ARBORI

1. NOȚIUNI INTRODUCTIVE

Cele mai importante structuri neliniare ce intervin în algoritmii de prelucrare cu calculatorul sunt structurile arborescente. Vorbind în general, structurile arborescente presupun relații de „ramificare” între noduri, asemănător configurației arborilor din natură.

Definiție. Structura arborescentă este o structură de date de tip arbore datele fiind înmagazinate în noduri, nodul de baza e rădăcină. Un arbore reprezintă un "graf" particular, în care substructurile legate la orice nod sunt disjuncte și în care exista un nod numit rădăcină din care e accesibil orice nod prin traversarea unui număr finit de arce. Conform teoriei grafurilor, un **arbore** este un graf neorientat, conex și fără cicluri. Arborii reprezintă grafurile cele mai simple ca structură din clasa grafurilor conexe, ei fiind cel mai frecvent utilizați în practică.

Pentru un graf arbitrar G cu n vârfuri si m muchii sunt echivalente următoarele afirmații:

1) G este arbore;
2) G este un graf conex si m = n - 1;
3) G este un graf aciclic si m = n - 1;
4) oricare doua vârfuri distincte (diferite) ale lui G sunt unite printr-un lanț simplu care este unic;
5) G este un graf aciclic cu proprietatea ca, daca o pereche oarecare de vârfuri neadiacente vor fi unite cu o muchie, atunci graful obținut va conține exact un ciclu.

Prin arbore se mai înțelege o mulțime finită de $n \geq 0$ elemente denumite *noduri* sau *vârfuri*, de același fel, care, dacă nu este vidă, atunci are :

a) un nod cu destinație specială, numit ***rădăcina arborelui***;

b) celelalte noduri sunt repartizate în $m \geq 0$ seturi disjuncte $A_1, A_2, \ldots, A_m$; fiecare set A_i constituind la rândul său un arbore.

Această definiție este recursivă. Există și alte definiții nerecursive pentru arbori (de exemplu, cea pe baza relației de preordine), dar definiția recursivă pare cea mai potrivită deoarece recursivitatea reprezintă o caracteristică naturală a structurilor arborescente.

Elementele arborelui sunt nodurile și legăturile dintre ele. Nodul rădăcină *r* îl considerăm ca fiind de *nivelul zero*. Întrucât $A_1, A_2, \ldots, A_m$ sunt la rândul lor arbori, fie $r_1, r_2, \ldots, r_m$ rădăcinile lor. Atunci $r_1, r_2, \ldots, r_m$ formează nivelul *unu* al arborelui.

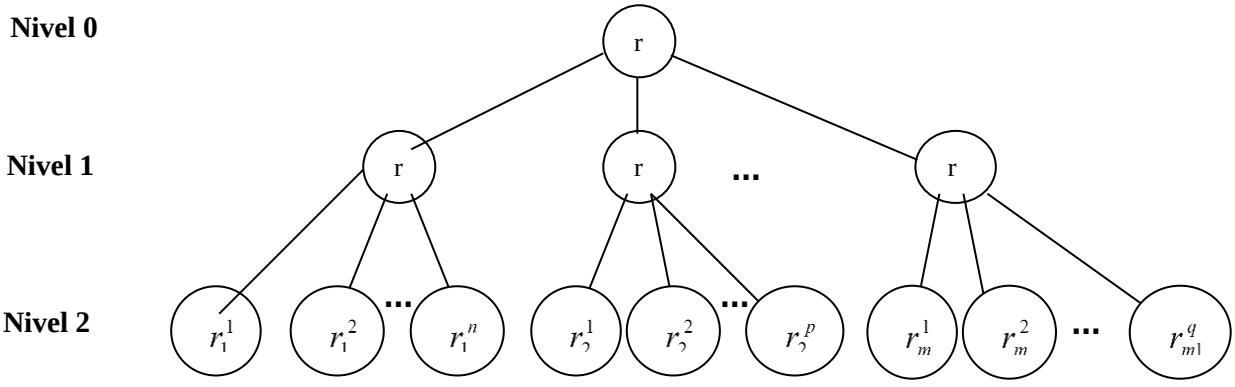


Fig. 1. Structura de tip arbore.

Nodul *r* va avea câte o legătură cu fiecare dintre nodurile $r_1, r_2, \ldots, r_m$. Continuând acest procedeu vom avea nivelurile 2, 3, … ale arborelui. Deci fiii tuturor nodurilor nivelului i formează nivelul i+1.

Dacă numărul nivelurilor este finit, atunci și arborele este finit. În caz contrar se numește **arbore infinit**.

O astfel de viziune asupra arborilor este o viziune ierarhică, nodurile fiind subordonate unele altora, fiecare nod fiind subordonat direct unui singur nod, excepție făcând rădăcina ce nu este

subordonată nici unui nod. Dacă un nod nu are nici un nod subordonat, atunci se numește *frunză* sau nod *terminal*. Numărul fiilor unui nod reprezintă *gradul* acelui nod. Un nod terminal are gradul zero. Un nod neterminal se numește deseori *nod de ramificare* sau *interior*.

Dacă nodul A are ca subordonați nodurile B și C, atunci A se numește *tată, părintele* lui B și C, B și C sunt *fiii* lui A, iar B este *fratele* lui C.

Gradul maxim al nodurilor unui arbore se numește *gradul arborelui* .

Dacă ordinea relativă a subarborilor A_1 , A_2 ,..., A_m în punctul b) al definiției are importanță, arborele se numește *arbore ordonat*. Exemplu: arborii din figura 2 sunt distincți (ca arbori ordonați).

