CAPITOLUL 2
2.1 ARBORI BINARI DE CĂUTARE
2.1.1 Definiție. Caracteristici

Definiție

Un **arbore binar de căutare** este un **arbore binar** cu proprietatea că, pentru orice **nod v** al său, cheile nodurilor din subarborele stâng sunt mai mici decât cheia nodului **v** și cheile nodurilor din subarborele drept sunt mai mari decât cheia nodului **v**.

Caracteristică

Într-un **arbore binar de căutare** nu există 2 noduri cu aceeași valoare a cheii.

În **Figura 2.1** este reprezentat un arbore binar de căutare:

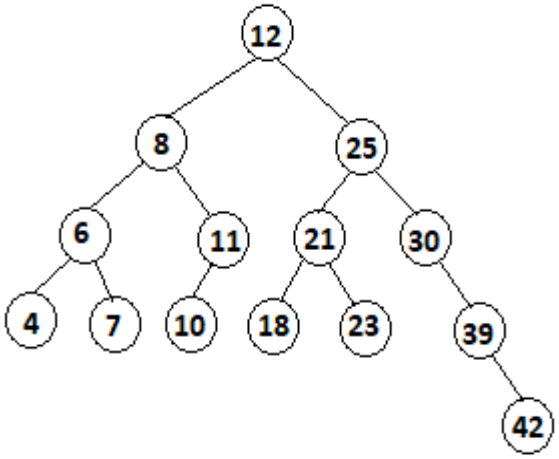


Figura 2.1 Arbore binar de căutare

2.1.2 Operații speciale cu arbori binari de căutare

Pentru operațiile de prelucrarea a unui arbore binar, se consideră arborele implementat dinamic. Deci avem declarația:

```
struct nod
{
int nod;
nod *stg, *dr;
};
```

Pentru că **nodul rădăcină r** este de **tip pointer la nod** cele 3 câmpuri ale nodului rădăcină se accesează astfel:

```
r->info
r->stg
r->dr
```

Deoarece **arborele binar de căutare** este un **arbore binar**, **algoritmii** folosiți pentru **prelucrarea arborilor binari** utilizează **tehnica recursivității** și **metoda Divide et Impera**: prelucrarea unui nod al arborelui se descompune în două **subprobleme**:

- prelucrarea subarborelui stâng;
- prelucrarea subarborelui drept;

la final având loc **îmbinarea** celor două **soluții**.

Toate subprogramele definite pentru prelucrarea arborelui binar de căutare se opresc la arborele vid pentru care **r==NULL.**

1. **Crearea**
2. **Parcurgerea**
3. **Căutarea unei valori printre etichete**
4. **Determinarea cheii cu valoare minimă**
5. **Determinarea cheii cu valoare maximă**
6. **Operații de actualizare**
    a. **Inserarea unui nod**
    b. **Ștergerea unui nod**

1. Crearea unui arbore binar de căutare

Se inițializează arborele cu **arborele vid,** atribuind rădăcinii **adresa NULL:**

r=NULL;

La crearea unui arbore binar de căutare se va realiza adăugarea fiecărui nod ca nod frunză, în poziția corespunzătoare pentru ca arborele să nu-și piardă proprietatea prezentată în definiția sa.

Pentru a căuta nodul la care se va adăuga nodul frunză ca succesor stâng sau succesor drept, se utilizează un pointer care va indica nodul curent. Pointerul se poziționează inițial pe rădăcina r arborelui. Se avansează pe nivelurile arborelui folosind acest pointer, fie prin succesorul stâng, fie prin succesor drept al nodului curent, conform rezultatului obținut în urma comparației dintre valoarea citită pentru cheie și valoarea cheii nodului curent.

Se avansează astfel până în momentul în care pointerul indică un succesor al nodului curent

care este arbore vid.

Valoarea care se inserează o notăm cu **val**.

În limbajul C++ subprogramul pentru crearea arborelui binar de căutare are definiția de mai jos:

```
void Creare (nod *&r, int val)
{
if(r!=NULL)
        {if(val<r->info)
                Creare(r->stg,val);
        else
          if(val>r->info)
                Creare(r->dr,val);
```

```
            else
                cout<<"Valoarea exista!\n";
        }
  else
      {r=new nod;
      r->info=val;
      r->stg=NULL;
      r->dr=NULL;
      }

}
```

Considerăm cazul când nu știm câte valori vor fi citite pentru cheile nodurilor arborelui, dar, ne oprim la întâlnirea valorii 0 care corespunde arborelui vid.

