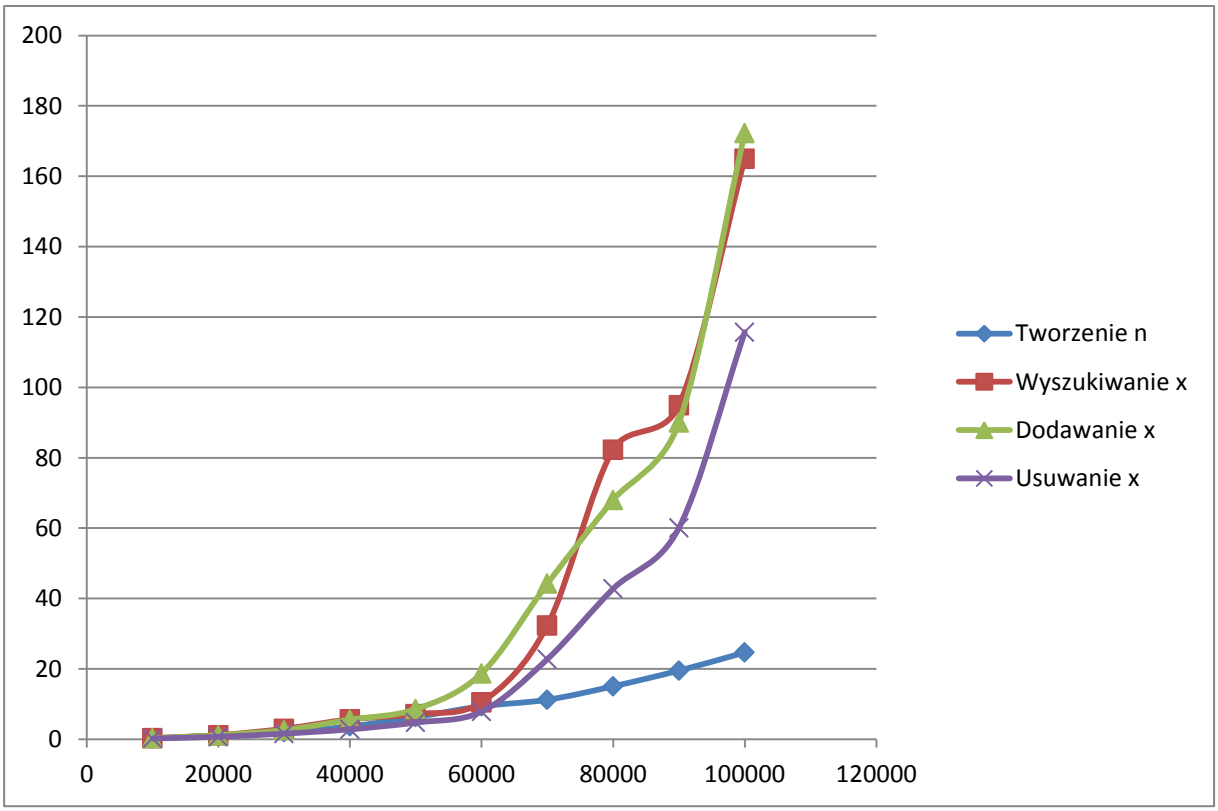
Sprawozdanie – struktury danych

LISTA



Złożoność:

- Wyszukiwanie elementu O(n)
- Dodawanie elementu O(n)
- Usuwanie elementu O(n)
- Tworzenie O(n^2)

Wyszukanie elementu na posortowanej liście wymaga wykonania maksymalnie n porównań w przypadku szukania największego elementu. Porównujemy każdą kolejną daną na liście z tą, którą chcemy znaleźć, aż nie dotrzemy do liczb większych od niej. Jeżeli do tego czasu jeszcze jej nie znaleźliśmy, zwracamy odpowiedź, że takiej liczby nie ma naszej liście. Na wykresie widać, że wyszukiwanie elementów przypomina wykres funkcji kwadratowej. Dzieje się tak, ponieważ mierzyliśmy czas wstawiania $\frac{n}{2}$ elementów.

