ARBORELE (TREE)

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

Definiţie 1 *Un arbore este o mulţime 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 partiţionate în k (≥ 0) arbori disjuncţi, $T_1, T_2, ..., T_k$, nodul R fiind legat de rădăcina R_i a fiecărui T_i $(1 \leq i \leq k)$ printr-o muchie. Arborii $T_1, T_2, ..., T_k$ se numesc subarbori ai lui R (nodurile R_i sunt* fii/descendenţi *ai lui R), iar R se numeşte* părintele *subarborilor T_i $(1 \leq i \leq k)$, respectiv a nodurilor R_i.*

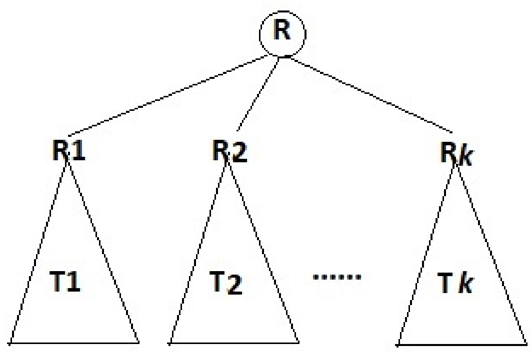


Figura 1: Arbore cu rădăcină.

Arbore ordonat - fiii fiecărui nod se consideră a forma o listă şi nu doar o mulţime – adică ordinea fiilor este bine definită şi 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 radacină la acel nod. Ca urmare, rădăcina arborelui este pe nivelul 0.

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

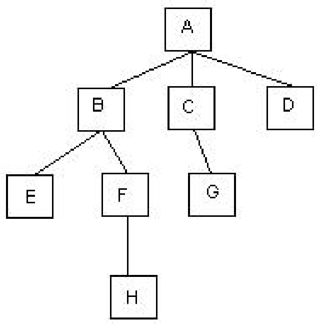


Figura 2: Arbore – exemplu.

Spre exemplu, arborele din Figura 2 are următoarele caracteristici:

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

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

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

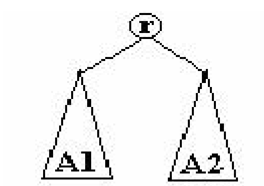


Figura 3: Arbore binar.

- Într-un arbore binar se face o distincţie clară între subarborele drept şi cel stâng.
- Dacă subarborele stâng este nevid, atunci rădăcina lui se numeşte fiul stâng al rădăcinii arborelui binar.
- Dacă subarborele drept este nevid, rădăcina lui se numeşte fiul drept al rădăcinii arborelui binar.
- Arborii binari din Figura 4 sunt distincţi, deşi conţin aceeaşi mulţime de noduri.

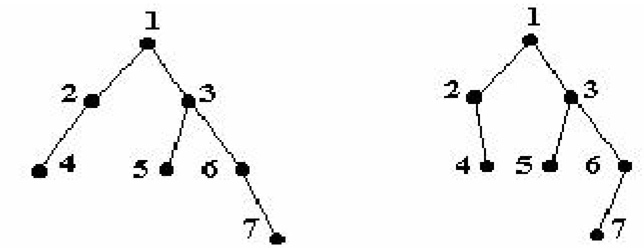


Figura 4: Arbori binari distincţi.

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 şi toate nodurile frunză au aceeaşi 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).

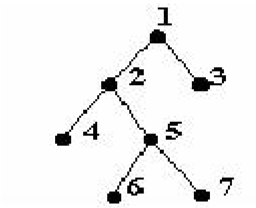


Figura 5: Arbore binar - nodurile interioare au 2 fii (*complet*).

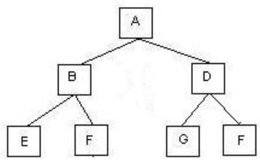


Figura 6: Arbore binar plin.

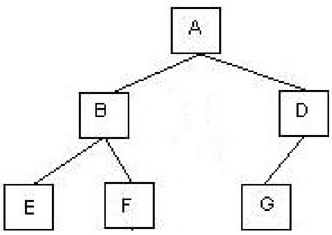


Figura 7: Arbore binar cu structură de ansamblu.

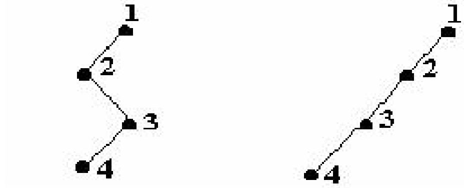


Figura 8: Arbori binari degeneraţi.

