Drzewa BST

Algorytmy i Struktury Danych

Drzewa – Definicje

Termin	Definicja
drzewo binarne	Drzewo, w którym każdy węzeł ma nie więcej niż dwoje dzieci
cykl	Jeżeli istnieje ścieżka umożliwiająca powrót do węzła wyjściowego, nazywamy ją cyklem
drzewo wyszukiwań binarnych	Drzewo binarne spełniające następującą własność - dla dowolnego węzła n, wszystkie węzły lewego poddrzewa n mają wartości mniejsze niż wartość n, a wszystkie węzły prawego poddrzewa n mają wartości większe niż wartość n
węzeł wewnętrzny	Węzeł, który ma co najmniej jedno dziecko
liść (węzeł zewnętrzny)	Węzeł, który nie ma dzieci
węzeł	Węzły są składnikami drzew. Węzeł zawiera pewną ilość danych oraz referencje do swojego rodzica oraz swoich dzieci (korzeń drzewa nie posiada rodzica, liście nie posiadają dzieci)
korzeń	Węzeł, który nie posiada rodzica. Każe drzewo posiada dokładnie jeden korzeń
poddrzewo	Poddrzewo o korzeniu w węźle n to część większego drzewa. Na to poddrzewo składa się węzeł n oraz wszyscy jego potomkowie

Drzewa

BST	$L\leq, R\geq$
Kopce	$L\geq, R\geq$

2

Drzewa

Złożoność obliczeniowa - zestawienie

Poniższa tabela zawiera listę operacji wykonywanych na drzewach BST wraz z charakteryzującą je złożonością obliczeniową

Operacja	Złożoność obliczeniowa w optymistycznym przypadku	Złożoność obliczeniowa w pesymistycznym przypadku
wyszukiwanie	$\log_2 n$	*N*
wstawienie	$\log_2 n$	*N*
usunięcie	$\log_2 n$	*N*
przejście za korzeniem (preorder)	*n*	*N*
przejście przez korzeń (inorder)	*n*	*N*
przejście przed korzeniem (postorder)	*n*	*N*

3

Drzewa

Definicje:

- Poziom wierzchołka w drzewie jest równy długości drogi łączącej go z korzeniem. Korzeń drzewa jest na poziomie 0.
- Wysokość drzewa równa jest maksymalnemu poziomowi drzewa, czyli długości najdłuższej spośród ścieżek prowadzących od korzenia do poszczególnych liści drzewa

Lematy:

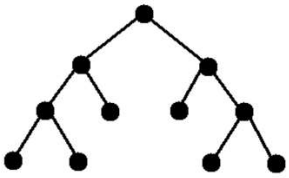
- Drzewo o wysokości *h* ma co najmniej 2^h węzłów.
- Każde drzewo o *n* węzłach ma wysokość co najwyżej log(*n*)

4

Drzewa

h(T) – wysokość drzewa

d(T) = $\sum_{x\_lisc\_w\_T} d_T(x)$ - całkowita długość ścieżek zewnętrznych



h(T) = 3

d(T) = 3+3+2+2+3+3=16

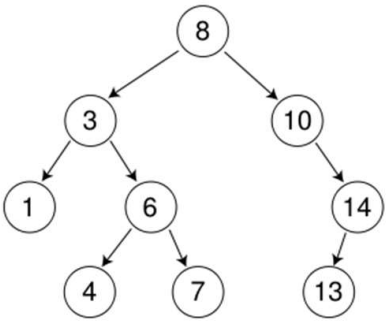
oczekiwana długość ścieżki zewnętrznej $\frac{d(T)}{3!} = \frac{16}{6} = 2{,}66$

5

Drzewa

Operacje na drzewie:

1 Dodawanie
2 Usuwanie
3 Wyszukiwanie
4 Wyliczenie wszystkich elementów



Binarne drzewo poszukiwań o wielkości równej 9, a wysokości równej 3; wierzchołek '8' jest tu korzeniem, a wierzchołki '1', '4', '7' i '13', to liście.

6

Drzewa

– Drzewo BST (binary search tree)

W każdym z węzłów drzewa BST przechowywany jest **klucz** (klucze są unikatowe).

Wartość klucza jest zawsze **większa** niż wartości wszystkich kluczy z lewego poddrzewa, a **mniejsza** niż wartości wszystkich kluczy z prawego poddrzewa (relacja może być odwrócona, to kwestia umowy);
przechodząc drzewo metodą inorder uzyskuje się ciąg wartości posortowanych rosnąco.

