Arbori binari de căutare (BST)

1. **Introducere**
2. **Inserarea și căutarea unui nod într-un arbore binar de căutare**
3. **Ștergerea unui nod dintr-un arbore binar de căutare**

1. Introducere

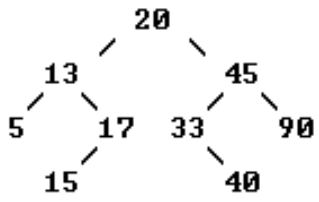
Arborii binari de căutare (*En.* **BST** – *Binary Search Tree*) sunt o implementare a tipului de date abstract numit "dicționar". Elementele unui dicționar pot fi ordonate, criteriul de sortare fiind dat de "cheia de sortare" (sau cheie de căutare).

Arborii binari de căutare implementează eficient următoarele operații:

search(arbore, k) - determină dacă un element specificat prin cheia de sortare k, există în arbore și-l returnează dacă există;
insert(arbore, x) - inserează în arbore elementul x, de cheie dată (**Nu există chei duble!**);
delete(arbore, k) - șterge un element specificat prin cheia k.

Proprietatea care definește structura unui arbore binar de căutare este următoarea: valoarea cheii memorate în rădacină este **strict** mai mare decât toate valorile cheilor conținute în subarborele stâng și **strict** mai mică decât toate valorile cheilor conținute în subarborele drept.
Această proprietate trebuie să fie îndeplinită pentru toți subarborii, de pe orice nivel din arborele binar de căutare. Prin urmare, afișarea în inordine a unui astfel de arbore va ordona nodurile în ordinea crescătoare a cheilor.

Exemplu (am reprezentat pentru fiecare nod numai cheile de căutare, de tip întreg în acest caz):



2. Inserarea și căutarea unui nod într-un arbore binar de căutare

a) varianta recursivă:

```
insert(r,a) // r - pointer la radacina (trasmis prin referinta)
            // a - atomul de inserat
{
         if r=0 then   r = make_nod(a)
         else if key(a) < key(data(r)) then
                             insert(lchild(r),a)
         else if key(a) > key(data(r)) then
                             insert(rchild(r),a)
}

make_nod(a)   // creeaza un nod nou in care memoreaza atomul a
{
    p = get_sp()   // aloca spatiu pentru un nod
    data(p) = a
    lchild(p) = rchild(p) = 0
    return (p)
}
```

Pentru varianta de mai sus, trebuie să permitem funcției ***insert*** să modifice valoarea argumentului **r**, pentru aceasta el va fi un parametru transmis prin referință. In implementarea C++ funcția ***insert*** va avea prototipul:

```
void insert(Nod*& r, Atom a);
```

O variantă care nu necesită argument referință (deci poate fi implementată în C) este dată mai jos.

```
insert(r,a)
{
     if r=0 then return ( make_nod(a) )
     else if key(a) < key(data(r)) then
                 lchild(r) = insert(lchild(r),a)
     else if key(a) > key(data(r)) then
                  rchild(r) = insert(rchild(r),a)
     return (r)
 }
```

Apelul acestei variante va avea de fiecare dată forma:

```
rad = insert(rad, a)
```

Procedura ***search*** întoarce pointer la nodul cu cheia de căutare dată sau pointerul NULL dacă nu există nodul respectiv.

```
search(r,k)
{
     if ( r=0 )  return NULL
     else if k < key(data(r)) then
                        return ( search(lchild(r),k) )
     else if k > key(data(r)) then
                        return ( search(rchild(r),k) )
     else return (r)
}
```

b) varianta nerecursivă

Trebuie observat că atât operația ***search*** cât și operația ***insert*** parcurg o ramura a arborelui (o cale de la rădăcină spre o frunză). Aceasta parcurgere poate fi efectuată iterativ. Este vorba de a parcurge o înlănțuire, deci se poate face o analogie cu parcurgerea unei liste înlănțuite.

```
Parcurgerea listei inlantuite
p=cap;
while(p!=0){
       ┌──────────────────────┐
       │ Prelucrare element   │
       └──────────────────────┘
        p = p->link;
}
```

```
Parcurgerea unei ramuri in arbore
p=rad;
while(p!=0){
       ┌──────────────────────┐
       │ Prelucrea nod        │
       └──────────────────────┘
        if(conditie) p=p->stg
        else p=p->drt;
}
```

Funcția de inserare în arborele binar de căutare, realizată nerecursiv, are următoarea formă (presupunem că ***r*** este parametru transmis prin referință):

```
insert(r,a)
{
     if ( r=0 ) r = make_nod(a)
     else {p = r
           while ( p<>0 )
             {
              p1 = p
              if key(a)<key(data(p)) then  p=lchild(p)
              else if key(a)>key(data(p)) then  p=rchild(p)
              else return
          }
           if ( key(a)<key(data(p1)) )
              lchild(p1) = make_nod(a)
           else rchild(p1) = make_nod(a)
     }
}
```

