Arbori binari de căutare.

1. Definiții şi generalități.

TAD Dictionar este o mulțime total ordonată după valoarea cheii, având ca operații reprezentative: căutarea, inserarea şi ştergerea. O implementare eficientă a dicționarelor se face cu arbori binari de căutare.

Intr-un arbore binar de căutare, pentru fiecare nod, cheile din subarborele stâng sunt mai mici decât cheia din nod, care este mai mică decât cheile din subarborele drept, adică:

`cheie(SS(x)) < cheie(x) < cheie(SD(x))`

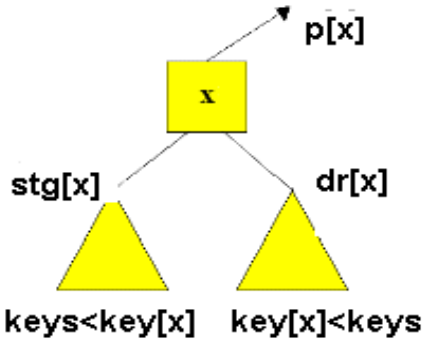


Fig. 1

Proprietățile arborilor binari de căutare:

- fiecare subarbore este un arbore binar de căutare
- cheile din subarborele stâng al unui nod sunt mai mici decât cheia din nod
- cheile din subarborele drept al unui nod sunt mai mari decât cheia din nod
- cheia minimă (/maximă) se află în cel mai din stânga (/dreapta) nod
- înălțimea arborelui binar de căutare este cuprinsă între $\log_2 n$ şi $n-1$.

2. Operații specifice arborilor binari de căutare.

Interfața cuprinde, în plus, față de **`TAD Arbore Binar`** funcțiile:

```
// determina adresa nodului cu cheia maxima
Arb  ABC_Max (Arb a);
// determina adresa nodului cu cheia minima
Arb  ABC_Min (Arb a);
// determina adresa nodului succesor in inordine
Arb  ABC_Succ (Arb x, Arb a);
// determina adresa nodului predecesor in inordine
Arb  ABC_Pred (Arb x, Arb a);
// determina adresa nodului cu cheia data
Arb  ABC_Search (Arb a, void *ch, PFC comp);
// insereaza cheia in arborele binar de cautare
void ABC_Insert (Arb *a, void *ch, PFC comp);
// sterge nodul cu cheia data din arborele binar de cautare
void ABC_Remove (Arb *a, void *ch, PFC comp);
```

Traversarea în inordine a unui arbore binar de căutare dă cheile sortate crescător. Această observație stă la baza unei metode de sortare (*treesort*).

1. Căutarea unei chei.

Căutarea unei chei într-un arbore binar de căutare este asemănătoare căutării binare: cheia căutată este comparată cu cheia din nodul curent (inițial nodul rădăcină). In funcție de rezultatul comparației se continuă căutarea în subarborele stâng , subarborele drept sau se termină în nodul curent întrucât s-a găsit cheia căutată.

```
// varianta recursiva
Arb ABC_Search (Arb a, void *ch, PFC comp){
  if(AB_Empty(a)) return NULL;
  if(comp(ch, AB_GetKey(a)) == 0) return a;
  if(comp(ch, AB_GetKey(a)) < 0)
     return ABC_Search(AB_GetSS(a), ch, comp);
  return ABC_Search (AB_GetSD(a), ch, comp);
}

// varianta nerecursiva
Arb ABC_Search (Arb a, void *ch, PFC comp){
  while(!AB_Empty(a)){
    if(comp(ch, AB_GetKey(a)) == 0)
       return a;
    if(comp(ch, AB_GetKey(a)) < 0)
       a = AB_GetSS(a);
    else
       a = AB_GetSD(a);
  };
  return NULL;
}
```

