

# ARBORELE (TREE)

- *Arborii* și variantele lor sunt printre cele mai comune și cele mai frecvent utilizate structuri de date, fiind utilizate într-o gamă foarte variată de aplicații cum ar fi teoria compilării, prelucrarea de imagini, etc., oferind o modalitate eficientă de memorare și manipulare a unei colecții de date.
- În teoria grafurilor, un arbore este un graf neorientat conex și fără cicluri.
- În informatică, *arborii cu rădăcină* sunt cei utilizați. De aceea, termenul *arbore* este asociat în informatică *arborelui cu rădăcină*.

**Definiție 1** *Un arbore este o mulțime finită  $\mathcal{T}$  cu 0 sau mai multe elemente numite noduri, care are următoarele caracteristici:*

- Dacă  $\mathcal{T}$  este vidă, atunci arborele este vid.
- Dacă  $\mathcal{T}$  este nevidă, atunci:
  - Există un nod special  $R$  numit rădăcină.
  - Celelalte noduri sunt partiționate în  $k$  ( $\geq 0$ ) arbori disjunși,  $T_1, T_2, \dots, T_k$ , nodul  $R$  fiind legat de rădăcina  $R_i$  a fiecărui  $T_i$  ( $1 \leq i \leq k$ ) printr-o muchie. Arborii  $T_1, T_2, \dots, T_k$  se numesc subarbori ai lui  $R$  (nodurile  $R_i$  sunt fii/descendenți ai lui  $R$ ), iar  $R$  se numește părintele subarborilor  $T_i$  ( $1 \leq i \leq k$ ), respectiv a nodurilor  $R_i$ .

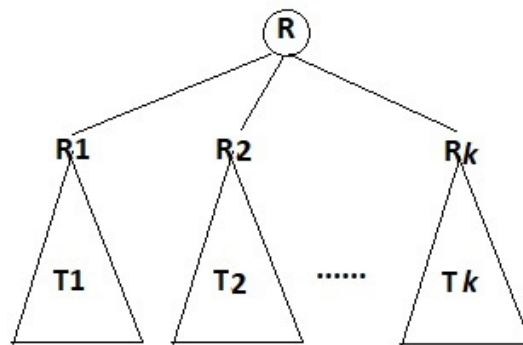


Figura 1: Arbore cu rădăcină.

**Arbore ordonat** - fiii fiecărui nod se consideră a forma o listă și nu doar o mulțime – adică ordinea filor este bine definită și relevantă.

- *gradul* unui nod este definit ca fiind numărul de fii ai nodului. Nodurile având gradul 0 se numesc *frunze*.
- *Adâncimea (nivelul)* unui nod în arbore este definită ca fiind lungimea (numărul de muchii) drumului unic de la rădăcină la acel nod. Ca urmare, rădăcina arborelui este pe nivelul 0.

- *Înălțimea* unui nod în arbore este definită ca fiind lungimea (numărul de muchii) celui mai lung drum de la nod la un nod frunză.
- *Înălțimea (adâncimea)* unui arbore este definită ca fiind înălțimea rădăcinii arborelui, adică nivelul maxim al nodurilor din arbore.

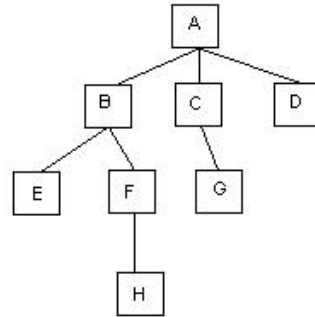


Figura 2: Arbore – exemplu.

Spre exemplu, arborele din Figura 2 are următoarele caracteristici:

- Rădăcina  $A$  este situată pe nivelul 0. Nodurile situate pe nivelul 1 sunt:  $B$ ,  $C$  și  $D$ . Nodurile situate pe nivelul 2 sunt:  $E$ ,  $F$  și  $G$ . Pe nivelul 3 există un singur nod, nodul  $H$ .
- Adâncimea (înălțimea) arborelui este 3.
- Nodul  $B$  are adâncimea 1 și înălțimea 2.

