B-Drzewa

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.

Każdy węzeł w B-drzewie zawiera 4 informacje:

- czy jest liściem (bool value)
- długość listy kluczy (n e(t ; 2t-1)) z wyjątkiem korzenia
- listę kluczy
- wskaniki na swoje dzieci (długość listy kluczy +1 nie pustych wskaźników)

```
procedure B-Tree-Create(T)
    x ← Allocate-Node()
    leaf[x] ← TRUE
    n[x] ← 0
    Disc − Write(x)
    root[T] ← x
```

```
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)
```

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.

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-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)
```

```
procedure 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
    Disc-Write(y);   Disc-Write(z);   Disc-Write(z)
```

3.4 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)
```

3.5 Usuwanie klucza z B-drzewa

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

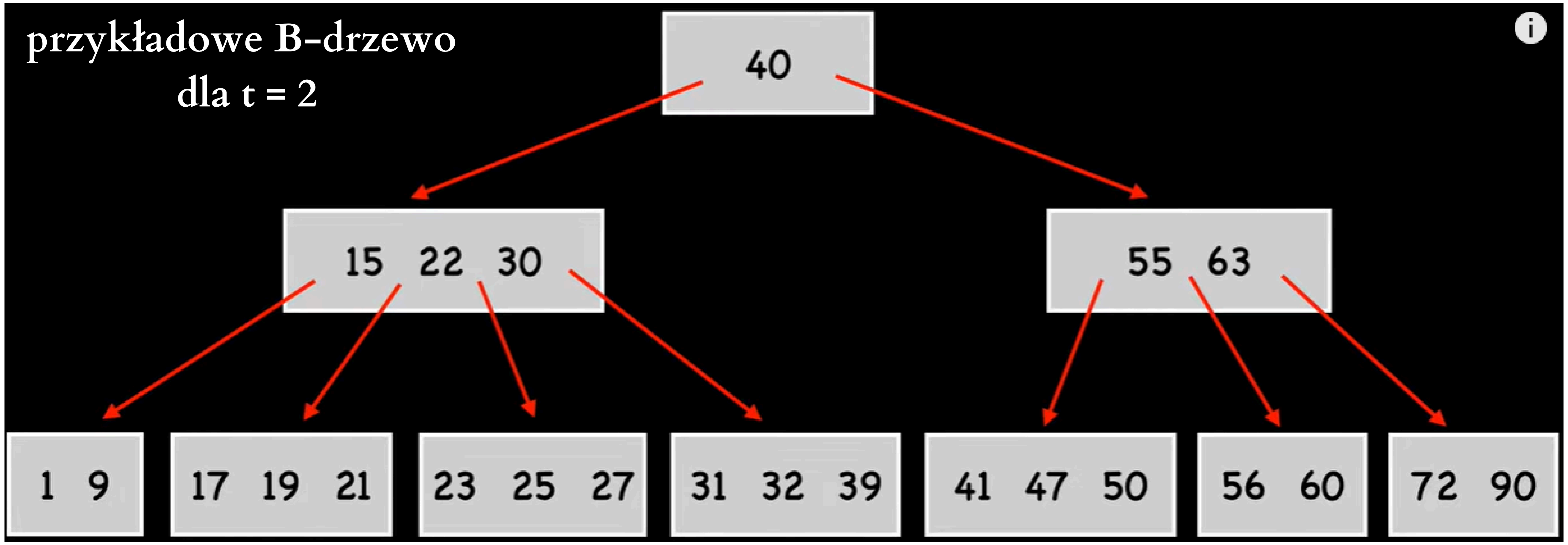
4 Koszt operacji

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.