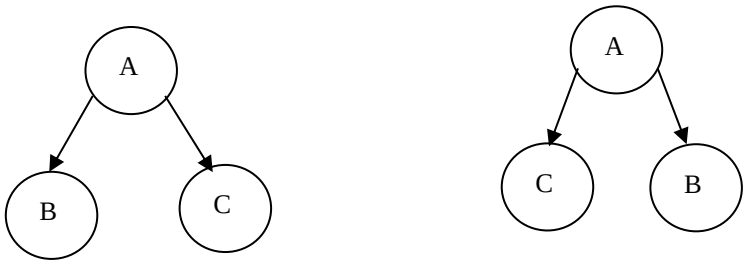


Fig. 2. Arbori binari

Nivelul maxim al nodurilor unui arbore se numește *înălțimea arborelui*. Fie arborele din figura 3. Acesta are înălțimea 4, nodul D este de grad 2, H este terminal, F este un nod intern, iar gradul arborelui este 3.

Prin subarborii unui arbore se înțeleg arborii în care se descompune acesta prin îndepărtarea rădăcinii. A_1 , A_2 ,..., A_m din definiția b) sunt subarborii rădăcinii.

Orice nod al unui arbore este rădăcina unui *arbore parțial*. Un arbore parțial nu este însă întotdeauna subarbore pentru arborele complet, dar orice subarbore este arbore parțial.

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 $1 \le i < k$, atunci această secvență se numește drum sau cale de la nodul n_i la nodul n_k .

Lungimea unui drum la un nod este un întreg având valoarea egală cu numărul de ramuri care trebuie traversate (parcurse) pentru a ajunge de la rădăcină la nodul respectiv. Rădăcina are lungimea drumului egală cu zero, fiii ei au lungimea drumului egală cu 2 și în general un nod situat la nivelul i are lungimea drumului i. Spre exemplu, în figura 3, lungimea drumului la nodul D este 1, iar la nodul P este 4. Dacă există un drum de la nodul x la nodul y, atunci nodul x se numește strămoș al lui y, iar nodul y se numește descendent al lui x. Exemplu: în aceeași figură strămoșii lui F sunt B, A, iar descendenții săi sunt L, K, M și P. Conform celor deja precizate, tatăl unui nod este strămoșul său direct sau predecesor, iar fiul unui nod este descendentul său direct sau succesor.

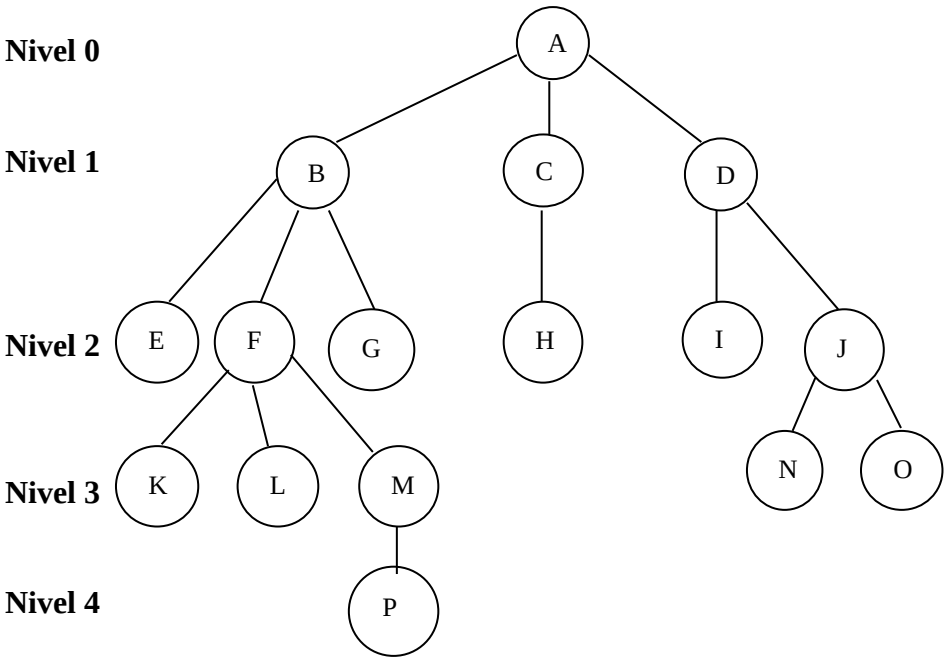


Fig. 3. Exemplu de arbore cu mai multe nivele.

Un strămoș sau un descendent al unui nod, altul decât nodul însuși, se numește *strămoș propriu*, respectiv *descendent propriu*. Într-un arbore, rădăcina este singurul nod fără un strămoș propriu. Nodurile terminale nu au descendenți proprii.

Înălțimea unui nod într-un arbore este lungimea celui mai lung drum de la nodul respectiv la un nod terminal. Pornind de la această definiție, ***înălțimea unui arbore*** se poate defini și ca fiind egală cu înălțimea nodului rădăcină. Deci, înălțimea unui arbore este egală cu nivelul maxim al acelui arbore. Adâncimea unui nod este egală cu lungimea drumului unic de la rădăcină la acel nod.

2. TRAVERSAREA ARBORILOR

Una din activitățile fundamentale care se execută asupra unei structuri de arbore este traversarea acestuia. Ca și în cazul listelor liniare, prin traversarea unui arbore se înțelege execuția unei anumite operații asupra tuturor nodurilor arborelui. În timpul traversării nodurile sunt vizitate într-o anumită ordine, astfel încât ele pot fi considerate ca și cum ar fi integrate într-o listă liniară. De fapt descrierea celor mai mulți algoritmi este mult ușurată dacă în cursul prelucrării se poate preciza elementul *următor* al structurii de arbore, respectiv se poate liniariza structura de arbore .

Există trei moduri de ordonare (liniarizare) mai importante care se referă la o structură de arbore, numite " preordine", "inordine" și " postordine". Toate trei se definesc recursiv, asemenea structurii de arbore. Ordonarea unui arbore se definește presupunând că subarborii săi s-au ordonat deja. Cum subarborii au noduri strict mai puține decât arborele complet, rezultă că, aplicând recursiv de un număr suficient de ori definiția, se ajunge la ordonarea unui arbore vid, care este evidentă. Cele trei moduri de traversare se definesc recursiv după cum urmează:

- Dacă arborele A este vid, atunci ordonarea lui A în preordine, inordine, postordine se reduce la lista vidă.
- Dacă A se reduce la un singur nod, atunci nodul însuși reprezintă traversarea lui A, în oricare dintre cele trei moduri.
- Pentru restul cazurilor fie arborele A cu rădăcina R și cu subarborii A_1 , A_2 ,..., A_k .

