8.3. Arbori binari ordonaţi

8.3.1. Definiţii

- Structura **arbore binar** poate fi utilizată pentru a reprezenta în mod convenabil o mulţime de elemente, în care elementele se regăsesc după o **cheie unică**.

  - Se **presupune** că avem o mulţime de `n` noduri definite ca articole, fiecare având câte o cheie care este număr întreg.

  - Dacă cele `n` articole se **organizează** într-o structură **listă liniară**, căutarea unei chei necesită în medie `n/2` comparaţii.

  - După cum se va vedea în continuare, **organizarea** celor `n` articole într-o **structură arbore binar convenabilă**, reduce numărul de căutări la maximum $\log_2 n$.

  - Acest lucru devine posibil utilizând structura **arbore binar ordonat.**

- Prin **arbore binar ordonat** se înţelege un **arbore binar** care are proprietatea că, parcurgând nodurile sale în **inordine**, secvenţa cheilor este **monoton crescătoare**.

- Un **arbore binar ordonat** se bucură şi de următoarea **proprietate**:

  - **Dacă** `n` este un nod oarecare al arborelui, având cheia `c`, **atunci:**

    - Toate nodurile din **subarborele stâng** a lui `n` **au cheile mai mici sau egale** cu `c`

    - Toate nodurile din **subarborele drept** al lui `n` **au chei mai mari sau egale** cu `c`.

- De aici rezultă un **procedeu de căutare** foarte simplu:

  - Începând cu rădăcina, se trece la fiul **stâng** sau la fiul **drept**, după cum cheia căutată este mai **mică** sau mai **mare** decât cea a nodului curent.

- Numărul **comparaţiilor de chei** efectuate în cadrul acestui procedeu este cel mult egal cu **înălţimea arborelui**.

- Din acest motiv aceşti arbori sunt cunoscuţi şi sub denumirea de **arbori binari de căutare** (“**Binary Search Trees**”).

- În general înălţimea unui arbore **nu** este determinată de **numărul** nodurilor sale.

  - Spre exemplu cu cele 9 noduri precizate în fig.8.3.1.a se poate construi atât arborele ordonat (a) de înălţime 4 cât şi arborele ordonat (b) de înălţime 6.

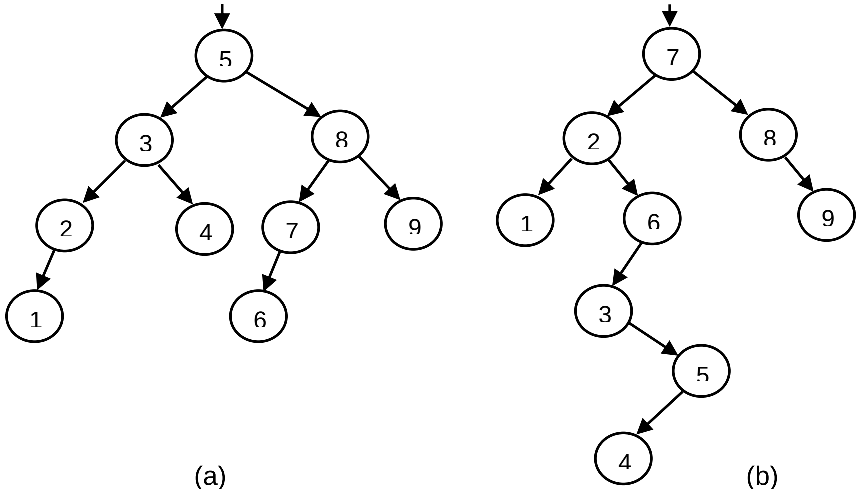


Fig.8.3.1.a. Arbori binari ordonaţi de diferite înălţimi

- Este simplu de observat că un arbore are înălţimea **minimă** dacă **fiecare** nivel al său conţine **numărul maxim de noduri**, cu excepţia posibilă a ultimului nivel.

- Deoarece numărul maxim de noduri al nivelului `i` este 2^{i-1}, rezultă că **înălţimea minimă** a unui arbore binar cu `n` noduri este:

$$h_{\min} = \lceil \log_2(n+1) \rceil$$

- Prin aceasta se justifică şi afirmaţia că o căutare într-un **arbore binar ordonat** necesită aproximativ **$\log_2 n$ comparaţii de chei**

    - Se precizează însă, că această afirmaţie este valabilă în **ipoteza** că nodurile sunt organizate într-o **structură arbore binar ordonat de înălţime minimă.**

- Dacă această condiţie **nu** este satisfăcută, **eficienţa** procesului de căutare poate fi mult redusă, în cazul cel mai defavorabil arborele degenerând într-o structură de **listă liniară.**

- Aceasta se întâmplă când subarborele drept (stâng) al tuturor nodurilor este **vid**, caz în care înălţimea arborelui devine egală cu `n`, iar căutarea **nu** este mai eficientă decât căutarea într-o **listă liniară** (O *(n/2)*).

- O altă proprietate importantă a ABO este aceea ca traversând un ABO **in inordine** se obţine **secvenţa ordonată crescător** a cheilor nodurilor arborelui.

- Traversarea în **inordine** a unui arbore binar se determină simplu **proiectând nodurile** structurii arbore pe **axa absciselor**. Secvenţa rezultată este ordonarea în inordine a nodurilor structurii.

8.3.2. Tipul de date abstract arbore binar ordonat

- Într-o manieră similară celei în care au fost definite **tipurile de date abstracte** pe parcursul acestui curs, şi în cazul **arborilor binari ordonaţi** se poate defini un astfel de **tip**.

- Acesta presupune desigur:

    - (1) Definirea **modelului matematic** asociat.

    - (2) Precizarea **setului de operatori**.

- Ca şi pentru celelalte structuri studiate şi în acest caz este greu de definit un set de operatori general valabil.

- Din mulţimea seturilor posibile se propune setul prezentat în [8.3.2.a].

<u>TDA Arbore Binar Ordonat (ABO)</u>

Modelul matematic: este un arbore binar, fiecare nod având asociată o cheie specifică. Pentru fiecare nod al arborelui este valabilă următoarea proprietate: cheia nodului respectiv este mai mare decât cheia oricărui nod al subarborelui său stâng şi mai mică decat cheia oricărui nod al subarborelui său drept.

Notatii:

TipCheie - tipul cheii asociate structurii nodului
TipElement - tipul asociat structurii unui nod
RefTipNod - referinţa la un nod al structurii
TipABO - tipul arbore binar ordonat
b: TipABO;
x,k: TipCheie;
e: TipElement;
p: RefTipNod;

[8.3.2.a]

Operatori:

1. ***Creaza****(b: TipABO);* - procedură care crează arborele binar vid *b*;

2. ***Cauta****(x: TipCheie, b: TipABO): RefTipNod;* - operator funcţie care caută în arborele *b* un nod având cheia identică cu *x* returnând referinţa la nodul în cauză respectiv indicatorul vid dacă un astfel de nod nu există;

3. ***Actualizeaza****(e: TipElement, b: TipABO);* - caută nodul din arborele *b* care are aceeaşi cheie cu nodul *e* şi îi modifică conţinutul memorând pe *e* în acest nod. Dacă un astfel de nod nu există, operatorul nu realizează nici o acţiune;

4. ***Insereaza****(e: TipElement, b: TipABO);* - inserează elementul *e* în arborele *b* astfel încât acesta să rămână un ABO;

5. ***SuprimaMin****(b: TipABO, e: TipElement);* - extrage nodul cu cheia minimă din arborele cu rădacina *b* şi îl

returnează în *e*. În urma suprimării arborele *b* rămâne un ABO;

6. ***Suprima****(x: TipCheie, b: TipABO);* - suprimă nodul cu cheia *x* din arborele *b*, astfel încât arborele să rămână ordonat. Dacă nu există un astfel de nod, procedura nu realizeză nimic.

8.3.3. Tehnici de căutare în arbori binari ordonaţi

- **Specificarea problemei**:

  - Fie `b` o referinţă care indică rădăcina unui **arbore binar ordonat**, ale cărui noduri au structura definită în [8.3.3.a].

  - Fie `x` un număr întreg dat.

  - Se cere să se găsească în arborele binar ordonat `b` acel nod care are cheia egală cu `x`.

- Funcţia **Cauta**`(x,b)` precizată în secvenţa [8.3.3.b] rezolvă această problemă

  - Căutarea se realizează în conformitate cu procedeul descris în paragraful anterior.

  - Funcţia **Cauta** returnează valoarea **NIL** dacă **nu** găseşte nici un nod cu cheia `x`, altminteri valoarea ei este egală cu pointerul care indică acest nod.

```
------------------------------------------------------------
{Structura de date Arbore Binar Ordonat}

TYPE RefTipNod=^TipNod;
     TipNod=RECORD
              cheie: TipCheie;                  [8.3.3.a]
              stang,drept: RefTipNod;
            END;
------------------------------------------------------------
{Căutare în ABO (Varianta iterativă)}

FUNCTION Cauta(x:TipCheie; VAR b:TipABO):RefTipNod;
  VAR gasit:boolean;
  BEGIN
    gasit:=false;
    WHILE(b<>NIL) AND NOT gasit DO              [8.3.3.b]
      BEGIN
        IF b^.cheie=x THEN gasit:=true ELSE
        IF x<b^.cheie THEN b:=b^.stang ELSE
          b:=b^.drept
      END;
    Cauta:=b
  END; {Cauta}
------------------------------------------------------------
```

* Acelaşi proces de căutare poate fi implementat şi în **variantă recursivă** ţinând cont de faptul ca arborele binar este definit ca şi o **structură de date recursivă**.

* Varianta recursivă a căutării apare în secvenţa [8.3.3.c].

    * Se face însă precizarea că această implementare este **mai puţin performantă** deoarece principial căutarea în arborii binari ordonaţi este o operaţie pur secvenţială care **nu** necesită memorarea drumului parcurs.

```
------------------------------------------------------------
{Căutare în ABO (Varianta recursivă)}

FUNCTION CautaRecursiv(x:TipCheie; b:TipABO):RefTipNod;
  BEGIN
    IF b=NIL THEN
        CautaRecursiv:=b
     ELSE
       IF x<b^.cheie THEN
           b:=CautaRecursiv(x,b^.stang)              [8.3.3.c]
         ELSE
           IF x>b.cheie THEN
               b:=CautaRecursiv(x,b^.drept)
             ELSE
               CautaRecursiv:=b
  END; {CautaRecursiv}
------------------------------------------------------------
```

8.3.4. Inserţia nodurilor în ABO. Crearea arborilor binari ordonaţi

* În cadrul acestui paragraf se tratează:

    * (1) **Inserţia nodurilor** într-un arbore binar ordonat.

    * (2) Problema **construcţiei unui arbore binar ordonat**, pornind de la o mulţime dată de noduri.

* Procesul de creare al unui ABO constă în **inserţia** câte unui nod într-un arbore binar ordonat care iniţial este vid.

    * Problema care se pune este aceea de a executa inserţia de o asemenea manieră încât arborele să rămână **ordonat** şi după adăugarea noului nod.

    * Acesta se realizează **traversând** arborele începând cu rădăcina şi selectând fiul **stâng** sau fiul **drept**, după cum cheia de inserat este mai **mică** sau mai **mare** decât cheia nodului parcurs.

    * Aceasta proces se **repetă** până când se ajunge la un pointer NIL.

    * În continuare inserţia se realizează modificând acest pointer astfel încât să indice noul nod.

* Se precizează că inserţia noului nod **trebuie** realizată chiar dacă arborele conţine deja un nod cu cheia egală cu cea nouă.

- În acest caz, dacă se ajunge la un nod cu cheia egală cu cea de inserat, se procedează ca şi cum aceasta din urmă ar fi **mai mare**, deci se trece la fiul **drept** al nodului curent.

- În felul acesta la parcurgerea în **inordine** a arborelui binar ordonat se obţine o **sortare stabilă** a cheilor arborelui. (Vol.1 &3.1).

- În fig.8.3.4.a se prezintă inserţia unei noi chei cu numărul `8` în structura existentă de arbore ordonat.

    - La parcurgerea în inordine a acestui arbore, se observă că cele două chei egale sunt parcurse în ordinea în care au fost inserate.

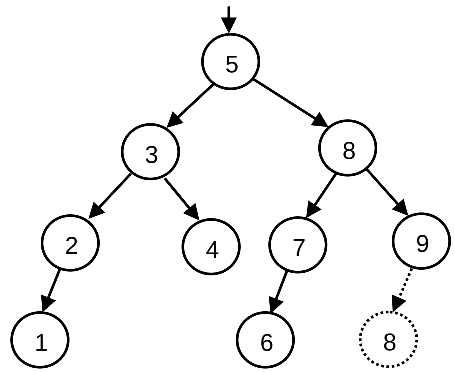


Fig.8.3.4.a. Inserţia unui nod nou cu o cheie existentă

- În continuare se prezintă o **procedură recursivă** pentru **inserţia unui nod într-un arbore binar ordonat**, astfel încât acesta să rămână ordonat.