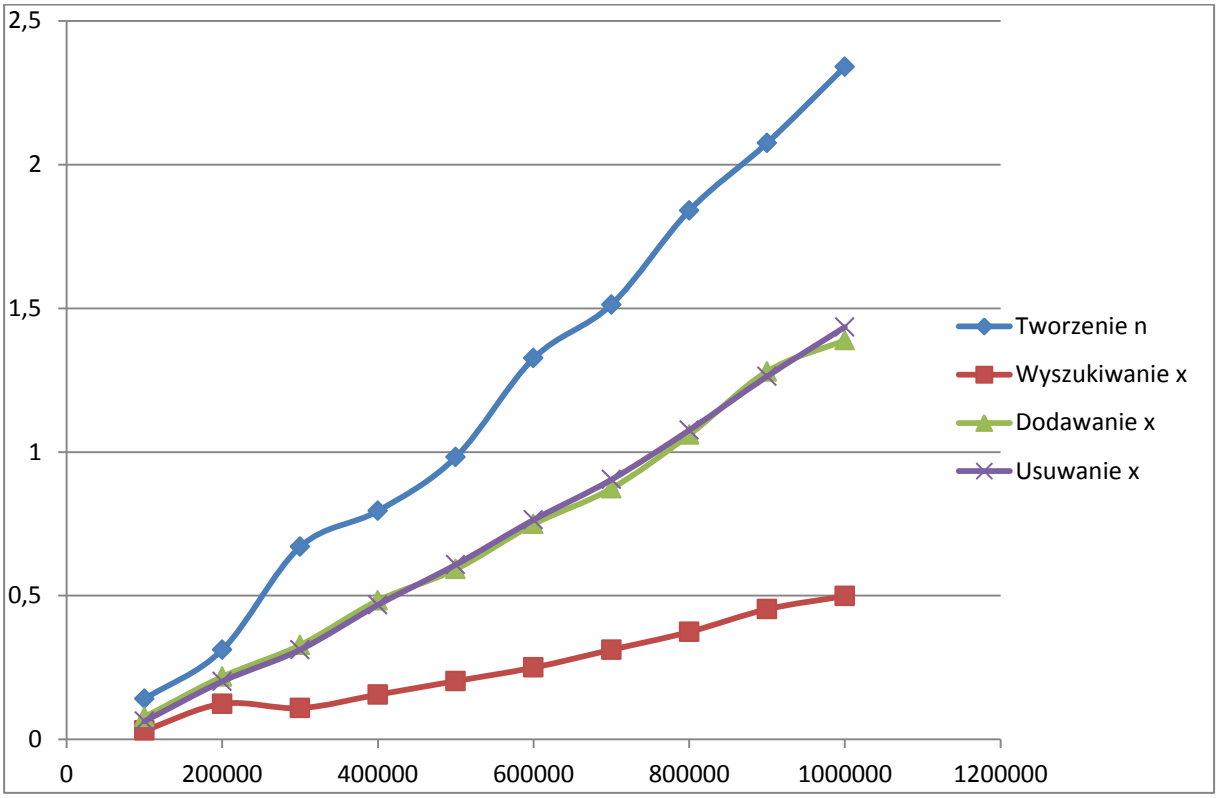
Dodawanie elementu również wymaga wykonania maksymalnie n porównań, gdy chcemy dodać liczbę większą od wszystkich znajdujących się na liście. By dodać element, przechodzimy przez listę, aż nie natrafimy na liczbę niemniejszą niż klucz, który chcemy wstawić. Wówczas znamy już odpowiednie dla niego miejsce. Wstawiamy go przed pierwszą liczbę niemniejszą od niego. Na wykresie dodawanie elementów przypomina parabolę, ponieważ dodawaliśmy $\frac{n}{2}$ dodatkowych elementów.

Usuwanie elementu przypomina jego wyszukiwanie, ponieważ najpierw musimy znaleźć miejsce, w którym on się znajduje, a później za wskaźnik na następny poprzedzającego go elementu podstawić

wskaźnik na następny znalezionego elementu. Następnie możemy zwolnić pamięć zajmowaną przez tę liczbę. Mierząc czas operacji, usuwaliśmy $\frac{n}{2}$ elementów.

