

# Algorytmy i Struktury Danych, 10. ćwiczenia

2018-12-12

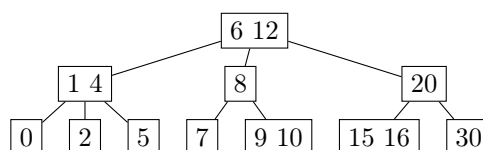
## Spis treści

1	2-3 drzewa	1
2	Pokazać, że przy pomocy rotacji można zawsze przejść z jednego BST do drugiego	5
3	ASD Zadania — wzbogacanie struktur danych	6

## 1 2-3 drzewa

2-3 drzewa to drzewa zrównoważone o następujących własnościach:

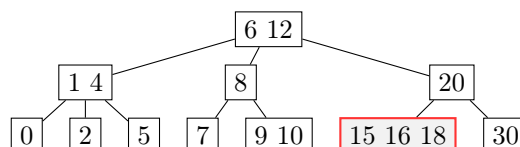
- każdy węzeł przechowuje 1 lub 2 klucze,
- każdy węzeł wewnętrzny (oprócz korzenia) ma 2 lub 3 synów,
- wszystkie liście mają tę samą głębokość,
- zachowany jest porządek kluczy w poddrzewach (mniej więcej jak w drzewach BST),



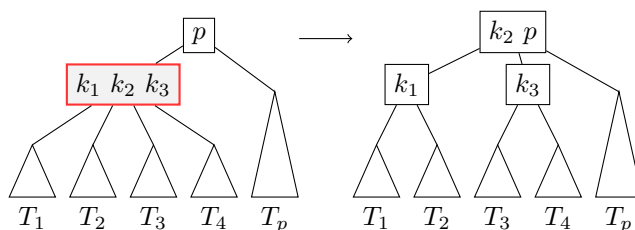
Rysunek 1: Przykładowe 2-3 drzewo

Wstawienia do drzewa:

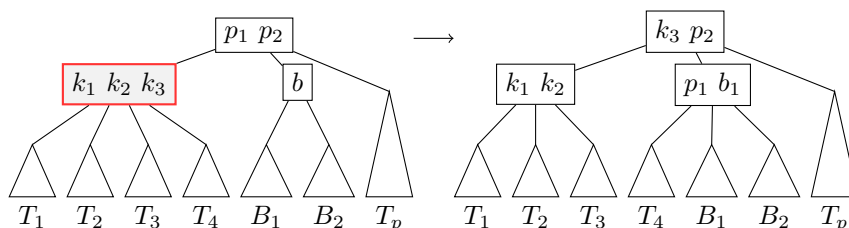
- znajdujemy liść w którym powinien znaleźć się nowy klucz, dodajemy go do węzła (może okazać się, że po tej operacji węzeł posiada 3 klucze). Przykładowo po dodaniu klucza 18 do drzewa z rysunku otrzymamy:



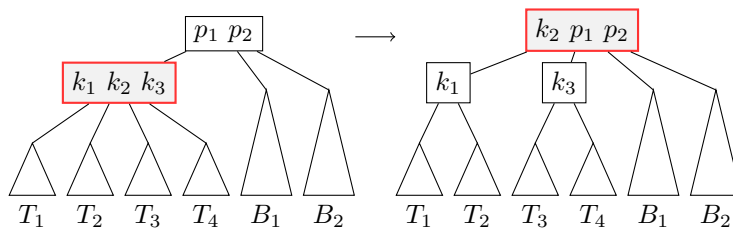
- jeśli przez wstawienie otrzymaliśmy węzeł z 3 kluczami to musimy naprawić drzewo. W trakcie naprawy, drzewo może chwilowo zawierać jeden węzeł z 3 kluczami (oraz 4 synami)
- Naprawianie polega na próbie przeniesienia nadmiarowego klucza do sąsiadujących węzłów
  - jeśli ojciec nadmiarowego węzła ma tylko jeden klucz, to możemy rozbić nadmiarowy węzeł na dwa (zawierające 1 klucz) a 3 (środkowy co do wartości) klucz przenieść do ojca



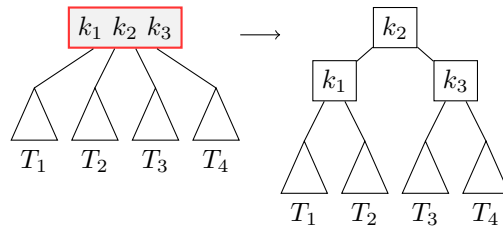
- jeśli ojciec nadmiarowego węzła ma dwa klucze, ale któryś z braci ma tylko jeden klucz, to możemy przenieść jeden klucz do ojca, a stąd jeden klucz do brata (teoretycznie można sobie poradzić bez tego przypadku, ale dzięki niemu można potencjalnie szybciej zakończyć poprawianie drzewa)



