

# 4

## STRUCTURI ARBORESCENTE

În clasa grafurilor conexe, structurile cele mai simple, dar care apar cel mai frecvent în aplicații, sunt cele arborescente (arbori). În acest capitol sunt prezentate principalele caracteristici ale arborilor, algoritmi pentru calculul arborelui parțial de cost minim, arbori direcționați, arbori cu rădăcină și arbori binari. Pe lângă operațiile primitive asupra arborilor – căutarea unei informații, inserarea unui nod, extragerea unui nod și metode de parcurgere, sunt prezentate două clase importante de arbori binari, și anume arbori de sortare și arbori de structură.

### 4.1 Grafuri de tip arbore

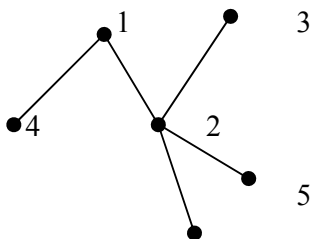
#### 4.1.1 Definiții și caracterizări ale grafurilor de tip arbore

*Definiția 4.1.1.* Graful  $G$  este *arbore* dacă  $G$  este aciclic și conex.

*Definiția 4.1.2.* Fie  $G=(V,E)$  graf arbore. Subgraful  $H=(V_1,E_1)$  al lui  $G$  este un subarbore al lui  $G$  dacă  $H$  este graf arbore.

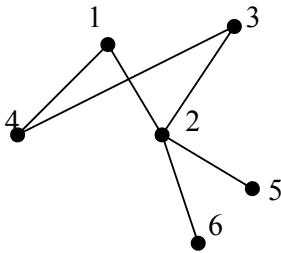
**Exemple:**

4.1. Graful



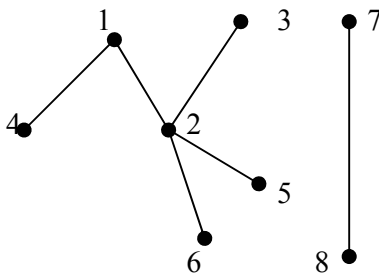
este arbore, deoarece pentru orice pereche de vârfuri  $i,j$ ,  $1 \leq i,j \leq 6$ ,  $i \neq j$ , există un  $i$ - $j$  drum și graful nu conține cicluri.

#### 4.2. Graful



nu este arbore deoarece este conex, dar drumul  $\Gamma : 1,4,3,2,1$  este un ciclu.

#### 4.3. Graful



nu este arbore deoarece are două componente conexe  $\{1,2,3,4,5,6\}$ ,  $\{7,8\}$ .

Proprietatea unui graf de a fi arbore poate fi verificată prin algoritmi care testează calitățile de conexitate și aciclicitate. Verificarea proprietății unui graf de a fi arbore poate fi realizată și pe baza următoarelor proprietăți:

*Proprietatea 1.:* Un graf  $G=(V,E)$ , cu  $|V| = n$ ,  $|E| = m$  este de tip arbore dacă și numai dacă  $G$  este aciclic și  $n=m+1$ . Cu alte cuvinte, problema revine la verificarea aciclicității grafului și a relației existente între numărul vârfurilor și numărul muchiilor grafului.

*Proprietatea 2:* Un graf  $G=(V,E)$ , cu  $|V| = n$ ,  $|E| = m$  este de tip arbore dacă și numai dacă  $G$  este conex și  $n=m+1$ .

Fie  $G=(V,E)$  un graf. Următoarele afirmații sunt echivalente:

1.  $G$  este graf arbore;
2.  $G$  este graf conex minimal (oriare ar fi  $e \in E$ , prin eliminarea muchiei  $e$  graful rezultat nu este conex);

3.  $G$  este graf aciclic maximal (prin adăugarea unei noi muchii în graf rezultă cel puțin un ciclu).

**Definiția 4.1.3.** Un graf orientat  $D=(V,E)$ , cu proprietatea că pentru  $\forall u,v \in E$ ,  $u,v \in E$ , atunci  $vu \notin E$  se numește *graf asimetric*. Digraful  $D$  este *simetric* dacă  $u,v \in E$ ,  $uv \in E$ , dacă și numai dacă  $vu \in E$ .

**Definiția 4.1.4.** Fie  $D=(V,E)$  digraf netrivial. Graful  $G=(V,E')$ , unde  $E'=\{uv/ uv \in E \text{ sau } vu \in E\}$  se numește *suport* al digrafului  $D$ .

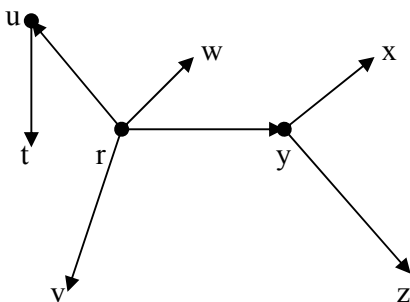
**Definiția 4.1.5.** Un *arbore direcționat* este un graf orientat asimetric cu proprietatea că graful suport corespunzător lui este graf arbore. Arborele direcționat  $T=(V,E)$  este cu rădăcină dacă există  $r \in V$  astfel încât, pentru orice  $u \in V$ ,  $u \neq r$ , există  $r$ - $u$  drum în  $T$ . Vârful  $r$  se numește *rădăcina* arborelui direcționat  $T$ .

**Definiția 4.1.6.** Dacă  $T=(V,E)$  este arbore direcționat, atunci  $T_1=(V_1,E_1)$  este subarbore al lui  $T$  dacă  $V_1 \subseteq V$ ,  $E_1 \subseteq E$  și  $T_1$  este arbore direcționat.

Deoarece graful suport al unui arbore direcționat este aciclic, rezultă că pentru orice  $u \in V$ ,  $u \neq r$ ,  $r$ - $u$  drumul în  $T$  este unic. De asemenea, un arbore direcționat are cel mult o rădăcină. În consecință, pentru orice  $u \in V$ ,  $u \neq r$ , distanța de la rădăcină la vârful  $u$  este egală cu numărul de muchii ale  $r$ - $u$  drumului în  $T$ .

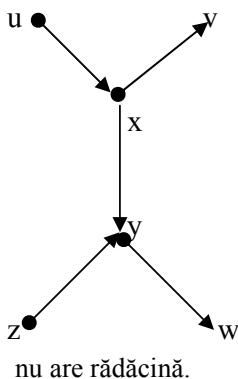
### Exemple:

#### 4.4. Arborele direcționat

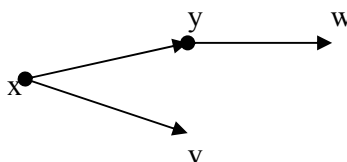


este cu rădăcină (vârful  $r$  este rădăcina arborelui).

#### 4.5. Arborele direcționat



#### 4.6. Arborele



este un subarbore cu rădăcină  $x$  al arborelui din exemplul 4.5

### 4.1.2 Arbori orientați; reprezentări și parcurgeri

*Definiția 4.1.7.* Un *arbore orientat* este un arbore direcționat cu rădăcină.

Deoarece un arbore orientat este un caz particular de digraf, pentru reprezentarea unui arbore orientat poate fi utilizată oricare din modalitățile prezentate în §3.1. În plus, există și posibilitatea obținerii unor reprezentări mai eficiente pentru acest tip de graf.

