UNELE ASPECTE ALE ARBORILOR DE TIP B

Adela GOREA, *lect. univ.,*
Universitatea de Stat „Alecu Russo" din Bălţi

Summary: *In some applications, data capture dominates query processing. For example, monitoring moving objects often requires more insertions and updates than queries. Data gathering using automated sensors often exhibits this imbalance. More generally, indexing streams is considered an unsolved problem.*

For those applications, B-tree indexes are good choices if some trade-off decisions are tilted towards optimization of updates rather than towards optimization of queries. This paper surveys some techniques that let B-trees sustain very high update rates, up to multiple orders of magnitude higher than traditional B-trees, at the expense of query processing performance. Not surprisingly, some of these techniques are reminiscent of those employed during index creation, index rebuild, etc., while other techniques are derived from well known technologies such as differential files and log-structured file systems.

Key-words: *tree, data structures, tree structures, representation, scroll, binary search, binary trees, B-trees.*

Introducere

Regăsirea unor informaţii în timp scurt este azi problema cea mai frecvent întîlnită în practică. Un astfel de exemplu este căutarea unui cuvînt în dicţionar sau a unui număr în cartea de telefon. Aceste informaţii sînt reprezentate, de regulă, sub forma unor înregistrări, fiecare înregistrare conţinînd un cîmp numit cheie, ce permite identificarea fiecărei înregistrări. O astfel de situaţie o vedem într-un dicţionar unde înregistrările au fiecare un cuvînt ce reprezintă cîmpul cheie pentru pronunţia şi definiţia corespunzătoare a cuvîntului respectiv.

Arborii sînt analizaţi ca structuri de date neliniare predominante atît în memoria internă, cît şi în memoria secundară. Azi, prin mărirea considerabilă a capacităţii de memorare a sistemelor de calcul, importanţa acestor structuri a crescut semnificativ. Doar în ultimii ani, modelul de date relaţional, apărut in 1970, s-a dezvoltat în numeroase direcţii, de la sisteme de informare geografică, tipuri de date abstracte şi modele orientate obiect, la baze de date temporale şi procesare analitică on-line [2].

În momentul actual, eficienţa procesului de căutare multidimensională sau căutare după chei secundare pentru datele unei colecţii este o cerinţă imperativă. Dar se constată că problema căutării pe domenii multidimensionale e mult mai dificilă decît similara ei în cazul unidimensional. Se constată azi că şi pentru probleme simple de căutare multidimensională, nu avem tehnici unanim recunoscute ca fiind cele mai bune. Dar, o mare realizare este că avem multe structuri de date, de cele mai multe ori structuri arborescente prin care se rezolvă probleme de căutare conceptual similare. De obicei, mulţimea acestor date este dinamică şi problema se reduce la alegerea structurii de date potrivite în funcţie de tipul şi statistica datelor care urmează a fi indexate, de proprietăţile intrinseci ale datelor multidimensionale şi nu în ultimul rînd de utilizarea lor.

Structura de date foarte des folosită pentru implementarea indecşilor este arborele de căutare. Articolele memorate pot fi oricît de complexe, dar ele conţin un cîmp numit cheie ce serveşte la identificarea acestora. Să notăm cu *C* mulţimea chei-

lor posibile ce vor trebui regăsite cu ajutorul arborelui de căutare. Dacă arborele de căutare este astfel construit încît foloseşte o relaţie de ordine totală pe *C*, atunci vom spune că arborele de căutare este bazat pe ordinea cheilor. Arborii de căutare, bazaţi pe ordinea cheilor, sînt de două feluri [3]:

(1) arbori binari de căutare;

(2) multicăi de căutare.

Performanţele unui index se îmbunătăţesc în mod semnificativ prin mărirea factorului de ramificare a arborelui de căutare folosit. Arborii multicăi de căutare sînt o generalizare a arborilor binari de căutare. Astfel, unui nod oarecare, în loc să i se ataşeze o singură cheie care permite ramificarea în doi subarbori, i se ataşează un număr de m chei, ordonate strict crescător, care permit ramificarea în *m + 1* subarbori. Numărul m diferă de la nod la nod, dar în general pentru fiecare nod trebuie să fie între anumite limite (ceea ce va asigura folosirea eficientă a mediului de stocare). Cele m chei ataşate unui nod formează o pagină. Determinarea poziţiei cheii căutate în cadrul unui nod se realizează secvenţial în cazul paginilor cu număr mic de chei sau prin căutare binară [1]. Un exemplu de arbore multicăi de ordinul *N* este prezentat în figura 1.