- Se precizează că iniţial, arborele poate fi vid.

- **Structura arbore** la care se vor face referiri este cea precizată în secvenţa [8.3.3.a].

- Procedura **`Insereaza`** realizează inserţia unui nod cu cheia `x` într-un arbore binar ordonat [8.3.4.a].

    - Se precizează că `x` este un număr întreg reprezentând cheia nodului de inserat şi `b` un pointer care indică rădăcina arborelui

```
------------------------------------------------------------
{Inserţia unui nod într-un arbore binar ordonat}

PROCEDURE Insereaza(x:TipCheie; VAR b:TipABO);
  BEGIN
    IF b<> NIL THEN
        IF x<b^.cheie THEN
            Insereaza(x,b^.stang)
          ELSE
            Insereaza(x,b^.drept)                    [8.3.4.a]
      ELSE {b este NIL}
        BEGIN
          new(b); {completarea înlănţuirii}
          b^.cheie:=x; b^.stang:=NIL; b^.drept:=NIL
        END
```

```
  END; {Insereaza}
------------------------------------------------------------
```

* Se observă că pentru funcţionarea corectă a acestei proceduri este esenţial ca `b` să fie parametru variabil, deoarece numai astfel noua valoare pe care o primeşte `b` prin instrucţiunea `new(b)`, se asignează şi parametrului actual corespunzător.

* În secvenţa [8.3.4.b] se prezintă un fragment de **program principal** care utilizează procedura de mai sus în vederea **creării unui arbore binar ordonat.**

    * Se presupune că:

        * (1) Toate cheile sunt diferite de zero.

        * (2) Cheile se citesc de la dispozitivul de intrare.

        * (3) Secvenţa de chei se încheie cu o cheie fictivă egală cu zero pe post de terminator.

```
------------------------------------------------------------
{Construcţia unui arbore binar ordonat}

VAR radacina:RefTipNod;
    c:TipCheie;
  BEGIN
    radacina:=NIL;
    Read(c);                                          [8.3.4.b]
    WHILE c<>0 DO
      BEGIN
        Insereaza(c,radacina);
        Read(c)
      END
  END;
------------------------------------------------------------
```

8.3.4.1. Inserţia nodurilor în ABO. Varianta iterativă

* În continuare se descrie o **variantă nerecursivă** a procedurii `Inserează`.

    * În cadrul acestei variante se disting două părţi şi anume:

        * (1) **Parcurgerea** arborelui pentru găsirea locului unde trebuie inserat noul nod.

        * (2) **Inserţia** propriu-zisă.

* Prima parte se implementează cu ajutorul a **doi pointeri** `q1` şi `q2`, urmând un algoritm similar celui utilizat la liste (tehnica celor doi pointeri - Vol.1 &6.4.2).

    * Cei doi pointeri indică mereu **două noduri** "**consecutive**" ale arborelui:

        * `q2^` este **nodul curent** (iniţial rădăcina).

        * `q1^` este **fiul său stâng sau fiul drept**, după cum `x`, cheia care se caută, este mai mică respectiv mai mare decât cheia nodului curent indicat de `q2`.

    * Pointerii avansează în tandem, din nod în nod de-a lungul arborelui, până când pointerul `q1` devine `NIL`

    * În acest moment se realizează inserţia propriu-zisă a noului nod drept fiu al lui `q2`.

    * Se precizează că este nevoie şi de o variabilă întreagă `d`, pentru a preciza dacă nodul nou trebuie inserat ca fiu stâng sau ca fiu drept al lui `q2^`.

        * Această variabilă se asignează în timpul parcurgerii arborelui şi se testează în cadrul inserţiei propriu-zise [Wi76].

* Spre deosebire de varianta recursivă în care traseul parcurs este **memorat implicit** de către mecanismul de implementare al recursivităţii cu ajutorul unei stive, în acest caz **nu** este nevoie de stivă întrucât **nu** trebuie să se revină în arbore decât cu un singur nivel (pentru a realiza înlănţuirea), motiv pentru care sunt suficienţi **doi pointeri consecutivi** (fig.8.3.4.b (b)).

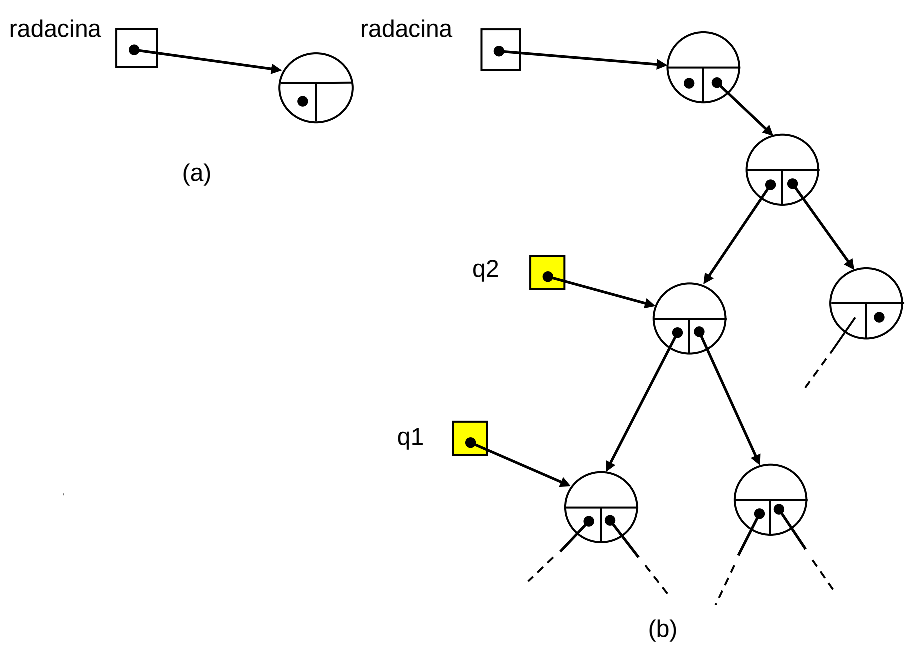


Fig.8.3.4.b. Arbori binari ordonaţi. Tehnica celor doi pointeri

* Procedura care realizează inserţia într-un **arbore binar ordonat** în manieră nerecursivă apare în secvenţa [8.3.4.c].

```
------------------------------------------------------------
{Inserţia în ABO (Varianta nerecursivă)}

PROCEDURE InsereazaNerecursiv(x:TipCheie; b:TipABO);
  VAR q1,q2:RefTipNod;
      d:integer;
  BEGIN
    q2:=b;              {iniţializare pointeri}
```

```
    q1:=q2^.drept;
    d:=1;
    WHILE q1<>NIL DO {parcurgere arbore în tandem}
      BEGIN
        q2:=q1;
        IF x<q1^.cheie THEN
            BEGIN
              q1:=q1^.stang;
              d:=-1.
            END
          ELSE
            BEGIN
              q1:=q1^.drept;                        [8.3.4.c]
              d:=1
            END
      END; {terminare parcurgere}
    new(q1); {inserţie}
    q1^.cheie:=x;
    q1^.stang:=NIL;
    q1^.drept:=NIL
    IF d<0 THEN         {IF (x<q2^cheie) THEN ...}
        q2^.stang:=q1
      ELSE
        q2^.drept:=q1
  END; {InsereazaNerecursiv}
-------------------------------------------------------------
```

- Este uşor de văzut că această procedură funcţionează corect **numai** dacă arborele are **cel puţin un nod.**

- Din acest motiv în implementarea structurii arborelui se utilizează **tehnica nodului fictiv**.

  - Astfel iniţial arborele va conţine un **nod fictiv** a cărui înlănţuire pe dreapta indică primul **nod efectiv** al arborelui.

- În această accepţiune **arborele binar vid** arată ca şi în figura 8.3.4.b.(a).

  - Drept consecinţă cei doi pointeri vor putea fi poziţionaţi în mod corespunzător chiar şi pentru arborele vid: `q2` indică nodul fictiv iar `q1` este `NIL`.

- De asemenea se face precizarea că se poate renunţa la variabila `d`.

  - Faptul că noul nod trebuie inserat ca fiu stâng sau ca fiu drept al lui `q2` se stabileşte comparând cheia lui `q2` cu cheia de inserat `x`.

  - Acest procedeu este sugerat ca şi comentariu în secvenţa [8.3.4.c.]

8.3.4.2. Considerente generale referitoare la crearea ABO

- Cu privire la crearea arborilor binari ordonaţi se poate menţiona faptul că **înălţimea** arborilor obţinuţi prin procedurile prezentate, depinde de **ordinea** în care se furnizează iniţial cheile.

- Dacă spre exemplu, secvenţa cheilor iniţiale este `5,3,8,2,4,7,9,1,6` atunci se obţine arborele din figura 8.3.1.a stânga, având o înălţime minimă (`4`).

- Dacă aceleaşi chei se furnizează în ordinea `7,2,8,1,6,9,3,5,4` atunci rezultă arborele mai puţin avantajos din aceeaşi figura dreapta cu înălţimea `6`.

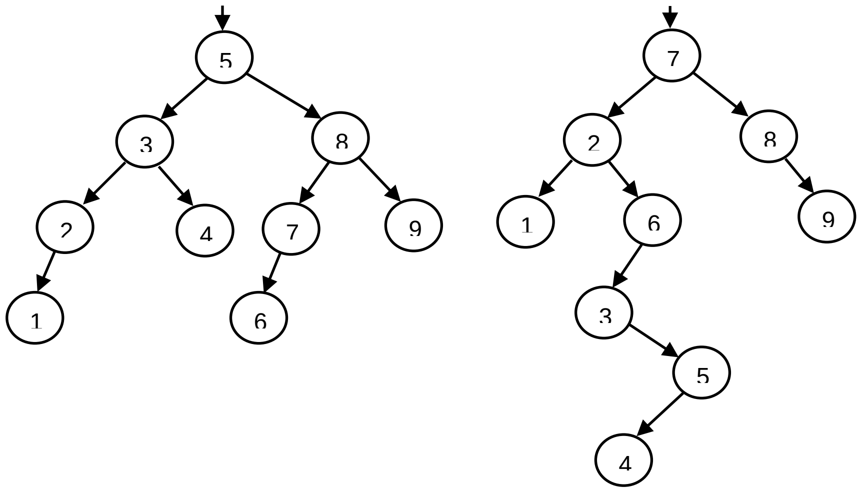


Fig.8.3.1.a. Arbori binari ordonaţi de diferite înălţimi (reluare)

- În cazul cel mai **defavorabil**, arborele poate degenera în **listă liniară**, lucru care se întâmplă în cazul în care cheile sunt furnizate în vederea inserţiei în **secvenţă ordonată crescător** respectiv **descrescător** (fig.8.3.4.c.(a),(b)).

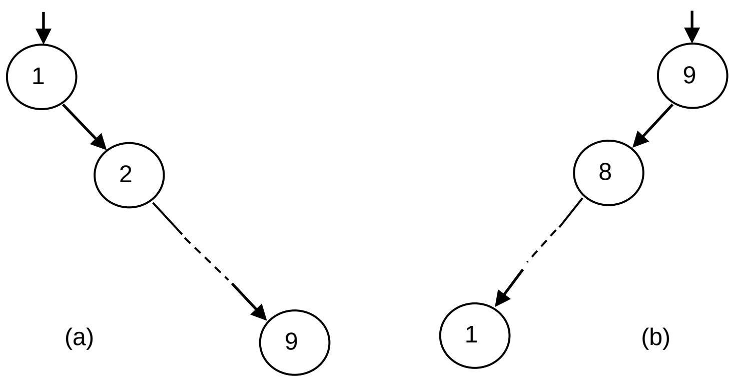


Fig.8.4.3.c. Arbori binari ordonaţi degeneraţi în liste liniare.

- Este evident faptul că în astfel de situaţii performanţa căutării scade catastrofal fiind practic egală cu cea a căutării într-o listă **liniară ordonată**.

- Din fericire, probabilitatea ca să apară astfel de situaţii este destul de redusă, fenomen ce va fi analizat mai târziu în cadrul acestui capitol.

8.3.5. Suprimarea nodurilor în arbori binari ordonaţi

* Se consideră o structură **arbore binar ordonat** şi o cheie precizată `x`.
* Se cere să se **suprime** din structura arbore binar ordonat nodul având cheia `x`.
    * Pentru aceasta, în prealabil se **caută** dacă există un nod cu o astfel de cheie.
    * Dacă **nu**, suprimarea s-a încheiat şi se emite eventual un mesaj de eroare.
    * În caz contrar se execută suprimarea propriu-zisă, de o asemenea manieră încât arborele să rămână **ordonat** şi după terminarea ei.
* Se disting două cazuri, după cum nodul care trebuie suprimat are:
    * (1) **Cel mult un fiu**
    * (2) **Doi fii.**
* **(1) Primul caz** în care nodul de suprimat are **cel mult un fiu**, se rezolvă conform figurii 8.3.5 (a,b,c) în care se prezintă cele trei variante posibile.

