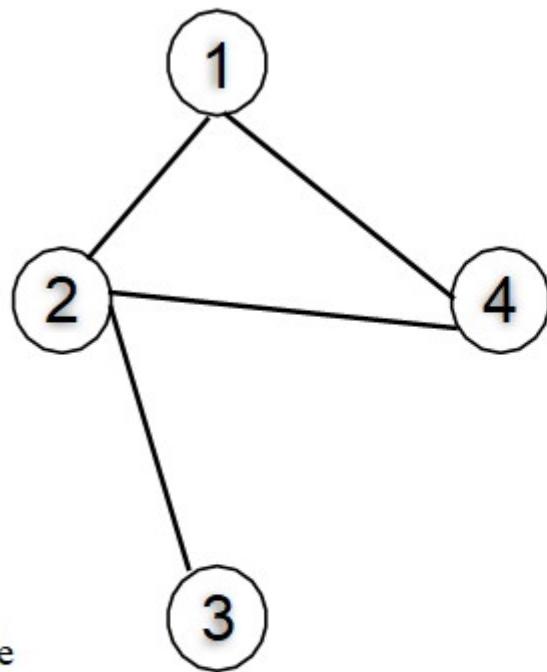


Drum

Un drum de la nodul x_0 la nodul x_n este o sevență de muchii referită prin secvența de noduri x_0, x_1, \dots, x_n astfel încât există o muchie de la x_{i-1} la x_i pentru $1 \leq i \leq n$. n este lungimea drumului (adică numărul de muchii traversate)



Drum

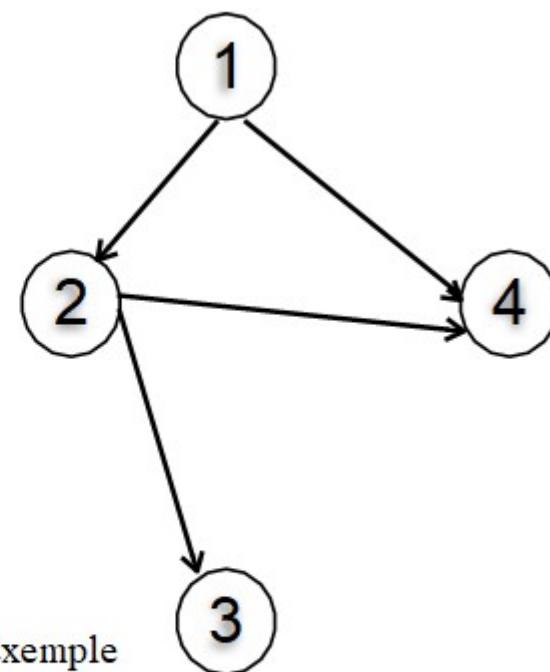


Exemplu

1, 2, 3 de lungime 2

1, 4, 2, 3 de lungime 3

(Notă: muchiile pot fi traversate în orice sens)

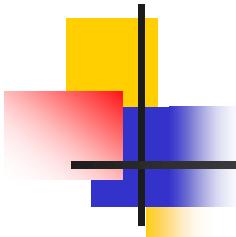


Exemplu

1,2,3 de lungime 2

1,2,4 de lungime 2

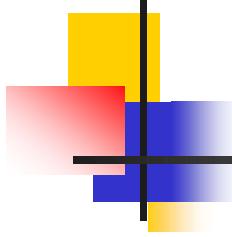
(Notă: muchiile pot fi traversate doar în sensul indicat)



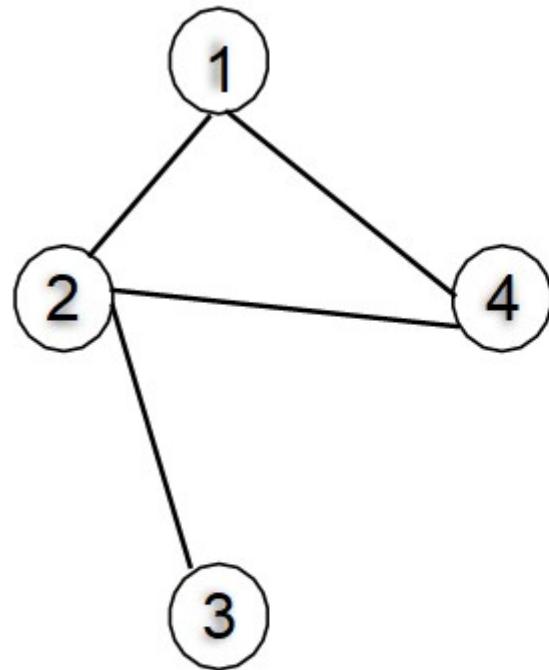
Cicluri

Un ciclu (ori circuit) este un drum ale cărui noduri de început și sfârșit sunt egale și în care nici o muchie nu apare de mai multe ori.

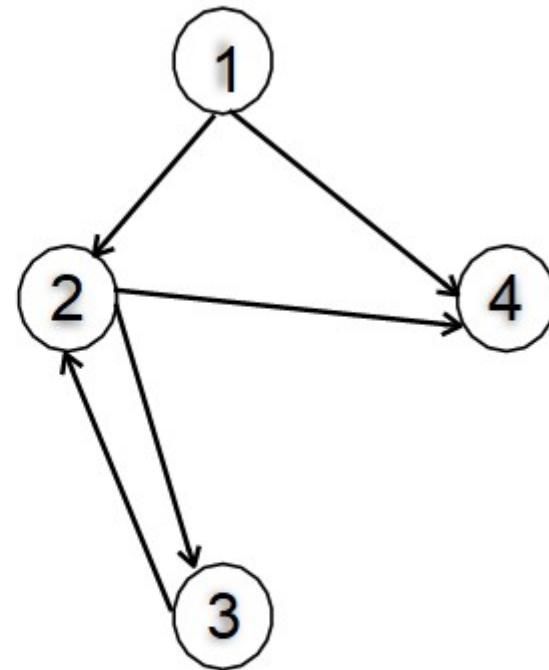
Un graf fără cicluri se numește aciclic.



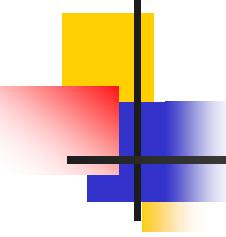
Cicluri



Exemplu
1,2,4,1

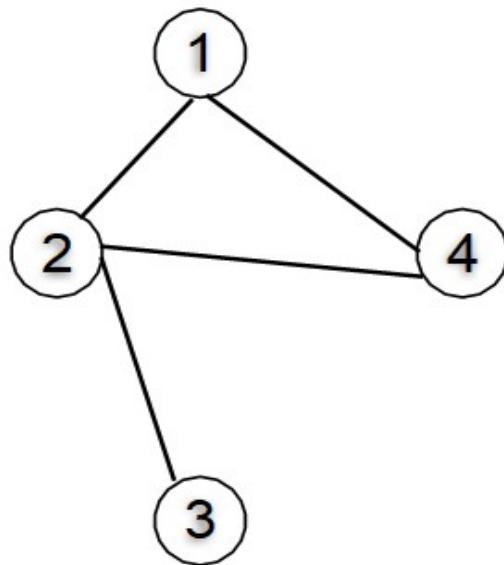


Exemplu
2,3,2

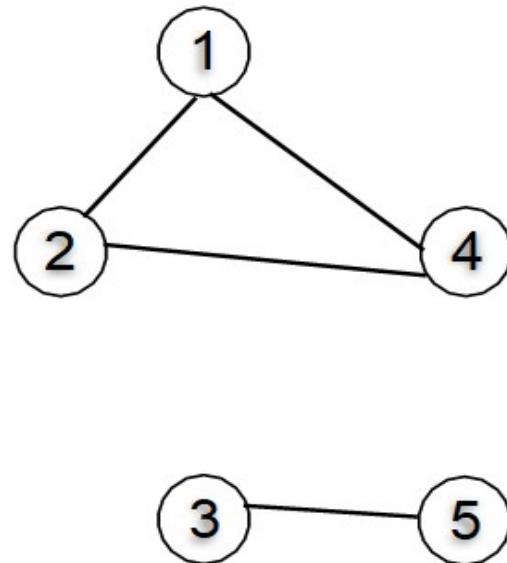


Grafuri neorientate conexe

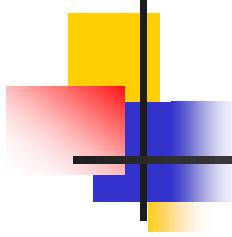
Un graf neorientate este conex dacă există un drum între orice pereche de noduri.



Conex

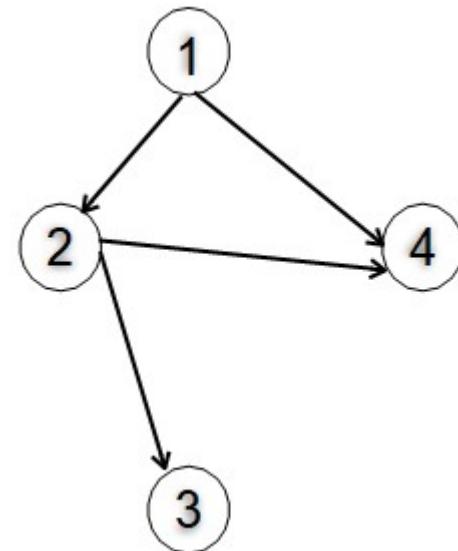


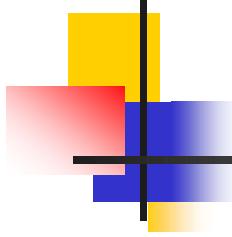
Nu e conex



Grafuri orientate (slab) conexe

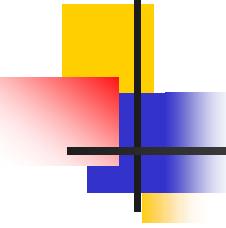
Un graf orientat este (slab) conex dacă înlocuind muchiile orientate (numite și arce) cu muchii neorientate graful neorientat obținut este conex



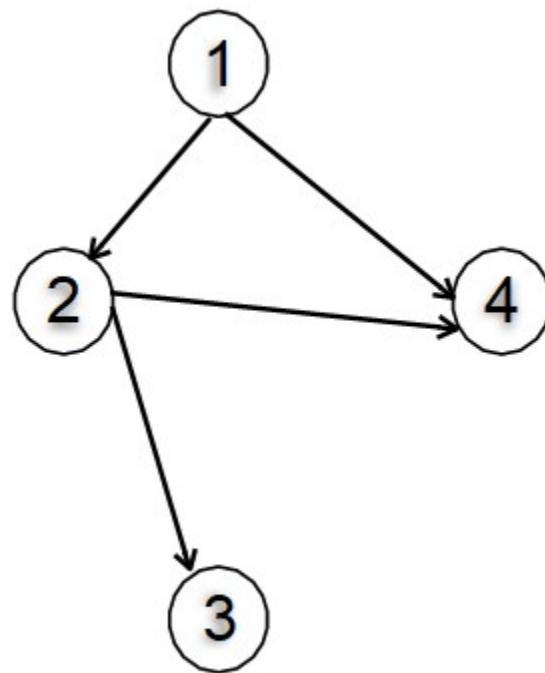


Grafuri orientate tare conexe

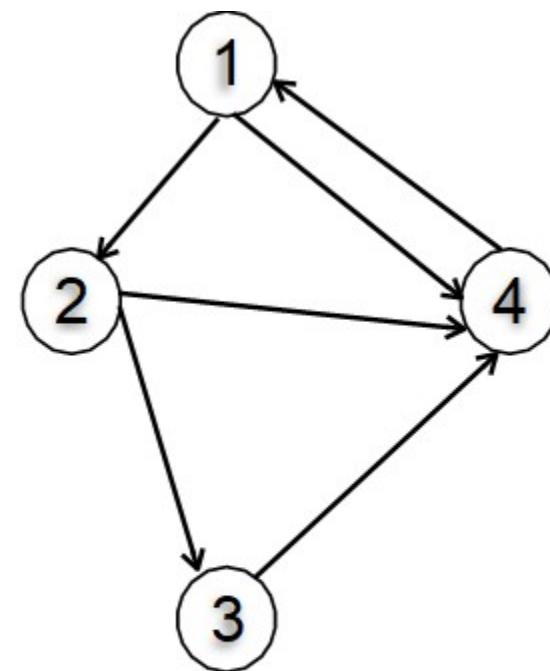
Un graf orientat este tare conex dacă și numai dacă există un drum de la v la u și un drum de la u la v pentru orice pereche de noduri u, v .



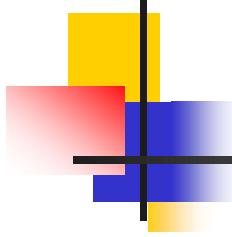
Grafuri orientate tare conexe



Nu e tare conex

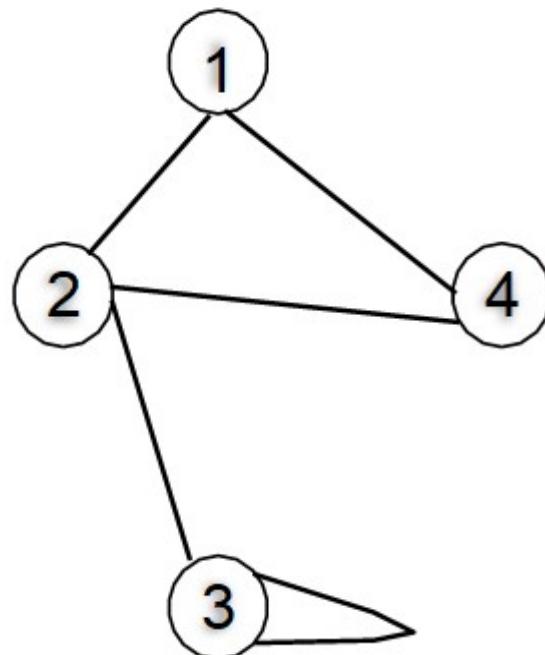


E tare conex



Gradul

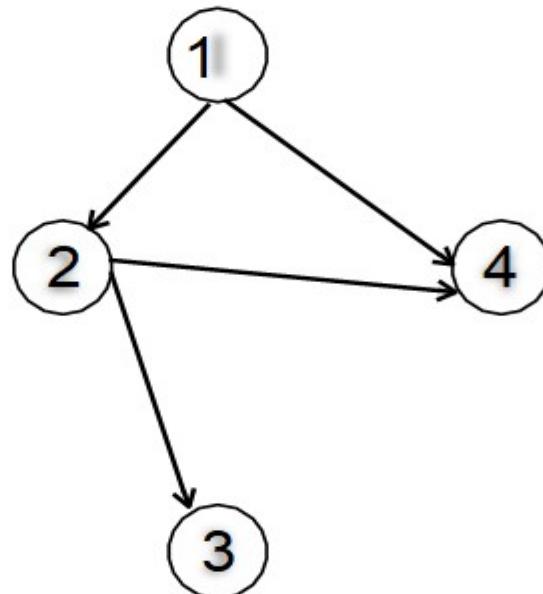
Într-un graf neorientat gradul unui nod este numărul de muchii pe care le atinge nodul.



Exemplu $\text{degree}(4) = 2$
 $\text{degree}(3) = 3$ (adunăm 2 în caz de buclă)

Gradul de intrare; gradul de ieșire

Într-un graf neorientat gradul unui nod este numărul de muchii pe care le atinge nodul.

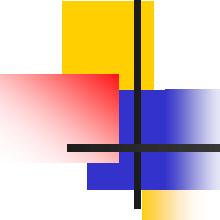


Exemplu

$\text{InDegree}(1) = 0$; $\text{OutDegree}(1) = 2$
 $\text{InDegree}(4) = 2$; $\text{OutDegree}(4) = 0$

Când $\text{InDegree}(v) = 0$, v se numește sursă

Când $\text{OutDegree}(v) = 0$, v se numește scurgere



Drumuri speciale

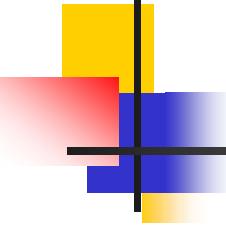
Un drum ce conține toate muchiile unui graf exact o dată se numește drum Eulerian.

Un ciclu Eulerian este un drum ce începe și se termină în același nod și în care fiecare muchie e traversată exact o dată.

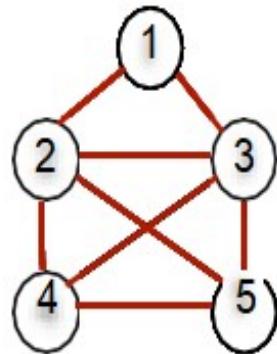
Conditii de existență în (multi) grafuri neorientate.

Există un ciclu Eulerian dacă și numai dacă toate nodurile au grad par.

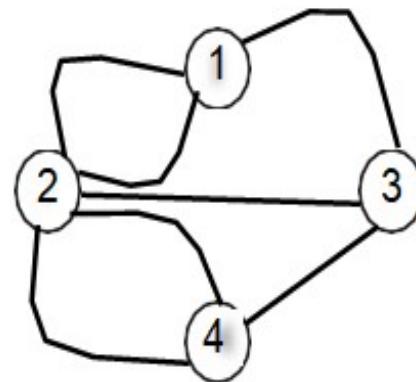
Există o cale Euleriană dacă graful are exact două noduri cu grad impar.



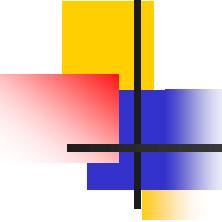
Drumuri speciale



Are drum Eulerian (ex.
4,2,1,3,2,5,3,4,5) dar nu ciclu
Eulerian



Nu are cale Euleriană și nici ciclu Eulerian

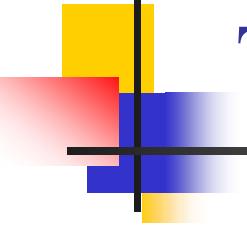


Traversarea Grafurilor

O traversare a unui graf începe într-un nod v și vizitează toate nodurile x ce pot fi atinse de la v călătorind pe un drum de la v la x. Dacă un nod a fost deja vizitat nu mai este vizitat încă o dată.

Cele mai des întâlnite moduri de traversare:

- Breadth-First (în lățime)
- Depth-First (în adâncime)



Traversarea Breadth-First (în lățime)

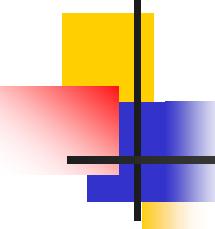
visit(v,k) este o procedură care vizitează fiecare nod x, nevizitat încă, pentru care există un drum de lungime k de la v la x.

Breadth-First:

```
for k := 0 to n - 1 do visit(v,k) od
```

//n - număr de noduri

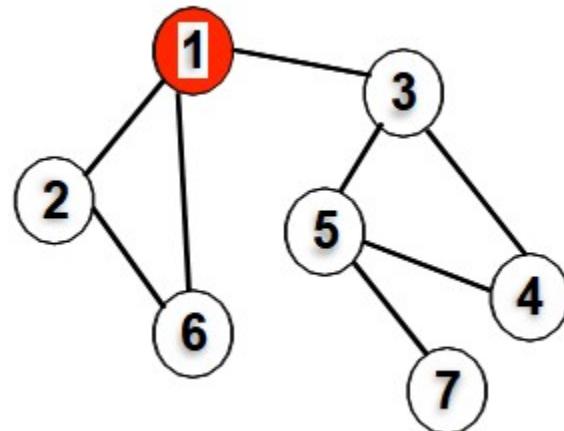
//v - nodul de start



Exemplu – pasul 1

$\text{visit}(1,0) = \{1\}$

Rezultat pentru $v = 1$: 1

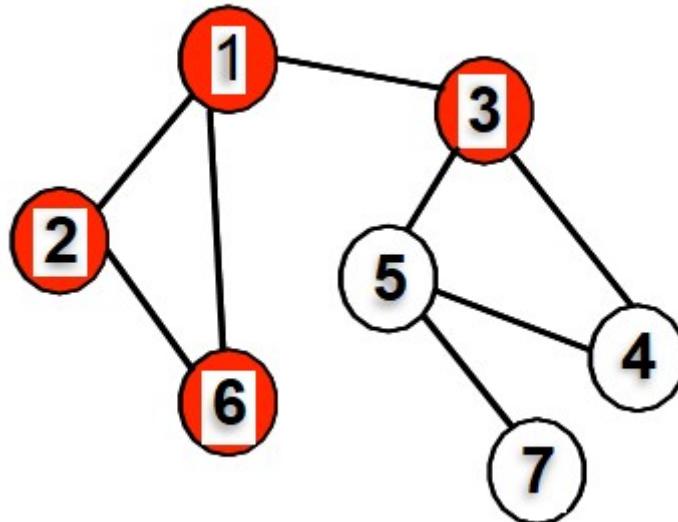


Exemplu – pasul 2

$\text{visit}(1,0) = \{1\}$

$\text{visit}(1,1) = \{2,6,3\}$

Rezultat pentru $v = 1: 1, 2, 6, 3$



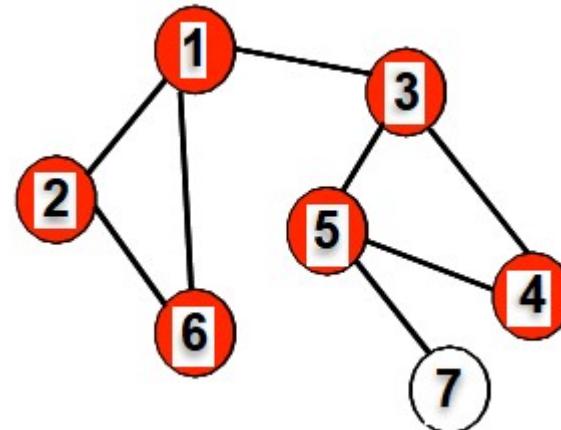
Exemplu – pasul 3

$\text{visit}(1,0) = \{1\}$

$\text{visit}(1,1) = \{2,6,3\}$

$\text{visit}(1,2) = \{5,4\}$

Rezultat pentru $v = 1: 1, 2, 6, 3, 5, 4$



Exemplu – pasul 4

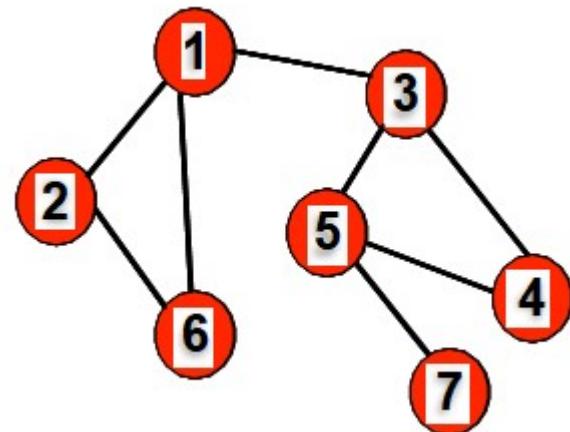
$\text{visit}(1,0) = \{1\}$

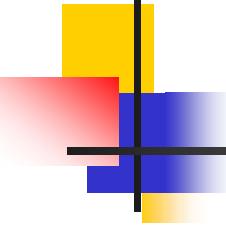
$\text{visit}(1,1) = \{2,6,3\}$

$\text{visit}(1,2) = \{5,4\}$

$\text{visit}(1,3) = \{7\}$

Rezultat pentru $v = 1: 1, 2, 6, 3, 5, 4, 7$





Traversarea Depth-First (în adâncime)

Depth-First(v):

 if v nu a fost vizitat then

 visit v; // marchează nodul ca vizitat, dar
 în lucru (gri)

 for fiecare muchie de la v la x do Depth-
 First(x); od

 // marchează nodul ca terminat (albastru)

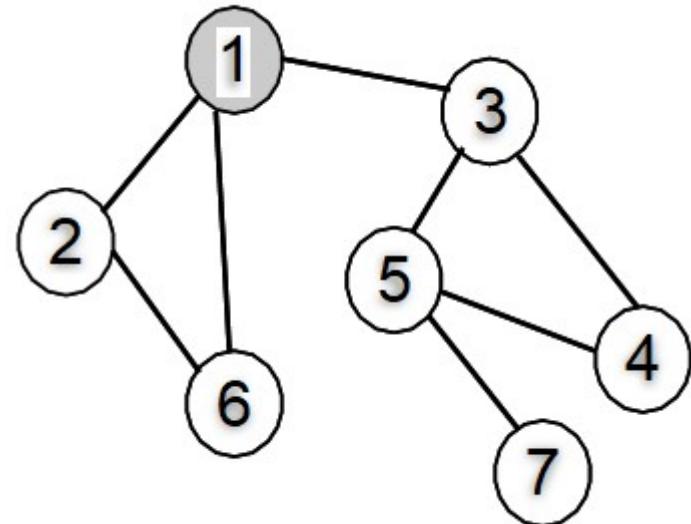
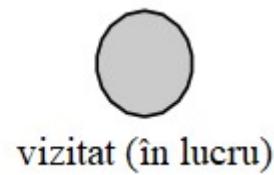
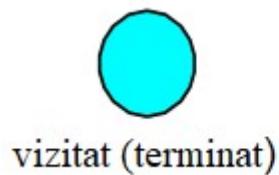
fi;

Exemplu – pasul 1

Stivă de urmărire

$df(1) - \{(1,2),(1,6),(1,3)\}$

Rezultatul pornind de la 1: 1



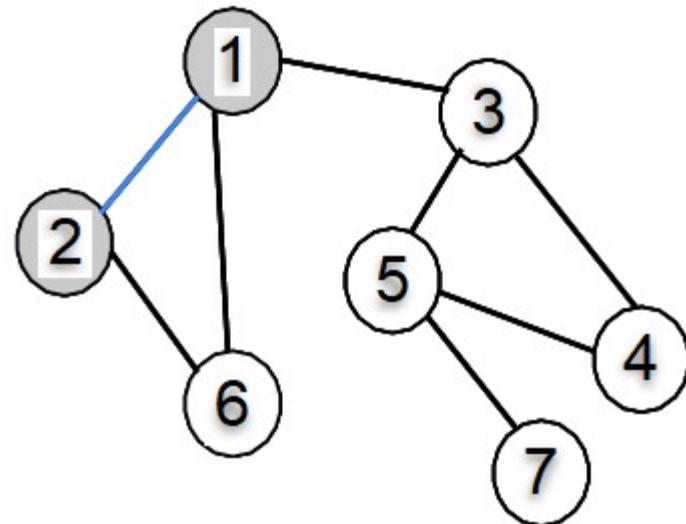
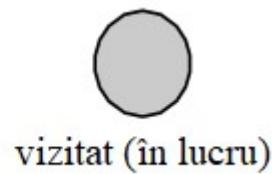
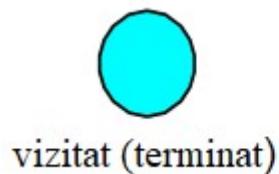
Exemplu – pasul 2

Stivă de urmărire

$$df(2) - \{(2,6),(1,2)\}$$

$$df(1) - \{(1,2),(1,6),(1,3)\}$$

Rezultatul pornind de la 1: 1, 2



Exemplu – pasul 3

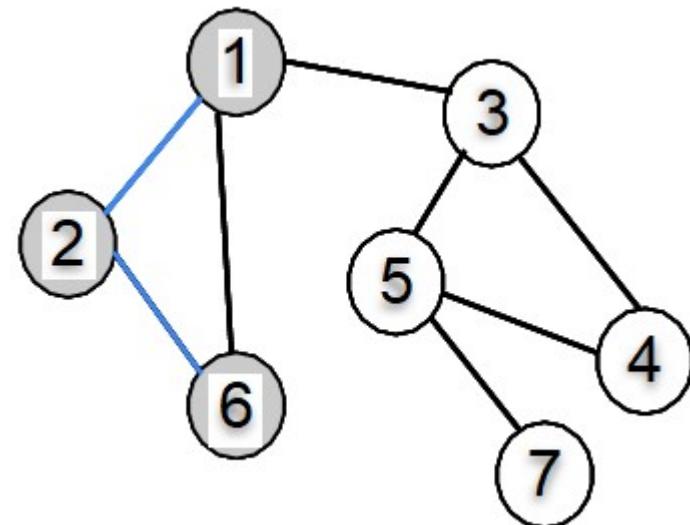
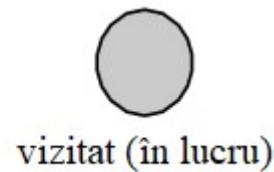
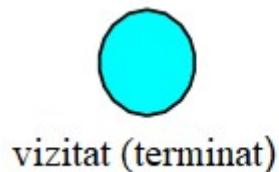
Stivă de urmărire

