8. Arbori

8.1. Arbori generalizați

8.1.1. Definiții

- În definirea **noțiunii de arbore** se pornește de la noțiunea de **vector**.

  - Fie **V** o mulțime având elementele a_1, a_2,....a_n.

  - Pe mulțimea **V** se poate defini o așa numită **"relație de precedență"** în felul următor: se spune că a_i precede pe a_j dacă i < j. Aceasta se notează: $a_i \prec a_j$.

  - Se poate verifica ușor că relația astfel definită are următoarele proprietăți, valabile pentru structura de vector:

---

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

- Din proprietățile (1) și (2) rezultă că relația de precedență **nu** determină în **V** **"bucle închise"**, adică **nu** există nici o secvență de elemente care se preced două câte două și în care ultimul element este același cu primul, cum ar fi de exemplu $a \prec b \prec c \prec d \prec a$.

- Proprietatea (3) precizează că relația de precedență este definită pentru **oricare** două elemente a și b ale lui **V**, cu singura condiție ca $a \neq b$.

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

  - Dacă această relație posedă proprietățile [8.1.1.a], atunci ea **imprima** peste mulțimea **V** o **structură de vector** (fig.8.1.1.a).

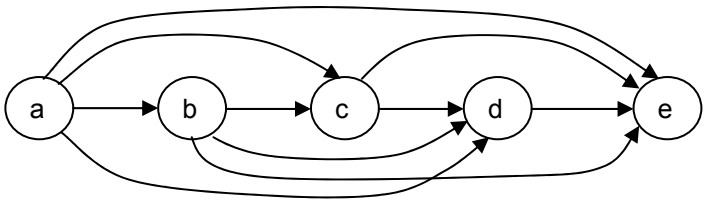

Fig. 8.1.1.a. Structură de vector

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

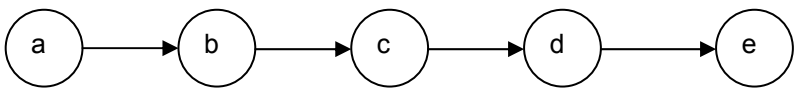

Fig.8.1.1.b. Relația succesor

- Această relație se definește cu ajutorul relației de precedență după cum urmează: dacă între elementele a şi b ale lui **V** este valabilă relația $a \prec b$ 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 relația "**succesor**" (mulțimea săgeților din figura 8.1.1.b.), **precizează** relația "**precedență**" fără a fi însă **identică** cu ea. Spre exemplu, există relația $b \prec e$ (prin tranzitivitate), dar nici o săgeată nu conectează pe b cu e.

- În figura8.1.1.c apare o aşa numită **structură de arbore** care se definește prin **generalizarea** structurii de **vector**.