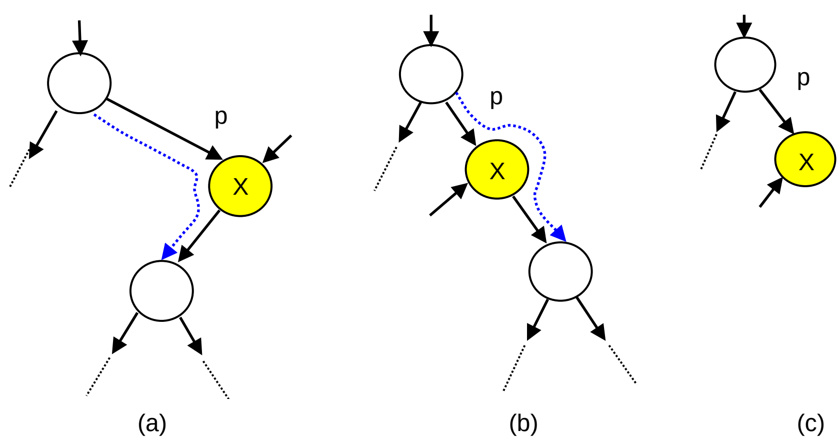


Fig.8.3.5.a. Suprimarea unui nod într-un ABO. Cazul 1.

* **Regula generală** care se aplică în acest caz este următoarea:
    * Fie `p` câmpul referintă aparţinând **tatălui** nodului `x`, referinţă care indică nodul `x`.
    * Valoarea lui `p` se **modifică** astfel încât acesta să indice unicul fiu al lui `x` (dacă un astfel de fiu există - fig. 8.3.5.a (a),(b)) sau dacă un astfel de fiu nu există, `p` devine `NIL` (fig.8.3.5.a (c)).
* Fragmentul de cod care apare în continuare ilustrează acest procedeu [8.3.5.a].

```
------------------------------------------------------------
{Suprimarea unui nod într-un ABO. Cazul 1: nodul de suprimat
are un singur sau niciun fiu}

q:=p; {p indică nodul de suprimat}
IF q^.drept=NIL THEN
```

```
      p:=q^.stang
  ELSE                                                  [8.3.5.a]
      IF q^.stang=NIL THEN
        p:=q^.drept;
-------------------------------------------------------------
```

- Ca **exemplu**, se prezintă în continuare implementarea operatorului `SuprimaMin` care suprimă şi în acelaşi timp returnează **cel mai mic element** al unui arbore binar ordonat (secvenţa [8.3.5.b]).

    - **Cel mai mic** element al unui arbore binar ordonat, este cel mai din **stânga** nod al arborelui, nod la care se ajunge înaintând mereu spre stânga pornind de la rădacină.

    - Primul nod care **nu** are înlănţuire spre stânga (**nu** are fiu stâng) este nodul căutat.

    - Suprimarea lui este imediată în baza procedeului precizat mai sus.

```
-------------------------------------------------------------
{Operatorul SuprimaMin în ABO}

PROCEDURE SuprimaMin(VAR b:TipABO; VAR min:TipElement);
  VAR temp:RefTipNod;
  BEGIN
    IF b<>NIL THEN
      IF b^.stang<>NIL THEN
          SuprimaMin(b^.stang, min)
        ELSE
          BEGIN                                        [8.3.5.b]
            min:=b^.info; temp:=b;
            b:=b^.drept;    {suprimare}
            DISPOSE(temp)
          END
  END; {SuprimaMin}
-------------------------------------------------------------
```

- Într-o manieră similară se poate implementa operatorul **`SuprimaMax`**

    - Acesta realizează suprimarea celui mai mare nod al arborelui, care este evident nodul situat cel mai la **dreapta** în arbore.

- (2) **Cel de-al doilea caz**, în care nodul de suprimat are **doi fii** se rezolvă astfel:

    - (1) Se caută **predecesorul** nodului de suprimat `x` în **ordonarea în inordine a arborelui.**

        - Fie acesta `y`. Se demonstrează că nodul `y` există şi că el nu are fiu drept.

    - (2) Se **modifică** nodul `x`, asignând toate câmpurile sale, cu excepţia cîmpurilor `stâng` şi `drept` cu câmpurile corespunzătoare ale lui `y`.

        - În acest moment în structura arbore, nodul `y` se găseşte în dublu exemplar: în locul său iniţial şi în locul fostului nod `x`.

- (3) Se **suprimă** nodul `y` iniţial, conform fragmentului [8.3.5.a] deoarece nodul nu are fiu drept.

- Cu privire la nodul `y`, se poate demonstra că el se detectează după următoarea **metodă**:

  - Se construieşte o secvenţă de noduri care începe cu fiul **stâng** al lui `x`, după care se alege drept succesor al fiecărui nod, fiul său **drept.**

  - Primul nod al secvenţei care **nu** are fiu drept este `y` (fig.8.3.5.b).

    - Este de fapt **cel mai mare nod** al subarborelui stâng al subarborelui binar care are rădăcina `x`.

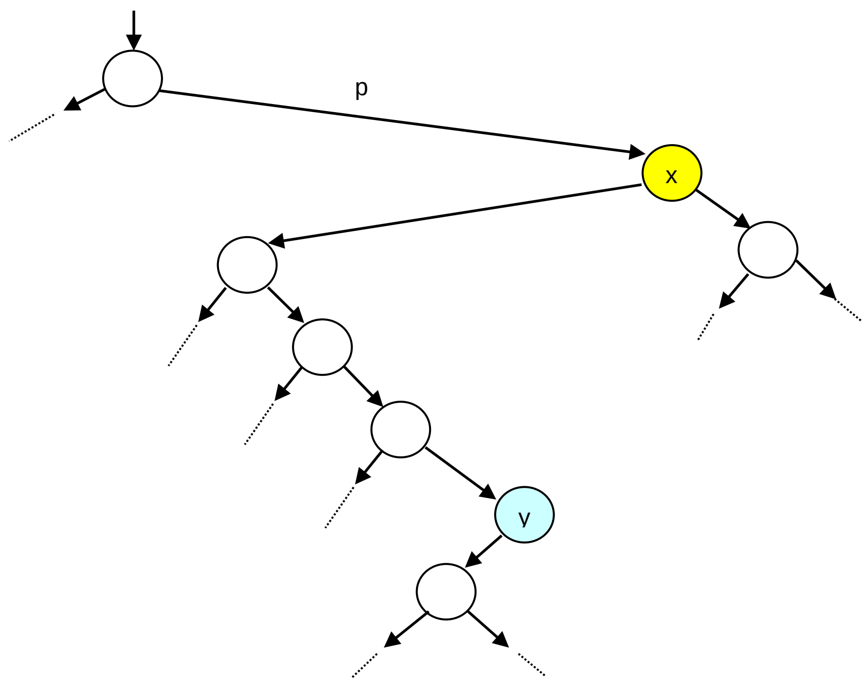


Fig. 8.3.5.b. Suprimarea unui nod într-un ABO. Cazul 2.

- Procedura care realizează **suprimarea** unui nod într-o structură **arbore binar ordonat** apare în secvenţa [8.3.5.c].

  - Procedura locală `SuprimaPred`, caută **predecesorul în inordine** al nodului `x`, realizând suprimarea acestuia conform metodei descrise (are cel mult un fiu).

  - După cum se observă, procedura `SuprimaPred` se utilizează numai în situaţia în care nodul `x` are doi fii.

```
------------------------------------------------------------
{Suprimarea unui nod într-un ABO. Cazul general}

PROCEDURE Suprima(x:TipCheie; VAR b:TipABO);
  VAR q:RefTipNod;
```

```
PROCEDURE SuprimaPred(VAR r:RefTipNod);
  BEGIN
    IF r^.drept<>NIL THEN
        SuprimaPred(r^.drept)
      ELSE
        BEGIN
          q^.cheie:=r^.cheie; {mută conţinutul lui r în q}
          q^.numar:=r^.numar;
          q:=r;
          r:=r^.stang {suprimă nodul r}
        END
  END; {SuprimaPreded}

BEGIN {Suprimare}
  IF b=NIL THEN
      WRITELN(' nodul nu se gaseste')        [8.3.5.c]
    ELSE
      IF x<p^.cheie THEN
          Suprima(x,p^.stang)
        ELSE
          IF x>p^.cheie THEN
              Suprima(x,p^.drept)
            ELSE
              BEGIN
                q:=p;
                IF q^.drept=NIL THEN  {Cazul 1}
                    p:=q^.stang
                  ELSE
                    IF q^.stang=NIL THEN  {Cazul 1}
                        p:=q^.drept
                      ELSE
                        SuprimaPred(q^.stang); {Cazul 2}
                {DISPOSE(q)}
              END
END; {Suprimare}
```

- Procedura `SuprimaPred`:
  - (1) Găseşte pointerul `r` care indică nodul având cea mai mare cheie, dintre cheile **subarborelui stâng**, al arborelui care are drept rădăcină nodul cu cheia `x` (nodul de suprimat).
  - (2) Înlocuieşte câmpurile nodului cu cheia `x`, indicat de pointerul `q`, cu câmpurile nodului indicat de `r` (cu excepţia înlănţuirilor).
  - Suprimă nodul indicat de `r`, acesta din urmă având un singur fiu (sau niciunul).
- Pentru a ilustra comportarea acestei proceduri în fig.8.3.5.c se prezintă:
  - O structură de arbore binar ordonat (a)
  - Din care se suprimă în mod succesiv nodurile având cheile 7, 8, 4, şi 6 (fig.8.3.5.c (b-e)).

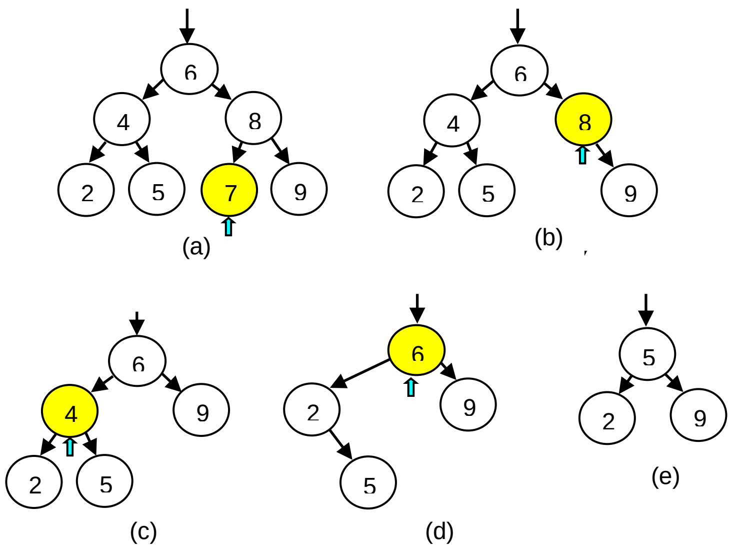


Fig. 8.3.5.c. Suprimarea nodurilor într-o structură arbore binar ordonat

- Există şi **o altă soluţie** de a rezolva suprimarea în cazul în care nodul `x` are doi fii şi anume:

    1. Se caută **succesorul** nodului `x` în ordonarea în inordine a cheilor arborelui. Se demonstrează că el există şi ca **nu** are fiu stâng.

    2. Pentru suprimare se procedează analog ca şi în cazul anterior, cu deosebirea că totul se realizează **simetric** (în oglindă).

- În acest caz de fapt se caută nodul cu **cea mai mică cheie** a **subarborelui drept** al subarborelui care-l are pe `x` drept rădăcină

- Pentru o mai bună înţelegere a celor prezentate se reaminteşte o **proprietate** a arborilor binari ordonaţi:

    - **Proiecţia pe abscisă** a nodurilor unui arbore binar, conduce la **ordonarea lor în inordine.**

    - În cazul **arborilor binari ordonaţi** se obţine de fapt secvenţa ordonată a cheilor arborelui.

8.3.6. Analiza căutării în arbori binari ordonaţi

- În general, în activitatea de programare se manifestă o anumită suspiciune faţă de căutarea şi inserţia nodurilor într-o structură **arbore binar ordonat**.

- Această suspiciune este motivată de faptul că programatorul în general **nu** are controlul creşterii arborelui şi ca atare **nu** poate anticipa cu suficientă precizie forma acestuia.

- După cum s-a precizat, efortul de căutare al unei chei variază între O*(log₂ n)* pentru **arborele binar perfect echilibrat** (de înălţime minimă) şi O*(n/2)* pentru **arborele binar** degenerat într-o **listă liniară**.

- Cele două situaţii reprezintă extremele situaţiilor reale iar probabilitatea ca ele să apară în este în general redusă [Wi76].

- Cazul general care va fi analizat în continuare, se referă la:

    - Dacă se dau `n` **chei**, ele pot fi permutate în `n!` **moduri** şi în consecinţă cu cele `n` chei se pot construi `n!` **arbori binari ordonaţi**, deoarece pentru fiecare permutare rezultă un arbore.

