Zadanie 2: Złożone struktury danych
Jakob Wolitzki 136830 I2

Wymagania:

Znajomość struktury listy jednokierunkowej i drzewa poszukiwań binarnych (ang. Binary Search Tree – BST) ze szczególnym uwzględnieniem:

- tworzenia oraz usuwania listy i drzewa,
- wyszukiwania elementu w liście i drzewie,
- usuwania elementu z listy i drzewa,
- przeglądania drzewa w 3 porządkach: wzdłużnym, poprzecznym i wstecznym,

definicji drzewa wyważonego i dokładnie wyważonego

Przebieg ćwiczenia:

1. Napisać program umożliwiający utworzenie spisu studentów. Program powinien umożliwiać pamiętanie danych na dynamicznej liście jednokierunkowej oraz w BST. Powinien on ponadto umożliwiać dopisywanie elementu do istniejącej struktury, poszukiwanie elementu oraz usuwanie struktury. Każdy element składa się z nazwiska, imienia i numeru indeksu. (12znaków Imię, 12znaków Nazwisko, 7 cyfr Nr Indeksu)

2. Przygotować zbiory danych do testowania szybkości wykonywania poszczególnych operacji na badanych strukturach danych. Zbiór taki powinien zawierać *n* rekordów (nazwisko, imię, nr indeksu), przy czym tylko nr indeksu będzie traktowany jako klucz przy operacjach wykonywanych na strukturach danych (Dane do tablicy wejściowej wczytywane z pliku tekstowego!).

3. Zbadać czas potrzebny na zapisanie do rozważanych struktur zbiorów danych o różnych rozmiarach *n* oraz usunięcie wszystkich elementów (usuwając element po elemencie losując poszczególne elementy). Ponadto zbadać czas niezbędny do wyszukania w każdej ze struktur wybranego elementu (mierząc czas średni wyszukiwania kolejno wszystkich elementów). W tym przypadku należy badać listę posortowaną, drzewo BST i dokładnie wyważone (BBST). Poszukiwany element powinien być wybrany losowo.

Wyniki eksperymentów należy przedstawić w tabelach oraz na wykresach:

1) zależności czasu wstawiania elementów od ich liczby - wspólny dla danych struktur,
2) poszukiwania wybranego elementu - wspólny dla danych struktur.

4. Przedstaw wioski z przeprowadzonych pomiarów.

Sprawozdanie zacznę od opisania poszczególnych struktur danych.

Lista jednokierunkowa jest to struktura dynamiczna o zmieniającej się dynamicznie wielkości. Każdy element listy zawiera jakieś dane, w naszym przypadku są to structy przedstawiające studentów z imieniem, nazwiskiem i numerem indeksu. Każdy element oprócz tych informacji, również wskazuję na następny, a ostatni zawsze wskazuje na wartość NULL w pamięci. Warto zwrócić uwagę, że listę jednokierunkową można przeglądać tylko w jednym kierunku, gdyż zawsze posiadamy adres głowy listy, co zostanie dokładniej opisane i przedstawione na wykresach.

Drzewo poszukiwań binarnych jest drzewem binarnym, w którym każdy węzeł musi spełniać określone warunki. Wszystkie węzły w lewym poddrzewie mają wartości mniejsze lub równe od wartości danego węzła oraz wszystkie węzły w prawym poddrzewie są większe lub równe od danego węzła. Podobnie jak w liście, każdy z węzłów w naszym drzewie jest structem zawierającym dane studentów, a budowanie drzewa odbywa się dzięki porównywaniu numerów indeksów studentów.

