Rodzina B-drzew [24.01.2011]

Piotr Walkowski, 221135

Wrocław, 03.02.2011

Spis treści

1 Rodzina B-drzew

Podstawową jednostką danych w dyskowych operacjach wejscia wyjscia jest blok. Kiedy dane z dysku są odczytywane, do pamięci wczytywany jest cały blok, a podczas i zapisywania na dysku, zapisywany jest również cały blok. Zawsze, kiedy żąda się jakichś danych z dysku, trzeba je najpierw znalezc, a nastepnie umiescić głowicę nad odpowiednim obszarem i poczekać, aż dysk obróci się o tyle, aby głowica mogła je odczytać. Oznacza to, że na czas dostępu do danych składa się przynajmniej kilka czynników:

$$\textit{czas dostępu} = \textit{czas wyszukiwania} + \textit{opóźnienie rotacyjne} + \textit{czas transferu}$$

Bez wątpienia proces tenże jest wyjątkowo powolny, w porównaniu z przenoszeniem danych w pamięci. Szczególnie długi jest pierwszy składnik, *czas wyszukiwania*, jako, że zależy on od mechanicznego przemieszczania głowicy nad właściwa sciezkę dysku. *Opóźnienie rotacyjne* to czas potrzebny na to, aby głowica znalazła się dokładnie nad odpowiednim blokiem i średnio jest to czas potrzebny na wykonanie połowy obrotu. Jeżeli program stale używa informacji z zewnętrznych nośników, projektując dany system należy okreslić ich naturę. Przykładowo drzewo binarne poszukiwania może rozciągać się na wiele różnych bloków dysku, tak, że np. średnio ma miejsce dostęp do dwóch bloków. Kiedy takie drzewo jest często używane, dostępy te mogą znacząco spowolnić działanie programu. Poza tym wstawianie kluczy do drzewa i ich usuwanie wymaga także wielu dostępów do poszczególnych bloków danych. Binarne drzewo poszukiwania, które jest notabene dobrym narzędziem, kiedy znajduję się calkowicie w pamięci okazuje się być nieodpowiednie. W przypadku zewnętrznych nośników danych taka struktura danych zawodzi z racji konieczności sięgania stale do bloków danych co znacznie spowalnia pracę. Prostą konkluzją jest to, że lepiej, w takim przypadku sięgać po duże ilości danych na raz zamiast przemieszczać się ciągle w celu odczytu niewielkich bloków danych. Najlepiej jest zatem tak zorganizować dane, aby takich odczytów było możliwie najmniej.

2 B-Drzewo

Jedna ze struktur danych, która znajduje szczególne zastosowanie, w programach bazodanowych jest **B-drzewo**. Została ona wymyślona przez parę: **Bayer**, **McCreight**, w roku 1972. B-drzewa są ściśle związane z nośnikami zewnętrznymi i mogą być dostosowane do nośnika tak, aby minimalizować związane z nim opóźnienia. Ważna cecha tej struktury jest to, że wielkość każdego węzła może być równa wielkości bloku. Liczba kluczy, w węźle może być różna, w zależności od wielkości tychże kluczy, organizacji danych czy oczywiście od wielkości bloku. Ilość iformacji zapisywana w pojedynczym weźle B-drzewa może być więc stosunkowo duża.

Formalnie **B-Drzewo** to struktura drzewiasta struktura danych z korzeniem T, spełniająca następujące warunki:
(1) Korzeń, jeśli nie jest liście to ma przynajmniej dwa poddrzewa
(2) Każdy węzeł niebędący liściem ani korzeniem ma k-1 kluczy i k wskaźników na poddrzewa. Gdzie $[\frac{m}{2}] <= k <= m$

(3) Każdy węzeł który jest liściem ma k-1 kluczy, gdzie [m-2] <=k<=m
(4) Wszystkie liście znajdują się na tym samym poziomie

Zgodnie z podanymi warunkami B-drzewo jest zawsze przynajmniej do połowy wypełnione, ma kilka poziomów i jest całkowicie zrównoważone. Węzeł B-drzewa często implementuje sie jako klase zawierajaca tablice m-1 komorek na klucze, tablice z m komórkami na wskazniki na inne wezły oraz ewentualnie inne wskaźniki ułatwiające obsługę drzewa, jak liczba kluczy w węźle oraz flaga określająca czy dany węzeł jest liściem.
Istnieje pewne podobienstwo do drzew czerwono czarnych, bowiem w obojgu strukturach liście znajdują się na tym samym poziomie, a także każda operacja jest wykonywana w czasie logarytmicznym wzgledem wysokości drzewa.

