AVL

-algoritmi, explicatii-

AVL :

- ABC care satisfice conditia:
    - inaltimea subarborelui stang al unui nod difera de inaltimea subarborelui drept al acestuia cu 0, 1 sau -1 (0, 1 sau -1 se numeste factor de echilibrare).
- Inaltimea lui este $\log_2 n$
- Se reechilibreaza in cazul adaugarii si a stergerii unui element
- Tipuri de rotatie folosite pentru reechilibrare:
    - O singura rotatie spre stanga (SRS)
    - Dubla rotatie spre stanga (DRS)
    - O singura rotatie spre dreapta (SRD)
    - Dubla rotatie spre dreapta (DRD)
- Relatie predefinita in acest document : ≤

Reprezentare inlantuita:

Nod

e: TElement // informatia utila nodului, are in componenta sa un camp c de tip TCheie

st: ↑ Nod // adresa la care e memorat descendentul stang

dr: ↑ Nod // adresa la care e memorat descendentul drept

h: Intreg // inaltimea nodului

AVL:

rad: ↑ Nod // radacina arborelui

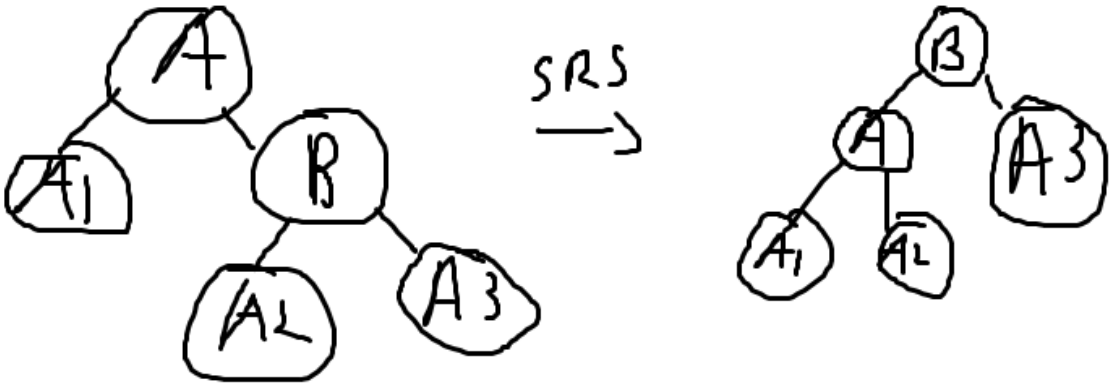
{pre: e : TElement}

{post: creeaza un nod avand informatia utila e si cei doi descendenti NIL}

{complexitate timp: theta(1)}

Functie CreeazaNod(e)

```
        aloca(p)
        [p].e ← e
        [p].st ← NIL
        [p].dr ← NIL
        [p].h ← 0
SfFunctie

{pre: p : ↑ Nod}
{post: se returneaza inaltimea lui p}
{complexitate timp: theta(1)}

Functie h(p)
        daca p =NIL atunci
                h ← -1
        altfel
                h ← [p].h
        SfDaca
SfFunctie



{pre: p : ↑ Nod}
{post: recalculeaza inaltimea lui p pe baza inaltimilor subarborilor lui p}
{complexitate timp: theta(1)}

Functie inaltime(p)
        daca p =NIL atunci

                inaltime ← -1
        altfel
                inaltime ← max(h([p].st), h([p].dr)) + 1
        SfDaca
SfFunctie


SRS
```



{pre: p este adresa unui nod; p : ↑ Nod este radacina unui subarbore}

{post: se returneaza radacina noului subarbore rezultat in urma unei SRS aplicate arborelui cu radacina p}

