

## 8. Arbori

### 8.1. Arbori generalizați

#### 8.1.1. Definiții

- În definirea **noțiunii de arbore** se pornește de la noțiunea de **vector**.
  - Fie  $V$  o mulțime având elementele  $a_1, a_2, \dots, a_n$ .
  - Pe mulțimea  $V$  se poate defini o așa numită "**relație de precedență**" în felul următor: se spune că  $a_i$  precede pe  $a_j$  dacă  $i < j$ . Aceasta se notează:  $a_i < a_j$ .
  - Se poate verifica ușor că relația astfel definită are următoarele proprietăți, valabile pentru structura vector:

- 
- (1) Oricare ar fi  $a \in V$  avem  $a \not< a$ . (S-a notat cu  $\not<$  relația "nu precede");  
(antireflexivitate)
- (2) Dacă  $a < b$  și  $b < c$  atunci  $a < c$  (tranzitivitate); [ 8.1.1.a ]
- (3) Oricare ar fi  $a \in V$  și  $b \in V$ , dacă  $a \neq b$  atunci avem fie  $a < b$  fie  $b < a$ .
- 

- Din proprietățile (1) și (2) rezultă că relația de precedență **nu** determină în  $V$  "**bucle închise**", adică **nu** există nici o secvență de elemente care se preced două câte două și în care ultimul element este același cu primul, cum ar fi de exemplu  $a < b < c < d < a$ .
- Proprietatea (3) precizează că relația de precedență este definită pentru **oricare** două elemente  $a$  și  $b$  ale lui  $V$ , cu singura condiție ca  $a \neq b$ .
- Fie  $V$  o mulțime finită peste elementele căreia s-a definit o relație de precedență, stabilind referitor la fiecare pereche de elemente, care dintre ele îl precede pe celălalt.
  - Dacă această relație posedă proprietățile [8.1.1.a], atunci ea **imprima** peste mulțimea  $V$  o **structură vector** (fig.8.1.1.a).

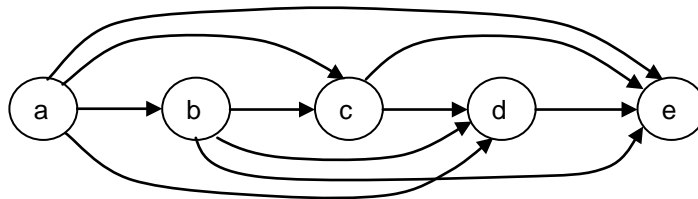
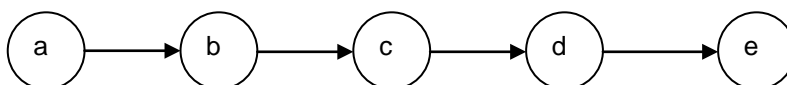


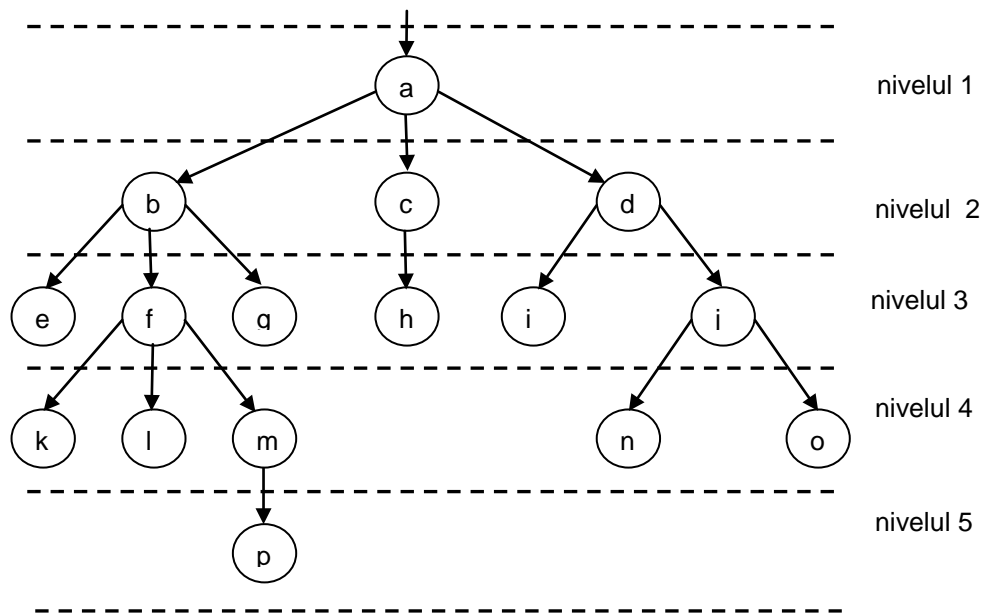
Fig. 8.1.1.a. Structură vector

- În figura 8.1.1.b apare o altă reprezentare intuitivă a unei structuri **vector**. Săgețile din figură indică relația "**succesor**".



**Fig.8.1.1.b. Relația succesor**

- Relația "**succesor**" se definește cu ajutorul relației de **precedență** după cum urmează:
  - Dacă între elementele  $a$  și  $b$  ale lui  $\mathbf{V}$  este valabilă relația  $a < b$  și nu există nici un  $c \in \mathbf{V}$  astfel ca  $a < c < b$  atunci se zice că  $b$  este **succesorul** lui  $a$ .
- Se observă că relația "**succesor**" (mulțimea săgeților din figura 8.1.1.b.), **precizează** relația "**precedență**" fără a fi însă **identică** cu ea.
  - Spre exemplu, există relația  $b < e$  (prin tranzitivitate), dar nici o săgeată nu conectează pe  $b$  cu  $e$ .
- În figura 8.1.1.c apare o așa numită **structură arbore** care se definește prin **generalizarea** structurii **vector**.



**Fig. 8.1.1.c. Structură arbore**

- Astfel, dacă în cazul **vectorului**, **toate elementele** cu excepția ultimului au **exact un succesor**, la **structura de arbore** se admite ca **fiecare element** să aibă un **număr oarecare de succesori**, inclusiv zero, cu **restricția** ca două elemente distincte să **nu** aibă același succesor.
- Relația **succesor** definește o relație de **precedență** pe structura arbore. Astfel, din figura avem  $b < p$ ,  $d < n$ , etc.
- **Relația de precedență** definită pe structura arbore se bucură de proprietățile (1) și (2) de la [8.1.1.a] dar **nu** satisface proprietatea (3).
  - Într-adevăr în figura 8.1.1.c, pentru elementele  $b$  și  $c$  **nu** este valabilă nici una din relațiile  $b < c$  sau  $c < b$ , la fel pentru elementele  $d$  și  $k$ .
  - Prin urmare, relația de precedență este definită numai pentru **o parte** a perechilor de elemente ale arborelui, cu alte cuvinte este o **relație parțială**.
- În general, o **structură arbore** se definește ca o mulțime  $\mathbf{A}$  de  $n \leq 0$  noduri de același tip, peste care s-a definit o relație de precedență având proprietățile (1) și (2) de la [8.1.1.a] precum și următoarele două proprietăți [8.1.1b]:

- 
- (3) Oricare ar fi  $b, c \in A$ , astfel încât  $b \not\prec c$  și  $c \not\prec b$ , dacă  $b \prec d$  și  $c \prec e$  atunci  $d \neq e$ .

- Cu alte cuvinte, două elemente oarecare între care **nu** există relația de precedență **nu** pot avea același succesor.

[ 8 . 1 . 1 . b ]

- (4) Dacă **A nu** este vidă ( $n > 0$ ) atunci există un element numit **rădăcină**, care precede toate celelalte elemente.
- 

- Pentru structura arbore se poate formula și o altă definiție echivalentă cu cea de mai sus.
- Prin **arbore**, se înțelege o mulțime de  $n \geq 0$  noduri de același tip, care dacă **nu** este vidă, atunci are un anumit nod numit **rădăcină**, iar restul nodurilor formează **un număr finit de arbori**, doi câte doi **disjuncți**.
- Se constată că atât această definiție, cât și structura pe care o definește, sunt **recursive**, lucru deosebit de important deoarece permite prelucrarea simplă a unei astfel de structuri cu ajutorul unor **algoritmi recursivi**.
- În continuare se vor defini câteva **noțiuni** referitoare la arbori.
- Prin **subarborii** unui arbore, se înțeleg arborii în care se descompune acesta prin îndepărtarea rădăcinii.
  - Spre exemplu arborele din figura 8.1.1.c, după îndepărtarea rădăcinii  $a$ , se descompune în trei subarbori având rădăcinile respectiv  $b, c$  și  $d$ .
- Oricare nod al unui arbore este rădăcina unui **arbore parțial**.
  - Spre exemplu în aceeași figură,  $f$  este rădăcina arborelui parțial format din nodurile  $f, k, l, m$  și  $p$ .
- Un arbore parțial **nu** este întotdeauna **subarbore** pentru arborele complet, în schimb orice **subarbore** este în același timp și **arbore parțial**.
- Într-o structură de arbore, succesorul unui nod se mai numește și "**fiul**" sau "**urmașul**" său.
- Dacă un nod are unul sau mai mulți fii, atunci el se numește "**tatăl**" sau "**părintele**" acestora.
- Dacă un nod are mai mulți fii, aceștia se numesc "**frați**" între ei.
  - Spre exemplu în fig. 8.1.1.c nodul  $b$  este tatăl lui  $e, f$  și  $g$  care sunt frați între ei și sunt în același timp fiii lui  $b$ .
- Se observă că într-o structură arbore, **arborele parțial** determinat de orice nod diferit de rădăcină, este **subarbore** pentru **arborele parțial** determinat de tatăl său.

- Astfel  $f$  este tatăl lui  $m$ , iar arborele parțial determinat de  $m$  este subarbore pentru arborele parțial determinat de  $f$ .
- Într-o structură arbore se definesc **niveluri** în felul următor: rădăcina formează nivelul 1, fiii ei formează nivelul 2 și în general fiii tuturor nodurilor nivelului  $n$  formează nivelul  $n+1$  (fig.8.1.1.c).
- Nivelul maxim al nodurilor unui arbore se numește **înălțimea** arborelui.
- Numărul fiilor unui nod definește **gradul** nodului respectiv.
- Un nod de grad zero se numește nod **terminal (frunză)**, iar un nod de grad diferit de zero, nod **intern**.
- **Gradul** maxim al nodurilor unui arbore se numește **gradul arborelui**.
  - Arborele din figura 8.1.1.c are înălțimea 5, nodul  $d$  este de grad 2, nodul  $h$  este terminal,  $f$  este un nod intern iar gradul arborelui este 3.
- Dacă  $n_1, n_2, \dots, n_k$  este o **secvență** de noduri aparținând unui arbore, astfel încât  $n_i$  este părintele lui  $n_{i+1}$  pentru  $1 \leq i < k$ , atunci această secvență se numește "**drum**" sau "**cale**" de la nodul  $n_1$  la nodul  $n_k$ .
- **Lungimea** unui **drum** este un întreg având valoarea egală cu numărul de ramuri (săgeți) care trebuie traversate pentru a ajunge de la rădăcină la nodul respectiv.
- Rădăcina are lungimea drumului egală cu 1, fiii ei au lungimea drumului egală cu 2 și în general un nod situat pe nivelul  $i$  are lungimea drumului  $i$ .
  - Spre exemplu, în figura 8.1.1.c, lungimea drumului la nodul  $d$  este 2 iar la nodul  $p$  este 5.
- Dacă există un drum de la nodul  $a$  la nodul  $b$ , atunci nodul  $a$  se numește **strămoș** sau **ancestor** al lui  $b$ , iar nodul  $b$  **descendent** sau **urmaș** al lui  $a$ .
  - Spre exemplu în aceeași figură, strămoșii lui  $f$  sunt  $f, b$  și  $a$  iar descendenții săi  $f, k, l, m$  și  $p$ .
- Conform celor deja precizate **tatăl** unui nod este **strămoșul său direct (predecesor)** iar **fiul** unui nod este **descendentul său direct (succesor)**.
- Un strămoș respectiv un descendent al unui nod, altul decât nodul însuși, se numește **strămoș propriu** respectiv **descendent propriu**.
- Într-un arbore, **rădăcina** este **singurul nod** fără **nici** un strămoș propriu.
- Un nod care **nu** are descendenți proprii se numește **nod terminal (frunză)**.
- **Înălțimea** unui nod într-un arbore este **lungimea celui mai lung drum** de la **nodul respectiv** la un **nod terminal**.

- Pornind de la această definiție, **înălțimea** unui arbore se poate defini și ca fiind egală cu **înălțimea nodului rădăcină**.
- **Adâncimea** unui nod este egală cu lungimea **drumului unic** de la rădăcină la acel nod.
- În practică se mai definește și **lungimea drumului unui arbore** numită și **lungimea drumului intern**, ca fiind egală cu suma lungimilor drumurilor corespunzătoare tuturor nodurilor arborelui.
- Formal, **lungimea medie a drumului intern al unui arbore**, notată cu  $P_I$  se definește cu ajutorul formulei [8.1.1.c].

$$P_I = \frac{1}{n} \sum_{i=1}^h n_i * i$$

unde  $n$  = numărul total de noduri

$i$  = nivelul nodului

[ 8 . 1 . 1 . c ]

$n_i$  = numărul de noduri pe nivelul  $i$

$h$  = înălțimea arborelui

- În scopul definirii **lungimii drumului extern**, ori de câte ori se întâlnește în arborele inițial un **subarbore vid**, acesta se înlocuiește cu un **nod special**.
- În structura care se obține după această activitate, toate nodurile originale vor fi de același **grad** (gradul arborelui).
- Se precizează că **nodurile speciale** care au înlocuit ramurile absente **nu** au descendenți, prin definiție.
  - Spre exemplu, arborele (a) din figura 8.1.1.d, extins cu noduri speciale reprezentate prin patrate, ia forma (b)..
- **Lungimea drumului extern** se definește ca fiind suma lungimilor drumurilor la toate nodurile speciale.
- Dacă numărul nodurilor speciale la nivelul  $i$  este  $m_i$ , atunci **lungimea medie a drumului extern**  $P_E$  este cea definită de formula [8.1.1.d].

$$P_E = \frac{1}{m} \sum_{i=1}^{h+1} m_i * i$$

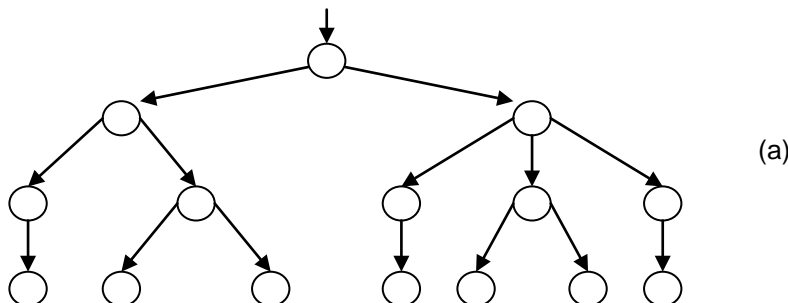
unde  $m$  = numărul total de noduri speciale

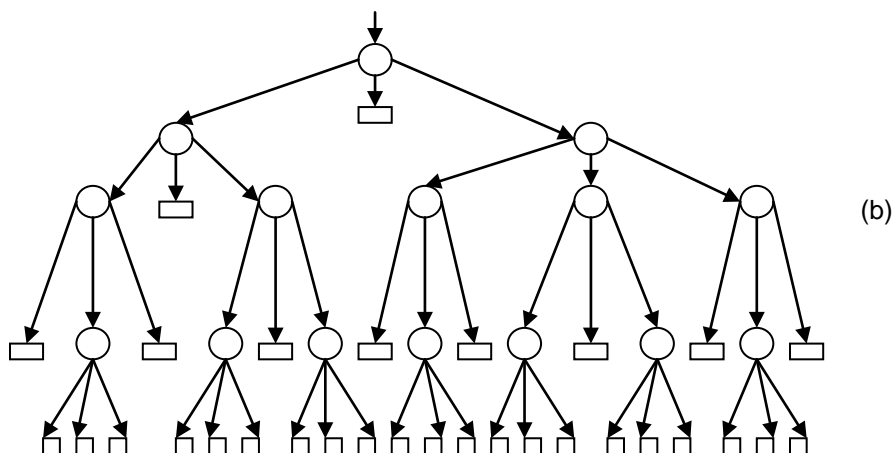
$i$  = nivelul nodului special

[ 8 . 1 . 1 . d ]

$m_i$  = numărul de noduri speciale pe nivelul  $i$

$h$  = înălțimea arborelui





**Fig. 8.1.1.d.** Arbore ternar extins cu noduri speciale

### 8.1.2. Tipul de date abstract arbore generalizat

- La fel ca și în cazul altor tipuri de structuri de date, este dificil de stabilit un set de operatori care să fie valabil pentru toate aplicațiile posibile ale structurilor de tip **arbore**.
- Cu toate acestea, din mulțimea operatorilor posibili se recomandă pentru **TDA Arbore** generalizat forma prezentată în secvența [8.1.2.a].