Tworzenie struktury ma złożoność kwadratową, ponieważ w najgorszym przypadku dla każdej wstawianej liczby musimy przejść listę aż do końca. Czynność tę powtarzamy n razy.

BST – Binary Search Tree



Złożoność:

- Najgorszy przypadek, gdy dane są posortowane.
  - Wyszukiwanie elementu O(n)
  - Dodawanie elementu O(n)
  - Usuwanie elementu O(n)
  - Tworzenie O(n^2)
- Średni przypadek dla danych losowych.
  - Wyszukiwanie elementu O$(\log_2 n)$
  - Dodawanie elementu O$(\log_2 n)$
  - Usuwanie elementu O$(\log_2 n)$
  - Tworzenie O$(n \log_2 n)$

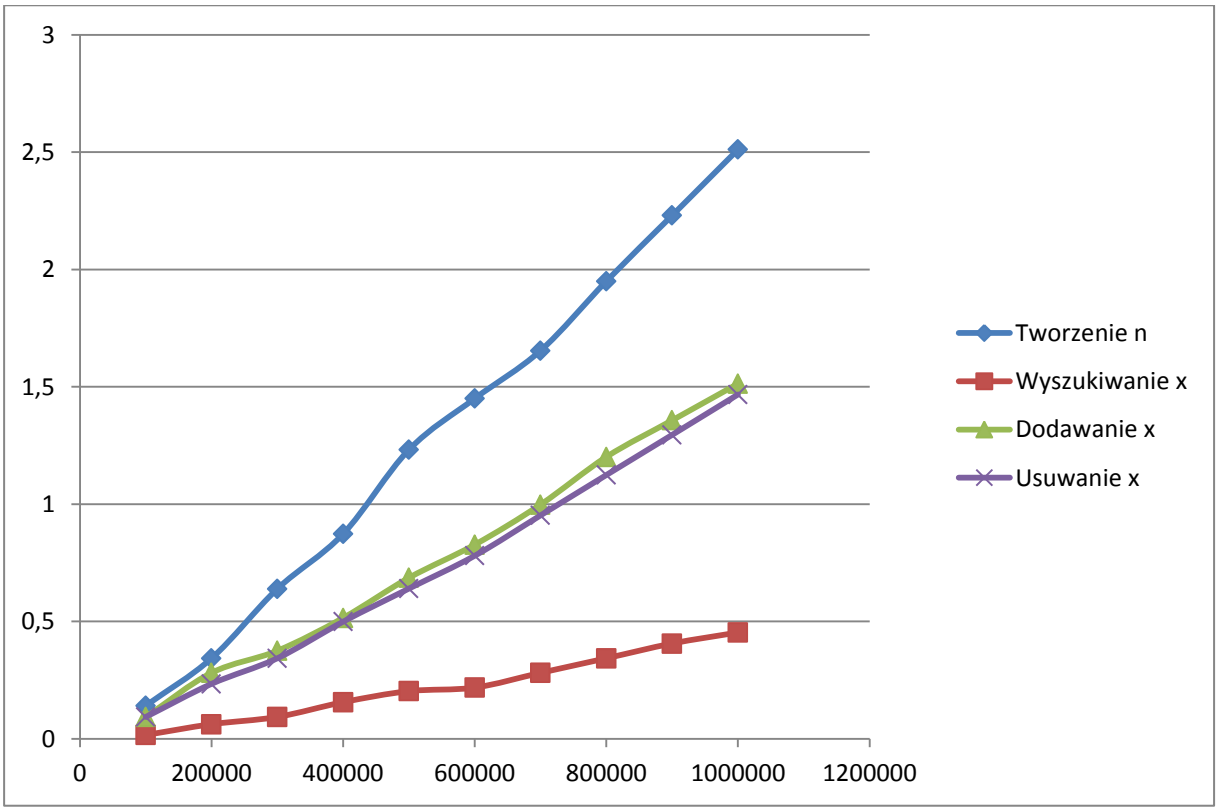
Wyszukiwanie elementu w drzewie BST polega na sprawdzeniu w każdym napotkanym węźle, czy liczba się w nim znajdująca jest większa, równa, czy mniejsza od szukanej. Gdy liczby są równe, możemy uznać wyszukiwanie za zakończone. Jeżeli szukana liczba jest większa, idziemy do prawego dziecka, jeżeli zaś mniejsza – do lewego (pod warunkiem, że takie dzieci są). Jeśli nie drzewo nie ma dziecka, do którego powinniśmy się udać, możemy zakończyć wyszukiwanie z negatywnym rezultatem. Złożoność tej operacji jest najgorsza w przypadku drzewa zbudowanego z danych posortowanych, a wynika to bezpośrednio z kształtu drzewa, które w takim przypadku przypomina listę, mając w każdym węźle tylko jedno dziecko. Średnio zaś wysokość drzewa jest logarytmiczna i tyle zajmuje dotarcie do jego liści.