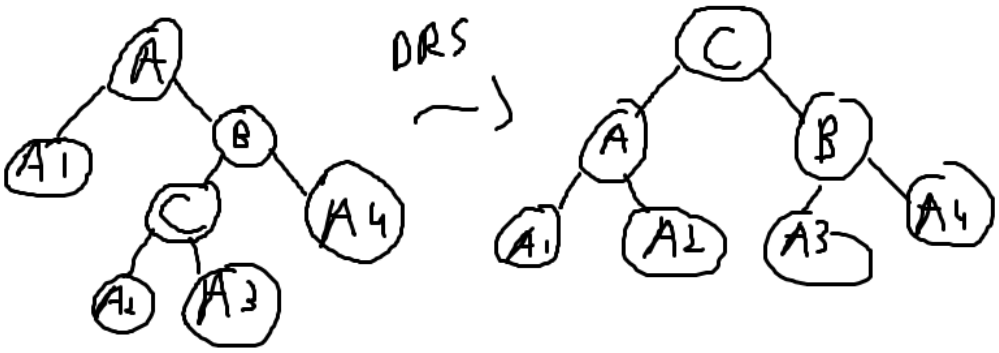
{complexitate timp: theta(1)}

```
Functie SRS(p)
        a ← p
        b ← [p].dr
        [a].dr ← [b].st
        [b].st ← a
        [a].h ← inaltime(a)
        [b].h ← inaltime(b)
        SRS ← b
SfFunctie
```

In curs (nu e intuitiv si nu ne place):

```
Functie SRS(p)
        pd ← [p].dr
        [p].dr ← [pd].st
        [pd].st ← p
        [p].h ← inaltime(p)
        [pd].h ← inaltime(pd)
        SRS ← pd
SfFunctie
```

DRS



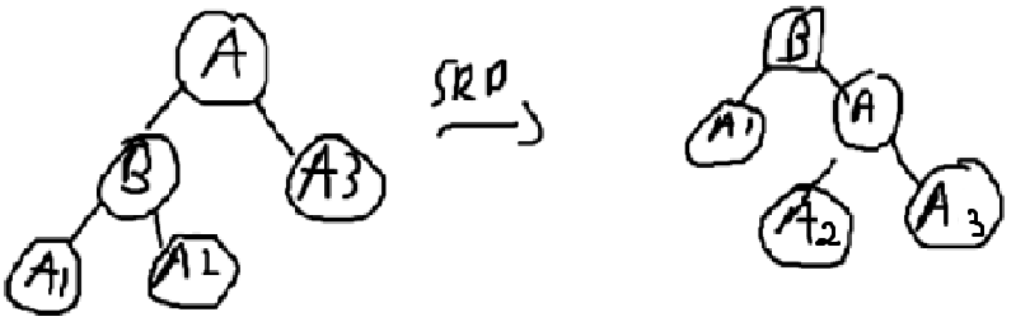
{pre: p este adresa unui nod; p : ↑ Nod este radacina unui subarbore}

{post: se returneaza radacina noului subarbore rezultat in urma unei DRS aplicate arborelui cu radacina p}

{complexitate timp: theta(1)}

```
Functie DRS(p)
        a ← p
        b ← [p].dr
        c ← [b].st
        [a].dr ← [c].st
        [b].st ← [c].dr
        [c].st ← a
        [c].dr ← b
        [a].h ← inaltime(a)
        [b].h ← inaltime(b)
        [c].h ← inaltime(c)
        DRS ← c
SfFunctie
```

SRD



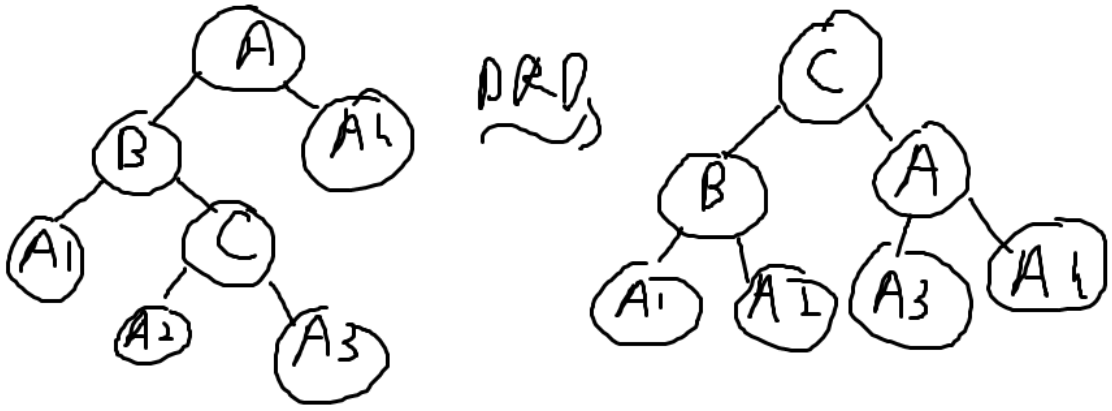
{pre: p este adresa unui nod; p : ↑ Nod este radacina unui subarbore}

