Arbori binari de căutare

1 Noţiuni de bază

Definiţie: un arbore binar de căutare este un arbore binar cu următoarele proprietăţi.

- Fiecare nod are o valoare numită cheie
- Pentru fiecare nod este valabil:
  - Toate nodurile din subarborele stâng au cheile mai mici decât cheia părintelui.
  - Toate nodurile din subarborele drept au cheile mai mari decât cheia părintelui.

În cazul în care relaţia de ordine nu este strictă, dacă nodurile cu chei egale se inserează pe aceeaşi parte a arborelui, inserţia multor noduri cu chei egale are ca urmare obţinerea unui arbore relativ dezechilibrat, având ca urmare o creştere a complexităţii operaţiilor. În continuare se consideră arbori binari de căutare cu chei distincte. Cazul cheilor egale se va discuta separat la sfârşitul cursului.

În figura 1 a) este prezentat un exemplu de arbore binar de căutare.

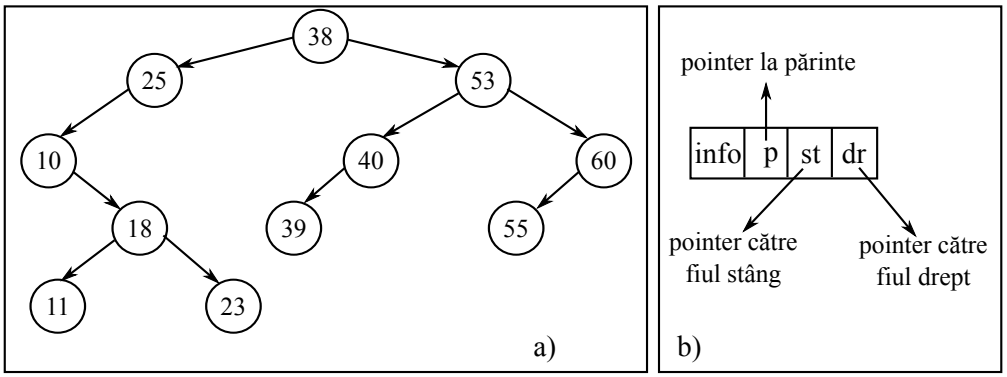

Figura 1: a) Exemplu de arbore binar de căutare; b) Structura unui nod din arbore.

Penru fiecare nod al arborelui se consideră structura din figura 1 b), în care fiecare nod x are câmpurile: $x.info$ = cheia nodului, $x.st$ şi $x.dr$ = fiul stâng şi respectiv fiul drept, $x.p$= părintele nodului x.

2 Operaţii într-un arbore binar de căutare

Operaţiile de bază într-un arbore binar de căutare sunt:

- Căutarea binară a unei chei.
- Determinarea nodului cu cheia minimă/maximă din arbore.
- Căutarea binară a succesorului / predecesorului unui nod
- Inserţie/ştergere a unui nod cu o anumită cheie
- Sortarea cheilor arborelui, prin parcurgerea acestuia în inordine .

Observaţii:

1. Complexitatea operaţiilor într-un arbore binar de căutare este proporţională cu înălţimea arborelui. De fapt, dacă arborele conţine n noduri atunci $O(log_2 n) \leq T(n) \leq O(n)$, $T(n)$ = complexitatea algoritmului utilizat.
2. În cazul unui arbore binar de căutare oarecare nu poate fi garantată complexitatea căutării binare, adică $O(log_2 n)$.

Există arbori binari de căutare care se autobalansează, de exemplu arborii AVL şi arborii roşu-negru. Pentru aceştia se demonstrează faptul că au complexitatea operaţiilor de ordinul $O(\log_2 n)$.

2.1 Căutarea binară

Problemă: Considerând un arbore binar de căutare T cu rădăcina $T.rad$, să se determine nodul cu o cheie dată k.

Soluţie: La fiecare moment dat compar cheia nodului curent x, $x.info$, cu k. Dacă $x.info = k$ atunci se returnează nodul x. Dacă $x.info < k$ atunci se continuă căutarea în subarborele drept al lui x, altfel se continuă căutarea în subarborele stâng al lui x.

