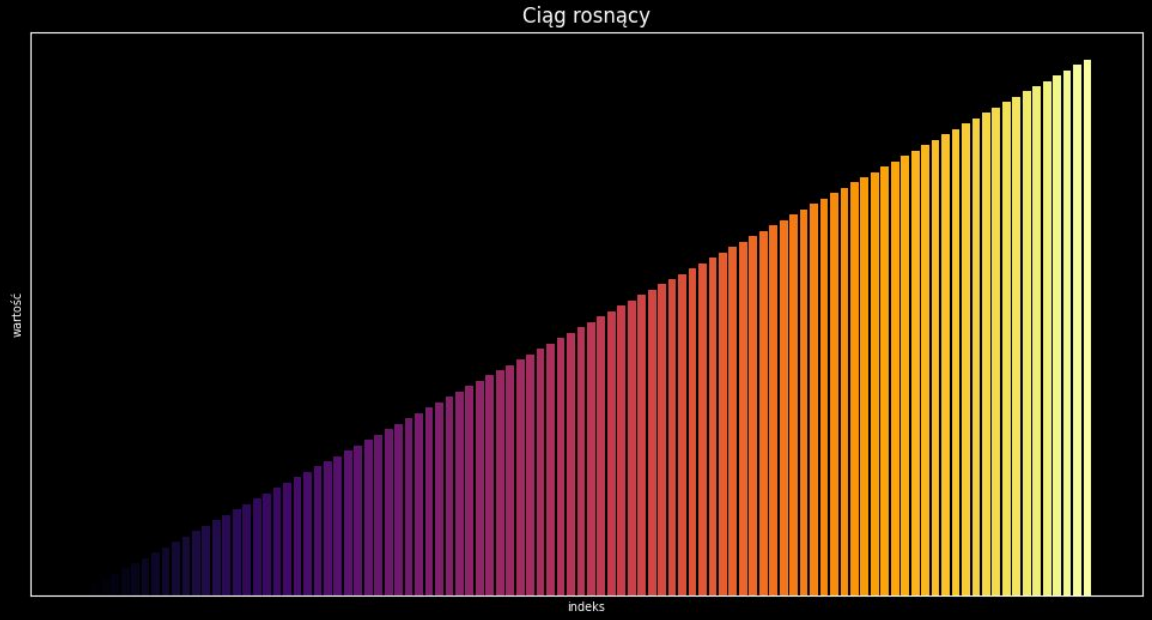
Sprawozdanie

Drzewa binarne

Jan Cieśla 155945

Dane wejściowe

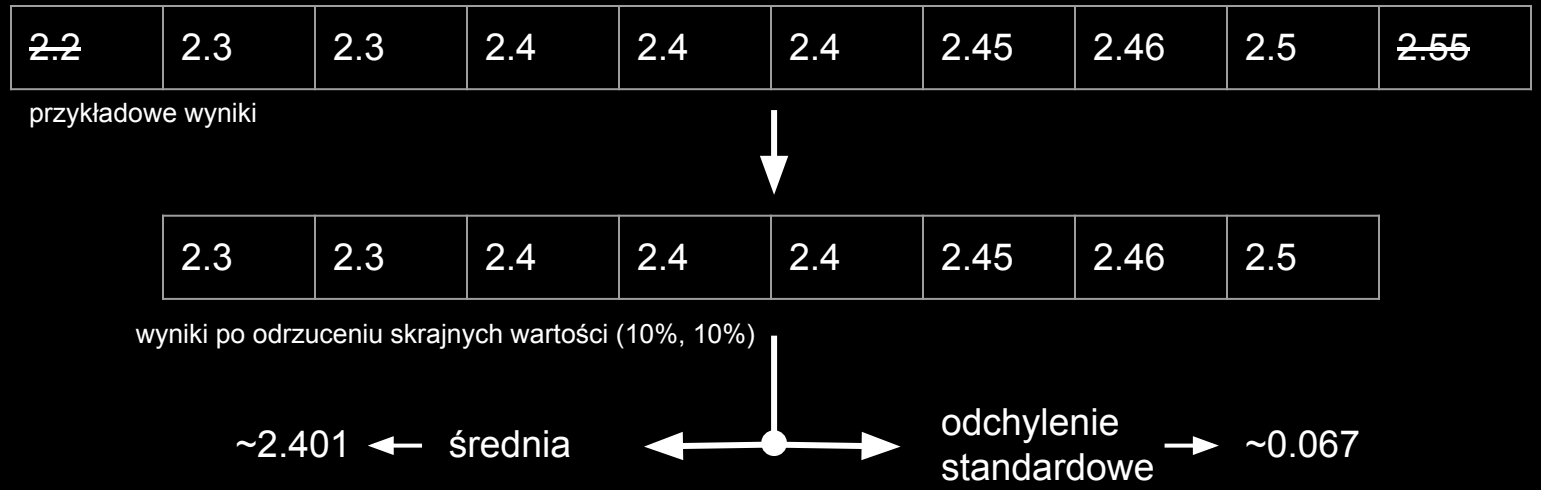


Dane wejściowe generowane są jako n-elementowy ciąg arytmetyczny o wyrazie minimalnym i maksymalnym odpowiednio 1 i n.

Do porównania efektywności algorytmów wybraliśmy n’y zaczynające się od 5,000 zwiększające się co 5,000 do 50,000

Testy algorytmów

W ramach testów algorytmów dwudziestokrotnie zmierzyliśmy czas ich działania dla każdego zestawu danych. Odrzuciliśmy skrajne wyniki (25% najwyższych i 25% najniższych), a z pozostałych wyników obliczyliśmy średnią oraz odchylenie standardowe - te dane są przedstawione na wykresach. W opisach algorytmów n będzie oznaczało ilość węzłów, a h wysokość drzewa



Tworzenie drzewa z tablicy

W implementacji drzewa są tworzone przez różne algorytmy

AVL - tworzenie drzewa przez binarne dzielenie tablicy

BST - tworzenie drzewa przez wstawianie kolejnych elementów tablicy do drzewa

Tworzenie drzewa przez binarne dzielenie tablicy

Przypadek optymistyczny

Czasowa: O(n)

Pamięciowa: O(n)

Przypadek średni

Czasowa: $O(n \cdot \log(n))$

Pamięciowa: O(n)

Przypadek pesymistyczny

Czasowa: $O(n \cdot \log(n))$

Pamięciowa: O(n)

