Złożone struktury danych

SPRAWOZDANIE II

PAWEŁ KOLEC 155873

Wprowadzenie:

W niniejszym sprawozdaniu zostaną zaprezentowane wyniki badań wydajności drzew BST oraz AVL w języku Python. W trakcie eksperymentów zostanie przeprowadzone pięć testów, podczas których zmierzymy czas wykonania trzech podstawowych operacji na drzewach: tworzenie struktury, wyszukiwanie minimum oraz wypisywanie elementów w kolejności in-order. Na każdym wykresie zostaną przedstawione dwie krzywe - jedna dla drzewa BST, druga dla drzewa AVL. Ponadto, zostanie wykonany wykres prezentujący zależność czasu równoważenia od liczby elementów w losowym drzewie BST.

Eksperymenty zostaną przeprowadzone dla różnych rozmiarów drzew, a wyniki przedstawione na wykresach pozwolą na analizę i porównanie wydajności obu struktur danych. Celem badań jest określenie, która struktura działa szybciej dla drzew o różnych rozmiarach oraz jak zmienia się czas działania w zależności od liczby elementów w drzewie.

Do przeprowadzenia badań zostanie wykorzystane środowisko programistyczne PyCharm na na systemie Microsoft Windows 11 Home z procesorem Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz. Wyniki uzyskane podczas eksperymentów pozwolą na dokładne porównanie wydajności drzew BST i AVL oraz wybór najlepszej struktury danych dla konkretnych zastosowań. Dane wejściowe przyjęte do badań będą miały posortowane malejąco oraz ich ilość to odpowiednio 1000, 5000, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000.

Wstęp teoretyczny:

Drzewa BST (Binary Search Tree) i AVL (Adelson-Velsky and Landis) to struktury danych wykorzystywane do przechowywania elementów w sposób pozwalający na ich szybkie wyszukiwanie, wstawianie oraz usuwanie. Drzewo BST jest strukturą, w której każdy węzeł ma maksymalnie dwoje dzieci, a wartość węzła po lewej stronie jest mniejsza niż wartość węzła po prawej stronie. Dzięki temu, przeszukiwanie drzewa jest wykonywane szybko, ponieważ w zależności od wartości szukanego elementu, przemieszczamy się w odpowiednim kierunku drzewa.

Drzewo AVL natomiast to drzewo BST z dodatkowymi ograniczeniami, zapewniającymi zrównoważenie drzewa. Oznacza to, że wysokość lewego i prawego poddrzewa każdego węzła może różnić się co najwyżej o jeden. Dzięki temu, złożoność czasowa operacji na drzewie AVL jest równoważna logarytmicznej, co zapewnia szybkie wyszukiwanie, wstawianie i usuwanie elementów.

Cel badania:

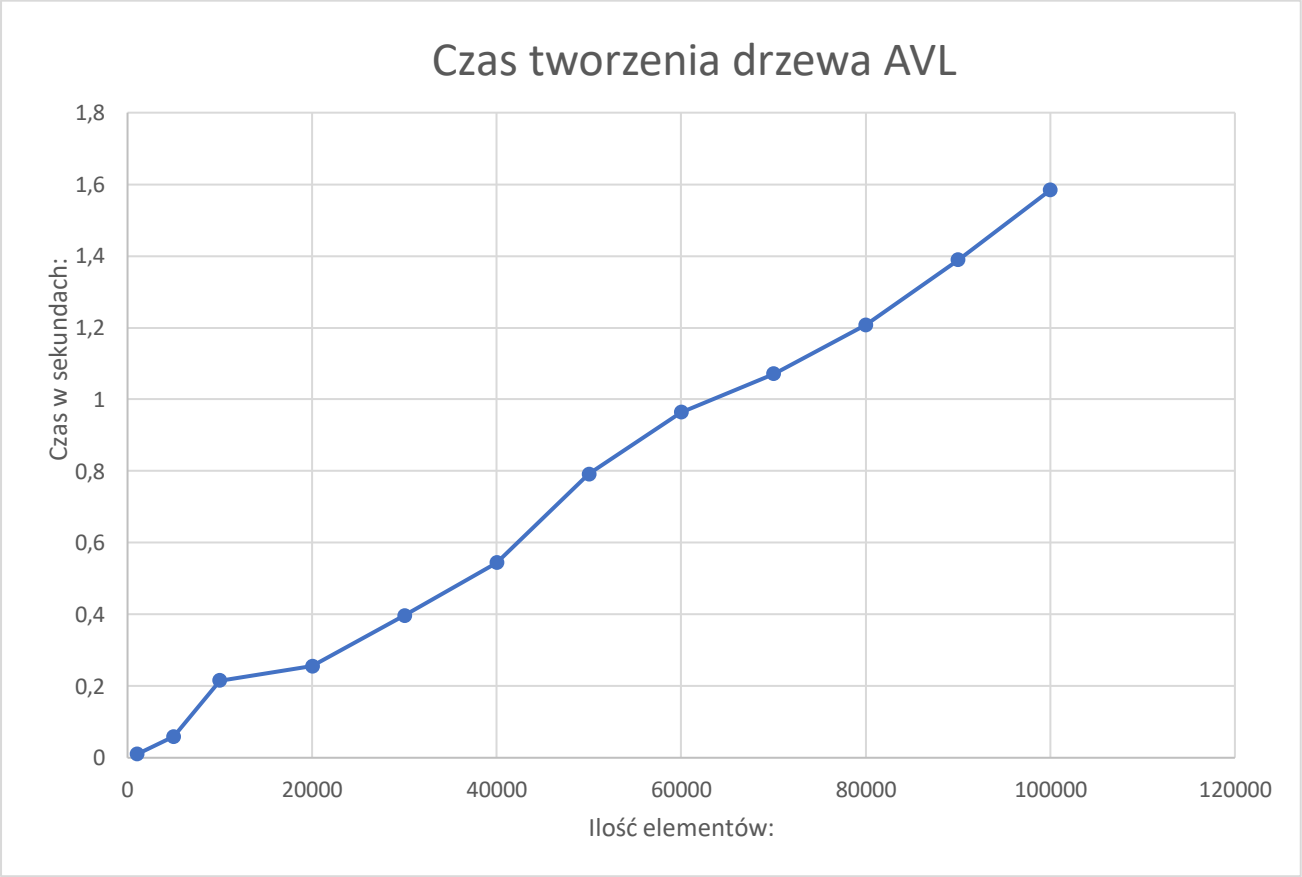
Celem badania jest określenie, która struktura danych, drzewo BST czy AVL, działa szybciej dla różnych rozmiarów drzew oraz w jaki sposób czas działania zależy od liczby elementów w drzewie. Badanie ma na celu porównanie wydajności tych struktur danych i określenie, która z nich jest bardziej odpowiednia dla danego zastosowania.