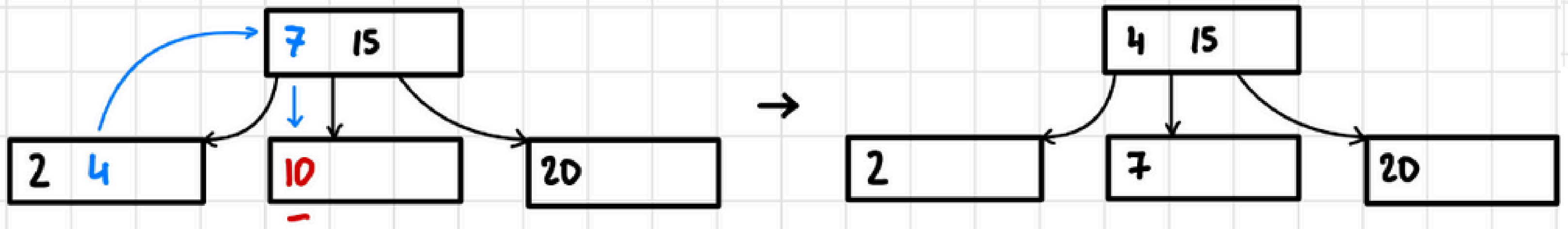
procedura	liczba operacji dyskowych	koszt pozostałych operacji
B-Tree-Search	$O(h)$	$O(th)$
B-Tree-Create	$O(1)$	$O(1)$
B-Tree-Split-Child	$O(1)$	$O(t)$
B-Tree-Insert	$O(h)$	$O(th)$
B-Tree-Delete	$O(h)$	$O(th)$



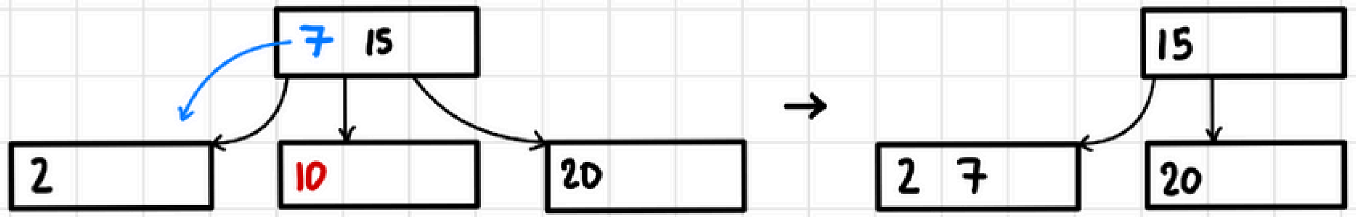
Usuwanie elementu w B-Drzewie

1° Po prostu usuwamy element

2° Przeciągamy przez tablicę z lewej lub prawej strony



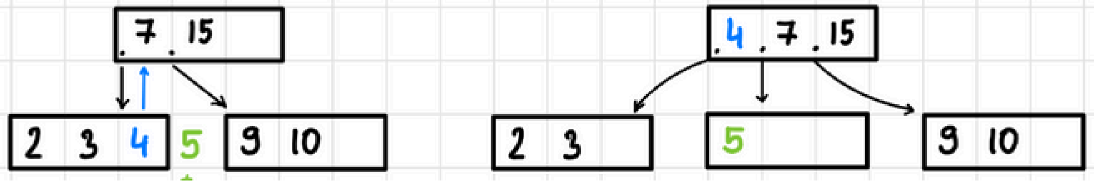
3° Usuwamy węzeł gdy sąsiedzi nie mogą pożyczyć wartości



Budowa B-Drzewa

Budowa B-Drzewa działa tak, że dodajemy elementy do liści. Jeśli nowy element w danym liściu się nie mieści, to rozdzielamy go na dwa nowe

Przykład



```python
def split_child(self, x, i):
    t = self.t

    # y is a full child of x
    y = x.children[i]

    # create a new node and add it to x's list of children
    z = Node(y.leaf)
    x.children.insert(i + 1, z)

    # insert the median of the full child y into x
    x.keys.insert(i, y.keys[t - 1])

    # split apart y's keys into y & z
    z.keys = y.keys[t: (2 * t) - 1]
    y.keys = y.keys[0: t - 1]

    # if y is not a leaf, we reassign y's children to y &
    if not y.leaf:
        z.children = y.children[t: 2 * t]
        y.children = y.children[0: t - 1]
```