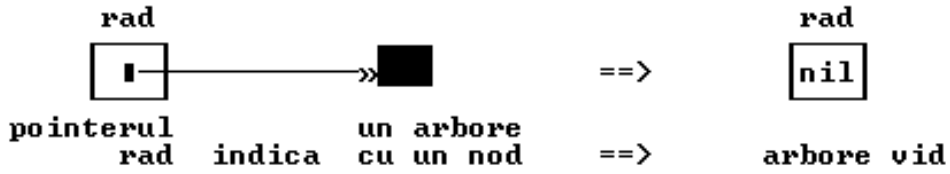
3. Ștergerea unui nod dintr-un arbore binar de căutare

Pentru ștergerea unei valori din arbore este necesară mai întâi identificarea nodului care conține această valoare. Vom folosi pentru aceasta tehnica prezentată la operația ***search***. Pentru simplitate, considerăm nodurile etichetate cu numere întregi care vor contitui chiar cheile de căutare `(key(data(p) = data(p))`.

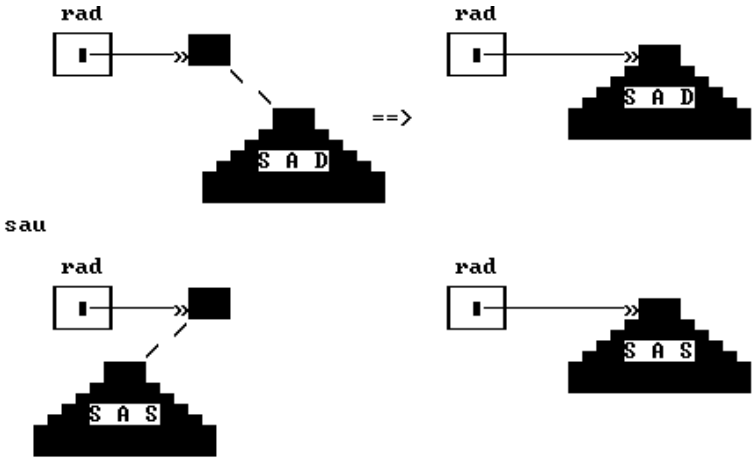
```
struct Nod{
     int data;
     Nod* stg, *drt;
     };

void delete(Nod*& rad, int a)
{
      if(rad==NULL)
           printf("Eroare: Valoarea %d nu este in arbore!", a);
      else if( a<rad->data )  delete(rad->stg,a)
      else if( a>rad->data )  delete(rad->drt,a)
      else deleteRoot(rad);
}
```

Am redus sarcina inițială la a scrie funcția ***deleteRoot*** care șterge rădăcina unui arbore binar de căutare nevid. Pentru aceasta, avem următoarele cazuri:

- Atunci când rădăcina nu are niciun descendent ștergerea este o operație imediată.

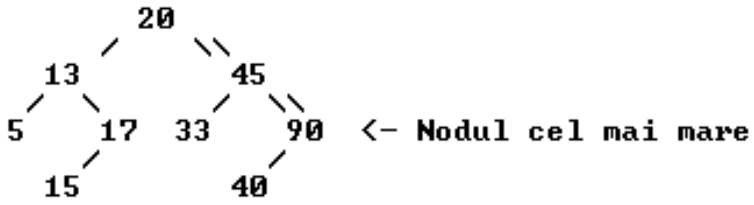


Atunci când rădăcina are un sigur descendent nodul șters va fi înlocuit cu subarborele descendent.