---

#### TDA Arbore generalizat

**Modelul matematic:** arbore definit în sens matematic.

Elementele arborelui aparțin unui aceluiași tip, numit și *tip de bază*.

#### **Notății:**

*TipNod* - tipul unui nod al arborelui;

*TipArbore* - tipul arbore;

*TipCheie* - tipul cheii unui nod;

*N*: *TipNod*;

*A*: *TipArbore*;

*v*: *TipCheie*;

[8.1.2.a]

#### **Operatori:**

1. **Tata**(*N*: *TipNod*, *A*:*TipArbore*): *TipNod*; - operator care returnează tatăl (părintele) nodului *N* din arborele *A*. Dacă *N* este chiar rădăcina lui *A* se returnează "nodul vid";

2. **PrimulFiu**(*N*: *TipNod*, *A*: *TipArbore*): *TipNod*; -

operator care returnează cel mai din stânga fiu al nodului  $N$  din arborele  $A$ . Dacă  $N$  este un nod terminal, operatorul returnează "nodul vid".

3. **FrateDreapta**( $N: \text{TipNod}, A: \text{TipArbore}$ ):  $\text{TipNod}$ ;  
operator care returnează nodul care este fratele dreapta "imediat" al nodului  $N$  din arborele  $A$ . Acest nod se definește ca fiind nodul  $M$  care are același părinte  $T$  ca și  $N$  și care în reprezentarea arborelui apare imediat în dreapta lui  $N$  în ordonarea de la stânga la dreapta a fiilor lui  $T$ .
4. **Cheie**( $N: \text{TipNod}, A: \text{TipArbore}$ ):  $\text{TipCheie}$ ; - operator care returnează cheia nodului  $N$  din arborele  $A$ .
5. **Creaza<sub>i</sub>**( $v: \text{TipCheie}, A_1, A_2, \dots, A_i: \text{TipArbore}$ ):  
 $\text{TipArbore}$ ; este unul din operatorii unei familii, care are reprezentanți pentru fiecare din valorile lui  $i=0,1,2,\dots$ . Funcția **Creaza<sub>i</sub>** generează un nod nou  $R$ , care are cheia  $v$  și căruia îi asociază  $i$  fii. Aceștia sunt respectiv subarborii  $A_1, A_2, \dots, A_i$ . În final se generează de fapt un arbore nou având rădăcina  $R$ . Dacă  $i=0$ , atunci  $R$  este în același timp rădăcina și nod terminal.
6. **Radacina**( $A: \text{TipArbore}$ ):  $\text{TipNod}$ ; - operator care returnează nodul care este rădăcina arborelui  $A$  sau "nodul vid" dacă  $A$  este un arbore vid.
7. **Initializeaza**( $A: \text{TipArbore}$ ):  $\text{TipArbore}$ ; - crează arborele  $A$  vid.
8. **Insereaza**( $N: \text{TipNod}, A: \text{TipArbore}, T: \text{TipNod}$ ); - operator care realizează inserția nodului  $N$  ca fiu al nodului  $T$  în arborele  $A$ . În particular se poate preciza și poziția nodului în lista de fii ai tatălui  $T$  (prin convenție se acceptă sintagma "cel mai din dreapta fiu").
9. **Suprima**( $N: \text{TipNod}, A: \text{TipArbore}$ ); - operator care realizează suprimarea nodului  $N$  și a descendenților săi din arborele  $A$ . Suprimarea se poate defini diferențiat funcție de aplicația în care este utilizată structura de date în cauză.
10. **Inordine**( $A: \text{TipArbore}, \text{ProcesareNod}(\dots): \text{TipProcedura}$ ); - procedură care parcurge nodurile arborelui  $A$  în "inordine" și aplică fiecărui nod procedura de prelucrare  $\text{ProcesareNod}$ .
11. **Preordine**( $A: \text{TipArbore}, \text{ProcesareNod}(\dots): \text{TipProcedura}$ ); - procedură care parcurge nodurile arborelui  $A$  în "preordine" și aplică fiecărui nod procedura de prelucrare  $\text{ProcesareNod}$ .
12. **Postordine**( $A: \text{TipArbore}, \text{ProcesareNod}(\dots):$

*TipProcedura);* - procedură care parcurge nodurile arborelui A în "postordine" și aplică fiecărui nod procedura de prelucrare *ProcesareNod*.

-----

- Structura **arbore generalizat** este **importantă** deoarece apare frecvent în **practică**, spre exemplu arborii familiali, sau structura unei cărți defalcată pe capitole, secțiuni, paragrafe și subparagrafe.
- Din punctul de vedere al reprezentării lor în memorie, **arborii generalizați** au marele **dezavantaj** că **au noduri de grade diferite**, ceea ce conduce la **structuri neuniforme** de date sau la **structuri uniforme parțial utilizate**.

### 8.1.3. Traversarea arborilor generalizați

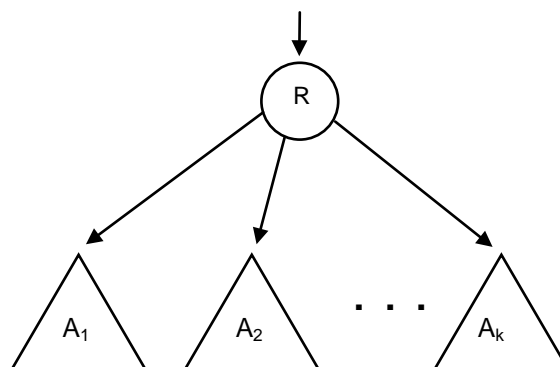
- Una din activitățile fundamentale care se execută asupra unei structuri arbore este **traversarea** acesteia.
- Ca și în cazul listelor liniare, prin **traversarea** unui arbore se înțelege execuția unei anumite operații asupra tuturor nodurilor arborelui.
- În timpul traversării, nodurile sunt **vizitate** într-o anumită **ordine**, astfel încât ele pot fi considerate ca și cum ar fi integrate într-o listă liniară.
- Descrierea și înțelegerea celor mai mulți algoritmi este mult ușurată dacă în cursul prelucrării se poate preciza **elementul următor** al structurii arbore, respectiv se poate **liniariza** structura arbore.
- În principiu tehnicile de traversare a arborilor generalizați se încadrează în **două** mari categorii:
  - (1) Tehnici bazate pe **căutarea în adâncime** ("depth-first search")
  - (2) Tehnici bazate pe **căutarea prin cuprindere** ("breadth-first search").
- Aceste tehnici își au sorginea în traversarea structurilor de date **graf**, dar prin particularizare sunt aplicabile și altor categorii de structuri de date, respectiv structuri de date **arbore generalizat**.

#### 8.1.3.1. Traversarea arborilor generalizați prin tehnici bazate pe căutarea în adâncime: preordine, inordine și postordine

- **Principiul căutării în adâncime (depth-first search)** presupune:
  - (1) Parcurgerea "**în adâncime**" a structurii, prin îndepărtare de punctul de pornire, până când acest lucru **nu** mai este posibil.



- (2) În acest moment se **revine** pe drumul parcurs până la proximal punct care permite o nouă posibilitate de înaintare în adâncime.
- (3) Procesul **continuă** în aceeași manieră până când structura de date este parcursă în întregime.
- Există trei moduri de **traversare (liniarizare)** care se referă la o structură de date arbore generalizat, bazate pe căutarea în adâncime și anume:
  - (1) **Traversarea în preordine**
  - (2) **Traversarea în inordine**
  - (3) **Traversarea în postordine**
- Cele trei modalități de traversare, se **definesc** de regulă în manieră **recursivă**, asemeni structurii de arbore și anume, **ordonarea** unui arbore se definește presupunând că **subarborii** săi s-au ordonat **deja**.
  - Cum subarborii au noduri strict mai puține decât arborele complet, rezultă că, aplicând recursiv de un număr suficient de ori definiția, se ajunge la ordonarea unui arbore vid, care este evidentă.
- Cele trei moduri de traversare ale unui arbore generalizat **A** se definesc recursiv după cum urmează:
  - Dacă arborele **A** este **nul**, atunci ordonarea lui **A** în preordine, inordine și postordine se reduce la **lista vidă**.
  - Dacă **A** se reduce la **un singur nod**, atunci nodul însuși, reprezintă traversarea lui **A**, în oricare din cele trei moduri.
  - Pentru restul cazurilor, fie arborele **A** cu rădăcina **R** și cu subarborii **A<sub>1</sub>, A<sub>2</sub>, . . . , A<sub>k</sub>**. (fig. 8.1.3.1.a):



**Fig.8.1.3.1.a.** Structură de arbore generalizat

- 1°. Traversarea în **preordine** a arborelui **A** presupune:
- Traversarea rădăcinii **R**
  - Traversarea în **preordine** a lui **A<sub>1</sub>**

- Traversarea în **preordine** a lui  $A_2, A_3$  și așa mai departe până la  $A_k$  inclusiv.

2°. Traversarea în **inordine** a arborelui **A** presupune:

- Traversarea în **inordine** a subarborelui  $A_1$
- Traversarea nodului **R**
- Traversările în **inordine** ale subarborilor  $A_2, A_3, \dots, A_k$ .

3°. Traversarea în **postordine** a arborelui **A** constă în:

- Traversarea în **postordine** a subarborilor  $A_1, A_2, \dots, A_k$
- Traversarea nodului **R**.

- **Structurile de principiu** ale procedurilor care realizează aceste traversări apar în secvența [8.1.3.1.a] iar un exemplu de **implementare generică** în secvența [8.1.3.1.b].

-----  
{Traversarea arborelui generalizat - modalități principale}

```
Preordine(A) : R, Preordine(A1), Preordine(A2), ...,
              Preordine(Ak).
Inordine(A)  : Inordine(A1), R, Inordine(A2), ...,
              Inordine(Ak). [8.1.3.1.a]
Postordine(A): Postordine(A1), Postordine(A2), ...,
              Postordine(Ak), R.
```

-----  
{Traversarea în **Preordine** a arborelui generalizat}

```
procedure Preordine(r: TipNod);
  *listeaza(r);
  pentru fiecare fiu f al lui r, (dacă există vreunul),
    în ordine de la stânga spre dreapta execută
    Preordine(f);
□
```

-----  
{Traversarea în **Inordine** a arborelui generalizat}

```
procedure Inordine(r: TipNod);
  dacă *r este nod terminal atunci *listează(r); [8.1.3.1.b]
  altfel
    Inordine(cele mai din stânga fiu al lui r);
    *listează(r);
    pentru fiecare fiu f al lui r, cu excepția celui
      mai din stânga, în ordine de la stânga spre
      dreapta execută
      Inordine(f);
  □
□
```

-----  
{Traversarea în **Postordine** a arborelui generalizat}

```

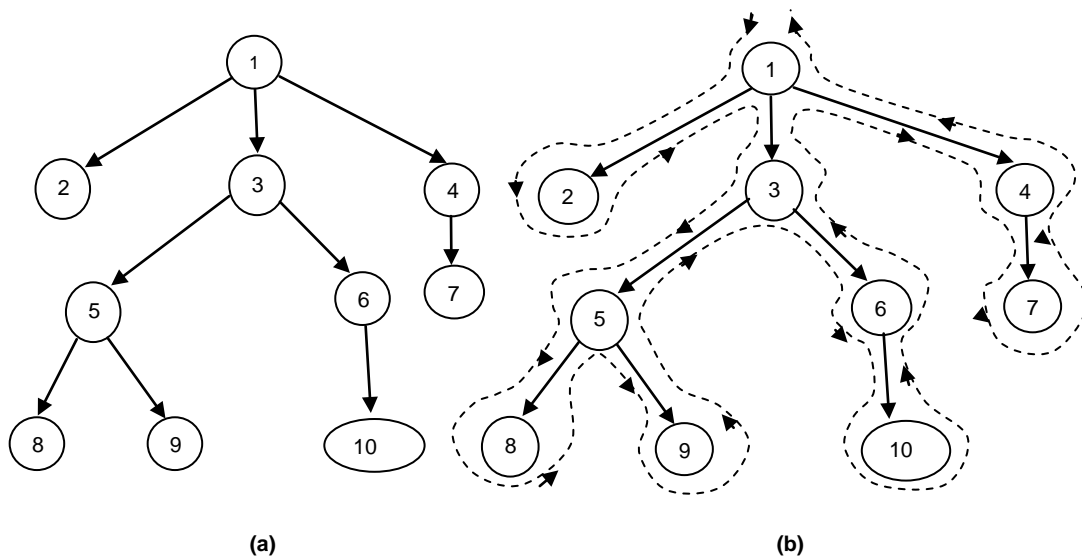
procedure Postordine(r: TipNod);
pentru fiecare fiu f al lui r, (dacă există vreunul),
    în ordine de la stânga spre dreapta execută
    Postordine(f);
□
*listeaza(r);

```

---

- Spre exemplu pentru arborele din figura 8.1.3.1.b (a), traversările anterior definite, conduc la următoarele secvențe de noduri:

- preordine: 1,2,3,5,8,9,6,10,4,7.
- postordine: 2,8,9,5,10,6,3,7,4,1.
- inordine: 2,1,8,5,9,3,10,6,7,4.



**Fig.8.1.3.1.b.** Traversarea unui arbore generalizat

- O metodă practică simplă de parcurgere a unui arbore este următoarea:
  - Dându-se o **structură arbore generalizat**, se imaginează parcurgerea acesteia în sens trigonometric pozitiv, rămânând cât mai aproape posibil de arbore (fig.8.1.3.1.b (b)).
  - Pentru **preordine**, nodurile se listează de **prima** dată când sunt întâlnite: 1, 2, 3, 5, 8, 9, 6, 10, 4, 7;
  - Pentru **postordine** nodurile se listează **ultima** dată când sunt întâlnite, respectiv când sensul de parcurgere este spre părinții lor: 2, 8, 9, 5, 10, 6, 3, 7, 4, 1;
  - Pentru **inordine**, un **nod terminal** se listează când este întâlnit **prima** oară, iar un **nod interior** când este întâlnit **a doua oară**: 2, 1, 8, 5, 9, 3, 10, 6, 7, 4.

---

□ **Exemplul 8.1.3.1.a.** Implementarea traversării în preordine a unui arbore utilizând operatorii definiți în cadrul TDA Arbore generalizat, varianta recursivă.

- Procedura din secvența următoare realizează tipărirea în **preordine** a cheilor nodurilor arborelui generalizat **A**
- Procedura este apelată prin următorul apel:  
TraversarePreordine(Radacina(A)).
- Se presupune că nodurile arborelui sunt de tip TipNod.

```

-----
PROCEDURE TraversarePreordine(r: TipNod);
{Implementare bazată pe setul de operatori TDA Arbore
generalizat}
{listează cheile descendenților lui r în preordine}

VAR f: TipNod;
BEGIN
    Write(Cheie(r,a));
    f:=PrimulFiu(r,a);
    WHILE f<>0 DO
        BEGIN
            TraversarePreordine(f);
            f:=FrateDreapta(f,a)
        END
    END; {Preordine}

```

□-----□

**Exemplul 8.1.3.1.b.** Implementarea traversării în preordine a unui arbore utilizând operatorii definiți în cadrul **TDA Arbore generalizat**, varianta nerecursivă.

