MODULUL 8 B-ARBORI

B-arborii sunt arbori balansaţi în care toate frunzele au aceeaşi adâncime. Acest tip de arbori reprezintă o generalizare a arborilor binari de căutare. Fiecare nod poate stoca mai multe chei, de la câteva, la mai multe mii. Alături de chei, un nod al arborelui poate conţine şi alte informaţii de interes stocate în structura de date.

O variantă utilizată frecvent a B-arborilor sunt arborii B+, care sunt B-arbori cu proprietatea că informaţiile suplimentare diferite de chei sunt stocate doar în frunze. Nodurile interne conţin doar chei.

1 Elemente de bază

Proprietăţi

1. Fiecare nod x are următoarele atribute (câmpuri):
   (a) $x.n$ numărul de chei stocate în nodul x.
   (b) Vectorul de chei $x.key$
   (c) $x.leaf =$ un câmp boolean care este TRUE, dacă x este frunză şi FALSE altfel.
   (d) Vectorul de legături către fiii lui x: $x.c$
2. Fiecare nod intern x are $x.n + 1$ fii.
3. Valorile cheilor $x.key(i), i = 1, \ldots, x.n$, separă valorile cheilor fiilor lui x după cum urmează: notăm cu $y_i = x.c(i), i = 1, \ldots, x.n + 1$ atunci:

$$y_1.key(1) \leq y_1.key(2) \leq \ldots \leq y_1.key(y_1.n) \leq x.key(1) \leq$$

$$\leq y_2.key(1) \leq y_2.key(2) \leq \ldots \leq y_2.key(y_2.n) \leq x.key(2) \leq$$

$$\ldots$$

$$\leq y_{x.n+1}.key(1) \leq y_{x.n+1}.key(2) \leq \ldots \leq y_{x.n+1}.key(y_{x.n+1}.n)$$

De fapt cheile $x.key(i-1)$ şi $x.key(i)$ definesc intervalul în care se pot afla valorile cheilor din subarborele $x.c(i), i = \overline{2, x.n}$. Cheile din subarborele de rădăcină $x.c(1)$ sunt mai mici decât cheia $x.key(1)$ şi cheile din subarborele de rădăcină $x.c(x.n+1)$ sunt mai mari decât cheia $x.key(x.n)$.

4. Toate frunzele au aceeaşi adâncime = adâncimea/înălţimea h a arborelui.
5. Numărul de chei ale unui nod este limitat inferior şi superior pe baza unei constante t, $t \geq 2$, numită ***gradul minim*** al arborelui, după cum urmează:
   (a) Fiecare nod x cu excepţia rădăcinii, conţine cel puţin $t-1$ chei şi are deci cel puţin t copii. Rădăcina conţine cel puţin o cheie.
   (b) Fiecare nod x conţine cel mult $2t-1$ chei, deci are cel mult $2t$ copii. Un nod care conţine $2t-1$ chei se numeşte ***nod plin***.

Un exemplu de B-arbore cu $t = 2$ în care cheile sunt litere ale alfabetului este prezentat în figura 1. Ordinea considerată este cea lexicografică.

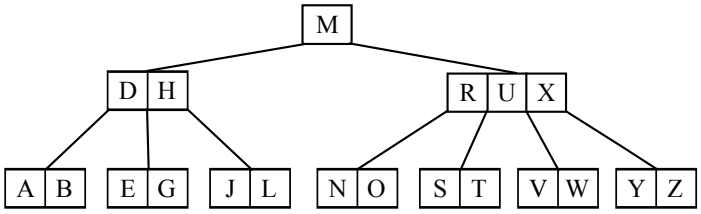

Figure 1: Exemplu de B-arbore

Înălţimea unui B-arbore

Considerăm un B-arbore de grad minim $t \geq 2$ cu n chei. Atunci înălţimea h a arborelui respectă inegalitatea:

$$h \leq \log_t \left(\frac{n+1}{2} \right)$$

Demonstraţie: vom calcula înălţimea unui B-arbore T cu n chei pornind de la un B-arbore T_1 cu aceeaşi înălţime h, dar cu număr minim de chei. Evindent $n \geq$ numărul de chei T_1.
Numărul de chei ale lui T_1, reprezentat în figura 2, se calculează astfel:

Nivel 0 : rădăcina conţine o singură cheie

Nivel 1 : conţine 2 noduri a câte $t-1$ chei

Nivel 2 : conţine $2 * t$ noduri a câte $t-1$ chei

Nivel 3 : conţine $2 * t^2$ noduri a câte $t-1$ chei

.....

