B-drzewa

B-drzewa są uogólnieniem drzew poszukiwań binarnych. Węzły mają większy stopień rozgałęzienia niż 2, uporządkowanie jest podobne jak w drzewach poszukiwań binarnych. Wymagane jest też zrównoważenie drzewa.

Są to struktury danych wykorzystywane w bazach danych (indeksy baz). Typowy rozmiar węzła jest zbliżony do rozmiaru bufora odczytywanego/zapisywanego przy jednym fizycznym odczycie/zapisie na dysku.

Definicja. B-drzewo o stopniu minimalnym t:

1. (Struktura węzła.) Węzeł `x` zawiera następujące informacje:

   `x.n` - ilość kluczy zawartych w węźle `x`

   `x.key[1], x.key[2], ... , x.key[x.n]` - klucze zawarte w węźle `x`

   `x.leaf` - wartość boolowska: czy węzeł `x` jest liściem

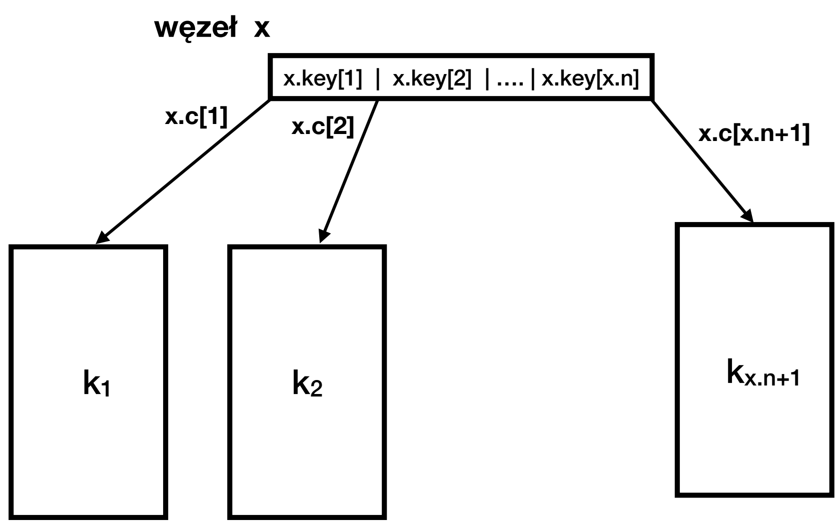
   `x.c[1], x.c[2], ... , x.c[x.n+1]` - "wskaźniki" do synów węzła `x`

2. (Porządek.) Przy oznaczeniach jak w punkcie 1, niech dodatkowo k_i będzie dowolnym kluczem w poddrzewie o korzeniu `x.c[i]`. Zachodzą nierówności:

$$\texttt{x.key}[1] \leq \texttt{x.key}[2] \leq \ldots \leq \texttt{x.key}[\texttt{x.n}]$$

oraz

$$k_1 \leq \texttt{x.key}[1] \leq k_2 \leq \texttt{x.key}[2] \leq k_3 \leq \ldots \leq k_{x.n} \leq \texttt{x.key}[\texttt{x.n}] \leq k_{x.n+1}$$



3. (Rozmiar węzłów.) Ilość kluczy w węźle różnym od korzenia jest w zakresie od $t-1$ do $2t-1$. Ilość kluczy w korzeniu jest w zakresie od 1 do $2t-1$.

4. (Zrównoważenie drzewa.) Wszystkie liście drzewa są na tym samym poziomie

Szukanie

Szukanie jest analogiczne jak w drzewach poszukiwań binarnych. Musimy jednak uwzględnić odczyty z dysku tych węzłów, które odwiedzamy.

```
Oznaczenia:
t stopień drzewa
x  węzeł B-drzewa
x.n - ilość kluczy w węźle x
x.leaf - czy x jest liściem
x.k[i] - klucze w węźle x; i=1,2...(x.n)
x.c[i] - synowie węzła x; i=1,2...(x.n+1)
----------------------------------------

BTreeSearch(x,k)
// szuka klucza k w poddrzewie o korzeniu x

  i=1
  while i<=x.n and k>x.k[i]
    i=i+1
  if j<=x.n and k==x.k{i]
    return (x,i)
  if x.leaf
    return NIL //klucza nie ma
  else
    DISK-READ(x.c[i])
    return BTreeSearch(x.c[i],k)
```

Wstawianie

Definicje.

Węzeł B-drzewa o minimalnym stopniu t nazywamy *pełnym*, jeżeli jest w nim $2t - 1$ kluczy, czyli maksymalna możliwa ilośc kluczy.

Węzeł B-drzewa o minimalnym stopniu t nazywamy *minimalnym*, jeżeli jest w nim minimalna możliwa ilośc kluczy, czyli 1 klucz w przypadku korzenia lub $t - 1$ kluczy dla węzłów różnych od korzenia.

Idea wstawiania

Nowy klucz wstawiamy do liścia drzewa zachowując uporządkowanie kluczy, a więc wędrujemy od korzenia ścieżką w dół drzewa, zgodnie z porządkiem drzewa aż dojdziemy do liścia. Po drodze rozbijamy wszystkie napotkane pełne węzły (łącznie z korzeniem). Rozbity węzeł przestaje być pełny. Pozwala to uniknąć takiej sytuacji, że liść, do którego ostatecznie dojdziemy jest pełny.