{post: se returneaza radacina noului subarbore rezultat in urma unei SRD aplicate arborelui cu radacina p}

{complexitate timp: theta(1)}

```
Functie SRD(p)
        a ← p
        b ← [p].st
        [a].st ← [b].dr
        [b].dr ← a
        [a].h ← inaltime(a)
        [b].h ← inaltime(b)
        SRD ← b
SfFunctie
```

DRD



{pre: p este adresa unui nod; p : ↑ Nod este radacina unui subarbore}

{post: se returneaza radacina noului subarbore rezultat in urma unei DRD aplicate arborelui cu radacina p}

{complexitate timp: theta(1)}

```
Functie DRD(p)
        a ← p
        b ← [p].st
        c ← [b].dr
        [b].dr ← [c].st
        [a].st ← [c].dr
        [c].st ←b
        [c].dr ←c
        [a].h ← inaltime(a)
        [b].h ← inaltime(b)
        [c].h ← inaltime(c)
        DRD ← c
SfFunctie
```

Adaugarea unui nod in AVL

{pre: p este adresa unui nod; p : ↑ Nod este radacina unui subarbore, e : TElement}

{post: se adauga informatia utila e in subarborele de radacina p si se returneaza noua radacina a subarborelui}
{complexitate timp: $O(\log_2 n)$}

```
Functie adauga_rec(p, e)
        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
                                        p ← SRS(p)
                                altfel
                                        p ← DRS(p)
                                SfDaca
                        altfel
                                [p].h ← inaltime(p)
                        SfDaca
                altfel
                        daca e.c < p[e].c atunci
                                [p].st ← adauga_rec([p].st, e)
                                daca h([p].st) - h([p].dr) = 2 atunci
                                        Daca e.c > [[p].st].e.c atunci
                                                p ← DRD(p)
                                        altfel
                                                p ← SRD(p)
                                        sfDaca
                                altfel
                                        [p].h ← inaltime(p)
                                sfDaca
                        altfel
```

```
                    @cheie duplicat – nu e permisa in AVL
                SfDaca
        SfDaca
        adauga_rec ← p
SfFunctie

{pre: ab este un arbore, e : TElement}
{post: se adauga informatia utila e in arborele ab si se returneaza arborele rezultat}
{complexitate timp: O(log2n)}

Subalgoritm adauga (ab, e)
        ab.rad ← adauga_rec(ab.rad, e)
SfSubalgoritm
```

Reprezentare pe tablou:

AVL:

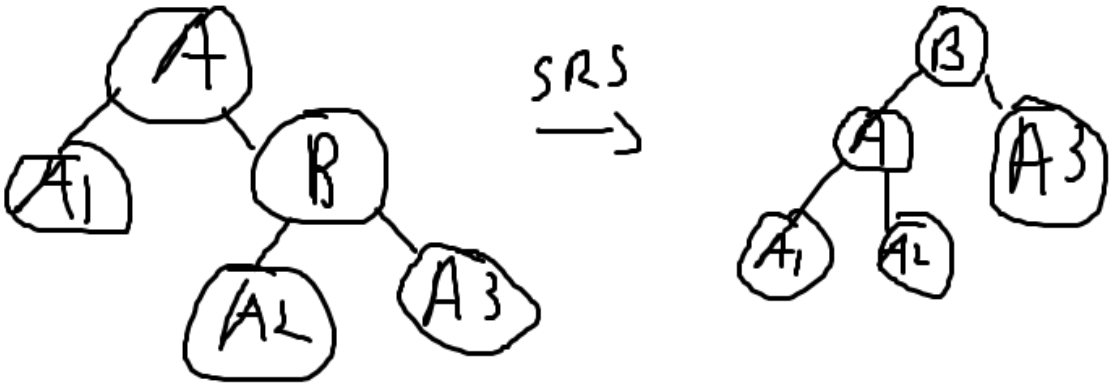
```
        elems: TElement[]   // informatia utila, are in componenta sa un camp c de tip TCheie
        st: Intreg[]   // vector de pozitii pentru subarborele stang
        dr: Intreg[]   // vector de pozitii pentru subarborele drept
        h: Intreg[]    // vector de inaltimi pentru fiecare element
        rad: Intreg    // pozitia radacinii

{pre: ab este arbore, p este pozitia radacinii unui subarbore}
{post: se returneaza inaltimea lui p}
{complexitate timp: theta(1)}

Functie h(ab, p)
        daca p = -1 atunci
                h ← -1
        altfel
```

```
            h ← ab.h[p]
        SfDaca
SfFunctie
```

