Jan Kwiatkowski
Jarosław Nachyła

Laboratorium 2 - Drzewa BST

W ramach zajęć zaimplementowano dwa drzewa binarne, każde w oddzielnym pliku. Implementacją drzewa ***avltree*** zajał się ***Jarosław Nachyła.***

Główne moduły w aplikacji:

- **BST.py** zawiera implementację Binarnego Drzewa Wyszukiwań. stworzył ***Jan Kwiatkowski.***

  Plik zawiera klasy Node i BST. W klasie BST zaimplementowano wszystkie metody potrzebne do wykonania pomiarów: add(value), search(value) i remove(value).

- plik commons.py zawiera wspólne funkcje do rysowania drzew w konsolo
- avlTree.py zawiera także klasę Node i klasę AVL z analogicznymi metodami. Dodatkowo w do klasy AVL stworzono testy jednostkowe testujące wszystkie operacje na drzewie algorytmem AVL łącznie z rotacjami .LewoLewo, Prawo Prawo, Lewo Prawo i Prawo Lewo. Każdy test jest wizualizowany w konsoli.

- ***main.py*** został stworzony wspólnie.

  Służy on do zliczania pomiaru czasu budowania i usuwania elementów drzew BST oraz AVL o zadanych wielkościach.

  Wartości dodawane do drzew zostały wylosowane z przedziału od 1 do 30000 (***CreateListOf10kRandomNumbers***()) i znajdują się w liście o nazwie ***random_numbers.***

  Stworzono listę przechowującą 10 obiektów każdej z klas (BST oraz AVL, funkcją ***StworzListeObiektowDrzewa()).***

  Każde z drzew stworzono na podstawie listy wejściowej o odpowiedniej ilości argumentów (1k,2k,...,10k) funkcją ***StworzDrzewaXYZ()*** i zmierzono czas wykonywania operacji wstawiania. Analogicznie postąpiono z usuwaniem ***ZlikwidujDrzewaXYZ().***

Algorytm drzewa BST skład się z dwóch klas. Jednej “DrzewoWezel” której obiekty przechowują informacje na temat każdego z węzłów oraz „BST”, która tworzy drzewa składające się z obiektów klasy „Drzewo Wezel”.
Metody tej klasy służące do operacji na drzewie to:

- add(wartość) – do drzewa dodawana jest nowa wartość w odpowiednie miejsce.
- wyświetl() – wyświetlanie LVR polega na wyświetleniu wartości znajdującej się maksymalnie po lewej stronie drzewa następnie w wierzchołku danego poddrzewa a potem po prawej stronie zaczynając od korzenia. Dzięki temu otrzymujemy wyświetlone wartości od najmniejszej do największej.
- search(wartość) – zwraca adres węzła w której znajduje się wartość przekazana w argumencie funkcji.
- remove(wartość) – usuwa wartość przekazana w argumencie funkcji.
- Z internetu pobrano i przerobiono także metodę rysującą drzewo w konsoli jednak ze względu na czytelność sprawdza się ona dla małej ilości danych.

Wyniki i wnioski.