**Definiție 2 (Arbore binar)** *Un arbore ordonat în care fiecare nod poate să aibă cel mult 2 subarbori se numește arbore binar. Mai exact, putem defini arborele binar ca având următoarele proprietăți:*

- *Un arbore binar poate fi vid.*
- *Într-un arbore binar nevid, fiecare nod poate avea cel mult 2 fii (subarbori). Subarborii sunt identificați ca fiind subarborile stâng, respectiv drept. În Figura 3, nodul  $r$  are subarborile stâng  $A1$  și subarborile drept  $A2$ .*

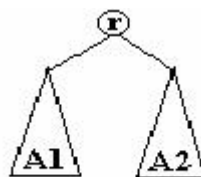


Figura 3: Arbore binar.

- Într-un arbore binar se face o distincție clară între subarboarele drept și cel stâng.
- Dacă subarboarele stâng este nevid, atunci rădăcina lui se numește fiul stâng al rădăcinii arborelui binar.
- Dacă subarboarele drept este nevid, rădăcina lui se numește fiul drept al rădăcinii arborelui binar.
- Arborii binari din Figura 4 sunt distincți, deși conțin aceeași mulțime de noduri.

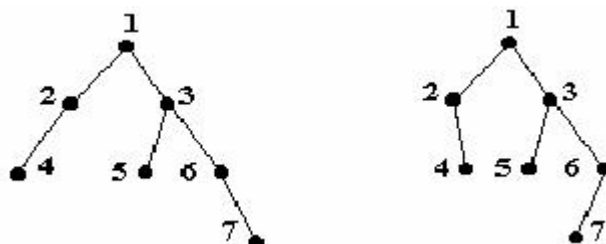


Figura 4: Arbori binari distincți.

Între arborii binari putem deosebi câteva categorii speciale:

- Un arbore binar pentru care fiecare nod interior are 2 fii (vezi Figura 5).
- Un arbore binar îl numim *plin* dacă fiecare nod interior are 2 fii și toate nodurile frunză au aceeași adâncime (vezi Figura 6).
- Un arbore binar are o structură de *ansamblu* (*heap*) dacă arborele este plin, exceptând ultimul nivel, care este plin de la stânga la dreapta doar până la un anumit loc (vezi Figura 7).

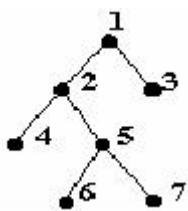


Figura 5: Arbore binar - nodurile interioare au 2 fii (*complet*).

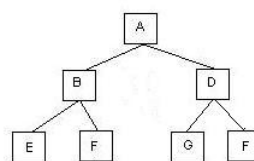


Figura 6: Arbore binar plin.

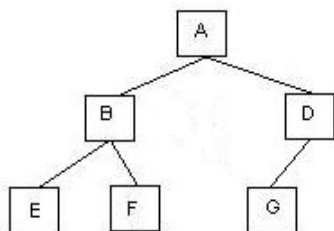


Figura 7: Arbore binar cu structură de ansamblu.



Figura 8: Arbori binari degenerați.

- Arborii binari se poate spune că au *formă*, forma lor fiind determinată de numărul nodurilor și de distanțele dintre noduri.
- Arborii binari din Figura 8 se numesc *degenerați*, deoarece au forma unui lanț de valori.
- Forma unui arbore influențează timpul necesar localizării unei valori în arbore.

**Arbore binar echilibrat** este un arbore binar cu proprietatea că înălțimea subarborelui său stâng nu diferă cu mai mult de  $\pm 1$  de înălțimea subarborelui său drept (și această proprietate este verificată pentru orice nod din arbore).

### Proprietăți ale AB:

1. Un arbore (nu neapărat binar) cu  $N$  vârfuri are  $N - 1$  muchii.
2. Numărul de noduri dintr-un arbore binar plin de înălțime  $N$  este  $2^{N+1} - 1$ .
3. Numărul maxim de noduri dintr-un arbore binar de înălțime  $N$  este  $2^{N+1} - 1$ .
4. Un arbore binar cu  $n$  vârfuri are înălțimea cel puțin  $\lceil \log_2 n \rceil$ .
5. Un arbore binar având o structură de ansamblu și  $n$  vârfuri are înălțimea  $\theta(\log_2 n)$ .

## Parcurgeri ale arborilor binari

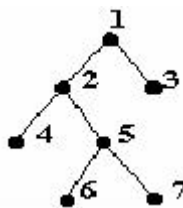


Figura 9: Arbore binar.

### Parcurgere în preordine

Pentru a parcurge în *preordine* un arbore binar, se vizitează rădăcina, apoi se parcurge în preordine subarborele stâng, apoi se parcurge în preordine subarborele drept (**RSD**).

