8.6. Arbori multicăi

8.6.1. Generalități

- Până în prezent au fost studiate cu predilecție structuri de arbori în care fiecare nod avea **cel mult** doi descendenți.
    - Desigur, acest lucru este pe deplin justificat dacă spre exemplu, se dorește să se reprezinte **descendența unei persoane** din punctul de vedere al strămoşilor,
    - În acest caz, fiecărei persoane i se asociază cei doi părinți ai săi.
- Dacă 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 conțin astfel de noduri se numesc **arbori multicăi**.
    - Astfel de structuri ridică însă unele **probleme** în prelucrare.
    - Este evident faptul că modalitățile de reprezentare a arborilor sugerate până în prezent **nu** corespund într-o manieră eficientă arborilor multicăi.
- În practică însă, există un mare interes referitor la arborii multicăi.
    - Este vorba despre construcția şi exploatarea **arborilor de foarte mari dimensiuni.**
        - Arbori în care se fac frecvent **inserții** şi **suprimări**
        - Arbori pentru care dimensiunile memoriei centrale sunt **insuficiente** sau a căror memorare vreme îndelungată în memoria sistemului de calcul este prea **costisitoare**.
- Această aplicabilitate se bazează pe o **tehnică specială de implementare** a acestei categorii de arbori.
    - 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 soluție perfectă a acestei probleme.

- 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 memorați pe disc ca unități în zone 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 situația în care arborele creşte aleator, în cel mai **defavorabil caz** (când degenerează în lista liniară) numărul de accese poate ajunge însă la 10^4.

  - Este evident faptul că 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 modalități de implementare a arborilor multicăi o reprezintă **arborii B**.

8.6.2. Arbori-B

8.6.2.1. Definire

- Din discuția asupra **mecanismului** de control al creşterii arborilor multicăi, **arborii perfect echilibrați** 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 cu excepția uneia, conține 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 `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 puțin 50%, deoarece orice pagină este cel puțin pe jumătare plină.

- În plus schema preconizată necesită **algoritmi simpli** pentru căutare, inserție şi suprimare în comparație 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 **proprietăți**:

  1. Fiecare pagină a arborelui-B conține **cel mult** `2n` noduri (chei);

  2. Fiecare pagină, cu excepția paginii rădăcină conține **cel puțin** `n` noduri;

  3. Fiecare pagină este fie:

     - O **pagină terminală** - caz în care **nu** are descendenți

     - O **pagina interioară** - caz în care are `m+1` descendenți unde `m` este numărul de chei din pagină `(n≤m≤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 conțin 2, 3 sau 4 noduri cu excepția paginii rădăcină care conține unul singur.

  - Toate paginile terminale apar pe nivelul 3.

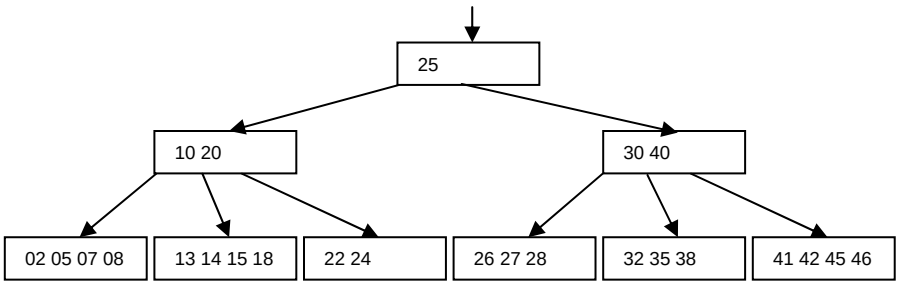

Fig.8.9.2.1.a. Arbore-B de ordinul 2

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

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

8.6.2.2. 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 unui sistem de calcul, pentru **căutarea cheii** x printre cheile $k_1, \ldots, k_m$ aparținând paginii, se poate utiliza o **metodă de căutare convențională**.