Algorytm wymaga wstępnego posortowania tablicy. W implementacji użyliśmy Timsort - połączenie sortowania przez wstawianie i sortowania przez scalanie. W najlepszym przypadku osiąga on złożoność czasową O(n) i pamięciową O(n) dla danych posortowanych. Najgorszy przypadek jest równy średniemu - są to złożoności $O(n \cdot \log(n))$ czasowa i O(n) pamięciowa.

Główna część algorytmu tworzy nowy węzeł dla każdego elementu tablicy. Stąd złożoność pamięciowa O(n). Wybór elementu dla którego utworzymy węzeł opiera się na wybraniu środkowego elementu tablicy dla którego wykonywana jest stała liczba porównań i przypisań oraz wywołaniu się (bez sortowania) dla powstałych przez to podtablic, skutkuje to złożonością pamięciową O(log(n)) ze względu na głębokość zagłębienia rekursji i czasową O(n) ponieważ węzeł tworzony jest w stałym czasie i każdy element tablicy jest wybierany jednokrotnie, nie jest wymagane wyszukiwanie miejsca do wstawienia węzła, ta informacja jest przekazywana w rekurencji.

Podsumowując złożoność pamięciowa to $O(n) + O(\log(n)) = O(n)$ i czasowa $O(n) + O(n) = O(n)$ optymistycznie i $O(n \cdot \log(n)) + O(n) = O(n \cdot \log(n))$ w przeciwnym wypadku

Ważną cechą algorytmu jest to, że w każdym przypadku tworzy on drzewo zrównoważone.

Tworzenie drzewa przez wstawianie kolejnych elementów tablicy do drzewa

Przypadek optymistyczny

Czasowa: $O(n \cdot \log(n))$

Pamięciowa: $O(n)$

Przypadek średni

Czasowa: ?