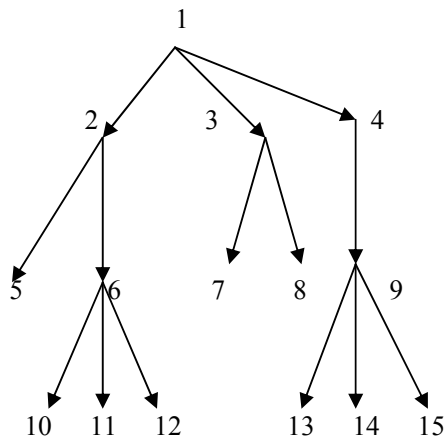
Una dintre modalități este reprezentarea FIU-FRATE, care constă în numerotarea convențională a vârfurilor grafului și reținerea, pentru fiecare vârf  $i$  al arborelui, a următoarelor informații:

- FIU( $i$ ), care reprezintă numărul atașat primului descendent al vârfului  $i$ ;
- FRATE( $i$ ), care reprezintă numărul atașat vârfului descendent al tatălui vârfului  $i$  și care urmează imediat lui  $i$ ;
- INF( $i$ ), care reprezintă informația atașată vârfului  $i$  (de obicei valoarea  $i$ ).

Pentru reprezentarea arborelui se rețin rădăcina și numărul nodurilor. Absența “fiului”, respectiv a “fratelui” unui vârf, este marcată printr-o valoare diferită de numerele atașate vârfurilor (de obicei valoarea 0).

**Exemplu:**

## 4.7. Arborele orientat



este reprezentat astfel:

$N=15$  (numărul nodurilor arborelui)

$R=1$  (rădăcina),  $FIU=(2,5,7,9,0,10,0,0,13,0,0,0,0,0,0)$

“fiul” lui 1 este 2  
 $FRATE=(0,3,4,0,6,0,8,0,0,11,12,0,14,15,0)$

vârful 1 nu are frate

vârful 14 are fratele 15

Utilizând structurile de date dinamice pentru arbori orientați, se obține o reprezentare descrisă în continuare. Presupunând că fiecare vârf al arborelui are cel mult  $n$  descendenți, fiecărui vârf îi este atașată structura:

identificatorul vârfului	vector de legături către descendenții vârfului		
	legătură către fiul 1	.....	legătură către fiul n

Dacă un vârf are  $p < n$  descendenți, atunci primele  $p$  legături sunt către descendenți, iar ultimele  $n-p$  legături sunt *nil*. Pentru  $n \leq 10$ , descrierea structurii de date în limbajul Pascal este:

```

type ptnod=^nod;
   nod=record
       inf:integer;
       leg:array[1..10] of ptnod;
   end;

```

O parcurgere revine la aplicarea sistematică a unei reguli de vizitare a vârfurilor grafului. Cele mai uzuale reguli de parcurgere a arborilor orientați sunt prezentate în continuare.

### **A. Parcurgerea în A-preordine**

Inițial vârful curent este rădăcina arborelui. Se vizitează vârful curent și sunt identificați descendenții lui. Se aplică aceeași regulă de vizitare pentru arborii care au ca rădăcini descendenții vârfului curent, arborii fiind vizitați în ordinea dată de numerele atașate vârfurilor rădăcină corespunzătoare.

#### **Exemplu:**

4.8. Pentru arborele orientat din exemplul 4.7, prin aplicarea parcurgerii în A-preordine, rezultă: 1,2,5,6,10,11,12,3,7,8,4,9,13,14,15.

Pentru un arbore reprezentat FIU-FRATE, implementarea parcurgerii în A-preordine se bazează pe următoarea procedură recursivă, având ca parametru de intrare rădăcina arborelui curent (vârful curent în momentul apelului).

```
procedure A_preordine (R);  
if R≠0 then  
  vizit (R);  
  A_preordine(FIU[R]);  
  A_preordine(FRATE[R]);  
endif;  
end;
```

### **B. Parcurgerea A-postordine**

Regula de vizitare a vârfurilor în parcurgerea în A-postordine diferă de cea în A-preordine numai prin faptul că rădăcina fiecărui arbore este vizitată după ce au fost vizitate toate celelalte vârfuri ale arborelui.

#### **Exemplu:**

4.9. Pentru arborele orientat din exemplul 4.7, ordinea de vizitare a vârfurilor este: 5,10,11,12,6,2,7,8,3,13,14,15,9,4,1.

Pentru arbori reprezentați prin structuri de date arborescente, implementarea parcurgerii în A-postordine poate fi obținută pe baza următoarei proceduri recursive. Unicul parametru (de intrare) reprezintă rădăcina arborelui curent în momentul apelului.

```
procedure A_postordine (R);  
if R≠nil then  
  do-for i=1,n,1  
    A_postordine(R^.leg[i]);
```

```

    enddo;
    vizit (R);
  endif;
end;

```

Procedurile A-preordine și A-postordine sunt variante de parcurgeri în adâncime, fiind prioritare vârfurile aflate la distanță maximă față de rădăcina arborelui inițial.

### C. Parcurgerea pe niveluri

*Definiția 4.1.8.* Un vârf  $v$  al unui arbore orientat cu rădăcină  $r$  se află pe *nivelul*  $i$  al arborelui, dacă distanța de la vârf la rădăcină (lungimea  $r$ - $v$  drumului) este egală cu  $i$ . Rădăcina arborelui este de nivel 0.

Parcurgerea unui arbore orientat pe niveluri constă în vizitarea vârfurilor sale în ordinea crescătoare a distanțelor față de rădăcină.

#### Exemplu:

4.10. Pentru arborele din exemplul 4.7, prin aplicarea parcurgerii pe niveluri, rezultă: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15.

Implementarea parcurgerii pe niveluri se bazează pe utilizarea unei structuri de coadă  $C$ . La momentul inițial, rădăcina arborelui este unicul element din  $C$ . Atâta timp cât coada este nevidă, se extrage (cu ștergere) un vârf din  $C$ , este vizitat și sunt introduși în coadă descendenții săi. Calculul se încheie în momentul în care, la tentativa de extragere a unui vârf din  $C$ , se constată  $C=\emptyset$ .

Parcurgerea pe niveluri este realizată de următoarea procedură care are ca parametri de intrare reprezentarea FIU-FRATE a grafului. Procedurile *push* și *pop* realizează operațiile de acces introducere-extragere în  $C$ .

```

procedure parcurgere_pe_niveluri(R,FIU,FRATE,n)
C:ptcoada;
C=nil;push(C,R);
while C≠nil do
  pop(C,v);
  VIZIT(v);
  v=FIU[v];
  while v≠0 do
    push(C,v);
    v=FRATE[v];
  endwhile;
endwhile;
end;

```

## Exemplu

4.11. Pentru arborele de la exemplul 4.7, evoluția algoritmului este:

$\begin{smallmatrix} \diagdown \\ t \end{smallmatrix} \begin{smallmatrix} \diagup \\ C \end{smallmatrix}$						
t=1	1					
t=2	2	3	4			
t=3	3	4	5	6		
t=4	4	5	6	7	8	
t=5	5	6	7	8	9	
t=6	6	7	8	9		
t=7	7	8	9	10	11	12
t=8	8	9	10	11	12	
t=9	9	10	11	12		
t=10	10	11	12	13	14	15
t=11	11	12	13	14	15	
t=12	12	13	14	15		
t=13	13	14	15			
t=14	14	15				
t=15	15					
t=16						

deci vârfurile sunt vizitate în ordinea: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15.