- Se presupune că nodurile arborelui sunt de tip TipNod.
- În cadrul variantei nerecursive se utilizează **stiva** s, care conține elemente de TipNod.
- Când se ajunge la prelucrarea nodului n, stiva va conține memorat **drumul** de la rădăcină la nodul n, cu rădăcina la baza stivei și nodul n în vârf.
- Procedura care apare în secvența [8.1.3.1.d], are **două moduri de acțiune**.
  - (1) În **primul mod**, se descinde de-a lungul celui mai stâng drum neexplorat din cadrul arborelui, tipărind (**prelucrând**) nodurile întâlnite pe drum și **introducându-le** în stivă, până se ajunge la un nod terminal.
  - (2) Se trece în cel de-al **doilea mod** de operare care presupune revenirea pe drumul memorat în stivă, **eliminând** pe rând nodurile, până la întâlnirea primului nod care are frate dreapta.
  - (3) Se **revine** în primul mod și reîncepe descinderea cu acest frate încă neexplorat.
  - (4) Se continuă alternarea modurilor de parcurgere până la parcurgerea integrală a arborelui.
- Procedura începe în modul unu și se termină când stiva devine vidă.

```

-----
PROCEDURE TraversarePreordineNerecursiva(a: TipNod);
{Implementare nerecursivă bazată pe structura stivă}
{Se utilizează operatorii TDA Arbore generalizat, TDA Stiva}

```

```

VAR m: TipNod;
      s: TipStiva;
      gata: boolean;

```

```

BEGIN

```

```

    Initializează(s);

```

```

    m:=Radacina(a);

```

```

    gata:=false;

```

```

    WHILE NOT gata DO

```

```

        IF m<>0 THEN

```

```

            BEGIN

```

```

                Write(Cheie(m,a));

```

```

                Push(m,s);

```

```

                m:=PrimulFiu(m,a) {exploarează fiul stâng}

```

```

            END

```

```

        ELSE

```

```

            IF Stivid(s) THEN

```

```

                gata:=true

```

```

                [8.1.3.1.d]

```

```

            ELSE

```

```

                BEGIN

```

```

                    m:=Fratedreapta(VarfSt(s),a);

```

```

                    Pop(s)

```

```

                END

```

```

    END; {TraversarePreordineNerecursiva}

```

```

-----□

```

- Traversările în preordine și postordine ale unui arbore sunt utile în obținerea de **informații anterioare (ancestrale)** referitoare la noduri.
  - Astfel, se consideră ordinea liniară a nodurilor unui arbore generalizat parcurs în postordine.
  - Se definește  $POSTORDINE(n)$  ca fiind poziția exprimată ca număr întreg, a nodului  $n$  în această ordonare.
  - $POSTORDINE(n)$  se numește “**număr postordine al lui  $n$** ”.
  - Se definește de asemenea  $DESC(n)$  ca fiind egal cu numărul descendenților proprii ai nodului  $n$ .
    - Spre exemplu în fig.8.1.3.1.b, numerele postordine ale nodurilor "9", "6" și "1" sunt respectiv 3,6 și 10.
- Numerele **postordine** asignate nodurilor se bucură de următoarea **proprietate**:
  - Într-un arbore generalizat, nodurile oricărui subarbore având rădăcina  $n$  sunt numerotate **consecutiv** de la  $(POSTORDINE(n) - DESC(n))$  la  $POSTORDINE(n)$ .
- Astfel, pentru a verifica dacă un nod  $x$  este **descendentul** unui nod  $y$ , este suficient să se verifice condiția [8.1.3.1.e]:

-----  
POSTORDINE(y) - DESC(y) < POSTORDINE(x) < POSTORDINE(y) [8.1.3.1.e]  
-----

- O proprietate similară se poate defini și pentru ordonarea în **preordine**.

### 8.1.3.2. Traversarea arborilor generalizați prin tehnica căutării prin cuprindere

- Tehnica căutării prin **cuprindere** derivă tot din traversarea **grafurilor**, dar ea este utilizată, e adevărat mai rar, și la traversarea arborilor [Ko86].
- Se mai numește și traversare pe niveluri (“**level traversal**”) [We94] și este utilizată cu precădere în reprezentarea grafică a arborilor.
- **Principiul** acestei tehnici este simplu: nodurile nivelului  $i+1$  al structurii arbore sunt vizitate **doar** după ce au fost vizitate toate nodurile nivelului  $i$  ( $0 < i < h$ , unde  $h$  este înălțimea arborelui).
- Pentru implementarea acestei tehnici de parcurgere a arborilor generalizați, se utilizează drept structură de date suport, **structura coadă**.
- În secvența [8.1.3.2.a] apare schița de principiu a acestei traversări bazată pe **TDA Coadă**.

-----  
**procedure** TraversarePrinCuprindere(r: TipNod)  
{Implementare bazată pe TDA Coadă}  
    Coadă: TipCoadă;  
  
    **Initializeaza**(Coadă);  
    **daca** r **nu** este nodul vid **atunci**  
        **Adauga**(r,Coadă);     {procesul de amorsare}  
    **Cât timp** NOT Vid(Coadă) **execută**     [8.1.3.2.a]  
        r ← **Cap**(Coadă); **Scoate**(Coadă);  
        \*listeaza(r);  
        **pentru** fiecare fiu y al lui r, (dacă există vreunul),  
            în ordine de la stânga la dreapta **execută**  
            **Adauga**(y,Coadă);  
        □  
    □  
□-----

**Exemplul 8.1.3.2.** Implementarea **traversării prin cuprindere** a unui arbore utilizând operatorii definiți în cadrul **TDA Arbore generalizat** și **TDA Coadă** este ilustrată în secvența [8.1.3.2.b]. Nodurile vizitate sunt afișate.

-----  
**PROCEDURE** TraversarePrinCuprindere(r: TipNod);  
{Implementare bazată pe TDA Arbore generalizat, TDA Coadă}  
  
    **VAR** Coadă: TipCoadă;  
        f: TipNod;  
    **BEGIN**  
        **Initializeaza**(Coadă);  
        **Adauga**(r,Coadă);  
        **WHILE** NOT Vid(Coadă) **DO**

```

BEGIN
    r:=Cap(Coadă); Scoate(Coadă);
    WRITE(Cheie(r));
    f:=PrimulFiu(r);
    IF f<> NIL THEN [8.1.3.2.b]
        BEGIN
            Adauga(f,Coadă);
            f:=FrataDreapta(f)
            WHILE f<> NIL DO
                BEGIN
                    Adauga(f,Coadă);
                    f:=FrataDreapta(f)
                END
            END
        END
    END
END; {TraversarePrinCuprindere}

```

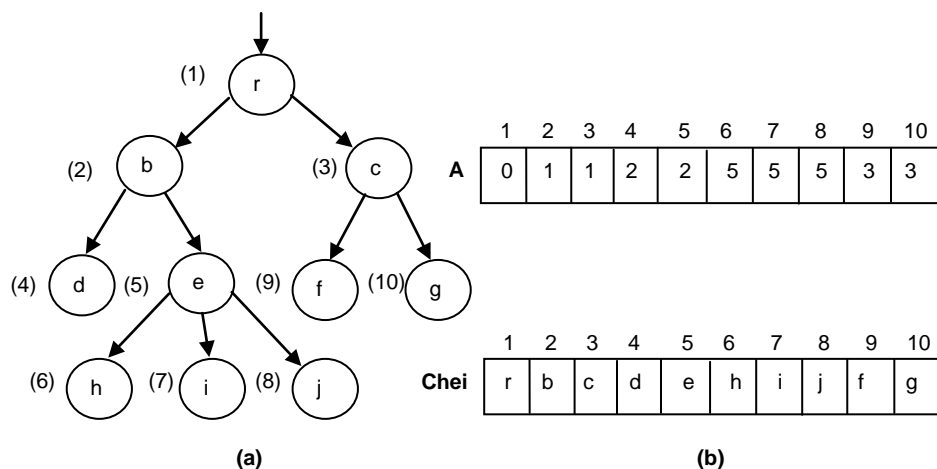
---

#### 8.1.4. Tehnici de implementare a TDA arbore generalizat

- În cadrul acestui subcapitol se vor prezenta câteva din **implementările** posibile ale **structurii arbore generalizat**, corelate cu aprecieri referitoare la capacitatea acestora de a suporta operatorii definiți în cadrul **TDA Arbore generalizat**.

##### 8.1.4.1. Implementarea arborilor generalizați cu ajutorul tablourilor

- Se bazează pe următoarea **tehnologie**:
  - Fie A un arbore generalizat cu n noduri.
  - Se numerotează nodurile arborelui A de la 1 la n.
  - Se asociază nodurilor arborelui elementele unui tablou A, astfel încât nodului i al arborelui îi corespunde locația A[i] din tablou.
  - Cea mai simplă manieră de reprezentare a unui arbore, care permite implementarea operației **Tata**, este cea conform căreia fiecare intrare A[i] din tablou memorează un indicator (cursor) la **părintele** nodului i.
  - Rădăcina se poate distinge prin valoarea zero a cursorului.
  - Cu alte cuvinte, dacă A[i] = j atunci nodul j este **părintele** nodului i iar dacă A[i]= 0, atunci nodul i este **rădăcina** arborelui.
- Acest mod de reprezentare, face uz de proprietatea arborilor care stipulează că orice nod are **un singur părinte**, motiv pentru care se numește și "**indicator spre părinte**".
- Găsirea părintelui unui nod se face într-un interval **constant** de timp **O(1)** iar parcurgerea arborelui în direcția nod – rădăcină, se realizează într-un interval de timp proporțional cu adâncimea nodului.



**Fig. 8.1.4.1.a.** Reprezentarea arborilor generalizați cu ajutorul tablourilor (varianta "indicator spre părinte")

- Această reprezentare poate implementa simplu și operatorul **Cheie** dacă se adaugă un alt tablou **Cheie**, astfel încât **Cheie[i]** este cheia nodului **i**, sau dacă elementele tabloului **A** se definesc de tip articol având câmpurile **cheie** și **cursor**.
- În figura 8.1.4.1.a apare o astfel de reprezentare (b) a arborelui generalizat (a).
- Reprezentarea "indicator spre părinte" are însă **dezavantajul** implementării dificile a operatorilor referitori la fii.
  - Astfel, unui nod dat **n**, **i** se determină cu dificultate fiii sau înălțimea.
  - În plus, reprezentarea "indicator spre părinte", **nu** permite specificarea ordinii fiilor unui nod, astfel încât operatorii **PrimulFiu** și **FrateDreapta** **nu** sunt bine precizați.
  - Pentru a da acuratețe reprezentării, se poate impune o **ordine artificială** a nodurilor în tablou, respectând următoarele convenții:
    - (a) - numerotarea fiilor unui nod se face numai după ce nodul a fost numărat;
    - (b) - numerele fiilor cresc de la stânga spre dreapta. Nu este necesar ca fiii să ocupe poziții adiacente în tablou.
- În accepțiunea respectării acestor convenții, în secvența [8.1.4.1.a] apare redactată funcția **FrateDreapta**.
- Tipurile de date presupuse sunt cele precizate în antetul procedurii. Se presupune că **nodul vid** este reprezentat prin zero.



{Implementarea Arborilor generalizați cu ajutorul tablourilor  
variantea "Indicator spre părinte"}

```

TYPE TipNod=integer;
      TipArbore=ARRAY[1..maxnod] OF TipNod;

```

{Exemplu de implementare a operatorului *FrateDreapta*}

```

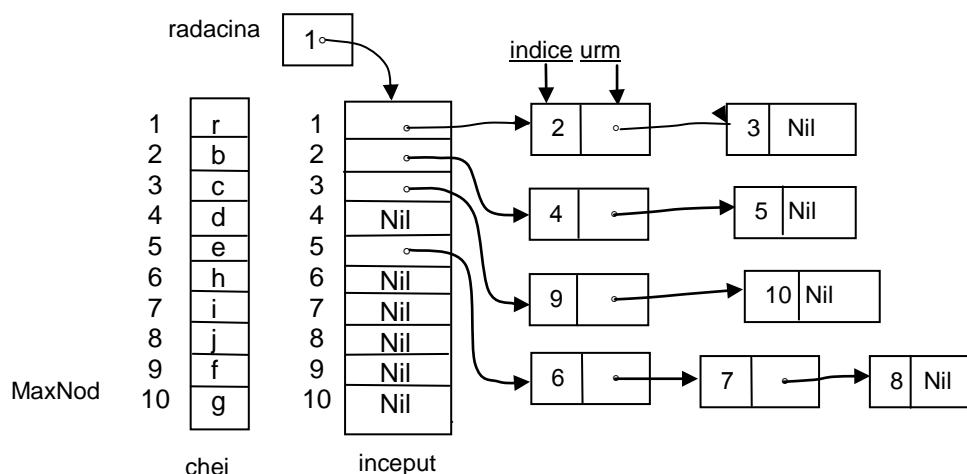
FUNCTION FrateDreapta(n: TipNod; a: TipArbore): TipNod;
  VAR i,parinte: TipNod;
  BEGIN
    parinte:=a[n];
    FrateDreapta:=0;
    i:=n;
    REPEAT
      i:=i+1;
      IF a[i]=parinte THEN FrateDreapta:=i
    UNTIL(FrateDreapta<>0) OR (i=maxnod)
  END; {FrateDreapta}

```

---

#### 8.1.4.2. Implementarea arborilor generalizați cu ajutorul listelor

- O manieră importantă și utilă de implementare a arborilor generalizați este aceea de a crea pentru **fiecare nod** al arborelui o **listă** a fiilor săi.
- Datorită faptului că numărul fiilor poate fi **variabil**, o variantă potrivită de implementare o reprezintă utilizarea **listelor înlănțuite**.
- În fig.8.1.4.2.a se sugerează maniera în care se poate implementa arborele din figura 8.1.4.1.a.(a).
  - Se utilizează un **tablou** (inceput) care conține câte o locație pentru fiecare nod al arborelui.
  - Fiecare element al tabloului inceput este o referință la o **listă înlănțuită simplă**, ale cărei elemente sunt nodurile fii ai nodului corespunzător intrării, adică elementele listei indicate de inceput[i] sunt fiii nodului i.



- În continuare se prezintă două posibilități de implementare.
- Pentru prima se sugerează tipurile de date prevăzute în secvența [8.1.4.2.a]. Este vorba despre o implementare bazată pe liste înlanțuite.
  - Se presupune că rădăcina arborelui este memorată în câmpul specific *radacina*.
  - Nodul vid se reprezintă prin valoarea NIL.
  - În aceste condiții în secvența [8.1.4.2.b] apare implementarea operatorului *PrimulFiu*.

---

```

{Reprezentarea arborilor generalizați utilizând liste
înlanțuite simple implementate cu ajutorul pointerilor}
TYPE TipPointreNod=^TipNodList;
      TipNodList=RECORD
                indice:1..MaxNod;
                urm    :TipPointerNod
      END;
      TipNod=0..MaxNod;                                [8.1.4.2.a]
      TipLista=TipPointerNod;
      TipArbore=RECORD
        inceput:ARRAY[1..MaxNod] OF TipLista;
        {chei:ARRAY[1..MaxNod] OF TipCheie;}
        radacina:TipNod
      END;

```

---

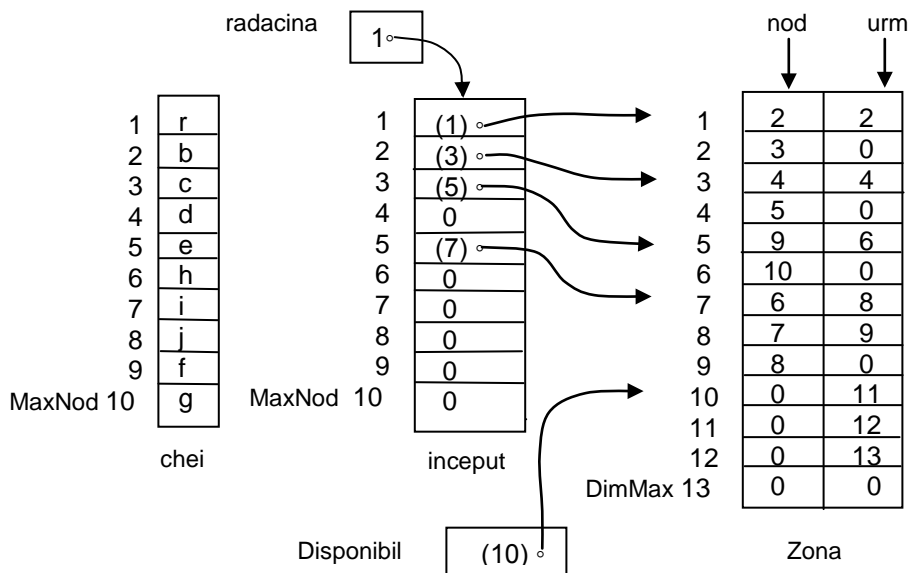
```

{Exemplu de implementare al operatorului PrimulFiu}
{se utilizează TDA Listă, varianta restrânsă}
FUNCTION PrimulFiu(n: TipNod; a: TipArbore): TipNod;
  VAR l: TipLista;
  BEGIN
    l:=a.inceput[n];                                [8.1.4.2.b]
    IF Fin(l) THEN {n este un nod terminal}
      PrimulFiu:=0
    ELSE
      PrimulFiu:=Furnizeaza(Primul(l),l)
    END; {PrimulFiu}

```

---

- Cea de-a doua modalitate de implementare se bazează pe **implementarea listelor** cu ajutorul **cursorilor** (Vol.1 &6.3.3).
  - În această reprezentare atât listele ca atare cât și înlanțuirile din cadrul lor sunt întregi utilizați ca și cursori într-o tabelă de elemente (Zona).
  - În fig.8.1.4.2.b se prezintă maniera în care se poate implementa arborele generalizat din figura 8.1.4.1.a.(a).
  - Se utilizează și în acest caz tabloul *inceput* care conține referințe la listele înlanțuite memorate în tabloul *Zona*, liste ale căror elemente sunt nodurile fii ai nodului corespunzător intrării.