- de exemplu, pentru arborele binar din Figura 9 parcurgerea în preordine este: 1, 2, 4, 5, 6, 7, 3.

## Parcurgere în inordine (ordine simetrică)

Pentru a parcurge în *inordine* (*ordine simetrică*) un arbore binar, se parcurge în inordine subarboarele stâng, se vizitează rădăcina, apoi se parcurge în inordine subarboarele drept (**SRD**).

- de exemplu, pentru arborele binar din Figura 9 parcurgerea în inordine este: 4, 2, 6, 5, 7, 1, 3.

## Parcurgere în postordine

Pentru a parcurge în *postordine* un arbore binar, se parcurge în postordine subarboarele stâng, apoi se parcurge în postordine subarboarele drept, după care se vizitează rădăcina (**SDR**).

- de exemplu, pentru arborele binar din Figura 9 parcurgerea în postordine este: 4, 6, 7, 5, 2, 3, 1.

## Parcurgere în lățime (pe niveluri)

Pentru a parcurge în *lățime* un arbore binar, se vizitează nodurile pe niveluri, în ordine de la stânga la dreapta: nodurile de pe nivelul 0, apoi nodurile de pe nivelul 1, nodurile de pe nivelul 2, etc.

- de exemplu, pentru arborele binar din Figura 9 parcurgerea în lățime este: 1, 2, 3, 4, 5, 6, 7.
- traversarea BFS (*Breadth first search*)

# TAD ArboreBinar (BINARY TREE)

## Observații

- Sunt aplicații în care am avea nevoie de memorarea datelor sunt forma unui arbore binar.
  - de exemplu, memorarea informațiilor dintr-un arbore genealogic.
- În majoritatea bibliotecilor existente, containerul **Tree** apare pe partea de GUI (ex. Java - clasa **JTree**).
  - arborele binar (de căutare) este folosit ca structură de date pentru a implementa containere: ex. în Java - `TreeSet`, `TreeMap`, etc)
- Dăm în continuare interfața minimală TAD ArboreBinar.

- Pe lângă operațiile de mai jos, pot fi adăugate operații care:
  - să **șteargă** un subarboare;
  - să **modifice** informația utilă a rădăcinii unui subarboare;
  - să **caute** un element în arbore, etc.

## TAD ArboreBinar

domeniu:

$$\mathcal{AB} = \{ab \mid ab \text{ arbore binar cu noduri care conțin informații de tip } TElement\}$$

operatii (interfața minimală):

- creeaza( $ab$ )
  - $pre : adevarat$
  - $post : ab \in \mathcal{AB}, ab = \text{arbore vid } (a = \Phi)$
- creeazaFrunza( $ab, e$ )
  - $pre : e \in TElement$
  - $post : ab \in \mathcal{AB}, ab$  arbore având un singur nod și informația din nodul rădăcină este egală cu  $e$
- creeazaArb( $ab, st, e, dr$ )
  - $pre : st, dr \in \mathcal{AB}, e \in TElement$
  - $post : ab \in \mathcal{AB}, ab$  arbore cu subarboare stâng =  $st$ , cu subarboare drept =  $dr$  și informația din nodul rădăcină este egală cu  $e$
- adaugaSubStang( $ab, st$ )
  - $pre : ab, st \in \mathcal{AB}$
  - $post : ab' \in \mathcal{AB}, ab'$  are subarboarele stâng =  $st$
- adaugaSubDrept( $ab, dr$ )
  - $pre : ab, dr \in \mathcal{AB}$
  - $post : ab' \in \mathcal{AB}, ab'$  are subarboarele drept =  $dr$
- element( $ab$ )
  - $pre : ab \in \mathcal{AB}, ab \neq \Phi$
  - $post : element = e, e \in TElement, e$  este informația din nodul rădăcină
- stang( $ab$ )
  - $pre : ab \in \mathcal{AB}, ab \neq \Phi$
  - $post : stang = st, st \in \mathcal{AB}, st$  este subarboarele stâng al lui  $ab$
- drept( $ab$ )
  - $pre : ab \in \mathcal{AB}, ab \neq \Phi$
  - $post : drept = dr, dr \in \mathcal{AB}, dr$  este subarboarele drept al lui  $ab$