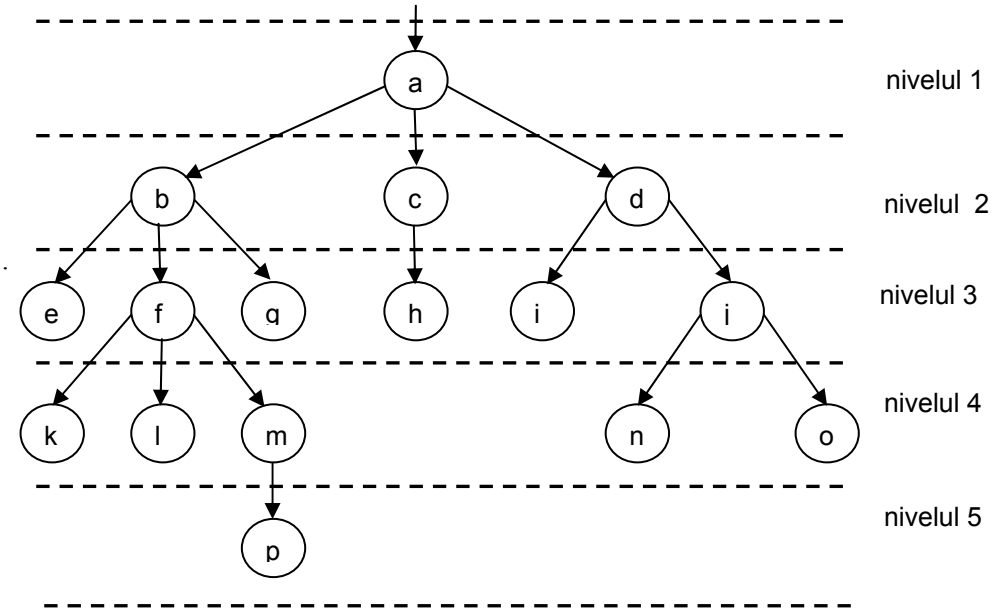

Fig. 8.1.1.c. Structură de arbore

- Astfel, dacă în cazul **vectorului**, **toate elementele** cu excepția ultimului au **exact** un **succesor**, la **structura de arbore** se admite ca **fiecare element** să aibă un **număr oarecare** de **succesori**, inclusiv zero, cu **restricția** ca două elemente distincte să **nu** aibă acelaşi succesor.
- Relația **succesor** definește o relație de **precedență** pe structura de arbore. Astfel, din figura avem $b \prec p$, $d \prec n$, etc.

- **Relația de precedență** definită pe structura de arbore se bucură de proprietățile (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 relațiile $b \prec c$ sau $c \prec b$, la fel pentru elementele d şi k. Prin urmare, relația de precedență este definită numai pentru **o parte** a perechilor de elemente ale arborelui, cu alte cuvinte este o **relație parțială**.

- În general, o **structură de arbore** se definește ca o mulțime **A** de $n \leq 0$ noduri de acelaşi tip, peste care s-a definit o relație de precedență având proprietățile (1) şi (2) de la [8.1.1.a] precum şi următoarele două proprietăți [8.1.1b]:

    - (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 relația de precedeță **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 de arbore se poate formula şi o altă definiție echivalentă cu cea de mai sus.

- Prin **arbore**, se înțelege o mulțime 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 **disjuncți**.

- Se constată că atât această definiție, 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 **noțiuni** referitoare la arbori.

- Prin **subarborii** unui arbore, se înțeleg arborii în care se descompune acesta prin îndepărtarea rădăcinii.

    - Spre exemplu arborele din figura 8.1.1.c, după îndepărtarea rădăcinii a, se descompune în trei subarbori având rădăcinile respectiv b,c şi d.

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

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

- Un arbore parțial **nu** este întotodeauna **subarbore** pentru arborele complet, dar orice **subarbore** este un **arbore parțial**.

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

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

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

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

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

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

- Într-o structură de 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 **înălțimea** 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 înălțimea 5, nodul d este de grad 2, nodul h este terminal, f este un nod intern iar gradul arborelui este 3.

- Dacă n_1, n_2,....,n_k este o **secvență** de noduri aparținând unui arbore, astfel încât n_i este părintele lui n_{i+1} pentru $1 \leq i < k$, atunci această secvență se 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 (săgeți) 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 descendenții 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 descendenți proprii se numeşte **nod terminal (frunză).**

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

- Pornind de la această definiție, **înălțimea** unui arbore se poate defini şi ca fiind egală cu **înălțimea 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 = înalțimea arborelui

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 aplicațiile posibile ale structurilor de **tip arbore**.

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

TDA Arbore generalizat

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

Notații:

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

Operatori:

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

2. ***PrimulFiu****(N: TipNod, A: TipArbore): TipNod;* - operator care returnează cel mai din stânga fiu al nodului *N* din arborele *A*. Dacă *N* este un nod terminal, operatorul returnează "nodul vid".

3. ***FrateDreapta****(N: TipNod, A:TipArbore): TipNod;* operator care returnează nodul care este fratele drept *"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. ***Cheie****(N: TipNod, A: TipArbore): TipCheie;* - operator care returnează cheia nodului *N* din arborele *A*.

