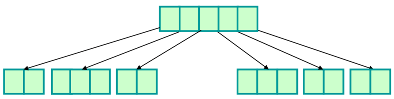
Algorytmy i struktury danych

Wykład 8-9

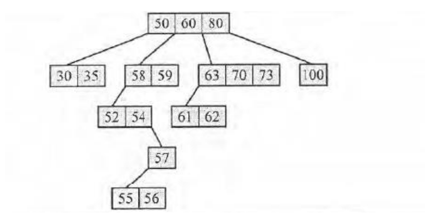
Drzewa wielokierunkowe. B-drzewa



Drzewa wielokierunkowe

- Drzewo wielokierunkowe jest drzewem, w którym każdy węzeł może mieć więcej niż 2 następniki;
- Drzewo poszukiwań rzędu ***m*** (*m*-kierunkowe drzewo poszukiwań) jest drzewem wielokierunkowym, w którym:
  - każdy węzeł ma co najwyżej ***m-1*** kluczy i ***m*** następników;
  - klucze w każdym węźle są uporządkowane rosnąco;
  - klucze pierwszych ***i*** następników są mniejsze niż *i*-ty klucz;
  - klucze ostatnich ***m-i*** następników są większe niż *i*-ty klucz;
- *m*-kierunkowe drzewo poszukiwań jest uogólnieniem drzewa BST;
- Najpopularniejszymi drzewami wielokierunkowymi są tzw. B-drzewa

Drzewa wielokierunkowe

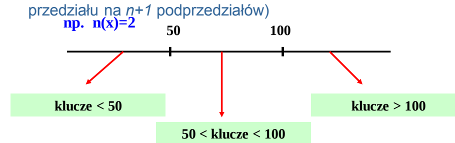


Przykład 4-kierunkowego drzewa poszukiwań

B-drzewa

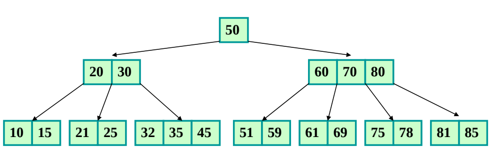
- B-drzewa są naturalnym uogólnieniem drzew BST
- B-drzewa są zrównoważonymi drzewami poszukiwań zaprojektowanymi w celu efektywnego zapisu w pamięciach zewnętrznych o dostępie bezpośrednim (np. dyskowych)
- Węzły B-drzewa mogą zawierać więcej niż jeden klucz (jeden klucz może zawierać tylko korzeń)
- Jeżeli węzeł wewnętrzny *x* w B-drzewie zawiera *n(x)* kluczy, to ma on *n(x)+1* następników (bo *n* wartości wyznacza podział przedziału na *n+1* podprzedziałów)

np. n(x)=2



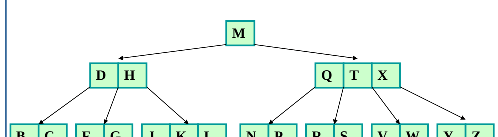
B-drzewa

- Klucze w węźle *x* są wartościami dzielącymi przedział kluczy odpowiadający węzłowi *x* na *n(x)+1* podprzedziałów odpowiadających następnikom *x;*



B-drzewa

Przykład B-drzewa



B-drzewa

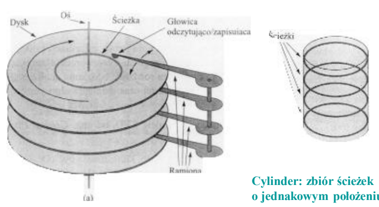
- Podczas wyszukiwania klucza w B-drzewie decyzja o kierunku dalszego wyszukiwania, podejmowana w każdym węźle *x*, zależy od wyników porównań wyszukiwanego klucza z *n(x)* wartościami pamiętanymi w tym węźle;
- Liście mają inną strukturę niż węzły wewnętrzne i znajdują się na tym samym (ostatnim) poziomie;

B-drzewa

- Współczynnik rozgałęzienia B-drzewa może być bardzo duży i wynika zazwyczaj z charakterystyki używanej jednostki dyskowej;
- Każde n-węzłowe B-drzewo ma wysokość rzędu $O(\log_t n)$, gdzie t jest tzw. *minimalnym stopniem drzewa*, $t \geq 2$;
- Ponieważ współczynnik rozgałęzienia B-drzewa może być dużo większy od *2*, wysokość B-drzewa może być znacząco mniejsza od wysokości drzewa typu BST, np. drzewa czerwono-czarnego;
- Najczęściej spotykane współczynniki w B-drzewach wahają się pomiędzy *50* a *2000*, zależnie od rozmiaru klucza w stosunku do rozmiaru bufora transmisyjnego

