Ioana Cristina Plajer

Structuri de date

Curs

- 2017 -

Cuprins

1 Arbori binari de căutare **5**

1.1 Noţiuni de bază 5

1.1.1 Căutarea binară 6

1.1.2 Minimul dintr-un arbore de căutare 7

1.1.3 Succesorul binar 7

1.1.4 Inserarea unui nod 9

1.1.5 Ştergerea unui nod 10

1.2 Arbori binari care se auto-balansează 14

1.2.1 Arbori AVL 15

1.3 Arbori Roşu-Negru 31

1.3.1 Inserţia într-un arbore roşu-negru 33

1.3.2 Ştergerea dintr-un arbore roşu-negru 38

2 Îmbogăţirea structurilor de date **49**

2.0.1 Arbori pentru statistici de ordine 50

2.0.2 Îmbogăţirea arborilor RN - teorema complexităţii . . 56

2.0.3 Arbori de intervale 57

Capitolul 1

Arbori binari de căutare

1.1 Noţiuni de bază

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

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

În cazul în care relaţia de ordine nu este strictă, dacă inserţia în cazul nodurilor cu chei egale se face pe aceeaşi parte a arborelui, în cazul multor noduri cu chei egale se obţine 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 1.1 a) este prezentat un exemplu de arbore binar de căutare.

Penru fiecare nod al arborelui se consideră structura din figura 1.1 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.

Operaţii de bază asupra unui arbore binar de căutare:

- Căutareă binară a unui nod.

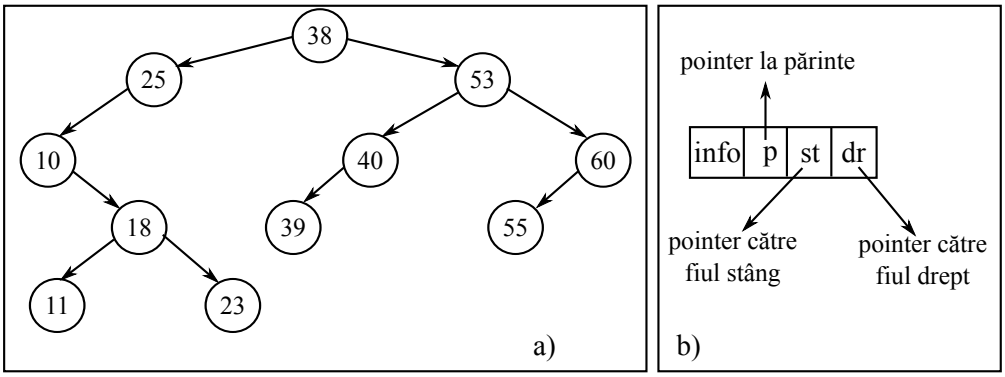


Figura 1.1: a) Exemplu de arbore binar de căutare; b) Structura unui nod din arbore.

- Determinarea minimului/maximului.
- Căutarea binară a succesorului
- Inserţie/ştergere a unui nod
- Sortarea unui vector care conţine cheile arborelui, prin parcurgerea 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 $O(log_2 n)$.

În secţiunile următoare vor fi prezentate două exemple de arbori binari de căutare care se autobalansează: aborii AVL şi arborii roşu-negru.

1.1.1 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.st
          sfarsit daca
  sfarsit cat timp
RETURN x
```

1.1.2 Minimul dintr-un arbore de căutare

Nodul cu informaţia minimă dintr-un arbore binar de căutare poate fi găsit parcurgând descendenţii stângi până la cea mai din stânga frunză. Funcţia AB_MIN returnează nodul cu informaţia minimă.

Algoritm:

```
AB_MIN(T)
  x = T.rad
  cat timp x.st ≠ NULL
           x = x.st
  sfarsit cat timp
RETURN x
```

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

1.1.3 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 comaparaţii
- 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, altfel.

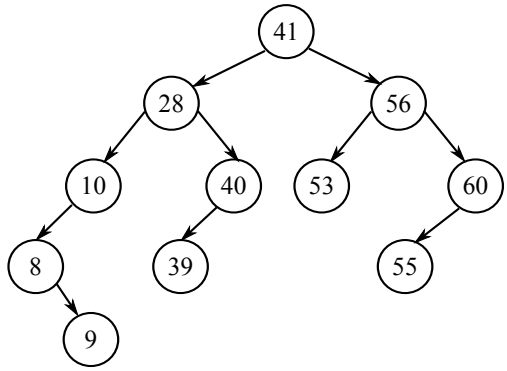


Figura 1.2: Arbore binar de căutare.

Exemplu: În arborele din figura 1.2 obţinem:

AB_SUCCESOR(41) = 53
AB_SUCCESOR(39) = 41
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
```

1.1.4 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 1.3 este ilustrat algoritmul de inserţie a unui nod cu cheia 38 într-un arbore binar de căutare.

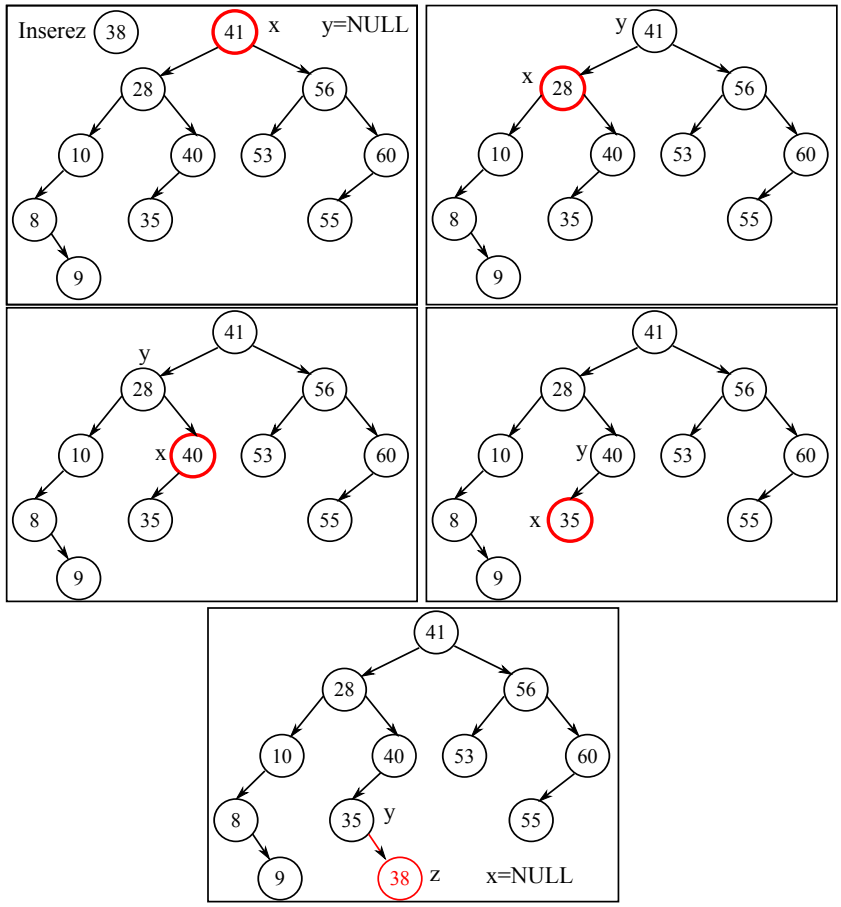


Figura 1.3: Inserarea nodului cu cheia 28 într-un arbore binar. Nodul curent cu care se compară nodul ce se inserează este marcat cu roşu.

1.1.5 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 interschimbă nodul z cu nodul y şi se şterge y din subarbore - adică se revine la cazul 2 de ştergere.

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. Funcţia AB_STERGE are ca argumente arborele T şi z=pointer către nodul care va fi şters. 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 [1] 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 fii 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 1.4.

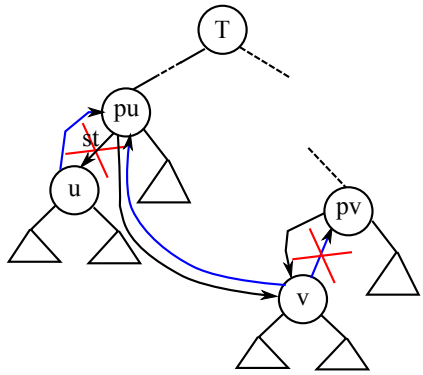


Figura 1.4: Managementul legăturilor între părintele nodului u şi nodul v.

În continuare este prezentat algoritmul de ştergere a unui nod z dintr-un arbore binar de căutare T, conform [1].

```
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.dr
        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 1.5.

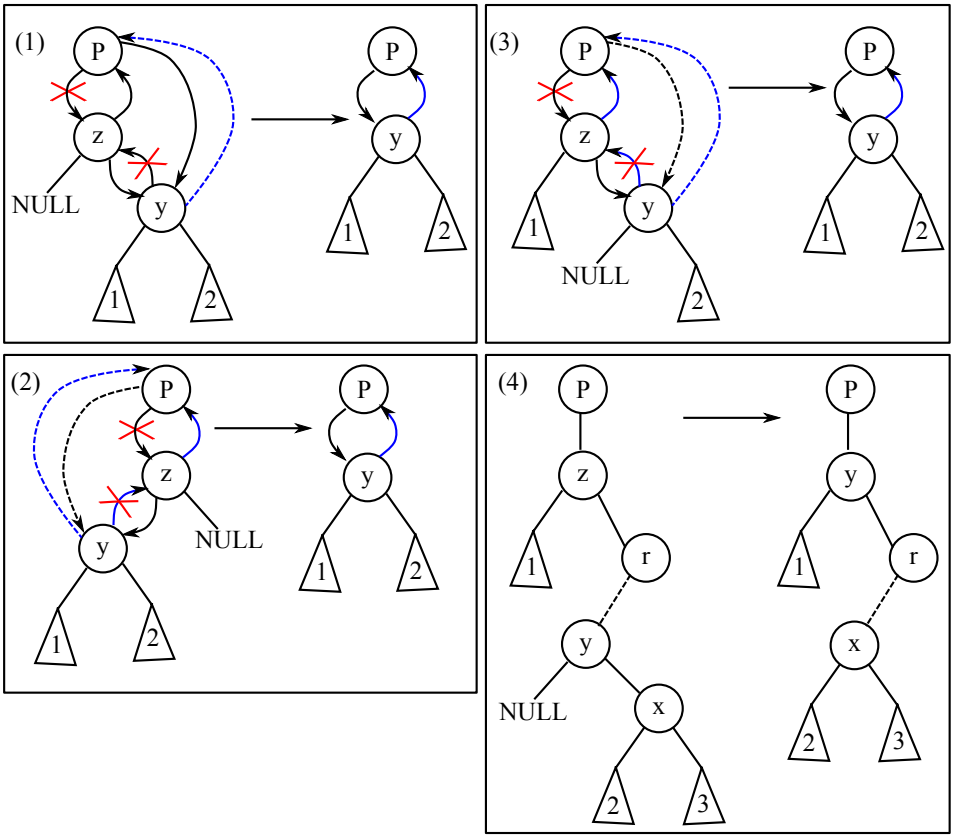


Figura 1.5: Ştergerea unui nod z care: (1) are fiul stâng nul; (2) are fiul drept nul; (3) are ambii fii nenuli, dar succesorul y este descendent direct al lui z; (4) are ambii fii nenuli, dar succesorul y nu este descendent direct al lui z.

***Observaţie*:** Se poate simplifica algoritmul, dacă în loc să se înlocuiască nodul z cu nodul y, se preia în z doar informaţia din y, iar apoi se şterge nodul y. Acest lucru se pretează, acolo unde nodurile nu conţin decât un câmp de informa ctie, nu şi alte câmpuri, care altfel ar trebui tratate separat.

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

Construcţia unui arbore binar de căutare prin inserţii succesive

Cel mai defavoriabil caz: elementele sunt sortate în ordine crescătoare sau descrescătoare şi sunt inserate succesiv.

Arbori binari de căutare construiţi aleator: se demonstrează faptul că pentru arborii binari de căutare construiţi prin inseţia aleatoare a elementelor, înălţimea este $O(log_2 n)$ [1].