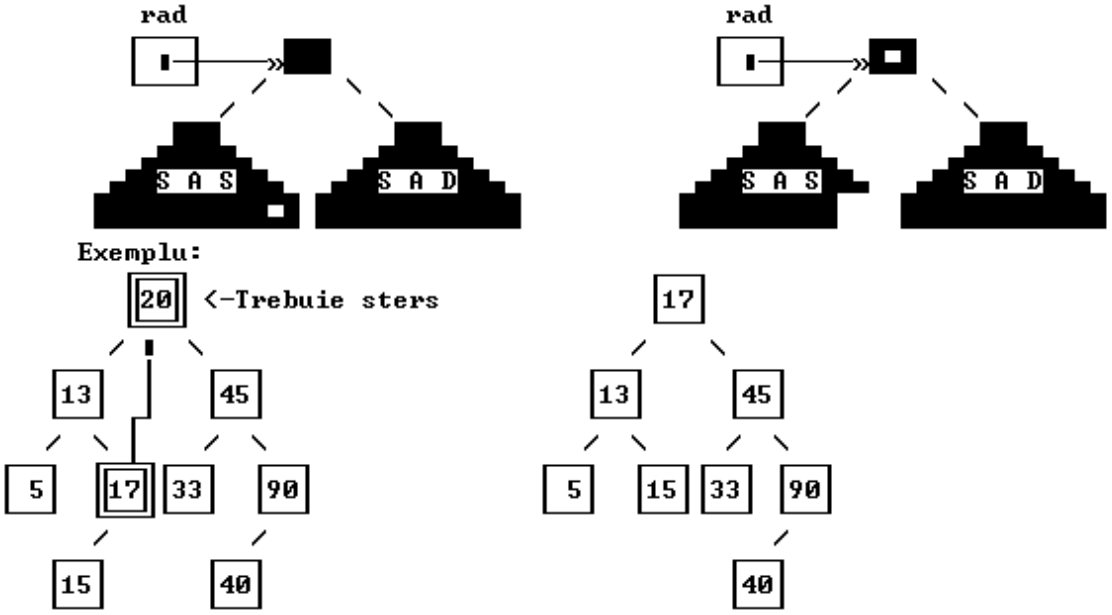


- Atunci când rădăcina are doi descendenți, aceasta va fi înlocuită cu nodul cu nodul de cheie maximă din subarborele stâng, acest nod având întotdeuana cel mult un descendent. Nodul cel mai mare dintr-un arbore (subarbore) binar de căutare se găsește pornind din rădăcină și înaintând cât se poate spre dreapta.

De exemplu



Deci:



Următoarea funcție **detașează** (**nu șterge!**) dintr-un arbore binar de căutare nevid nodul cu valoarea cea mai mare și întoarce pointer la acest nod.

```
Nod* removeGreatest(Nod*& r)
{
     Nod* p;
     if( r->drt==0 ) {
                       p = r;
                       r = r->stg;
                       return p;
                       }
     else return removeGreatest(r->rchild);
}
```

Varianta prezentată este recursivă. Se poate scrie ușor și o variantă nerecursivă pentru această procedură.

Ținând cont de cazurile posibile prezentate procedura ***deleteRoot*** va trata separat cazurile:

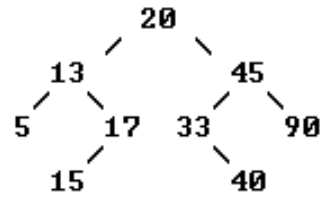
- dacă subarborele stâng este vid: promovează subarborele drept. Cazul în care și subarborele drept este vid nu trebuie tratat separat, în acest caz se promovează arborele vid (rad devine NULL);
- altfel, dacă subarborele drept este vid: promoveaza subarborele stâng.
- altfel (ambii subarbori nu sunt vizi): înlocuiește rădăcina cu cel mai mare nod din subarborele stâng.

```
void deleteRoot(Nod*& rad)
{
     Nod* p = rad;
     if( rad->stg==0) rad = rad->drt;
     else if( rad->drt==0) rad = rad->stg;
     else {
           rad = removeGreatest (rad->stg);
           rad->stg = p->stg;
           rad->drt = p->drt;
          }
     delete p;
}
```

TEMA

Se citește de la intrare un șir de valori numerice întregi, pe o linie, separate de spații, șir care se încheie cu o valoare 0 :

a) Să se introducă valorile citite într-un arbore binar de căutare (exclusiv valoarea 0 care încheie șirul) ;
b) Să se afișeze în inordine conținutul arborelui ;
c) Se citește o valoare pentru care să se verifice dacă este sau nu conținută în arbore ;
d) Se citește o valoare care să fie ștearsă din arbore și apoi să se afișeze arborele în inordine ;
e) Să se scrie funcții pentru afișarea nodurilor de cheie minimă respectiv maximă din arborele binar de căutare ;
f) Să se afișeze arborele indentat, asemănător afișării folderelor în Explorer. De exemplu, arborele



va fi afișat (”-” semnifică descendent nul) :

20
13
5
-
-
17
15
-
-
-
45
33
-
40
-
-
90
-
-

g) Dealocați zona de memorie ocupată de arbore.

Exerciții:

1. Se introduc următoarele valori (chei) într-un BST: 30, 20, 38, 10, 35, 25, 42, 40, 22. Precizați care este înălțimea arborelui binar de căutare și câte frunze sunt. Cum arată afișarea în postordine?
2. Reprezentarea în preordine a unui arbore binar este: 22, 11, 5, 7, 15, 33, 30, 41, 35. Este acesta un BST? Nodurile de cheie minimă și maximă sunt frunze?
3. Reprezentați BST-ul corespunzător următoarei afișări în postordine: 15, 8, 30, 25, 20, 38, 40, 44, 35.