    - Ne propunem să determinăm **lungimea medie** a_n **a drumului de căutare,** corespunzător tuturor celor `n` chei şi tuturor celor `n!` arbori care pot fi generaţi pornind de la cele `n!` permutări ale celor `n` chei originale.

    - Se consideră că cele `n` chei sunt distincte având valorile `1,2,...,n,` şi se presupune că sosesc în ordine aleatoare cu o **distribuţie normală** a probabilităţii de apariţie.

    - În acest context **lungimea medie a drumului de căutare** într-un arbore binar cu `n` noduri, a_n se defineşte ca fiind o sumă de `n` termeni, fiecare termen fiind produsul dintre nivelul unui nod al arborelui (care este chiar lungimea drumului la nodul în cauză) şi probabilitatea sa de acces.

    - Dacă se presupune că toate nodurile sunt în mod egal căutate (au probabilitatea de acces `1/n`), atunci formal **lungimea medie a drumului de căutare** a_n apare în [8.3.6.a] unde p_i este lungimea drumului de la rădăcină la nodul `i` (adâncimea nodului `i`).

$$a_n = \frac{1}{n}\sum_{i=1}^{n} p_i \qquad \text{[8.3.6.a]}$$

- Calculând valoarea **lungimii medii a drumului de căutare** a_n se ajunge la formula recursivă [8.3.6.g].

$$a_n = \frac{1}{n^2}((n^2-1)a_{n-1} + 2n - 1) \qquad \text{[8.3.6.g]}$$

- Pe de altă parte, a_n poate fi exprimat într-o formă nerecursivă utilizând termenii **funcţiei armonice** `H` [8.3.6.h] după cum se prezintă în relaţia [8.3.6.i]

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \qquad \text{[8.3.6.h]}$$

$$a_n = 2\frac{n+1}{n}H_n - 3 \qquad \text{[8.3.6.i]}$$

----

* Se poate verifica faptul că relaţia [8.3.6.i] verifică relaţia recursivă [8.3.6.g].

* Dar valoarea aproximativă a lui H_n poate fi determinată în baza **formulei lui Euler** [8.3.6.j].

$$H_n = \gamma + \ln(n) + \frac{1}{12n^2} + \ldots \qquad [8.3.6.j]$$

----

unde $\gamma \approx 0.577$ este constanta lui Euler.

* Dacă se înlocuieşte această valoare în formula [8.3.6.i] rezultă următoarea valoare pentru **lungimea medie a drumului de căutare** într-un **arbore binar ordonat oarecare** cu n chei [8.3.6.k].

$$a_n \approx 2[\ln(n) + \gamma] = 2\ln(n) - c \qquad [8.3.6.k]$$

----

* Întrucât lungimea medie a drumului de căutare într-un **arbore binar perfect echilibrat** este [8.3.6.l]:

$$a_n^{/} = \log_2(n) - 1 \qquad [8.3.6.l]$$

----

* Neglijând termenii constanţi care pentru valori mari ale lui n devin neglijabili şi trecând la **limită**, obţinem relaţia finală [8.3.6.m].

$$\lim_{n\to\infty} \frac{a_n}{a_n^{/}} = \frac{2\ln(n)}{\log_2(n)} = \frac{2\ln(n)}{\frac{\ln(n)}{\ln 2}} = 2\ln 2 \approx 1.386 \qquad [8.3.6.m]$$

----

* **Concluzia** este că înlocuind **arborele binar perfect echilibrat** cu un **arbore binar aleatoriu,** efortul de căutare creşte în medie cu 39 %.

  * Desigur, creşterea acestui efort poate fi mult mai mare, dacă arborele aleatoriu este nefavorabil, spre exemplu degenerat într-o listă, dar această situaţie are o probabilitate foarte mică de a se realiza.

* Cele 39 % impun practic **limita efortului adiţional de calcul** care poate fi cheltuit în mod profitabil pentru reorganizarea structurii după inserarea cheilor.

  * În acest sens un rol esenţial îl joacă **raportul** dintre numărul de accese la noduri (căutări) şi numărul de inserţii realizate în arbore.

  * Cu cât acest raport este mai mare cu atât reorganizarea structurii este mai justificată.

* În general valoarea 39 % este suficient de redusă pentru ca în majoritatea aplicaţiilor să se recurgă la tehnici directe de inserare şi să **nu** se facă uz de reorganizare decât în situaţii deosebite.

8.3.7. Arbori binari parţial ordonaţi

* O structură arbore binar aparte o reprezintă structura **arbore binar parţial ordonat**.

  * **Caracteristica esenţială** a unui **arbore binar parţial ordonat** este aceea că cheia oricărui nod este mai mare (mică) decât cheile fiilor săi.

  * **Consecinţa imediată**: la parcurgerea în inordine, secvenţa cheilor nu mai este ordonată.

* Un exemplu de astfel de arbore apare în figura 8.3.7.a.

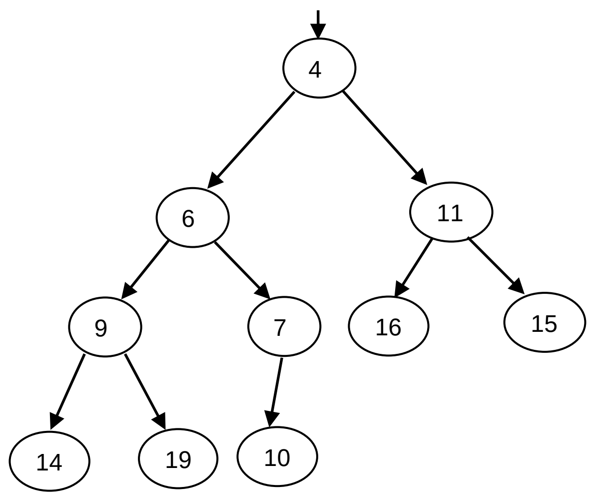


Fig.8.3.7.a. Arbore binar parţial ordonat

* Deoarece un arbore binar parţial ordonat este de fapt un arbore binar, se poate realiza o reprezentare eficientă a sa cu ajutorul unui tablou liniar aplicând tehnica specificată la paragraful 8.2.4.1.

  * Această reprezentare este cunoscută şi sub numele de **ansamblu** (heap) şi a fost definită în partea I (sortare prin metoda ansamblelor - Vol.1 &3.2.5.)

* Spre exemplu **arborele binar parţial ordonat** din figura 8.3.7.a apare reprezentat ca un **ansamblu** în figura 8.3.7.b.

1	2	3	4	5	6	7	8	9	10
4	6	11	9	7	16	15	14	19	10

Fig.8.3.7.b. Reprezentarea unui arbore binar parţial ordonat ca un ansamblu

* **Structura ansamblu** permite implementarea eficientă şi foarte elegantă atât a unor **metode de sortare** (**sortarea prin metoda ansamblelor** - Vol.1 &3.2.5) cât şi a unor **structuri de date** derivate din **liste** (**cozi bazate pe prioritate** - Vol.1 &6.5.5.3).

8.3.8. Aplicaţii ale arborilor binari ordonaţi

8.3.8.1. Problema concordanţei

- În cadrul acestui paragraf se propune reluarea **problemei concordanţei** prezentată în Vol.1 şi rezolvarea ei cu ajutorul structurilor de date arbore.
    - Se reaminteşte că **problema concordanţei** constă de fapt în **determinarea frecvenţelor de acces** ale cuvintelor unui text dat.
- **Problema** se formulează astfel:
    - Se consideră un **text** format dintr-o succesiune de **cuvinte**.
    - Se parcurge textul şi se delimitează cuvintele.
    - Pentru fiecare cuvânt se verifică dacă **este** sau **nu** la prima apariţie.
        - Dacă este la prima apariţie, cuvântul se înregistrează şi contorul asociat se iniţializează pe valoarea 1.
        - Dacă cuvântul a mai fost căutat, se incrementează contorul asociat, acesta contorizând numărul de apariţii.
    - În final se dispune de **lista (**ordonată) a tuturor cuvintelor şi de numărul de apariţii ale fiecăruia.
- În acest scop, nodurile reprezentând cuvintele sunt organizate într-o **structură arbore binar ordonat**, pornind de la un arbore vid.
- Procesul se desfăşoară după cum urmează.
    - Se citeşte un nou cuvânt şi se caută în arbore:
        - Dacă **nu** se găseşte atunci cuvântul se inserează.
        - Dacă cuvântul se găseşte, atunci se incrementează contorul de apariţii al cuvântului respectiv.
    - Procesul continuă până la epuizarea tuturor cuvintelor textului analizat
- Se presupune că un nod al structurii **arbore binar ordonat** are structura precizată în secvenţa [8.3.8.1.a].

```
------------------------------------------------------------
{Problema concordanţei. Implementare bazată pe arbori binari
ordonaţi}

TYPE RefTipNod=^cuvant
     cuvant=RECORD
              cheie:integer;                      [8.3.8.1.a]
              contor:integer;
```

```
              stang,drept:RefTipNod
            END;
```

- Fie `radacina` o variabilă pointer care indică rădăcina arborelui binar ordonat.
- Programul care rezolvă problema concordanţei apare în secvenţa [8.3.8.1.b].

```
PROGRAM Concordanta;

TYPE  RefTipNod=^cuvant;
      cuvant=RECORD
               cheie:integer;
               contor:integer;
               stang,drept:RefTipNod
             END;
VAR radacina:RefTipNod; cuv:integer;

PROCEDURE Imprarbore(r:RefTipNod);
  BEGIN
    IF r<>NIL THEN
      BEGIN
        Imprarbore(r^.stang);
        WRITELN(r^.cheie,r^.contor);
        Imprarbore(r^.drept)
      END
  END; {Imprarbore}

PROCEDURE Cauta(x:integer; VAR p:RefTipNod);
  BEGIN
    IF p=NIL THEN {cuvântul nu se găseşte, deci inserţie}
        BEGIN
          new(p);                                  [8.3.8.1.b]
          p^.cheie:=x; p^.contor:=1;
          p^.stang:=NIL; p^.drept:=NIL
        END
      ELSE
        IF x<p^.cheie THEN
            Cauta(x,p^.stang)
          ELSE
            IF x>p^.cheie THEN
                Cauta(x,p^.drept)
              ELSE {cuvântul s-a găsit, incrementare contor}
                p^.contor:=p^.contor+1
  END; {Cauta}

BEGIN {PROGRAM principal}
  radacina:=NIL; {*}
  Read(cuv);
  WHILE cuv<>0 DO
    BEGIN
      Cauta(cuv,radacina);
      Read(cuv)
    END;
  Imprarbore(radacina)
END.
```

- Pentru simplificare se presupune ca **textul** analizat constă dintr-o succesiune de **numere întregi** care modelează cuvintele textului, iar cifra `0` este utilizată ca **terminator**.
- **Programul principal** realizează următoarele:
    - Iniţializează structura cu arborele vid (`radacina`).
    - Citeşte pe rând cuvintele textului în variabila `cuv` prin tastarea numerelor care le reprezintă (bucla `WHILE`).
    - Pentru fiecare cuvânt (număr) tastat apelează procedura **`Cauta`**`(cuv,radacina)`.
    - Programul se finalizează cu tastarea numărului `0` considerat drept terminator.
- Procedura **`Cauta`** realizează următoarele:
    - (1) Caută cheia `cuv` în arborele indicat de pointerul `radacina`.
    - (2) Dacă **nu** o găseşte, inserează cheia în arbore.
    - (3) Dacă găseşte cheia incrementează contorul corespunzător.
- După cum se observă, această procedură este o **combinaţie** a căutării şi creării arborilor binari ordonaţi.
    - Metoda de **parcurgere** a arborelui este cea prezentată la căutarea în arbori binari ordonaţi varianta recursivă (&8.3.3)
- Dacă **nu** se găseşte nici un nod cu cheia `cuv` atunci are loc inserţia similară celei utilizate în cadrul procedurii `Insereaza1` definită la inserţia în arbori binari ordonaţi varianta 1 (&8.3.4)
- Procedura recursivă `Imprarbore` parcurge nodurile arborelui în **inordine** afişându-le unele sub altele, fără a reflecta însă şi structura arborelui, element care diferenţiază această procedură de cea prezentată în secvenţa [8.2.7.1.a.].

8.4. Arbori binari echilibraţi. Arbori AVL

8.4.1. Definirea arborilor echilibraţi AVL

- Din **analiza** căutării în **arbori binari ordonaţi** prezentată în &8.3.6. rezultă în mod evident că o procedură de inserare care restaurează structura de arbore astfel încât ea să fie **tot timpul** *perfect echilibrată* **nu** este viabilă, deoarece activitatea de restructurare este foarte **complexă**.

    - Cu toate acestea sunt posibile anumite **ameliorări**, dacă termenul **"echilibrat"** este definit într-o manieră **mai puţin strictă**.