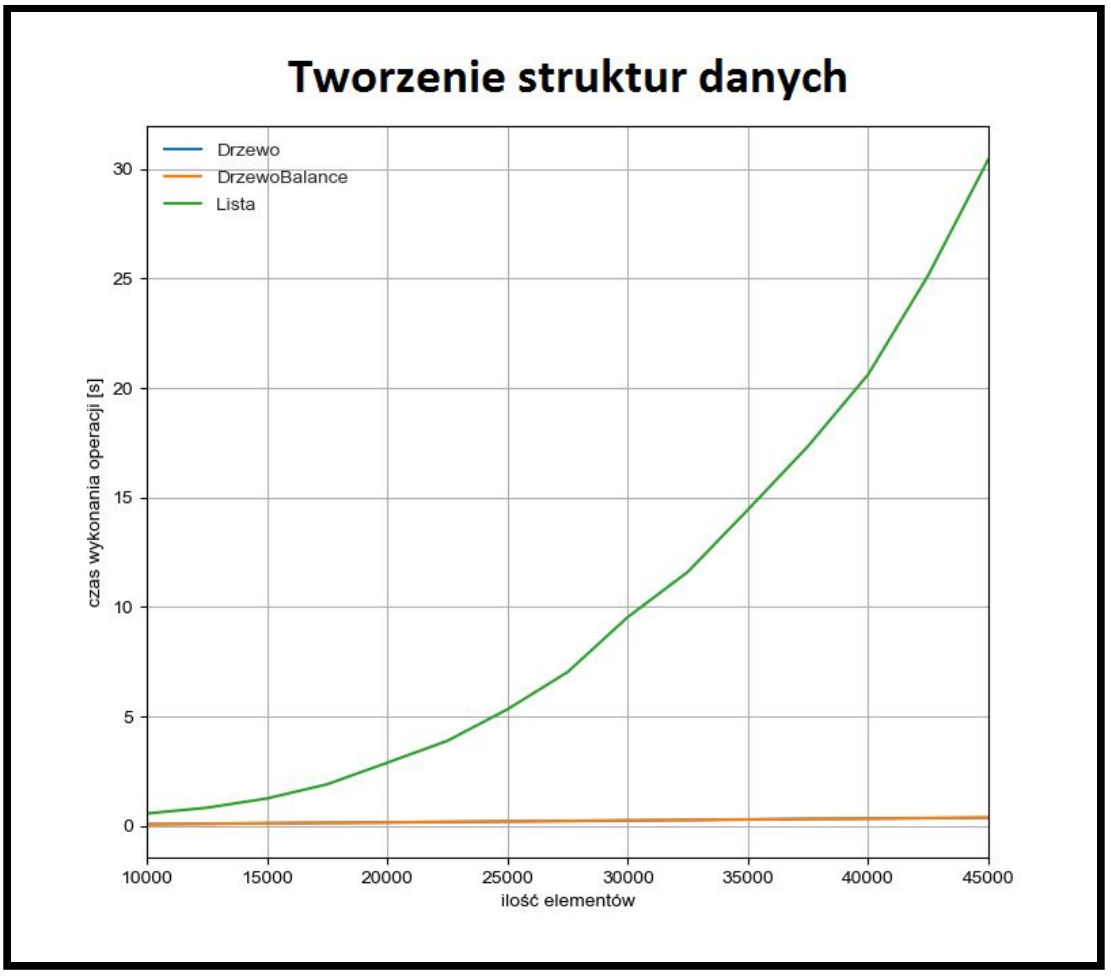
Szczególnym przykładem drzewa binarnego, jest **dokładnie wyważone drzewo binarne**. Różni się ono budową, gdyż jego wysokość jest najmniejsza możliwa dla danych wejściowych. Drzewo dzięki przeprowadzonym rotacją, najpierw w prawo, a następnie w lewo, sprowadza się do wyważonej struktury danych, w której ilość danych w lewym poddrzewie jest równa lub bardzo zbliżona do ilości w prawym.

Na wstępie chciałbym również napisać, iż wykresy opracowane na podstawie wyników z przeprowadzonych badań nie odpowiadają w pełni złożonością obliczeniowym algorytmów. Dzieje się tak, gdyż wyniki mogą być zakłócone przez takie czynniki jak działanie funkcji losujących w C++ powodujących niedokładnie jednorodny rozkład danych losowych. Powoduje to drobne odchylenia na wykresach.