B-drzewa

Typowy napęd dyskowy



Cylinder: zbiór ścieżek o jednakowym położeniu w pionie

B-drzewa

- Typowy schemat obsługi żądania dostępu do rekordu *x* zapisanego na dysku:
  - Disk-Read(x);
  - operacje odwołujące się do x;
  - Disk-Write(x);
- czas dostępu do pamięci półprzewodnikowej: 100 ns
- prędkość obrotowa dysku: 5400 – 15000 obr/min, najczęściej 7200 obr/min czyli jeden obrót zajmuje 8,33 ms (czyli ok. 83 000 razy dłużej niż czas dostępu do PAO!)

MS Excel

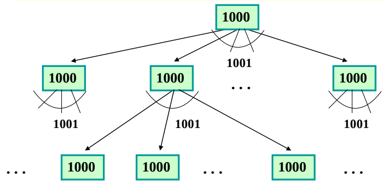
- uwzględniając czas potrzebny na przemieszczenia ramienia głowicy średni czas dostępu do dysku jest rzędu 10-50 ms

czas dostępu = czas wyszukiwania + opóźnienie rotacyjne + czas transmisji

- np. czas dostępu = 20 ms + 10 ms + 5 ms = 35 ms

B-drzewa

Przykład B-drzewa o wysokości h=3, zawierającego ponad miliard kluczy (*minimalny stopień drzewa t = 1001*)



1 węzeł: 1000 kluczy

1001 węzłów:
1 001 000 kluczy

1 001 001 węzłów:
1 001 001 000 kluczy

B-drzewa

- Najprostsze B-drzewo ma minimalny stopień *t=2*;
- Wówczas:
  - Każdy węzeł wewnętrzny ma 1, 2 lub 3 klucze
  - Każdy węzeł wewnętrzny ma 2, 3 lub 4 następniki; takie drzewo jest nazywane 2-3-4 drzewem;
- W praktyce używa się zazwyczaj dużo większych wartości *t*.

Własności B-drzewa

B-drzewo *T* jest drzewem (z korzeniem *root[T]*) o następujących własnościach 1-5:

1. Każdy węzeł x ma następujące atrybuty (pola):
   - *n[x]* – liczba kluczy aktualnie zapisanych w węźle *x*;
   - *wektor n[x]* kluczy pamiętanych w porządku rosnącym
     $key_1[x] < key_2[x] < \ldots < key_{n[x]}[x]$
   - *leaf[x]* – pole logiczne, którego wartością jest *TRUE*, jeśli *x* jest liściem lub *FALSE*, jeśli *x* jest węzłem wewnętrznym;
2. Każdy węzeł wewnętrzny *x* zawiera także *n[x] + 1* wskaźników do swoich następników $c_1[x], c_2[x], \ldots, c_{n[x]+1}[x]$,

B-drzewa

3. Klucze $key_1[x], key_2[x], \ldots, key_{n[x]}[x]$ rozdzielają wartości kluczy pamiętane w poddrzewach: jeśli k_i jest dowolnym kluczem z poddrzewa o korzeniu $c_i[x]$, to:
   $$k_1 < key_1[x] < k_2 \le key_2[x] < \ldots < key_{n[x]}[x] < k_{n[x]+1}$$
4. Wszystkie liście leżą na tej samej głębokości, równej wysokości drzewa.

B-drzewa

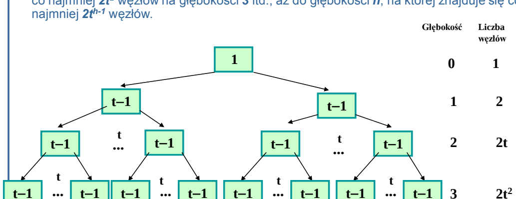
5. Minimalna i maksymalna liczba kluczy przechowywanych w danym węźle. Zależą od ustalonej liczby całkowitej $t \ge 2$, nazywanej *minimalnym stopniem B-drzewa:*
   - każdy węzeł różny od korzenia musi mieć co najmniej ***t-1*** kluczy; każdy węzeł wewnętrzny różny od korzenia musi mieć co najmniej ***t*** następników;
   - jeśli drzewo jest niepuste korzeń musi mieć co najmniej jeden klucz.
   - każdy węzeł może zawierać co najwyżej ***2t -1*** kluczy; każdy węzeł wewnętrzny może mieć co najwyżej ***2t*** następników;
   - węzeł jest pełny, jeśli zawiera dokładnie ***2t -1*** kluczy.