Metoda BF pentru parcurgerea grafurilor este o generalizare a tehnicii de parcurgere pe niveluri a arborilor orientați. O alternativă de implementare a parcurgerii pe niveluri poate fi descrisă prin intermediul procedurilor recursive *frați* și *parc*. Coada *C* este o variabilă globală și este inițializată cu rădăcina arborelui. Parcurgerea este obținută prin apelul *parc(C)*.

```

procedure frați(v);
  if v≠0 then
    push(C,v);frați(FRATE[v]);
  endif;
end;
```

```

procedure parc;
  if C≠nil then
    pop(C,v);VIZIT(v);
    frați(FIU[v]); parc;
  endif;
end;
```



### 4.1.3 Arbori parțiali; algoritmul Kruskal

*Definiția 4.1.9.* Fie  $G$  graf. Subgraful parțial  $H$  este un arbore parțial al lui  $G$  dacă  $H$  este graf arbore.

*Definiția 4.1.10.* Fie  $(V, E, w)$  un graf ponderat conex. Dacă  $T=(V, E_0)$  este un arbore parțial al grafului  $G=(V, E)$ , ponderea arborelui  $T$  este definită prin:  
$$W(T) = \sum_{e \in E_0} w(e).$$

*Definiția 4.1.11.* Fie  $T(G)$  mulțimea arborilor parțiali corespunzători grafului  $G$ .  $T_0 \in T(G)$  este arbore parțial minim pentru  $G$  dacă  $W(T_0) = \min \{W(T); T \in T(G)\}$ .

Dacă  $G$  este graf finit, atunci  $T(G)$  este mulțime finită, deci orice graf finit ponderat și conex are cel puțin un arbore parțial minim.

Pentru calculul unui arbore parțial minim sunt cunoscuți mai mulți algoritmi. În continuare este prezentat algoritmul Kruskal pentru determinarea unui arbore parțial minim al unui graf ponderat conex  $G=(V, E, w)$ .

*Pasul 1:*  $i=1$ ;  $E_0=\emptyset$

*Pasul 2:* Determină mulțimea

$R=\{e/e \in E \setminus E_{i-1} \text{ astfel încât graful } (V, E_{i-1} \cup \{e\}) \text{ este aciclic}\}$

Dacă  $R=\emptyset$ , atunci stop;

altfel, selectează  $e_i \in R$  cu  $w(e_i) = \min \{w(e), e \in R\}$ ;

$E_i = E_{i-1} \cup \{e_i\}$

*Pasul 3:*  $i=i+1$  și reia pasul 2.

Structura  $(V, E_{i-1})$  calculată de procedură este arbore parțial minim al grafului conex ponderat  $G$ .

Ideea algoritmului Kruskal revine la alegerea și includerea în mulțimea de muchii curente a unei muchii de cost minim încă neselectate și astfel încât să nu formeze un ciclu cu muchiile selectate la etapele precedente. Algoritmul se încheie atunci când nici o alegere nu mai este posibilă. Aplicarea metodei la grafuri neconexe calculează o mulțime de arbori parțiali minimi, câte un arbore pentru fiecare componentă conexă.

Pentru implementarea algoritmului Kruskal, graful conex ponderat este reprezentat sub formă tabelară, muchiile fiind ordonate crescător după ponderi. Muchiile selectate de algoritm pot fi menținute, de asemenea, într-o structură tabelară, sau doar marcate ca fiind incluse în mulțimea muchiilor arborelui parțial minim a cărui construcție este dorită. În varianta prezentată în continuare muchiile selectate sunt afișate.

Pentru verificarea condiției ca muchia selectată să nu formeze nici un ciclu cu muchiile selectate la etapele precedente, este utilizat un vector, TATA. Pentru fiecare vârf  $i$  (vârfurile grafului fiind numerotate de la 1 la  $n$ , unde  $n$  este numărul nodurilor grafului), componenta TATA  $[i]$  este predecesorul său în arborele care conține vârfurile  $i$  construit până la momentul curent, dacă  $i$  nu este rădăcina acelui

arbore, respectiv  $TATA[i]$  este egal cu *–numărul de vârfuri ale arborelui de rădăcină  $i$* , în caz contrar. Componentele vectorului TATA sunt inițializate cu valoarea -1. Calculul care realizează adăugarea unei noi muchii poate fi descris astfel:

- este determinată o muchie de cost minim,  $e=v_1v_2$ , care nu a fost selectată anterior;
- se verifică proprietatea de aciclicitate a grafului care rezultă prin eventuala adăugare a muchiei selectate astfel: dacă vârfurile  $v_1$  și  $v_2$  nu aparțin aceluiași arbore, atunci proprietatea de aciclicitate este îndeplinită și muchia  $e$  este adăugată la structura curentă. Determinarea valorii  $k$  reprezentând rădăcina arborelui care conține vârful dat  $v$  rezultă prin parcurgerea vectorului TATA:

```
k=v;
while TATA[k] >0 do k=TATA[k]
endwhile;
```

- adăugarea muchiei  $e$  selectate este realizată prin reunirea arborilor cu rădăcini  $r_1$  și  $r_2$ , din care fac parte  $v_1$  și respectiv  $v_2$ , astfel: dacă  $TATA[r_1] < TATA[r_2]$  (arborele de rădăcină  $r_1$  conține mai multe vârfuri decât arborele de rădăcină  $r_2$ ), atunci arborele rezultat prin reunirea celor doi arbori are ca rădăcină vârful  $r_1$ , iar vârful  $r_2$  devine fiu al lui  $r_1$  ( $TATA[r_2]=r_1$  și  $TATA[r_1] = TATA[r_2] + TATA[r_1]$  (numărul muchiilor arborelui cu rădăcină  $r_1$  este crescut cu numărul muchiilor arborelui cu rădăcină  $r_2$ ). În caz contrar se procedează analog, rădăcina arborelui rezultat prin reunire fiind  $r_2$ , iar  $r_1$  devenind fiu al rădăcinii.

Calculul se încheie după ce a fost adăugată cea de-a  $(n-1)$ -a muchie.

## 4.2 Arbori binari

### 4.2.1 Reprezentare; parcurgeri

*Definiția 4.2.1.* Un arbore binar este un arbore orientat cu proprietatea că pentru orice vârf  $v$ ,  $od(v) \leq 2$ . În cazul  $od(v)=2$ , cei doi descendenți sunt desemnați ca descendent stâng (fiu stânga) respectiv descendent drept (fiu dreapta). Pentru vârfurile cu  $od(v)=1$ , unicul descendent este specificat fie ca fiu stânga, fie ca fiu dreapta.

*Definiția 4.2.2.* Se numește *nod terminal* orice vârf  $v$  al arborelui cu  $od(v)=0$ . Nodul  $v$  este *neterminal* dacă  $od(v)>0$ .

Reprezentarea unui arbore binar este realizată prin reținerea, pentru fiecare nod, a legăturilor către descendenții lui. Absența unui descendent este reprezentată prin nil.

identificator nod	legătură fiu stânga	legătură fiu dreapta
----------------------	------------------------	-------------------------

Structura de date Pascal este:

```
type arb = ^nod;  
  nod = record;  
    inf: integer;  
    fiu, fiud: arb;  
  end;
```

*Definiția 4.2.3.* Fie  $T=(V,E)$  un arbore binar cu rădăcina  $R$ . *Subarboarele stâng* al lui  $T$  este  $ST=(V\setminus\{R\}, E\setminus\{RS\})$ , unde  $S$  este fiul stânga al rădăcinii. *Subarboarele drept* al lui  $T$  este  $DT=(V\setminus\{R\}, E\setminus\{RD\})$ , unde  $D$  este fiul dreapta al rădăcinii.