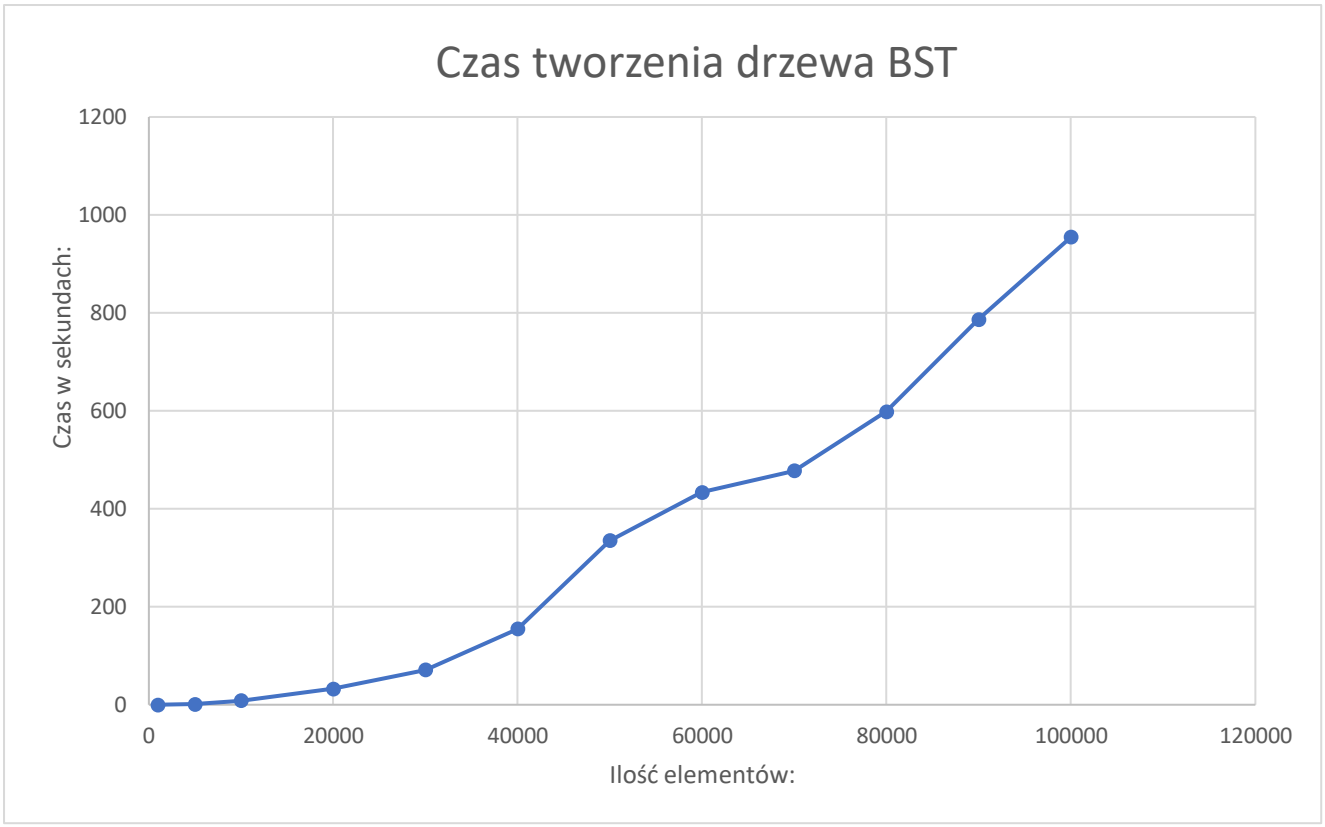
Porównanie czasu tworzenia drzewa AVL i BST dla danych posortowanych malejąco:

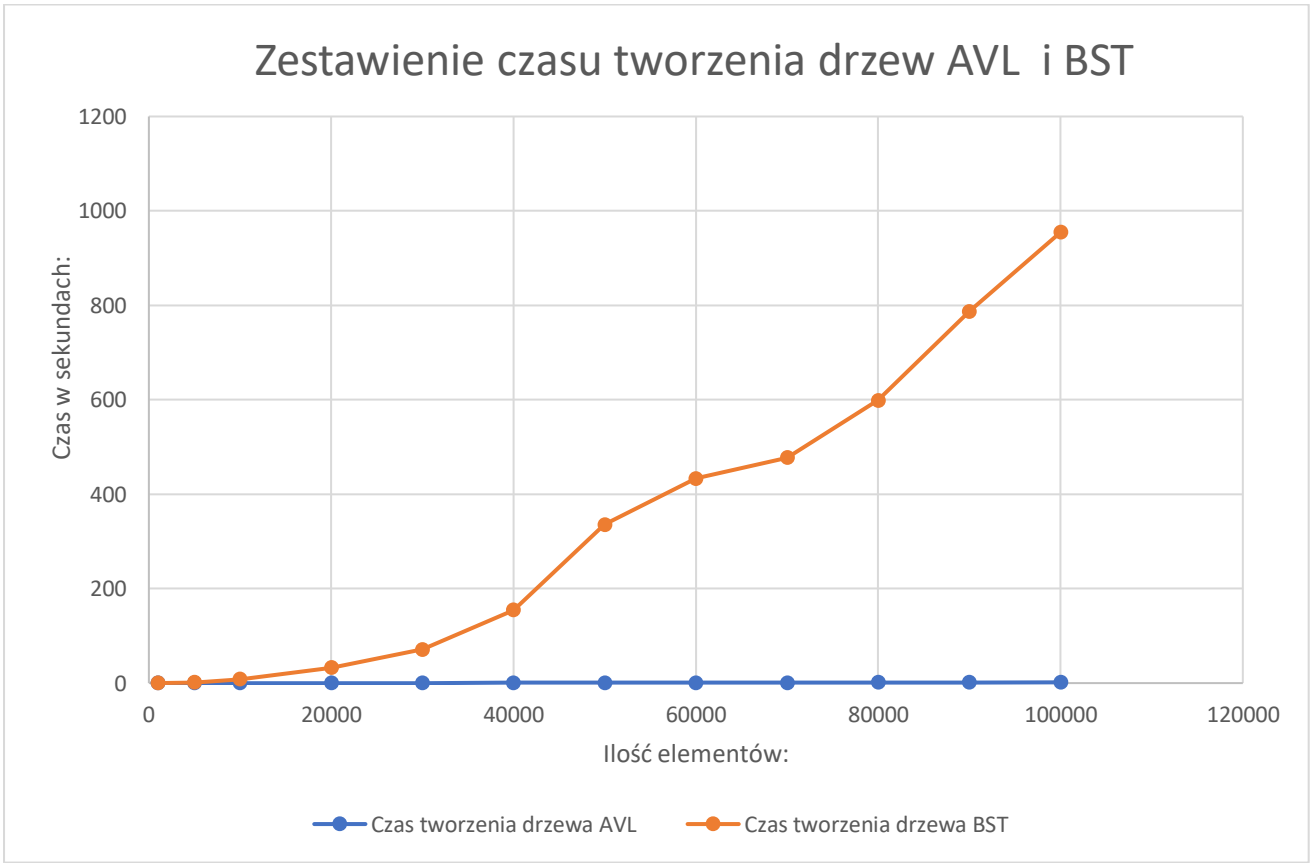
Wstep:

Drzewa AVL i BST to dwie popularne struktury danych, które służą do przechowywania i organizowania danych w postaci drzewa. Drzewo BST (Binary Search Tree) jest drzewem binarnym, w którym dla każdego węzła lewe poddrzewo ma klucze mniejsze od klucza węzła, a prawe poddrzewo ma klucze większe od klucza węzła. Drzewo AVL (Adelson-Velsky and Landis Tree) to rodzaj drzewa BST, które jest automatycznie zrównoważone poprzez rotacje węzłów w celu utrzymania zbalansowanej struktury.

Zestawienie czasu tworzenia drzew AVL i BST:




Czas tworzenia drzewa BST
Czas w sekundach:
1200
1000
800
600
400
200
0
0
20000
40000
60000
80000
100000
120000
Ilość elementów:


Zestawienie czasu tworzenia drzew AVL i BST
Czas w sekundach:
1200
1000
800
600
400
200
0
0
20000
40000
60000
80000
100000
120000
Ilość elementów:
Czas tworzenia drzewa AVL
Czas tworzenia drzewa BST

Złożoność:

Złożoność czasowa operacji zależy od wysokości drzewa. W drzewie BST, wysokość drzewa może osiągnąć wartość O(n) dla danych posortowanych, co prowadzi do pesymistycznej złożoności czasowej O(n), gdzie n to liczba elementów w drzewie.