- jeśli obaj bracia mają po dwa klucze, to rozdzielamy nadmiarowy węzeł i przenosimy jeden klucz do ojca (który stanie się nadmiarowy)

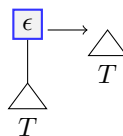
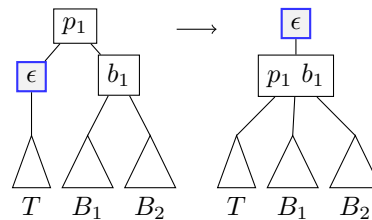
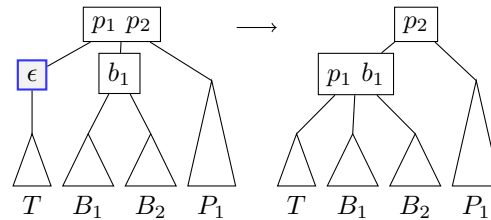
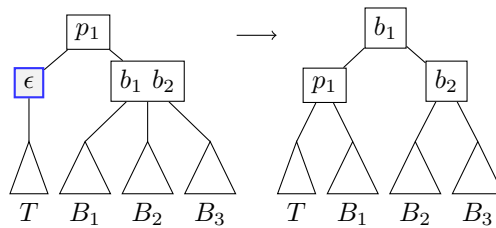


- jeśli korzeń jest nadmiarowy, to rozdzielamy go na trzy węzły (zawierające po jednym kluczu), z których środkowy klucz staje się nowym korzeniem



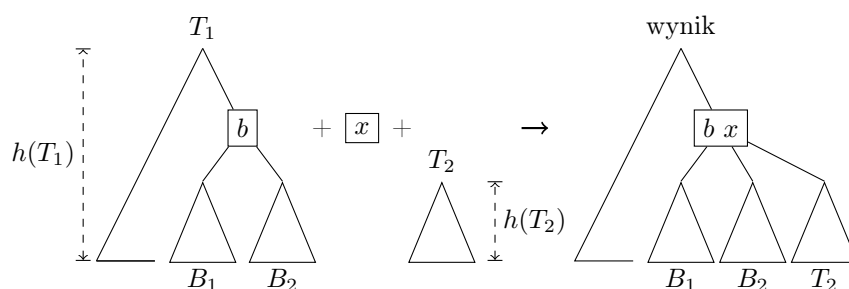
Usuwanie z drzewa:

- jeśli klucz leży w węźle wewnętrznym, to zamieniamy go z następnikiem (lub poprzednikiem), i redukujemy problem do usuwania z liścia,
- jeśli klucz leży w liściu, to usuwamy go z węzła,
- jeśli okazuje się, że jakiś węzeł zawiera 0 kluczy, to poprawiamy drzewo idąc od tego węzła do korzenia i próbując zastąpić puste miejsce kluczem pożyczonym od sąsiadów. W trakcie poprawiania drzewo może zawierać co najwyżej jeden węzeł zawierający 0 kluczy i jednego syna.
- możliwe przypadki:



Scalanie  $\text{JOIN}(T_1, x, T_2)$ :

- jest to operacja, która jako argumenty dostaje 2-3 drzewa  $T_1$  i  $T_2$  oraz klucz  $x$ , (przy czym  $\max\{y \in T_1\} < x < \min\{y \in T_2\}$ ), operacja zwraca 2-3 drzewo zawierające wartości  $T_1 \cup \{x\} \cup T_2$ .
- jeśli  $h(T_1) = h(T_2)$ , to tworzymy drzewo o korzeniu z kluczem  $x$ , oraz lewym poddrzewem  $T_1$ , prawym poddrzewem  $T_2$ ,
- jeśli  $h(T_1) > h(T_2)$  (symetryczny przypadek  $h(T_1) < h(T_2)$ ): na skrajnie prawej ścieżce  $T_1$ , odnajdujemy węzeł  $b$ , leżący w odległości  $h(T_1) - h(T_2) - 1$  od korzenia (dzięki temu synowie  $b$  mają wysokości  $h(T_2)$ ),
- do węzła  $b$  dodajemy klucz  $x$  oraz jako skrajnie prawe poddrzewo  $T_2$ ,



- może to spowodować powstanie węzła o 3 kluczach i w takim wypadku poprawiamy drzewo analogicznie do wstawiania.

Złożoność czasowa:  $O(|h(T_1) - h(T_2)| + 1)$ .