5. ***Creeaza$_i$***`(v: TipCheie, A`$_1$`,A`$_2$`,...,A`$_i$`: TipArbore): TipArbore;` este unul din operatorii unei familii, care are reprezentanți pentru fiecare din valorile lui *i=0,1,2,..* Funcția ***Creeaza$_i$*** 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. ***Radacina****(A: TipArbore): TipNod;* - operator care returnează nodul care este rădăcina arborelui *A* sau "nodul vid" dacă *A* este un arbore vid.

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

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

9. ***Suprima****(N: TipNod, A: TipArbore);* - operator care realizează suprimarea nodului *N* şi a descendenților săi din arborele *A*. Suprimarea se poate defini diferențiat funcție de aplicația în care este utilizată structura de date în cauză.

10. ***Inordine****(A: TipArbore, ProcesareNod(...): TipProcedura);* - procedură care parcurge nodurile arborelui *A* în *"inordine"* şi aplică fiecărui nod procedura de prelucrare *ProcesareNod*.

11. ***Preordine****(A: TipArbore, ProcesareNod(...): TipProcedura);* - procedură care parcurge nodurile arborelui *A* în *"preordine"* şi aplică fiecărui nod procedura de prelucrare *ProcesareNod*.

12. ***Postordine****(A: TipArbore, ProcesareNod(...): TipProcedura);* - procedură care parcurge nodurile arborelui *A* în *"postordine"* şi aplică fiecărui nod procedura de prelucrare *ProcesareNod*.

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

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

8.1.3. Traversarea arborilor generalizați

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

- Ca şi în cazul listelor liniare, prin **traversarea** unui arbore se înțelege execuția unei anumite operații asupra tuturor nodurilor arborelui.

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

- Descrierea şi înțelegerea celor mai mulți 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 generalizați 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 generalizați 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 cînd 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 modalități de traversare, se **definesc** de regulă în manieră **recursivă**, asemeni structurii de arbore şi anume, **ordonarea** unui arbore se defineşte presupunând că **subarborii** săi s-au ordonat **deja**.

    - Cum subarborii au noduri strict mai puține decât arborele complet, rezultă că, aplicând recursiv de un număr suficient de ori definiția, se ajunge la ordonarea unui arbore vid, care este evidentă.

- Cele trei moduri de traversare ale unui arbore generalizat **A** se definesc recursiv după cum urmează:

    - Dacă arborele **A** este **nul**, atunci ordonarea lui **A** în preordine, inordine 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):

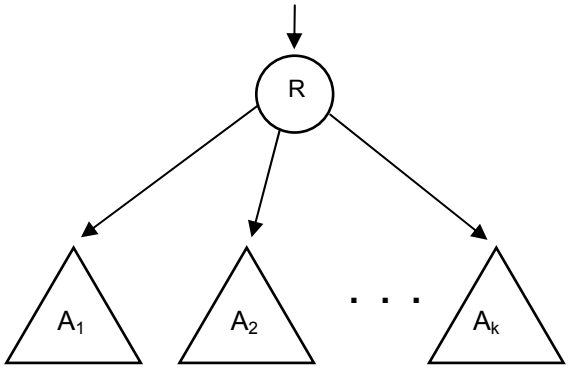

Fig.8.1.3.1.a. Structură de arbore generalizat

1°. Traversarea în **preordine** a arborelui **A** presupune:
- Traversarea rădăcinii **R**
- Traversarea în **preordine** a lui $\mathbf{A_1}$
- Traversarea în **preordine** a lui $\mathbf{A_2}$, $\mathbf{A_3}$ 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 $\mathbf{A_2, A_3,...,A_k}$.

3°. Traversarea în **postordine** a arborelui A constă în:
- Traversarea în **postordine** a subarborilor $\mathbf{A_1}$, $\mathbf{A_2, ...,A_k}$
- Traversarea nodului **R**.

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

{Traversarea arborelui generalizat - modalităţi principiale}

Preordine(A) :R,Preordine(A_1),Preordine(A_2),...,
Preordine(A_k).
Inordine(A) :Inordine(A_1),R,Inordine(A_2),...,
Inordine(A_k). [8.1.3.1.a]
Postordine(A):Postordine(A_1), Postordine(A_2),...,
Postordine(A_k),R.