2.1 oszacowanie wysokości

Przypadek pesymistyczny dla wyszukiwania w drzewie to sytuacja kiedy B-drzewo ma najmniejszą dozwoloną liczbę wskaźników, w węzłach niebędących korzeniem, $q = [\frac{m}{2}]$, zaś szukany jest lisć (który może, ale nie musi istnieć).
W takim wypadku jeżeli wysokość oznaczymy jako h, jest:

$$\begin{gathered}
\text{Jeden klucz w korzeniu} \\
2(q-1) \text{ kluczy na drugim poziomie} \\
2q(q-1) \text{ kluczy na trzecim poziomie} \\
\dots \\
2q^h - 2(q-1) \text{ kluczy w lisiach (poziom h)}
\end{gathered}$$

co korzystając z wzoru na sumę elementów ciągu geometrycznego dają:
$1 + (\sum_{i=0}^{h-2} 2q^i)(q-1)$ kluczy, w B-drzewie.

w związku z tym można wyrazić liczbę kluczy B-drzewa, w przypadku pesymistycznym jako:

$$1 + 2(q-1)(\sum_{i=0}^{h-2} q^i) = 1 + 2(q-1)(\tfrac{q^{h-1}-1}{q-1}) = -1 + 2q^{h-1}$$

Zatem związek liczby kluczy n i wysokości h wyraża się:

$$n >= -1 + 2q^{h-1}$$

Po zlogarytmowaniu względem zmiennej h otrzymujemy:

$$h <= log_q \tfrac{n+1}{2} + 1$$

2.2 podstawowe operacje

2.3 Wyszukiwanie w drzewie

2.3.1 Opis

Algorytm znajdowania klucza w B-drzewie jest dość prosty i nie wymaga specjalnego komentarza. Najprościej biorąc jest on 'rozszerzeniem' wyszukiwania stosowanego w drzewach BST z tym, że, w każdym węźle decydujemy do którego z dzieci zejść, gdzie ich liczba jest zwykle dużo większa od dwóch.

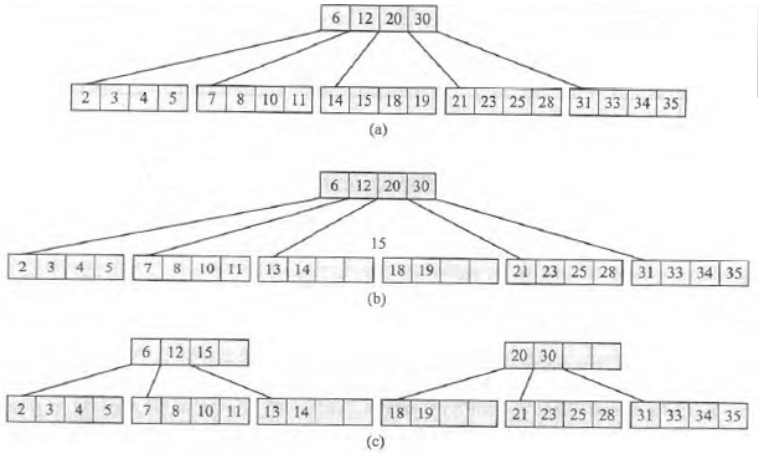
2.3.2 Pseudokod

```
BTreeNode find(key K, Node *n)
  if(node != 0)
    for(i=1; i <= node->keysArity && node->keys[i-1] < K; i++);
    if(i > node->keysArity || node->keys[i-1] > K)
       return find(K, node->ptr[i-1]);
    else return node
  else return nil
```