Fie valorile citite pentru chei: **17, 12, 9, 7, 10, 14, 16, 22, 19, 25, 23, 0.**

Arborele binar de căutare care se creează este reprezentat în **Figura 2.2**. **a.**

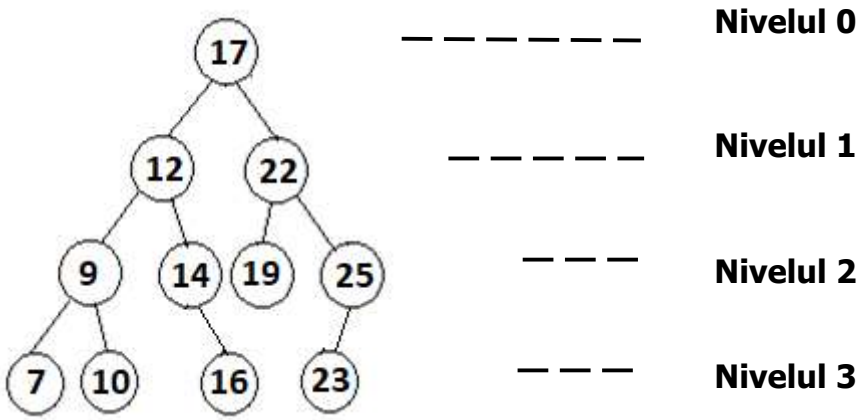


Figura 2.2 a. Arbore binar de căutare de înălțime 3

S-a creat astfel un **arbore binar de căutare** cu **înălțimea egală cu 3.**

Observație

Înălțimea unui **arbore binar de căutare** depinde de **ordinea** în care se introduc **valorile cheii.**

Astfel , dacă valorile citite pentru etichete **sunt: 12, 10, 17, 8, 14, 21, 15, 20, 7, 23, 5, 0** se obține **arborele binar de căutare** din **Figura 2.2 b.**, care are **înălțimea** egală cu **4.**

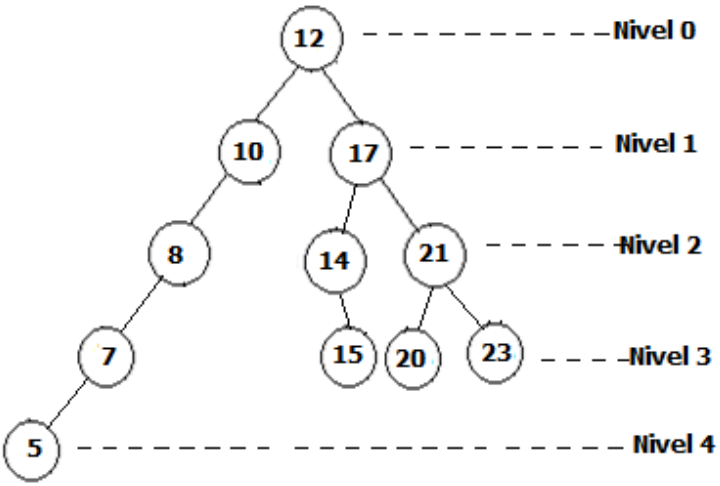


Figura 2.2 b. Arbore binar de căutare de înălțime 4

Arborele binar de căutare din **Figura 2.2 c.** s-a obținut prin citirea valorilor: **10, 8, 12, 14, 17, 15, 20, 21, 23, 0.**

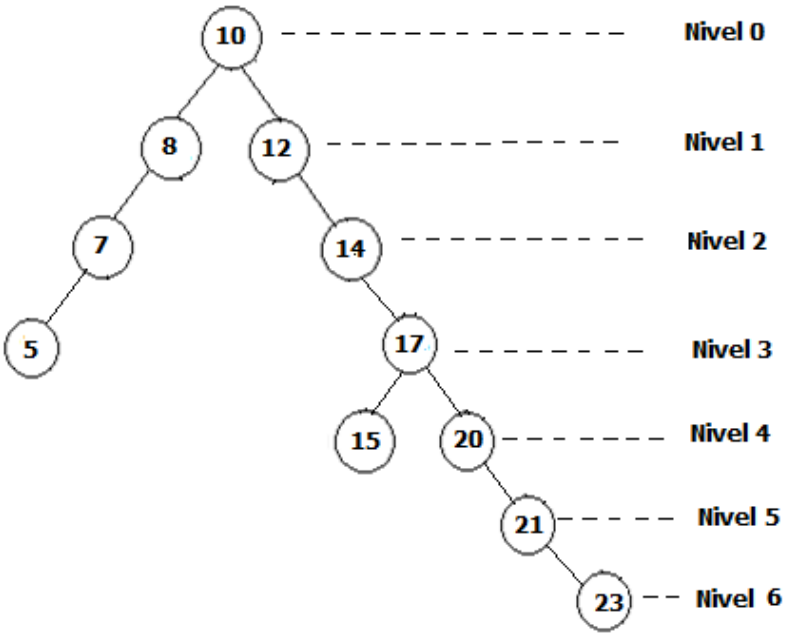


Figura 2.2 c. Arbore binar de căutare de înălțime 6

Dacă șirul de valori pentru cheile nodurilor arborelui, se citește în ordine crescătoare sau descrescătoare a valorilor, se obține un **arbore binar de căutare degenerat**, cu câte un nod pe nivel.

2. Parcurgerea unui arbore binar de căutare

Parcurgerea unui **arbore binar de căutare** înseamnă a vizita fiecare nod al arborelui o singură dată, în scopul prelucrării informației.

Avem două tipuri de parcurgere:

- În **lățime Breadth First** asemănătoare cu cea de la grafuri neorientate

- În **adâncime** care se utilizează cel mai des

Parcurgerea în adâncime

Algoritmii de parcurgere în adâncime sunt specifici arborilor binari și anume:

- **Algoritmul RSD** pentru traversarea în **preordine**: se prelucrează rădăcina, subarborele stâng și apoi subarborele drept;
- **Algoritmul SRD** pentru traversarea în **inordine**: se prelucrează subarborele stâng, rădăcina și apoi subarborele drept;
- **Algoritmul SDR** pentru traversarea în **postordine**: se prelucrează subarborele stâng, subarborele drept și apoi rădăcina.