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

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

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

```
procedure Inordine(r: TipNod);
  dacă *r este nod terminal atunci  *listează(r);
   |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}

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

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

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

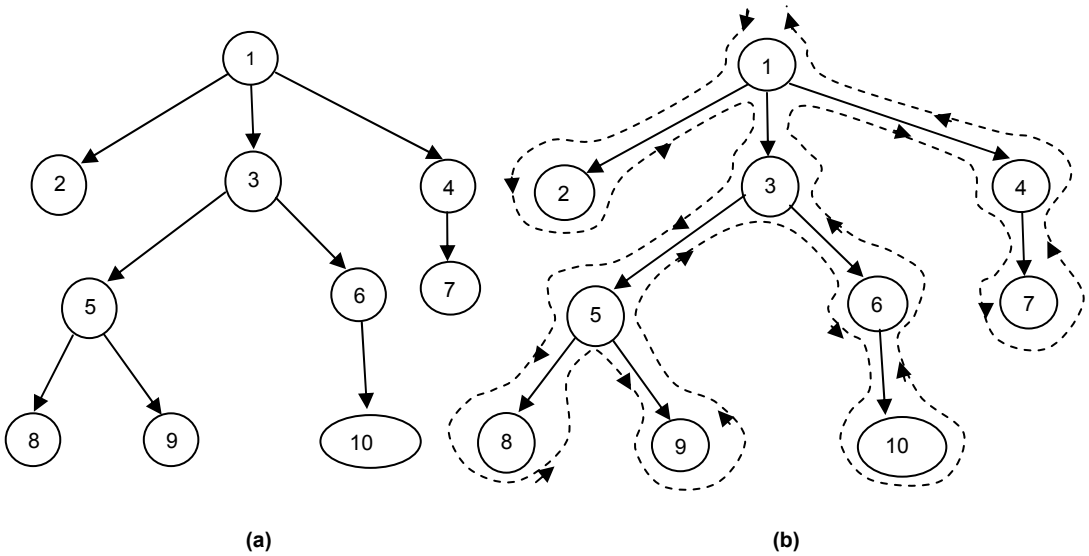

Fig.8.1.3.1.b. Traversarea unui arbore generalizat

- O **metodă practică simplă de parcurgere** a unui **arbore** este următoarea:

    - Dându-se o structură de 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 părinții lor: 2, 8, 9, 5, 10, 6, 3, 7, 4, 1;

    - Pentru **inordine**, un **nod terminal** se listează când este întâlnit **prima** oară, iar un **nod interior** când este întâlnit **a doua oară**: 2, 1, 8, 5, 9, 3, 10, 6, 7, 4.

□----------

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

- Procedura din secvența următoare realizează tipărirea în **preordine** a cheilor nodurilor arborelui generalizat **A**

- Procedura este apelată prin următorul apel:
  `TraversarePreordine(`*`Radacina`*`(A)).`

- Se presupune că nodurile arborelui sunt de tip `TipNod`.

```
PROCEDURE TraversarePreordine(r: TipNod);
{Implementare bazată pe setul de operatori TDA Arbore
generalizat}
{listeaza cheile descendentilor lui r in preordine}

  VAR f: TipNod;
    BEGIN
      Write(Cheie(r,a));
      f:=PrimulFiu(r,a);
      WHILE f<>0 DO
        BEGIN                                          [8.1.3.1.c]
          TraversarePreordine(f);
          f:=FrateDreapta(f,a)
        END
    END; {Preordine}
```

□----------□

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

- Se presupune că nodurile arborelui sunt de tip `TipNod`.
- În cadrul variantei nerecursive se utilizează **stiva** `s`, care conține elemente de `TipNod`.
- Când se ajunge la prelucrarea nodului n, stiva va conține memorat **drumul** de la rădăcină la nodul n, cu rădăcina la baza stivei şi nodul n în vârf.
- Procedura care apare în secvența [8.1.3.1.d], are **două moduri de acțiune**.
  - (1) În **primul mod**, se descinde de-a lungul celui mai stâng drum neexplorat din cadrul arborelui, tipărind (**prelucrând**) nodurile întâlnite pe drum şi **introducându**-le în stivă, până se ajunge la un nod terminal.
  - (2) În acest moment 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 drept.
  - În acest moment se **revine** în primul mod şi reîncepe descinderea cu acest frate încă neexplorat.
  - Procedura începe în modul unu şi se termină când stiva devine vidă.