- $\text{vid}(ab)$

$$\begin{aligned} \text{pre} : & \quad ab \in \mathcal{AB} \\ \text{post} : & \quad \text{vid} \begin{cases} \text{true} & \text{dacă } ab = \Phi \\ \text{false}, & \text{altfel} \end{cases} \end{aligned}$$

- $\text{iterator}(ab, \text{ordine}, i)$

$$\begin{aligned} \text{pre} : & \quad ab \in \mathcal{AB}, \text{ordine este ordinea de traversare a arborelui} \\ \text{post} : & \quad i \in \mathcal{I}, i \text{ este un iterator pe arborele } ab \text{ în ordinea} \\ & \quad \text{precizată de } \text{ordine} \end{aligned}$$

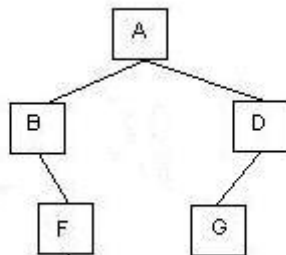
- $\text{distruge}(ab) \{\text{destructor}\}$

$$\begin{aligned} \text{pre} : & \quad ab \in \mathcal{AB} \\ \text{post} : & \quad ab \text{ a fost 'distrus' (spațiul de memorie alocat a fost eliberat)} \end{aligned}$$

## Reprezentări posibile pentru arbori binari

### A. Reprezentarea secvențială pe tablou

- Se folosește ca schemă de memorare un ansamblu  $(A[1..Max])$ :
  - $A_1$  este elementul din nodul rădăcină.
  - fiul stâng al lui  $A_i$  este  $A_{2 \cdot i}$ .
  - fiul drept al lui  $A_i$  este  $A_{2 \cdot i + 1}$ .
  - părintele lui  $A_i$  este  $A_{\lfloor i/2 \rfloor}$ .



Indice

[1]  
[2]  
[3]  
[4]  
[5]  
[6]  
[7]

Tablou

A
B
D
-1
F
G
-1

### B. Reprezentarea înlănțuită

Într-o astfel de reprezentare, în fiecare nod reprezentat al arborelui sunt memorate:

- informația utilă a nodului din arbore;
- două referințe spre cei doi fii – stâng și drept.

Arborele va fi identificat prin referința spre nodul rădăcină.

## B.1 Reprezentarea înlănțuirilor folosind alocarea dinamică a memoriei.

- Referințele sunt în acest caz pointeri (adrese de memorie).
- Pointerul NIL indică un arbore vid.

De exemplu, pentru un **Container** oarecare (de ex. Colecție) reprezentat sub forma unui AB:

### Nod

e: TElement //informația utilă nodului

st: ↑ Nod //adresa la care e memorat descendentul stâng

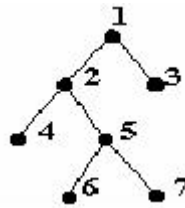
dr: ↑ Nod //adresa la care e memorat descendentul drept

### Container

rad: ↑ Nod //adresa rădăcinii AB

## B.2. Reprezentarea înlănțuirilor pe tablou.

Pentru arborele



reprezentarea (B.2) este

Indice	1	2	3	4	5	6	7	8	9	10
Element	3	-	5	1	-	2	6	-	4	7
Stanga	0	8	7	6	0	9	0	5	0	0
Dreapta	0		10	1		3	0		0	0

Tabela 1:  $radacina=4$ ,  $primLiber=2$

### Observație:

1. Capacitatea vectorilor poate fi mărită dinamic, dacă este necesar - numărul de elemente din arbore depășește numărul de locații alocat inițial (vezi vectorul dinamic).

În directorul asociat Cursului 11 găsiți implementarea parțială, în limbajul C++, a containerului **ArboreBinar** (reprezentarea este înlănțuită, cu alocare dinamică a nodurilor). Iteratorii nu sunt implementați.



## Parcurgeri recursive ale arborilor binari

- Variantele recursive de parcurgere sunt simplu de implementat (datorită definiției recursive a arborelui binar).
- Dezavantajul procedurilor recursive: supraîncărcarea stivei de execuție.