Istnieje alternatywny algorytm, w którym nie rozbijamy węzłów w czasie wędrówki w dół drzewa, ale dopiero wtedy, gdy okaże się, że liść do którego ostatecznie dotarliśmy jest pełny. Może to jednak spowodować cofanie się w górę drzewa i rozbijanie kolejnych przodków aż do korzenia włącznie.

Rozbijanie węzła

```
BtreeSplitChild(x,i,y)
// rozbija węzeł y, który jest i-tym synem węzła x
  utworz nowy wezel z    // nowy węzeł
  z.leaf = y.leaf   // z jest liściem, jeżeli y był liściem
  z.n = t-1
  for j=1 to t-1    // przepisujemy część kluczy do z
      z.k[j]=  y.k[j+t]
  if not y.leaf
      for j=1 to t   // przepisujemy odpowiednich synów do z
          z.c[j]= y.c[j+t]
  y.n = t-1
  for j = x.n+1 downto i+1     // robimy miejsce na nowego syna w x
       x.c[j+1]=  x.c[j]
  x.c[i+1] = z    // z jest tym synem
  for j=x.n downto i     // robimy miejsce na nowy klucz w x
        x.k[j+1]=  x.k[j]
  x.k[i] = y.k[t]   // środkowy klucz węzła y jest tym nowym kluczem
  x.n = x.n+1
  DISK-WRITE(y)
  DISK-WRITE(z)
  DISK-WRITE(x)
```

Wstawianie do poddrzewa, którego korzeń jest niepełny

```
B-TREE-INSERT-NONFULL(x,k)
// wstawia klucz k do poddrzewa o korzeniu x
//  węzeł x jest niepełny (x.n < 2*t-1)
  i = x.n
  if x.leaf    // wstawiamy k do x zachowując porządek
     while i>=1 and k < x.k[i]
          x.k[i+1] = x.k[i]
          i = i-1
     x.k[i+1] = k
     x.n = x.n + 1
     DISK-WRITE(x)
  else    // rekursywne zejście w dół drzewa
      while i>=1 and k < x.k[i]
          i = i-1
      i = i+1
      DISK-READ(x.c[i])
      if (x.c[i]).n = 2*t - 1
          B-TREE-SPLIT-CHILD(x,i,x.c[i])
          if k > x.k[i]
             i = i+1
       B-TREE-INSERT-NONFULL(x.c[i],k)
```

Wstawianie do B-drzewa

```
B-TREE-INSERT(T,k)
// wstawia klucz k do drzewa T
  r = T.root
  if r.n == 2*t - 1
      utworz nowy węzeł s
      s = T.root
      s.leaf = FALSE
      s.n = 0
      s.c[1] = r
      B-TREE-SPLIT-CHILD(s,1,r)
      B-TREE-INSERT-NONFULL(s,k)
  else
      B-TREE-INSERT-NONFULL(r,k)
```

Złożoność czasowa

Stwierdzenie. B-drzewo stopnia minimalnego t zawierające n kluczy ($n > 0$) ma wysokość nie większą niż $log_t \frac{n+1}{2}$.

Dowód. Analizujemy B-drzewo stopnia t o wysokości h o najmniejszej możliwie ilości kluczy.

Poziom 0 (korzeń): 1 węzeł, który ma 2 synów i zawiera 1 klucz
Poziom 1: 2 węzły, które mają w sumie $2t$ synów i zawierają $2(t-1)$ kluczy
Poziom 2: $2t$ węzłów, w sumie $2t^2$ synów, kluczy $2t(t-1)$
Poziom 3: $2t^2$ węzłów, w sumie $2t^3$ synów, kluczy $2t^2(t-1)$
...
Poziom h: $2t^{h-1}$ węzłów, kluczy $2t^{h-1}(t-1)$

Zatem sumaryczna ilość kluczy n w drzewie o wysokości h spełnia warunek

$$n \geq 1 + 2(t-1) + 2t(t-1) + 2t^2(t-1) + \ldots + 2t^{h-1}(t-1) =$$
$$= 1 + 2(t-1)(t^0 + t^1 + t^2 + \ldots + t^{h-1})$$
$$= 1 + 2(t-1)\frac{t^h - 1}{t-1}$$

czyli

$$n \geq 1 + 2(t^h - 1)$$
$$n \geq 2t^h - 1$$
$$t^h \leq \frac{n+1}{2}$$

i po zlogarytmowaniu przy podstawie t

$$h \leq log_t \frac{n+1}{2}$$

Stwierdzenie. Pesymistyczna złożoność czasowa operacji wstaw, szukaj, usuń wykonywanych na B-drzewie o minimalnym stopniu t zawierającym n kluczy jest

$\Theta(log_t n)$ jeżeli liczymy tylko operacje dyskowe

$\Theta(t log_t n)$ jeżeli liczymy wszystkie operacje

Dowód. Wstawienie, szukanie, usunięcie polega na przejściu ścieżką w drzewie od korzenia maksymalnie do liścia. Wysokość drzewa jest w najgorszym razie $\Theta(log_t n)$ a ilość wykonanych operacji dyskowych na każdym odwiedzonym poziomie drzewa jest ograniczona przez (niewielką) stałą. Ponadto, ilość wszystkich operacj na każdym poziomie drzewa jest $\Theta(t)$.