8. Arbori

8.1. Arbori generalizaţi

8.1.1. Definiţii

- În definirea **noţiunii de arbore** se porneşte de la noţiunea de **vector**.

  - Fie **V** o mulţime având elementele $a_1, a_2, \ldots, a_n$.

  - Pe mulţimea **V** se poate defini o aşa numită **"relaţie de precedenţă"** în felul următor: se spune că a_i precede pe a_j dacă $i<j$. Aceasta se notează: $a_i \prec a_j$.

  - Se poate verifica uşor că relaţia astfel definită are următoarele proprietăţi, valabile pentru structura vector:

---

(1) Oricare ar fi $a \in V$ avem $a \nprec a$. (S-a notat cu $\nprec$ relaţia "nu precede");
(*antireflexivitate*)
(2) Dacă $a \prec b$ şi $b \prec c$ atunci $a \prec c$ (*tranzitivitate*); [8.1.1.a]
(3) Oricare ar fi $a \in V$ şi $b \in V$, dacă $a \neq b$ atunci avem fie $a \prec b$ fie $b \prec a$.

- Din proprietăţile (1) şi (2) rezultă că relaţia de precedenţă **nu** determină în **V** **"bucle închise"**, adică **nu** există nici o secvenţă de elemente care se preced două câte două şi în care ultimul element este acelaşi cu primul, cum ar fi de exemplu $a \prec b \prec c \prec d \prec a$.

- Proprietatea (3) precizează că relaţia de precedenţă este definită pentru **oricare** două elemente a şi b ale lui **V**, cu singura condiţie ca $a \neq b$.

- Fie **V** o mulţime finită peste elementele căreia s-a definit o relaţie de precedenţă, stabilind referitor la fiecare pereche de elemente, care dintre ele îl precede pe celălălalt.

  - Dacă această relaţie posedă proprietăţile [8.1.1.a], atunci ea **imprimă** peste mulţimea **V** o **structură vector** (fig.8.1.1.a).

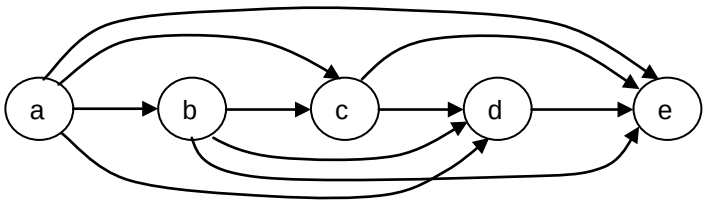


Fig. 8.1.1.a. Structură vector

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

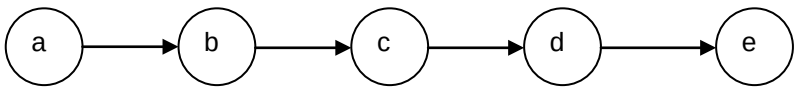


Fig.8.1.1.b. Relaţia succesor

- Relaţia ”**succesor”** se defineşte cu ajutorul relaţiei de **precedenţă** după cum urmează:
    - Dacă între elementele a şi b ale lui $\mathbf{V}$ este valabilă relaţia $a \prec b$ şi nu există nici un $c \in \mathbf{V}$ astfel ca $a \prec c \prec b$ atunci se zice că b este **succesorul** lui a.
- Se observă că relaţia "**succesor**" (mulţimea săgeţilor din figura 8.1.1.b.), **precizează** relaţia "**precedenţă**" fără a fi însă **identică** cu ea.
    - Spre exemplu, există relaţia $b \prec e$ (prin tranzitivitate), dar nici o săgeată nu conectează pe b cu e.
- În figura8.1.1.c apare o aşa numită **structură arbore** care se defineşte prin **generalizarea** structurii **vector**.

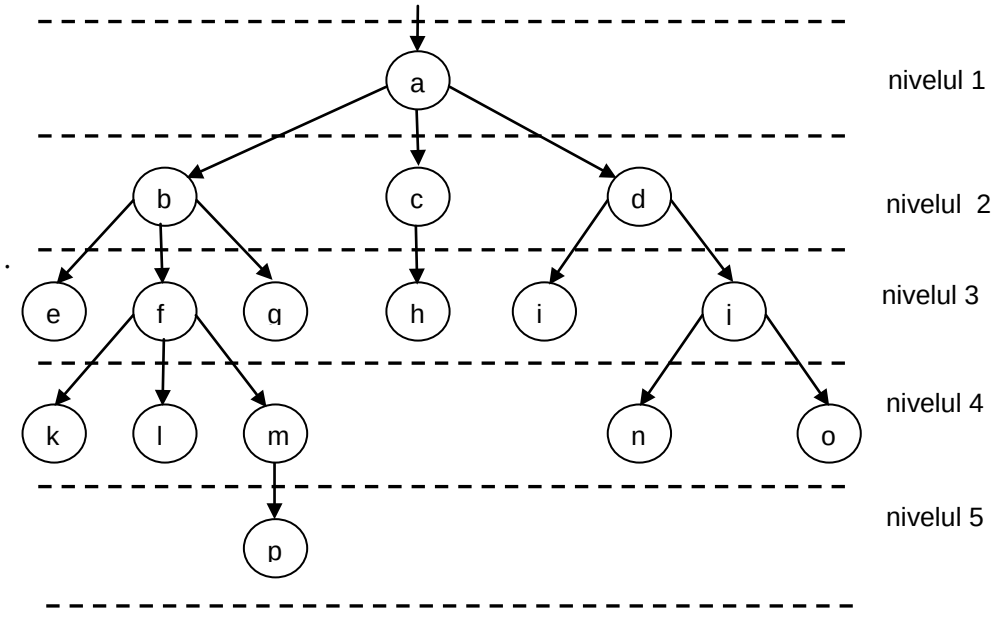


Fig. 8.1.1.c. Structură arbore

- Astfel, dacă în cazul **vectorului**, **toate elementele** cu excepţia ultimului au **exact** un **succesor**, la **structura de arbore** se admite ca **fiecare element** să aibă un **număr oarecare** de **succesori**, inclusiv zero, cu **restricţia** ca două elemente distincte să **nu** aibă acelaşi succesor.
- Relaţia **succesor** defineşte o relaţie de **precedenţă** pe structura arbore. Astfel, din figura avem $b \prec p, d \prec n,$ etc.
- **Relaţia de precedenţă** definită pe structura arbore se bucură de proprietăţile (1) şi (2) de la [8.1.1.a] dar **nu** satisface proprietatea (3).
    - Într-adevăr în figura 8.1.1.c, pentru elementele b şi c **nu** este valabilă nici una din relaţiile $b \prec c$ sau $c \prec b$, la fel pentru elementele d şi k.
    - Prin urmare, relaţia de precedenţă este definită numai pentru **o parte** a perechilor de elemente ale arborelui, cu alte cuvinte este o **relaţie parţială**.
- În general, o **structură arbore** se defineşte ca o mulţime $\mathbf{A}$ de $n \leq 0$ noduri de acelaşi tip, peste care s-a definit o relaţie de precedenţă având proprietăţile (1) şi (2) de la [8.1.1.a] precum şi următoarele două proprietăţi [8.1.1.b]:

- (3) Oricare ar fi $b, c \in \mathbf{A}$, astfel încît $b \nprec c$ şi $c \nprec b$, dacă $b \prec d$ şi $c \prec e$ atunci $d \neq e$.
    - Cu alte cuvinte, două elemente oarecare între care **nu** există relaţia de precedenţă **nu** pot avea acelaşi 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 şi o altă definiţie echivalentă cu cea de mai sus.

- Prin **arbore**, se înţelege o mulţime de $n \geq 0$ noduri de acelaşi 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 **disjuncţi**.

- Se constată că atât această definiţie, cât şi structura pe care o defineşte, 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 **noţiuni** referitoare la arbori.

- Prin **subarborii** unui arbore, se înţeleg 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 şi d.

- Oricare nod al unui arbore este rădăcina unui **arbore parţial**.

    - Spre exemplu în aceeaşi figură, f este rădăcina arborelui parţial format din nodurile f, k, l, m şi p.

- Un arbore parţial **nu** este întotodeauna **subarbore** pentru arborele complet, în schimb orice **subarbore** este în acelaşi timp şi **arbore parţial**.

- Într-o structură de arbore, succesorul unui nod se mai numeşte şi **"fiul"** sau **"urmaşul"** său.

- Dacă un nod are unul sau mai mulţi fii, atunci el se numeşte **"tatăl"** sau **"părintele"** acestora.

- Dacă un nod are mai mulţi fii, aceştia se numesc **"fraţi"** între ei.

    - Spre exemplu în fig. 8.1.1.c nodul b este tatăl lui e, f şi g care sunt fraţi între ei şi sunt în acelaşi timp fiii lui b.

- Se observă că într-o structură arbore, **arborele parţial** determinat de orice nod diferit de rădăcină, este **subarbore** pentru **arborele parţial** determinat de tatăl său.

- Astfel f este tatăl lui m, iar arborele parţial determinat de m este subarbore pentru arborele parţial determinat de f.

- Într-o structură arbore se definesc **niveluri** în felul următor: rădăcina formează nivelul 1, fiii ei formează nivelul 2 şi în general fiii tuturor nodurilor nivelului n formează nivelul n+1 (fig.8.1.1.c).

- Nivelul maxim al nodurilor unui arbore se numeşte **înălţimea** arborelui.

- Numărul fiilor unui nod defineşte **gradul nodului** respectiv.

- Un nod de grad zero se numeşte nod **terminal (frunză)**, iar un nod de grad diferit de zero, nod **intern**.

- **Gradul** maxim al nodurilor unui arbore se numeşte **gradul arborelui**.

    - Arborele din figura 8.1.1.c are înălţimea 5, nodul d este de grad 2, nodul h este terminal, f este un nod intern iar gradul arborelui este 3.

- Dacă $n_1, n_2, \ldots, n_k$ este o **secvenţă** de noduri aparţinând unui arbore, astfel încât n_i este părintele lui n_{i+1} pentru $1 \leq i < k$, atunci această secvenţă se numeşte **"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 (săgeţi) care trebuiesc 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 şi 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 numeşte **strămoş** sau **ancestor** al lui b, iar nodul b **descendent** sau **urmaş** al lui a.

    - Spre exemplu în aceeaşi figură, strămoşii lui f sunt f,b şi a iar descendenţii săi f,k,l,m şi p.

- Conform celor deja precizate **tatăl** unui nod este **strămoşul său direct** (**predecesor**) iar **fiul** unui nod este **descendentul său direct** (**succesor**).

- Un strămos respectiv un descendent al unui nod, altul decât nodul însuşi, se numeşte **strămoş propriu** respectiv **descendent propriu**.

- Într-un arbore, **rădăcina** este **singurul nod** fără **nici** un strămoş propriu.

- Un nod care **nu** are descendenţi proprii se numeşte **nod terminal (frunză).**

- **Înălţimea** unui nod într-un arbore este **lungimea celui mai lung drum** de la **nodul respectiv** la un **nod terminal**.

- Pornind de la această definiţie, **înălţimea** unui arbore se poate defini şi ca fiind egală cu **înălţimea nodului rădăcină**.

- **Adîncimea** unui nod este egală cu lungimea **drumului unic** de la rădăcină la acel nod.

- În practică se mai defineşte şi **lungimea drumului unui arbore** numită şi **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 $\mathbf{P_I}$ se defineşte 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 = înalţimea arborelui

- În scopul definirii **lungimii drumului extern**, ori de câte ori se întâlneşte în arborele iniţial un **subarbore vid**, acesta se înlocuieşte cu un **nod special**.

- În structura care se obţine după această activitate, toate nodurile originale vor fi de acelaşi **grad** (gradul arborelui).

- Se precizează că **nodurile speciale** care au înlocuit ramurile absente **nu** au descendenţi, prin definiţie.

    - Spre exemplu, arborele (a) din figura 8.1.1.d, extins cu noduri speciale reprezentate prin patrate, ia forma (b)..

- **Lungimea drumului extern** se defineşte 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 $\mathbf{P_E}$** este cea definită de formula [8.1.1.d].

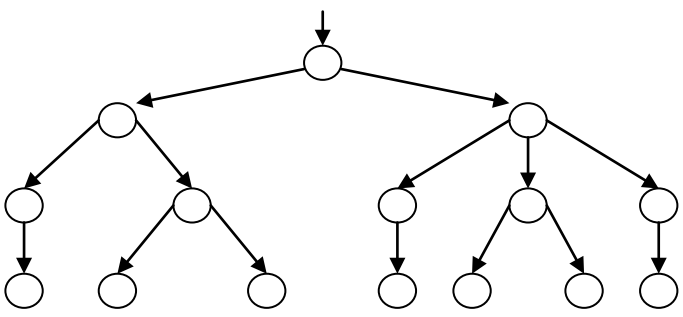
$$P_E = \frac{1}{m}\sum_{i=1}^{h+1} m_i * i$$

unde m = numărul total de noduri speciale

i = nivelul nodului special [8.1.1.d]

n_i =numărul de noduri speciale pe nivelul i

h = înalţimea arborelui



(a)

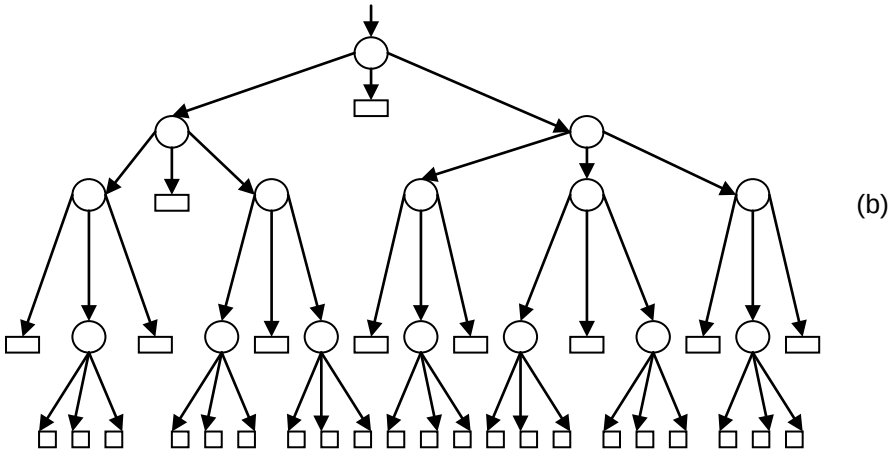


Fig. 8.1.1.d. Arbore ternar extins cu noduri speciale

8.1.2. Tipul de date abstract arbore generalizat

- La fel ca şi în cazul altor tipuri de structuri de date, este dificil de stabilit un set de operatori care să fie valabil pentru toate aplicaţiile posibile ale **structurilor arbore**.