Algoritm 1: Căutare binară

Intrare: Arborele binar de căutare T şi elementul k care se caută
Iesire: nodul x cu cheia k sau Nil
$x \leftarrow T.rad$
cat_timp $x \leq Nil$ *şi* $x.info \neq k$ **executa**
  daca $k < x.info$ **atunci**
    $x \leftarrow x.st$
  sfarsit_daca
  altfel
    $x \leftarrow x.dr$
  sfarsit_daca
sfarsit_cat_timp
return x

2.2 Minimul dintr-un arbore de căutare

Nodul cu informaţia minimă din subarborele de rădăcină x a unui arbore binar de căutare poate fi găsit pornind de la nodul x şi coborând în descendenţii stângi până la cea mai din stânga frunză. Funcţia AB_MIN returnează nodul cu informaţia minimă.

Algoritm 2: AB_MIN

Intrare: Arborele binar de căutare T şi nodul x
Iesire: nodul cu cheia minimă din subarborele de rădăcină x
$y \leftarrow x$
cat_timp $y.st \neq Nil$ **executa**
  $y \leftarrow y.st$
sfarsit_cat_timp
return y

Observaţie: maximul se determină în mod similar şi anume parcurgând descendenţii drepţi până la cea mai din dreapta frunză.

2.3 Succesorul binar

Succesorul unui nod x într-un arbore binar de căutare este acel nod y din arbore, a cărui cheie are valoarea imediat următoare cheii lui x în şirul sortat al valorilor din arbore.

- Poate fi determinat prin comparaţii
- Dacă există, este:
  - Cel mai mic element din $x.dr$, dacă $x.dr \neq Nil$
  - Un nod părinte y pentru care x se află în subarborele stâng al lui y, dacă x nu are descendent drept.
- Dacă x este nodul cu cea mai mare cheie, atunci x nu are succesor.

Exemplu: În arborele din figura 2 obţinem:

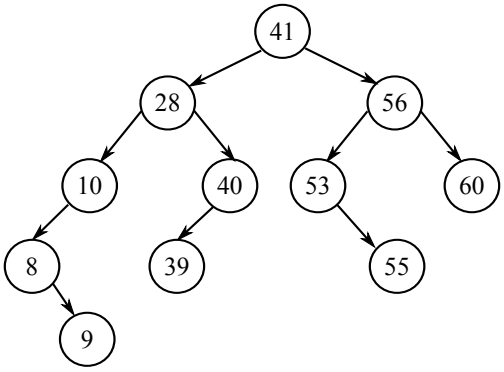

Figura 2: Arbore binar de căutare.

AB_SUCCESOR(41) = 53
AB_SUCCESOR(39) = 40
AB_SUCCESOR(9) = 10
AB_SUCCESOR(60) - nu există

Algoritm 3: Succesor

Intrare: Arborele binar de căutare T şi nodul x
Iesire: nodul y, sucesor binar al lui x

```
daca x.dr ≠ Nil atunci
|  y = AB_MIN(x.dr)
sfarsit_daca
altfel
|  y ← x.p
|  cat_timp y ≠ Nil şi x = y.dr executa
|  |  x ← y
|  |  y ← y.p
|  sfarsit_cat_timp
sfarsit_daca
return y
```

2.4 Inserarea unui nod

Se consieră arborele binar de căutare T cu rădăcina $T.rad$ şi nodul z, care are câmpurile

$$z.info = k, z.st = Nil, z.dr = Nil.$$

Se doreşte inserarea acestui nod în arborele binar T.
Ideea principală este următoare: pornind de la rădăcină se coboară în arbore, până la un nod, care are cel mult un fiu şi care poate fi părintele nodului z. Pentru a respecta proprietatea de arbore binar de căutare, dacă informaţia nodului curent x este mai mare decât $z.info$, atunci z se va insera în subarborele stâng al lui x, altfel se va insera în

subarborele drept. În algoritmul următor x reprezintă nodul curent, care iniţial este $T.rad$ = rădăcina lui T, y reprezintă părintele lui x, iniţial Nil.