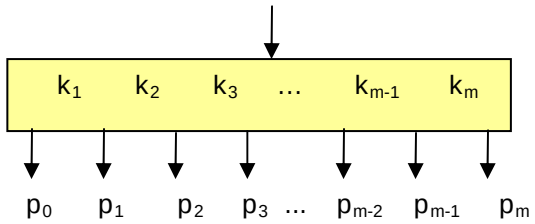

Fig. 8.9.2.2.a. Pagină cu m chei a unui arbore-B

- Spre exemplu, dacă m este mare se poate utiliza **căutarea binară**, în caz **contrar, căutarea liniară**.

    - Trebuie subliniat faptul că **timpul de căutare** în memoria centrală este probabil **neglijabil** în comparație 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 situații:

    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.

8.6.2.3. Inserția nodurilor în arbori B

- În ceea ce priveşte **inserția nodurilor** în arborii-B de ordinul n, există mai multe cazuri.

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

- În acest caz inserția se realizează simplu în pagina respectivă inserând cheia corespunzătoare la **locul potivit** în secvența ordonată a cheilor,

- Spre exemplu inserția 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 conține deja `2n` chei,

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

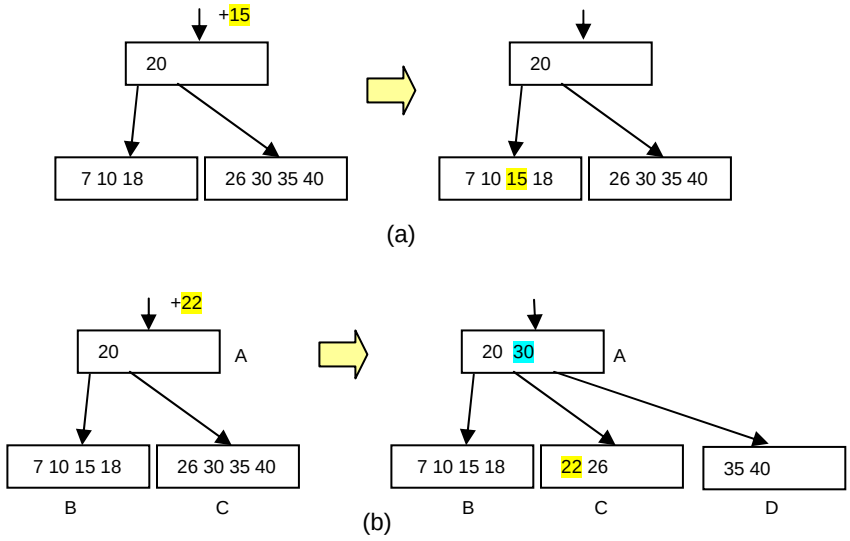

Fig.8.9.2.3.a Inserția nodurilor în arbori-B. Exemple

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

  1. Se caută cheia 22 şi se descoperă că ea lipseşte, iar inserția în pagina C este imposibilă deoarece aceasta este plină (conține `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.

- Această schemă păstrează toate proprietățile 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 inferioare ale structurii, în cazul extrem până la rădăcină.

  - Aceasta este de fapt **singura** posibilitate ca un arbore-B **să crească în înălțime**.

- 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 inserție în **arbori echilibrați** (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}

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 elemente în pagină}
               p0:RefPagina;
               elemente:ARRAY[1..nn] OF nod
             END;
-------------------------------------------------------------
```

- Referitor la structura `nod`:

  - Câmpul `cheie` precizează cheia nodului respectiv

  - Câmpul `p` indică pagina urmaş care conține chei mai mari decât cheia nodului în cauză

  - Câmpul `contor` este utilizat ca **numărător de accese**.

- Referitor la structura `pagina`:

  - Fiecare pagină oferă spațiu pentru `2n` noduri.

  - Variabila `m` indică **numărul curent** de noduri memorate în pagina respectivă

- Tabloul `elemente` 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 excepția rădăcinii), se garantează o utilizare a memoriei de cel puțin 50 %.

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

  - Structura sa de principiu este asemănătoare cu cea a algoritmului de căutare binară, cu **excepția** faptului că decizia de ramificație în arbore **nu** e binară ci este cea specifică **arborilor-B** (&8.9.2.2).

  - În schimb căutarea în interiorul unei pagini este o căutare binară efectuată în tabloul `elemente` al paginii curente.

- Forma pseudocod a procedurii `Cauta` apare în secvența [8.9.2.3.b].