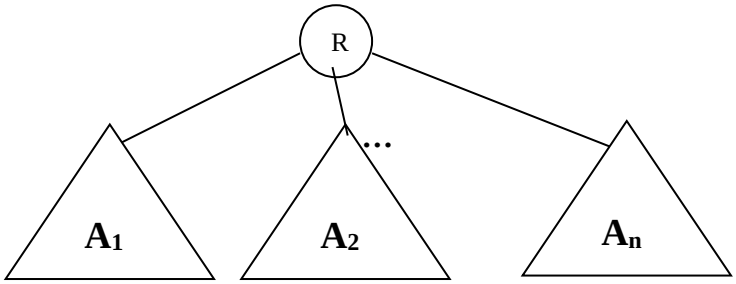


Fig. 4. Parcurgerea arborilor oarecare

1. ***Traversarea în preordine*** a arborelui A presupune traversarea rădăcinii R, urmată de traversarea în preordine a lui A_1, apoi de traversarea în preordine a lui A_2 și așa mai departe, până la A_k inclusiv.
2. ***Traversarea în inordine*** presupune parcurgerea în inordine a subarborelui A_1, urmată de nodul R și în continuare de parcurgerile în inordine a subarborilor A_2,..., A_k .
3. ***Traversarea în postordine*** a arborelui A constă în traversarea în postordine a lui A_1,A_2,..., A_k și în final a nodului R.

Spre exemplu, pentru arborele din figura 5 traversările definite anterior conduc la următoarele secvențe de noduri:

- preordine: 1, 2, 5, 6, 10, 11, 12, 7, 3, 8, 4, 9;
- inordine: 5, 2, 10, 6, 11, 12, 7, 1, 8, 3, 9, 4;
- postordine: 5, 10, 11, 12, 6, 7, 2, 8, 3, 9, 4, 1.

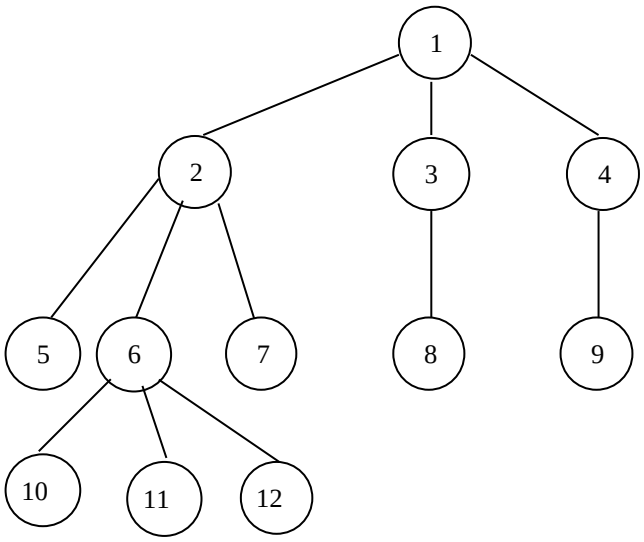


Fig. 5. Arbore cu mai multe nivele

3. REPREZENTAREA STRUCTURILOR ARBORESCENTE ÎN PROGRAME

Stocarea datelor sub formă de structuri arborescente, presupune definirea unor tipuri de date ce să permită accesarea datelor conform logicii arborilor. Pentru reprezentarea structurilor pe suport a structurilor arborescente se folosește alocarea înlănțuită, **obținând structuri dinamice de tip arbore.**

Cea mai cunoscută metodă este cea a folosirii unor pointeri multipli, adică a unor referințe spre fii fiecărui nod . Fiecare nod conține, pe lângă informația propriu-zisă, n pointeri p_1, p_2, …, p_n, unde n este gradul nodului și a unui câmp care să păstreze gradul acelui nod. Pointerii p_1, p_2, …, p_n vor conține adresele succesorilor imediați al nodului (adică fii nodului) .

Să considerăm exemplul din figura 6 :

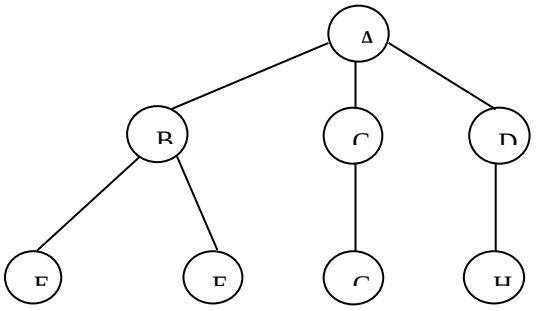


Fig. 6. Arbore oarecare

Acest arbore se reprezintă în memorie, cu metoda de mai sus, ca în figura 7.

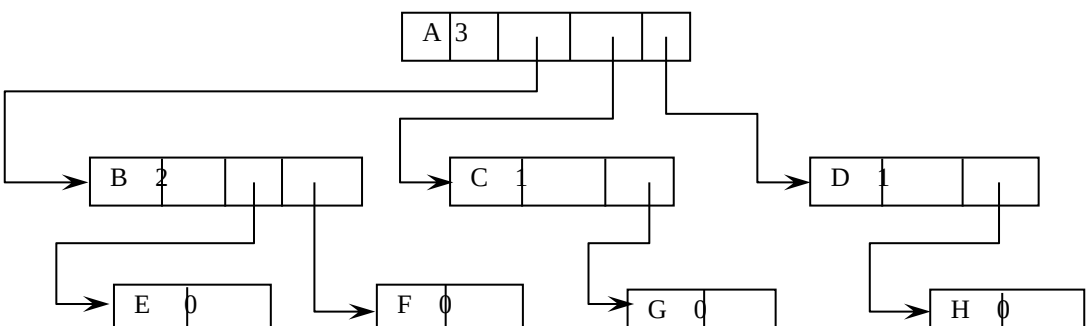


Fig. 7. Reprezentarea în memorie a unui arbore

O soluție asemănătoare cu cea mai sus este de a folosi n pointeri, unde n este în schimb gradul arborelui (adică numărul maxim de fii ai unui nod). Dacă unul din subarbori lipsește, pointerul corespunzător va fi vid, conținând referința null. În figura 8 este prezentat exemplul de mai sus, reprezentat în varianta descrisă.

