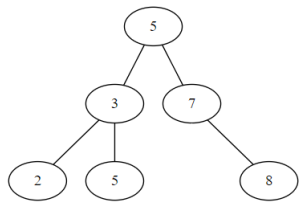


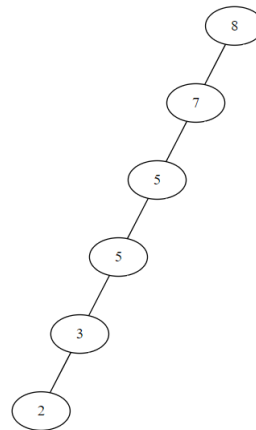
ARBORI DE CĂUTARE ECHILIBRAȚI (BALANCED SEARCH TREES)

Analiza arborilor binari de căutare

- operațiile specifice se execută în timp dependent de înălțimea arborelui (complexitate timp $O(h)$).
- în cel mai rău caz pentru n elemente înălțimea este $n - 1$ (arbore degenerat) $\Rightarrow \theta(n)$ înălțime (complexitate în caz defavorabil pentru operații).
- cazul ideal: arbore echilibrat a cărui înălțime să fie $O(\log_2 n)$.



(a) ABC echilibrat.



(b) ABC degenerat (lanț).

- ideea: la fiecare nod să păstrăm *echilibrarea*.
- când un nod își pierde *echilibrul* \Rightarrow **reechilibrare** (prin rotații specifice).
- sunt mai multe moduri de definire a echilibrării \Rightarrow variante de arbori de căutare echilibrați.
 - **arbori AVL**, arbori splay, arbori roșu-negru, B-arbori, etc.
 - caracteristică comună: înălțimea arborelui este $O(\log_2 n)$.

ARBORI AVL

Definiție 1 Un **Arbore AVL** (Adelson Velski Landis) este un ABC care satisface următoarea proprietate (**invariant AVL**):

- dacă x este un nod al AVL, atunci:
 - înălțimea subarborelui stâng al lui x diferă de înălțimea subarborelui drept al lui x cu 0, 1 sau -1 (0, 1 sau -1 se numește **factor de echilibrare**).

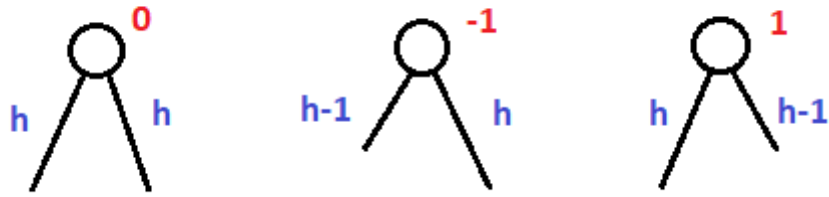


Figura 2: Factori de echilibrare posibili la orice nod al unui AVL.

- în AVL cheile (elementele) memorate în noduri sunt distincte.

Exemplu Presupunem că în container avem cheile 4 5 6 7 8 10 și relația $\mathcal{R} = \leq$. În Figura 3 sunt indicați 2 ABC care conțin aceste chei. Cel din stânga nu este AVL, pe când cel din dreapta este AVL.

- la aborele din Figura 3(a) va fi necesară o rotație în subarborele marcat, pentru a-l echilibra.

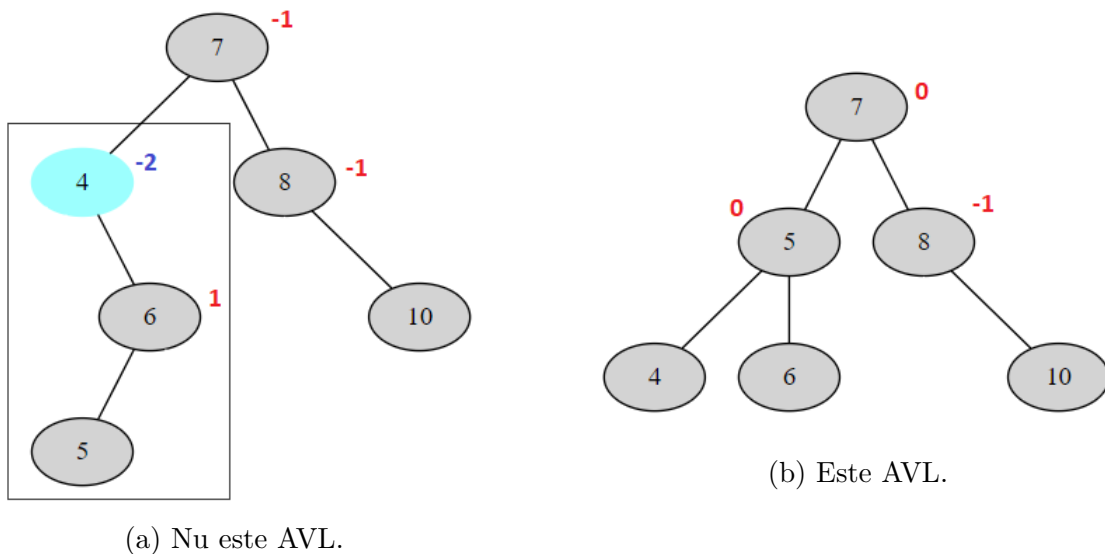


Figura 3: Arbori care conțin aceeași mulțime de chei: cel din stânga nu e echilibrat, cel din dreapta e echilibrat.

Proprietate. Înălțimea unui arbore AVL cu n noduri este $\theta(\log_2 n)$.