```
PROCEDURE TraversarePreordineNerecursiva(a: TipNod);
{Implementare nerecursiva bazata pe structura stiva}
{Se utilizează operatorii TDA Arbore generalizat, TDA Stiva}

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

  BEGIN
    Initializează(s);
    m:=Radacina(a);
    gata:=false;
    WHILE NOT gata DO
```

```
      IF m<>0 THEN
          BEGIN
            Write(Cheie(m,a));
            Push(m,s);
            m:=PrimulFiu(m,a)  {explorează fiul stâng}
          END
        ELSE
          IF Stivid(s) THEN
              gata:=true                          [8.1.3.1.d]
            ELSE
              BEGIN
                m:=FrateDreapta(VarfSt(s),a);
                Pop(s)
              END
  END; {TraversarePreordineNerecursiva}
-------------------------------------------------------------□
```

8.1.3.2. Traversarea arborilor generalizați prin tehnica căutării prin cuprindere

- Tehnica căutării prin **cuprindere** derivă tot din traversarea **grafurilor**, dar ea este utilizată, e 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 înălțimea arborelui).

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

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

```
-------------------------------------------------------------
procedure TraversarePrinCuprindere(r: TipNod)
{Implementare bazată pe TDA Coadă}
  Coada: TipCoada;

  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);
    *listeaza(r);
    pentru fiecare fiu y al lui r,(dacă există vreunul),
            în ordine de la stânga la dreapta execută
       Adauga(y,Coada);
    □
  □
□-------------------------------------------------------------
```

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

```
-------------------------------------------------------------
PROCEDURE TraversarePrinCuprindere(r: TipNod);
{Implementare bazată pe TDA Arbore generalizat, TDA Coada}