Fie **arborele binar de căutare** din **Figura 2.3.** Se observă că nodul rădăcina are cheia 12, deci

r->info=12.

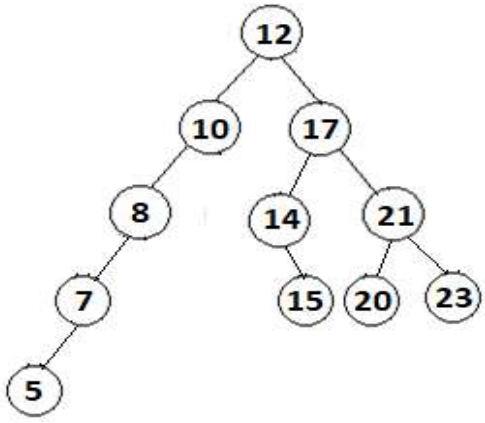


Figura 2.3

Parcurgerile în adâncime sunt:

RSD(12): 12 10 8 7 5 17 14 15 21 20 23
SRD(12): 5 7 8 10 12 14 15 17 21 21 23
SDR(12): 5 7 8 10 15 14 20 23 21 17 12

Funcția pentru:

- **parcurgerea** în **preordine**:

```
void RSD(nod *r)
{if(r!=NULL)
        { cout<<r->info<<" ";
          RSD(r->stg);
          RSD(r->dr);
        }
}
```

Apelul funcției este: **RSD(r);**

Funcția pentru:

- **parcurgerea** în **inordine**:

```
void SRD(nod *r)
{if(r!=NULL)
        { SRD(r->stg);
          cout<<r->info<<" ";
          SRD(r->dr);
        }
}
```

Apelul funcției este: **SRD(r);**

Funcția pentru:

- **parcurgerea** în **postordine**:

```
void SDR(nod *r)
{if(r!=NULL)
        { SDR(r->stg);
          SDR(r->dr);
          cout<<r->info<<" ";
        }
}
```

Apelul funcției este: **SDR(r);**

Observații:

1) Dacă se utilizează **parcurgerea SRD** ***etichetele nodurilor*** vor fi **afișate** în **ordine crescătoare**;
2) **Cheia** cu **valoare minimă** se găsește în **nodul cel mai din stânga** al arborelui, iar **căutarea** se realizează pornind din **rădăcină** și parcurgând numai pe **legătura** cu **succesorul stâng**;
3) **Cheia** cu **valoare maximă** se găsește în **nodul cel mai din dreapta** al arborelui, iar **căutarea** se realizează pornind din **rădăcină** și parcurgând numai pe **legătura** cu **succesorul drept**;

3. Căutarea unei valori printre etichete

Operația de căutare este foarte importantă deoarece se utilizează în majoritatea operațiilor de prelucrare a arborilor binari de căutare.

Se notează cu **x valoarea pentru căutare**, **x** fiind o **valoare întreagă**.

Valoarea x se compară cu cheia rădăcinii arborelui adică **r->info**:

- dacă **informația nodului rădăcină r->info** este **egală** cu valoarea **x** atunci **căutarea** se **încheie cu succes**;
- dacă **informația nodului rădăcină r->info>valoarea x,** se continuă căutarea în subarborele stâng al rădăcinii, altfel se continuă căutarea în subarborele drept al rădăcinii.

Avansarea continuă astfel până când se găsește un nod cu cheia căutată sau până când se ajunge la un arbore vid dacă nu există cheia căutată.

Se prezintă mai jos, în limbajul C++ două subprograme pentru operația de căutare.

Subprogram recursiv cu numele **Cautare ( )**, care realizează **căutarea unei valori x** printre **cheile** arborelui cu **adresa rădăcinii r** și returnează:

- **adresa nodului** care are **cheia cu valoarea** x dacă **valoarea x** se **găsește** în arbore sau
- **NULL** în **caz contrar**

```
nod *Cautare(nod *r, int x)
{if(r!=NULL)
   { if(r->info==x)
        return r;
     else
        if(r->info>x)
                return Cautare(r->stg,x);
         else
                return Cautare(r->dr,x);
    }
 else return NULL;
}
```

Apelul acestui **subprogram** este**:**

p=Cautare(r,x);

unde **p** este de **tip *nod**

Subprogram recursiv cu numele **Cautare**, care are rolul de a **căuta o valoare x** printre **cheile** arborelui cu **adresa rădăcinii r** și care returnează:

- **1** dacă **există cheia** cu **valoarea x** în arbore sau
- **0** în **caz contrar**

```
int  Cautare(nod *r, int x)
{if(r!=NULL)
```

```
        { if(r->info==x)
         return 1;
          else
         if(r->info>x)
                return Cautare(r->stg,x);
             else
                return Cautare(r->dr,x);
         }
     else return 0;
}
```

Apelul acestui **subprogram** este**:**

```
    if(Cautare(r,x)==1)
          cout<<"Valoarea s-a gasit!";
        else
          cout<<"Valoarea NU s-a gasit!";
```