Pamięciowa: $O(n)$

Przypadek pesymistyczny

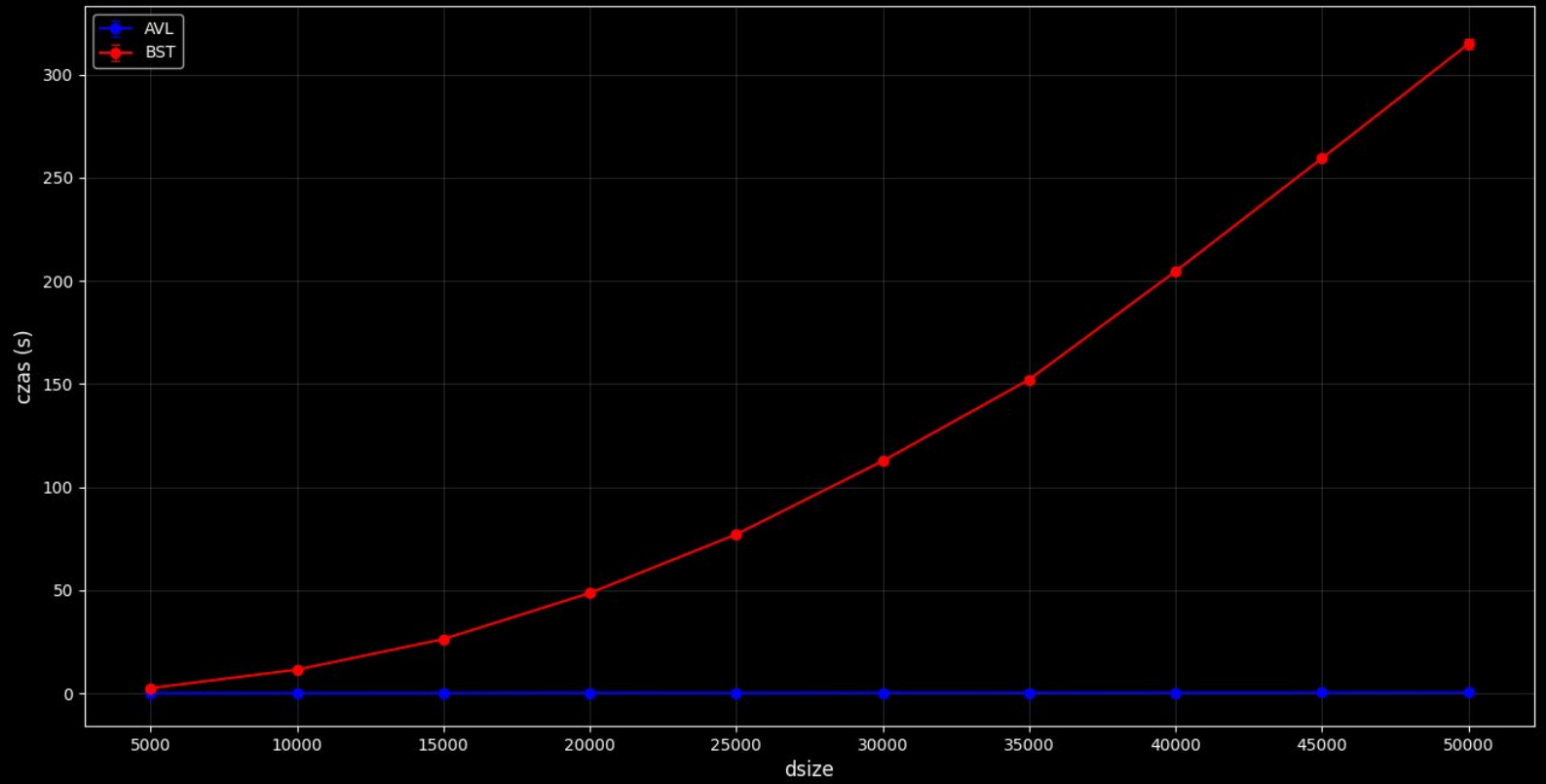
Czasowa: $O(n^2)$

Pamięciowa: $O(n)$

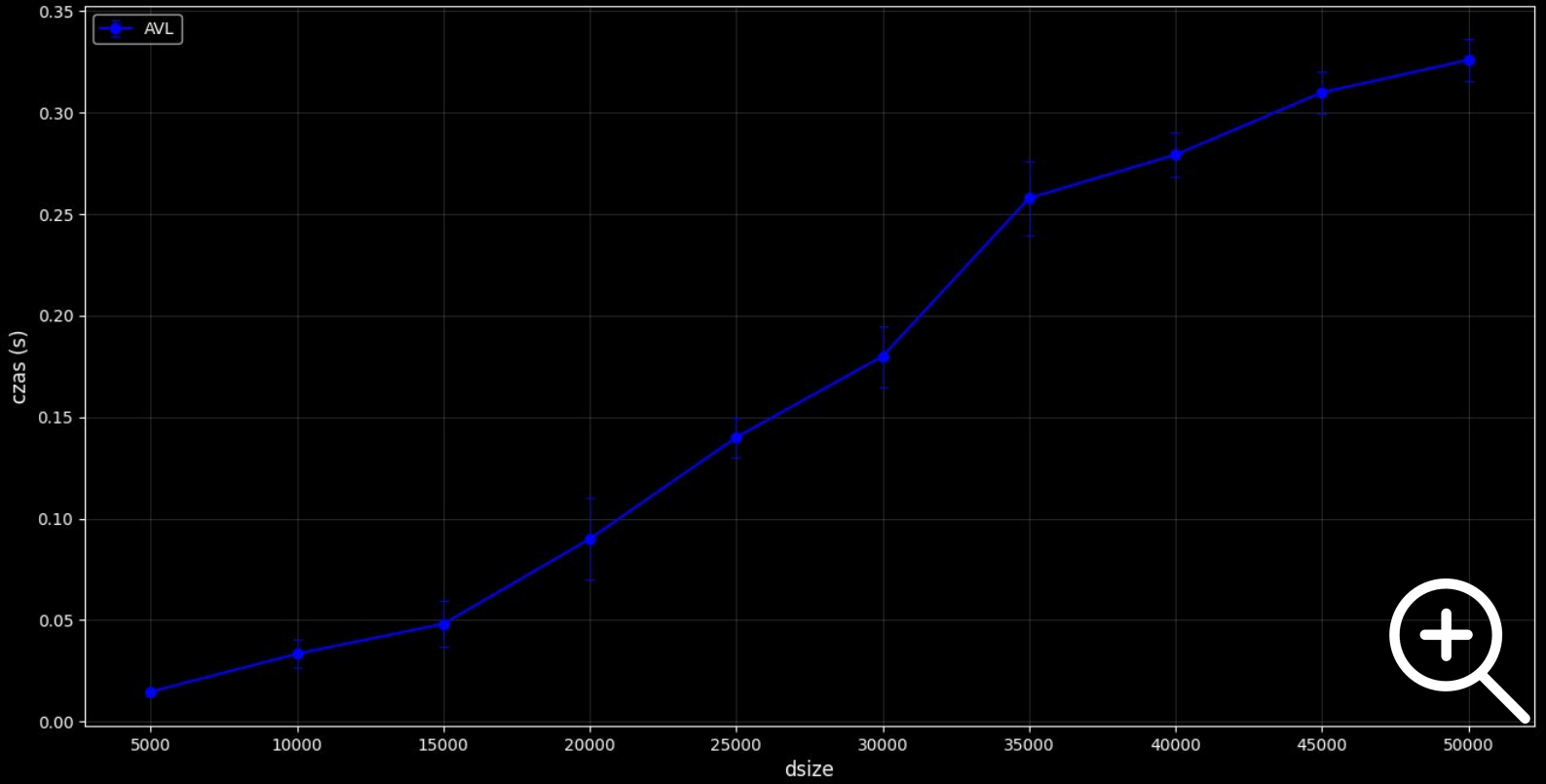
Algorytm wstawia kolejne elementy tablicy do drzewa. Dla drzewa pustego pierwszy wstawiany element staje się korzeniem, później zaczynając od niego w zależności od tego czy klucz nowego węzła jest mniejszy czy większy od aktualnego przemieszcza się odpowiednio do lewego lub prawego syna. W momencie natrafienia na puste miejsce węzeł jest dopisywany (ustawiane są wskaźniki). Złożoność czasowa zależy tylko od długości tablicy i długości ścieżki wstawiania to w przypadku drzewa idealnego będzie miała długość $\log_2 n$ a zdegenerowanego n. Średnio długość ścieżki w pesymistycznym przypadku (zawsze wstawiamy węzeł o kluczu większym od już wstawionych) wyniesie: $0 + 1 + 2 + \ldots n = (n^2 + n) \cdot 0.5$. Z tego wynika optymistyczna złożoność czasowa $O(n \cdot \log(n))$ i pesymistyczna $O(n^2)$.

Nowy węzeł jest tworzony dla każdego elementu tablicy stąd złożoność pamięciowa $O(n)$

Problemem algorytmu jest to, że najczęściej stworzy on drzewo niezrównoważone, a dla danych posortowanych winorośl.


make time
AVL
BST
czas (s)
0
50
100
150
200
250
300
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
dsize

make time


AVL
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
czas (s)
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
dsize

AVL

Algorytm połowienia binarnego nie ma pesymistycznych i optymistycznych przypadków, jednak posortowana tablica jest przypadkiem optymistycznym Timsort'a. Wykres zgodnie z oczekiwaną złożonością przypomina funkcję liniową.