În plus față de metodele deja prezentate pentru parcurgerea arborilor generali și care sunt aplicabile și în acest caz particular, parcurgerile în *preordine*(RSD), *inordine*(SRD) și respectiv *postordine*(SDR) sunt special considerate pentru arbori binari și au multiple aplicații. Regula de vizitare revine la parcurgerea subarboarelui stâng, a subarboarelui drept corespunzători vârfului curent. La momentul inițial, vârful curent este rădăcina arborelui. Diferența între cele trei tipuri de parcurgere este dată de momentul în care este vizitat fiecare vârf al arborelui. În parcurgerea RSD (rădăcină-subarboare stâng-subarboare drept), fiecare vârf al arborelui este vizitat în momentul în care devine vârf curent; în parcurgerea SRD (subarboare stâng-rădăcină-subarboare drept), vizitarea vârfului este efectuată după ce a fost parcurs subarboarele stâng; în parcurgerea SDR (subarboare stâng-subarboare drept-rădăcină) vizitarea fiecărui vârf este efectuată după ce au fost parcurși subarborii aferenți lui.

Procedura *preordine* realizează parcurgerea în preordine, metodele SRD și SDR fiind propuse cititorului ca exerciții. Procedura *preordine* este recursivă și are ca parametru de intrare vârful curent în momentul apelului.

```
procedure preordine(r:arb);  
begin  
  if r<>nil then  
    begin  
      write(r^.inf);  
      preordine(r^.fiu);  
      preordine(r^.fiud);  
    end;  
end;
```

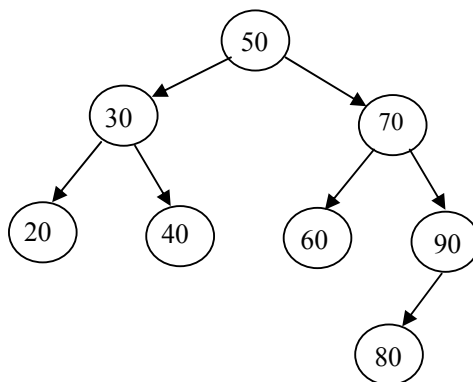
## 4.2.2 Arbori binari de sortare

*Definiția 4.2.4.* Un arbore de sortare este un arbore binar cu proprietățile:

- ⇒ fiecărui nod  $i$  al arborelui îi este atașată o informație  $INF(i)$  dintr-o mulțime ordonată de valori;
- ⇒ pentru fiecare nod  $i$ ,  $INF(i)$  este mai mare decât  $INF(j)$ , pentru toate nodurile  $j$  din subarboarele stâng al arborelui cu rădăcină  $i$ ;
- ⇒ pentru fiecare nod  $i$ ,  $INF(i)$  este mai mică decât  $INF(j)$ , pentru toate nodurile  $j$  din subarboarele drept al arborelui cu rădăcină  $i$ ;
- ⇒ pentru orice vârfuri  $i, j$ , dacă  $i \neq j$ , atunci  $INF(i) \neq INF(j)$ .

### Exemplu:

4.12. Arborele binar



este de sortare.

Operațiile uzuale efectuate asupra arborilor de sortare sunt inserarea unui nod, ștergerea unui nod și parcurgerea arborelui (în preordine, inordine sau postordine). Inserarea și ștergerea nodurilor într-un arbore de sortare trebuie realizate astfel încât arborele rezultat să fie, de asemenea, arbore de sortare.

Parcurgerea în inordine a unui arbore de sortare determină secvența vârfurilor arborelui în ordinea crescătoare a informațiilor atașate.

### Exemplu:

4.13. Pentru arborele de sortare descris în exemplul 4.12, parcurgerea în inordine determină secvența de valori: 20,30,40,50,60,70,80,90, adică exact vectorul de informații asociate nodurilor ordonat crescător.

### Inserarea unui nod într-un arbore de sortare

Algoritmul de inserare a unei informații  $nr$  în arborele de sortare de rădăcină  $rad$  este recursiv și constă în efectuarea operațiilor. Vârful curent  $v$  la momentul inițial este rădăcina arborelui.

- 1. dacă arborele de rădăcină  $v$  este vid ( $v=nil$ ), este generat arborele cu un singur nod, având  $nr$  ca informație atașată;
- 2. altfel
  - a) dacă informația atașată nodului  $v$  este mai mare decât  $nr$ , atunci vârful curent devine fiul stânga al lui  $v$ ;
  - b) dacă informația atașată nodului  $v$  este egală cu  $nr$ , atunci stop (se previne duplicarea informațiilor atașate vârfurilor arborelui);
  - c) dacă informația atașată nodului  $v$  este mai mică decât  $nr$ , atunci vârful curent devine fiul dreapta al lui  $v$ .

#### Exemplu:

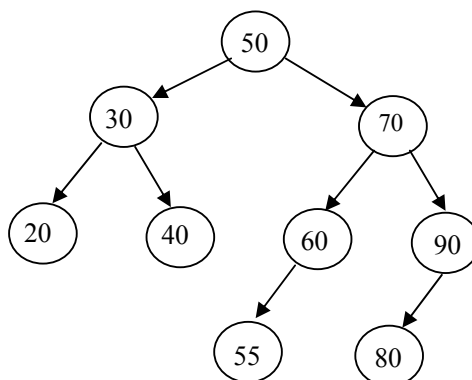
4.14. Aplicarea algoritmul descris pentru inserarea informației 55 în arborele de sortare din exemplul 4.12 determină următoarele operații:

$INF(v)=50<55$ : se decide inserarea în subarborele drept cu rădăcina având informația atașată 70 (cazul 2.c);

$INF(v)=70>55$ : se decide inserarea în subarborele stâng cu rădăcina având informația atașată 60 (cazul 2.a);

$INF(v)=60>55$ : se decide inserarea în subarborele stâng cu rădăcina nil (situația de la 1). Se decide crearea nodului cu informație 55, fiu stâng al nodului de informație 60.

Arborele rezultat este:



### Ștergerea unei informații dintr-un arbore de sortare

Algoritmul pentru ștergerea unei informații  $nr$  din arborele de sortare de rădăcină  $rad$  este recursiv este descris în continuare. Vârful curent  $v$  la momentul inițial este rădăcina arborelui.