4. Determinarea cheii cu valoare minimă

Așa cum s-a observat**, cheia** cu **valoare minimă** se găsește în **nodul cel mai din stânga** al arborelui, iar **căutarea** se realizează pornind din **rădăcină** și parcurgând numai pe **legătura** cu **succesorul stâng.**

Atunci funcția care determină cheia cu valoare minimă are definiția:

```
int Val_Min(nod *r)
{
if(r->stg!=NULL)
         return Val_Min(r->stg);
  else
         return r->info;
}
```

Apelul funcției este**:**

```
cout<<"Cheia minima="<<Val_Min(r);
```

5. Determinarea cheii cu valoare minimă

Cheia cu **valoare maximă** se găsește în **nodul cel mai din dreapta** al arborelui, iar **căutarea** se realizează pornind din **rădăcină** și parcurgând numai pe **legătura** cu **succesorul drept.**

Atunci funcția care determină cheia cu valoare minimă are definiția:

```
int Val_Max(nod *r)
{
if(r->dr!=NULL)
        return Val_Max(r->dr);
  else
        return r->info;
}
```

Apelul funcției este**:**

```
cout<<"Cheia maxima="<<Val_Max(r);
```

6. Operații de actualizare

a. Operația de inserare a unui nod

Se notează cu x valoarea pentru inserare și se utilizează un pointer pentru căutare care pornește din nodul rădăcină și care va indica nodul curent analizat în arbore. Conform rezultatului obținut în urma comparației dintre valoarea cheii nodului curent și cheia care se inserează, pointerul avansează fie prin succesorul stâng, fie prin succesorul drept.

Înainte de inserare se caută valoarea x:

- dacă se găsește inserarea este abandonată pentru că, într-un arbore binar de căutare nu este permisă existența a două noduri cu chei egale
- dacă valoarea x nu se găsește se ajunge la arborele vid când r devine NULL.

Cheia se va adăuga ca succesor stâng sau drept al nodului curent.

Funcția pentru inserarea unui nod cu valoarea x în arbore este prezentată mai jos:

```
void Inserare(nod *&r, int x)
{if(r!=NULL)
  {if(r->info==x)
        cout<<"Valoarea exista in arbore!";
  else
        if(r->info>x)
              Inserare(r->stg,x);
        else
```

```
            Inserare(r->dr,x);
  }
else
        {r=new nod;
        r->info=x;
        r->stg=NULL;
        r->dr=NULL;
        }
}
```

Apelul subprogramului este: **Inserare(r,x);**

b. Operația de ștergere a unui nod

Valoarea pentru ștergere se notează cu **k**.

Pentru a realiza **ștergerea nodului** cu **eticheta k** dintr-un **arbore binar de căutare** au loc etapele:

1) Se caută nodul cu **eticheta k** în arbore;
2) În urma căutării apar două situații:
   - Dacă **nodul nu există** se va încheia operația de ștergere și se va afișa pe ecran mesajul **"Cheia NU exista!";**
   - Dacă nodul există, el se va șterge.

Atunci când există în arbore nodul cu cheia k, pentru ștergere există următoarele situații:

Caz1 − Nodul este nod frunză și se va șterge efectiv;

Caz 2 − Nodul are un singur succesor: **a.** Doar **succesor drept**; **b.** Doar **succesor stâng.**

În acest caz fiul său îl va înlocui în arbore.

Caz 3 − Nodul are ambii subarbori nevizi. El va fi înlocuit cu nodul cu cea mai mare etichetă din subarborele stâng al său.

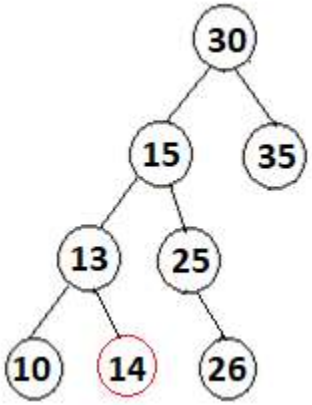
În continuare se prezintă cele patru cazuri.

Se notează cu r pointerul la nodul care urmează să fie șters. Se pornește cu pointerul r de la rădăcină și se poziționează pe nodul care are cheia cu valoarea k.

Caz1 − Nodul este nod frunză, adică nu are succesori

Deci **r>stg==NULL** și **r->dr==NULL.**

Fie **arborele binar de căutare** din **Figura2.4.a**



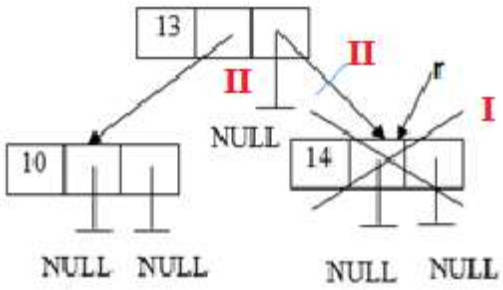


Figura2.4.a **Figura2.4.b**

Figura 2.4 Ștergerea unui nod Caz 1

Se realizează ștergerea nodului cu cheia 14. Adresa acestui nod va fi înlocuită în nodul părinte cu adresa arborelui vid așa cum se prezintă schematic în **Figura2.4 b.**

Algoritmul este:

delete r; **I**

r=NULL; **II**

Caz 2 – Nodul are un singur succesor

a. Doar succesor drept, deci **r->stg==NULL**

Se consideră arborele binar de căutare din **Figura 2.5a.** Se va șterge nodul cu eticheta 25.

