8.9. Arbori multicăi

8.9.1. Generalităţi

- Până în prezent au fost studiate cu predilecţie structuri arbore în care fiecare nod avea **cel mult** doi descendenţi.

- Desigur, acest lucru este pe deplin justificat dacă spre exemplu, se doreşte să se reprezinte **descendenţa unei persoane** din punctul de vedere al strămoşilor,

    - În acest caz, fiecărei persoane i se asociază cei doi părinţi ai săi.

- Dacă problema se abordează însă punctul de vedere al **urmaşilor**, atunci o familie poate să aibă mai mult de doi copii, rezultând astfel noduri cu mai multe ramuri (gradul arborelui este mai mare ca 2).

    - Structurile care conţin astfel de noduri se numesc, după cum s-a mai precizat, **arbori generalizaţi**.

- Astfel de structuri ridică însă unele **probleme** în implementare.

    - Spre **exemplu**, în situaţia anterioară, dacă se cunoaşte **numărul de copii**, atunci referinţele la aceştia pot fi memorate într-un **tablou**, care devine o componentă a nodului afectat persoanei respective.

    - Dacă **numărul de copii** variază în limite largi, aceasta poate conduce la utilizarea ineficientă a memoriei.

- O altă soluţie, mai eficientă, este aceea de a crea cu **referinţele** la copii, **o listă liniară** al cărei început se păstrează în nodul părinte.

- Această structură de date, poate fi şi mai mult complicată prin introducerea unor **componente suplimentare** în nodul corespunzător unei persoane, pentru a putea reprezenta spre exemplu diferite **grade de rudenie**.

    - O astfel de structură poate progresa rapid spre o **bază de date relaţională** care poate îngloba mai mulţi arbori în ea.

    - **Algoritmii** care operează asupra unei astfel de structuri, depind în mod **intim** de **structurile de date** definite, precizarea unor reguli şi tehnici cu caracter general în acest caz fiind lipsită de sens.

    - Este însă evident faptul că modalităţile de **reprezentare a arborilor** sugerate până în prezent **nu** corespund într-o manieră eficientă unor astfel de structuri.

- În acest context se pot utiliza **arborii multicăi**, care reprezintă o categorie specială de **arbori generalizaţi**. Specificaţia acestora este următoarea:

- (1) Este vorba despre construcţia şi exploatarea **arborilor de foarte mari dimensiuni**.

- (2) Arbori în care se fac frecvent **inserţii** şi **suprimări**.

- (3) Arbori pentru care **dimensiunile memoriei centrale** sunt **insuficiente** sau a căror memorare vreme îndelungată în memoria sistemului de calcul este prea **costisitoare**.

- (4) Arbori în care **operaţia de căutare** trebuie sa fie cât mai performantă.

- În acest scop a fost dezvoltată o **tehnică specială de implementare** a **arborilor multicăi**.

  - Să presupune că nodurile unui arbore trebuiesc memorate într-o **memorie secundară**, spre exemplu pe un **disc magnetic**.

  - **Structurile de date dinamice** definite în acest curs se pretează foarte bine şi acestui scop:

    - Astfel, **pointerii** care de regulă indică **adrese de memorie** pot indica în acest caz **adrese de disc**.

    - Utilizând spre exemplu un **arbore binar echilibrat** cu 10^6 noduri, căutarea unei chei necesită aproximativ $\log_2 10^6 \approx 20$ paşi.

    - Deoarece în acest caz, fiecare pas necesită un **acces** la disc (care este lent) se impune cu necesitate **o altă organizare** pentru **reducerea numărului de accese**.

  - **Arborii multicăi** reprezintă o **soluţie perfectă** a acestei probleme.

- Se porneşte de la următoarea constatare:

  - Este cunoscut faptul că după realizarea **accesului** (de regulă mecanic) la un anumit element de pe disc (**pistă**) sunt uşor accesibile (electronic) un întreg grup de elemente (**sectoarele corespunzătoare**).

- Aceasta **sugerează** faptul că:

  - Un arbore poate fi divizat în **subarbori**.

  - **Subarborii** pot fi memoraţi pe disc ca unităţi la care accesul se realizează foarte rapid.

  - Aceşti subarbori se numesc **pagini**.

- Considerând că accesul la **fiecare pagină** presupune un acces disc, dacă spre exemplu se plasează 100 noduri pe o pagină, atunci căutarea în arborele cu 10^6 noduri presupune $\log_{100} 10^6 = 3$ accese disc în loc de 20.

  - În situaţia în care arborele creşte aleator, în cel mai **defavorabil caz** (când degenerează în lista liniară) numărul de accese poate ajunge însă la 10^4.

- Ca atare este evident faptul că şi în cazul **arborilor multicăi**, trebuie avut în vedere un **mecanism de control al creşterii acestora**.

- Există mai multe variante de implementare a **arborilor multicăi**.

- Una dintre cele mai cunoscute modalităţi de implementare a arborilor multicăi o reprezintă **arborii-B**.

8.9.2. Arbori-B

8.9.2.1. Definire

- Din discuţia asupra **mecanismului** de control al creşterii arborilor multicăi, **arborii perfect echilibraţi** se exclud de la început din cauza costului ridicat al echilibrării.

- Un **criteriu** foarte potrivit în acest scop a fost postulat de **R.Bayer** în 1970 şi anume:

    - Fiecare **pagină** a arborelui multicăi, cu excepţia uneia, conţine între n şi $2n$ noduri, unde n este o constantă dată.

        - Astfel într-un **arbore** cu N noduri, a cărui dimensiune maximă a unei pagini este cuprinsă între n şi $2n$ noduri, în cel mai rău caz, se fac $\log_n N$ **accese la pagini** pentru a căuta o cheie precizată.

        - Factorul de **utilizare al memoriei** este de cel puţin 50%, deoarece orice pagină este cel puţin pe jumătate plină.

        - În plus schema preconizată presupune **algoritmi simpli** pentru **căutare**, **inserţie** şi **suprimare** în comparaţie cu alte metode.

- Structurile de date propuse de **Bayer** se numesc **arbori-B** iar n se numeşte **ordinul arborelui-B**.

- **Arborii-B** se bucură de următoarele **proprietăţi**:

    - (1) Fiecare pagină a arborelui-B conţine **cel mult** $2n$ noduri (chei).

    - (2) Fiecare pagină, cu excepţia paginii rădăcină conţine **cel puţin** n noduri.

    - (3) Fiecare pagină este fie:

        - O **pagină terminală** - caz în care **nu** are descendenţi.

        - O **pagina interioară** - caz în care are $m+1$ descendenţi unde m este numărul de chei din pagină $(n \leq m \leq 2n)$.

    - (4) Toate **paginile terminale** sunt la acelaşi **nivel**.

- În figura 8.9.2.1.a apare reprezentat un arbore-B de ordinul 2 cu 3 niveluri.

    - Toate paginile conţin 2, 3 sau 4 noduri cu excepţia paginii rădăcină care conţine unul singur.

    - Toate paginile terminale apar pe nivelul 3.

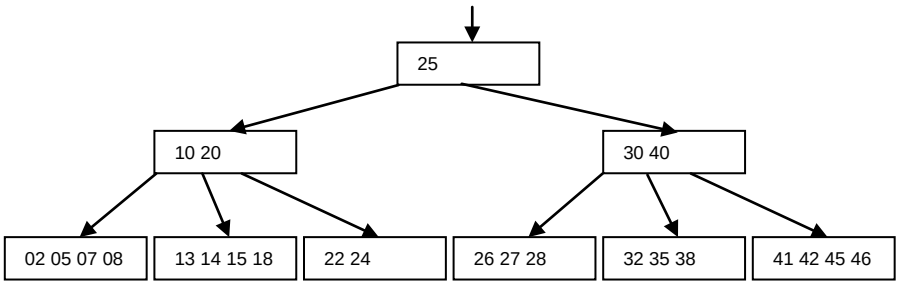

Fig.8.9.2.1.a. Arbore-B de ordinul 2

- Dacă această structură se **liniarizează** prin inserarea cheilor descendenţilor printre cheile strămoşilor lor, cheile nodurilor apar în **ordine crescătoare** de la stânga la dreapta.

- Acestă structurare reprezintă o **extensie naturală** a **structurii arbore binar ordonat** şi ea stă la baza **metodei de căutare** ce va fi prezentată în continuare.

8.9.2.2. Căutarea cheilor în arbori-B

- Se consideră o **pagină** a unui arbore-B de forma prezentată în fig.8.9.2.2.a şi o **cheie** dată x.

- Presupunând că pagina a fost transferată în memoria centrală a sistemului de calcul, pentru **căutarea cheii** x printre cheile $k_1, \ldots, k_m$ aparţinând paginii, se poate utiliza o **metodă de căutare convenţională**.

    - Se face următoarea **precizare** importantă: cheile k_i sunt **ordonate crescător** în cadrul paginii.

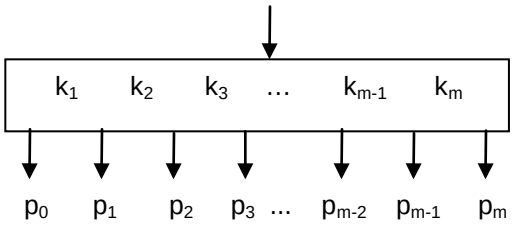

Fig8.9.2.2.a. Pagină cu m chei a unui arbore-B

- Astfel, dacă m este mare se poate utiliza **căutarea binară**, altfel, **căutarea liniară**.

- Trebuie subliniat faptul că **timpul de căutare** în memoria centrală este probabil **neglijabil** în comparaţie cu **timpul de transfer** al unei pagini din memoria secundară în cea primară.

- Dacă cheia x **nu** se găseşte în **pagina curentă** este valabilă una din următoarele situaţii:

  - (1) $k_i<x<k_{i+1}$, pentru $1 \le i < m$. Căutarea continuă în pagina p_i.

  - (2) $k_m<x$. Căutarea continuă în pagina p_m.

  - (3) $x<k_1$. Căutarea continuă în pagina p_o.

- Dacă **pointerul la pagina** desemnată de algoritmul de mai sus este **vid,** atunci **nu** există nici un nod cu cheia x şi **căutarea este terminată**, adică s-a ajuns la baza arborelui într-o **pagină terminală**.

8.9.2.3. Inserţia nodurilor în arbori B

- În ceea ce priveşte **inserţia nodurilor** în arborii-B de ordinul n, aceasta se realizează implicit într-o **pagină terminală**.

  - Există în principiu două situaţii:

- (1) Nodul trebuie inserat într-o pagină conţinând $m<2n$ noduri.

  - În acest caz inserţia se realizează simplu în pagina respectivă **inserând** cheia corespunzătoare la **locul potivit** în secvenţa ordonată a cheilor.

    - Spre **exemplu** inserţia cheii cu numărul 15 în arborele-B din fig.8.9.2.3.a (a).

- (2) Nodul trebuie inserat într-o pagina care este **plină**, adică conţine deja 2n chei.

  - În acest caz structura arborelui se modifică conform exemplului prezentat în fig.8.9.2.3.a (b).

  - Astfel, spre **exemplu** inserţia cheii cu numărul 22 în arborele-B se realizează în următorii paşi:

    - (1) Se caută cheia 22 şi se descoperă că ea lipseşte, iar inserţia în pagina destinaţie C este imposibilă deoarece aceasta este plină (conţine 2n chei).

    - (2) Pagina C se **scindează** în două prin alocarea unei noi pagini D.

    - (3) Cele 2n+1 chei ale paginii C sunt distribuite după cum urmează: primele n (cele mai mici) în pagina C, ultimele n (cele mai mari) în pagina D iar cheia mediană este translatată pe nivelul inferior în pagina strămoş A.