Nivel h : conţine $2 * t^{n-1}$ noduri a câte $t-1$ chei

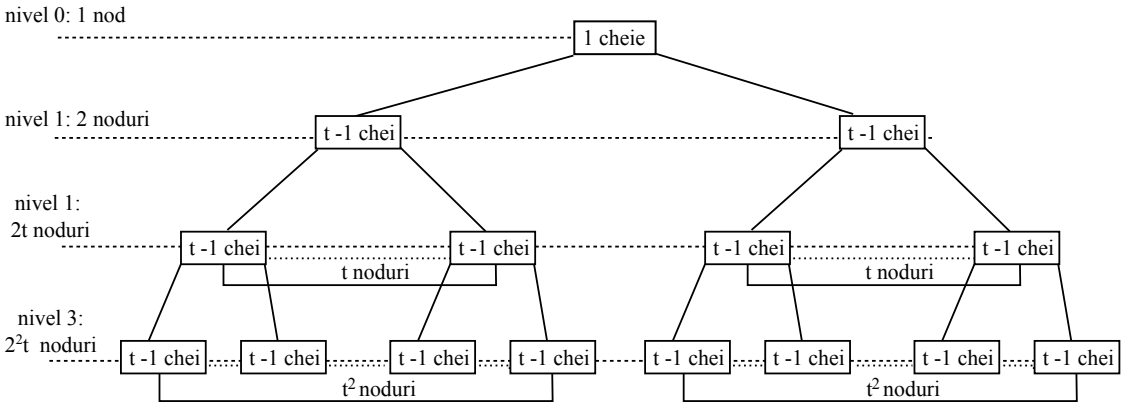

Figure 2: Numărul minim de chei într-un B-arbore de grad minim t şi înălţime h

În total numărul de chei ale arborelui T_1 este:

$$1 + 2(t-1) + 2t(t-1) + 2t^2(t-1) + ... + 2t^{n-1}(t-1) =$$
$$= 1 + 2(t-1)(1 + t + t^2 + ... + t^{n-1}) =$$
$$= 1 + 2(t-1)(t^h - 1)/(t-1) = 2t^h - 1.$$

Deci:

$$n \geq 2t^h - 1 \Rightarrow h \leq \log_t \left(\frac{n+1}{2} \right).$$

Înălţimea unui arbore este cu atât mai mică, cu cât gradul minim t este mai mare. Complexitatea operaţiilor de căutare, inserţie şi ştergere depinde de înălţimea arborelui, deci de t.

Domenii de utilizare

B-arborii pot fi utilizaţi cu succes pentru accesul rapid al datelor de pe hard-disk, prin minimizarea numărului de operaţii de citire/scriere. În memoria principală (în memoria RAM), cu acces rapid, se păstrează rădăcina B-arborelui, iar nodurile din subarbore se păstrează în memoria secundară, mai lentă. Accesul la un nod din memoria secundară se realizează în complexitate $O(\log_t n)$, unde baza logaritmului depinde de gradul minim al arborelui, iar numărul de accesări ale discului în acest scop este dat de lungimea drumului de la rădăcină la nodul căutat. Cu cât gradul minim al arborelui este mai mare, cu atât înălţimea este mai mică şi accesul la informaţie este mai rapid.

Multe baze de date utilizează B-arbori sau variante ale acestora pentru memorarea datelor.

2 Operaţii

Observaţie: precum s-a specificat anterior, B-arborii sunt utilizaţi în general pentru optimizarea accesului la memorie. Nodul rădăcină este memorat în memoria principală

(în memoria RAM), astfel încât accesul la rădăcină nu necesită operaţii de citire pe disc. Accesul la noduri interne necesită citire pe disc.

Notăm prin: DR, DW operaţiile de citire/scriere pe disc.

2.1 Căutarea unei chei

Căutarea într-un B-arbore este de fapt o generalizare a căutării binare, doar că spre deosebire de aceasta, pentru fiecare nod nu avem doar o decizie între două ramuri, ci avem o decizie între $n+1$ ramuri, n fiind numărul de chei ale nodului curent.

Funcţia BT_CAUT(x, k) este definită recursiv şi are ca parametri rădăcina x a subarborelui în care s-a ajuns cu căutarea şi cheia căutată, k. Funcţia returnează o pereche (x, i), unde x = nodul care conţine cheia k şi i indicele cheii în nodul x, adică $x.key(i) = k$. Dacă în arbore nu se găseşte cheia k, atunci funcţia returnează NULL.