```
-------------------------------------------------------------
(Schița de principiu a procedurii de căutare în arbori-B}

PROCEDURE Cauta(x:integer; a:RefPagina; VAR h:boolean;
                VAR v:nod);
  VAR u:nod;
  BEGIN
    IF a=NIL THEN
        BEGIN {x nu este in arbore}
          {*se crează nodul v, i se atribuie cheia x şi se
           pune h pe adevarat indicând pasarea nodului v spre
           rădăcină}
        END
      ELSE
        BEGIN {se caută x în pagina curentă a^}
          {*căutare binară într-un tablou liniar }
          IF găsit THEN
              {*incrementează contorul de accese}
            ELSE                                        [8.9.2.3.b]
              BEGIN
                Cauta(x,urmas,h,u);
                IF h THEN Insereaza  {nodul u a fost pasat}
              END
        END
  END; {Cauta}
-------------------------------------------------------------
```

- Algoritmul de inserție este formulat ca şi o procedură aparte (procedura `Insereaza`) care este activată după ce procesul de căutare indică faptul că unul din noduri trebuie **pasat** spre 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 în direcția rădăcinii.

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

    - Noul nod, este transmis imediat, via parametrul `u` paginii terminale pentru adevărată inserție.

- Pornind de la aceste precizări, structura de principiu a procedurii `Insereză` apare în secvența [8.9.2.3.c].

```
-----------------------------------------------------------
{Schița de principiu a inserției nodurilor în arbori-B}

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 crează o nouă pagină b^}
          {* se redistribuie cheile paginii a^, primele n
           pe a^, ultimele n pe b^ si se pasează nodul v=u
           conținând cheia mediană spre nivelul inferior
           iar h rămâne poziționat 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`.

    - În figura 8.9.2.3.b apare urma execuției 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. Inserția ultimei chei (25) cauzează două scindări şi alocarea a 3 noi pagini.