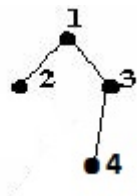
Considerând TAD `ArboreBinar` definit anterior, indiferent de reprezentarea acestuia, se pot tipări nodurile arborelui (ex. în `PREORDINE`), folosind procedura recursivă de mai jos:

```
Subalgoritm preordine(ab)
    {complexitate timp:  $\theta(n)$ }
pre:   ab este un arbore binar
post:  se afișează în preordine elementele din arbore
    {dacă arborele nu este vid}
Dacă  $\neg \text{vid}(ab)$  atunci
    {se prelucrează informația din rădăcina arborelui}
    element(ab, e)
    {prelucrează e}
    {se prelucrează recursiv subarborele stâng}
    preordine(stang(ab))
    {se prelucrează recursiv subarborele drept}
    preordine(drept(ab))
SfDacă
SfSubalgoritm
```

## Parcurgeri nerecursive ale arborilor binari

Pentru a implementa iteratorii pe arbore, avem nevoie de o parcurgere iterativă.

Presupunem (în cele ce urmează) reprezentare înlănțuită cu alocare dinamică.  
Pentru a exemplifica parcurgerile, considerăm AB de mai jos



### PREORDINE

Se va folosi o **STIVĂ** auxiliară.

```
Subalgoritm preordine(ab)
    {complexitate timp:  $\theta(n)$ }
```

*pre:*    *ab* este un arbore binar  
*post:*   se afișează în preordine elementele din arbore  
           {stiva va conține adrese de noduri}  
       creeaza(*S*)  
       Daca *ab.rad*  $\neq$  *NIL*    atunci  
           {arborele nu e vid}  
           adauga(*S, ab.rad*)  
           {se adauga radacina in stiva}  
       SfDaca  
       CatTimp  $\neg$ vida(*S*) executa  
           sterge(*S, p*)  
           {se sterge nodul din varful stiva}  
           @tipareste[*p*].*e*  
       Daca [*p*].*dr*  $\neq$  *NIL*    atunci  
           {exista legatura dreapta}  
           adauga(*S, [p].dr*)  
           {se adauga descendentul drept in stiva}  
       SfDaca  
       Daca [*p*].*st*  $\neq$  *NIL*    atunci  
           {exista legatura stanga}  
           adauga(*S, [p].st*)  
           {se adauga descendentul stang in stiva}  
       SfDaca  
       SfCatTimp  
       SfSubalgoritm

## Exemplificare

Stiva <i>S</i>	<i>p</i>	Ce se tipărește
①	-	-
∅	①	1
②	-	-
③	-	-
③	②	2
∅	③	3
④	-	-
∅	④	4
STOP		

## LĂȚIME (parcuregere pe niveluri)

Se va folosi o **COADĂ** auxiliară.

      Subalgoritm niveluri(*ab*)  
           {complexitate timp:  $\theta(n)$ }  
*pre:*    *ab* este un arbore binar  
*post:*   se afișează în lățime elementele din arbore  
           {coada va conține adrese de noduri}

```

creeaza(C)
Daca  $ab.rad \neq NIL$  atunci
    {arborele nu e vid}
    adauga( $C, ab.rad$ )
    {se adauga radacina in coada}
SfDaca
CatTimp  $\neg vida(C)$  executa
    sterge( $C, p$ )
    {se sterge nodul din coada}
    @tipareste[p].e
Daca  $[p].st \neq NIL$  atunci
    {exista legatura stanga}
    adauga( $C, [p].st$ )
    {se adauga descendentul stang in coada}
SfDaca
Daca  $[p].dr \neq NIL$  atunci
    {exista legatura dreapta}
    adauga( $C, [p].dr$ )
    {se adauga descendentul drept in coada}
SfDaca
SfCatTimp
SfSubalgoritm

```

## INORDINE

Se va folosi o **STIVĂ** auxiliară.

```

Subalgoritm inordine( $ab$ )
    {complexitate timp:  $\theta(n)$ }
pre:     $ab$  este un arbore binar
post:   se afişează în inordine elementele din arbore
        {stiva va conţine adrese de noduri}
creeaza( $S$ )
 $p \leftarrow ab.rad$ 
{nodul curent}
CatTimp  $\neg vida(S) \vee (p \neq NIL)$  executa
    CatTimp  $p \neq NIL$  executa
        {se adauga in stiva ramura stanga a lui p}
        adauga( $S, p$ )
         $p \leftarrow [p].st$ 
    SfCatTimp
    sterge( $S, p$ )
    {se sterge nodul din varful stivei}
    @tipareste[p].e
     $p \leftarrow [p].dr$ 
SfCatTimp
SfSubalgoritm

```

## Iterator INORDINE

Nod	AB	IteratorInordine
-----	----	------------------

$e:TElement$	$rad:\uparrow Nod$	$a:AB$
$st, dr:\uparrow Nod$		$curent:\uparrow Nod$
		$s:Stiv\grave{a}$

- se va folosi o **Stivă** care va conține  $\uparrow Nod$  și care are în interfață operațiile specifice:  $creeaz\grave{a}(s)$ ,  $vid\grave{a}(s)$ ,  $adaug\grave{a}(s,e)$ ,  $element(s,e)$ ,  $\grave{s}terge(s,e)$ .

