POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: ELEKTRONIKA
SPECJALNOŚĆ: ZASTOSOWANIE INŻYNIERII KOMPUTEROWEJ W TECHNICE

EFEKTYWNOŚĆ PROCEDUR OBLICZENIOWYCH

ZADANIE LABORATORYJNE NR 2

Badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w drzewiastych strukturach danych takich jak drzewo BST, drzewo Czerwono-Czarne oraz kopiec binarny.

AUTOR:
Michał Musiał

Prowadzący:
Dr inż. Tomasz Kapłon

WROCŁAW 2017

Spis treści

1. Cel ćwiczenia

Celem ćwiczenia było przeprowadzenie badań na strukturach drzewiastych polegających na pomiarze czasu wykonywania operacji wstawiania, usuwania oraz wyszukiwania elementów w takich strukturach jak:

- Drzewo BST
- Kopiec binarny
- Drzewo Czerwono-czarne

2. Plan eksperymentu

Do przetestowania struktur wylosowanych zostało 9 zestawów danych, zgodnie z ich rozmiarem oraz wielkością liczb podanych w instrukcji. Zostały oznaczone według schematu:
l**1m**.txt

Cyfra określa ilość danych, czyli dla 1 jest to 50 elementów, 2 – 1/2Max, 3 – Max liczba elementów.

Znak m, s lub d oznacza z jakiego zakresu zawierają liczby. Odpowiednio: m oznacza najmniejsze liczby z zakresu -100 do 100, s oznacza liczby z zakresu -1/2Max do 1/2Max rozmiaru, d oznacza liczby z maksymalnego zakresu podanego w zadaniu.

Jako maksymalny dostępny rozmiar pamięci przyjęto 2000000 bajtów, czyli największy plik będzie zawierał 500 000 elementów oraz liczby z zakresu -2000000 do 2000000.

Przyjąłem następujący algorytm przeprowadzania pomiarów:

1. Wybierz strukturę

2. Wczytaj zestaw danych

3. Wykonaj żądaną operację.

4. Odczytaj wynik pomiaru czasu, dane zapisz do tabeli.

5. Usuń strukturę

6. Powtórz

Dla każdej operacji, powyżej podany schemat powtarzałem pięciokrotnie, a następnie wyznaczałem średni czas z wykonanych pomiarów. Operacje pomiaru czasu zautomatyzowałem poprzez zastosowanie pętli *for.* Operacje dodawania, usuwania, wyszukiwania wykonują się w tejże pętli pięciokrotnie, wg. przedstawionego algorytmu uwzględniając czas równoważenia dla struktur z algorytmem DSW.

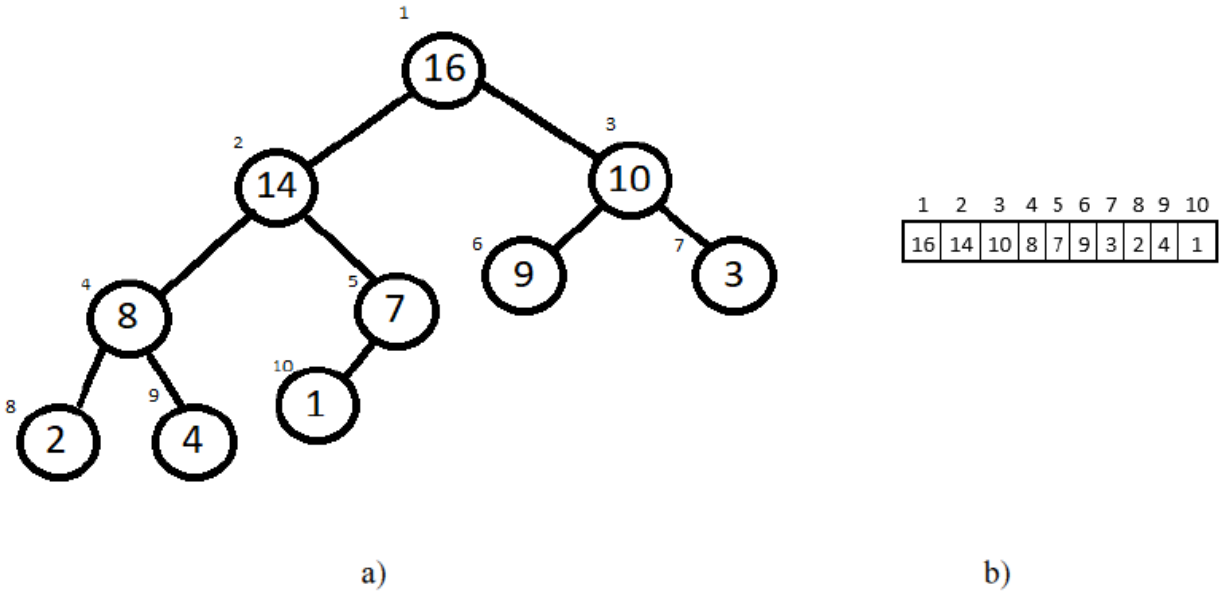
Mając do przebadania 9 zestawów danych, na każdym z nich do wykonania 3 operacje, powtórzone każdorazowo 5 razy daje to łączną liczbę 135 pomiarów dla każdej struktury.

Dla każdej struktury zachowałem ten sam plan eksperymentu.

3. Kopiec binarny

3.1 Wstęp teoretyczny

Kopiec jest to tablicowa struktura danych mająca duże znaczenie w algorytmice, jako że jest podstawą dla bardziej skomplikowanych obiektów. Kopiec posiada bardzo przydatną własność – jego największy (lub najmniejszy) element znajduje się zawsze w korzeniu drzewa dzięki czemu odczytanie największej (lub najmniejszej) wartości jest wykonywalne w jednym kroku.
Kopiec (binarny) jest strukturą danych tablicową, którą rozpatrywać można jako pełne drzewo binarne (Rys.1), co oznacza, że posiada zapełnione wszystkie poziomy za wyjątkiem ostatniego, a ostatni jest zapełniany bez przerw, poczynając od strony lewej do prawej. Każdy węzeł drzewa binarnego odpowiada elementowi tablicy, w którym jest podana wartość węzła. Tablica A reprezentująca kopiec ma dwa atrybuty: length[A], który określa liczbę elementów tablicy oraz heap-size[A], który określa liczbę elementów kopca znajdujących się w tablicy. Korzeniem drzewa jest A[1], a posiadając indeks i- węzła, można obliczyć indeksy ojca, syna lewego i prawego. Korzeń drzewa przechowywany jest zawsze w pierwszej komórce tablicy.



Rysunek 1 Kopiec można rozpatrywać jako (a) drzewo binarne i (b) jako tablicę [1]

3.2 Działanie programu

W zadaniu laboratoryjnym działanie kopca binarnego oparte jest o tablicę.

Pomiar czasu jest realizowany za pomocą funkcji podanej w zadaniu.