2. Inserarea unui nod

Inserarea unui nod se face, în funcție de rezultatul comparației cheilor, în subarborele stâng sau drept. Dacă cheia există, nu se mai inserează. Dacă arborele este vid, se crează un nod care devine nod rădăcină al arborelui. În caz contrar, cheia se inderează ca fiu stânga sau fiu dreapta a unui nod din arbore.

```
void ABC_Insert (Arb *a, void *ch, PFC comp){
  if(AB_Empty(*a)){
    *a = AB_CrNod(ch, NULL, NULL, NULL, 0);
    return;
  };
  Arb prec = 0, crt = *a;
  while(crt){
    if(comp(ch, AB_GetKey(crt))==0) return;
    prec = crt;
    if(comp(ch, AB_GetKey(crt)) < 0){
       crt = AB_GetSS(crt);
       dir = -1;
    }
    else {
```

```
        crt = AB_GetSD(crt);
        dir = 1;
    }
  crt = AB_CrNod(ch, 0, 0, prec, dir);
}
```

3. Cheia maximă dintr-un arbore binar de căutare.

Adresa nodului având cea mai mare cheie dintr-un arbore binar de căutare se determină simplu prin coborârea pe subarborele drept:

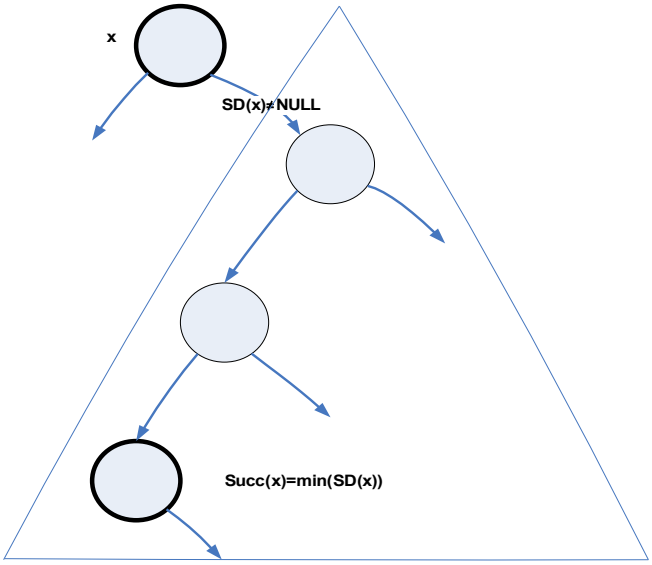
```
Arb ABC_Max (Arb a){
  if(AB_Empty(a)) return NULL;
  while(AB_GetSD(a))
    a = AB_GetSD(a);
  return a;
}
```

4. Succesorul şi predecesorul unui nod.

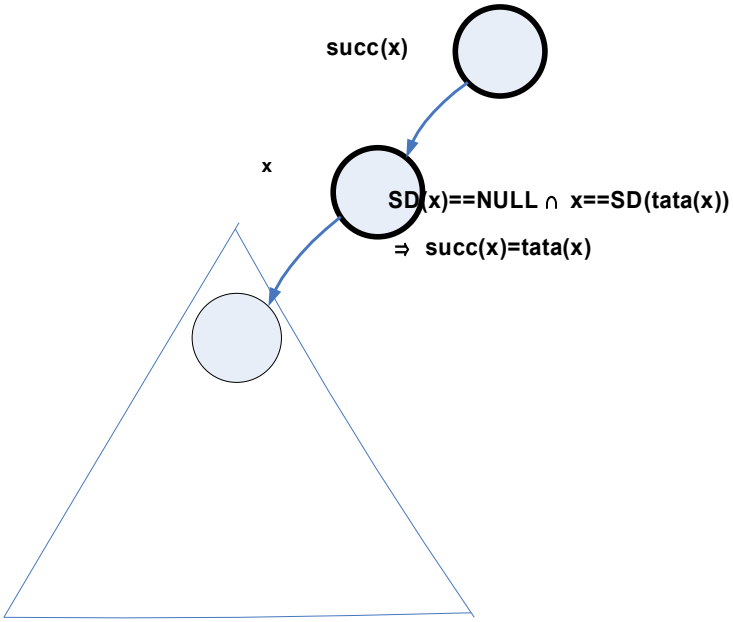
Succesorul unui nod, la traversarea în inordine, se defineşte ca:

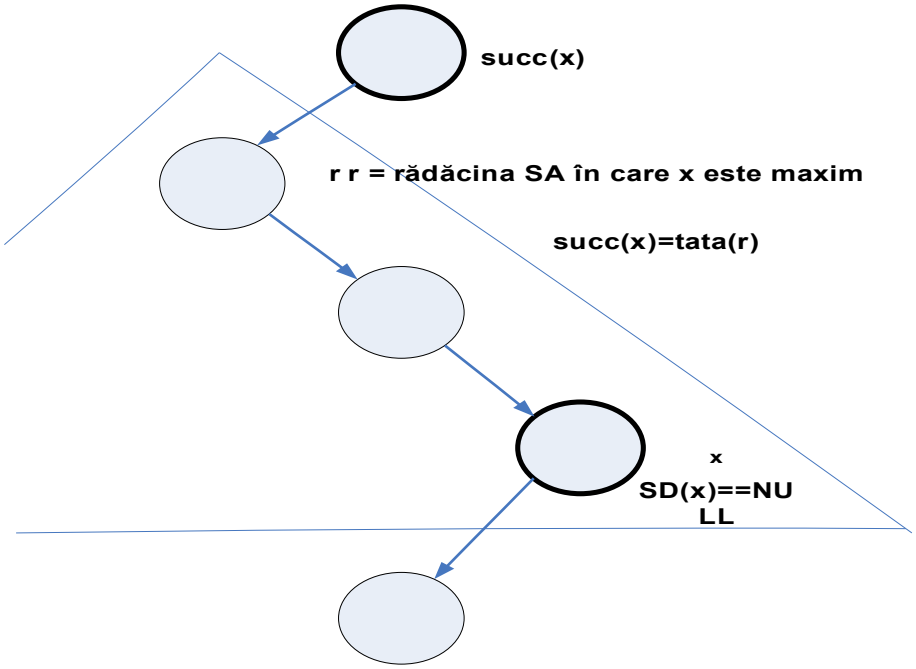
- elementul minim din subarborele drept al nodului, dacă nodul are subarbore drept

y = succ(x) = min(SD(x)) dacă SD(x) ≠ NULL



- dacă nodul nu are subarbore drept, el va fi element maxim într-un subarbore. Părintele rădăcinii acestui subarbore este nodul succesor.






```
Arb ABC_Succ (Arb x, Arb a){
  if(AB_Empty(a) || AB_Empty(x) || x==ABC_Max (a))
    return NULL;
  if(AB_GetSD(x))
    return ABC_Min (AB_GetSD(x));
  Arb y = AB_GetPred(x);
  while(AB_GetSD(y) == x){
    x = y;
    y = AB_GetPred(y);
  };
```

```
  return y;
}
```

Similar, predecesorul unui nod este:

- elementul maxim din subarborele stâng al nodului, dacă nodul are subarbore stâng

y = pred(x) = max(SS(x)) dacă **SS(x) ≠ NULL**

- dacă nodul nu are subarbore stâng, el va fi element minim într-un subarbore. Părintele rădăcinii acestui subarbore este nodul succesor.

```
Arb ABC_Pred (Arb x, Arb a){
  if(AB_Empty(a) || AB_Empty(x) || x==ABC_Min(a))
    return NULL;
  if(AB_GetSS(x))
    return ABC_Max(AB_GetSS(x));
  Arb y = AB_GetPred(x);
  while(AB_GetSS(y) == x){
    x = y;
    y = AB_GetPred(y);
  };
  return y;
}
```

5. Ştergerea unui nod.

Ştergerea unui nod este o operație mai complicată, întrucât presupune o rearanjare a nodurilor.

Pentru ştergerea unui nod având o cheie dată se foloseşte funcția **ABC_Search()** pentru a găsi nodul ce urmează a fi şters. Sunt posibile următoarele cazuri:

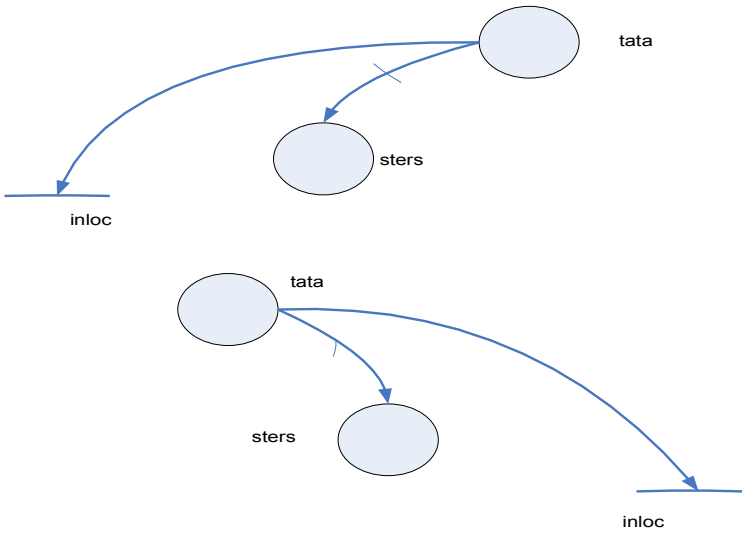
1. nodul de şters nu există; operația se consideră încheiată.
2. nodul de şters este frunză sau are un singur succesor.
3. nodul de şters are 2 succesori.

In raport cu nodul care urmează a fi şters vom defini predecesorul acestuia (tatăl) şi nodul de înlocuire.

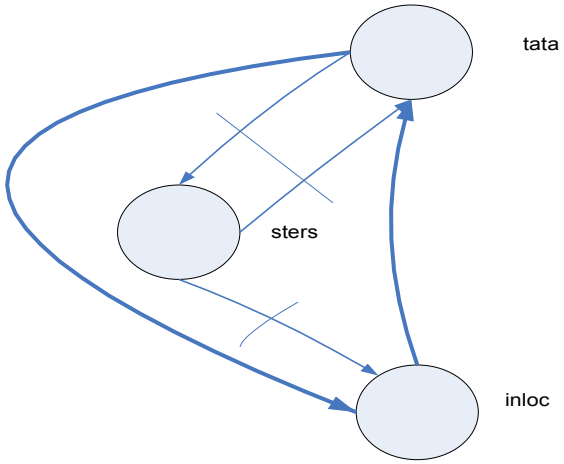
In cazul ştergerii unui nod frunză, sau a unui nod având un singur succesor, legătura de la tatăl nodului de şters la acesta este modificată pentru a indica nodul de înlocuire.

Dacă se şterge un nod frunză, nodul de înlocuire nu există (este **NULL**):

```
inloc = NULL;
if(AB_GetSS(tata)==sters)  //sters e fiu stanga
  AB_SetSS(tata, inloc);
else
  AB_SetSD(tata, inloc);
```



In cazul unui nod cu un singur fiu, nodul de înlocuire va fi acel fiu. Legătura la fiul şters va fi înlocuită prin legătura la nodul de înlocuire, iar predecesorul nodului de înlocuire se modifică, devenind nodul tată.



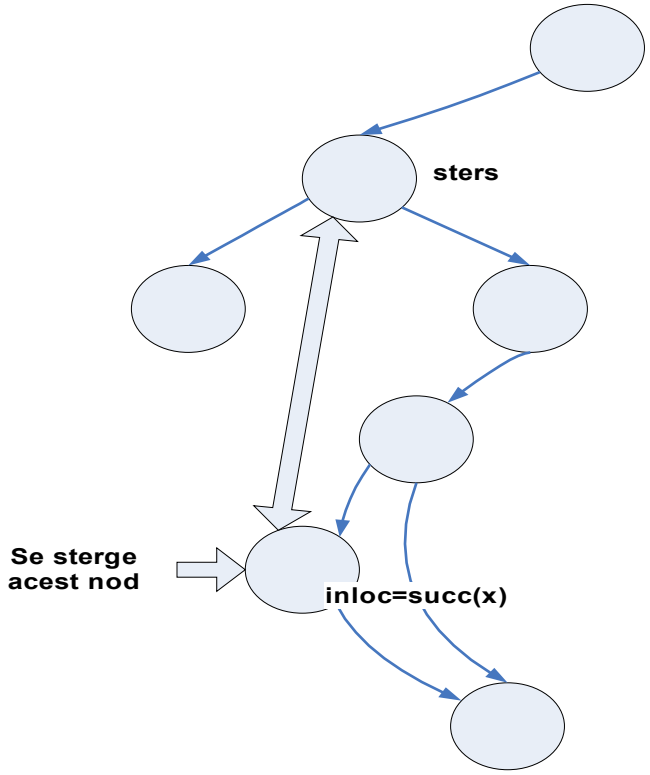

```
int dir;
// determinare nod de inlocuire
if(AB_GetSS(sters)                  //determina inloc
   inloc = AB_GetSS(sters);
else
   inloc = AB_GetSD(sters);
// actualizarea legaturii predecesorului
if(AB_GetSS(tata)==sters)
   dir = -1;
else
   dir = 1;
// actualizare legatura la predecesor nod de inlocuire
AB_SetPred(inloc, tata, dir);
```