Podział drzewa  $\text{SPLIT}(x, T)$ :

- operacja usuwa klucz  $x$  z drzewa i tworzy dwa 2-3 drzewa,  $T_1 = \{y \in T : y < x\}$  oraz  $T_2 = \{y \in T : y > x\}$
- szukamy ścieżki od korzenia do węzła zawierającego  $x$  (bez straty ogólności możemy założyć, że  $x \in T$ , w przeciwnym przypadku możemy go sztucznie dodać), węzły na tej ścieżce oznaczamy przez  $V = v_0 = \text{root}(T), v_1, \dots, v_k$ ,
- gdybyśmy usunęli ścieżkę  $V$  z drzewa  $T$ , to rozpadnie się ono na  $O(\log |T|)$  2-3 drzew, każde z nich będzie w całości albo zawierało klucze większe od  $x$  lub mniejsze od  $x$
- niech  $L$  to maksymalna kolekcja poddrzew  $T$  spoza ścieżki  $V$ , zawierająca klucze mniejsze od  $x$ :

$$L = \{T_v : v \in T \text{ oraz } v \notin V \text{ oraz } \text{parent}(v) \in V \text{ oraz } \max(T_v) < x\}$$

- analogicznie definiujemy  $R$  (zawierające poddrzewa o wartościach większych od  $x$ )

$$R = \{T_v : v \in T \text{ oraz } v \notin V \text{ oraz } \text{parent}(v) \in V \text{ oraz } \min(T_v) > x\}$$

- analogicznie dzielimy klucze ze ścieżki  $V$ :

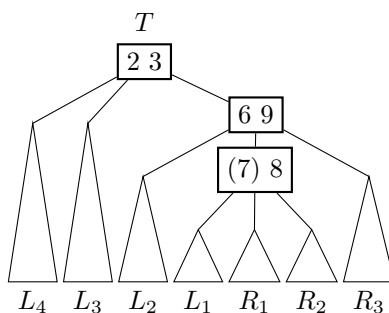
$$X_L = \{y : keys(V) \text{ oraz } y < x\}$$

$$X_R = \{y : keys(V) \text{ oraz } y > x\}$$

- żeby otrzymać drzewo  $T_1$  scalamy poddrzewa z  $L$  (wg rosnących wysokości) używając kluczy z  $X_L$  jako elementów separujących,
- analogicznie, żeby otrzymać drzewo  $T_2$  scalamy poddrzewa z  $R$  (wg rosnących wysokości) używając kluczy z  $X_R$  jako elementów separujących.

Złożoność czasowa:  $O(|h(T)|) = O(\log n)$ . Dzięki temu, że scalamy drzewa wg. rosnących wysokości i obserwacji, że scalanie wymaga czasu  $O(|h_1| - |h_2|)$  suma składa się teleskopowo.

Przykład  $SPLIT(7, T)$ :



- $V = (\{2\ 3\}, \{6\ 9\}, \{7\ 8\})$ ,
- $L = (L_4, L_3, L_2, L_1)$ ,
- $R = (R_1, R_2, R_3)$ ,
- $X_L = \{2, 3, 6\}$ ,  $X_R = \{8, 9\}$ ,
- $T_1 = JOIN(L_4, 2, JOIN(L_3, 3, JOIN(L_2, 6, L_1)))$
- $T_2 = JOIN(JOIN(R_1, 8, R_2), 9, R_3)$

## 2 Pokazać, że przy pomocy rotacji można zawsze przejść z jednego BST do drugiego

Za pomocą rotacji możemy dowolne drzewo BST zamienić na listę (i na odwrót). Wstarczy tak długo jak drzewo zawiera węzeł z lewym synem, wykonać na nim (i lewym synie) prawą rotację.

### 3 ASD Zadania — wzbogacanie struktur danych

#### Zadanie 3.24

Zaprojektuj strukturę danych umożliwiającą wykonywanie w czasie  $O(\log n)$  następujących operacji na zbiorze  $S$ :

- $make\ set(S) :: S := \emptyset$
- $insert((x, y), S) :: S := S \cup \{(x, y)\}$
- $minx(S) ::$  usunięcie z  $S$  pary  $(x, y)$  o najmniejszej pierwszej składowej,
- $miny(S) ::$  usunięcie z  $S$  pary  $(x, y)$  o najmniejszej drugiej składowej,
- $searchx(x, S) ::$  wyznaczenie takiej pary  $(a, b) \in S$ , że  $x = a$ ,
- $searchy(y, S) ::$  wyznaczenie takiej pary  $(a, b) \in S$ , że  $y = b$ .

**Rozwiązanie:** dwa drzewa AVL (jedno ze współrzędnymi  $x$ , drugie ze współrzędnymi  $y$ ), dodatkowo każdy węzeł trzyma dowiązanie do odpowiadającego mu węzła w drugim drzewie.