Arbori binari de căutare cu chei egale

Algoritmii prezentaţi mai sus porneau de la presupunerea că arborii nu conţin chei egale.

Se pune problema inserării cheilor egale în arbore, astfel încât, dacă există multe chei egale, să nu ducă la creşterea semnificativă a înălţimii arborelui pe una dintre ramuri. Există mai multe moduri de rezolvare a problemei.

Se consideră z nodul care se inserează şi x nodul curent cu care se compară z.

1. Pentru fiecare nod x se utilizează un câmp ce conţine o variabliă booleană care ia alternativ valorile *true*/*false* atunci când se detectează $z.info = x.info$. În funcţie de valoarea câmpului boolean al lui x se continuă inserţia pe stânga sau pe dreapta lui x.

2. În fiecare nod se păstrează o listă a nodurilor cu chei egale, iar dacă $x.info = z.info$ se inserează z în lista lui x.

3. Dacă $x.info = z.info$ se inserează z în mod aleator pe stânga sau pe dreapta lui x.

1.2 Arbori binari care se auto-balansează

Ideea creării arborilor binari de căutare a fost aceea de a permite operaţii eficiente într-un set de date după modelul căutarii binare, în complexitate logaritmică. Aşa cum s-a precizat mai sus, complexitatea operaţiilor pe

un arbore binari de căutare depinde de înălţimea acestuia, deci sunt $O(h)$. Pentru un arbore binar de căutare obişnuit nu există nici o garanţie, că această înălţime este de ordin logaritmic. Pentru a obţine complexitate logaritimică, un arbore binar de căutare trebuie să fie cât mai echilibrat. În scopul acesta au fost dezvoltaţi o serie de arbori care sa auto-echilibrează. Dintre aceştia vom discuta în cele ce urmează arborii AVL şi arborii roşu-negru.

1.2.1 Arbori AVL

Arborii AVL [2] [3] [4] sunt arbori binari de căutare aproape balansaţi. Denumirea provine de la autorii acestor arbori, doi matematicieni ruşi G.M. Adelson-Velsky şi E.M. Landis. Considerăm pentru fiecare nod x un *factor de balansare* dat prin

$$x.fb = h(x.dr) - h(x.st)$$

adică, factorul de balansare al unui nod x este reprezentat de către diferenţa dintre înălţimea subarborelui său drept şi înălţimea subarborelui său stâng. În unele documentaţii diferenţa se realizează între înălţimea subarborelui stâng şi cea a subarborelui drept.

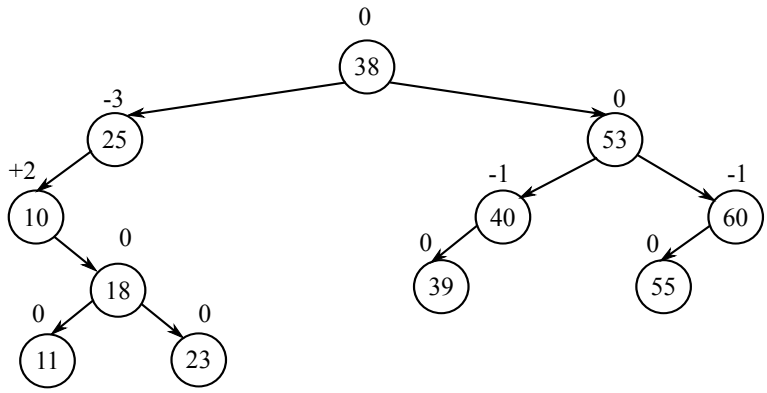


Figura 1.6: Arbore binar de căutare împreună cu factorii de balansare asociaţi nodurilor.

Definiţie - nod balansat: Un nod x se numeşte balansat, dacă $x.fb \in \{-1, 0, 1\}$.

În figura 1.6 este prezentat un exemplu de arbore binar de căutare împreună cu factorii de balansare corespunzători fiecărui nod. Se observă faptul că nodurile cu cheile 10 şi 25 nu sunt balansate.

Definiţie - arbore AVL: Un arbore binar de căutare se numeşte *arbore AVL*, dacă fiecare nod al său este balansat.

Un exemplu de arbore AVL este prezentat în figura 1.7.

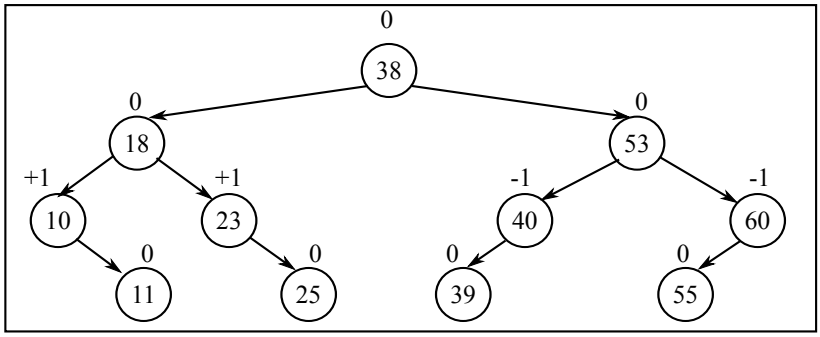


Figura 1.7: Arbore AVL.

Înălţimea unui arbore AVL

Pentru a determina înălţimea maximă a unui arbore AVL care conţine n noduri se porneşte de la următoarea idee. Se notează cu $N(h)$ numărul minim de noduri interne ale unui arbore AVL de înălţime h. Atunci, datorită faptului că factorul de balansare al rădăcinii este minim -1 şi maxim 1, rezultă că pentru un număr minim de noduri interne, unul dintre subarbori are înălţimea $h-1$ iar celălalt are înălţimea $h-2$.

$$N(h) = N(h-1) + N(h-2) + 1.$$

Cazurile de bază:

$$N(0) = 1$$
$$N(1) = 2$$

Evident: $N(h-1) > N(h-2)$, de unde rezultă $N(h) > 2N(h-2) > 4N(h-4) > \ldots > 2^i N(h-2i)$.
Ajungem la un caz elementar pentru $h-2i = 1$ pentru h par - sau $h-2i = 2$ pentru h impar. De aici rezultă:

$N(h) > 2^{(h+1)/2}$ - h par - sau $N(h) > 2^{h/2}$ - h impar.
Deci $h < 2log_2(N(h))$.

Deoarce complexitatea operaţiilor de căutare, inserţie şi ştergere dintr-un arbore binar de căutare este $O(h)$ rezultă că într-un arbore AVL complexitatea acestor operaţii este $O(log_2 n)$, dacă nu se ţine cont de operaţiile de reechilibrare ale arborelui.

Prin operaţii de inserţie/ştergere se poate produce o debalansare a anumitor noduri. Pentru refacerea proprietăţii de arbore AVL este necesară o schimbare a structurii de pointeri. Schimbarea structurii de pointeri se realizează prin operaţii de **rotaţie**.

Rotaţia

Este o operaţie locală care schimbă structura de pointeri într-un arbore binar, dar păstrează proprietăţile acestuia.

Tipuri de rotaţie:

- Rd = rotaţie spre dreapta în jurul nodului x
- Rs = rotaţie spre stânga în jurul nodului y

Operaţia de rotaţie este ilustrată în figura 1.8.

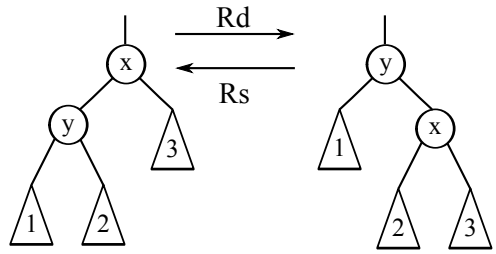


Figura 1.8: Rotaţii într-un arbore binar de căutare.

Observaţii:

- pentru a putea efectua o rotaţie spre drapta în jurul nodului x, este necesar ca nodul x să aibă un descendent stâng diferit de NULL.

- pentru a putea efectua o rotaţie spre stânga în jurul nodului y, este necesar ca nodul y să aibă un descendent drept diferit de NULL.

Algoritm pentru rotaţia spre stânga în jurul nodului y