În cazul în care arborele are număr maxim de noduri (n), acesta este **plin** (orice nod interior are factorul de echilibrare 0). $\Rightarrow h = \theta(\log_2 n)$

- Notăm cu $N(h)$ numărul minim de noduri ale unui arbore AVL de înălțime h .
 - toate nodurile interioare au factor de echilibrare -1 sau 1.
 - înălțimea unui arbore (a cărei rădăcină este p) se poate defini recursiv ca fiind $1 + \max(\text{înălțimea subarborelui stâng și înălțimea subarborelui drept})$
 $h(p) = 1 + \max(h([p].st), h([p].dr))$.

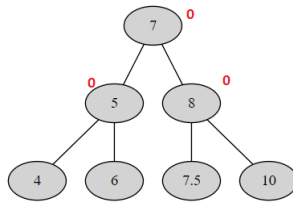


Figura 4: AVL cu număr maxim de noduri n .

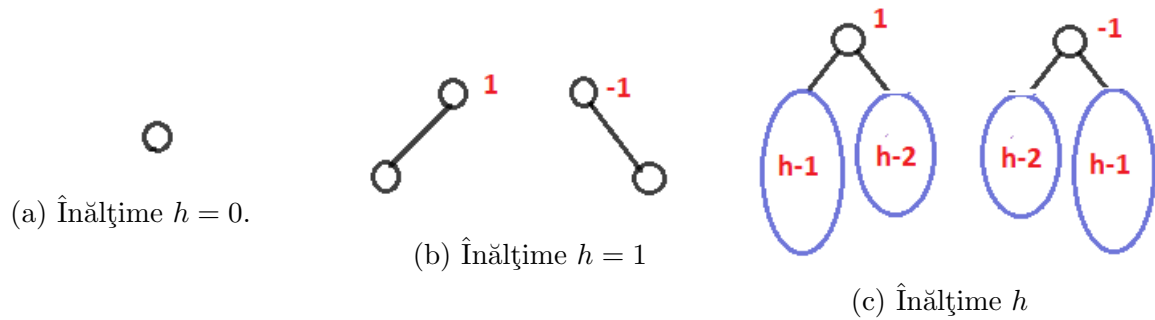


Figura 5: Număr minim de noduri în AVL

$$N(h) = \begin{cases} 1 & h = 0 \\ 2 & h = 1 \\ N(h-1) + N(h-2) + 1 & \text{altfel} \end{cases} \quad (1)$$

– se poate arăta că $N(h) \approx \phi^h$, unde $\phi = \frac{1+\sqrt{5}}{2}$ este numărul de aur (*golden ratio*), $\phi = 1.618\dots$

$$\Rightarrow h \approx \log_{\phi} n \in \theta(\log_2 n)$$

Rotații și situații de reechilibrare în AVL

- 6 situații de reechilibrare (Knuth);
 - în cazul **adăugării** unui element
 - în cazul **ștergerii** unui element
- 4 tipuri de **rotații** pentru reechilibrare:
 1. o singură rotație spre stânga (**SRS**);
 2. dublă rotație spre stânga (**DRS**);
 3. o singură rotație spre dreapta (**SRD**);
 4. dublă rotație spre dreapta (**DRD**).

Situații de reechilibrare la adăugare

Caz I - rotații spre stânga

Caz Ia) - e necesară o SRS (Figura 6)

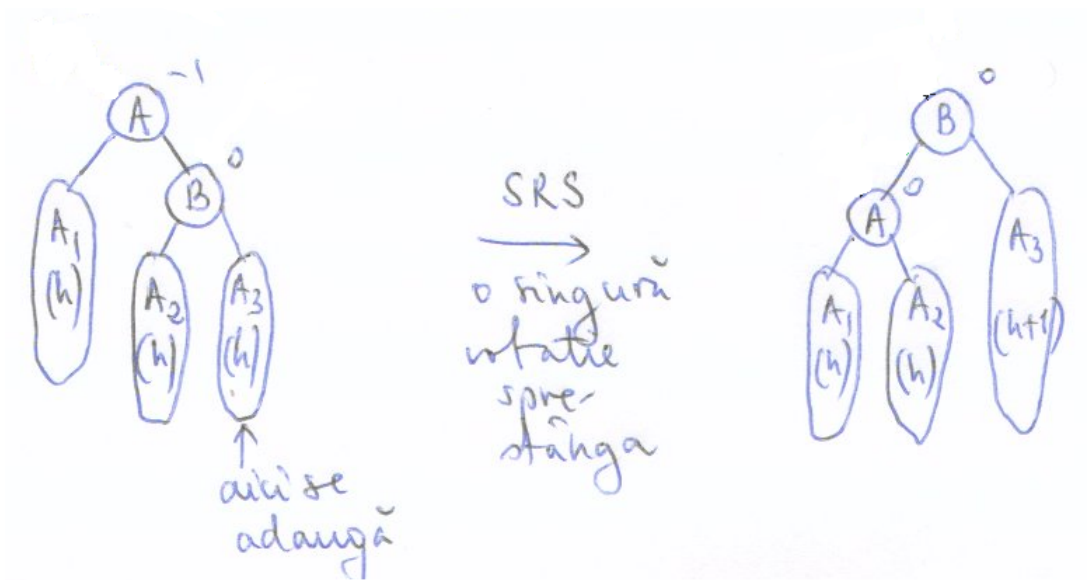


Figura 6: Caz Ia) la adăugare - e necesară o SRS pentru reechilibrare.

