MODULUL 5 - ARBORI BIANRI 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 un arbore relativ dezechilibrat, producând 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.

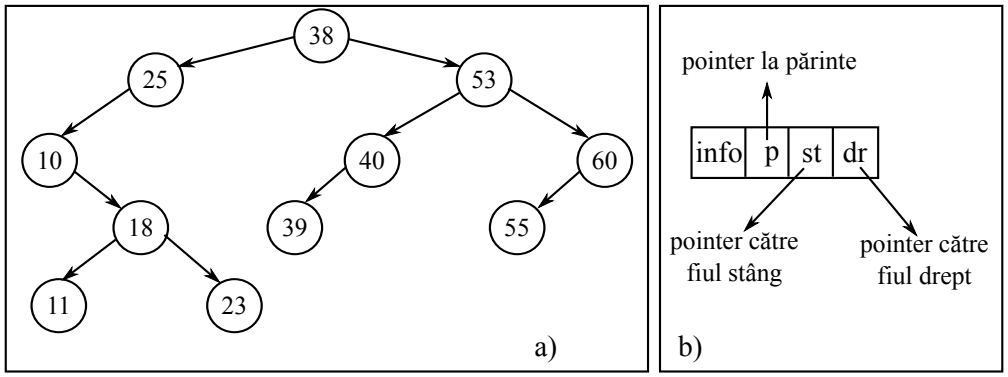

Figure 1: a) Exemplu de arbore binar de căutare; b) Structura unui nod din arbore.

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 unui nod.
- Determinarea minimului/maximului.
- Căutarea binară a succesorului / predecesorului
- Inserţie/ştergere a unui nod
- Sortarea cheilor arborelui, prin parcurgerea acestuia în inordine a arborelui.

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 cu rădăcina T, 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, altfel se continuă căutarea în subarborele stâng al lui x.

Algoritm:

```
AB_CAUT(T,k)
  x=T.rad
  cat timp x ≠ NULL si x.info ≠ k
           daca k < x.info atunci
                x = x.st
           altfel x = 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:

```
AB_MIN(x)
  y = x
  cat timp y.st ≠ NULL
           y = 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 $x.dr \neq NULL$
  - 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.

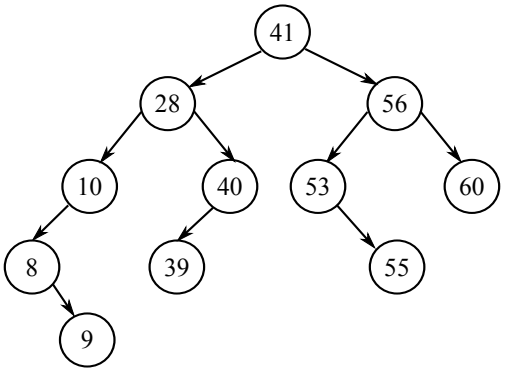

Figure 2: Arbore binar de căutare.

Exemplu: În arborele din figura 2 obţinem:

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

Algoritm

```
AB_SUCCESOR(x)
   daca x.dr ≠ NULL atunci
        y = AB_MIN(x.dr)
        RETURN y
   sfarsit daca
   y = x.p
   cat timp y ≠ NULL si x = y.dr
            x = y
            y = y.p
   sfarsit cat timp
 RETURN y
```

2.4 Inserarea unui nod

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

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

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 $NULL$.