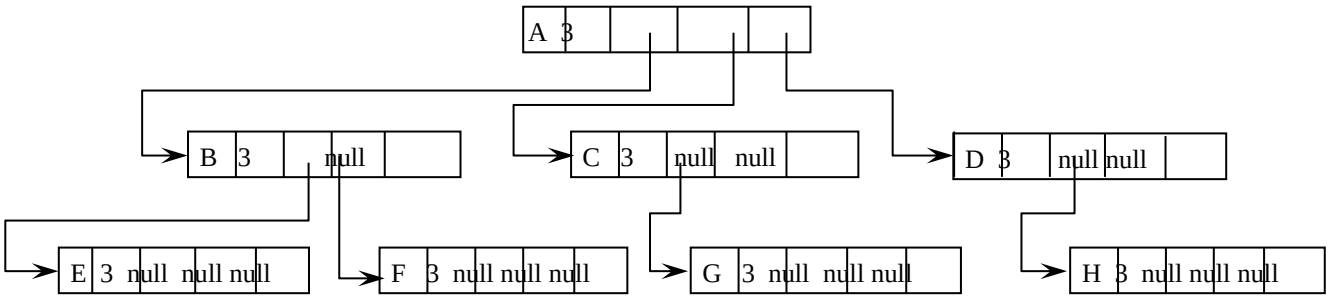


Fig. 8 Reprezentarea în memorie a unui arbore binar utilizând pointeri

Se observă în această situație se face o risipă de memorie. Anumite zone de memorie sunt nefolosite, ele conținând doar referința vidă NULL. Există soluții de eliminarea acestor neajunsuri, prin păstrarea referințelor către fii unui nod nu într-un vector ci folosind o listă simplu înlănțuită în varianta dinamică .

Un caz particular este acela al arborilor binari, care au doar maxim doi fii. În acest caz, structura de mai sus se simplifică și se definește astfel :

```
struct Arb
   {int Info; //Informația nodului respectiv
    Arb *st, *dr; //Adresele fiului stâng și drept
    };
Arb *Rad; //Se păstrează adresa rădăcinii, cu ajutorul ei se accesează arborele
```

Sunt prezentate mai jos modalitățile de creare a unui arbore binar și de parcurgere a sa în cele trei variante . Ele se pot generaliza imediat pentru cazul în care avem arbori oarecare.

```
#include<conio.h>
#include<iostream.h>
struct Arb
   {int Info;
    Arb *st, *dr;
    };
Arb *Rad;
Arb* Creare(Arb *Rad)
{
 char R;
 Rad = new Arb;
 cout<<"Informația nodului ";
 cin>>Rad->Info;
 cout<<"Aveți subarbore stâng";
 cin>>R;
 if (R=='D')
  Rad->st=Creare(Rad->st);
```

```
  else
    Rad->st=NULL;
  cout<<"Aveți subarbore drept";
  cin>>R;
  if (R=='D')
      Rad->dr=Creare(Rad->dr);
  else
     Rad->dr=NULL;
 return Rad;
}
void ParcurgerePre(Arb *Rad)
{
 if (Rad!=NULL)
  {
   cout<<" "<<Rad->Info;
   ParcurgerePre(Rad->st);
   ParcurgerePre(Rad->dr);
}
}
void ParcurgereIn(Arb *Rad)
{
 if (Rad!=NULL)
  {
   ParcurgereIn(Rad->st);
   cout<<" "<<Rad->Info;
   ParcurgereIn(Rad->dr);
}
}
void ParcurgerePost(Arb *Rad)
{
 if (Rad!=NULL)
  {
   ParcurgerePost(Rad->st);
   ParcurgerePost(Rad->dr);
   cout<<" "<<Rad->Info;
}
}
void main()
{
 Rad= NULL;
 Creare(Rad);
 cout<<"Nodurile parcurse in preordine sunt:"<<endl;
 ParcurgerePre(Rad);
 cout<<"Nodurile parcurse in inordine sunt:"<<endl;
 ParcurgereIn(Rad);
 cout<<"nodurile parcurse in postordine sunt:"<<endl;
 ParcurgerePost(Rad);
}
```

4. ARBORI BINARI ORDONAȚI

Structura de arbore poate fi utilizată pentru a reprezenta în mod convenabil o mulțime de elemente în care elementele se regăsesc după o cheie unică. Se presupune că avem o mulțime de *n*

noduri definite ca articole, având câte o cheie care este un număr întreg. Dacă cele *n* articole se organizează într-o structură de listă liniară, căutarea unei chei necesită în medie *n*/2 comparații. După cum se va vedea în continuare, organizarea celor n articole într-o structură convenabilă de arbore binar reduce numărul de comparații la maximum $\log_2 n$. Acest lucru devine posibil utilizând structura de **arbore binar ordonat.**

Prin **arbore binar ordonat** (sau arbore de căutare) se înțelege un arbore binar care are proprietatea că, parcurgând nodurile în inordine, secvența cheilor este monoton crescătoare. Un arbore ordonat se bucură de următoarea proprietate: dacă N este un nod oarecare al arborelui, având cheia c, atunci toate din subarborele stâng al lui N au cheile mai mici decât c și toate nodurile din subarborele drept al lui N au cheile mai mari sau egale cu c.

De aici rezultă un procedeu foarte simplu de căutare: începând cu rădăcina, se trece la fiul stâng sau la fiul drept, după cum cheia căutată este mai mică sau mai mare decât cea a nodului curent. Numărul comparărilor de chei efectuate în acest procedeu este cel mult egal cu înălțimea arborelui. În general, înălțimea arborelui nu este determinată de numărul nodurilor sale. Spre exemplu, cu 9 noduri se poate construi atât arborele ordonat de înălțime 3 (a) cât și arborele ordonat de înălțime 5 (b).