    - Astfel de criterii de echilibrare "**imperfectă**" pot conduce la **tehnici** mai simple de **reorganizare** a **structurii arbore binar ordonat**, al căror cost deteriorează într-o măsură redusă performanţa medie de căutare.

- Una dintre aceste definiţii ale echilibrării **arborilor binari ordonaţi** este cea propusă de **Adelson, Velskii** şi **Landis** în 1962 şi care are următorul enunţ:

    - Un **arbore binar ordonat** este **echilibrat** dacă şi numai dacă pentru oricare nod al arborelui, înălţimile celor doi subarbori diferă cu cel mult 1.

    - Arborii care satisfac acest criteriu se numesc "**arbori AVL**" după numele inventatorilor.

- În cele ce urmează, sintagma "**arbori echilibraţi AVL**" este sinonimă cu "**arbori AVL**"

    - Se atrage atenţia asupra faptului că arborii **perfect echilibraţi** sunt de asemenea **arbori AVL**.

- Această definiţie are câteva avantaje:

    - (1) Este foarte **simplă.**

    - (2) Conduce la o **procedură de reechilibrare** viabilă.

    - (3) Asigură o **lungime medie a drumului de căutare** practic identică cu cea a unui **arbore perfect echilibrat.**

- În acest context se vor studia următorii operatori definţi în cadrul **structurii arbore echilibrat AVL**:

    1^o **Inserţia** unui nod cu o cheie dată.

    2^o **Suprimarea** unui nod cu o cheie dată.

- Toţi aceşti operatori necesită un **efort de calcul** de ordinul O*(log n),* unde n este numărul nodurilor structurii, chiar în cel mai **defavorabil caz**.

- **Optimul** este atins de arborii echilibraţi având un număr de noduri $n=2^k-1$.

8.4.2. Inserţia nodurilor în arbori echilibraţi AVL

- Se dă un **arbore AVL** având rădăcina `R`, subarborele `S` de înălţime h_S pe post de subarbore stâng şi subarborele `D` de înălţime h_D pe post de subarbore drept.

  - Se cere **să se insereze** un nod nou în acest arbore.

- Se presupune că nodul nou se **inserează** în **subarborele stâng S**, determinând creşterea cu 1 a înălţimii acestuia.

  - Se disting trei cazuri:

    1. $h_S=h_D$: în urma inserţiei `S` şi `D` devin de înălţimi inegale, fără însă a viola criteriul echilibrului.

    2. $h_S<h_D$: în urma inserţiei `S` şi `D` devin de înălţimi egale, echilibrul fiind îmbunătăţit.

    3. $h_S>h_D$: criteriul echilibrului este violat şi arborele trebuie **reechilibrat**.

- Lucrurile se întâmplă similar, dacă nodul nou se **inserează** în **subarborele drept D**, determinând creşterea cu 1 a înălţimii acestuia, cu deosebirea ca totul se reflectă în oglindă, adică se schimba `S` cu `D` respectiv `D` cu `S`.

- Astfel, în arborele echilibrat din figura 8.5.3.a:

  - Nodurile `9` sau `11` pot fi inserate **fără** reechilibrare.

  - Inserţia unuia din nodurile `1,3,5` sau `7` necesită însă **reechilibrarea** arborelui.

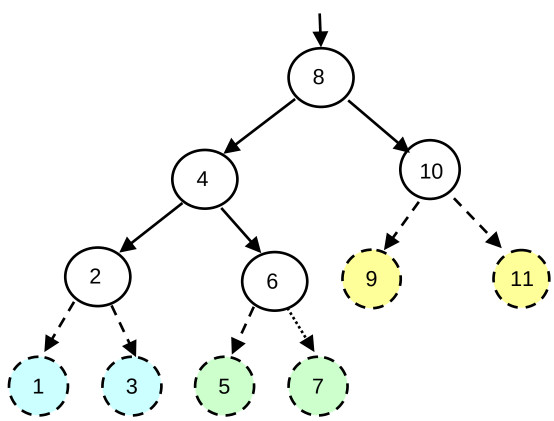


Fig.8.5.3.a. Arbore echilibrat AVL

- O analiză atentă a situaţiilor posibile care rezultă în urma inserţiei evidenţiază faptul că există numai **două configuraţii** care necesită tratamente speciale.

  - Celelelate configuraţii pot fi reduse la aceste două situaţii din considerente de **simetrie.**

* **Prima situaţie** se referă la inserţia nodurilor `1` sau `3` în arborele reprezentat cu linie continuă în figura 8.5.3.a

* **Cea de-a doua situaţie** se referă la inserţia nodurilor `5` sau `7` în arborele din figura 8.5.3.a

  * Cele două situaţii sunt prezentate în figurile 8.5.3.b şi 8.5.3.c, fiecare în câte trei ipostaze (a), (b) şi (c) care evoluează de la simplu la complicat.

  * Cele două situaţii sunt denumite cazul "**1 Stânga**" respectiv cazul "**2 Stânga**".

  * Ambele cazuri presupun creşterea **subarborelui stâng** `S`, ca atare reprezintă un **caz stânga**.

    * **Cazul 1 Stînga** presupune creşterea **subarborelui stâng** al **subarborelui stâng** al arborelui în cauză.

    * **Cazul 2 Stânga** presupune creşterea **subarborelui drept** al **subarborelui stâng** al arborelui în cauză.

  * Elementele adăugate prin inserţie apar cu linie punctată.

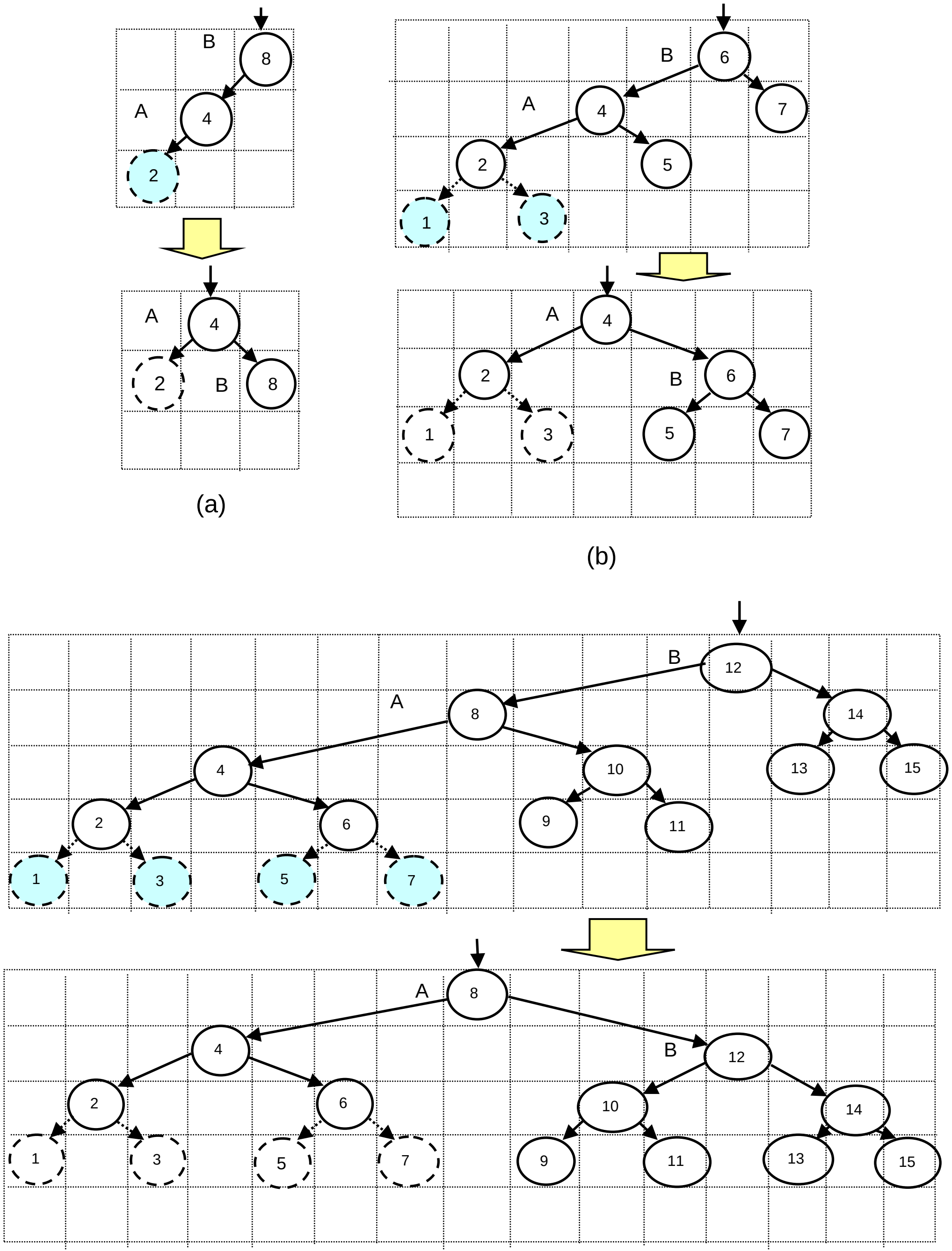
  * Prin transformări simple, structurile de arbori se reechilibrează.

    * În **cazul 1 Stânga** este vorba despre **o rotaţie simplă** de două noduri `A` respectiv `B`

    * În cazul **2 Stânga** este vorba despre **o rotaţie dublă** în care sunt implicate trei noduri: `A,B` şi `C`.

      * Se subliniază faptul că arborii AVL fiind arbori ordonaţi, singurele mişcări permise ale nodurilor sunt cele pe verticală.

      * Poziţiile relative ale proiecţiilor pe orizontală ale nodurilor aparţinând unui arbore AVL, trebuie să rămână nemodificate.



(c)

Fig.8.5.3.b. Echilibrarea arborilor AVL. Cazul **1Stânga**

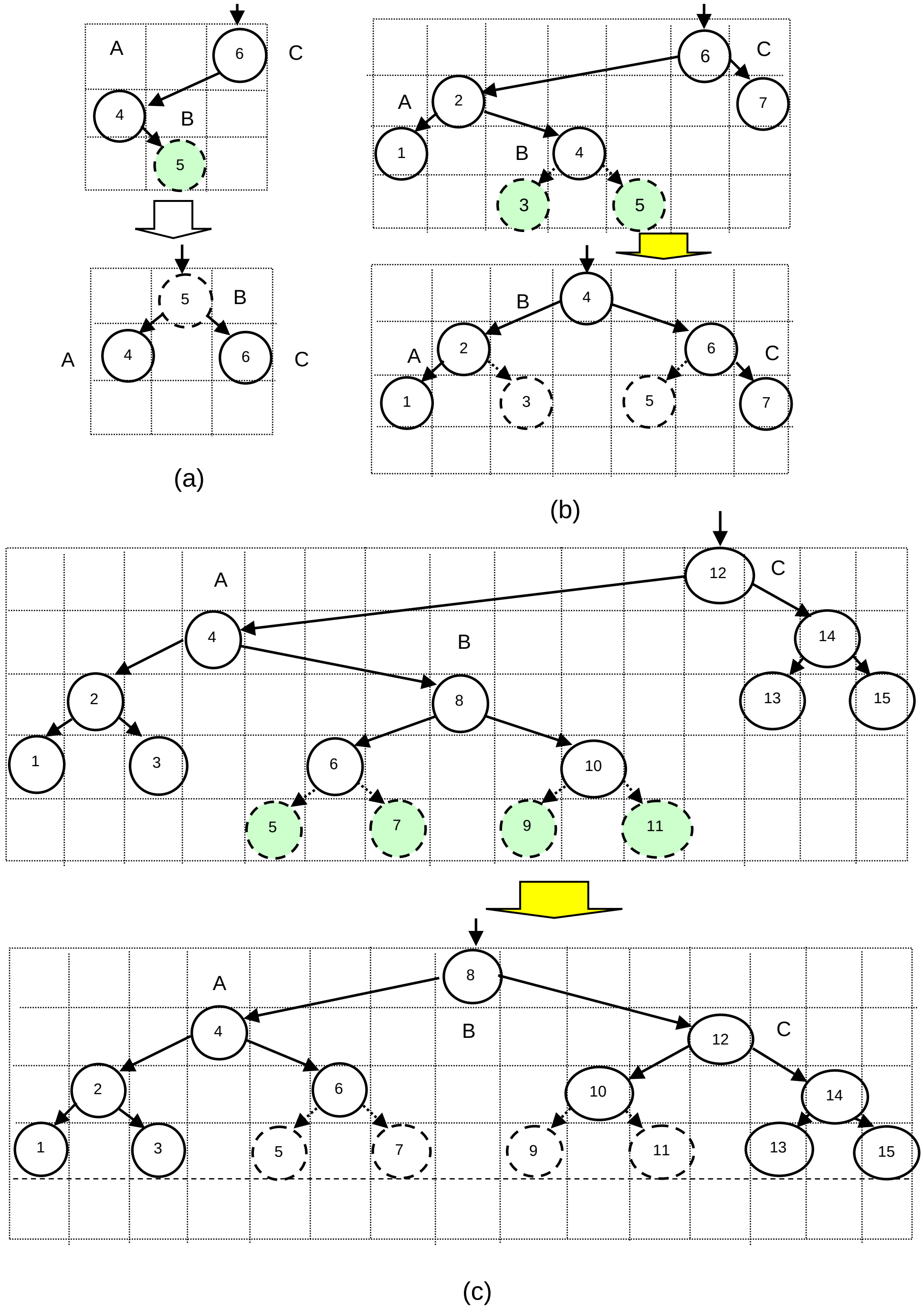


Fig.8.5.3.c. Echilibrarea arborilor AVL. Cazul **2 Stânga**

- Sinteza acestor cazuri precum şi modul sintetic în care se realizează procesul de echilibrare pentru cazurile pe stânga sunt prezentate în figurile 8.5.3.d şi 8.5.3.e.