- 1. dacă arborele este vid ( $v=nil$ ) atunci stop ( $nr$  nu se află în mulțimea informațiilor atașate nodurilor arborelui);
- 2. altfel
  - a) dacă informația atașată nodului  $v$  este mai mare decât  $nr$ , atunci vârful curent devine fiul stânga al lui  $v$ ;
  - b) dacă informația atașată nodului  $v$  este mai mică decât  $nr$ , vârful curent devine fiul dreapta al lui  $v$ ;
  - c) dacă  $INF(v)=nr$  atunci:
    - c1) dacă subarborele stâng este vid ( $v^.fius=nil$ ), atunci adresa vârfului  $v$  este memorată într-o celulă suplimentară  $aux$ ,  $v$  devine fiul dreapta al lui  $v$ , iar celula  $aux$  este eliberată din memorie (este disponibilizată celula corespunzătoare vârfului din arbore de informație  $nr$ );
    - c2) dacă subarborele stâng este nevid, atunci se determină cel mai mare element din subarborele stâng (este parcurs subarborele stâng pe legăturile din dreapta, cât timp acest lucru este posibil, cu păstrarea și a adresei nodului părinte corespunzător fiecărui vârf atins) :  
 $p:=v^.fius$ ;  
while  $p^.fiud \neq nil$  do  
begin  
   $p1:=p$ ;  
   $p:=p^.fiud$ ;  
end;  
c2.1) dacă fiul stânga al lui  $v$  nu are subarbore drept ( $v^.fius^.fiud=nil$ ), atunci informația atașată fiului stânga se transferă în vârful curent, iar fiul stânga ( $v^.fius$ ) este înlocuit cu fiul său stânga ( $v^.fius^.fius$ ) și este eliberată memoria corespunzătoare celulei  $v^.fius$ .  
c2.2) altfel, se transferă în rădăcină informația atașată ultimului nod  $p$  determinat la c2), nodul  $p$  este înlocuit cu fiul său stâng și celula corespunzătoare lui  $p$  este eliberată din memorie.

#### Exemple:

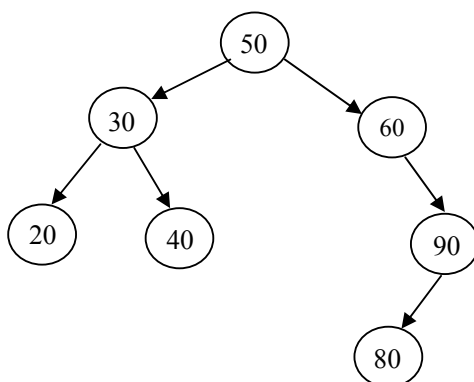
4.15. Ștergerea informației 70 în arborele de sortare din exemplul 4.12 este realizată astfel:

70>50, decide ștergerea din subarborele drept (cu rădăcină 70) - situația 2.b;

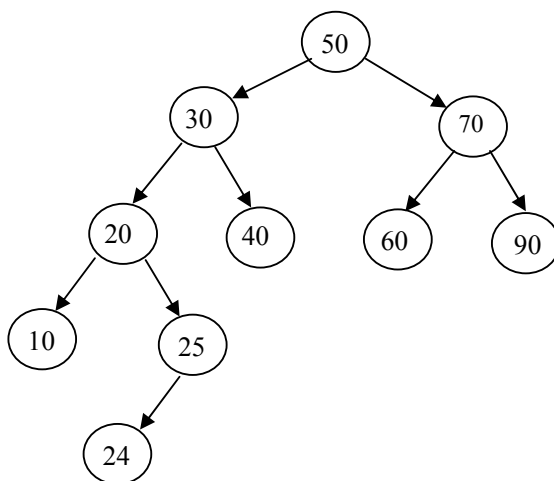
70=70, decide ștergerea din arborele curent: rădăcina etichetată cu 70 - situația 2.c;

Există subarbore stâng (rădăcina lui  $p$  este etichetată cu 60) iar acesta nu are subarbore drept - situația 2.c.1.: nodul cu informație 70 este etichetat cu 60, iar  $p$  este înlocuit cu subarboarele său stâng (vid)

Arborele de sortare rezultat este:



4.16. Ștergerea informației 30 din arborele de sortare:



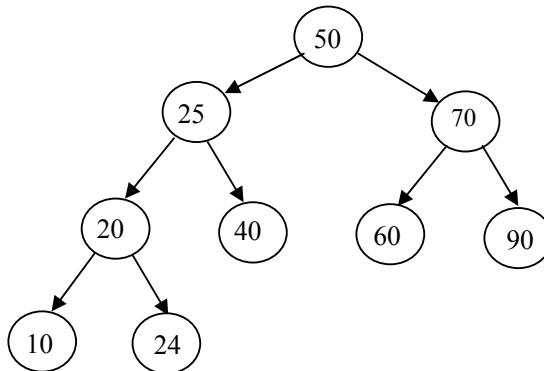
este realizată astfel:

$30 > 50$ , decide ștergerea din subarboarele stâng (cu rădăcină 30) - situația 2.a;

$30 = 30$ , decide ștergerea din arborele curent: rădăcina etichetată cu 70 – situația 2.c;

Există subarbore stâng (rădăcina este etichetată cu 20), iar acesta are subarbore drept - situația 2.c.2: nodul cu informație 30 este etichetă cu 25 (informația ultimului nod  $p$  detectat la c2)), iar  $p$  este înlocuit cu subarborele său stâng (cu rădăcină 24).

Arborele rezultat este:



Punctul c) de la pasul 2 poate fi înlocuit cu:

c) dacă INF( $v$ )=nr atunci:

- c1) dacă subarborele drept este vid ( $v^{\wedge}.fiud=nil$ ), atunci adresa vârfului  $v$  este memorată într-o celulă suplimentară  $aux$ ,  $v$  devine fiul stânga al lui  $v$ , iar celula  $aux$  este eliberată din memorie (este eliberat vârful de informație  $nr$ );

- c2) dacă subarborele drept este nevid, atunci se determină cel mai mic element din subarborele drept (este parcurs subarborele drept pe legăturile din stânga, cât timp acest lucru este posibil, cu păstrarea și a adresei nodului părinte corespunzător fiecărui vârf atins) :

```

p:=v^fiud;
while p^fius <> nil do
begin
p1:=p;p:=p^fius;
end;
    
```

c2.1.) dacă fiul dreapta al lui  $v$  nu are subarbore stâng ( $v^{\wedge}.fiud^{\wedge}.fius=nil$ ), atunci informația atașată fiului dreapta se transferă în vârful curent, iar fiul dreapta este înlocuit cu fiul său dreapta ( $v^{\wedge}.fiud^{\wedge}.fiud$ ) și este eliberată memoria corespunzătoare celulei  $v^{\wedge}.fiud$ .

c2.2) altfel, se transferă în rădăcină informația atașată ultimului nod  $p$  determinat la c2), nodul  $p$  este înlocuit cu fiul său dreapta și celula corespunzătoare lui  $p$  este eliberată din memorie.



### 4.2.3 Arbori de structură

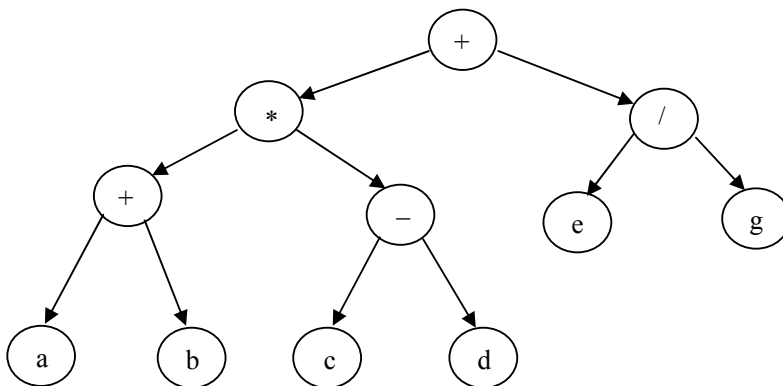
Expresiile aritmetice în care intervin numai operatori binari pot fi reprezentate prin intermediul arborilor strict binari (fiecare nod neterminal are doi fii). Un *arbore de structură* are vârfurile etichetate astfel:

- fiecare nod neterminal este etichetat cu un simbol corespunzător unuiu dintre operatori;
- fiecare nod terminal este etichetat cu un operand (variabilă sau constantă);

Construcția arborelui de structură corespunzător unei expresii aritmetice date se realizează pe baza “parantezării” existente în expresie și a priorităților convențional asociate operatorilor (ordinea operațiilor) astfel încât rădăcina fiecărui subarbore este etichetată cu operatorul care se execută ultimul în evaluarea subexpresiei corespunzătoare acelu subarbore.