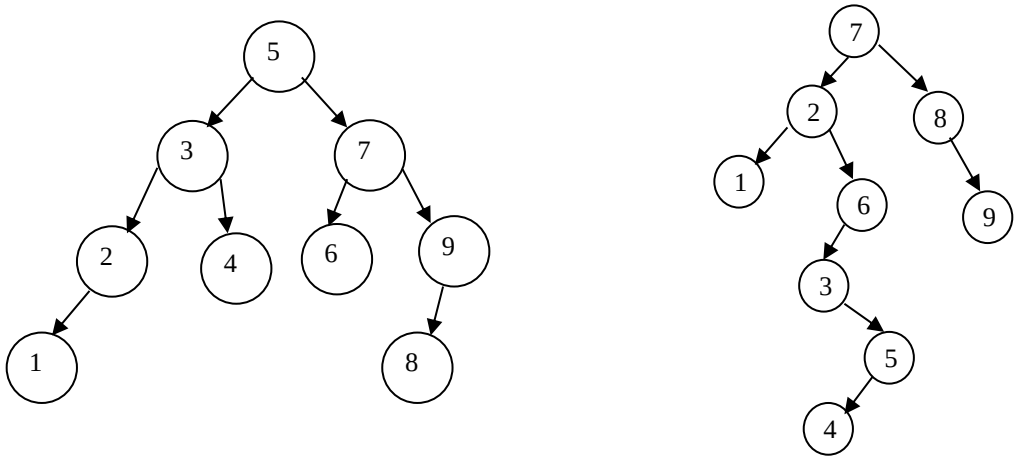


Fig.9. Arbori binari ordonați

(a) înălțime 3 **(b) înălțime 5**

Este simplu de observat că un arbore are înălțimea minimă dacă fiecare nivel conține numărul maxim de noduri, cu excepția posibilă a ultimului nivel. Deoarece numărul maxim de noduri al nivelului i este 2^i , rezultă că înălțimea minimă a unui arbore binar cu n noduri este $h_{min} = [\log_2 n]$. Prin aceasta s-a justificat afirmația că o căutare într-un arbore binar ordonat necesită aproximativ $\log_2 n$ comparații de chei.

Observație. Această afirmație este valabilă în ipoteza că nodurile s-au organizat într-un arbore ordonat de înălțime minimă. Dacă această condiție nu este satisfăcută, eficiența procesului de căutare poate fi mult redusă, în cazul cel mai defavorabil arborele degenerând într-o structură de listă liniară. Aceasta se întâmplă când subarborele drept (stâng) al tuturor nodurilor este vid, caz în care înălțimea arborelui devine n-1, iar căutarea nu este mai eficientă decât într-o listă liniară.

5. TEHNICI DE CĂUTARE ÎNTR-UN ARBORE BINAR ORDONAT

Căutarea cu metoda fanionului

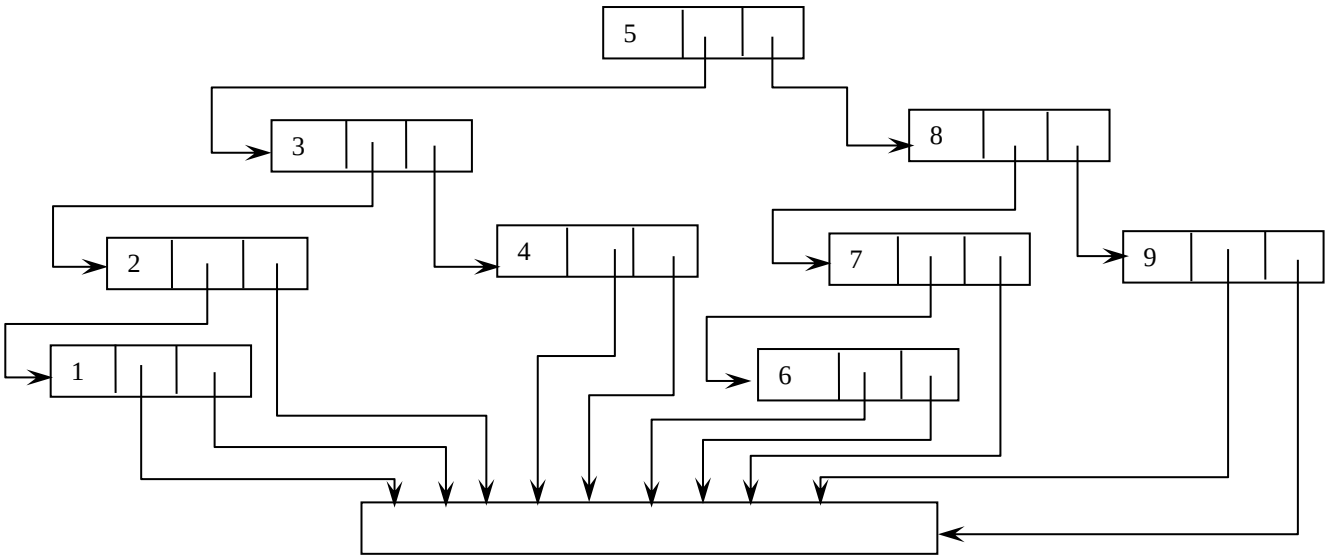
Tehnica de căutare se poate simplifica dacă se aplică **metoda fanionului**. În cazul arborilor, această metodă presupune completarea structurii cu un nod fictiv, fanionul, indicat de un pointer, notat cu F, astfel că toate ramurile arborelui se termină cu acest nod, deci toate referințele egale cu NULL le vom înlocui cu F. Arborele din figura 9 (a) a se va reprezenta ca în figura 10.