Drzewo AVL z kolei zapewnia zrównoważoną strukturę drzewa poprzez automatyczne wykonywanie rotacji, dzięki czemu gwarantuje, że wysokość drzewa jest zawsze ograniczona przez logarytmiczny rozmiar drzewa. Oznacza to, że złożoność czasowa operacji w drzewie AVL wynosi O(log n), co jest znacznie bardziej wydajne niż w przypadku drzewa BST.

Wnioski:

Na podstawie przedstawionych wyników można zauważyć, że dla danych posortowanych malejąco, czas tworzenia drzewa AVL jest znacznie krótszy niż czas tworzenia drzewa BST.

Dla mniejszych rozmiarów danych (N = 1000, 5000), różnica w czasie tworzenia obu drzew jest jeszcze niewielka, jednakże wraz ze wzrostem rozmiaru danych, czas tworzenia drzewa BST rośnie znacznie szybciej niż czas tworzenia drzewa AVL.

Można zauważyć, że dla N = 100000 czas tworzenia drzewa AVL wynosi 1,5840355, podczas gdy czas tworzenia drzewa BST wynosi aż 954,6185795. Różnica w czasie tworzenia drzewa AVL i BST jest tutaj ponad 600 razy większa.

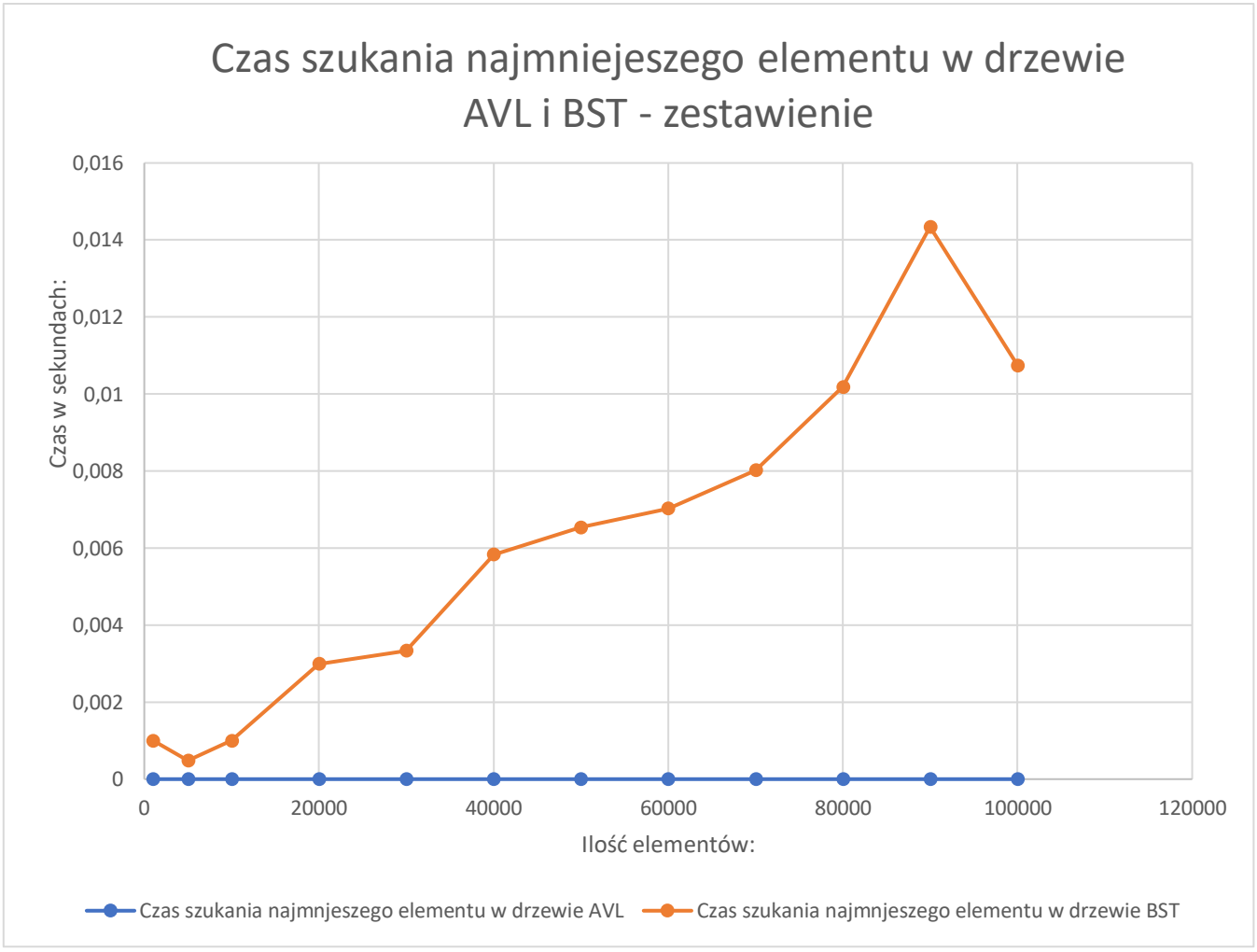
Podsumowując, dla danych posortowanych malejąco, drzewo AVL jest bardziej efektywne od drzewa BST, szczególnie dla większych rozmiarów danych.