### Varianta 1

```
Subalgoritm creeaz\grave{a}(i,ab)
  {constructorul iteratorului}
  i.ab ← ab
  i.curent ← ab.rad
  {constructorul stivei}
  creeaz\grave{a}(i.s)
SfSubalgoritm
```

```
Functia valid(i)
  {validitatea iteratorului}
  valid ← (i.curent ≠ NIL) ∨ ¬ vida(i.s)
SfFunctia
```

```
Subalgoritm element(i,e)
  {elementul curent al iteratorului}
  CatTimp i.curent ≠ NIL executa
    {se adauga in stiva ramura stanga a elementului curent}
    adauga(i.s, i.curent)
    i.curent ← [i.curent].st
  SfCatTimp
  sterge(i.s, i.curent)
  {se sterge nodul din varful stivei}
  e ← [i.curent].e
SfSubalgoritm
```

```
Subalgoritm urmator(i)
  {deplaseaz\grave{a} iteratorul}
  i.curent ← [i.curent].dr
SfSubalgoritm
```

### Varianta 2

```
Subalgoritm creeaz\grave{a}(i,ab)
  {constructorul iteratorului}
  i.ab ← ab
  {constructorul stivei}
  creeaz\grave{a}(i.s)
  i.curent ← ab.rad
  CatTimp i.curent ≠ NIL executa
    {se adauga in stiva ramura stanga a elementului curent}
    adauga(i.s, i.curent)
    i.curent ← [i.curent].st
```

```

SfCatTimp
Daca  $\neg \text{vida}(i.s)$  atunci
    {se acceseaza nodul din varful stivei}
     $\text{element}(i.s, i.curent)$ 
SfDaca
SfSubalgoritm

Functia  $\text{valid}(i)$ 
    {validitatea iteratorului}
     $\text{valid} \leftarrow (i.curent \neq \text{NIL})$ 
SfFunctia

Subalgoritm  $\text{element}(i, e)$ 
    {elementul curent al iteratorului}
     $e \leftarrow [i.curent].e$ 
SfSubalgoritm

Subalgoritm  $\text{urmator}(i)$ 
    {se sterge nodul din varful stivei}
     $\text{sterge}(i.s, i.curent)$ 
    Daca  $[i.curent].dr \neq \text{NIL}$  atunci
        {deplasează iteratorul}
         $i.curent \leftarrow [i.curent].dr$ 
        CatTimp  $i.curent \neq \text{NIL}$  executa
            {se adauga in stiva ramura stanga a elementului curent}
             $\text{adauga}(i.s, i.curent)$ 
             $i.curent \leftarrow [i.curent].st$ 
        SfCatTimp
    SfDaca
    Daca  $\neg \text{vida}(i.s)$  atunci
        {se acceseaza nodul din varful stivei}
         $\text{element}(i.s, i.curent)$ 
    altfel
         $i.curent \leftarrow \text{NIL}$ 
    SfDaca
SfSubalgoritm

```

## POSTORDINE

Se va folosi o **STIVĂ** auxiliară. Un element din stivă va fi de forma  $[p, k]$  unde:

- $p$  e adresa nodului;
- $k$  este 0 (dacă nu s-a trecut la partea dreaptă a lui  $p$ ) sau 1 (s-a trecut la parcurgerea subarborelui drept al lui  $p$ ).

```

Subalgoritm  $\text{postordine}(ab)$ 
    {complexitate timp:  $\theta(n)$ }
pre:   $ab$  este un arbore binar
post: se afișează în postordine elementele din arbore