Înainte de demararea procesului de căutare propriu-zisă se asignează cheia fanionului cu x. În procesul căutării nodul cu cheia x se găsește acum cu certitudine; dacă acest nod va fi fanionul,

atunci în arborele inițial nu există un nod cu cheia x, în caz contrar nodul găsit este cel căutat. În prezența unei astfel de structuri, funcția *Loc* se modifică devenind *Loc*1, în care se simplifică condiția din instrucțiunea while :

```
Arbore * Loc1(int X, Arbore*T)
{ int gasit;
 gasit=0;
 F->key = x;
 while (T->key != x)
        if (x < T->key) T=T->st;
              else T=T->dr;
 return T;
}
```

Deci dacă *Loc*1(X, T) ia valoarea F, atunci nu există nici un nod cu cheia x, în caz contrar valoarea funcției este un pointer care indică nodul căutat.



Căutare binară

Dacă structura în care se efectuează o operație de căutare este statică (nu trebuie să se adauge sau să se elimine noi elemente), atunci arborele de căutare se poate reprezenta eficient cu ajutorul unui tablou A cu n elemente (n este numărul nodurilor din arbore).

Rădăcina este elementul cu indicele *rad* = [(n+1)/2], subarborele stâng este format din elementele tabloului cu indicele mai mic ca *rad*, iar cel drept cu indicele mai mare ca *rad.* Rădăcina subarborelui drept, respectiv a celui stâng, se determină în același mod. Se observă că tabloul va trebui să conțină de fapt elementele arborelui de căutare în inordine, adică elementele tabloului vor trebui să fie ordonate strict crescător. De asemenea, se observă că arborele obținut este total echilibrat.

Algoritmul de căutare se poate descrie în felul următor:

1) se inițializează: *s*:= 1 și *d* = *n* ;

2) dacă *st* > *dr* avem căutare fără succes (algoritmul se termină);

3) astfel *st* ≤ *dr* (arborele nu este vid) și se calculează *rad* := [(*st*+*dr*)/2] (se determină rădăcina arborelui) și se compară cheia x cu A [*rad*]; dacă A [*rad*] = x avem căutare cu succes (algoritmul se termină) ;

4) dacă x >A [*rad*] se calculează *st* = *rad* + 1 (se trece la subarborele drept) și algoritmul se reia de la pasul 2;

5) altfel x < A [*rad*] și se calculează *d*:= *rad* –1 (se trece la subarborele stâng) și algoritmul se reia de la pasul 2.

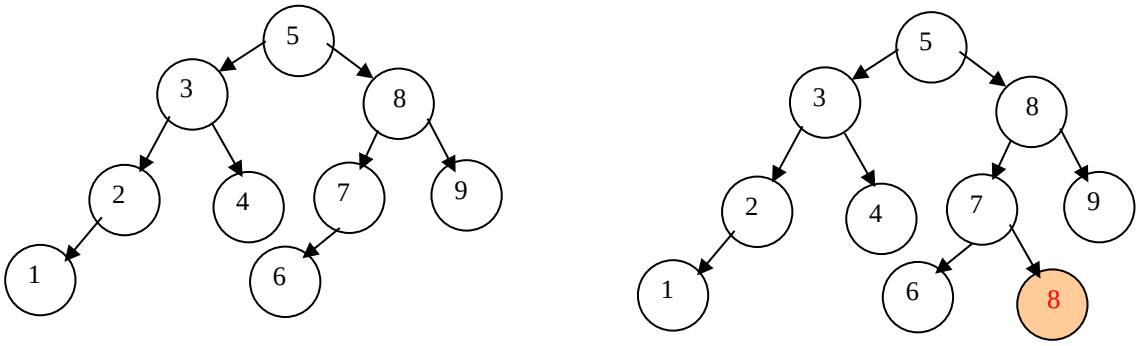
```
#include <iostream.h>
int a[10],n ;
int b ; //b este valoarea căutată

void cautare()
{ int st, dr, rad ; // rad este indicele elementului de mijloc
  int gasit;
  st = 0;
  dr = n-1;
  rad = (st + dr ) / 2;
  gasit = 0;
  while (! gasit && st <= dr )
    {
      if ( b >a [rad]) // căutarea se continua in jumătatea dreapta a șirului
        st = rad + 1 ;
      else // căutarea se continua in jumătatea stânga a șirului
        dr = rad -1;
   rad = (st + dr) / 2;
   if (st <= dr) gasit = b == a[rad];
 }
 if (st > dr) cout<<" Nu este in tablou.";
    else cout<<" S-a găsit pe poziția "<< rad ;
}

void main()
{ cout<<"n=";cin>>n;
 cout<<"Se da șirul ordonat";
 int i,j;
 for (i= 0;i<n;i++)
  cin>>a[i];
 for (j =1; j<=4;j++)
  {
    cout<<"b="; cin>>b;
    cautare();
  }
}
```

6. CREAREA ARBORILOR BINARI ORDINAȚI

În cadrul acestui paragraf se va trata problema construcției unui arbore binar ordonat, pornind de la o mulțime dată de noduri.

Procesul de creare constă în inserția a câte unui nod într-un arbore binar ordonat, care inițial este vid. Problema care se pune este aceea de a executa inserția de o asemenea manieră încât arborele să rămână ordonat și după adăugarea noului nod. Aceasta se realizează traversând arborele începând cu rădăcina și selectând fiul stâng sau drept după cum cheia de inserat este mai mică sau mai mare decât cea a nodului parcurs. Aceasta se repetă până când se ajunge la un pointer NULL. În continuare inserția se realizează modificând acest pointer astfel încât să indice noul nod. Se precizează că inserția noului nod se poate realiza chiar dacă arborele conține deja un nod cu cheia egală cu cea nouă. În acest caz dacă se ajunge la un nod cu cheia egală cu cea de inserat, se procedează ca și cum aceasta din urmă ar fi mai mare, deci se trece la fiul drept al nodului. În felul acesta se obține o metodă de sortare stabilă.

Considerăm arborele din figura 1(a). În figura 11(b) se prezintă inserția unui nou nod cu cheia 8 în structura existentă de arbore ordonat. La parcurgerea în inordine a acestui arbore se observă că cele două noduri de chei egale sunt parcurse în ordinea în care au fost inserate.



a. b.