Teraz przedstawię wyniki eksperymentów.

Pierwszym typem badanych danych jest wstawianie elementów do struktur danych.

Tworzenie struktur danych [s]			
Ilość elementów	Lista jednokierunkowa	Drzewo BST	Drzewo BBST
10000	0.57442	0.08366	0.08746
12500	0.83713	0.10336	0.10482
15000	1.26162	0.13019	0.12383
17500	1.90762	0.15007	0.14547
20000	2.89304	0.16541	0.16660
22500	3.89256	0.18856	0.19069
25000	5.33250	0.21247	0.21125
27500	7.03246	0.23209	0.23469
30000	9.54086	0.25575	0.25621
32500	11.60818	0.27759	0.27996
35000	14.44621	0.30093	0.30396
37500	17.34855	0.32919	0.31964
40000	20.62435	0.34088	0.34091
42500	25.15558	0.36258	0.36512
45000	30.45236	0.38420	0.38327



Przy opisywaniu powyższego zestawu danych, na pierwszy rzut oka widzimy niemalże dokładnie wyniki przy tworzeniu drzewa BST i drzewa BBST. Dzieje się tak, dlatego iż operacja wyważania jest wykonywana tylko raz, a jej szybkość nie wpływa znacząco na wydłużenie czasu wykonania struktury.
Porównując czasy działania operacji tworzenia listy jednokierunkowej i drzew, możemy stwierdzić, że tworzenie listy trwa znacznie dłużej.

Tworzenie listy jednokierunkowej posortowanej w najgorszym przypadku jest bardzo nieoptymalne, gdyż przy każdym wstawianiu nowego elementu należy przejść przez całą listę i wstawić element na jej końcu. Takie zjawisko zostało przedstawione w tym przykładzie, gdzie łączna suma porównań wynosi 1+2+3+..+n-2+n-1. Złożoność funkcji wstawiającej wynosi O(1), jest stała, jednakże, czas na dodanie elementu na sam koniec listy wynosi O(n), za czym idzie złożoność obliczeniowa $O(n^2)$, gdy weźmiemy pod uwagę wszystkie elementy.

Najbardziej optymalne byłoby dodawanie elementu na sam początek listy, gdyż zawsze mamy wskaźnik na jej początek, wtedy złożoność obliczeniowa tworzenia listy wynosiłaby O(n). Jest to jednak wyjątkowy przypadek danych, gdzie dane są posortowane od największego do najmniejszego elementu.

Ogólnie zaleca się tworzyć listę dla mniejszych ilości danych, gdzie n nie przekracza 300 elementów.