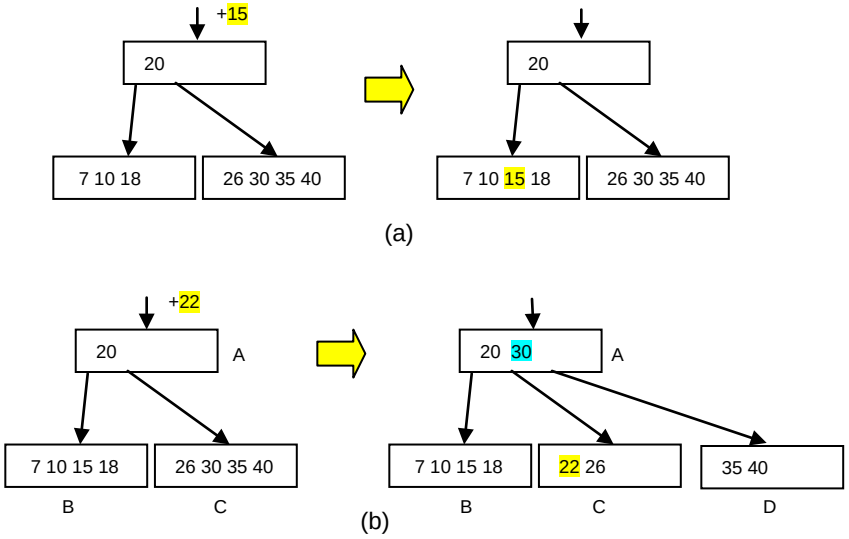

Fig.8.9.2.3.a Inserţia nodurilor în arbori-B. Exemple

- Această schemă păstrează toate proprietăţile caracteristice ale aborilor-B.
    - Se observă că paginile rezultate din scindare au exact n noduri.
- Desigur este posibil ca scindarea să se **propage** spre nivelurile superioare ale structurii, în cazul extrem până la rădăcină.
    - Aceasta este de fapt **singura** posibilitate ca un arbore-B să crească în înălţime.
- Maniera de creştere a unui astfel de arbore este **inedită**: el creşte de la nodurile terminale spre rădăcină.
- În continuare se va dezvolta un **program** care materializează conceptele prezentate.
- Pornind de la proprietatea de **propagare a scindării paginii,** se consideră că formularea **recursivă** a algoritmului este cea mai convenabilă.
- Structura generală a programului este similară programului de inserţie în **arbori echilibraţi** (vezi &8.5.3).
- Pentru început se precizează structurile de date utilizate în implementarea arborilor-B [8.9.2.3.a].

```
-----------------------------------------------------------------
/*Structură de date pentru arbori-B - varianta C*/

int n=...; /*n este ordinul arborelui B*/
int nn=2*n;

typedef struct nod {
          int cheie;  /*cheie nod*/
          struct pagina * p; /*referinţa la pagina cu chei
                               mai mari*/
          int contor; /*contor chei*/
       } NOD;                              /*[8.9.2.3.a]*/
```

```
typedef struct pagina {
          int m; /*nr curent de elemente în pagină*/
          struct pagina* p0; /*referinţa la p0*/
          NOD elem[nn];
       } PAGINA;

typedef PAGINA* refPagina;
-------------------------------------------------------------
{Structură de date pentru arbori-B - varianta PASCAL}

CONST nn=2*n;

TYPE  RefPagina=^pagina;
      indice=0..nn;

      nod=RECORD
            cheie:integer;
            p:RefPagina;
            contor:integer
          END;                                    [8.9.2.3.a]

      pagina=RECORD
               m:indice; {nr curent de noduri în pagină}
               p0:RefPagina; {referinţă la prima pagină}
               elem:ARRAY[1..nn] OF nod
             END;
-------------------------------------------------------------
```

- Referitor la structura `nod`:
  - Câmpul `cheie` precizează cheia nodului respectiv.
  - Câmpul `p` indică pagina urmaş care conţine **chei mai mari** decât cheia nodului în cauză.
  - Câmpul `contor` este utilizat ca **numărător de accese**.
- Referitor la structura `pagina`:
  - Fiecare pagină oferă spaţiu pentru `2n` noduri.
  - Variabila `m` indică **numărul curent de noduri** memorate în pagina respectivă.
  - Tabloul `elem` memorează nodurile din pagina curentă în ordinea crescătoare a cheilor.
  - `p0` indică pagina urmaş cu chei mai mici decât cea mai mică cheie din pagină.
- Deoarece `m≥n`, (cu excepţia rădăcinii), se garantează o utilizare a memoriei de cel puţin 50 %.

- În programul [8.9.2.3.d] apare **algoritmul de căutare şi inserţie** materializat de procedura **Cauta**.

  - Structura de principiu a procedurii **Cauta** este asemănătoare cu cea a algoritmului de căutare în arbori binari, cu **excepţia** faptului că decizia de ramificaţie în arbore **nu** e binară ci este cea specifică **arborilor-B** (&8.9.2.2).

  - Căutarea în interiorul unei pagini este o **căutare binară** efectuată în tabloul `elemente` al paginii curente.

- Forma **pseudocod** a procedurii **Cauta** apare în secvenţa [8.9.2.3.b].

```
-------------------------------------------------------------
/*Schiţa de principiu a procedurii de căutare în arbori-B -
varianta pseudocod*/

Subprogram Cauta(int x, refPagina a, boolean *h, NOD *v)

  NOD u;
  daca(a==null)
     |/*x nu este în arbore*/
     |*se crează nodul v;
     |*i se atribuie cheia x;
     |*se face h=TRUE indicând pasarea nodului v spre
     | părintele său;
     |□ /*daca*/
    altfel
     |/*se caută x în pagina curentă a*/
     |*căutare binară într-un tablou liniar;
     |daca(găsit)
     |    *incrementează contorul de accese;
     |  altfel                                    [8.9.2.3.b]
     |    |Cauta(x,urmas,&h,&u);
     |    |daca(*h) Insereaza;   /*nodul u a fost pasat
     |    |                       Spre părintele său*/
     |    |□ /*altfel*/
     |□ /*altfel*/
/*Cauta*/
-------------------------------------------------------------
(Schiţa de principiu a procedurii de căutare în arbori-B -
varianta PASCAL}

PROCEDURE Cauta(x:integer; a:RefPagina; VAR h:boolean;
                VAR v:nod);
  VAR u:nod;

  BEGIN
    IF a=NIL THEN
        BEGIN {x nu este în arbore}
          {*se creează un nod nou v}
          {*se atribuie cheia x nodului v şi se pune h pe
           adevarat indicând pasarea nodului v spre rădăcina}
        END
      ELSE
        BEGIN {se caută x în pagina curentă a^}
          {*căutare binară într-un tablou liniar}
          IF gasit THEN
```

```
            {*incrementează contorul de accese}
          ELSE                                          [8.9.2.3.b]
            BEGIN
              Cauta(x,urmas,h,u);
              IF h THEN Insereaza {nodul u care se pasează}
            END
        END
  END; {Cauta}
```

- **Algoritmul de inserţie** este formulat ca şi o procedură aparte (procedura **Insereaza**) care este activată după ce procesul de căutare indică faptul că unul din noduri este **pasat** spre **pagina părinte**.

- Acest lucru este precizat de către valoarea "adevărat" a parametrului h returnat de procedura **Cauta**.

    - Dacă h este adevărat, parametrul u indică nodul care trebuie pasat paginii părinte, în direcţia rădăcinii.

- Se precizează faptul că procesul de inserţie începe într-o **pagină ipotetică**, de tip "**nod special**" situată virtual **sub nivelul terminal**.

    - Noul nod creat, este transmis via parametrul u paginii terminale pentru adevărata inserţie.

- Pornind de la aceste precizări, structura de principiu a procedurii **Inserează** apare în secvenţa [8.9.2.3.c].

```
/*Schiţa de principiu a inserţiei nodurilor în arbori-B -
varianta pseudocod*/

Subprogram Insereaza
  daca((numărul de noduri m al paginii a)<nn)
      *se inserează nodul u în pagina a la locul potrivit
        şi se face h=fals;
    altfel                                      [8.9.2.3.c]
      |*se crează o nouă pagină b;
      |*se insereaza cheia u la locul potrivit intre
      | cheile paginii a;
      |*se redistribuie cheile paginii a, primele n
      | pe a, ultimele n pe b;
      |*se pasează nodul v=u conţinând cheia mediană spre
      | nivelul inferior;
      |*h rămâne poziţionat pe valoarea true;
      |□ /*altfel*/
/*Insereaza*/
```

```
{Schiţa de principiu a inserţiei nodurilor în arbori-B -
varianta PASCAL}

PROCEDURE Insereaza
  BEGIN
    IF (numărul de noduri a^.m al paginii a^)<2n THEN
        *se insereaza nodul u în pagina a^ la locul potrivit
          şi se face h=fals
```

```
      ELSE
        BEGIN                                          [8.9.2.3.c]
          * se creează o nouă pagină b^;
          * se redistribuie cheile paginii a^, primele n pe
           a^, ultimele n pe b^;
          *se pasează nodul v=u conţinând cheia mediană
            spre nivelul superior;
          *h rămâne poziţionat pe valoarea true
        END
  END; {Insereaza}
-------------------------------------------------------------
```

- Dacă parametrul `h` devine adevărat după apelul procedurii **`Cauta`** din **programul principal**, acesta indică necesitatea **scindării** paginii **rădăcină**.

- Deoarece **pagina rădăcină** are un rol **special**, acest proces trebuie programat separat.

    - El constă de fapt din alocarea unei **noi pagini rădăcină** şi inserarea unui singur nod transmis prin parametrul `u`.

- Implementarea inserţiei în arbori-B apare în programul **`ArboriB`**, secvenţa [8.9.2.4.a] din paragraful următor, procedurile **`Cauta`** respectiv **`Insereaza`**.

- În legătură cu **inserţia nodurilor** în arbori-B se fac următoarele observaţii:

    - (1) Deoarece paginile arborelui sunt alocate în memoria secundară, este necesar un mecanism pentru realizarea **transferului paginii curente** în memoria primară.

    - (2) Întrucât fiecare activare a procedurii **`Cauta`** implică o alocare de pagină în memoria principală, vor fi necesare cel mult $k=\log_n N$ apeluri recursive, unde `n` este ordinul arborelui-B iar `N` numărul total de noduri.

    - (3) Prin urmare dacă arborele conţine `N` noduri, în memoria principală trebuie să încapă **cel puţin** $k=\log_n N$ pagini.

    - (4) Acesta este unul din factorii care limitează **dimensiunea** `2n` a paginii.

- În cadrul secvenţei [8.9.2.4.a] instrucţia **`WITH`** rezolvă aceste aspecte.

    - (1) În primul rând, indică faptul cunoscut că referirile sunt relative la pagina `a`.

    - (2) În al doilea rând, deoarece paginile sunt alocate în memoria secundară, instrucţia **`WITH`** presupune şi realizarea **transferului paginii vizate** în memoria primară.

- De fapt în memorie trebuie să existe mai mult de `k` pagini din cauza scindărilor care apar.

    - O consecinţă a acestei maniere de lucru este faptul că **pagina rădăcină** trebuie să fie **tot timpul** în memoria principală, ea fiind punctul de pornire al tuturor activităţilor.

- Un alt **avantaj** al structurii de date de tip arbore-B se referă la **actualizarea simplă** şi **eficientă**, în mod secvenţial a întregii structuri.

    - În acest caz, fiecare pagină este adusă în memorie exact odată.

- Se observă de asemenea faptul că **arborii-B** cresc relativ greu în înălţime, inserţia unei noi pagini respectiv adăugarea unui nou nivel se realizează după inserţia unui număr semnificativ de chei.

- În figura 8.9.2.3.b apare urma execuţiei programului la inserarea următoarei succesiuni de chei: 20; 40, 10, 30, 15; 35, 7, 26, 18, 22; 5; 42, 13, 46, 27, 8, 32; 38, 24, 45, 25;

- Punctul şi virgula precizează momentele la care au avut loc alocări de pagini.

- Inserţia ultimei chei (25) cauzează două scindări şi alocarea a 3 noi pagini.

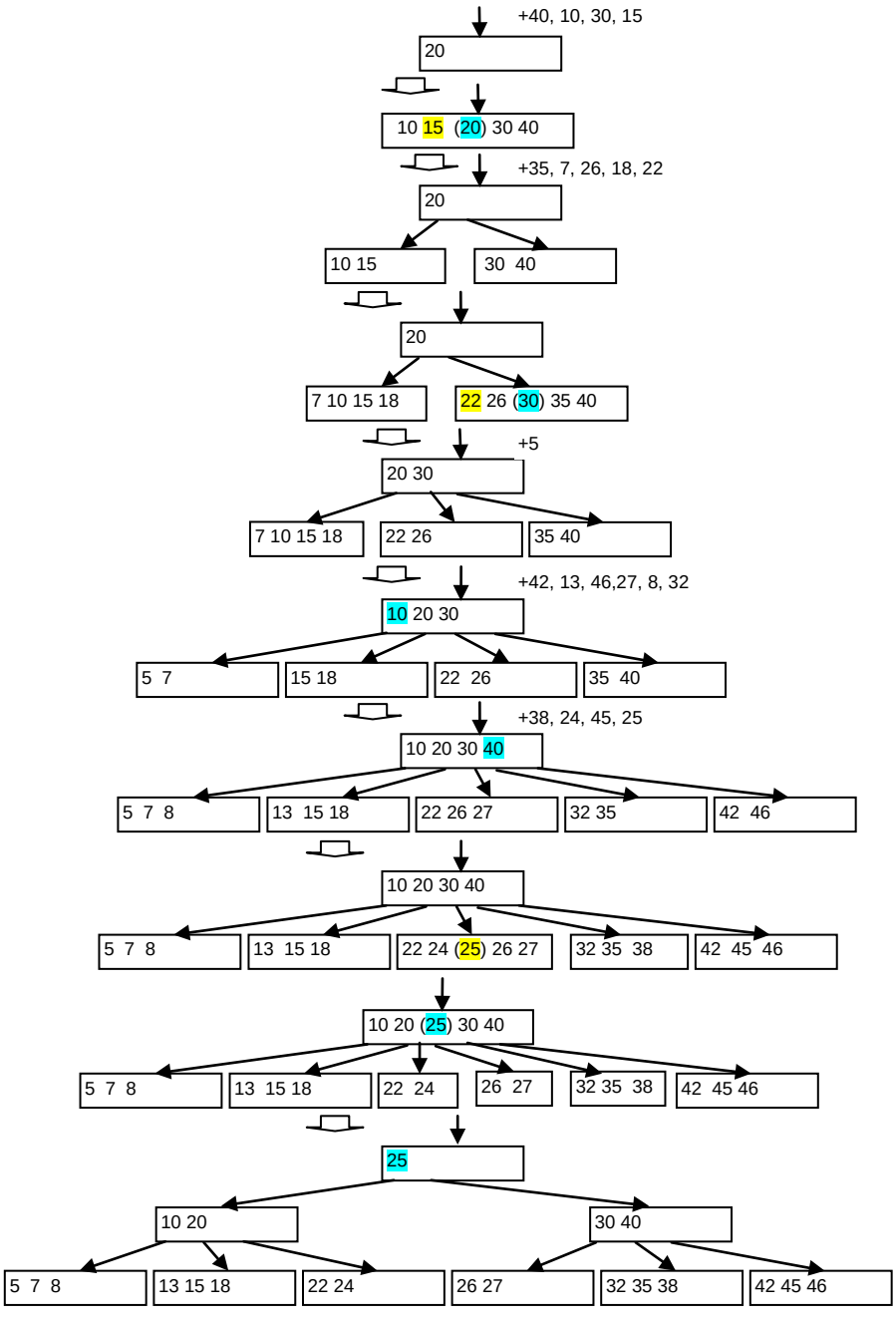

Fig.8.9.3.2.b. Construcţia unui arbore-B de ordinul 2

8.9.2.4. Suprimarea nodurilor în arbori-B

- Principial, suprimarea nodurilor în arborii-B, este o operaţie simplă, la nivel de detaliu însă ea devine complicată.