Przykład jej użycia w programie:

```
performanceCountStart = startTimer();//zaczynamy pomiar

kopiec->usun(250000);

performanceCountEnd = endTimer(); //zapamiętujemy koniec czasu

tm = performanceCountEnd.QuadPart - performanceCountStart.QuadPart;//Wyznacznie różnicy Koniec-Początek

cout << endl << "Time:" << tm / freq.QuadPart * 1000 << endl;//Przeskalowanie wyniku do ms.
```

Wynik pomiaru musimy podzielić przez częstotliwość taktowania naszego procesora, a następnie pomnożyć wyniki razy 1000, aby wynik był prezentowany w milisekundach. Dla każdej kolejnej badanej struktury korzystałem z tej samej metody pomiaru czasu.

3.3 Badane operacje

Dodawanie elementu:

Dodawany element wstawiamy jako ostatni liść kopca. Następnie sprawdzamy kolejno, czy jest on mniejszy lub równy swojemu rodzicowi. Jeśli nie, to zamieniamy wstawiony element z jego rodzicem. Operację kontynuujemy, aż znajdziemy rodzica większego lub równego elementowi lub dojdziemy do korzenia drzewa. Operacja dodawania nowego elementu do kopca posiada klasę złożoności obliczeniowej **O(log n)**

Usuwanie elementu:

Usuwamy korzeń drzewa, na miejsce którego wstawia się ostatni z liści. Następnie poprzez ruch kolejnymi poziomami w dół drzewa, sprawdzany zostaje warunek kopca. Jeśli nie jest spełniony to zamienia się ojca z największym z synów. Operację kontynuuje się do czasu w którym spełnią się warunki kopca lub osiągnięcia liścia. Operacja usuwania elementu z kopca ma klasę złożoności obliczeniowej równą **O(log n)**

Wyszukiwanie Elementu:

Przy wyszukiwaniu elementu algorytm musi przeszukać całą strukturę, więc czas tej operacji będzie zależał od liczby elementów znajdujących się w kopcu, złożoność takiego algorytmu będzie zatem **O(n).**
Wyszukiwanie jest takie same jak w przypadku zwykłych tablic i zależy od liczby elementów, które przechowuje struktura.

3.4 Pomiary

Dodawanie:

DODAWANIE				
Lp.	Liczba elementów	Wielkość liczb	Średnia Pomiaru czasu [ms]	Średnia OLE [ms]
1	50	M	0,00951686	0,009610163
2	50	S	0,009610168	
3	50	D	0,009703462	
4	250 000	M	0,010263292	0,01044543
5	250 000	S	0,0105298	
6	250 000	D	0,0105432	
7	500 000	M	0,010543192	0,010605397
8	500 000	S	0,0106365	
9	500 000	D	0,0106365	