$df(6) - \{(2,6),(1,6)\}$

$df(2) - \{(2,6),(1,2)\}$

$df(1) - \{(1,2),(1,6),(1,3)\}$

Rezultatul pornind de la 1: 1, 2, 6



Exemplu – pasul 4

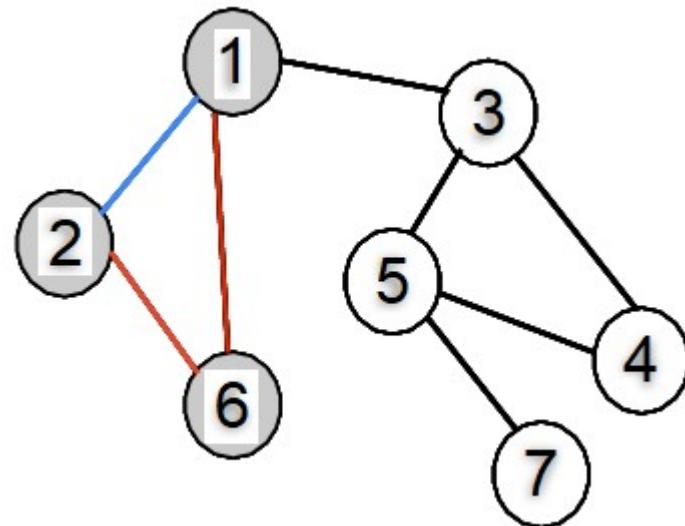
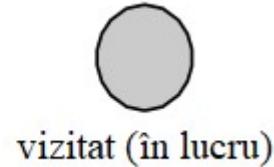
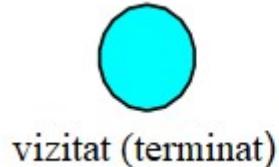
Stivă de urmărire

$df(6) - \{(2,6),(1,6)\}$

$df(2) - \{(2,6),(1,2)\}$

$df(1) - \{(1,2),(1,6),(1,3)\}$

Rezultatul pornind de la 1: 1, 2, 6



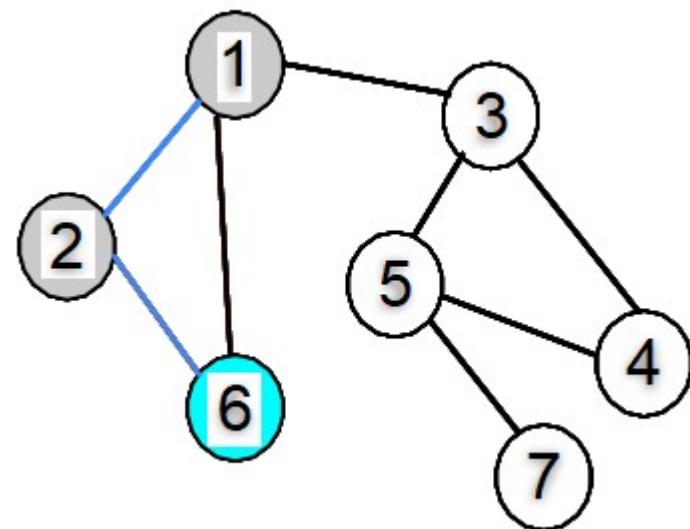
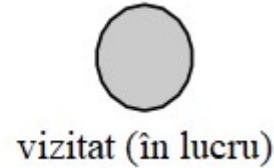
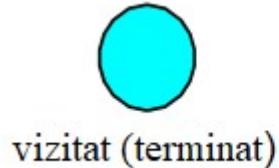
Exemplu – pasul 5

Stivă de urmărire

$$df(2) - \{(2,6),(1,2)\}$$

$$df(1) - \{(1,2),(1,6),(1,3)\}$$

Rezultatul pornind de la 1: 1, 2, 6



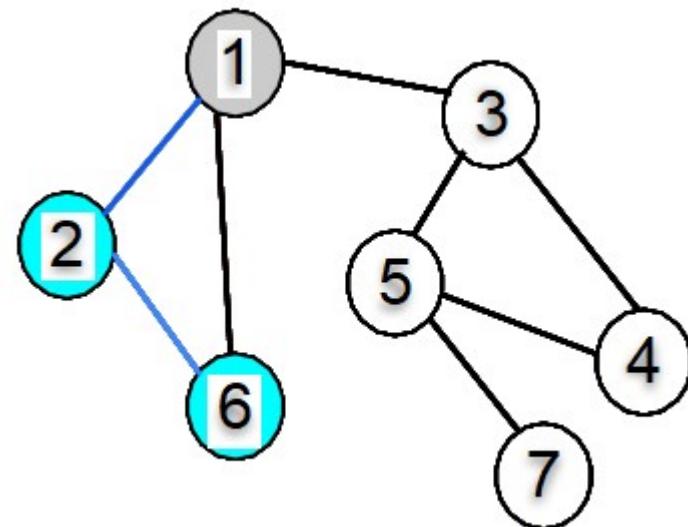
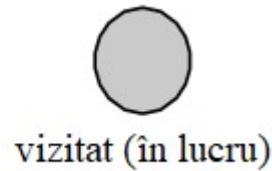
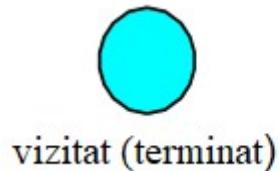
Exemplu – pasul 6

Stivă de urmărire

$$df(2) - \{(2,6),(1,2)\}$$

$$df(1) - \{(1,2),(1,6),(1,3)\}$$

Rezultatul pornind de la 1: 1, 2, 6



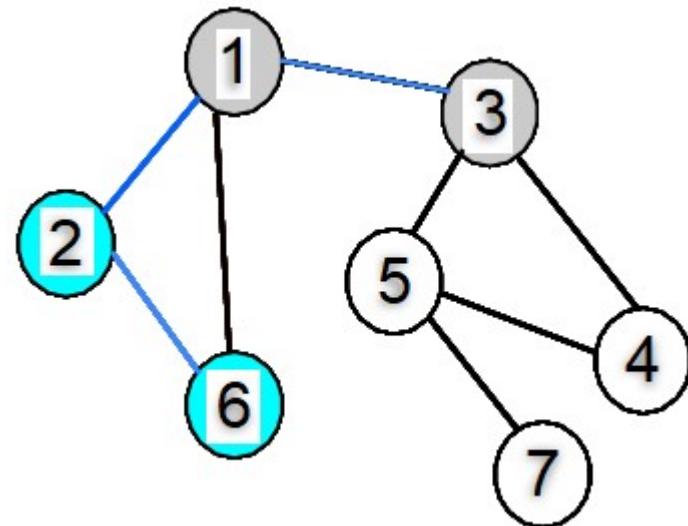
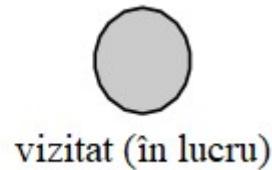
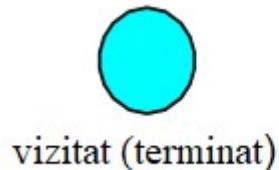
Exemplu – pasul 7

Stivă de urmărire

$$df(3) - \{(3,5),(3,4),(1,3)\}$$

$$df(1) - \{(1,2),(1,6),(1,3)\}$$

Rezultatul pornind de la 1: 1, 2, 6, 3



Exemplu – pasul 8

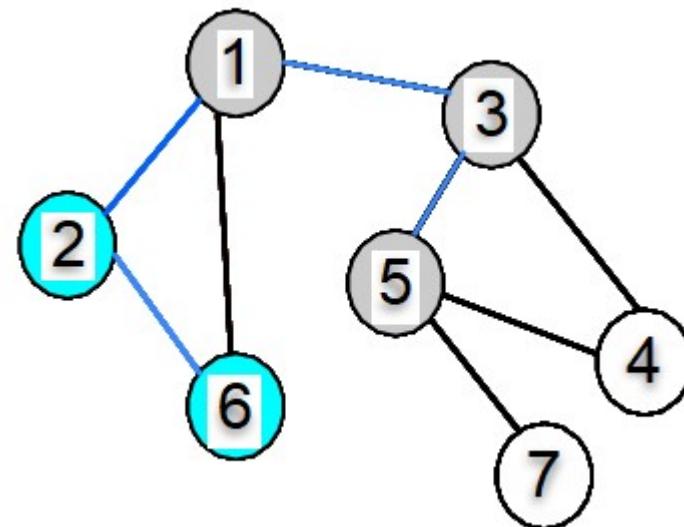
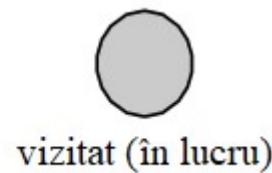
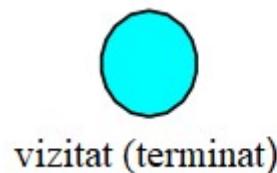
Stivă de urmărire

$df(5) - \{(5,7),(5,4),(5,3)\}$

$df(3) - \{(3,5),(3,4),(1,3)\}$

$df(1) - \{(1,2),(1,6),(1,3)\}$

Rezultatul pornind de la 1: 1, 2, 6, 3, 5



Exemplu – pasul 9

Stivă de urmărire

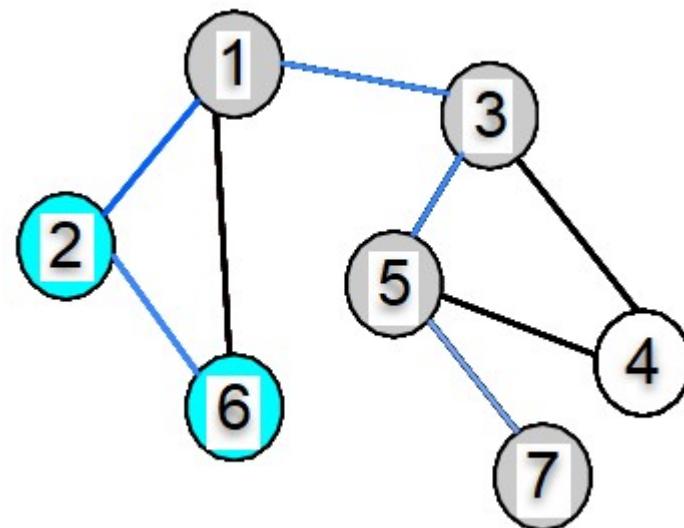
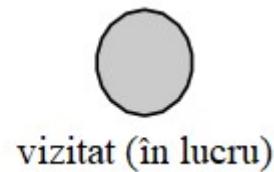
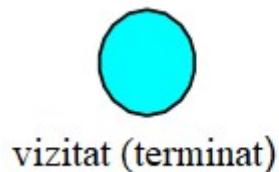
$$df(7) - \{(5,7)\}$$

$$df(5) - \{(5,7), (5,4), (5,3)\}$$

$$df(3) - \{(3,5), (3,4), (1,3)\}$$

$$df(1) - \{(1,2), (1,6), (1,3)\}$$

Rezultatul pornind de la 1: 1, 2, 6, 3, 5, 7



Exemplu – pasul 10

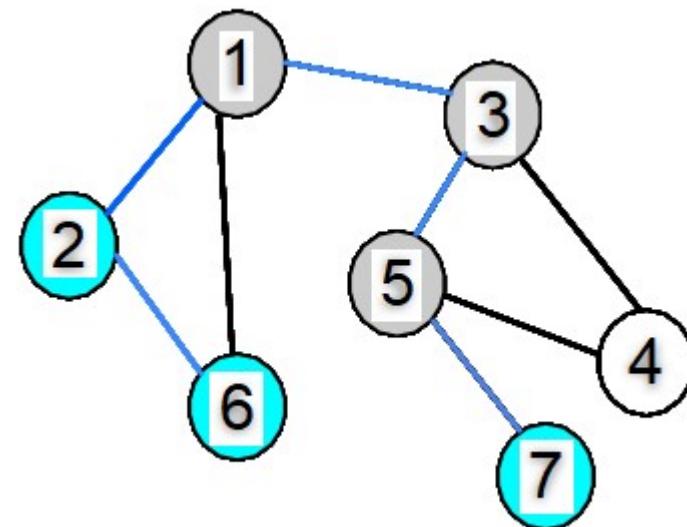
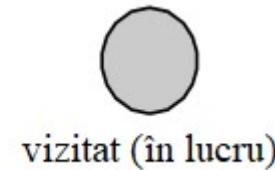
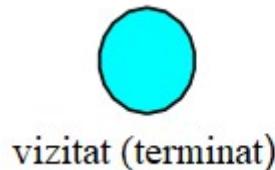
Stivă de urmărire

$df(5) - \{(5,7),(5,4),(5,3)\}$

$df(3) - \{(3,5),(3,4),(1,3)\}$

$df(1) - \{(1,2),(1,6),(1,3)\}$

Rezultatul pornind de la 1: 1, 2, 6, 3, 5, 7



Exemplu – pasul 11

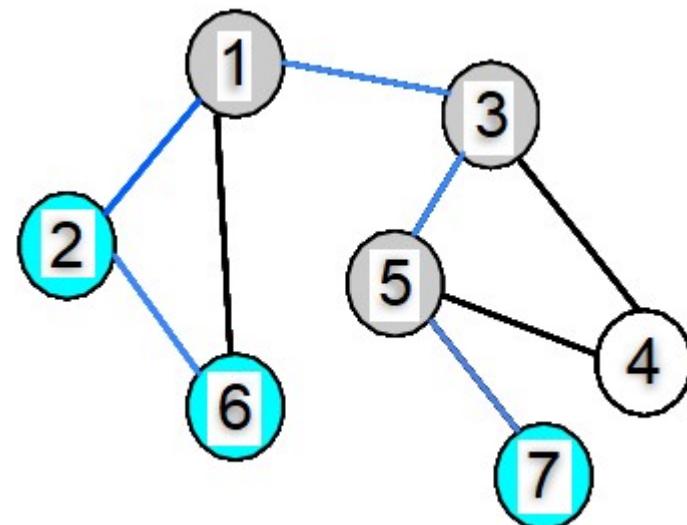
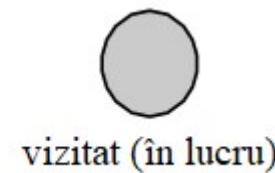
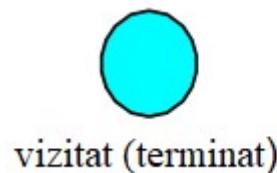
Stivă de urmărire

$$df(5) - \{(5,7),(5,4),(5,3)\}$$

$$df(3) - \{(3,5),(3,4),(1,3)\}$$

$$df(1) - \{(1,2),(1,6),(1,3)\}$$

Rezultatul pornind de la 1: 1, 2, 6, 3, 5, 7



Exemplu – pasul 12

Stivă de urmărire

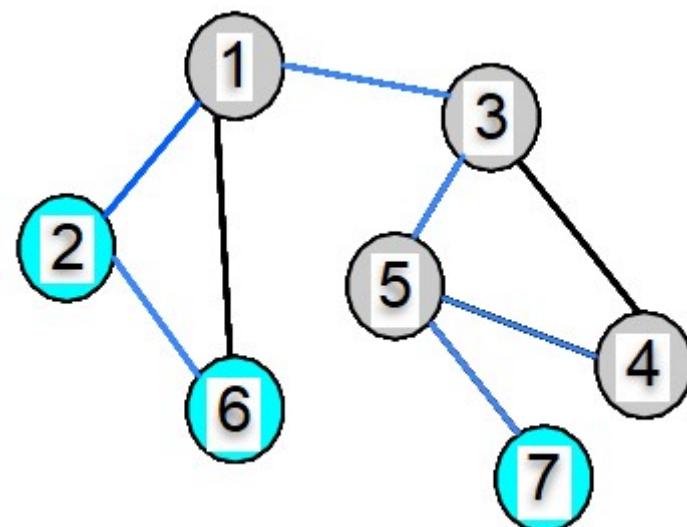
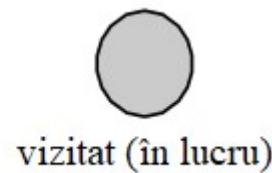
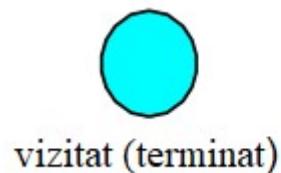
$$df(4) - \{(5,4),(3,4)\}$$

$$df(5) - \{(5,7),(5,4),(5,3)\}$$

$$df(3) - \{(3,5),(3,4),(1,3)\}$$

$$df(1) - \{(1,2),(1,6),(1,3)\}$$

Rezultatul pornind de la 1: 1, 2, 6, 3, 5, 7, 4



Exemplu – pasul 13

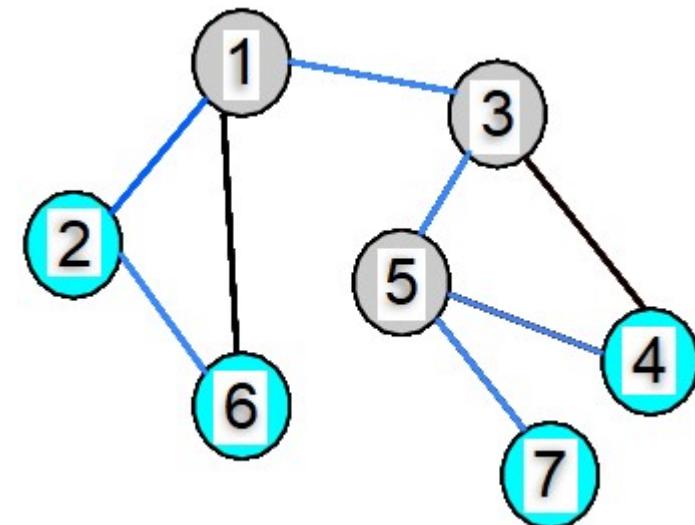
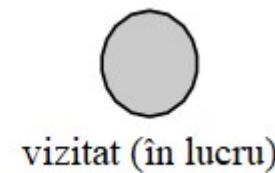
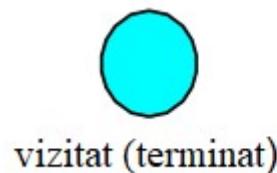
Stivă de urmărire

$df(5) - \{(5,7),(5,4),(5,3)\}$

$df(3) - \{(3,5),(3,4),(1,3)\}$

$df(1) - \{(1,2),(1,6),(1,3)\}$

Rezultatul pornind de la 1: 1, 2, 6, 3, 5, 7, 4



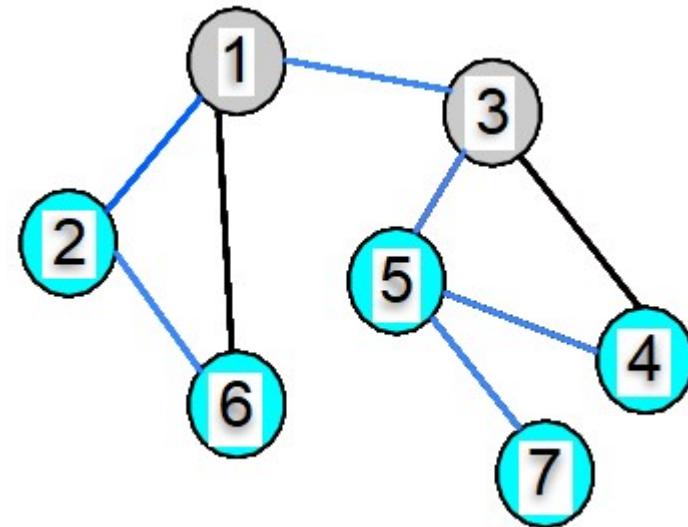
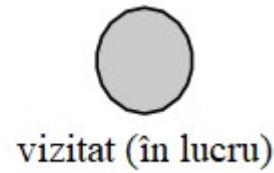
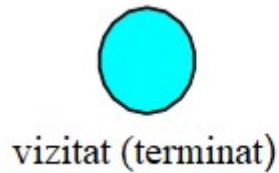
Exemplu – pasul 14

Stivă de urmărire

$$df(3) - \{(3,5),(3,4),(1,3)\}$$

$$df(1) - \{(1,2),(1,6),(1,3)\}$$

Rezultatul pornind de la 1: 1, 2, 6, 3, 5, 7, 4

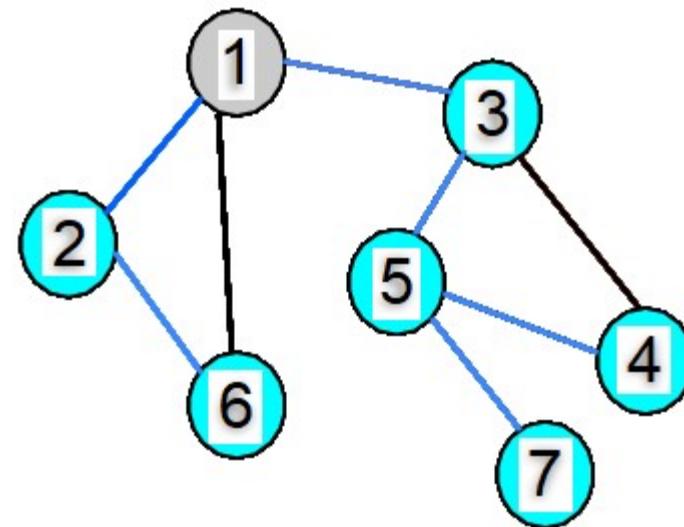
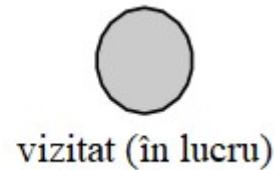
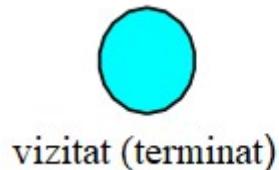


Exemplu – pasul 15

Stivă de urmărire

$$df(1) - \{(1,2), (1,6), (1,3)\}$$

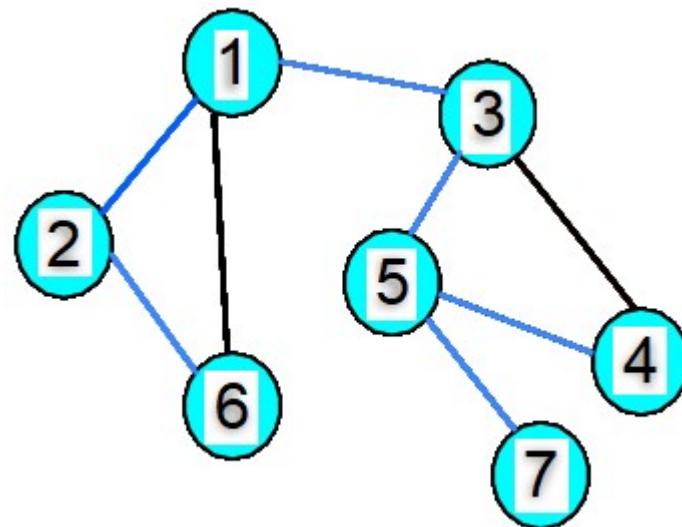
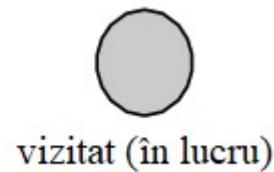
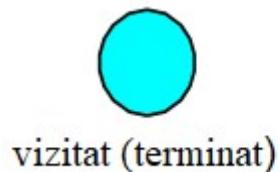
Rezultatul pornind de la 1: 1, 2, 6, 3, 5, 7, 4

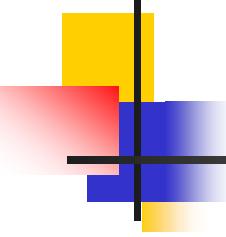


Exemplu – pasul 16

Stivă de urmărire

Rezultatul pornind de la 1: 1, 2, 6, 3, 5, 7, 4





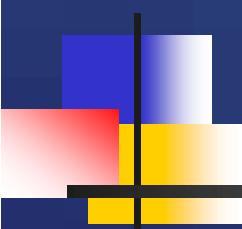
Rezultate

Pentru cele două moduri de traversare:

- Breadth-First (în lățime)
- Depth-First (în adâncime)

rezultatele prezentate nu sunt unice.

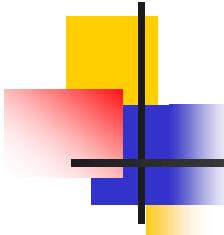
Rezultatele pot dифeri în func^{ie} de o ordine cerută, de exemplu de la stânga la dreapta.



Logica si structuri discrete

**Universitatea Politehnica Timisoara
Anul universitar 2018-2019**

S1.dr.ing.Mirella Amelia MIOC



Curs 11

Arbore Binari de Căutare

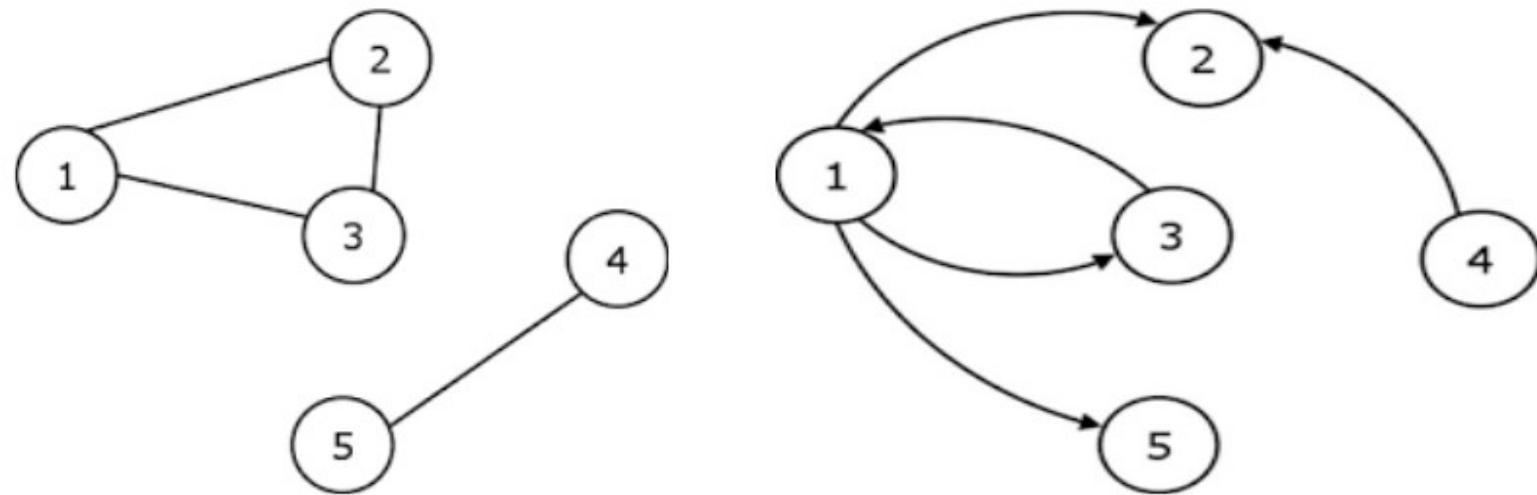
Noțiuni Generale

- Arboare; Arboare binare
- Traversare
- Inserare
- Căutare
- Stergere

Elemente teoretice despre grafuri

Grafuri: Un graf $G = (V, E)$ reprezintă o mulțime de noduri/vârfuri (V) unite între ele prin muchii/arce (E)

Grafuri orientate și neorientate

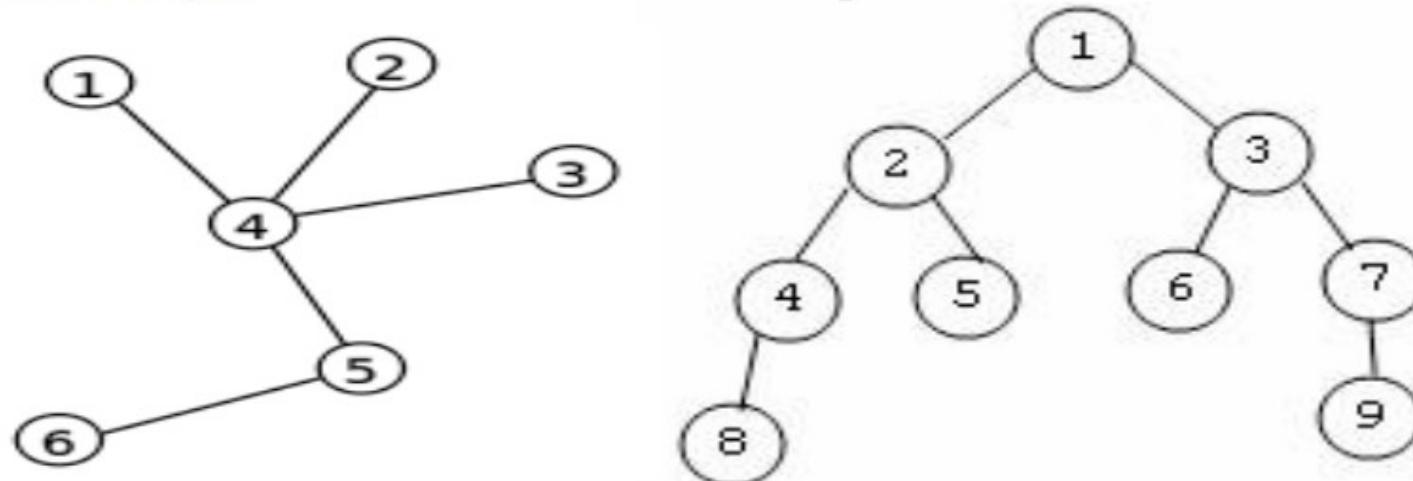


Structuri arborescente

Definiție : Un arbore este un graf neorientat conex și fără cicluri în care unul dintre noduri este rădăcină.