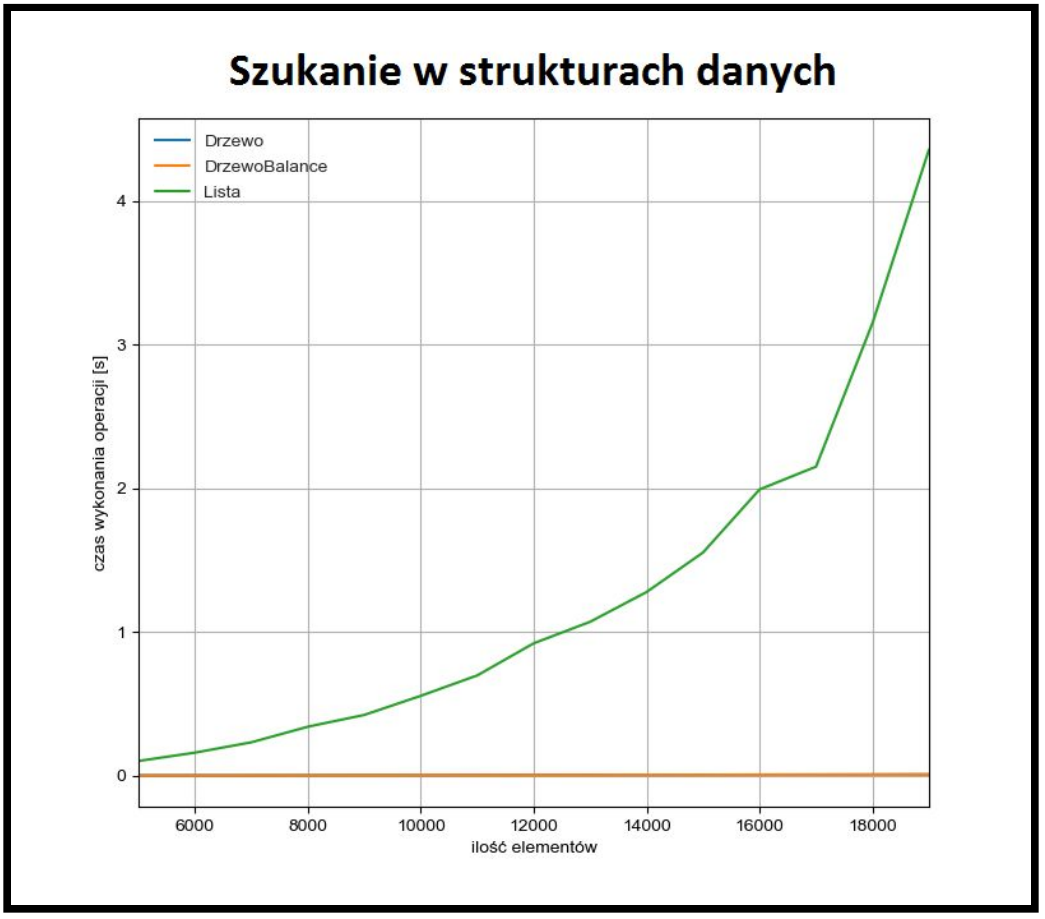
Dla losowego rozkładu danych, bardzo szybkie i optymalne okazało się być drzewo BST i BBST. Ich struktura i operacje porównania bardzo sprzyjają danym losowym, które są rozłożone jednorodnie.

Najgorszym zestawem danych dla drzewa BST jest ciąg danych posortowanych. W tym przypadku drzewo BST zachowuje się dokładnie tak samo jak lista. Gdyż elementy (w zależności czy dane są malejące czy rosnące) układają się jak schodki w jedną stronę. W takim przypadku szukanie elementu ma złożoność O(n), a stworzenie takiej struktury $O(n^2)$ tak samo jak w przypadku listy jednokierunkowej.