Algoritm:

```
BT_CAUT(x,k)
  i=1
  cat timp i≤ x.n si x.key(i) < k
           i=i+1
  sfarsit cat timp
  daca i≤ x.n si x.key(i) = k atunci
           RETURN(x,i)
  altfel
           daca x.leaf = TRUE atunci
                RETURN NULL
           altfel
                DR(x.c(i)) //citeste de pe disc un nou nod
                RETURN BT_CAUT(x.c(i),k)
           sfarsit daca
  sfarsit daca
RETURN
```

Complexitate: căutarea în fiecare nod este de ordinul $O(t)$, iar căutarea în arbore depinde de înălţimea h a arborelui, care este de ordinul $\log_t n$. Astfel complexitatea totală a algoritmului este $O(t \log_t n)$.

2.2 Crearea unui arbore vid

Crearea unui arbore vid, în care apoi se inserează pe rând chei, presupune întâi alocarea unei zone de memorie pe disc, rezervată rădăcinii. Această procedură se realizează în timp $O(1)$.

```
BT_CREEAZA(T)
  aloca memeorie pentru x
  x.leaf=TRUE
  x.n=0
  DW(x) //scrie pe disc un nou nod
  T.rad=x
RETURN
```

2.3 Inserarea unei chei

Inserarea unei chei într-un B-arbore diferă de inserarea într-un arbore binar prin faptul că nu putem pur şi simplu crea o nouă frunză în B-arbore în care să inserăm cheia respectivă. Un astfel de procedeu ar duce la dezechilibrarea arborelui şi ar contraveni proprietăţii unui B-arbore de a avea toate frunzele pe acelaşi nivel.

Inserţia unei chei se realizează prin inserarea cheii într-un nod frunză deja existent. Problema apare atunci când frunza în care ar trebui inserat nodul este plină. În acest caz trebuie aplicată o procedură de DIVIZARE a frunzei în jurul cheii mediane (aflate pe poziţia din mijloc în sirul cheilor).

Operaţia de divizare a unui nod

Se consideră un nod plin y, deci cu $2t-1$ chei. Presupunem că y este al i-lea descendent al nodului x, care nu este plin. Atunci divizarea se realizează în modul următor:

- $x.n$ creşte cu o unitate: $x.n = x.n + 1$.
- cheile lui x începând de la poziţia i se deplasează cu o poziţie la dreapta în nod.
- se deplasează pointerii către descendenţii $x.c(i+1), \ldots, x.c(x.n)$ cu o poziţie spre dreapta în nodul x.
- cheia $y.key(t)$ urcă în nodul x pe poziţia i
- nodul y se divizează în două noduri noi care conţin câte $t-1$ chei şi anume primul nod conţine cheile $y.key(1), \ldots, y.key(t-1)$ şi al doilea nod conţine cheile $y.key(t+1), \ldots, y.key(2t-1)$
- noile noduri create devin descendenţii $x.c(i)$ şi $x.c(i+1)$.

Această procedură este ilustrată în figura 3.

Observaţii:

1. Dacă x este la rândul său plin, nu putem efectua divizarea. De aceea la inserţie trebuie să ne asigurăm deja pe parcursul căutării frunzei în care se realizează inserţia că pe drum nu întâlnim noduri pline. În caz contrar se realizează o divizare.