BST

Posortowana rosnąco tablica jest przypadkiem pesymistycznym dla tworzenia drzewa przez wstawianie kolejnych elementów tablicy. Za każdym razem wstawiany element dołączamy na końcu winorośli,wydłużając ścieżkę dla następnego elementu. Wykres zgodnie z oczekiwaną złożonością przypomina odwróconą połówkę paraboli.

AVL i BST odnosi się do drzew tworzonych wg zadania

Wyszukiwanie elementu o maksymalnym kluczu

Przypadek optymistyczny

Czasowa: O(1)

Pamięciowa: O(1)

Przypadek średni

Czasowa: O(h)

Pamięciowa: O(1)

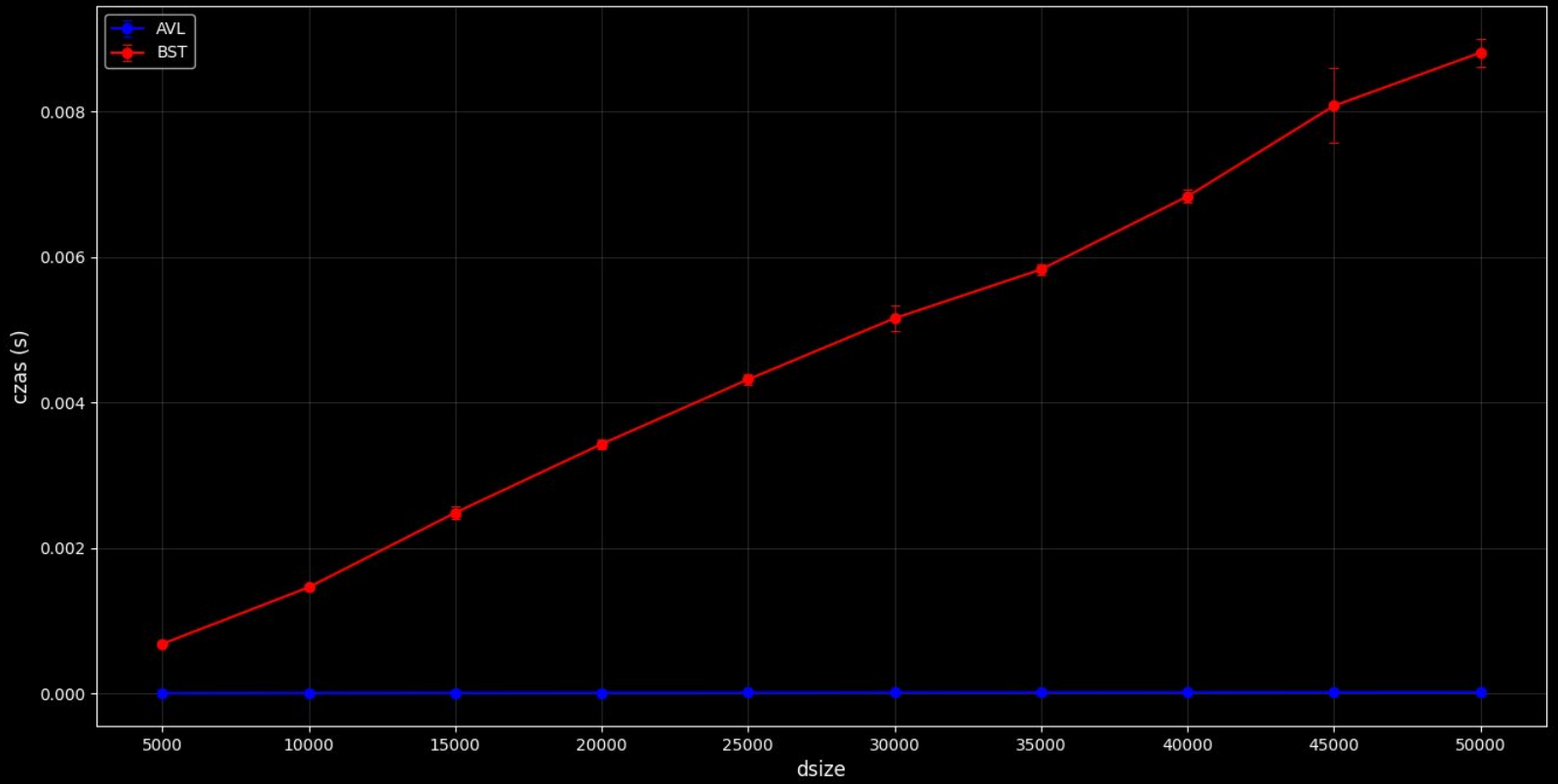
Przypadek pesymistyczny

Czasowa: O(n)