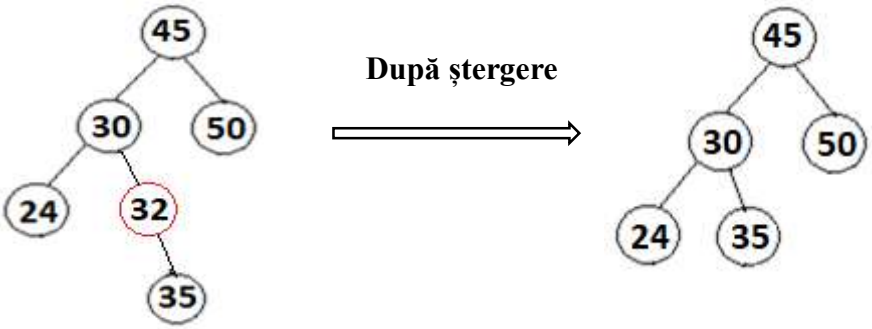


Figura 2.5a Ștergerea unui nod Caz 2 a.

Atunci se va stabili o legătură între părintele său și fiul lui, adică între nodul cu eticheta 30 și nodul cu eticheta 35.

Fie **q pointerul** la **succesorul drept** al **nodului** care va fi **șters**.

Algoritmul este:

q=r->dr; **I**

delete r; **II**

r=q; **III**

Reprezentarea grafică pentru implementarea dinamică apare în **Figura 2.5 b.**

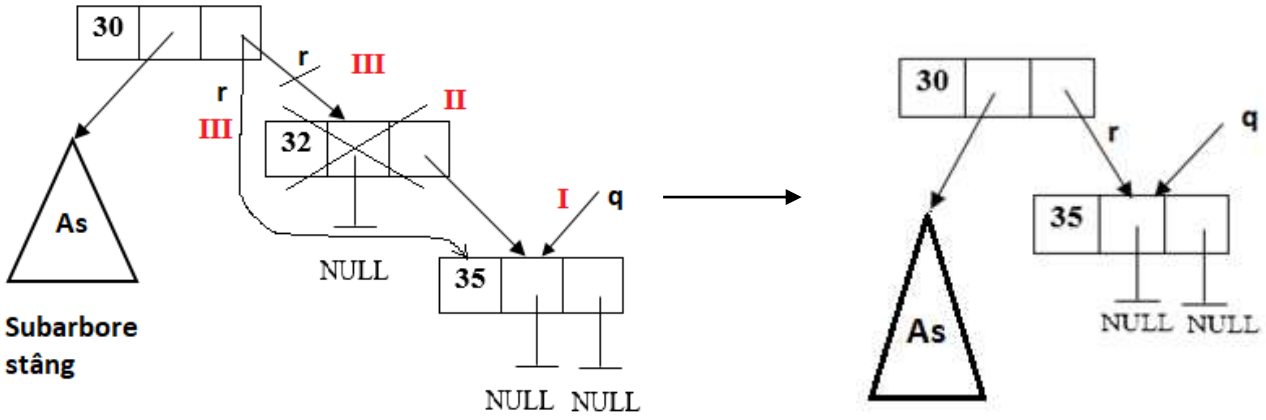


Figura 2.5b Ștergerea unui nod ce are doar succesor drept

b. Doar succesor stâng, deci **r->dr==NULL**

Fie **arborele binar de căutare** din **Figura2.6. a**

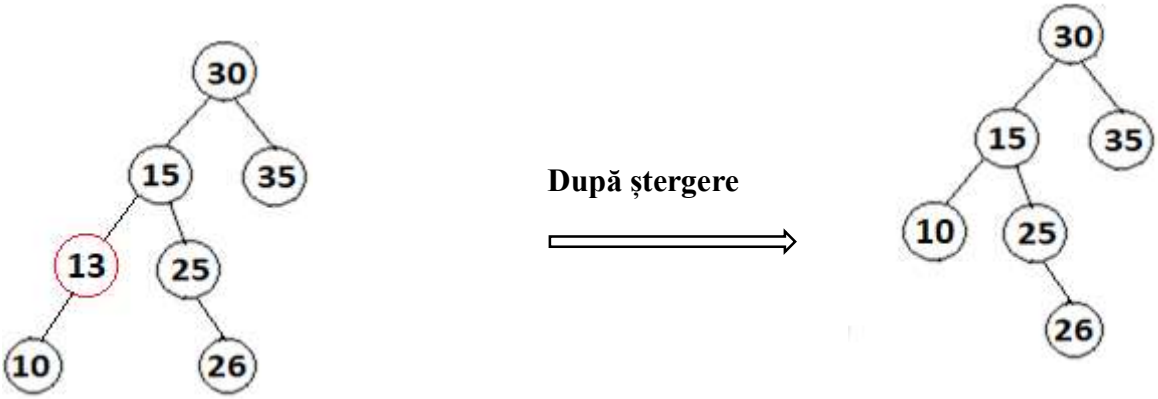


Figura2.6. Ștergerea unui nod Caz 2 b.