- Cu toate acestea, din mulţimea operatorilor posibili se recomandă pentru **TDA Arbore generalizat** forma prezentată în secvenţa [8.1.2.a].

TDA Arbore generalizat

Modelul matematic: arbore definit în sens matematic.
Elementele arborelui aparţin unui aceluiaşi tip, numit şi *tip de bază*.

Notaţii:

```
TipNod - tipul unui nod al arborelui;
TipArbore - tipul arbore;
TipCheie - tipul cheii unui nod;
TipNod N;
TipArbore A;
TipCheie v;                                    [8.1.2.a]
```

Operatori:

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

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

- operator care returnează nodul care este fratele dreapta *"imediat"* al nodului *N* din arborele *A*. Acest nod se defineşte ca fiind nodul *M* care are acelaşi părinte *T* ca şi *N* şi care în reprezentarea arborelui apare imediat în dreapta lui *N* în ordonarea de la stânga la dreapta a fiilor lui *T*.

4. *TipCheie* ***Cheie****(TipNod N, TipArbore A)*
   - operator care returnează cheia nodului *N* din arborele *A*.

5. *TipArbore* ***Creaza$_i$****(TipCheie v, TipArbore $A_1,A_2,...,A_i$)*
   - este o familie de operatori care are reprezentanţi pentru fiecare din valorile lui *i=0,1,2,..* Funcţia ***Creaza$_i$*** generează un nod nou *R*, care are cheia *v* şi căruia îi asociază *i* fii. Aceştia sunt respectiv subarborii A_1, $A_2,...,A_i$. În final se generează de fapt un arbore nou având rădăcina *R*. Dacă *i=0*, atunci *R* este în acelaşi timp rădăcina şi nod terminal.

6. *TipNod* ***Radacina****(TipArbore A)*
   - operator care returnează nodul care este rădăcina arborelui *A* sau "nodul vid" dacă *A* este un arbore vid.

7. *TipArbore* ***Initializeaza****(TipArbore A)*
   - crează arborele *A* vid.

8. ***Insereaza****(TipNod N, TipArbore A, TipNod T)*
   - operator care realizează inserţia nodului *N* ca fiu al nodului *T* în arborele *A*. În particular se poate preciza şi poziţia nodului în lista de fii ai tatălui *T* (prin convenţie se acceptă sintagma *"cel mai din dreapta fiu"*).

9. ***Suprima****(TipNod N, TipArbore A)*
   - operator care realizează suprimarea nodului *N* şi a descendenţilor săi din arborele *A*. Suprimarea se poate defini diferenţiat funcţie de aplicaţia în care este utilizată structura de date în cauză.

10. ***Inordine****(TipArbore A, TipSubprogram ProcesareNod(...))*
    - operator care parcurge nodurile arborelui *A* în *"inordine"* şi aplică fiecărui nod subprogramul de prelucrare *ProcesareNod*.

11. ***Preordine****(TipArbore A, TipSubprogram ProcesareNod(...))*
    - operator care parcurge nodurile arborelui *A* în *"preordine"* şi aplică fiecărui nod subprogramul de prelucrare *ProcesareNod*.

12. ***Postordine****(TipArbore A, TipSubprogram ProcesareNod(...))*
    - operator care parcurge nodurile arborelui *A* în *"postordine"* şi aplică fiecărui nod subprogramul de prelucrare *ProcesareNod*.

- Structura **arbore generalizat** este **importantă** deoarece apare frecvent în **practică**, spre exemplu arborii familiali, sau structura unei cărţi defalcată pe capitole, secţiuni, paragrafe şi subparagrafe.

- Din punctul de vedere al reprezentării lor în memorie, **arborii generalizaţi** au marele **dezavantaj** că **au noduri de grade diferite**, ceea ce conduce la **structuri neuniforme** de date sau la **structuri uniforme parţial utilizate**.

8.1.3. Traversarea arborilor generalizaţi

- Una din activităţile fundamentale care se execută asupra unei structuri arbore este **traversarea** acesteia.

- Ca şi în cazul listelor liniare, prin **traversarea** unui arbore se înţelege execuţia unei anumite operaţii asupra tuturor nodurilor arborelui.

- În timpul traversării, nodurile sunt **vizitate** într-o anumită **ordine**, astfel încât ele pot fi considerate ca şi cum ar fi integrate într-o listă liniară.

- Descrierea şi înţelegerea celor mai mulţi algoritmi este mult uşurată 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 generalizaţi se încadrează în **două** mari categorii:

    - (1) Tehnici bazate pe **căutarea în adîncime** (**"depth-first search"**)

    - (2) Tehnici bazate pe **căutarea prin cuprindere** (**"breadth-first search"**).

- Aceste tehnici îşi au sorgintea în traversarea structurilor de date **graf**, dar prin particularizare sunt aplicabile şi altor categorii de structuri de date, respectiv structurii de date **arbore generalizat**.

8.1.3.1. Traversarea arborilor generalizaţi prin tehnici bazate pe căutarea în adâncime: preordine, inordine şi 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 proximul punct care permite o nouă posibilitate de înaintare în adâncime.

    - (3) Procesul **continuă** în aceeaşi 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 şi anume:

    - **(1) Traversarea în preordine**

    - **(2) Traversarea în inordine**

    - **(3) Traversarea în postordine**

- Cele trei modalităţi de traversare, se **definesc** de regulă în manieră **recursivă**, asemeni structurii arbore şi anume, **ordonarea** unui arbore se defineşte presupunând că **subarborii** săi s-au ordonat **deja**.

    - Cum subarborii au noduri strict mai puţine decât arborele complet, rezultă că, aplicând recursiv de un număr suficient de ori definiţia, 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 şi postordine se reduce la **lista vidă**.

    - Dacă **A** se reduce la **un singur nod**, atunci nodul însuşi, reprezintă traversarea lui **A**, în oricare din cele trei moduri.

    - Pentru restul cazurilor, fie arborele **A** cu rădăcina **R** şi cu subarborii $\mathbf{A_1, A_2, ..., A_k}$. (fig. 8.1.3.1.a):

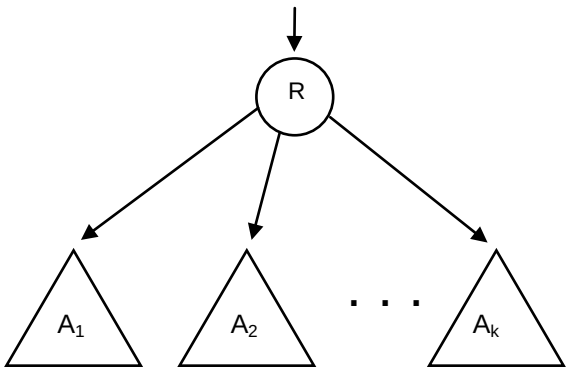


Fig.8.1.3.1.a. Structură de arbore generalizat

1°. Traversarea în **preordine** a arborelui **A** presupune:
- Traversarea rădăcinii **R**
- Traversarea în **preordine** a lui $\mathbf{A_1}$
- Traversarea în **preordine** a lui $\mathbf{A_2, A_3}$ şi aşa mai departe până la $\mathbf{A_k}$ inclusiv.

2°. Traversarea în **inordine** a arborelui **A** presupune:
- Traversarea în **inordine** a subarborelui $\mathbf{A_1}$
- Traversarea nodului **R**

- Traversările în **inordine** ale subarborilor $A_2, A_3, \ldots, A_k$.

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

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

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

```
-------------------------------------------------------------
/*Traversarea arborelui generalizat - modalităţi
principiale*/

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 – varianta
pseudocod*/

subprogram Preordine(TipNod r)
  *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 – varianta
pseudocod*/

subprogram Inordine(TipNod r)
  dacă (r este nod terminal) atunci
      *listează(r);
   altfel                                          [8.1.3.1.b]
      Inordine(cel mai din stânga fiu al lui r);
      *listează(r);
      pentru (fiecare fiu f al lui r, cu excepţia celui
            mai din stânga, în ordine de la stânga spre
            dreapta) execută
         Inordine(f);
      □
   □
-------------------------------------------------------------
/*Traversarea în Postordine a arborelui generalizat –
varianta pseudocod*/

subprogram Postordine(TipNod r)
  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 secvenţe 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.

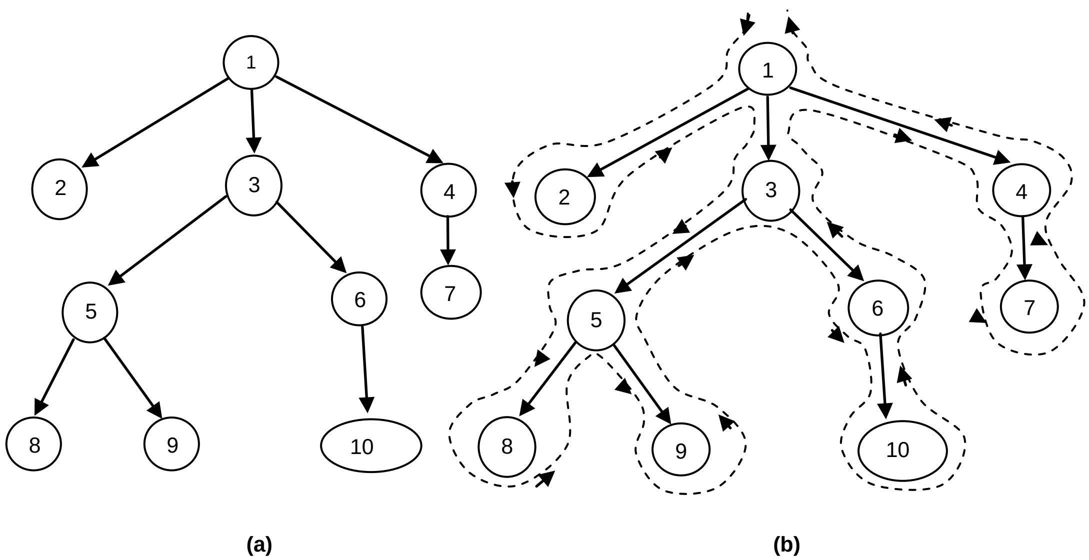


Fig.8.1.3.1.b. Traversarea unui arbore generalizat

- 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 întîlnite, respectiv când sensul de parcurgere este spre părinţii 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 definiţi în cadrul **TDA Arbore generalizat**, varianta recursivă.

- Procedura din secvenţa 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 `tip_nod`.

```
-------------------------------------------------------------
void TraversarePreordine (tip_nod r);

/*Listeaza cheile descendenţilor lui r în preordine*/
/*Implementare bazată pe setul de operatori TDA Arbore
generalizat*/

 {  tip_nod f;
    scrie(Cheie(r,a));
    f= PrimulFiu(r,a);
      while(!f==0)
       {                                    [8.1.3.1.c]
         TraversarePreordine (f);
         f= FrateDreapta(f,a)
       }
  } /*TraversarePreordine*/
□-------------------------------------------------------------□
```

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

- Se presupune că nodurile arborelui sunt de tip `tip_nod`.
- În cadrul variantei nerecursive se utilizează **stiva** `s`, care conţine elemente de `tip_nod`.
- Când se ajunge la prelucrarea nodului `n`, stiva va conţine memorat **drumul** de la rădăcină la nodul `n`, cu rădăcina la baza stivei şi nodul `n` în vârf.
- Procedura care apare în secvenţa [8.1.3.1.d], are **două moduri de acţiune**.
  - (1) În **primul mod**, se descinde de-a lungul celui mai stâng drum neexplorat din cadrul arborelui, tipărind (**prelucrând**) nodurile întâlnite pe drum şi **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 şi 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 şi se termină când stiva devine vidă.