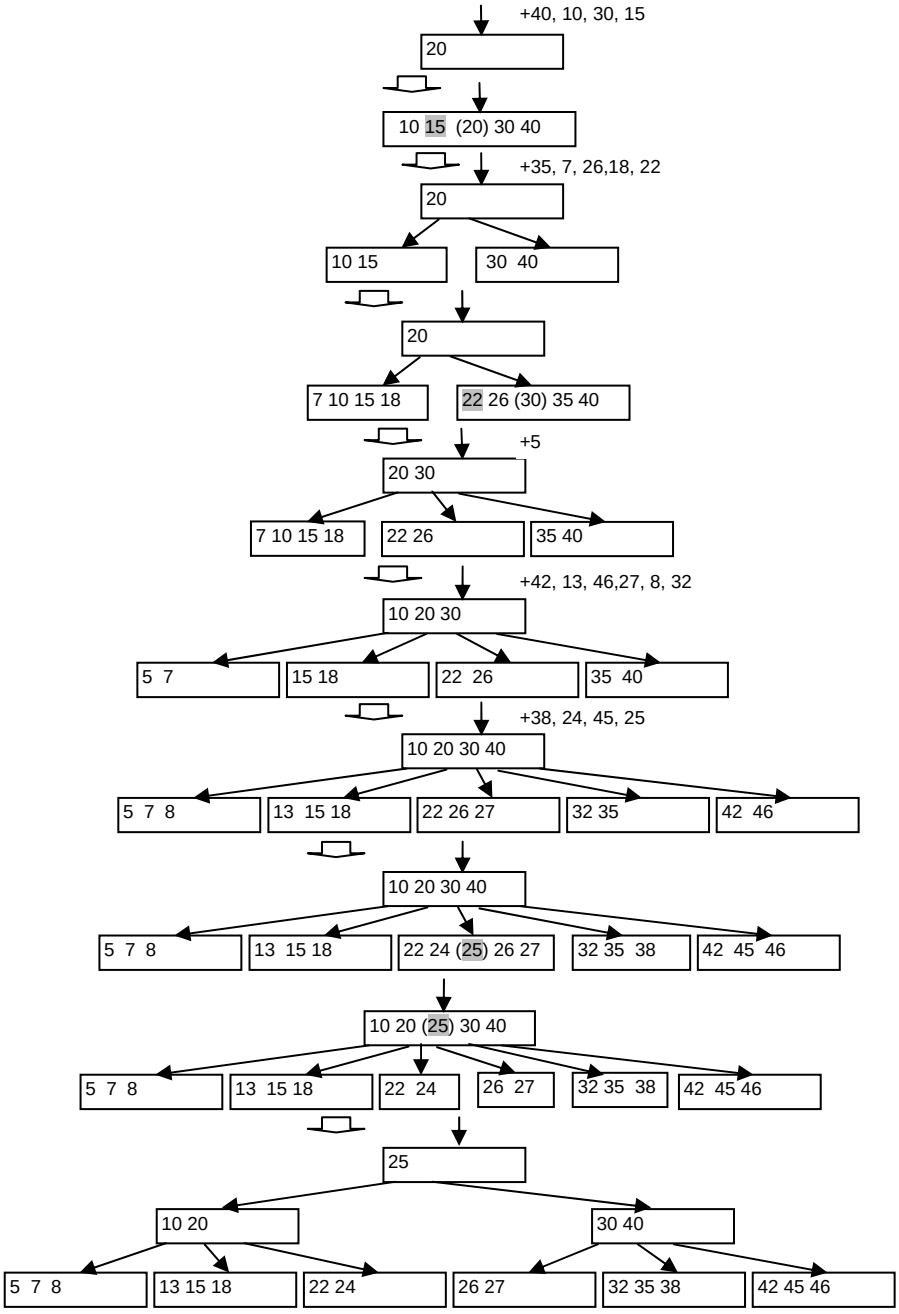

Fig.8.9.3.2.b. Construcția unui arbore-B de ordinul 2

- În legătură cu inserția nodurilor în arbori-B se fac următoarele observații:

  - (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 conține `N` noduri, în memoria principală trebuie să încapă **cel puțin** `k` pagini.

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

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

  - O consecință 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 activităților.

- Un alt **avantaj** al structurii de date de tip arbore-B se referă la actualizarea simplă şi eficientă, în mod secvențial 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 înălțime, inserția unei noi pagini respectiv adăugarea unui nou nivel se realizează după inserția unui număr semnificativ de chei.

8.6.2.4. Suprimarea nodurilor în arbori-B

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

- Se disting două situații:

  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 adevărată a lui `h`.

    - În acest caz soluția de rezolvare este aceea de a "**împrumuta**" un nod de la pagina vecină.

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

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

    - În situația 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 , funcție de situație, ştergerea celeilalte pagini

        - Acesta este procesul invers scindării paginii, proces care poate fi urmărit în figura 8.9.2.3.a, dacă se imaginează suprimarea cheii cu numărul 22.

        - Din nou, extragerea cheii din mijloc din pagina strămoş, poate determina subdepăşirea, situație 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 înălțimii arborelui-B.

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

    - În figura 8.9.2.4.a se prezintă evoluția unui arbore-B, rezultată din suprimarea următoarei secvențe 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.