7

Drzewa

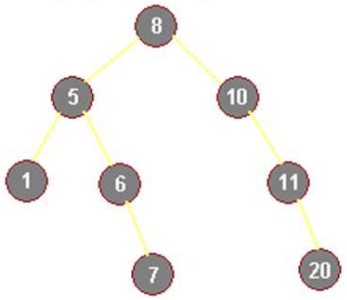
– Drzewo BST (binary search tree)

Drzewo BST jest **drzewem binarnym**. Oprócz pola wartości drzewo BST posiada jeszcze dwa pola: **L** i **P**, wskazujące odpowiednio na lewy i prawy następnik.

Drzewo BST ma szczególną własność:

- jeżeli element drzewa znajduje się **w lewej** gałęzi to jest **mniejszy** od swego poprzednika
- jeżeli element drzewa znajduje się **w prawej** gałęzi to jest **większy** od swego poprzednika

Oto przykładowe drzewo BST:



Przeszukiwanie drzewa:

- Wzdłużna - *preorder(VLR)*: korzeń, lewe poddrzewo, prawe poddrzewo.
- Poprzeczna - *inorder(LVR)*: lewe poddrzewo, korzeń, prawe poddrzewo.
- Wsteczne - *postorder(LRV)*: lewe poddrzewo, prawe poddrzewo, korzeń.

8

Drzewa

– Drzewo BST (binary search tree)

Sposoby przechodzenia drzewa binarnego

Istnieje 6 sposobów przejścia **drzewa binarnego**:
VLR, LVR, LRV, VRL, RVL, RLV,
gdzie:
Visit - "odwiedź" węzeł, **L**eft - idź w lewo, **R**ight - idź w prawo**.**

Wyróżnia się 3 pierwsze:
- VLR - **pre-order**, przejście **wzdłużne** (korzeń, lewe poddrzewo, prawe poddrzewo)**,**
- LVR - **in-order**, przejście **poprzeczne** (lewe poddrzewo, korzeń, prawe poddrzewo),
- LRV - **post-order**, przejście **wsteczne** (lewe poddrzewo, prawe poddrzewo, korzeń).

9

Drzewa

– Wagi wierzchołków drzewa binarnego

Obliczanie wag wierzchołków:
- Dla każdego wierzchołka drzewa: w(x) = h(LD) - h(PD),
- gdzie LD i PD są odpowiednio lewym i prawym poddrzewem drzewa o korzeniu w x;

10

Drzewa

– Drzewo BST (binary search tree)

Dodawanie elementów:

Jeśli drzewo BST jest puste (korzeń = nil) należy wstawić element (nie porównujemy go z innymi), w przeciwnym wypadku porównujemy wartość elementu z następnikami każdego węzła (zaczynając od korzenia). Jeżeli wartość elementu jest niewiększa od wartości porównywanego wierzchołka to przechodzimy do lewego następnika, w przeciwnym razie przechodzimy do prawego następnika.

Krok ten powtarzamy aż znajdziemy dla naszego elementu odpowiednie miejsce, tzn. gdy następnik, do którego powinniśmy iść jest pusty (nil).

Następnie wstawiamy element jako odpowiedni następnik (prawy, jeśli element jest większy od węzła, lewy jeśli niewiększy). odpowiednio na lewy i prawy następnik.

11

Drzewa

– Drzewo BST (binary search tree)

Usuwanie elementów:

Jeżeli usuwany element nie ma następników to można zwolnić przydzieloną mu pamięć.

Jeśli element do usunięcia ma jeden następnik należy go połączyć (następnik) z poprzednikiem usuwanego elementu.
Jeśli element ma dwa następniki można go usunąć na dwa sposoby:

- połączyć poprzednik elementu z wierzchołkiem o najmniejszej wartości z prawego poddrzewa usuwanego
- połączyć poprzednik elementu z wierzchołkiem o największej wartości z lewego poddrzewa

12

Drzewa

– Drzewo BST (binary search tree)

Usuwanie drzewa:

Można to zrobić na dwa sposoby: od końca, schodząc za każdym razem od korzenia lub idąc od liścia do korzenia.
Druga metoda jest znacznie szybsza, wymaga jednak zastosowania stosu, na którym umieszczamy następniki usuwanego elementu.
Po usunięciu elementu zdejmujemy ze stosu następny do usunięcia, umieszczamy na stosie jego następniki itd...

13

Drzewa

– Drzewo BST (binary search tree)

Koszt operacji na drzewie BST

Pesymistyczny koszt każdej z operacji na drzewie BST (o liczbie węzłów *n*), tj. wyszukiwania, dodania lub usunięcia klucza, zależy od wysokości drzewa i wynosi

- $\log_2 n$ - dla drzewa zrównoważonego - najlepszy przypadek;
- *n* - dla drzewa zdegenerowane do listy, tj. takiego, w którym każdy z węzłów oprócz liścia ma tylko jednego syna - najgorszy przypadek.