2.4 Wstawianie

2.4.1 Opis

Zarówno wstawianie nowego klucza, jak i jego usuwanie są operacjami trudnymi, jeśli wziąć pod uwagę fakt. ze wszystkie liście muszą znajdować się na ostatnim poziomie. Implementowanie wstawiania elementów staje się prostsza, kiedy zmieni się metodę tworzenia drzewa. Jeśli pierwszy napotkany klucz jest najmniejszym z kluczy, klucz ten jest wstawiany do korzenia, a ostatecznie korzeń nie ma wcale lewego poddrzewa, chyba że podjęte zostaną kroki w celu zrównoważenia drzewa. Drzewo mozna jednak budować takze od dołu tak, aby korzeń stale się zmieniał, a jego ostateczna wartość była znana dopiero po zakończeniu budowy drzewa. Tak właśnie wstawia się klucze do B-drzew. Jeśli dany jest klucz do wstawienia, przechodzi się bezpośrednio do liścia i, jeśli jest tam miejsce, wstawia się go tam. Kiedy liść jest już pełny, tworzony jest nowy liść, klucze są dzielone między te dwa liście (stary i nowy), jeden zaś przechodzi do rodzica. Jeśli takze rodzic jest pełny, proces jest powtarzany aż do skutku; może się zdarzyć, że stary korzeń zostanie zastąpiony przy tym nowym. Aby opisany proces bardziej uporządkować, omówione zostaną trzy typowe sytuacje związa- ne ze wstawianiem do B-drzew nowych kluczy:



Złożona operacja wstawiania klucza 13 do drzewa

2.4.2 Pseudokod

```
void insert(k)
  znajdz lisc, w ktorym wstawiony zostanie klucz K
    while(true)
      znajdz odpowiednie miejsce na K w tablicy keys;
      if wezel node nie jest pelny
         wstaw K i zwieksz keyArity;
         return
      else podziel node na node1 node2 # node1 = node, node2 jest nowy
         rozloz klucze i wskazniki rowno miedzy node1 i node2 oraz zainicjuj keyArity
         K = srodkowy klucz sposrod kluczy z node i K
         if node byl korzeniem
            utworz nowy korze  jako rodzica node1 i node2
            wstaw K i wskazniki na node1 i node2 w korzeniu, ustaw jego keyArity na 1
         else node = jego rodzic # teraz przetwarzany jest rodzic wezla node
            return
```

16

3 Usuwanie

Usuwanie z B-drzewa jest analogiczne do wstawienia, choć jest trochę bardziej skomplikowane. Zamiast podawać pseudokod dla operacji usuwania, lepiej opisać ją słowami. Usuwanie klucza jest realizowane za pomocą rekurencyjnej procedury

Wszystko jest realizowane dzięki rekurencyjnej procedurzę B-Tree-Delete. Załóżmy, że chcemy usunąć klucz k z poddrzewa o korzeniu w węźle x. Procedura B-Tree-Delete jest zaprojektowana w taki sposób, żeby byl spełniony następujący niezmiennik: Kiedykolwiek następuje wywołanie rekureneyjne B-Tree-Delete dla węzła x różnego od korzenia, w tym węźle jest co najmniej t kluczy, gdzie t jest minimalnym stopnian węzła w drzewie. Zauważmy, iż dzięki temu, że x ma o jeden klucz więcej, niż jest to wymagane, istnieje możliwość przesunięcia w razie konieczności jednego klucza do syna, do którego następuje zejście rekurencyjne. Powyższy niezmiennik umożliwia usunięcie klucza

z drzewa w jednym przejściu W dół drzewa, bez konieczności powracania w kierunku korzenia (z jednym wyjątkiem, o którym poniżej)

3.0.3 pseudokod

Usuwanie z B-drzewa wykonujemy następująco:

1. Jeśli klucz k jest w węźle x i x jest liściem to usun klucz k z x

2. Jeżeli klucz k jest w węźle x i x jest węzłem wewnętrznym to wykonaj co następuje:

(a) Niech y będzie synem x poprzedzającym k. Jeśli y ma co najmniej t kluczy, to w poddrzewie o korzemu w y wyznacz poprzednik k' klucza k. Rekurencyjnie usuń k' i w węźle x zastąp k przez k' (Wyznaczenie k' i usunięcie go z poddrzewa moze być wykonane w jednym przejsciu w dół drzewa)

(b) Symetryczme, jeśli syn z, ktory występuje po k w węźle x, ma co najmniej t kluczy, to wyznacz następnik k' dla k w poddrzewie o korzeniu w z. Rekuręncyjnie usuń k' i zastąp k przez k' w x. (Wyznaczenie k' i usunięcie go z poddrzewa może być wykonane w jednym przejści w dół poddrzewa).