Exemplu caz Ia)

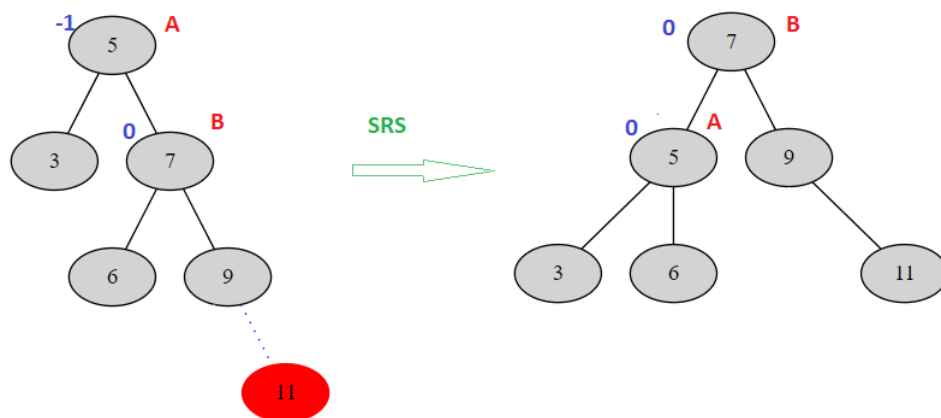


Figura 7: Exemplu caz Ia) la adăugare - e necesară o SRS pentru reechilibrare.

!!! Atenție !!!

- la inserarea unui element, rotațiile se aplică în subarboarele de înălțime minimă a cărui rădăcină și-a pierdut echilibrul (de jos în sus - de la frunze spre rădăcină)

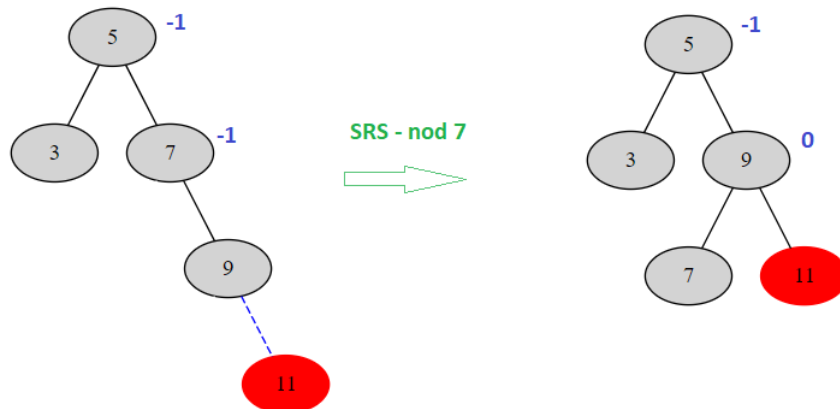


Figura 8: Echilibrul se strică la nodul 7. În subarborele de rădăcină 7 se aplică SRS pentru reechilibrare.

Implementare SRS

- pentru implementarea operațiilor, pp. în cele ce urmează reprezentare înlănțuită folosind alocare dinamică.
- pp. că fiecare nod memorează:
 - informația utilă (e);
 - adresa celor doi subarbori (stâng st și drept dr);
 - înălțimea nodului în arbore (h).

Nod

e : TElement //informația utilă nodului
 st : \uparrow Nod //adresa la care e memorat descendentul stâng
 dr : \uparrow Nod //adresa la care e memorat descendentul drept
 h : \uparrow Intreg //înălțimea nodului

Funcția $h(p)$

{complexitate timp: $\theta(1)$ }

pre: $p : \uparrow$ Nod

post: se returnează înălțimea lui p

{dacă e subarbore vid}

Dacă $p = \text{NIL}$ atunci

$h \leftarrow -1$

altfel

$h \leftarrow [p].h$

SfDacă

SfFuncția

Funcția $inaltime(p)$

{complexitate timp: $\theta(1)$ }

pre: $p : \uparrow$ Nod

post: recalculează înălțimea lui p pe baza înălțimilor subarborilor lui p

{dacă e subarbore vid}

Dacă $p = \text{NIL}$ atunci

$inaltime \leftarrow -1$

```

altfel
{se recalculează înălțimea lui  $p$  pe baza înălțimilor celor doi fii}
inaltime  $\leftarrow \max(h([p].st), h([p].dr))+1$ 
SfDaca
SfFunctia