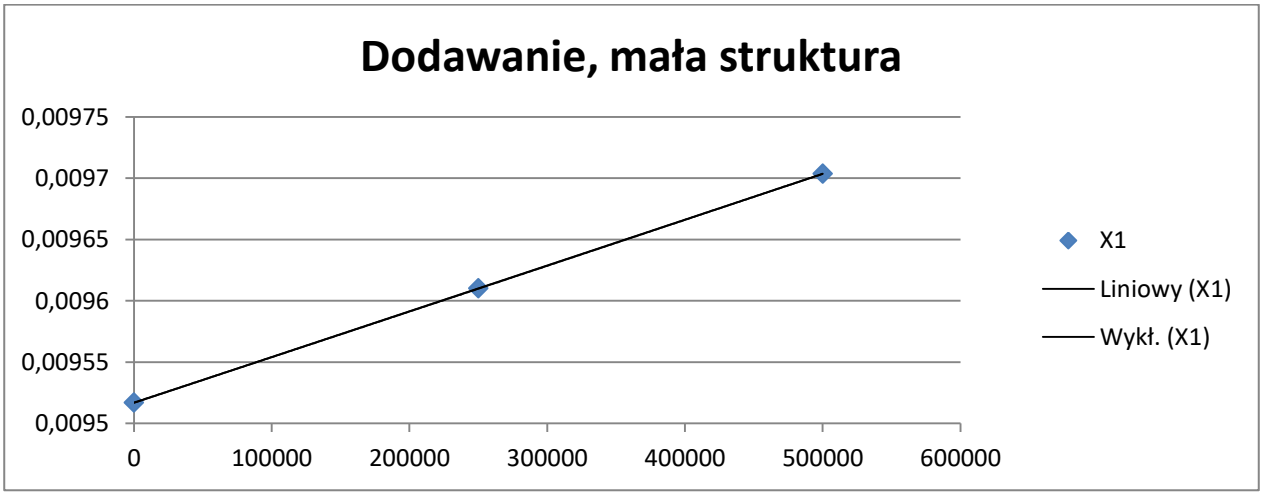
Tabela 1 Dodawanie do kopca

M-Liczby -100 do 100

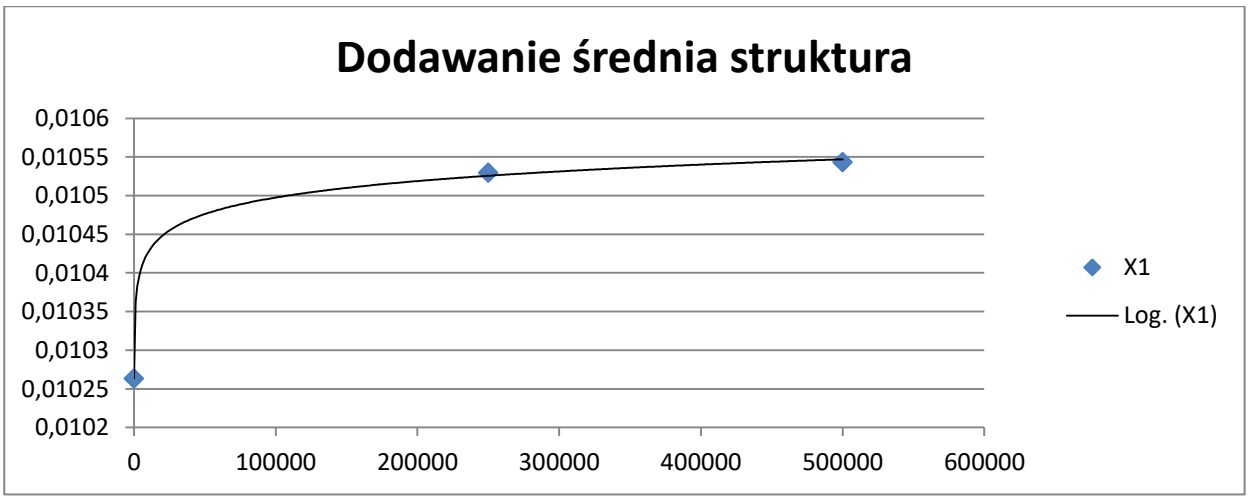
S- Liczby -1 000 000 do 1 000 000

D- Liczby -2 000 000 do 2 000 000

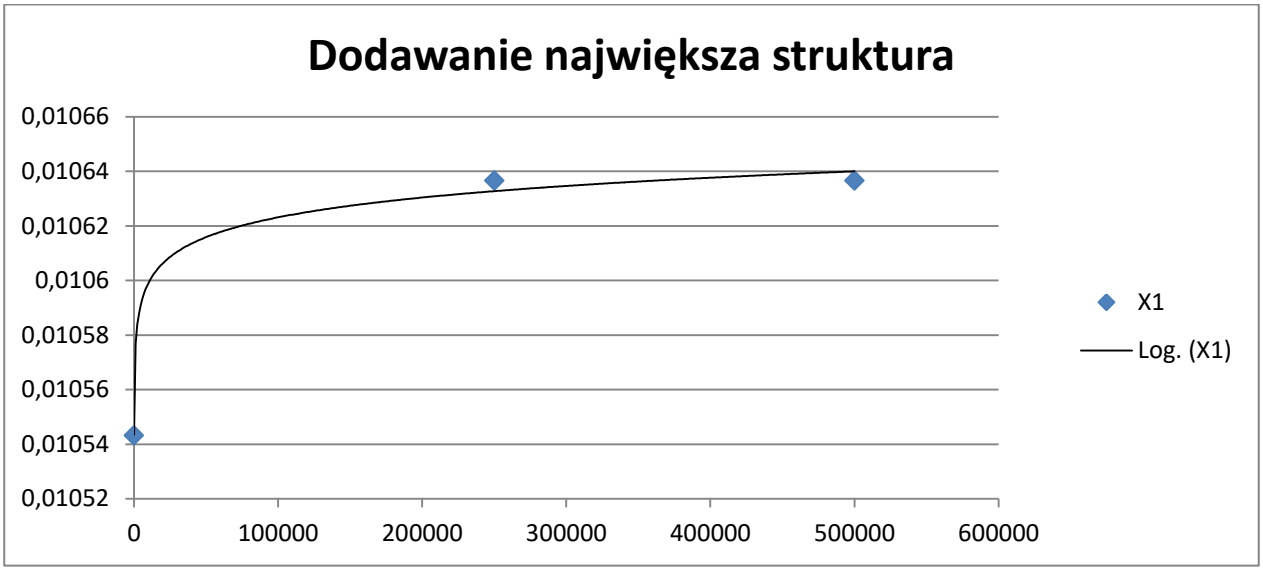
Średnia OLE- Średnia Od Liczby Elementów



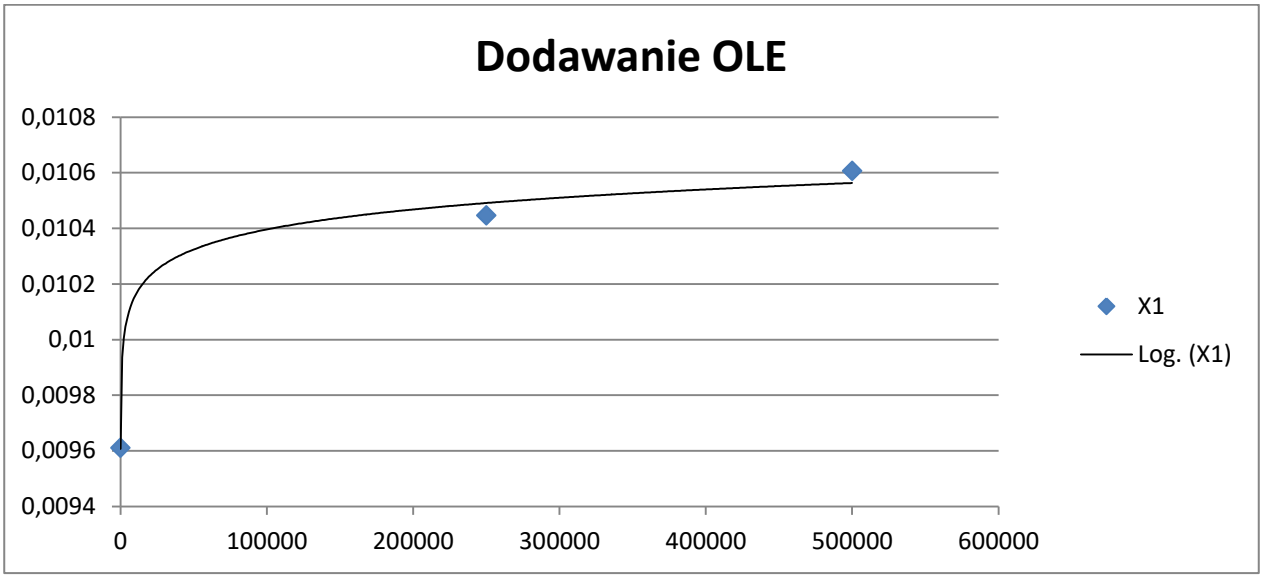
Wykres 1 Dodawanie do kopca binarnego dla liczby elementów: 50



Wykres 2 Dodawanie do kopca binarnego dla liczby elementów: 250 000



Wykres 3 Dodawanie do kopca binarnego dla liczby elementów: 500 000



Wykres 4 Dodawanie do kopca binarnego - wykres średniej od liczby elementów

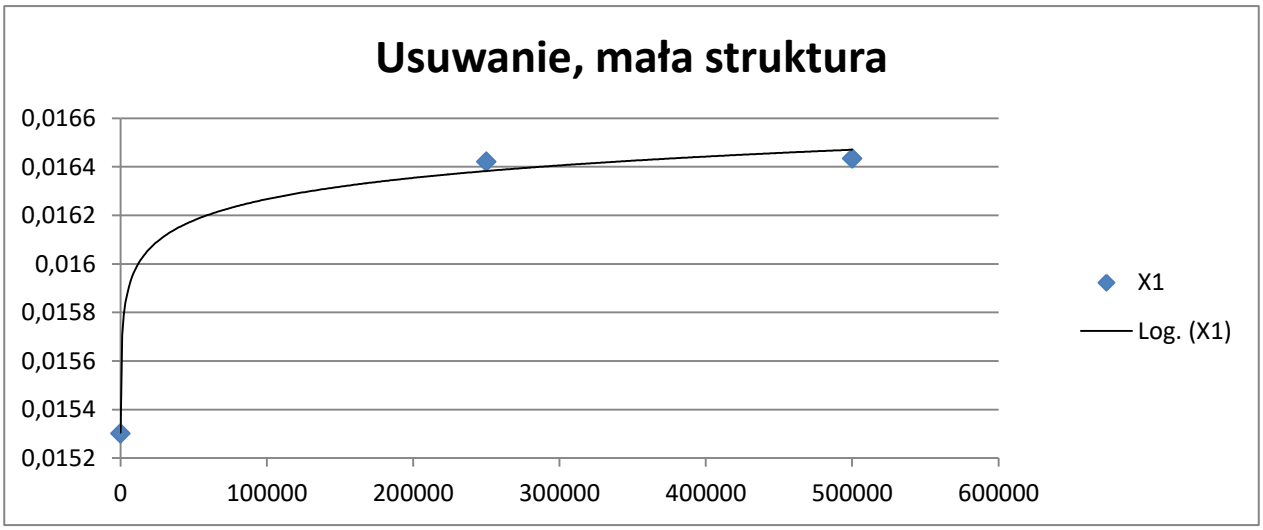
Usuwanie:

Usuwanie				
lp.	Liczba elementów	Wielkość liczb	Średnia czasu	Średnia OLE
1	50	M	0,01530162	0,0160525
2	50	S	0,01642124	
3	50	D	0,01643464	
4	250 000	M	3,037648	3,51664
5	250 000	S	3,769518	
6	250 000	D	3,742754	
7	500 000	M	6,469142	6,713547
8	500 000	S	6,812644	
9	500 000	D	6,858854	

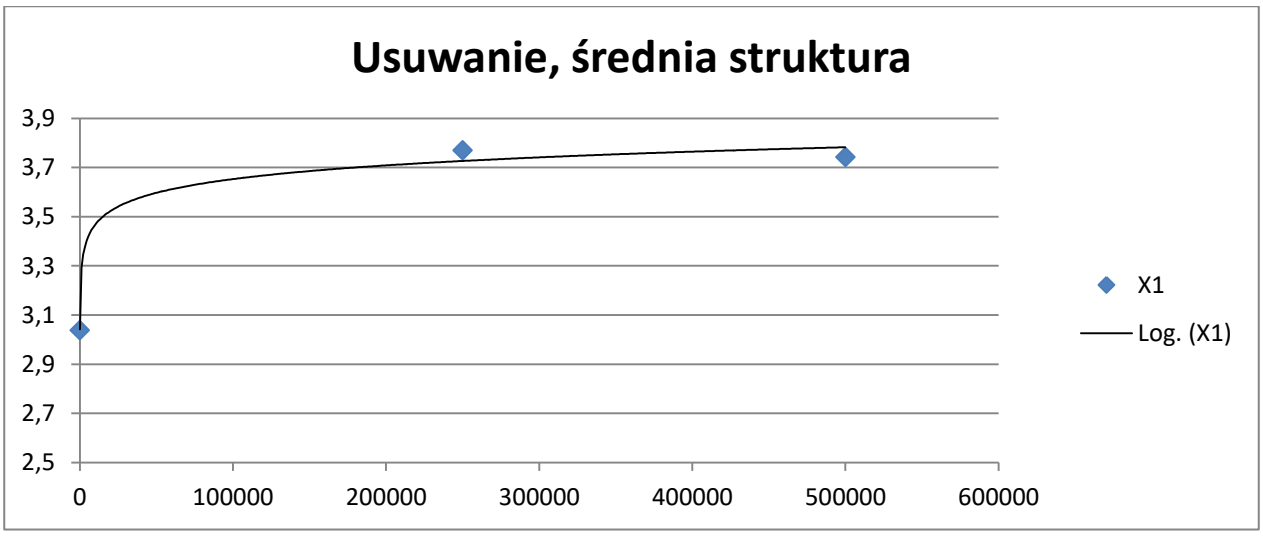
Tabela 2 Usuwanie elementu z kopca

Średnia OLE- Średnia Od Liczby Elementów

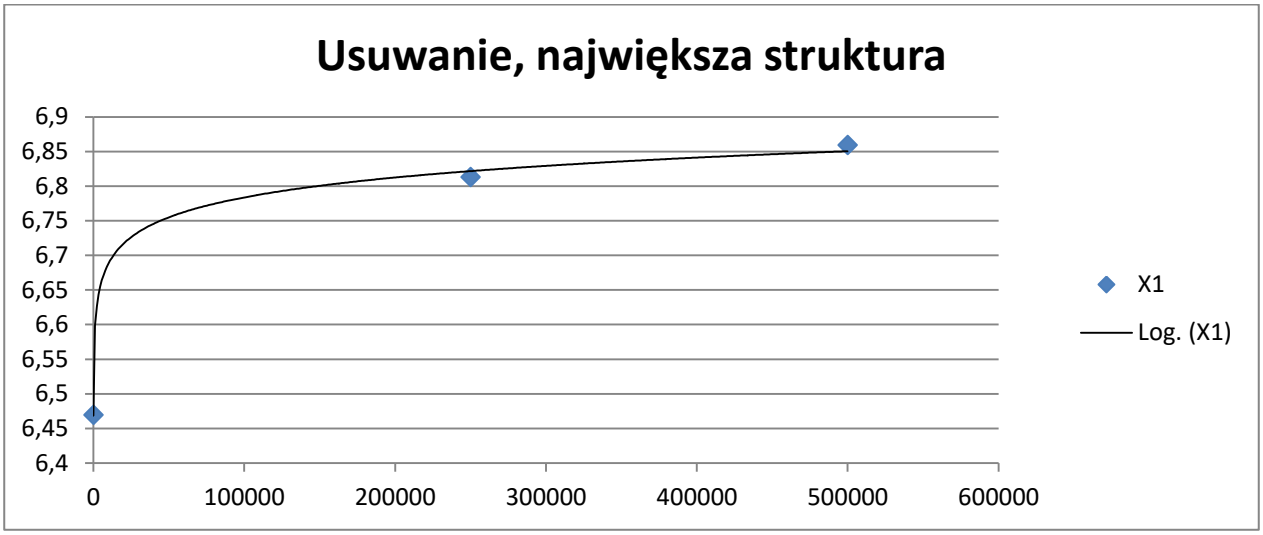
.



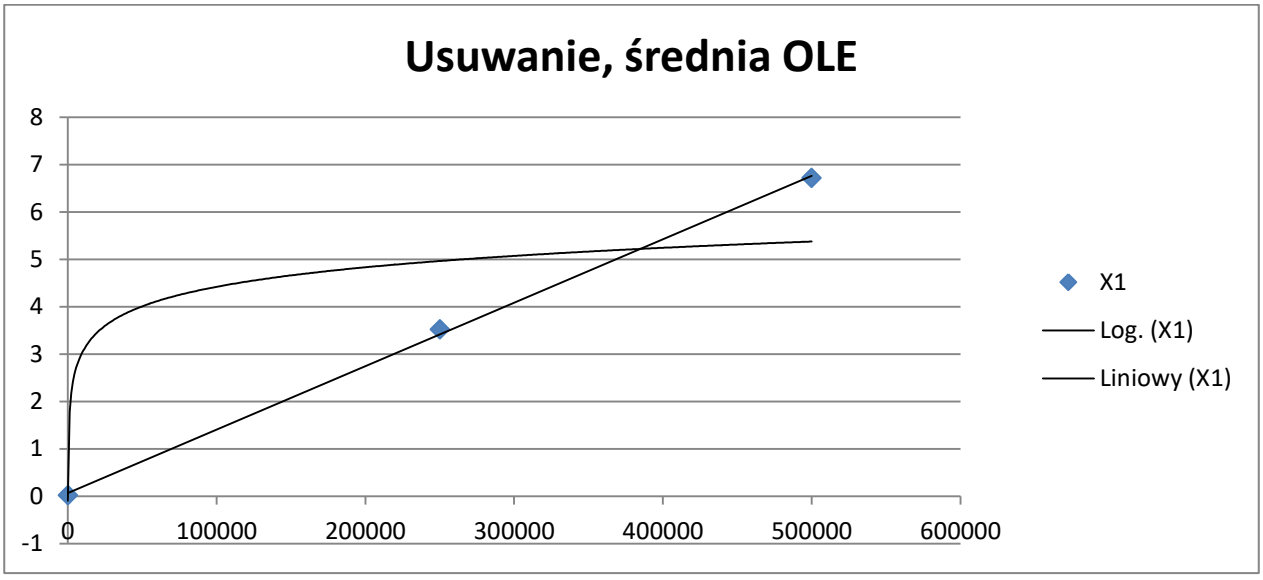
Wykres 5 Usuwanie z kopca binarnego dla liczby elementów: 50