- Conexitate : există drum între oricare două noduri.
- Ciclul este un drum de forma $(v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_n \rightarrow v_0)$

Observație : Multe drumuri trec prin rădăcină.

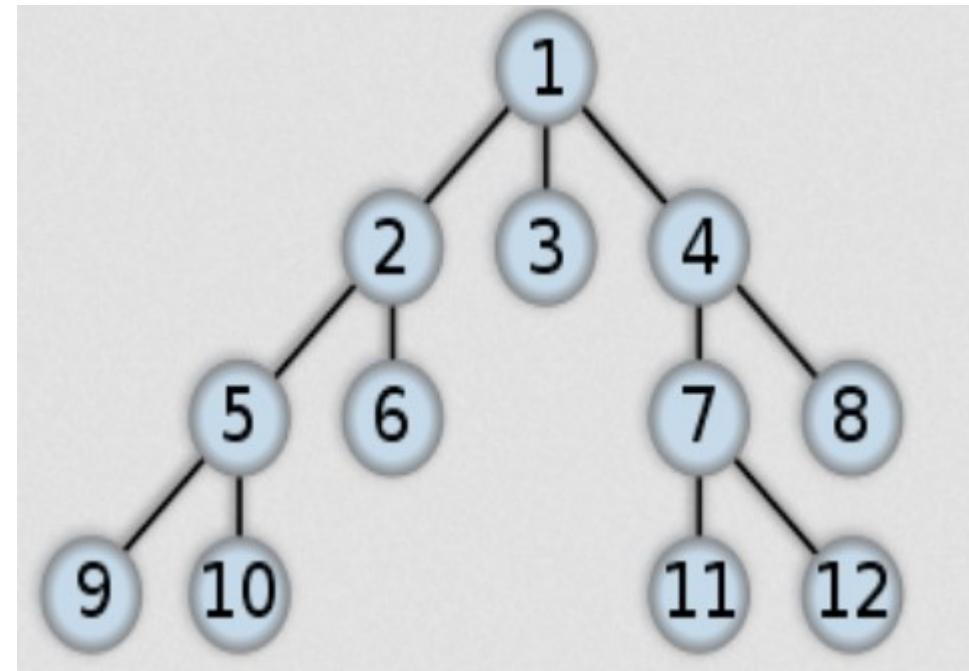


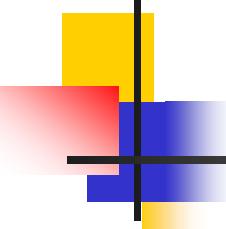
Elementele unui arbore

Nod unic numit **rădăcină**. Un nod are un unic **nod părinte** (legătura superioară) și o mulțime de **noduri fii**. Nodurile fără fii sunt numite **noduri frunze** (terminale).

```
struct treenode{  
    treenode * parent;  
    listnode * children;  
    int val;  
}treenode;
```

```
struct listnode {  
    treenode * node;  
    listnode * next;  
} listnode ;
```

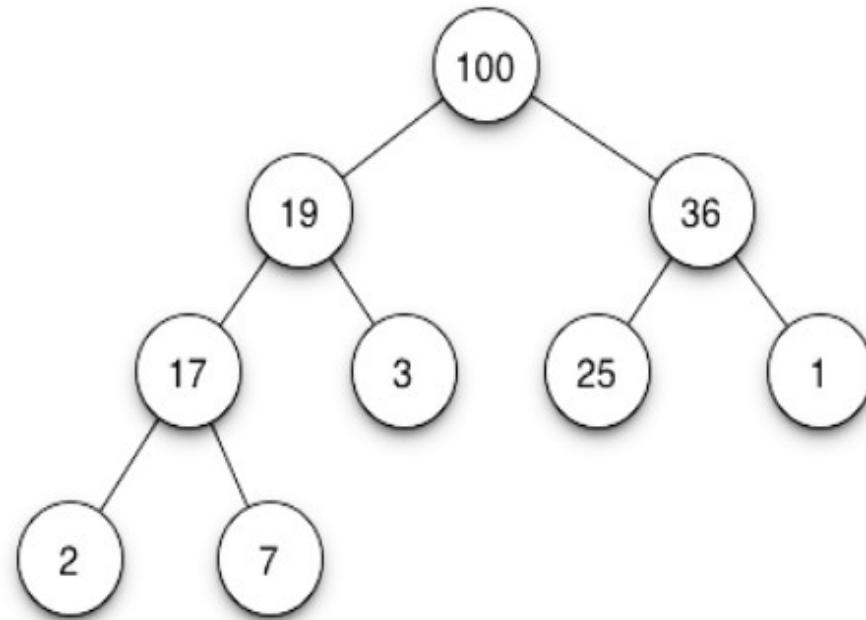


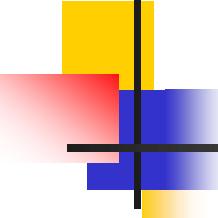


Arbore binari

Fiecare nod are maxim două noduri copii.

```
struct treenode {  
    treenode * parent; // optional  
    treenode * left;  
    treenode * right;  
    int val;  
}treenode;  
  
treenode* root;
```





Traversarea arborilor binari - I

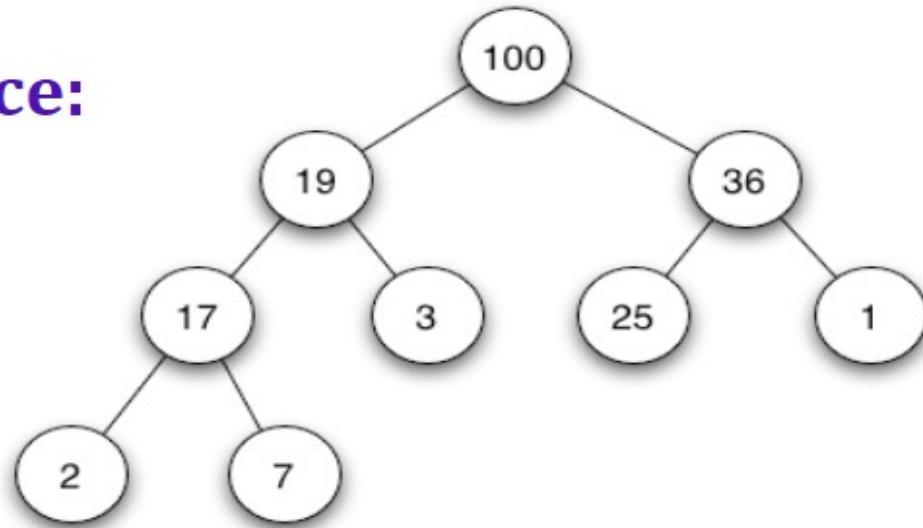
Parcurgeri generale din teoria grafurilor :

- adâncime (100, 19, 17, 2, 7, 3, 36, 25, 1).
- lățime (100, 19, 36, 17, 3, 25, 1, 2, 7).

Notiunea de **subarbore** (din stânga sau din dreapta).

Parcurgeri specifice:

- inordine (SRD)
- preordine (RSD)
- postordine (SDR)

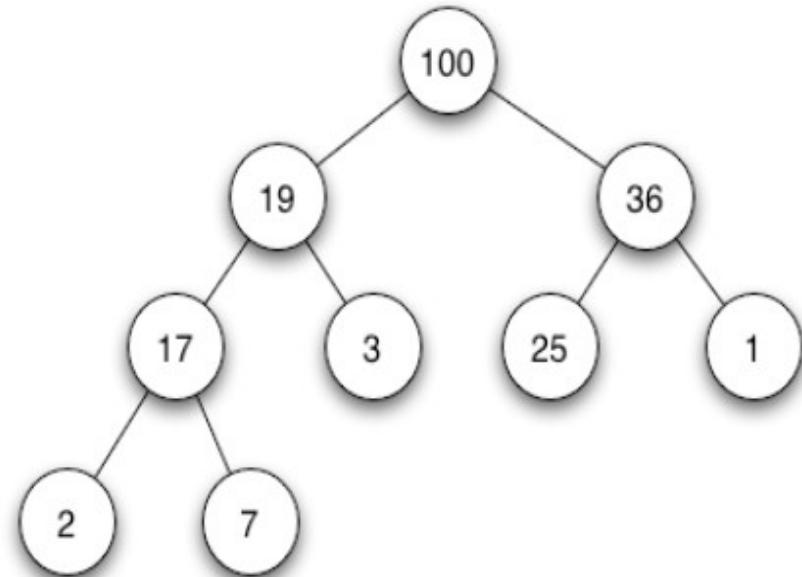


Traversarea în inordine (SRD)

```
void inordine(treenode * subroot) {  
    if(subroot==NULL) return;  
    inordine(subroot->left);  
    printf("Valoare nod %d\n",subroot->val);  
    // alte eventuale procesări  
    inordine(subroot->right);  
}
```

Elementele afişate :

2, 17, 7, 19, 3, 100, 25, 36, 1

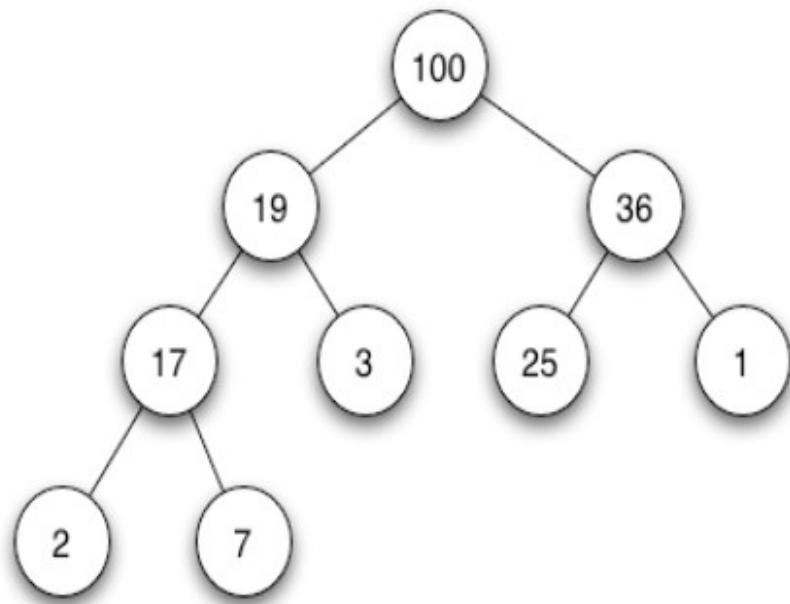


Traversarea în preordine (RSD)

```
void preordine(treenode * subroot) {  
    if(subroot==NULL) return;  
    printf("Valoare nod %d \n",subroot->val);  
    // alte eventuale procesări  
    preordine(subroot->left);  
    preordine(subroot->right);  
}
```

Elementele afișate :

100, 19, 17, 2, 7, 3, 36, 25, 1

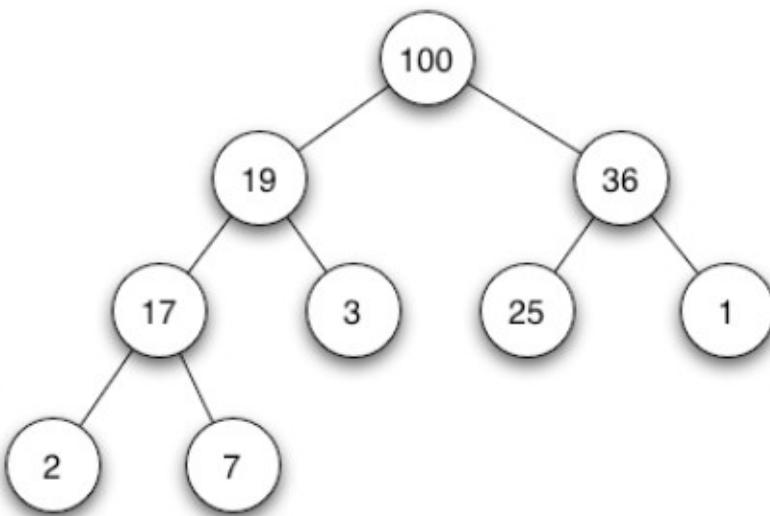


Traversarea în postordine (SDR)

```
void postordine(treenode * subroot) {  
    if(subroot==NULL) return;  
    postordine(subroot->left);  
    postordine(subroot->right);  
    printf("Valoare nod %d\n",subroot->val);  
    // alte eventuale procesări  
}
```

Elementele afişate :

2, 7, 17, 3, 19, 25, 1, 36, 100

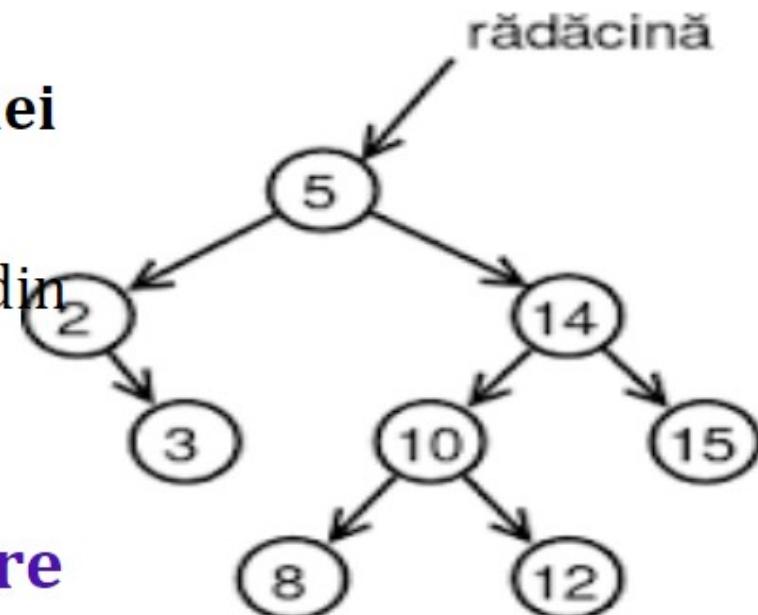


Arbore binare de căutare

Una dintre informațiile unui nod este **cheie** și suportă relația de ordine (numerică, lexicografică, etc.).

Proprietate : pentru orice nod, cheia nodului rădăcină

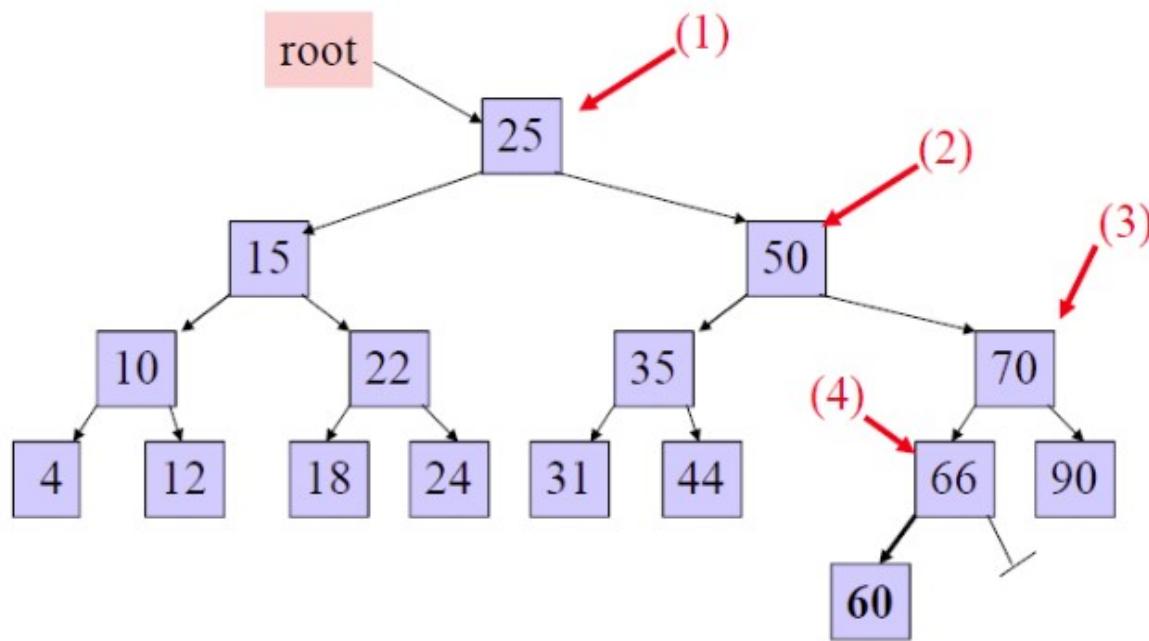
este **superioară oricărui chei** din subarborele din **stânga** și **inferioară oricărui chei** din subarborele din **dreapta**.

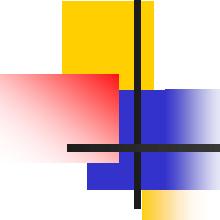


Inordine (SRD) = Sortare

Inserarea unui nou element – I

Inserarea unui nou element (ex. 60) se face comparând valoarea sa cu valoarea nodului rădăcina și apoi coborând recursiv în arbore conform relației de ordine.



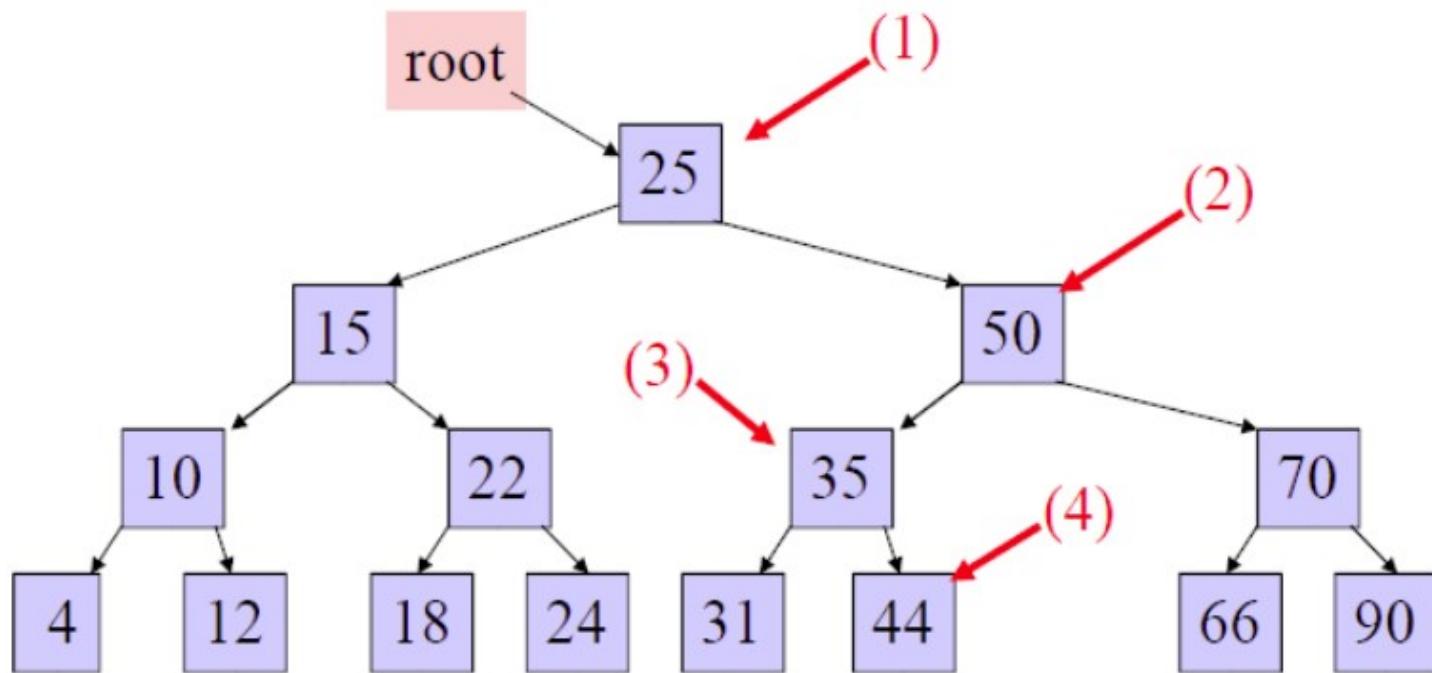


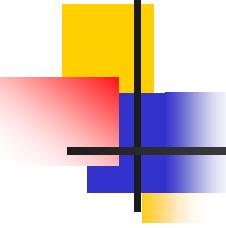
Inserarea unui nou element - II

```
void insert(int val, treenode* & root) {
    treenode* node=(treenode*)malloc(sizeof(treenode));
    node->val=val; node->left=node->right=NULL;
    if(root==NULL) { root=node; return; }
    treenode* p=root;
    while(p!=NULL){
        if(val<p->val) {
            if(p->left==NULL) p->left=node; return;
            else p=p->left; }
        else {
            if(p->right==NULL) p->right=node; return;
            else p=p->right;
        }
    }
}
```

Căutarea/modificarea unui element

Căutarea unui element (cheia 44/45) se face comparând valoarea sa cu valoarea nodului rădăcină și apoi coborând recursiv în arbore conform relației de ordine ...





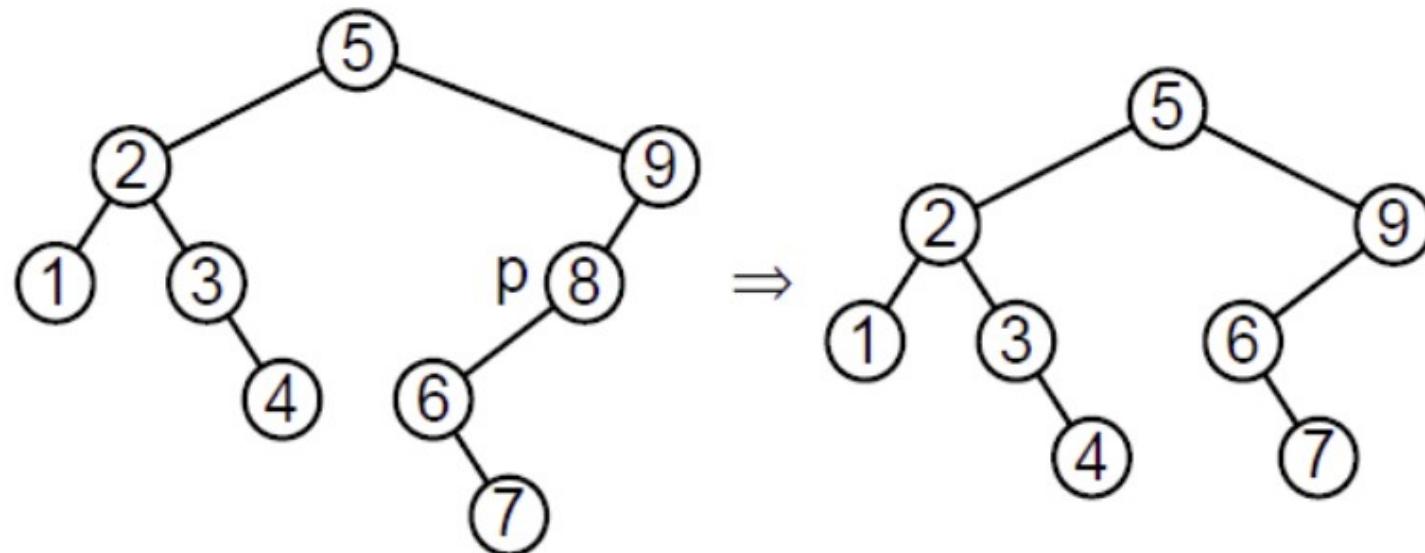
Căutarea/modificarea unui element

```
treenode* search(int val, treenode * root) {  
    treenode* p=root;  
    while(p!=NULL){  
        if(val==p->val) // găsirea elementului  
            return p;  
        if(val<p->val) // coborârea în partea stângă p=p->left;  
            else // coborârea în partea dreaptă  
                p=p->right;  
    }  
    return NULL; // elementul nu a fost găsit  
}
```

Ştergerea unui element – cazul I

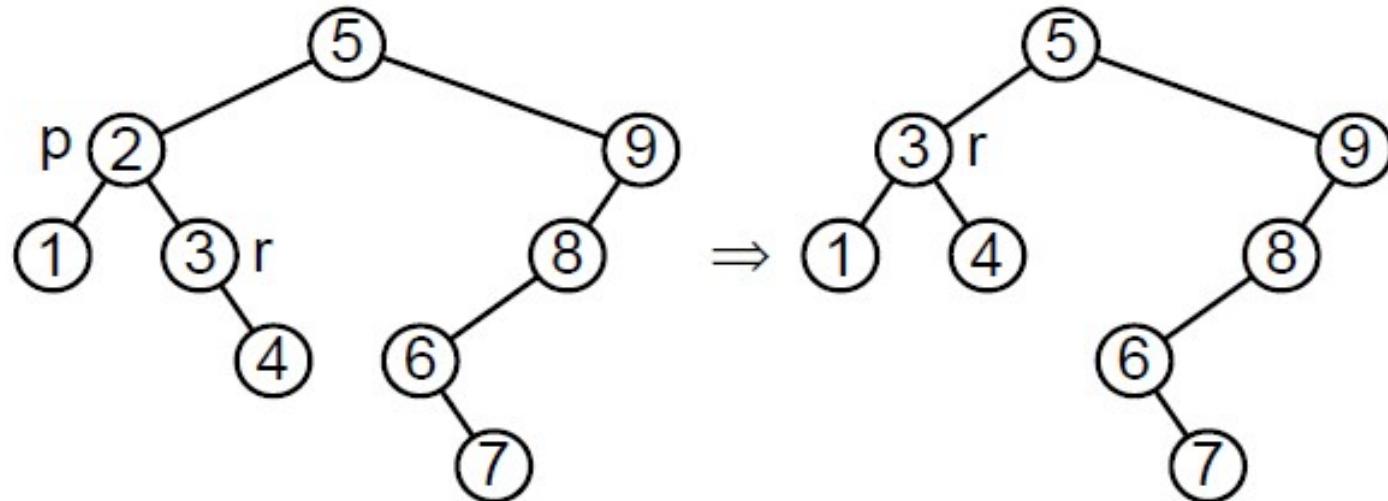
Operări mai complicate deoarece trebuie refăcute legăturile pentru a menține proprietatea arborelui.

Cazul I : nodul căutat nu are copil în dreapta.