Algoritm 4: Insert

Intrare: Arborele binar de căutare T şi cheia k
Iesire: Arborele în care s-a inserat nodul z cu cheia k

```
aloca memorie pentru nodul z
z.info ← k
z.st ← Nil
z.dr ← Nil
x ← T.rad
y ← Nil
cat_timp X ≠ Nil executa
    y ← x
    daca z.info < x.info atunci
        x ← x.st
    sfarsit_daca
    altfel
        x ← x.dr
    sfarsit_daca
sfarsit_cat_timp
z.p ← y
daca y = Nil atunci
    T.rad ← Z
sfarsit_daca
altfel
    daca z.info < y.info atunci
        y.st ← z
    sfarsit_daca
    altfel
        y.dr ← z
    sfarsit_daca
sfarsit_daca
```

În figura 3 este ilustrat algoritmul de inserţie a unui nod cu cheia 38 într-un arbore binar de căutare.

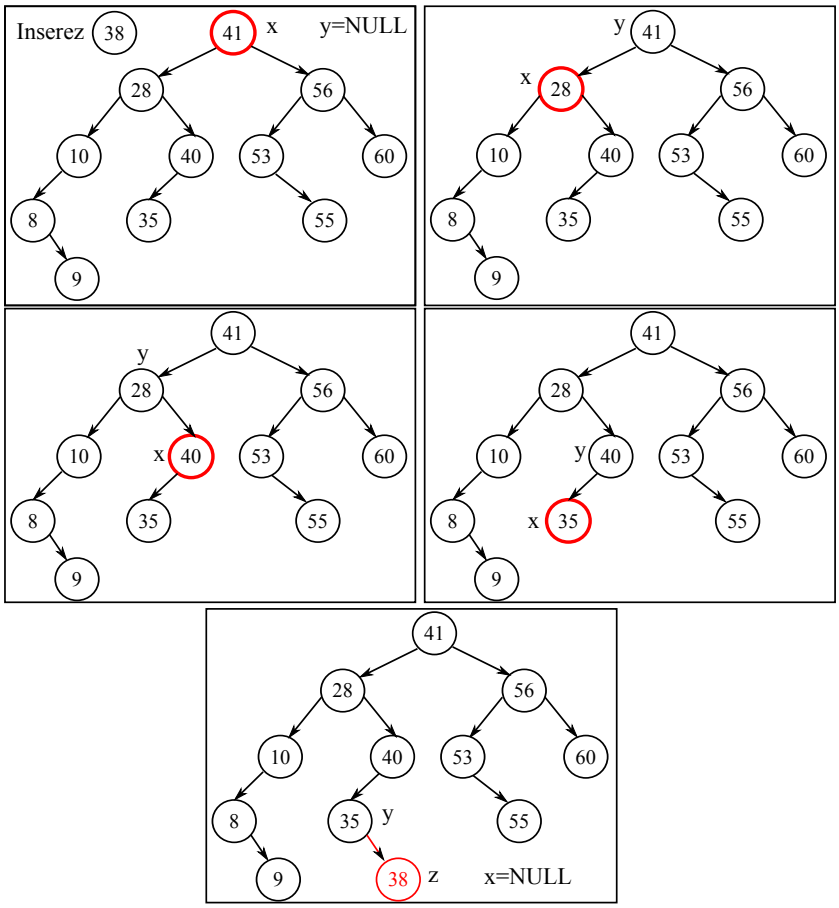

Figura 3: Inserarea nodului cu cheia 28 într-un arbore binar. Nodul curent cu care se compară nodul ce se inserează este marcat cu roşu.

2.5 Ştergerea unui nod

Ştergerea unui nod z dintr-un arbore binar T cu rădăcina $T.rad$ este ceva mai elaborată decât inserarea. Sunt luate în considerare următoarele cazuri:

1. z nu are fii şi atunci este pur şi simplu înlocuit cu Nil
2. z are un singur fiu nenul. Atunci se înlocuieşte z cu acel fiu
3. z are doi fii nenuli. Atunci se determină succesorul y al lui z care se află în subarborele drept al lui z şi evident nu are descendent stâng. Apoi se înlocuieşte nodul z cu nodul y, iar y se înlocuieşte cu fiul său drept.