(c) Jeśli obaj synowie y i z mają tylko po t-1 kluczy, to przenieś k oraz wszsytko z węzła z do y. W wyniku tej operacji klucz k i wskaźnik do z zostają usunięte z x. Następnie zwolnij pamięć przydzieloną dla z i usuń rekurencyjnie k z y.
3. Jeśli klucz k nie wystepuje w wewnetrznym węźle x, to wymaż korzeń $c_i[x]$ poddrzewa, w którym musi znajdować się k (jeśli tylko jest w drzewie). Jeśli $c_i[x]$ ma tylko t - 1 kluczy, to wykonaj podkrok 3a lub 3b w celu zagwarantowania, że zejście rekurencyjne następuje do węzła zawierającego conajmniej t kluczy. Następnie usuń rekurencyjnie k z właściwego poddrzewa.

(a) Jeśli w węźle $c_i[x]$ jest tylko t-1 kluczy, ale jeden z jego sąsiednich braci ma t kluczy, to umieść w $c_i[x]$ dodatkowy klucz, przesuwajac odpowiedni klucz z x, a w jego miejsce przenosząc klucz z lewego lub prawego brata - z tego, który zawiera t kluczy. Na koniec przesuń jeszcze z wybranego brata do $c_i[x]$ wskaźnik do odpowiedniego syna.

(b) Jeśli $c_i[x]$ i sąsiedzi bracia mają po $t-1$ kluczy to połącz c_i z jednym z sąsiednich braci, przesuwając odpowiedni klucz z x do nowo powstałego węzła. Przesunięty klucz jest kluczem środkowym w nowym węźle.

4 B*-drzewa

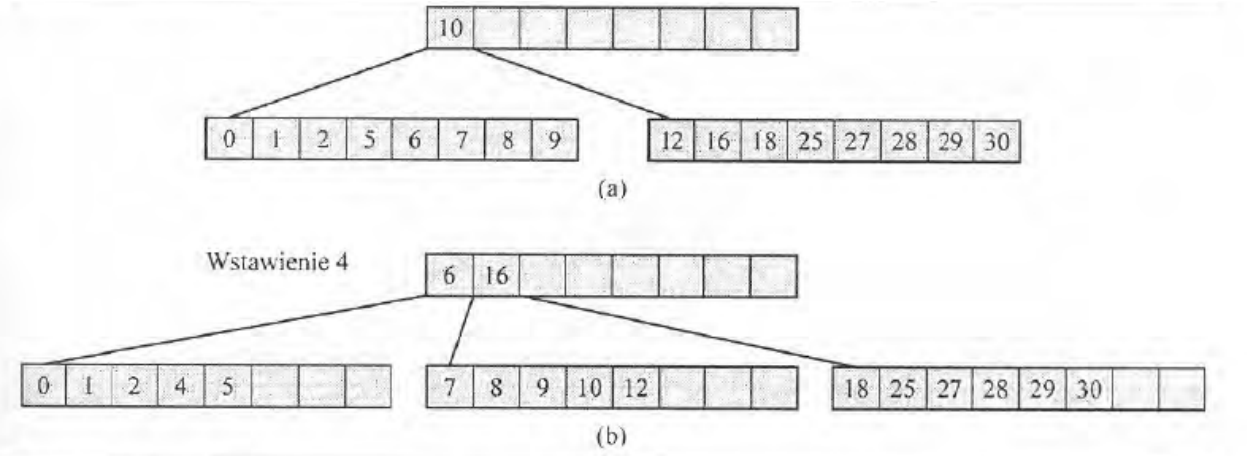
4.1 Charakterystyka

B*-drzewo to odmiana B-drzewa stworzona przez Donalda Knutha, jego nazwa została za- proponowane przez Douglasa Comera. W B*-drzewie wszystkie węzły poza korzeniem muszą być

wypełnione przynajmniej w dwóch trzecich, a nie tylko w połowie, jak zwykłe B-drzewa. Ściślej rzecz biorąc, liczba kluczy we wszystkich węzłach B-drzewa rzędu m niebędących korzeniami musi wynosić k dla $[\frac{2m-1}{3}] <= k <= m-1$ Częstość podziałów węzłów ogranicza się, opóźniając podział, a kiedy juz do niego dochodzi, dzieląc dwa węzły na trzy, a nie jeden na dwa. Badania statystyczne pokazują, że średnia zajętość B*-drzew wynosi 81/100 (Leung l984). Struktura ta znalazła zastosowanie chociażby w systemie plików reiserFS.