Porównanie czasu szukania najmniejszego elementu w drzewie AVL i BST:

Wstep:

Wyszukiwanie najmniejszego elementu w drzewie BST i AVL polega na poruszaniu się po drzewie w kierunku lewego poddrzewa, aż do osiągnięcia liścia (węzeł, który nie ma żadnych dzieci). Ten liść jest najmniejszym elementem w drzewie.

Zestawienie czasu szukania najmniejszego elementu w drzewie AVL i BST:



Złożoność:

Złożoność czasowa wyszukiwania najmniejszego elementu w drzewie BST i AVL wynosi O(h), gdzie h to wysokość drzewa. W najgorszym przypadku, gdy drzewo jest niesymetryczne, wysokość drzewa wynosi n-1, gdzie n to liczba węzłów w drzewie. W takim przypadku złożoność czasowa wynosi O(n), jednakże w przypadku zbalansowanego drzewa AVL, złożoność wynosi O(log n), co oznacza, że czas potrzebny na znalezienie najmniejszego elementu rośnie logarytmicznie wraz z ilością węzłów w drzewie.

Wnioski:

Drzewa AVL są bardziej wydajne niż drzewa BST w przypadku operacji wyszukiwania, w tym operacji znajdowania najmniejszego elementu. Drzewa AVL zapewniają zrównoważoną strukturę, co oznacza, że wysokość drzewa jest utrzymywana na stałym poziomie. Dzięki temu czas potrzebny na wykonanie operacji wyszukiwania jest ograniczony i zależy tylko od logarytmicznej ilości węzłów w drzewie. W przypadku drzewa BST, gdzie struktura może być niesymetryczna, czas potrzebny na wykonanie operacji wyszukiwania może rosnąć liniowo wraz z ilością węzłów w drzewie.

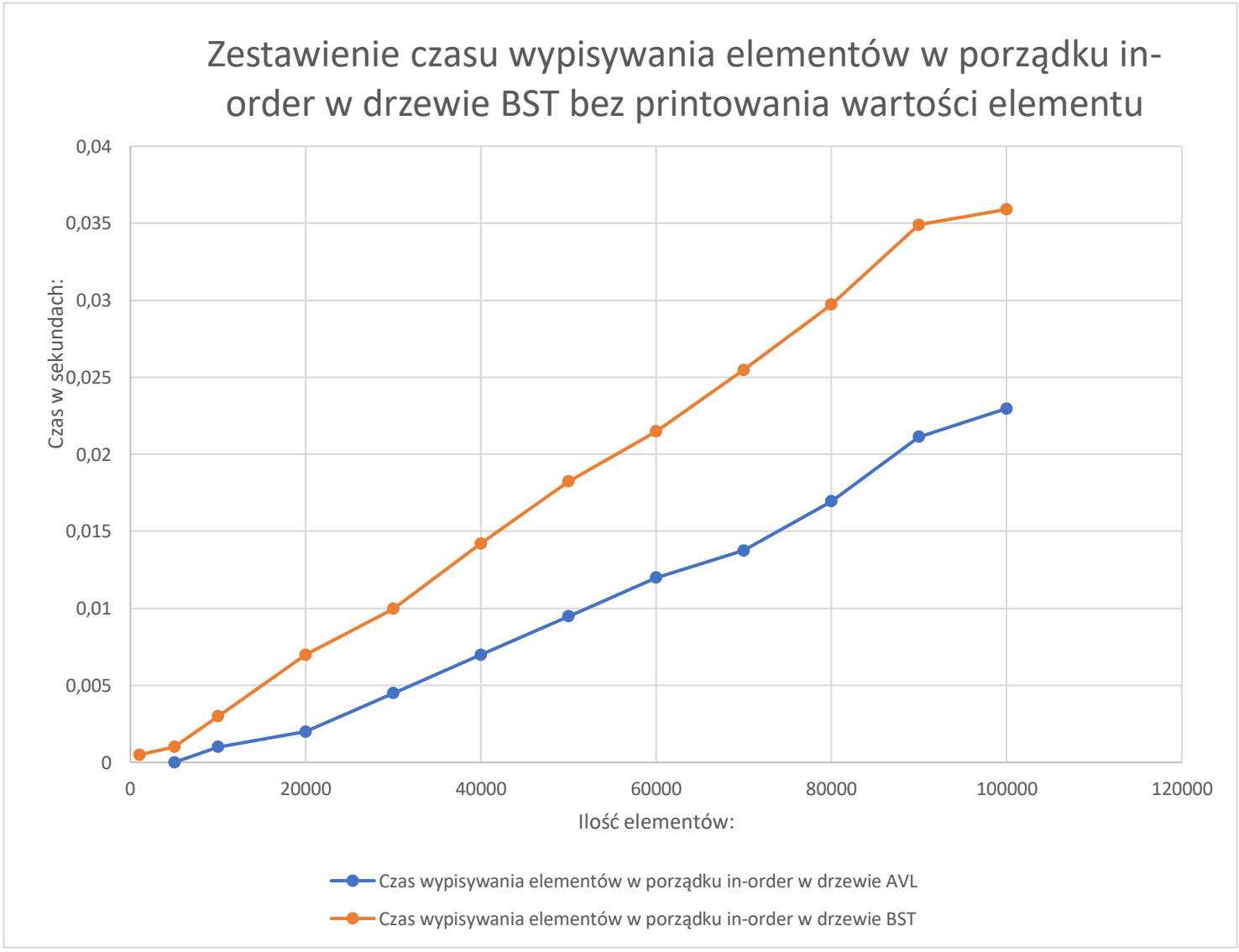
Porównanie czasu wypisywania elementów w porządku in-order w drzewie AVL i BST:

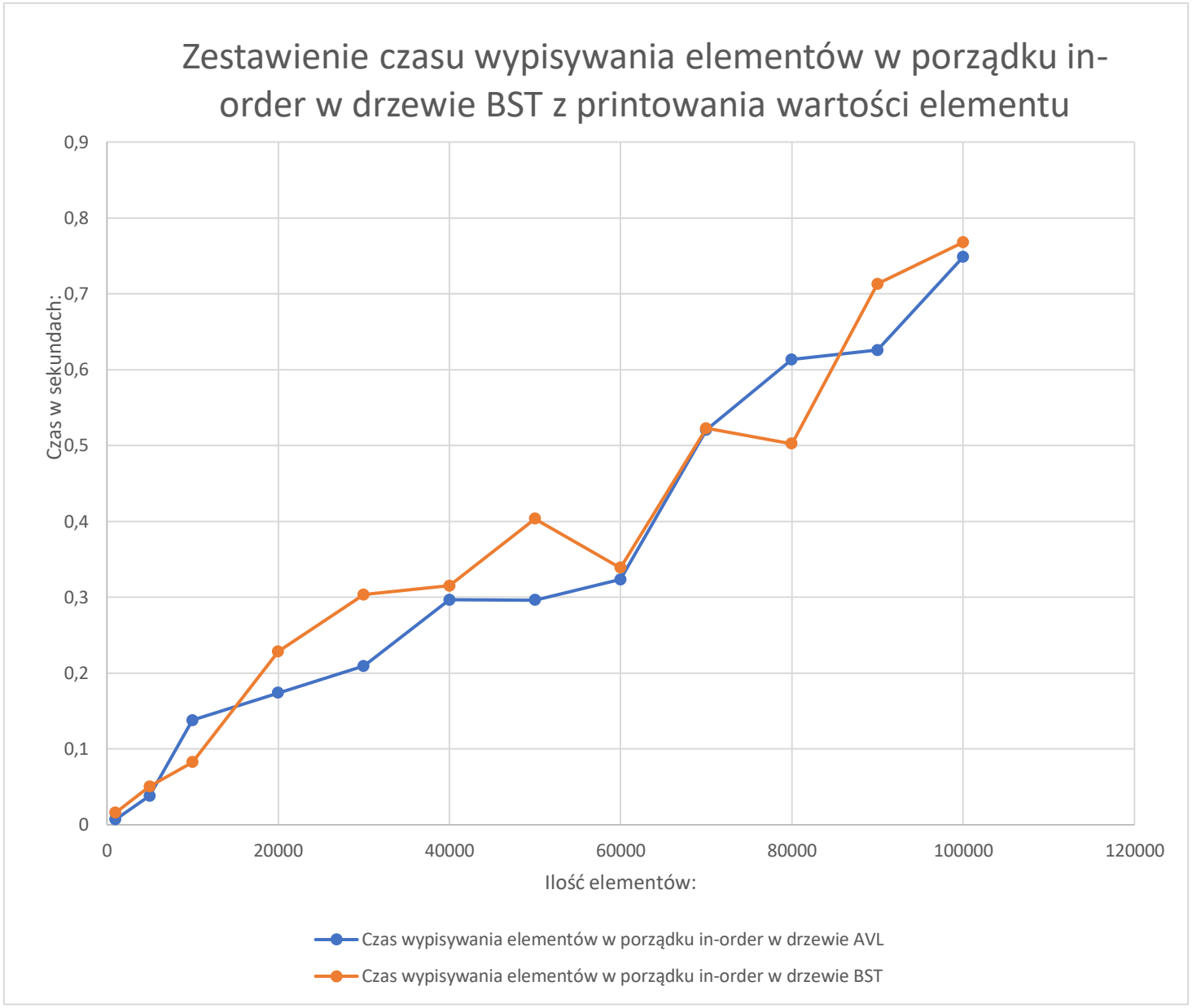
Wstep:

Przechodzenie drzewa binarnego w porządku in-order polega na odwiedzeniu węzła lewego poddrzewa, następnie samego węzła i na końcu węzła prawego poddrzewa.