Gdy chcemy dodać element, musimy znaleźć dla niego odpowiednie miejsce tak, by zachować własności drzewa BST. Zaczynamy od korzenia drzewa i sprawdzamy, czy liczba, którą chcemy wstawić, jest od niego mniejsza, czy większa. Jeżeli element, który chcemy dodać jest mniejszy, idziemy do lewego dziecka, a jeżeli większy – do prawego (jeżeli owo istnieje). Gdy okazuje się, że nie ma dziecka, do którego mieliśmy przejść, znaleźliśmy właśnie miejsce dla nowej liczby w drzewie. To ona zostaje brakującym dzieckiem. Złożoność tej operacji zależy od czasu dojścia do liścia drzewa. Gdy drzewo bardziej przypomina listę, bo zostało utworzone z posortowanych danych, operacja ta ma złożoność liniową. W średnim przypadku zaś logarytmiczną.

Usuwanie elementu zaczyna się od znalezienia jego miejsca w drzewie. Sytuacja jest prosta, gdy owa liczba jest liściem. Wówczas wystarczy ją usunąć. Może zdarzyć się też, iż ma ona tylko jedno dziecko. Wtedy również nie ma problemu, bo zastępowana jest po prostu przez ów dziecko. Największe komplikacje pojawiają się, gdy usuwany element ma dwójkę dzieci. W takim przypadku musimy znaleźć odpowiednią liczbę, którą moglibyśmy go zastąpić. Wybieramy najbardziej lewy element w prawym poddrzewie lub najbardziej prawy w lewym poddrzewie. Jego wartość podstawiamy w usuwanym węźle, a liść, w którym się on znajduje usuwamy. Operacja ta wymaga dostania się do liścia drzewa. Gdy mamy do czynienia ze strukturą przypominającą listę złożoność jest liniowa. W średnim przypadku jest znacznie lepsza, czyli logarytmiczna.

Tworzenie struktury wymaga powtórzenia n razy operacji dodawania elementu do struktury. Złożoność operacji dodawania została już opisana i waha się pomiędzy logarytmiczną a liniową. Tak więc, w pesymistycznym przypadku tworzenie BST ma złożoność $n * n$, a w średnim $n * \log_2 n$.

AVL (Adelson-Velski, Landis)



Drzewo AVL jest wyważonym drzewem BST. Drzewo nazywamy wyważonym wtedy i tylko wtedy, gdy dla każdego węzła wysokości dwóch jego poddrzew różnią się co najwyżej o jeden.

Wyszukiwanie elementu będzie wyglądać tak samo, jak w przypadku drzew BST. Jednakże złożoność tej operacji zawsze będzie logarytmiczna, ponieważ drzewo jest wyważone i nie ma tu mowy o tak złym przypadku jak dla struktury listo podobnej.

