Arbori binari de cautare

1. Arbori binari de cautare

1.1. Introducere

Arborii binari de cautare sint o implementare a tipului de date abstract numit "dictionar". Elementele unui dictionar pot fi ordonate, criteriul de sortare fiind dat de "cheia de sortare" (sau cheie de cautare).

Arborii binari de cautare implementeaza eficient urmatoarele operatii:

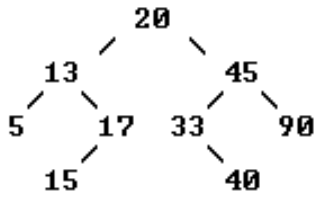
search(arbore, k) - determina daca un element specificat prin cheia de sortare k, exista in arbore si-l returneaza daca exista.

insert(arbore, x) - insereaza in arbore elementul x.

delete(arbore, k) - sterge un element specificat, specificat prin cheia k.

Proprietatea care defineste structura unui arbore binar de cautare este urmatoarea: Valoarea cheii memorate in radacina este mai mare decit toate valorile cheilor continute in subarborele sting si mai mica decit toate valorile cheilor continute in subarborele drept. Aceasta proprietate trebuie sa fie indeplinita pentru toti subarborii, de pe orice nivel in arborele binar de cautare.

Exemplu (am reprezentat pentru fiecare nod numai cheile de cautare):



1.2. Inserarea si cautarea unui nod intr-un arbore binar de cautare

a) varianta recursiva :

```
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 sa permitem functiei ***insert*** sa modifice valoarea argumentului **r**, pentru aceasta el va fi un parametru transmis prin referinta. In implementarea C++ functia ***insert*** va avea prototipul:

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

O varianta care nu necesita argument referinta (deci poate fi implementata in C) este data 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 data forma:

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

Procedura ***search*** intoarce pointer la nodul cu cheia de cautare data sau pointerul NULL daca nu exista 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 nerecursiva

Trebuie observat ca atit operatia ***search*** cit si operatia ***insert*** parcurg o ramura a arborelui (un lant de la radacina spre o frunza). Aceasta parcurgere poate fi efectuata iterativ. Este vorba de a parcurge o inlantuire, deci se impune o analogie cu parcurgerea listei inlantuite.

```
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;
      }
```

Procedura de inserare in arborele binar de cautare, realizata nerecursiv, are urmatoarea forma (presupunem ca ***r*** este parametru transmis prin referinta):

```
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)
     }
}
```

1.3. Stergerea unui nod dintr-un arbore binar de cautare

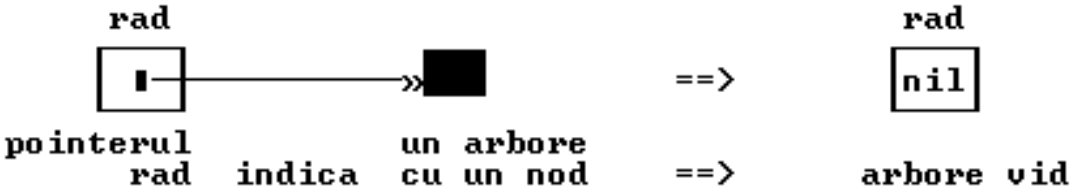
In continuare vom scrie functia C++ pentru stergerea unei valori dintr-un arbore binar de cautare care contine numere intregi.

Pentru stergerea unei valori din arbore este necesara mai intii identificarea nodului care contine aceasta valoare. Vom folosi pentru aceasta tehnica prezentata la operatia ***search***. Pentru simplitate consideram nodurile etichetate cu numere intregi care vor contitui chiar cheile de cautare `(key(data(p) = data(p))`.

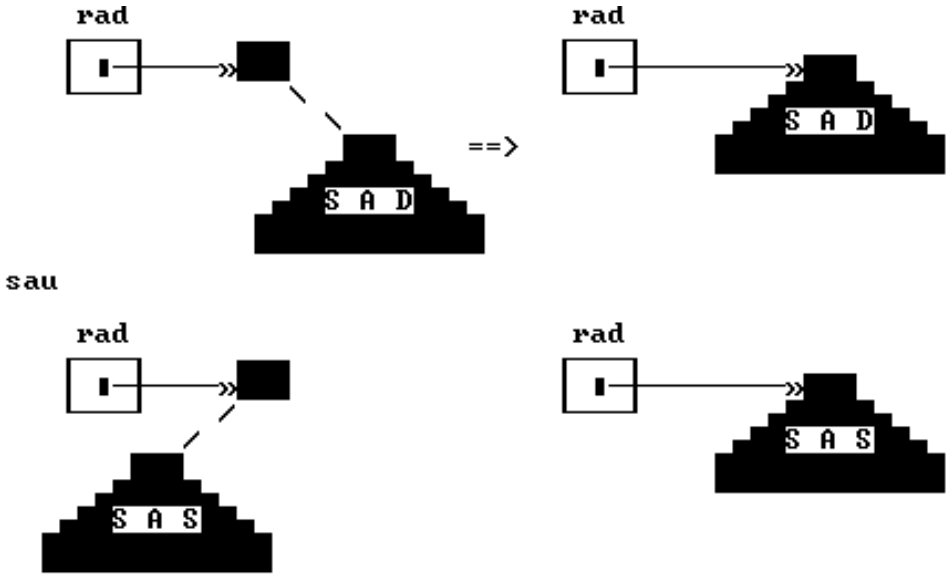
```
struct Nod{
     int data;
     Nod* stg, *drt;
     };


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 initiala la a scrie functia ***deleteRoot*** care sterge radacina unui arbore binar de cautare nevid. Pentru aceasta avem urmatoarele cazuri:

- Atunci cind radacina nu are nici un descendent stergerea este o operatie imediata.