```
ROT_ST(T,y)
    x = y.dr
    y.dr = x.st //subarb 2 din imag se mută de la x la y
    daca x.st ≠ NULL atunci
         x.st.p = y//fac legatura subarb.  2 la noul părinte y
    sfarsit daca
    x.p = y.p
    daca y.p = NULL atunci
         T.rad = x
    altfel
         daca y = y.p.st atunci //fac legatura de la parintele
                                //lui y la x
              y.p.st = x
         altfel y.p.dr = x
         sfarsit daca
    sfarsit daca
    x.st = y //fac legatura de la x la y
    y.p = x //fac legatura de y la x
 RETURN
```

Rotaţia înspre dreapta este simetrică.

Complexitate: $O(1)$

Inserţia

Prin inserarea unui nod nou se poate produce o debalansare în arbore. Acest lucru înseamnă că, cel puţin un nod din arbore va avea, după recalcularea factorilor de balansare un factor -2 sau 2.

Rebalansarea se realizează în modul următor: Nodul nou inserat are factorul de balansare este 0. Se porneşte de la nodul inserat înspre rădăcină. Pentru fiecare nod pe acest drum se recalculează factorul de balansare.

Notăm cu x nodul curent. Pornind de la acest nod se calculează factorul de balansare al părintelui $x.p$ în modul următor:

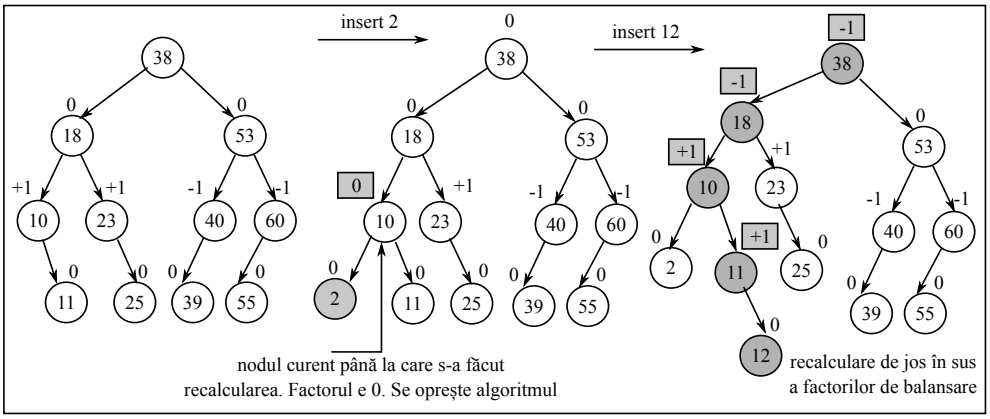


Figura 1.9: Exemplu de recalculare a factorilor de balansare de la nodul inserat în sus.

- Dacă x este chiar nodul inserat, atunci,
  - dacă x a fost inserat la stânga lui $x.p$, a crescut înălţimea pe stânga lui $x.p$, deci factorul de balansare a lui $x.p$, care este $h(x.p.dr) - h(x.p.st)$ scade cu 1.
  - dacă x a fost inserat la dreapta lui $x.p$, a crescut înălţimea pe dreapta lui $x.p$, deci factorul de balansare a lui $x.p$, care este $h(x.p.dr) - h(x.p.st)$ creşte cu 1.

- Dacă x este nodul curent şi factorul său de balansare recalculat este 0, atunci nu a crescut înălţimea subarborelui de rădăcină x, faţă de cât era înainte de inserţie, doar s-a echilibrat. Din acest motiv, nu mai are sens continuarea urcării în arbore. Dat fiind că nu s-a modificat înălţimea subarborelui, nu se vor mai modifica nici factorii de balansare mai sus şi algoritmul de rebalansare se poate opri.

- Dacă x este nodul curent şi factorul său de balansare recalculat este -1 sau +1, atunci s-a produs o crecştere a înălţimii subarborelui de rădăcină x pe una dintre ramuri (stângă respectiv dreaptă). Acest lucru produce o modificare a factorului de balansare al părintelui $x.p$ şi anume dacă x este pe stânga lui $x.p$, atunci scade factorul lui $x.p$ cu 1, dacă x este pe dreapta lui $x.p$, atunci creşte factorul lui $x.p$ cu 1. Acest lucru poate produce modificări mai sus şi se continuă cu procesul de reechilibrare de la $x.p$ ca fiind noul nod curent.

Un exemplu de astfel de recalculare este prezentat în figura 1.9.

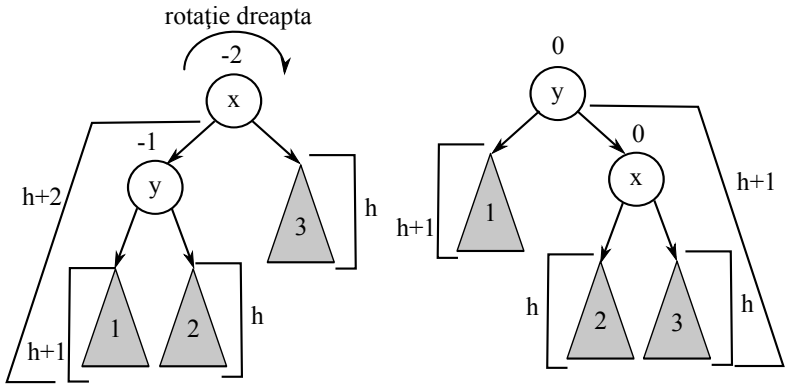


Figura 1.10: Rebalansarea arbore AVL în cazul 1 a).

Dacă la momentul curent factorul de balansare al nodului curent x este -2 sau 2, atunci trebuie rebalansat arborele.

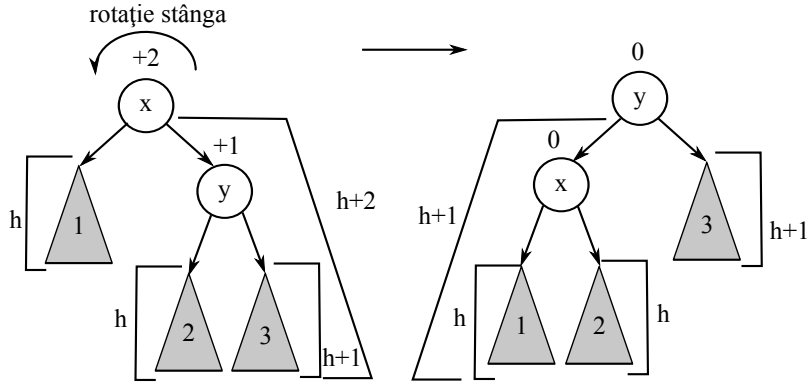


Figura 1.11: Rebalansarea arbore AVL în cazul 1 b).

Cazul 1

a) $x.fb = -2$. Deoarece debalansarea s-a făcut prin inserţia unui nod într-un arbore AVL valid rezultă că inserţia s-a făcut la stânga lui x. Se verifică factorul de balansare al lui $y = x.st$. Dacă $y.fb = -1$

atunci: se efectuează o rotaţie spre dreapta în jurul lui x. Rezultatul acestei operaţii este ilustrat în figura 1.10. Se observă faptul că noii factori de balansare sunt 0 atât pentru x cât şi pentru y. În plus, înainte de inserţie, subarborele stâng al nodului x avea înălţimea $h+1$, iar cel drept h. După rebalansare ambii subarbori au înălţimea $h+1$. Rezultă că înălţimea subarborelui care acum are rădăcina y nu a crescut, faţă de înălţimea înainte de inserţie, când rădăcina era x. Astfel, nu este necesar să continuăm urcarea în arbore, deoarece mai sus nu vor exista modificări în factorul de balansare.

b) $x.fb = 2$. Deoarece debalansarea s-a făcut prin inserţia unui nod într-un arbore AVL valid rezultă că inserţia s-a făcut la dreapta lui x. Se verifică factorul de balansare al lui $y = x.dr$. Dacă $y.fb = 1$ atunci: se efectuează o rotaţie la stânga în jurul lui x. Rezultatul acestei operaţii este ilustrat în figura 1.11. Observaţii similare cazului 1.a pot fi făcute şi în acest caz.

Cazul 2:

a) $x.fb = -2$. Deoarece debalansarea s-a făcut prin inserţia unui nod într-un arbore AVL valid rezultă că inserţia s-a făcut la stânga lui x. Se verifică factorul de balansare al lui $y = x.st$. Presupunem că $y.fb = 1$. Ce se întâmplă dacă se efectuează o rotaţie spre dreapta în jurul lui x? Rezultatul unei astfel de rotaţii este ilustrat în figura 1.12.

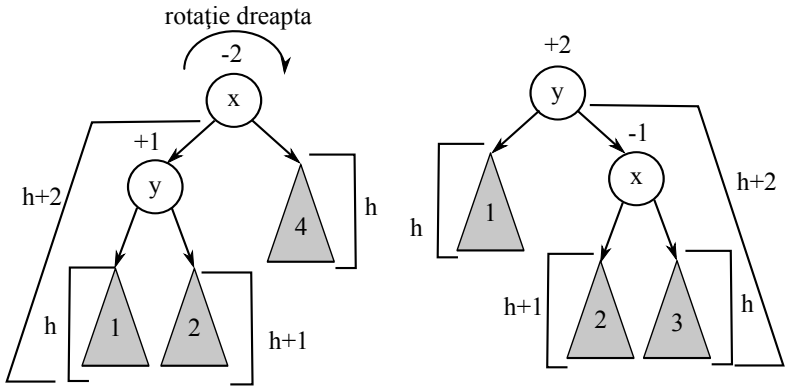


Figura 1.12: în cazul 2, o rotaţie simplă nu rezolvă problema debalansării arborelui

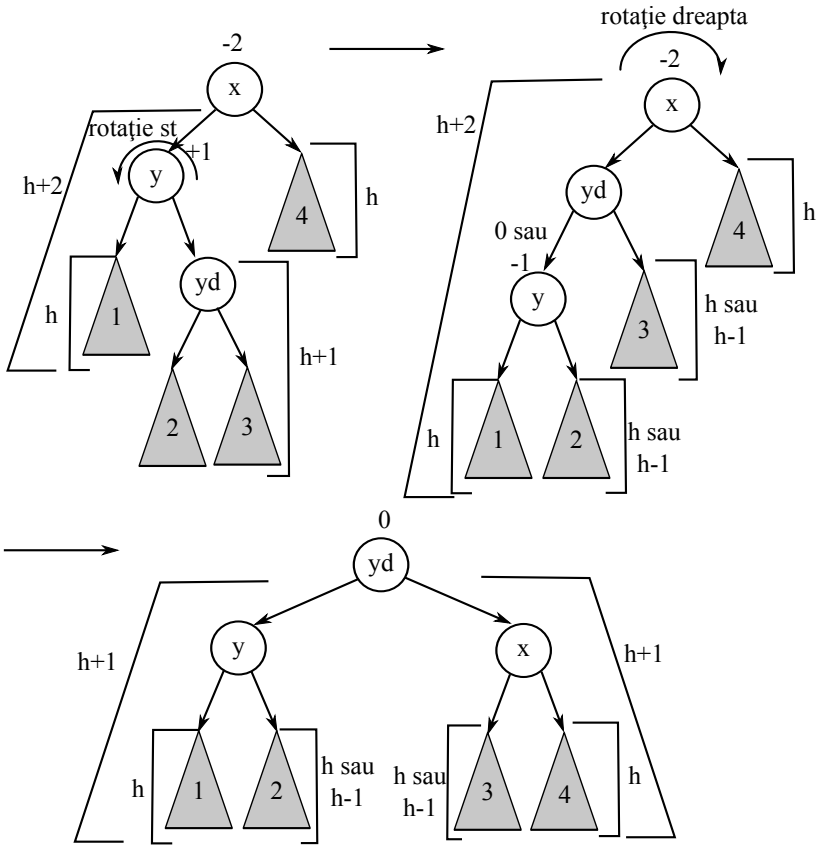


Figura 1.13: Rebalansarea arborelui în cazul 2.

Deci nu se rezolvă debalansarea, ci se mută pe cealaltă parte a arborelui. Soluţia este următoarea:

- întâi rotaţie la stânga în jurul lui y
- apoi rotaţie la dreapta în jurul lui x.

Se obţine rezultatul din fig. 1.13.

b) $x.fb = 2$. Deoarece debalansarea s-a făcut prin inserţia unui nod într-un arbore AVL valid rezultă că inserţia s-a făcut la dreapta lui x. Se verifică factorul de balansare al lui $y = x.dr$. Presupunem că $y.fb = 1$. Dacă efectuăm o rotaţie la dreapta în jurul lui x se obţine un efect similar ca la punctul a. Soluţia este deci:

– întâi rotaţie la dreapta în jurul lui y

– apoi rotaţie la stânga în jurul lui x

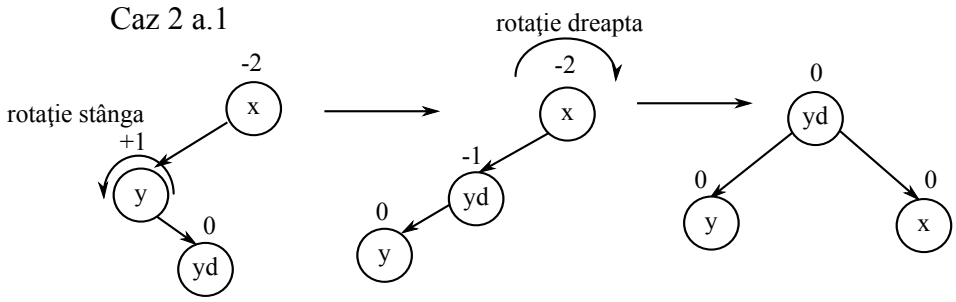


Figura 1.14: Recalcularea factorilor de balansare pentru x, y, yd, atunci când yd a avut factorul 0. Acest caz nu poate apărea decât atunci când nodul inserat a fost yd, y a fost o frunză, iar nodul curent x a avut doar fiul y.

Observaţii:

- Pentru $x.fb = -2 \Rightarrow$ inserţia s-a făcut pe stânga şi în mod sigur $x.st.fb$ nu poate să fie 0, în caz contrar nu s-ar fi debalansat x.