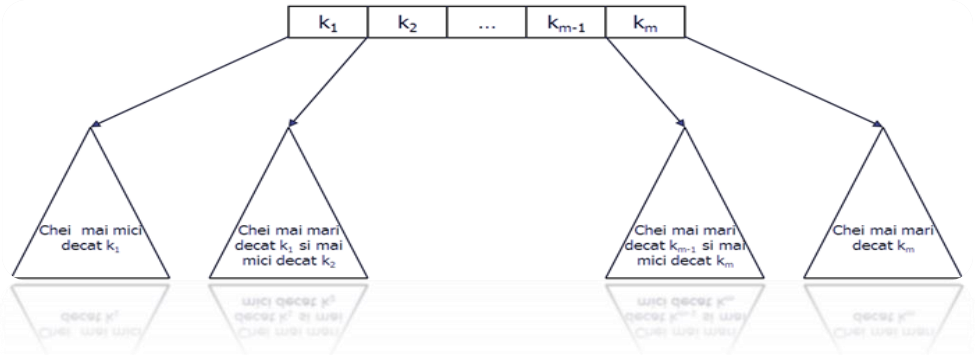

Figura 1. *Arbore de tip B*

Proprietăţile arborilor de tip B

Un arbore B de ordin *N* este un arbore multicăi de căutare şi are următoarele proprietăţi:

(I) toate nodurile frunză sînt pe acelaşi nivel;

(II) rădăcina are cel puţin doi descendenţi, dacă nu este frunză;

(III) fiecare pagină conţine cel puţin *n/2* chei (excepţie face rădăcina care poate avea mai puţine chei, dacă este frunză);

(IV) nodul este fie frunză, are n + 1 descendenţi (unde n este numărul de chei din nodul respectiv, $n/2 \leq m \leq n-1$);

(V) fiecare pagină conţine cel mult n-1 chei; din acest motiv, un nod poate avea maxim n descendenţi.

Proprietatea (*I)* menţine arborele balansat. Proprietatea *(II)* forţează arborele să se ramifice devreme. Proprietatea *(III)* ne asigură că fiecare nod al arborelui este cel puţin pe jumătate plin.

Înălţimea maximă a unui arbore B dă marginea superioară a numărului de accese la disc necesare pentru a localiza o cheie.

Operaţiile de bază ale arborelui de tip B.

Procesul de *căutare* într-un arbore B este o extindere a căutării într-un arbore binar. Operaţia de căutare în arborele B se realizează comparînd cheia căutată *x* cu cheile nodului curent, plecînd de la nodul rădăcină. Dacă nodul curent are *n* chei, atunci se disting următoarele cazuri [5]:

(I) $ci < x < ci+1, i=1,n$ se continuă căutarea în nodul indicat de Pi;

(II) $cn < x$ se continuă căutarea în nodul indicat de Pn;

(III) $x < c0$ – se continuă căutarea în nodul indicat de P0.

Arborele B suportă *căutarea secvenţială* a cheilor. Arborele este traversat secvenţial prin referirea în inordine a nodurilor. Un nod este referit de mai multe ori întrucît el conţine mai multe chei. Subarborele asociat fiecărei chei este referit înainte ca următoarea cheie sa fie accesata. Arborii B sînt optimi pentru accesul direct la o cheie. Pentru accesul secvenţial la o cheie nu se obţin performanţe satisfăcătoare [7].

Condiţia ca toate frunzele să fie pe acelaşi nivel duce la un comportament caracteristic pentru arborii de tip B:

(1) faţă de arborii binari de căutare,

(2) arborilor de tip B nu le este permis să crească la frunze;

(3) sînt forţaţi sa crească la rădăcină.

Operaţia de *inserare* a unei chei în arborele B este precedată de operaţia de căutare. În cazul unei căutări cu succes nu se mai pune problema inserării întrucît cheia se afla deja în arbore. Dacă cheia nu a fost găsită, operaţia de căutare se va termina într-un nod frunză. În acest nod frunză se va insera noua cheie, în funcţie de gradul de umplere al nodului frunzele afectate, se disting în următoarele cazuri:

(I) nodul are mai puţin de *n–1* chei, inserarea se efectuează fără să se modifice structura arborelui;

(II) nodul are deja numărul maxim de *n–1* chei; în urma inserării nodul va avea prea multe chei, de aceea el va "fisiona". În urma fisionării vom obţine două noduri care se vor găsi pe acelaşi nivel şi o cheie mediană care nu se va mai găsi în nici unul din cele două noduri.

Se observă că procesul de inserare a unei chei garantează că fiecare nod intern va avea cel puţin jumătate din numărul maxim de descendenţi. În urma operaţiilor de inserare arborele va deveni mai înalt şi mai lat, figura 2.