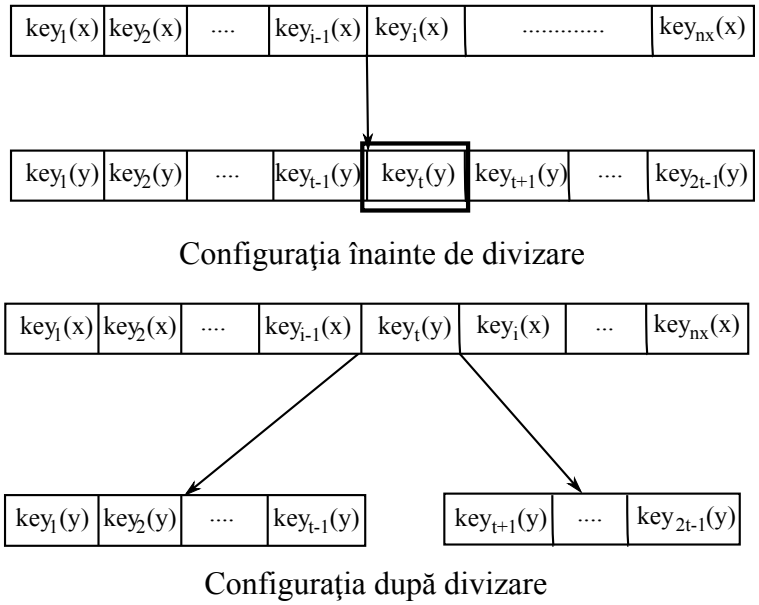

Figure 3: Divizarea unui nod plin

2. Dacă nodul plin y este rădăcina, atunci nu există un părinte x, în care să se insereze cheia mediană din y. Astfel, dacă nodul care trebuie divizat este chiar rădăcina trebuie procedat în modul următor:

   Divizarea rădăcinii

   - Se creează un nou nod vid, care va fi noua rădăcină.
   - Se leagă vechea rădăcină ca descendent al noii rădăcini.
   - Se realizează operaţia de divizare.

3. Înălţimea unui B-arbore creşte doar prin divizarea rădăcinii.

Exemplu: Inserarea cheii P în arborele din figura 4 cu $t = 2$. Căutarea începe de la

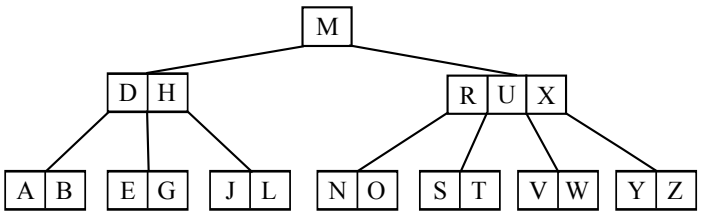

Figure 4: B-arbore în care se va realiza inserţia de chei

nodul rădăcină. $T.rad.n = 1$, iar $T.rad.key(1) < P$, deci trebuie coborât la descendentul $T.rad.c(2)$. Dar Se observă că $T.rad.c(2).n = 2t - 1 = 3$, deci nodul este plin. Înainte de a continua, nodul $T.rad.c(2)$ trebuie divizat. Se obţine arborele din fig 5:

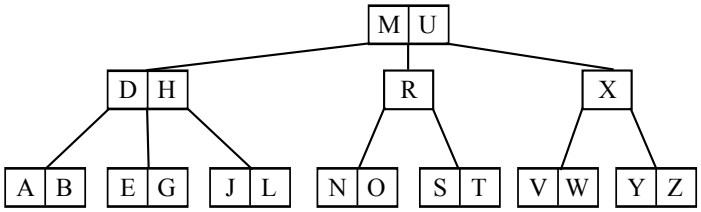

Figure 5: B-arborele după divizarea nodului cu cheile R, U, X

Acum $T.rad.key(1) = M < P$ şi $T.rad.key(2) = U > P$, deci se coboară la descendentul $x_1 = T.rad.c(2)$.

Avem: $x_1.n = 1, x_1.key(1) = R > P$, deci se coboară la nodul $x_2 = x_1.c(1)$, care este frunză. $x_2.n = 2 < 2t-1$, deci nodul nu este plin şi se poate insera P pe poziţia potrivită. Rezultă arborele din fig. 6.

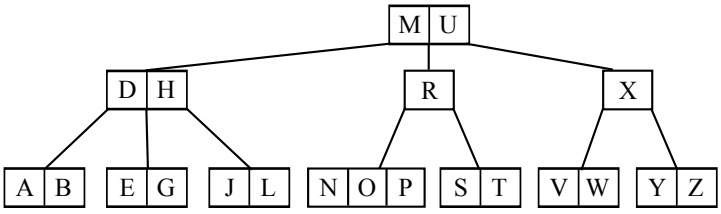

Figure 6: B-arborele rezultat după inserţia cheii P

2.4 Eliminarea unei chei

Eliminarea unei chei dintr-un B-arbore este ceva mai delicată decât inserţia, deoarece uneori trebuie eliminată şi o cheie dintr-un nod intern. Acest lucru ar avea însă ca urmare reducerea numărului de chei din nod şi ca urmare a numărului de descendenţi ai nodului respectiv. Este deci necesară o rearanjare a cheilor şi a descendenţilor în arbore. În plus, la eliminarea unei chei trebuie avut grijă ca numărul de chei dintr-un nod sa nu devină mai mic decât $t-1$. Astfel nu se poate pur şi simplu extrage o cheie dintr-un nod cu $t-1$ chei.