```

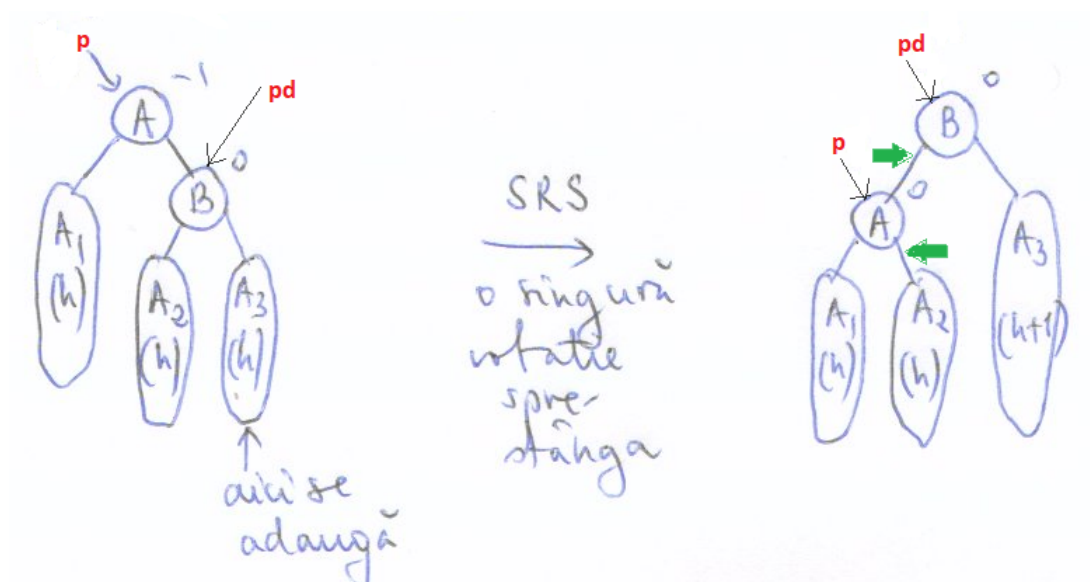


Figura 9: Situația de SRS pentru reechilibrare.

```

Functia SRS( $p$ )
{complexitate timp:  $\theta(1)$ }
pre:  $p$  este adresa unui nod;  $p:\uparrow Nod$  este rădăcina unui subarbore
post: se returnează rădăcina noului subarbore rezultat în urma unei SRS aplicate arborelui cu
rădăcina  $p$ 
{  $pd:\uparrow Nod$  e fiul drept }
 $pd \leftarrow [p].dr$ 
{ se restabilesc legăturile între noduri conform SRS }
 $[p].dr \leftarrow [pd].st$ 
 $[pd].st \leftarrow p$ 
{se recalculează înălțimile conform SRS}
 $[p].h \leftarrow \text{inaltime}(p)$ 
 $[pd].h \leftarrow \text{inaltime}(pd)$ 
 $SRS \leftarrow pd$ 
SfFunctia

```

Caz Ib) - e necesară o DRS (Figura 10)

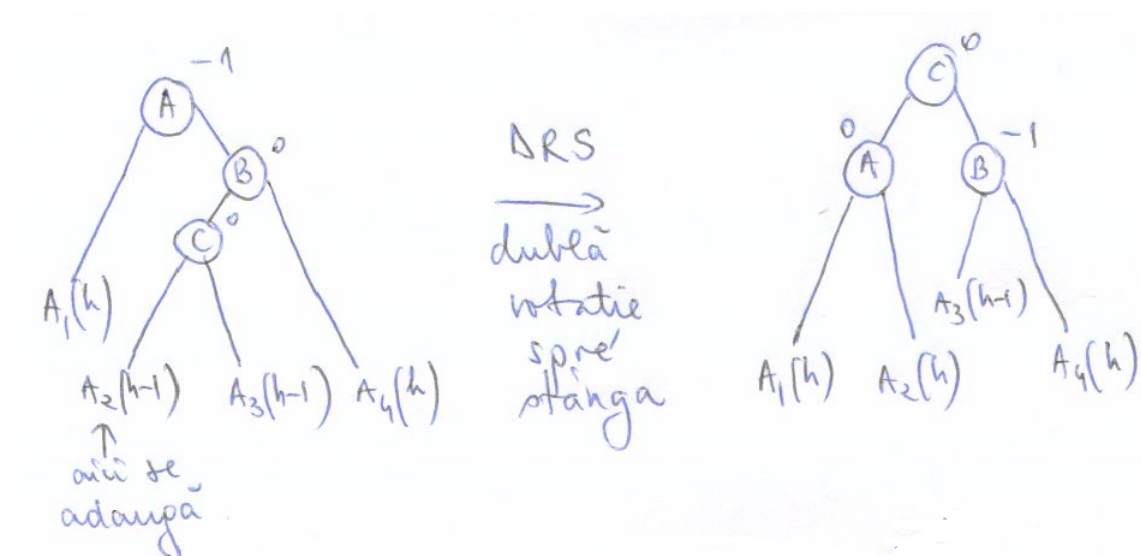


Figura 10: Caz Ib) la adăugare - e necesară o DRS pentru reechilibrare.

Exemplu caz Ib)

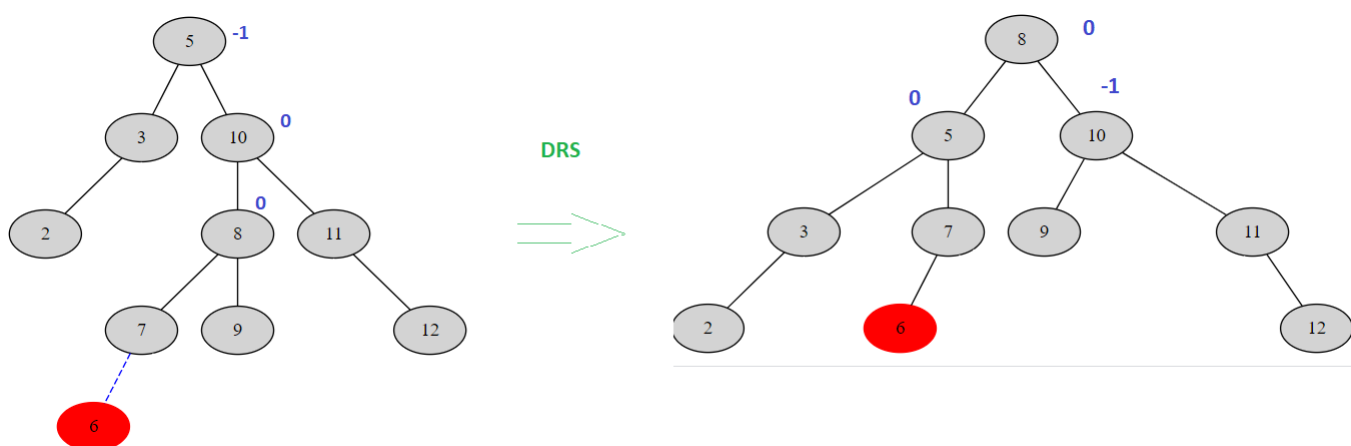


Figura 11: Exemplu caz Ib) la adăugare - e necesară o DRS pentru reechilibrare.