```
-------------------------------------------------------------
void TraversarePreordineNerecursiva(tip_nod a);

/*Implementare nerecursivă bazată pe structura stivă
Se utilizează operatorii TDA Arbore generalizat, TDA Stivă*/

 {  tip_nod m;
    tip_stiva s;
```

```
    boolean gata;

    Initializează(s); /*iniţializare stivă*/
    m= Radacina(a);   /*stabilire nod de pornire - rădăcina*/
    gata= false;
    while(!gata)
      if(!m==0) THEN   /*primul mod de parcurgere*/
          {
            *scrie(Cheie(m,a)); /*prelucare nod*/
            Push(m,s);          /*introducere nod în stivă*/
            m= PrimulFiu(m,a);  /*explorează fiul stâng*/
          }
        else                                        [8.1.3.1.d]
          if Stivid(s)   /*terminare parcurgere*/
              gata= true;
            else       /*modul al doilea de parcurgere*/
              {
                m= FrateDreapta(VarfSt(s),a); /*explorează
                                              frate dreapta*/
                Pop(s);  /*extragere nod din stivă*/
              }
  } /*TraversarePreordineNerecursiva*/
```

- Traversările în preordine şi postordine ale unui arbore sunt utile în obţinerea de **informaţii anterioare** (**ancestrale**) referitoare la noduri.

    - Astfel, se consideră ordinea liniară a nodurilor unui arbore generalizat parcurs în postordine.

    - Se defineşte **POSTORDINE**(n) ca fiind poziţia exprimată ca număr întreg, a nodului n în această ordonare.

    - **POSTORDINE**(n) se numeşte **"număr postordine al lui n"**.

    - Se defineşte de asemenea **DESCENDETI**(n) ca fiind egal cu numărul descendenţilor proprii ai nodului n.

        - Spre exemplu în fig.8.1.3.1.b, numerele postordine ale nodurilor "9", "6" şi "1" sunt respectiv 3,6 şi 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)-**DESCENDENTI**(n)) la **POSTORDINE**(n).

- Astfel, pentru a verifica dacă un nod x este **descendentul** unui nod y, este suficient să se verifice condiţia [8.1.3.1.e]:

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

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

8.1.3.2. Traversarea arborilor generalizaţi prin tehnica căutării prin cuprindere

- Tehnica căutării prin **cuprindere** derivă tot din traversarea **grafurilor**, dar ea este utilizată, e adevarat mai rar, şi la traversarea arborilor [Ko86].

- Se mai numeşte şi traversare pe niveluri (**"level traversal"**) [We94] şi 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 înălţimea arborelui).

- Pentru implementarea acestei tehnici de parcurgere a arborilor generalizaţi, se utilizează drept structură de date suport, **structura coadă**.

- În secvenţa [8.1.3.2.a] apare schiţa de principiu a acestei traversări bazată pe **TDA Coadă.**

```
-----------------------------------------------------------
subprogram TraversarePrinCuprindere1(tip_nod r)

/*Varianta pseudocod bazată pe TDA Coadă*/

  tip_coada coada;
  tip_nod y;
  Initializeaza(coada);
  daca (r nu este nodul vid) atunci
    Adauga(r,coada); /*procesul de amorsare*/
| cât timp NOT Vid(coada)execută                    [8.1.3.2.a]
|   r<-Cap(coada); Scoate(coada); /*scoate nodul din coadă*/
|   *listeaza(r); /*prelucrare nod*/
|   | pentru fiecare fiu y al lui r,(dacă există vreunul),
|   |         în ordine de la stânga la dreapta execută
|   |    Adauga(y,coada); /*adaugă nodul în coadă*/
|   | □
| □
-----------------------------------------------------------
```

Exemplul 8.1.3.2. Implementarea **traversării prin cuprindere** a unui arbore utilizând operatorii definiţi în cadrul **TDA Arbore generalizat** şi **TDA Coadă** este ilustrată în secvenţa [8.1.3.2.b]. Nodurile vizitate sunt afişate.

```
-----------------------------------------------------------
void TraversarePrinCuprindere2(tip_nod r);

/*Implementare bazată pe TDA Arbore generalizat, TDA Coada*/

{   tip_coada coada;
    tip_nod f;

    Initializeaza(coada);
    Adauga(r,coada);
    while (!Vid(coada))
      {
        r=Cap(coada); Scoate(coada);
        *scrie(Cheie(r));
        f=PrimulFiu(r);
```

```
        if (!f==NIL)                                    [8.1.3.2.b]
          {
            Adauga(f,coada);
            f=FrateDreapta(f)
            while (!f==NIL)
              {
                Adauga(f,coada);
                f=FrateDreapta(f)
              }
          }
      }
 } /*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 definiţi în cadrul **TDA Arbore generalizat**.

8.1.4.1. Implementarea arborilor generalizaţi 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 locaţia `A[i]` din tablou.
    - Cea mai simplă manieră de reprezentare a unui arbore, care permite implementarea operaţiei ***`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 numeşte şi **"indicator spre părinte"**.
- Găsirea părintelui unui nod se face într-un interval **constant** de timp **O***(1)* iar parcurgerea arborelui în direcţia nod – rădăcină, se realizează într-un interval de timp proporţional cu adâncimea nodului.

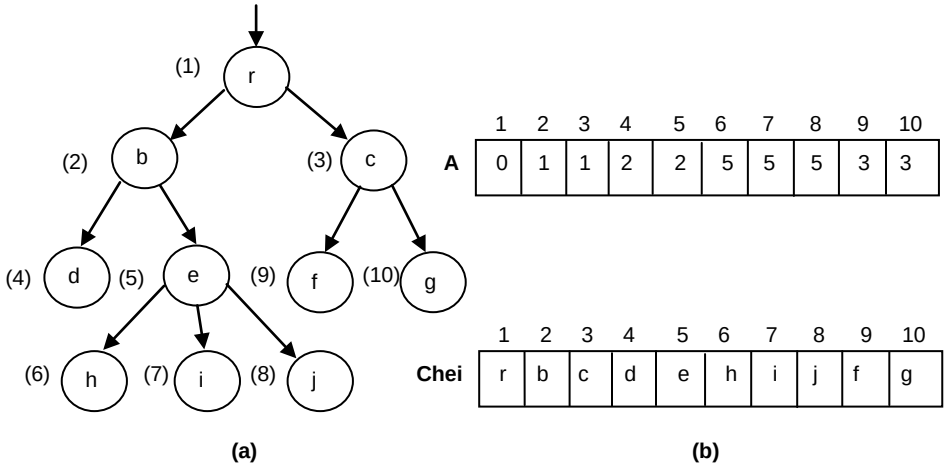


Fig. 8.1.4.1.a. Reprezentarea arborilor generalizaţi cu ajutorul tablourilor (varianta "indicator spre părinte")

- Această reprezentare poate implementa simplu şi operatorul ***Cheie*** dacă se adaugă un alt tablou `Chei`, astfel încât `Chei[i]` este cheia nodului `i`, sau dacă elementele tabloului `A` se definesc de tip articol având câmpurile `cheie` şi `cursor`.

- În figura 8.1.4.1.a apare o astfel de reprezentare (b) a aborelui generalizat (a).

- Reperezentarea "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 înălţimea.

  - În plus, reprezentarea "indicator spre părinte", **nu** permite specificarea ordinii fiilor unui nod, astfel încât operatorii ***PrimulFiu*** şi ***FrateDreapta*** **nu** sunt bine precizaţi.

  - Pentru a da acurateţe reprezentării, se poate impune o **ordine artificială** a nodurilor în tablou, respectând următoarele convenţii:

    (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 poziţii adiacente în tablou.

- În accepţiunea respectării acestor convenţii, în secvenţa [8.1.4.1.a] apare redactată funcţia ***FrateDreapta***.

- Tipurile de date presupuse sunt cele precizate în antetul procedurii. Se presupune că **nodul vid** este reprezentat prin zero.

```
-------------------------------------------------------------
/*Implementarea Arborilor generalizaţi cu ajutorul
tablourilor - varianta "Indicator spre părinte"*/

typedef int tip_nod;
typedef tip_nod tip_arbore[MaxNod];
```

```
/*Exemplu de implementare a operatorului FrateDreapta*/

tip_nod FrateDreapta(tip_nod n, tip_arbore a)

{   tip_nod i,rezultat,parinte;

    parinte=a[n];                          /*[8.1.4.1.a]*/
    rezultat=0;
    i=n;
    do
      {
       i++;
       if (a[i]=parinte) rezultat=i;
      }
    while(!rezultat==0) || (i=MaxNod))
    return rezultat;
 }  /*FrateDreapta*/
```

8.1.4.2. Implementarea arborilor generalizaţi cu ajutorul listelor

- O manieră importantă şi utilă de implementare a arborilor generalizaţi 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 înlănţuite**.

- 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 conţine câte o locaţie pentru fiecare nod al arborelui.

    - Fiecare element al tabloului `inceput` este o referinţă la o **listă înlănţuită simplă**, ale cărei elemente sunt nodurile fii ai nodului corespunzător intrării, adică elementele listei indicate de `inceput[i]` sunt fiii nodului `i.`

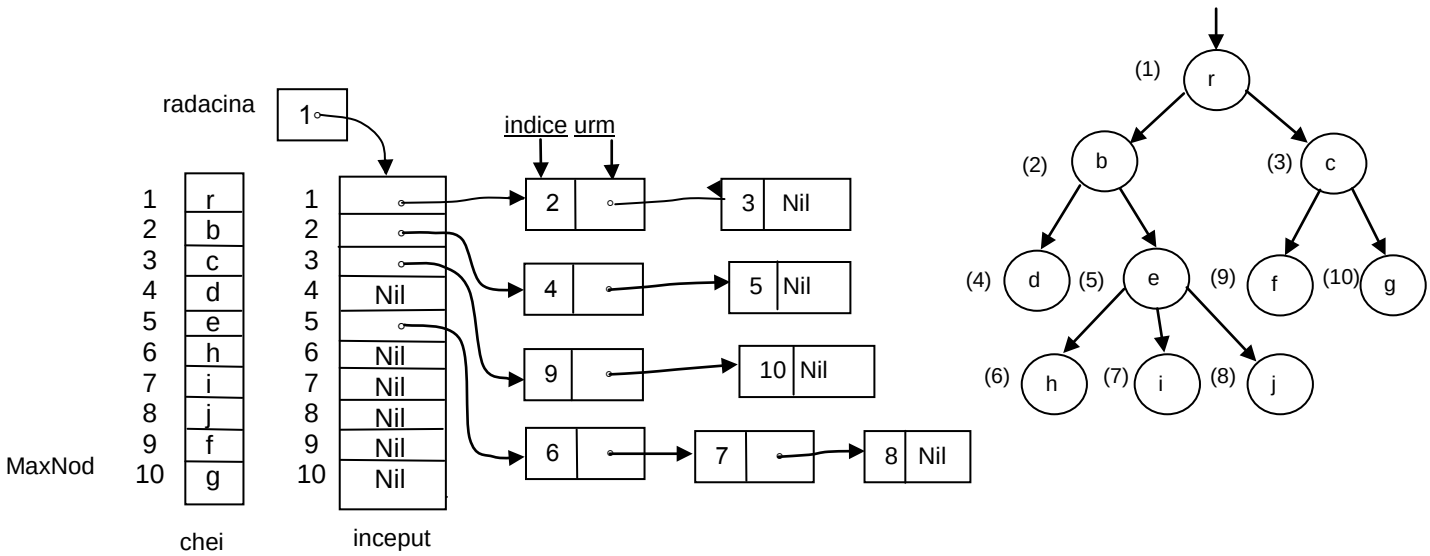


Fig.8.1.4.2.a. Reprezentarea arborilor generalizaţi cu ajutorul listelor înlănţuite

.

- În continuare se prezintă două posibilităţi de implementare.

  - Pentru prima se sugerează tipurile de date prevăzute în secvenţa [8.1.4.2.a]. Este vorba despre o implementare bazată pe liste înlănţuite.

    - Se presupune că rădăcina arborelui este memorată în câmpul specific `radacina`.

    - Nodul vid se reprezintă prin valoarea NIL.