  VAR Coada: TipCoada;
      f: TipNod;
  BEGIN
    Initializeaza(Coada);
    Adauga(r,Coada);
    WHILE NOT Vid(Coada) DO
      BEGIN
        r:=Cap(Coada); Scoate(Coada);
        WRITE(Cheie(r));
        f:=PrimulFiu(r);
        IF f<> NIL THEN                                  [8.1.3.2.b]
          BEGIN
            Adauga(f,Coada);
            f:=FrateDreapta(f)
            WHILE f<> NIL DO
              BEGIN
                Adauga(f,Coada);
                f:=FrateDreapta(f)
              END
          END
      END
  END;{TraversarePrinCuprindere}
-------------------------------------------------------------
```

8.1.4. Tehnici de implementare a TDA arbore generalizat

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

8.1.4.1. Implementarea arborilor generalizați cu ajutorul tablourilor

- Se bazează pe următoarea **metodă**:
  - Fie A un arbore generalizat cu n noduri.
  - Se numerotează nodurile arborelui A de la 1 la n.
  - Se asociază nodurilor arborelui elementele unui tablou A, astfel încât nodului i al arborelui îi corespunde locația A[i] din tablou.
  - Cea mai simplă manieră de reprezentare a unui arbore, care permite implementarea operației ***Tata***, este cea conform căreia fiecare intrare A[i] din tablou memorează un indicator (cursor) la **părintele** nodului i.
  - Rădăcina se poate distinge prin valoarea zero a cursorului.

- Cu alte cuvinte, dacă A[i] = j atunci nodul j este **părintele** nodului i iar dacă A[i]= 0, atunci nodul i este **rădăcina** arborelui.

- Acest mod de reprezentare, face uz de proprietatea arborilor care stipulează că orice nod are **un singur părinte**, motiv pentru care se 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 - părinte, se realizează într-un interval de timp proporţional cu adîncimea nodului.

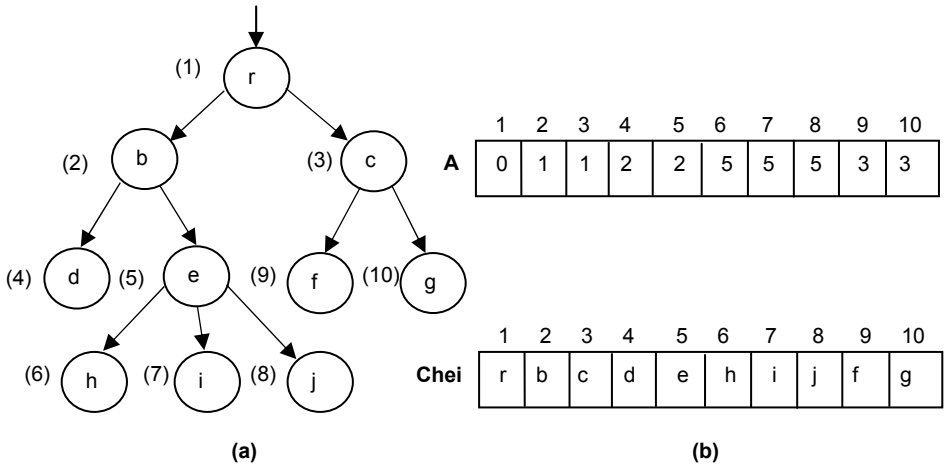

Fig. 8.1.4.1.a. Reprezentarea arborilor generalizaţi cu ajutorul tablourilor (varianta "indicator spre părinte")

- 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 ***FrateDrepta***.

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

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

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

{Exemplu de implementare a operatorului FrateDreapta}

FUNCTION FrateDreapta(n: TipNod; a: TipArbore): TipNod;
  VAR i,parinte: TipNod;
  BEGIN
    parinte:=a[n];                          [8.1.4.1.a]
    FrateDreapta:=0;
    i:=n;
    REPEAT
      i:=i+1;
      IF a[i]=parinte THEN FrateDreapta:=i
    UNTIL(FrateDreapta<>0) OR (i=maxnod)
  END;{FrateDreapta}
```

8.1.4.2. Implementarea arborilor generalizați cu ajutorul listelor

- O manieră importantă şi utilă de implementare a arborilor generalizați este aceea de a crea pentru **fiecare nod** al arborelui **o listă** a fiilor săi.

- Datorită faptului că numărul fiilor poate fi **variabil**, o variantă potrivită de implementare o reprezintă utilizarea **listelor înlănțuite**.

- În fig.8.1.4.2.a se sugerează maniera în care se poate implementa arborele din figura 8.1.4.1.a.(a).

    - Se utilizează un **tablou** `(inceput)` care conține câte o locație pentru fiecare nod al arborelui.

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

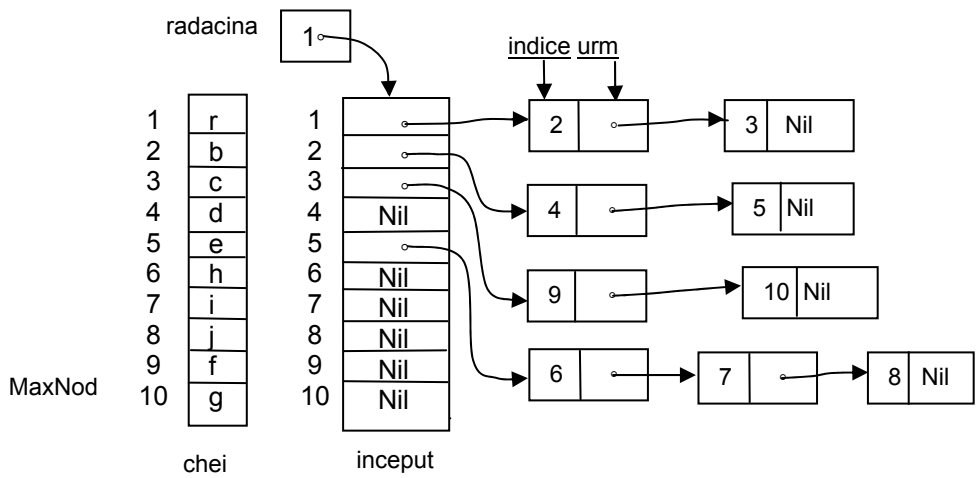

Fig.8.1.4.2.a. Reprezentarea arborilor generalizaţi cu ajutorul listelor înlănţuite

.