Caz Ic) - e necesară o DRS (Figura 12)

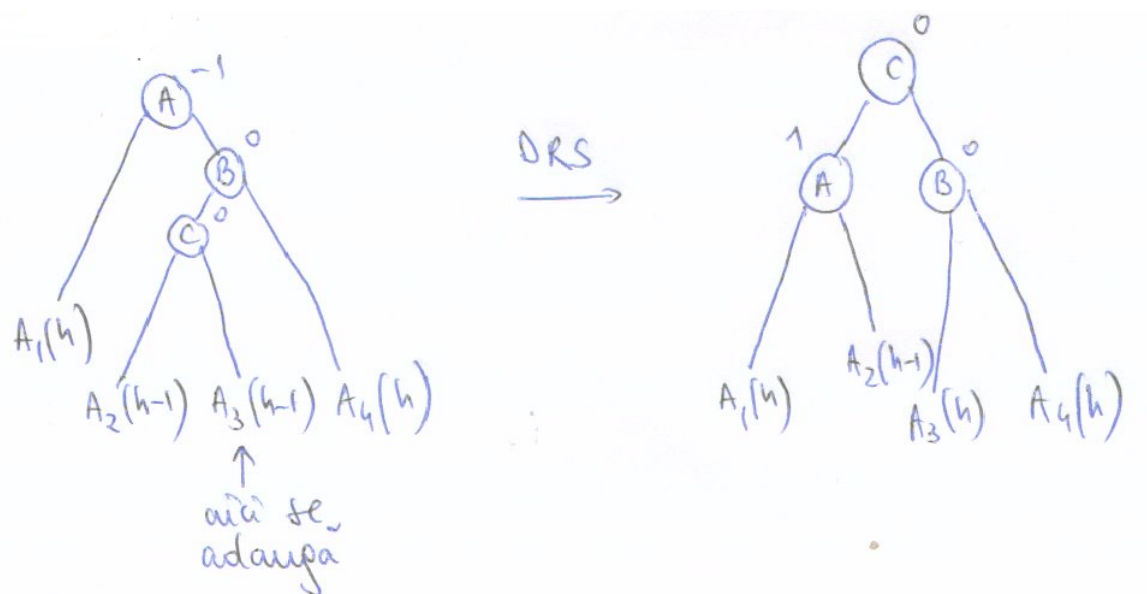


Figura 12: Caz Ic) la adăugare - e necesară o DRS pentru reechilibrare.

Exemplu caz Ic)

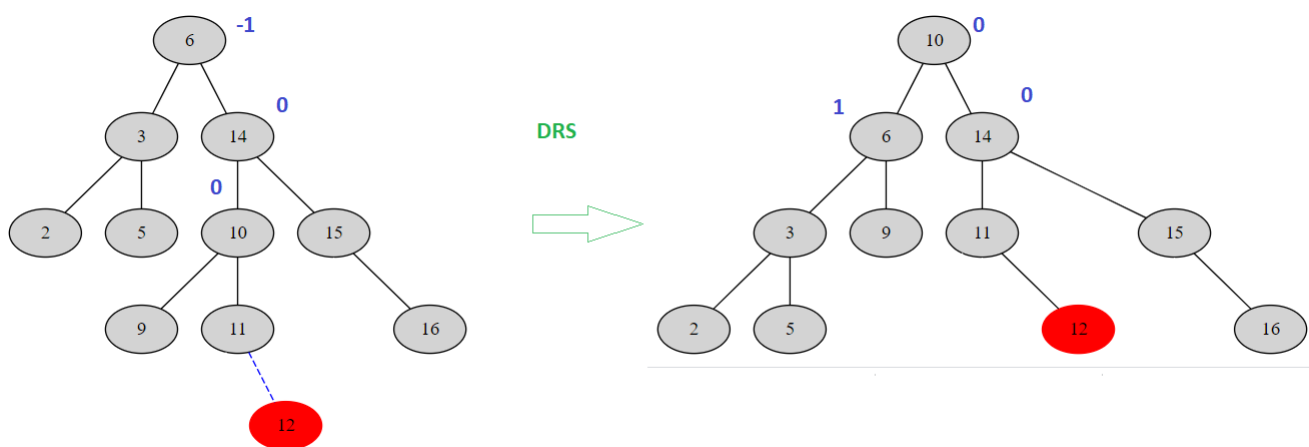


Figura 13: Exemplu caz Ic) la adăugare - e necesară o DRS pentru reechilibrare.

Caz II - rotații spre dreapta

- simetric cu cele trei situații de la cazul I
- **Caz IIa)** - e necesară o SRD (caz simetric ca cel descris în Figura 6)
- **Caz IIb)** - e necesară o DRD (caz simetric ca cel descris în Figura 10)
- **Caz IIc)** - e necesară o DRD (caz simetric ca cel descris în Figura 12)

Exemplu caz IIa)

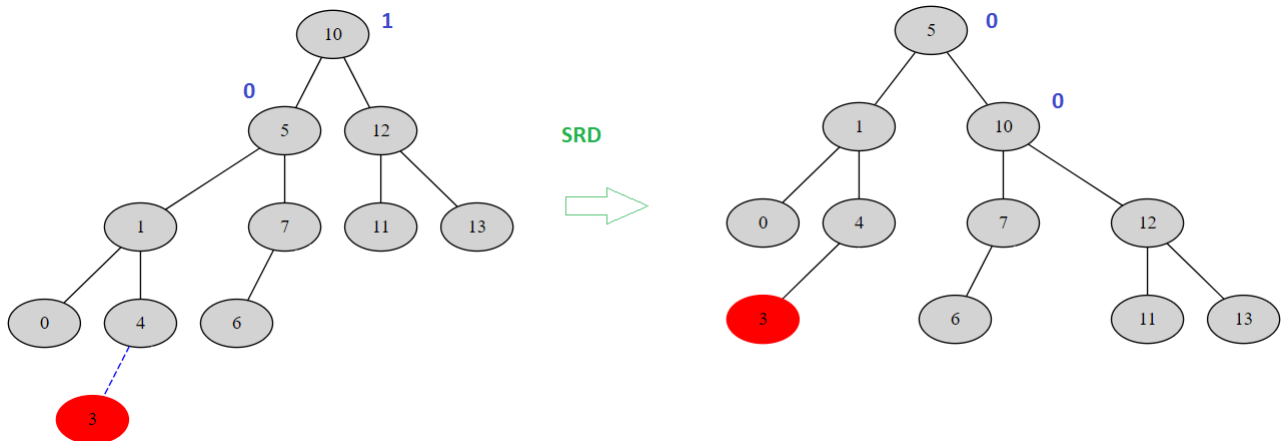


Figura 14: Exemplu caz IIa) la adăugare - e necesară o SRD pentru reechilibrare.