- Arborii binari se poate spune că au *formă*, forma lor fiind determinată de numărul nodurilor şi de distanţele dintre noduri.
- Arborii binari din Figura 8 se numesc *degeneraţi*, deoarece au forma unui lanţ de valori.
- Forma unui arbore influenţează timpul necesar localizării unei valori în arbore.

Arbore binar echilibrat este un arbore binar cu proprietatea că înălţimea subarborelui său stâng nu diferă cu mai mult de ± 1 de înălţimea subarborelui său drept (şi această prorietate este verificată pentru orice nod din arbore).

Proprietăţi 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 înălţime N este $2^{N+1}-1$.
3. Numărul maxim de noduri dintr-un arbore binar de înălţime N este $2^{N+1}-1$.
4. Un arbore binar cu n vârfuri are înălţimea cel puţin $[log_2 n]$.
5. Un arbore binar având o structură de ansamblu şi n vârfuri are înălţimea $\theta(log_2 n)$.

Parcurgeri ale arborilor binari

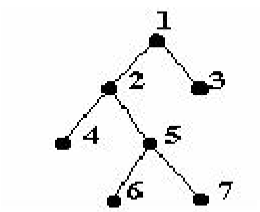


Figura 9: Arbore binar.

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 subarborele stâng, se vizitează rădăcina, apoi se parcurge în inordine subarborele 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 subarborele stâng, apoi se parcurge în postordine subarborele 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 lăţime (pe niveluri)

Pentru a parcurge în *lăţime* 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 lăţime este: $1, 2, 3, 4, 5, 6, 7$.
- traversarea BFS (*Breadth first search*)

TAD ArboreBinar
(BINARY TREE)

Observaţii

- Sunt aplicaţii în care am avea nevoie de memorarea datelor sunt forma unui arbore binar.
  - de exemplu, memorarea informaţiilor 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 continuarea interfaţa minimală TAD ArboreBinar.

- Pe lângă operaţiile de mai jos, pot fi adăugate operaţii care:
  - să **şteargă** un subarbore;
  - să **modifice** informaţia utilă a rădăcinii unui subarbore;
  - să **caute** un element în arbore, etc.

TAD ArboreBinar

domeniu:

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

operatii (interfaţa minimală):

- creeaza(ab)

  $pre:$ $adevarat$
  $post:$ $ab \in \mathcal{AB}, ab =$ arbore vid ($a = \Phi$)

- creeazaFrunza(ab, e)

  $pre:$ $e \in TElement$
  $post:$ $ab \in \mathcal{AB}, ab$ arbore având un singur nod şi informaţia 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 subarbore stâng $= st$, cu subarbore drept $= dr$ şi informaţia din nodul rădăcină este egală cu e

- adaugaSubStang(ab, st)

  $pre:$ $ab, st \in \mathcal{AB}$
  $post:$ $ab' \in \mathcal{AB}, ab'$ are subarborele stâng $= st$

- adaugaSubDrept(ab, dr)

  $pre:$ $ab, dr \in \mathcal{AB}$
  $post:$ $ab' \in \mathcal{AB}, ab'$ are subarborele drept $= dr$

- element(ab)

  $pre:$ $ab \in \mathcal{AB}, ab \neq \Phi$
  $post:$ $element = e, e \in TElement, e$ este informaţia din nodul rădăcină

- stang(ab)

  $pre:$ $ab \in \mathcal{AB}, ab \neq \Phi$
  $post:$ $stang = st, st \in \mathcal{AB}, st$ este subarborele stâng al lui ab

- drept(ab)

  $pre:$ $ab \in \mathcal{AB}, ab \neq \Phi$
  $post:$ $drept = dr, dr \in \mathcal{AB}, dr$ este subarborele stâng al lui ab

- vid(ab)

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

- iterator ($ab, ordine, i$)

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

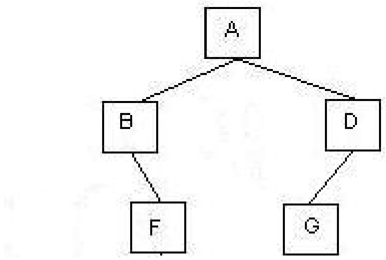
- distruge(ab) {destructor}

$pre:$ $ab \in \mathcal{AB}$
$post:$ ab a fost 'distrus' (spaţiul de memorie alocat a fost eliberat)

Reprezentări posibile pentru arbori binari

A. Reprezentarea secvenţială pe tablou

- Se foloseşte 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 $A_{[i/2]}$.