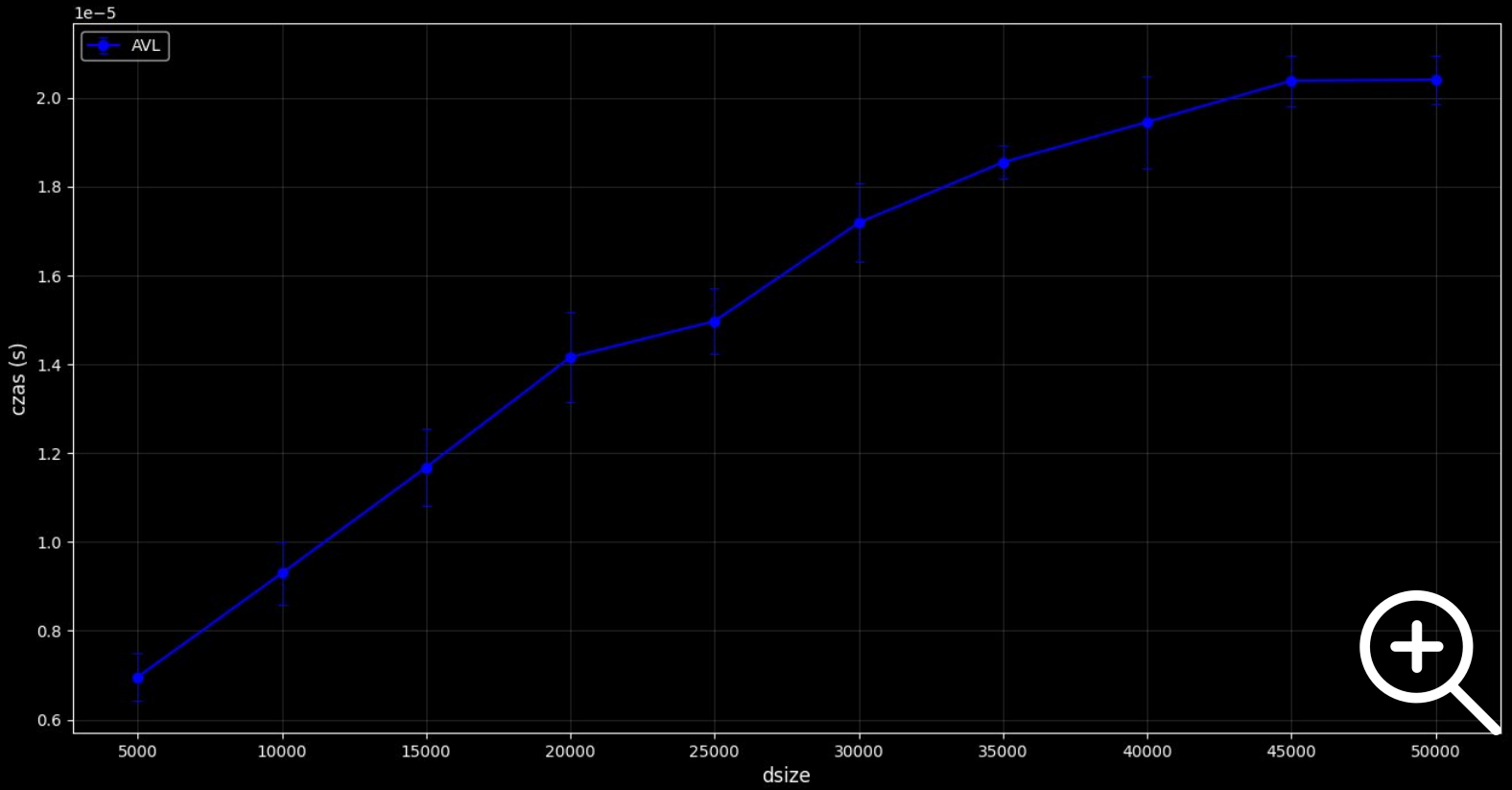
Pamięciowa: O(1)

Element o maksymalnym kluczu zawsze będzie skrajnie prawym liściem w binarnym drzewie poszukiwań. Aby się do niego dostać zaczynając od korzenia będziemy przechodzić do prawego syna dopóki ten istnieje. Złożoność czasowa tej operacji zależy od długości ścieżki dostępu. Pesymistycznie będzie to ostatni węzeł w drzewie zdegenerowanym do skrajnie prawej gałęzi i optymistycznie pierwszy węzeł w drzewie zdegenerowanym do skrajnie lewej gałęzi. Z tego wynika optymistyczna złożoność czasowa O(1) i pesymistyczna O(n). W przypadku drzewa idealnego długość ścieżki wynosi $\log_2 n$.