Wykres 6 Usuwanie z kopca binarnego dla liczby elementów: 250 000



Wykres 7 Usuwanie z kopca binarnego dla liczby elementów: 500 000



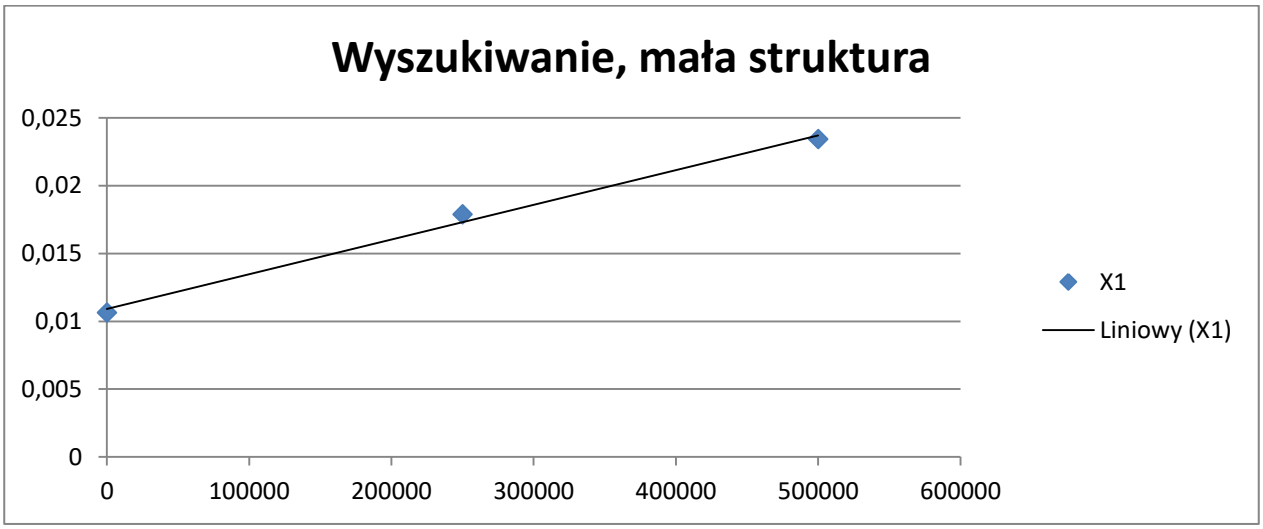
Wykres 8 Usuwanie z kopca binarnego - wykres średniej od liczby elementów

Wyszukiwanie:

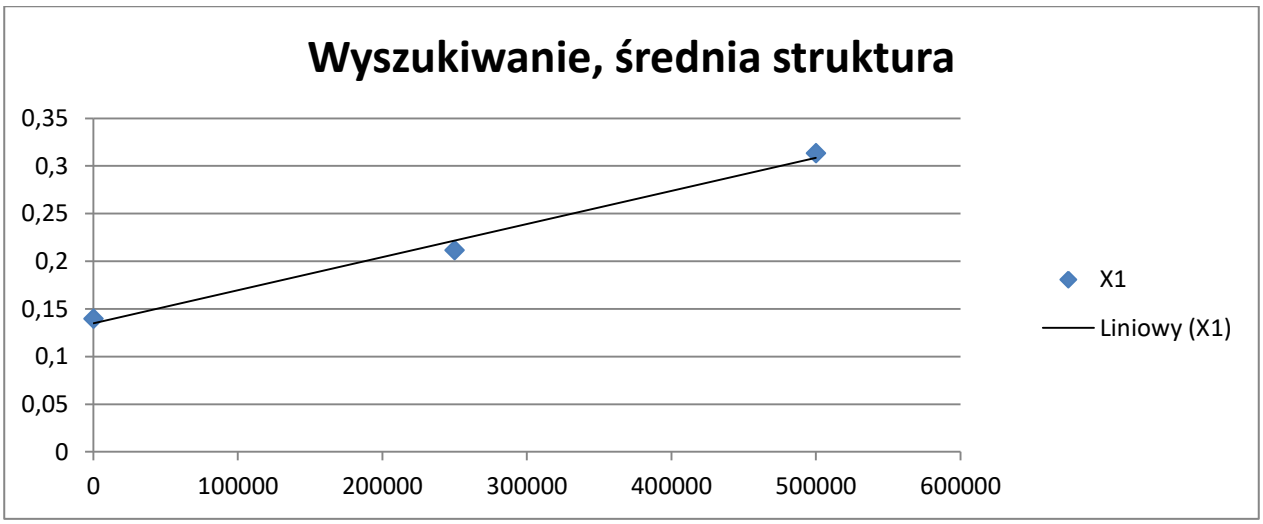
Wyszukaj				
lp.	Liczba elementów	Wielkość liczb	Średnia czasu	Średnia OLE
1	50	M	0,0106365	0,01730962
2	50	S	0,017873442	
3	50	D	0,02341892	
4	250 000	M	0,139767	0,22162102
5	250 000	S	0,21168244	
6	250 000	D	0,31341362	
7	500 000	M	0,148724	0,278152
8	500 000	S	0,291967	
9	500 000	D	0,393765	