Aceste probleme se rezolvă prin operaţii de rotaţie şi prin fuziuni.

Rotaţia într-un B-arbore

Se consideră nodul x.

- O rotaţie la dreapta în jurul cheii $k = x.key(i)$:
  - mută ultima cheie a fiului $y_i = x.c(i)$ în nodul x în locul cheii k
  - mută cheia k pe prima poziţie în nodul $y_{i+1} = x.c(i+1)$

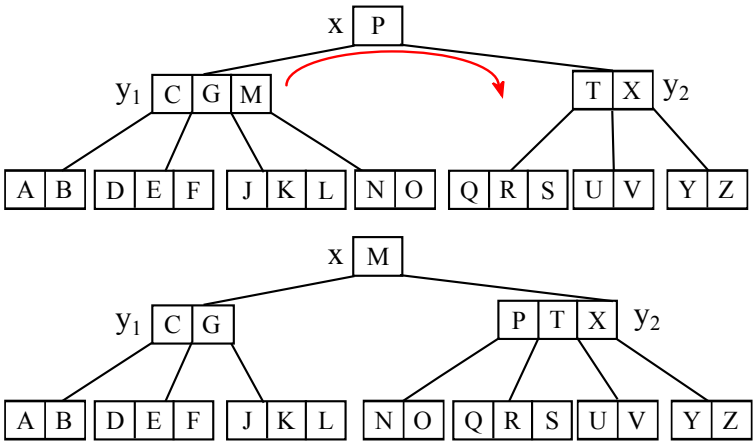

Figure 7: Rotaţie la dreapta în jurul cheii P din rădăcină.

  - mută ultimul fiu al lui y_i ca prim fiu al lui y_{i+1}.

- O rotaţie la stânga în jurul cheii $k = x.key(i)$:
  - mută prima cheie a fiului $y_{i+1} = x.c(i+1)$ în nodul x în locul cheii k
  - mută cheia k pe ultima poziţie din nodul $y_i = x.c(i)$
  - mută primul fiu al lui y_{i+1} ca ultim fiu al lui y_i.

Un exemplu de rotaţie la dreapta într-un B-arbore este prezentat în figura 7.

Fuziunea a două noduri vecine într-un B-arbore

O fuziune nu poate fi realizată decât între doi fraţi vecini, fii ai aceluiaşi nod x, şi care au ambii exact $t-1$ chei. De asemenea părintele x trebuie să aibă cel puţin t chei. Se consideră nodul x, iar descendenţii care fuzionează sunt $y = x.c(i)$ şi $z = x.c(i+1)$. Fuziunea lui y cu z presupune reunirea celor două noduri într-unul singur, iar cheia $x.key(i)$ coboară din nodul x în noul nod rezultat pe poziţia aflată între cheile lui y şi cele ale lui z.

Această operaţie este reprezentată grafic în fig. 8.

Un exemplu de fuziune a descendenţilor rădăcinii din fig. 9 este prezentat în **??**.

Observaţii:

- Înălţimea unui arbore se micşorează atunci când rădăcina are doi fii care fuzionează, ca în exemplul din fig. **??**.
- O fuziune se poate realiza doar dacă părintele nodurilor care fuzionează are cel puţin t chei. Din acest motiv, aşa cum la inserţie pe parcursul căutării nodului în care se inserează, toate nodurile pline întâlnite pe parcurs trebuie divizate, în cazul eliminării unei chei, pe parcursul căutării cheii respective, înainte de a coborî la un

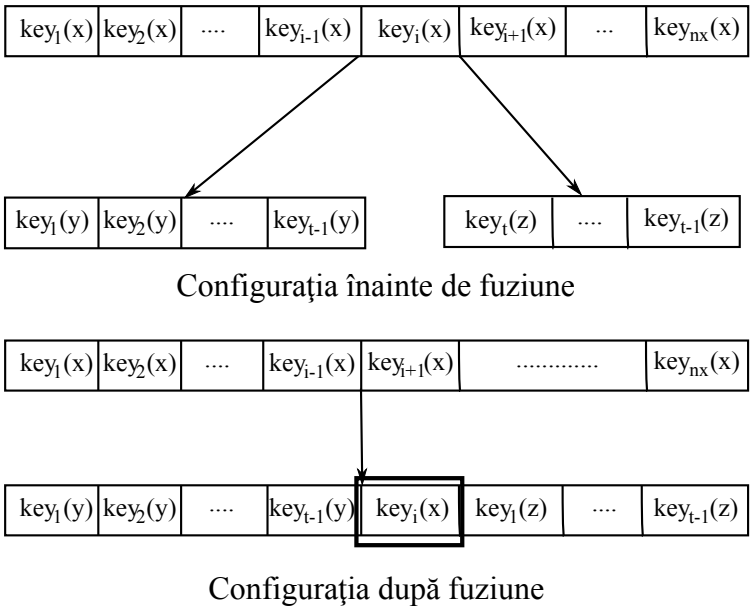

Figure 8: Fuziunea a două noduri vecine.

descendent, trebuie asigurat acestui descendent un minim de t chei.