4.2 Operacje

Wobec tego, ze kazdy węzeł B-drzewa odpowiada blokowi nośnika zewnętrznego, sięganie do jednego węzła oznacza jeden dostęp do tego nośnika, co jest kosztowne w porównaniu z dostępem do kluczy węzła znajdujących się w pamięci. Wobec tego im mniej węzłów zostanie utworzonych. tym lepiej. Podział w B*-drzewie opóźnia się, próbując rozłożyć klucze między węzeł a jego sąsiada w razie niedopełnienia. Łatwo zauważyć,że nie tylko równo dzieli się klucze, ale w węźle, który był pełny, pojawiają się puste miejsca pozwalające wstawić nowe klucze.Ponadto jeśli pelny jest także sąsiad, dochodzi do podziału - tworzony jest jeden nowy węzeł, klucze z węzła i jego sąsiada (wraz z kluczem rozdzielającym z rodzica) są równomiemie rozkładane między trzy węzły, zaś do rodzica wstawiane są dwa klucze rozdzielające. Wszystkie trzy węzły uczestniczące w podziale będą na pewno pelne przynajmniej w dwóch trzecich.



O ile węzeł oraz jego sąsiedzi są pełni, w B* drzewie dochodzi do podziału - tworzony jest nowy węzeł, natomiast klucze są rozdzielane równo pomiędzy trzy.

5 B+drzewa

5.1 Charakterystyka

Wobec faktu, ze jeden węzeł B-drzewa odpowiada jednej stronie lub blokowi nośnika zewnętrznego, przejście z jednego węzła do innego wymaga czasochłonnego zmieniania strony. Wobec tego wskazana byłaby niewielka ilość dostępów do węzła. Przchodzenie bezpośrednie po drzewie pozwala łatwo to zaimplementować, ale w przypadku węzłów innych niz końcowe równocześnie pokazywany jest tylko jeden klucz, a potem konieczny jest dostęp do innej strony. Wobec tego dobrze byłoby rozszerzyć B-drzewa tak, aby umożliwić szybsze sięganie do kolejnych danych niż przy zastosowaniu przechodzenia bezpośredniego. Rozwiązanie takie udostępnia B+drzewo W B-drzewie odwołania do danych mogą pojawiać się w dowolnych węzłach; zaś w B+drzewie tylko w liściach. Węzły wewnętrzne B+drzewa to indeksy pozwalające na szybki dostęp do danych; ta część drzewa nazywana jest zbiorem indeksów. Liście mają imią strukturę niż reszta węzłów B+drzewa i zwykle są połączone w jak lista cykliczna. Warto zauważyć, ze węzły wewnętrzne zawierają klucze, wskaźniki i licznik kluczy. Liście zawierają klucze, wskaźniki rekordów W pliku danych związanym z kluczami oraz wskaźniki na następny liść. Za twórcę tejże struktury uważa się **D. Comera**, który na ramach ACM zaprezentował pracę pt. "the ubiquitous B-tree"w roku 1979, jednakże sam autor wspominał, że koncepcje B+drzew były rozważane nieco wcześniej (być może nawet w 1974) przez pracowników z IBM research.

5.2 Operacje

Operacje B+-drzewa nieznacznie różnią się od operacji B-drzewa. Wstawianie klucza do liścia mającego jeszcze wolne miejsca wymaga ustawienia kluczy tego liścia w kolejności. W zbiorze indeksów nie powoduje to zadnych zmian. Jeśli klucz jest wstawiany do pełnego już liścia, liść jest dzielony, nowy liść jest dołączany do drzewa, zaś klucze są dzielone między stare i nowy liść; pierwszy klucz z nowego węzła jest kopiowany (nie przenoszony, jak w B-drzewie) do rodzica. Jeśli rodzic nie jest pełny, może to wymagać co najwyżej lokalnego przesunięcia kluczy w rodzicu.

Literatura

[1] Douglas Comer, The Ubiquitous B-Tree, ACM 1979

[2] Raghu Ramakrishnan, Johannes Gehrke; Database Management Systems, 2000

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Wprowadzenie do algorytmów, WNT, 2005