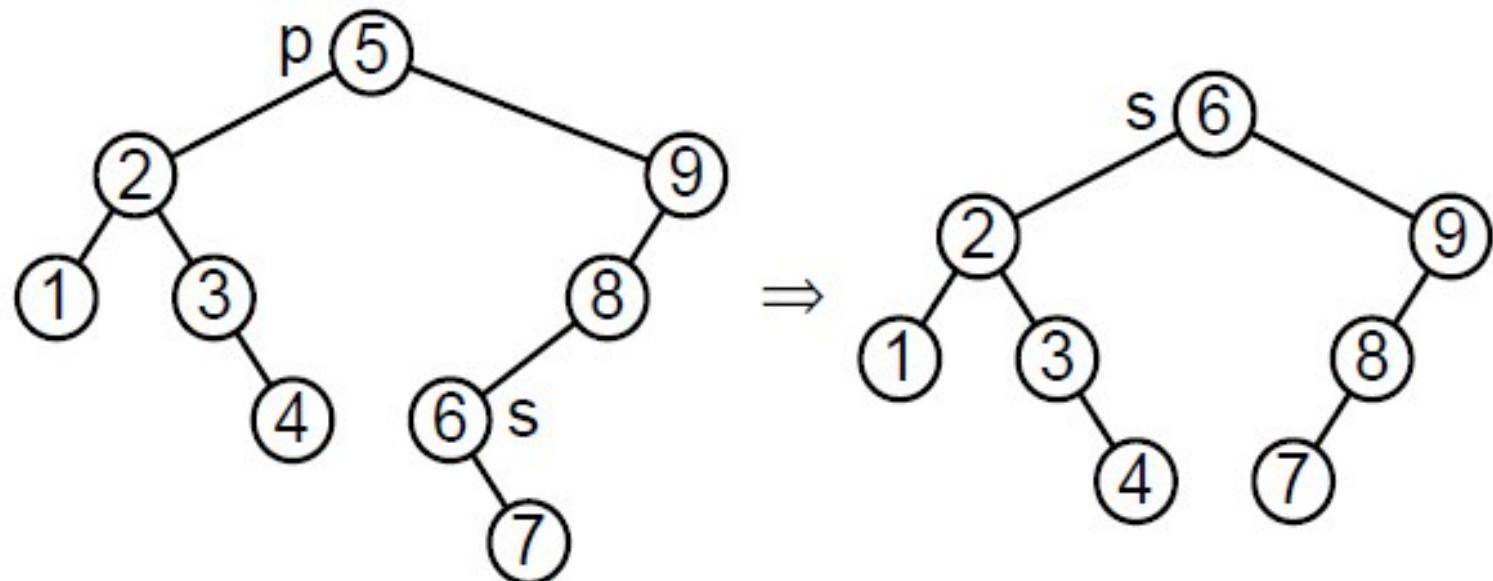
Ștergerea unui element – cazul II

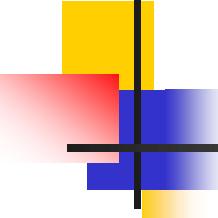
Cazul II : nodul căutat (p) are copil în dreapta (r).
(r) nu are nod copil în stânga.



Ştergerea unui element – cazul III

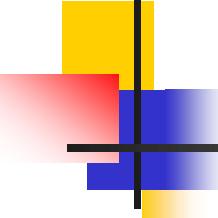
**Cazul III : nodul căutat (p) are copil în dreapta, iar acesta are un nod copil în stânga.
s reprezintă nodul cel mai din stânga al subarborelui drept.**





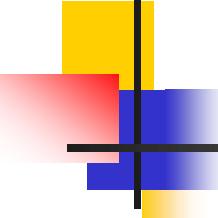
Implementarea ștergerii unui nod

```
void remove(int val, treenode * & root) {
    treenode* p=search(val,root); if(p==NULL) return;
    treenode* parent=p->parent;
    if(p->right==NULL) { //cazul I
        if(parent==NULL) {// radacina este eliminata
            root=p->left;
            if(root!=NULL) root->parent=NULL;
        }
        else{
            if(parent->left==p) parent->left=p->left;
            else parent->right=p->left;
            if(p->left!=NULL) p->left->parent=parent;
        }
        free(p); return;
    }
}
```



Implementarea stergerii unui nod

```
treenode* r=p->right;
if(r->left==NULL){ // cazul II
    if(parent==NULL) { // radacina este eliminata
        root=r; root->parent=NULL;
        r->left=p->left;
        if(p->left!=NULL) p->left->parent=r;
    }
    else{
        if(parent->left==p) parent->left=r;
        else parent->right=r;
        r->parent=parent;
        if(p->left!=NULL) p->left->parent=r;
    }
    free(p); return;
}
```



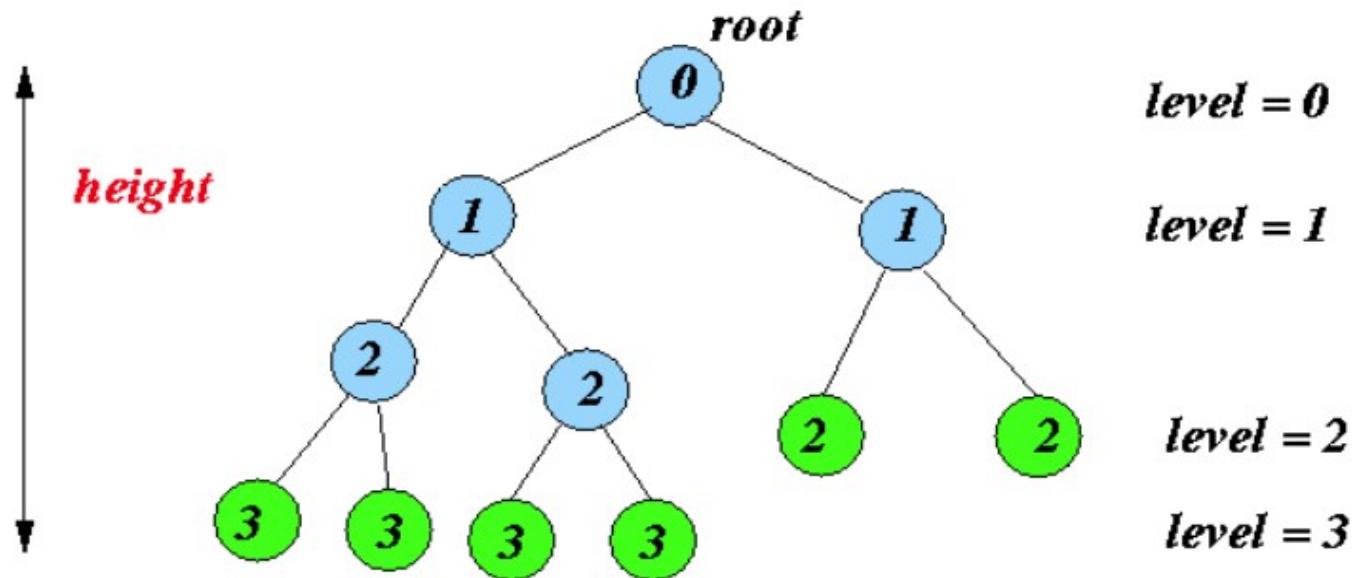
Implementarea ștergerii unui nod

```
// cazul III – cautam elementul s cel mai din stanga din
// subarborele din stanga nodului r (partea din dreapta a lui p)
treenode* s=r->left;
while(s->left!=NULL)
    s=s->left;
// inlocuirea valorii lui p cu cea din nodul s
p->val=s->val;
parent=s->parent;
parent->left=s->right;
if(s->right!=NULL)
    s->right=parent;
free(s);
return;
}
```

Înălțimea unui arbore

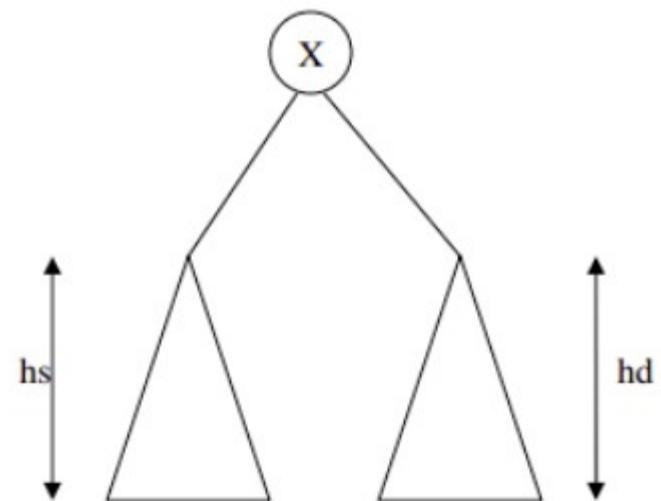
Înălțimea unui arbore : drumul (numărul de muchii) între rădăcină și cea mai îndepărtată frunză.

Nivelul unui nod : distanța față de rădăcină.



Înălțimea unui arbore binar

```
int height(treenode * subroot) {  
    if(subroot==NULL) return 0;  
    if(subroot->left!=NULL || subroot->right!=NULL){  
        int hs=height(subroot->left);  
        int hd=height(subroot->right);  
  
        if(hs<hd) return hd+1;  
        else return hs+1;  
    }  
    return 0;  
}
```

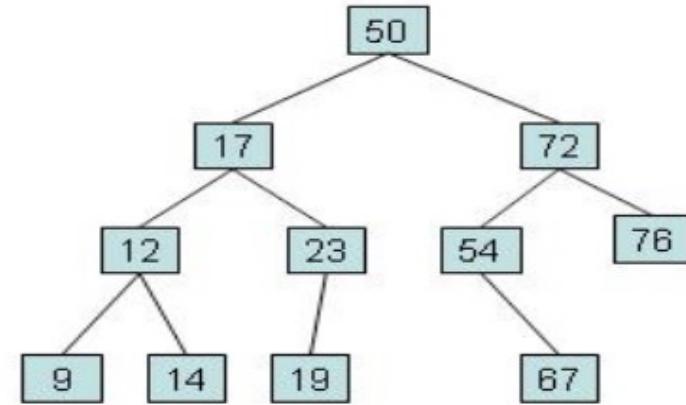
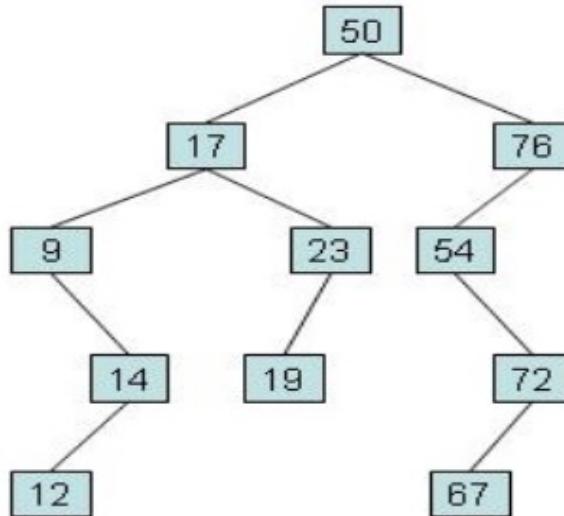


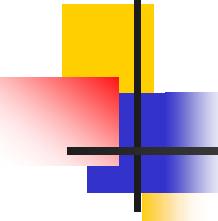
Analiza complexității operațiilor

$\Omega(1) \leq \text{adăugare, căutare și ștergere} \leq O(h)$

Înalțimea h depinde de ordinea de inserare sau ștergere.

Echilibrarea arborelui : disponerea compactă.



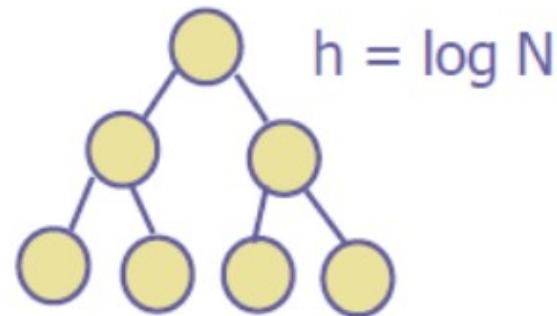
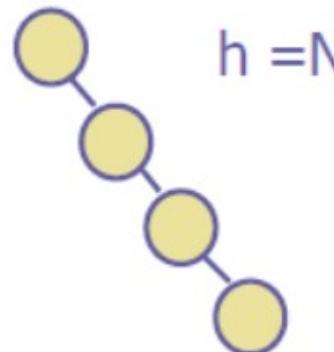


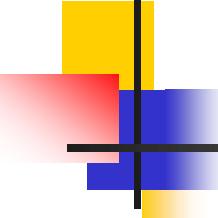
Arbore degenerati și echilibrați

Arbore degenerati (elemente în ordine crescătoare)

Arbore perfect echilibrați de înălțime h : ex. toate nodurile frunză sunt pe același nivel și orice nod de pe nivelurile intermediare are numărul maxim de fii.

Înălțimea minimă pentru un arbore cu N noduri, unde $2^{n-1} \leq N < 2^n$ este $h = n - 1 = [\log_2 N]$

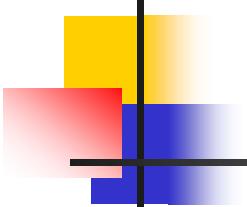




Exerciții propuse

- 1) Dat fiind un arbore binar ale cărui noduri conțin valori întregi, să se verifice dacă este unul de căutare.
- 2) Sa se găsească elementul minim/maxim dintr-un arbore binar de căutare.
- 3) Să se reconstruiască un arbore binar pe baza parcurgerilor sale inordine (SRD) și postordine (SDR).
- 4) Să se determine numărul de arbori binari de căutare distincți care conțin numerele de la 1 la N.

Curs10. Structuri de date dinamice



1. Structuri de date statice și dinamice
2. Tipul pointer
3. Utilizarea pointerilor
4. Pointeri structuri și tablouri
5. Alocarea dinamică a memoriei
6. Liste liniare simplu înlățuite
7. Liste ordonate
8. Liste dublu înlățuite
9. Liste multiplu înlățuite

Structuri de date statice și dinamice

Principii

- Structurile de date fundamentale ale limbajului C (tabloul, structura și uniunea) sunt fixe din punct de vedere al formei și al dimensiunii. Necesarul de memorie pentru reprezentarea unei variabile de un astfel de tip, numită variabilă structurată, rezultă din tipul și numărul componentelor sale, se calculează la faza de compilare a programului și rămâne constant pe parcursul execuției. Aceste structuri de date se numesc **statische**.
- În practica programării însă, există multe probleme care presupun structuri ale informației mult mai complicate (de ex.: liste, arbori, grafuri), a căror caracteristică principală este aceea că se modifică structural în timpul execuției programului. Rezultând o modificare dinamică atât a formei cât și a dimensiunii lor, aceste structuri de date se numesc **dinamice**.

Structuri de date statice și dinamice

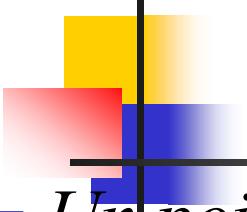
Caracteristici

- În general, atât structurile de date dinamice cât și cele statice sunt formate, în ultimă instanță, din componente de dimensiune constantă care se încadrează într-unul din tipurile acceptate de limbaj.
- În cazul structurilor dinamice aceste componente se numesc **noduri**. Natura dinamică a structurii rezultă din faptul că numărul nodurilor și legăturile între noduri se pot modifica pe durata rulării.

Structuri de date statice și dinamice

Caracteristici

- Atât structurile dinamice în ansamblu cât și nodurile lor individuale se deosebesc de structurile statice prin faptul că ele nu se declară ca variabile și, în consecință, nu se poate face referire la ele utilizând numele lor deoarece, practic, nu au nume.
- Referirea unor astfel de structuri se face cu ajutorul unor variabile statice, definite special în acest scop, numite **pointeri** sau **variabile-indicator**. Un pointer poate conține la un moment dat o **referință** la un nod al unei structuri dinamice materializată prin **adresa** nodului respectiv.



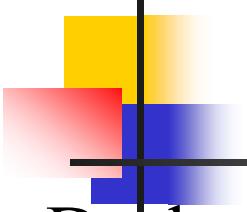
Pointeri

Tipul pointer

- *Un pointer este o variabilă care are ca valoare o adresă de memorie.*

Spunem că variabila pointer indică locația de memorie de la adresa pe care o conține.

- Cel mai comun caz este acela în care valoarea unei variabile pointer este adresa unei alte variabile.
- În C, pointerii se referă la un anumit tip: tipul datelor conținute în locația de memorie indicată de variabila pointer.
=> O variabilă pointer în C este de regulă *pointer la un anumit tip*.



Pointeri

Tipul pointer

- Declararea variabilelor pointer se face în felul următor:

*tip * nume_variabila_pointer;*

variabila *nume_variabila_pointer* conține adrese de zone de memorie alocate datelor de tipul *tip*.

* semnifică faptul că variabila este de tip pointer la tipul respectiv.

- De exemplu,

```
int *p;
```

definește o variabilă *p* pointer la întreg. Variabila *p* poate conține adrese de locații în care se memorează valori întregi.



Pointeri

Tipul pointer

- Declarațiile variabilelor pointer la un anumit tip pot să apară în aceeași listă cu alte variabile obișnuite de acel tip:

```
int x, *p;
```

definește o variabilă *x* de tip întreg și o variabilă *p* de tip pointer la întreg.

- Se pot declara și *pointeri generici*, de tip *void ** (tipul datelor indicate de ei nu este cunoscut). Un pointer de tip void reprezintă doar o adresă de memorie a unui obiect oarecare:

- dimensiunea zonei de memorie indicate și interpretarea informației nu sunt definite;
- poate apărea în atribuirile, în ambele sensuri, cu pointeri de orice alt tip;
- folosind o conversie explicită de tip cu operatorul *cast* : (*tip **), el poate fi convertit la orice alt tip de pointer(pe același calculator, toate adresele au aceeași lungime).

Pointeri

Utilizare – operatorii de adresare și dereferențiere

- Ca și în cazul variabilelor de orice tip, și variabilele pointer declarate și neinitializate conțin valori aleatoare.
- Pentru a atribui variabilei p valoarea adresei variabilei x , se folosește operatorul de adresă într-o expresie de atribuire de forma:

$p = \&x;$

În urma acestei operații, se poate spune că pointerul p indică spre variabila x .

- Pentru a obține valoarea obiectului indicat de un pointer se utilizează *operatorul de derefențiere (indirectare)* (operatorul $*$):

Dacă p este o variabilă de tip pointer care are ca valoare adresa locației de memorie a variabilei întregi x (p indică spre x) atunci expresia $*p$ reprezintă o valoare întreagă (valoarea variabilei x).

Pointeri

Utilizare– operatorii de adresare și dereferențiere

```
int x, *p;  
x=3;  
p=&x;  
printf("%d", *p);  
*p=5;  
printf("%d", x);
```

Expresia $*p$ care se afișează cu primul *printf* reprezintă valoarea obiectului indicat de p , deci valoarea lui x . Se va afișa valoarea 3.

Atribuirea $*p=5;$ modifică valoarea obiectului indicat de p , adică valoarea lui x . Ultimul *printf* din secvență afișează 5 ca valoare a lui x .

Pointeri

Utilizare– operatorii de adresare și dereferențiere

- Întotdeauna când se aplică operatorul `*` asupra unui pointer `ptr` declarat ca și

```
tip *ptr;
```

expresia obținută prin derefențiere (`*ptr`) este de tipul `tip`.

- În cazul unui pointer generic (de tip `void *`), derefențierea trebuie precedată de *cast* (altfel nu știm ce tip de valori indică pointerul).
- Când se utilizează operatorul de derefențiere trebuie avut grijă ca variabila pointer asupra căreia se aplică să fi fost *initializată*, adică să conțină adresa unei locații de memorie valide. O secvență ca cea de mai jos este incorectă și poate genera erori grave la execuție:

```
int *p;  
*p=3;
```

Pentru a indica adresă inexistentă, se utilizează ca valoare a unui pointer, constanta `NULL`.

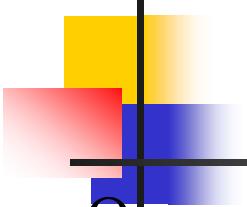
Pointeri

Utilizare– operatorii de adresare și dereferențiere

- Variabilele pointer pot fi utilizate în expresii și direct (fără operația de indirectare):

```
int x, *p1, *p2;  
x=3;  
p1=&x;  
p2=p1; /* atribuire de pointeri */
```

- Atribuirea $p2=p1$; copiază conținutul lui $p1$ în $p2$, deci $p2$ va fi făcut să indice spre același obiect ca și $p1$. În contextul secvenței, $p2$ va indica tot spre x .



Pointeri

Pointeri și structuri

- Operatorul de derefențiere poate avea și o altă reprezentare în cazul pointerelor la structuri.

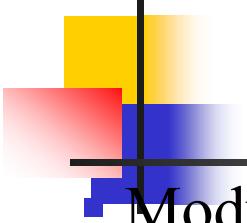
```
struct ts {  
    int x;  
  
    ...  
}
```

și variabilele

```
struct ts structura, *pstructura;  
...  
pstructura = &structura;
```

Pentru a selecta câmpul *x* al structurii indicate de *pstructura*, se poate scrie:

$(*pstructura).x$ sau, mai concis:
 $pstructura -> x$



Pointeri

Pointeri ca argumente de funcții

Modul implicit de transmitere a parametrilor funcțiilor în C este prin valoare:

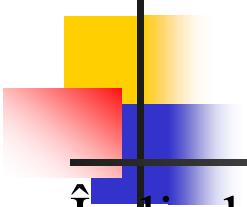
```
void schimba1(int x, int y) {  
    int aux ;  
    aux = x; x = y; y = aux ; }
```

schimba1(a,b) / schimbă doar copiile lui a și b */*

- Pointerii permit accesul direct la locația de memorie a unui argument.

```
void schimba2(int *x, int *y) {  
    int aux ;  
    aux = *x; *x = *y; *y = aux ; }
```

schimba2(&a, &b) / schimbă valorile variabilelor a și b */*



Pointeri

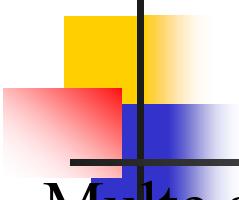
Pointeri și tablouri

- În limbajul C, numele unui tablou este echivalent cu un pointer care conține adresa primului său element.
- Fie următoarele declarații: *tab* un tablou de elemente de tip *T* și *p* un pointer la tipul *T*:
 - T tab[N];*
 - T * p;*

Cele 3 atribuiriri următoare sunt echivalente, și au ca efect faptul că *p* va indica adresa primului element al tabloului *tab*:

p=tab; p=&tab; p=&tab[0];

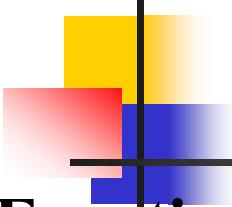
- Atribuirea *tab=p;* nu este permisă.
- Numele tabloului indică o adresă constantă: adresa primului element al tabloului este o valoare constantă din momentul în care tabloul a fost alocat static, deci nu mai poate fi modificată printr-o atribuire ca cea de mai sus.



Alocarea dinamică a memoriei

Principii

- Multe aplicații pot fi optimizate dacă memoria necesară stocării datelor lor este alocată *dinamic* în timpul execuției programului.
- Alocarea dinamică de memorie înseamnă alocarea de zone de memorie și eliberarea lor în timpul execuției programelor, în momente stabilite de programator.
- Zona de memorie în care se alocă, la cerere, datele dinamice este diferită de zona de memorie în care se alocă datele statice (stiva de date) și se numește *heap*.
- Spațiul de memorie alocat dinamic este accesibil programatorului printr-un pointer care conțin adresa sa. Pointerul care indică un obiect din *heap* este de regulă stocat într-o variabilă alocată static.
- Funcțiile de gestionare a memoriei alocate dinamic au prototipurile în fișierele header *alloc.h* și *stdlib.h*.



Alocarea dinamică a memoriei

Funcții pentru gestiunea memoriei dinamice

Funcția de alocare dinamică a memoriei (*malloc*)

*void * malloc(size_t size);*

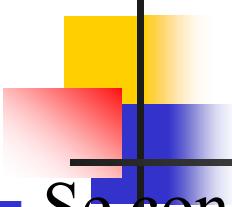
Funcția alocă în *heap* un bloc de dimensiune *size*; dacă operația reușește returnează un pointer la blocul alocat, altfel returnează NULL. Este posibil ca cererea de alocare să nu poată fi satisfăcută dacă nu mai există suficientă memorie în zona de *heap* sau nu există un bloc compact de dimensiune cel puțin egală cu *size*. Dimensiunea *size* se specifică în octeți.

- Rezultatul funcției *malloc* este un pointer de tip *void*. Se va folosi operatorul *cast* pentru conversii de tip la atribuire.
- Alocarea dinamică a unei zone pentru o valoare de tip *int* :

*int * ip;*

*ip=(int *)malloc(sizeof(int));*

**ip=5;*



Alocarea dinamică a memoriei

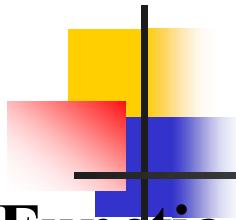
Funcții pentru gestiunea memoriei dinamice

- Se consideră tipul structurat persoana, definit în felul următor:

```
struct persoana {  
    char nume[30];  
    int varsta;  
};
```

- O variabilă dinamică de tip pointer la tipul structurat persoana se alocă și se utilizează în felul urmator:

```
struct persoana * pp;  
if ((pp=(persoana *) malloc (sizeof( persoana)))==NULL)) {  
    printf("\n Eroare alocare memorie \n");  
    exit(1);  
}  
pp->varsta=30;  
strcpy(pp->nume, "ion");
```



Alocarea dinamică a memoriei

Funcții pentru gestiunea memoriei dinamice

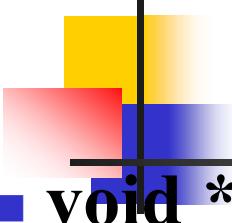
Funcția de eliberare dinamică a memoriei (free)

```
void free(void * block);
```

Funcția eliberează blocul alocat anterior cu *malloc*, având pointerul transmis ca parametru. Transferul unei valori invalide (un pointer la un bloc de memorie care nu a fost rezultatul unui apel al funcției *malloc*) poate compromite funcționarea sistemului de alocare.

- Chiar dacă variabila pointer care indică o astfel de zonă de memorie nu mai există (și-a depășit domeniul de viață) zona alocată rămâne încă ocupată, devenind însă înaccesibilă pentru program.
- Eroare frecventă de acest gen poate apare la utilizarea variabilelor dinamice în cadrul funcțiilor:

```
void fct() {  
    int * p;  
    p=(int *)malloc(10*sizeof(int)); }
```



Alocarea dinamică a memoriei

Funcții pentru gestiunea memoriei dinamice

- **void *calloc(size_t nitems, size_t size);**

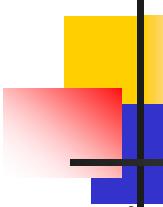
Funcția alocă (*în heap*) un bloc de dimensiune $size * nitems$ octeți, pe care îi pune pe 0; dacă operația reușește returnează un *pointer la blocul alocat*, altfel returnează *NULL*. Este posibil ca cererea de alocare să nu poată fi satisfăcută dacă nu există un bloc compact de dimensiune cel puțin egală cu $size * nitems$. Dimensiunea *size* se specifică în octeți.

Rezultatul funcției *calloc* fiind un pointer de tip *void*, se va folosi operatorul *cast* pentru conversii de tip la atribuire.

- Exemplu (alocarea dinamică a unui tablou de întregi):

```
int *tab;
```

```
tab=(int *)calloc( N, sizeof(int) );
```



Alocarea dinamică a memoriei

Funcții pentru gestiunea memoriei dinamice

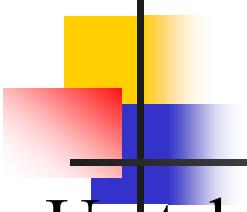
- **void *realloc(void *block, size_t size);**

Funcția ajustează mărimea blocului de la adresa *block* la dimensiunea *size*; dacă nu avem suficientă memorie la adresa *block*, va copia conținutul blocului la o adresă nouă; dacă operația reușește *returnează* un *pointer la blocul alocat*, dacă blocul nu poate fi reallocate sau *size=0* returnează *NULL*.

block – punctează către o zonă de memorie obținută anterior în mod dinamic

size – mărimea în octeți a noului bloc alocat

Dacă *block=NULL*, *realloc* se comportă ca un *malloc*.



Alocarea dinamică a memoriei

Alocarea dinamică a tablourilor

- Un tablou poate fi alocat dinamic printr-o secvență ca cea de mai jos:

`TIP * p;`

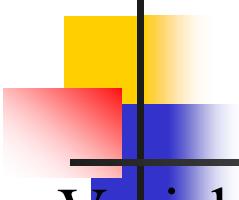
`p= (TIP *) malloc(N*sizeof(TIP));`

Pointerul p va indica un bloc suficient de mare pentru a conține N elemente de tipul TIP .