Observaţie: În cazul în care z are doi descendenţi nenuli, el poate fi înlocuit şi cu predecesorul său.

În funcţia AB_STERGE se consideră T arborele binar, z nodul care trebuie şters şi y nodul cu care se înlocuieşte z. Cele 3 cazuri descrise mai sus vor fi cuprinse în funcţie în următoarele cazuri:

1. z nu are fiu stâng $\Rightarrow$ se înlocuieşte z cu fiul drept - eventual Nil. Acest caz include şi cazul în care z nu are nici un fiu.
2. z nu are fiu drept $\Rightarrow$ se înlocuieşte z cu fiul stâng
3. z are ambii fii nenuli $\Rightarrow$ y=sucesorul lui z care se află în subarborele drept al lui z şi are fiul stâng Nil
   a. y este descendentul drept direct al lui $z \Rightarrow$ se înlocuieşte z cu y (fiul drept al lui y rămâne neschimbat iar fiul stâng al lui z devine fiul stâng al lui y)
   b. y nu este descendentul drept direct al lui $z \Rightarrow$ se înlocuieşte y cu fiul său drept iar apoi se înlocuieşte nodul z cu nodul y.

În bibliografie (T.H. Cormen - *Introduction to Algorithms*) - este propusă utilizarea unei funcţii ajutătoare TRANSPLANT(T,u,v) care înlocuieşte în arborele T nodul u ca subarbore cu nodul v - de fapt această funcţie realizează doar managementul legaturilor între părintele lui u şi nodul v, legăturile cu fiii se realizează separat în funcţia de ştergere propriu-zisă.

Cazurile luate în considerare de către funcţia AB_DELETE sunt ilustrate în figura 4.

Algoritm 5: Transplant

Intrare: Arborele binar de căutare T, nodrile u şi v
Iesire: Arborele în care s-a înlocuit u cu v ca fiu al lui $u.p$

```
daca u.p = Nil atunci
 | T.rad ← v
sfarsit_daca
altfel
 | daca u = u.p.st atunci
 |  | u.p.st ← v
 | sfarsit_daca
 | altfel
 |  | u.p.dr ← v
 | sfarsit_daca
sfarsit_daca
daca v ≠ Nil atunci
 | v.p ← u.p
sfarsit_daca
```

Modul de funcţionare al funcţiei AB_TRANSPLANT este ilustrat în figura 5.

În continuare este prezentat algoritmul de ştergere a unui nod z dintr-un arbore binar de căutare T.