- Se disting două situaţii:

    - (1) Nodul se găseşte într-o **pagină terminală**, caz în care suprimarea este imediată.

    - (2) Nodul se găseşte într-o **pagină internă**.

        - În acest caz nodul în cauză trebuie **înlocuit** cu unul dintre cele două noduri **adiacente**, care sunt în pagini terminale şi prin urmare pot fi uşor suprimate.

        - De regulă înlocuirea se realizează cu **predecesorul** nodului respectiv.

- **Căutarea cheii adiacente** este similară celei utilizate la suprimarea nodurilor într-un **arbore binar ordonat** (vezi &8.3.5).

    - (1) Se înaintează spre **pagina terminală** `P` de-a lungul celor mai din **dreapta** pointeri ai **subarborelui stâng** al cheii de suprimat.

    - (2) Se înlocuieşte nodul de suprimat cu **cel mai din dreapta nod** al lui `P` .

    - (3) Se reduce dimensiunea lui `P` cu 1.

- **Reducerea dimensiunii paginii** trebuie să fie urmată de **verificarea** numărului `m` de noduri din pagină.

    - Dacă `m<n` apare fenomenul numit "**subdepăşire**" care este indicat de valoarea adevărat a lui `h`.

    - În acest caz soluţia de rezolvare este aceea de a "**împrumuta**" un nod de la paginile vecine.

        - Întrucât această operaţie presupune aducerea unei pagini vecine `(Q)` în memoria principală - o operaţie relativ costisitoare - se preferă exploatarea la maxim a acestei situaţii prin împrumutarea mai multor noduri.

        - Astfel, în mod uzual, nodurile se distribuie în mod egal în paginile `P` şi `Q`, proces numit **echilibrare**.

- În situaţia în care **nu** poate fi împrumutat nici un nod din pagina `Q` - aceasta având dimensiunea minimă `n`, paginile `P` şi `Q` care împreună au `2n-1` noduri se **contopesc** într-una singură.

    - Acest lucru presupune extragerea nodului **median** din **pagina părinte** a lui `P` şi `Q`, gruparea tuturor nodurilor într-una din pagini, adăugarea nodului median şi ştergerea celeilalte pagini.

    - Acesta este procesul invers scindării paginii, proces care poate fi urmărit în figura 8.9.2.3.b, la suprimarea cheii cu numărul `22`.

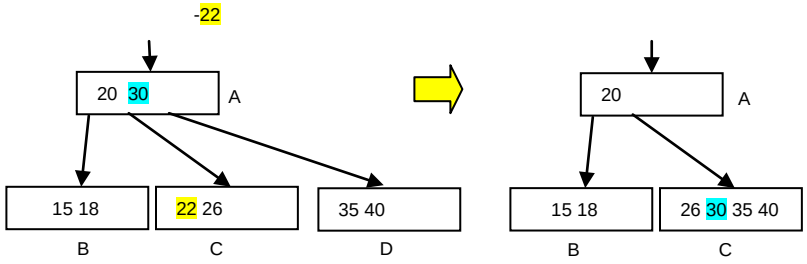

Fig.8.9.2.3.b Suprimarea nodurilor în arbori-B

    - Din nou, extragerea cheii din mijloc din pagina strămoş, poate determina **subdepăşirea**, situaţie care poate fi rezolvată fie prin echilibrare fie prin contopire.

    - În caz extrem, procesul de contopire se poate propaga până la **rădăcină**.

    - Dacă rădăcina este redusă la dimensiunea 0, ea dispare cauzând **reducerea înălţimii** arborelui-B.

    - Aceasta este de fapt **singura cale de reducere** a dimensiunii unui arbore-B.

- În figura 8.9.2.4.a se prezintă evoluţia unui arbore-B, rezultată din suprimarea următoarei secvenţe de chei: 45, 25, 24; 38, 32; 8, 27, 46, 13, 42; 5, 22, 18, 26; 7, 35, 15.

    - Şi în acest caz punctul şi virgula precizează momentele la care sunt eliberate pagini.

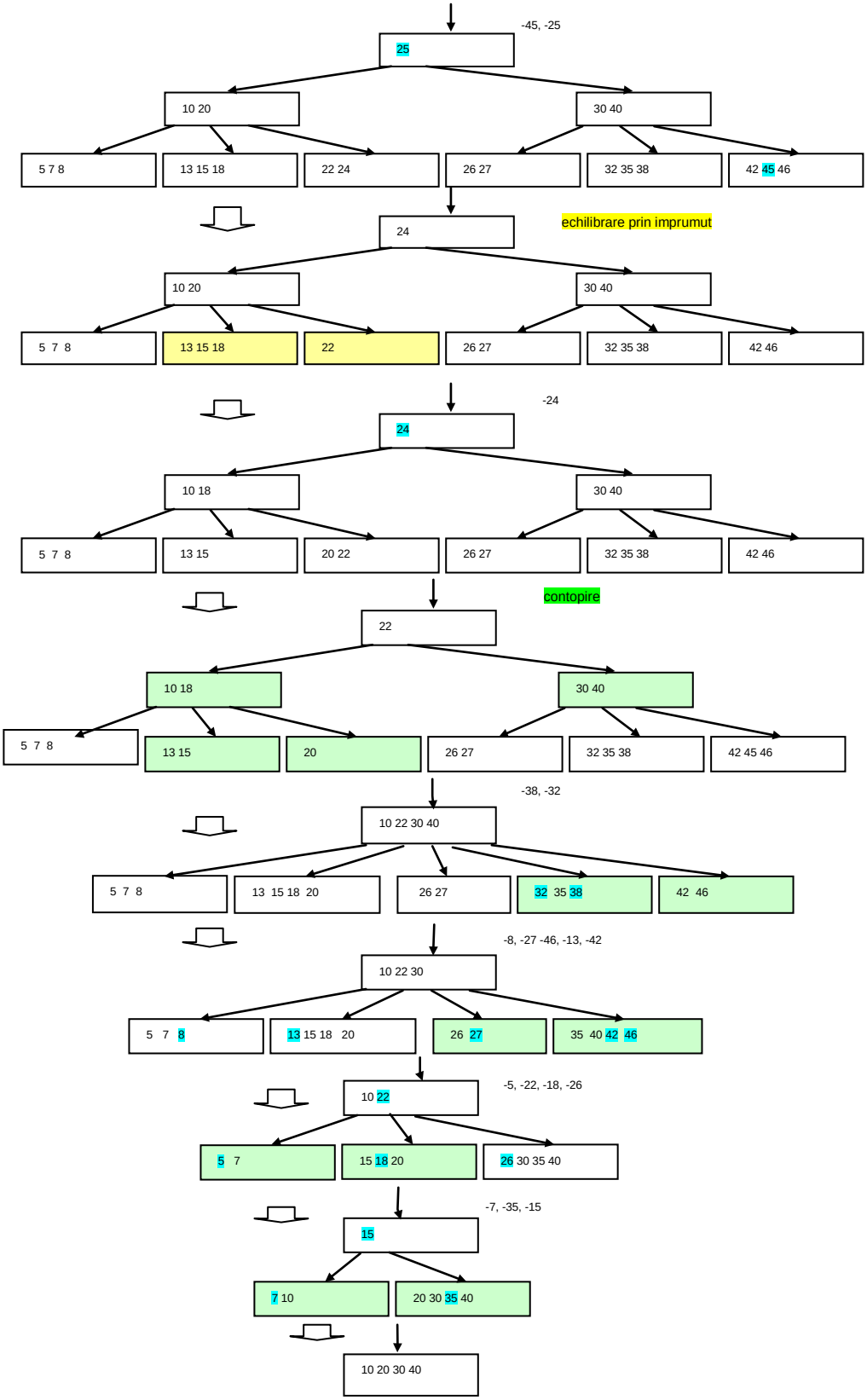

Fig.8.9.4.2.a. Suprimarea nodurilor în arbori-B

- Algoritmul de suprimare este inclus în programul din secventa [8.9.2.4.a] conceput ca şi exemplu de aplicaţie pentru utilizarea arborilor-B.

- Referitor la **suprimare**, în cadrul aplicaţiei sunt dezvoltate mai multe proceduri şi anume.

- (1) Procedura **`Suprima`**:

  - Realizează parcurgerea arborelui-B căutând cheia de suprimat în manieră recursivă.

  - Dacă găseşte cheia într-o pagină **terminală**, o suprimă, mută cheile mai mari cu o poziţie spre stânga şi îl asignează pe `h=m<n`, semnalând dacă este cazul **subdepăşirea**.

  - În toate situaţiile se verifică valoarea lui `h` şi se apelează procedura **`Subdepasire`** dacă `h=TRUE`.

- (2) Procedura **`Supr:`**

  - Caută în manieră recursivă, **predecesorul** cheii de suprimat, substituie cheia de suprimat cu predecesorul găsit, îl suprimă şi dacă este cazul semnalează subdepăşire.

  - La revenirea din fiecare apel recursiv, verifică pe `h` şi apelează dacă este cazul procedura **`Subdepasire.`**

- (3) Procedura **`Subdepasire`**:

  - Rezolvă problema echilibrării respectiv a contopirii paginilor adiacente funcţie de situaţia concretă (pagina din dreapta respectiv cea din stânga paginii subdepăşite indicate de referinţa `a`).

- (4) Procedura **`TipArb`**:

  - Parcurge arborele B şi afişează într-o manieră specifică structura acestuia.

```
{Program arbori-B Varianta PASCAL}

PROGRAM ArboriB;

  {Cautare,inserţie şi suprimare în arbori B}

  CONST n=2; nn=4; {dimensiune pagina}

  TYPE  refPagina=^pagina;

        TipNod=RECORD
          cheie:integer;
          p:refPagina; {referinţă la pagina cu chei mai mari}
          contor:integer
        END;

        pagina =RECORD
          m:0..nn;        {m= numărul de noduri}
          p0:refPagina;  {referinţa la pagina zero}
          e: ARRAY[1..nn] OF TipNod
```

```
      END;

VAR radacina,q:refPagina; x:integer;          [8.9.2.4.a]
    h:boolean; u:nod;

PROCEDURE Cauta(x:integer; VAR a:refPagina; VAR h:boolean;
                VAR v:nod);

   {Caută cheia x în arborele B de rădăcina a. Dacă o
   găseşte incrementează contorul,altfel inserează un nod
   nou cu cheia x şi contor:=1. Dacă un nod trebuie pasat
   spre un nivel interior, el este atribuit lui v. h:=TRUE
   semnifică că arborele a devenit mai înalt}

  VAR k,s,d:integer; q:refPagina; u:nod;

  PROCEDURE Insereaza;
    VAR i:integer; b:refPagina;
    BEGIN {inserează pe u în dreapta lui a^.elem[d] }
      WITH a^ DO
        BEGIN
          IF m<nn THEN
              BEGIN
                m:=m+1; h:=false;
                FOR i:=m DOWNTO d+2 DO e[i]:=e[i-1];
                e[d+1]:=u
              END
            ELSE
              BEGIN {pagina a^ e plină; scindează pagina şi
                     atribuie nodul median lui v}
                new(b); {se creează o pagină nouă b}
                IF d<=n THEN
                    BEGIN
                      IF d=n THEN{inserează pe u}
                          v:=u                  [8.9.2.4.a]
                        ELSE
                          BEGIN
                            v:=e[n];
                            FOR i:=n DOWNTO d+2 DO
                              e[i]:=e[i-1];
                            e[d+1]:=u
                          END;
                      FOR i:=1 TO n DO
                        b^.e[i]:=a^.e[i+n]
                    END
                  ELSE
                    BEGIN
                      d:=d-n; v:=e[n+1];
                      FOR i:=1 TO d-1 DO
                        b^.e[i]:=a^.e[i+n+1];
                      b^.e[d]:=u;
                      FOR i:=d+1 TO n DO
                       b^.e[i]:=a^.e[i+n];
                    END;
                m:=n; b^.m:=n; b^.p0:=v.p; v.p:=b
              END
        END {WITH}
    END; {Insereaza}
```

```
   BEGIN {caută cheia x în pagina a^; h:=fals}
     IF a=NIL THEN
         BEGIN {nodul cu cheia x nu este în arbore}
           h:=true;
           WITH v DO   {se creează noul nod v}
             BEGIN
               cheie:=x; contor:=1; p:=NIL
             END
         END
       ELSE
         WITH a^ DO
           BEGIN
             s:=1; d:=m; {căutare binară}
             REPEAT
               k:=(s+d) DIV 2;
               IF x<=e[k].cheie THEN d:=k-1;
               IF x>=e[k].cheie THEN s:=k+1;
             UNTIL d<s;
             IF s-d>1 THEN {găsit, incrementare contor}
                 BEGIN
                   e[k].contor:=e[k].contor+1;
                   h:=false
                 END
               ELSE
                 BEGIN {nodul nu e în această pagină}
                   IF d=0 THEN
                       q:=p0
                     ELSE
                       q:=e[d].p;
                   Cauta(x,q,h,u);
                   IF h THEN Insereaza
                 END
           END
   END; {Cauta}

PROCEDURE Suprima(x:integer; VAR a:refPagina;
   VAR h:boolean);

{Caută şi suprimă nodul cu cheia x din arborele-B având
 rădăcina a. Dacă o pagină devine subdimensionată se
 încearcă fie echilibrarea cu o pagină adiacentă (dacă este
 posibil), fie contopirea. h:=TRUE semnifică că pagina a
 este subdimensionată}

   VAR i,k,s,d:integer; q:refPagina;

   PROCEDURE Subdepasire(VAR c,a:refPagina; VAR s1:integer;
     VAR h:boolean);
   {a=pagina subdepăşită; c=pagina strămoş}
     VAR b:refPagina; i,k,mb,mc:integer;
     BEGIN
       mc:=c^.m; {h=true; a^.m=n-1}
       IF s1<mc THEN
           BEGIN {b este pagina din dreapta lui a}
             s1:=s1+1;
             b:=c^.e[s1].p; mb:=b^.m; k:=(mb-n+1) DIV 2;
             {k=nr. de noduri disponibile în pagina
```

```
                adiacentă b}
              a^.e[n]:=c^.e[s1]; a^.e[n].p:=b^.p0;
              IF k>0 THEN
                  BEGIN {mută k noduri din b în a }
                    FOR i:=1 TO k-1 DO a^.e[i+n]:=b^.e[i];
                    c^.e[s1]:=b^.e[k]; c^.e[s1].p:=b;
                    b^.p0:=b^.e[k].p; mb:=mb-k;
                    FOR i:=1 TO mb DO b^.e[i]:=b^.e[i+k];
                    b^.m:=mb; a^.m:=n-1+k; h:=false
                  END
                ELSE
                  BEGIN {contopeşte paginile a şi b}
                    FOR i:=1 TO n DO a^.e[i+n]:=b^.e[i];
                    FOR i:=s1 TO mc-1 DO c^.e[i]:=c^.e[i+1];
                    a^.m:=nn; c^.m:=mc-1  {Dispose(b)}
                  END
            END
          ELSE
            BEGIN {b este pagina din stânga lui a}
              IF s1=1 THEN b:=c^.p0 ELSE b:=c^.e[s1-1].p;
              mb:=b^.m+1; k:=(mb-n) DIV 2;
              IF k>0 THEN
                  BEGIN {mută k noduri din pagina b în a}
                    FOR i:=n-1 DOWNTO 1 DO
                      a^.e[i+k]:=a^.e[i];
                    a^.e[k]:=c^.e[s1]; a^.e[k].p:=a^.p0;
                    mb:=mb-k;
                    FOR i:=k-1 DOWNTO 1 DO
                      a^.e[i]:=b^.e[i+mb];
                    a^.p0:=b^.e[mb].p;
                    c^.e[s1]:=b^.e[mb]; c^.e[s1].p:=a;
                    b^.m:=mb-1; a^.m:=n-1+k; h:=false;
                  END
                ELSE
                  BEGIN {contopire pagini a şi b}
                    b^.e[mb]:=c^.e[s1]; b^.e[mb].p:=a^.p0;
                    FOR i:=1 TO n-1 DO b^.e[i+mb]:=a^.e[i];
                    b^.m:=nn; c^.m:=mc-1  {Dispose(a)}
                  END
            END
      END; {Subdepasire}

  PROCEDURE Supr(VAR p:refPagina; VAR h:boolean);
    VAR q:refPagina; {a şi k variabile globale}
    BEGIN
      WITH p^ DO
        BEGIN
          q:=e[m].p;
          IF q<>NIL THEN
              BEGIN
                Supr(q,h);
                IF h THEN Subdepasire(p,q,m,h)
              END
            ELSE
              BEGIN
                p^.e[m].p:=a^.e[k].p; a^.e[k]:=p^.e[m];
                m:=m-1; h:=m<n
              END
```

```
      END
  END; {Supr}

  BEGIN {Suprima}
    IF a=NIL THEN
        BEGIN
          WriteLn('   cheia nu exista in arbore');
          h:=false
        END
      ELSE
        WITH a^ DO
          BEGIN {căutare binară}
            s:=1; d:=m;
            REPEAT
              k:=(s+d) DIV 2;
              IF x<=e[k].cheie THEN d:=k-1;
              IF x>=e[k].cheie THEN s:=k+1;
            UNTIL s>d;
            IF d=0 THEN
                q:=p0
              ELSE
                q:=e[d].p;
            IF s-d>1 THEN
                BEGIN {găsit; se şterge (suprimă) e[k]}
                  IF q=NIL THEN
                      BEGIN {a este o pagină terminală}
                        m:=m-1; h:=m<n;
                        FOR i:=k TO m DO e[i]:=e[i+1];
                      END
                    ELSE
                      BEGIN{a nu este o pagină terminală}
                        Supr(q,h);
                        IF h THEN Subdepasire(a,q,d,h)
                      END
                END
              ELSE {cautare pe pagina urmatoare}
                BEGIN
                  Suprima(x,q,h);
                  IF h THEN Subdepasire(a,q,d,h)
                END
          END
  END; {Suprima}

PROCEDURE TipArb(p:refPagina; l:integer);
  VAR i:integer;
  BEGIN                                    [8.9.2.4.a]
    IF p<>NIL  THEN
      WITH p^ DO
        BEGIN
          FOR i:=1 TO l DO Write('   ');
          FOR i:=1 TO m DO Write(e[i].cheie);
          WriteLn;
          TipArb(p0,l+1);
          FOR i:=1 TO m DO TipArb(e[i].p,l+1)
        END
  END; {TipArb}

BEGIN {programul principal}
```

```
    radacina:=NIL; Read(x);
    WHILE x<>0 DO
      BEGIN
        WriteLn('cauta cheia  ',x);
        Cauta(x,radacina,h,u);
        IF h THEN
          BEGIN {inserează noua pagină de bază}
            q:=radacina; new(radacina);
            WITH radacina^ DO
              BEGIN
                m:=1; p0:=q; e[1]:=u
              END;
          END;
          TipArb(radacina,1); Read(x)
      END;
    Read(x);
    WHILE  x<>0 DO
      BEGIN
        WriteLn('sterge cheia  ',x);
        Suprima(x,radacina,h);
        IF h THEN
          BEGIN {pagina de bază s-a redus ca dimensiune}
            IF radacina^.m=0 THEN
              BEGIN
                q:=radacina; radacina:=q^.p0
                {Dispose(q)}
              END
          END;
        TipArb(radacina,1); Read(x)
      END
  END.
-------------------------------------------------------------
```

- Analiza performanţei arborilor-B pune în evidenţă o puternică **dependenţă** a dimensiunii `n` a paginii, faţă de caracteristicile memoriei şi de cele ale sistemului de calcul.

    - Ca şi o observaţie se poate menţiona faptul că la inserţia nodurilor în arborii-B, **scindarea** paginii poate fi amânată, încercând în prealabil **echilibrarea** paginilor alăturate.

- Diferite variante de **arbori B** sunt discutate de către Knuth [Kn76].

8.9.3. Arbori-B binari

- O categorie aparte de arbori-B o reprezintă cei de ordinul 1 (`n=1`), care se mai numesc "**arbori-B binari**" sau "**arbori BB**".

- Întrucât paginile unui astfel de arbore conţin **unul** sau **două noduri**, utilizarea lor în contextul reprezentării unor masive de date în memoria secundară a sistemului de calcul este în afara discuţiei din cauza risipei de memorie.

    - Aproximativ 50 % din pagini vor conţine un singur element.

- În continuare, studiul acestei categorii de arbori se va aborda în accepţiunea **rezidenţei** lor **integrale** în memoria centrală.

- Conform **definiţiei** arborilor-B:

    - (1) Toate **paginile terminale** ale unui arbore-B binar apar la **acelaşi nivel**.

    - (2) **Paginile interioare** conţin `1` sau `2` chei şi în consecinţă `2` respectiv `3` descendenţi.

- Din acest motiv **arborii-B binari** se mai numesc şi "**arbori 2-3**".

- Pentru reprezentarea paginilor unui astfel de arbore se pot utiliza **tablouri liniare** sau **pointeri**.

    - Din considerente de ocupare a memoriei se preferă **pointerii**.

- Fiecare **pagină** a unui arbore 2-3 este de fapt o listă înlănţuită de `1` sau `2` noduri.

- Deoarece **fiecare pagină** are cel mult trei descendenţi, există posibilitatea **combinării** pointerilor ce indică **descendenţii** cu cei ce realizează **înlănţuirea** în cadrul listei, după cum rezultă din figura 8.9.3.a.

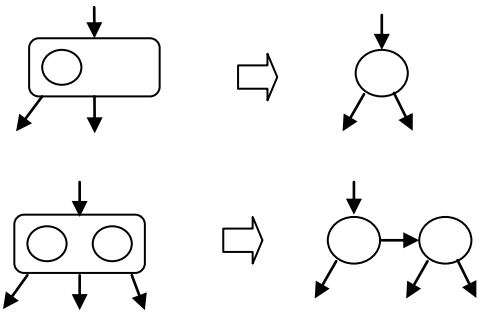

Fig.8.9.3.a. Reprezentarea arborilor-B binari

- Astfel noţiunea de **pagină** dispare, nodurile apărând într-o structură asemănătoare unui arbore binar obişnuit.

- Totuşi trebuie precizat faptul că în acest caz, se definesc **două categorii de pointeri**:

    - Pointerii care se referă la **descendenţi** (cei verticali).

    - Pointerii care se referă la **fraţi** (cei orizontali).

- Deoarece **numai** pointerul pe dreapta poate fi orizontal, este suficientă o variabilă booleană indicator (`orizontal`) în cadrul unui nod care să precizeze acest fapt.

- În aceste condiţii se pot utiliza următoarele structuri de date [8.9.3.a].

```
-------------------------------------------------------------
/*Structură de date pentru reprezentarea unui nod al unui
arbore-B binar - varianta C*/

typedef struct tip_nod
```

```
        {
            /*diferite campuri*/
            char cheie;
            struct tip_nod* stang;                    [8.3.3.a]
            struct tip_nod* drept;
            boolean orizontal;
        } NOD_BB;

typedef tip_nod * ref_tip_nod_BB;
-------------------------------------------------------------
{Structură de date pentru reprezentarea unui nod al unui
arbore-B binar - varianta Pascal}

TYPE RefNod=^Nod
       Nod=RECORD
          cheie:integer;
          {alte câmpuri}                          [8.9.3.a]
          stang,drept:RefNod;
          orizontal:boolean
        END;
-------------------------------------------------------------
```

- Acest mod de organizare asigură o **lungime maximă a drumului de căutare** în arbore $p = 2\lceil \log_2 N \rceil$.

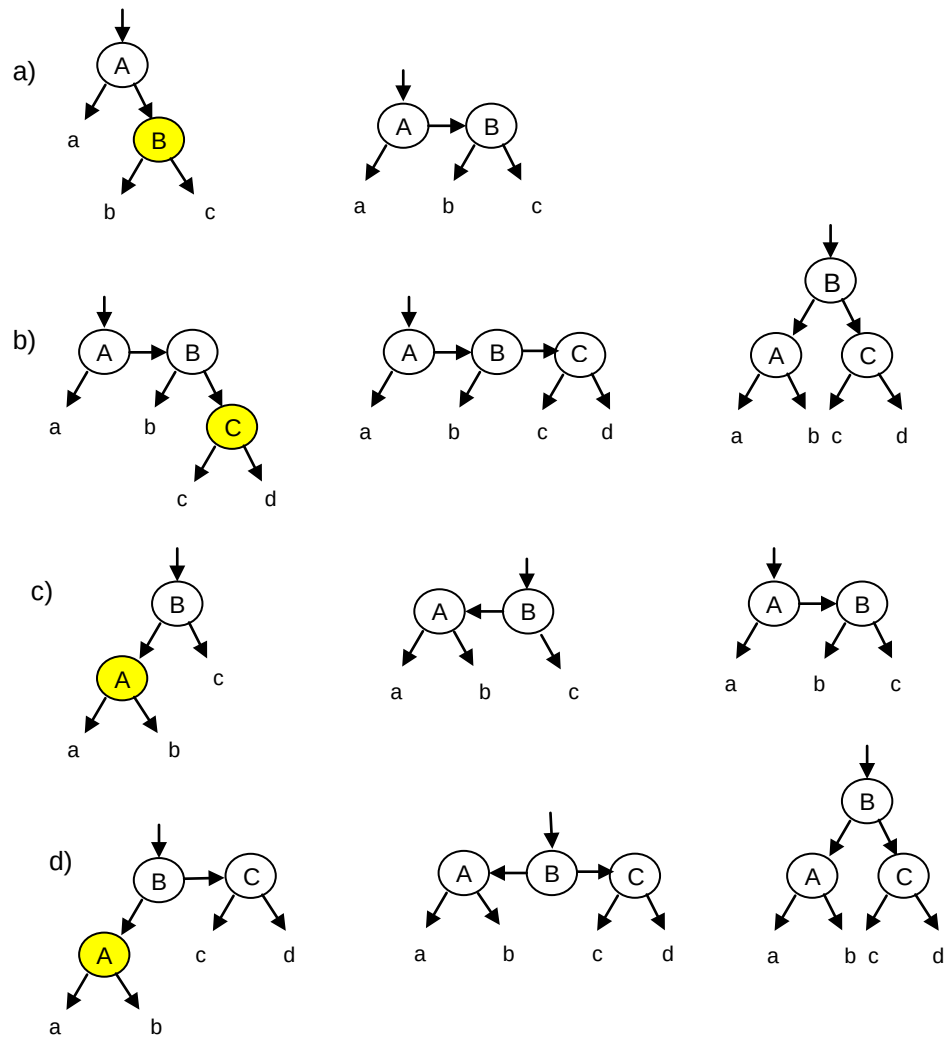

Fig.8.9.3.b. Inserţia nodurior în arbori-B binari

- În ceea ce priveşte **inserţia** unui nod nou într-un arbore-B binar, se disting patru situaţii posibile, după cum creşte **subarborele** său **stâng** sau **subarborele** său **drept** (fig.8.9.3.b).