```

```

creeaza(S)

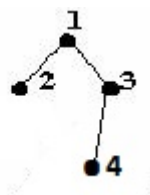

p ← ab.rad


{nodul curent}
CatTimp ¬ vida(S) ∨ (p ≠ NIL) executa
  CatTimp p ≠ NIL executa
    {se adauga in stiva ramura stanga a lui p}
    adauga(S, [p, 0])
    p ← [p].st
  SfCatTimp
  sterge(S, [p, k])
  {se sterge nodul din varful stivei}
  Daca k = 0 atunci
    {nu s-a traversat subarborele drept al lui p}
    adauga(S, [p, 1])
    p ← [p].dr
  altfel
    @tipareste[p].e
    p ← NIL
    {trebuie extras un nou nod din stiva - al doilea ciclu CatTimp nu trebuie sa se mai execute}
  SfDaca
SfCatTimp
SfSubalgoritm

```

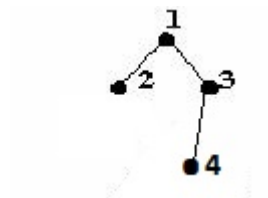
## Probleme

1. Implementați iteratori cu parcurgere în preordine, postordine și lățime a nodurilor unui arbore binar.
2. Descrieți în Pseudocod operația pentru căutarea într-un arbore binar a unei informații date. Se va folosi o implementare iterativă.
3. Descrieți în Pseudocod operația pentru determinarea înălțimii unui arbore binar. Se va folosi o operație iterativă.
4. Să se scrie o procedură iterativă pentru determinarea nivelului pe care apare într-un arbore binar o informație dată.
5. Scrieți o operație nerecursivă care determină părintele unui nod *p* dintr-un AB.
6. Cunoscând preordinea și inordinea nodurilor unui arbore binar, să se descrie o operație care construiește arborele. (vezi SEMINAR 7)
  - de exemplu. dacă preordinea (RSD) este **1** 2 3 4 și inordinea (SRD) este **1** 4 3, atunci arborele este



7. Cunoscand postordinea si inordinea nodurilor unui arbore binar, sa se descrie o operatie care construiește arborele. (vezi SEMINAR 7)

- de exemplu, dacă postordinea (SDR) este 2 4 3 **1** și inordinea (SRD) este 2 **1** 4 3, atunci arborele este



## Expresii aritmetice

O expresie aritmetică se poate reprezenta printr-un arbore binar ale cărui noduri terminale sunt asociate cu variabile sau constante și ale cărui noduri interne sunt asociate cu unul dintre operatorii:  $+$ ,  $-$ ,  $\times$ , și  $/$ . (Vezi Figura 11.)

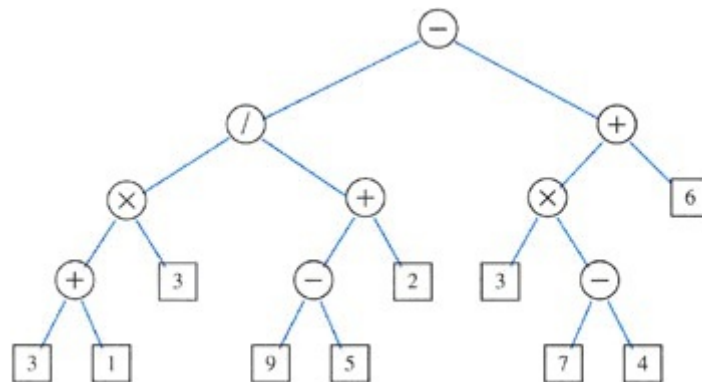


Figura 10: Arbore corespunzător expresiei  $((((3+1) \times 3) / ((9-5)+2)) - ((3 \times (7-4)) + 6))$ .

Fiecare nod dintr-un asemenea arbore are o valoare asociată:

- Dacă nodul este terminal valoarea sa este cea a variabilei sau constantei asociate.
- Dacă nodul este neterminal valoarea sa este definită prin aplicarea operației asociate asupra fiilor lui.

1. **Evaluarea unei expresii aritmetice din forma postfixată.** Fie  $EPost$  o expresie aritmetică CORECTĂ în forma postfixată, conținând operatorii binari:  $+$ ,  $-$ ,  $*$ ,  $/$ , iar ca operanzi cifre 0-9 (ex:  $EPost = 1\ 2\ +\ 3\ *\ 4\ /\$ ). Se cere să se determine valoarea expresiei (ex: valoarea este 2.25). **Indicație:** Se va folosi o stivă în care se vor adăuga operanzii. În final, stiva va conține valoarea expresiei.

Vom putea folosi următorul algoritm.