- Pentru $x.fb = 2 \Rightarrow$ inserţia s-a făcut pe dreapta şi în mod sigur $x.dr.fb$ nu poate să fie 0 , în caz contrar nu s-ar fi debalansat x.
- după rebalansare nu trebuie continuată urcarea în arbore, deoarece, din aceleaşi considerente ca la cazul I, înălţimea subarborelui din figură nu se modifică faţă de înălţimea înainte de inserţie, şi deci nu pot apărea debalansări mai sus în arbore.

Să studiem însă ce se întâmplă cu factorii de balansare ai nodurilor x, y şi yd, care trebuie recalculaţi. Acest lucru este ilustrat în figurile 1.14şi 1.15 pentru cazul 2.a). Acest caz este împărţit în 2 subcazuri, în funcţie de factorul de balansare al nodului yd. Cazul 2 b) se rezolvă în mod simetric.

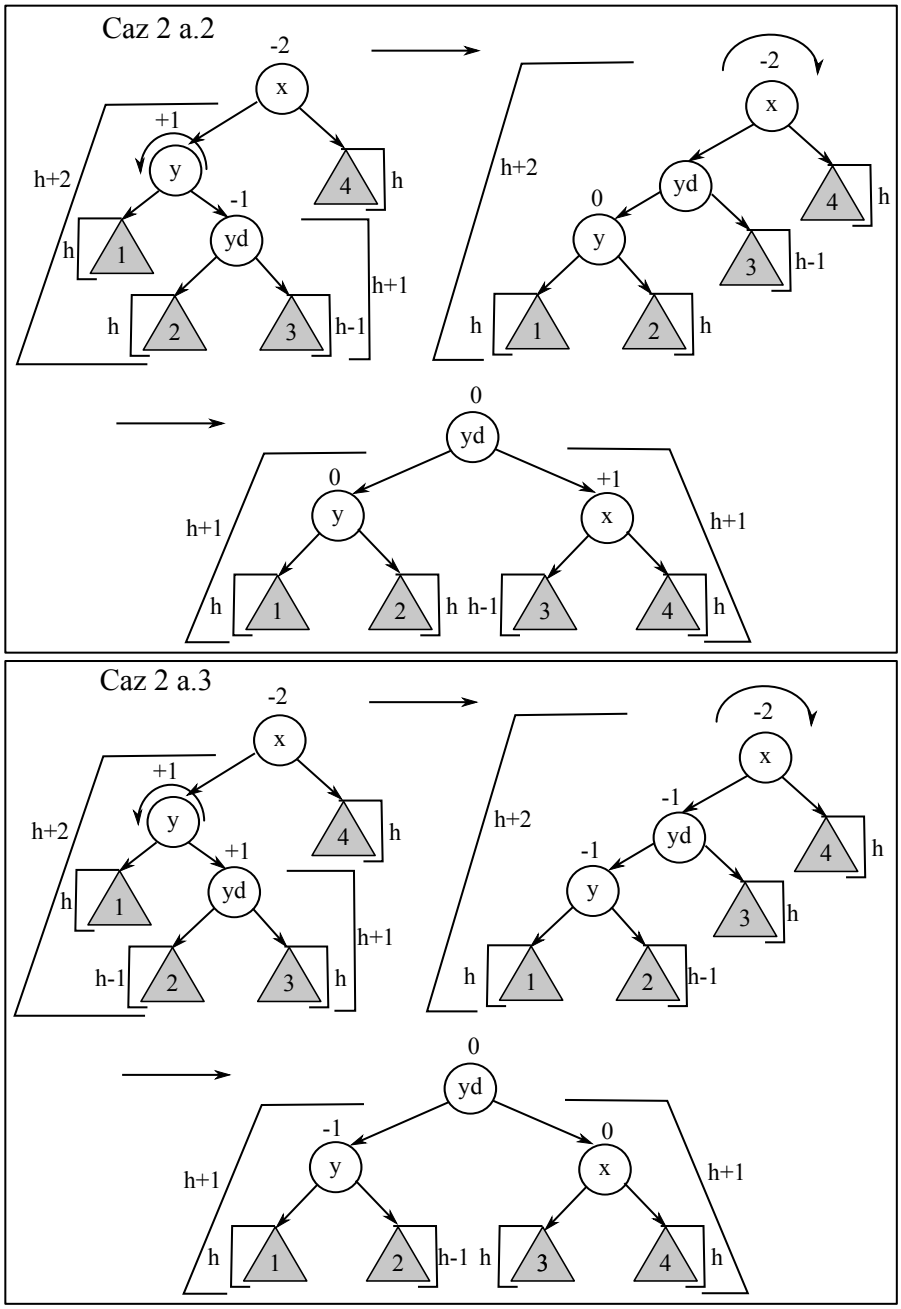


Figura 1.15: Recalcularea factorilor de balansare pentru x, y, yd, atunci când yd a avut factorul -1, respectiv $+1$

Un exemplu de inserţie într-un arbore AVL este prezentat în figura 1.16.

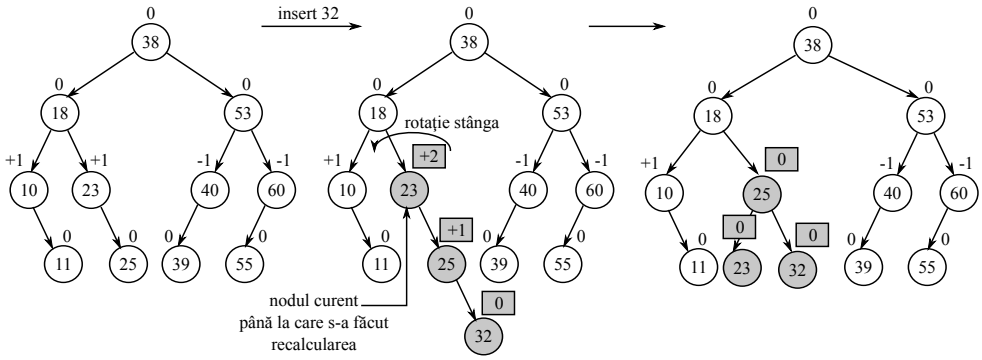


Figura 1.16: Inserţia nodului cu cheia 32 se inserează în arborele AVL din figură.

Ştergerea unui nod

Ştergerea unui nod se realizează la fel ca pentru arborii binari de căutare obişnuiţi. Dacă notăm cu z nodul care trebuie şters atunci:

- dacă z are cel mult un descendent, se porneşte rebalansarea de la părintele lui z
- dacă z are doi descendenţi, atunci se determină succesorul lui y, se înlocuieşte nodul y cu descendentul său drept x, iar nodul z cu nodul y.

Se realizează rebalansarea începând de la părintelel lui x. Dacă s-a şters o frunză, atunci, dacă ştergerea a avut loc la stânga va creşte factorul de balansare a al părinterlui nodului şters, dacă ştergerea a avut loc la dreapta, scade factorul de balansare al părintelui nodului şters.

Spre deosebire de inserţie, algoritmul nu se opreşte atunci când factorul de balansare recalculat al nodului curent x este 0, deoarece în acest caz a scăzut înălţimea subarborelui de rădăcină x. Acest lucru, ilustrat în figura 1.17, produce modificări ale factorului de balansare şi la părintele său în modul următor: dacă x se află la stânga părintelui său, factorul acestui părinte va creşte cu 1, altfel va scădea cu 1 şi deci continuă urcarea în arbore.

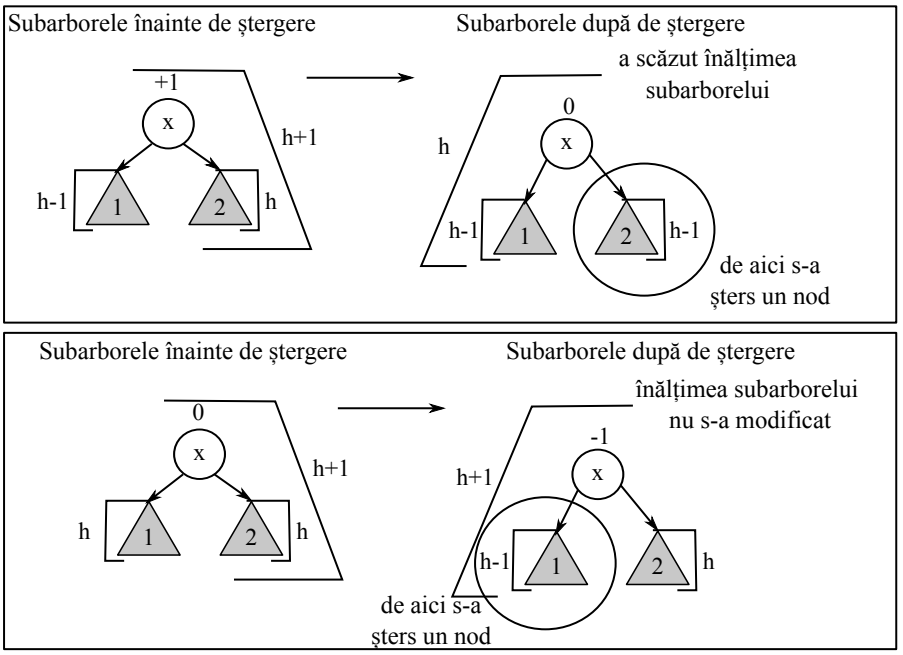


Figura 1.17: Modificarea înălţimii subarborelui curent după şteregerea unui nod.

În schimb, atunci când factorul de balansare al nodului x devine -1 sau $+1$, înseamnă că doar s-a produs o scădere a înălţimii pe una dintre ramurile lui x, dar nu şi a subarborelui de rădăcină x (fig. 1.17), deci algoritmul de rebalansare se poate opri.

De asemenea, algoritmul nu se opreşte însă după prima rebalansare a unui nod cu factorul -2 sau $+2$, ci în cele mai multe cazuri trebuie să continue de la nodul curent la părinte până la rădăcină.

Considerând nodul curent x, rebalansarea are următoarele cazuri:

Cazul 1

a. Dacă x are factorul de balansare nou -2 şi factorul lui $y = x.st$ este -1 sau 0, atunci rotaţie la dreapta în jurul lui x. În figura 1.18 este ilustrat modul de recalculare a factorilor de balansare pentru nodurile x şi y. Se observă şi următorul fapt important. Subarborele de rădăcină x a avut înainte de ştergere înălţimea $h+2$. În cazul în 1.a.1, în care y a avut factorul de balansare 0 înainte de rebalansare, după rebalansare se observă că nu s-a modificat înălţimea subarborelui, care acum are

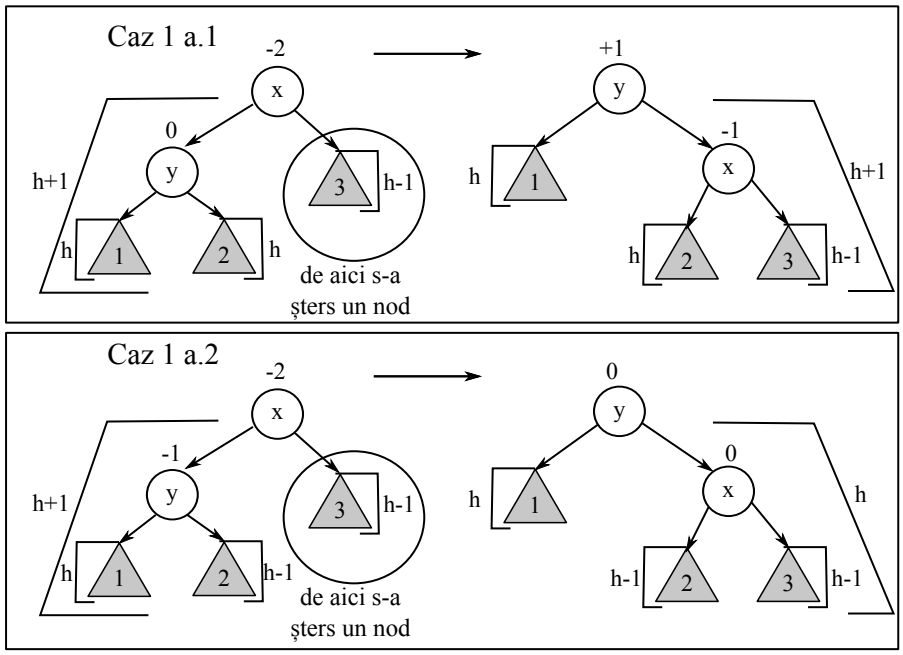


Figura 1.18: Rebalansarea unui arborele AVL după ştergerea unui nod, cazul 1 a.

rădăcina y, şi deci nu mai este necesară continuarea urcării în arbore.

În schimb, daca y a avut factorul de balansare -1, cazul 1.a.2, se observă din figură, că după rebalansare, înălţimea subarborelui care acum are rădăcina y, a scăzut cu 1, ceea ce produce eventuale debalansări mai sus în arbore, deci trebuie continuat la părintele lui x.

b. Dacă x are factorul de balansare nou 2 şi factorul de balansare al lui $y = x.dr$ este 0 sau 1 atunci rotaţie la stânga în jurul lui x. În figura 1.19 este ilustrat modul de recalculare a factorilor de balansare pentru nodurile x şi y. La fel ca în cazul 1. a, se observă şi următoarele: Subarborele de rădăcină x a avut înainte de ştergere înălţimea $h + 2$. În cazul în 1.b.1, în care y a avut factorul de balansare 0 înainte de rebalansare, după rebalansare se observă că nu s-a modificat înălţimea subarborelui, care acum are rădăcina y, şi deci nu mai este necesară continuarea urcării în arbore.

În schimb, daca y a avut factorul de balansare $+1$, cazul 1.b.2, se observă din figură, că după rebalansare, înălţimea subarborelui care

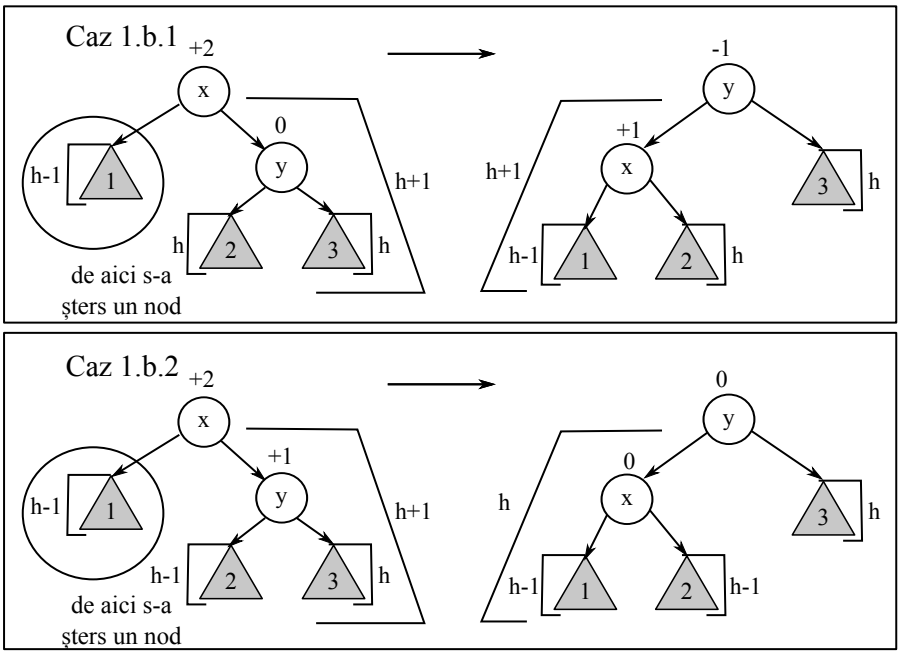


Figura 1.19: Rebalansarea unui arborele AVL după ştergerea unui nod, cazul 1 b.

acum are rădăcina y, a scăzut cu 1, ceea ce produce eventuale debalansări mai sus în arbore, deci trebuie continuat la părintele lui x.

Cazul 2

a. Dacă x are factorul de balansare nou -2 şi nodul $y = x.st$ are factorul de balansare 1, atunci pentru rebalansare se efectuează:
   - Întâi rotaţie la stânga în jurul lui y
   - Apoi rotaţie la dreapta în jurul lui x.

b. Dacă x are factorul de balansare nou 2 şi $y = x.dr$ are factorul de balansare -1, atunci pentru rebalansare se efectuează:
   - Întâi rotaţie la dreapta în jurul lui y
   - Apoi rotaţie la stânga în jurul lui x.

Modul de calcul al factorilor de balansare pentru x, y şi $y.dr$ în cazul 2.a este prezentat în figurile 1.20 şi 1.21. Tot din aceste figuri se observă faptul că, înălţimea subarborelui care are ca rădăcină nodul curent, a scăzut

cu o unitate, ceea ce face necesară continuarea rebalansării nodului de la părintele acestei rădăcini. În cazul 2.b modul de calcul este analog, simetric.

Complexitate: rebalansarea se realizează pentru maxim h noduri, unde h este înălţimea arborelui, deci complexitatea este $O(log_2 n)$.

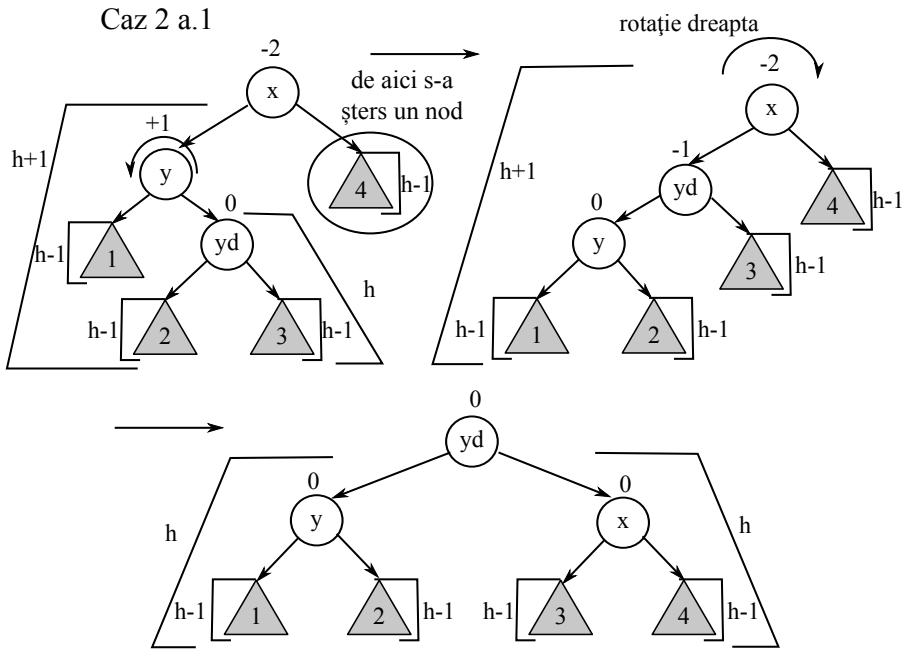


Figura 1.20: Rebalansarea unui arborele AVL după ştergerea unui nod, cazul 2 a1, în care $y.dr$ are factorul de balansare 0.