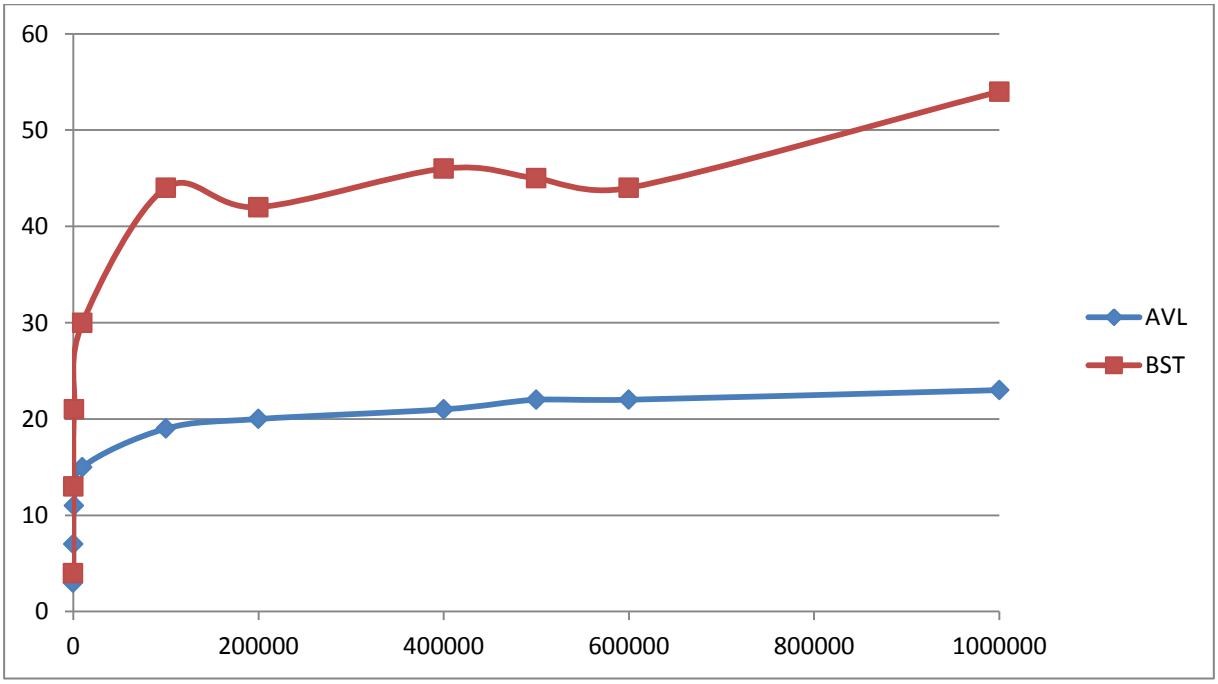
Dodawanie elementu również przebiega analogicznie do drzew BST. Niemniej jednak po dołożeniu liczby może zostać zachwiana równowaga w strukturze i drzewo przestanie być wyważone. Wówczas trzeba na nowo zrównoważyć drzewo poprzez wykonanie rotacji. Ta dodatkowa operacja wpływa na złożoność dodawania elementu i czas wstawiania liczby jest nieco większy od czasu jej wyszukiwania.

Usuwanie odbywa się tak samo jak w drzewach BST. Po wykasowaniu elementu należy natomiast wykonać rotacje, jeżeli równowaga struktury została zachwiania. W przypadku usuwania może się okazać, że ilość rotacji będzie całkiem spora. Musimy sprawdzić wyważanie każdego węzła aż do korzenia.

Tworzenie drzewa to powtórzenie n razy operacji dodawania.