Fig. 11. Inserarea unui nod cu cheia 8

```
#include<conio.h>
#include<iostream.h>
struct Arb
   {int Info;
    Arb *st, *dr;
    };
Arb *Rad;
int n;
Arb * Inserare(Arb *Rad, int x)
{
  if (Rad==NULL)
   { Rad = new Arb;
    Rad->Info=x;
     Rad->st=NULL;
     Rad->dr=NULL;
   } else if (x<Rad->Info) Rad->st=Inserare(Rad->st,x);
         else Rad->dr=Inserare(Rad->dr,x);
  return Rad;
}
void ParcurgerePre(Arb *Rad)
{
 if (Rad!=NULL)
  {
   cout<<" "<<Rad->Info;
   ParcurgerePre(Rad->st);
   ParcurgerePre(Rad->dr);
  }
}
void ParcurgereIn(Arb *Rad)
{
 if (Rad!=NULL)
  {
   ParcurgereIn(Rad->st);
   cout<<" "<<Rad->Info;
   ParcurgereIn(Rad->dr);
  }
```

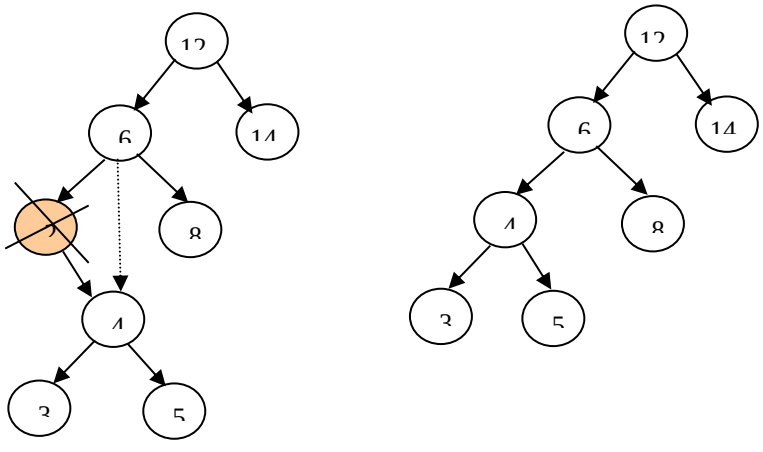
```
}
void ParcurgerePost(Arb *Rad)
{
 if (Rad!=NULL)
  {
   ParcurgerePost(Rad->st);
   ParcurgerePost(Rad->dr);
   cout<<" "<<Rad->Info;
  }
}
void main()
{
 cout<<"Număr de noduri:";
 cin>>n;
 int x,i;
 Rad= NULL;
 for (i=1;i<=n;i++)
 { cout<<"Informația :";
   cin>>x;
   Rad=Inserare(Rad,x);
  }
 cout<<"Nodurile parcurse in preordine sunt:"<<endl;
 ParcurgerePre(Rad);
 cout<<"Nodurile parcurse in inordine sunt:"<<endl;
 ParcurgereIn(Rad);
 cout<<"nodurile parcurse in postordine sunt:"<<endl;
 ParcurgerePost(Rad);
}
```

Tehnica ștergerii unui nod dintr-un arbore binar ordonat

Se consideră o structură de arbore binar ordonat și x o cheie precizată. Pentru ștergerea nodului cu cheia x, în prealabil se caută dacă există un nod cu o astfel de cheie. Dacă nu, suprimarea s-a încheiat și se emite eventual un mesaj de eroare. În caz contrar se execută suprimarea propriu-zisă, de o asemenea manieră încât arborele să rămână ordonat și după terminarea ei.

Dacă elementul ce se șterge este un nod terminal sau un nod cu un singur descendent, atunci ștergerea este directă. Dificultatea apare atunci când nodul care se șterge are doi descendenți căci nu putem pointa în două direcții cu un singur pointer. În această situație elementul care se șterge este înlocuit fie cu cel mai din dreapta element al subarborelui său stâng, fie cu cel mai din stânga element al subarborelui său drept, ambele având maxim un descendent.

1) Dacă nodul care trebuie suprimat are cel mult un fiu se procedează astfel: dacă *P* este câmpul de referință al tatălui lui *x*, adică referința care indică nodul *x*, valoarea lui *P* se modifică astfel încât aceasta să indice unicul fiu al lui *x*, dacă acesta există (fig. 12 a, b) sau, în caz contrar, *P* devine NULL.



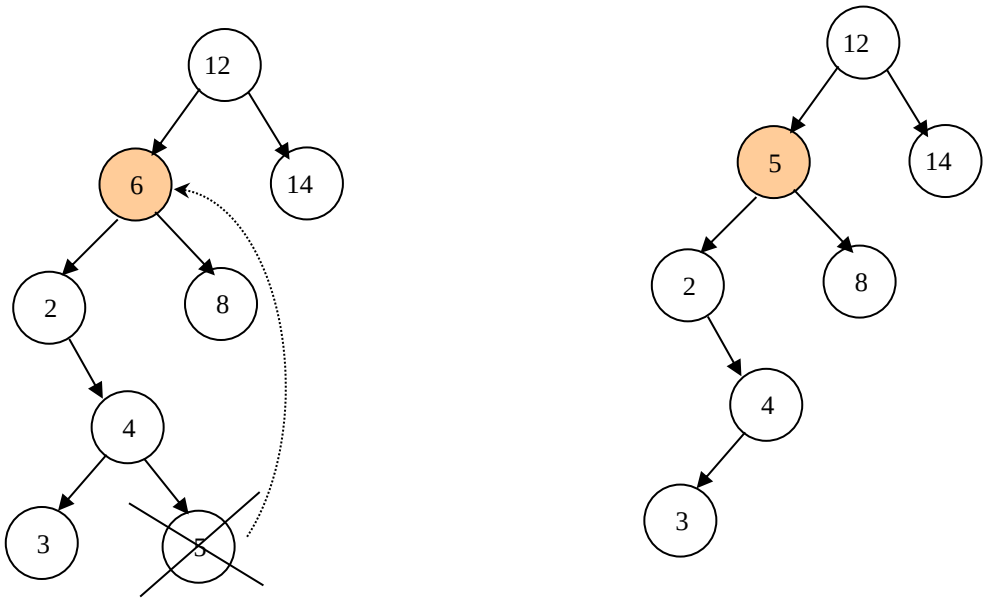
a **b**

Fig. 12. Tehnica suprimării unui nod care are cel puțin un fiu egal cu NULL

2) Cel de-al doilea caz, în care nodul cu cheia *x* are doi fii se rezolvă astfel:

Fie arborele din figura 13. Dacă nodul care conține cheia x are ambii fii diferiți de NULL, procedeul de eliminare este mai complicat. Pentru ștergerea nodului cu cheia 6 din figura 13a se caută predecesorul (respectiv succesorul) acestui nod în ordonarea arborelui în inordine. Fie acesta *y* (în cazul nostru nodul cu cheia 5 sau 8). Nodul *y* există și este unic deoarece la traversarea în inordine se traversează în cazul unui nod cu doi fii mai întâi subarborele său stâng în inordine, deci va fi vizitat cel puțin încă un nod; dar predecesorul în inordine este unic, deci *y* există și este unic.

Se modifică nodul *X* asignând toate câmpurile sale, cu excepția câmpurilor *st* și *dr* cu câmpurile corespunzătoare ale lui *y*. În acest moment, în structura de arbore, nodul *y* se găsește în dublu exemplar: în locul său inițial și în locul fostului nod *x* .



a **b**

Fig. 13. Tehnica suprimării unui nod care are ambii fii diferiți de NULL

Se suprimă nodul *Y* inițial conform cazului 1) deoarece nodul nu are un fiu drept. Cu privire la nodul *Y* se poate demonstra că el se detectează după următoarea metodă: se construiește o secvență de noduri care încep cu fiul stâng al lui *X*, după care se alege drept succesor al fiecărui nod fiul său drept. Primul nod al secvenței care nu are fiu drept este *Y* (nodul cu cheia 5 din fig. 13. a) .

Procedura *Suprimare* realizează suprimarea unui nod într-o structură de arbore binar ordonat. Procedura locală *Suprfd* caută predecesorul în inordine al nodului *X*, realizând suprimarea acestuia conform metodei descrise. Această procedură se utilizează numai în situația în care nodul *X* are doi fii:

```
#include<conio.h>
#include<iostream.h>
struct Arb
  { int Info;
    Arb *st, *dr;
  };
Arb *Rad, *q;
int n;
Arb * Inserare(Arb *Rad, int x)
{
```

```
 if (Rad==NULL)
  { Rad = new Arb;
   Rad->Info=x;
      Rad->st=NULL;
    Rad->dr=NULL;
  } else if (x<Rad->Info) Rad->st=Inserare(Rad->st,x);
       else Rad->dr=Inserare(Rad->dr,x);
return Rad;
}
void ParcurgerePre(Arb *Rad)
{
 if (Rad!=NULL)
  {
   cout<<" "<<Rad->Info;
   ParcurgerePre(Rad->st);
   ParcurgerePre(Rad->dr);
  }
}
void ParcurgereIn(Arb *Rad)
{
 if (Rad!=NULL)
  {
   ParcurgereIn(Rad->st);
   cout<<" "<<Rad->Info;
   ParcurgereIn(Rad->dr);
  }
}
void ParcurgerePost(Arb *Rad)
{
 if (Rad!=NULL)
  {
   ParcurgerePost(Rad->st);
   ParcurgerePost(Rad->dr);
   cout<<" "<<Rad->Info;
  }
}
int cauta(Arb *Rad, int x)
{ if (Rad==NULL) return 0;
  else if (Rad->Info==x) return 1;
     else if (Rad->Info<x) return cauta(Rad->dr,x);
        else return cauta(Rad->st,x);
}
Arb * suprfd (Arb *r)
{
  if (r->dr!=NULL)  r->dr= suprfd (r->dr); //căutăm Y
   else {
            q->Info = r->Info; // X = Y
            q = r;
            r = q->st ; //suprim pe Y
            delete q; //eliberarea spațiului
       }
  return r;
 }
```

```
 Arb * suprimare (Arb *Rad, int x )
{  if (Rad==NULL)
    cout<<" Nodul"<< x<<" nu a fost găsit, deci nu se poate șterge";
 else if (x < Rad->Info)
        Rad->st= suprimare (Rad->st,x);
        else if ( x >Rad->Info)
           Rad->dr= suprimare (Rad->dr,x);
           else
                { q = Rad; //Rad conține adresa lui x
                  if ( q->dr == NULL) Rad= q->st; // nu are fiu drept
                  else if (q->st ==NULL) Rad = q->dr; // nu are fiu stâng
                      else q=suprfd (q->st); //are 2 fii
        }
 return Rad;
}
void main()
{
 cout<<"Număr de noduri:";
 cin>>n;
 int x,i;
 Rad= NULL;
 for (i=1;i<=n;i++)
 { cout<<"Informația :";
   cin>>x;
   Rad=Inserare(Rad,x);
  }
 cout<<"Nodurile parcurse in preordine sunt:"<<endl;
 ParcurgerePre(Rad);
 cout<<"Nodurile parcurse in inordine sunt:"<<endl;
 ParcurgereIn(Rad);
 cout<<"nodurile parcurse in postordine sunt:"<<endl;
 ParcurgerePost(Rad);
 cout<<"Informația căutată:";cin>>x;
 if (cauta(Rad,x)) cout<<"Este";
 else cout<<"Nu este";
 cout<<endl;
 cout<<"Informația care se șterge:";
 cin>>x;
 Rad=suprimare(Rad,x);
 cout<<"Nodurile ramase parcurse in preordine sunt:"<<endl;
 ParcurgerePre(Rad);
 cout<<"Nodurile ramase parcurse in inordine sunt:"<<endl;
 ParcurgereIn(Rad);
 cout<<"nodurile ramase parcurse in postordine sunt:"<<endl;
 ParcurgerePost(Rad);
}
```