Un exemplu de ştergere a unui nod dintr-un arbore AVL este prezentat în figura. **??**.

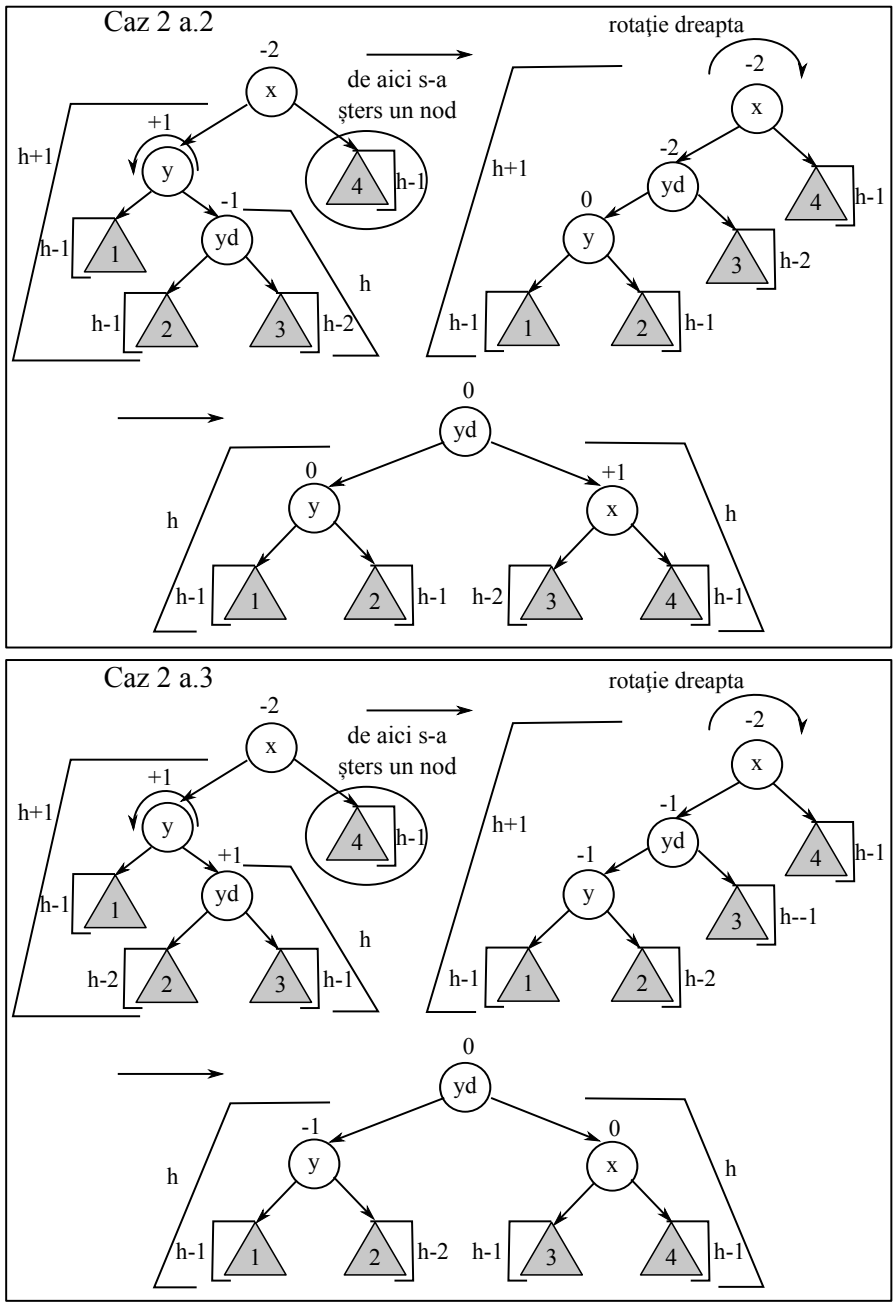


Figura 1.21: Rebalansarea unui arborele AVL după ştergerea unui nod, cazurl 2 a1, în care $y.dr$ are factorul de balansare -1, reaspectiv +1.

1.3 Arbori Roşu-Negru

Definiţie: Un arbore roşu-negru (ARN) este un arbore binar de căutare în care fiecărui nod i se asociază o culoare - roşu sau negru - şi care are următoarele proprietăţi:

1. Fiecare nod este roşu sau negru -are deci un câmp suplimentar *color*
2. Rădăcina este neagră
3. Fiecare frunză este neagră şi NULL
4. Dacă un nod este roşu, ambii fii sunt negri ⇒ părintele unui nod roşu este negru.
5. Pentru fiecare nod, oricare drum de la nod la o frunză are acelaşi număr de noduri negre (inclusiv frunza null şi exclusiv nodul de la care se porneşte) ⇒ pe fiecare drum de la rădăcina r la o frunză se găsesc acelaşi număr de noduri negre = înălţimea neagră a arborelui = *bh - black height*.

Un exemplu de arbore roşu-negru este prezentat în figura 1.22.

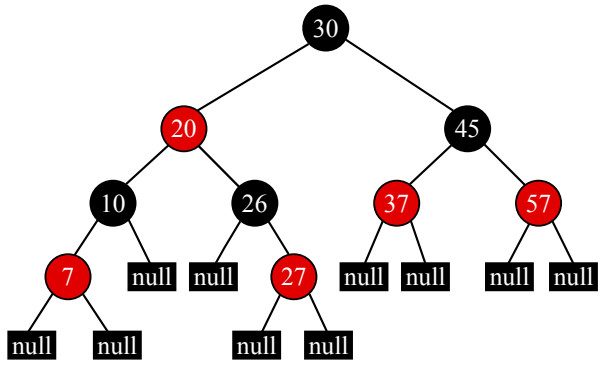


Figura 1.22: Exemplu de arbore roşu - negru.

Observaţii:

- Într-un ARN nici un drum de la rădăcină la o frunză nu poate fi mai lung decât dublul unui alt drum la altă rădăcină. Acest lucru asigură o oarecare balansare a arborelui .

- Înălţimea maximă a unui arbore ARN este $2\log_2(n+1)$.

  Se demonstrează prin inducţie că orice subarbore de rădăcină x conţine cel puţin $2^{bh(x)} - 1$ noduri interne. Notând cu h înălţimea arborelui, din proprietatea 4 se obţine $bh(r) \geq h/2$. Dar $n \geq 2^{bh(r)} - 1 \geq 2^{h/2} - 1$ de unde rezultă $h \leq 2\log_2(n+1)$

- Definirea unui nod NULL pentru fiecare frunză presupune un consum inutil de memorie. Din acest motiv se poate considera în locul acestor frunze un singur nod santinelă T.nil către care să pointeze acele noduri interne care au ca descendenţi frunze NULL. Un exemplu de ARN cu santinelă este prezentat în fig. 1.23.

- Pentru simplitate în continuare vom ignora în desene nodurile NULL.

- Datorită faptului că operaţiile de căutare, maxim, minim, succesor, predecesor depind de înălţimea h a arobrelui, înseamnă că aceste operaţii au complexitatea $O(\log_2 n)$.

- Operaţiile de inserţie şi ştergere sunt ceva mai complicate decât în cazul arborilor binari de căutare simpli, deoarece după inserţie/ştergere trebuie eventual refăcută structura de arbore roşu-negru.

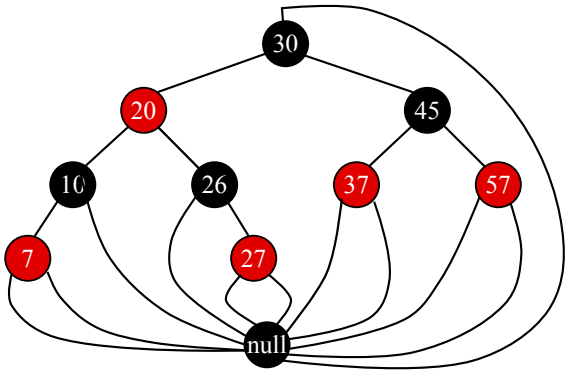


Figura 1.23: Exemplu de arbore roşu - negru cu santinelă.

Pentru refacerea proprietăţii de arbore roşu-negru sunt necesare operaţii de recolorare a nodurilor şi de schimbare a structurii de pointeri. Schimbarea structurii de pointeri se realizează prin operaţii de rotaţie.

1.3.1 Inserţia într-un arbore roşu-negru

Inserţia propriu-zisă a unui nod într-un arbore roşu-negru se realizează după acelaşi algoritm ca şi inserţia într-un arbore binar de căutare. Practic se porneşte de la rădăcină şi se compară la fiecare pas cheia nodului curent x cu cheia nodului z care se inserează, coborându-se în subarborele stâng dacă $z.info < x.info$ şi în cel dreapt altfel. Apoi se refac proprietăţile ARN.

Observaţii:

- Întotdeauna nodul care se introduce are culoarea roşie
- Se utilizează o santinelă T.nil

Proprietăţile care pot fi neîndeplinite în cazul inserţiei:

- Proprietăţile 1, 3 şi 5 se păstrează, datorită faptului că se inserează un nod roşu, care are ca descendenţi două frunze NULL, deci se leagă de nodul santinelă T.nil
- Proprietatea 2 poate fi contrazisă dacă nodul inserat este chiar rădăcina. În acest caz este suficientă recolorarea nodului negru.
- Proprietatea 4 poate fi contrazisă, dacă părintele de care s-a legat noul nod are culoarea roşie. În acest caz trebuie refăcută proprietatea. Există în această situaţie 3 cazuri care vor fi discutate în continuare.

Funcţia de refacere a proprietăţilor RN este apelată doar dacă părintele nodului inserat este roşu.

Observaţii:

- Dacă notăm cu z nodul inserat (nodul curent), cu P părintele său, cu U unchiu (fratele părintelui) iar cu B bunicul, atunci z se poate afla la stânga sau la dreapta lui P. Cele două cazuri se tratează în mod similar, prin simetrie. Vom considera în continuare inserţia pe stânga bunicului B.
- Datorită faptului că P are culoarea roşie, deja înainte de inserţie, iar inserţia s-a produs într-un arbore roşu-negru valid, înseamnă că P nu este rădăcină şi deci există nodul $B \neq T.nil$

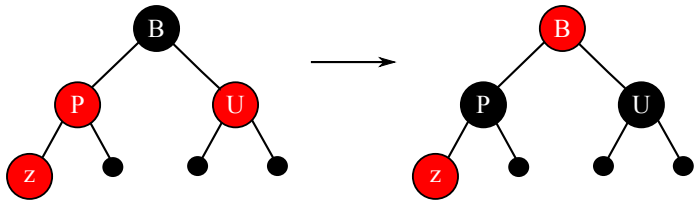


Figura 1.24: Cazul 1 pentru inserţie.

Cazurile de refacere a proprietăţii 4 în urma inserţiei

Cazu 1: unchiul lui z este roşu (fig. 1.24).

Deoarece U, P sunt roşii rezultă că B are culoarea neagră, altfel s-ar contrazice proproietatea 4, $\Rightarrow$ este suficientă recolorarea P, U, B, adică P şi U devin negri iar B roşu.

În urma acestei modificări poate avea loc o contrazicere a proprietăţii 4 pentru B şi părintele său, deci se reia procedura de refacere a proprietăţilor roşu-negru, considerând de data aceasta $z = B$.

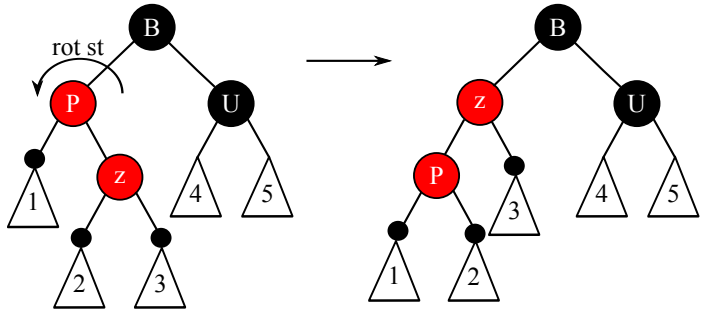


Figura 1.25: Cazul 2 pentru inserţie.

Observaţie: Înălţimea neagră a arborelui nu se modifică.

Acelaşi procedeu se aplică şi în cazul în care z se află pe dreapta lui P.

Cazul 2: unchiul este negru şi z se află pe dreapta lui P (P se află la stânga lui B) - fig.1.25

Cazul 3: unchiul este negru şi z se află pe stânga lui P (P se află la stânga lui B) - fig.1.26

Observaţie: printr-o rotaţie la stânga în jurul lui P, cazul 2 se reduce

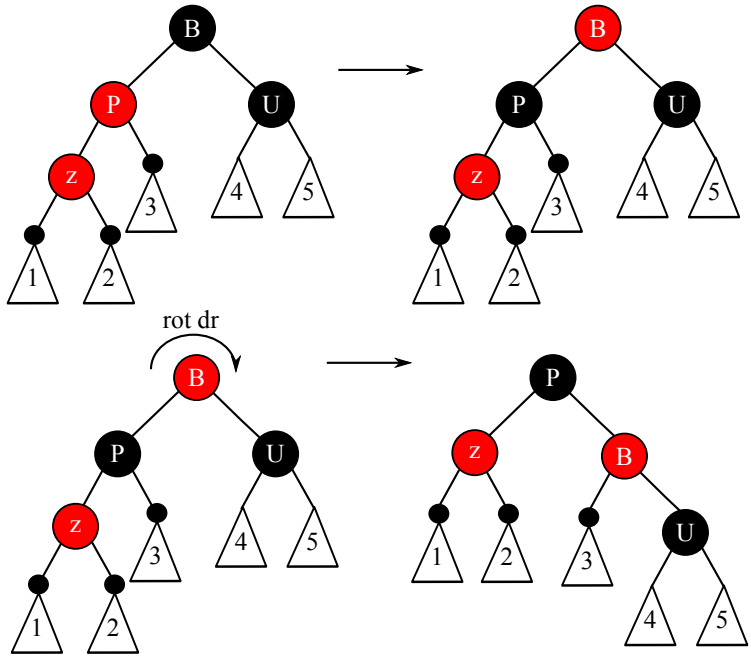


Figura 1.26: Cazul 3 pentru inserţie.

la cazul 3.

Cazul 3 se rezolvă în modul următor:

1. Recolorare B, P.
2. Rotaţie la dreapta în jurul lui B.

Se observă faptul că înălţimea neagră nu se modifică pe nici una dintre cele două ramuri.

Un exeplu de construcţie a unui ARN prin inserţie succesivă de chei este prezentat în figura 1.27.

Algoritm: inserarea nodului z în arborele T