Wysokości drzew

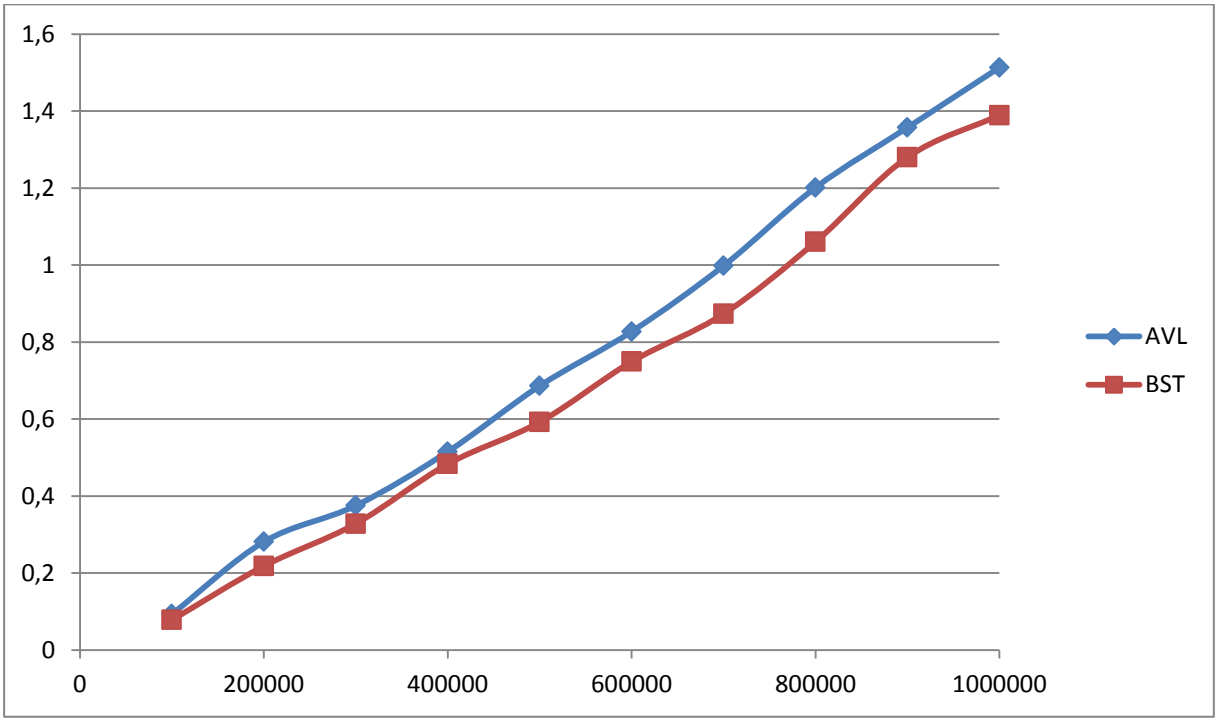
Wysokości drzew mierzone były dla danych losowych.



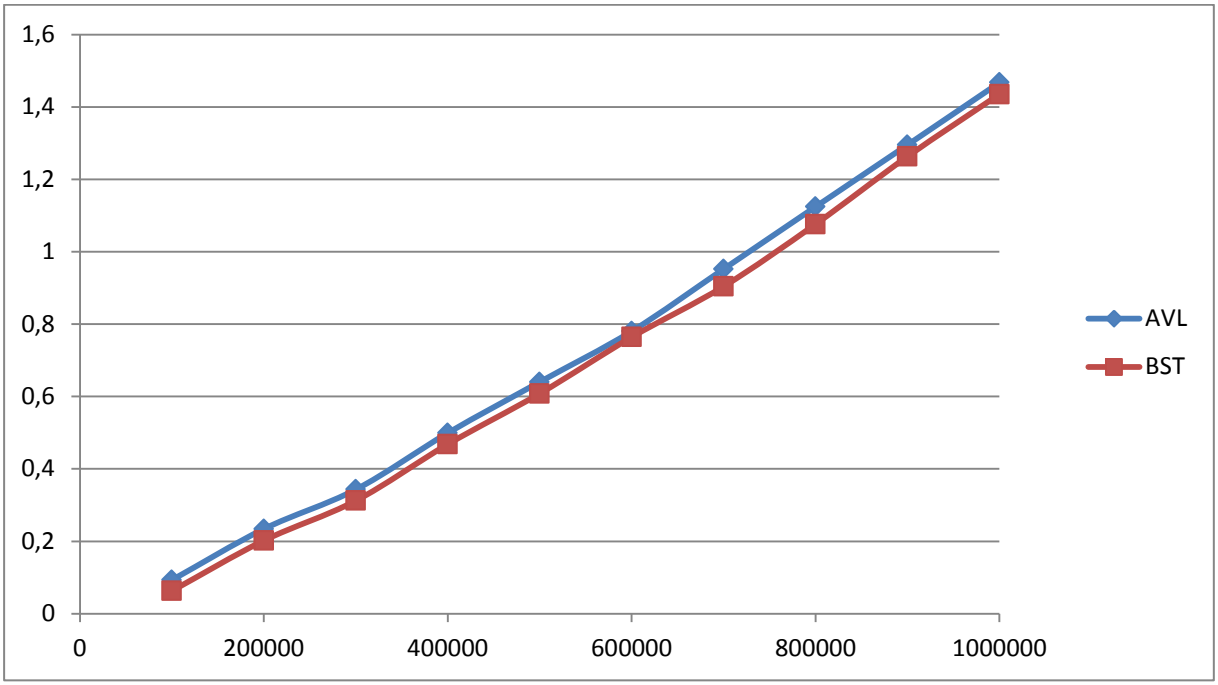
N	AVL	BST
10	3	4
100	7	13
1000	11	21
10000	15	30
100000	19	44
200000	20	42
400000	21	46
500000	22	45
600000	22	44
1000000	23	54