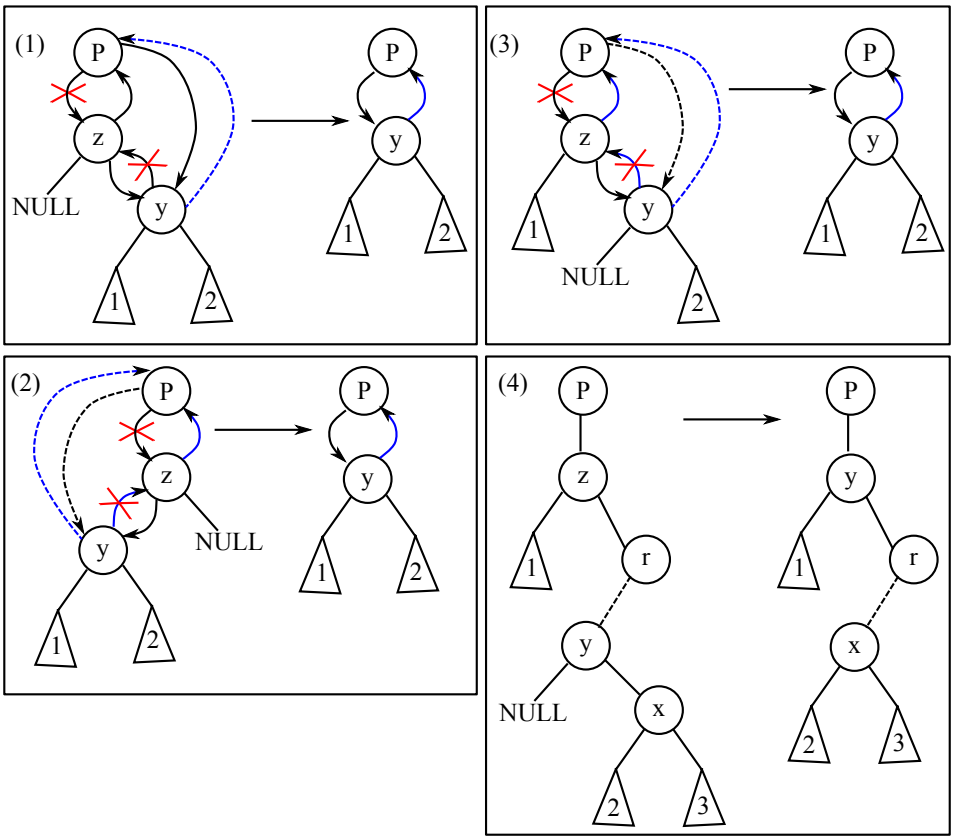

Figura 4: Ştergerea unui nod z care: (1) are fiul stâng nul; (2) are fiul drept nul; (3) are ambii fii nenuli, dar succesorul y este descendent direct al lui z; (4) are ambii fii nenuli, dar succesorul y nu este descendent direct al lui z.

Algoritm 6: AB_DELETE

Intrare: Arborele binar de căutare T şi nodul z, găsit cu funcţia *Search*
Iesire: Arborele din care s-a şters z

```
daca z.st = Nil atunci
    Tansplant(z, z.dr)
sfarsit_daca
altfel
    daca z.dr = Nil atunci
        Tansplant(z, z.st)
    sfarsit_daca
    altfel
        y ← succesor(z)
        daca y ≠ z.dr atunci
            Transplant(y, y.dr)
            y.dr ← z.dr
            z.dr.p ← y
        sfarsit_daca
        Transplant(z, y)
        y.st ← z.st
        z.st.p ← y
    sfarsit_daca
sfarsit_daca
```

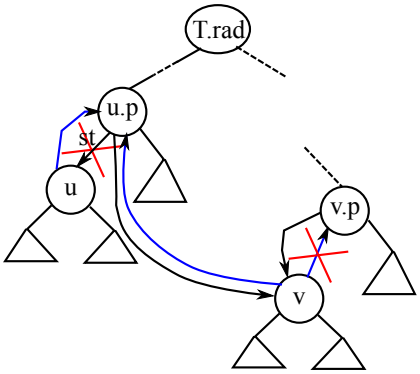

Figura 5: Managementul legăturilor între părintele nodului u şi nodul v.

În figura 6 este prezentat un exemplu de ştergere dintr-un arbore binar de căutare..

Observaţie: în multe documentaţii este prezentată o metodă mai simplă de ştergere a unui nod z cu doi fii nenuli dintr-un arbore biar de căutare. În loc să se înlocuiască nodul z cu succesorul său y si apoi să se şteargă z, se înlocuieşte informaţia lui z cu informaţia lui y şi apoi se şterge y, care are un singur fiu. Problema cu acest mod de ştergere este că, de fapt nu se va şterge pointerul care a fost reimis către funcţie, ci un alt pointer. Dacă alte componente al programului păstrează pointeri către y, acest lucru va avea consecinţe nefaste, aceşti pointeri fiind invalidaţi. În schimb metoda de mai sus, prezentată şi în Cormen, garantează faptul că pointerul şters este chiar cel trimis ca parametru către funcţia de ştegere

Complexitatea: tuturor operaţiilor descrise mai sus, începând de la căutarea binară, au complexitatea $O(h)$, unde h = înălţimea arborelui.

3 Exerciţii rezolvate

1. Care este numărul minim de chei într-un arbore binar (de căutare sau nu) cu înălţimea h?
   Soluţie: Numărul minim de chei se obţine atunci când pe fiecare nivel avem număr minim de noduri, adica unul singur. În acest caz numărul de noduri este $h + 1$.