**Fig.8.1.4.2.b.** Reprezentarea arborilor generalizați cu ajutorul listelor înlănțuite implementate cu ajutorul cursorilor

- În secvența [8.1.4.2.c] apar structurile de date aferente acestei implementări, iar în secvența [8.1.4.2.d] implementările operatorilor **PrimulFiu** și **Tata**.
- După cum se remarcă, implementarea operatorului **Tata** se realizează mai dificil, întrucât trebuie realizată o baleere a tuturor listelor de fii în vederea stabilirii cărei liste îi aparține nodul furnizat ca parametru.

---

{Reprezentarea arborilor generalizați utilizând liste  
înlănțuite implementate cu ajutorul cursorilor}

```

TYPE Cursor=0..DimMax;
      TipNod=0..MaxNod;
      TipArbore=RECORD
        inceput:ARRAY[1..MaxNod] OF Cursor;
        {chei:ARRAY[1..MaxNod] OF TipCheie;
        radacina:TipNod
      END;
VAR Zona:ARRAY[1..DimMax] OF RECORD
        nod:TipNod;
        urm:Cursor
      END;

      Disponibil:Cursor;
      A:TipArbore;
  
```

[ 8.1.4.2.c ]

---

{Exemplu de implementare al operatorilor **PrimulFiu** și **Tata**}

```

FUNCTION PrimulFiu(n: TipNod; a: TipArbore): TipNod;
  VAR l:Cursor; {cursor la începutul listei fiilor lui n}
  BEGIN
    l:=a.inceput[n];
    IF l=0 THEN {n este nod terminal}
      PrimulFiu:=0
    ELSE
      PrimulFiu:=Zona[l].nod
    END; {PrimulFiu}
  
```

```

FUNCTION Tata(n: TipNod; a: TipArbore): TipNod;
  VAR p:TipNod; {parcurge tații posibili ai lui n}
        i:Cursor; {parcurge copiii lui p}
  BEGIN
    Tata:=0; p:=0;
    REPEAT [8.1.4.2.d]
      p:=p+1; i:=a.inceput[p];
      WHILE i<>0 AND (Tata=0) DO {verifică dacă n este
                                   printre fiii lui p}

        IF Zona[i].nod=n THEN
          Tata:=p
        ELSE
          i:=Zona[i].urm
      UNTIL (Tata<>0) OR (p=MaxNod)
  END; {Tata}

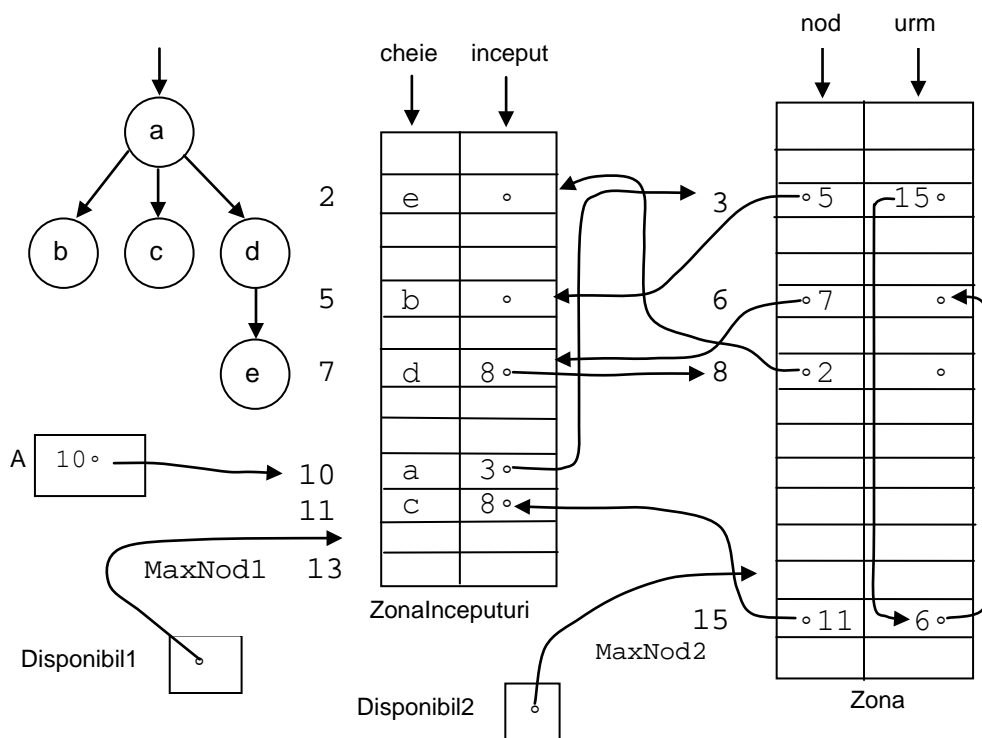
```

---

### 8.1.4.3. Implementarea structurii arbore generalizat pe baza relațiilor "primul-fiu" și "frate-dreapta"

- Implementările structurilor de arbori generalizați, descrise până în prezent, printre alte **dezavantaje**, îl au și pe acela de a **nu** permite implementarea simplă a operatorului **Creaza** și deci de a **nu** permite dezvoltarea facilă a unor **structuri complexe** pornind de la **structuri simple**.
- Această dificultate provine din faptul că, în timp ce, **spre exemplu** în reprezentarea bazată pe **cursori**, **listele fiilor** partajează o singură zonă comună (Zona), fiecare arbore are propriul său tablou de începuturi (inceput).
- Astfel, pentru a implementa spre **exemplu** operatorul **Creaza<sub>2</sub>**(*v:TipCheie*, *A<sub>1</sub>,A<sub>2</sub>:TipArbore*):*TipArbore* este necesar ca arborii A<sub>1</sub> și A<sub>2</sub> să fie recopiați într-un al treilea arbore, al cărui tablou de începuturi conține tablourile celor doi arbori, completat cu locația corespunzătoare lui v indicând o listă de fii care conține rădăcinile lui A<sub>1</sub> și A<sub>2</sub>.
- Pentru a depăși această dificultate, în structura de date implementată în paragraful anterior cu ajutorul cursorilor, se fac două modificări:
  - (1) Tabloul inceput se înlocuiește cu un tablou care cuprinde începuturile listelor de fii pentru **toate nodurile** corespunzătoare tuturor arborilor luați în considerare (ZonaInceputuri din fig.8.1.4.3.a).
    - Elementele acestui tablou sunt articole cu două câmpuri: cheie și inceput.
    - Câmpul cheie identifică un nod al arborelui iar inceput este un cursor care indică lista fiilor săi memorată în tabloul Zona.
    - Acest cursor materializează de fapt relația “**primul-fiu**”.
  - (2) Nodurile **nu** mai sunt numerotate în ordinea 1,2,...,n (ca în tabela inceput) ci ele sunt reprezentate printr-un indice arbitrar în ZonaInceputuri.
    - Din acest motiv, este normal ca în cadrul tabloului Zona, câmpul nod să **nu** mai indice direct "numărul" unui nod, ci el să aibă valoarea unui cursor în ZonaInceputuri, valoare care indică **poziția** acelui nod.

- Câmpul urm din cadrul aceluiași tablou realizează înlănțuirea nodurilor fii și materializează relația “frate-dreapta”.
- În această reprezentare TipAbore este de fapt un cursor în ZonaInceputuri.
- În figura 8.1.4.3.a este prezentată structura de date aferentă arborelui generalizat reprezentat în colțul stânga sus al figurii.



**Fig.8.1.4.3.a.** Reprezentarea unui arbore generalizat cu ajutorul relațiilor “primul-fiu” și “frate-dreapta” (Varianta 1)

- Cheile nodurilor sunt a,b,c,d și e cărora li s-au atribuit prin mecanismul de alocare pozițiile 10,5,11,7 respectiv 2 în tabloul ZonaInceputuri.
- Printr-un mecanism similar au fost atribuite și locațiile tabloului Zona.
- În acest scop, cele două tablouri dispun de **liste de disponibili specifice** (Disponibil1 respectiv Disponibil2) care înlățuite locațiile libere și din care se face alocarea spațiului necesar nodurilor.
- Evident, în aceleași liste sunt înlănțuite nodurile disponibilizate.
- Din punct de vedere formal, structurile de date aferente acestei reprezentări apar în secvența [8.1.4.3.a].

---

{Reprezentarea arborilor generalizați bazată pe relațiile  
primul-fiu și frate-dreapta (Varianta 1)}

```

TYPE TipIndicatorInceputuri=0..MaxNod1;
        TipIndicatorZona=0..MaxNod2;
        TipArbore=TipIndicatorInceputuri;

```

[8.1.4.3.a]

```

VAR ZonaInceputuri:ARRAY[1..MaxNod1] OF RECORD
    cheie:TipCheie;
    inceput:TipIndicatorZona
END;

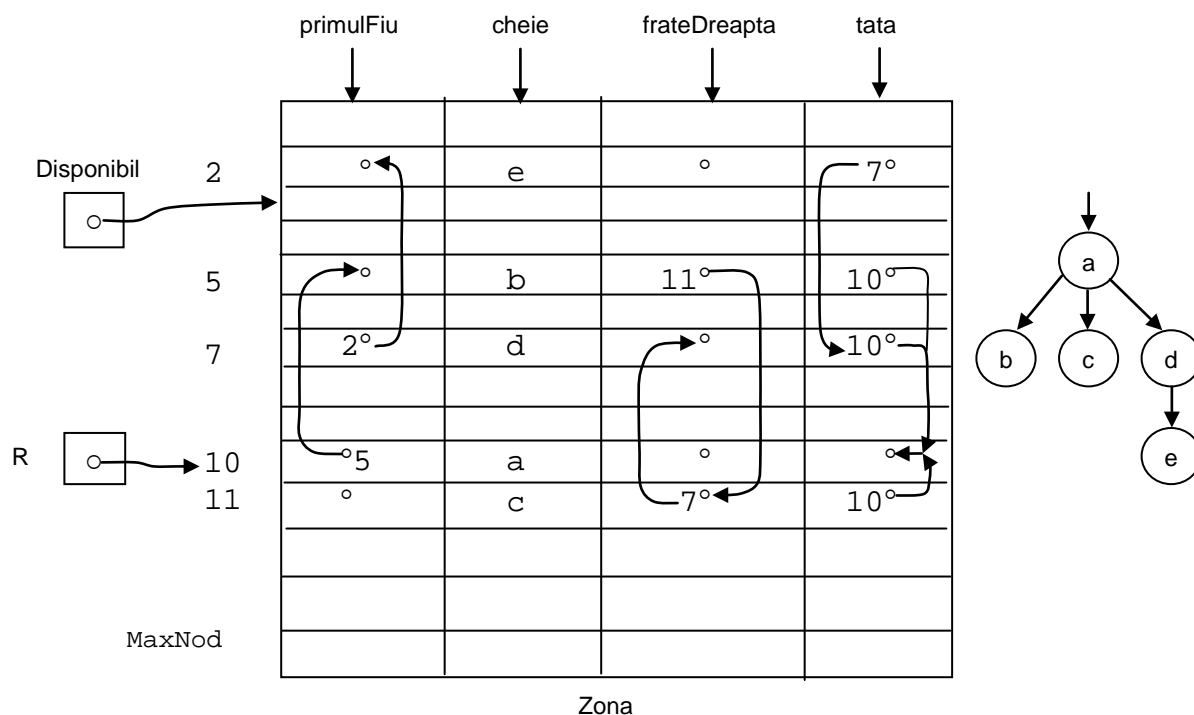
Zona:ARRAY[1..MaxNod2] OF RECORD
    nod:TipIndicatorInceputuri;
    urm:TipIndicatorZona
END;

Disponibil1:TipIndicatorInceputuri;
Disponibil2:TipIndicatorZona;
A:TipArbore;

```

---

- Structura de date poate fi utilizată în implementarea eficientă a operatorului **Creaza** întrucât permite **combinarea** simplă a structurilor de arbori.
- În cadrul acestei structuri, câmpul urm al tabloului Zona, precizează chiar fratele dreapta al nodului în cauză.
- De asemenea, fiind dat un nod al cărui cursor în ZonaInceputuri este i, câmpul ZonaInceputuri[i].inceput indică cursorul în spațiul Zona al primului fiu al lui i. Fie j valoarea acestui cursor. În continuare cheia primului fiu se poate afla simplu știind că Zona[j].nod reprezintă cursorul acestui nod în ZonaInceputuri.
- Pentru a **simplifica** și mai mult lucrurile se poate renunța la tabloul ZonaInceputuri.
- Astfel, în continuare un nod **nu** va mai fi identificat prin cursorul său în ZonaInceputuri ci prin indexul celulei pe care el o ocupă în tabloul Zona, cu alte cuvinte câmpul cheie din ZonaInceputuri va înlocui câmpul nod din Zona.
- În aceste condiții, câmpul urm devine frateDreapta și va indica direct fratele dreapta al nodului respectiv.
- Pentru a se păstra informația referitoare la primul fiu, Zona se prelungește cu un câmp auxiliar numit primulFiu.



**Fig.8.1.4.3.b.** Reprezentarea unui arbore generalizat cu ajutorul relațiilor “primul-fiu” și “frate-dreapta” (Varianta 2)

- O structură de date încadrată în TipArbore, este desemnată în aceste condiții printr-un **cursor** în tabloul Zona, cursor care indică nodul rădăcină al arborelui.
- Acest mod de reprezentare este însă impropriu implementării operatorului **Tata**, care presupune balearea integrală a tabloului Zona.
- Pentru a implementa în mod eficient și acest operator se sugerează **adăugarea** unui al 4-lea câmp (tata), care să indice direct părintele fiecărui nod .
- În fig.8.1.4.3.b, apare reprezentarea arborelui generalizat din dreapta figurii, iar în secvența [8.1.4.3.b] apare definiția formală a structurii de date corespunzătoare acestui mod de implementare al arborilor generalizați.

---

{Reprezentarea arborilor generalizați bazată pe relațiile  
primul-fiu și frate-dreapta (Varianta 2)}

```
TYPE TipCursor=0..MaxNod;
    TipArbore=TipCursor;                                [8.1.4.3.b]
VAR  Zona:ARRAY[1..MaxNod] OF RECORD
                                primulFiu:TipCursor;
                                cheie:TipCheie;
                                frateDreapta:TipCursor;
                                tata:TipCursor
                                END;
    Disponibil:TipCursor;
    R:TipArbore;
```

---

- În secvența [8.1.4.3.c] este prezentat un exemplu de implementare a operatorului **Creaza<sub>2</sub>**, pornind de la reprezentarea propusă.
- Se presupune că există o listă a liberilor în tabloul Zona, indicată prin cursorul Disponibil, în cadrul căreia elementele sunt înlanțuite prin intermediul câmpului frateDreapta.