Widać, że dzięki rotacjom wysokości drzew AVL są mniejsze od wysokości drzew BST. Wyliczono, że wysokość drzewa AVL w najgorszym przypadku wyniesie $1{,}44 * \log_2 n$, co jest niewiele gorszym wynikiem od drzewa dokładnie wyważonego, które jest trudniejsze w implementacji. Drzewa BST mogą mieć maksymalnie wysokość n dla danych posorotwanych. Doskonale widać jednak, że średnio dla losowych danych zachowują się dużo lepiej, mając wysokość logarytmiczną.

Operacje na drzewach

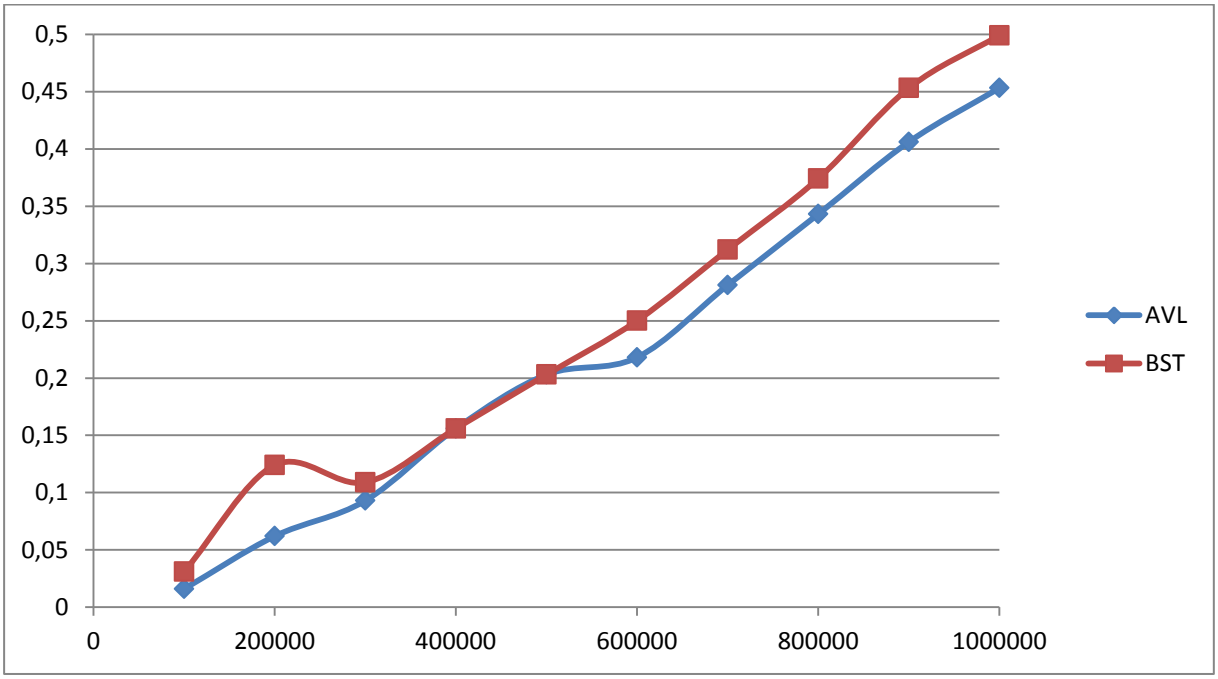


Dodawanie $\frac{n}{2}$ dodatkowych elementów



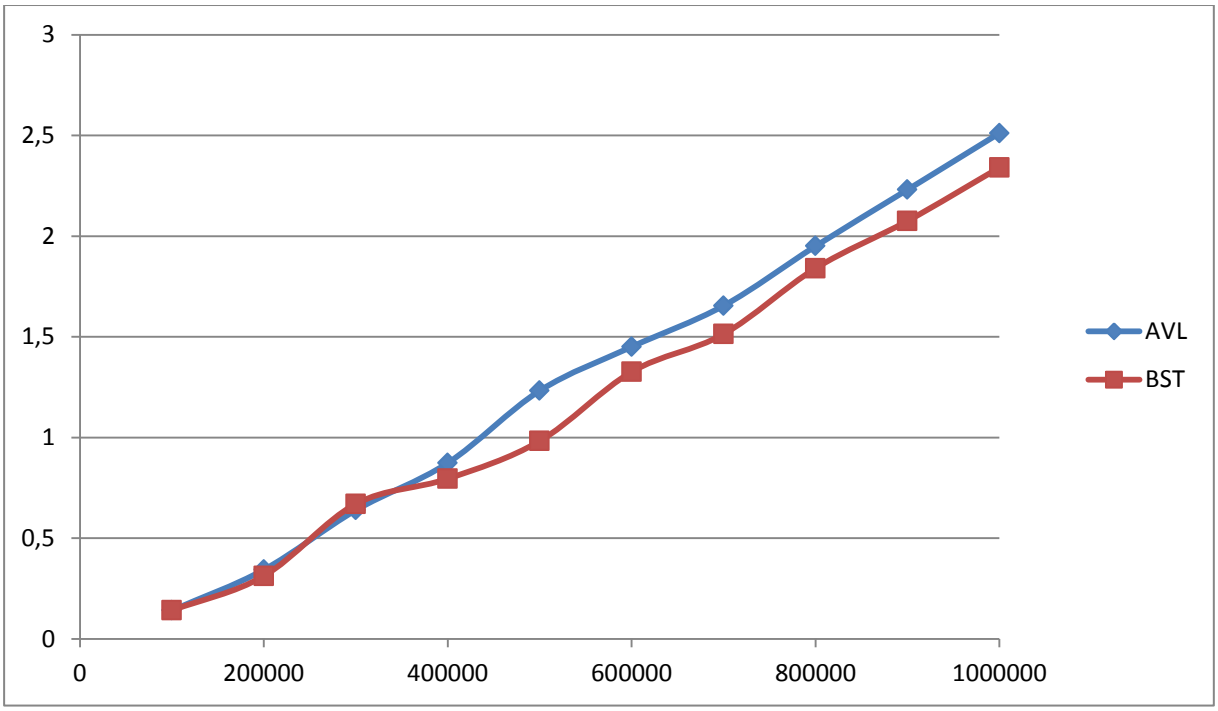
Usuwanie $\frac{n}{2}$ elementów

Na powyższych wykresach można zauważyć, że operacje dodawania i usuwania elementów zajmują trochę mniej czasu w przypadku drzew BST. W prawdzie wysokość drzewa BST jest większa (ale nieznacznie dla losowych danych, dla których wykonane były testy) i dłużej trwa dostanie się do ów elementu, ale nie musimy wykonywać rotacji tak, jak dla struktury AVL.



Wyszukiwanie $\frac{n}{2}$ elementów

Właśnie ta operacja jest największą zaletą drzew AVL. Niezależnie od ułożenia danych wejściowych drzewo to ma wysokość logarytmiczną, zatem znalezienie elementu ma taką złożoność. W AVL jesteśmy w stanie szybko dostać się do potrzebnego elementu.



Tworzenie struktury

Tworzenie struktury polega na powtórzeniu n razy operacji dodawania elementu. Zajmuje więcej czasu w przypadku drzew AVL przez konieczność zrównoważania drzewa.