În continuare, variabila p poate fi utilizată ca și cum ar fi fost declarată ca un tablou de forma:

`TIP p[N];`

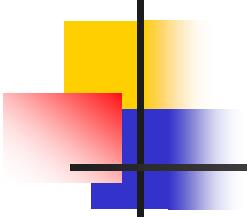
- Avantajul alocării dinamice a unui tablou este că dimensiunea sa poate fi specificată doar în timpul execuției.



Alocarea dinamică a memoriei

Concluzii

- Variabilele dinamice se numesc astfel pentru că ele apar și pot să dispară pe parcursul execuției programului, în mod dinamic. Alocarea spațiului necesar pentru o asemenea variabilă ca și relocarea (eliberarea) lui, se efectuează în mod explicit în program, prin apelurile *malloc* și *free*.
- Timpul scurs din momentul creării locației unei variabile și până la desființarea acestei locații se numește **durata de viață a variabilei**. Pentru variabilele obișnuite (automatice), durata de viață coincide cu durata de activare a blocului de care aparțin. Pentru variabilele dinamice apariția și disparația lor are loc independent de structura textului programului iar durata de viață este intervalul de timp scurs între apelurile funcțiilor *malloc* și *free* pentru acele variabile.



Liste liniare simplu înlăntuite

Definiții

Lista liniară este o secvență având un număr arbitrar de componente numite **noduri** de un anumit tip (același) numit **tip de bază**.

Nodurile unei liste pot fi ordonate liniar, funcție de poziția lor în cadrul listei :

$$a_1, a_2, \dots, a_n \qquad n \geq 0$$

Numărul n de noduri se numește **lungimea** listei.

Dacă $n = 0$ avem de-a face cu o **listă vidă**.

Liste liniare simplu înlăntuite

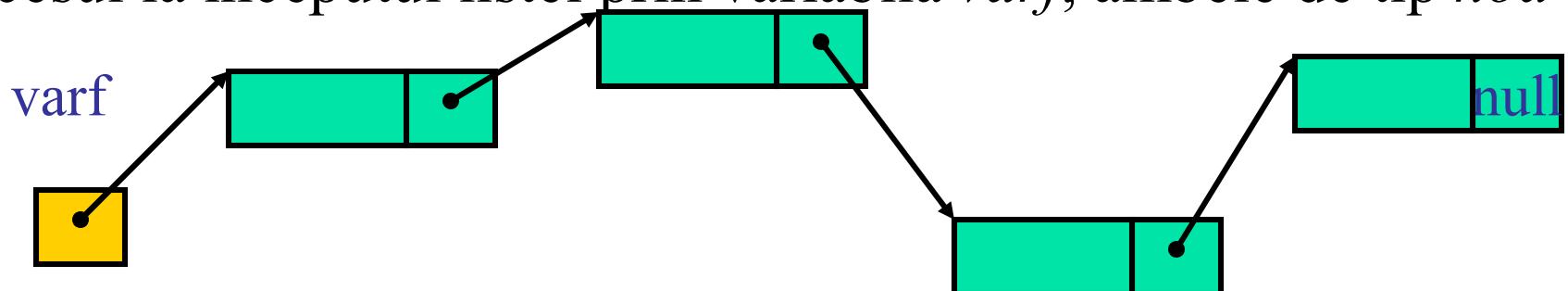
Reprezentare

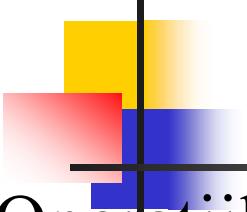
```
#include <stdlib.h>

typedef struct elem {
    int info;
    struct elem *urmator;
} nod;

nod *varf;
```

Legătura dintre noduri se realizează prin câmpul *urmator* iar accesul la începutul listei prin variabila *varf*, ambele de tip *nod **.



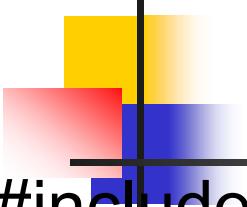


Liste liniare simplu înlăntuite

Operații de bază

Operațiile care se pot efectua asupra listelor sunt:

- crearea listei;
- parcurgerea listei;
- inserarea unui nod într-o anumită poziție (nu neapărat prima);
- căutarea unui nod cu o anumită proprietate;
- furnizarea conținutului unui nod dintr-o anumită poziție sau cu o anumită proprietate;
- suprimarea unui nod dintr-o anumită poziție sau cu o anumită proprietate.

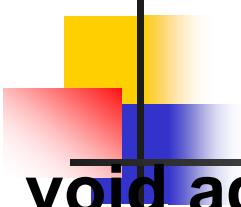


Liste liniare simplu înlăntuite

Operații de bază - exemple

```
#include <stdio.h>

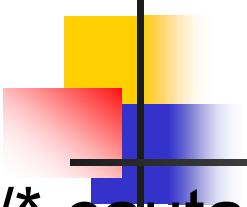
typedef struct elem {
    int info;
    struct elem *urmator;
} nod ;
nod *varf, *sfarsit;
// adaug un nod ( transmis ca argument ) in lista
void adaugare_inceput(nod *nou, nod **prim){
    /* introduce noul nod in capul listei */
    nou->urmator=*prim;
    if(!*prim) sfarsit=nou;
    *prim=nou;
}
```



Liste liniare simplu înlăntuite

Operații de bază - exemple

```
void adaugare_sfarsit(nod *nou, nod **ultim){  
    nou->urmator = NULL;  
    if(!(*ultim))  varf = nou;  
    else  
        (*ultim)->urmator=nou;  
    *ultim = nou;  
}  
// parcurgere si afisare lista  
void afisare_lista(nod *varf){  
    while(varf) {  
        printf("%d\n", varf->info);  
        varf=varf->urmator;}  
}
```

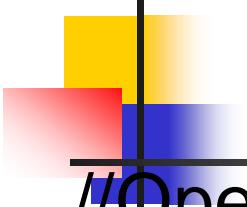


Liste liniare simplu înlăntuite

Operații de bază - exemple

```
/* cautare nod; in cazul in care se gaseste, vom returna  
pointerul catre el, altfel vom returna NULL */
```

```
nod *cauta(nod *varf, int informatie){  
    while(varf){  
        if(varf->info==informatie) return varf;  
        varf=varf->urmator;  
    }  
    return NULL;  
}
```



Liste liniare simplu înlățuită

Operații de bază – program complet

//Operatii de baza asupra listei

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
typedef struct elem {
```

```
    int info;
```

```
    struct elem *urmator;
```

```
} nod;
```

```
nod *varf, *sfarsit;
```

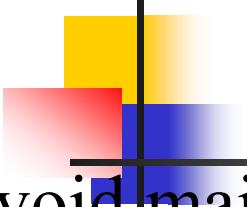
```
void adaugare_inceput(nod *,nod**);
```

```
void adaugare_sfarsit(nod *,nod**);
```

```
void afisare_lista(nod*);
```

```
nod* cauta(nod *,int);
```

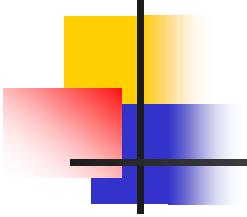
```
void AfisezMeniu(void);
```



Liste liniare simplu înlăntuite

Operații de bază – program complet

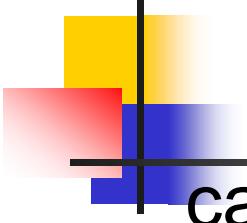
```
void main()
{
    char opt;  int aux;  nod *q;
    while(1) {
        AfisezMeniu();
        opt=(char)getc(stdin);
        switch(opt) {
            case 'i': q = (nod*)malloc(sizeof(nod));
                        printf("Introduceti valoarea: ");
                        scanf("%d", &aux);
                        q->info=aux;
                        adaugare_inceput(q, &varf);
                        break;
```



Liste liniare simplu înlăntuite

Operații de bază – program complet

```
case 's':    q = (nod*)malloc(sizeof(nod));
              printf("Introduceti valoarea: ");
              scanf("%d", &aux);
              q->info=aux;
              adaugare_sfarsit(q, &sfarsit);
              break;
case 'a':    afisare_lista(varf);
              break;
```



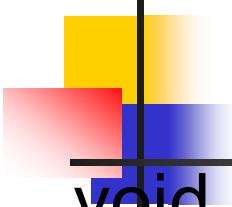
Liste liniare simplu înlăntuite

Operații de bază – program complet

```
case 'c':    printf("val. nodului cautat este: ");
              scanf("%d", &aux);
              if(cauta(varf, aux))
                  printf("Nodul se afla in lista\n");
              else
                  printf("Nodul nu se afla in lista\n");
              break;
case 'x':    exit(0);
default:     puts("Comanda eronata\n");
}
```

` }

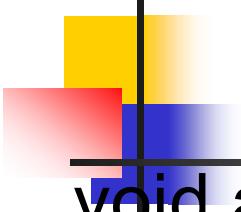
}



Liste liniare simplu înlăntuite

Operații de bază – program complet

```
void adaugare_inceput(nod *nou,nod **prim) {  
    nou->urmator=*prim;  
    if(!*prim)  
        sfarsit=nou;  
    *prim=nou; }  
  
void adaugare_sfarsit(nod *nou, nod **ultim) {  
    nou->urmator=NULL;  
    if(!(*ultim))  
        varf=nou;  
    else  
        (*ultim)->urmator=nou;  
    *ultim=nou; }
```

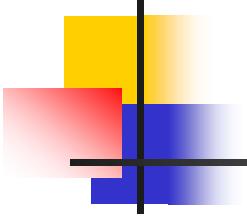


Liste liniare simplu înlăntuite

Operații de bază – program complet

```
void afisare_lista(nod *varf) {
    if(!varf)
        printf("Lista vida!\n");
    while(varf) {
        printf("%d\n", varf->info);
        varf=varf->urmator; }

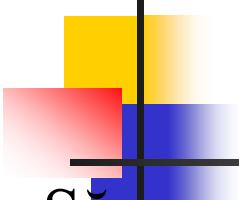
nod *cauta(nod *varf,int informatie) {
    while(varf) {
        if(varf->info==informatie) return varf;
        varf=varf->urmator; }
    return NULL; }
```



Liste liniare simplu înlăntuite

Operații de bază – program complet

```
void AfisezMeniu(void)
{
    puts("\ni ----- adaugare inceput");
    puts("\ns ----- adaugare sfarsit");
    puts("\na ----- afisare lista");
    puts("\nc ----- cautare nod in lista");
    puts("\nx ----- parasire program");
}
```



Liste liniare simplu înlăntuite

Exemplu de program complet

- Să se scrie un program care realizează evidența datelor de stare civilă a unui grup de persoane. Pentru fiecare persoană se rețin numele, vârstă și adresa. Programul principal va primi și va executa, în mod repetat, următoarele comenzi:

a= adaugă o persoană nouă în evidență

v= afișează vârstă unei persoane pentru care se dă numele

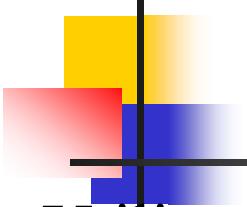
m= modifică adresa unei persoane pentru care se dă numele

d= șterge o persoană din evidență

l= listează toate persoanele

x= terminare program.

În această aplicație, numărul de persoane din evidență nu este cunoscut inițial și se modifică în timpul execuției programului.



Liste liniare simplu înlățuită

Exemplu de program complet

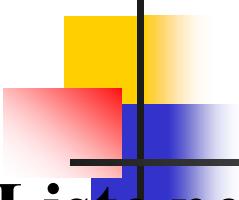
- Utilizarea unui tablou de structuri (fie acesta alocat static sau chiar dinamic) nu este o soluție eficientă pentru că spațiul necesar pentru tablou este *variabil* chiar în timpul execuției programului.
- Pentru memorarea unei evidențe ca cea descrisă în exemplu, se poate însă utiliza o structură de date dinamică, de tip listă simplu înlățuită. O astfel de structură este compusă din *noduri alocate dinamic* pe măsură ce este nevoie de ele.
- Fiecare nod conține, pe lângă informațiile despre o persoană, și adresa locației de memorie în care se găsește următorul nod din listă.
- Pentru accesul la listă trebuie să se cunoască adresa primului nod.

Liste liniare simplu înlăntuite

Exemplu de program complet

```
typedef struct pers {    /* tipul unui nod din lista */
    char *nume;
    int varsta;
    char *adresa;
    struct pers *urm;
} persoana;
persoana * lista = NULL;
```

- Câmpul *urm* este câmp de înlănțuire - pointer către următorul nod din listă.
- Trebuie cunoscută numai adresa primului element de tip *persoana*. Ultimul element din listă trebuie să aibă câmpul *urm* egal cu NULL.
- Adăugarea și eliminarea unui element din listă se realizează prin modificări ale legăturilor elementelor vecine. Spațiul de memorie poate fi alocat dinamic pentru fiecare element, individual.



Liste liniare simplu înlăntuite

Exemplu de program complet

Liste neordonate

■ Adăugarea într-o listă neordonată

Adăugarea unei noi persoane în listă presupune întâi alocarea de memorie pentru un nou nod. Inserția noului nod se face la începutul listei, el devenind cap de listă. Aceasta este varianta cea mai simplă de adăugare de noduri, în cazul listelor neordonate.

```
void adauga(void) {
```

```
    /* creaza un nou nod si il adauga la inceputul listei */  
    char nume[100], adr[100];
```

```
    int varsta;
```

```
    persoana *nou;
```

```
    printf("introduce numele, varsta si adresa: \n");  
    scanf("%s %d %s", nume, &varsta, adr);
```

Liste liniare simplu înlăntuite

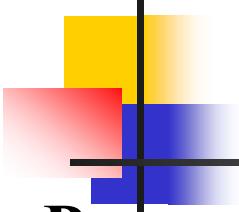
Exemplu de program complet

```
/* aloca spatiu pentru noul nod */
if (((nou=(persoana *)malloc(sizeof(persoana)))==NULL) ||
    ((nou->nume=(char *)malloc(strlen(nume)+1))==NULL)||  

    ((nou->adresa=(char *)malloc(strlen(adr)+1))==NULL)) {  

    printf("\n Eroare alocare memorie \n");
    exit(1);
}  

/*copiaza informatiile in nod */
strcpy(nou->nume, nume);
strcpy(nou->adresa, adr);
nou->varsta=varsta;
/* introduce noul nod in capul listei */
nou->urm=list;
list=nou;
}
```



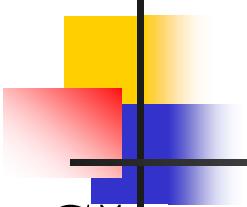
Liste liniare simplu înlăntuite

Exemplu de program complet

■ Parcursarea listei

Parcursarea (traversarea) unei liste este necesară pentru executarea unei anumite operații asupra tuturor nodurilor listei. Se utilizează un pointer curent care parcurge toate nodurile listei, de la primul nod până la sfârșitul listei. Sfârșitul listei este indicat de înlăntuirea nulă (NULL).

```
void list(void) {  
    /* afiseaza informatiile din toate nodurile */  
    persoana *p;  
    printf("Lista persoanelor este \n");  
    for (p=lista; p!=NULL; p=p->urm)  
        printf("%s %d \n",p->nume, p->varsta);  
}
```



Liste liniare simplu înlăntuite

Exemplu de program complet

■ Căutarea în lista neordonată

Lista este parcursă până când se întâlnește un nod având câmpul nume egal cu cel căutat sau până se ajunge la sfârșitul listei (NULL), dacă în listă nu este prezent un nod cu numele căutat.

```
persoana* cauta(char *nume) {
    persoana *p;
    for (p=list; p!=NULL; p=p->urm)
        if (strcmp(p->nume,nume)==0) return p;
    return NULL;
}
```

Utilizând funcția de căutare după nume, se pot realiza simplu funcțiile care afișează vârstă unei persoane cu nume dat sau care modifică adresa acelei persoane.

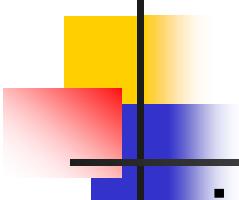
Liste liniare simplu înlăntuite

Exemplu de program complet

```
void varsta(void) {
    /* afiseaza varsta unei persoane */

    char nume[100];
    persoana *p;

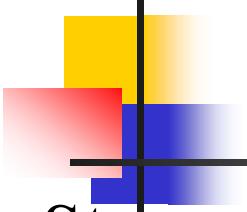
    printf("Introduce numele");
    scanf("%s",&nume);
    p=cauta(nume);
    if (p!=NULL)
        printf("Varsta lui %s : %d \n", p->nume, p->varsta);
    else printf("%s nu exista in lista\n",nume);
}
```



Liste liniare simplu înlăntuite

Exemplu de program complet

```
void modifica(void) {
    /* modifica adresa unei persoane */
    char nume[100], adr[100];
    persoana *p;
    printf("Introdu numele"); scanf("%s", nume);
    p=cauta(nume);
    if (p!=NULL) {
        printf("Introdu noua adresa");
        scanf("%s", adr);
        free(p->adresa);
        /* aloca spatiu pentru adresa noua */
        if ((p->adresa=
            (char *)malloc((strlen(adr)+1)))==NULL) {
            printf("\n Eroare aloc. mem. pt.adresa \n"); exit(1); }
        strcpy(p->adresa, adr); }
    else printf("%s nu exista in lista \n",nume);
}
```



Liste liniare simplu înlăntuite

Exemplu de program complet

■ Stergerea unui nod din listă

Fie $q1$ un pointer care indică nodul din listă care trebuie eliminat, iar $q2$ nodul care îl precede pe $q1$. Eliminarea lui $q1$ implică refacerea legăturilor, după cum urmează:

$$q2->urm = q1->urm.$$

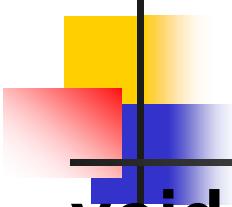
Un caz particular apare când nodul care trebuie şters este primul nod din listă. În acest caz, $q1=q2=listă$. Eliminarea lui $q1$ implică trecerea lui $q1->urm$ pe poziția de noul cap al listei.

Concluzie: Pentru eliminarea unui nod cu cheie dată din listă, sunt folosiți doi pointeri care parcurg lista, în căutarea nodului care trebuie eliminat. Un pointer $q1$ va căuta nodul care va trebui eliminat, și $q2$ nodul anterior lui.

Liste liniare simplu înlăntuită

Exemplu de program complet

```
void sterge(void) {
    char nume[100];
    persoana *q1,*q2;
    printf("Introd numele ");
    scanf("%s", nume);
    for (q1=q2=list; q1!=NULL; q2=q1, q1=q1->urm)
        if (strcmp(q1->nume,nume)==0) {
            if (q1==list) { /* sterge capul listei */
                list=q1->urm; }
            else { /* sterge un nod din interiorul listei */
                q2->urm=q1->urm; }
            free(q1->nume); free(q1->adresa); free(q1);
            return; }
    printf("Stergere: %s nu a fost gasit in lista \n",nume);
}
```

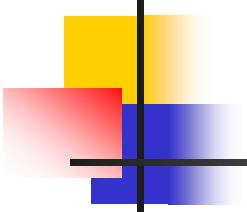


Liste liniare simplu înlăntuite

Exemplu de program complet

```
void afis_meniu(void){  
    printf("\n\nAlegeti optiunea: \n");  
    printf("m= modifica adresa unei persoane\n");  
    printf("v= afla varsta unei persoane \n");  
    printf("a= adauga o noua persoana\n");  
    printf("d= sterge o persoana din evidenta\n");  
    printf("l= listeaza toate persoanele \n");  
    printf("x= terminare program\n");  
}
```

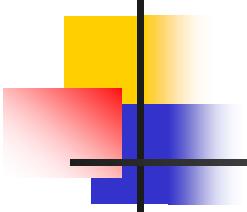
```
void main(void) {  
    char cmd;  
    int terminat=0;
```



Liste liniare simplu înlăntuite

Exemplu de program complet

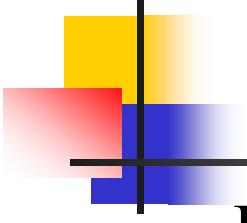
```
do {
    afis_meniu();
    cmd=getch();
    switch (cmd) {
        case 'm': modifica(); break;
        case 'v': varsta(); break;
        case 'a': adauga(); break;
        case 'd': sterge(); break;
        case 'l': list(); break;
        case 'x': terminat=1; break;
        default: printf("comanda gresita\n");
    }
} while (!terminat);
}
```



Liste ordonate

Principii

- Listele ordonate se construiesc de la bun început ordonate, plecând de la faptul că lista vidă este în mod intrinsec ordonată, iar nodurile ulterioare se inserează astfel încât să nu se strice ordinea deja existentă.
- Transpunerea directă a metodelor utilizate, cu succes, la tablouri sau fișiere, în cazul listelor înlănțuite este, cel mai adesea, total nerecomandabilă.

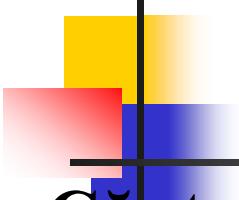


Liste ordonate

Exemplu de program

Programul anterior se completează cu cerința ca afișarea persoanelor să fie făcută întotdeauna în ordine alfabetică.

- Cel mai bine ar fi ca lista să fie păstrată în permanență ordonată alfabetic după nume.
- Orice operație de adăugare a unei noi persoane va trebui să țină cont de ordonarea listei și să insereze noul element la locul potrivit. În plus, faptul că lista este ordonată crește performanța operațiilor de căutare după nume în listă.
- Membrul care servește la identificarea nodurilor și asupra căruia operează criteriul de ordonare este numit membrul sau câmpul cheie (membrul *nume* din structură).



Liste ordonate

Exemplu de program

■ Căutarea unui element într-o listă ordonată

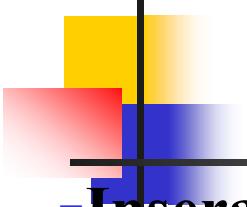
Căutarea unui element cu o cheie dată într-o listă ordonată se face traversând lista atâta timp cât nu s-a ajuns la sfârșitul listei ($p!=NULL$) și cheile din nodurile curente sunt mai mici decât cheia căutată ($strcmp(p->nume, nume)<0$).

În cazul în care nu există în listă un nod cu cheia căutată, în cele mai multe cazuri nu se mai ajunge cu parcurgerea listei până la sfârșitul ei, ci căutarea se oprește în momentul întâlnirii primului nod care are cheia mai mare decât cheia căutată.

Liste ordonate

Exemplu de program—căutarea unui element

```
persoana* cauta_ordonat(char *nume) {
    /*Cauta daca este prezenta o persoana cu numele dat.
    Daca da, returneaza un pointer la nodul in care se gaseste;
    Daca nu, returneaza NULL */
    persoana *p;
    for (p=lista; p!=NULL && strcmp(p->nume,nume)<0;
        p=p->urm) ;
    if (p!=NULL && strcmp(p->nume,nume)==0)
        return p;
    else
        return NULL;
}
```



Liste ordonate

Exemplu de program

■ Inserarea unui element într-o listă ordonată

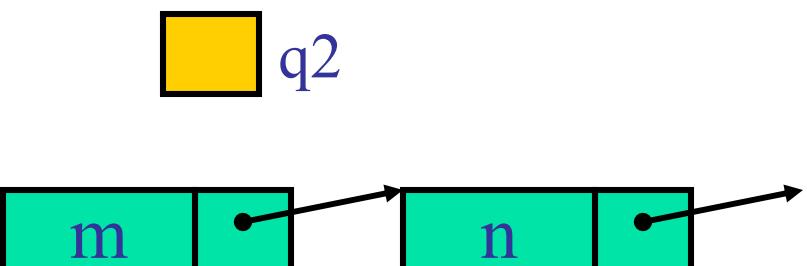
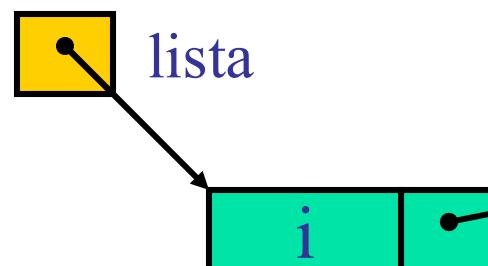
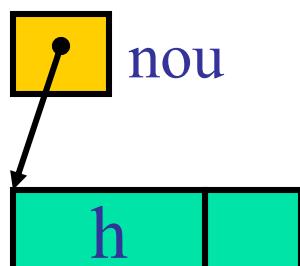
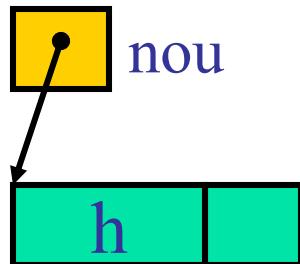
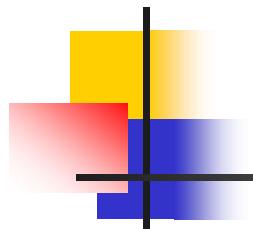
Adăugarea unui nou element la lista aflată în stare ordonată se face astfel încât lista să rămană ordonată și după adăugarea noului element. Pentru aceasta se face o căutare a poziției de inserare în listă: noul nod va fi inserat înaintea primului nod care are cheia mai mare decât cheia lui.

Se utilizează un pointer curent $q1$ care avansează în listă, atât timp cât nodurile curente au chei mai mici decât noua cheie de inserat și nu s-a ajuns la sfârșitul listei. Dacă $q1$ s-a oprit pe primul nod cu cheie mai mare decât cheia de inserat, noul nod trebuie inserat înaintea nodului $q1$. Este necesar deci accesul și la nodul anterior lui $q1$, pentru a putea reface legăturile.

Se utilizează pentru aceasta tehnica parcurgerii listei cu 2 pointeri, $q1$ și $q2$, $q2$ fiind tot timpul cu un pas în urma lui $q1$.

Liste ordonate

Cazurile 1 și 2



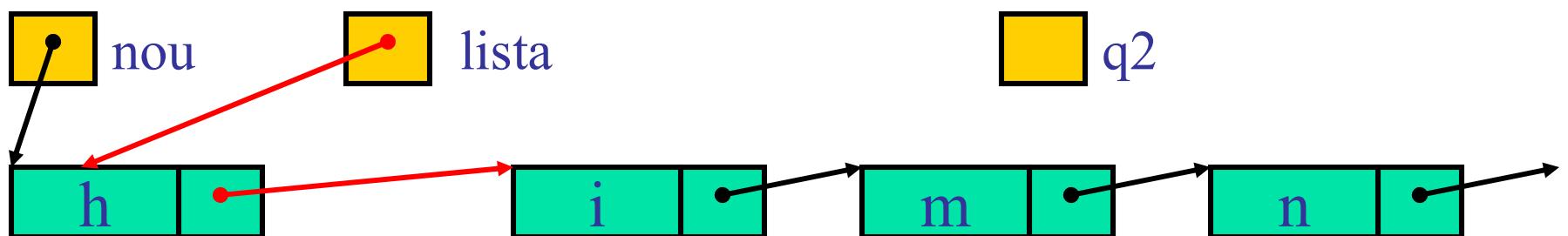
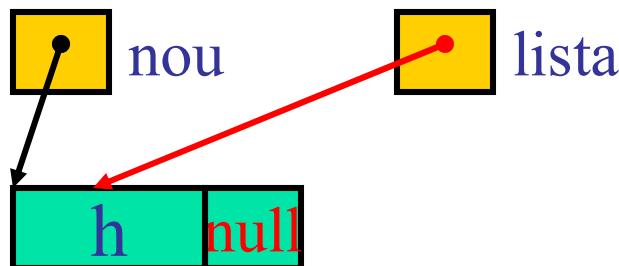
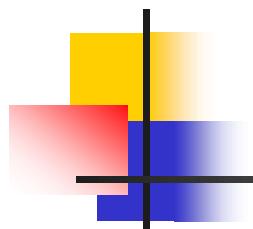
...