Indice	Tablou
[1]	A
[2]	B
[3]	D
[4]	-1
[5]	F
[6]	G
[7]	-1

B. Reprezentarea înlănţuită

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

- informaţia utilă a nodului din arbore;
- două referinţe spre cei doi fii – stâng şi drept.

Arborele va fi identificat prin referinţa spre nodul rădăcină.

B.1 Reprezentarea înlănţuirilor folosind alocarea dinamică a memoriei.

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

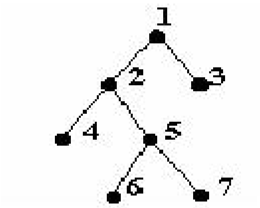
De exemplu, pentru un **Container** oarecare (de ex. Colecţie) reprezentat sub forma unui AB:

Nod
 e: TElement *//informaţia utilă nodului*
 st: $\uparrow$ Nod *//adresa la care e memorat descendentul stâng*
 dr: $\uparrow$ Nod *//adresa la care e memorat descendentul drept*

Container
 rad: $\uparrow$ Nod *//adresa rădăcinii AB*

B.2. Reprezentarea înlănţuirilor 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

Observaţie:

1. Capacitatea vectorilor poate fi mărită dinamic, dacă este necesar - numărul de elemente din arbore depăşeşte numărul de locaţii alocat iniţial (vezi vectorul dinamic).

În directorul asociat Cursului 10 găsiţi implementarea parţială, în limbajul C++, a containerului **ArboreBinar** (reprezentarea este înlănţuită, cu alocare dinamică a nodurilor). Iteratorii nu sunt implementaţi.

Parcurgeri recursive ale arborilor binari

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

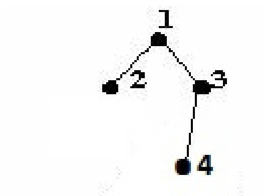
Considerând TAD ArboreBinar definit anterior, indiferent de reprezentarea acestuia, se pot tipări nodurile arborelui (ex. în PREORDINE), folosind procedura recursivă de mai jos:

```
Subalgoritm preordine(ab)
    {complexitate timp: θ(n)}
pre:    ab este un arbore binar
post:   se afişează în preordine elementele din arbore
    {dacă arborele nu este vid}
    Daca ¬ vid(ab) atunci
        {se prelucrează informaţia din rădăcina arborelui}
        element(ab, e)
        {prelucrează e}
        {se prelucrează recursiv subarborele stâng}
        preordine(stang(ab))
        {se prelucrează recursiv subarborele drept}
        preordine(drept(ab))
    SfDaca
SfSubalgoritm
```

Parcurgeri nerecursive ale arborilor binari

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

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



PREORDINE

Se va folosi o **STIVĂ** auxiliară.

```
Subalgoritm preordine(ab)
    {complexitate timp: θ(n)}
```

pre: ab este un arbore binar
post: se afişează în preordine elementele din arbore

```
  {stiva va conţine adrese de noduri}
  creeaza(S)
  Daca ab.rad ≠ NIL atunci
    {arborele nu e vid}
    adauga(S, ab.rad)
    {se adauga radacina in stiva}
  SfDaca
  CatTimp ¬vida(S) executa
    sterge(S, p)
    {se sterge nodul din varful stiva}
    @tipareste[p].e
    Daca [p].dr ≠ NIL atunci
      {exista legatura dreapta}
      adauga(S, [p].dr)
      {se adauga descendentul drept in stiva}
    SfDaca
    Daca [p].st ≠ NIL atunci
      {exista legatura stanga}
      adauga(S, [p].st)
      {se adauga descendentul stang in stiva}
    SfDaca
  SfCatTimp
SfSubalgoritm
```

Exemplificare

Stiva S	p	**Ce se tipăreşte**
1	-	-
$\emptyset$	1	1
2 3	-	-
3	2	2
$\emptyset$	3	3
4	-	-
$\emptyset$	4	4
STOP		

LĂŢIME (parcurgere pe niveluri)