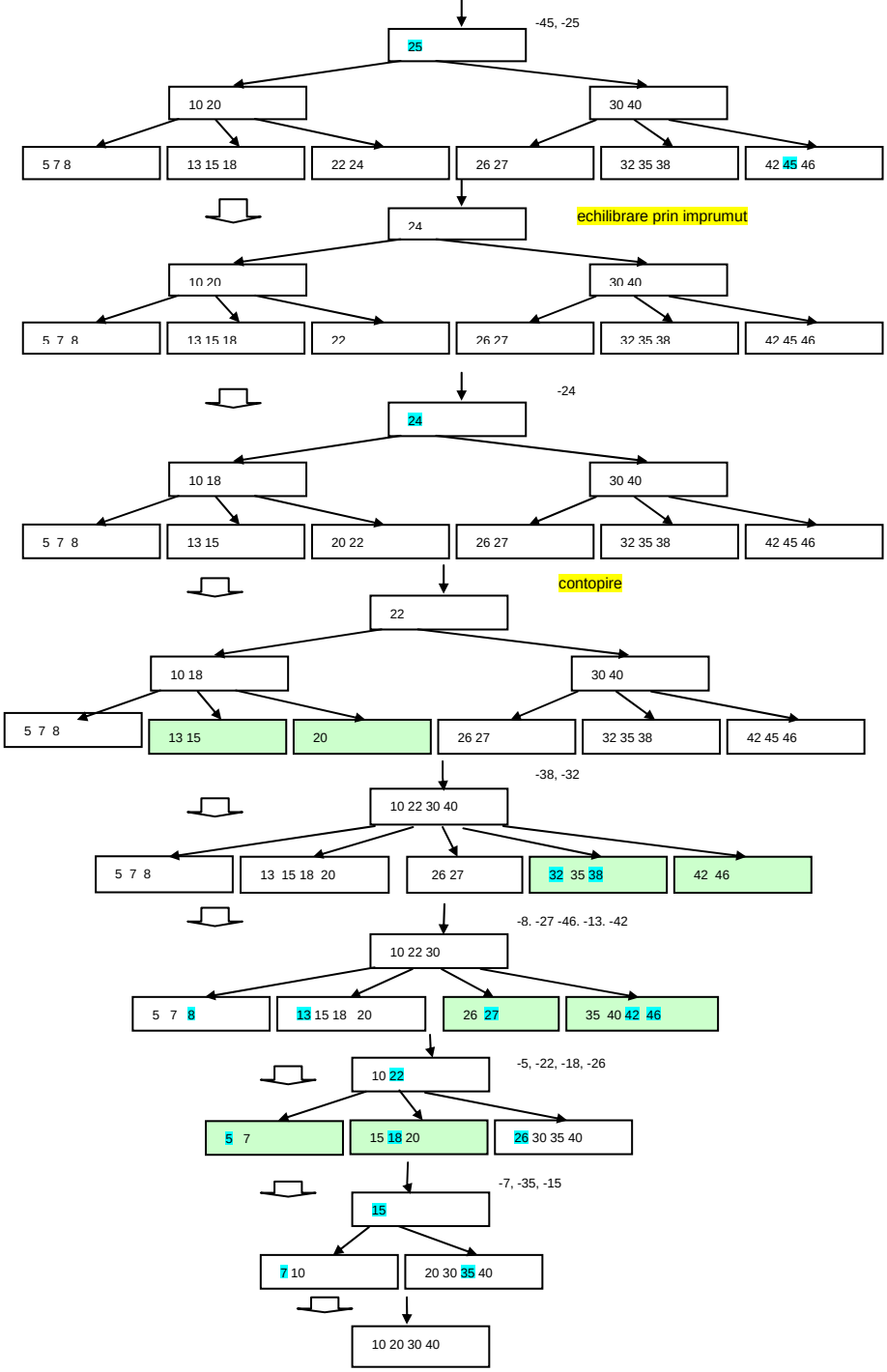

Fig.8.9.4.2.a. Suprimarea nodurilor în arbori-B