    - În aceste condiţii în secvenţa [8.1.4.2.b] apare implementarea operatorului ***PrimulFiu***.

```
-------------------------------------------------------------
/*Reprezentarea arborilor generalizaţi utilizând liste
înlănţuite simple implementate cu ajutorul pointerilor*/

int MaxNod=...; /*număr maxim noduri*/

typedef struct nod_lista {   /*structura nod listă*/
  int indice;
  struct nod_lista * urm;
} tip_nod;
                                                  [8.1.4.2.a]
typedef struct tip_arbore {   /*structura arbore*/
  tip_nod * inceput[MaxNod];
  tip_cheie chei[MaxNod];
  int radacina;
} tip_arbore

-------------------------------------------------------------
/*Exemplu de implementare al operatorului PrimulFiu}
{se utilizează TDA Listă, varianta restrânsă*/

tip_nod  PrimulFiu(tip_nod n, tip_arbore a)

  { tip_nod * lista, rezultat;
    lista=a.inceput[n];                          [8.1.4.2.b]
    if Fin(lista)   /*n este un nod terminal*/
        rezultat=0;
      else
        rezultat=Furnizeaza(Primul(lista),lista);
  } /*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 şi înlănţuirile din cadrul lor sunt întregi utilizaţi ca şi 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 şi în acest caz tabloul `inceput` care conţine referinţe la listele înlănţuite memorate în tabloul `Zona`, liste ale căror elemente sunt nodurile fii ai nodului corespunzător intrării.

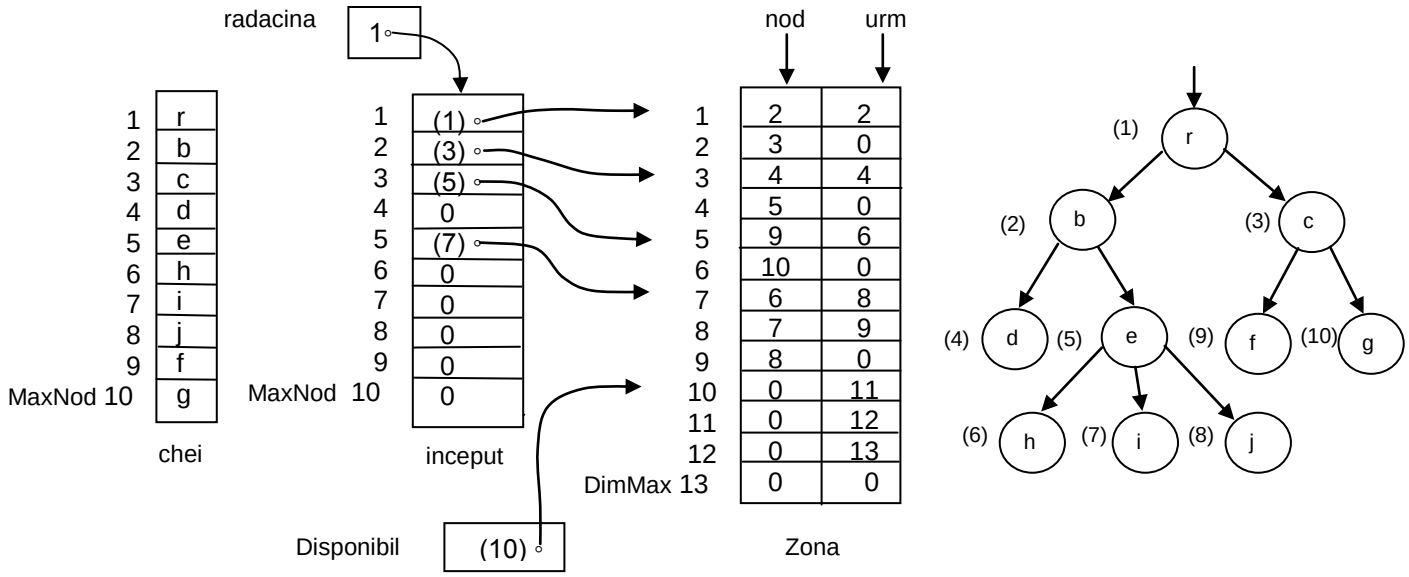


Fig.8.1.4.2.b. Reprezentarea arborilor generalizaţi cu ajutorul listelor înlănţuite implementate cu ajutorul cursorilor

- În secvenţa [8.1.4.2.c] apar structurile de date aferente acestei implementări, iar în secvenţele [8.1.4.2.d,e] implementările operatorilor ***PrimulFiu*** respectiv ***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 aparţine nodul furnizat ca parametru.

```
-----------------------------------------------------------------
/*Reprezentarea arborilor generalizaţi utilizând liste
 înlănţuite implementate cu ajutorul cursorilor*/

int MaxNod=...;  /*număr maxim noduri*/
int DimMax=...;  /*dimensiune tablou Zona*/

typdef int tip_cursor;
typedef int tip_nod;

typedef struct tip_arbore {       /*structura arbore*/
    tip_cursor inceput[MaxNod];
    tip_cheie  chei[MaxNod];                /*[8.1.4.2.c]*/
    tip_nod radacina;
} tip_arbore;

typedef struct element_zona {    /*structura element Zona*/
    tip_nod nod;
    tip_cursor urm;
} element_zona

element_zona Zona[DimMax];      /*definire tablou Zona*/
```

```
tip_cursorDisponibil;  /*cursor inceput lista disponibil*/
tip_arbore R;
-----------------------------------------------------------------
/*Exemplu de implementare al operatorului PrimulFiu*/

tip_nod PrimulFiu(tip_nod n, tip_arbore a)

 {  tip_cursor c; {cursor la începutul listei fiilor lui n}
    tip_nod rezultat;

    c=a.inceput[n];
    if (c=0) {n este nod terminal}  /*[8.1.4.2.d]*/
        rezultat=0;
      else
        rezultat=Zona[c].nod;
    return rezultat;
  } /*PrimulFiu*/
-----------------------------------------------------------------
/*Exemplu de implementare al operatorului Tata*/

tip_nod Tata(tip_nod n, tip_arbore a)
 {
    tip_nod p;     /*parcurge taţii posibili ai lui n*/
    tip_cursor c;  /*parcurge fiii lui p*/
    tip_nod rezultat;

    rezultat=0; p= -1;                /*[8.1.4.2.e]*/
    do {
      p=p+1; c=a.inceput[p];  /*lista fiilor lui p*/
      while ((c!=0) && (rezultat=0)) /*verifică dacă n este
        {                            printre fiii lui p*/
          if (Zona[c].nod=n) rezultat=p;
            else c=Zona[c].urm;
        }
    while((rezultat!=0) || (p=MaxNod-1));
    return rezultat;
  } /*Tata*/
-----------------------------------------------------------------
```

8.1.4.3. Implementarea structurii arbore generalizat pe baza relaţiilor "primul-fiu" şi "frate-dreapta"

- Implementările structurilor arbori generalizaţi, descrise până în prezent, printre alte **dezavantaje**, îl au şi pe acela de a **nu** permite implementarea simplă a operatorului ***Creaza*** şi 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 *TipArbore* ***Creaza***$_2$*(TipCheie v, TipArbore* A_1*,*A_2*)* este necesar ca arborii A_1 şi A_2 să fie recopiaţi într-un al treilea arbore, al cărui tablou de începuturi conţine tablourile celor

doi arbori, completat cu locaţia corespunzătoare lui v indicând o listă de fii care conţine rădăcinile lui A_1 şi A_2.

- Pentru a depăşi această dificultate, pornind de la structura de date implementată în paragraful anterior cu ajutorul cursorilor, se propun două variante de rezolvare.

- **Varianta 1**

  - (1) Tabloul `inceput` se înlocuieşte cu un tablou care cuprinde începuturile listelor de fii pentru **toate nodurile** corespunzătoare tuturor arborilor luaţi în considerare (`ZonaInceputuri` din fig.8.1.4.3.a).
    - Elementele acestui tablou sunt articole cu două câmpuri: `cheie` şi `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 relaţia **"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 **poziţia** acelui nod.
    - Câmpul `urm` din cadrul aceluiaşi tablou realizează înlănţuirea nodurilor fii şi materializează relaţia **"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 colţul stânga sus al figurii.

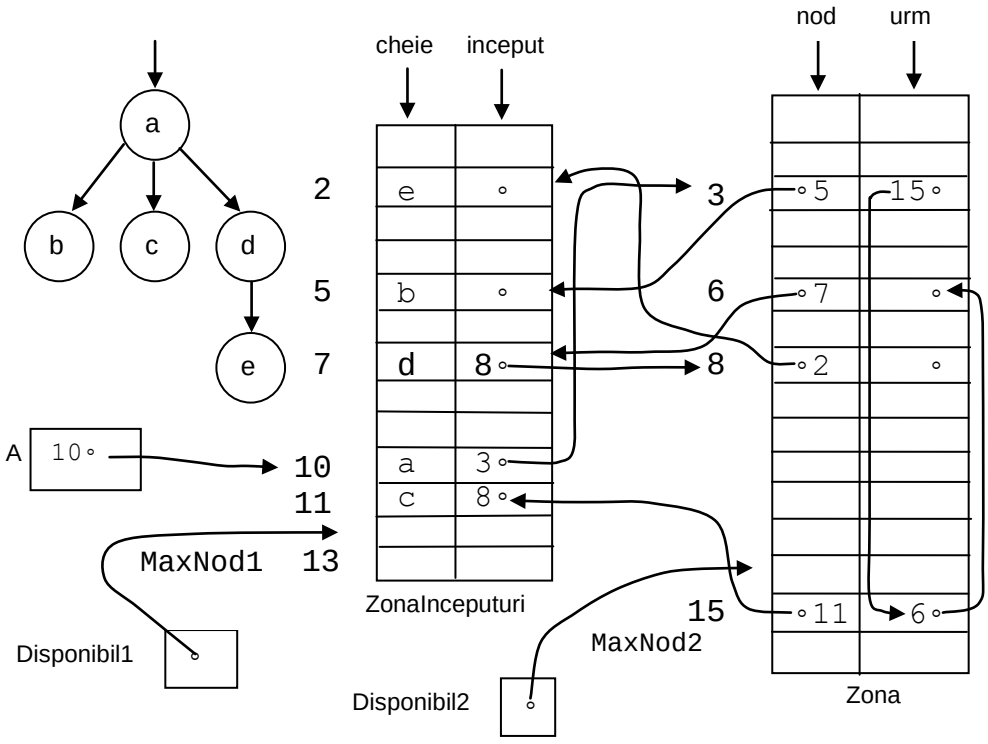


Fig.8.1.4.3.a. Reprezentarea unui arbore generalizat cu ajutorul relaţiilor "primul-fiu" şi "frate-dreapta" (Varianta 1)

- Cheile nodurilor sunt a,b,c,d şi e cărora li s-au atribuit prin mecanismul de alocare poziţiile 10,5,11,7 respectiv 2 în tabloul `ZonaInceputuri`.

- Printr-un mecanism similar au fost atritribuite şi locaţiile tabloului `Zona`.

- În acest scop, cele două tablouri dispun de **liste de disponibili specifice** (`Disponibil1` respectiv `Disponibil2`) care înlăţuite locaţiile libere şi din care se face alocarea spaţiului necesar nodurilor.

- Evident, în aceleaşi liste sunt înlanţuite nodurile disponibilizate.

- Din punct de vedere formal, structurile de date aferente acestei reprezentări apar în secvenţa [8.1.4.3.a].

```
-------------------------------------------------------------
/*Reprezentarea arborilor generalizaţi bazată pe relaţiile
primul-fiu şi frate-dreapta (Varianta 1) C*/

int MaxNod1=...;  /*dimensiune tablou ZonaInceputuri*/
int MaxNod2=...;  /*dimensiune tablou Zona*/

typdef int tip_cursor;
typdef int tip_nod;
typdef tip_cursor tip_arbore;

typedef struct linie_tablou_inceputuri {
    tip_cheie  cheie;
    tip_cursor inceput;
} linie_tablou_inceputuri;                  /*[8.1.4.3.a]*/

typedef struct linie_tablou_zona {
    tip_nod  nod;
    tip_cursor urm;
} linie_tablou_zona;

linie_tablou_inceputuri ZonaInceputuri[MaxNod1];   /*definire
                                  tablou ZonaInceputuri*/
linie_tablou_zona Zona[MaxNod2];    /*definire tablou Zona*/

tip_cursor Disponibil1;  /*cursor inceput lista disponibili a
                           tabloului ZonaInceputuri*/
tip_cursor Disponibil2;  /*cursor inceput lista disponibili a
                           tabloului Zona*/
tip_arbore R;
-------------------------------------------------------------
```

- 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 spaţiul `Zona` al primului fiu al lui `i`. Fie `j` valoarea acestui cursor. În continuare cheia primului fiu se poate afla simplu ştiind că `Zona[j].nod` reprezintă cursorul acestui nod în `ZonaInceputuri`.

- **Varianta 2**

- Pornind de la reprezentarea bazată pe cursori, pentru a simplifica şi mai mult lucrurile se poate renunţa la tabloul `inceput`.

- Astfel, în continuare un nod **nu** va mai fi precizat prin cursorul său în tabloul `inceput` ci prin indexul celulei pe care el o ocupă în tabloul `Zona`.

- Pentru identificarea unui nod, în tabloul `Zona` se adaugă câmpul `cheie`.

- În aceste condiţii, câmpul `urm` devine `frateDreapta` şi va indica direct fratele dreapta al nodului respectiv.

- Pentru a se păstra informaţia referitoare la primul fiu, `Zona` se prelungeşte cu un câmp auxiliar numit `primulFiu`.

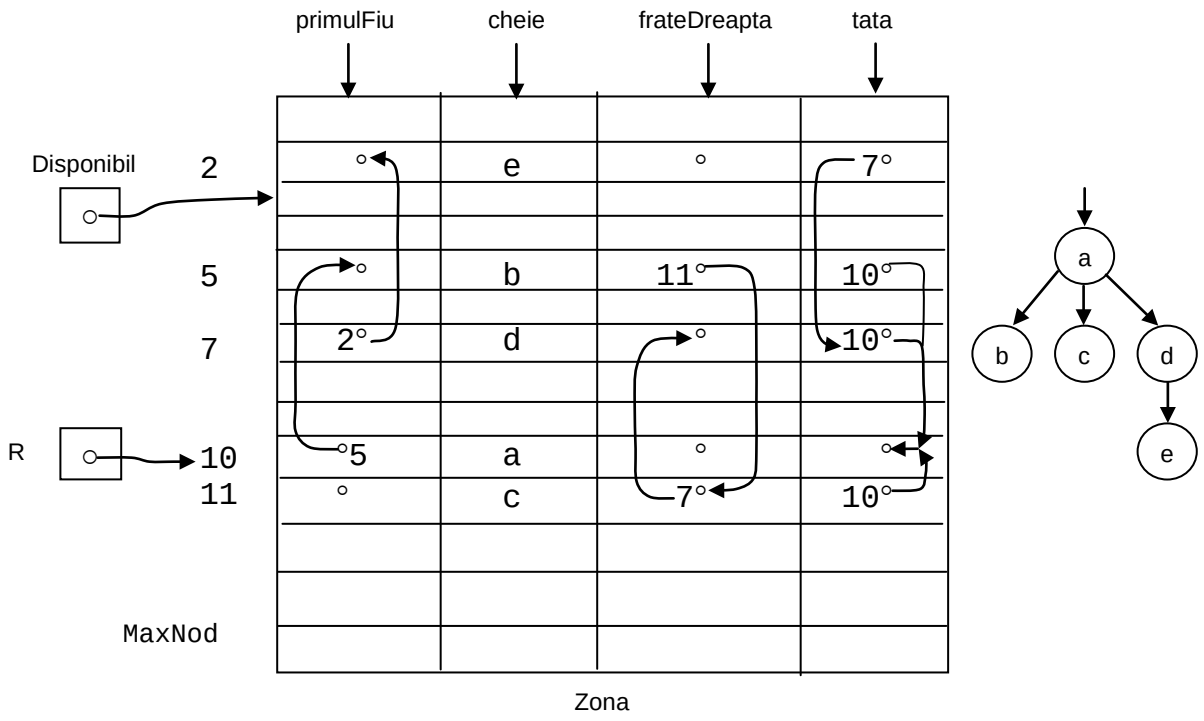


Fig.8.1.4.3.b. Reprezentarea unui arbore generalizat cu ajutorul relaţiilor "primul-fiu" şi "frate-dreapta" (Varianta 2)

- O structură de date încadrată în `TipArbore`, este desemnată în aceste condiţii 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 baleerea integrală a tabloului `Zona`.

- Pentru a implementa în mod eficient şi acest operator se adaugă un al 4-lea câmp (`tata`), care indică direct părintele nodului în cauză.

- În fig.8.1.4.3.b, apare reprezentarea arborelui generalizat din dreapta figurii, iar în secvenţa [8.1.4.3.b] apare definiţia formala a structurii de date corespunzătoare acestui mod de implementare al arborilor generalizaţi.