{pre: ab este arbore, p este pozitia radacinii unui subarbore}
{post: recalculeaza inaltimea lui p pe baza inaltimilor subarborilor lui p}
{complexitate timp: theta(1)}

```
Functie inaltime(ab, p)
        daca p = -1 atunci
            inaltime ← -1
        altfel
            inaltime ← max(ab, ab.st[p]), h(ab, ab.dr[p])) +1
        SfDaca
SfFunctie
```

SRS



{Pre: ab este arbore, p este pozitia radacinii unui subarbore}
{Post: se returneaza pozitia radacinii noului subarbore rezultat in urma unei SRS aplicate arborelui cu radacina pe pozitia p}
{complexitate timp: theta(1)}

```
Functie SRD(ab, p)
```

a ← p

b ← ab.dr[a]

ab.dr[a] ← ab.st[b]

ab.st[b] ← a

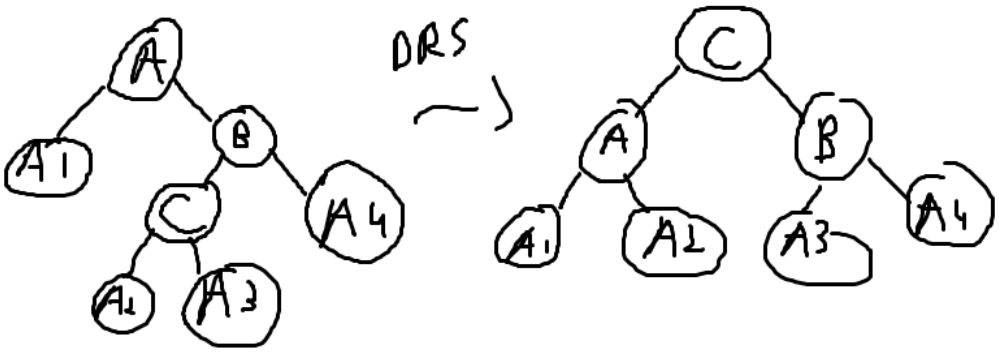
ab.h[a] ← inaltime(ab, a)

ab.h[b] ← inaltime(ab, b)

SRD ← b

SfFunctie

DRS



{Pre: ab este arbore, p este pozitia radacinii unui subarbore}

{Post: se returneaza pozitia radacinii noului subarbore rezultat in urma unei DRS aplicate arborelui cu radacina pe pozitia p}

{complexitate timp: theta(1)}

Functie DRS(ab, p)

a ← p

b ← ab.dr[a]

c ← ab.st[b]

ab.dr[a] ← ab.st[c]

ab.st[b] ← ab.dr[c]

ab.st[c] ← a

ab.dr[c] ← b

ab.h[a] ← inaltime(ab, a)

ab.h[b] ← inaltime(ab, b)

ab.h[c] ← inaltime(ab, c)

DRS ← c

SfFunctie

SRD

{Pre: ab este arbore, p este pozitia radacinii unui subarbore}

{Post: se returneaza pozitia radacinii noului subarbore rezultat in urma unei SRD aplicate arborelui cu radacina pe pozitia p}