```
ARN_INSERT(T,z)
    y = T.nil
    x = T
    cat timp x ≠ T.nil
```

```
                y = x
                daca z.info < y.info atunci
                        x = x.st
                altfel x = x.dr
                sfarsit daca
        sfarsit pentru
        z.p = y
        daca y = T.nil atunci
                r = z
        altfel
                daca z.info < y.info atunci
                        y.st = z
                altfel
                        y.dr = z
                sfarsit daca
        sfarsit daca
        z.st = T.nil
        z.dr = T.nil
        z.color = rosu
        ARN_INSERT_REPARA(T,z)
    RETURN
```

Algoritm: refacerea proprietăţilor de arbore roşu-negru

```
    ARN_INSERT_REPARA(T,z)
        cat timp z.p.color = rosu
                daca z.p = z.p.p.st atunci
                        y = z.p.p.dr
_____________________________________________________ caz 1
                        daca y.color = rosu atunci
                                z.p.color = negru
                                y.color = negru
                                z.p.p.color = rosu
                                z = z.p.p
                        altfel
_____________________________________________________ caz 2
                                daca z = z.p.dr atunci
                                        z = z.p
                                        ROT_ST(T, z)
```

```
                    sfarsit daca
                                                        caz 3
                    z.p.color = negru
                    z.p.p.color = rosu
                    ROT_DR(T,z.p.p)
                sfarsit daca
                altfel
                //similar dar simetric pentru nodul z
                //aflat la stânga bunicului
            sfarsit daca
    sfarsit cat timp
    T.rad.color = negru
RETURN
```

Complexitate: Inserţia are aceeaşi complexitate ca şi în cazul arborilor binari de căutare simpli, deci $O(\log_2 n)$, iar algoritmul de refacere al proprietăţilor de arbore roşu-negru porneşte de jos înspre rădăcină pe o ramură, deci complexitatea este tot $O(\log_2 n)$. Rezultă complexitatea pentru inserţie este $O(\log_2 n)$.

1.3.2 Ştergerea dintr-un arbore roşu-negru

Operaţia de ştergere dintr-un arbore roşu-negru este mai complicată decât operaţia de inserţie. În prima etapă, operaŁŁia de Łztergere are la bază ştergerea dintr-un arbore binar de căutare, cu câteva modificări, după care trebuie efectiuată o operaţie de refacere a proprietăţilor de arbore roşu-negru.

Reamintim faptul că, în cazul ştergerii unui nod z dintr-un arbore binar de căutare se luau în considerare cazurile:

1. z nu are descendent stâng, deci $z.st = T.nil$ - fig.1.28 a)
2. z nu are descendent drept, deci $z.dr = T.nil$ - fig.1.28 b)
3. z are doi descendenţi (diferiţi de T.nil) şi $y = sucesor(T, z)$
   a. y descendent direct al lui z - fig.1.28 c)
   b. y nu este descendent direct al lui z - fig.1.28 d)

Observaţii:

- În cazurile de ştergere 1 şi 2 y reprezintă chiar nodul care va fi şters, deoare-ce este iniţializat cu z. În cazul 3, y reprezintă nodul cu care va fi înlocuit z.
- Prin operaţiile de transplantare, pointerii lui y se pot modifica, deci şi culoarea. Din acest motiv se utilizează o variabilă care reţine valoarea originală a lui y.
- În cazurile 1 şi 2, prin ştergerea nodului z (reprezentat şi prin y) şi înlocuirea cu unul dintre descendenţii direcţi, se poate produce o modificare a culorii în poziţia deţinută anterior de z, ceea ce poate duce la violarea proprietăţilor de arbore roşu-negru. La fel în cazul 3, la înlocuirea lui y cu x.
- Nodul x este acela care se deplasează în poziţia deţinută anterior de nodul y. Se observă faptul că, x poate fi santinela T.nil

Refacerea proprietăţilor de arbore roşu-negru:

- Dacă nodul y a avut culoarea iniţială roşu, atunci se păstrează proprietăţile de arbore RN, deoarece
  1. Nu s-a modificat înălţimea neagră pe nici o ramură prin eliminarea unui nod roşu
  2. Nici un nod roşu nu a căpătat descendenţi roşii
  3. Deoarece nodul y nu a fost rădăcină (culoarea originală a fost roşie) nu s-a modificat nici culoarea rădăcinii, care a rămas neagră
- Rezultă deci că, proprietăţile de arbore RN sunt violate doar dacă a fost eliminat un nod negru. Există mai multe cazuri care trebuie tratate.
- Prin eliminarea unui nod negru din arbore
  1. În primul rând se micşorează înălţimea neagră pe ramura respectivă
  2. Poate apărea vecinătate între două noduri roşii

Cazul 0: culoarea nodului x cu care s-a făcut înlocuirea este roşie. În acest caz singurul lucru care trebuie făcut este recolorarea lui x în negru. Acest lucru rezolvă atât problema (1) cât şi problema (2).

Notaţii: P=părintele lui x, F=fratele lui x, hp = înălţimea neagră originală a subarborelui cu rădăcina P (exclusiv nodul P).

În continuare vom considera că nodul x se află la stânga nodului P. Pentru x la dreapta lui P modul de rezolvare este simetric.

Cazul 1: F are culoarea roşie - fig. 1.29

Facem următoarele observaţii:

- Datorită faptului că înainte de ştergere arborele era RN valid, din F roşu şi F descendent direct al lui P rezultă culoarea lui P este neagră.
- Copiii lui F sunt ambii negri (eventual T.nil)

- Prin ştergere, s-a micşorat înălţimea neagră pe partea stângă a lui P, adică subarborele drept are înălţimea neagră hnp, iar subarborele stâng $hp-1$. Cum F este roşu rezultă că ambii săi subarbori au înălţimea hnp.

Refacerea proprietăţilor de arbore roşu-negru:

- Recolorare P şi F
- Rotaţie la stânga în jurul lui P

În această situaţie F, acum negru, urcă în locul lui P. La dreapta lui F înălţimea neagră a rămas hnp. Înălţimea neagră a subarborelui drept al lui P este hnp, iar înălţimea neagră subarborelui stâng al lui P este $hp-1$. Deci problema în nodul P a rămas, dar descendentul drept al lui P nu mai este roşu ci negru, ceea ce conduce la unul dintre cazurile 2, 3 sau 4.

Deci din cazul 1 se trece la cazul 2, 3 sau 4!

În figura 1.30 este ilustrată procedura de refacere a proprietăţilor RN în cazul 1.

Cazul 2: F are culoarea neagră şi ambii descendenţi ai săi sunt negri (eventual T.nil) - fig. 1.31.

Observaţii:

- F sigur este diferit de $T.nil$, pentru că pe partea stângă a lui P s-a şters un nod negru, iar înainte de ştergere pe ambele părţi ale lui P înălţimea neagră era aceeaşi şi cel puţin 1.
- P poate fi roşu sau negru
- Prin ştergere s-a micşorat înălţimea neagră pe partea stângă a lui P, adică subarborele drept are înălţimea neagră hp, iar subarborel stâng $hp-1$. Cum F este negru rezultă că ambii săi subarbori au înălţimea $hp-1$.

Refacerea proprietăţilor de arbore RN: - fig. 1.32

- Se recolorează F roşu $\Rightarrow$ înălţimea neagră a subarborelui drept al lui P este $hp-1$, deci egală cu cea a subarborelui stâng, dar înălţimea neagră a arborelui P este mai mică decât înainte de ştergere, deci cu o

unitate mai mică decât a fratelui său (dacă există). Această problemă se rezolvă astfel:

- Dacă culoarea lui P este roşie atunci este suficient să recolorăm P cu negru - fig.1.32 a).
- Altfel problema de refacere a proprietăţilor RN se reia pentru subarborele care are ca rădăcină părintele lui P. Se observă din figura 1.32 b) că, în acest caz înălţimea neagră a întregului arbore care are rădăcina P a scăzut cu 1 şi trebuie reechilibrat de la acest nod în sus.

Cazul 3: F are culoarea neagră, descendentul stâng al lui F notat cu FS este roşu, iar cel drept notat cu FD este negru. (fig. 1.33)