Algorytm in-order dla drzewa binarnego działa rekurencyjnie. Dla każdego węzła, algorytm przechodzi do lewego poddrzewa, odwiedza węzeł, a następnie przechodzi do prawego poddrzewa. Dzięki temu, algorytm wypisuje elementy w porządku rosnącym, zaczynając od najmniejszego elementu, a kończąc na największym elemencie.

Zestawienie czasu wypisywania elementów w porządku in-order w drzewie AVL i BST:





Złożoność:

Złożoność czasowa algorytmu in-order dla drzewa binarnego wynosi O(n), gdzie n to liczba węzłów w drzewie. Algorytm musi odwiedzić każdy węzeł w drzewie dokładnie jeden raz, aby wypisać wszystkie elementy w porządku in-order.

W przypadku drzewa AVL, złożoność czasowa in-order jest taka sama jak dla drzewa BST, czyli O(n), ponieważ algorytm działa identycznie dla obu typów drzew.

W przypadku niesymetrycznego drzewa BST, złożoność czasowa in-order może wynieść nawet O(n^2), jeśli drzewo ma postać listy. W takim przypadku algorytm musi przejść przez każdy węzeł drzewa, co może zająć dużo czasu.

Podsumowując, złożoność czasowa in-order dla drzewa binarnego wynosi O(n), a drzewo AVL nie wprowadza żadnych zmian w tym względzie w porównaniu do drzewa BST. Jednak w przypadku

drzewa BST, który jest niesymetryczny, algorytm in-order może być bardziej czasochłonny niż dla drzewa AVL.

Wnioski:

Wniosek z porównania czasu wypisywania elementów w porządku in-order w drzewie AVL i BST jest taki, że dla obu typów drzew złożoność czasowa algorytmu in-order wynosi O(n).

Drzewa AVL i BST różnią się od siebie sposobem balansowania, a nie sposobem wypisywania elementów w porządku in-order. W obu przypadkach algorytm działa tak samo i wymaga odwiedzenia każdego węzła drzewa dokładnie jeden raz.

Należy jednak pamiętać, że w przypadku niesymetrycznego drzewa BST, wypisanie elementów w porządku in-order może zająć więcej czasu ze względu na konieczność odwiedzenia każdego węzła drzewa, co może prowadzić do złożoności czasowej O(n^2).

Podsumowując, zarówno drzewo AVL, jak i BST są efektywnymi strukturami danych do wypisywania elementów w porządku in-order, pod warunkiem, że drzewo BST jest zrównoważone, aby zapobiec nierównomiernemu rozmieszczeniu węzłów.

Równoważenie drzewa BST:

Wstep:

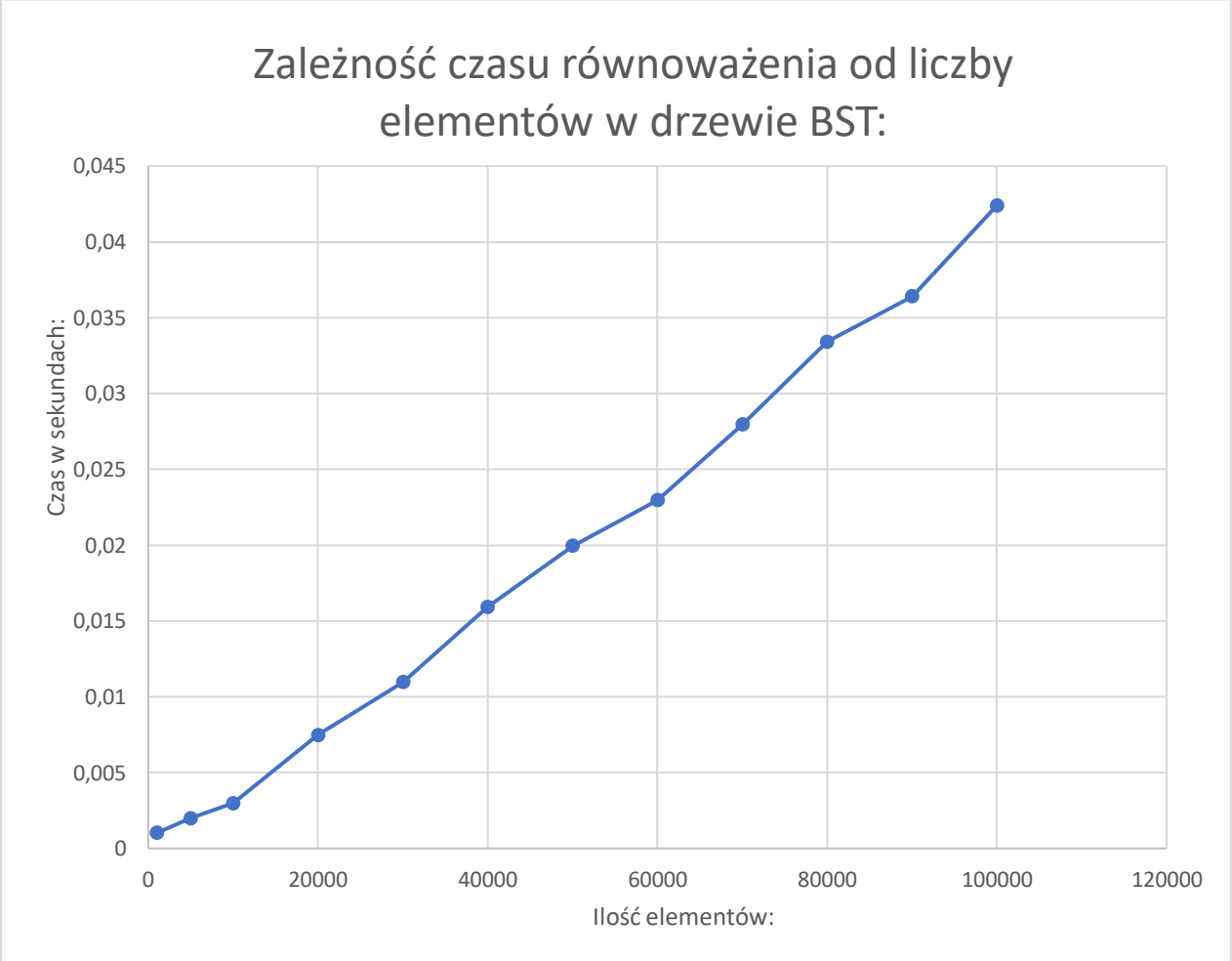
Algorytm DSW (Day-Stout-Warren) to algorytm, który służy do równoważenia drzewa BST (Binary Search Tree). Algorytm DSW działa na zasadzie rotacji węzłów, w celu uzyskania drzewa o jak najmniejszej wysokości.