Algorytm nie potrzebuje dodatkowej pamięci.


max time
AVL
BST
0.008
0.006
0.004
0.002
0.000
czas (s)
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
dsize

max time


1e−5
AVL
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
czas (s)
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
dsize

AVL

Wyszukiwanie elementu o maksymalnym kluczu w drzewie zrównoważonym takim jak AVl sprowadza się do przejścia skrajnie prawej krawędzi, która może mieć długość najwyżej $1.44\log_2 n$. Wykres zgodnie z oczekiwaną złożonością przypomina funkcję logarytmiczną.

BST

Wyszukiwanie elementu o maksymalnym kluczu w drzewie zdegenerowanym do jednej skrajnie prawej gałęzi wymaga przejścia wszystkich węzłów, jest to przypadek pesymistyczny tego algorytmu. Wykres zgodnie z oczekiwaną złożonością przypomina funkcję liniową

AVL i BST odnosi się do drzew utworzonych wg zadania

Wypisanie drzewa in-order

Przypadek optymistyczny

Czasowa: O(n)

Pamięciowa: O(log(n))

Przypadek średni

Czasowa: O(n)

Pamięciowa: O(h)

Przypadek pesymistyczny

Czasowa: O(n)

Pamięciowa: O(n)

Złożoność czasowa nie zależy od zrównoważenia drzewa. Algorytm odwiedza wszystkie węzły raz co prowadzi do złożoności czasowej O(n).

Ze względu na to, że algorytm odkłada na stosie węzły których przetwarzania nie zakończył wymagana jest dodatkowa pamięć. Jej ilość jest zależna od wysokości drzewa. Dla drzewa zdegenerowanego złożoność pamięciowa wynosi O(n), jest to przypadek pesymistyczny. W przypadku drzewa AVL wysokość jest najwyżej równa $1.44\log_2 n$ w tym wypadku złożoność pamięciowa jest logarytmiczna.

Zgodnie z określoną złożonością czasy algorytmów w drzewach AVL i BST nie odbiegają od siebie, a ich wykresy przypominają funkcje liniowe.

w ostatnim akapicie AVL i BST odnosi się do drzew utworzonych wg zadania


inorder time
AVL
BST
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
czas (s)
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
dsize

Równoważenie drzewa DSW

Przypadek optymistyczny

Czasowa: O(n)

Pamięciowa: O(1)

Przypadek średni

Czasowa: O(n)

Pamięciowa: O(1)