Wysokość B-drzewa

- ***Twierdzenie:***
  Jeśli $n \ge 1$, to dla każdego B-drzewa o ***n*** kluczach, wysokości ***h*** i minimalnym stopniu $t \ge 2$ zachodzi:
  $$h \le \log_t \frac{n+1}{2}$$
- *Dowód: kolejne 2 slajdy*

Minimalna liczba kluczy w B-drzewie o wysokości *h*

- Jeśli B-drzewo ma wysokość ***h***, to jego korzeń ma co najmniej jeden klucz a wszystkie pozostałe węzły zawierają co najmniej po ***t–1*** kluczy;
- Mamy zatem co najmniej ***2*** węzły na głębokości ***1***, co najmniej ***2t*** węzłów na głębokości ***2***, co najmniej $2t^2$ węzłów na głębokości ***3*** itd., aż do głębokości ***h***, na której znajduje się co najmniej $2t^{h-1}$ węzłów.



liczba kluczy: $n \ge 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1}$

Dlaczego?

Minimalna liczba kluczy w B-drzewie o wysokości h

- Liczba kluczy *n* spełnia zatem nierówność:

$$S_k = a_1 \frac{1-q^k}{1-q}, \quad q \ne 1$$

$$n \ge 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} =$$

$$= 1 + 2(t-1)\frac{t^h - 1}{t-1} = 2t^h - 1$$

czyli

$$n \ge 2t^h - 1$$

- Zatem

$$t^h \le \frac{n+1}{2}$$

skąd

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

B-drzewa

- **Przykład**
  t=1001
  Liczba kluczy n = 1 001 001 000

$$h \le \log_t \frac{n+1}{2} = \log_{1001} \frac{1001001000+1}{2} = 3$$

- **Jaką minimalną wysokość miałoby drzewo binarne o liczbie kluczy n = 1 001 001 000 ?**

Ms Excel

B-drzewa

Podstawowe operacje na B-drzewach:

- Tworzenie nowego (pustego) B-drzewa (*B-Tree-Create*)
- Wyszukiwanie klucza w B-drzewie (*B-Tree-Search*)
- Wstawianie klucza do B-drzewa (*B-Tree-Insert*)
- Usuwanie klucza z B-drzewa (*B-Tree-Delete*)

- Założenia:
  - korzeń B-drzewa zawsze znajduje się w PAO;
  - ilekroć korzeń się zmienia, jest zapisywany na dysku;
  - każdy węzeł przekazywany jako parametr musi być wcześniej wczytany z dysku;

B-drzewa

Tworzenie nowego (pustego) B-drzewa (*BTree-Create*)

```
B-TREE-CREATE(T)
x ← ALLOCATE-NODE()
leaf[x] ← TRUE
n[x] ← 0
DISK-WRITE(x)
root[T] ← x
```

- Żeby zbudować nowe B-drzewo najpierw za pomocą procedury *B-Tree-Create* tworzymy pusty korzeń;
- Następnie dodajemy nowe klucze za pomocą funkcji *B-Tree-Insert*
- Wykorzystujemy przy tym pomocniczą funkcję *Allocate-Node,* która w czasie *O(1)* przydziela pamięć na nowy węzeł

B-drzewa

Wyszukiwanie klucza w B-drzewie (*B-Tree-Search*)

- Operacja jest bardzo podobna do wyszukiwania w drzewie BST;
- W każdym węźle B-drzewa dokonuje się wyboru poddrzewa,w którym ma być kontynuowane wyszukiwanie;
- Wyboru dokonujemy spośród poddrzew, których liczba zależy od liczby kluczy w tym węźle (takich kluczy w węźle *x* jest *n[x],* stąd liczba wskaźników na poddrzewa wynosi *n[x] + 1*);