Se va folosi o **COADĂ** auxiliară.

```
Subalgoritm niveluri(ab)
  {complexitate timp: θ(n)}
```
pre: ab este un arbore binar
post: se afişează în lăţime elementele din arbore

```
  {coada va conţine adrese de noduri}
```

```
    creeaza(C)
    Daca ab.rad ≠ NIL  atunci
      {arborele nu e vid}
      adauga(C, ab.rad)
      {se adauga radacina in coada}
    SfDaca
    CatTimp ¬vida(C) executa
      sterge(C, p)
      {se sterge nodul din coada}
      @tipareste[p].e
      Daca [p].st ≠ NIL  atunci
        {exista legatura stanga}
        adauga(C, [p].st)
        {se adauga descendentul stang in coada}
      SfDaca
      Daca [p].dr ≠ 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: θ(n)}
pre:    ab este un arbore binar
post:   se afişează în inordine elementele din arbore
    {stiva va conţine adrese de noduri}
    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)
        p ← [p].st
      SfCatTimp
      sterge(S, p)
      {se sterge nodul din varful stivei}
      @tipareste[p].e
      p ← [p].dr
    SfCatTimp
  SfSubalgoritm
```

Iterator INORDINE

Nod	**AB**	**IteratorInordine**
e:TElement	rad:↑Nod	a:AB
st, dr:↑Nod		$curent$:↑Nod
		s:Stivă

- se va folosi o **Stivă** care va conţine ↑**Nod** şi care are în interfaţă operaţiile specifice: *creează*(s), *vidă*(s), *adaugă*(s,e), *element*(s,e), *şterge*(s,e).

<u>Varianta 1</u>

```
Subalgoritm creează(i, ab)
   {constructorul iteratorului}
   i.ab ← ab
   i.curent ← ab.rad
   {constructorul stivei}
   creează(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)
   {deplasează iteratorul}
   i.curent ← [i.curent].dr
SfSubalgoritm
```

Observaţie

- în Varianta 1 de implementare a iteratorului, în subagoritmul `element` se modifică atributul `curent` al iteratorului $\Longrightarrow$ metoda `element` din implementarea C++ nu va putea fi declarată ca fiind `const`.

Varianta 2

```
Subalgoritm creează(i, ab)
   {constructorul iteratorului}
   i.ab ← ab
   {constructorul stivei}
   creează(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 ¬ vida(i.s) atunci
      {se acceseaza nodul din varful stivei}
      element(i.s, i.curent)
   SfDaca
SfSubalgoritm

Functia valid(i)
   {validitatea iteratorului}
   valid← (i.curent ≠ NIL)
SfFunctia


Subalgoritm element(i, e)
   {elementul curent al iteratorului}
   e ← [i.curent].e
SfSubalgoritm

Subalgoritm urmator(i)
   {se sterge nodul din varful stivei}
   sterge(i.s, i.curent)
   Daca [i.curent].dr ≠ NIL atunci
      {deplasează iteratorul}
      i.curent ← [i.curent].dr
      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
   SfDaca
   Daca ¬ vida(i.s) atunci
      {se acceseaza nodul din varful stivei}
      element(i.s, i.curent)
   altfel
      i.curent ← NIL
   SfDaca
SfSubalgoritm
```

Observaţie

- în Varianta 2 de implementare a iteratorului, subagoritmul `element` nu modifică iteratorul $\Longrightarrow$ metoda `element` din implementarea C++ va putea fi declarată ca fiind `const`.

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 postordine(ab)
  {complexitate timp: θ(n)}
pre:   ab este un arbore binar
post:  se afişează 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. Implementaţi iteratori cu parcurgere în preordine, postordine şi lăţime a nodurilor unui arbore binar.
2. Descrieţi în Pseudocod operaţia pentru căutarea într-un arbore binar a unei informaţii date. Se va folosi o implementare iterativă.

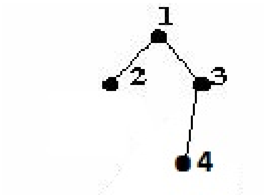
3. Descrieţi în Pseudocod operaţia pentru determinarea înălţimii unui arbore binar. Se va folosi o operaţie iterativă.