---

{Exemplu de implementare al operatorului **Creaza<sub>2</sub>**}

```
FUNCTION Creaza2(v:TipCheie; t1,t2:TipArbore):TipArbore;
VAR temp:TipCursor; {păstrează indexul primei locații
                    disponibile pentru radacina noului arbore}
BEGIN
    temp:=Disponibil;
    Disponibil:=Zona[Disponibil].frateDreapta;
    Zona[temp].primulFiu:=t1;                                [8.1.4.3.c]
    Zona[temp].cheie:=v;
    Zona[temp].frateDreapta:=0;   Zona[temp].tata:=0;
    Zona[t1].frateDreapta:=t2;   Zona[t1].tata:=temp;
    Zona[t2].frateDreapta:=0;   Zona[t2].tata:=temp;
    Creaza2:=temp
END; {Creaza2}
```

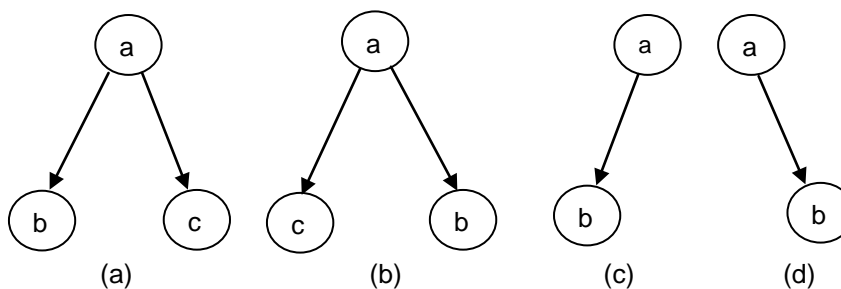
---

- Pentru a utiliza mai puțin spațiu (dar mai mult timp), se poate renunța la câmpul `tata` și în locul valorii zero care apare în câmpul `frateDreapta` al celui mai din dreapta fiu (ultimul), se poate poziționa un **cursor** la părintele său.
- Pentru a evita confuziile, mai este nevoie de un câmp indicator pentru fiecare nod care să precizeze dacă câmpul `frateDreapta` indică **fratele** sau **părintele**.
- În aceste circumstanțe, fiind dat un nod oarecare, tatăl său se poate afla parcurgând frații săi pe relația frate-dreapta. Ultimul dintre aceștia își indică tatăl.
- Timpul necesar acestei activități depinde de poziția nodului în lista fraților.

## 8.2. Arbori binari

### 8.2.1. Definiții

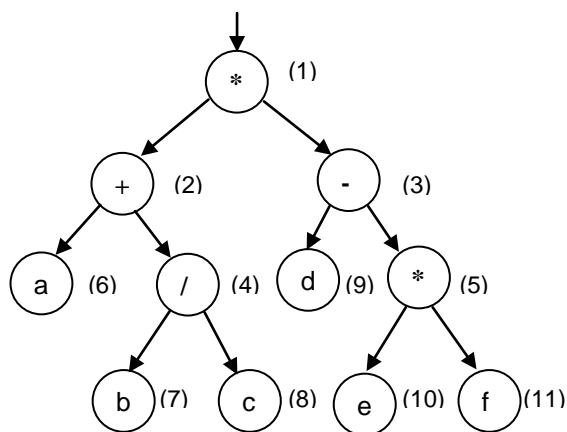
- Prin arbore binar se înțelege o mulțime de  $n \geq 0$  noduri care dacă nu este vidă, conține un anumit nod numit **rădăcină**, iar restul nodurilor formează doi arbori binari disjuncți numiți: **subarboarele stâng** respectiv **subarboarele drept**.
- Ca și exemple pot fi considerați arborii binari reprezentați în figura 8.2.1.a.



**Fig.8.2.1.a.** Structuri de arbori binari

- Structurile (a) și (b) din fig.8.2.1.a. deși conțin noduri identice, reprezintă arbori binari **diferiți** deoarece se face o distincție netă între subarboarele stâng și subarboarele drept al unui arbore binar.
- Acest lucru este pus și mai pregnant în evidență de structurile (c) și (d) din fig.8.2.1.a. care de asemenea reprezintă arbori **diferiți**.
  - O **expresie aritmetică** obișnuită poate fi reprezentată cu ajutorul unui arbore binar întrucât operatorii aritmetici sunt operatori binari..
  - Spre exemplu, pentru expresia  $(a+b/c)*(d-e*f)$ , arborele binar corespunzător care se mai numește și arbore de parcurgere ("**parse tree**"), apare în figura 8.2.1.b.



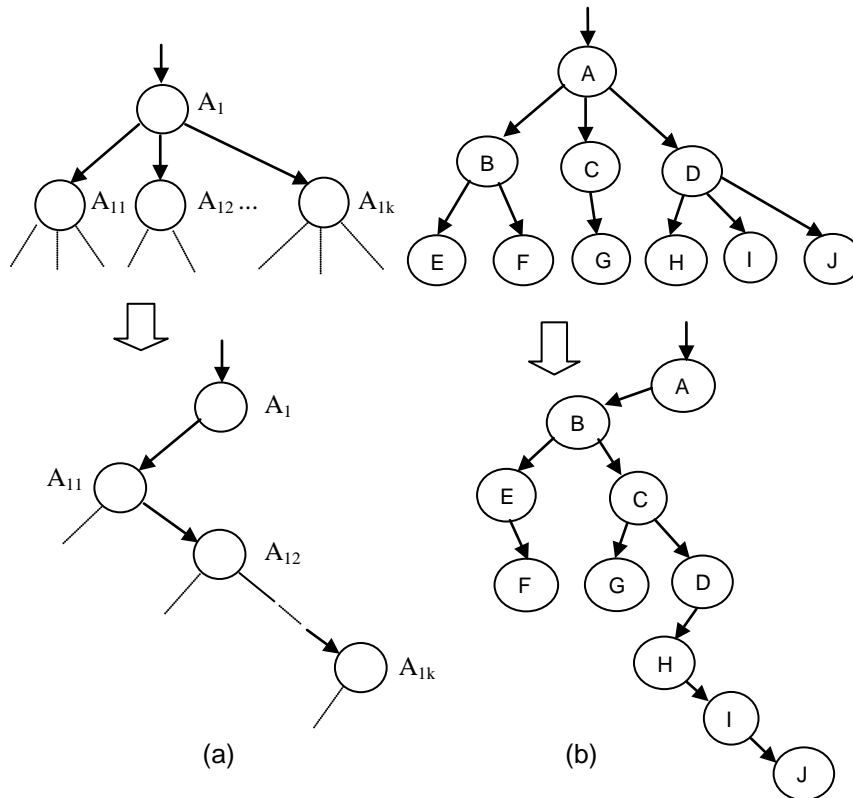


**Fig.8.2.1.b.** Arbore binar asociat unei expresii aritmetice

- Structura **arbore binar** este deosebit de importantă deoarece:
  - (1) Este simplu de **reprezentat**
  - (2) Este ușor de **prelucrat**, bucurându-se de o serie de proprietăți specifice
  - (3) **Orice** structură **arbore**, poate fi transformată într-o structură **arbore binar**.

### 8.2.2. Tehnica transformării unei structuri arbore generalizat într-o structură arbore binar.

- Fie un **arbore generalizat** oarecare  $A$ , care are rădăcina  $A_1$  și subarborii  $A_{11}, A_{12}, \dots, A_{1k}$ .
- **Transformarea** acestuia într-un **arbore binar** se realizează după cum urmează:
  - (1) Se ia  $A_1$  drept rădăcină a arborelui binar,
  - (2) Se face subarborii  $A_{11}$  fiul său stâng,
  - (3) Fiecare subarbore  $A_{1i}$  se face fiul drept al lui  $A_{1,i-1}$  pentru  $2 \leq i \leq k$ .
  - (4) Se continuă în aceeași manieră transformând după același algoritm fiecare din subarborii rezultați, până la parcurgerea integrală a arborelui inițial.
- Grafic, această tehnică apare reprezentată în figura 8.2.2.a, (a)-cazul general și (b)-un exemplu.



**Fig.8.2.2.a.** Transformarea unui arbore oarecare în arbore binar

### 8.2.3. TDA Arbore binar

- Tipul de date abstract **arbore binar**, ca de altfel orice TDA presupune:
  - (1) Definirea **modelului matematic** asociat structurii arbore binar
  - (2) Definirea **setului de operatori** care gestionează structura.
- Ca și în cadrul altor TDA-uri, este greu de stabilit un set de operatori care să ofere satisfacție în toate clasele de aplicații posibile.
- În cadrul cursului de față se propune pentru **TDA Arbore binar** forma prezentată în [8.2.3.a].

---

#### TDA Arbore binar

**Modelul matematic:** o structură de noduri. Toate nodurile sunt de același tip. Fiecare nod este rădăcina unui subarbore și este alcătuit din trei câmpuri: element, indicator stâng și indicator drept. Indicatorii precizează subarboarele stâng respectiv subarboarele drept al subarboarelui în cauza. În cadrul câmpului element poate exista un câmp cheie care identifică nodul.

**Notatii:**

*TipNod* - tipul asociat nodurilor arborelui  
*TipArboreBinar* - tipul arbore binar  
*TipIndicatorNod* - indicator la nod

*t,s,d: TipArboreBinar;*  
*r,n: TipIndicatorNod;*  
*w: TipNod;*  
*b: boolean;*

[8.2.3.a]

**Operatori:**

1. **CreazaArboreVid**(*t: TipArboreBinar*); - procedură care creează arborele vid *t*;
2. **ArboreVid**(*t: TipArboreBinar*): *boolean*; - funcție care returnează valoarea adevărat dacă arborele *t* este vid și fals în caz contrar.
3. **Radacina**(*n: TipIndicatorNod* ): *TipIndicatorNod*; - returnează indicatorul rădăcinii arborelui binar căruia îi aparține nodul *n*;
4. **Parinte**(*n: TipIndicatorNod, t:TipArboreBinar*) :*TipIndicatorNod*; - returnează indicatorul Părintelui (tatălui) nodului *n* aparținând arborelui *t*;
5. **FiuStanga**(*n: TipIndicatorNod,t:TipArboreBinar*) :*TipIndicatorNod*; - returnează indicatorul fiului stâng al nodului *n* aparținând arborelui *t*;
6. **FiuDreapta**(*n: TipIndicatorNod,t: TipArboreBinar*): *TipIndicatorNod*; - returnează indicatorul fiului drept al nodului *n* aparținând arborelui *t*;
7. **SuprimaSub**(*n: TipIndicatorNod,t: TipArboreBinar*); - suprimă subarborele a cărui radacină este nodul *n*, aparținând arborelui binar *t*;
8. **InlocuiesteSub**(*n: TipIndicatorNod, r,t: TipArboreBinar*); - inserează arborele binar *r* în *t*, cu rădăcina lui *r* localizată în poziția nodului *n*, înlocuind subarborele indicat de *n*. Operatorul se utilizează de regulă când *n* este o frunză a lui *t*, caz în care poate fi asimilat cu operatorul *adauga*;
9. **Creaza2**(*s,d: TipArboreBinar,r: TipIndicatorNod*): *TipArboreBinar*; - creează un arbore binar nou care are nodul *r* pe post de rădăcină și pe *s* și *d* drept subarbore stâng respectiv subarbore drept
10. **Furnizeaza**(*n: TipIndicatorNod,t: TipArboreBinar*): *TipNod*; - returnează conținutul nodului indicat de *n* din arborele binar *t*. Se presupune că *n* există în *t*;

11. **Actualizeaza**(*n*: TipIndicatorNod, *w*: TipNod, *t*: TipArboreBinar); - înlocuiește conținutul nodului indicat de *n* din *t* cu valoarea *w*. Se presupune că *n* există.
  12. **Cauta**(*w*: TipNod, *t*: TipArboreBinar): TipIndicatorNod;  
returnează indicatorul nodului aparținând arborelui binar *t*, al cărui conținut este *w*;
  13. **Preordine**(*t*: TipArboreBinar, Vizitare(*listaArgumente*));  
- operator care realizează parcurgerea în preordine a arborelui binar *t* aplicând fiecărui nod, procedura *Vizitare*;
  14. **Inordine**(*t*: TipArboreBinar, Vizitare(*listaArgumente*)); -  
operator care realizează parcurgerea în inordine a arborelui binar *t* aplicând fiecărui nod, procedura *Vizitare*;
  15. **Postordine**(*t*: TipArboreBinar, Vizitare(*listaArgumente*));  
- operator care realizează parcurgerea în postordine a arborelui binar *t* aplicând fiecărui nod, procedura *Vizitare*.
- 

#### 8.2.4. Tehnici de implementare a arborilor binari

- În aplicațiile curente se utilizează **două** modalități de implementare a structurii arbore binar:
  - (1) Implementarea bazată pe **tablouri**, (mai rar utilizată)
  - (2) Implementarea bazată pe **pointeri**.

##### 8.2.4.1. Implementarea arborilor binari cu ajutorul structurii tablou

- Pentru implementarea unui arbore binar se poate utiliza o structură **tablou liniar** definit după cum urmează [8.2.4.1.a]:

---

{Implementarea unui arbore binar cu ajutorul unei structuri tablou liniar}

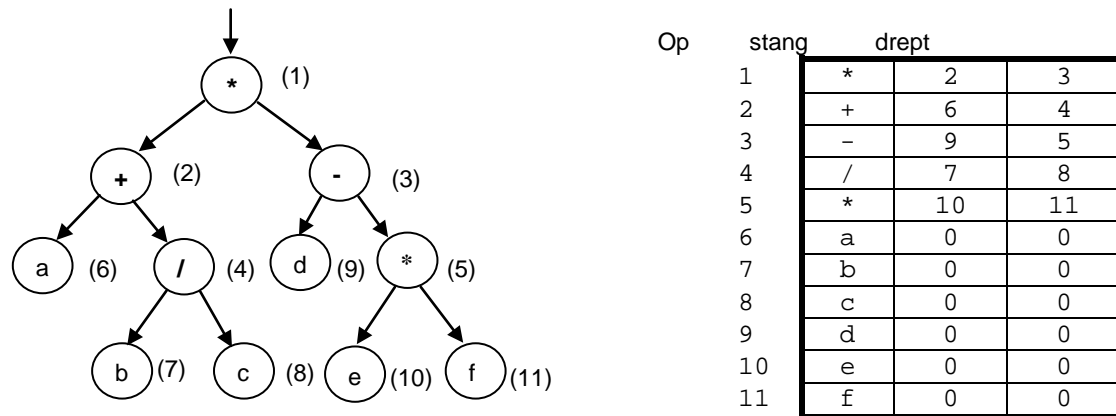
```

TYPE TipCursor=0..MaxNod;
      TipElement=RECORD
                    Op:char;                                [8.2.4.1.a]
                    stang,drept:TipCursor
      END;
      TipArbore=ARRAY[TipCursor] OF TipElement;
VAR T:TipArbore;
  
```

---

- În prealabil fiecărui nod al arborelui *i* se asociază în mod **aleator** o intrare în tabloul liniar.

- În principiu, acesta este o implementare bazată pe **cursori**.
- În fig.8.2.4.1.a apare o astfel de reprezentare a unui arbore binar.

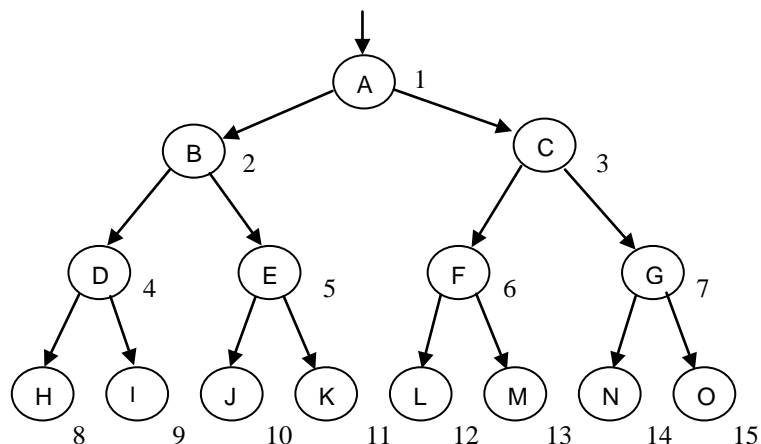


**Fig.8.2.4.1.a.** Reprezentarea unui arbore binar cu ajutorul unui tablou liniar

- Un alt mod de reprezentare cu ajutorul structurilor tablou se bazează pe următoarele două **leme**.
- **Lema 1.** Numărul maxim de noduri al nivelului  $i$  al unui arbore binar este  $2^{i-1}$ .
- Ca atare, numărul maxim de noduri al unui arbore de înălțime  $h$  este [8.2.4.1.b].

$$NrMax = \sum_{i=1}^h 2^{i-1} = 2^h - 1 \quad \text{pentru } h > 0 \quad [8.2.4.1.b]$$

- Arborele binar de înălțime  $h$  care are exact  $2^h - 1$  noduri se numește **arbore binar plin de înălțime  $h$** .
- Se **numeratează** secvențial nodurile unui arbore binar plin de înălțime  $h$ , începând cu nodul situat pe **nivelul 1** și continuând număratoarea nivel cu nivel de **sus în jos**, și în cadrul fiecărui nivel, de la **stânga la dreapta**.
- Pornind de la această numerotare se poate realiza o **implementare elegantă** a structurii arbore binar, **asociind** fiecărui nod al arborelui, locația corespunzătoare numărului său într-un **tablou liniar** (fig.8.2.4.1.b.).