- Se reaminteşte faptul că arborii-B cresc spre rădăcină şi că toate nodurile lor terminale sunt la acelaşi nivel.

- (a) Cazul (a). Se referă la situaţia în care **subarborele drept** al nodului A creşte, iar a este singura cheie din pagina (ipotetică) corespunzătoare.

    - Nodul B care se adaugă va deveni fratele drept al lui A, pointerul drept vertical al celui din urmă devenind orizontal.

- (b) Cazul (b). Simpla modificare a tipului pointerului **nu** este posibilă dacă A are frate drept.

    - În această situaţie se obţine o pagină cu 3 noduri, care se scindează.

    - Nodul B din mijloc este trecut în sus pe nivelul superior.

- (c) Cazul (c). Dacă subarborele stâng al unui nod B a crescut în înălţime şi B este singur în pagină (cazul (c)), atunci A poate fi adus în aceeaşi pagină.

    - A devine frate stânga al lui B, element care contravine definiţiei, întrucât un pointer stânga nu poate fi orizontal.

    - Se modifică rădăcina arborelui astfel încât să-l indice pe A, iar pointerul orizontal dreapta al lui A îl va indica pe B.

- (d) Cazul (d). Dacă însă B are frate drept, inserţia lui A face necesară scindarea paginii. B este transmis în sus pe nivelul superior.

    - A şi C devin descendenţii nodului B.

- Se precizează faptul că în cadrul **procesului de căutare**, **nu** se face nici o diferenţă între pointerii orizontali şi cei verticali.

8.9.3.1. Arbori-B binari simetrici

- Se observă însă în figura 8.9.3.b o **diferenţiere** referitoare la tratarea diferită a cazurilor de creştere a subarborelui drept (cazul (a)) respectiv a celui stâng (cazul (c)), element care conferă structurii de arbore-B binar un **caracter asimetric**.

    - Evitarea acestei asimetrii a condus la conceptul de **arborii-B binari simetrici** (arbori **BBS**).

- **Arborii-B binari simetrici** sunt arbori-B în care atât pointerul stâng cât şi cel drept pot fi de tip "**orizontal**" sau "**vertical**".

- Din acest motiv, fiecare nod necesită în acest caz două variabile booleene pentru precizarea naturii celor doi pointeri respectiv `OS` şi `OD`.

- În medie **procesul de căutare** în astfel de arbori este **mai eficient**, în schimb algoritmii de inserţie şi suprimare sunt mai complicaţi.

- În continuare se abordează în detaliu doar studiul **inserţiei** nodurilor în arbori BBS, pentru care se disting patru cazuri (fig.8.9.3.1.a).

    - Simetria este evidentă.

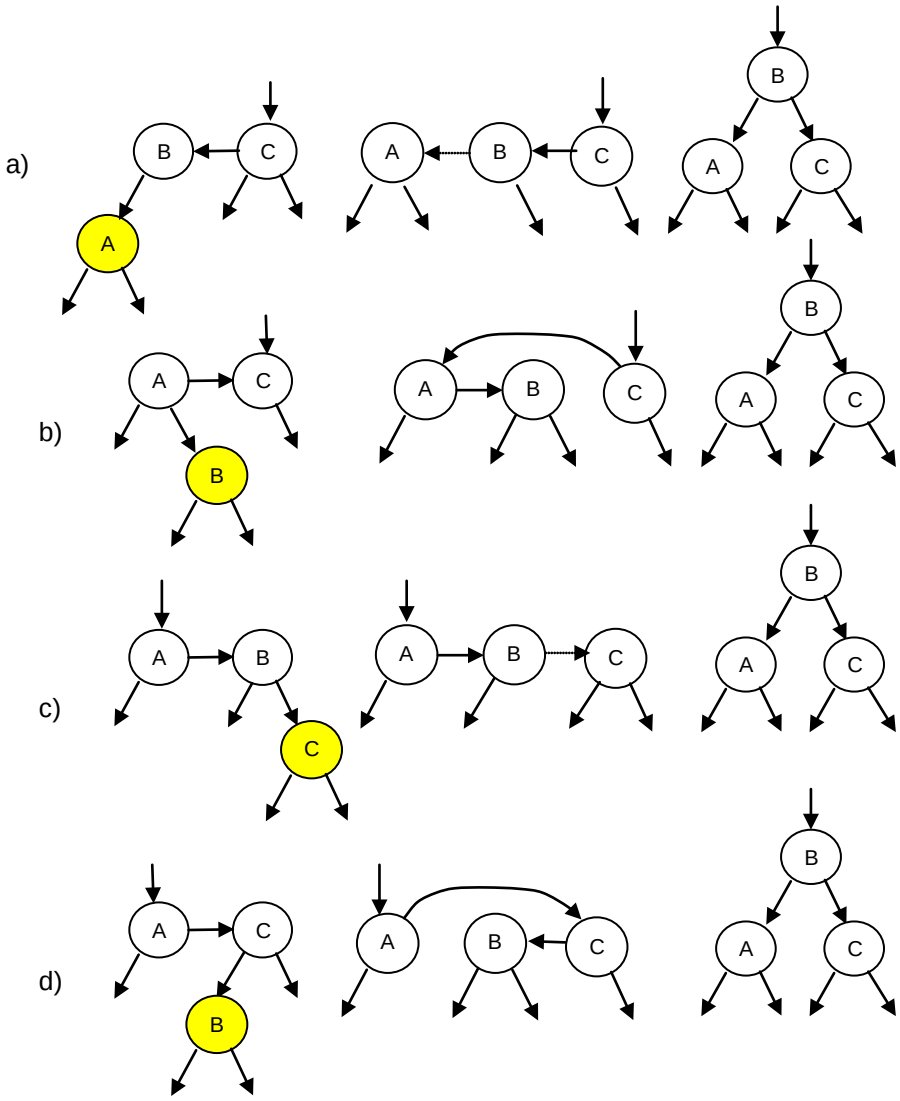

Fig.8.9.3.1.a. Inserţia nodurilor în arbori-B binari simetrici. (a) Cazul 1 Stânga, (b) Cazul 2 Stânga, (c) Cazul 1 Dreapta, (d) Cazul 2 Dreapta.

- Se precizează faptul că, situaţia în care creşte un subarbore a unui nod `A` fără frate, se tratează simplu:

    - Nodul rădăcină al subansamblului respectiv devine fratele lui `A`.

    - Această situaţie **nu** apare reprezentată în figură.

- Cazurile prezentate în figura 8.9.3.1.a conduc la **depăşirea paginii** şi în consecinţă la **scindare.**

    - Pentru fiecare din aceste cazuri din figura 8.9.3.1.a:

        - Coloana din dreapta precizează **situaţia iniţială**.

        - Coloana din mijloc precizează **situaţia intermediară** rezultată în urma ridicării nodului în "pagină", urmată de creşterea numărului de subarbori.

        - Coloana din dreapta **situaţia finală** rezultată în urma rearanjării structurii (scindării).

- În continuare, **se renunţă la conceptul de pagină** care de altfel a stat la baza arborilor-B.

- Ceea ce interesează în continuare este **limitarea lungimii maxime a drumului de căutare** la $2\log_2 N$.

    - Pentru aceasta este necesar ca pe **nici un drum** să **nu** apară **doi pointeri orizontali succesivi.**

    - Această restricţie **nu** interzice însă existenţa unor noduri având **ambii pointeri orizontali** (unul spre stânga, celălalt spre dreapta).

- Cu aceste precizări, un **arbore B binar simetric** se defineşte ca fiind structura arbore care satisface următoarele proprietăţi:

    - (1) Fiecare nod conţine o cheie şi cel mult doi pointeri de subarbori.

    - (2) Fiecare pointer este fie orizontal, fie vertical.

    - (3) Nu există doi pointeri orizontali succesivi în nici un drum de căutare.

    - (4) Toate nodurile terminale apar la acelaşi nivel.

- Din această definiţie rezultă că cel mai lung drum de căutare **nu** poate depăşi dublul înălţimii arborelui ($2\log_2 N$).

- În figura 8.9.3.1.b se prezintă maniera de dezvoltare a arborilor BBS.

    - Secvenţele de chei care se inserează corespunzător celor patru situaţii prezentate în figură sunt următoarele:

        - (a) 1, 2 ; 3 ; 4, 5, 6 ; 7 ;
        - (b) 5, 4 ; 3 ; 1, 2, 7, 6 ;
        - (c) 6, 2 ; 4 ; 1, 7, 3, 5 ;
        - (d) 4, 2, 6 ; 1, 7 ; 3, 5 ;

        - Punctul şi virgula precizează momentul realizării reprezentării.

        - Prima dintre situaţii (a) este prezentată mai detaliat, pentru înţelegerea mai facilă a procedurii de reechilibrare.

- Se remarcă două **caracteristici fundamentale**:
  - (1) Toate **nodurile terminale** apar la acelaşi nivel.
  - (2) **Echilibrarea** se realizează dacă apar doi pointeri orizontali succesivi indicând acelaşi sens.

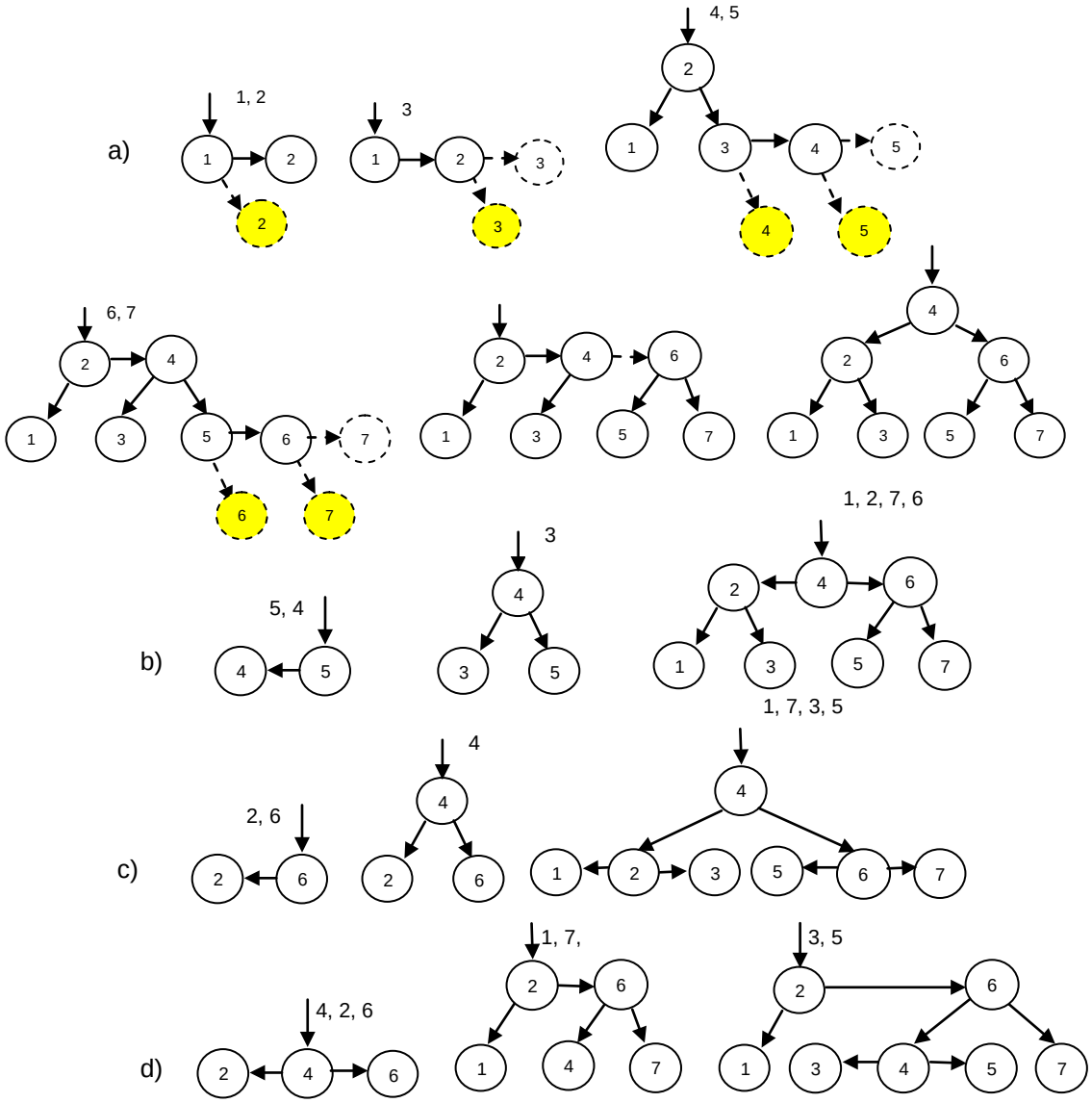

Fig.8.9.3.1.b. Dezvoltarea arborilor-B binari simetrici.