Se dorește ștergerea nodului cu eticheta 13. Atunci se va stabili o legătură între părintele său și fiul lui, adică între nodul cu eticheta 15 și nodul cu eticheta 10.

Fie **q pointerul** la **succesorul stâng** al **nodului** care va fi **șters**.

Algoritmul este:

q=r->stg; **I**

delete r; **II**

r=q; **III**

Aceste etape sunt prezentate în **Figura 2.6 b.**

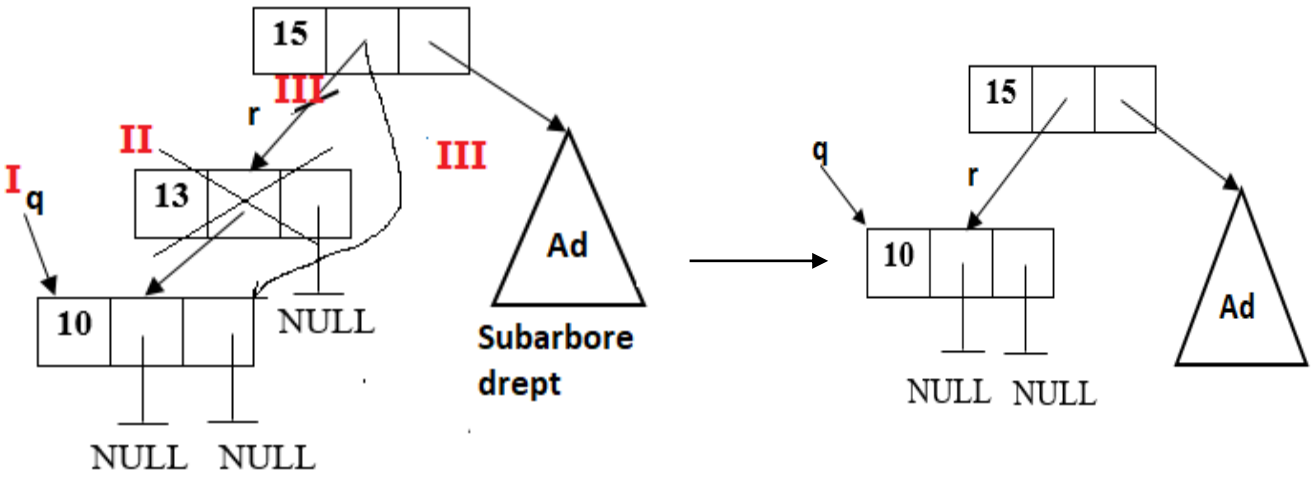


Figura 2.6 b. Ștergerea unui nod ce are doar succesor stâng

Caz 3 − Nodul are ambii subarbori nevizi

Fie arborele binar de căutare din **Figura 2.7 a.** Se dorește ștergerea nodului cu cheia 14.

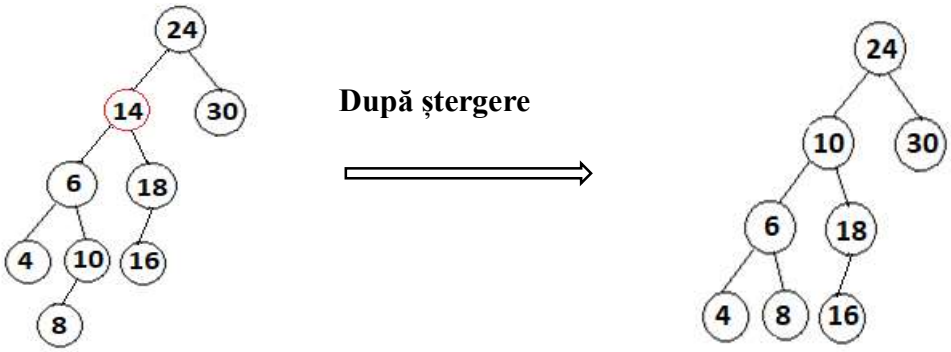


Figura 2.7 a Ștergerea unui nod Caz 3

Pașii sunt:

- Se poziționează **pointerul cmax** pe nodul cu **cheia maximă** din **subarborele stâng** al lui 14, deci pointerul c**max** se va poziționa pe **nodul cu cheia 10.** Nodul cu cheia 10 este cea mai mare valoare din subarborele stâng al nodului cu cheia 14 și deci nu are succesor drept. Înseamnă că, eticheta 10 este mai mică decât toate etichetele din subarborele drept al nodului cu eticheta 14 și deci se poate înlocui valoarea 14 cu valoarea 10;
- Se înlocuiește eticheta nodului 14 cu eticheta nodului 10;
- Ștergem nodul cu valoarea 10, adică un nod ce are doar un succesor stâng(Caz 2b Ștergere).