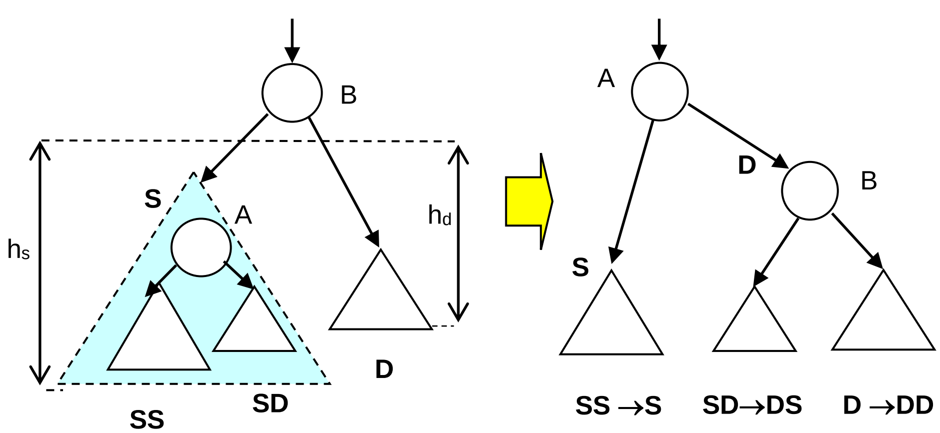


Fig.8.5.3.d. Echilibrarea arborilor AVL**. Cazul 1 Stânga**. Schema generală

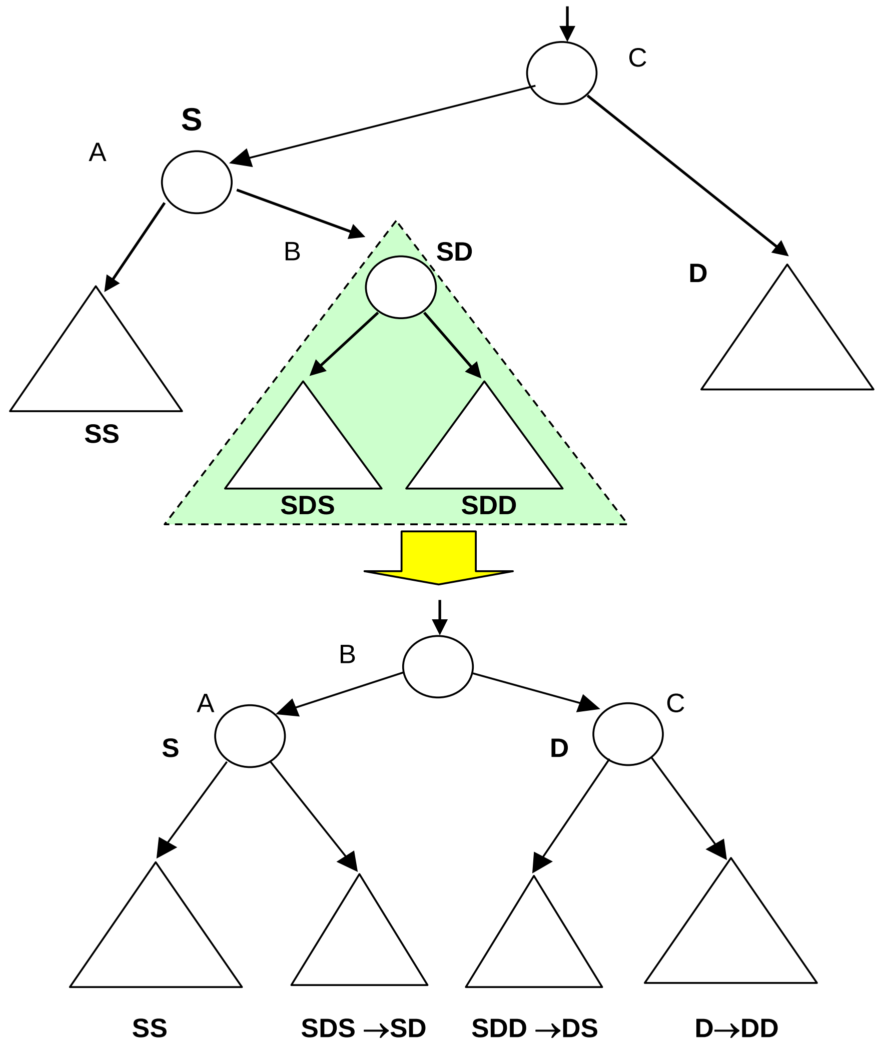


Fig.8.5.3.e. Echilibrarea arborilor AVL. **Cazul 2 Stânga**. Schema generală

- În oglindă cu cazurile pe stânga, pentru **dreapta** se pot distinge următoarele cazuri:
  - **Cazul 1 Dreapta** care presupune creşterea **subarborelui drept** al **subarborelui drept** al arborelui original
  - **Cazul 2 Dreapta** care presupune creşterea **subarborelui stâng** al **subarborelui drept** al arborelui original.
- Şi în acest caz, reechilibrarea se rezolvă prin **una** sau **două rotaţii** ale nodurilor `A` şi `B`, respectiv ale nodurilor `A`, `B` şi `C`.
  - Aceleaşi scheme sintetice de data aceasta pentru cazurile pe **dreapta** apar în figurile 8.5.3.f respectiv 8.5.3.g.
    - Este vorba despre **cazurile 1** respectiv **2 Dreapta**.

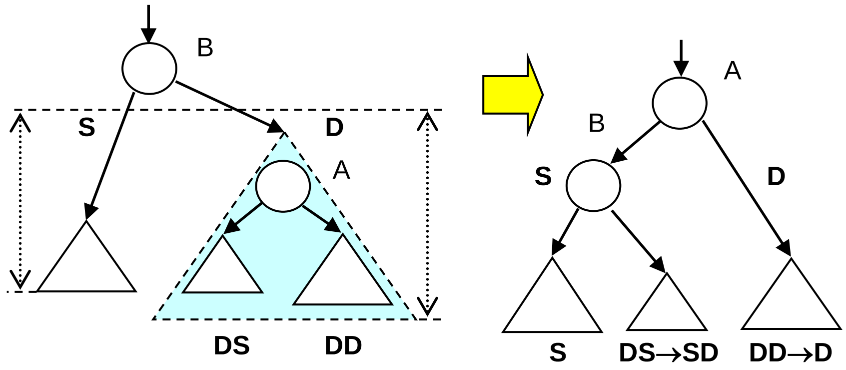


Fig.8.5.3.f. Echilibrarea arborilor AVL. **Cazul 1 Dreapta**. Schema generală

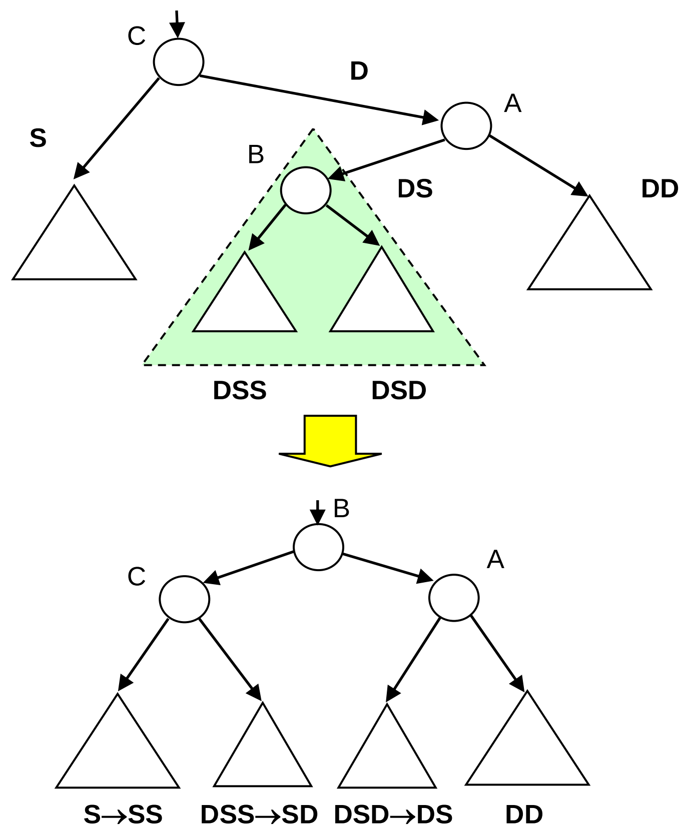


Fig.8.5.3.g. Echilibrarea arborilor AVL. **Cazul 2 Dreapta**. Schema generală

- **Principial, procesul de echilibrare** împreună cu modalitatea efectivă de **restructurare** apar pentru fiecare din cele două cazuri în figurile mai sus precizate.

- Un **algoritm pentru inserţie şi reechilibrare** depinde în **mod critic** de maniera în care este memorată informaţia referitoare la **situaţia echilibrului** arborelui.

- O soluţie este aceea prin care se atribuie **fiecărui nod** un **factor explicit de echilibru**.

    - **Factorul de echilibru** se referă la subarborele a cărui rădăcină o constituie nodul în cauză.

- **Factorul de echilibru al unui nod**, va fi interpretat ca şi **diferenţa** dintre înălţimea **subarborelui său drept** şi înălţimea **subarborelui său stâng**.

- În acest caz structura unui nod devine [8.5.3.a]:

```
------------------------------------------------------------
{Structura unui nod al unui arbore AVL}

TYPE TipRef=^TipNod
        TipNod=RECORD
           cheie:integer;                            [8.5.3.a]
           contor:integer;
           stang,drept:TipRef;
           ech:(-1,0,+1){diferenţa dintre hd şi hs}
         END;
------------------------------------------------------------
```

- Pornind de la **structura nod** definită în secvenţa [8.5.3.a], **inserţia** unui nod se desfăşoară în trei etape:

  1. Se **parcurge** arborele binar, pentru a verifica dacă nu cumva cheia există deja.

  2. Se **inserează** noul nod şi se iniţializează factorul său de echilibru pe valoarea zero.

  3. Se **revine** pe drumul de căutare şi se verifică factorul de echilibru pentru fiecare nod întâlnit, procedându-se la **echilibrare** acolo unde este cazul.

- Această metodă realizează unele verificări redundante deoarece:

  - Odată echilibrul stabilit, **nu** mai este necesară verificarea factorului de echilibru pentru strămoşii nodului

- Cu toate acestea, se va face totuşi uz de ea, deoarece:

  - (1) Este uşor de înţeles.

  - (2) Se poate implementa printr-o **extindere** a procedurilor recursive de căutare şi inserţie a nodurilor în arbori binari ordonaţi, descrise în & 8.3.4.

- Aceste proceduri care includ **operaţia de căutare a unui nod**, datorită formulării lor **recursive**, asigură în manieră implicită "**revenirea de-a lungul drumului de căutare**".

  - Informaţia care trebuie transmisă la revenirea din fiecare pas este cea referitoare **la modificarea înălţimii subarborelui** în care s-a făcut inserţia.

  - Din acest motiv, în lista de parametri ai procedurii de inserţie se introduce parametrul variabil de tip boolean `h`, a cărui valoare "adevărat" semnifică **creşterea înălţimii subarborelui** din care se revine.

- Se presupune că procedura de inserţie revine din **subarborele stâng** la un nod `p^` (vezi fig.8.5.3.h), cu indicaţia că **înălţimea** sa a crescut.

- Se pot distinge trei situaţii referitoare la înălţimea subarborelui **înaintea** respectiv **după** realizarea inserţiei:

    1. $h_S<h_D$, deci `p^.ech=+1`; După inserţie factorul de echilibru devine `p^.ech=0`, ca atare inechilibrul anterior referitor la nodul `p` a fost rezolvat.

    2. $h_S=h_D$, deci `p^.ech=0`; După inserţie factorul de echilibru devine `p^.ech=-1`, în consecinţă greutatea este acum înclinată spre stânga, dar arborele rămâne echilibrat în sensul AVL.