```
nou->urm=lista;  
lista=nou;
```

...

Liste ordonate

Cazurile 1 și 2

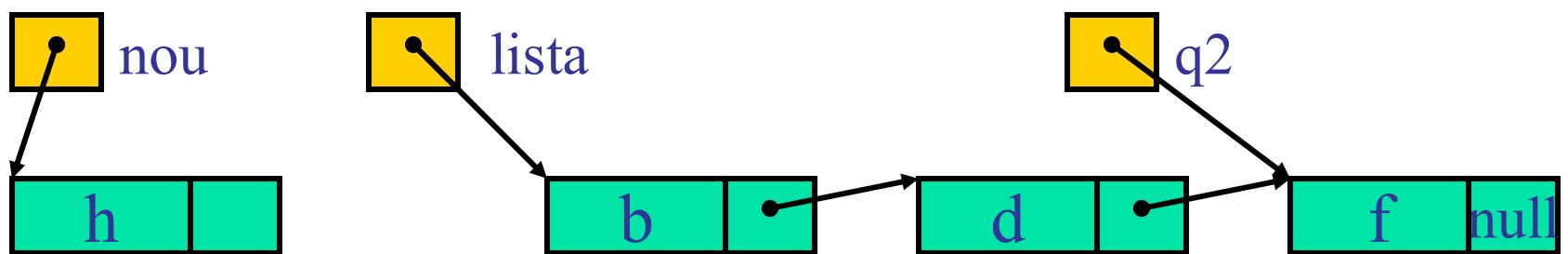
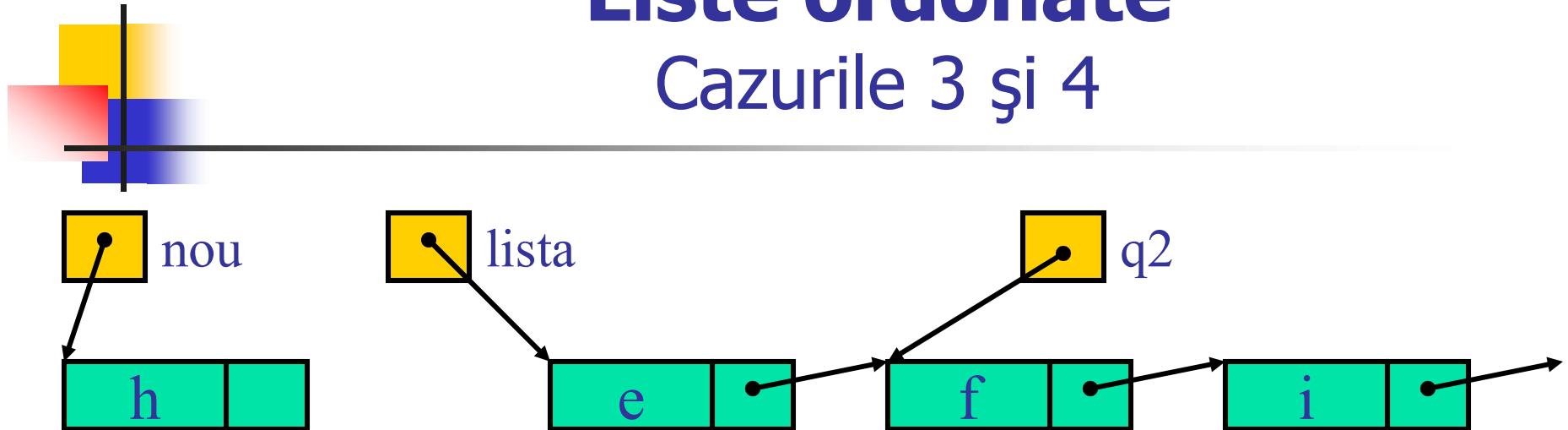


```
nou->urm=list;  
lista=nou;
```

...

Liste ordonate

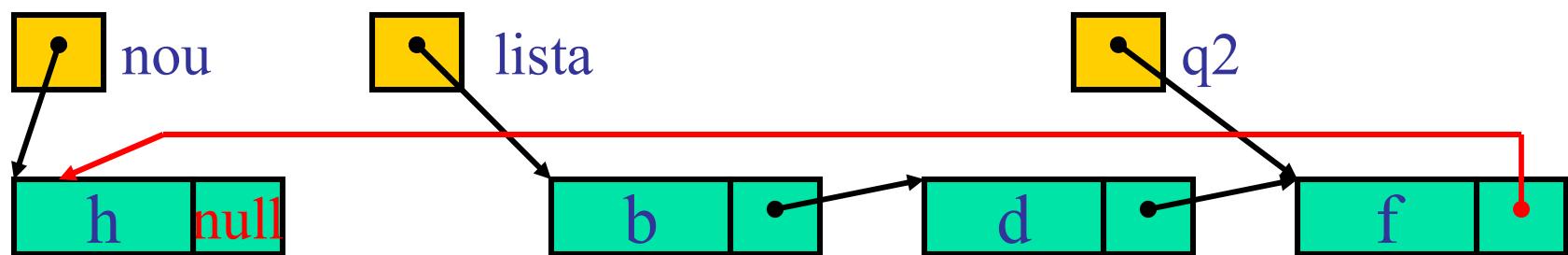
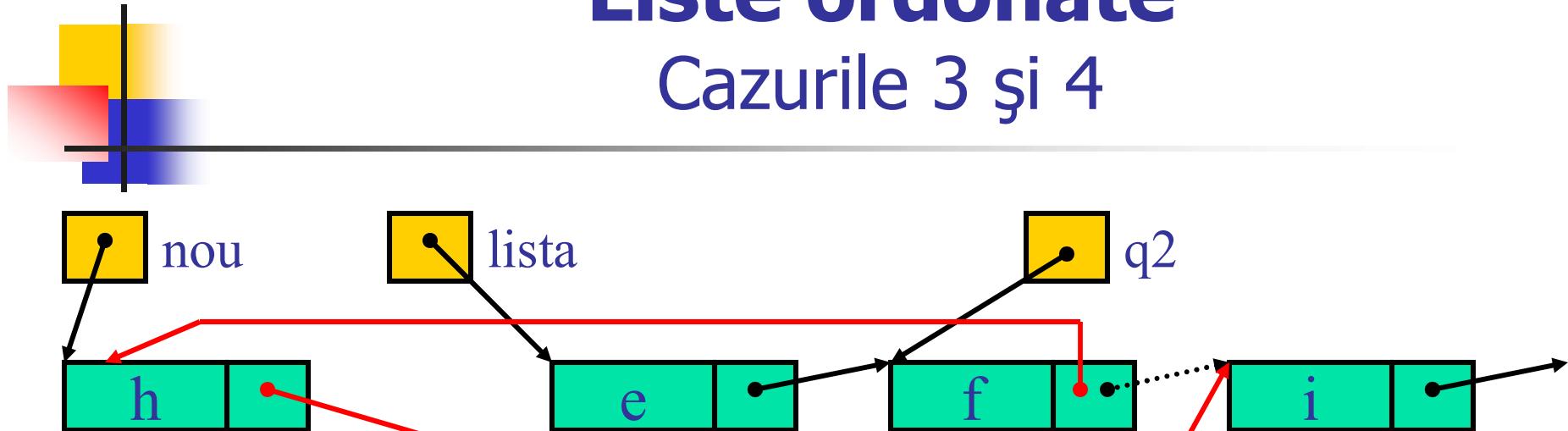
Cazurile 3 și 4



```
...
(q1=q2->urm;)
q2->urm=nou;
nou->urm=q1;
...
```

Liste ordonate

Cazurile 3 și 4



```
...  
(q1=q2->urm;)  
q2->urm=nou;  
nou->urm=q1;  
...
```

Liste ordonate

Exemplu de program–inserarea unui element

```
void adauga_ordonat(void) {
    /* se va insera in lista in ordinea alfabetica a numelui */
    char nume[100], adr[100];
    int varsta;
    persoana *nou, *q1, *q2;
    printf("introduce numele, varsta si adresa \n");
    scanf("%s %d %s", nume, &varsta, adr);
    /* aloca spatiu pentru noul nod */
    if(((nou=(persoana *)malloc(sizeof(persoana)))==NULL)||((nou->nume=(char *)malloc(strlen(nume)+1))==NULL)||((nou->adresa=(char *)malloc(strlen(adr)+1))==NULL)) {
        printf("\n Eroare alocare memorie \n"); exit(1); }
```

Liste ordonate

Exemplu de program–inserarea unui element

```
/*copiaza informatiile in nod */  
strcpy(nou->nume,nume);  
strcpy(nou->adresa, adr);  
nou->varsta=varsta;
```

```
/* introduce noul nod in lista,cauta locul de insertie */  
for(q1=q2=lista;q1!=NULL&&strcmp(q1->nume, nume)<0;  
q2=q1, q1=q1->urm)  
;
```

Liste ordonate

Exemplu de program – inserarea unui element

```
if (q1!=NULL && strcmp(q1->nume, nume)==0) {  
    printf("Eroare: %s apare deja in lista\n", nume);  
    return; }  
else  
if (q1!=q2) {  
/* inserarea nu se face la inceput - cazurile 3 si 4*/  
    q2->urm=nou;  
    nou->urm=q1; }  
else {  
/* inserarea se face la inceputul listei –cazurile 1 si 2 */  
    nou->urm=lista;  
    lista=nou; } }
```



Liste ordonate

Exemplu de program

■ Eliminarea unui element dintr-o listă ordonată

Eliminarea unui nod din lista ordonată se face la fel ca și în cazul listei neordonate, diferă doar modul de localizare a nodului ce trebuie șters: în căutare se tine cont de faptul că lista este ordonată și căutarea se face doar atâta timp cât nodurile curente au cheia mai mică decât cheia căutată.

```
void sterge_ordonat(void) {  
    /* scoate din evidență o persoana */  
    char nume[100];  
    persoana *q1,*q2;  
    printf("Introd numele ");  
    scanf("%s",&nume);  
    for(q1=q2=lista;q1!=NULL&&strcmp(q1->nume,nume)<0;  
        q2=q1,q1=q1->urm) ;
```

Liste ordonate

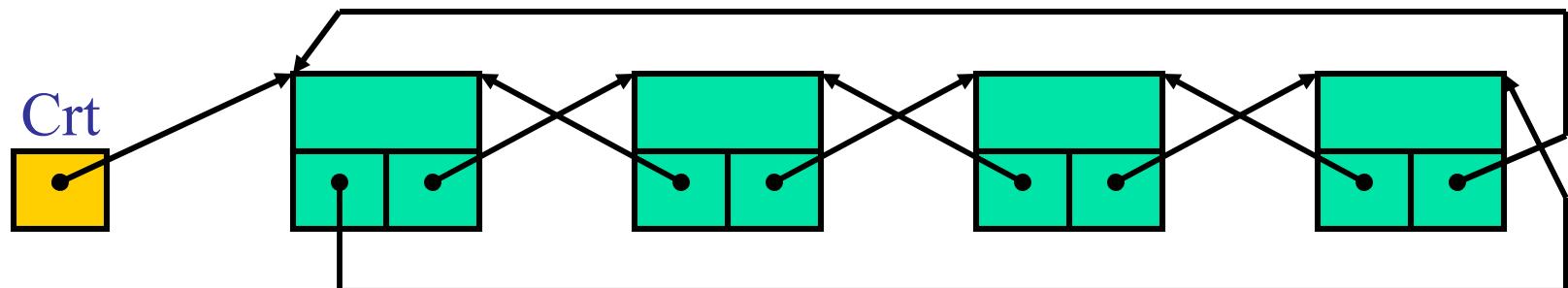
Exemplu de program–eliminarea unui element

```
if (q1!=NULL && strcmp(q1->nume,nume)==0) {  
    if (q1==lista) /* sterge capul listei */  
        lista=q1->urm;  
    else /* sterge un nod din interiorul listei */  
        q2->urm=q1->urm;  
    free(q1->nume);  
    free(q1->adresa);  
    free(q1);  
    return;  
}  
/* aici se ajunge daca numele nu a fost gasit */  
printf("Stergere: %s nu a fost gasit in lista \n",nume);  
}
```

Liste dublu înlăntuite

Listele dublu înlăntuite pot fi parcuse în ambele sensuri și se pot organiza ușor ca liste circulare (în care nu există nici un element prim sau ultim).

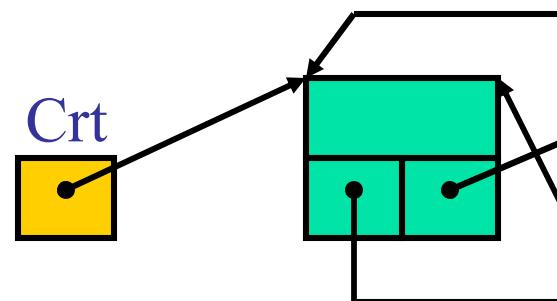
Caracteristica generală a listelor dublu înlăntuite este aceea că nodurile listei conțin 2 câmpuri de legătură (unul spre nodul anterior și altul spre nodul următor).

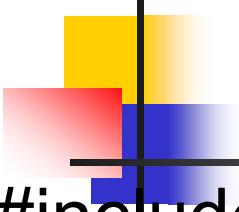


Liste dublu înlăntuite circulare

Avantajul listelor dublu înlăntuite circulare constă în ușurința și eleganța căutării în ele.

Dacă inițial în listă există un nod (ca în figură) funcțiile de lucru (inserare, căutare, suprimare etc.) sunt simple și generale, neexistând situații particulare pentru capetele listei. Nodul inițial poate fi un nod fictiv.





Liste dublu înlăntuite circulare

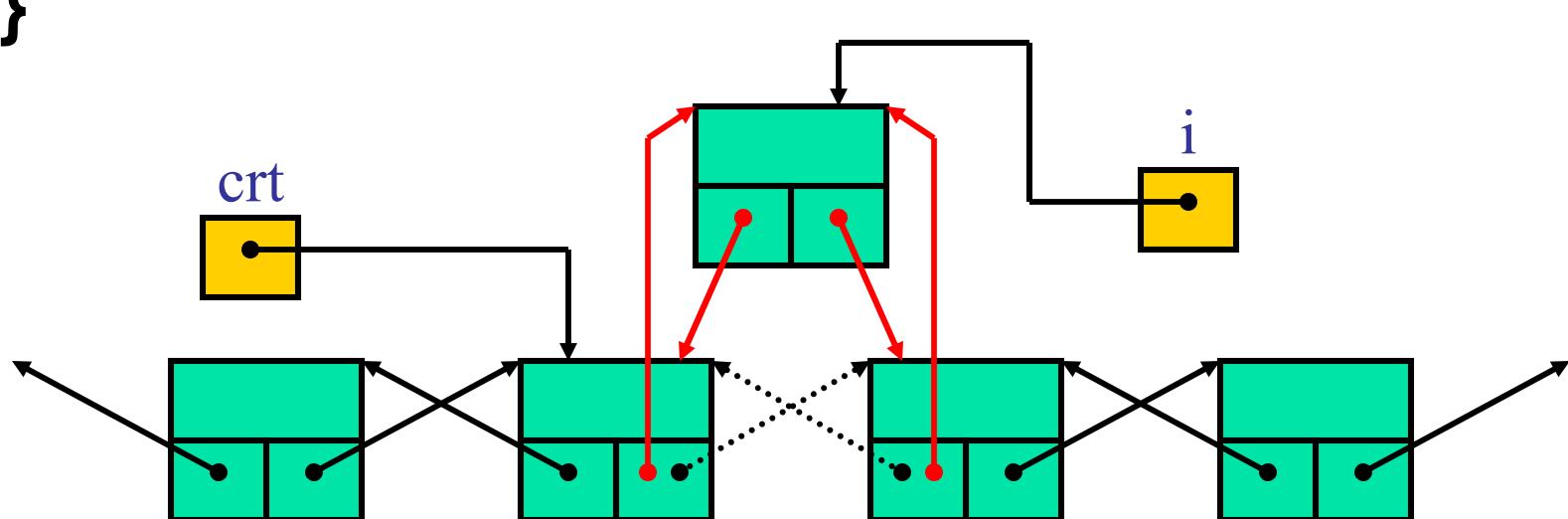
Descrierea listei

```
#include <stdio.h>
#include <stdlib.h>
typedef struct elem{
    int info;
    struct elem *urmator;
    struct elem *anterior;
}nod;
```

Liste dublu înlăntuite circulare

Inserare după nodul curent

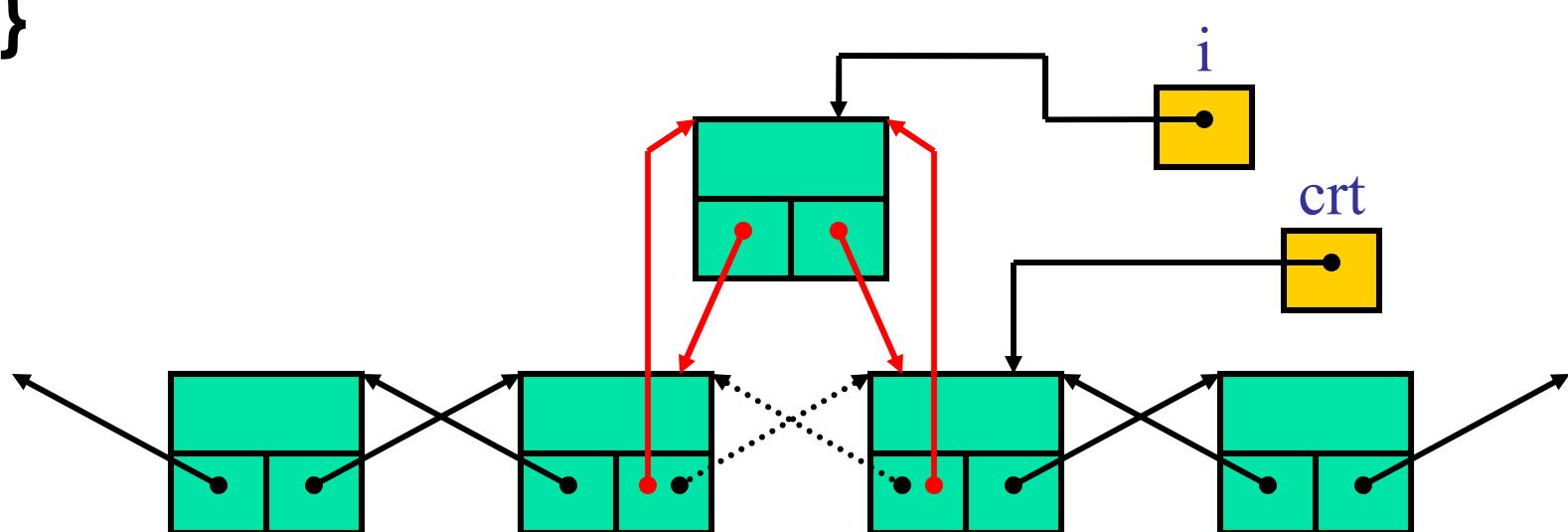
```
void adaug_dupa(nod *i, nod **crt){  
    i->urmator = (*crt)->urmator;  
    i->anterior = *crt;  
    (*crt)->urmator = i;  
    (i->urmator)->anterior = i;  
}
```



Liste dublu înlățuită circulară

Inserare înainte de nodul curent

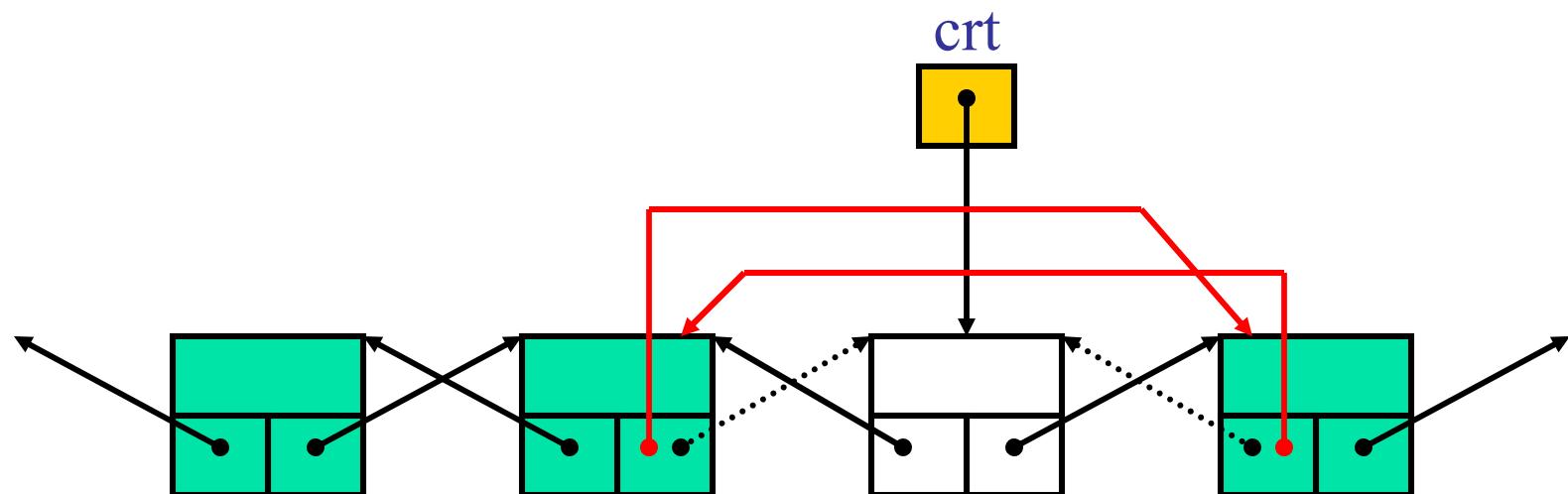
```
void adaug_inainte(nod *i, nod **crt){  
    i->anterior = (*crt)->anterior;  
    i->urmator = *crt;  
    (*crt)->anterior = i;  
    (i->anterior)->urmator = i;  
}
```

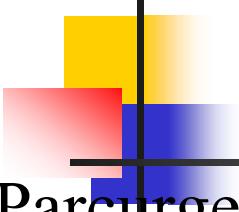


Liste dublu înlățuite circulare

Ștergerea nodului curent

```
void sterge(nod *crt){  
    crt->anterior->urmator=crt->urmator;  
    crt->urmator->anterior=crt->anterior;  
    free(crt);  
}
```





Liste dublu înlăntuite circulare

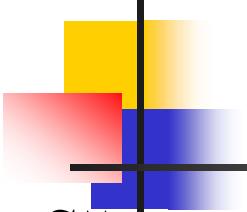
Parcurgerea listei circulare

Parcurgerea unei liste circulare necesită marcarea elementului cu care a început căutarea. Presupunând că *prim* este un pointer spre acest element, parcurgerea completă a listei se poate face astfel:

```
void parcurgere(nod **prim){  
    nod *p=*prim;  
    printf("%d\n", p->info);  
    p=p->urmator;  
    while(p!=(*prim)){  
        printf("%d\n", p->info);  
        p=p->urmator; } }
```

Dacă lista are un singur element nu se intră în ciclul *while*.

Pentru a parcurge lista în sens invers se folosesc legăturile către nodurile anterioare.



Liste multiplu înlăntuite

Structuri multilistă (optional)

- Să se scrie un program care realizează evidența datelor de stare civilă ale unui grup de persoane. Pentru fiecare persoană se rețin numele și vârsta.

Programul principal va primi și executa, în mod repetat, următoarele comenzi:

a = Adaugă în evidență o nouă persoană ale cărei date se citesc

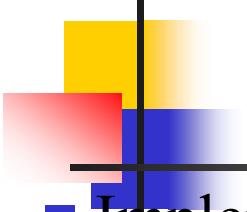
t = Caută un nume în evidență

s = Sterge o persoană din evidență

n = Afisează alfabetic persoanele din evidență

v = Afisează persoanele în ordinea vîrstei

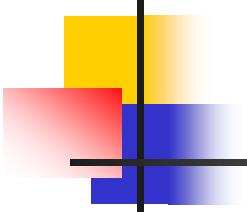
x = Terminare program



Liste multiplu înlăntuite

Structuri multilistă

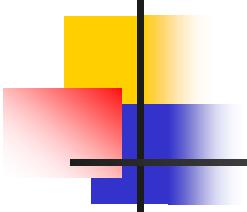
- Implementarea evidenței sub formă de listă simplu înlăntuită ordonată după nume sau ca listă simplu înlăntuită ordonată după vârstă este nesatisfăcătoare, deoarece s-ar impune o operație de reordonare a listei de fiecare dată când se comandă afișarea după celălalt criteriu.
- O soluție este ca persoanele să fie păstrate într-o listă multiplu înlăntuită. Această listă are în noduri informațiile despre persoane, noduri care sunt înlăntuite simultan în două liste ordonate, una după nume și una după vârstă. Se subliniază faptul că **aceleași noduri fizice fac parte simultan din ambele liste!** (informația despre fiecare persoană apare o singură dată). Altfel, dacă informațiile ar fi păstrate în 2 liste ordonate independente, redundanța informațiilor care se introduce astfel complică operațiile de adăugare și ștergere a unei persoane din evidență (aceste operații ar trebui făcute pe ambele liste).



Liste multiplu înlăntuite

Structuri multilistă

- Se definește tipul structurat *persoana* pentru tipul nodurilor listei. Pe lângă câmpurile de date (*nume*, *varsta*), aceasta structură conține două câmpuri de înlăntuire: *urm_nume* și *urm_varsta*. Câmpul *urm_nume* indică adresa următorului nod care are numele alfabetic după numele din nodul curent, iar câmpul *urm_varsta* indică următorul nod care are vîrstă mai mare decât vîrstă din nodul curent.
- Pentru accesul la structura de date, sunt necesari doi pointeri spre cele două începuturi ale listei - un pointer spre capul listei ordonată după nume, în variabila globală *lista_nume*, și un pointer spre capul listei după vîrstă în variabila globală *lista_varsta*.



Liste multiplu înlăntuite

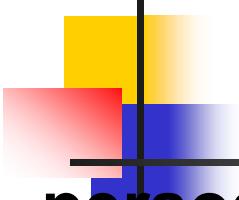
Structuri multilistă

- Dintre operațiile pe structura de date corespunzătoare evidenței persoanelor, operațiile de căutare și afișare se efectuează ca operații obișnuite pe liste simplu înlăntuite.