```
-------------------------------------------------------------
/*Reprezentarea arborilor generalizaţi bazată pe relaţiile
primul-fiu şi frate-dreapta (Varianta 2) C*/

int MaxNod=...;

typdef int tip_cursor;
typdef tip_cursor tip_arbore;

typedef struct linie_tablou {     /*structura linie tablou*/
    tip_cursor primulFiu;
    tip_cheie  cheie;
    tip_cursor frateDreapta;              /*[8.1.4.3.b]*/
    tip_cursor tata;
} linie_tablou;

linie_tablou Zona[MaxNod];     /*definire tablou Zona*/
tip_cursorDisponibil;  /*cursor inceput lista disponibili*/
tip_arbore R;
-------------------------------------------------------------
```

- În secvenţa [8.1.4.3.c] este prezentat un exemplu de implementare a operatorului ***Creaza₂***, 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 înlănţuite prin intermediul câmpului `frateDreapta`.

- Lista `Disponibil` este utilizată pentru alocarea/eliberarea dinamică de către utilizator a nodurilor în tabloul `Zona`.

```
-------------------------------------------------------------
/*Exemplu de implementare al operatorului Creaza2*/

tip_arbore Creaza2(tip_cheie v, tip_arbore t1,t2)
 {
    tip_cursor temp; /*păstrează indexul rădăcinii noului
                       arbore*/
    temp=Disponibil;  /*alocare nod nou*/
    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;
    return temp;
  } /*Creaza2*/
-------------------------------------------------------------
```

8.2. Arbori binari

8.2.1. Definiţii

- Prin arbore binar se înţelege o mulţime de n ≥ 0 noduri care dacă nu este vidă, conţine un anumit nod numit **rădăcină**, iar restul nodurilor formează doi arbori binari disjuncţi numiţi: **subarborele stâng** respectiv **subarborele drept**.

- Ca şi exemple pot fi consideraţi arborii binari reprezentaţi în figura 8.2.1.a.

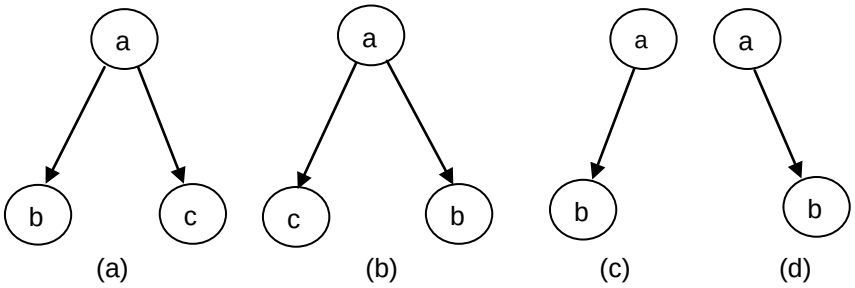


Fig.8.2.1.a. Structuri de arbori binari

- Structurile (a) şi (b) din fig.8.2.1.a. deşi conţin noduri identice, reprezintă arbori binari **diferiţi** deoarece se face o distincţie netă între subarborele stâng şi subarborele drept al unui arbore binar.

- Acest lucru este pus şi mai pregnant în evidenţă de structurile (c) şi (d) din fig.8.2.1.a. care de asemenea reprezintă arbori **diferiţi**.

  - O **expresie aritmetică** obişnuită poate fi reprezentată cu ajutorul unui arbore binar întrucît operatorii aritmetici sunt operatori binari..

  - Spre exemplu, pentru expresia **`(a+b/c)*(d-e*f)`**, arborele binar corespunzător care se mai numeşte şi arbore de parcurgere **("parse tree"),** apare în figura 8.2.1.b.

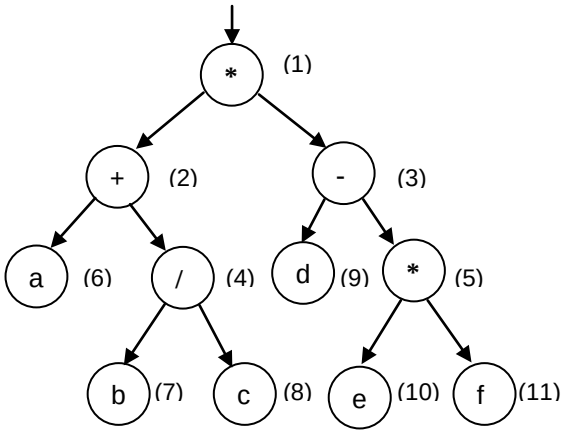


Fig.8.2.1.b. Arbore binar asociat unei expresii aritmetice

- Structura **arbore binar** este deosebit de importantă deoarece:
  - (1) Este simplu de **reprezentat**
  - (2) Este uşor de **prelucrat**, bucurându-se de o serie de proprietăţi 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 şi subarborii A_{11}, $A_{12}...,A_{1k}$.
- **Transformarea** acestuia într-un **arbore binar** se realizează după cum urmează:
  - (1) Se ia A_1 drept rădăcină a arborelui binar.
  - (2) Se face subarborele A_{11} fiul său stâng.
  - (3) Fiecare subarbore A_{1i} se face fiul drept al lui $A_{1,i-1}$ pentru $2 \le i \le k$.
  - (4) Se continuă în aceeaşi manieră transformând după acelaşi algoritm fiecare din subarborii rezultaţi, până la parcurgerea integrală a arborelui iniţial.
- Grafic, această tehnică apare reprezentată în figura 8.2.2.a, (a)-cazul general şi (b)-un exemplu.

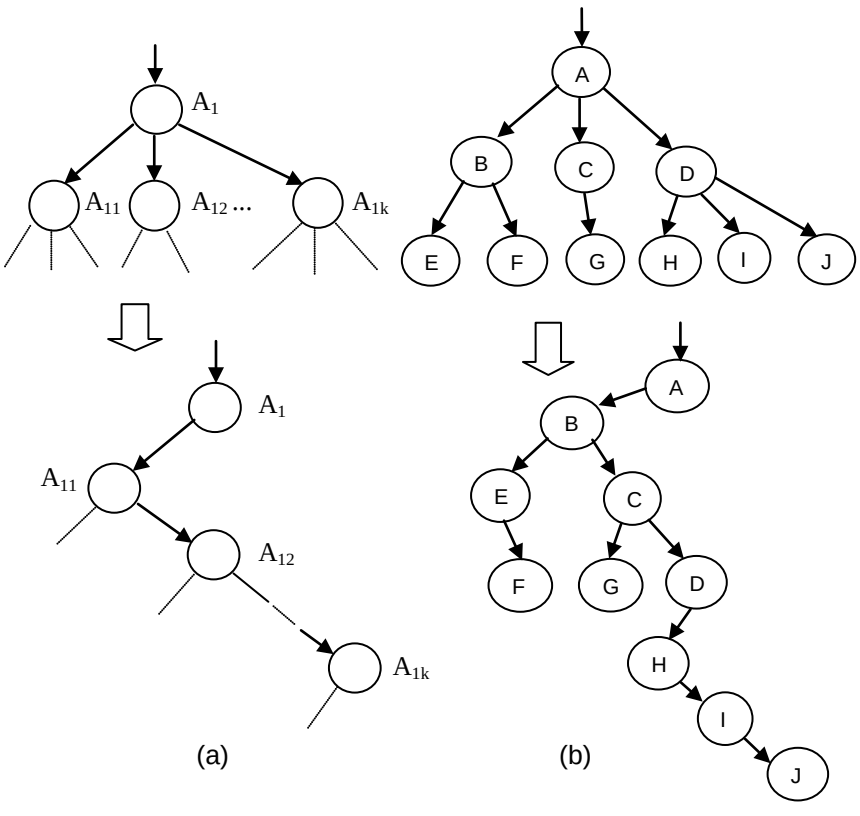


Fig.8.2.2.a. Transformarea unui arbore oarecare în arbore binar

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 şi în cadrul altor TDA-uri, este greu de stabilit un set de operatori care să ofere satisfacţie în toate clasele de aplicaţii posibile.
- În cadrul cursului de faţă 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 acelaşi tip. Fiecare nod este rădacina unui subarbore şi este alcătuit din trei câmpuri: element, indicator stâng şi indicator drept. Indicatorii precizează subarborele stâng respectiv subarborele drept al subarborelui în cauza. În cadrul câmpului element poate exista un câmp cheie care identifică nodul.

Notaţii:

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

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

[8.2.3.a]

Operatori:

1. ***CreazaArboreVid****(TipArboreBinar * t)* - procedură care crează arborele vid *t;*
2. *boolean* ***ArboreVid****(TipArboreBinar t)* - funcţie care returnează valoarea adevărat dacă arborele *t* este vid şi fals în caz contrar.
3. *TipIndicatorNod* ***Radacina****(TipIndicatorNod n)* - returnează indicatorul rădăcinii arborelui binar căruia îi aparţine nodul *n*;
4. *TipIndicatorNod* ***Parinte****(TipIndicatorNod n, TipArboreBinar t)* - returnează indicatorul părintelui(tatălui) nodului *n* aparţinând arborelui *t*;

5. *TipIndicatorNod* ***FiuStanga****(TipIndicatorNod n, TipArboreBinar t)* - returnează indicatorul fiului stâng al nodului *n* aparţinând arborelui *t*;

6. *TipIndicatorNod* ***FiuDreapta****(TipIndicatorNod n, TipArboreBinar t)* - returnează indicatorul fiului drept al nodului *n* aparţinând arborelui *t*;

7. ***SuprimaSub****(TipIndicatorNod n, TipArboreBinar t);* - suprimă subarborele a cărui radacină este nodul *n*, aparţinând arborelui binar *t*;

8. ***InlocuiesteSub****(TipIndicatorNod n, TipArboreBinar r,t)* - inserează arborele binar *r* în *t*, cu rădacina lui *r* localizată în poziţia 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 *adaugă*;

9. *TipArboreBinar* ***Creaza2****(TipArboreBinar s,d, TipIndicatorNod r)* - crează un arbore binar nou care are nodul *r* pe post de rădăcină şi pe *s* şi *d* drept subarbore stâng respectiv subarbore drept

10. *TipNod* ***Furnizeaza****(TipIndicatorNod n, TipArboreBinar t)* - returnează conţinutul nodului indicat de *n* din arborele binar *t*. Se presupune că *n* există în *t*;

11. ***Actualizeaza****(TipIndicatorNod n, TipNod w, TipArboreBinar t);* - înlocuieşte conţinutul nodului indicat de *n* din *t* cu valorea *w*. Se presupune că *n* există.

12. *TipIndicatorNod* ***Cauta****(TipNod w, TipArboreBinar t)* - returnează indicatorul nodului aparţinând arborelui binar t, al cărui conţinut este w;

13. ***Preordine****(TipArboreBinar t, Vizitare(listaArgumente))* - operator care realizează parcurgerea în preordine a arborelui binar *t* aplicând fiecărui nod, procedura *Vizitare*;

14. ***Inordine****(TipArboreBinar t, Vizitare(listaArgumente))* - operator care realizează parcurgerea în inordine a arborelui binar *t* aplicând fiecărui nod, procedura *Vizitare*;

15. ***Postordine****(TipArboreBinar t, 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 aplicaţiile curente se utilizează **două** modalităţi 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

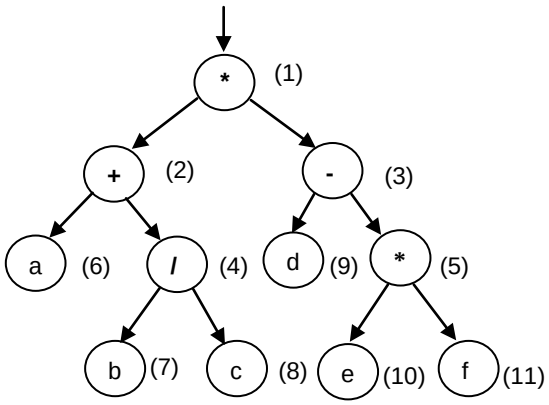
- Pentru implementarea unui arbore binar se poate utiliza o structură **tablou liniar** definit după cum urmează [8.2.4.1.a]:

```
-----------------------------------------------------------------
/*Implementarea unui arbore binar cu ajutorul unei structuri
tablou liniar*/

typedef struct element {
  char op;
  int stang,drept;                              /*[8.2.4.1.a]*/
} tip_element

typedef tip_element tip_arbore[MaxNod];

tip_arbore t;
-----------------------------------------------------------------
```

- 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.



	op	stang	drept
1	*	2	3
2	+	6	4
3	-	9	5
4	/	7	8
5	*	10	11
6	a	0	0
7	b	0	0
8	c	0	0
9	d	0	0
10	e	0	0
11	f	0	0

Fig.8.2.4.1.a. Reprezentarea unui abore 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 înălţime `h` este [8.2.4.1.b].

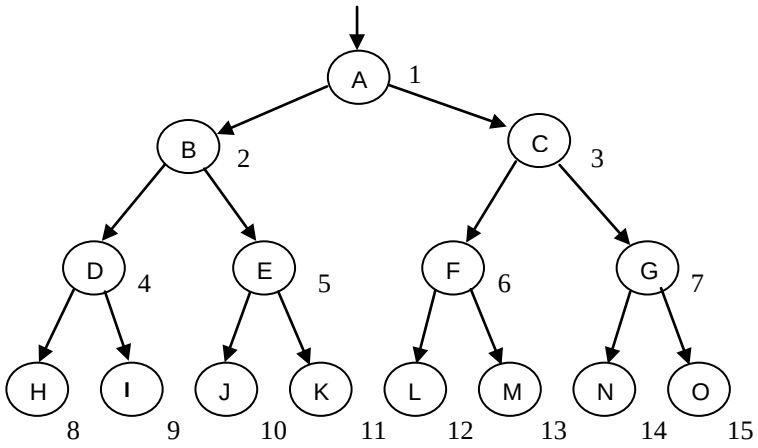
```
-----------------------------------------------------------------
```

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

- Arborele binar de înălţime h care are exact 2^h-1 noduri se numeşte **arbore binar plin de înălţime h.**

- Se **numerotează** secvenţial nodurile unui arbore binar plin de înălţime h, începând cu nodul situat pe **nivelul** 1 şi continuând numărătoarea nivel cu nivel de **sus în jos**, şi 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, locaţia 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 şi reprezentarea sa cu ajutorul unui tablou liniar

- Un arbore binar cu n noduri şi de înălţime h se numeşte **arbore binar complet** dacă nodurile sale corespund nodurilor care sunt numerotate de 1 la n într-un arbore binar plin de înălţime h.

- O consecinţă a acestei definiţii este aceea că într-un **arbore binar complet**, nodurile terminale apar pe **cel mult două niveluri adiacente**.

- Asemeni arborelui binar plin şi arborele complet poate fi memorat într-un **tablou liniar** de dimensiune n.

- Diferenţa dintre un **arbore binar complet** şi un **arbore binar plin**, ambele de înalţime 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 înalţime 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)

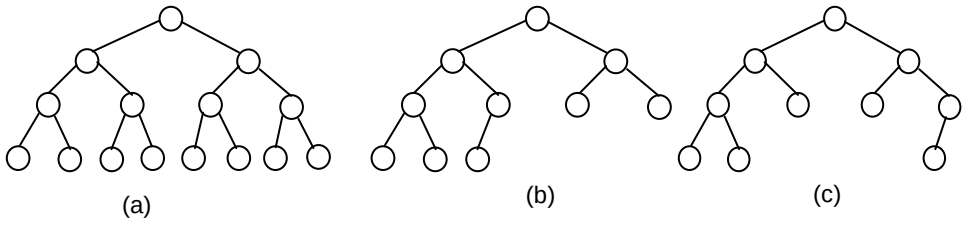


Fig.8.2.4.1.c. Arbori binari:plin (a), complet (b), de înălţime minimă (echilibrat) (c)

- Lema următoare precizează maniera deosebit de simplă în care se poate determina părintele, fiul stâng şi fiul drept al unui nod precizat, fără memorarea explicită a nici unui fel de informaţie de legătură.

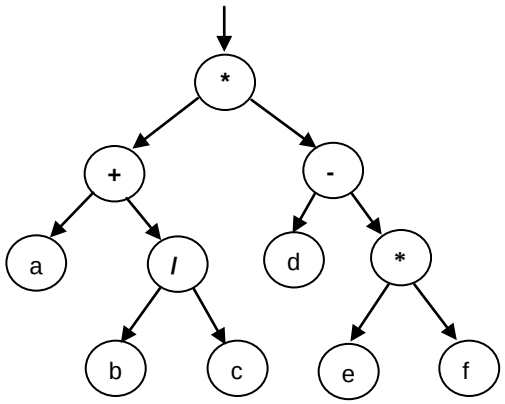
- **Lema 2.** Dacă se reprezintă un **arbore binar complet** într-o manieră conformă cu cele anterior precizate, atunci pentru orice nod având indicele $i, 1 \le i \le n$ sunt valabile relaţiile:

    1. *Parinte(i)* este nodul având indicele $\lfloor i/2 \rfloor$ dacă $i \ne 1$. Dacă $i=1$ nodul indicat de i este nodul rădăcină care nu are părinte.
    2. *FiuStanga(i)* este nodul având indicele $2*i$ dacă $2*i \le n$. Dacă $2*i > n$, atunci nodul i nu are fiu stâng.
    3. *FiuDreapta(i)* este nodul având indicele $2*i+1$ dacă $2*i+1 \le n$. 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 înălţime de arbore binar plin** care "cuprinde" arborele în cauză;
    - Se numerotează şi **nodurile lipsă** ale arborelui de reprezentat, ele fiind înlocuite cu nodul vid Φ în cadrul reprezentării.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
*	+	-	a	/	d	*	Φ	Φ	b	c	Φ	Φ	e	f

Fig.8.2.4.1.d. Reprezentarea unui arbore binar oarecare

- Se remarcă similitudinea dintre acest mod de reprezentare şi reprezentarea **ansamblelor** (Vol.1 &3.2.5).

- Această manieră de reprezentare a arborilor binari este bună pentru **arborii compleţi** şi este în general **ineficientă** pentru toţi ceilalţi.

- De asemenea, ea suferă şi de **neajunsul** specific reprezentărilor bazate pe **tablouri liniare** şi anume, dimensiunea fixă a zonei alocate.

- **Concluzionând**, referitor la această manieră de implementare a arborilor binari se pot face următoarele precizări:

    - **Avantaje:** Simplitate, absenţa legăturilor; parcurgerea simplă a arborelui în ambele sensuri.
    - **Dezavantaje:** Dimensiunea limitată, implementarea relativ complicată a modificărilor (inserţii, suprimări), utilizare ineficientă 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 şi 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 secvenţiale bazate pe structura tablou.

    - Pe de altă parte face uz de proprietăţile **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 C (secvenţa [8.2.4.2.a]) respectiv în variantă Pascal (secvenţa [8.2.4.2.b]).

```
-------------------------------------------------------------
/*Implementarea arborilor binari cu ajutorul pointerilor*/

typedef struct tip_nod
     {
       /*diferite câmpuri*/
       char cheie;
       struct tip_nod* stang;           /*[8.2.4.2.a]*/
       struct tip_nod* drept;
     } tip_nod;

typedef tip_nod * ref_tip_nod;
-------------------------------------------------------------
```

- În fig.8.2.4.2.a se **poate** urmări reprezentarea **arborelui binar** din figura 8.2.4.1.a bazată pe definiţia din secvenţa [8.2.4.2.a.].

- Este uşor de văzut că practic **orice arbore** poate fi reprezentat în acest mod.

- În cadrul cursului de faţă, această modalitate de reprezentare va fi utilizată în mod preponderent.

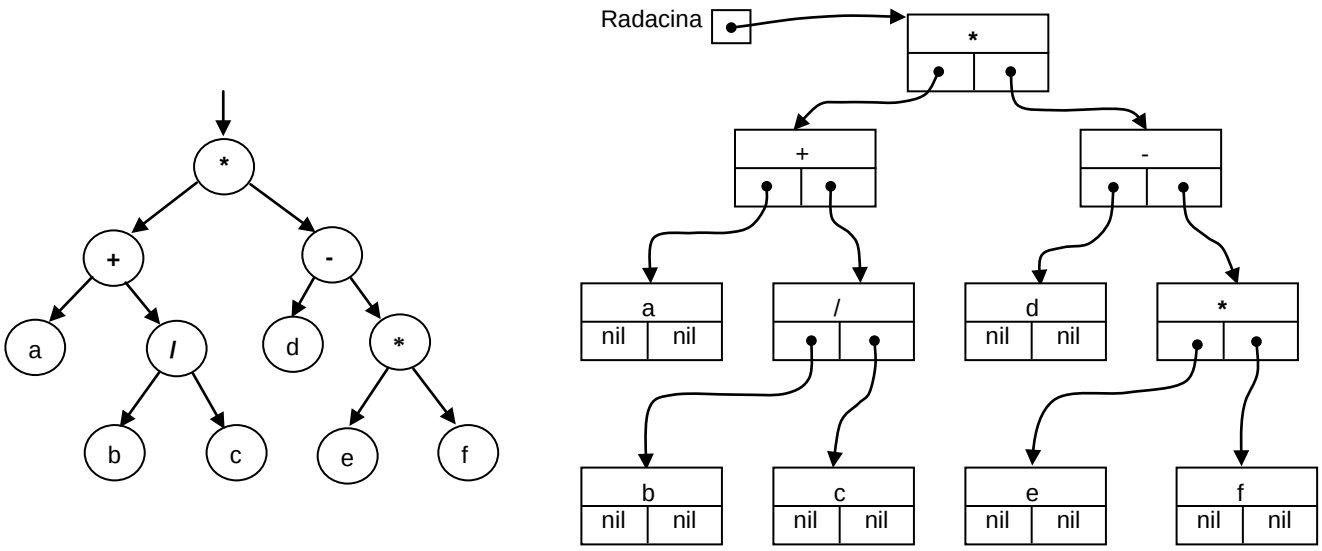


Fig.8.2.4.2.a. Arbore binar reprezentat cu ajutorul pointerilor

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 generalizaţi**.

- În consecinţă şi în acest caz se disting tehnicile bazate pe **căutarea în adâncime** (preordine, inordine şi postordine) precum şi tehnica **căutării prin cuprindere**.

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

- Traversarea în adâncime **(depth first)**, aşa cum s-a precizat şi în cadrul arborilor generalizaţi, 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 proximul nod care are fii neexploraţi şi parcurgerea se reia în aceeaşi 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** şi **postordine**.

- Considerentele avute în vedere la traversarea arborilor generalizaţi rămân valabile şi pentru arborii binari.

- Astfel, referindu-ne la arborele binar din fig.8.2.5.1.a şi considerând pe R drept rădacină, iar pe SS şi SD drept subarborele său stâng, respectiv subarborele său drept, atunci cele **trei moduri de parcurgere** sunt evidenţiate de modelele recursive din secvenţa [8.2.5.1.a].

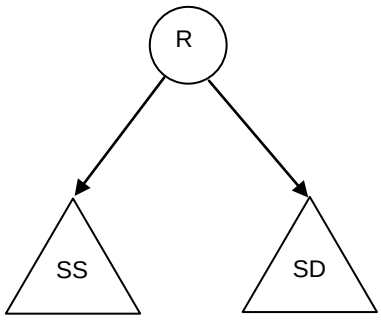


Fig.8.2.5.1.a. Model de arbore binar

```
-------------------------------------------------------------
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 şi înregistrând caracterul corespunzător pe măsură ce sunt vizitate nodurile arborelui, se obţin următoarele ordonări:

  - Preordine: **`*+a/bc-d*ef`**
  - Inordine: **`a+b/c*d-e*f`**
  - Postordine: **`abc/+def*-*`**

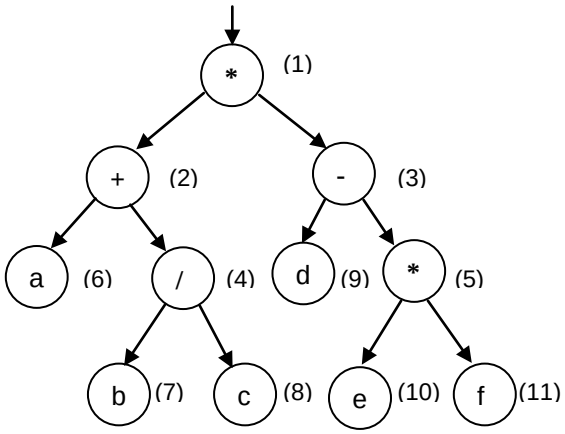


Fig.8.2.1.b. Arbore binar corespunzător unei expresii aritmetice (reluare)

- Se precizează faptul că ultima ordonare este cunoscută în matematică sub denumirea de **notaţie poloneză** (**postfix**).

- Prin scrierea unei expresii aritmetice în **notaţie poloneză** se înţelege scrierea operatorului **după** ce cei doi operanzi, în loc de a-l scrie între ei, ca în notaţia algebrică obişnuită (**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 uşor de observat că **forma poloneză** a unei expresii reprezentate printr-un arbore binar se obţine **parcurgând** arborele în **postordine**.