Algoritm de eliminare a unei chei dintr-un B-arbore

Eliminarea chei k din B-arborele T cu gradul minim t va fi tratată în mod recursiv. Se porneşte de la rădăcină şi se coboară în arbore până la întâlnirea nodului ce conţine cheia k.

Notăm cu x nodul curent. Pe parcurs ne asigurăm că orice nod în care se coboară are cel puţin $t-1$ chei.

Cazul I: x este frunză şi k este cheie a lui x. Din faptul că s-a avut grijă ca nodurile în care se coboară să aibă cel puţin t chei, x are cel puţin t chei, deci pur şi simplu se extrage cheia k din nodul x şi se încheie algoritmul.

Cazul II: x nu este frunză şi $k = x.key(i)$. Atunci:

(a) **Dacă** $y = x.c(i)$ (copilul care îl precede pe k) are $y.n \geq t$ atunci se caută k' predecesorul lui k în subarborele de rădăcină y, se şterge recursiv k' din subarborele de rădăcină y şi se înlocuieşte k cu k' în x.

(b) **Altfel dacă** $y = x.c(i+1)$ (copilul care îi urmează lui k) are $y.n \geq t$ atunci se caută k'' succesorul lui k în subarborele de rădăcină y, se şterge recursiv k'' din acest subarbore şi se înlocuieşte k cu k'' în x.

(c) **Altfel** - adică ambii copii aflaţi de o parte şi de alta a lui k au exact $t-1$ chei - atunci: se realizează o fuziune între cei doi copii $x.c(i)$ şi $x.c(i+1)$. Astfel se obţine

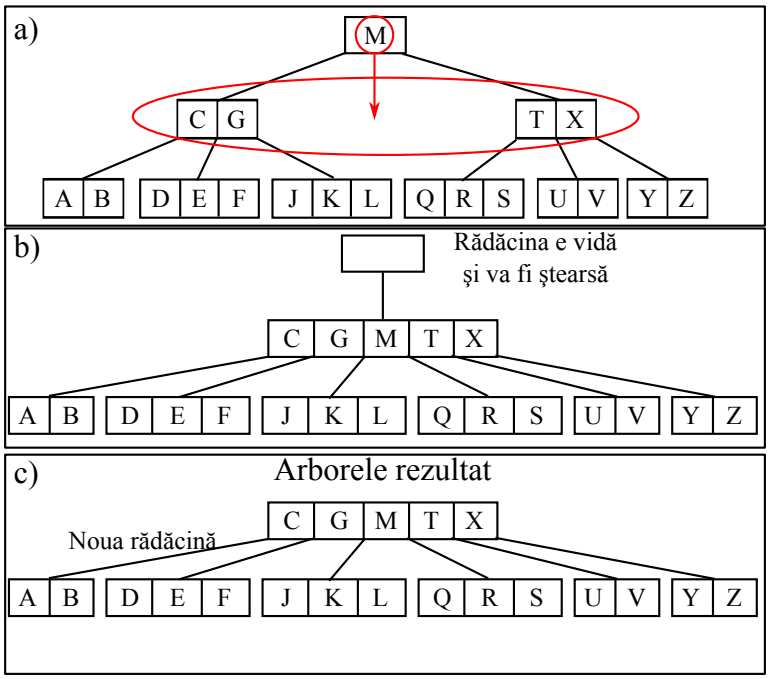

Figure 9: a) Nodurile care urmează să fuzioneze împreună cu cheia M a rădăcinii; b) Arborele după fuzionare; c) Rezultatul după eliminarea rădăcinii vide.

un nou nod, care va conţine cheia k. Aceasta se şterge recursiv din noul nod obţinut prin fuziune.