{complexitate timp: theta(1)}

Functie SRD(ab, p)

a ← p

b ← ab.st[a]

ab.st[a] ← ab.dr[b]

ab.dr[b] ← a

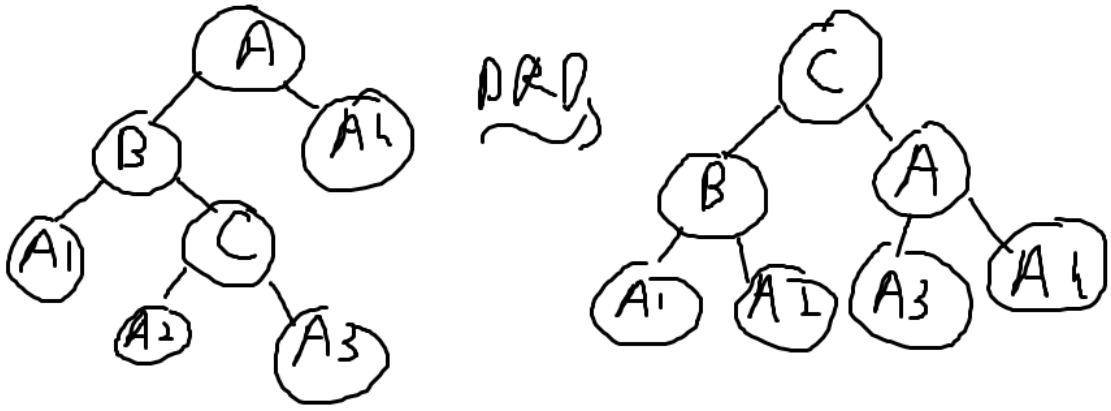
ab.h[a] ← inaltime(ab, a)

ab.h[b] ← inaltime(ab, b)

SRD ← b

SfFunctie

DRD



{Pre: ab este arbore, p este pozitia radacinii unui subarbore}

{Post: se returneaza pozitia radacinii noului subarbore rezultat in urma unei DRD aplicate arborelui cu radacina pe pozitia p}

{complexitate timp: theta(1)}

```
Functie DRD(ab, p)
        a ← p
        b ← ab.st[a]
        c ← ab.dr[b]
        ab.st[a] ← ab.dr[c]
        ab.dr[b] ← ab.st[c]
        ab.st[c] ← b
        ab.dr[c] ← a
        ab.h[a] ← inaltime(ab, a)
        ab.h[b] ← inaltime(ab, b)
        ab.h[c] ← inaltime(ab, c)
        DRD ← c
SfFunctie
```

Adaugarea unui element in AVL

{pre: ab este un arbore, p este pozitia radacinii subarborelui pe care il analizam curent, e : TElement }

{post: se adauga informatia utila e in subarborele de radacina p si se returneaza noua radacina a subarborelui}
{complexitate timp: $O(\log_2 n)$}

```
Functie adauga_rec(ab, p, e)
        daca ab.elems[p] = -1 atunci
                ab.elems[p] ← e
        altfel
                daca e.c > ab.elems[p].c atunci
                        ab.dr[p] ← adauga_rec(ab, ab.dr[p], e)
                        daca h(ab, ab.dr[p]) - h(ab, ab.st[p]) = 2 atunci
                                daca e.c > ab.elems[ab.dr[p]].c atunci
                                        p ← SRS(ab, p)
                                altfel
                                        p ← DRS(ab, p)
                                SfDaca
                        altfel
                                ab.h[p] ← inaltime(ab, p)
                        SfDaca
                altfel
                        daca e.c < ab.elems[p].c atunci
                                ab.st[p] ← adauga_rec(ab, ab.st[p], e)
                                daca h(ab, ab.st[p]) - h(ab, ab.dr[p]) = 2 atunci
                                        Daca e.c > ab.elems[ab.st[p]].c atunci
                                                p ← DRD(ab, p)
                                        altfel
                                                p ← SRD(ab, p)
                                        sfDaca
                                altfel
                                        ab.h[p] ← inaltime(ab, p)
                                sfDaca
```

```
                altfel
                    @cheie duplicat – nu e permisa in AVL
                SfDaca
        SfDaca
        adauga_rec ← p
SfFunctie
```

{pre: ab este un arbore, e : TElement}
{post: se adauga informatia utila e in arborele ab si se returneaza pozitia in care s-a adaugat elementul}
{complexitate timp: $O(\log_2 n)$}

```
Subalgoritm adauga (ab, e)
        ab.rad ← adauga_rec(ab, ab.rad, e)
SfSubalgoritm
```