- În continuare se prezintă structurile de date utilizate la definirea unui nod al unui **arbore-B binar simetric** (secvenţa [8.9.3.1.a] precum şi algoritmul care realizează construcţia unui arbore-B binar simetric (secvenţa [8.9.3.1.b]).

```
-------------------------------------------------------------
/*Structură de date pentru reprezentarea unui nod al unui
arbore-B binar simetric - varianta C*/

typedef struct tip_nod
        {
             /*diferite campuri*/
             char cheie;
             int contor;
```

```
            struct tip_nod* stang;       /*[8.9.3.1.a]*/
            struct tip_nod* drept;
            boolean os,od;
        } NOD_BBS;

typedef tip_nod * ref_tip_nod_BBS;
```

{Structura de date nod arbore-B binar simetric – varianta PASCAL}

```
  TYPE nod=RECORD
            cheie: integer;
            contor: integer;                    [8.9.3.1.a]
            stang,drept: refNod;
            os,od: boolean
          END;
```

{Construcţia unui arbore nod arbore-B binar simetric - Varianta PASCAL}

```
PROCEDURE Cauta(x:integer; VAR p:refNod; VAR h:integer);

{Caută cheia x într-un arbore-B binar simetric. Dacă o găseşte îi incrementează contorul asociat. Dacă nu o găseşte crează un nod nou, inserează cheia şi funcţie de situaţie restructurează arborele}

  VAR p1,p2:refNod;
  BEGIN
    IF p=NIL THEN
        BEGIN {cheia nu e în arbore; se inserează}
          new(p); h:=2;
          WITH p^ DO
            BEGIN
              cheie:=x; contor:=1; stang:=NIL;
              drept:=NIL; os:=false; od:=false
            END
        END
      ELSE
        IF x<p^.cheie THEN {parcurgere subarbore stâng}
            BEGIN
              Cauta(x,p^.stang,h);
              IF h<>0 THEN
                IF p^.os THEN
                    BEGIN
                      p1:=p^.stang; h:=2; pq.os:=false;
                      IF p1^.os THEN
                          BEGIN {Cazul 1 Stânga}
                            p^.stang:=p1^.drept;
                            p1^.drept:=p; p1^.os:=false;
                            p:=p1
                          END
                        ELSE
                          IF p1^.od THEN
                            BEGIN {Cazul 2 Stânga}
                              p2:=p1^.drept;
                              p1^.od:=false;
                              p1^.drept:=p2^.stang;
```

```
                              p2^.stang:=p1;
                              p^.stang:=p2^.drept;
                              p2^.drept:=p;
                              p:=p2
                            END
                      END
                    ELSE
                      BEGIN
                        m:=h-1;
                        IF h<>0 THEN p^.os:=true
                      END                              [8.9.3.1.b]
              END
            ELSE
              IF x>p^.cheie THEN {parcurgere subarbore drept}
                  BEGIN
                    Cauta(x,p^.drept,h);
                    IF h<>0 THEN
                      IF p^.od THEN
                          BEGIN
                            p1:=p^.drept; h:=2;
                            p^.od:=false
                            IF p1^.od THEN
                                BEGIN {Cazul 1 Dreapta}
                                  p^.drept:=p1^.stang;
                                  p1^.stang:=p;
                                  p1^.od:=false;
                                  p:=p1
                                END
                              ELSE
                                IF p1^.os THEN
                                  BEGIN {Cazul 2 Dreapta}
                                    p2:=p1^.sting;
                                    p1^.os:=false;
                                    p1^.stang:=p2^.drept;
                                    p2^.drept:=p1;
                                    p^.drept:=p2^.stang;
                                    p2^.stang:=p; p:=p2
                                  END
                          END
                        ELSE
                          BEGIN
                            h:=h-1;
                            IF h<>0 THEN p^.od:=true;
                          END
                    END
                  ELSE   {cheia există în arbore}
                    BEGIN
                      p^.contor:=p^.contor+1: h:=0
                    END
  END; {Cauta}
-------------------------------------------------------------
```

- Procedura recursivă **Cauta** se bazează pe schema clasică a inserţiei în arbori binari ordonaţi (vezi &8.3.4.).

- Parametrul `h`, care corespunde parametrului cu acelaşi nume din programul de căutare în arbori-B, are rolul de a indica faptul că arborele având rădăcina `p` s-a **modificat**.

- Reprezentarea "paginilor" ca şi "liste înlănţuite" are drept consecinţă traversarea unei "pagini" în unul sau două apeluri ale procedurii de căutare.

- Din acest motiv trebuie să se facă distincţie între:
  - (1) Un subarbore (indicat printr-un **pointer vertical**) care a crescut.
  - (2) Un nod frate (indicat printr-un **pointer orizontal**) care a mai obţinut un frate şi care necesită astfel scindarea paginii.
- Problema se rezolvă introducând trei valori pentru variabila `h`:
  - (1) `h=0` - subarborele `p` **nu** determină modificarea structurii arborelui.
  - (2) `h=1` - nodul `p` a obţinut un frate.
  - (3) `h=2` - subarborele `p` a crescut în înălţime.
- Se observă că rotirea pointerilor este foarte asemănătoare cu cea implementată de algoritmul referitor la **arborii echilibraţi** (&8.5.3).

- Se poate demonstra faptul că **arborii AVL** echilibraţi sunt de fapt **un subset** al **arborilor BBS**.
  - Rezultă, în consecinţă că drumul mediu al arborilor BBS este mai lung decât cel corespunzător clasei AVL.
  - Pe de altă parte însă, rearanjarea nodurilor este necesară mai puţin frecvent, pentru clasa BBS.
- În **concluzie**:
  - (1) **Arborii echilibraţi AVL** sunt de preferat în acele aplicaţii în care operaţia de căutare a cheilor este mai frecventă decât inserţia sau suprimarea.
  - (2) Dacă raportul acestor operaţii este moderat, se preferă **arborii BBS**.
  - (3) Desigur, limite în acest sens sunt greu de precizat, deoarece pe lângă raportul dintre frecvenţa căutărilor şi frecvenţa modificărilor structurale, intervin cu o pondere foarte semnificativă detaliile de implementare.

8.9.4. Arbori 2-3

8.9.4.1. Definire

- Arborii-B binari se încadrează în categoria arborilor-B, în particular fiind definiţi drept "**arbori B de ordinul 1**".
- Conform **definiţiei**:
  - Toate **nodurile terminale** ale unui astfel de arbore apar la acelaşi nivel.
  - **Nodurile interioare** conţin unul sau două elemente şi în consecinţă au doi respectiv trei descendenţi.
- Din acest motiv **arborii-B binari** se mai numesc şi **arbori 2-3**.

- În paragraful anterior (&8.9.3) au fost prezentate unele modalităţi de implementare a arborilor 2-3 bazate pe structuri de **arbori binari modificaţi**.
- În cadrul paragrafului de faţă va fi prezentată o altă **modalitate de reprezentare** [AH85].
    - Ca şi în alte situaţii se presupune că **arborele** va reprezenta o **mulţime de elemente** peste care este definită o **relaţie de ordonare** notată cu "<".
    - Elementele mulţimii sunt plasate în **nodurile terminale** care sunt situate toate **pe acelaşi nivel** al arborelui.
    - Dacă un element `x` se găseşte la stânga unui element `y` atunci este valabilă relaţia `x<y`.
    - În fiecare **nod interior** al structurii arbore 2-3:
        - (1) Pe **prima poziţie** va fi memorată **cheia celui mai mic descendent terminal** al celui **de-al doilea fiu** al nodului.
        - (2) Dacă nodul conţine şi un **al treilea fiu**, atunci pe **poziţia următoare** în nod se înregistrează **cheia celui mai mic descendent terminal** al celui **de-al treilea fiu**.
            - Se precizează că prin **cel mai mic descendent terminal** se înţelege elementul **terminal** cu **cheia cea mai mică** în contextul considerat.
    - Astfel în figura 8.9.4.1.a. luând în considerare nodul rădăcină (`7,16`) se observă că:
        - (1) Prima cheie memorată în acest nod este cel mai mic descendent terminal al celui de-al doilea fiu al său care este nodul (`9,14`) adică cheia `7.`
        - (2) Pe a doua poziţie în cadrul nodului rădăcină apare valoarea celui mai mic descendent terminal al celui de-al treilea fiu al său care este nodul (`22,-`), adică cheia `16`.
    - Se observă că, principial se obţine o structură asemănătoare cu structura de arbore–B reprezentată cu ajutorul paginilor (& 8.9.2).

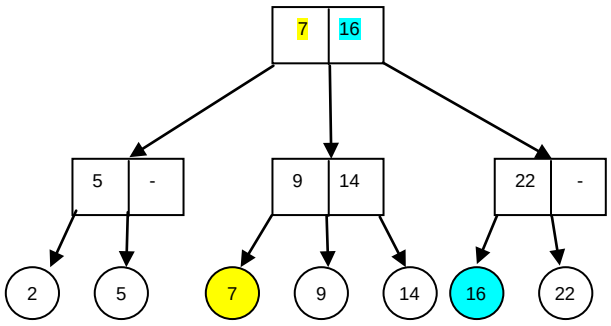

Fig.8.9.4.1.a. Arbore 2-3

- Un arbore 2-3 cu `h` niveluri (de înălţime `h`), conţine între 2^{h-1} şi 3^{h-1} noduri.
    - Cu alte cuvinte, arborele 2-3 care reprezintă o mulţime de `n` elemente are înălţimea cuprinsă între $1+\log_3 n$ şi $1+\log_2 n$.

- Astfel, **lungimea maximă** a drumului de căutare pentru un nod terminal al arborelui este O*(log₂ n)*.

- **Căutarea** unei chei într-un **arbore 2-3** se realizează cu un efort O*(log₂ n)* prin particularizarea căutării într-un **arbore-B**.
  - Astfel căutarea unei chei `x` se realizează simplu parcurgând arborele de la rădăcină spre nodurile terminale.
  - În **fiecare** nod parcurs care conţine cheile (`a,-`) sau (`a,b`) se compară `x` cu `a`.
    - (1) Dacă `x<a` se trece la **primul fiu** al nodului.
    - (2) Dacă `x≥a` şi nodul are numai 2 fii se trece la **fiul al doilea**.
    - (3) Dacă nodul în cauză are 3 fii se compară `x` cu `b` şi dacă `x<b` se trece la **fiul al doilea** al nodului, altfel se trece la **fiul al treilea**.
  - Elementul `x` există în arbore dacă şi numai dacă există un nod terminal cu cheia `x`.
  - Căutarea se poate opri la depistarea unor egalităţi de genul `x=a` sau `x=b` în cadrul **nodurilor interioare** dacă se urmăreşte numai **apartenenţa**, dar trebuie să continue până la depistarea nodului terminal dacă se doreşte prelucrarea acestuia.

- Implementarea acestei reprezentări se poate realiza în limbajul PASCAL utilizând structura de tip articol cu variante (secvenţa [8.9.4.1.a]).

```
-------------------------------------------------------------
{Implementarea arborilor-B binari (2-3)}

type TipElement = record
                    cheie: REAL;
                    {alte câmpuri utile}
                  end;

     FelNod = (terminal,interior);

     RefNodArbore2-3 = ^TipNodArbore2-3;      [8.9.4.1.a]

     TipNodArbore2-3 = record
       case fel: FelNod of
         terminal: (element: TipElement);
         interior: (cheieStg,cheieDr: REAL;
                    fiuUnu,fiuDoi,fiuTrei:
                          RefNodArbore2-3)
       end {case}
     end; {record}
-------------------------------------------------------------
```

8.9.4.2. Inserţia nodurilor în arbori 2-3

- Inserţia unui nod nou într-un arbore 2-3 se realizează principial ca şi inserţia într-un arbore–B.
- Inserţia unui nod cu cheia `x` începe cu căutarea cheii `x` în arbore.

- Dacă se ajunge la un nod aparţinând nivelului situat chiar deasupra nivelului terminal şi se constată ca fii acestuia **nu** îl conţin pe `x`, se procedează la inserţie.
- Sunt posibile următoarele situaţii:
- (1) Dacă nodul respectiv are numai **doi fii**, se face `x` cel de-a treilea fiu al nodului, plasându-l la locul potrivit şi reajustând structura nodului astfel încât acesta să reflecte noua situaţie.
    - În figura 8.9.4.2.a este prezentată inserţia nodului cu cheia `21` în structura de arbore din figura 8.9.4.1.a, inserţie realizată în maniera mai sus precizată.