Cazul III: x nu este fruză şi k nu este cheie a lui x.
Atunci se determină i astfel încât $x.key(i-1) < k < x.key(i)$. Rezultă faptul că trebuie căutată cheia k în subarborele de rădăcină $x.c(i)$. Înainte de a coborî la descendentul $x.c(i)$ al lui x trebuie să ne asigurăm că acest descendent are cel puţin t chei. Dacă $x.c(i)$ are doar $t-1$ chei atunci:

(a) Dacă $x.c(i)$ are un frate vecin cu cel puţin t chei se realizează o rotaţie astfel încât o cheie din acel frate să urce la părintele x şi o cheie din x să coboare la nodul $x.c(i)$. Apoi se şterge recursiv k din $x.c(i)$.

(b) Dacă ambii fraţi vecini au $t-1$ chei trebuie realizată fuziunea lui $x.c(i)$ cu unul dintre aceşti fraţi. Apoi se şterge recursiv k din nodul rezultat prin fuziune.

Exemplu: Considerăm arborele din fig. 10 cu $t = 3$.

Se şterge cheia G: se porneşte de la rădăcină $G < M$ şi suntem în cazul III. Trebuie coborât la primul copil. Acesta însă are exact $t-1 = 2$ chei. Fratele drept al acestui nod are t chei, deci suntem în cazul III (a) $\Rightarrow$ rotaţie la stânga în jurul cheii M din rădăcină. Se obţine arborele din fig. 11 a).

Acum se coboară la nodul x care conţine cheile C, G, M. Se observă că acesta nu e frunză, dar conţine cheia $G = x.key(2)$ şi ambii fii $x.c(2)$ şi $x.c(3)$ au t chei, deci pot aplica cazul

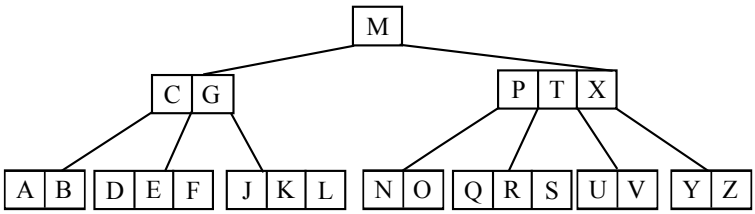

Figure 10: B-arbore cu $t = 3$ în care vor fi efectuate ştergeri de chei

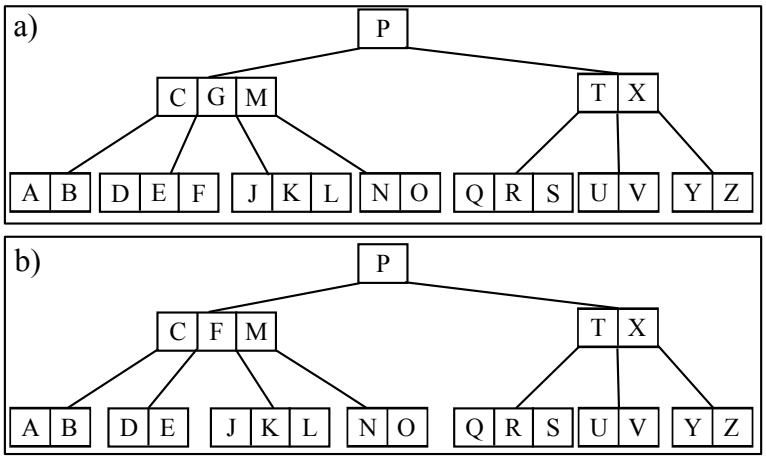

Figure 11: a) B-arborele din fig. 10 după rotaţie; b) Arborele după ştergerea cheii G.

II (a) $\Rightarrow$ caut predecesorul lui G, care este F. Înlocuiesc G cu F şi şterg F din fiul $x.c(2)$. Se obţine arborele din fig. 11 b).

Se şterge cheia B: se porneşte cu $x = T.rad$, se compară $B < x.key(1) \Rightarrow$ se studiază fiul $x.c(1)$. Acesta are t chei $\Rightarrow x = x.c(1)$.

Se observă că în acest moment $B < x.key(1) = C \Rightarrow$ se studiază fiul $x.c(1)$. Acest fiu nu are decât $t - 1$ chei, iar singurul său frate are tot $t - 1$ chei $\Rightarrow$ suntem în cazul III (b) $\Rightarrow$ trebuie realizată o fuziune între $x.c(1)$ şi $x.c(2)$. Se obţine arborele din fig. 12.

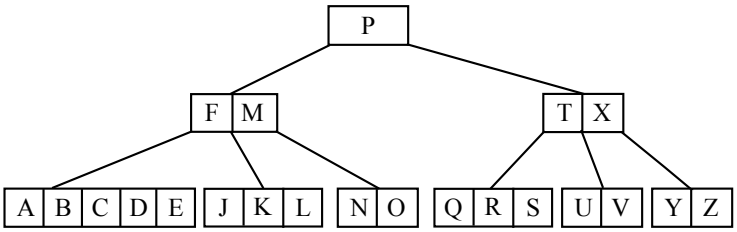

Figure 12: B-arborele după fuziunea nodurilor cu cheile A, B şi respectiv D, E împreună cu cheia C.

Se coboară acum la $x = x.c(1)$ şi se şterge cheia B din nod. Rezultă arborele din fig. 13.

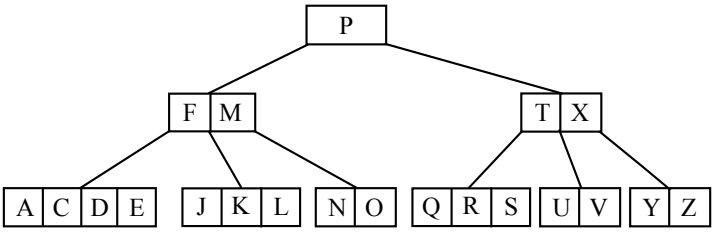

Figure 13: B-arborele după şergerea cheii B.

Se şterge cheia X: se porneşte de la rădăcina $x = T.rad$. $X > x.key(1) \Rightarrow$ se studiază fiul $x.c(2)$. Acesta are doar $t-1$ chei. Unicul frate al său are tot $t-1$ chei $\Rightarrow$ fuziune. Rădăcina iniţială rămâne fără nici o cheie, deci se şterge şi noua rădăcină va fi nodul rezultat în urma fuziunii. Rezultă arborele din figura 14.

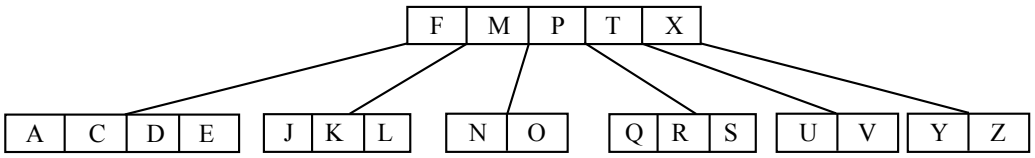

Figure 14: B-arborele după fuziunea fiilor rădăcinii şi ştergerea acesteia.

Cheia X se găseşte în rădăcină şi ambii fii, de o parte şi de alta a lui X au exact $t-1$ chei $\Rightarrow$ fuziune. Rezultă arborele din fig. 15.

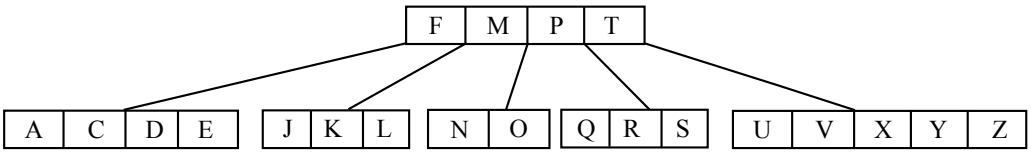

Figure 15: B-arborele după fuziunea nodurilor U, V şi Y, Z.

Acum se poate şterge X din frunza care îl conţine şi se obţine arborele din fig. 16.

Complexitate: Operaţiile într-un B-arbore presupun căutare în interiorul unui nod, care este $O(t)$ precum şi coborâre în arbore, care depinde de înălţimea h, deci este $O(\log_t n)$. Astfel complexitatea este $O(t \log_t n)$. Pentru t mare, poate fi utilizată căutarea binară într-un nod, astfel scade complexitatea acestei operaţii la $O(\log_2 t)$.

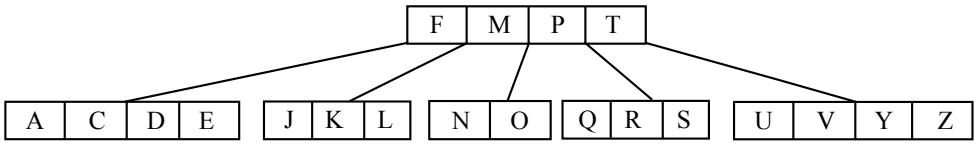

Figure 16: B-arborele după ştergerea cheii X.