- Pentru fiecare  $e \in EPost$  (în ordine de la stânga la dreapta)
  - (a) Dacă  $e$  este operand, atunci se adaugă în stivă.
  - (b) Dacă  $e$  este operator, atunci se scot din stivă doi operanzi ( $op_1$  și  $op_2$ ), se efectuează operația  $e$  între cei doi operanzi ( $v = op_2 \ e \ op_1$ ), după care se adaugă  $v$  în stivă. **Obs.** Aici s-ar putea depista dacă expresia nu ar fi corectă în forma postfixată (s-ar încerca extragerea unui element dintr-o stivă vidă).
- În presupunerea noastră că expresia în forma postfixată este validă, stiva va conține la final o singură valoare, care se va extrage din stivă. Această valoare reprezintă valoarea expresiei.

**Exemplu.** Fie expresia  $EPost$   $1\ 2\ 3\ *\ +\ 4\ 5\ *\ -$ . Arborele asociat ar fi

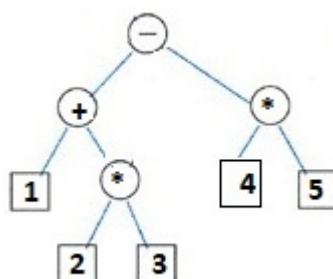


Figura 11: Arbore corespunzător expresiei în forma postfixată  $1\ 2\ 3\ *\ +\ 4\ 5\ *\ -$ .

Aplicarea algoritmului indicat mai sus este ilustrată în Tabelul de mai jos.

<del>3</del>		<del>5</del>		
<del>2</del>	<del>6</del>	<del>4</del>	<del>20</del>	
1	<del>1</del>	7	<del>7</del>	-13

Tabela 2: Stiva pe parcursul aplicării algoritmului

2. **Translatarea unei expresii aritmetice din forma infixată în forma postfixată.** Fie  $E$  o expresie aritmetică CORECTĂ în forma infixată, fără paranteze, conținând operatorii binari:  $+$ ,  $-$ ,  $*$ ,  $/$ , iar ca operanzi cifre 0-9 (ex:  $E = 1 + 2 * 3$ ). Se cere să se determine forma postfixată  $EPost$  a expresiei (ex:  $EPost = 1\ 2\ 3\ *\ +$ ). **Indicație:** Se va folosi o stivă în care se vor adăuga operatorii și o coadă  $EPost$  care va conține în final forma postfixată a expresiei.

Vom putea folosi următorul algoritm.

- Pentru fiecare  $e \in E$  (în ordine de la stânga la dreapta)
  - (a) Dacă  $e$  este operand, atunci se adaugă în coada  $EPost$ .



(b) Dacă  $e$  este operator, atunci se scot din stivă operatorii având prioritatea de evaluare mai mare sau egală decât a lui  $e$  și se adaugă în coada  $EPost$ , după care se adaugă  $e$  în stivă.

- Se scot din stivă operatorii rămași și se adaugă în coada  $EPost$ .
- În final,  $EPost$  va conține forma postfixată a expresiei.

**Exemplu.** Considerăm expresia  $E$  în forma infixată corespunzătoare exemplului din Figura 11:  $E = 1 + 2 * 3 - 4 * 5$ .

- $EPost$  ar trebui să fie  $1\ 2\ 3\ *\ +\ 4\ 5\ *\ -$

Aplicarea algoritmului indicat mai sus este ilustrată în Tabelul de mai jos.

		*	
		+	

$EPost$						
1	2	3				
1	2	3	*	+		
1	2	3	*	+	4	5
1	2	3	*	+	4	5

3. Translați o expresie aritmetică din forma infixată în forma postfixată. Expresia conține paranteze, care indică ordinea de evaluare. De exemplu, expresia  $(2+4)*6-(3+2)$  are forma postfixată  $2\ 4\ +\ 6\ *\ 3\ 2\ +\ -$  **Indicație:** Ideea/algoritmul de bază este același ca și în cazul în care expresia nu ar conține paranteze. Paranteza deschisă "(" se va adăuga în stivă. Va trebui să identificați pașii care vor trebui efectuați la întâlnirea unei paranteze închise ")".
4. Translați o expresie aritmetică din forma infixată în forma prefixată/postfixată. Folosiți un AB intermediar.
5. Evaluați o expresie aritmetică în forma infixată folosind un AB intermediar.