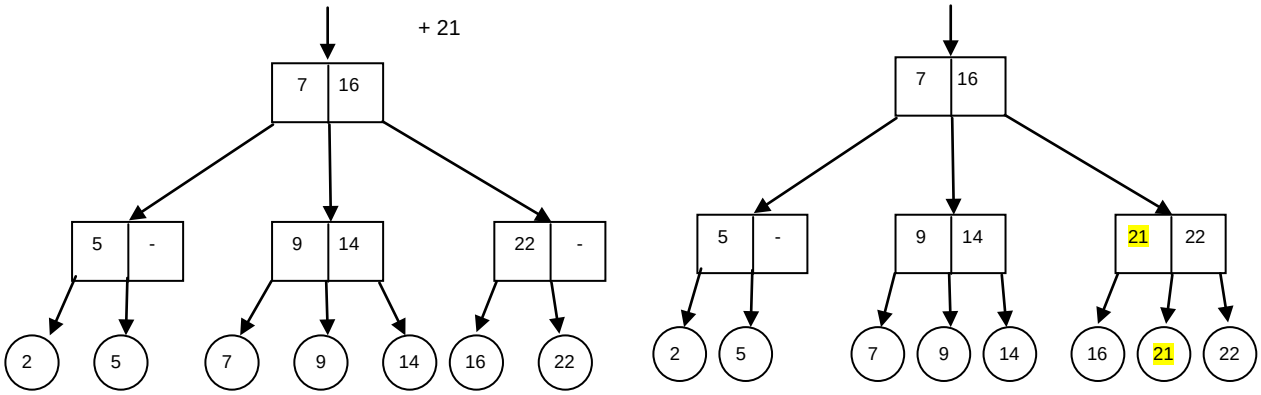

Fig.8.9.4.2.a. Inserţia unui nod nou într-un arbore 2-3

- (2) Dacă nodul considerat are **trei fii** atunci adăugarea celui de-al patrulea fiu fiind imposibilă, nodul se **scindează** în două noduri, iar cei patru fii sunt redistribuiţi:
    - Cei doi fii cu chei mai mici, fostului nod.
    - Ceilalţi doi fii cu chei mai mari, nodului nou apărut prin scindare.
- În continuare nodul nou apărut trebuie inserat în structura arbore.
- În urma acestui proces **scindarea** se poate propaga spre nivelurile superioare ale structurii, în cazul extrem până la rădăcină.
    - Aceasta este de fapt singura posibilitate ca un arbore 2-3 să crească în înălţime.
- În figura 8.9.4.2.b se prezintă inserţia nodului cu cheia `11` în arborele 2-3 din fig. 8.9.4.2.a.
    - După cum se observă, inserţia se realizează în două etape.
    - (1) În prima etapă are loc scindarea nodului (`9,14`) în două noduri (`9,-`) respectiv (`14,-`) cu repartizarea corespunzătoare a nodurilor terminale (fig. 8.9.4.2.b (a)).
    - (2) În etapa următoare se inserează nodul nou creat în structura de arbore, lucru care necesită o nouă scindare, de data aceasta a nodului rădăcină şi crearea unei noi rădăcini (fig. 8.9.4.2.b (b)).

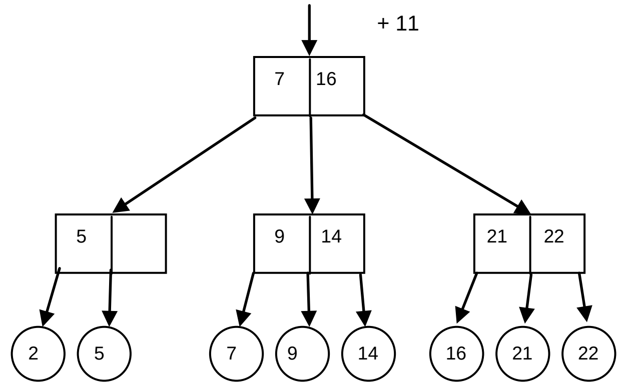

Fig.8.9.4.2.a. Inserţia unui nod nou într-un arbore 2-3 (reluare)

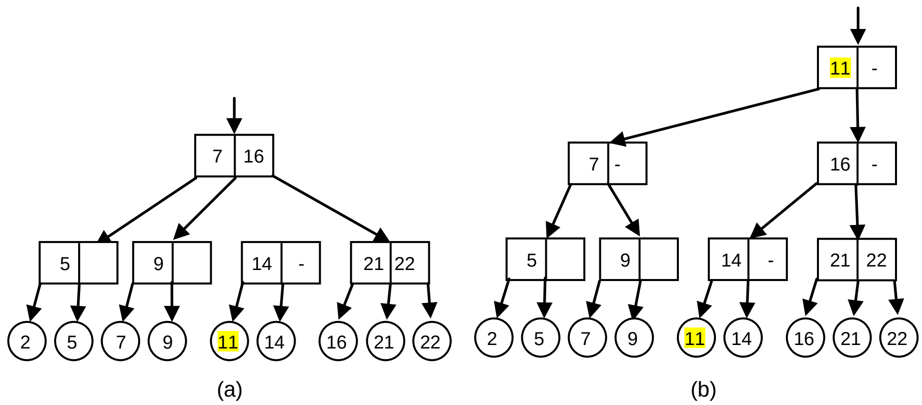

Fig.8.9.4.2.b. Inserţie într-un arbore binar 2-3

- Deşi principial inserţia se realizează simplu, implementarea sa efectivă presupune o serie de **detalii**.
- În cele ce urmează se vor prezenta două proceduri:
  - (1) Procedura **`Inserţie`** care se apelează pentru rădăcină.
  - (2) Procedura **`Inserţie1`** care se apelează pentru celelate noduri ale structurii şi care parcurge arborele în manieră recursivă.
- Se presupune că arborele 2-3 în care se realizează inserţia **nu** este nici arbore vid nici arbore cu un singur nod intern.
  - În aceste două cazuri inserţia se realizează direct de către procedura **`Insertie`**.
- Procedura **`Inserţie1`** returnează **pointerul la nodul nou creat** (dacă se crează un astfel de nod) şi **cheia celui mai mic element** al subarborelui determinat de acest nod.
  - Acest lucru este realizat cu ajutorul parametrilor `Pnou` respectiv `Mic` care se atribuie în situaţia în care se returnează un nod nou.
- Structura de principiu a procedurii **`Inserţie1`** apare în secvenţa [8.9.4.2.a] iar implementarea detaliată în secvenţa [8.9.4.2.b].
- Se precizează faptul că în secvenţa [8.9.4.2.a] din motive de reducere a complexităţii, o serie de detalii ale procesului de inserţie, care apar de altfel în secvenţa [8.9.4.2.b] sunt omise.

```
/*Inserţia în arborii 2-3*/

/*Valabilă pentru toate nodurile cu excepţia nodului
   rădacină*/
/*Schiţa de principiu a algoritmului - varianta pseudocod*/

Subprogram Insertie1(RefNodArbore2-3 Nod, TipElement X;
                var RefNodArbore2-3 Pnou:, var REAL Mic);
     /* X - element care urmează a fi inserat în
            subarborele Nod*/
     /* Pnou - pointer la nodul nou creat la dreapta lui Nod
           (dacă este cazul)*/
     /* Mic  - cheia celui mai mic element al subarborelui
            indicat de Pnou */

  RefNodArbore2-3 Pprim,W;
  REAL MicPrim;

  Pnou= NULL;
  daca (Nod este terminal)
     |daca X nu este elementul lui Nod
     | |*creează un nod nou terminal indicat de Pnou;
     | |*atribuie pe X noului nod;
     | |Mic= X.cheie;
     | |□ /*daca*/
     |□ /*daca*/
    altfel                                          [8.9.4.2.a]
     |/*Nod este un nod interior*/
     |*fie W acel fiu al lui Nod, căruia îi va aparţine X;
     |Insertie1(W,X,Pprim,MicPrim);
     |daca (Pprim<>NULL) /*a fost creat un nod nou*/
     |   |*inserează pointerul Pprim printre fiii lui
     |   |   Nod chiar în dreapta lui W;
     |   |daca Nod are 4 fii
     |   | |*creează un nod nou indicat de Pnou;
     |   | |*asociază acestui nod fiii trei şi patru
     |   | |     ai lui Nod;
     |   | |*actualizează valorile cheieStg şi
     |   | |     cheieDr în Nod şi în noul nod;
     |   | |*asignează pe Mic cu cea mai mică
     |   | |     cheie aparţinând fiilor noului nod
     |   | |□ /*daca*/
     |   |□ /*daca*/
     |□ /*altfel*/
  revenire;
/*Insertie1*/
-------------------------------------------------------------
{Inserţia în arborii 2-3}

{Valabilă pentru toate nodurile cu excepţia nodului
radacină}
{Schiţa de principiu a algoritmului - varianta PASCAL like}

procedure Insertie1(Nod: RefNodArbore2-3; X: TipElement;
                var Pnou: RefNodArbore2-3; var Mic: REAL);
    {X    - element care urmează a fi inserat în
            subarborele Nod
     Pnou - pointer la nodul nou creat la dreapta lui Nod
```

```
            (dacă este cazul)
      Mic   - cheia celui mai mic element al subarborelui
              indicat de Pnou}

  var Pprim,W: RefNodArbore2-3; MicPrim: REAL;

  begin
    Pnou= nil;
    if Nod este terminal then
      begin
        if X nu este elementul lui Nod then
          begin
            *creează un nod nou terminal indicat de Pnou;
            *atribuie pe X noului nod;
            Mic:= X.cheie
          end
      end
    else                                              [8.9.4.2.a]
      begin {Nod este un nod interior}
        *fie W acel fiu al lui Nod, căruia îi va
          aparţine X;
        Insertie1(W,X,Pprim,MicPrim);
        if Pprim <> nil then {a fost creat un nod nou}
          begin
            *inserează pointerul Pprim printre fiii lui
              Nod chiar în dreapta lui W;
            if Nod are 4 fii then
              begin
                *creează un nod nou indicat de Pnou;
                *asociază acestui nod fiii trei şi patru
                  ai lui Nod;
                *actualizează valorile cheieStg şi
                  cheieDr în Nod şi în noul nod;
                *asignează pe Mic cu cea mai mică
                  cheie aparţinând fiilor noului nod
              end
          end
      end
  end; {Insertie1}
-------------------------------------------------------------
{Inserţia normală în arbori 2-3 - varianta PASCAL}

{Valabilă pentru toate nodurile cu excepţia nodului
rădacină}

procedure Insertie1(Nod: RefNodArbore2-3;X: TipElement;
            var Pnou: RefNodArbore2-3; var Mic: REAL);

  var Pprim: RefNodArbore2-3;
      MicPrim: REAL;
      Fiu: 1..3; {indică care dintre fiii nodului este
                  selectat în apelul recursiv}
      W: RefNodArbore2-3; {pointer la fiu}

  begin
    Pnou := nil;
    if Nod^.fel=terminal then {Nod este un nod terminal}
      begin
```

```
      if Nod^.element.cheie<>X.cheie then
        begin {se creează un nod nou care-l conţine pe
               X şi se returnează pointerul acestui nod}
          NEW(Pnou);
          Pnou^.fel:= terminal;
          if Nod^.element.cheie<X.cheie then
            begin {X se plasează în noul nod care este
                   situat  la dreapta nodului curent,
                   fiind mai mare}
              Pnou^.element:= X; Mic:= X.cheie
            end
          else
            begin {X se plasează în stânga elementului
                  din nodul curent prin interschimbare}
              Pnou^.element:= Nod^.element;
              Nod^.element:= X;
              Mic:= Pnou^.element.cheie
            end
        end                                        [8.9.4.2.b]
    end
  else
    begin {Nod este un nod interior}
      {se selectează fiul corespunzător W pentru
       parcurgere}
      if X.cheie<Nod^.cheieStg then
        begin
          Fiu:= 1; W:=Nod^.fiuUnu
        end
      else
        if(Nod^.fiuTrei=nil)or
              (X.cheie<Nod^.cheieDr) then
          begin {X aparţine celui de-al doilea
                 subarbore}
            Fiu:= 2; W:= Nod^.fiuDoi
          end
        else
          begin {X aparţine celui de-al treilea
                 subarbore}
            Fiu:= 3; W:= Nod^.fiuTrei
          end;
      Insertie1(W,X,Pprim,MicPrim);
      if Pprim<>nil then
        {trebuie inserat noul fiu al lui Nod}
        if Nod^.fiuTrei=nil then
          {Nod are numai doi fii; noul nod se inserează
           la locul potrivit}
          if Fiu=2 then {Fiu=2}
            begin
              Nod^.fiuTrei:= Pprim;
              Nod^.cheieDr:= MicPrim
            end
          else
            begin {Fiu=1}
              Nod^.fiuTrei:= Nod^.fiuDoi;
              Nod^.cheieDr:= Nod^.cheieStg;
              Nod^.fiuDoi:= Pprim;
              Nod^.cheieStg:= MicPrim
            end
```

```
          else                                        [8.9.4.2.b]
            begin {Nod are trei fii}
              NEW(Pnou); {se creaza un nod nou}
              Pnou^.fel := interior;
              if Fiu=3 then
                begin {Pprim şi fiul trei devin fiii
                       noului nod}
                  Pnou^.fiuUnu:= Nod^.fiuTrei;
                  Pnou^.fiuDoi:= Pprim;
                  Pnou^.fiuTrei:= nil;
                  Pnou^.cheieStg:= MicPrim; {cheieDr
                          este nedefinită pentru Pnou}
                  Mic:= Nod^.cheieDr;
                  Nod^.fiuTrei:= nil
                end
              else
                begin {Fiu<=2; Se mută fiul trei al lui
                       Nod în Pnou}
                  Pnou^.fiu_2:= Nod^.fiuTrei;
                  Pnou^.cheieStg:= Nod^.cheieDr;
                  Pnou^.fiuTrei:= nil;
                  Nod^.fiuTrei:= nil
                end;
              if Fiu=2 then
                begin {Pprim devine primul fiu al lui
                       Pnou}
                  Pnou^.fiuUnu:= Pprim;        [8.9.4.2.b]
                  Mic:= MicPrim
                end;
              if Fiu=1 then
                begin {fiul doi al lui Nod se mută în
                       Pnou; Pprim devine fiul doi al
                       lui Nod}
                  Pnou^.fiuUnu:= Nod^.fiuDoi;
                  Mic:= Nod^.cheieStg;
                  Nod^.fiuDoi:= Pprim;
                  Nod^.cheieStg:= MicPrim
                end
            end
        end
    end; {Insertie1}
-------------------------------------------------------------
```