2. Care este numărul maximde chei într-un arbore binar (de căutare sau nu) cu înălţimea h?
   Soluţie: Numărul maxim de chei se obţine atunci când pe fiecare nivel k avem număr maxm de noduri, adica unul 2^k noduri. În acest caz numărul de noduri este $2^{h+1} - 1$. Pentru detalii se poate vedea exerţitiul 1 de la problemele rezolvate la capitolul de heap-uri, pentru că în acest caz este acelaşi calcul.

3. Care este înălţimea minimă / maximă a unui arbore cu n noduri?
   Soluţie: înălțimea maximă se obţine pentru număr minim de noduri per nivel şi

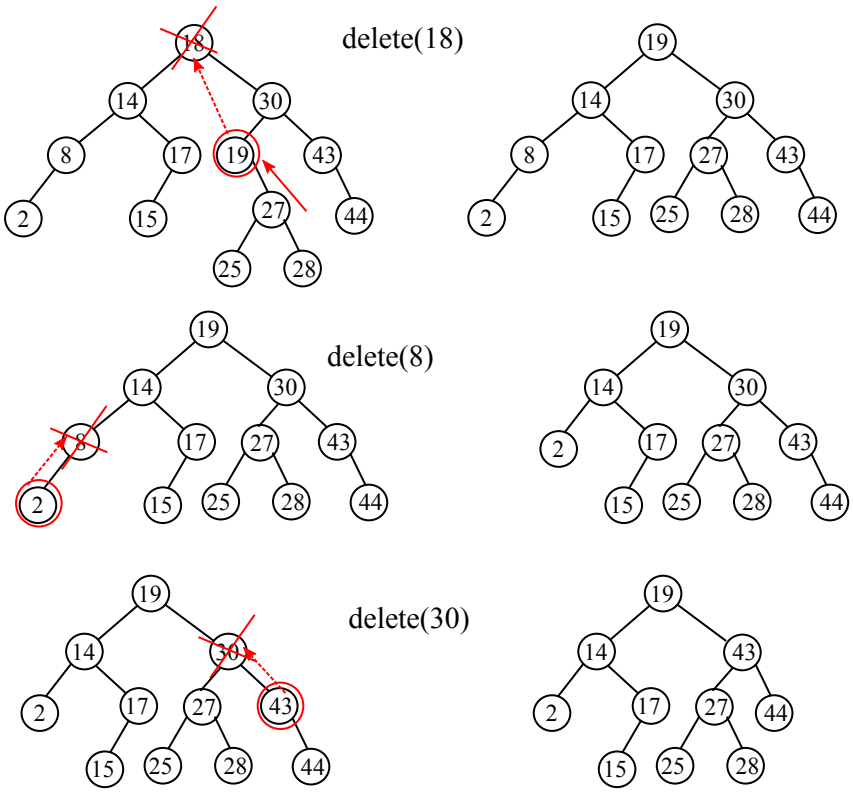

Figura 6: Se şterg pe rând cheile 18, 8, 30

este $n-1$ şi înălţimea minimă se obţine pentru număr maxim de noduri per nivel şi este $\log_2 n$.

4. Cunoscându-se parcurgerea în preordine a unui arbore binar de căutare, să se refacă arborele. Exemplu: RSD: 23, 17, 10, 15, 19, 35, 26, 24, 30, 37.
   Soluţie:

   var. 1 Observăm faptul că, prin parcurgerea în inordine a unui arbore binar de căutare se obţine şirul cheilor sortat crescător. Deci, cunoscând care sunt cheile arborelui, din parcurgerea în preordine (RSD), putem imediat obţine parcurgerea în inordine şi apoi putem aplica algoritmul discutat la tema3. (vezi documentaţie de pe elearning).

   var. 2 Dacă luăm pur şi simplu cheile în ordinea în care apar în parcurgerea RSD şi le inserăm într-un arbore binar de căutare iniţial vid, obţinem arborele cerut.

   Pentru cazul din exemplu, arborele este prezentat în fig. 7:

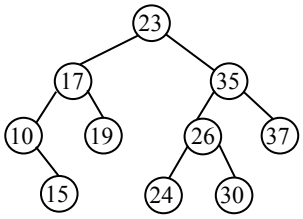

Figura 7: Arborele binar de căutare a cărui parcurgere în preordine este: 23, 17, 10, 15, 19, 35, 26, 24, 30, 37.