Se notează cu:

p − pointerul la nodul care trebuie șters, adică la nodul cu eticheta 14

q − pointerul la nodul cu eticheta maximă din subarborele stâng al nodului cu cheia 14

Algoritmul de ștergere este:

p->info=cmax->info; **I**

q=cmax; **II**

cmax=cmax->stg; **III**

delete q; **IV**

Considerând implementarea dinamică avem **Figura 2.7 b.**

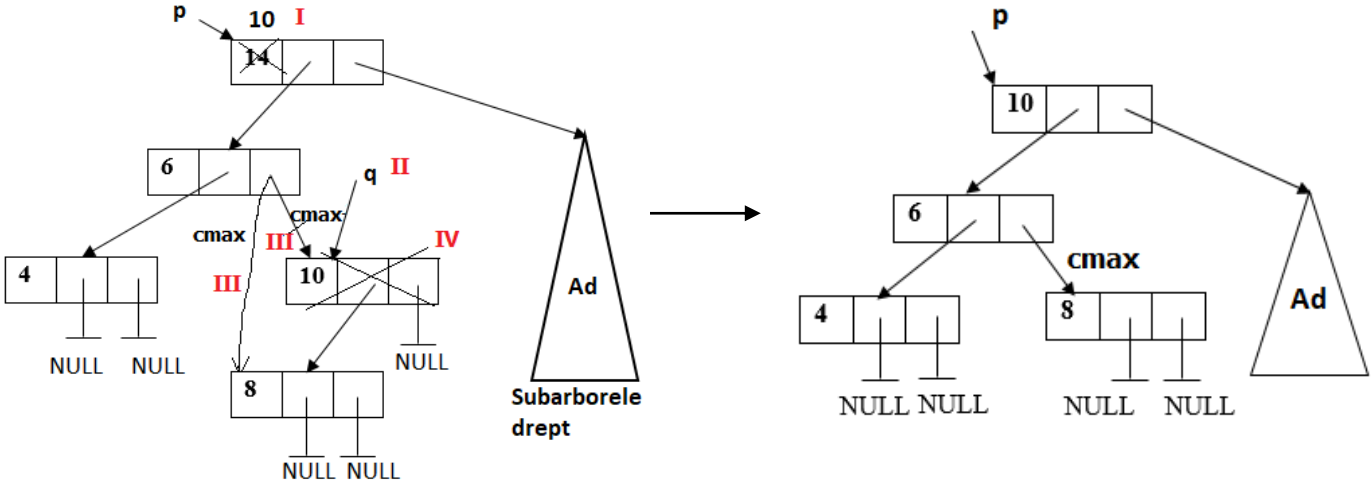


Figura 2.7 b. Ștergerea unui nod cu doi succesori

Mai jos se prezintă definițiile pentru subprogramul **Sterge_nod( )** care realizează ștergerea unui nod care are ambii subarbori nevizi și pentru subprogramul **Stergere( )** care are rolul de a realiza ștergerea unui nod, conform situațiilor prezentate anterior.

```
void StergeNod(nod *&p,nod *&cmax)
{nod *q;
   if(cmax->dr!=NULL) StergeNod(p, cmax->dr);
        else
         {p->info=cmax->info;
          q=cmax;
          cmax=cmax->stg;
          delete q;
         }
}
void Stergere(nod *&r, int k)
{nod *q;
if(r!=NULL)
        {if(r->info==k)
           if(r->stg==NULL&&r->dr==NULL)
             //Caz 1
```

```
            {
            delete r;
            r=NULL;
            }
          else
         //Caz 2a
          if(r->stg==NULL)
            {q=r->dr;
             delete r;
             r=q;
                 }
             else
           //Caz 2b
            if(r->dr==NULL)
             {q=r->stg;
               delete r;
               r=q;
              }
             else
           //Caz 3
                StergeNod(r,r->stg);
       else
        if(r->info>k)
            Stergere(r->stg,k);
         else
            Stergere(r->dr,k);
       }
        else cout<<"Cheia NU exista"<<endl;
}
```

Apelul subprogramului **Stergere ( )** este**: Stergere(r,k);**

2.1.3 APLICAȚIE ARBORI BINARI DE CĂUTARE

Pe prima linie a fișierului **date.in** se găsește un șir de valori întregi, despărțite prin câte un spațiu, șir care se termină cu valoarea 0, primele valori fiind nenule. Aceste valori reprezintă etichetele nodurilor unui ***arbore binar de căutare implementat dinamic***.

a. Să se creeze arborele prin citirea valorilor din fișier;
b. Să se realizeze parcurgerile **SRD** și **SDR;**
c. Să se determine **înălțimea arborelui;**
d. Să se verifice dacă o valoare întreagă **x** citită de la tastatură *există* printre cheile arborelui;
e. Să se afișeze **nodurile frunză de pe fiecare nivel al arborelui;**
f. Să se *insereze* un nod ce are cheia cu **valoarea** egală cu **dublul cheii maxime** din arbore
g. Să *șteargă* nodul cu **cheia minimă** și **nodul rădăcină**.