Algoritm

```
AB_INSERT(T,z)
   y = NULL //pastreaza parintele nodului curent x
   x = T.rad
   cat timp x ≠ NULL
            y = x
            daca z.info < x.info atunci
                 x = x.st
            altfel x = x.dr
            sfarsit daca
```

```
  sfarsit cat timp
  z.p = y
  daca y = NULL atunci T.rad = z //inserarea radacinii
  alfel //verific pe care parte se face insertia
          daca z.info < y.info atunci
                y.st = z
          altfel y.dr = z
          sfarsit daca
  sfarsit daca
RETURN
```

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

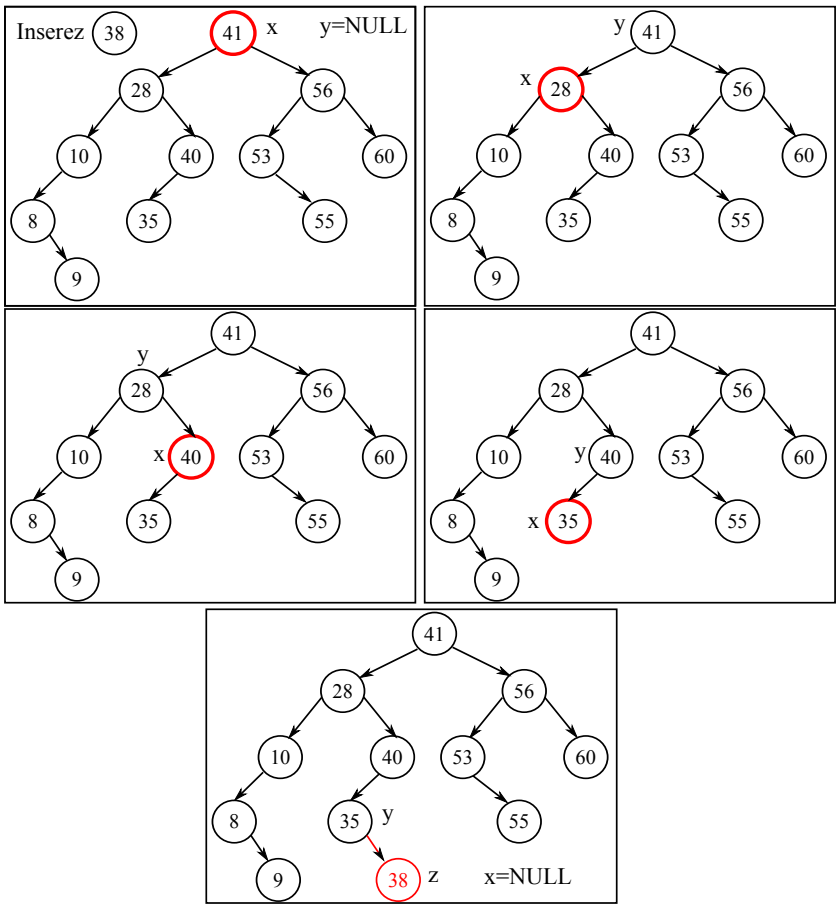

Figure 3: Inserarea nodului cu cheia 28 într-un arbore binar. Nodul curent cu care se compară nodul ce se inserează este marcat cu roşu.

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 NULL
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 NULL. 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 NULL
   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.

Algoritm

```
AB_TRANSPLANT(T,u,v)
    daca u.p = NULL atunci
         T.rad = v
    altfel
         daca u = u.p.st atunci
              u.p.st = v
         altfel
```

```
            u.p.dr = v
        sfarsit daca
    sfarsit daca
    daca v ≠ NULL atunci
        v.p = u.p
RETURN
```

Modul de funcţionare al funcţiei AB_TRANSPLANT este ilustrat în figura 4.

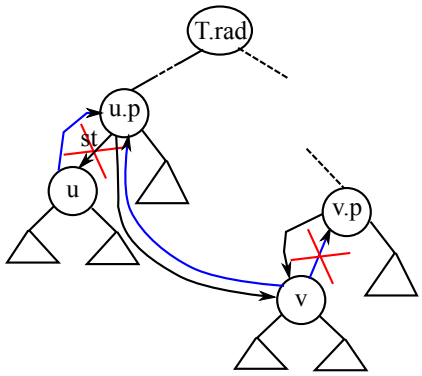

Figure 4: Managementul legăturilor între părintele nodului u şi nodul v.