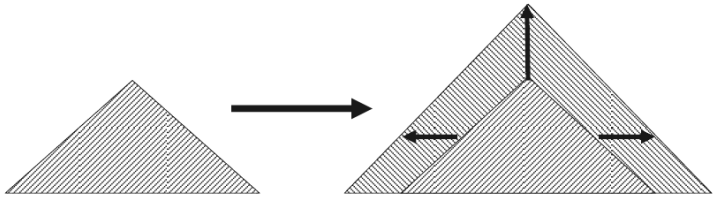

Figura 2. *Modificările dimensionale ale unui arbore B după efectuarea inserării*

Uzual, în special pentru arbori B de ordin mare, un nod părinte are suficient spaţiu disponibil pentru a primi valoarea unei chei şi un pointer către un nod descendent. În cel mai rău caz algoritmul de fisionare este aplicat pe întreaga înălţime a arborelui. În acest mod arborele va creşte în înălţime, lungimea drumului de căutare crescînd cu 1.

O sinteză a algoritmul de inserare a unei valori de cheie în arbore este prezentat în figura următoare [4].

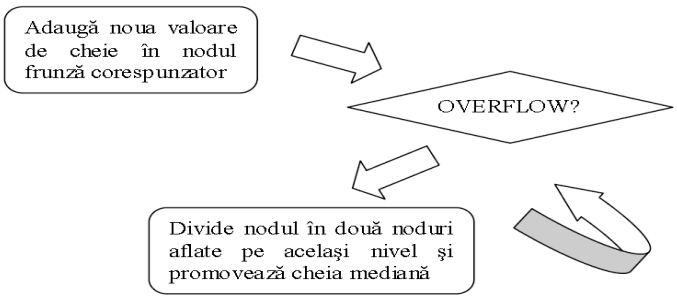

Figura 3. *Algoritmul de inserare a unei chei în arbore*

(I) inserează noua valoare de cheie în nodul frunză corespunzător;
(II) nodul_curent = nodul_frunza;
(III) while (starea pentru nodul_curent este *OVERFLOW*:
- divide nodul_curent în două noduri aflate pe acelaşi nivel şi promovează cheia mediană în nodul părinte pentru nodul_curent;
- nodul_curent = nodul părinte pentru nodul_curent.

Operaţia de *ştergere* dintr-un arbore B este ceva mai complicată decît operaţia de inserare. Operaţia de ştergere se realizează simplu dacă valoarea de cheie care urmează a fi ştearsă se află într-un nod frunză. Dacă nu, cheia va fi ştearsă logic, fiind înlocuită cu o alta, vecină în inordine, care va fi ştearsă efectiv. În urma ştergerii se disting următoarele cazuri [6]:

(I) dacă nodul conţine mai mult de *N/2* chei, ştergerea nu ridică probleme;
(II) dacă nodul are numărul minim de *N/2* chei , după ştergere numărul de chei din nod va fi insuficient. De aceea se împrumută o cheie din nodul vecin dacă acesta are cel puţin *N/2* chei, caz în care avem de-a face cu o *partajare*.

Marele avantaj al utilizării unor structuri de date arborescente este acela că facilitează regăsirea rapidă a unor submulţimi ale datelor reprezentate şi astfel, operaţiile se implementează cu algoritmi care au un timp de execuţie deosebit de performant. Mai mult, structurile de date arborescente sînt utile datorită abilităţii lor de a se focaliza asupra unei submulţimi a datelor iniţiale.

Datorită avantajelor lor, unele structuri arborescente, într-o varietate foarte mare, sînt generatoare de metode moderne şi foarte utilizate la reprezentarea şi arhivarea datelor.

Referinţe bibliografice

1. CORMEN, C., LEISERSON, R., Introducere în algoritmi, Editura Computer Libris Agora, Cluj-Napoca, 2000.
2. D. Lomet. 2001. The Evolution of Effective B-tree: Page Organization and Techniques: A Personal Account. In SIGMOD Record.
3. Goetz, Graefe, Sorting and Indexing with Partitioned B-Trees. Article Conference on Innovative Data Systems Research, 2003.
4. Ionesco, Constantin, Structuri arboriscente cu aplicațiile lor, Editura Tehnică, București, 1998.
5. VLDB Journal, The International Journal on Very Large Data Bases 2 (4), 361 – 406.
6. Mond, Y, Raz, Y. Concurrency Control in B+-trees Databases Using Preparatory Operations. In Eleventh International Conference on Very Large Data Bases 2002.
7. Михалкович С. Основы программирования: Динамические массивы. Списки. Ассоциативные массивы. Деревья. Рос-тов н/Д.: УПЛ ЮФУ, 2007. 48 с.