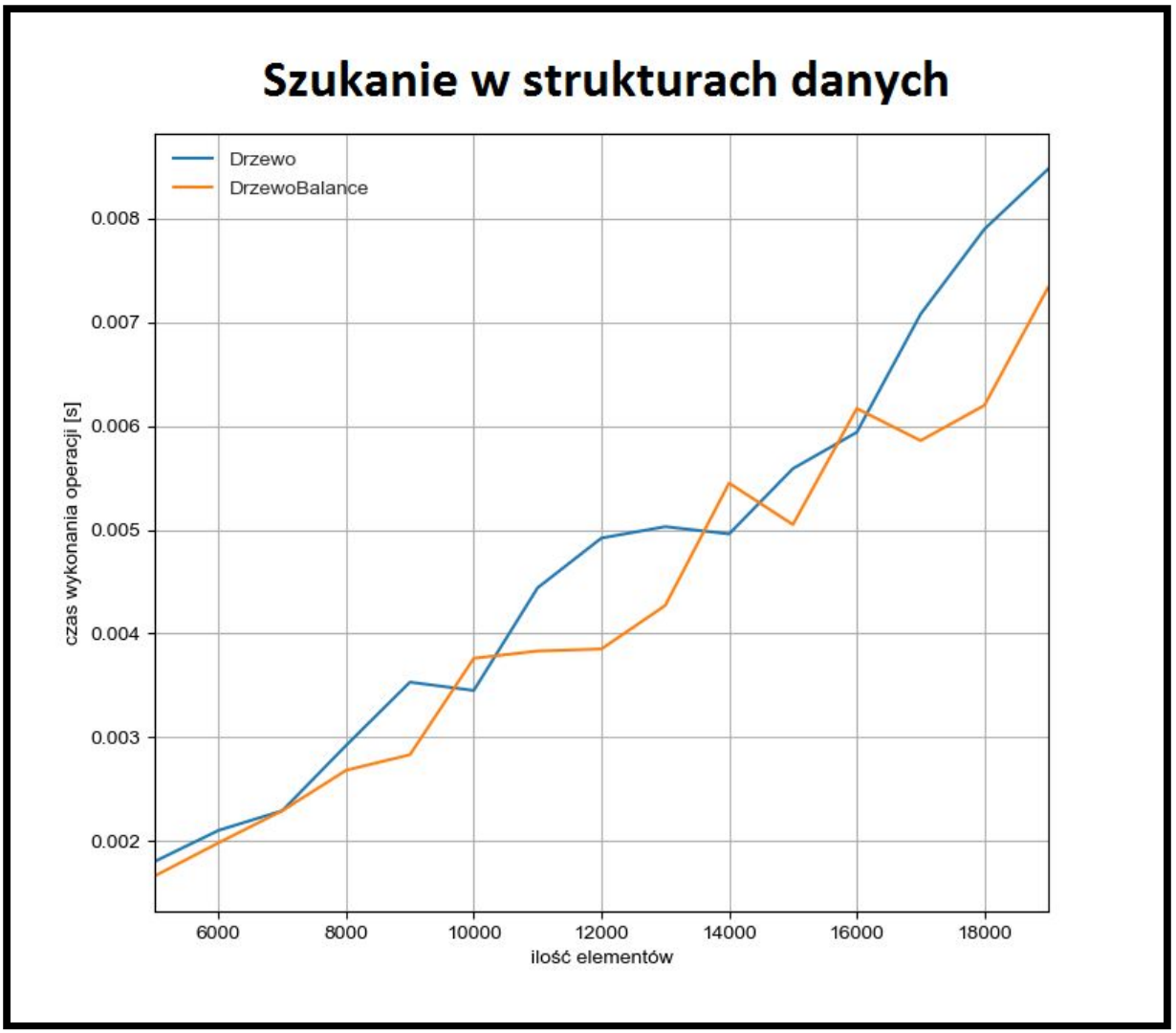
Podsumowanie:
Po analizie poszczególnych krzywych, można łatwo stwierdzić, iż występuje znacząca różnica między szybkością tworzenia listy a drzewa BST i BBST. Widzimy różnice pomiędzy złożonością kwadratową, a logarytmiczną. Struktury drzewiaste wygrywają z listami jednokierunkowymi.

Kolejnym typem badanych danych jest wyszukiwanie elementów w strukturach danych.

Szukanie w strukturach danych [s]			
Ilość elementów	Lista jednokierunkowa	Drzewo BST	Drzewo BBST
5000	0.10178	0.00180	0.00166
6000	0.15979	0.00210	0.00198
7000	0.23184	0.00229	0.00229
8000	0.34032	0.00292	0.00268
9000	0.42231	0.00353	0.00283
10000	0.55526	0.00345	0.00376
11000	0.69734	0.00444	0.00383
12000	0.92091	0.00492	0.00385
13000	1.07062	0.00503	0.00427
14000	1.27789	0.00496	0.00545
15000	1.55331	0.00559	0.00505
16000	1.99166	0.00594	0.00617
17000	2.15021	0.00708	0.00586
18000	3.14966	0.00790	0.00620
19000	4.35686	0.00848	0.00734



Na osobnym wykresie przedstawię wyniki wyszukiwania elementów drzewa BST oraz BBST, aby dokładniej zobrazować różnicę pomiędzy nimi.