B-drzewa

Wyszukiwanie klucza w B-drzewie (B-Tree-Search)

```
B-TREE-SEARCH(x, k)
i ← 1
while i ≤ n[x] i k > key_i[x]
    do i ← i + 1
if i ≤ n[x] i k = key_i[x]
    then return (x, i)
if leaf[x]
    then return NIL
    else DISK-READ(c_i[x])
        return B-TREE-SEARCH(c_i[x], k)
```

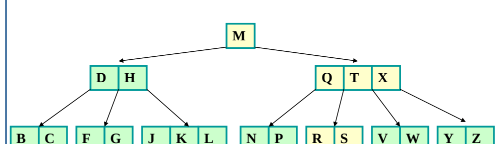
znajdowanie najmniejszego *i*, takiego, że $k \le key_i\ [x]$

sprawdzanie, czy został znaleziony poszukiwany klucz

- **Jeżeli klucz *k* jest w drzewie to ww. funkcja zwraca parę *(x, i)*, składającą się z węzła *x* oraz indeksu *i*, przy czym zachodzi $key_i\ [x]=k$**
- **W przeciwnym razie zwracana jest wartość NIL**

B-drzewa

- **Przykład wyszukiwania klucza o wartości *R***



B-drzewa

Wstawianie klucza do B-drzewa

- Wstawianie klucza *k* do B-drzewa jest znacznie bardziej skomplikowane aniżeli wstawianie klucza do drzewa BST;
- Wstawianie nowego klucza odbywa się poprzez wstawienie tego klucza do odpowiedniego liścia
- Tak jak w drzewie BST najpierw szukamy pozycji w liściu do wstawienia nowego klucza;
- W B-drzewie nie możemy jednak utworzyć nowego liścia i wstawić tam nowy klucz (dlaczego?);
- Zamiast tego wstawiamy klucz do istniejącego liścia, jeśli nie jest on pełny;
- Jeśli taki liść jest pełny wykonujemy najpierw operację polegającą na rozbiciu pełnego węzła (mającego **2t-1** kluczy) na dwa węzły po **t-1** kluczy (dlaczego?);

B-drzewa

Wstawianie klucza do B-drzewa

- Rozbicia dokonujemy względem środkowego klucza **key$_i$[y]**, który jest przesuwany do poprzednika węzła ***y***, wyznaczając punkt podziału między dwoma nowymi węzłami;
- Gdyby poprzednik węzła **y** również był pełny, także należałoby go rozbić itd. w górę drzewa (być może aż do korzenia);
- Aby uniknąć ww. „cofania się" **podczas schodzenia w dół drzewa w celu znalezienia miejsca wstawienia nowego klucza rozbijamy wszystkie spotkane pełne węzły**;
- Dzięki temu wstawianie klucza ***k*** do B-drzewa o wysokości ***h*** jest wykonywane w **jednym przebiegu** w dół drzewa, wymagającym ***O(h)*** dostępów do dysku;

B-drzewa

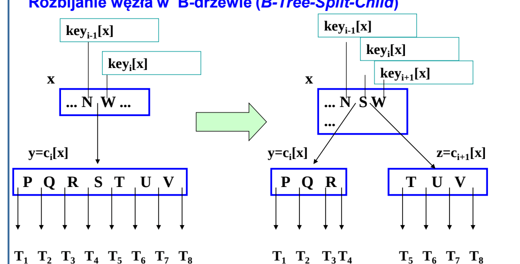
Rozbijanie pełnego węzła w B-drzewie (*B-Tree-Split-Child*)

Dane wejściowe:

- niepełny węzeł wewnętrzny **x**,
- indeks **i** (numer klucza)
- węzeł **y** (zakładamy, że węzły **x, y** znajdują się w PAO), taki że **y=c$_i$ [x]** jest pełnym następnikiem niepełnego węzła **x**)

B-drzewa

Rozbijanie węzła w B-drzewie (*B-Tree-Split-Child*)



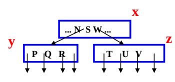
Rozbijanie węzła dla t=4. Węzeł *y* zostaje rozbity na dwa węzły: *y* oraz *z*, a środkowy klucz *S* wędruje z węzła *y* w górę do jego poprzednika