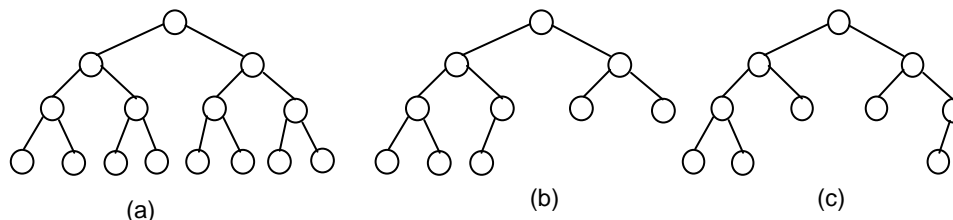


# Arbore

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

**Fig.8.2.4.1.b.** Arbore binar plin și reprezentarea sa cu ajutorul unui tablou liniar

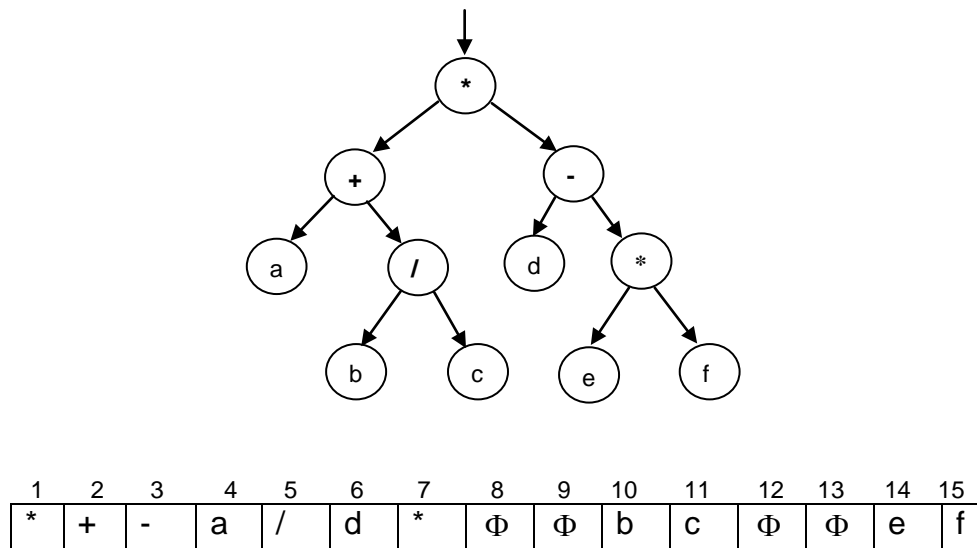
- Un arbore binar cu  $n$  noduri și de înălțime  $h$  se numește **arbore binar complet** dacă nodurile sale corespund nodurilor care sunt numerotate de 1 la  $n$  într-un arbore binar plin de înălțime  $h$ .
- O consecință a acestei definiții este aceea că într-un **arbore binar complet**, nodurile terminale apar pe **cel mult două niveluri adiacente**.
- Asemeni arborelui binar plin și arborele complet poate fi memorat într-un **tablou liniar** de dimensiune  $n$ .
- Diferența dintre un **arbore binar complet** și un **arbore binar plin**, ambele de înălțime  $h$ , este aceea că primul are un număr  $n$  de noduri unde  $n < 2^h - 1$ , iar cel de-al doilea are **exact**  $2^h - 1$  noduri.
- Ambii arbori sunt de fapt **arbori binari de înălțime minimă** care se bucură de proprietatea că nodurile lor terminale sunt distribuite pe **cel mult două niveluri** ale arborelui (fig.8.2.4.1.c)



**Fig.8.2.4.1.c.** Arbori binari:plin (a), complet (b), de înălțime minimă (echilibrat) (c)

- Lema următoare precizează maniera deosebit de simplă în care se poate determina părintele, fiul stâng și fiul drept al unui nod precizat, fără memorarea explicită a nici unui fel de informație de legătură.
- **Lema 2.** Dacă se reprezintă un **arbore binar complet** într-o manieră conformă cu cele anterior precizate, atunci pentru orice nod având indicele  $i$ ,  $1 \leq i \leq n$  sunt valabile relațiile:
  1.  $Parinte(i)$  este nodul având indicele  $\lfloor i/2 \rfloor$  dacă  $i \neq 1$ . Dacă  $i = 1$  nodul indicat de  $i$  este nodul rădăcină care nu are părinte.
  2.  $FiuStanga(i)$  este nodul având indicele  $2*i$  dacă  $2*i \leq n$ . Dacă  $2*i > n$ , atunci nodul  $i$  nu are fiu stâng.
  3.  $FiuDreapta(i)$  este nodul având indicele  $2*i+1$  dacă  $2*i+1 \leq n$ . Dacă  $2*i+1 > n$ , atunci nodul  $i$  nu are fiu drept.

- Acest mod de implementare poate fi în mod evident utilizat pentru **orice structură arbore binar**, în marea majoritate a cazurilor rezultând însă o utilizare **ineficientă** a zonei de memorie alocate tabloului.
- Spre exemplu în fig.8.2.4.1.d apare reprezentarea arborelui binar din fig.8.2.4.1.a. În acest caz se fac următoarele precizări:
  - h este **cea mai mică înălțime de arbore binar plin** care "cuprinde" arborele în cauză;
  - Se numerotează și **nodurile lipsă** ale arborelui de reprezentat, ele fiind înlocuite cu nodul vid  $\Phi$  în cadrul reprezentării.



**Fig.8.2.4.1.d.** Reprezentarea unui arbore binar oarecare

- Se remarcă similitudinea dintre acest mod de reprezentare și reprezentarea **ansamblor** (Vol.1 & 3.2.5).
- Această manieră de reprezentare a arborilor binari este bună pentru **arborii compleți** și este în general **ineficientă** pentru toți ceilalți.
- De asemenea, ea suferă și de **neajunsul** specific reprezentărilor bazate pe **tablouri liniare** și anume, dimensiunea fixă a zonei alocate.
- **Concluzionând**, referitor la această manieră de implementare a arborilor binari se pot face următoarele precizări:
  - **Avantaje:** Simplitate, absența legăturilor; parcurgerea simplă a arborelui în ambele sensuri.
  - **Dezavantaje:** Dimensiunea limitată, implementarea relativ complicată a modificărilor (inserții, suprimări), utilizare inefficientă în cazul arborilor rari.
  - **Se recomandă:** În cazul arborilor binari cu o dinamică redusă a modificărilor, în care se realizează multe parcurgeri sau cautări.

#### 8.2.4.2. Implementarea arborilor binari cu ajutorul pointerilor

- Maniera cea mai naturală și cea mai flexibilă de reprezentare a **arborilor binari** este cea bazată pe **TDA Indicator**.
- Acest mod de reprezentare are două **avantaje**:
  - Pe de-o parte înlătură **limitările** reprezentării secvențiale bazate pe structura tablou.
  - Pe de altă parte face uz de proprietățile **recursive** ale structurii arbore, implementarea realizându-se cu ajutorul **structurilor de date dinamice**.
- Un arbore binar poate fi descris cu ajutorul unei structuri de date dinamice în variantă Pascal (secvența [8.2.4.2.a]) respectiv în variantă C (secvența [8.2.4.2.b]).

-----  
{Implementarea arborilor binari cu ajutorul pointerilor}

```
TYPE RefTipNod=^TipNod;
      TipNod=RECORD
          cheie:char;                                [8.2.4.2.a]
          stang,drept: RefTipNod
      END;
      TipArboreBinar= RefTipNod;
```

-----

// implementare C

```
typedef struct nod
{
    //diferite campuri
    char cheie;
    struct nod* stang;                                [8.2.4.2.b]
    struct nod* drept;
}nod;
```

-----

- În fig.8.2.4.2.a se **poate** urmări reprezentarea **arborelui binar** din figura 8.2.4.1.a bazată pe definiția din secvența [8.2.4.2.a].
- Este ușor de văzut că practic **orice arbore** poate fi reprezentat în acest mod.
- În cadrul cursului de față, această modalitate de reprezentare va fi utilizată în mod preponderent.



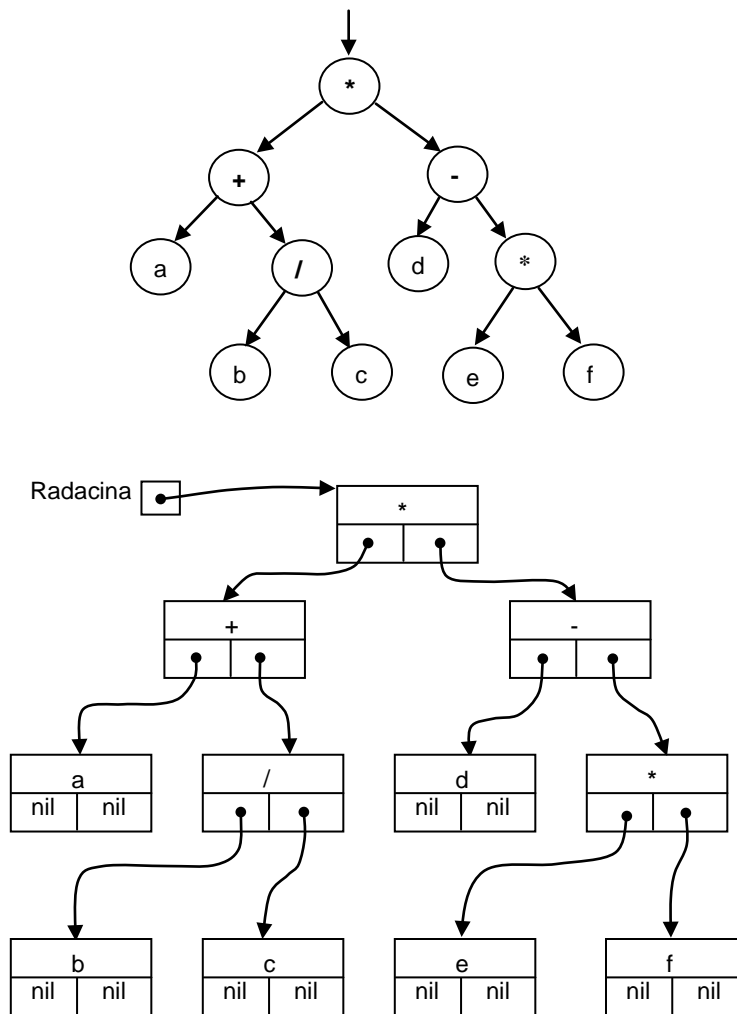


Fig.8.2.4.2.a. Arbore binar reprezentat cu ajutorul pointerilor

### 8.2.5. Traversarea arborilor binari

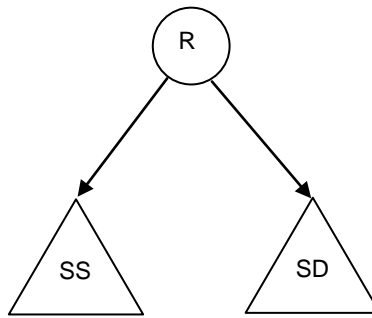
- Referitor la **tehnicile de traversare a arborilor binari** se face precizarea că ele derivă direct prin **particularizare** din tehnicile de traversare a **arborilor generalizați**.
- În consecință și în acest caz se disting tehnicile bazate pe **căutarea în adâncime** (preordine, înordine și postordine) precum și tehnica **căutării prin cuprindere**.

#### 8.2.5.1. Traversarea arborilor binari prin tehnici bazate pe căutarea în adâncime

- Traversarea în adâncime (**depth first**), așa cum s-a precizat și în cadrul arborilor generalizați, presupune **parcurgerea în adâncime** a arborelui binar atât de departe cât se poate plecând de la rădăcină.
- Când s-a ajuns la o frunză, **se revine** pe drumul parcurs la proximal nod care are fii neexplorați și parcurgerea se reia în aceeași manieră până la traversarea integrală a structurii.
- Schema este bună dar presupune **memorarea** drumului parcurs, ca atare necesită fie o

implementare recursivă fie utilizarea unei stive.

- **Traversarea în adâncime** are trei variante: **preordine**, **inordine** și **postordine**.
- Considerentele avute în vedere la traversarea arborilor generalizați rămân valabile și pentru arborii binari.
- Astfel, referindu-ne la arborele binar din fig.8.2.5.1.a și considerând pe R drept rădăcină, iar pe SS și SD drept subarboarele său stâng, respectiv subarboarele său drept, atunci cele **trei moduri de parcurgere** sunt evidențiate de modelele recursive din secvența [8.2.5.1.a].



**Fig.8.2.5.1.a.** Model de arbore binar

---

{Modele recursive pentru traversarea arborilor binari prin tehnica căutării în adâncime}

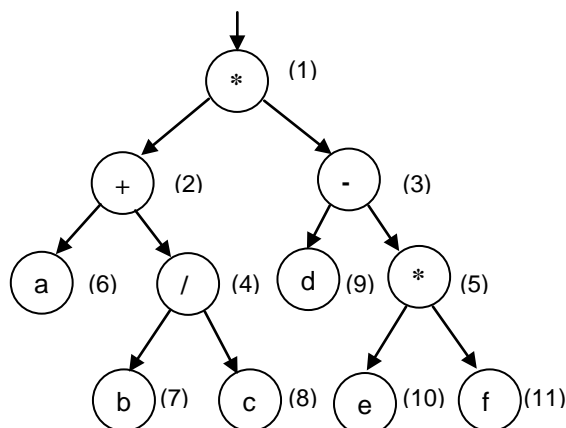
**Preordine(A) : R, Preordine(SS), Preordine(SD)**

**Inordine(A) : Inordine(SS), R, Inordine(SD)** [ 8.2.5.1.a]

**Postordine(A) : Postordine(SS), Postordine(SD), R**

---

- Traversarea unei structuri de date este de fapt o **ordonare liniară** a componentelor sale în baza unui anumit **protocol**.
- Spre **exemplu**, traversând arborele care memorează **expresia aritmetică**  $(a+b/c)*(d-e*f)$  din fig.8.2.1.b și înregistrând caracterul corespunzător pe măsură ce sunt vizitate nodurile arborelui, se obțin următoarele ordonări:



**Fig.8.2.1.b.** Arbore binar corespunzător unei expresii aritmetice (reluare)

- Preordine:  $*+a/bc-d*ef$
  - Inordine:  $a+b/c*d-e*f$
  - Postordine:  $abc/+def*-*$
- Se precizează faptul că ultima ordonare este cunoscută în matematică sub denumirea de **notație poloneză (postfix)**.
  - Prin scrierea unei expresii aritmetice în **notație poloneză** se înțelege scrierea operatorului **după** ce cei doi operanzi, în loc de a-l scrie între ei, ca în notația algebrică obișnuită (**infix**).
  - Astfel spre exemplu:

$a+b$	devine	$ab+$
$a*b+c$	devine	$ab*c+$
$a*(b+c)$	devine	$abc+*$
$a-b/c$	devine	$abc/-$
$(a-b)/c$	devine	$ab-c/$

- Este ușor de observat că **forma poloneză** a unei expresii reprezentate printr-un arbore binar se obține **parcurgând** arborele în **postordine**.
- În mod similar, în matematică se definește **notație poloneză inversă (prefix)** în care operatorul se scrie **în fața** celor doi operanzi. De exemplu expresia  $a+b$  se scrie  $+ab$ .
- Se observă de asemenea că **notația prefix** corespunde **traversării** în **preordine** a arborelui expresiei.
- O proprietate interesantă a notației poloneze este aceea că în ambele forme ea păstrează **semnificația** expresiei aritmetice **fără** a utiliza **paranteze**.
- Cu privire la ordonarea în **inordine**, se face precizarea că ea corespunde notației obișnuite a expresiei aritmetice dar cu **omiterea parantezelor**, motiv pentru care semnificația matematică a expresiei se pierde, cu toate că arborele binar corespunzător păstrează în structura sa această semnificație (vezi fig.8.2.1.b).
- În continuare, cele trei metode de traversare vor fi concretizate în trei **proceduri recursive** în care:
  - $r$  este o **variabilă** de tip pointer care indică rădăcina arborelui.
  - **Viziteaza**( $r^{\wedge}$ ) reprezintă **operația** care trebuie executată asupra fiecărui nod.
- Considerând pentru structura **arbore binar** definiția din secvența [8.2.5.1.b], structura de principiu a celor trei metode de traversare este prezentată în [8.2.5.1.c].