Afișarea rezultatelor se va face în fișierul **date.out.**

Exemplu:

date.in
15 11 20 17 24 12 18 8 23 26 5 30 7 27 0

De la tastatură se citește x=100

date.out
SRD:5 7 8 11 12 15 17 18 20 23 24 26 27 30
SDR:7 5 8 12 11 18 17 23 27 30 26 24 20 15
Inaltimea=5
Valoarea x=100 nu s-a gasit
Nodurile frunza:7 12 18 23 27
Cheia minima=5
Cheia maxima=30
SDR după inserare:7 5 8 12 11 18 17 23 27 60 30 26 24 20 15
SDR dupa stergere minim:7 8 12 11 18 17 23 27 60 30 26 24 20 15
SDR dupa stergere radacina:7 8 11 18 17 23 27 60 30 26 24 20 12

Observații

1) Înălțimea arborelui se va calcula recursiv astfel:
1+ valoarea maximă dintre înălțimile subarborilor stâng și drept ai arborelui.
Se utilizează subprogramele **Max ( )** și **Inalt ( )**
2) Folosind funcția **Frunze_Niv ( )** se afișează nodurile frunză, în parcurgere SRD a arborelui.

Implementare în limbajul C++

```
#include <iostream>
#include <fstream>

using namespace std;

ifstream fin("date.in");
ofstream fout("date.out");
 struct nod
 {
    int info;
    nod *stg,*dr;
 };
int h,k;
void Creare (nod *&r, int val)
{
```

```
if(r!=NULL)
          {if(val<r->info)
                    Creare(r->stg,val);
            else
             if(val>r->info)
                     Creare(r->dr,val);
      else
       cout<<"Valoarea exista!\n";
          }
          else
      {r=new nod;
      r->info=val;
      r->stg=NULL;
      r->dr=NULL;
      }
}
void SRD(nod *r)
{if(r!=NULL)
   { SRD(r->stg);
     fout<<r->info<<" ";
     SRD(r->dr);
   }
}
void SDR(nod *r)
{if(r!=NULL)
   { SDR(r->stg);
     SDR(r->dr);
     fout<<r->info<<" ";
   }
}
int Max (int a,int b)
{
   if(a>b) return a;
   else return b;
}
int Inalt(nod *r)
{
   if(r==NULL)
```

```
      return -1;
   else
   return 1+Max(Inalt(r->stg),Inalt(r->dr));
}
int  Cautare(nod *r, int x)
{if(r!=NULL)
    { if(r->info==x)
          return 1;
      else
          if(r->info>x)
                return Cautare(r->stg,x);
          else
                return Cautare(r->dr,x);
     }
 else return 0;
}
void Frunze_Niv(nod *r)
{
   if(r!=NULL)
      {
      Frunze_Niv(r->stg);
      if(r->stg==NULL&r->dr==NULL)
           fout<<r->info<<" ";
      Frunze_Niv(r->dr);
      }
}

int Val_Min(nod *r)
{
if(r->stg!=NULL)
          return Val_Min(r->stg);
  else
         return r->info;
}
int Val_Max(nod *r)
{
if(r->dr!=NULL)
          return Val_Max(r->dr);
```

```
 else
        return r->info;
}
void Inserare(nod *&r,int x)
{if(r!=NULL)
   {if(r->info==x)
         cout<<"Valoarea exista in arbore!";
   else
         if(r->info>x)
                  Inserare(r->stg,x);
         else
                  Inserare(r->dr,x);
  }
else
   {r=new nod;
   r->info=x;
   r->stg=NULL;
   r->dr=NULL;
   }
}
void StergeNod(nod *&p,nod *&cmax)
{nod *q;
    if(cmax->dr!=NULL) StergeNod(p,cmax->dr);
         else
           {p->info=cmax->info;
            q=cmax;
            cmax=cmax->stg;
           delete q;
           }
}
void Stergere(nod *&r, int k)
{nod *q;
if(r!=NULL)
         {if(r->info==k)
            if(r->stg==NULL&&r->dr==NULL)
                  //Caz 1
                  {
                delete r;
```

```
                r=NULL;
                }
      else
        //Caz 2a
       if(r->stg==NULL)
         {q=r->dr;
          delete r;
          r=q;
         }
         else
       //Caz 2b
          if(r->dr==NULL)
             {q=r->stg;
              delete r;
              r=q;
              }
           else
             //Caz 3
             StergeNod(r,r->stg);
        else
         if(r->info>k)
             Stergere(r->stg,k);
          else
              Stergere(r->dr,k);
        }
          else cout<<"Cheia NU exista"<<endl;
}

int main()
{
  int x,val,cmin,cmax;
  nod *r;
  r=NULL;
  fin>>val;
  while(val!=0)
  {
    Creare(r,val);
    fin>>val;
```

```
    }
    fout<<"SRD:";
    SRD(r);
    fout<<"\nSDR:";
    SDR(r);
    h=Inalt(r);
    fout<<"\nInaltimea="<<h;
    cout<<"x=";
    cin>>x;
    if(Cautare(r,x)==1)
       fout<<"\nValoarea x="<<x<<" s-a gasit!";
    else
       fout<<"\nValoarea x="<<x<<" nu s-a gasit!";
    fout<<"\nNodurile frunza";
    Frunze_Niv(r);
    cmin=Val_Min(r);
    cmax=Val_Max(r);
    fout<<"\nCheia minima="<<cmin;
    fout<<"\nCheia maxima="<<cmax;
    Inserare(r,2*cmax);
    fout<<"\nSDR dupa inserare:";
    SDR(r);
    Stergere(r,cmin);
    fout<<"\nSDR dupa stergere minim:";
    SDR(r);
    Stergere(r,r->info);
    fout<<"\nSDR dupa stergere radacina:";
    SDR(r);
    return 0;
}
```