B-drzewa

Rozbijanie węzła w *B-drzewie (B-Tree-Split-Child*)

```
B-TREE-SPLIT-CHILD(x, i, y)
 z ← ALLOCATE-NODE()
 leaf[z] ← leaf[y]
 n[z] ← t − 1
 for j ← 1 to t − 1
   do key_j[z] ← key_{j+t}[y]
 if not leaf[y]
   then for j ← 1 to t
          do c_j[z] ← c_{j+t}[y]
 n[y] ← t − 1
 for j ← n[x] + 1 downto i + 1
   do c_{j+1}[x] ← c_j[x]
 c_{i+1}[x] ← z
 for j ← n[x] downto i
   do key_{j+1}[x] ← key_j[x]
 key_i[x] ← key_t[y]
 n[x] ← n[x] + 1
 DISK-WRITE(y)
 DISK-WRITE(z)
 DISK-WRITE(x)
```

tworzenie węzła ***z*,** który z węzła *y* przejmuje *t-1* największych kluczy oraz *t* odpowiednich następników



węzeł ***z*** zostaje nowym następnikiem ***x***; środkowy klucz jest przesuwany z węzła ***y*** do ***x***

zapis zmodyfikowanych węzłów do pamięci dyskowej

B-drzewa

Wstawianie klucza do B-drzewa (w jednym przebiegu)

- **funkcja *B-Tree-Insert-Nonfull* wstawia klucz *k* do drzewa o niepełnym korzeniu;**
- **funkcja *B-Tree-Insert-Nonfull* schodzi rekurencyjnie w dół drzewa, dbając o to, aby węzeł do którego następuje zejście nie był pełny;**
- **w razie potrzeby (gdy węzeł jest pełny) wywoływana jest procedura *B-Tree-Split-Child*;**

B-drzewa

Wstawianie klucza do B-drzewa (w jednym przebiegu)

- Wstawianie klucza **k** do B-drzewa T o wysokości **h** jest wykonywane w jednym przebiegu w dół drzewa, wymagającym **O(h)** dostępów do dysku;

```
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
        c1[s] ← r
        B-TREE-SPLIT-CHILD(s, 1, r)
        B-TREE-INSERT-NONFULL(s, k)
   else B-TREE-INSERT-NONFULL(r, k)
```

korzeń ***r*** jest pełny; po jego rozbiciu węzeł ***s*** (z dwoma następnikami) zostaje korzeniem

B-drzewa

Funkcja B-Tree-Insert-Nonfull

```
B-TREE-INSERT-NONFULL(x, k)
i ← n[x]
if leaf[x]
   then while i ≥ 1 i 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
        DISK-WRITE(x)
   else while i ≥ 1 i k < key_i[x]
           do i ← i − 1
        i ← i + 1
        DISK-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)
```

węzeł *x* jest liściem i klucz *k* ma się znaleźć w tym węźle

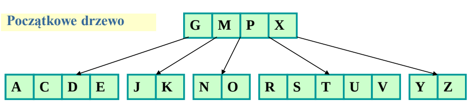
wyznaczenie następnika, do którego następuje zejście rekurencyjne

B-drzewa

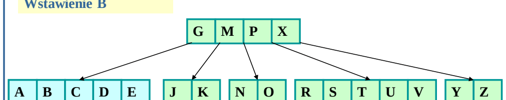
Wstawianie kluczy do B-drzewa - Przykłady

- **Minimalny stopień B-drzewa t=3 (w każdym węźle można zapamiętać co najmniej 2 klucze (z wyjątkiem korzenia) i co najwyżej 5 kluczy)**