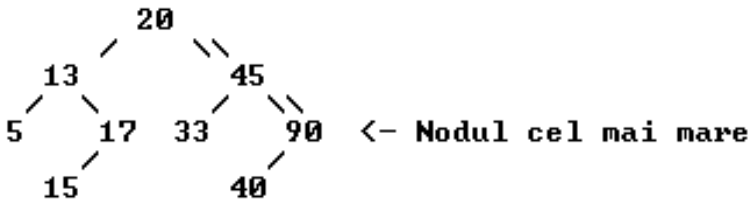


- Atunci cind radacina are un sigur descendent nodul sters va fi inlocuit cu subarborele descendent.

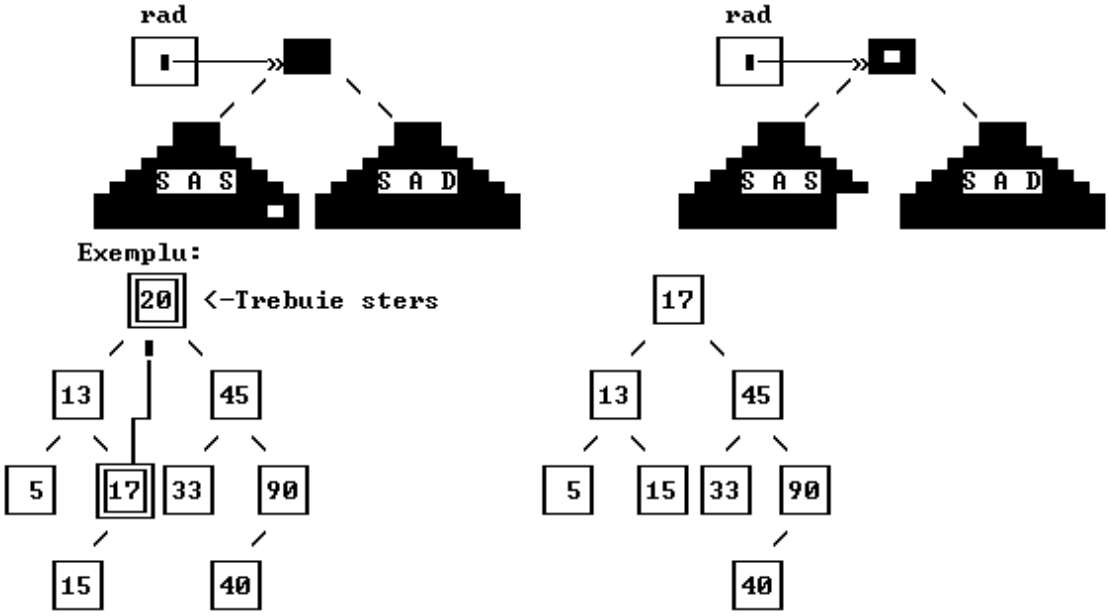


- Atunci cind radacina are doi descendenti, ea va fi inlocuita cu nodul cu valoarea cea mai mare din subarborele sting, acest nod avind intotdeuana cel mult un descendent. Nodul cel mai mare dintr-un arbore (subarbore) binar de cautare se gaseste pornind din radacina si inaintind cit se poate spre dreapta.

De exemplu



Deci:



Urmatoarea functie detaseaza dintr-un arbore binar de cautare nevid nodul cu valoarea cea mai mare si intoarce 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 prezentata este recursiva. Se poate scrie usor si o varianta nerecursiva pentru aceasta procedura.

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

- daca subarborele sting este vid: promoveaza subarborele drept. Cazul in care si subarborele drept este vid nu trebuie tratat separat, in acest caz se promoveaza arborele vid (rad devine NULL);
- altfel daca, subarborele drept este vid: promoveaza subarborele sting.
- altfel (ambii subarbori nu sint vizi): inlocuieste radacina cu cel mai mare nod din subarborele sting.

```
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;
}
```

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 10

TEMA

1. Se citeste de la intrare un sir de valori numerice intregi, pe o linie, separate de spatii, sir care se incheie cu o valoare 0.

a) Sa se introduca valorile citite intr-un arbore binar de cautare (exclusiv valoarea 0 care incheie sirul).

b) Sa se afiseze continutul arborelui in inordine, preordine si postordine.

c) Se citeste o valoare pentru care sa se raspunda daca este sau nu continuta in arbore.

d) Se citeste o valoare care sa fie stearsa din arbore si apoi sa se afiseze arborele in inordine.

Implementati variantele recursive ale functiilor de cautare, inserare, stergere.

2. Reluati problema anterioara, dar de aceasta data realizati implementarile iterative (nerecursive) ale functiilor de cautare, inserare, stergere.

NOTARE

Problema 1: citire valori 0.5p; a) 1.5p; b) 1.5p; c) 1.5p; d) 2p;

Problema 2: cautare 1.5p; inserare 1.5p; stergere 3p.