Z powyższych wykresów, widzimy, że najdłużej szukamy elementów w liście jednokierunkowej. W najgorszym przypadku, należy przejść przez całą listę w celu znalezienie elementu, czyli złożoność obliczeniowa wynosi O(n).

Wyszukanie pojedynczego elementu w drzewach jest znacznie szybsze. W drzewie BST, w najgorszym przypadku, wyszukanie elementu jest rzędu O(n), jednakże tylko w momencie gdy drzewo byłoby zbudowane z posortowanych danych wejściowych. W średnim przypadku, czyli dla danych losowych rozłożonych równomiernie, szukanie elementu wykonuje znacznie mniej operacji niż n.

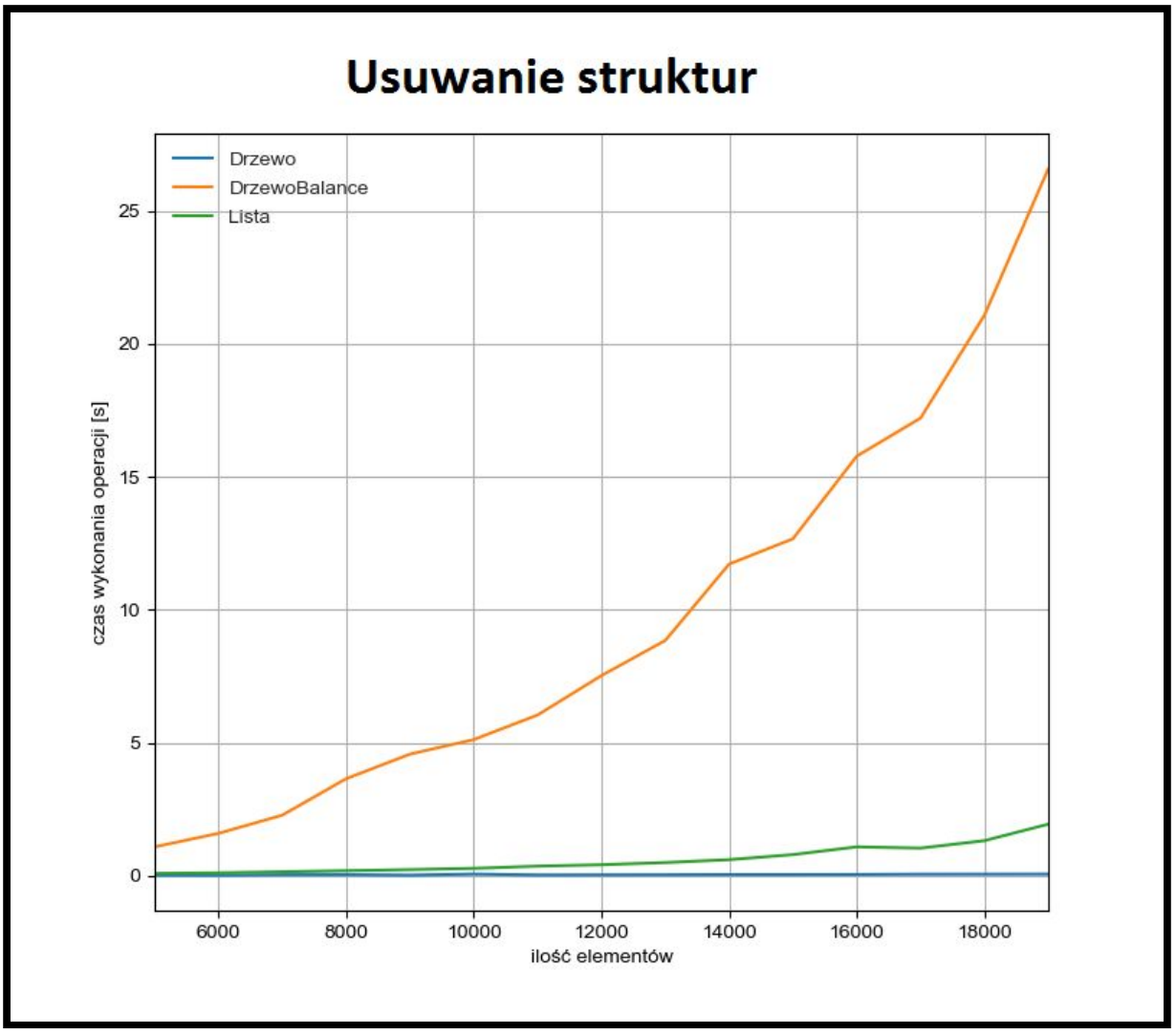
Szukanie elementu w drzewie BBST jest jeszcze szybsze niż w BST. Wynika to z faktu iż w drzewie BBST różnica liczby węzłów lewego i prawego poddrzewa wynosi maksymalnie 1 dla każdego węzła. Wysokość takiego drzewa jest mniejsza niż w drzewie BST i wynosi $\log_2 n$, za czym idzie znacznie mniejsza ilość porównań w celu znalezienia elementu.