Początkowe drzewo

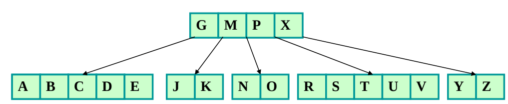


Wstawienie B

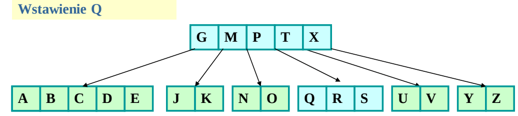


B-drzewa

Wstawianie kluczy do B-drzewa

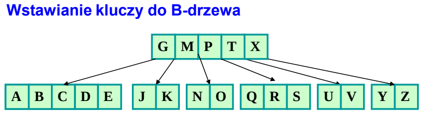


Wstawienie Q

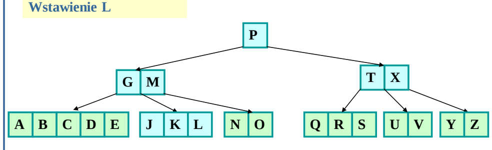


B-drzewa

Wstawianie kluczy do B-drzewa

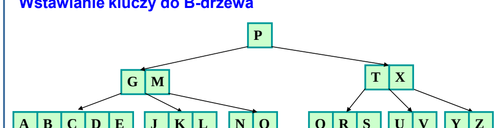


Wstawienie L

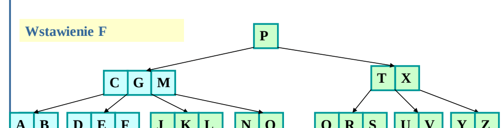


B-drzewa

Wstawianie kluczy do B-drzewa



Wstawienie F



B-drzewa

Usuwanie klucza z B-drzewa (*B-Tree-Delete*)

- usuwanie klucza z B-drzewa jest podobne do operacji wstawiania;
- klucz może być usunięty z dowolnego węzła (nie tylko z liścia), przy czym usunięcie klucza z węzła wewnętrznego wymaga manipulowania jego następnikami;
- należy uważać, aby nie utworzyć drzewa nie będącego B-drzewem: np. aby rozmiar węzła nie zmniejszył się poniżej *t-1* kluczy;
- procedura *B-Tree-Delete* jest zaprojektowana w taki sposób, aby spełniony był następujący niezmiennik: **kiedykolwiek następuje wywołanie rekurencyjne *B-Tree-Delete* dla węzła *x* różnego od korzenia, w tym węźle jest co najmniej *t* kluczy, gdzie *t* jest minimalnym stopniem B-drzewa (o 1 więcej ponad minimum);**
- pozwala to na przesunięcie w razie konieczności jednego klucza do następnika, do którego następuje zejście rekurencyjne;
- umożliwia to usunięcie klucza w jednym przebiegu;

B-drzewa

Idea algorytmu *B-Tree-Delete(x, k)* usuwania klucza z B-drzewa

1. Jeśli klucz ***k*** jest w węźle ***x***, będącym liściem o liczbie kluczy większej od ***t-1***, to usuń klucz ***k*** z węzła ***x***;
2. Jeśli klucz ***k*** jest w wewnętrznym węźle ***x***, to wykonaj jedną z poniższych akcji:
   a) Niech ***y*** będzie następnikiem węzła ***x***, o kluczach mniejszych od ***k***. Jeśli ***y*** ma co najmniej ***t*** kluczy, to w poddrzewie o korzeniu ***y*** wyznacz poprzednik ***k'*** klucza ***k***. Rekurencyjnie usuń ***k'*** i w węźle ***x*** zastąp ***k*** przez ***k'***.
   b) Jeśli następnik ***z*** węzła ***x***, o kluczach większych od ***k***, ma co najmniej ***t*** kluczy, to w poddrzewie o korzeniu ***y*** wyznacz następnik ***k'*** klucza ***k***. Rekurencyjnie usuń ***k'*** i w węźle ***x*** zastąp ***k*** przez ***k'***.
   c) Jeśli obydwa (wyznaczone przez klucz ***k***) następniki ***y*** oraz ***z*** węzła ***x*** mają tylko po ***t-1*** kluczy, to przenieś klucz ***k*** z węzła ***x*** oraz całą zawartość węzła ***z*** do węzła ***y***. Zwolnij pamięć przydzieloną dla ***z*** oraz usuń rekurencyjnie klucz ***k*** z węzła ***y***.

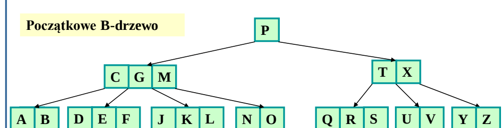
B-drzewa

Idea algorytmu usuwania klucza z B-drzewa (B-Tree-Delete)