```
typedef struct pers{  
    char *nume;  
    int varsta;  
    struct pers *urm_nume;  
    struct pers *urm_varsta;  
} persoana;
```

```
persoana *lista_nume=NULL, *lista_varsta=NULL
```

```
;
```



Liste multiplu înlăntuite

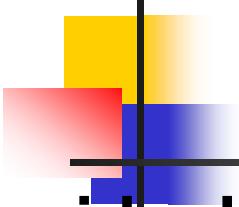
Structuri multilistă

```
persoana *cauta( char *n) {
    persoana *p;
    for( p = lista_nume; p !=NULL ; p =p->urm_nume)
        if( strcmp(p->nume,n ) == 0) return p;
    return NULL; }

void afis_nume( void ) {
    persoana *p;
    for( p =lista_nume; p != NULL ; p =p->urm_nume)
        printf("%s %d\n",p->nume, p->varsta);

}

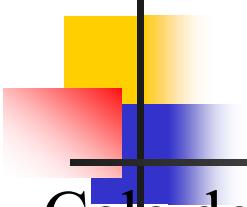
void afis_varsta( void ) {
    persoana *p;
    for( p =lista_varsta; p != NULL ; p =p->urm_varsta)
        printf("%s %d\n",p->nume, p->varsta);
}
```



Liste multiplu înlăntuite

Structuri multilistă

```
void adauga(char *n, int v) {
    persoana *t;
    if(( t = cauta( n ) ) != NULL) {
        printf("Eroare: %s exista deja \n", n);
        return; }
    else
    if(( t = (persoana*)malloc(sizeof(persoana))) == NULL ||
       ( t->nume = (char*)malloc(strlen(n)+1)) == NULL ) {
        printf("Eroare : memorie insuficienta\n"); exit( 1 ); }
    else {
        strcpy( t->nume,n);
        t->varsta = v;
        adauga_nume(t);
        adauga_varsta(t); }
}
```

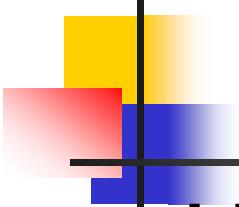


Liste multiplu înlăntuite

Structuri multilistă

- Cele două funcții, adauga_nume și adauga_varsta, realizează înlănțuirile nodului nou în cele 2 liste, lista_varsta respectiv lista_nume. Fiecare dintre aceste două funcții este o operație simplă de inserție a unui nou nod în lista ordonată respectivă.

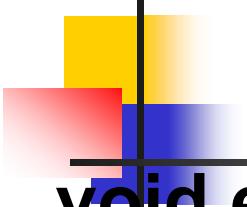
```
void adauga_nume(persoana *nou) {  
    persoana *q1,*q2;  
    for(q1=q2=list_nume; q1!=NULL &&  
        strcmp(q1->nume, nou->nume)<0;  
        q2=q1, q1=q1->urm_nume);  
    nou->urm_nume = q1;  
    if(q1 == q2) list_nume=nou;  
    else  
        q2->urm_nume = nou;  
}
```



Liste multiplu înlăntuite

Structuri multilistă

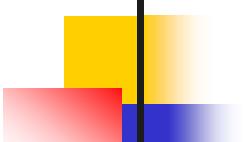
```
void adauga_varsta( persoana *nou) {  
    persoana *q1,*q2;  
    for(q1=q2=list_varsta;  
        q1!=NULL && q1->varsta<nou->varsta;  
        q2=q1,q1=q1->urm_varsta);  
    nou->urm_varsta = q1;  
    if(q1 == q2) list_varsta=nou;  
    else  
        q2->urm_varsta = nou;  
}
```



Liste multiplu înlăntuite

Structuri multilistă

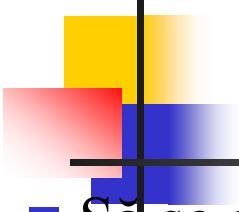
```
void elimin( char *s){  
    scot_nume( s );  
    scot_varsta( s );  
}  
void scot_nume( char *l) {  
    persoana *q1,*q2;  
    for(q1=q2=list_nume; q1!= NULL &&  
        strcmp( q1->nume,l)<0; q2=q1, q1=q1->urm_nume);  
    if( q1 != NULL && strcmp (q1->nume,l) ==0)  
        if(q1 == q2 ) list_nume=list_nume->urm_nume;  
        else  
            q2->urm_nume = q1->urm_nume;  
    else  
        printf(" Eroare: %s nu apare in evidenta\n",l);  
}
```



Liste multiplu înlăntuite

Structuri multilistă

```
void scot_varsta( char *l) {
    persoana *q1,*q2;
    for(q1=q2=list_varsta; q1!= NULL &&
        strcmp( q1->nume,l)!=0; q2=q1, q1=q1-
>urm_varsta);
    if( q1 != NULL && strcmp (q1->nume,l) ==0)
        if(q1 == q2 ) list_varsta= list_varsta-
>urm_varsta;
    else
        q2->urm_varsta = q1->urm_varsta;
    else
        printf(" Eroare: %s nu apare in evidenta\n",l);
}
```



Liste multiplu înlăntuite

Exemplu de program

- Să se scrie un program care realizează evidența datelor privitoare la întreprinderile și salariații unui județ. Pentru fiecare întreprindere, se rețin denumirea, profilul activității și lista persoanelor angajate.

Pentru fiecare persoană se rețin numele și vârsta. Programul principal va primi și executa, în mod repetat, următoarele comenzi:

i = Adaugă în evidență o nouă întreprindere a cărei date se citesc

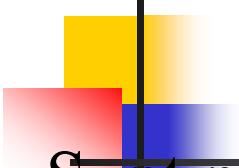
a = Adaugă un nou angajat la o anumită întreprindere

s = Afisează alfabetic persoanele angajate la o anumită întreprindere

l = Afisează alfabetic toate întreprinderile

x = Terminare program

Structura de date avantajoasă în acest caz este o listă de întreprinderi, fiecare întreprindere conținând la rândul său o listă a persoanelor angajate.

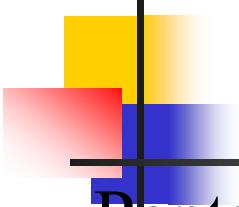


Liste multiplu înlăntuite

Exemplu de program

- Sunt necesare două tipuri de noduri: un tip de nod pentru lista întreprinderilor (lista principală) și un alt tip de nod pentru persoanele din listele de salariați.

```
typedef struct pers {  
    char *nume;  
    int varsta;  
    struct pers *urm;  
} persoana;  
  
typedef struct intr {  
    char * nume;  
    char * profil;  
    persoana * salariati;  
    struct intr * urm;  
} intreprindere;
```

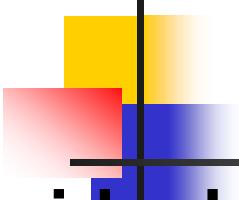


Liste multiplu înlăntuite

Exemplu de program

- Pentru a accesa întreaga structură de date, este nevoie de un pointer la primul nod al listei principale, lista de intreprinderi:

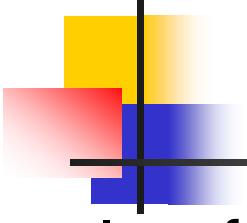
```
intreprindere * lista=NULL;
```
- Avantajele utilizării multilistei:
 - Se evită repetarea informațiilor întreprinderi;
 - Permite generarea simplă de statistici/listări ale datelor pe fiecare intreprindere.
- Operațiile pe lista principală (lista de intreprinderi) sunt operații obișnuite pe o listă simplu înlăntuită



Liste multiplu înlăntuită

Exemplu de program

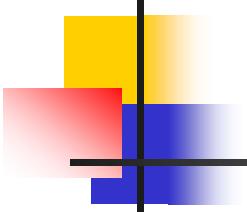
```
void adauga_intr( char *nume, char *profil) {  
    intreprindere *q1,*q2, *nou;  
    for(q1=q2=lista;q1!=NULL&&strcmp(q1->nume,nume)<0;  
        q2=q1, q1=q1->urm);  
  
    if((nou=(intreprindere *)malloc(sizeof(intreprindere)))==  
        NULL||  
        (nou->nume=(char*)malloc(strlen(nume)+1)) ==  
        NULL ||  
        (nou->profil=(char*)malloc(strlen(profil)+1))== NULL){  
        printf("Eroare : memorie insuficienta\n");  
        exit(1);  
    }  
}
```



Liste multiplu înlăntuită

Exemplu de program

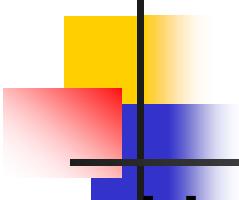
```
else {  
    strcpy( nou->nume,nume);  
    strcpy(nou->profil,profil);  
    nou->salariati=NULL;  
}  
nou->urm = q1;  
if(q1 == q2 )  
    lista=nou;  
else  
    q2->urm = nou;  
}
```



Liste multiplu înlățuită

Exemplu de program

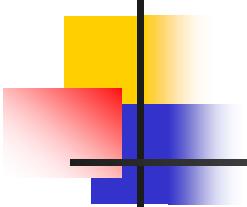
```
intreprindere * cauta_intr(char * n) {  
    intreprindere *intr;  
  
    for(intr=lista; intr!=NULL &&(strcmp(intr->nume, n)<0);  
        intr=intr->urm);  
    if (intr !=NULL && strcmp(intr->nume, n)==0)  
        return intr;  
    return NULL;  
}
```



Liste multiplu înlăntuită

Exemplu de program

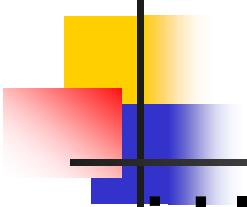
```
void adauga_persoana(char *i, char *n, int v) {  
    persoana *t; intreprindere *intr;  
    intr=cauta_intr(i);  
    if (intr==NULL) {  
        printf("Eroare: intreprinderea %s nu exista \n", i);  
        return; }  
    else  
    if((t =(persoana*)malloc(sizeof(persoana))) == NULL ||  
        (t->nume=(char*)malloc(strlen(n)+1)) == NULL ) {  
        printf("Eroare : memorie insuficienta\n");  
        exit( 1 ); }  
    else { strcpy( t->nume,n);  
        t->varsta = v;  
        intr->salariati =adauga(intr->salariati,t); }  
}
```



Liste multiplu înlăntuită

Exemplu de program

```
persoana*adauga( persoana*lista_pers,persoana *nou)
{
    persoana *q1,*q2;
    for(q1=q2=lista_pers; q1!=NULL &&
        strcmp(q1->nume,nou->nume)<0;q2=q1, q1=q1->urm);
    nou->urm = q1;
    if(q1 == q2 ) return nou;
    else {
        q2->urm = nou;
        return lista_pers;
    }
}
```

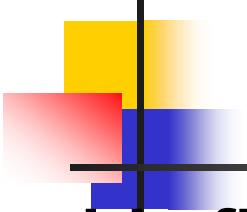


Liste multiplu înlăntuită

Exemplu de program

```
void listeaza_salariati(persoana * l) {
    persoana * p;
    for (p=l; p!=NULL; p=p->urm)
        printf(" %s    %2d \n", p->nume, p->varsta);
}

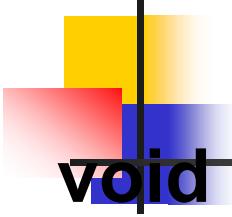
void listeaza(void) {
    intreprindere * intr;
    for (intr=lista; intr!=NULL; intr=intr->urm) {
        printf("Intreprinderea %s cu profil de %s \n",
               intr->nume, intr->profil);
        printf("Lista salariatilor: \n ");
        listeaza_salariati(intr->salariati);
    }
}
```



Liste multiplu înlăntuite

Exemplu de program

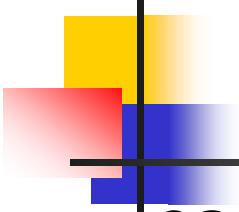
```
void afis_meniu(void){  
    printf(" i = adauga o noua intreprindere \n");  
    printf(" a = adauga un salariat la o intreprindere\n");  
    printf(" l = listeaza tot \n");  
    printf(" s = listeaza toti salariatii unei intreprinderi\n");  
    printf(" x = Terminarea programului \n\n");  
}
```



Liste multiplu înlăntuite

Exemplu de program

```
void main(void) {
    char cmd;
    int terminat=0;
    char s1[30], s2[30]; int v;
    intreprindere * intr;
    do {
        afis_meniu();
        cmd=getch();
        switch (cmd) {
            case 'i':
                printf("Introduceti numele si profilul intr:");
                scanf("%s %s", &s1, &s2);
                adauga_intr(s1,s2);
                break;
        }
    } while (!terminat);
```



Liste multiplu înlăntuite

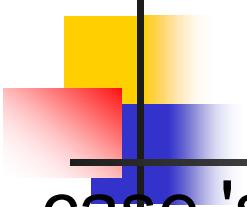
Exemplu de program

```
case 'a':
```

```
    printf("Introduceti intreprinderea:");
    scanf("%s", s1);
    printf("Introduceti numele si varsta \n");
    scanf("%s %d", s2, &v);
    adauga_persoana(s1, s2, v);
    break;
```

```
case 'l':
```

```
    listeaza();
    break;
```



Liste multiplu înlăntuite

Exemplu de program

```
case 's':  
    printf("Introduceti numele intreprinderii \n");  
    scanf("%s", &s1);  
    intr=cauta_intr(s1);  
    if (!intr)  
        printf("Eroare: intr. %s nu exista \n", s1);  
    else  
        listeaza_salariati(intr->salariati);  
    break;  
  
case 'x': terminat=1;  
    break;  
default: printf("comanda gresita\n"); } }  
while (!terminat); }
```

Probleme rezolvate

Metoda lui Quine

1. Gasiti valorile de adevar pentru A, B si C astfel incat wff $(A \rightarrow B) \rightarrow C$ sa nu fie echivalent cu $A \rightarrow (B \rightarrow C)$.

2. Utilizati metoda lui Quine sa analizati valoarea de adevar pentru urmatorul wff.

$$(A \wedge B \rightarrow C) \wedge (A \rightarrow B) \rightarrow (A \rightarrow C).$$

3. Dovediti fiecare dintre echivalentele urmatoare fara tabele de adevar.

a. $(A \rightarrow B) \equiv \neg(A \wedge \neg B)$.

b. $(A \wedge B \rightarrow C) \rightarrow (A \wedge C \rightarrow B) \equiv A \wedge C \rightarrow B$.

Solutii

1. Un raspuns este $A = \text{False}$, $C = \text{False}$ iar B poate fi True sau False.

2. Daca W este wff, atunci se calculeaza $W(C/\text{True})$ si $W(C/\text{False})$ ca in continuare:

$$W(C/\text{True}) = (A \wedge B \rightarrow \text{True}) \wedge (A \rightarrow B) \rightarrow (A \rightarrow \text{True}) \equiv \text{True} \wedge (A \rightarrow B) \rightarrow \text{True} \equiv \text{True}.$$

$$W(C/\text{False}) = (A \wedge B \rightarrow \text{False}) \wedge (A \rightarrow B) \rightarrow (A \rightarrow \text{False}) \equiv \neg(A \wedge B) \wedge (A \rightarrow B) \rightarrow \neg A.$$

If $U = W(C/\text{False})$, atunci se calculeaza $U(A/\text{True})$ si $U(A/\text{False})$ ca in continuare:

$$U(A/\text{True}) \equiv \neg(\text{True} \wedge B) \wedge (\text{True} \rightarrow B) \rightarrow \neg \text{True} \equiv \neg B \wedge B \rightarrow \text{False} \equiv \text{False} \rightarrow \text{False} \equiv \text{True}.$$

$$U(A/\text{False}) \equiv \neg(\text{False} \wedge B) \wedge (\text{False} \rightarrow B) \rightarrow \neg \text{False} \equiv \neg \text{False} \wedge \text{True} \rightarrow \text{True} \equiv \text{True}.$$

Cat timp toate expresiile evaluate sunt True, rezulta ca W este o tautologie

$$\begin{aligned}3. \quad a. \neg(A \wedge \neg B) &\equiv \neg A \vee \neg \neg B \equiv \neg A \vee B \equiv A \rightarrow B. \\b. (A \wedge B \rightarrow C) &\rightarrow (A \wedge C \rightarrow B) \\&\equiv \neg(A \wedge B \rightarrow C) \vee (A \wedge C \rightarrow B) \\&\equiv \neg(A \wedge B \wedge \neg C) \vee (A \wedge C \rightarrow B) \\&\equiv \neg(A \wedge B \wedge \neg C) \vee \neg(A \wedge C) \vee B \\&\equiv (A \vee \neg A \vee \neg C \vee B) \wedge (B \vee \neg A \vee \neg C \vee B) \wedge (\neg C \vee \neg A \vee \neg C \vee B) \\&\equiv \text{True} \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg A \vee B) \\&\equiv B \vee \neg A \vee \neg C \\&\equiv B \vee \neg(A \wedge C) \equiv A \wedge C \rightarrow B.\end{aligned}$$

Logică și Structuri Discrete

Functii

După James Hein
Discrete Structures, Logic and Computability
Section 2.1(pp.60-63); Section2.3(pp.89-93);
Section 2.2 (pp.77-78 only Composition);Section 3.3(pp. 146- 150);

De ce?

**Functiile pot fi programe iar programele
pot fi functii [combinari de functii]**

Definiție

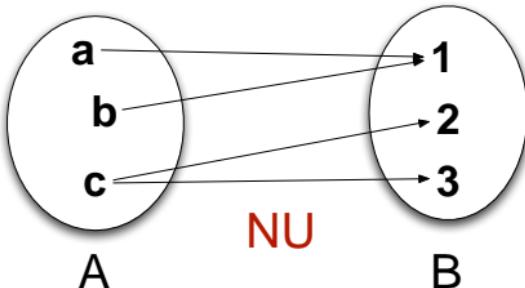
Fie **A** și **B** două multimi și pentru fiecare element din A **asociem** un element din B. O astfel de asociere se numește **funcție** de la A la B.

Important

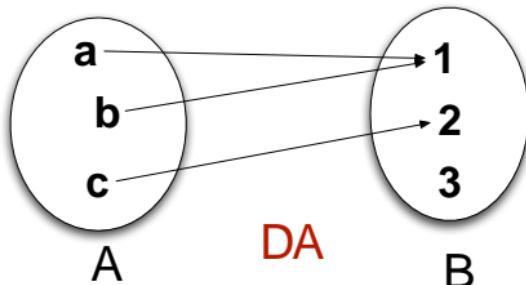
Un element din A este asociat cu **exact un** element din B.

Altfel spus, dacă $x \in A$ e asociat cu $y \in B$ atunci x nu este asociat cu un alt element din B.

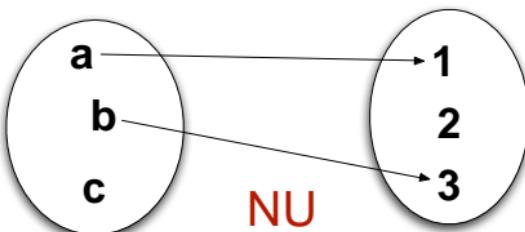
Sunt acestea Funcții ?



NU



DA



NU

Este de fapt o funcție parțială
(nu e definită pentru orice element din
A)

A B

Definirea Funcțiilor (1)

Scriem toate asocierile

Exemplu

$$A = \{a, b, c\}$$

$$B = \{1, 2, 3\}$$

$$g(a)=1, g(b) = 1, g(c) = 2$$

Definirea Funcțiilor (2)

Utilizăm o formulă

Exemplu

Vrem ca funcția f să asocieze orice număr natural cu pătratul numărului

$$f(x) = x^2$$

Definirea Funcțiilor (3)

Definirea după caz

Exemplu

$\text{abs} : \mathbb{R} \rightarrow \mathbb{R}$

$$\text{abs}(x) = \begin{cases} x & \text{dacă } x \geq 0 \\ -x & \text{dacă } x < 0 \end{cases}$$

Exemplu (utilizând reguli if-then-else)

$$\text{abs}(x) = \text{if } x \geq 0 \text{ then } x \text{ else } -x$$

Notății și Terminologie (1)

1. Funcțiile sunt notate ușual cu litere mici precum f , g , h sau prin **nume descriptive** (ex. **succ** (de la **successor/succesor**))

2. Dacă f este o funcție de la A la B care asociază $x \in A$ cu $y \in B$ scriem $f(x) = y$ sau $y = f(x)$

. $f(x)$ se citește "f în x", "f de x", "f aplicat lui x"

. $f(x) = y$ se citește "f mapează pe x în y"

. x este un **argument** de-al lui f

. y este o **valoare** de-a lui f

3. Dacă f este o funcție de la A la B scriem $f : A \rightarrow B$

. spunem că f are **tipul** $A \rightarrow B$

. $A \rightarrow B$ reprezintă **mulțimea tuturor funcțiilor de la A la B**

. A este **domeniul** lui f

. B este **codomeniul** lui f

Notății și Terminologie (2)

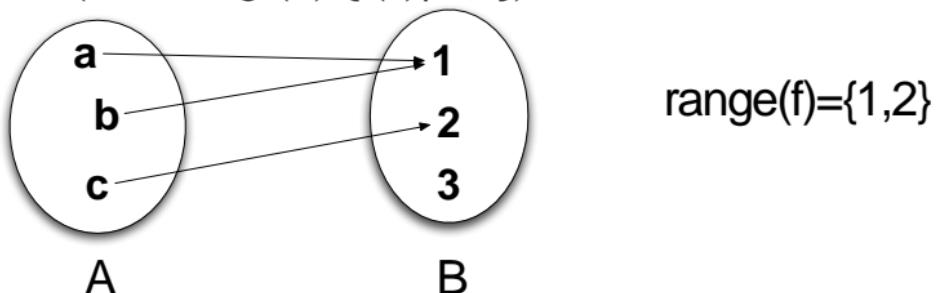
4. Pentru $f : A_1 \times A_2 \times \dots \times A_n \rightarrow B$ spunem că f are **arietatea n** sau că **f are n argumente** și scriem $f(x_1, x_2, \dots, x_n)$

5. O funcție cu două argumente se numește **funcție binară**

. $f(x,y)$ se poate scrie în forma **infix** ca $x f y$

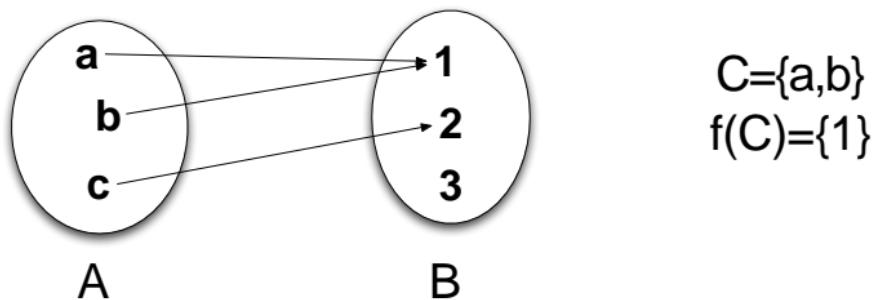
.pentru $+ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ putem scrie $+(x,y)$ sau $x + y$

6. **Imaginea funcției f (sau mulțimea valorilor lui f)** este mulțimea elementelor din B care sunt asociate cu un element din A (adică $\text{range}(f) = \{f(a) | a \in A\}$)



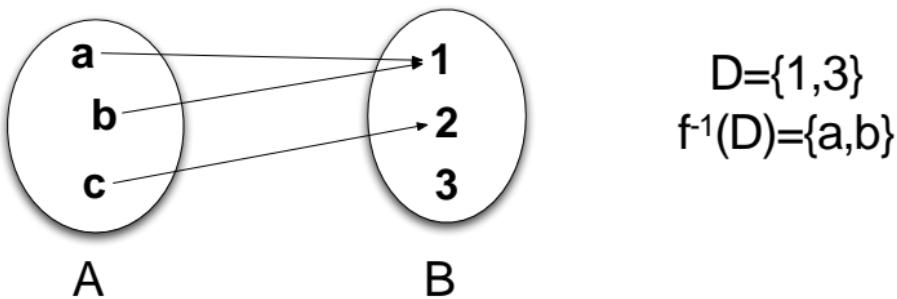
Notății și Terminologie (3)

7. Dacă $C \subset A$, **îmaginea** lui C prin f este multimea tuturor valorilor din B asociate elementelor din C
(adică $f(C) = \{f(a) | a \in C\}$)



Notății și Terminologie (4)

8. Dacă $D \subset B$, **imaginea inversă (pre-imaginea)** a lui D prin f este multimea valorilor din A asociate cu un element din D (adică $f^{-1}(D) = \{x \in A \mid f(x) \in D\}$)



Egalitatea Funcțiilor

Dacă f și g sunt funcții de tipul $A \rightarrow B$, f și g sunt egale dacă $f(x) = g(x)$ pentru orice $x \in A$ și scriem

$$f = g$$

Exemplu

$$f : N \rightarrow N \quad g : N \rightarrow N$$

$$f(x) = x^2 \quad g(x) = x * x$$

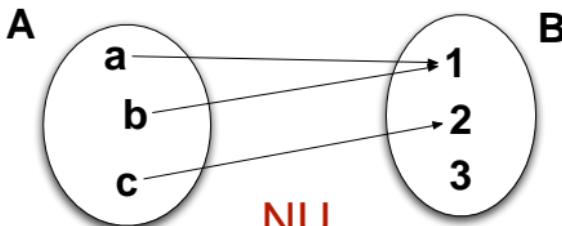
e ușor de văzut că $f = g$:)

Proprietăți - Funcții Injective

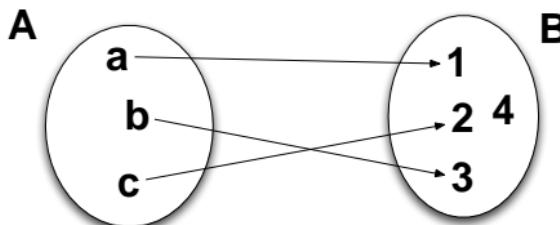
O funcție $f : A \rightarrow B$ este **injectivă** dacă
nicicare două elemente din A nu se
mapează în același element din B

Formal, $f : A \rightarrow B$ este **injectivă** dacă
pentru orice $x, y \in A$, $x \neq y$, $f(x) \neq f(y)$

Sunt acestea Injecții ?



NU



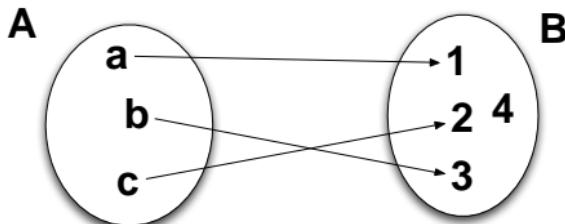
DA

Observați că în general dacă A și B sunt seturi finite și $f : A \rightarrow B$ este injectivă atunci $|A| \leq |B|$

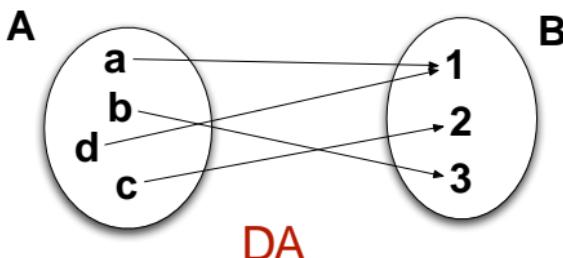
Proprietăți - Funcții Surjective

O funcție $f : A \rightarrow B$ este **surjectivă** dacă orice element $b \in B$ poate fi scris ca $b = f(x)$ pentru un $x \in A$ (adică $\text{range}(f) = B$)

Sunt acestea Surjectii ?



NU



DA

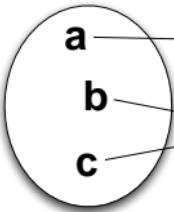
Observați că în general dacă A și B sunt seturi finite și $f : A \rightarrow B$ este surjectivă atunci $|A| \geq |B|$

Proprietăți - Funcții Bijective

O funcție $f : A \rightarrow B$ este **bijectivă** dacă
este injectivă și surjectivă

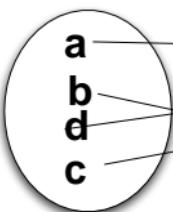
Sunt acestea Bijectii ?

A



NU

B A



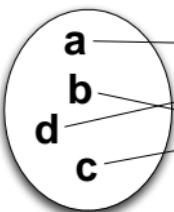
DA

Observați că în general dacă A și B sunt finite și $f : A \rightarrow B$ este bijectivă atunci

$$|A| = |B|$$

(spune că A și B sunt seturi echipotente)

A



NU

Construirea Funcțiilor - Compunerea

Putem construi funcții noi combinând funcții simple. **Compunerea** este o metodă de combinare.

Construirea Funcțiilor - Compunerea

Fie $f : A \rightarrow B$ și $g : B \rightarrow C$

Compunerea este o funcție $g \circ f : A \rightarrow C$
 $(g \circ f)(x) = g(f(x))$

Informal, aplicăm f lui x și apoi aplicăm g rezultatului anterior

Construirea Funcțiilor - Compunerea

Cmpunerea este **associativă**

$$(f \circ g) \circ h = f \circ (g \circ h)$$

$$((f \circ g) \circ h)(x) = (f \circ g)(h(x)) = f(g(h(x)))$$

$$(f \circ (g \circ h))(x) = f((g \circ h)(x)) = f(g(h(x)))$$

Construirea Funcțiilor - Compunerea

În general, compunerea **nu e comutativă**

$$f : N \rightarrow N, f(x) = x + 1 \quad g : N \rightarrow N, g(x) = x^2$$

$$(f \circ g)(x) = f(g(x)) = f(x^2) = x^2 + 1$$

$$(g \circ f)(x) = g(f(x)) = g(x+1) = (x+1)^2$$

Construirea Funcțiilor - Componerea

Functia identitate (id) întoarce tot timpul
valoarea argumentului său

$$id_N : N \rightarrow N, id_N(x) = x \quad f : N \rightarrow N$$

$$f \circ id_N = f = id_N \circ f$$

Exemplu de Componere

Utilizând doar succ : N→N, succ(x) = x + 1 construiți o funcție g : N→N care adună 2 la valoarea argumentului ei.

$$\begin{aligned}g(x) &= x + 2 \\&= (x + 1) + 1 \\&= \text{succ}(x) + 1 \\&= \text{succ}(\text{succ}(x)) \\&= (\text{succ} \circ \text{succ})(x)\end{aligned}$$

$$g = \text{SUCC} \circ \text{SUCC}$$

Construirea Funcțiilor Recursive

O funcție este definită **recursiv** dacă ea este definită în termenii ei însăși

Cu alte cuvinte, f este definită recursiv dacă cel puțin o valoare $f(x)$ este definită în termenii altelor valori $f(y)$ unde $x \neq y$

Construirea Funcțiilor Recурсиве

Multe funcții utile au ca domeniu multimi definite inductiv

Amintișivă că, o definire inductivă a unui set constă din:

Baza : Listarea unor elemente specifice din S (cel puțin unul)

Inducția : Dăm cel puțin o regulă de construcție de noi elemente din S pe baza unor elemente deja existente în S

Construirea Funcțiilor Recursive

Pentru aceste cazuri putem construi funcții recursive în modul următor:

Dacă S este un set inductiv, putem construi un f cu domeniul S astfel:

Baza : Pentru fiecare element de bază $x \in S$, specificăm o valoare pentru $f(x)$

Inducția : Dăm una sau mai multe reguli care, pentru orice element $x \in S$ definit inductiv, va defini $f(x)$ în termenii unei valori de-a lui f definite anterior

Exemplul I

Setul numerelor naturale $N = \{0, 1, 2, 3, 4, 5, \dots\}$ este inductiv

Baza: $0 \in N$

Inducția: if $n \in N$ then $n + 1 \in N$

Fie $f : N \rightarrow N$, $f(n) = 1 + 3 + \dots + (2n+1)$

O definire recursivă a lui f **Baza:** $f(0) = 1$

Inducția: $f(n+1) = 1 + 3 + \dots + (2n+1) + (2(n+1)+1)$
 $= 1 + 3 + \dots + (2n+1) + 2n+3$
 $= f(n) + 2n + 3$

Forma de definire Pattern Matching

Evaluarea lui $f(x)$ se realizează potrivind-o cu $f(0)$ sau
 $f(n+1)$ ex. $f(3)$ se potrivește cu $f(n+1)$ cu $n = 2$

Exemplul I

Baza: $f(0) = 1$

Inducția: $f(n) = f(n-1) + 2*n + 1$ if $n > 0$

Forma Condițională

$f(n) = \text{if } n = 0 \text{ then } 1 \text{ else } f(n-1) + 2*n + 1$

Forma Condițională cu Reguli if-then-else

Evaluarea Funcțiilor Recursive - Unfolding (sau “desfășurarea”)

$f(n) = \text{if } n = 0 \text{ then } 1 \text{ else } f(n-1) + 2*n + 1$

Se prelucreaza după definiție toate expresiile de forma $f(x)$
până când nu mai avem nici una

$$\begin{aligned}f(4) &= f(3) + 2 * 4 + 1 \\&= f(2) + 2 * 3 + 1 + 2 * 4 + 1 \\&= f(1) + 2 * 2 + 1 + 2 * 3 + 1 + 2 * 4 + 1 \\&= f(0) + 2 * 1 + 1 + 2 * 2 + 1 + 2 * 3 + 1 + 2 * 4 + 1 \\&= 1 + 2 * 1 + 1 + 2 * 2 + 1 + 2 * 3 + 1 + 2 * 4 + 1 \\&= 25\end{aligned}$$

Exemplul II

Setul numerelor naturale $N = \{0, 1, 2, 3, 4, 5, \dots\}$ este inductiv

Baza: $0 \in N$

Inducția: if $n \in N$ then $n + 1 \in N$

Definiți o funcție recursivă pentru $n! = 1 * 2 * 3 * \dots * n$ (factorial: $N \rightarrow N$)

Baza: $\text{factorial}(0) = 1$

Inducția: $\text{factorial}(1) = 1$,

$$\begin{aligned}\text{factorial}(n+1) &= 1 * 2 * \dots * n * (n+1) \\ &= \text{factorial}(n) * (n+1)\end{aligned}$$

Forma Pattern Matching

Baza: $\text{factorial}(0) = 1$ **Inducția:** $\text{factorial}(1) = 1$,

$\text{factorial}(n) = \text{factorial}(n-1) * n$ if $n > 1$

Forma Conditională

$\text{factorial}(n) = \text{if } n = 0 \text{ then } 1 \text{ else if } n = 1 \text{ then } 1 \text{ else } \text{factorial}(n-1) * n$

Forma Conditională cu Reguli if-then-else

Exemplul II

$\text{factorial}(n) = \text{if } n = 0 \text{ then } 1 \text{ else if } n = 1 \text{ then } 1 \text{ else } \text{factorial}(n-1) * n$

$\text{factorial}(4) = \text{factorial}(3) * 4$
= $\text{factorial}(2) * 3 * 4$
= $\text{factorial}(1) * 2 * 3 * 4$
= $1 * 2 * 3 * 4$
= 24

$\text{factorial}(n) = \text{if } n = 0 \text{ then } 1 \text{ else } \text{factorial}(n-1) * n$

O altă definire

$\text{factorial}(4) = \text{factorial}(3) * 4$
= $\text{factorial}(2) * 3 * 4$
= $\text{factorial}(1) * 2 * 3 * 4$
= $\text{factorial}(0) * 1 * 2 * 3 * 4$
= $1 * 1 * 2 * 3 * 4$
= 24

Exemplul III

Setul numerelor naturale $N = \{0, 1, 2, 3, 4, 5, \dots\}$ este inductiv

Baza: $0 \in N$

Inducția: if $n \in N$ then $n + 1 \in N$

Numerele Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, ... în care fiecare număr este obținut însumând anterioarele două numere; definiți o funcție recursivă care întoarce al n -lea număr Fibonacci

Basis: $\text{fib}(0) = 0$ Induction: $\text{fib}(1) = 1$,
 $\text{fib}(n+1) = \text{fib}(n) + \text{fib}(n-1)$

Forma Pattern Matching

Basis: $\text{fib}(0) = 0$,
Induction: $\text{fib}(1) = 1$
 $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$ if $n > 1$

Forma Conditională

$\text{fib}(n) = \text{if } n = 0 \text{ then } 0 \text{ else if } n = 1 \text{ then } 1 \text{ else } \text{fib}(n-1) + \text{fib}(n-2)$

Forma Conditională cu Reguli if-then-else

Construirea Procedurilor Definite Recursiv

Procedura = un subprogram care realizează anumite acțiuni și care poate deasemenea să întoarcă orice număr de valori (inclusiv nici una)

Exemple

print(x) - tipărește valoarea lui x pe dispozitivul de ieșire (1 acțiune)

sort(L) - acțiunea de sortare a elementelor listei L și întoarce lista sortată ca L

Se pot construi proceduri recursive pentru procesarea elementelor unui set inductiv într-un mod similar ca pentru funcții

Construirea Procedurilor Definite Recursiv

Dacă S este un set inductiv, putem construi o procedură P pentru procesarea elementelor lui S în modul următor:

Baza : Pentru fiecare element de bază $x \in S$, se specifică un set de acțiuni pentru $P(x)$

Inducție : Se dă una sau mai multe reguli care, pentru orice element definit inductiv $x \in S$, va defini acțiunile lui $P(x)$ în termenii unor acțiuni de-a lui P deja definite

Exemplu

Scriți o procedură care tipărește numerele naturale mai mici sau egale cu n