Ştergerea unui nod frunză sau a unui nod cu un singur succesor poate fi exprimată prin funcţia **sterge1():**

```
void sterge1(Arb* sters, Arb* a){
```

```
    int dir;
    Arb tata, inloc;
    tata = AB_GetPred(*sters);
    if(AB_GetSS(*sters))
       inloc = AB_GetSS(*sters);
    else
       inloc = AB_GetSD(*sters);
    if(tata)
       if(AB_GetSS(tata)==*sters)
          dir = -1;
       else
          dir = 1;
    else
       *a = inloc;
    if(inloc)
       AB_SetPred(inloc, tata, dir);
    else
       if(dir==-1)
          AB_SetSS(tata, NULL);
       else
          AB_SetSD(tata, NULL);
    free(*sters);
}
```

Dacă nodul de şters are 2 succesori, se caută succesorul în inordine al nodului de şters (nodul de înlocuire). Acesta va avea un singur succesor (sau niciunul). Se mută cheia din nodul de înlocuire în nodul de şters şi se şterge nodul de înlocuire (care are cel mult un succesor cu **`sterge1()`**).



```
void ABC_Remove(Arb* a, void *ch, PFC comp){
   Arb sters, inloc, tata;
   sters = ABC_Search (*a, ch, comp);
   if(!sters) return;
   if(AB_GetSS(sters) && AB_GetSD(sters)){
      inloc = ABC_Succ (sters);
      AB_SetKey(sters, AB_GetKey(inloc));
      sterge1(&inloc, a);
   }
   else
      sterge1(&sters, a);
}
```

Complexitatea operațiilor **`Search(), Min(), Max(), Succ(), Pred(), Insert(), Remove(), Traversare()`** într-un arbore binar de căutare este $O(d)=O(\log_2 n)$.

Treesort.

Probleme propuse.

1. TAD Listă Ordonată se implementează eficient cu arbori binari de căutare. Primul element din listă este elementul minim din arborele binar de căutare, ultimul element din listă este elementul maxim, avansul la următorul element din listă poate fi făcut cu funcția succesor, căutarea unui element în lista ordonată se poate face cu funcția **`SearchABC()`**, cu complexitate logaritmică şi nu secvențială, etc.

2. Considerăm un arbore binar complet având chei caractere preluate din şirul de caractere "PORTUGALIA'.
   a. Scrieți parcurgerile în preordine, inordine şi postordine
   b. Creați, folosind şirul de caractere de mai sus, un arbore binar de căutare.
   c. Scrieți o funcție care crează un arbore binar de căutare folosind drept chei literele dintr-un şir de caractere dat ca parametru, litere convertite în prealabil în majuscule. O cheie egală cu cheia dintr-un nod este inserată în subarborele stâng.

3. Scrieți o funcție care stabileşte dacă un arbore binar reprezentat cu pointeri este sau nu arbore binar de căutare.

4. Functie care creaza un arbore binar de cautare folosind drept chei caracterele dintr-un sir dat ca parametru. Functia are prototipul: `cr_arb(Arb* a, char* expr);`

5. Scrieți o funcție nerecursivă pentru traversarea în inordine a unui arbore binar de căutare, afişând cheile nodurilor.
Indicație: Traversareea porneşte din nodul cu cheia minimă şi se opreşte în nodul cu cheia maximă. Se avansează folosind succesorul în inordine.