Exemplu caz IIc)

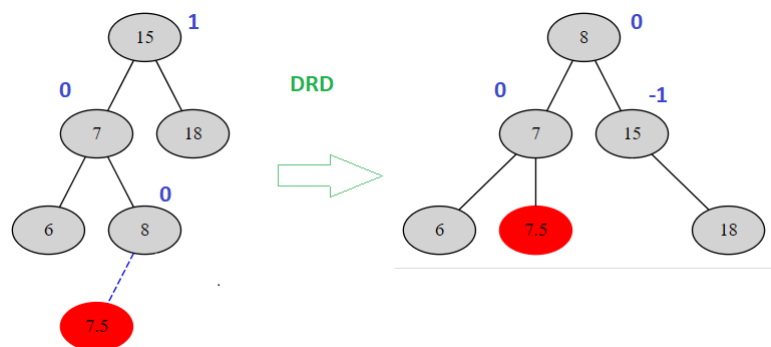


Figura 15: Exemplu caz IIc) la adăugare - e necesară o DRD pentru reechilibrare.

Operația de adăugare în arbore AVL

PP. reprezentare înlănțuită cu alocare dinamică, fiecare nod memorează și înălțimea sa.

- un element identificat de o *cheie*

Funcția creeazaNod(e) este $\{O(1)\}$
 {creeaza un nod avand informatia utila 'e' si cei doi descendenti NIL}
 aloca(p) {p: ↑Nod}
 [p].e ← e
 [p].st ← NIL
 [p].dr ← NIL
 [p].h ← 0
 creeazaNod ← p
sf creeazaNod

Funcția `adauga_rec(p, e)` este $\{ O(\log_2 n) \}$
 {se adauga informatia utila 'e' in subarborele de radacina 'p' si se returneaza noua radacina a subarborelui }

```

Daca p=NIL atunci
    p ← creeazaNod(e)
altfel
    daca e.c > [p].e.c atunci
        [p].dr ← adauga_rec([p].dr, e)
        daca h([p].dr) - h([p].st) = 2 atunci
            daca e.c > [[p].dr].e.c atunci {caz Ia}
                p ← SRS (p)
            altfel {caz Ib, Ic}
                p ← DRS (p)
        sfDaca
        altfel
            [p].h ← inaltime(p)
        sfDaca
    altfel
        daca e.c < [p].e.c atunci
            @ simetric pe partea stanga – rotatii spre dreapta
        altfel
            @ cheie duplicat – nu e permisa in AVL
        sfDaca
    sfDaca
sfDaca
adauga_rec ← p
sf_adauga_rec

```

Subalgoritm `adauga(ab, e)` este $\{ O(\log_2 n) \}$
 {se adauga informatia utila 'e' in arborele 'ab' si se returneaza arborele rezultat}
 ab.rad ← adauga_rec(ab.rad, e)
sf_adauga

Observații

- alte reprezentări posibile pentru arborele AVL (ca și pentru un ABC)
 - reprezentare înlănțuită cu reprezentare înlănțuiri pe tablou.
 - reprezentare secvențială, folosind ca schemă de memorare un ansamblu.
- în locul înălțimii fiecărui nod, se poate memora *factorul de echilibrare* al acestuia

Situații de reechilibrare la ștergere

Caz I - rotații spre stânga

Caz Ia) și Ib) - dacă prin ștergere din **A1**, înălțimea devine $h-1$, e necesară o SRS (Figura 16)

- e posibil ca prin ștergere din A1, înălțimea să rămână $h \Rightarrow$ nu e necesară rotație

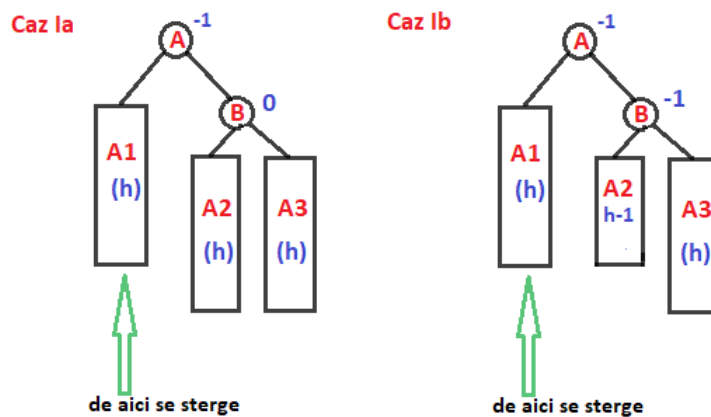


Figura 16: Caz Ia) și Ib) la ștergere - e necesară o SRS pentru reechilibrare.

Exemple caz Ia) și caz Ib) în care sunt necesare rotații

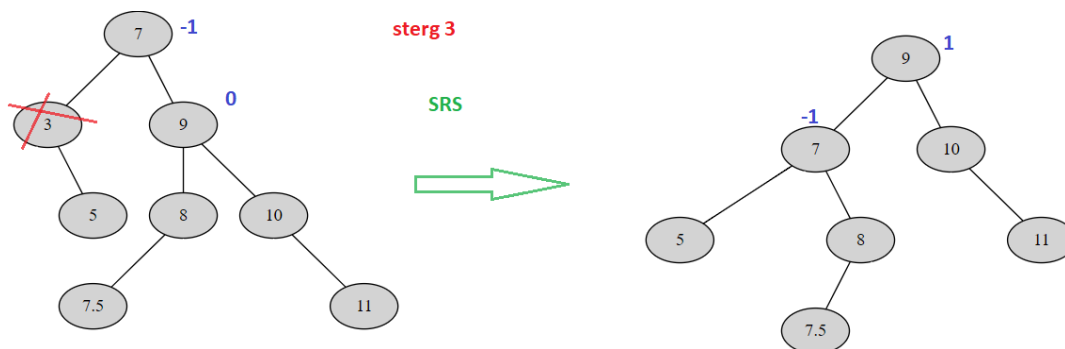


Figura 17: Exemplu caz Ia) la ștergere - e necesară o SRS pentru reechilibrare.