#### Zadanie 3.25

Zaprojektuj strukturę danych umożliwiającą wykonywanie w czasie  $O(\log n)$  następujących operacji na zbiorze  $S$ :

- $construct(S) ::$  utworzenie ciągu pustego  $S$ ,
- $insert(S, x) :: S := S \cup \{x\}$ ,
- $delete(S, x) :: S := S - \{x\}$ ,
- $search(S, x) ::$  sprawdzenie, czy  $x$  znajduje się w zbiorze  $S$ ,
- $elem(S, i) ::$  wyznaczenie  $i$ -tego co do wielkości elementu zbioru  $S$ ,
- $numb(S, x) ::$  wyznaczenie numeru elementu  $x$  w zbiorze  $S$  (względem wielkości).

**Rozwiązanie:** drzewo AVL z atrybutami rozmiar poddrzewa.

#### Zadanie 3.26

Zaprojektuj strukturę danych do wykonywania ciągów następujących operacji (dla elementów  $x$  pochodzących z dowolnego zbioru liniowo uporządkowanego):

- $initialization :: S_i = \emptyset$  dla  $i = 1, 2, \dots, n$ ,
- $insert(S_i, x) :: S_i := S_i \cup \{x\}$ , pod warunkiem, że  $x$  nie występuje w żadnym zbiorze  $S_j$ ,  $1 \leq j \leq n$ ,
- $deletemin(S_i) ::$  usunięcie ze zbioru  $S_i$  najmniejszego elementu,
- $find(x) ::$  wyznaczenie numeru zbioru do którego należy element  $x$ .

**Rozwiązanie:**  $S_i$  jako zwykle kopce, dodatków utrzymujemy słownik par  $(x, numerzbioru)$

### Zadanie 3.27

Zaprojektuj strukturę danych umożliwiającą wykonywanie w czasie  $O(\log n)$  następujących operacji na ciągu  $S$ :

- $construct(S) ::$  utworzenie ciągu pustego  $S$ ,
- $insert(S, i, x) ::$  wstawienie  $x$  na  $i$ -te miejsce w ciągu  $S$ , tzn.  $S_i = x$  pod warunkiem, że  $i \leq |S| + 1$ ,
- $sum(S, i, j) ::$  obliczenie sumy  $\sum_{k=i}^j S_k$ ,

**Rozwiązanie:** AVL z dodatkowym atrybutem suma elementów poddrzewa.

### Zadanie 3.28

**Rozwiązanie:** AVL z atrybutem rozmiar poddrzewa.

### Zadanie 3.29

Zaprojektuj strukturę danych umożliwiającą wykonywanie w czasie  $O(\log n)$  następujących operacji na zbiorze  $S$  zawierającym przedziały liczb rzeczywistych  $[l, r]$ :

- $empty(S) :: S = \emptyset$ ,
- $add(S, I) :: S = S \cup \{I\}$ ,
- $delete(S, I) :: S = S - \{I\}$ ,
- $is(S, x) ::$  sprawdzenie czy element  $x$  należy do jakiegoś przedziału w zbioru  $S$ ;
- $intersect(S, I) ::$  sprawdzenie czy przedział  $I$  ma niepuste przecięcie z jakimś przedziałem należącym do  $S$ .

**Rozwiązanie:** Utrzymujemy słownik z parami  $(x, z)$  (gdzie  $x$  to liczba rzeczywista, a  $z + 1$  lub  $-1$ ). Dodatkowo każdy węzeł ma dodatkowy atrybut  $suma$  oznaczający sumę wartości  $z$  w poddrzewie. Możemy w takim drzewie w czasie  $O(\log n)$  obliczyć  $sum(q)$  oznaczającą sumę wszystkich atrybutów  $z$  par  $(x, z)$ , takich, że  $x \leq q$ .

- $add(S, I)$  – dodajemy do słownika pary  $(l, +1)$  i  $(r, -1)$ ,
- $delete(S, I)$  – usuwamy ze słownika pary  $(l, +1)$  i  $(r, -1)$ ,
- $is(S, x)$  – jeśli słownik zawiera pary  $(x, +1)$  lub  $(x, -1)$  to zwracamy  $true$ , wpp. obliczamy  $sum(x)$  i jeśli suma jest  $> 0$  to zwracamy  $true$ , jeśli  $sum(x) \leq 0$ , to zwracamy  $false$ .
- $intersect(S, I)$ , jeśli  $is(S, l)$  lub  $is(S, r)$  to zwracamy  $true$ , jeśli istnieje w słowniku para  $(x, z)$ , t.ż.  $l \leq x \leq r$ , to zwracamy  $true$ , wpp zwracamy  $false$ .