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

```
AB_DELETE(T, z)
    daca z.st = NULL atunci
        AB_TRANSPLANT(T, z, z.dr) //înlocuieşte z cu z.dr
    altfel
        daca z.dr = NULL atunci
            AB_TRANSPLANT(T, z, z.st) //înlocuieşte z cu z.st
        altfel
            y = AB_SUCCESOR(z)
            daca y ≠ z.dr atunci
                AB_TRANSPLANT(T, y, y.dr)
                y.dr = z.dr //descendentul drept al lui z
                z.dr.p = y //devine descendent drept al lui y
            sfarsit daca
            AB_TRANSPLANT(T, z, y)
            y.st = z.st //descendentul stang al lui z
            z.st.p = y //devine descendent stang al lui y
        sfarsit daca
    sfarsit daca
RETURN
```

Cazurile luate în considerare de către funcţia AB_DELETE sunt ilustrate în figura 5.

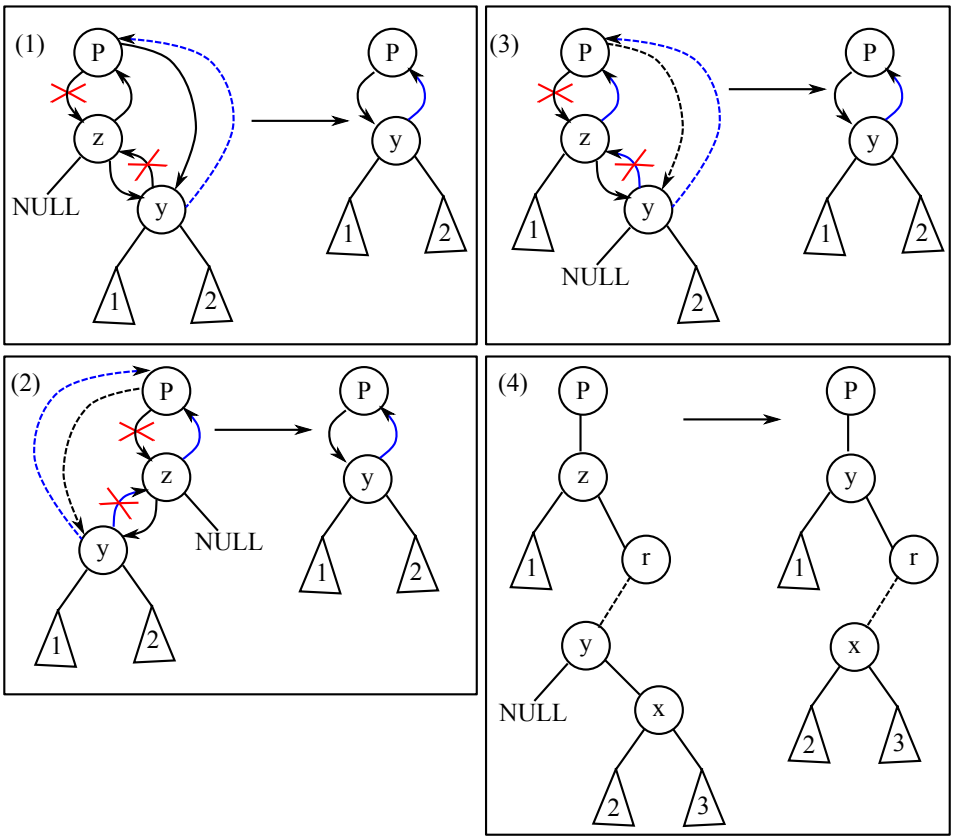

Figure 5: Ştergerea unui nod z care: (1) are fiul stâng nul; (2) are fiul drept nul; (3) are ambii fii nenuli, dar succesorul y este descendent direct al lui z; (4) are ambii fii nenuli, dar succesorul y nu este descendent direct al lui z.

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