Przypadek pesymistyczny

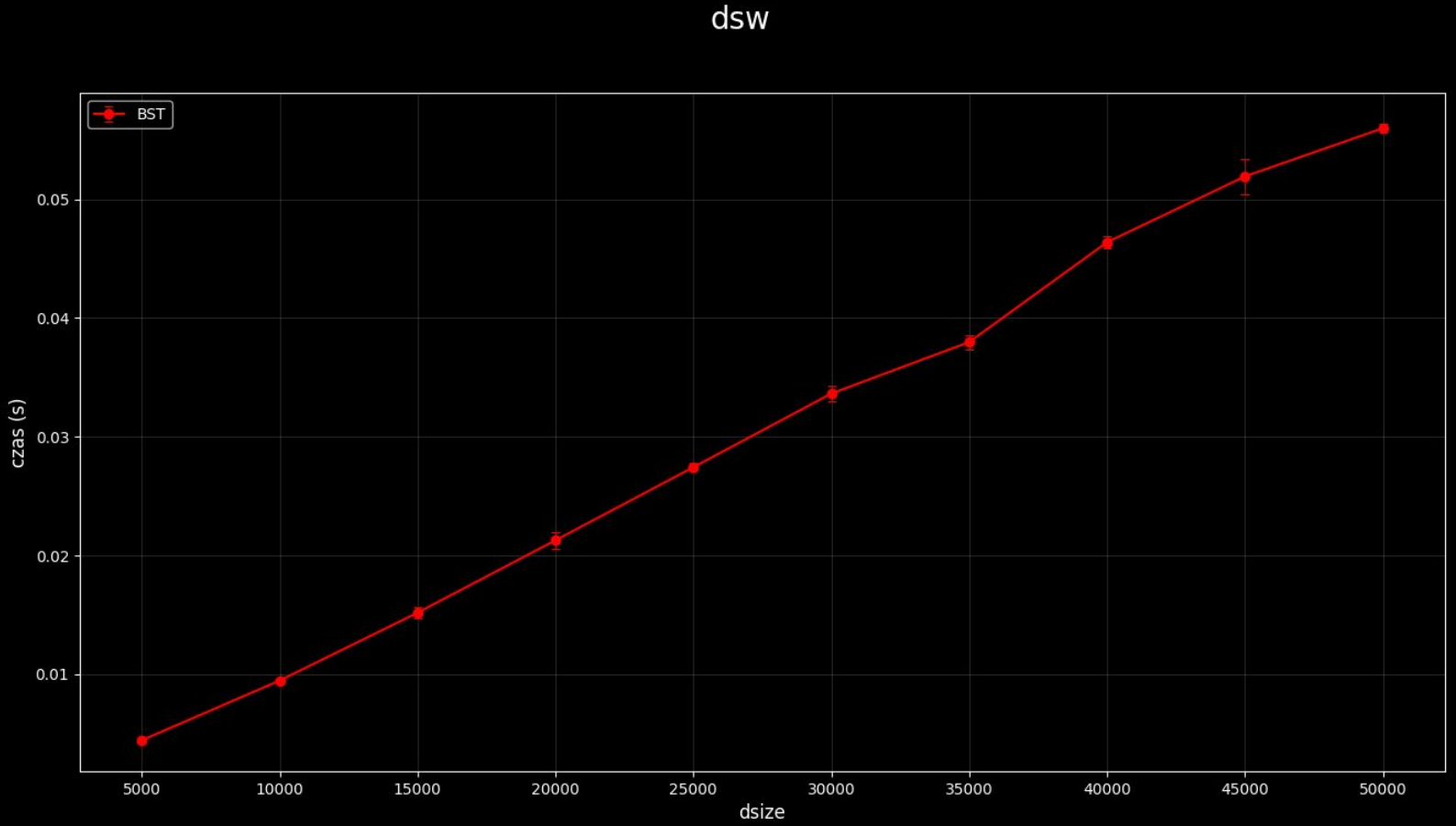
Czasowa: O(n)

Pamięciowa: O(1)

Algorytm składa się z dwóch części. W pierwszej “prostujemy” drzewo do winorośli przez rotację węzłów w prawo (LL). Rotacja jest przeprowadzana przez stałą liczbę operacji na wskaźnikach stąd jej złożoność czasowa i pamięciowa są stałe. Zaczynając od korzenia sprawdzamy czy węzeł ma lewego syna, jeśli tak to wykonujemy rotację i przechodzimy do nowo utworzonego rodzica w przeciwnym wypadku przechodzimy do prawego syna. algorytm kończy się gdy ten nie ma prawego syna. podsumowując w tej części wykonujemy stałą liczbę operacji dla wszystkich węzłów. Złożoność czasowa jest liniowa a pamięciowa stała.

Druga część algorytmu wykonuje rotację w lewo (RR) na co drugim węźle skrajnie prawej gałęzi. w pierwszej fazie wykonuje się tyle rotacji ile liści jest na ostatnim poziomie drzewa, później rotacje wykonywane są w iteracjach. Za każdym razem zaczynając od korzenia wykonywane jest połowa rotacji z poprzedniej iteracji zaczynając od połowy liczby liści kończąc gdy ilość rotacji do przeprowadzenia spadnie poniżej 1. Sumując n/2 + n/2 + n/4+n/8.. wykonujemy najwyżej 3/2 n rotacji. Złożoność czasowa jest liniowa a pamięciowa stała.

Podsumowując obie części działają w czasie liniowym bez dodatkowej pamięci zatem algorytm DSW również działa w czasie liniowym bez dodatkowej pamięci.


dsw
BST
czas (s)
0.01
0.02
0.03
0.04
0.05
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
dsize

BST

Drzewo zdegenerowane do skrajnie prawej gałęzi jest optymistycznym przypadkiem dla DSW pomijane są wszystkie rotacje w pierwszej części algorytmu. nie zmienia to jednak ogólnej złożoności programu. Wykres zgodnie z założoną złożonością przypomina funkcję liniową.

BST odnosi się do drzewa utworzonego wg zadania