- În mod similar, în matematică se defineşte **notaţie poloneză inversă** (**prefix**) în care operatorul se scrie **în faţa** celor doi operanzi. De exemplu expresia **a+b** se scrie **+ab**.

- Se observă de asemenea că **notaţia prefix** corespunde **traversării** în **preordine** a arborelui expresiei.

- O proprietate interesantă a notaţiei poloneze este aceea că în ambele forme ea păstrează **semnificaţia** expresiei aritmetice **fără** a utiliza **paranteze**.

- Cu privire la ordonarea în **inordine**, se face precizarea că ea corespunde notaţiei obişnuite a expresiei aritmetice dar cu **omiterea parantezelor**, motiv pentru care semnificaţia matematică a expresiei se pierde, cu toate că arborele binar corespunzător păstrează în structura sa această semnificaţie (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^)` reprezintă **operaţia** care trebuie executată asupra fiecărui nod.

- Considerând pentru structura **arbore binar** definiţia din secvenţa [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*/

typedef struct tip_nod   /*definire structură arbore binar*/
          {
               /*diferite câmpuri*/
               char cheie;
               struct tip_nod* stang;     /*[8.2.5.1.b]*/
               struct tip_nod* drept;
          } tip_nod;
```

```
typedef tip_nod* ref_tip_nod;
-------------------------------------------------------------
void Preordine(ref_tip_nod r)  /*traversare in preordine*/
 {
    if (r!=NULL)
      {
        Viziteaza(*r);
        Preordine((*r).stang);   /* Preordine(r->stang); */
        Preordine((*r).drept);   /* Preordine(r->drept); */
      }
 } /*Preordine*/

void Inordine(ref_tip_nod r)   /*traversare in inordine*/
  {
    if (r!=NULL)
      {
        Inordine((*r).stang);              /*[8.2.5.1.c]*/
        Viziteaza(*r);
        Inordine((*r).drept);
      }
  } /*Inordine*/

void Postordine(ref_tip_nod r)  /*traversare in postordine*/
  {
    if (r!=NULL)
      {
        Postordine((*r).stang);
        Postordine((*r).drept);
        Viziteaza(*r);
      }
  } /*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 ş.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 generalizaţi (secvenţa [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 **evidenţierea nivelurilor arborelui binar**, variantă care utilizează în tandem **două structuri coadă** (secvenţa [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 şi procesul se reia până la parcurgerea integrală a arborelui binar, adică golirea ambelor cozi.

```
-------------------------------------------------------------
Traversarea prin cuprindere unui arbore binar cu
evidenţierea nivelurilor acestuia - varianta pseudocod

subprogram TraversarePrinCuprindereArboreBinar(tip_nod
radacina)

/*Se utilizeaza TDA Arbore binar si TDA Coada*/
{
  tip_coada coada1,coada2;

  r=radacina;
  Initializeaza(coada1); Initializeaza(coada2);
  daca (r nu este nodul vid) atunci
    Adauga(r,coada1); /*procesul de amorsare*/
repetă
   cât timp (NOT Vid(coada1))execută
     r<-Cap(coada1); Scoate(coada1);
     *listeaza(r);
     dacă (FiuStanga(r) nu este nodul vid) atunci
       Adauga(FiuStanga(r),coada2);
     dacă (FiuDreapta(r) nu este nodul vid) atunci
       Adauga(FiuDreapta(r),coada2);
   □
   cât timp (NOT Vid(coada2))execută            [8.2.5.2.a]
     r<-Cap(coada2); Scoate(coada2);
     *listeaza(r);
     dacă (FiuStanga(r) nu este nodul vid) atunci
       Adauga(FiuStanga(r),coada1);
     dacă (FiuDreapta(r) nu este nodul vid) atunci
       Adauga(FiuDreapta(r),coada1);
   □
până când (Vid(coada1) AND Vid(coada2))
□
-------------------------------------------------------------
```

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** şi **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 creştere corespunzătoare a complexităţii algoritmilor de prelucrare.

- O **primă soluţie** 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 înlănţuite**.

- Dacă nodul are o structură suficient de complexă, un câmp de înlănţuire suplimentar **nu** conduce la un consum excesiv de memorie [8.2.6.a].

```
-------------------------------------------------------------
/*Arbori binari cu trei indicatori – structura de date*/

typedef struct tip_nod
     {
       tip_info info;
       struct tip_nod * stang;                /*[8.2.6.a]*/
       struct tip_nod * drept;
       struct tip_nod * parinte;
     } tip_nod;

typedef tip_nod * ref_tip_nod;
-------------------------------------------------------------
```

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

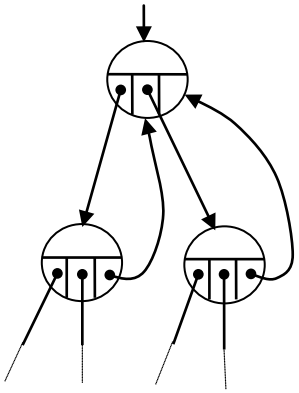


Fig.8.2.6.a. Arbore binar cu legături

- O **altă posibilitate** larg utilizată în implementarea arborilor binari se bazează pe observaţia 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 **înlănţuirii vide pe dreapta** să apară legătura la **nodul următor** la parcurgerea în **inordine** iar în locul **înlănţuirii vide pe stânga** să apară legătura la **nodul anterior** în aceeaşi parcurgere.

  - Întrucât câmpurile indicator au fost **supraîncărcate** (”overloaded”) este necesar ca în structura nodului să fie efectuate modificări care să evidenţieze natura acestor câmpuri: **pointeri** (înlănţuiri) 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 arbore se numeşte **arbore cu legături** (”threaded tree”).

- Un arbore cu legături poate fi:
    - (1) **Parţial** (dacă numai **unul** din câmpurile unui nod 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 secvenţa [8.2.6.1.a].

```
-------------------------------------------------------------
/*Arbori binari cu legături plini  (Fully Threaded Trees)*/

typedef tip_nod_legaturi * ref_tip_nod;

typedef struct tip_nod_legaturi
     {
       tip_info info;
       boolean legatura_stg, legatura_dr;
       ref_tip_nod stang;                  /*[8.2.6.1.a]*/
       ref_tip_nod drept;
     } tip_nod_legaturi;

typdef ref_tip_nod tip_arbore_binar_leg;
-------------------------------------------------------------
```

- Câmpurile booleene `legătura_stg` respectiv `legătura_dr` au valoarea **adevărat** dacă câmpurile indicator corespunzătoare sunt legături respectiv au valoarea **fals** când acestea sunt înlănţuiri.

- 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**.

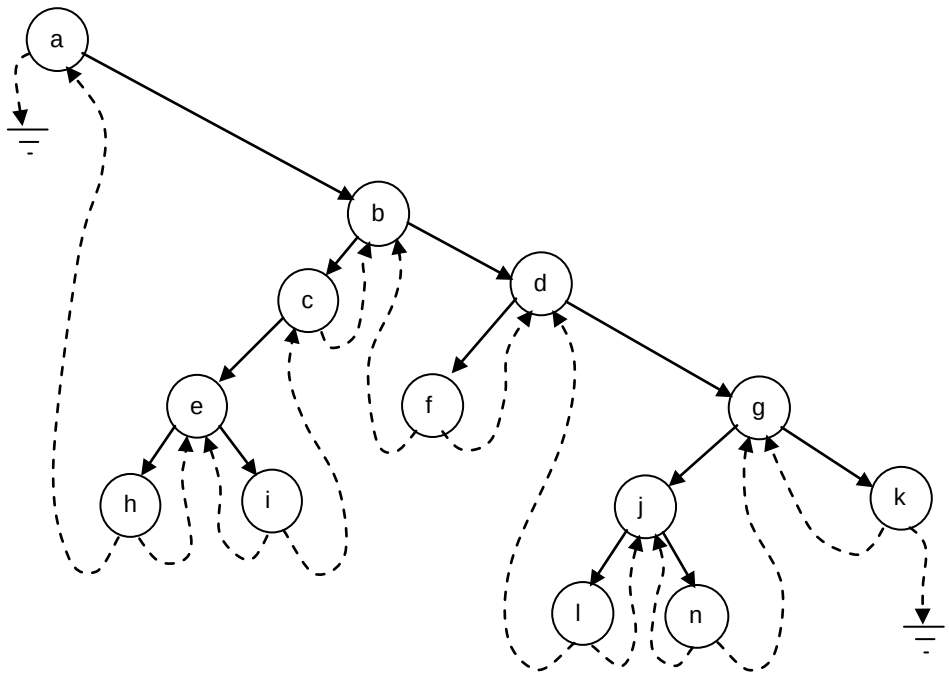


Fig.8.2.6.1.a. Arbore cu legături plin

- Ţinând 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 drepta) 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 acelaşi lucru este valabil şi pentru cel mai din **dreapta nod** al fiecărui **subarbore stâng** (`n` pe `g`).
  - **Cel mai din stânga** şi **cel mai din dreapta** nod al structurii au legătura pe stânga respectiv pe dreapta **vidă** (nodurile `a` şi `k`).
- Se menţionează faptul că **parcurgerea în inordine** poate fi evidenţiată practic simplu realizând **proiecţia 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 aceeaşi manieră a **subarborelui drept**.
  - Se continuă în aceeaşi 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`** şi apare în secvenţa [8.2.6.1.b].

```
-----------------------------------------------------------------
/*Traversarea unui arbore cu legături - Varianta iterativă*/

void InorderIterativ (ref_tip_nod n);
/*Traversează arborele binar cu legături în inordine începând
cu nodul n*/

{
  while (n != NULL) /*până la atingerea celui mai din dreapta
                      nod*/
    {
      while ((!n->legatura_stg) && (n->stang!=NULL))
        n=n->stang; /*avansare pe stânga cât este posibil*/
      viziteaza(n); /*vizitare cel mai din stânga nod*/
      while (n->legatura_dr) /*urcare în arbore în inordine*/
        {
```

```
            n=n->drept;
            viziteaza(n);                    /*[8.2.6.1.b]*/
          } /*while*/
        n=n->drept;   /*se trece la subarborele drept*/
      } /*while*/
} {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** definiţi de TDA arbore binar se implementează într-o **manieră similară** cu reprezentarea bazată pe **pointeri**.

- **Diferenţa** apare la inserţia sau suprimarea unui subarbore.

    - Se consideră spre exemplu operatorul ***InlocuieşteSub(n,r,t)*** unde `n` este de `TipIndicatorNod`, iar `r` şi `t` de `TipArboreBinar`.

    - La iniţiativa acestui operator, **subarborele** având rădăcina `r` trebuie plasat în locul precizat de indicatorul `n`.

    - Pentru simplificarea lucrurilor, se va ignora posibilitatea de a **reutiliza** spaţiul alocat subarborelui indicat de `n` precum şi eventualele supraîncărcări ("aliasing").

    - **Inserţia** subarborelui indicat de `r` presupune de fapt poziţionarea **legăturii** pe **dreapta** a **celui mai din dreapta nod** al subarborelui `r` astfel încât să indice poziţia precizată de nodul similar din subarborele `n` şi în mod analog pentru **cel mai din sânga nod** al subarborelui `r`, după cum rezultă din figura 8.2.6.1.b.

- În secvenţa [8.2.6.1.c] apare implentarea operatorului ***InlocuieşteSub***.

```
/*Operatorul Inlocuieste Subarbore – varianta C*/

void InlocuiesteSub(ref_tip_nod n, tip_arbore_binar_leg r, t)
 {
   ref_tip_nod cel_mai_stg,cel_mai_dr; /*noduri extreme în
                                         subarborele n*/
   ref_tip_nod r_stg,r_dr;/*noduri extreme în subarborele r*/

   /*caută nodurile extreme în arborele original n*/

   /*determinare nod extrem stânga*/
   cel_mai_stg=n;
   do
     cel_mai_stg= cel_mai_stg->stang;
   while ((cel_mai_stg->legatura_stg)||
            (cel_mai_stg->stang == NULL));

   /*determinare nod extrem dreapta*/
   cel_mai_dr=n;
   do                                        /*[8.2.6.1.c]*/
      cel_mai_dr= cel_mai_dr->drept;
```

```
   while ((cel_mai_dr->legatura_dr)||
          (cel_mai_dr->drept == NULL));

   /*înlocuieşte nodul n cu nodul r*/
   n=r;

   /*caută şi modifică legăturile extreme ale lui r*/

   /*modificare legătură extremă pe stânga*/
   r_stg=n;
   while (r_stg->stang!=NULL)
      r_stg=rStg->stang;
   r_stg->stang= cel_mai_stg;
   r_stg->legatura_stg=TRUE;

   /*modificare legătură extremă pe dreapta*/
   r_dr=n;
   while (r_dr->drept!=NULL)
     r_dr=r_dr->drept;
   r_dr->drept= cel_mai_dr;
   r_dr->legatura_dr=TRUE;