- În continuare se prezintă un exemplu de implemntare de implementare utilizând liste înlănţuite simple bazate pe pointeri

  - Tipurile de date propuse apar în secvenţa [8.1.4.2.a].

    - 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***.

    - Se presupune că se utilizează operatorii definiţi peste tipul de date abstract listă prezentaţi în Vol. 1, &6.2.1

```
-------------------------------------------------------------
{Reprezentarea arborilor generalizaţi utilizând liste
înlănţuite simple implementate cu ajutorul pointerilor}
TYPE TipPointreNod=^TipNodList;
     TipNodList=RECORD
          indice:1..MaxNod;
          urm   :TipPointerNod
     END;
     TipNod=0..MaxNod;                              [8.1.4.2.a]
     TipLista=TipPointerNod;
     TipArbore=RECORD
       inceput:ARRAY[1..MaxNod] OF TipLista;
       {chei:ARRAY[1..MaxNod] OF TipCheie;}
       radacina:TipNod
     END;
-------------------------------------------------------------
{Exemplu de implementare al operatorului PrimulFiu}
{se utilizează TDA Listă, varianta restrânsă}
FUNCTION PrimulFiu(n: TipNod; a: TipArbore): TipNod;
  VAR l: TipLista;
  BEGIN
    l:=a.inceput[n];                              [8.1.4.2.b]
    IF Fin(l) THEN  {n este un nod terminal}
        PrimulFiu:=0
      ELSE
        PrimulFiu:=Furnizeaza(Primul(l),l)
  END;{PrimulFiu}
-------------------------------------------------------------
```

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

- Implementările structurilor de 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**.

- Pentru a rezolva această problemă, pentru implementarea structurii arbore generarlizat se poate utiliza o structură ca cea din figura 8.1.4.3.b, în forma tabloului `Zona` care are următoarele caracteristici:

    - Fiecare nod al arborelui este identificat prin indexul celulei pe care el o ocupă în tabloul `Zona`

    - Alocarea nodurilor se realizează în manieră dinamică utilizând locația `Disponibil`. Nodurile disponibile se înlănțuie prin intermediul câmpului `primulFiu`