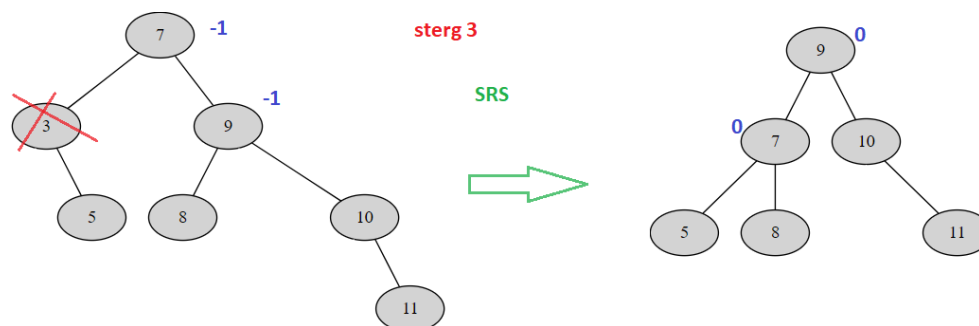


Figura 18: Exemplu caz Ib) la ștergere - e necesară o SRS pentru reechilibrare.

Exemplu caz Ia) în care nu e necesară rotație

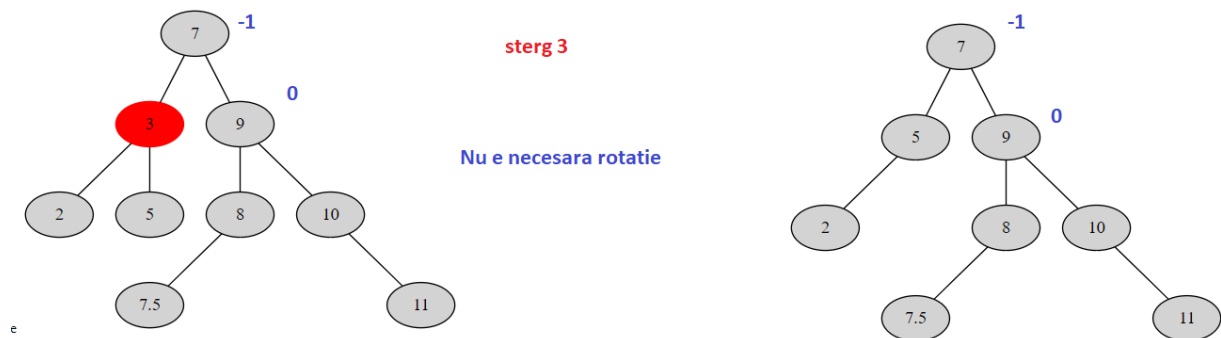


Figura 19: Exemplu caz Ia) la ștergere - nu e necesară rotație.

Caz Ic) - dacă prin ștergere din **A1**, înălțimea devine $h-1$, e necesară o DRS (Figura 20)

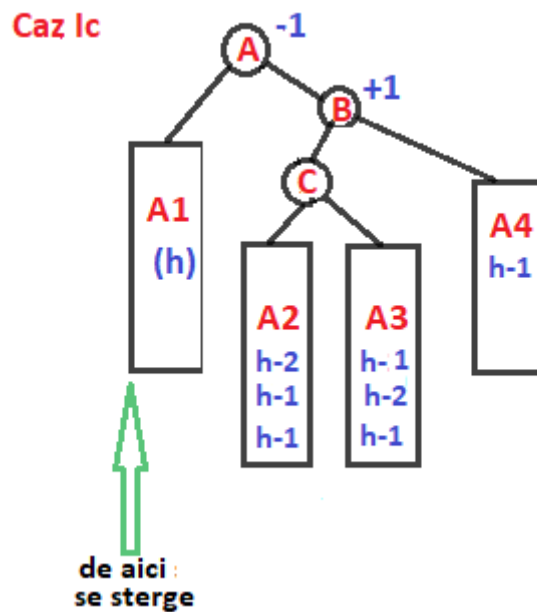


Figura 20: Caz Ic) la ștergere - e necesară o DRS pentru reechilibrare.

Exemplu caz Ic)

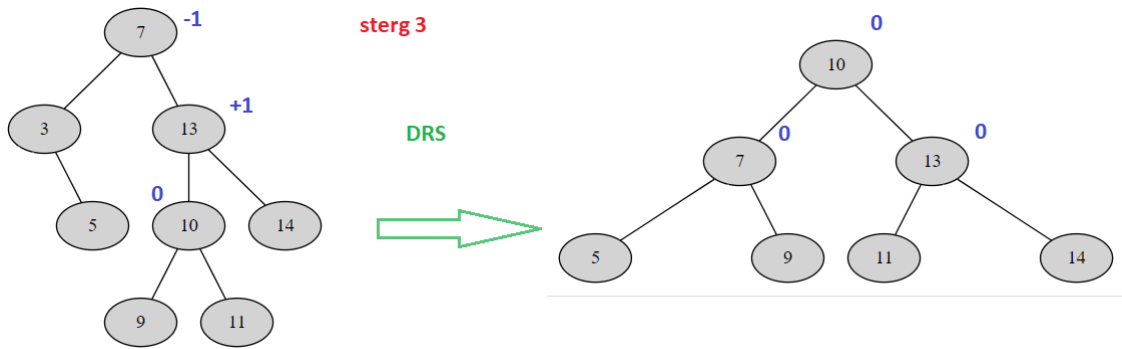


Figura 21: Exemplu caz Ic) la ștergere - e necesară o DRS pentru reechilibrare.