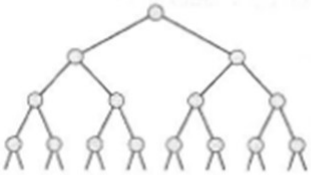
Drzewo jest **zrównoważone** wtedy i tylko wtedy, gdy dla każdego węzła wysokości dwóch jego poddrzew różnią się co najwyżej o 1.

Drzewa spełniające ten warunek są często nazywane drzewami **AVL**.

14

Drzewa

– Drzewo BST (binary search tree)



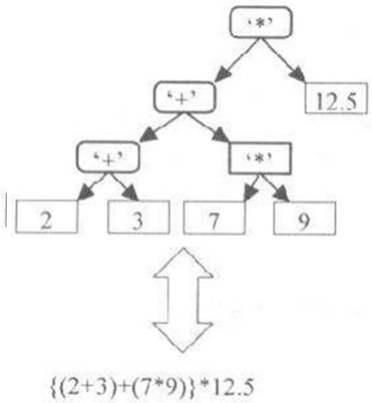
poziom	Węzłów na poziom	Węzłów na wszystkich poziomach
1	$2^0 = 1$	$1 = 2^1 - 1$
2	$2^1 = 2$	$3 = 2^2 - 1$
3	$2^2 = 4$	$7 = 2^3 - 1$
4	$2^3 = 8$	$15 = 2^4 - 1$
⋮		
11	$2^{10} = 1024$	$2047 = 2^{11} - 1$
⋮		
14	$2^{13} = 8192$	$16383 = 2^{14} - 1$
⋮		
h	$2^h - 1$	$n = 2^h - 1$
⋮		

Rys.
Maksymalna liczba węzłów w drzewach binarnych o różnych wysokościach.

15

Drzewa

– Drzewo BST (binary search tree)



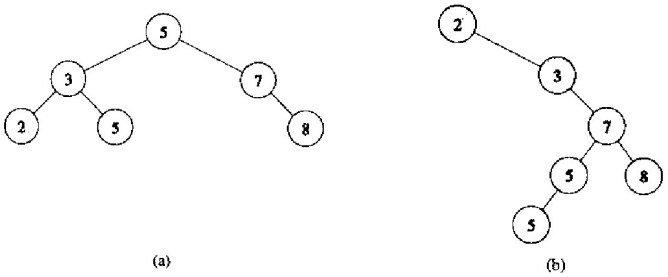
{(2+3)+(7*9)}*12.5

Rys.
Przykład zapisu wyrażenie matematycznego.

16

Drzewa

– Drzewo BST (binary search tree)



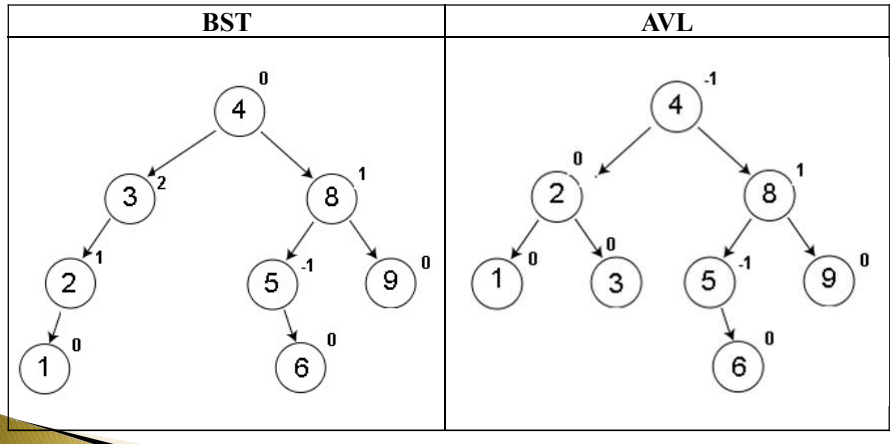
Rys. Drzewo poszukiwań binarnych. Dla każdego węzła x klucze znajdujące się w lewym poddrzewie są nie większe od key[x], a klucze znajdujące się w prawym poddrzewie są nie mniejsze od key[x]. Ten sam zbiór wartości może być przedstawiony za pomocą wielu różnych drzew BST. Pesymistyczny czas działania większości operacji na drzewach BST jest proporcjonalny do wysokości drzewa.
(a) Drzewo BST o wysokości 2 składające się z 6 węzłów. (b) Nieco mniej efektywne drzewo BST zawierające te same klucze; jego wysokość wynosi 4.

Narysuj drzewo BST o wysokości 2, 3, 4, 5 oraz 6 zawierające następujący zbiór kluczy {1, 4, 5, 10, 18, 17, 21}

17

Drzewa

– Drzewo AVL i BST



18

DZIĘKUJĘ ZA UWAGĘ