    - Câmpul `frateDreapta` indică fratele dreapta al nodului respectiv

    - Câmpul `primulFiu` indică primul fiu al nodului respectiv

    - Câmpul `tata` indică părintele nodului respectiv

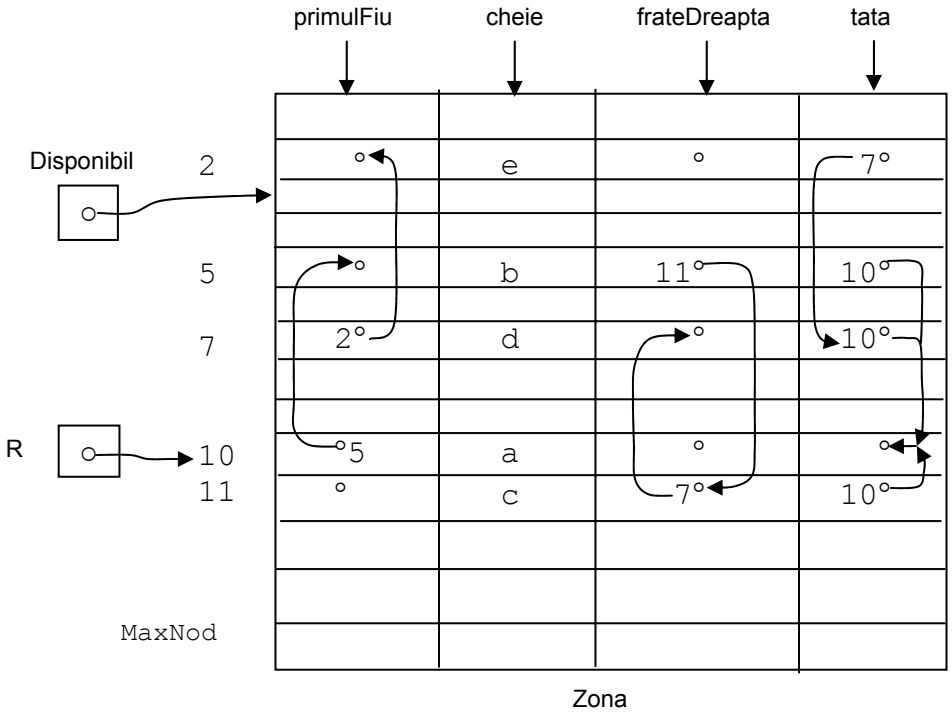

Fig.8.1.4.3.b. Reprezentarea unui arbore generalizat cu ajutorul relațiilor "primul-fiu" şi "frate-dreapta" (Varianta 2)

- O structură de date încadrată în `TipArbore`, este desemnată în aceste condiții printr-un **indice** în tabloul `Zona`, indice care indică nodul rădăcină al arborelui.

- În fig.8.1.4.3.b, apare reprezentarea arborelui din figura 8.1.4.3.a., iar în secvența [8.1.4.3.b] apare definiția formala a structurii de date corespunzătoare acestui mod de implementare al arborilor generalizați.

```
-------------------------------------------------------------
{Reprezentarea arborilor generalizaţi bazată pe relaţiile
primul-fiu şi frate-dreapta (Varianta 2)}

TYPE TipCursor=0..MaxNod;
     TipArbore=TipCursor;                          [8.1.4.3.b]
VAR  Zona:ARRAY[1..MaxNod] OF RECORD
                                 primulFiu:TipCursor;
```

```
                                    cheie:TipCheie;
                                    frateDreapta:TipCursor;
                                    tata:TipCursor
                                END;
    Disponibil:TipCursor;
    R:TipArbore;
-------------------------------------------------------------
```

- În secvența [8.1.4.3.c] este prezentat un exemplu de implementare a operatorului ***Creaza₂***, pornind de la reprezentarea propusă.

- Se reamintește că există o listă a liberilor în tabloul `Zona`, indicată prin cursorul `Disponibil`, în cadrul căreia elementele sunt înlănțuite prin intermediul câmpului `primulFiu`.

```
-------------------------------------------------------------
{Exemplu de implementare al operatorului Creaza₂}

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

8.2. Arbori binari

8.2.1. Definiții

- Prin arbore binar se înțelege o mulțime de n ≥ 0 noduri care dacă nu este vidă, conține un anumit nod numit **rădăcină**, iar restul nodurilor formează doi arbori binari disjuncți numiți: **subarborele stâng** respectiv **subarborele drept**.

- Ca şi exemple pot fi considerați arborii binari reprezentați în figura 8.2.1.a.

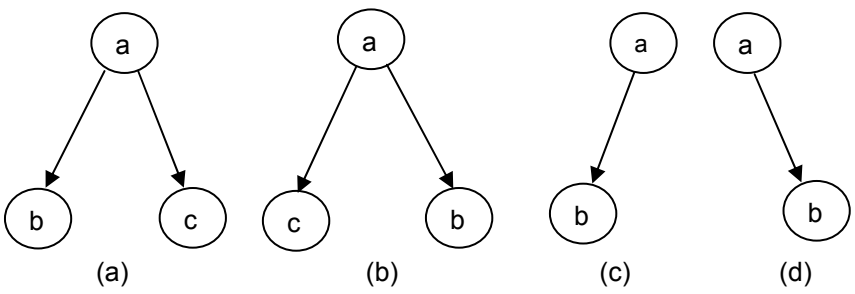

Fig.8.2.1.a. Structuri de arbori binari