    3. $h_S>h_D$, deci `p^.ech=-1`; ca atare este necesară **reechilibrarea** arborelui.

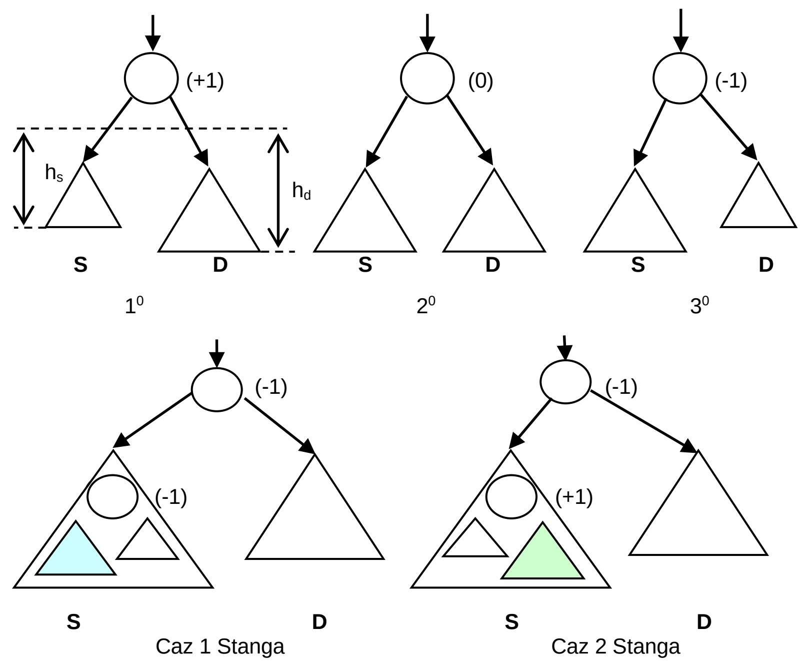


Fig.8.5.3.h. Inserţia în arbori AVL. Cazul Stânga. Schema generală

- În cazul 3^0, **inspectarea** factorului de echilibru al rădăcinii **subarborelui stâng** (`p1^.ech`) conduce la stabilirea cazului **1 Stânga** sau cazul **2 Stânga**.

    - (1) Dacă acest nod are la rândul său înălţimea subarborelui său stâng mai mare ca cea a celui drept, adică factorul de echilibru egal cu `(-1)`, suntem în cazul **1 Stânga.**

    - (2) Dacă factorul de echilibru al acestui nod este egal cu `(+1)` suntem în cazul **2 Stânga** (fig. 8.5.3.h).

    - (3) În această situaţie **nu** poate apare un subarbore stâng a cărui rădăcină are un factor de echilibru nul [Wi76].

- **Operaţia de reechilibrare** constă dintr-o secvenţă de reatribuiri de pointeri.

    - De fapt pointerii sunt schimbaţi ciclic, rezultând fie o **rotaţie simplă** fie o **rotaţie dublă** a două respectiv trei noduri implicate.

* În plus, pe lângă rotirea pointerilor, **factorii de echilibru** respectivi sunt reajustaţi.

* Procedura care realizează acest lucru apare în secvenţa[8.5.3.b]. Principiul de lucru este cel ilustrat în figura 8.5.3.h.

```
--------------------------------------------------------------
{Inserţia unui nod într-un arbore echilibrat AVL}

PROCEDURE InsertEchilibrat(x:TipCheie; VAR p:TipRef;
                           VAR h:BOOLEAN);
  VAR p1,p2:TipRef; {h=fals}

BEGIN
    IF p=NIL THEN
        BEGIN {cuvântul nu e arbore; se inserează}
          new(p); h:=TRUE;
          p^.cheie:=x; p^.contor:=1;
          p^.stang:=NIL; p^.drept:=NIL; p^.ech:=0
        END
      ELSE
        IF x<p^.cheie THEN
            BEGIN
              InsertEchilibrat(x,p^.stang,h);
              IF h THEN {ramura stângă a crescut în
                          înălţime}
                CASE p^.ech OF
                  +1: BEGIN
                         p^.ech:=0; h:=FALSE
                      END;                          [8.5.3.b]
                   0: p^.ech:=-1;
                  -1: BEGIN {reechilibrare}
                         p1:=p^.stang;
                         IF p1^.ech=-1 THEN
                             BEGIN {cazul 1 stânga}
                               p^.stang:=p1^.drept;
                               p1^.drept:=p;
                              p^.ech:=0; p:=p1
                             END
                           ELSE
                             BEGIN {cazul 2 stânga}
                              p2:=p1^.drept;
                              p1^.drept:=p2^.stang;
                              p2^.stang:=p1;
                              p^.stang:=p2^.drept;
                              p2^.drept:=p;
                              IF p2^.ech=-1 THEN
                                  p^.ech:=+1
                                ELSE
                                  p^.ech:=0;
                              IF p2^.ech:=+1 THEN
                                  p1^.ech:=-1
                                ELSE
                                  p1^.ech:=0;
                              p:=p2
                             END;
                         p^.ech:=0; h:=FALSE
```

```
                END
              END {CASE}
            END                                            [8.5.3.b]
          ELSE
            IF x>p^.cheie THEN
              BEGIN
                InsertEchilibrat(x,p^.drept,h);
                IF h THEN {ramura dreapta a crescut
                            în înălţime}
                  CASE p^.ech OF
                    -1: BEGIN
                          p^.ech:=0; h:=FALSE
                        END;
                     0: p^.ech:=+1;
                    +1: BEGIN {reechilibrare}
                          p1:=p^.drept;
                          IF p1^.ech=+1 THEN
                              BEGIN {cazul 1 dreapta}
                                p^.drept:=p1^.stang;
                                p1^.stang:=p;
                               p^.ech:=0; p:=p1
                              END
                            ELSE
                              BEGIN {cazul 2 dreapta}
                               p2:=p1^.stang;
                               p1^.stang:=p2^.drept;
                               p2^.drept:=p1;
                               p^.drept:=p2^.stang;
                               p2^.stang:=p;
                               IF p2^.ech=+1 THEN
                                   p^.ech:=-1
                                 ELSE
                                   p^.ech:=0;
                               IF p2^.ech=-1 THEN
                                   p1^.ech:=+1
                                 ELSE
                                   p1^.ech:=0;
                               p:=p2                [8.5.3.b]
                              END;
                          p^.ech:=0; h:=FALSE
                        END
                  END {CASE}
              END
            ELSE
              BEGIN  {cuvântul există, incrementare contor}
                p^.contor:=p^.contor+1;
              END
    END; {InsertEchilibrat}
------------------------------------------------------------
```

- Procedura `InsertEchilibrat` funcţionează după cum urmează:

  1. Iniţial se parcurge arborele indicat de referinţa `p` pe stânga respectiv pe dreapta după valoarea cheii `x` care se caută. Parcurgerea se realizează prin apeluri recursive ale procedurii `InsertEchilibrat`;

  2. Dacă se ajunge la o referinţă `p=nil` are loc inserţia, cu modificarea lui `h=TRUE` specificând astfel că înălţimea subarborelui a crescut;

3. După o astfel de inserţie se revine din apelul recursiv şi se verifică echilibrul nodului curent realizându-se eventual echilibrarea pe stânga (dacă se revine din stânga) sau pe dreapta (dacă se revine din dreapta).

4. Dacă se găseşte o cheie egala cu `x` se incrementează contorul nodului în cauză.

5. Cu privire la variabila `h` se fac următoarele precizări:
   - Inserţia îl poziţionează pe `h←TRUE;`
   - Revenirile prin noduri cu factorul de echilibru `0` nu îl modifică pe `h`;
   - Reechilibrarea îl poziţionează pe `h←FALSE;`

- Pentru exemplificare se consideră succesiunea de inserţii într-un arbore AVL precizată în figura 8.5.3.i.

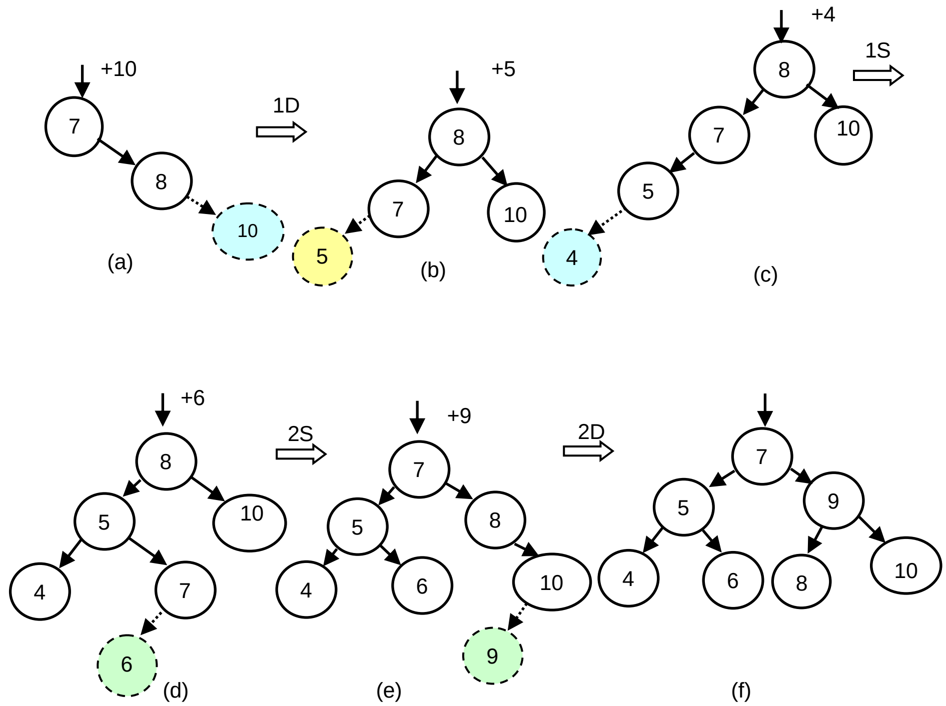


Fig.8.3.5.i. Inserţii succesive într-un arbore echilibrat AVL.

- Se consideră arborele echilibrat AVL (a).

- Inserţia cheii `10` conduce la un arbore dezechilibrat (cazul 1 Dreapta), a cărui echilibrare perfectă se realizează printr-o rotaţie simplă dreapta, fig.8.5.3.i (b).

- Inserţiile nodurilor `5` şi `4` conduc la dezechilibrarea subarborelui cu rădăcina `7`. Echilibrarea sa se realizează printr-o rotaţie simplă (cazul 1 Stânga) (d).

- Inserţia în continuare a cheii `6` produce din nou dezechilibrarea arborelui, a cărui echilibrare se realizează printr-o rotaţie dublă stânga rezultînd arborele (e) (cazul 2 Stânga).

    - În sfârşit, inserţia nodului `9` conduce la cazul 2 Dreapta, care necesită în vederea echilibrării arborelui cu rădăcina `8` o rotaţie dublă care conduce la arborele echilibrat AVL (f).

- În legătură cu **performanţa inserţiei într-un arbore echilibrat AVL** se ridică două probleme:

    1. Dacă toate cele `n!` permutări de `n` chei apar cu **probabilitate egală**, care este **înălţimea probabilă** a **arborelui echilibrat** care se construieşte?

    2. Care este **probabilitatea** ca o inserţie să necesite **reechilibrarea** arborelui?

- Analiza matematică a acestui complicat algoritm este încă o problemă nerezolvată.

- Teste empirice ale înălţimii arborilor generaţi de **algoritmul de inserţie echilibrată** [8.5.3.b.] conduc la valoarea `h=log(n)+c,` unde `c` este o constantă mică `(c≈0.25).`

    - Aceasta înseamnă că în practică, arborii echilibraţi AVL, se comportă **la fel de bine** ca şi **arborii perfect echilibraţi**, fiind însă mai uşor de realizat.

- Testele empirice sugerează de asemenea că în medie, **reechilibrarea** este necesară aproximativ la fiecare **două inserţii.**

    - Atât rotaţiile simple cât şi cele duble sunt echiprobabile.

- Complexitatea operaţiei de reechilibrare sugerează faptul că arborii echilibraţi trebuie utilizaţi de regulă când operaţiile de căutare a informaţiei sunt mult mai frecvente decât cele de inserare.

8.4.3. Suprimarea nodurilor în arbori echilibraţi AVL

- Şi în cazul arborilor echilibraţi AVL, suprimarea este o operaţie mai **complicată** decât inserţia.

- În principiu însă, operaţia de **reechilibrare** rămâne aceeaşi, reducîndu-se la una sau două **rotaţii** la stânga sau la dreapta.

- **Tehnica** care stă la baza suprimării nodurilor în arbori echilibraţi AVL este similară celei utilizate în cazul **arborilor binari ordonaţi** prezentată în &8.3.5.

    - Cazul evident este cel în care, nodul care se suprimă este un **nod terminal** sau are **un singur descendent**.