Observaţii:

- Culoarea lui P poate fi roşie sau neagră.
- F sigur este diferit de $T.nil$, pentru că pe partea stângă a lui P s-a şters un nod negru, iar înainte de ştergere pe ambele părţi ale lui P înălţimea neagră era aceeaşi şi cel puţin 1.
- Înălţimea neagră a subarborelui stâng al lui P este $hp-1$, înălţimea neagră a subarborelui cu rădăcina F este hp. Înălţimile neagre ale lui FS şi FD sunt $hp-1$.

Refacerea proprietăţilor de arbore RN: - fig. 1.34

- Recolorare F şi $FS \Rightarrow F$ devine roşu şi FS devine negru $\Rightarrow$ înălţimea neagră a lui FS devine hp şi înălţimea neagră a lui FD devine $hp-1$.
- Rotaţie la dreapta în jurul lui $F \Rightarrow FS$ urcă în locul lui F, înălţimea neagră a lui FS devine hp, înălţimea neagră a lui F devine $hp-1 =$ înălţimea neagră a descendentului stăng al lui FS.

Problema la care am ajuns în continuare este refacere proprietăţilor RN pentru cazul 4.

Cazul 4: F are culoarea neagră iar descendentul drept notat cu FD este roşu. Descendentul stâng, FS, poate avea oricare dintre cele două culori. (fig. 1.35)

Observaţii:

- Culoarea lui P poate fi roşu sau negru.
- F sigur este diferit de $T.nil$, pentru că pe partea stângă a lui P s-a şters un nod negru, iar înainte de ştergere pe ambele părţi ale lui P înălţimea neagră era aceeaşi şi cel puţin 1.
- Înălţimea neagră a subarborelui stâng al lui P este $hp - 1$, înălţimea neagră a subarborelui cu rădăcina F este hp. Înălţimile negre ale lui FS şi FD sunt $hp - 1$.

Refacerea proprietăţilor de arbore RN: - fig. 1.36.

- Colorare F cu culoarea lui P
- Colorare P cu negru
- Colorare FD cu negru
- Rotaţie la stânga în jurul lui P

Prin aceste operaţii rezultă: creşterea cu o unitate a înălţimii negre a subarborelui F, adică $bh(F) = hp + 1$. Dar $bh(x) = hp - 1$. Prin rotaţia în jurul lui P, P fiind acum negru, se rebalansează arborele.

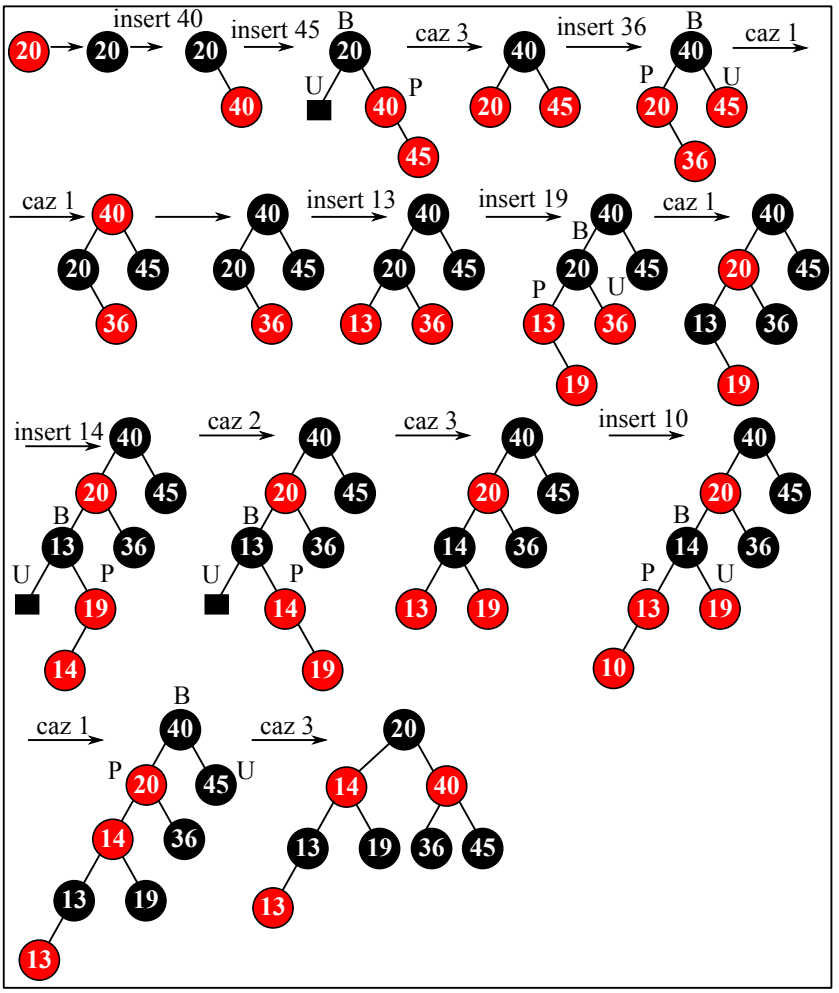


Figura 1.27: Inserarea cheilor 20, 40, 45, 36, 13, 14, 19, 10 într-un ARN iniţial vid.

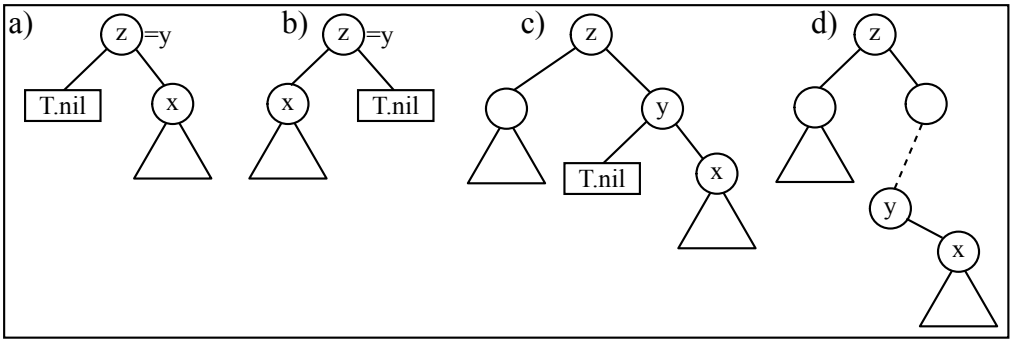


Figura 1.28: Cazurile pentru ştergerea nodului z. a) z nu are descendent stâng, b) z nu are descendent drept, c) z are 2 descendenţi nenuli şi succesorul lui z este fiu al lui z, d) z are doi descendenţi şi succesorul lui z nu este fiu al lui z.

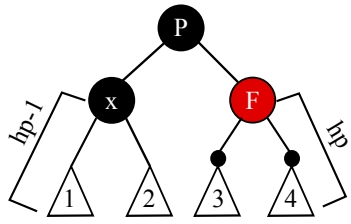


Figura 1.29: Cazul 1 în ARN

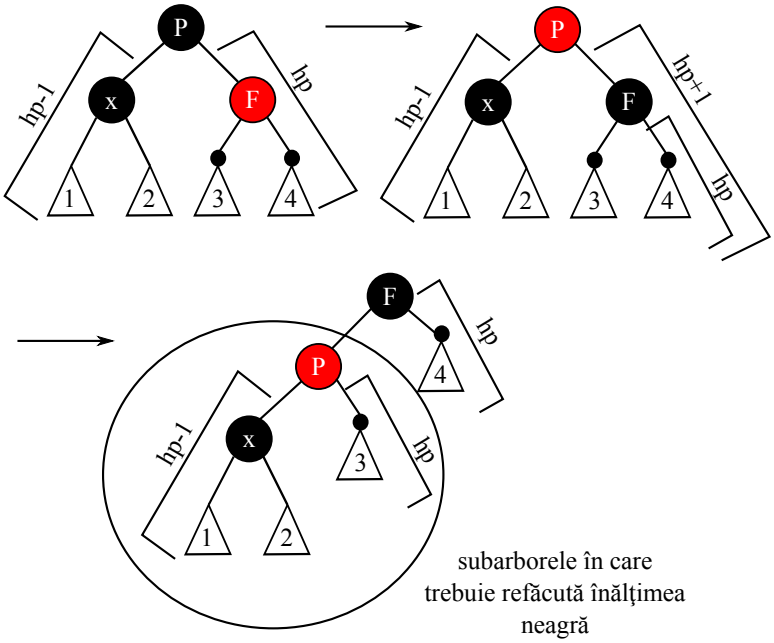


Figura 1.30: Refacerea proprietăţilor RN în cazul 1.

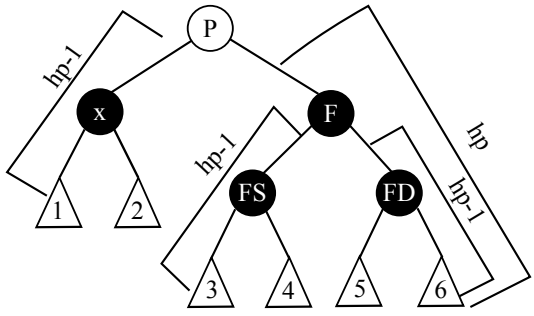


Figura 1.31: Cazul 2 în ARN

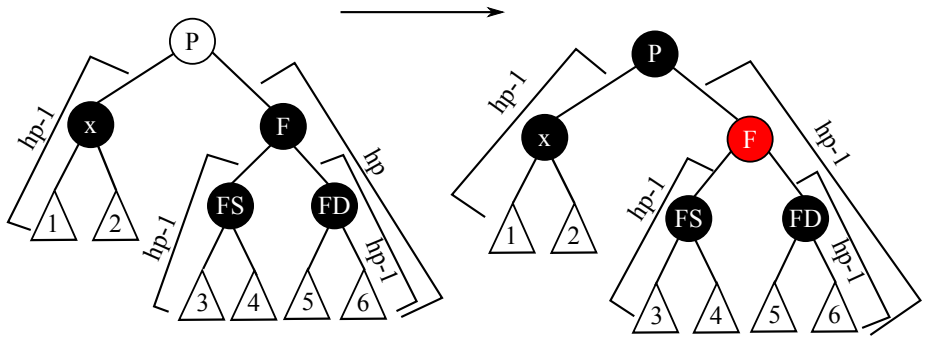


Figura 1.32: Soluţionarea cazului 2 în cazul ştergerii unei chei dintr-un ARN. Dacă P era roşu el va deveni negru. Se observă că, de fapt se mută problema refacerii înălţimii negre la nivelul nodului P.

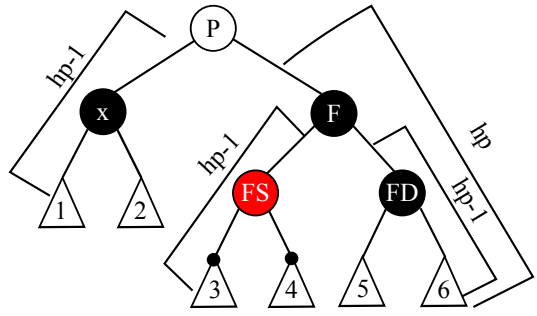


Figura 1.33: Cazul 3 în ARN

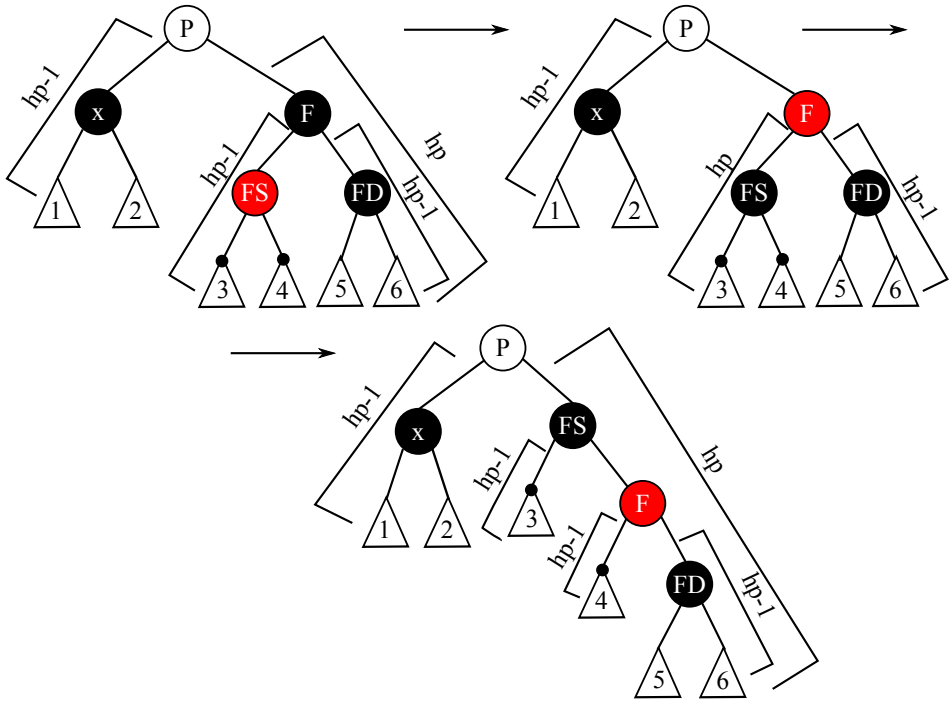


Figura 1.34: Soluţionarea cazului 3 în cazul ştergerii unei chei dintr-un ARN, conduce la cazul 4.

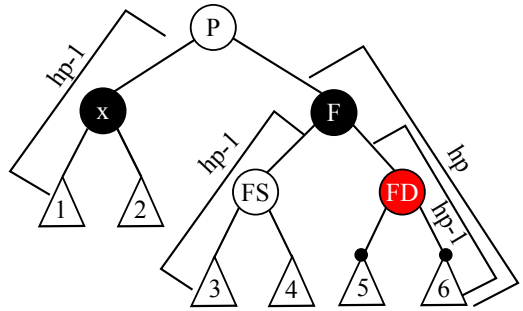


Figura 1.35: Cazul 4 în ARN

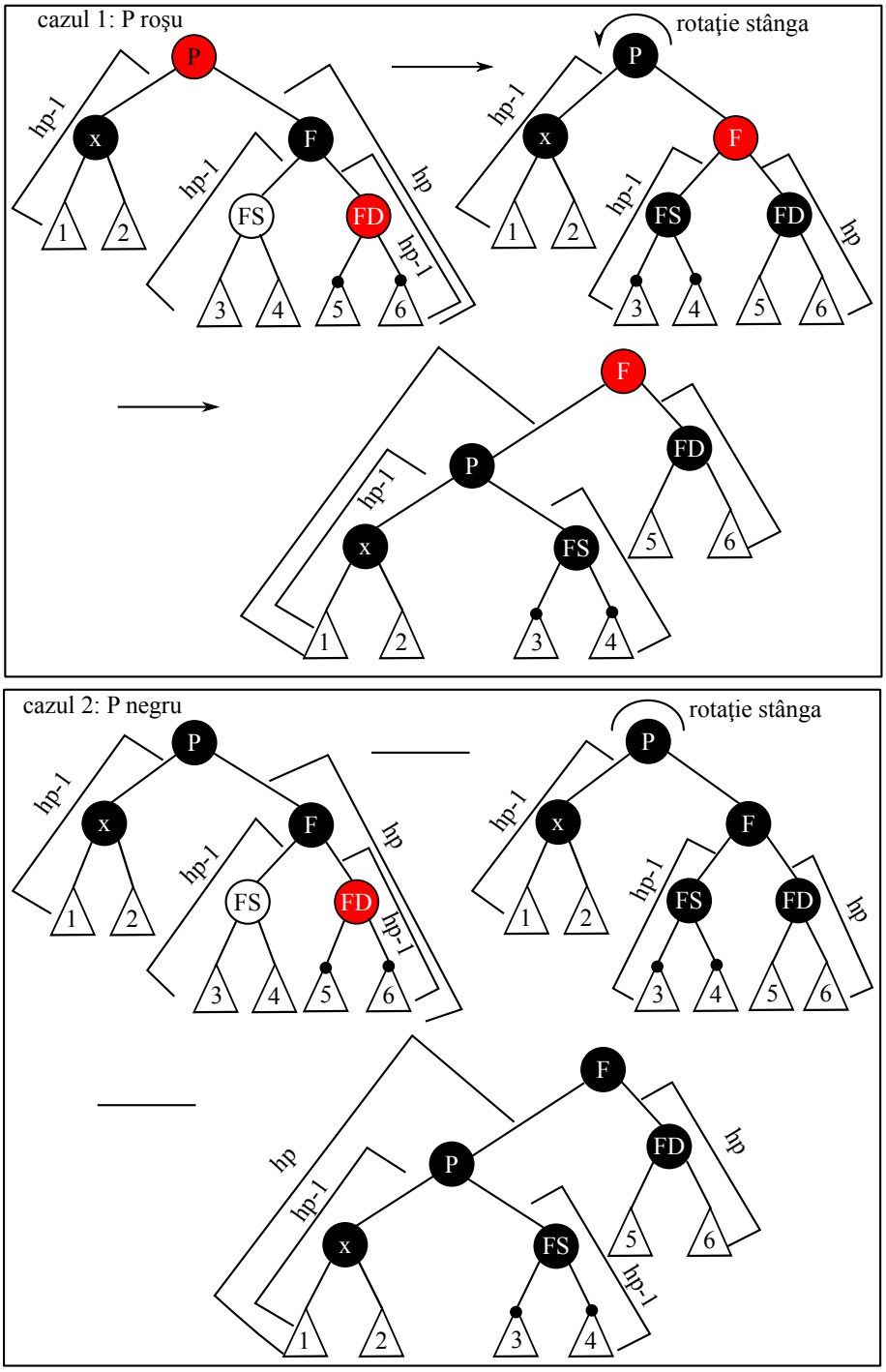


Figura 1.36: Soluţionarea cazului 4 în cazul ştergerii unei chei dintr-un ARN. Cu acest caz se încheie procedeul de refacere al proprietăţilor ARN.

Capitolul 2

Îmbogăţirea structurilor de date

Îmbogăţirea structurilor de date se realizează în general cu scopul de-a putea efectua şi alte operaţii -cereri - în afara celor uzuale, descrise în capitolele anterioare, sau pentru a creşte eficienţa celor uzuale. Îmbogăţirea unei structuri de date se referă la ataşarea de informaţii suplimentare elementelor structurii - adică adăugarea de câmpuri suplimentare fiecărui nod, astfel încât să permită efectuarea operaţiilor suplimentare dorite. Aceste câmpuri suplimentare trebuie actualizate odată cu modificarea mulţimii dinamice stocate cu ajutorul structurii de date.

Informaţia suplimentară adăugată nodurilor structurii de date poate fi o valoare, dar poate fi şi o referinţă (pointer)!

Procesul de îmbogăţire a unei structuri de date are 4 etape [1]:

1. Alegerea unei structuri de date de bază, pe care urmează să o îmbogăţim: de exempu un arbore roşu-negru.
2. Determinarea informaţiei suplimentare care trebuie păstrată în structura aleasă.
3. Verificarea faptului că, informaţia suplimentară aleasă poate fi actualizată pe parcursul operaţiilor de bază de modificare a structurii, respectiv inserţie/ştergere, fără a creşte complexitatea acestora. Va fi enunţată mai jos o teoremă, care permite în anumite condiţii bine definite, demonstrarea păstrării complexităţii logaritmice în cazul îmbogăţirii unui arbore binar care se autoechilibrează.

4. Dezvoltarea de noi operaţii utile asupra structurii de date îmbogăţite.

Un exemplu clasic pentru îmbogăţirea arborilor roşu-negru sunt arborii pentru statistici de ordine, prezentaţi în continuare.

2.0.1 Arbori pentru statistici de ordine

În cadrul cursului de algoritmică, odată cu descrierea algoritmului *Quicksort*, s-a discutat şi despre statistici de ordine. De fapt *statistica de ordin i* a unei mulţimi de n elemente $a_1, a_2, \ldots, a_n$, este acel element, care s-ar afla în şirul sortat al elementelor pe poziţia a i-a. S-a discutat cum poate fi modificat algoritmul *Quicksort*, astfel încât această problemă să fie rezolvată în timp liniar.

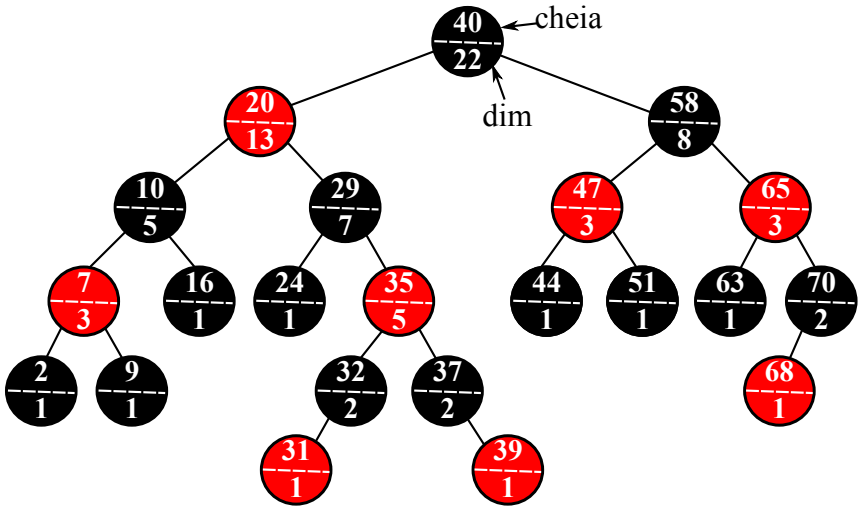


Figura 2.1: Exemplu de ARN pentru statistici de ordine. Câmpul dimensiune = *dim* al unui nod x = numărul de noduri din subarborele de rădăcină x

Prin îmbogăţirea structurilor de date se poate obţine însă o complexitate $O(\log_2 n)$ pentru determinarea statisticii de ordin i. În plus, pentru fiecare element din şir, poate fi calculat tot cu o complexitate $O(\log_2 n)$ rangul său, adică poziţia pe care s-ar afla în şirul sortat.

Definiţie: Un *arbore pentru statistici de ordine* este un ARN, în care fiecare nod x conţine un câmp suplimentar, $x.dim$, care reprezintă numărul

de noduri - fără frunzele nil - ale subarborelui cu rădăcina x. În cazul nodului santinelă $T.nil$, se consideră $T.nil.dim = 0$. Se observă uşor relaţia:

$$x.dim = x.st.dim + x.dr.dim + 1$$

În figura 2.1 este reprezentat un arbore roşu-negru pentru statistici de ordine.

Pentru un astfel de arbore îmbogăţit cu informaţia *dim* pot fi definite două operaţii noi:

(I) Căutarea elementului de rang i = elementul al cărei chei s-ar afla pe poziţia i în şirul sortat al cheilor din arbore.

(II) Determinarea rangului unui element = poziţia pe care s-ar afla cheia elementului respectivă în şirul sortat al cheilor din arbore.

Deoarece prin parcurgerea în inordine a unui arbore binar de căutare se obţine şirul sortat al cheilor, rangul unui element este de fapt poziţia sa în urma parcurgerii în inordine a arborelui.

(I) Căutarea elementului de rang i

Observaţi

- Pornind de la rădăcina $T.rad$, în cazul parcurgerii în inordine, întâi se parcurg toate elementele din subarborele drept, care sunt în număr de $T.rad.st.dim$, după care se afişază cheia rădăcinii, iar apoi se parcurg elementele din subarborele drept. Deci rangul lui $T.rad$ este $T.rad.st.dim + 1$.
- Pornind de la un nod x, rangul lui x în cadrul mulţimii formate din elementele aflate în subarborele x este $x.st.dim + 1$.

Putem astfel determina elementul de rang i prin următorul algoritm:

```
ArbStat_SELECT(T,i)
   x = T.rad
   cat timp x ≠ T.nil
        rang = x.st.dim + 1
        daca i = rang atunci
             RETURN x
        sfarsit daca
        daca i<rang atunci
```

```
            x=x.st
        altfel
            x=x.dr
            i=i-rang
        sfarsit daca
    sfarsit cat timp
RETURN x
```

Exemplul: În arborele din figura 2.1 se caută nodul de rang $r = 16$. Exemplul este ilustrat în figura 2.2

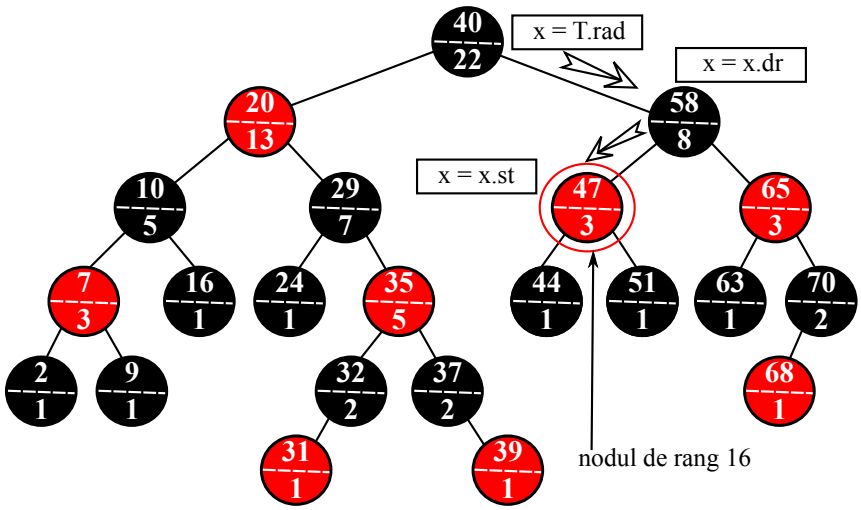


Figura 2.2: Determinarea elementului cu rangul 16.

Notăm cu x nodul curent.

- Iniţial $x = T.rad$, deci $rang = x.st.dim + 1 = 13{+}1{=}14$. Rădăcina se află pe poziţia 14 în parcurgerea în inordine, deoarece trebuie întâi să se parcurgă toate cele 13 noduri din subarborele stâng.

- $rang < r \Rightarrow$ cobor pe dreapta în arbore prin $x = x.dr$ iar variabila $r = r - rang = 2$, adică se caută elementul de rang 2 în subarborele de răcină $x.dr$, care are cheia 58. Rădăcina se află pe locul 14, iar nodul pe care îl caut se află cu 2 poziţii mai la dreapta de rădăcină, deci pe poziţia 2 în subarborele drept.

- În continuare nodul curent este $x = T.rad.dr$, în această situaţie $rang = x.st.dim + 1 = 4$, deci x se află pe poziţia 4 în subarborele curent, dar nodul căutat se află pe poziţia 2 în acest subarbore.

- Se intră deci pe prima ramură a instrucţiunii condiţionale şi $x = x.st$ Astfel nodul curent este $T.rad.dr.st$ cu cheia 47 şi variabila $rang = x.st.dim + 1 = 2$

- Nodul curent x cu cheia 47 are rangul $2 = r \Rightarrow$ am ajuns la nodul căutat.

Nodul cu rangul 16 este cel cu cheia 47. Se observă uşor faptul că şirul sortat al cheilor este {2, 7, 9, 10, 16, 20, 24, 29, 31, 32, 35, 37, 39, 40, 44, 47, 51, 58, 63, 65, 68, 70}, iar al 16-lea element este 47.

Complexitate: algoritmul coboară la fiecare apel iteraţie cu un nivel în arbore. Rezultă faptul că timpul de execuţie este direct proporţional cu înălţimea arborelui, deci este $O(\log_2 n)$.

(II) Determinarea rangului unui element

Observăm următoarele lucruri:

- Dacă la nodul x se ajunge doar prin coborârea în arbore pe descendenţi stângi, atunci rangul lui este chiar $x.st.dim + 1$, adică rangul său în cadrul subarborelui de rădăcină x.

- Dacă x se află la dreapta părintelui său, adică $x.p.dr = x$, atunci în parcurgerea în inordine subarborelui de rădăcină $x.p$, înaintea lui x se vor afla $x.p.st.dim + 1$ elemente.

Putem afla astfel rangul unui nod x în modul următor:

- Pornim de la nodul x către rădăcină, iniţializând rangul cu $x.st.dim+1$

- Atunci când nodul curent pe drumul spre rădăcină se află pe stânga părintelui său, se lasă rangul nemodificat

- Atunci când nodul curent pe drumul spre rădăcină se află pe dreapta părintelui său, la rang se adaugă $x.p.st.dim + 1$

- Algoritmul se încheie atunci când nodul curent l-a care s-a ajuns este chiar rădăcina

Algoritm

```
ArbStat_RANG(T,x)
    rang = x.st.dim + 1
    cat timp x ≠ T.rad
        y = x.p
        daca x = y.dr atunci
              rang = rang + y.st.dim + 1
        sfarsit daca
        x = y
    sfarsit cat timp
 RETURN rang
```

Exemplul: În arborele din figura 2.1 se determină rangul nodului x cu cheia 32.

- $x = x.p.st$, deci poziţia lui x în parcurgerea în inordine a arborelui de rădăcină $x.p$ este aceeaşi cu poziţia în parcurgerea în inordine a subarborelui cu rădăcina $x.p$, adică $x.st.dim + 1 = 2$, deci $rang = 2$

- nodul curent devine $x = x.p$ (adică nodul cu cheia 35), $rang = 2$. Noul nod curent cu cheia 35 se află pe dreapta părintelui, deci poziţia lui în parcuregrea în inordine a subarborelui de rădăcină $x.p$ este poziţia lui $x.p$ + rangul în subarborele de rădăcină x, adică $x.p.st.dim + 1 + rang = 4$.

- nodul curent devine din nou $x = x.p$ (adică nodul cu cheia 29). Acest nod se află pe dreapa părintelui, deci $rang = rang + x.p.st.dim + 1$ = 4+5+1=10

- se trece la nodul părinte, care este nodul cu cheia 20, care se află pe stânga rădăcinii

- Rezultatul final este: nodul cu cheia 32 are $rang = 10$, deci se află pe poziŁŁia 10 în parcurgerea în inordine a arborelui.

Complexitate: se parcurge arborele din nodul x către rădăcină, deci complexitatea este $O(\log_2 n)$

Refacerea informaţiei dimensiunii subarborilor după operaţi de modificare

1. **La inserţie:** la fiecare nod care este parcurs pe drumul spre locul de inserţie se incrementează câmpul *dim* cu o unitate. Nodul nou adăugat va avea câmpul $dim = 1$.

2. **La ştergere:** se parcurge arborele de la părintele nodul z care a fost şters (dacă acesta a avut cel mult un descendent) respectiv de la părintele succesorului lui z, către rădăcină şi pentru fiecare nod de pe acest drum, câmpul *dim* se decrementează cu o unitate.

3. **La refacerea proprietăţilor de ARN**, operaţia de rotaţie produce modifiări în numărul de noduri ale unui subarbore în modul următor: considerăm rotaţia la dreapta în jurul nodului x din figura 2.3.

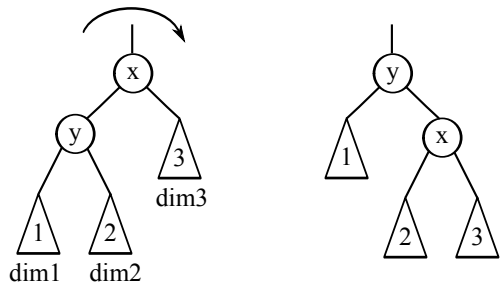


Figura 2.3:

Se observă din fig.2.3 că iniţial $x.dim = y.dim + dim3 + 1$. După rotaţie y urcă în locul lui x, dar numărul de noduri din subarborele considerat rămâne nemodificat, deci:

- $y.dim = x.dim$ (valoarea originală a câmpului).
- x coboară în dreapta lui y şi are noua valoare a câmpului dimensiune: $x.dim = dim2 + dim3 + 1 = x.st.dim + x.dr.dim + 1$

Deci la funcţia de rotaţie trebuie adăugate aceste două linii de cod:

$y.dim = x.dim$
$x.dim = x.st.dim + x.dr.dim + 1$

Etapele îmbogăţirii în cazul arborilor pentru statistici de ordine

1. Alegerea unei structuri de date de bază: un ARN
2. Determinarea informaţiei suplimentare: câmpul dimesiune = nr. de noduri di subarborele curent
3. Verificarea actualizării în timp logaritmic: - s-a arătat mai sus
4. Dezvoltarea de noi operaţii utile asupra structurii de date îmbogăţite: funcţiile: Stat_SELECT şi Stat_RANG.

2.0.2 Îmbogăţirea arborilor RN - teorema complexităţii

Următoarea teoremă demonstrează faptul că, dacă informaţia suplimentară păstrată în fiecare nod respectă anumite condiţii, atunci complexitatea operaţiilor de inserţie/ştergere într-un arbore RN îşi păstrează complexitatea nemodificată, adică $O(\log_2 n)$.

Teoremă: Considerăm un atribut/câmp suplimentar f prin care se îmbogăţeşte un arbore roşu-negru T cu n noduri. Dacă valoarea câmpului f al oricărui nod x depinde doar de informaţiile din nodurile x, $x.st$, $x.dr$ şi eventual de valorile $x.st.f$ şi $x.dr.f$, atunci valoarea câmpului suplimentar poate fi actualizată la orice operaţie de inserţie/ştergere fără a afecta complexitatea $O(\log_2 n)$ a acestor operaţii. [1]

Demonstraţie: În cazul inserţiei: dacă s-a inserat nodul x ca descendent al nodului $x.p$, valoarea lui $x.f$ se calculază instantaneu, deoarece depinde doar de valorile din x şi de santinela $T.nil$. Modificarea adusă de inserarea lui x se poate reflecta asupra părintelui $x.p$. Modificarea lui $x.p.f$ este de asemenea $O(1)$, iar modificările se refelectă doar asupra părintelui acestuia. Astfel se parcurge arborele de la x către rădăcină şi se actualizează doar câmpul f pentru nodurile aflate pe acest drum. Ceea ce duce la complexitate $O(\log_2 n)$. În cazul operaţiei de refacere a proprietăţilor RN, apar un număr limitat de rotaţii, la care doar câmpurile a două noduri sunt actualizate, iar modificările acestora se propagă doar către părinte şi astfel, iar către rădăcină într-un timp $O(\log_2 n)$. Astfel per total se păstrează complexitatea $O(\log_2 n)$.

Aceeaşi argumentaţie se poate efectua în cazul operaţiei de ştergere.

2.0.3 Arbori de intervale

Intervalele pot fi utilizate pentru reprezentarea şi gestionarea evenimentelor/proceselor care se desfăşoară în timp. Un arbore binar de căutare, care are ca informaţie un interval, poate fi utilizat de exemplu pentru căutarea într-o bază de date conţinând evenimente ce se desfăşoară în timp. Pentru a obţine operaţii de inserţie, ştergere şi căutare eficientă se poate utiliza ca structură de bază un arbore roşu-negru.

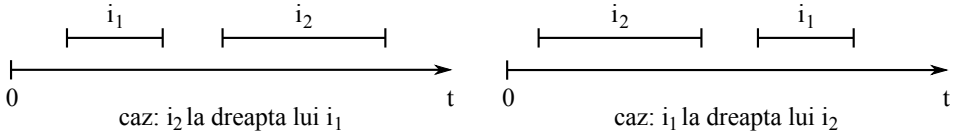
Un interval închis $i = [t_1, t_2]$ este caracterizat prin cele două capete ale sale t_1 şi t_2.

Notăm: $i.low = t_1$ şi $i.high = t_2$.

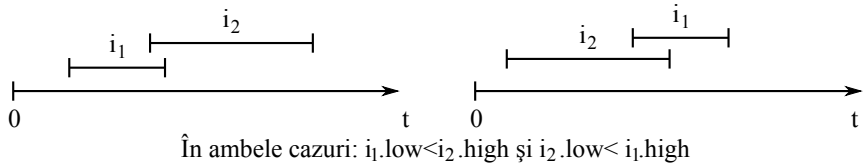
Putem compara două intervale prin verificarea dacă se intersectează sau nu.

Observăm următoarele: considerând două intervale i_1 şi i_2 putem avea următoarele situaţii:

a. Cele două intervale nu se intersectează deloc, atunci sau $i_1.high < i_2.low$ (adică i_2 e la dreapta lui i_1) sau $i_2.high < i_1.low$ (adică i_1 e la dreapta lui i_2).



b. Se intersectează, atunci $i_1.low \leq i_2.high$ şi $i_2.low \leq i_1.high$



Un arbore de intervale se construieşte respectând etapele generale definite mai sus:

1. Alegerea unei structuri de date de bază: un ARN, în care fiecare nod are ca informaţie un interval $x.int$, iar cheia după care se realizează inserţia/ştergerea în arbore este $x.int.low$, adică ordonearea

intervalelor se realizează în funcţie de capătul din stânga. Inserţia şi ştergerea se efectuează ca în orice arbore RN. Căutarea se efectuează prin INERVAL_SEARCH(T,i), care caută în T un nod x pentru care $x.int$ se intersectează cu i sau $T.nil$ dacă nu există nici un astfel de interval.

2. Informaţia suplimentară prin care se îmbogăţeşte structura de date: fiecare nod x conţine un câmp $x.max = max(y.int.high$, y nod din subarborele de rădăcină $x)$.

3. Actualizarea informaţiei suplimentare în urma modificării arborelui prin inserţie/ştergere: s-a enunţat înainte o teoremă prin care se demonstra că, dacă informaţia suplimentară dintr-un nod x depinde doar de cheia lui x, de cheile descendenţilor săi şi de informaţiile suplimentare din descendenţii săi, atunci actualizarea informaţiei nu creşte complexitatea operaţiilor de inserţie/ştergere. Dar în cazul unui arbore de intervale:

$$x.max = \max(x.int.high, x.st.max, x.dr.max)$$

deci, condiţiile se respectă.

4. Operaţia nouă care se dezvoltă pe baza informaţiei suplimentare este INTERVAL_SEARCH(T,i), al cărei algoritm este prezentat în continuare.

```
INTERVAL_SEARCH(T,i)
   x = T.rad
   cat timp x ≠ T.nil şi nu se intersectează cu x.int
       dacă x.st ≠ T.nil si i.low ≤ x.st.max atunci
            x = x.st
       altfel x = x.dr
RETURN x
```

Exemplu: Un exemplu de arbore de intervale este prezentat în fig. 2.4.

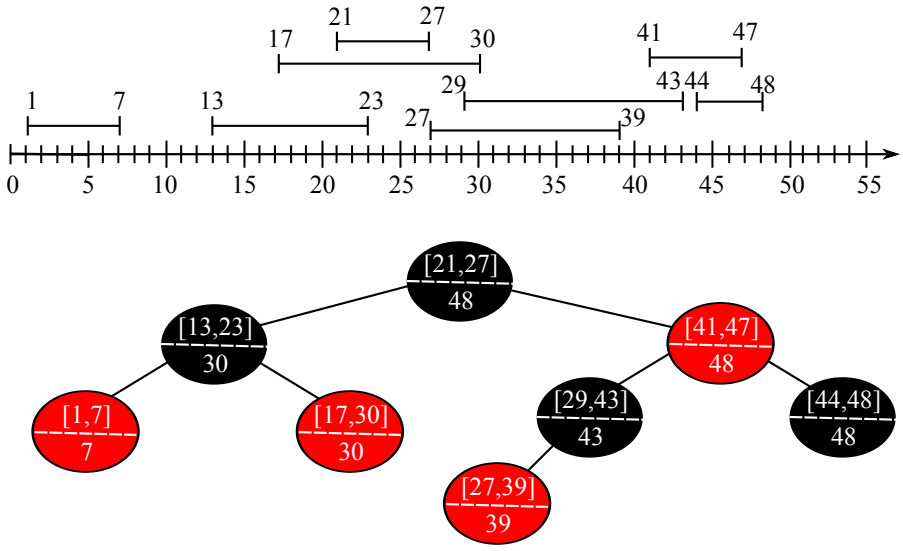


Figura 2.4: O mulţime de intervale împreună cu arborele RN corespunzător.

.

Bibliografie

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein *Introduction to Algorithms* Third Edition, MIT Press, Cambridge, Massachsusett London, 2009.

[2] G. Adelson-Velskii, E. M. Landis *An algorithm for the organization of information.* Proceedings of the USSR Academy of Sciences 146: 263-266. (1962)

[3] Granville Barnett, Luca Del Tongo *Data Structures and Algorithms: Annotated Reference with Examples*, 2008, http://dotnetslackers.com/

[4] Clifford A. Shaffer *Data Structures and Algorithm Analysis*, 2013, http://people.cs.vt.edu/Ëśshaffer/Book/errata.html