Aby zamknąć tryb pełnoekranowy, naciśnij

```python
def insert_non_full(self, x, k):
    t = self.t
    i = len(x.keys) - 1

    if x.leaf:
        x.keys.append(None)
        while i >= 0 and k < x.keys[i]:
            x.keys[i + 1] = x.keys[i]
            i -= 1
        x.keys[i + 1] = k
    else:
        while i >= 0 and k < x.keys[i]:
            i -= 1
        i += 1
        if len(x.children[i].keys) == (2 * t) - 1:
            self.split_child(x, i)
            if k > x.keys[i]:
                i += 1
        self.insert_non_full(x.children[i], k)
```

```python
class BTree():
    def __init__(self, t):
        self.root = Node(True)
        self.t = t
```

```python
class Node:
    def __init__(self, leaf=False):
        self.keys = []
        self.children = []
        self.leaf = leaf
```

```python
def insert(self, k):
    t = self.t
    root = self.root

    if len(root.keys) == (2 * t) - 1:
        new_root = Node()
        self.root = new_root
        new_root.children.insert(0, root)
        self.split_child(new_root, 0)
        self.insert_non_full(new_root, k)
    else:
        self.insert_non_full(root, k)
```

```python
def search(self, key, node=None):
    node = self.root if node == None else node

    i = 0
    while i < len(node.keys) and key > node.keys[i]:
        i += 1
    if i < len(node.keys) and key == node.keys[i]:
        return (node, i)
    elif node.leaf:
        return None
    else:
        return self.search(key, node.children[i])
```

Treść

Podaj minimalną i maksymalną ilość węzłów w B-drzewie o wys.4 (krawędziowej) i minimalnym stopniu wierzchołka t=3.

▼ Rozwiązanie

Skorzystam z faktów

1. dla mnimalnego stopnia t zachodzi t - 1 <= # kluczy <= 2t - 1 dla każdego z węzłów(oprócz korzenia)
2. węzeł z t kluczami ma t+1 dzieci

Minimalna ilość węzłów

Ilość węzłów dla następujących poziomów:

1. = 1
2. = 1 * 2
3. = 1 * 2 * 3
4. = 1 * 2 * 3 * 3
5. = 1 * 2 * 3 * 3 * 3

stąd minimalna ilość węzłów = 1 + 2 + 6 + 18 + 54 = 81

Maksymalna ilość węzłów

Ilość węzłów dla następujących poziomów:

1. = 1
2. = 1 * 6
3. = 1 * 6 * 6
4. = 1 * 6 * 6 * 6
5. = 1 * 6 * 6 * 6 * 6 stąd maksymalna ilość węzłów = 1 + 6 + 36 + 216 + 1296 = 1555

Komentarz

W zadaniu zakładamy, że wysokość krawędziowa, to ilość krawędzi z korzenia do liścia

Treść

Rozważamy B-drzewa, których wierzchołki mogą pamiętać od dwóch do czterech kluczy.

Narysuj jak będzie wyglądać takie B-drzewo po wstawieniu do początkowo pustego drzewa kolejno kluczy 1,2,.. ,10.

Treść

Rozwazamy B.drzewa, których wierzchołki mogą pamiętać od dwóch do czterech kluczy.

Narusuj, jak będzie wyglądać takie B-drzewo po wstawieniu do początkowo pustego drzewa kolejno kluczy 1, 10, 3, 8, 5, 6, 7, 4, 9, 2.

Treść

Maksymalna liczba odwołań do pamięci zewnętrznej przy B-drzewie 1000/2000

▼ Rozwiązanie

log_1000(n)