Caz II - rotații spre dreapta

- simetric cu Ia)

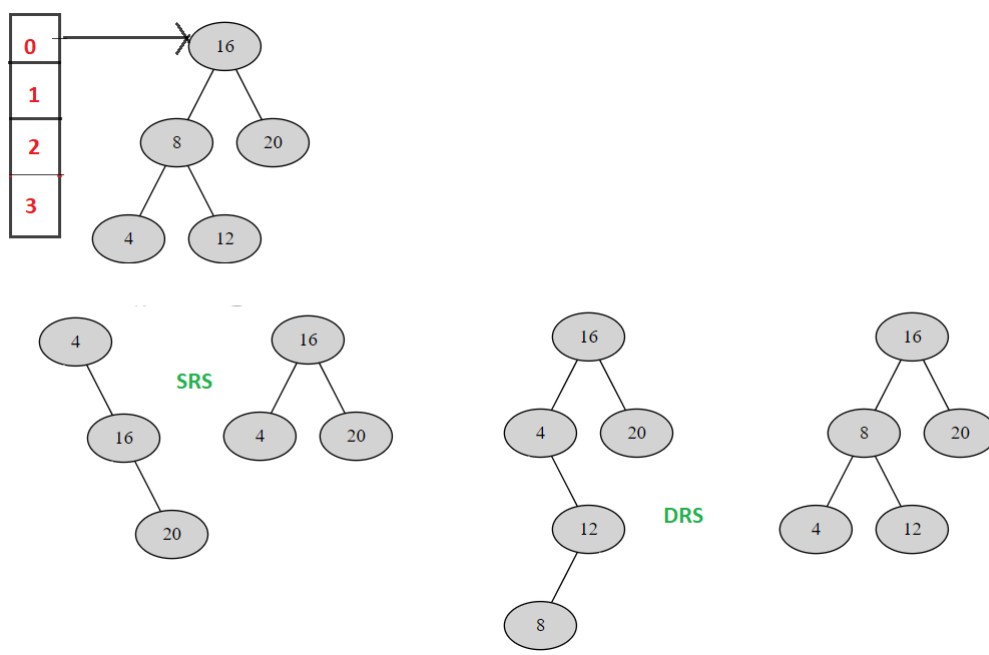
Observație

Tabelele de dispersie cu rezolvare coliziuni prin liste independente își pot memora listele folosind arbori AVL.

- se reduce complexitatea timp defavorabil la căutare de la $\theta(n)$ la $\theta(\log_2 n)$.

Fie $m = 4$ și funcția de dispersie prin divizare.

$c(\text{heie})$	4	5	16	9	20	7	12	8
$c \bmod m$	0	1	0	1	0	3	0	0



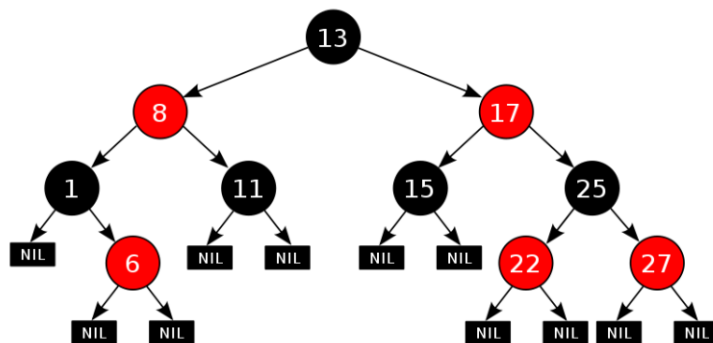
Probleme

1. Descrieți în Pseudocod următoarele rotații: DRS, SRD, DRD.
2. Dați exemple concrete în care apare necesitatea următoarelor tipuri de rotații la adăugare/ștergere: SRS, SRD, DRS, DRD.
3. Reprezentări alternative pentru AB, ABC, AVL folosind următoarele reprezentări:
 - reprezentare înlănțuită cu reprezentare înlănțuiri pe tablou.
 - reprezentare secvențială, folosind ca schemă de memorare un ansamblu.

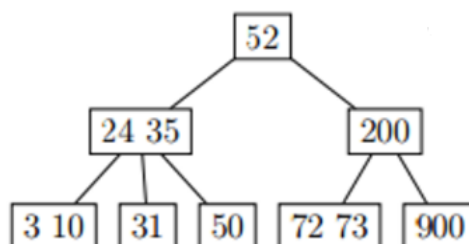
Alte tipuri de ABC echilibrați

Înălțimea este $O(\log_2 n)$

- arbori roșu-negru (*red-black trees*)
 - sunt ABC
 - nodurile au o culoare: *roșie* sau *neagră*
 - frunzele (**nil**) sunt negre
 - un nod roșu are cei doi fii de culoare neagră
 - pe orice drum de la rădăcină la o frunză, numărul nodurilor negre este același
 - * în urma constrângerii legate de modul în care se colorează nodurile \Rightarrow
 - niciun drum de la radacina la o frunza nu este mai lung decât dublul lungimii oricarui alt drum \Rightarrow arborele este aproximativ echilibrat.
 - $h < \log_2 n + 1$



- B-arbori



- generalizare ABC
- fiecare nod interior conține mai multe chei
- nodurile pot avea mai mult de 2 descendenți
 - * dacă un nod conține 2 chei c_1 și c_2 , atunci are 3 descendenți
- folosiți în *baze de date* și *sisteme de fișiere*