Algorytm DSW składa się z dwóch etapów: przekształcenia drzewa w listę poprzez wykonanie rotacji węzłów, a następnie przekształcenia listy w drzewo za pomocą sekwencji rotacji.

W pierwszym etapie algorytm przekształca drzewo w listę, poprzez wykonanie rotacji węzłów w celu wyprostowania gałęzi. W tym procesie jedna z gałęzi jest przekształcana w węzły listy, a druga gałąź jest usuwana.

W drugim etapie algorytm przekształca listę w drzewo, poprzez sekwencję rotacji węzłów. Rotacje te są wykonywane w taki sposób, aby uzyskać drzewo o jak najmniejszej wysokości.

Krzywa zależności czasu równoważenia od liczby elementów w drzewie BST:



Złożoność:

Złożoność algorytmu DSW to O(n), gdzie n to liczba węzłów w drzewie BST. Oznacza to, że algorytm działa w czasie liniowym względem liczby węzłów w drzewie.

W pierwszym etapie algorytmu DSW, przekształcenie drzewa w listę, wykonuje się O(log n) rotacji, co daje łącznie O(n log n) operacji. W drugim etapie algorytmu, przekształcenie listy w drzewo, wykonuje się O(log n) rotacji dla każdego węzła na liście, co daje łącznie O(n) operacji.

Algorytm DSW ma złożoność czasową O(n), co czyni go bardzo efektywnym sposobem na uzyskanie zrównoważonego drzewa BST.

Wnioski:

Algorytm DSW to skuteczny sposób na uzyskanie zrównoważonego drzewa BST. Dzięki zastosowaniu algorytmu DSW, drzewo BST staje się bardziej wydajne i efektywne, co przyspiesza czas wyszukiwania, wstawiania i usuwania elementów.

W porównaniu z zwykłym drzewem BST, które może mieć drastycznie różne wysokości w zależności od kolejności wstawianych elementów, drzewo BST zrównoważone algorytmem DSW ma minimalną wysokość, co pozwala na szybsze operacje na drzewie.

Wnioski:

Na podstawie wyników badań przedstawionych w niniejszym sprawozdaniu można stwierdzić, że drzewa AVL są bardziej wydajne od drzew BST dla większych rozmiarów drzew. Wyniki pokazują, że mimo większej złożoności czasowej operacji wstawiania elementu do drzewa AVL w porównaniu do drzewa BST, czas wykonania innych operacji takich jak wyszukiwanie minimum czy wypisywanie elementów w kolejności in-order jest znacznie krótszy dla drzewa AVL.

Dodatkowo, wyniki pokazują, że czas równoważenia drzewa BST rośnie wraz ze wzrostem liczby elementów w drzewie. Dlatego ważne jest, aby regularnie wykonywać operację równoważenia, aby utrzymać wydajność struktury danych na odpowiednim poziomie.

Ostatecznie, wybór między drzewem AVL a BST powinien być dokładnie przemyślany, biorąc pod uwagę specyfikę danego zastosowania. Dla mniejszych rozmiarów drzew BST może być wystarczające i bardziej efektywne, jednak dla większych drzew warto rozważyć użycie drzewa AVL ze względu na jego wyższą wydajność.