  } /*InlocuiesteSub*/
-----------------------------------------------------------------
```

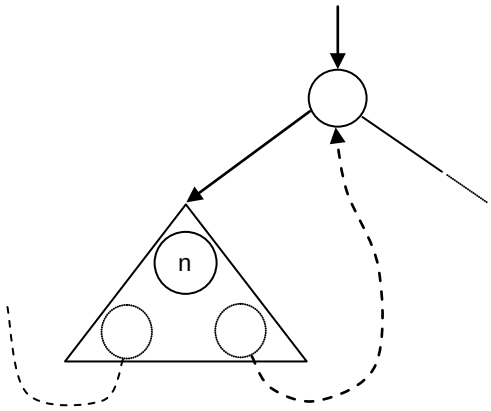


Fig.8.2.6.1.b. Implementarea operatorului `InlocuiesteSub`

- **Revenind** la problema de la care s-a pornit şi anume **aflarea părintelui unui nod precizat**, arborii cu legături pot fi utilizaţi 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 şi părintele acestuia la parcurgerea în **inordine**.
  - În consecinţă **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 dreapt** al tatălui său (fig.8.2.6.1.c.(b)).

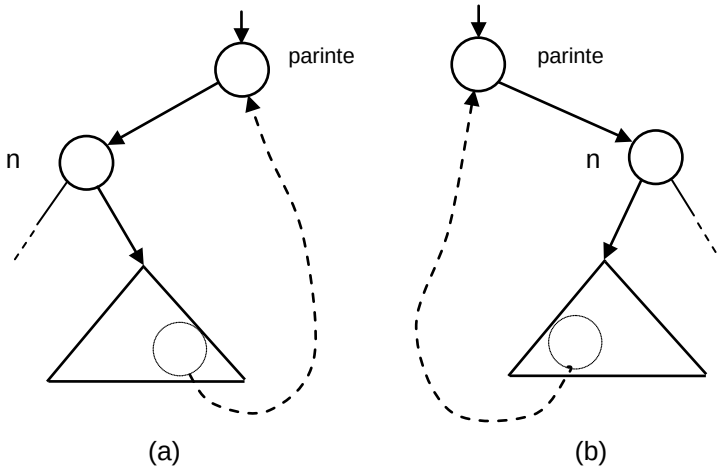


Fig.8.2.6.1.c. Stabilirea părintelui unui nod

- Cu alte cuvinte:
  - Dacă n este un **fiu stâng**, atunci părintele său poate fi găsit urmând înlănţuirile pe **dreapta** cât de departe este posibil şi parcurgând prima legătură pe dreapta.
  - Dacă n este un **fiu dreapta**, atunci părintele său poate fi găsit parcurgând înlănţuirile pe **stânga** cât de departe se poate şi apoi urmând prima legătură pe stânga.
  - Desigur, întrucât iniţial **nu** se cunoaşte dacă n este fiu stâng sau fiu drept al părintelui său, este necesară parcurgerea ambelor posibilităţi
  - **Varianta corectă** se determină verificând dacă nodul la care s-a ajuns are într-adevăr pe n ca fiu stânga respectiv dreapta.

8.2.6.2. Arbori binari cu legături parţiale

- 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) funcţie de sensul parcurgerii arborelui, rezultând astfel un **arbore binar cu legături parţiale**.
- 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 secvenţa [8.2.6.2.a].

```
-------------------------------------------------------------
/*Arbori binari cu legături parţiale*/

typedef tip_nod_legaturi * ref_tip_nod;

typedef struct tip_nod_leg_par
     {
       tip_element element;
```

```
      ref_tip_nod stang;                    /*[8.2.6.1.a]*/
      ref_tip_nod drept;
      boolean legatura;
    } tip_nod_leg_par;

typdef ref_tip_nod tip_arbore_binar_leg_par;
-------------------------------------------------------------
```

- Câmpul `legatura` se referă la câmpul `drept`.

- Astfel dacă `p` este de tip `ref_tip_nod` si `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).

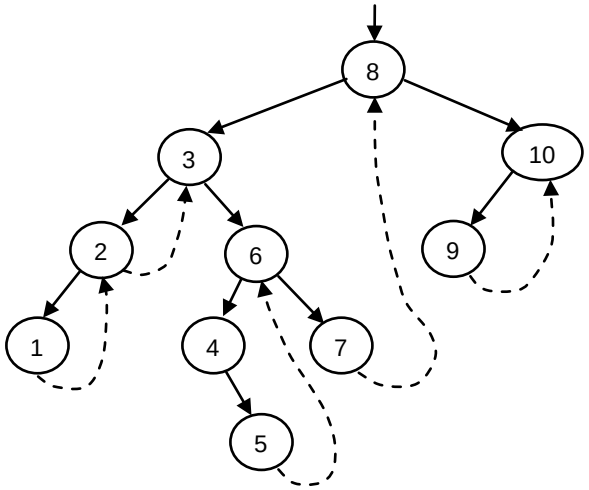


Fig.8.2.6.2.a. Arbore binar cu legături parţiale

- Această reprezentare suportă **implementarea** simplă (eventual cu anumite restricţii) a majorităţii operatorilor definiţi pe **TDA arbore binar**.

- Implementarea TDA arbore binar, pornind de la aceste considerente, poate fi considerată un **exerciţiu** incitant şi în acelaşi timp util.

8.2.7 Aplicaţii ale arborilor binari

- Dintre aplicaţiile specifice arborilor binari se prezintă câteva considerate mai reperezentative:

  - Construcţia unui arbore binar de **înalţime minină**

  - Generarea unui arbore binar pornind de la specificarea sa în **preordine**

  - Construcţia unui **arbore de parcurgere** pornind de la forma postfix a expresiei asociate

8.2.7.1. Construcţia şi reprezentarea grafică a unui arbore binar de înălţime minimă

- Se presupune că arborele binar ce trebuie construit conţine noduri încadrate în tipul clasic definit în secvenţa [8.2.4.2.b], în care cheile nodurilor sunt `n` **numere** care **se citesc** de la tastatură.

- Restricţia care se impune este aceea, ca arborele construit să aibă **înălţimea minimă.**

- Pentru aceasta este necesar ca **fiecărui nivel** al structurii arborelui care se construieşte să-i fie alocate **numărul maxim** de noduri posibile, cu excepţia 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 distribuţia 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 `ns=n DIV 2` noduri.

    3° Se generează subarborele drept cu `nd=n-ns-1` noduri.

- Acest algoritm permite construcţia simplă a unui arbore binar de înalţime minimă.

- Un **arbore binar** este considerat **perfect echilibrat** dacă pentru fiecare nod al său, numărul de noduri al subarborelui stâng diferă de cel al subarborelui drept cu **cel mult** o unitate.

- În mod evident că **arborele de înălţime minimă** construit de către acest algoritm este unul **perfect echilbrat**.

- Implementarea algoritmului **varianta C** apare în secvenţa [8.2.7.1.a].

    - Cheile nodurilor se introduc de la tastatură.

    - Prima valoare introdusă este numărul total de noduri `n`.

    - Sarcina construcţiei efective a arborelui binar revine funcţiei **`Arbore`** care:

        - Primeşte ca parametru de intrare numărul de noduri `n`.
        - Determină numărul de noduri pentru cei doi subarbori.
        - Citeşte cheia nodului rădăcină.
        - Construieşte în manieră recursivă arborele.
        - Returnează referinţa la arborele binar construit.

```
-------------------------------------------------------------
//Construcţia unui arbore binar de înălţime minimă

#include "stdafx.h"
#include <stdlib.h>

typedef struct nod
{
      int cheie;
      struct nod* stang;
```

```
	struct nod* drept;
} NOD;

int n;
NOD* radacina;

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

	if (n==0)  //arborele vid
		return NULL;

	ns= n/2;    //determinare număr noduri subarbore stâng
	nd= n-ns-1; //determinare număr noduri subarbore drept
	scanf_s("%d", &x); //citire cheie rădăcină

	//alocare memorie nod rădăcină arbore
	if ((NodNou = (NOD *)malloc(sizeof(NOD))) == NULL)
	{
		printf("Eroare la malloc");
		return NULL;
	}
	//completare conţinut nod rădăcină
	NodNou->cheie = x;
	NodNou->stang = Arbore(ns);  //completare înlănţuire
	NodNou->drept = Arbore(nd);  //completare înlănţuire
	return NodNou;
} //Arbore

void Afiseaza_Arbore(NOD* t, int h)
{
	//afişează 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_s("%d", &n); //numărul total de noduri
	radacina = Arbore(n); //construcţie arbore
	Afiseaza_Arbore(radacina, 50);
	return 0;
} //main
```

- Funcţia **`Afiseaza_Arbore`** realizează **reprezentarea grafică** a unei structuri de arbore binar răsturnat (rotit cu 90^o în sensul acelor de ceasornic), conform următoarei **specificaţii**:

1° Pe fiecare rând se afişează **exact un nod**

2° Nodurile sunt ordonate de sus în jos în **inordine**

3° Dacă un nod se afişează pe un rând precedat de h blancuri, atunci fiii săi (dacă există) se afişează precedaţi de h-d blancuri. d este o variabilă a cărei valoare este stabilită de către programator şi precizează **distanţa** dintre nivelurile arborelui măsurată în caractere "spaţiu"

4° Pentru **rădăcină**, se alege o valoare corespunzătoare a lui h, ţinând cont că arborele se dezvoltă spre stânga.

- Această modalitate de reprezentare grafică a arborilor binari este una dintre cele mai **simple** şi **imediate**, ea recomandându-se în mod deosebit în aplicaţiile didactice.

- Trebuie subliniat faptul că simplitatea şi transparenţa acestui program rezultă din utilizarea **recursivităţii**.

  - Aceasta reprezintă un **argument** în plus adus afirmaţiei, 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 funcţia care **construieşte** arborele cât şi de către procedura de **afişare** 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 faţă, îl constituie generarea unui arbore binar cu formă dirijată.

- Specificarea arborelui se face în **preordine**, enumerând nodurile (reprezentate în cazul de faţă printr-un singur caracter) începând cu **rădăcina** urmată mai întâi de **subarborele stâng** apoi de cel **drept**.

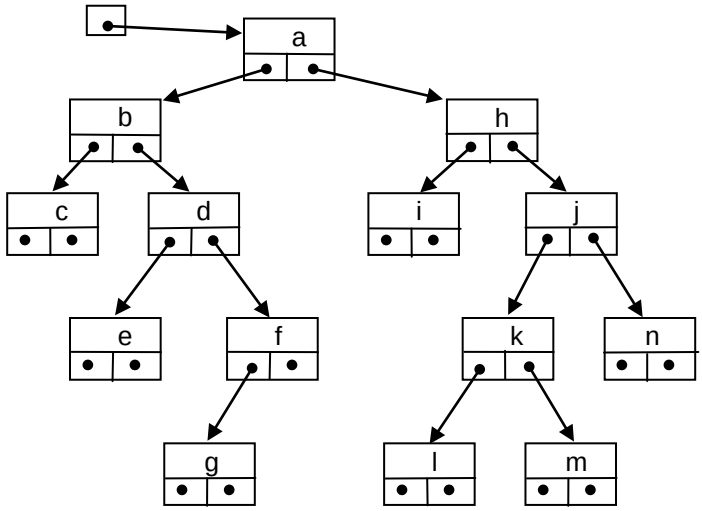


Fig.8.2.7.2.a. Structură arbore binar

- 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.

- Procedura care construieşte o astfel de structură pornind de la specificarea arborelui în preordine precizată printr-un şir de caractere apare în [8.2.7.2.a]

- Se precizează că sunt valabile structurile de date precizate în secvenţa [8.2.4.2.a].

```
-------------------------------------------------------------
/*Generarea unui AB pornind de la specificarea sa în
preordine*/

/*Generarea unui AB pornind de la specificarea sa în
preordine*/

typedef struct tip_nod
          {
               /*diferite campuri*/
               char cheie;
               struct tip_nod* stang;
               struct tip_nod* drept;
          } tip_nod;

typedef tip_nod * ref_tip_nod;

/*Procedura de creare a AB*/

void Creare(ref_tip_nod * p);
  {
    char ch;
    scanf("%c", &ch);
    if (ch!='.')                              /*[8.2.7.2.a]*/
       {
         (*p)=(tip_nod *)malloc(sizeof(tip_nod); /*completare
                                              înlănţuire*/
         (*p)->cheie=ch;     /*asignare cheie nod curent*/
         Creare(&(*p)->stang);     /*creare subarbore stâng*/
         Creare(&(*p)->drept);     /*creare subarbore drept*/
       } /*if*/
      else
        (*p)=NULL;

  } /*Creare*/
-------------------------------------------------------------
```

8.2.7.3. Construcţia 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 faţă se pune problema de a **construi** un astfel de arbore binar asociat unei **expresii aritmetice**.

- Algoritmul precizat de secvenţa [8.2.7.3.a] realizează acest lucru pornind de la notaţia **postfix** a expresiei.

    - Se presupune ca forma **postfix** a expresiei aritmetice apare ca un şir de caractere care este citit element cu element **de la dreapta la stânga** de către procedură [De89].

```
subprogram ConstrArbParcurgere(ref_tip_nod * p)

/*Construcţia unui arbore de parcurgere pentru o expresie
aritmetică pornind de la forma postfix a expresiei. Varianta
pseudocod. Expresia se parcurge de la dreapta la stânga*/

    tip_element t;
    citeste_element(t);  /*citeşte elementul curent şi
       avansează la elementul următor de la dreapta la
       stânga*/
    daca (t<>inceput)
     |*generează un nod nou indicat de p;
     |*plasează pe t în noul nod;
     |daca (t este un operator)                [8.2.7.3.a]
     |    |/*construcţie recursivă*/
     |    |ConstrArbParcurgere(&(*p)->drept);
     |    |ConstrArbParcurgere(&(*p)->stang)
     |    |□ /*daca*/
     |   altfel
     |    |/*elementele non operator sunt întotdeauna
     |    |          frunze*/
     |    |*se face subarborele drept al lui p, vid;
     |    |*se face subarborele stang al lui p, vid
     |    |□ /*altfel*/
     |□ /*daca*/
/*ConstrArbParcurgere*/
```