Tabela 3 Wyszukiwanie elementu w kopcu

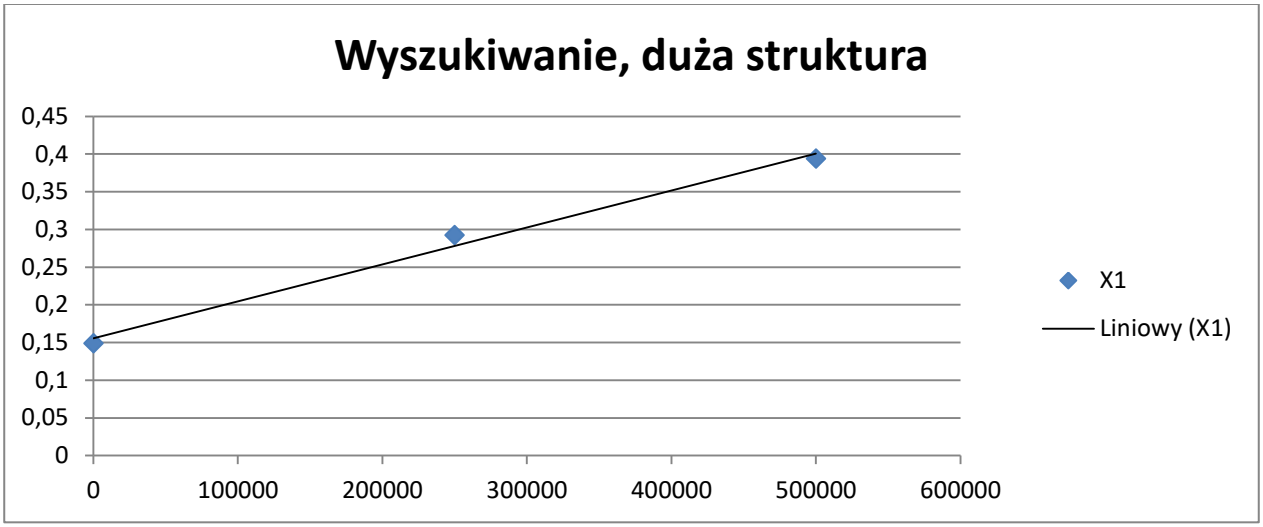
. *Średnia OLE- Średnia Od Liczby Elementów*



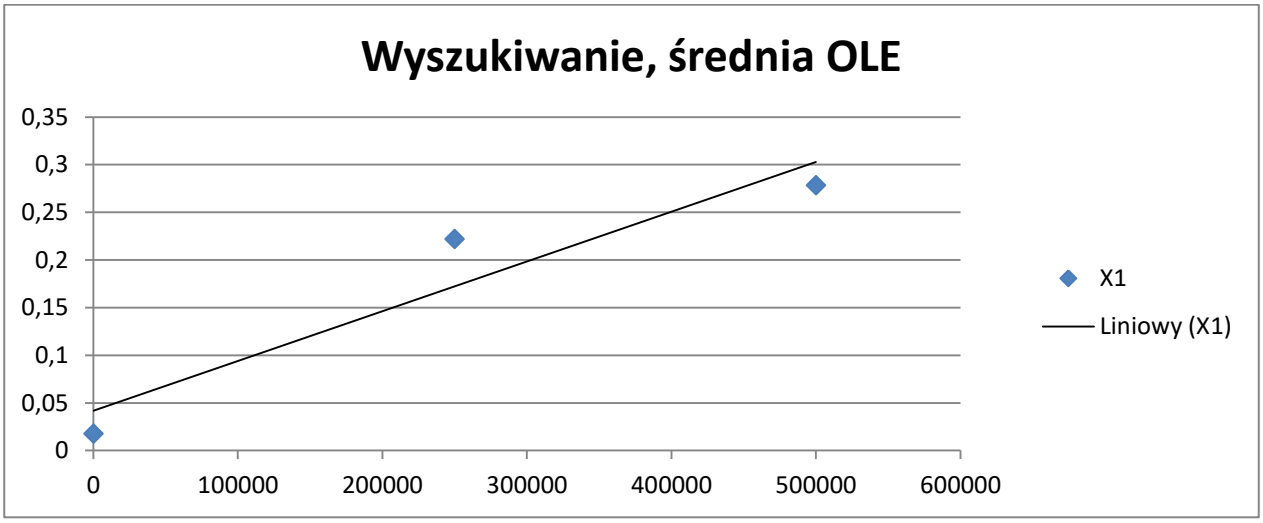
Wykres 9 Wyszukiwanie w kopcu binarnym dla liczby elementów: 50



Wykres 10 Wyszukiwanie w kopcu binarnym dla liczby elementów: 250 000



Wykres 11 Wyszukiwanie w kopcu binarnym dla liczby elementów: 500 000



Wykres 12 Wyszukiwanie w kopcu binarnym dla średniej OLE

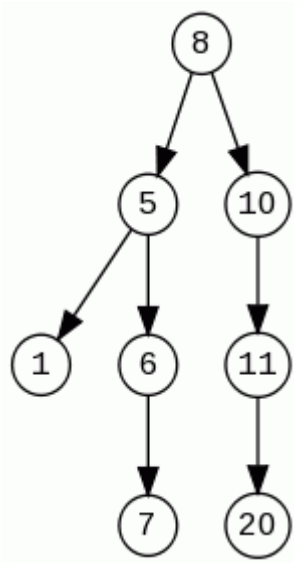
4. Drzewo BST

4.1 Wstęp teoretyczny

Drzewo BST (binary search tree) jest dynamiczną strukturą danych będącą drzewem binarnym. Oprócz pola wartości drzewo BST posiada jeszcze dwa pola: L i P, wskazujące odpowiednio na lewy i prawy następnik. Drzewo BST ma szczególną własność:

- jeżeli element drzewa znajduje się w lewej gałęzi to jest mniejszy od swego poprzednika
- jeżeli element drzewa znajduje się w prawej gałęzi to jest większy od swego poprzednika

Przykład drzewa BST:



Rysunek 2 Przykład drzewa BST

Węzły, oprócz klucza, przechowują wskaźniki na swojego lewego i prawego syna oraz na swojego ojca.

Koszt wykonania podstawowych operacji w drzewie BST (wstawienie, wyszukanie, usunięcie węzła) jest proporcjonalny do wysokości drzewa *h*, ponieważ operacje wykonywane są wzdłuż drzewa. Fakt ten w notacji Landaua zapisuje się O(*h*). Jeżeli drzewo jest zrównoważone, to jego wysokość bliska jest logarytmowi dwójkowemu liczby węzłów $(\boldsymbol{h \approx log_2(n)})$ zatem dla drzewa o *n* węzłach optymistyczny koszt każdej z podstawowych operacji wynosi O(log *n*). Z drugiej strony drzewo skrajnie niezrównoważone ma wysokość porównywalną z liczbą węzłów (w skrajnym przypadku drzewa zdegenerowanego do listy wartości te są równe: *h*=*n*), z tego powodu koszt pesymistyczny wzrasta do O(*n*).

Drzewa BST często stosuje się w zadaniach, w których wymagane jest względnie szybkie sortowanie lub wyszukiwanie elementów, na przykład różnego rodzaju słowniki, kolejki priorytetowe.

Drzewo BST zaimplementowane w moim programie oparte jest o listę, inaczej niż miało to miejsce w przypadku kopca binarnego.

4.2 Działanie programu

Wyszukiwanie:

Podstawową operacją wykonywaną w drzewie BST jest wyszukiwanie znajdującego się w nim klucza. Do wyszukiwania węzła w drzewie BST, można użyć procedury określonej w następujący sposób.

1. Mamy dany wskaźnik do korzenia drzewa oraz klucz k,
2. Procedura wyznacza wskaźnik do węzła zawierającego klucz k, jeżeli taki węzeł istnieje.
3. W przeciwnym wypadku w wyniku otrzymuje się wartość NIL.
4. Rozpoczyna się wyszukiwanie w korzeniu i procedura schodzi po ścieżce w dół drzewa.
5. W każdym napotkanym węźle porównywany zostaje klucz k z wartością w kluczu.
6. Jeżeli te wartości są równe to wyszukiwanie zostaje przerwane,
7. W przypadku gdy *k* jest większe od *k[x]* to dalsze wyszukiwanie przebiega jw. w prawym poddrzewie węzła x, ponieważ z własności drzewa BST wynika, że k nie może znajdować się w lewym poddrzewie. Analogicznie dzieje się w przypadku gdy k jest mniejsze od *k[x].*

Takie przeszukiwanie drzewa nosi nazwę Wzdłużnego – preorder.