Z powyższego wykresu, widzimy, że opłaca się równoważyć drzewo, gdy chcemy znaleźć jakiś element.

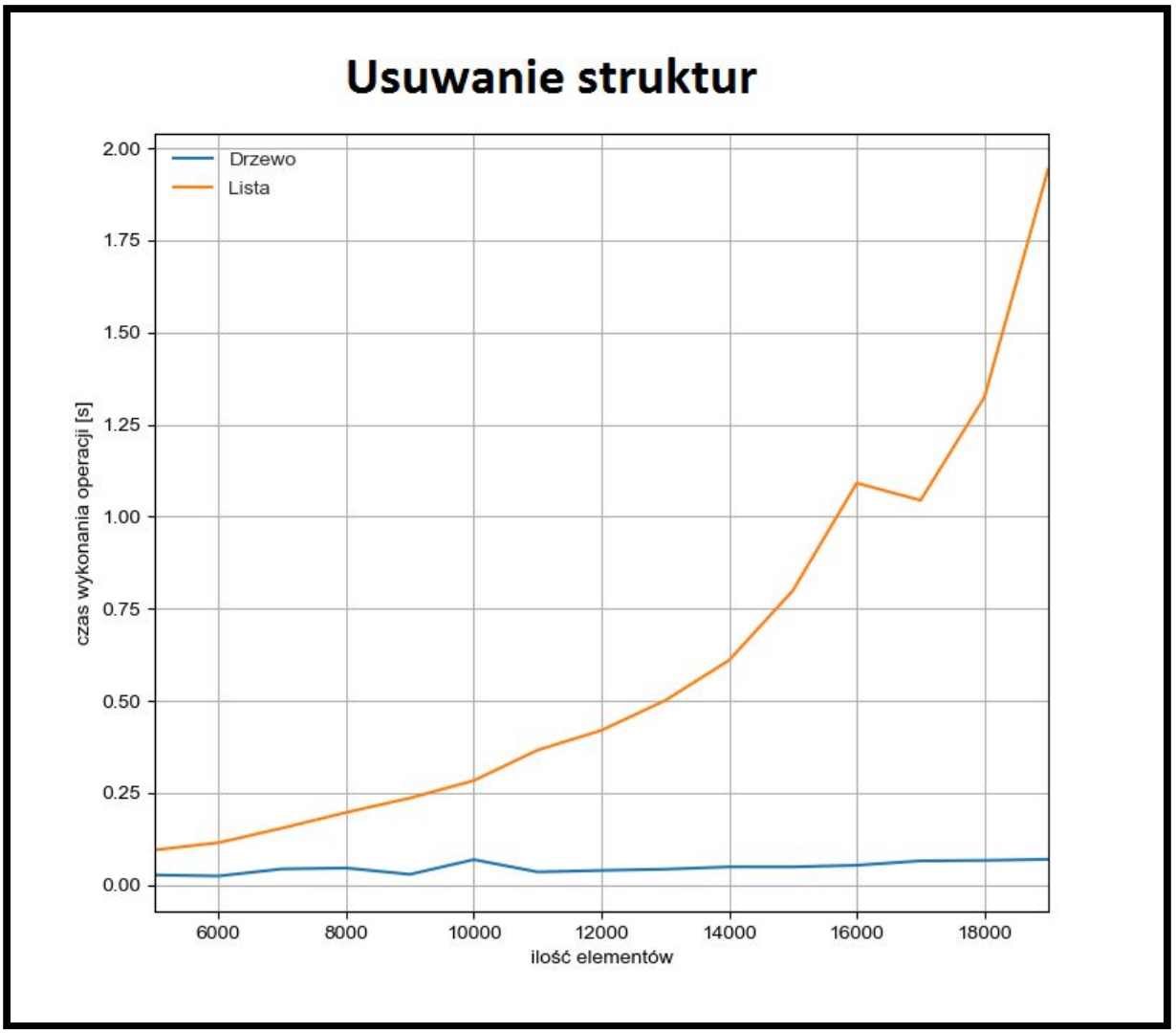
Podsumowanie:
W drzewie BST i BBST znacznie szybciej możemy znaleźć dany element niż w liście jednokierunkowej. Widzimy, że struktury drzewiaste oparte na zasadzie metody “Dziel i zwyciężaj” wykonują znacznie mniej operacji w celu znalezienia elementu. Jednakże czas poszukiwania elementów w drzewie BST może znacznie zależeć od danych wejściowych, dlatego opłaca się takie drzewo równoważyć.