-----  
 {Traversarea arborilor binari}

**TYPE** RefTipNod=<sup>^</sup>TipNod;

TipNod=**RECORD**

{diferite câmpuri}

[8.2.5.1.b]

```

        stang,drept:RefTipNod
    END;
-----
PROCEDURE Preordine(r:RefTipNod);
BEGIN
    IF r<>nil THEN
        BEGIN
            Viziteaza(r^);
            Preordine(r^.stang);
            Preordine(r^.drept)
        END
    END; {Preordine}

PROCEDURE Inordine(r:RefTipNod);
BEGIN
    IF r<>nil THEN
        BEGIN
            Inordine(r^.stang);           [8.2.5.1.c]
            Viziteaza(r^);
            Inordine(r^.drept)
        END
    END; {Inordine}

PROCEDURE Postordine(r:RefTipNod);
BEGIN
    IF r<>nil THEN
        BEGIN
            Postordine(r^.stang);
            Postordine(r^.drept);
            Viziteaza(r^)
        END
    END; {Postordine}
-----

```

### 8.2.5.2. Traversarea arborilor binari prin tehnica căutării prin cuprindere

- Traversarea prin **cuprindere (breadth-first)** presupune traversarea tuturor nodurilor de pe un nivel al structurii arborelui, după care se trece la nivelul următor parcurgându-se nodurile acestui nivel ș.a.m.d până la epuizarea tuturor nivelurilor arborelui.
- În cazul arborilor binari, rămâne valabilă **schema de principiu** a parcurgerii prin cuprindere bazată pe utilizarea unei structuri de date **coadă**, prezentată pentru arborii generalizați (secvența [8.1.3.2.a]), cu precizarea că nodurile pot avea cel mult **doi fii**.
- În continuare se prezintă o **altă variantă** de parcurgere care permite efectiv **evidențierea nivelurilor arborelui binar**, variantă care utilizează în tandem **două structuri coadă** (secvența [8.2.5.2.a]).
  - La parcurgerea nodurilor unui nivel al arborelui binar, noduri care sunt memorate într-una din cozi, fiii acestora se adaugă celeilalte cozi.
  - La terminarea unui nivel (golirea cozii curente parcurse), cozile sunt comutate și procesul se reia până la parcurgerea integrală a arborelui binar, adică golirea ambelor cozi.

-----  
{Traversarea prin cuprindere unui arbore binar cu  
evidențierea nivelurilor acestuia}

```
procedure TraversarePrinCuprindereArboreBinar(r: TipNod)
  Coadal, Coadal2: TipCoadal;

  Initializeaza(Coadal); Initializeaza(Coadal2);
  daca r nu este nodul vid atunci
    Adauga(r,Coadal); {procesul de amorsare}
  repetă
    cât timp NOT Vid(Coadal) execută
      r<-Cap(Coadal); Scoate(Coadal);
      *listeaza(r);
      dacă FiuStanga(r) nu este nodul vid atunci
        Adauga(FiuStanga(r),Coadal2);
      dacă FiuDreapta(r) nu este nodul vid atunci
        Adauga(FiuDreapta(r),Coadal2);
    □
    cât timp NOT Vid(Coadal2) execută [8.2.5.2.a]
      r<-Cap(Coadal2); Scoate(Coadal2);
      *listeaza(r);
      dacă FiuStanga(r) nu este nodul vid atunci
        Adauga(FiuStanga(r),Coadal);
      dacă FiuDreapta(r) nu este nodul vid atunci
        Adauga(FiuDreapta(r),Coadal);
    □
  până când Vid(Coadal) AND Vid(Coadal2)
  □
```

-----

### 8.2.6. Arbori binari cu legături ("Threaded Trees")

- Implementările bazate pe **indicatori** (cursori sau pointeri) permit parcurgerea simplă în manieră **descendentă** a unui arbore în (variantă recursivă sau iterativă) dar fac practic relativ dificilă **parcurgerea ascendentă**.
- În principiu această problemă poate fi rezolvată, memorând în fiecare moment al parcurgerii pe lângă **nodul curent** și **părintele acestuia**:
  - **Dezavantaj**: se realizează un **consum mai ridicat de memorie**.
  - **Avantaj**: se reduce gradul de complexitate al algoritmului de parcurgere.
- În acest caz, pentru a **reduce** consumul de memorie poate fi avantajos să se modifice în mod convenabil structura de date, acceptând o creștere corespunzătoare a complexității algoritmilor de prelucrare.
- O **primă soluție** pentru a realiza parcurgerea ascendentă simplă a unui arbore este aceea de a **memora** pentru **fiecare nod** în parte legătura la **părintele său**, rezolvând această problemă în maniera **listelor dublu înlănțuite**.
  - Dacă nodul are o structură suficient de complexă, un câmp de înlănțuire suplimentar **nu** conduce la un consum excesiv de memorie [8.2.6.a].

-----  
{Arbori binari cu trei indicatori}

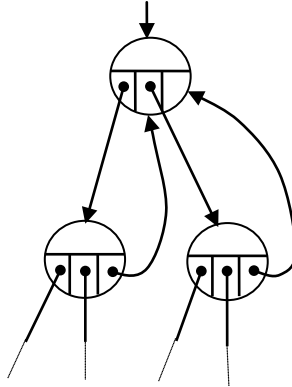
```

TYPE RefTipNod = ^TipNod;
    TipNod=RECORD
        info: TipInfo;                                [8.2.6.a]
        stang, drept, parinte: RefTipNod
    END;

```

---

- Reprezentarea grafică a acestei soluții apare în figura 8.2.6.a.



**Fig.8.2.6.a.** Arbore binar cu legături

- O **altă posibilitate** larg utilizată în implementarea arborilor binari se bazează pe observația că o **mare** parte a **nodurilor** unei structuri arbore au câmpurile indicator **nule**.
- Aceste câmpuri pot fi **modificate** astfel încât ele să indice noduri în arbore care **preced** sau **succed** nodul în cauză într-o anumită ordine de traversare.
  - Spre exemplu, referindu-ne la traversarea în **inordine**, se poate conveni ca în fiecare nod, în locul **înlănțuirii vide pe dreapta** să apară legătura la **nodul următor** la parcurgerea în **inordine** iar în locul **înlănțuirii vide pe stânga** să apară legătura la **nodul anterior** în aceeași parcurgere.
  - Întrucât câmpurile indicator au fost **supraîncărcate** ("overloaded") este necesar ca în structura nodului să fie efectuate modificări care să evidențieze natura acestor câmpuri: **pointeri** (înlănțuiri) sau **legături** ("threads").
  - Se precizează faptul că **pointerii** indică fii ai nodului în cauză, iar **legăturile** predecesori sau succesori la parcurgerea în inordine.
- O astfel de structură de arbore se numește **arbore cu legături** ("threaded tree").
- Un arbore cu legături poate fi:
  - (1) **Parțial** (dacă numai **unul** din câmpuri este folosit ca legătură)
  - (2) **Plin** ("fully threaded tree") dacă se utilizează ambele câmpuri.

### 8.2.6.1. Arbori binari cu legături plini ("Fully Threaded Trees")

- O posibilitate de implementare a unui **arbore cu legături plin** este cea precizată în secvența [8.2.6.1.a].

{Arbori binari cu legături plini (Fully Threaded Trees)}

```
TYPE RefTipNod=^TipNod;  
  TipNod=RECORD  
    info: TipInfo;  
    legaturaStg,legaturaDr: BOOLEAN;    [8.2.6.1.a]  
    stang,drept: RefTipNod  
  END;  
  TipArboreBinarLeg= RefTipNod;
```

- Câmpurile booleene **legăturăStg** respectiv **legăturăDr** au valoarea **adevărat** dacă câmpurile indicator corespunzătoare sunt **legături** respectiv au valoarea **fals** când acestea sunt **înlănțuiri**.
- Un **exemplu** de astfel de implementare apare în figura 8.2.6.1.a. **Legăturile** sunt marcate cu linii **întrerupte** iar **pointerii** cu linii **continue**.

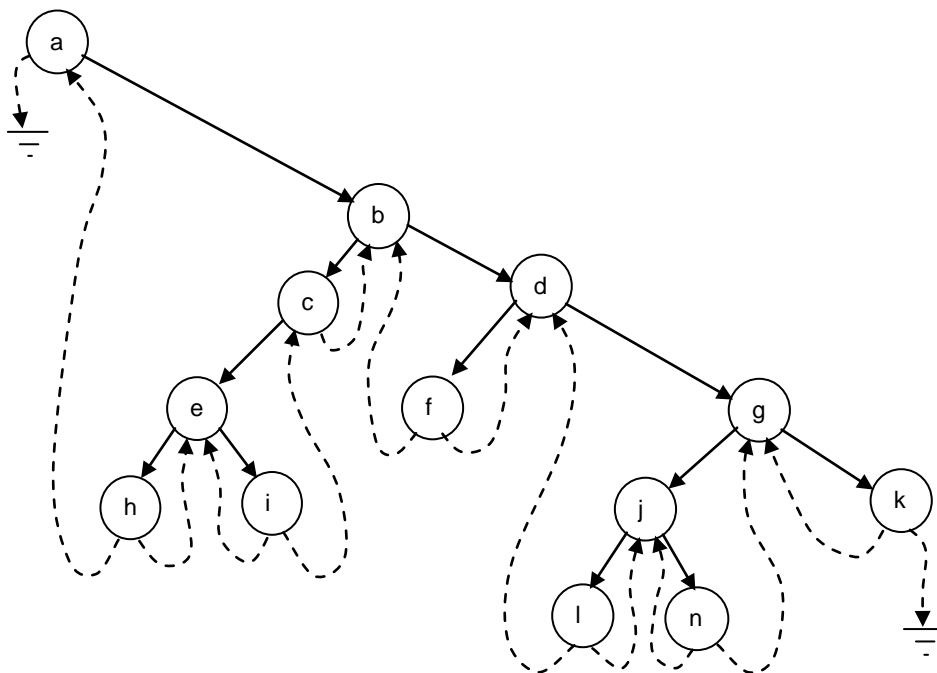


Fig.8.2.6.1.a. Arbore cu legături plin

- Ținând cont de faptul că parcurgerea în inordine a arborelui din figura 8.2.6.1.a. este:  
**a,h,e,i,c,b,f,d,l,j,n,g,k**
- Se observă că:

- Pentru nodurile terminale, legăturile precizează **predecesorul** (legătura pe stânga) respectiv **succesorul** (legătura pe dreapta) nodului în cauză la parcurgerea în inordine, spre exemplu e respectiv c pentru nodul i.
- Cel mai din **stânga nod** al fiecărui **subarbore drept** indică prin legătura sa pe stânga **părintele** arborelui (h pe a sau l pe d), după cum același lucru este valabil și pentru cel mai din **dreapta nod** al fiecărui **subarbore stâng** (n pe g).
- **Cel mai din stânga și cel mai din dreapta** nod al structurii au legătura pe stânga respectiv pe dreapta **vidă** (nodurile a și k).
- Se menționează faptul că **parcurgerea în inordine** poate fi evidențiată practic simplu realizând **proiecția pe abscisă** a nodurilor arborelui.
- Un **prim avantaj** al acestui mod de reprezentare este acela că permite **traversarea în inordine** a arborelui binar în **manieră iterativă**.
  - Într-adevăr, în această traversare **primul** nod vizitat este cel mai din stânga nod al subarborelui stâng.
  - După vizitarea acestui nod, urmează **părintele** acestuia la care se ajunge direct urmând legătura pe dreapta a nodului respectiv.
  - După vizitarea părintelui, se trece la parcurgerea în aceeași manieră a **subarborelui drept**
  - Se continuă în aceeași manieră până la vizitarea **celui mai din dreapta** nod al întregii structuri, nod care este singurul care are legătura pe dreapta vidă.
- Procedura care ilustrează această traversare este **InorderIterativ** și apare în secvența [8.2.6.1.b].

-----  
 {Traversarea unui arbore cu legături - Varianta iterativă}

```
PROCEDURE InorderIterativ (n: RefTipNod);
{Traversează arborele binar cu legături în inordine începând
cu nodul n}
```

```
BEGIN
```

```
  WHILE n<>NIL DO {până la atingerea celui mai din dreapta
                  nod}
```

```
    BEGIN
```

```
      WHILE (NOT n^.legaturaStg) AND (n^.stang<>NIL) DO
        n:=n^.stang; {avansare pe stânga cât este posibil}
```

```
      viziteaza(n); {vizitare cel mai din stânga nod}
```

```
      WHILE n^.legaturaDr DO {urcare în arbore în inordine}
```

```
        BEGIN
```

```
          n:=n^.drept;
```

```
          viziteaza(n) [8.2.6.1.b]
```

```
        END;
```

```
        n:=n^.drept {se trece la subarborele drept}
```

```
    END
```

```
END; {InorderIterativ}
```

-----



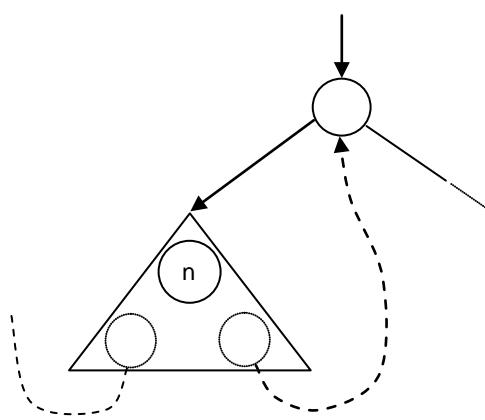
- Un **al doilea avantaj** al acestui nod de traversare este acela că traversarea poate demara cu **oricare** nod al arborelui, lucru care este practic imposibil în varianta recursivă.
- În această reprezentare **operatorii** definiți de TDA arbore binar se implementează într-o **manieră similară** cu reprezentarea bazată pe **pointeri**.
- **Diferența** apare la inserția sau suprimarea unui subarbore.
  - Se consideră spre exemplu operatorul ***InlocuiesteSub***(*n, r, t*) unde *n*: Indicator, iar *r, t*: ArboreBinar.
  - La inițiativa acestui operator, **subarboarele** având rădăcina *r* trebuie plasat în locul precizat de indicatorul *n*.
  - Pentru simplificarea lucrărilor, se va ignora posibilitatea de a **reutiliza** spațiul alocat subarborelui indicat de *n* precum și eventualele supraîncărcări (“aliasing”).
  - **Inserția** subarborelui indicat de *r* presupune de fapt poziționarea **legăturii** pe **dreapta** a **celui mai din dreapta nod** al subarborelui *r* astfel încât să indice poziția precizată de nodul similar din subarboarele *n* și în mod analog pentru **cel mai din stânga nod** al subarborelui *r*, după cum rezultă din figura 8.2.6.1.b.
- În secvența [8.2.6.1.c] apare implementarea operatorului ***InlocuiesteSub***.