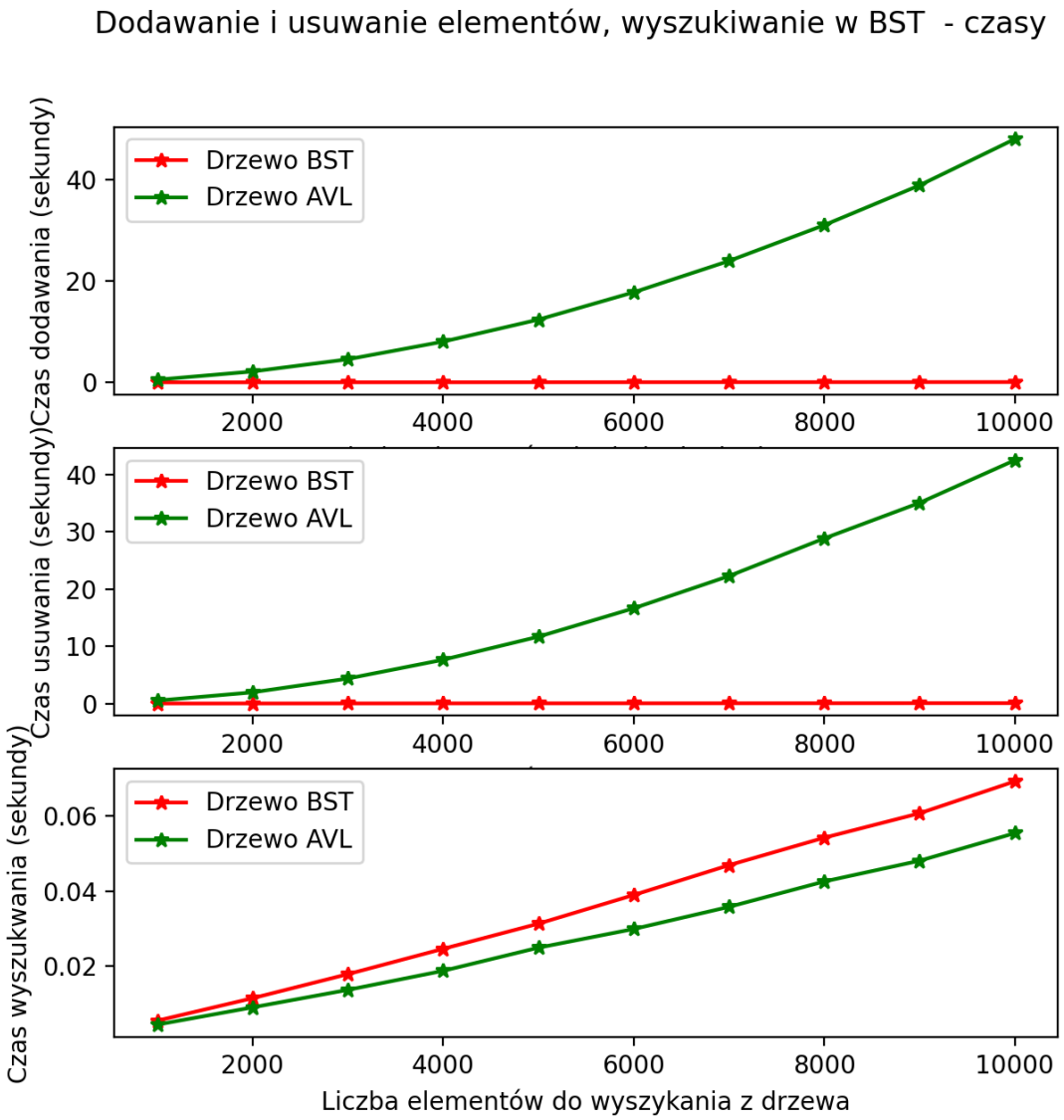
Poniżej przedstawiono tabelę złożoności BST:

	Średnia	**Pesymistyczna**
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

Analogicznie dla AVL:

	Średnia	**Pesymistyczna**
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$

Porównanie czasów budowania drzew oraz następnie usuwania z nich kolejno wszystkich węzłów przedstawiono na wykresie poniżej. Patrząc na wykresy spodziewaliśmy się znacznie większej przewagi algorytmu AVL szczególnie podczas wyszukiwania, ostatni wykres na dole.


Dodawanie i usuwanie elementów, wyszukiwanie w BST - czasy
Czas dodawania (sekundy)
Drzewo BST
Drzewo AVL
0
20
40
2000
4000
6000
8000
10000
Czas usuwania (sekundy)
Drzewo BST
Drzewo AVL
0
10
20
30
40
2000
4000
6000
8000
10000
Czas wyszukwania (sekundy)
Drzewo BST
Drzewo AVL
0.02
0.04
0.06
2000
4000
6000
8000
10000
Liczba elementów do wyszykania z drzewa