4. Să se scrie o procedură iterativă pentru determinarea nivelului pe care apare într-un arbore binar o informaţie dată.

5. Scrieţi o operaţie nerecursivă care determină părintele unui nod p dintr-un AB.

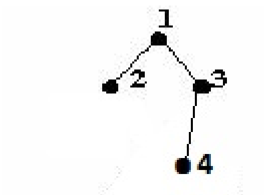
6. Cunoscand preordinea si inordinea nodurilor unui arbore binar, sa se descrie o operatie care construieste arborele. (vezi SEMINAR 7)

   - de exemplu. dacă preordinea (RSD) este **1** 2 3 4 şi inordinea (SRD) este 2 **1** 4 3, atunci arborele este



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

   - de exemplu, dacă postordinea (SDR) este 2 4 3 **1** şi 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 şi ale cărui noduri interne sunt asociate cu unul dintre operatorii: $+$, -, $\times$, şi $/$. (Vezi Figura 11.)

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 operaţiei asociate asupra fiilor lui.

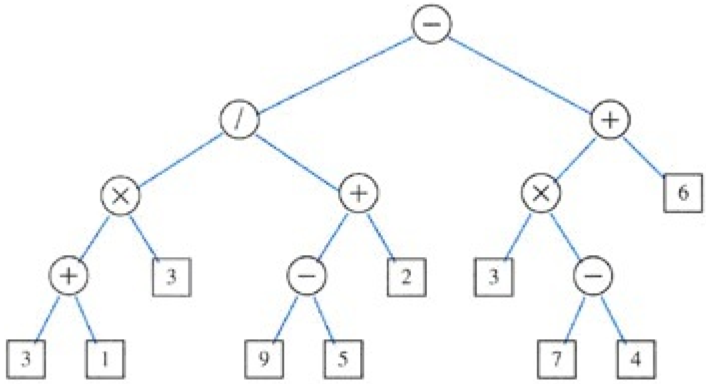


Figura 10: Arbore corespunzător expresiei $(((3+1)\times 3)/((9-5)+2))-((3\times(7-4))+6)$.

1. **Evaluarea unei expresii aritmetice din forma postfixată.** Fie *EPost* o expresie aritmetică CORECTĂ în forma postfixată, conţinând 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). **Indicaţie:** Se va folosi o stivă în care se se vor adăuga operanzii. În final, stiva va conţine 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 şi op_2), se efectuează operaţia e înte 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 conţine 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

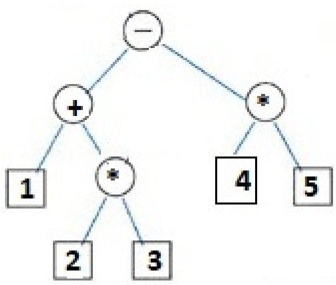


Figura 11: Arbore corespunzător expresiei în forma postfixată 1 2 3 * + 4 5 * -.

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

$\cancel{3}$		$\cancel{5}$		
$\cancel{2}$	$\cancel{6}$	$\cancel{4}$	$\cancel{20}$	
1	$\cancel{1}$	7	$\cancel{7}$	$\boxed{\textbf{-13}}$

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, conţinând operatorii binari: +, -, *, /, iar ca operanzi cifre 0-9 (ex: $E = 1 + 2 * 3$). Se cere să se determine forma postfixată $EPost$ a expresiei (ex: $EPost = 1\ 2\ 3\ *\ +$). **Indicaţie:** Se va folosi o stivă în care se se vor adăuga operatorii şi o coadă $EPost$ care va conţine 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 şi se adaugă în coada $EPost$, după care se adaugă e în stivă.
   - Se scot din stivă operatorii rămaşi şi se adaugă în coada $EPost$.
   - În final, $EPost$ va conţine 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.

	*	
	-	
*	$\cancel{*}$	
+	$\cancel{+}$	∅

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

3. Translataţi o expresie aritmetică din forma infixată în forma postfixată. Expresia conţine paranteze, care indică ordinea de evaluare. De exemplu, expresia (2+4)*6-(3+2) are forma postfixată 2 4 + 6 * 3 2 + - **Indicaţie:** Ideea/algoritmul de bază este acelaşi ca şi în cazul în care expresia nu ar conţine paranteze. Paranteza deschisă "("se va adăuga în stivă. Va trebui să identificaţi paşii care vor trebui efectuaţi la întâlnirea unei paranteze închise ")".

4. Translataţi o expresie aritmetică din forma infixată în forma prefixată/postfixată. Folosiţi un AB intermediar.

5. Evaluaţi o expresie aritmetică în forma infixată folosind un AB intermediar.