Ponadto istnieją jeszcze wyszukiwanie takie jak poprzeczne (inorder), czyli *lewe poddrzewo, korzeń, prawe poddrzewo* oraz Wsteczne (postorder) – *lewe poddrzewo, prawe poddrzewo, korzeń.*

Dodawanie oraz usuwanie

Jeśli drzewo BST jest puste (korzeń=nil) należy wstawić element (nie porównujemy go z innymi), w przeciwnym wypadku porównujemy wartość elementu z następnikami każdego węzła (zaczynając od korzenia). Jeżeli wartość elementu jest nie większa od wartości porównywanego wierzchołka to przechodzimy do lewego następnika, w przeciwnym razie przechodzimy do prawego następnika. Krok ten powtarzamy aż znajdziemy dla naszego elementu odpowiednie miejsce, tzn. gdy następnik, do którego powinniśmy iść jest pusty (nil). Następnie wstawiamy element jako odpowiedni następnik (prawy, jeśli element jest większy od węzła, lewy jeśli nie większy).

W przypadku usuwania argumentem procedury służącej do usuwania danego węzła *z* z drzewa BST jest wskaźnik do z. W tej procedurze rozpatrywane są trzy przypadki. Jeśli z nie ma synów, to w jego ojcu p[z] zastępujemy wskaźnik do *z* wartością NIL. Jeśli węzeł ma tylko jednego syna, to zostaje „wycięty" z ustalony wskaźnikiem między jego ojcem a jedynym synem. Podczas gdy węzeł ma dwóch synów, wycięty zostaje następnik *y* węzła *z*, o którym jest wiadome, że nie ma lewego syna, oraz zastąpiona jest zawartość z, z zawartością y.

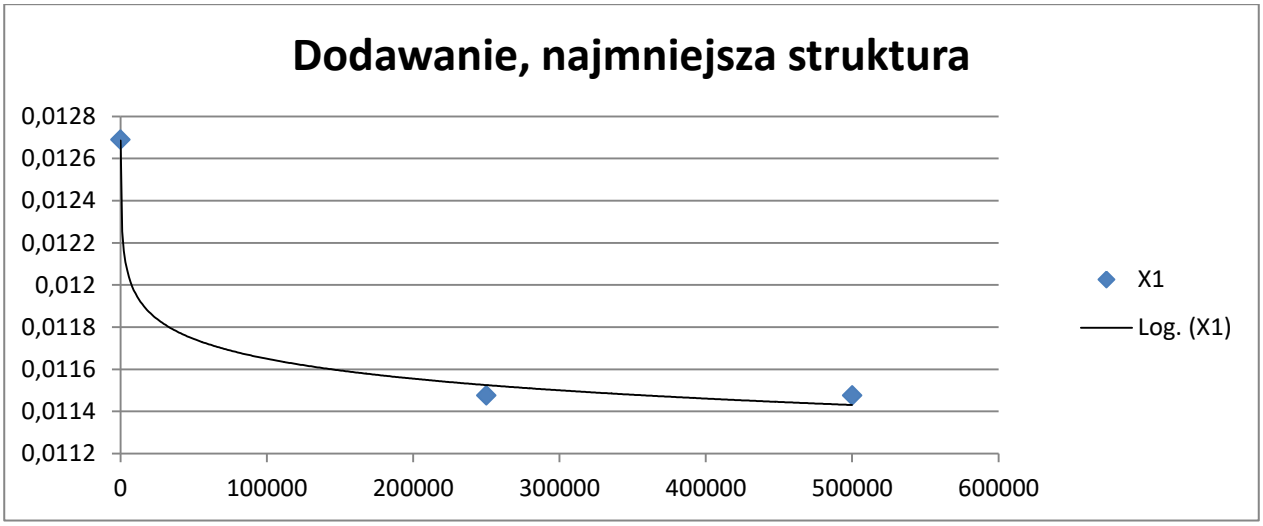
4.3 Pomiary

Dodawanie:

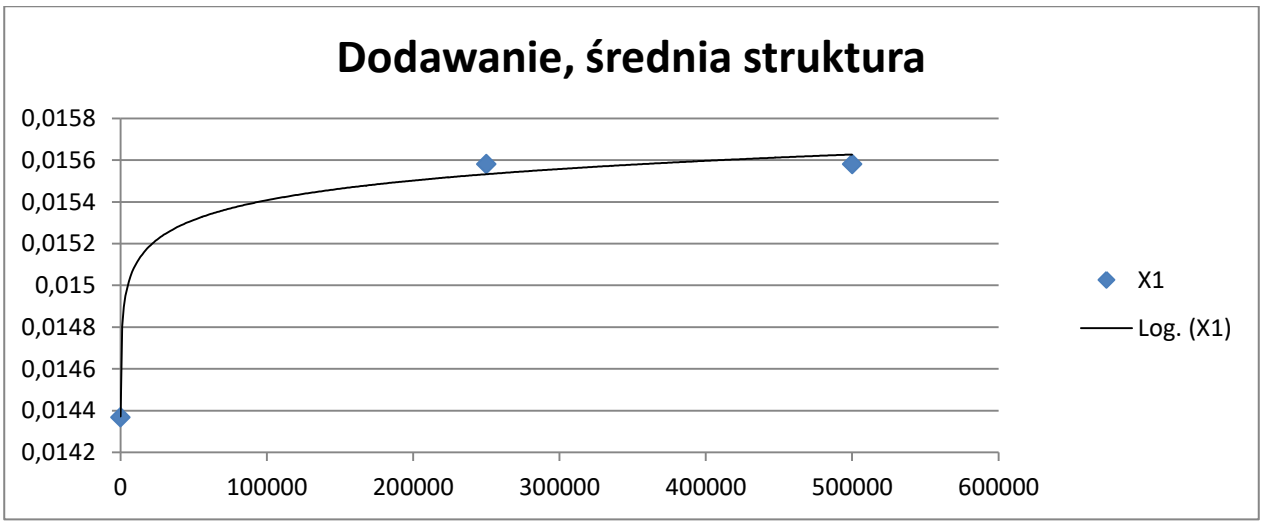
DODAWANIE				
lp.	Liczba elementów	Wielkość liczb	Średnia czasu	Średnia OLE
1	50	M	0,01268912	0,011880507
2	50	S	0,0114762	
3	50	D	0,0114762	
4	250 000	M	0,0143686	0,015177227
5	250 000	S	0,01558154	
6	250 000	D	0,01558154	
7	500 000	M	0,0148351	0,015208473
8	500 000	S	0,0153954	
9	500 000	D	0,01539492	