#### Exemplu:

4.17. Pentru expresia matematică  $(a+b)*(c-d)+e/g$ , arborele de structură este:



Construcția arborelui de structură pentru o expresie  $s$  se face în două etape, și anume:

1. Atașarea priorităților operatorilor și operanzilor (toți operanzii au aceeași prioritate, egală cu prioritatea maximă). Prioritățile atașate permit eliminarea parantezelor fără ca semnificația expresiei să se modifice;
2. Construcția propriu-zisă.

Prima etapă este realizată astfel:

- prioritatea inițială a operatorilor '+', '-', '\*' este 1 (dacă expresia nu conține paranteze, atunci în construcție operatorii vor fi primii luați în considerare, în ordinea de la dreapta la stânga);
- prioritatea inițială a operatorilor '/', '\*' este 10 (dacă expresia nu conține paranteze, aceștia sunt considerați după operatorii de prioritate 1 în ordinea de la dreapta la stânga);
- prioritatea fiecărui operator este incrementată cu valoarea 10 pentru fiecare pereche de paranteze în interiorul cărora se află;
- prioritatea atașată fiecărui operand este *maxint*.

După stabilirea sistemului de priorități, se elimină parantezele din expresie (ordinea de efectuare a operațiilor în cadrul expresiei este indicată de vectorul de priorități atașat).

### Exemplu:

4.18. Etapele calculului sistemului de priorități pentru expresia de la exemplul 4.17 sunt:

i	j	dim	vectorul <i>prioritate</i>
1	10	0	
2	10	1	(maxint)
3	10	2	(maxint,11)
4	10	3	(maxint,11,maxint)
5	0	3	(maxint,11,maxint)
6	0	4	(maxint,11,maxint,10)
7	10	4	(maxint,11,maxint,10)
8	10	5	(maxint,11,maxint,10,maxint)
9	10	6	(maxint,11,maxint,10,maxint,11)
10	10	7	(maxint,11,maxint,10,maxint,11,maxint)
11	0	7	(maxint,11,maxint,10,maxint,11,maxint)
12	0	8	(maxint,11,maxint,10,maxint,11,maxint,1)
13	0	9	(maxint,11,maxint,10,maxint,11,maxint,1,maxint)
14	0	10	(maxint,11,maxint,10,maxint,11,maxint,1,maxint,10)
15	0	11	(maxint,11,maxint,10,maxint,11,maxint,1,maxint,10,maxint)

După eliminarea parantezelor, expresia rezultată este  $s=a+b*c-d+e/g$ , având ca vector de priorități (maxint,11,maxint,10,maxint,11,maxint,1,maxint,10,maxint).

Construcția arborelui de structură pe baza expresiei  $s$ , din care au fost eliminate parantezele și a vectorului de priorități, poate fi realizează recursiv în modul descris în continuare. La momentul inițial expresia curentă este cea dată.

- Pentru expresia curentă se determină operatorul/operandul de prioritate minimă care se atașează ca etichetă a rădăcinii  $r$  a subarborului de structură corespunzător ei; fie  $i$  poziția acestuia în cadrul expresiei;
- Dacă expresia are un singur simbol (operand) atunci  $r^{\wedge}.fius=r^{\wedge}.fiud=nil$ ;
- Altfel, se consideră subexpresiile  $s1$  și  $s2$ , constând din simbolurile de pe pozițiile 1 până la  $i-1$  și respectiv  $i+1$  până la  $length(s)$ . Arborii de structură corespunzători subexpresiilor  $s1$  și  $s2$  se atașează ca subarbor stâng, respectiv subarbor drept vârfului  $r$ .

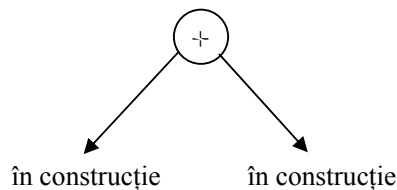
**Exemplu:**

4.19. Pentru expresia de la exemplul 4.17, după determinarea vectorului de priorități și a expresiei neparanțate corespunzătoare, procedura  $cr\_der$  realizează următoarea construcție:

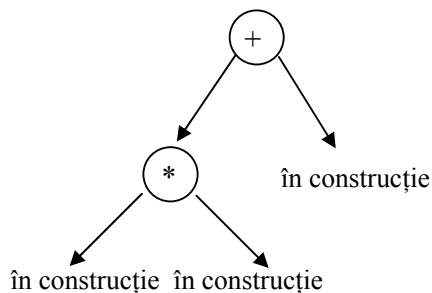
$s=a+b*c-d+e/g$ ,

$prioritate=(maxint,11,maxint,10,maxint,11,maxint,1,maxint,10,maxint)$

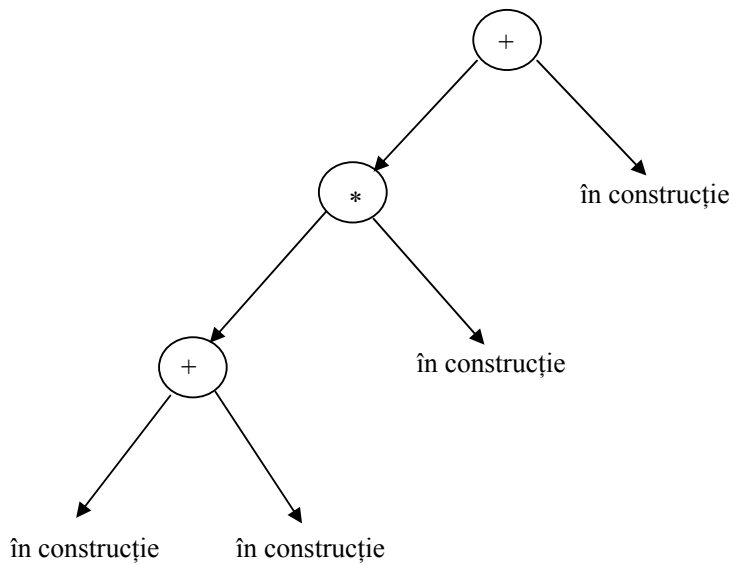
$p=1, u=11 \Rightarrow \min=1, i=8$ , arborele:



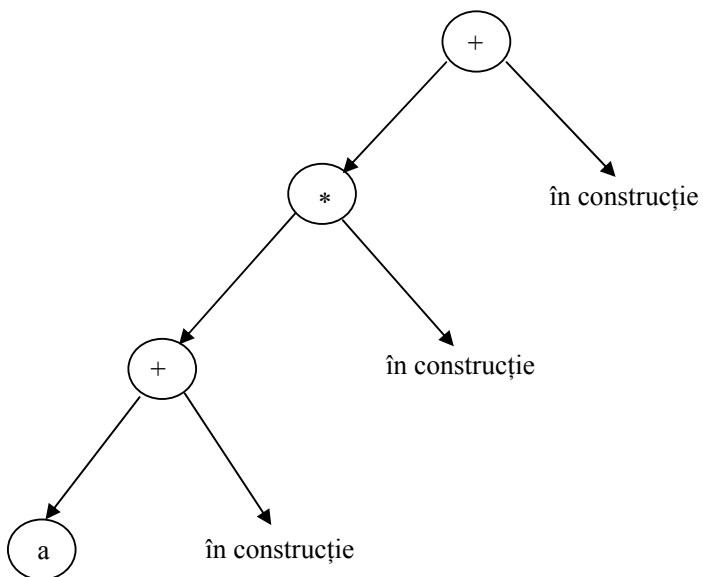
$p=1, u=7 \Rightarrow \min=10, i=4$ , arborele:



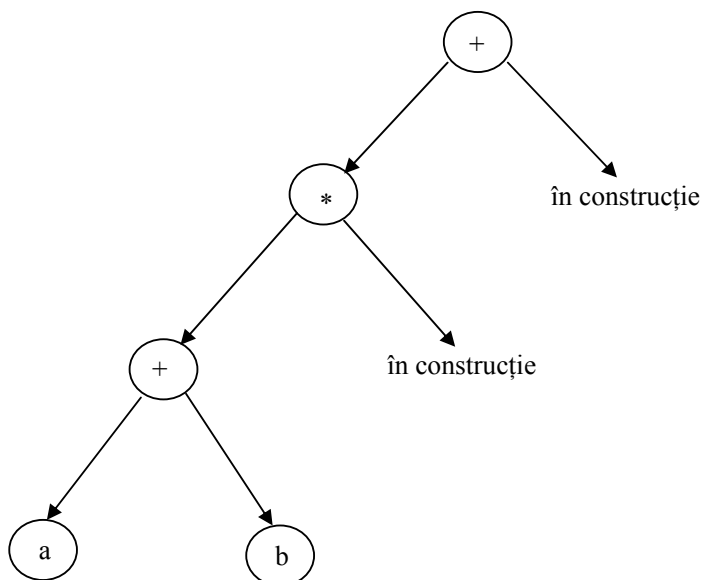
$p=1, u=3 \Rightarrow \min=11, i=2$ , arborele:



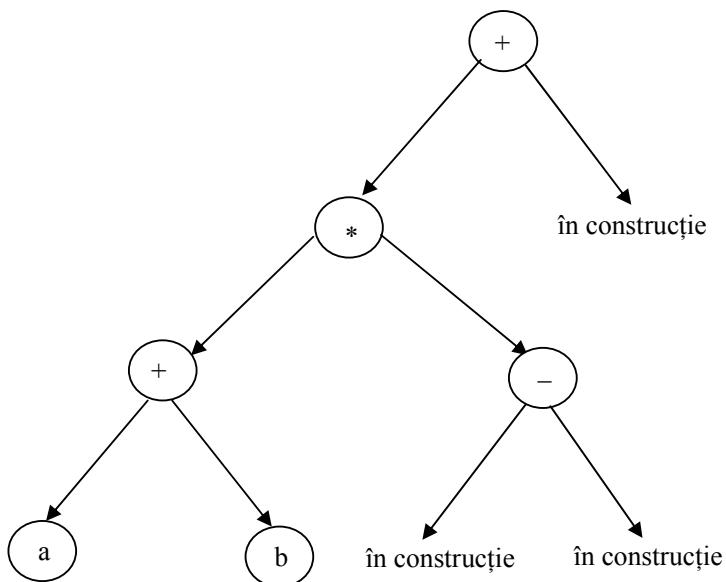
$p=u=1 \Rightarrow \min=\maxint, i=1$ , arborele:



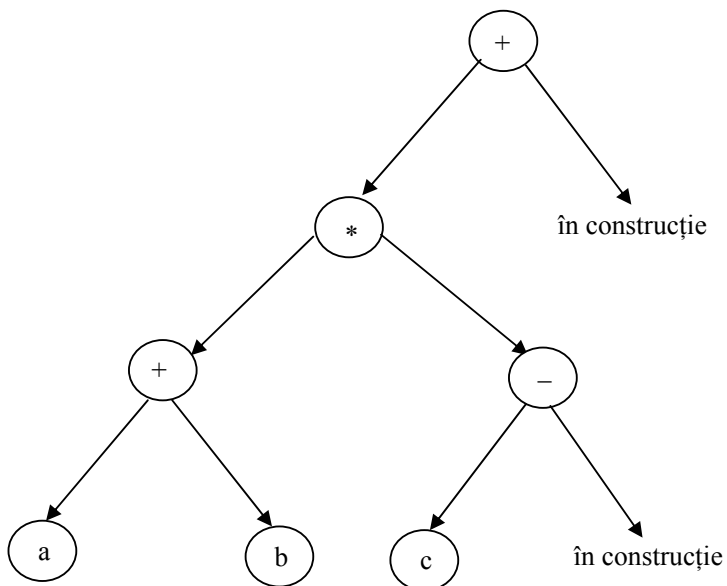
$p=3, u=3 \Rightarrow \min=\maxint, i=3$ , arborele:



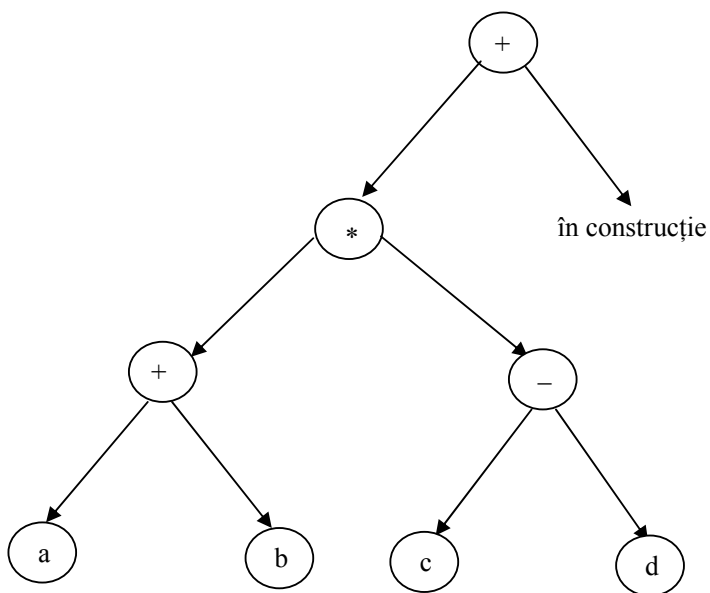
$p=5, u=7 \Rightarrow \min=11, i=6$ , arborele:



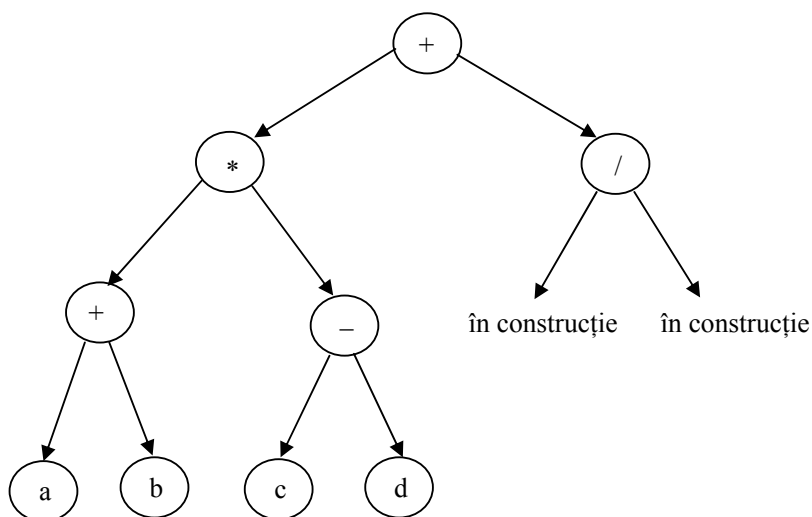
$p=5, u=5 \Rightarrow \text{min}=\text{maxint}, i=5$ , arborele:



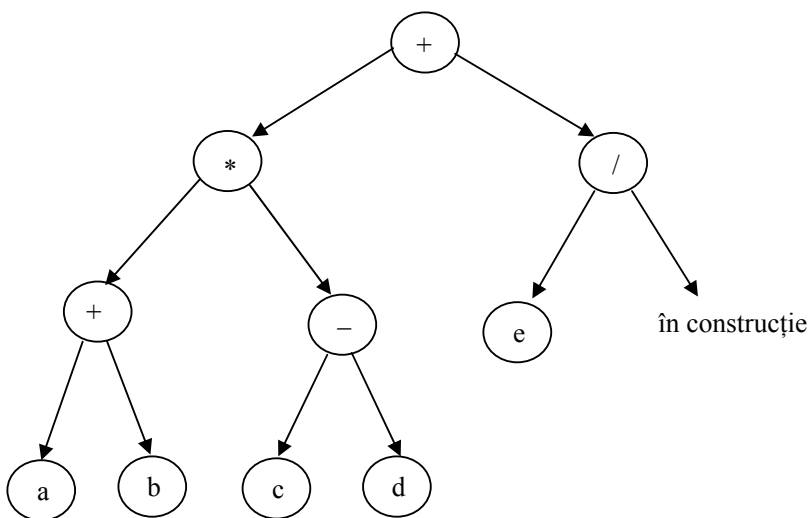
$p=7, u=7 \Rightarrow \text{min}=\text{maxint}, i=7$ , arborele:



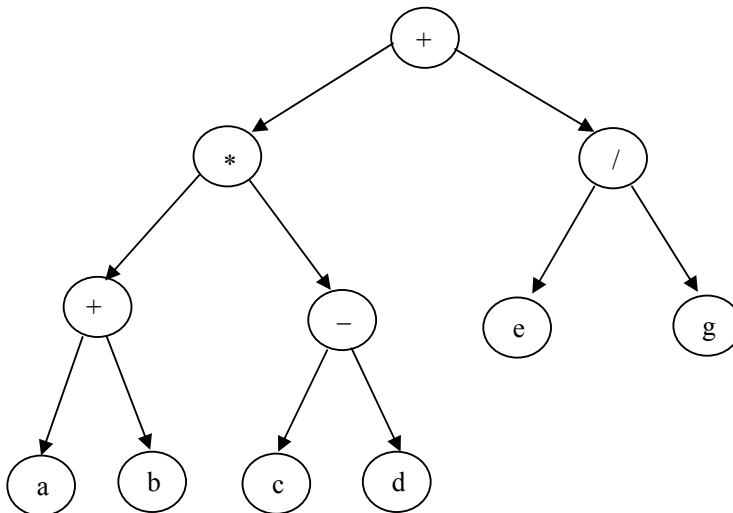
$p=9, u=11 \Rightarrow \min=10, i=10$ , arborele:



$p=9, u=9 \Rightarrow \min=\maxint, i=9$ , arborele:



$p=11, u=11 \Rightarrow \min=\maxint, i=11$ , arborele:



și construcția se termină.

Construcția arborelui de structură se face în ipoteza în care expresia este corectă. Procedura *cr der* se apelează cu:  $nil, 1, \text{length}(s), s, \text{prioritate}$ , unde  $s$  și *prioritate* sunt cei rezultați după apelarea procedurii *priorități*.

**Definiția 4.2.5.** Se numește *forma poloneză directă* a unei expresii, expresia rezultată în urma parcurgerii RSD a arborelui de structură. . Se numește *forma poloneză inversă* a unei expresii, expresia rezultată în urma parcurgerii SDR a arborelui de structură.

### Exemplu:

4.20. Pentru expresia considerată la exemplul 4.17, forma poloneză directă este  $+*+ab-cd/eg$ . Forma poloneză inversă a expresiei date este  $ab+cd-*eg/+$ .

Parcurea arborelui în inordine determină secvența de simboluri rezultată prin eliminarea parantezelor din expresia dată. Restaurarea unei forme parantezate poate fi realizată printr-o parcurgere SRD în modul următor. La momentul inițial, vârful curent este rădăcina arborelui de structură. Dacă vârful curent  $v$  nu este vârf terminal, atunci se generează  $(s1)$  eticheta( $v$ ) ( $s2$ ), unde eticheta( $v$ ) este operatorul etichetă a vârfului,  $s1$  este secvența rezultată prin traversarea SRD a subarborelui stâng,  $s2$  este secvența rezultată prin traversarea SRD a subarborelui drept. Dacă  $v$  este vârf terminal, atunci este generată secvența eticheta( $v$ ).



**Exemplu:**

4.21. Prin aplicarea traversării SRD a arborelui de structură al expresiei din 4.2.6, rezultă:

$$s=(((a)+(b))*((c)-(d)))+((e)/(g)).$$

Se observă că, prin aplicarea traversării SRD și a parantezării descrise, expresia rezultată are aceeași semnificație cu expresia inițială, dar apar multe paranteze “inutile”. Propunem ca exercițiu scrierea unei proceduri Pascal pentru eliminarea parantezelor inutile.

**Evaluarea expresiilor aritmetice pe baza arborilor de structură**

Traversarea SRD a arborelui de structură atașat unei expresii aritmetice permite evaluarea expresiei pentru valorile curente corespunzătoare variabilelor. Evaluarea poate fi efectuată în mod recursiv. La momentul inițial, vârful curent este rădăcina arborelui. Dacă  $v$  este vârf curent, atunci noua informație asociată lui  $v$  este:

- $\text{val}(\text{eticheta}(v))$ , dacă  $v$  este vârf terminal;
- $\text{val}(s1)\text{eticheta}(v)\text{val}(s2)$ , dacă  $v$  este neterminal,

unde  $\text{val}(s1)$ ,  $\text{val}(s2)$  sunt valorile rezultate prin evaluările subarborilor stâng și respectiv drept ai lui  $v$ ;  $\text{val}(\text{eticheta}(v))$  este valoarea curentă a variabilei, dacă eticheta lui  $v$  este variabilă, respectiv valoarea constantei, dacă eticheta lui  $v$  este o constantă.

Dacă  $v$  este vârf neterminal, atunci noua informație asociată lui  $v$  este  $\text{val}(s1)\text{eticheta}(v)\text{val}(s2)$ , ce reprezintă rezultatul operației eticheta( $v$ ) aplicată valorilor  $\text{val}(s1)$ ,  $\text{val}(s2)$ .

**Exemplu:**

4.22. Prin aplicarea metodei de evaluare descrise, se obține:

