Notatki z AiSD. Nr 13. 16 marca 2005

Drzewa zbalansowane:

B-drzewa

IIUWr. II rok informatyki. Przygotował: Krzysztof Loryś

21 Wstęp

Przypomnienie: *Słownikiem* nazywamy strukturę danych umożliwiającą pamiętanie zbioru pewnych elementów oraz wykonywanie na nim operacji wstawiania, wyszukiwania i usuwania elementu.

Gdy chcemy używać niewielkich słowników możemy przechowywać je w pamięci wewnętrznej. Strukturami danych nadającymi się do tego celu są przykładowo zbalansowane drzewa binarne (np. drzewa AVL, drzewa czerwono-czarne) i tablice hashujące.

Dla implementacji dużych słowników, nie mieszczących się w pamięci wewnętrznej idealnie nadają się B-drzewa. Są to zbalansowane drzewa przeszukiwań specjalnie zaprojektowane tak, by operacje na nich były efektywnie wykonywane wtedy gdy są one przechowywane w plikach dyskowych.
Cechy charakterystyczne B-drzew:

- Wszystkie liście B-drzewa leżą na tej samej głębokości.
- Każdy węzeł zawiera wiele elementów zbioru (są one uporządkowane).
- Nowe elementy zapamiętywane są w liściach.
- Drzewo rośnie od liści do korzenia: jeśli jakiś wezeł jest pełny to tworzony jest jego nowy brat, który przejmuje od niego połowę elementów a jeden z jego elementów (środkowy) wędruje wraz ze wskaźnikiem na nowego brata do ojca. Jeśli w ten sposób podzielony zostanie korzeń, to tworzony jest nowy korzeń, a stary będzie jednym z dwóch jego synów. Jest to jedyny moment, w którym może wzrosnąć wysokość B-drzewa.

22 Formalny opis

Definicja. *B-drzewo o minimalnym stopniu t* posiada następujące własności:

1. Każdy węzeł x ma następujące pola:

   a. $n[x]$ - liczba kluczy aktualnie pamiętanych w x,

   b. $2t-1$ pól $key_i[x]$ na klucze (pamiętane są one w porządku niemalejącym: $key_1[x] \leq key_2[x] \cdots key_{n[x]}[x]$),

   c. $leaf[x]$ - pole logiczne = TRUE iff x jest liściem.

2. Jeśli x jest węzłem wewnętrznym to posiada ponadto $2t$ pól $c_i[x]$ - na wskaźniki do swoich dzieci.

3. Klucze pamiętane w poddrzewie o korzeniu $c_i[x]$ są nie mniejsze od kluczy pamiętanych w poddrzewie o korzeniu $c_j[x]$ (dla każdego $j < i$) i nie większe od kluczy pamiętanych w poddrzewie o korzeniu $c_k[x]$ (dla każdego $i < k$).

4. Wszystkie liście mają tę samą głębokość (oznaczamy ją h).

5. $t \geq 2$ jest ustaloną liczbą całkowitą określającą dolną i górną granicę na liczbę kluczy pamiętanych w węzłach:

   a. Każdy węzeł różny od korzenia musi pamiętać co najmniej $t - 1$ kluczy (a więc musi mieć co najmniej t dzieci). Jeśli drzewo jest niepuste, to korzeń musi pamiętać co najmniej jeden klucz.

   b. Każdy węzeł może pamiętać co najwyżej $2t - 1$ kluczy (a więc może mieć co najwyżej $2t$ dzieci). Mówimy, że węzeł jest *pełny* jeśli zawiera dokładnie $2t - 1$ kluczy.

23 Operacje na B-drzewach

Zakładamy, że B-drzewo pamiętane jest na dysku. Jego węzły sprowadzane są do pamięci wewnętrznej operacją *disc-read*. Każdorazowo w pamięci wewnętrznej znajduje się tylko niewielka liczba węzłów. Tylko te węzły mogą być modyfikowane przez program. Po każdorazowej modyfikacji węzeł zapisywany jest operacją *disc-write* na dysk. Przyjmujemy, że operacja *disc-read* nie powoduje żadnej akcji gdy wydana jest do węzła znajdującego się aktualnie w pamięci.

23.1 Przeszukiwanie

Wykonuje się w podobny sposób jak w binarnych drzewach przeszukiwań. Jedyna różnica polega na tym, źe przechodząc wierzchołki drzewa dokonujemy wyboru między wieloma synami.
W poniźszej procedurze k jest poszukiwanym kluczem a x jest adresem węzła, od którego rozpoczynamy szukanie.

```
procedure B-Tree-Search(x, k)
    i ← 1
    while i ≤ n[x] and k > key_i[x]  do i ← i + 1
    if i ≤ n[x] and k = key_i[x]  then  return (x, i)
    if leaf[x]  then  return NIL
                else  disc-read(c_i[x])
                      return B-Tree-Search(c_i[x], k)
```

W przypadku gdy $n[x]$ jest duźe zamiast liniowego przeszukiwania kluczy w wierzchołku, może opłacić się zastosowanie przeszukiwania binarnego.

23.2 Tworzenie pustego B-drzewa

procedure *B-Tree-Create*(T)
$x \leftarrow$ *Allocate-Node*$()$
$leaf[x] \leftarrow$ TRUE
$n[x] \leftarrow 0$
$Disc - Write(x)$
$root[T] \leftarrow x$

23.3 Rozdzielanie węzła w B-drzewie

Znaczenie parametrów:
y - pełny wierzchołek, tj. zawierający $2t - 1$ kluczy, który należy rozdzielić;
x - ojciec y-ka, procedura *B-Tree-Split-Child* będzie wywoływana dla x-a, który jest niepełny;
i - określa, którym synem x-a jest y.

procedure *B-Tree-Split-Child*(x, i, y)
$z \leftarrow$ *Allocate-Node*$()$
$leaf[z] \leftarrow leaf[y]$
$n[z] \leftarrow t - 1$
for $j \leftarrow 1$ **to** $t - 1$ **do** $key_j[z] \leftarrow key_{j+t}[y]$
if not $leaf[y]$ **then** **for** $j \leftarrow 1$ **to** t **do** $c_j[z] \leftarrow c_{j+t}[y]$
$n[y] \leftarrow t - 1$
for $j \leftarrow n[x] + 1$ **downto** $i + 1$ **do** $c_{j+1}[x] \leftarrow c_j[x]$
$c_{i+1}[x] \leftarrow z$
for $j \leftarrow n[x]$ **downto** i **do** $key_{j+1}[x] \leftarrow key_j[x]$
$key_i[x] \leftarrow key_t[y]$
$n[x] \leftarrow n[x] + 1$
Disc-Write(y); *Disc-Write*(z); *Disc-Write*(z)

23.4 Umieszczanie klucza w B-drzewie

Umieszczenie klucza k w drzewie dokonuje się w procedurze *B-Tree-Insert-Nonfull.* Procedura *B-Tree-Insert* sprawdza jedynie czy T nie ma pełnego korzenia i jeśli tak jest, to tworzy nowy korzeń, a stary rozdziela na dwa węzły, które stają się synami nowego korzenia.

```
procedure B-Tree-Insert(T, k)
    r ← root[T]
    if n[r] = 2t − 1
        then s ← Allocate-Node()
             root[T] ← s
             leaf[s] ← FALSE
             n[s] ← 0
             c_1[s] ← r
             B-Tree-Split-Child(s, 1, r)
             B-Tree-Insert-Nonfull(s, k)
        else B-Tree-Insert-Nonfull(r, k)
```

Procedura *B-Tree-Insert-Nonfull* przechodzi ścieżkę od korzenia do odpowiedniego liścia, rozdzielając wszystkie pełne wierzchołki, które ma przejść. Chodzi o to, by w momencie wywołania tej procedury węzeł x był niepełny.

```
procedure B-Tree-Insert-Nonfull(x, k)
    i ← n[x]
    if leaf[x] then
                while i ≥ 1 and k < key_i[x]
                    do key_{i+1}[x] ← key_i[x]
                       i ← i − 1
                key_{i+1}[x] ← k
                n[x] ← n[x] + 1
                Disc-Write(x)
        else while i ≥ 1 and k < key_i[x] do i ← i − 1
             i ← i + 1
             Disc-Read(c_i[x])
             if n[c_i[x]] = 2t − 1
                 then B-Tree-Split-Child(x, i, c_i[x])
                      if k > key_i[x] then i ← i + 1
             B-Tree-Insert-Nonfull(c_i[x], k)
```

23.5 Usuwanie klucza z B-drzewa

```
procedure B-Tree-Delete(x, k)
    (* zadanie domowe *)
```

24 Koszt operacji

Twierdzenie 1 *Jeśli $n \geq 1$, to dla każdego B-drzewa o wysokości h i stopniu minimalnym $t \geq 2$ pamiętającego n kluczy: $h \leq \log_t \frac{n+1}{2}$.*

Przykład Jeśli przyjmiemy $t = 100$, to wówczas B-drzewo zawierające do 2000000 elementów ma wysokość nie większą niż 3. Tak więc wszystkie omawiane operacje na takim

B-drzewie będą wymagały dostępu do co najwyżej trzech węzłów (a więc trzeba będzie wykonać co najwyżej sześć operacji dyskowych).

Niech n będzie liczbą węzłów w B-drzewie a $h = \Theta(\log_t n)$ - wysokością drzewa.

procedura	liczba operacji dyskowych	koszt pozostałych operacji
B-Tree-Search	$O(h)$	$O(th)$
B-Tree-Create	$O(1)$	$O(1)$
B-Tree-Split-Child	$O(1)$	$O(t)$
B-Tree-Insert	$O(h)$	$O(th)$
B-Tree-Delete	$O(h)$	$O(th)$

25 Rada praktyczna

Należy rozważnie dobierać wartość t. Trzeba pamiętać, że wraz ze wzrostem t rośnie liczba operacji wykonywanych w pamięci wewnętrznej i może zniweczyć korzyści wynikające ze zmniejszenia liczby operacji dyskowych.