Ostatnim typem badanych danych jest usuwanie elementów w strukturach danych.

Usuwanie elementów w stukturach danych [s]			
Ilość elementów	Lista jednokierunkowa	Drzewo BST	Drzewo BBST
5000	0.09531	0.02739	1.09125
6000	0.11511	0.02467	1.59784
7000	0.15477	0.04386	2.28251
8000	0.19726	0.04646	3.65253
9000	0.23626	0.02920	4.57731
10000	0.28378	0.06921	5.12453
11000	0.36646	0.03579	6.04754
12000	0.42018	0.03978	7.52841
13000	0.50125	0.04321	8.84971
14000	0.61038	0.04946	11.71945
15000	0.79980	0.04939	12.67132
16000	1.09135	0.05391	15.77680
17000	1.04457	0.06577	17.20863
18000	1.32429	0.06702	21.07486
19000	1.94384	0.07006	26.57668



Na samym początku można zauważyć, że usuwanie drzewa BBST zajmuje najwięcej czasu z wszystkich struktur. Dzieje się tak, gdyż po każdym usunięciu elementu, drzewo należy dokładnie zrównoważyć i przywrócić jego strukturę. Dlatego na osobnym wykresie przedstawię różnice pomiędzy drzewem BST oraz listą jednokierunkową.



Usuwanie przeprowadzamy, wybierając losowy element ze struktury, który następnie usuwamy. Jak widzimy, usuwanie elementów w drzewie przebiega znacznie szybciej niż usuwanie ich z listy jednokierunkowej. Wynika to z czasu potrzebnego na odnalezienie elementu. Z faktu, iż w drzewach wykonujemy znacznie mniej operacji w celu znalezienia elementu, cały proces przebiega szybciej niż ma to miejsce w liście jednokierunkowej. Operacja usunięcia elementu w drzewach jak i w liście ma stałą złożoność, zatem nie wpływa ona na złożoność procesu usuwania całej struktury.

Podczas usuwania elementu ze środka drzewa, algorytm odpowiednio je odbudowywuje. Znalezienie elementu, który zastąpi nam korzeń ma złożoność O(logn), a samo usunięcie ma stałą złożoność obliczeniową, zatem nie wpływa to na całkowity proces usuwania elementu.