    - Dacă nodul de suprimat are însă **doi descendenţi**, el va fi înlocuit cu **predecesorul** adică cu cel mai din dreapta nod al subarborelul său stâng.

- Ca şi în cazul inserţiei, se utilizează variabila booleeană `h` a cărei poziţionare pe „valoare adevărată" semnifică **reducerea înălţimii subarborelui**.

    - Reechilibrarea se execută **numai** când `h` este adevărat.

* Variabila h se poziţionează pe adevărat după suprimarea unui nod al structurii, sau dacă reechilibrarea însăşi reduce înălţimea subarborelui.

- Tehnica suprimării nodurilor din arbori echilibraţi AVL este materializată de procedura **SuprimEchilibrat** secvenţa [8.5.4.a]

```
{Suprimarea unui nod într-un arbore echilibrat AVL}

PROCEDURE SuprimEchilibrat(x:TipCheie; VAR p:TipRef;
                           VAR h:BOOLEAN);
  VAR q:TipRef; {h=fals}

  PROCEDURE Echilibru1(VAR p:TipRef: VAR h:BOOLEAN);
    VAR p1,p2:TipRef;
        e1,e2:(-1,0,+1);
    BEGIN {h=adevărat,ramura stânga a devenit mai mică}
      CASE p^.ech OF
        -1: p^.ech:=0;
         0: BEGIN
              p^.ech:=+1; h:=FALSE
            END;
        +1: BEGIN {reechilibrare}                       [8.5.4.a]
              p1:=p^.drept; e1:=p1^.ech;
              IF e1>=0 THEN
                  BEGIN {cazul 1 dreapta}
                    p^.drept:=p1^.stang; p1^.stang:=p;
                    IF e1=0 THEN
                        BEGIN
                          p^.ech:=+1; p1^.ech:=-1;
                          h:=FALSE
                        END
                      ELSE
                        BEGIN
                          p^.ech:=0; p1^.ech:=0
                        END;
                    p:=p1
                  END
                ELSE
                  BEGIN {cazul 2 dreapta}
                    p2:=p1^.stang; e2:=p2^.ech;
                    p1^.stang:=p2^.drept; p2^.drept:=p1;
                    p^.drept:=p2^.sting;
                    p2^.stang:=p;
                    IF e2=+1 THEN
                        p^.ech:=-1
                      ELSE
                        p^.ech:=0;
                    IF e2=-1 THEN
                        p1^.ech:=+1
                      ELSE
                        p1^.ech:=0;
                    p:=p2; p2^.ech:=0
                  END
            END
      END  {CASE}                                       [8.5.4.a]
    END; {Echilibru1}
```

```
PROCEDURE Echilibru2(VAR p:TipRef; VAR h:BOOLEAN);
  VAR p1,p2:TipRef;
      e1,e2:(-1,0,+1);
  BEGIN {h=adevarat,ramura dreapta a devenit mai mică}
    CASE p^.ech OF
      +1: p^.ech:=0;
       0: BEGIN
            p^.ech:=-1; h:=FALSE
          END;
      -1: BEGIN  {reechilibrare}
            p1:=p^.stang; e1:=p1^.ech;
            IF e1<=0 THEN
                BEGIN {cazul 1 stânga}
                  p^.stang:=p1^.drept; p1^.drept:=p;
                  IF e1=0 THEN
                      BEGIN
                        p^.ech:=-1; p1^.ech:=+1;
                        h:=FALSE
                      END
                    ELSE
                      BEGIN
                        p^.ech:=0; p1^.ech:=0
                      END;
                  p:=p1
                END
              ELSE
                BEGIN {cazul 2 stânga}
                  p2:=p1^.drept; e2:=p2^.ech;
                  p1^.drept:=p2^.stang; p2^.stang:=p1;
                  p^.stang:=p2^.drept;
                  p2^.drept:=p;
                  IF e2=-1 THEN
                      p^.ech:=+1
                    ELSE
                      p^.ech:=0;
                  IF e2=+1 THEN
                      p1^.ech:=-1
                    ELSE
                      p1^.ech:=0;
                  p:=p2; p2^.ech:=0
                END
          END
    END {CASE}
  END; {Echilibru2}

PROCEDURE Suprima(VAR r:TipRef; VAR h:BOOLEAN);
  BEGIN {h=false}
    IF r^.drept<>NIL THEN
        BEGIN
          Suprima(r^.drept,h);
          IF h THEN Echilibru2(r,h)
        END
      ELSE
        BEGIN                                     [8.5.4.a]
          q^.cheie:=r^.cheie;
          q^.contor:=r^.contor;
          r:=r^.stang; h:=TRUE
```

```
        END
   END; {Suprima}

BEGIN {SuprimaEchilibrat}
  IF p=NIL THEN
      BEGIN
        WRITE('cheia nu e IN arbore'); h:=FALSE
      END
    ELSE
      IF x<p^.cheie THEN
          BEGIN
            SuprimaEchilibrat(x,p^.stang,h);
            IF h THEN Echilibru1(p,h)
          END
        ELSE
          IF x>p^.cheie THEN
              BEGIN
                SuprimaEchilibrat(x,p^.drept,h);
                IF h THEN Echilibru2(p,h)
              END
            ELSE
              BEGIN {suprima p^}
                q:=p;
                IF q^.drept=NIL THEN            [8.5.4.a]
                    BEGIN
                      p:=q^.stang; h:=TRUE
                    END
                  ELSE
                    IF q^.stang=NIL THEN
                        BEGIN
                          p:=q^.drept; h:=TRUE
                        END
                      ELSE
                        BEGIN
                          Suprima(q^.stang,h);
                          IF h THEN Echilibru1(p,h)
                        END;
                {DISPOSE(q)}
              END
END; {SuprimaEchilibrat}
```

- În cadrul procedurii **SuprimEchilibrat** se definesc trei proceduri:
    - (1) **Echilibru1** care se aplică când **subarborele stâng** s-a redus din înălţime;
    - (2) **Echilibru2** care se aplică când **subarborele drept** s-a redus din înălţime;
    - (3) **Suprima** – are rolul procedurii `Supred` la arbori binari ordonaţi:
        - (1) Găseşte şi înlocuieşte nodul de suprimat cu predecesorul său.
        - (2) Suprimă predecesorul.

- (3) În plus procedura **`Suprima`** realizează eventualele reechilibrari la revenirea recursivă pe drumul parcurs în arbore.

- Mersul procedurii **`SuprimEchilibrat`** este normal:

  - (1) Se parcurge recursiv arborele AVL pentru căutarea cheii de suprimat, (apeluri ale procedurii **`SuprimEchilibrat`** pe stânga sau pe dreapta după cum cheia care se caută e mai mică respectiv mai mare decât cea a nodului curent);

  - (2) Când se găseşte cheia ea se suprimă exact ca şi la arborii binari ordonaţi:

    - Cazul 1 fiu: se rezolvă prin suprimare directă;

    - Cazul 2 fii: se apelează procedura **`Suprima`** descrisă mai sus.

  - (3) Este important de reamintit faptul că după fiecare revenire dintr-un apel recursiv se verifică valoarea lui `h` şi dacă este necesar se apelează procedura corespunzătoare de echilibrare.

- Modul de lucru al procedurii, este prezentat în figura 8.5.4.a.

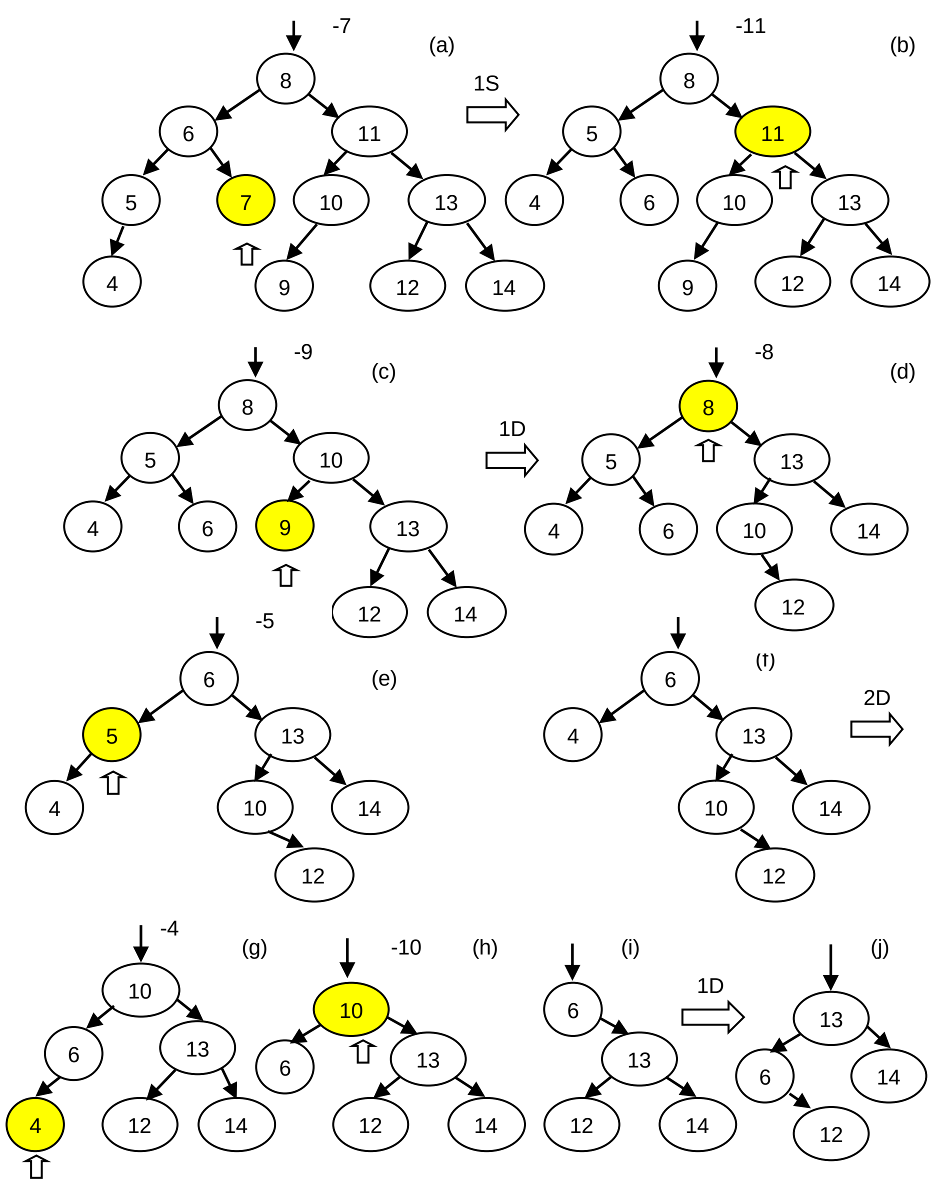


Fig.8.5.4.a. Suprimări succesive în arbori echilibraţi AVL

- Dându-se arborele binar echilibrat (a), se suprimă în mod succesiv nodurile având cheile `7,11,9,8,5,4` şi `10,` rezultând arborii (b)...(j).

  - Suprimarea cheii `7` este simplă însă conduce la subarborele dezechilibrat cu rădăcina `6`. Reechilibrarea acestuia presupune o rotaţie simplă (cazul 1 stânga).

  - Suprimarea nodului `11` nu ridică probleme.

  - Reechilibrarea devine din nou necesară după suprimarea nodului `9`; de data aceasta, subarborele având rădăcina `10`, este reechilibrat printr-o rotaţie simplă dreapta (cazul 1 dreapta).

  - Suprimarea cheii `8` este imediată

  - Deşi nodul `5` are un singur descendent, suprimarea sa presupune o reechilibrare mai complicată bazată pe o dublă rotaţie (cazul 2 dreapta).

  - Ultimul caz, cel al suprimării nodului cu cheia `10` presupune înainte de reechilibrare, înlocuirea acestuia cu cel mai din dreapta element al arborelui său stâng (nodul cu cheia `6`).

- În cazul arborilor binari echilibraţi, suprimarea unui nod se realizează în **cel mai defavorabil caz** cu performaţa O*(log n)*.

- Diferenţa esenţială dintre **inserţie** şi **suprimare** în cazul **arborilor echilibraţi AVL** este următoarea:

  - În urma unei **inserţii**, reechilibrarea se realizează prin una sau două rotaţii (a două sau trei noduri).

  - **Suprimarea** poate necesita în cel mai defavorabil caz, o rotaţie simplă sau dublă, a fiecărui nod situat pe drumul de căutare.

- În realitate, testele experimentale indică faptul suprinzător că:

  - (1) În cazul **inserţiei** reechilibrarea devine necesară aproximativ la fiecare **a 2-a** inserţie.

  - (2) În cazul **suprimării** reechilibrarea devine necesară aproximativ la fiecare a **5-a** suprimare.

  - (3) Există însă unele situaţii speciale la suprimare, în care reechilibrarea este necesară în fiecare din nodurile situate pe drumul de căutare.