- În continuare se trece la redactarea procedurii **Inserţie** care apelează procedura **Inserţie1**.
- Dacă **Inserţie1** returnează un nod nou, atunci **Inserţie** trebuie să creeze o nouă rădăcină.
  - Codul procedurii apare în secvenţa [8.9.4.2.c].

```
-------------------------------------------------------------
{Inserţia rădăcinii arborilor 2-3 - varianta PASCAL}

procedure Insertie(X: TipElement; var M: RefNodArbore2-3);
  var Pprim: RefNodArbore2-3; {pointer la nodul nou
                               returnat de Insertie1}
      MicPrim: REAL; {valoarea Mic a subarborelui
                      determinat de Pprim}
```

```
    Temp: RefNodArbore2-3; {utilizat pentru memorarea
                            temporară a valorii pointerului M}
begin
  if M este arbore vid sau arbore cu un singur nod then
     *se realizează direct inserţia;
    else
      begin
        Insertie1(M,X,Pprim,MicPrim);
        if Pprim <> nil then
           begin {se creează o nouă rădăcină; fiii
                  rădăcinii sunt cei indicaţi de M
                  respectiv Pprim}
             Temp:= M;                          [8.9.4.2.c]
             NEW(M);
             M^.fel:= interior;
             M^.fiuUnu:= Temp;
             M^.fiuDoi:= Pprim;
             M^.cheieStg:= MicPrim;
             M^.fiuTrei:= nil
           end
      end
end; {Insertie}
```

8.9.4.3. Suprimarea nodurilor în arbori 2-3

- Suprimarea nodurilor în arbori 2-3 se realizează într-o manieră similară suprimării în arbori-B.
- Se presupune că se dă un arbore 2-3 şi o cheie x şi se cere să se suprime nodul având cheia egală cu x.
- În prealabil se caută nodul cu cheia precizată după care se trece la suprimarea propriu-zisă.
- Astfel, fie n **părintele nodului de suprimat** şi fie p **părintele** lui n (bunicul nodului de suprimat). Sunt posibile mai multe situaţii:
  - (1) Dacă n (părintele nodului de suprimat) are **trei fii** suprimarea se realizează simplu prin reorganizarea nodului indicat de n.
  - (2) Dacă n (părintele nodului de suprimat) are **doi fii** în urma suprimării n rămâne cu un singur fiu, adică apare fenomenul cunoscut sub numele de ”**subdepăşire**”, arborele 2-3 trebuie reorganizat situaţie în care există trei cazuri:
    - (a) Dacă n este chiar nodul **rădăcină**, după suprimare, singurul său fiu va deveni noua rădăcină.
    - (b) Dacă p (bunicul nodului de suprimat) mai are un fiu situat la stânga sau la dreapta lui n şi acest fiu are la rândul său 3 fii, în cadrul unui proces denumit “**echilibrare**” se va realiza “împrumutul” unui fiu astfel încât n să aibă doi fii.
    - (c) Dacă însă fiul sau fii lui p, adiacenţi lui n au la rândul lor numai câte doi fii, echilibrarea **nu** se mai poate realiza şi în consecinţă se transferă singurul fiu al lui n unui frate adiacent şi se suprimă n.

- Dacă în urma acestui proces denumit "**cotopire**" `p` rămâne cu un singur fiu, se repetă procedura în mod recursiv substituind pe `n` cu `p`, trecând astfel pe nivelul superior al structurii arborelui.
- În caz extrem acest proces se poate propaga până la rădăcină, aceasta fiind practic singura cale de reducere a înălţimii arborelui.

- Spre exemplu se presupune că se doreşte suprimarea nodului cu cheia `11` din arborele din figura 8.9.4.3.a.(0).
  - Dacă nodul terminal `11` este suprimat, părintele său (`14,-`) rămâne cu un singur fiu.
  - În această situaţie se poate realiza **împrumutul** de la fratele din dreapta al nodului (`14,-`) care are trei fii.
  - Astfel este împrumutat nodul `16` şi în urma reactualizării cheilor rezultă arborele din figura 8.9.4.3.a.(a).

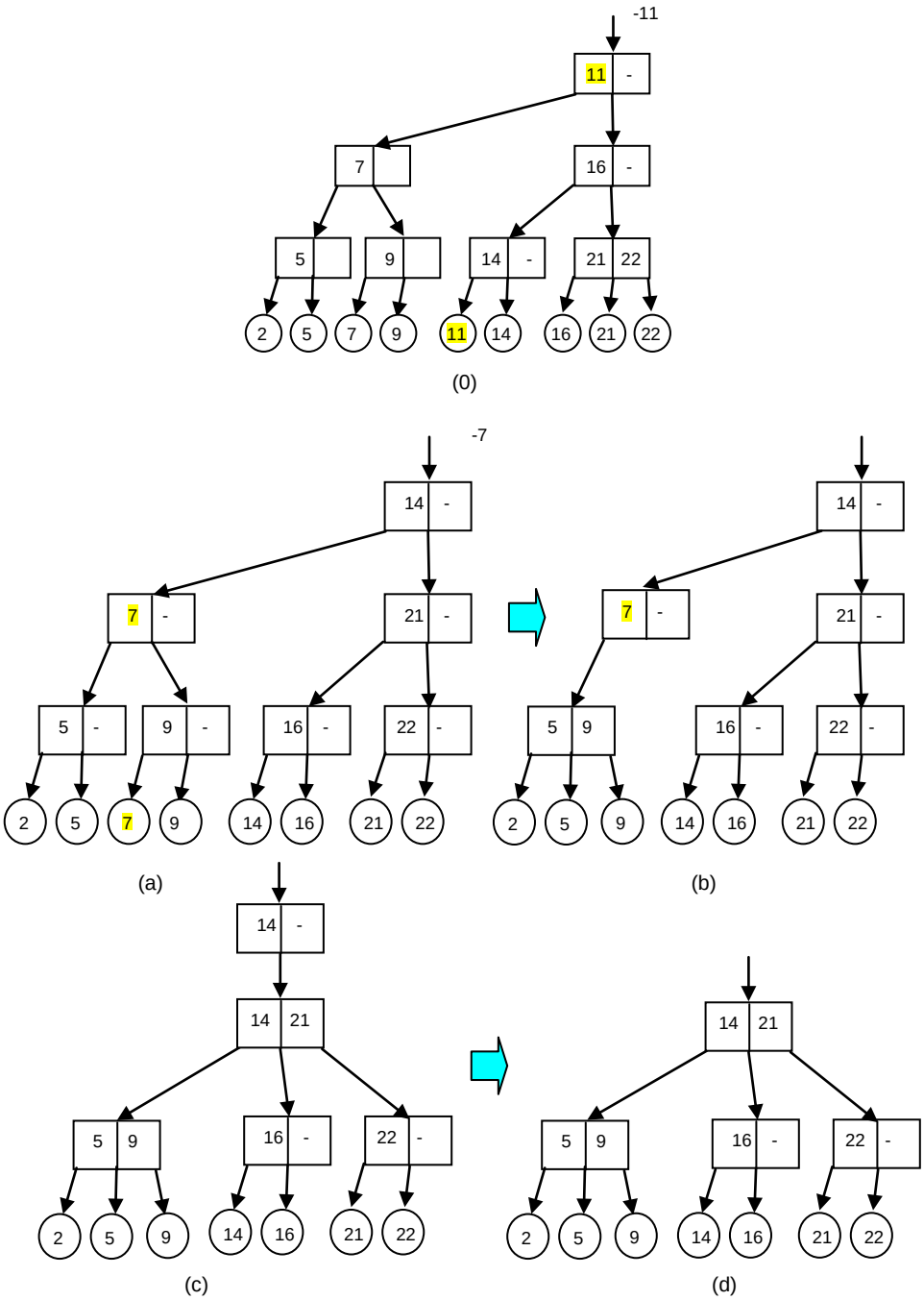

Fig.8.9.4.3.a. Suprimarea nodurilor într-un arbore 2-3

- În continuare se procedează la suprimarea nodului cu cheia 7 din arborele din figura 8.9.4.3.(a).
  - În consecinţă părintele său (9,-) rămâne cu un singur fiu.
  - Întrucât nu se poate realiza împrumutul deoarece unicul frate al nodului în cauză, adică cel situat în stânga sa, are numai doi fii, se procedează la **contopire** prin transferarea cheii 9 şi suprimarea nodului (9,-).
  - În urma acestui proces rezultă arborele (b) din aceeaşi figură.
  - După cum se observă nodul interior (7,-) are un singur fiu iar fratele său, nodul (21,-) are numai doi fii, în consecinţă se aplică din nou **contopirea** cu reactualizarea cheilor, obţinându-se un nod cu trei fii (fig.8.9.4.3.(c)).
  - Acum rădăcina are un singur fiu, deci ea poate fi suprimată şi înlocuită cu acest fiu (fig.8.9.4.3(d)).
- Din cele prezentate rezultă faptul că prelucrarea arborilor 2-3 necesită manipularea frecventă a valorilor memorate în nodurile interioare.
- Valorile necesare în procesul de prelucrare pot fi determinate în două moduri:
  - (1) Parcurgând arborele pentru fiecare valoare.
  - (2) Reţinând în timpul prelucrării arborelui **cea mai mică valoare** a **descendenţilor fiecărui nod** situat pe drumul de la rădăcină spre nodul de suprimat.
    - Această informaţie poate fi determinată simplu pentru fiecare nod parcurs în cadrul unui **algoritm recursiv** de **suprimare**.
- În cele ce urmează se va prezenta structura de principiu a unei funcţii **Suprima1** (secvenţa (8.9.4.3.a)) care primeşte ca parametri un pointer la un nod de arbore 2-3 (Nod) şi o cheie X.
  - Sarcina funcţiei **Suprima1** este aceea de a suprima nodul terminal cu cheia X dintre descendenţii nodului precizat, dacă există un astfel de nod.
- Pentru a se atinge acest obiectiv, în cadrul funcţiei se realizează o **parcurgere recursivă** a structurii arborelui în vederea determinării nodului terminal X, după care se revine pe drumul parcurs restructurând după necesităţi arborele.
- După fiecare apel, funcţia **Suprima1** returnează **valoarea adevărat** dacă după restructurare Nod rămâne cu **un singur fiu** respectiv returnează **valoarea fals** dacă Nod rămâne cu doi sau trei fii.

```
------------------------------------------------------------
{Suprimarea în arbori 2-3}

{Schiţa de principiu a algoritmului – varianta PASCAL like}

function Suprima1(Nod: RefNodArbore2-3; X: TipElement):
                    BOOLEAN;
  var NumaiUnul: BOOLEAN;{memorează valoarea returnată
                          de un apel al funcţiei Suprima1}
  begin
```

```
Suprima1:= FALSE;
if fiii lui nod sunt terminali then          [8.9.4.3.a]
  begin
    if X este printre aceşti terminali then
      begin
        *extrage pe X;
        *deplasează fiii lui Nod situaţi în dreapta
          nodului X cu o poziţie spre stânga;
        if Nod are acum un singur fiu then
          Suprima1:= TRUE
      end
  end
else
  begin {nodul nu are fii terminali el situându-se pe
         un nivel interior}
    *determină acel fiu al lui Nod, care l-ar putea
      avea pe X ca descendent: fie acesta W;
    NumaiUnul:= Suprima1(W,X); {W are valoarea
      Nod^.fiuUnu, Nod^.fiuDoi sau Nod^.fiuTrei
      după cum este cazul}
    if NumaiUnul then                        [8.9.4.3.a]
      begin {restructurare}
        if W este primul fiu al lui Nod then
          if Y (cel de-al doilea fiu al lui Nod) are
              trei copii then
            *împrumută primul fiu al lui Y şi
             transferă-l drept al doilea fiu al lui W
          else
            begin {Y are doi fii, deci contopire}
              *transferă fiul lui W, drept prim fiu
                al lui Y;
              *extrage pe W dintre fiii lui Nod;
              if Nod are acum un singur fiu then
                Suprima1:= TRUE
            end;
        if W este cel de-al doilea fiu
             al lui Nod then
            if Y,primul fiu al lui Nod,are
                  trei fii then
              *împrumută cel de-al treilea fiu al lui
                Y şi transferă-l drept primul fiu al
                lui W
            else {Y are doi fii}
              if Z,cel de-al treilea fiu al lui
                  Nod,există şi are trei copii then
                *împrumută primul fiu al lui Z şi
                  transferă-l drept al doilea fiu al
                  lui W
              else
                begin{nici un alt fiu al lui Nod,nu
                      are trei fii}
                  *transferă fiul lui W drept al
                    treilea fiu al lui Y;
                  *extrage pe W dintre fiii lui Nod;
                  if Nod are acum un singur fiu then
                    Suprima1:= TRUE
                end;
        if W este cel de-al treilea fiu al
```

```
                    lui Nod then
                  if Y, cel de-al doilea fiu al lui Nod, are
                     trei fii then
                    *împrumută cel de-al treilea fiu al lui
                      Y şi transferă-l drept primul fiu al
                      lui W
                  else                                [8.9.4.3.a]
                    begin {Y are doi fii}
                      *transferă fiul lui W drept cel de-al
                        treilea fiu al lui Y;
                      *extrage pe W dintre fii lui Nod
                    end {în acest caz lui Nod îi rămân cu
                         siguranţă doi fii}
            end
        end
    end;{Suprima1}
----------------------------------------------------------------
```

- Codul detaliat al funcţiei **Suprima1** este propus drept exerciţiu.
- Un alt exerciţiu propus este acela de a redacta procedura **Suprima**(m,x) care:
  - Verifică şi rezolvă direct cazurile speciale în care arborele m este vid sau conţine un singur element.
  - Dacă arborele conţine mai multe noduri, apelează funcţia **Suprima1**.
  - Dacă **Suprima1** returnează adevărat, procedura **Suprima**(m,x)suprimă rădăcina (nodul idicat de m) şi face pe m să indice pe fiul unic al rădăcinii.