-----  
 {Operatorul Inlocuieste Subarbore}

```

PROCEDURE InlocuiesteSub(VAR n: RefTipNod; r:
TipArboreBinarLeg; VAR t: TipArboreBinarLeg);
  VAR CelMaiStg, CelMaiDr: RefTipNod; {noduri extreme în
                                         subarboarele n}
      rStg, rDr: RefTipNod; {noduri extreme in subarboarele r}
  BEGIN {caută nodurile extreme în arborele original n}
    CelMaiStg:=n;
    REPEAT
      CelMaiStg:=CelMaiStg^.stang
    UNTIL (CelMaiStg^.legaturaStg)OR(CelMaiStg^.stang=NIL);
    CelMaiDr:=n;
    REPEAT [8.2.6.1.c]
      CelMaiDr:=CelMaiDr^.drept
    UNTIL (CelMaiDr^.legaturaDr) OR (CelMaiDr^.drept= NIL);
    n:=r; {înlocuiește pe n cu r}
    rStg:=n; {caută și modifica legaturile extreme ale lui r}
    WHILE rStg^.stang<>NIL DO
      rStg^:=rStg^.stang;
    rStg^.stang:= CelMaiStg; rStg^.legaturaStg:=TRUE;
    rDr:=n;
    WHILE rDr^.drept<>NIL DO
      rDr^:=rDr^.drept;
    rDr^.drept:=CelMaiDr; rDr^.legaturaDr:=TRUE;
  END; {InlocuiesteSub}
  
```

-----



**Fig.8.2.6.1.b.** Implementarea operatorului InlocuiesteSub

- **Revenind** la problema de la care s-a pornit și anume **afierea părintelui unui nod precizat**, arborii cu legături pot fi utilizați în acest scop.
  - Astfel considerând nodul  $n$ , pentru a afla **părintele** său se procedează după cum urmează:
  - În primul rând trebuie să se determine dacă nodul  $n$  este fiul **stâng** sau **drept** al părintelui său.
  - Presupunând că el este fiul **stâng**, atunci subarborele drept al lui  $n$  se află situat între nodul  $n$  și părintele acestuia la parcurgerea în **inordine**.
  - În consecință **cel mai din dreapta** descendent al lui  $n$  precede imediat părintele nodului  $n$ , deci urmând legătura sa pe dreapta se ajunge la nodul căutat (fig.8.2.6.1.c.(a)).
  - Problema se rezolvă într-o manieră similară dacă  $n$  este **fiul drept** al tatălui său (fig.8.2.6.1.c.(b)).
- Cu alte cuvinte:
  - Dacă  $n$  este un **fiu stâng**, atunci părintele său poate fi găsit urmând înlanțuirile pe **dreapta** cât de departe este posibil și parcurgând prima legătură pe dreapta
  - Dacă  $n$  este un **fiu drept**, atunci părintele său poate fi găsit parcurgând înlanțuirile pe **stânga** cât de departe se poate și apoi urmând prima legătură pe stânga.
  - Desigur, întrucât inițial **nu** se cunoaște dacă  $n$  este fiu stâng sau fiu drept al părintelui său, este necesară parcurgerea ambelor posibilități
  - **Varianta corectă** se determină verificând dacă nodul la care s-a ajuns are într-adevăr pe  $n$  ca fiu stângă respectiv dreapta.

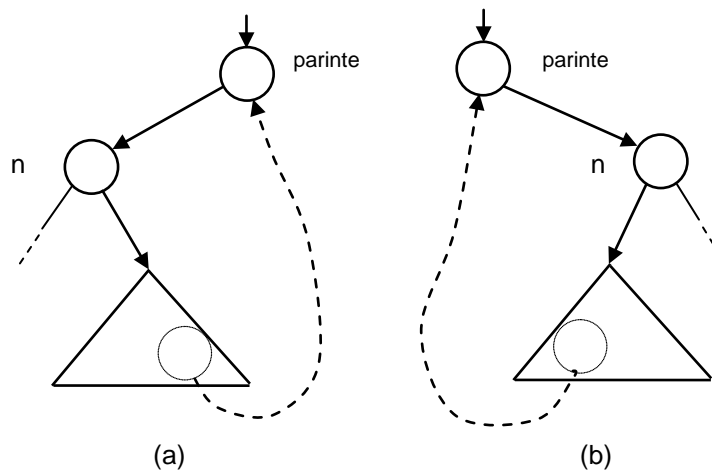


Fig.8.2.6.1.c. Stabilirea părintelui unui nod

### 8.2.6.2. Arbori binari cu legături parțiale

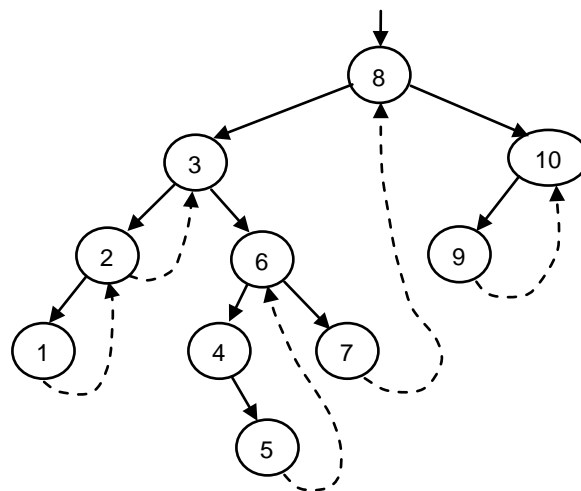
- După cum s-a precizat, în cazul arborilor binari este posibil să fie utilizat **numai unul** din câmpurile de legătură (cel stâng sau cel drept) funcție de sensul parcurgerii arborelui, rezultând astfel un **arbore binar cu legături parțiale**.
- În cazul parcurgerii în **inordine** este suficientă utilizarea câmpului **drept**.
- Un astfel de arbore apare în fig.8.2.6.2.a, iar structura de date asociată lui în secvența [8.2.6.2.a].

{Arbori binari cu legături parțiale}

```

TYPE RefTipNod=^TipNod;
      TipNod=RECORD
          element: TipElement;
          stang,drept: RefTipNod;
          legatura: BOOLEAN
      END;
      TipArboreBinarLegPar=RefTipNod;
  
```

- Câmpul `legatura` se referă la câmpul `drept`.
- Astfel dacă `p.legatura= FALSE` atunci `p.drept` indică subarborele drept al nodului `p`, altfel `p.drept` indică legătura nodului `p` (reprezentată punctat în fig.8.2.6.2.a).



**Fig.8.2.6.2.a.** Arbore binar cu legături parțiale

- Această reprezentare suportă **implementarea** simplă (eventual cu anumite restricții) a majorității operatorilor definiți pe **TDA arbore binar**.
- Implementarea TDA arbore binar, pornind de la aceste considerente, poate fi considerată un **exercițiu** incitant și în același timp util.

### 8.2.7 Aplicații ale arborilor binari

- Dintre aplicațiile specifice arborilor binari se prezintă câteva considerate mai reprezentative:
  - Construcția unui arbore binar de **înălțime minimă**
  - Generarea unui arbore binar pornind de la specificarea sa în **preordine**
  - Construcția unui **arbore de parcurgere** pornind de la forma postfix a expresiei asociate

#### 8.2.7.1. Construcția și reprezentarea grafică a unui arbore binar de înălțime minimă

- Se presupune că arborele binar ce trebuie construit conține noduri încadrate în tipul clasic definit în secvența [8.2.4.2.b], în care cheile nodurilor sunt  $n$  **numere** care **se citesc** de la tastatură.
- Restricția care se impune este aceea, ca arborele construit să aibă **înălțimea minimă**.
- Pentru aceasta este necesar ca **fiecărui nivel** al structurii arborelui să-i fie alocate **numărul maxim** de noduri posibile, cu excepția nivelului de bază.
- Acest lucru poate fi realizat prin **distribuirea** în mod egal a nodurilor care se introduc în structură pe **stânga** respectiv pe **dreapta** fiecărui nod deja introdus.
- **Regula** după care se asigură distribuția egală pentru un număr cunoscut de noduri  $n$  poate fi formulată simplu în **termeni recursivi**:

1° Se ia un nod drept rădăcină;

2° Se generează subarborele stâng cu  $n_s = n \text{ DIV } 2$  noduri;

3° Se generează subarborele drept cu  $n_d = n - n_s - 1$  noduri.

- Acest algoritm, permite construcția simplă a unui arbore binar de înălțime minimă.
- Un **arbore binar** este considerat **perfect echilibrat** dacă pentru fiecare nod, numărul de noduri al subarborelui stâng diferă de cel al subarborelui drept cu **cel mult** o unitate.
- În mod evident că **arborele de înălțime minimă** construit de către acest algoritm este unul **perfect echilibrat**.

- Implementarea algoritmului **varianta C** apare în secvența [8.2.7.1.a].
  - Cheile nodurilor se introduc de la tastatură.
  - Prima valoare introdusă este numărul total de noduri n.
  - Sarcina construcției efective a arborelui binar revine funcției **Arbore** care:
    - Primește ca parametru de intrare numărul de noduri n
    - Determină numărul de noduri pentru cei doi subarbori
    - Citește cheia nodului rădăcină
    - Construiește în manieră recursivă arborele
    - Returnează referința la arborele binar construit.

---

```

//Construcția unui arbore binar de înălțime minimă

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

typedef struct nod                                     [8.2.7.1.a]
{
    int cheie;
    struct nod* stang;
    struct nod* drept;
}NOD;

int n;
NOD* radacina;

NOD* Arbore(int n)
{
    //construcție arbore perfect echilibrat cu N noduri
    NOD* NodNou;
    int x, ns, nd;

    if (n==0)
        return NULL;

    ns = n / 2;                                     [8.2.7.1.a]
    nd = n - ns - 1;
    scanf("%d", &x);

    if ((NodNou = (NOD *)malloc( sizeof(NOD))) == NULL)
    {
        printf("Eroare la malloc");
        return NULL;
    }
    NodNou->cheie = x;
    NodNou->stang = Arbore(ns);    //completare înlănțuire
    NodNou->drept = Arbore(nd);    //completare înlănțuire
    return NodNou;
}
//Arbore

```

```

void Afiseaza_Arbore( NOD* t, int h)
{
    //afișează arborele t
    int i;
    if (t != NULL)
    {
        Afiseaza_Arbore(t->stang, h - 5);
        for(i=0; i<h; i++) printf( " ");
        printf("%d\r\n", t->cheie);
        Afiseaza_Arbore( t->drept, h - 5 );
    } //if
} //Afiseaza_Arbore

int main(int argc, char** argv)
{
    printf( "n=" );
    scanf("%d", &n); //numărul total de noduri
    radacina = Arbore(n); //construcție arbore
    Afiseaza_Arbore(radacina, 50);
    return 0;
} //main

```

---

- Funcția **Afiseaza\_Arbore** realizează **reprezentarea grafică** a unei structuri de arbore binar răsturnat (rotit cu 90° în sensul acelor de ceasornic), conform următoarei specificații:
  - 1° Pe fiecare rând se afișează **exact un nod**
  - 2° Nodurile sunt ordonate de sus în jos în **inordine**
  - 3° Dacă un nod se afișează pe un rând precedat de h blankuri, atunci fiii săi (dacă există) se afișează precedați de h-d blankuri. d este o variabilă a cărei valoare este stabilită de către programator și precizează **distanța** dintre nivelurile arborelui măsurată în caractere "spațiu"
  - 4° Pentru **rădăcină**, se alege o valoare corespunzătoare a lui h, ținând cont că arborele se dezvoltă spre stânga.
- Această modalitate de reprezentare grafică a arborilor binari este una dintre cele mai **simple** și **imEDIATE**, ea recomandându-se în mod deosebit în aplicațiile didactice.
- Trebuie subliniat faptul că simplitatea și transparența acestui program rezultă din utilizarea **recursivității**.
  - Aceasta reprezintă un **argument** în plus adus afirmației, că **algoritmii recursivi** reprezintă modalitatea cea mai potrivită de prelucrare a unor **structuri de date** care la rândul lor sunt definite **recursiv**.
  - Acest lucru este demonstrat atât de către funcția care **construiește** arborele cât și de către procedura de **afișare** a acestuia, procedură care este un exemplu clasic de parcurgere în inordine a unui arbore binar.

### 8.2.7.2. Generarea unui arbore binar pornind de la specificarea sa în preordine

- Obiectivul **exemplului** de față, îl constituie generarea unui arbore binar cu formă dirijată.
- Specificarea arborelui se face în **preordine**, enumerând nodurile (reprezentate în cazul de față printr-un singur caracter) începând cu **rădăcina** urmată mai întâi de **subarboarele stâng** apoi de cel **drept**.
- Astfel arborelui specificat în forma:  
**abc...de...fg...hi...jkl...m...n...**

unde punctul semnifică **arborele vid**, îi corespunde reprezentarea din figura 8.2.7.2.a.

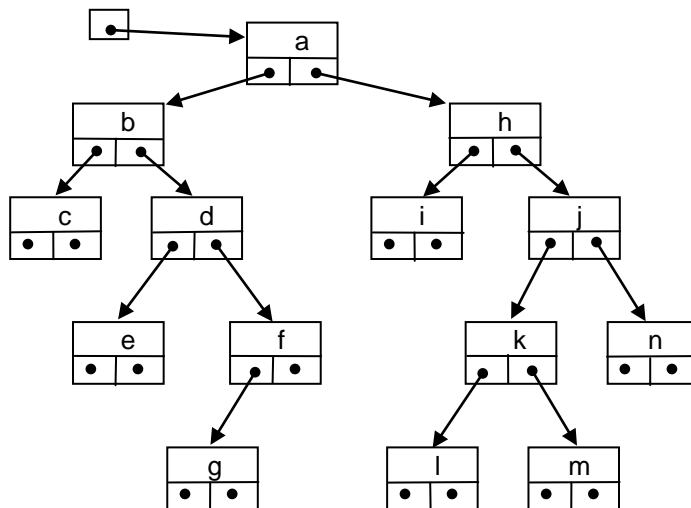


Fig.8.2.7.2.a. Structură arbore binar

- Procedura care construiește o astfel de structură pornind de la specificarea arborelui în preordine precizată printr-un șir de caractere apare în [8.2.7.2.a]
- Se precizează că sunt valabile structurile de date precizate în secvența [8.2.4.2.a].

---

{Generarea unui AB pornind de la specificarea sa în preordine}

```

PROCEDURE Creare(VAR p:RefTipNod);
BEGIN
  Read(ch);
  IF ch<>'.' THEN                                     [8.2.7.2.a]
    BEGIN
      new(p);                                           {completare înlănțuire}
      p^.cheie:=ch;
      Creare(p^.stang);                                {tehnica look-ahead}
      Creare(p^.drept);                                {tehnica look-ahead}
    END
  ELSE
    p:=NIL
  END; {Creare}

```

---

### 8.2.7.3. Construcția unui arbore de parcurgere ("Parse Tree")

- După cum s-a precizat în paragraful 8.2.1. oricărei expresii aritmetice i se poate asocia un arbore binar numit arbore de parcurgere ("**parse tree**").
  - În paragraful de față se pune problema de a **construi** un astfel de arbore binar asociat unei **expresii aritmetice**.
  - Algoritmul precizat de secvența [8.2.7.3.a] realizează acest lucru pornind de la notația **postfix** a expresiei.
    - Se presupune ca forma **postfix** a expresiei aritmetice apare ca un șir de caractere care este citit element cu element **de la dreapta la stânga** de către procedură [De89].
- 

**PROCEDURE ConstrArbParcurgere**(p:RefTipNod);

{Construcția unui arbore de parcurgere pentru o expresie aritmetică pornind de la forma postfix a expresiei. Expresia se parcurge de la dreapta la stânga}

**VAR** t:TipElement;

**BEGIN**

CitesteElement(t); {următorul element în ordine de  
la dreapta la stânga}

**IF** t<>sfarsit **THEN**

**BEGIN**

\*generează un nod nou indicat de p;

\*plasează pe t în noul nod;

**IF** t este un operator **THEN** [8.2.7.3.a]

**BEGIN** {construcție recursiva

**ConstrArbParcurgere**(p^.drept);

**ConstrArbParcurgere**(p^.stang)

**END**

**ELSE**

**BEGIN** {elementele non operator sunt întotdeauna  
frunze}

\*se face subarborele drept al lui p, vid;

\*se face subarborele stang al lui p, vid

**END**

**END**

**END;** {**ConstrArbParcurgere**}

---