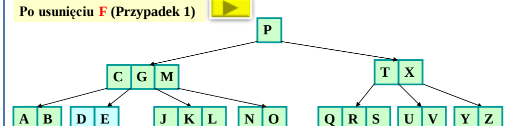
3. Jeśli klucza ***k*** nie ma w wewnętrznym węźle ***x***, to wyznacz korzeń $c_i[x]$ poddrzewa, w którym musi znajdować się klucz ***k*** (jeśli tylko jest w drzewie). Jeśli $c_i[x]$ ma tylko ***t-1*** kluczy, to wykonaj krok 3a lub 3b w celu zagwarantowania, że zejście rekurencyjne następuje do węzła zawierającego co najmniej ***t*** kluczy. Następnie usuń rekurencyjnie klucz ***k*** z właściwego poddrzewa.
   a) Jeśli w węźle $c_i[x]$ jest tylko ***t-1*** kluczy, ale jeden z jego sąsiadów (braci) ma ***t*** kluczy, to przesuń do węzła $c_i[x]$ odpowiedni klucz z węzła ***x***, a w jego miejsce przenieś do węzła ***x*** klucz z lewego lub prawego sąsiada $c_i[x]$ – tego, który zawiera ***t*** kluczy.
   b) Jeśli węzeł $c_i[x]$ i któryś z jego sąsiadów ma ***t-1*** kluczy, to połącz węzeł $c_i[x]$ z tym sąsiadem, przesuwając odpowiedni klucz rozdzielający z węzła ***x*** do nowo powstałego węzła. Przesunięty klucz jest kluczem środkowym w nowym węźle.

Usuwanie kluczy z B-drzewa (t=3) - przykłady

Początkowe B-drzewo

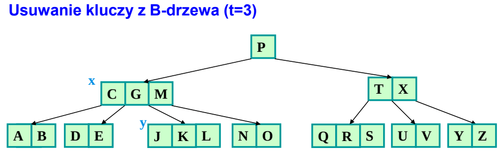


Po usunięciu F (Przypadek 1)

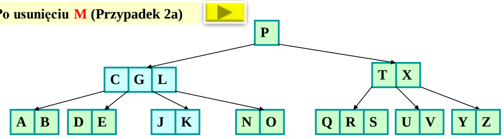


B-drzewa

Usuwanie kluczy z B-drzewa (t=3)

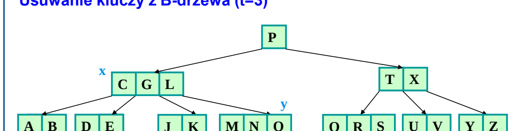


Po usunięciu M (Przypadek 2a)

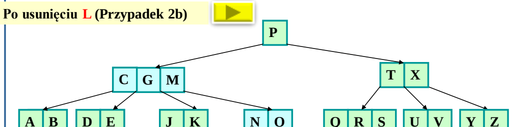


B-drzewa

Usuwanie kluczy z B-drzewa (t=3)

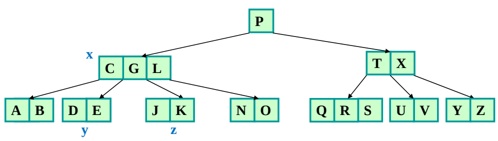


Po usunięciu L (Przypadek 2b)

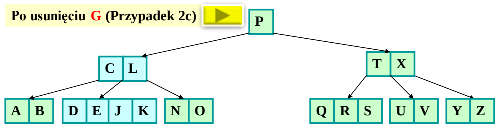


B-drzewa

Usuwanie kluczy z B-drzewa (t=3)

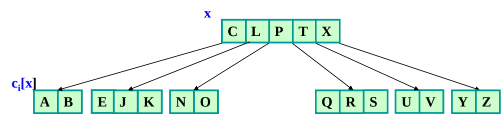


Po usunięciu G (Przypadek 2c)

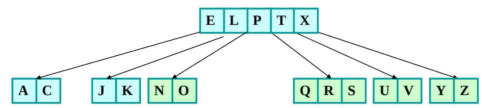


B-drzewa

Usuwanie kluczy z B-drzewa

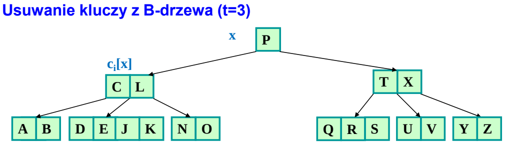


Po usunięciu B (Przypadek 3a)

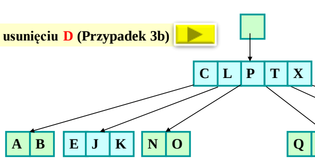


B-drzewa

Usuwanie kluczy z B-drzewa (t=3)



Po usunięciu D (Przypadek 3b)



Maleje wysokość drzewa!

Dziękuję za uwagę