```
printFirst(n) : if n = 0 then print(0);  
else { printFirst(n-1); print(n); }
```

De ce studiem injecția / surjectia ?

Numărabil și Nenumărabil +
Poate un calculator să calculeze orice ?

Cardinalitatea Multimilor (Mărimea)

Pentru seturi finite e ușor de înțeles

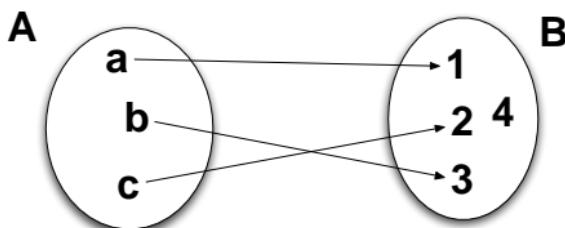
Exemplu

$$A = \{a, b, c\} \quad |A| = 3 \quad B = \{1, 2, 3, 4\} \quad |B| = 4$$

Dar dacă avem seturi infinite ?

Cardinalitatea Multimilor (Mărimea)

pentru seturi finite



Dacă există o
funcție
injectivă
 $f : A \rightarrow B$ atunci
 $|A| \leq |B|$

Similar se pot compara
cardinalitățile unor seturi infinite

Definiții (1)

În general, pentru două seturi A și B (**finite ori infinite**), dacă există o **injecție** $f : A \rightarrow B$ atunci cardinalitatea lui A este mai mică sau egală cu cardinalitatea lui B

$$|A| \leq |B|$$

Exemplu

$$\text{Par} = \{0, 2, 4, 6, \dots\}$$

$$\mathbb{N} = \{0, 1, 2, 3, 4, 5, \dots\}$$

$$f : \text{Par} \rightarrow \mathbb{N}, f(x) = x \text{ e injectivă deci } |\text{Even}| \leq |\mathbb{N}|$$

Definiții (2)

Când putem spune că $|A| = |B|$ (pentru seturi finite sau infinite) ?

Răspuns: când există o bijectie de la A la B

Când putem spune că $|A| < |B|$ (pentru seturi finite sau infinite) sau că B are mai multe elemente decât A ?

Răspuns: când există o injecție de la A la B și nu există nici o bijectie

Exemplu

Par = {0,2,4,6,...}

Impar = {1,3,5,7,...}

N = {0,1,2,3,4,5,...}

$g : \text{Par} \rightarrow \mathbb{N}$, $g(x) = x/2$ e o bijectie deci

$|\text{Par}| = |\mathbb{N}|$

$h : \text{Impar} \rightarrow \mathbb{N}$, $h(x) = (x - 1) / 2$ e o bijectie

deci $|\text{Impar}| = |\mathbb{N}|$

$k : \text{Impar} \rightarrow \text{Par}$, $k(x) = x - 1$ e o bijectie deci

$|\text{Impar}| = |\text{Par}|$

Numărabil și Nenumărabil

Un set C e numărabil dacă $|C| \leq |N|$

- Intuitiv, putem numărat elementele lui chiar dacă numărăm fără să ne oprim cândva
- dacă C e finit se spune că C e numărabil
- dacă $|C| = |N|$ se spune că C e infinit numărabil
- putem arăta că C e numărabil găsind o injecție $f : C \rightarrow N$ sau găsind o surjecție $g : N \rightarrow C$

Un set C care nu e numărabil se spune că e nenumărabil

Cum arătăm că un set e numărabil ?

Listând într-o formă toate elementele setului (arătând existența unei surjecții între \mathbb{N} și set)

Exemplu

\mathbb{Q}^+ (setul numerelor raționale pozitive)

$1/1$

(toate \mathbb{Q}^+ unde numitor+numărător=2)

$1/2, 2/1$

(toate \mathbb{Q}^+ unde numitor+numărător=2)

$1/3, 2/2, 3/1$

(toate \mathbb{Q}^+ unde numitor+numărător=2)

$1/4, 2/3, 3/2, 4/1$

(toate \mathbb{Q}^+ unde numitor+numărător=2)

...

...

Toate elementele sunt listate :)

Exercițiu

Dacă A este reuniunea unei colecții numărabile de seturi, și fiecare set e numărabil atunci A e numărabil

Din moment ce colecția (setul) de seturi e numărabil, putem lista toate seturile

s_0, s_1, s_2, \dots

$s_0 = s_{00}, s_{01}, s_{02}, \dots$ $s_1 = s_{10}, s_{11}, s_{12}, \dots$ $s_2 = s_{20}, s_{21}, s_{22}, \dots$

$s_3 = s_{30}, s_{31}, s_{32}, \dots$

⋮



Din moment ce fiecare set din colecție e numărabil, putem lista toate elementele lor

Deci avem o bijectie între N și elementele listate și deci A e numărabil

44

Problemă

Este setul tuturor programelor (AllProg), care pot fi scrise pe un computer, numărabil ori nenumărabil ?

Fie **A** setul tuturor caracterelor dintr-un limbaj de programare (ex. toate caracterele de la tastatură)

$A = \{a, b, c, d, \dots\}$ (acest set este finit)

Notă: Uzual A se numește alfabet

Fie **Pn** setul tuturor secvențelor de caractere din A de lungime n (aceste secvențe sunt de fapt programe)

Exemple $P_0 = \{\}$

$P_1 = \{a, b, c, d, \dots\}$

...

$P_{15} = \{\text{printf}(" \%d", x);, \dots\}$

Setul tuturor programelor dintr-un limbaj de programare e **infiinit numărabil**

Fiecare set P_i e finit și deci numărabil iar colecția tuturor P_i este countable infinite

AllProg (sau A^*) este reuniunea tuturor P_i și deci, după slideul anterior ...

Cum arătăm că două seturi au aceeași cardinalitate ?

Găsim o bijecție între seturi

- nu e tot timpul ușor

Dacă $|A| \leq |B|$ și $|B| \leq |A|$ atunci
 $|A| = |B|$

- găsim o injecție de la A la B
- găsim o injecție de la B la A

Problemă

$|(0,1)| = |\mathbb{R}|$?

Fie $f : (0,1) \rightarrow \mathbb{R}$, $f(x) = x$

E injectivă și deci $|(0,1)| \leq |\mathbb{R}|$

Fie $g : \mathbb{R} \rightarrow (0,1)$, $g(x) = 1/(2^x + 1)$

E injectivă și deci $|\mathbb{R}| \leq |(0,1)|$

Prin urmare,

$$|\mathbb{R}| = |(0,1)|$$

Cum arătăm că un set e nenumărabil ?

1. Găsim un alt set nenumărabil și evidențiem o bijecție între ele

2. Considerăm că setul e numărabil și găsim o contradicție

Problemă

E $(0,1)$ nenumărabil?

Presupunem că e numărabil. Deci, ar trebui să putem lista **toate** elementele din $(0,1)$ după cum urmează: (d_{ij} sunt cifrele de la 0 la 9)

$r_0 = 0, d_{00} d_{01} d_{02} \dots$ (această secvență e infinită)

$r_1 = 0, d_{10} d_{11} d_{12} \dots$

$r_2 = 0, d_{20} d_{21} d_{22} \dots$

$r_3 = 0, d_{30} d_{31} d_{32} \dots$

.

.

Să construim r_{weird} în modul următor: $r_{\text{weird}} = 0, s_0 s_1 s_2 \dots$ unde $s_i = \text{if } d_{ii} = 4 \text{ then } 5 \text{ else } 4$

ei bine $r_{\text{weird}} \in (0,1)$ dar el nu apare în listarea noastră

deoarece poziția i din r_{weird} (adică. s_i) diferă de poziția i din r_i (adică. d_{ii}) pentru orice i . Avem deci o contradicție.

Prin urmare,
 $(0,1)$ e nenumărabil
și R e nenumărabil
(deoarece $|(0,1)|=|R|$)
nenumărabil
(deoarece
 $|(0,1)|=|R|$)

Problemă

Acesta e un exemplu de utilizare a **tehnicii diagonalizării** ce poate fi utilizată pentru a construi un element care nu apare într-o listă.

...il ?

...umărabil. Deci, ar trebui să putem lista **toate** elementele din urmează: (d_{ij} sunt cifrele de la 0 la 9)

$d_{00} d_{01} d_{02} \dots$ (această secvență e infinită)

$r_1 = 0, d_{10} d_{11} d_{12} \dots$

$r_2 = 0, d_{20} d_{21} d_{22} \dots$

$r_3 = 0, d_{30} d_{31} d_{32} \dots$

.

.

Să construim r_{weird} în modul următor: $r_{\text{weird}} = 0, s_0 s_1 s_2 \dots$ unde $s_i = \text{if } d_{ii} = 4 \text{ then } 5 \text{ else } 4$

ei bine $r_{\text{weird}} \in (0, 1)$ dar el nu apare în listarea noastră

deoarece poziția i din r_{weird} (adică, s_i) diferă de poziția i din r_i (adică, d_{ii}) pentru orice i . Avem deci o contradicție.

Prin urmare,
 $(0,1)$ e nenumărabil
și R e nenumărabil
(deoarece $|(0,1)|=|R|$)
nenumărabil
(deoarece
 $|(0,1)|=|R|$)

Să revedem ce am găsit

1. Setul tuturor programelor dintr-un limbaj de programare e **infiit numărabil**

Există un număr numărabil de numere calculabile din R deoarece fiecare număr calculabil are nevoie de un program ce să-l calculeze !

2. dar, **R e nenumărabil**

Deci, există numere reale ce nu pot fi calculate !

1. Logica propozițiilor. Calculul predicatelor

Logica propozițiilor

Logica propozițiilor se bazează pe determinarea stării de adevărat sau fals ale conținutului acestora sau a modului de implicare (o afirmație are o implicație într-o altă afirmație), iar când avem mai multe propoziții se aplică axiomele pentru determinarea stării logice finale. Pentru acesta se trasează tabela de adevăr.

Se pune problema formalizării limbajului natural adică găsirii unui mod de substituție a limbajului natural cu un limbaj formal. Pentru o asemenea descriere vom folosi notații speciale pentru categoriile:

1. Constante referitoare la obiecte
2. Predicate referitoare la relația dintre obiecte
3. Argumente de tip predicat cu nume specifice (ex: studenții de la IDD sunt cu taxe=> cu taxe (studenți IDD))
4. Argumente de tip predicat pentru adjective (Leul este instabil => **instabil(leu)**)
5. Verbele sunt reprezentate ca predicate cu argumente care pot fi subiecte sau obiecte (ex: plouă =>plouă(); creșterea leului => crește(leu); A plătește lui B 100 lei =>**plateste(A,B,100)**)
6. Timpurile se introduc ca și argumente (A a plătit lui B 100 lei =>**plateste(A,B,100,trecut)**)
7. Articolele nedeterminate se reprezintă cu ajutorul cuantificatorilor existențiali “ \exists ”(există cel puțin un... pentru care propoziția este adevărată): *Un client a deschis un cont => $\exists(x,y):client(x) \wedge cont(y) \wedge deschis(x,y,trecut)$*
8. Analog se procedează cu expresii ca “este”, “există”, “sunt”: *Există un venit diferit de salariu sau pensie => $\exists(x):venit(x) \wedge este-un(x,y) \wedge diferit(y,salariu) \wedge diferit(y,pensie)$*
9. Expresii de genul “oricare”, “totul”, “fiecare” se reprezintă cu ajutorul cuantificatorului universal “ \forall ”(propoziție adevărată pentru oricare ..): *Orice persoană trebuie să plătească impozit => $\forall(x):persoana(x) \rightarrow trebuie-platit(x,impozit)$*
10. Implicațiile reprezintă expresii condiționale. IF este premisa sau condiția, iar THEN este acțiunea sau concluzia: *Dacă un om este fericit, el este prietenos => $\forall(x):fericit(x) \rightarrow prietenos(x)$*

O altă metodă de utilizare a logicii este inferarea, adică derivarea unor propoziții noi din propoziții existente.

P=Tribuna economică apare joi

Q=Astăzi este luni

R=Tribuna economică nu apare azi

Pentru a nu scrie întotdeauna toate propozițiile, acestea vor fi înlocuite cu litere.

Calculul propozițional are în vedere enunțurile declarative repartizate în două grupe: Adevărat și Fals. O propoziție compusă este formată din propoziții elementare legate cu conectorii \wedge (AND) \vee (OR), iar pentru negarea unei propoziții se folosește \sim (NOT) . Pentru a stabili valoarea de adevăr a unei propoziții vom folosi următorul tabel:

Tabela de adevăr pentru câteva propoziții logice

A	B	$\sim A$	$\sim B$	$A \wedge B$	$A \vee B$
A	A	F	F	A	A
A	F	F	A	F	A
F	A	A	F	F	A
F	F	A	A	F	F

Calculul predicatelor

Calculul predicatelor permite fragmentarea propozițiilor în obiecte (argumente) și predicate (aserții despre attributele obiectelor); calculul predicatelor permite, de asemenea, utilizarea variabilelor și funcțiilor de variabile. Aceste proprietăți fac din calculul predicatelor un instrument mai puternic decât calculul propozițional.

Trecerea de la propoziție la forma de calcul predicativ arată în felul următor:

“Creditează 101 cu 70.000.000” \Leftrightarrow creditează(101,70000.000) sau (creditează 101 70.000.000)

În exemplul de mai sus am utilizat valori concrete, dar la fel de bine se pot utiliza în descrierea formală a predicatelor variabile **cont-de-activ(x)**, deci putem scrie propoziții generalizate.

Cuantificatorii universal și existențial enumerați mai sus se pot folosi și la calculul predicatelor:

Ex: “Toți Popescu sunt cetăteni ai României” $\Leftrightarrow \forall(x), \text{Popescu}(x) \rightarrow \text{cetăean-român}(x)$
acesta se poate citi astfel Daca x este Popescu atunci x este român

$\forall(x), \text{cont}(x) \rightarrow \text{cont-de-activ}(x)$ AND $\text{cont}(5121) \Rightarrow \text{cont-de-activ}(5121)$

Contul 5121 este un cont de activ dacă 5121 este un cont și toate conturile sunt de activ

Exerciții:

1. Transformați următoarele predicate în propozițiiile corespunzătoare:

ESTE(INFLAȚIA MARE)

DEBITEAZĂ(X, CONT(5131))

2. Utilizați următoarele enunțuri pentru scrierea de formule în calculul predicatelor:

$P(x) = x$ este un economist

$S(x) = x$ este inteligent

$L(x,y) = x$ iubește y

- a) toți economiștii sunt inteligenți: $\forall(x):P(x) \rightarrow S(x)$
- b) unii economisti sunt inteligenți: $\exists(x):P(x) \rightarrow S(x)$
- c) nici un economist nu este intelligent: $\forall(x):P(x) \rightarrow \neg S(x)$
- d) unii sunt economisti: $\exists(x):P(x)$
- e) nici unul nu este economist: $\forall(x):\neg P(x)$
- f) unii economisti nu sunt inteligenți: $\exists(x):P(x) \rightarrow \neg S(x)$
- g) există economisti: $\exists(x):P(x)$
- h) unii iubesc pe alții $\exists(x) \exists(y):L(x,y)$
- i) toți îl iubesc pe economist: $\forall(x) \forall(y):L(y,P(x))$
- j) nimeni nu iubește economistul $\forall(x) \forall(y):\neg L(y,P(x))$
- k) unii economisti sunt iubiți $\exists(x) \forall(y) L(y,P(x))$
- l) toți iubesc economistii inteligenți $\forall(x) \exists(y):L(y,(P(x) \wedge S(x)))$

Notă: modul de scriere a formularelor poate varia în funcție de modul de interpretare.

2. Reguli de producție

Metoda regulilor de producție are la bază conceptul de reprezentare a cunoașterii sub forma unor reguli de formă: **DACĂ premisă ATUNCI concluzie1 ALTFEL concluzie2**. Atât premisele cât și concluziile sunt stabilite de expert. Premisa este, de obicei, o afirmație care este evaluată dacă este adevărat sau fals. Concluziile pot fi soluții-acțiuni. O concluzie poate deveni la rândul lui premisă într-o altă regulă. De asemenea, acest tip de reprezentare a cunoașterii permite folosirea variabilelor.

Regulile de producție pot fi clasificate după ordinea în care se merge în inferență (premise -> concluzie, sau concluzie -> premise) sau după tipul de entități conținute (constante sau variabile).

a) Reguli de producție deductive (RPD). Sunt de forma:

DACĂ
<premise>
ATUNCI
<concluzie>

În acest caz se utilizează implicația logică în sensul că dacă premisele sunt adevărate, atunci și concluzia este adevărată și aceasta se adaugă și ea bazei de fapte.

Ca urmare, se pot aplica alte reguli care conțin între premise și fapta nou adăugată.

b) Reguli de producție inductive (RPI). Sunt de forma:

<concluzie>
DACĂ
<premise>

Acste reguli declanșează deducții succesive, plecând de la concluzie și verificând fiecare premisă. Dacă premisele se găsesc a fi adevărate se conchide că și concluzia este adevărată.

Din punct de vedere logic, regulile deductive și cele inductive sunt formulări echivalente.

c) Reguli de producție cu variabile (RPV)

Utilizarea unor reguli care conțin doar constante este neeconomică datorită faptului că trebuie scrisă câte o regulă pentru fiecare caz în parte, chiar dacă sunt mai multe cazuri similare.

Ca și în cazul predicatorilor, folosirea variabilelor oferă o modalitate generală de exprimare.

Exemplu : Presupunem că sunt necesare reguli pentru pensionarea unui angajat de forma:

IF Ionescu V. este salariat

AND Ionescu V. a depășit vârsta de 65 ani

THEN Ionescu V. se poate pensiona

Astfel de reguli trebuie scrise pentru fiecare angajat, soluție greoaie și neeconomică. Se poate scrie însă o regulă cu variabile care să fie valabilă pentru toți angajații:

IF ?x este SALARIAT

AND ?x are vârsta > 65

THEN ?x se poate pensiona

În continuare vom reda o posibilă listă de reguli de producție care vor fi de folos la alegerea unui tip de imprimantă. Avem trei tipuri de imprimante:

1. cu ace (costul de întreținere redus, calitate rea, preț mediu) recomandat pentru contabilitate
2. cu jet (calitate bună, preț mic, costuri relativ mari) recomandat pentru birouri mici sau pentru casa cu număr redus de listări
3. cu laser (calitate foarte bună, costuri mici, preț foarte mare) recomandat birourilor, firmelor mari unde calitatea documentelor este importantă.

Vom face mai întâi un tabel în care vom include datele de mai sus:

Tip imprimantă	Ace	Jet	Laser
Proprietăți			
Preț	mediu	mic	mare
Costuri	F mic	mare	mic
Calitate	rea	buna	F bună

O regulă de producție ar arăta astfel:

DACĂ calitate=f_bună

AND costuri=mici

AND pret_achiz=mare

THEN Alege imprimanta laser

Putem constata că pentru a lua în considerare toate posibilitățile trebuie să scriem $3 \times 3 \times 3 = 27$ reguli de producție. Evident sunt combinații (ilogice) care nu ne duc la nici o soluție concretă (ex: preț-mare costuri-mari calitate-rea).

Exercițiu 1: scrieți celelalte reguli de producție logice.

DACĂ calitate=bună

AND costuri=mari
AND preț_achiz=mic
THEN Alege imprimanta jet

DACĂ calitate=rea
AND costuri=f_mic
AND preț_achiz=mediu
THEN Alege imprimanta ace

Exercițiu 2: Stabiliți criteriile și concepeți regulile de producție pentru un sistem expert care să ne asiste în alegerea unei anumite metode de investiții financiare (bancă, valută, acțiuni). Observație pentru a rezolva problema trebuie să utilizați posibilitatea utilizării variabilelor și valorilor numerice în scrierea regulilor de producție.

In acest exemplu vom urmări comisionul în cazul retragerii unei sume în euro și dobanda primită în cazul în care vom lasa banii într-un cont pentru o anumita perioadă de timp.

banca	BT	BCR	BRD
Proprietăți			
dobanda	medie	mica	mare
comision	F_mic	mare	mic
euro	mic	mediu	mare

Daca euro=mic	Daca euro=mediu	Daca euro=mare
And comision-=f_mic	And comision-=mare	And comision=mic
And dobanda=medie	And dobanda=mica	And dobanda=mare
Then Alege banca BT	Then Alege banca BCR	Then Alege banca BRD

- Find truth values for A , B , and C such that the wff $(A \square B) \square C$ is not equivalent to $A \square (B \square C)$.
- Use Quine's method to analyze the truth value of the following wff.
 $(A \exists B \square C) \exists (A \square B) \square (A \square C)$.

Solutions

- One answer is $A = \text{False}$, $C = \text{False}$ and B is either True or False.
- If W is the wff, then calculate $W(C/\text{True})$ and $W(C/\text{False})$ as follows:
 $W(C/\text{True}) = (A \exists B \square \text{True}) \exists (A \square B) \square (A \square \text{True})$
 $\alpha \text{ True} \exists (A \square B) \square \text{True} \alpha \text{ True}.$
 $W(C/\text{False}) = (A \exists B \square \text{False}) \exists (A \square B) \square (A \square \text{False})$
 $\alpha \neg (A \exists B) \exists (A \square B) \square \neg A.$
If $U = W(C/\text{False})$, then calculate $U(A/\text{True})$ and $U(A/\text{False})$ as follows:
 $U(A/\text{True}) \alpha \neg (\text{True} \exists B) \exists (\text{True} \square B) \square \neg \text{True}$
 $\alpha \neg B \exists B \square \text{False} \alpha \text{ False} \square \text{False} \alpha \text{ True}.$
 $U(A/\text{False}) \alpha \neg (\text{False} \exists B) \exists (\text{False} \square B) \square \neg \text{False}$
 $\alpha \neg \text{False} \exists \text{True} \square \text{True} \alpha \text{ True}.$
Since all expressions evaluate to True, it follows that W is a tautology.