Tabela 4 Dodawanie do drzewa BST

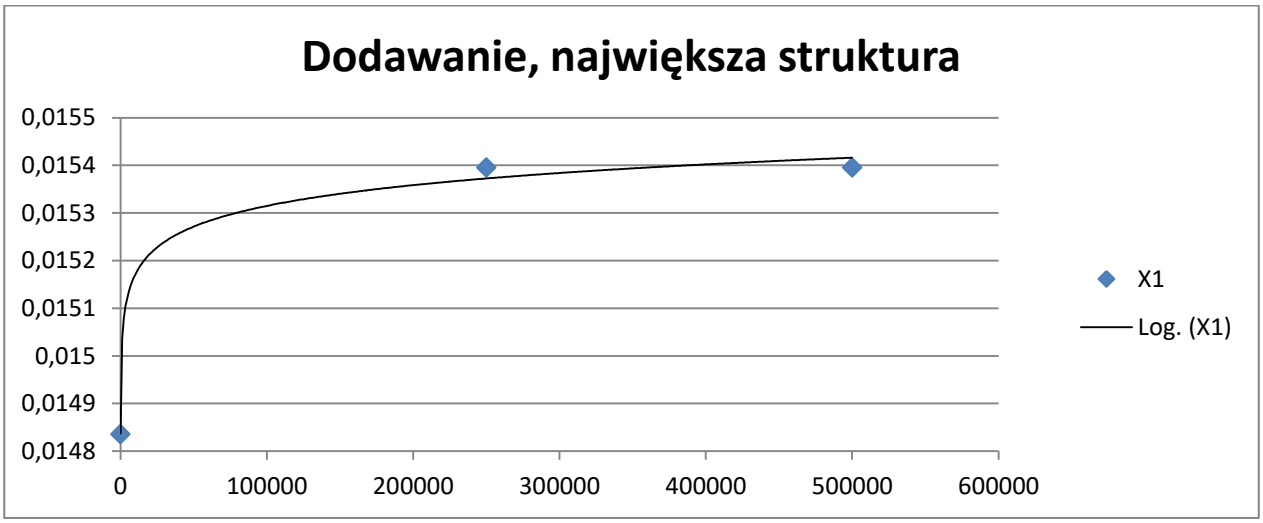
Średnia OLE- Średnia Od Liczby Elementów



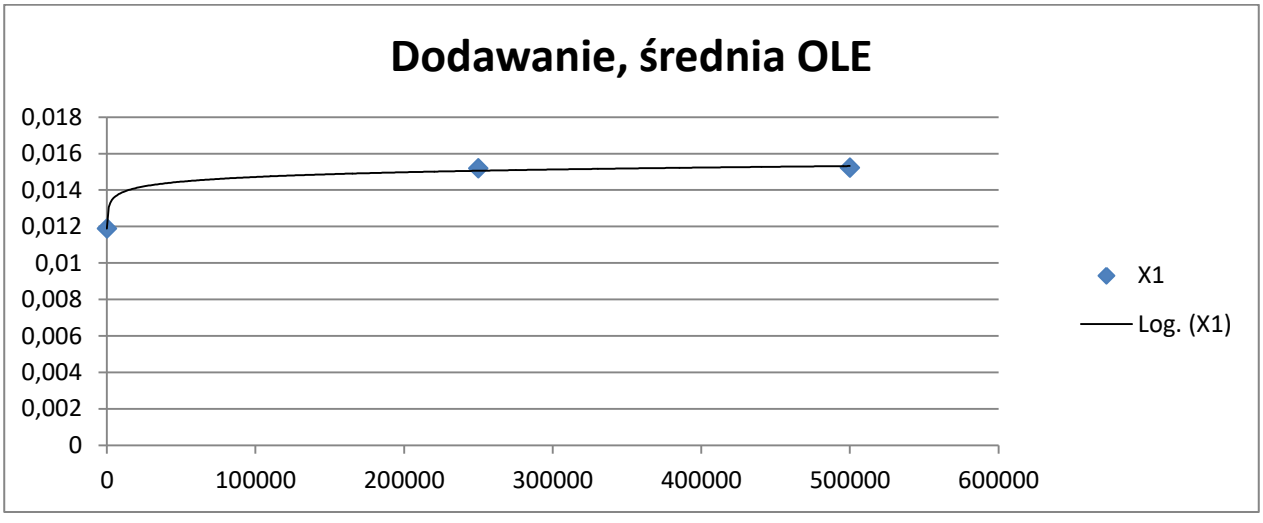
Wykres 13 Dodawanie do drzewa BST, dla struktury danych liczącej 50 elementów



Wykres 14 Dodawanie do drzewa BST, dla struktury danych liczącej 250 000 elementów



Wykres 15 Dodawanie do drzewa BST, dla struktury danych liczącej 500 000 elementów

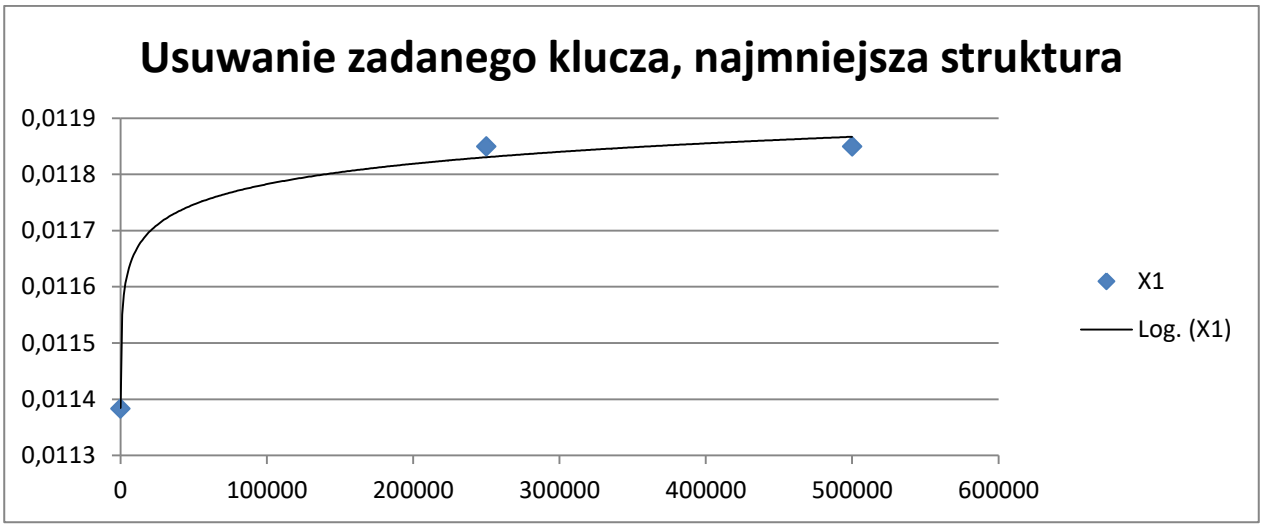


Wykres 16 Wykres przedstawia zależność średniej (pomiaru czasu) od liczby elementów.

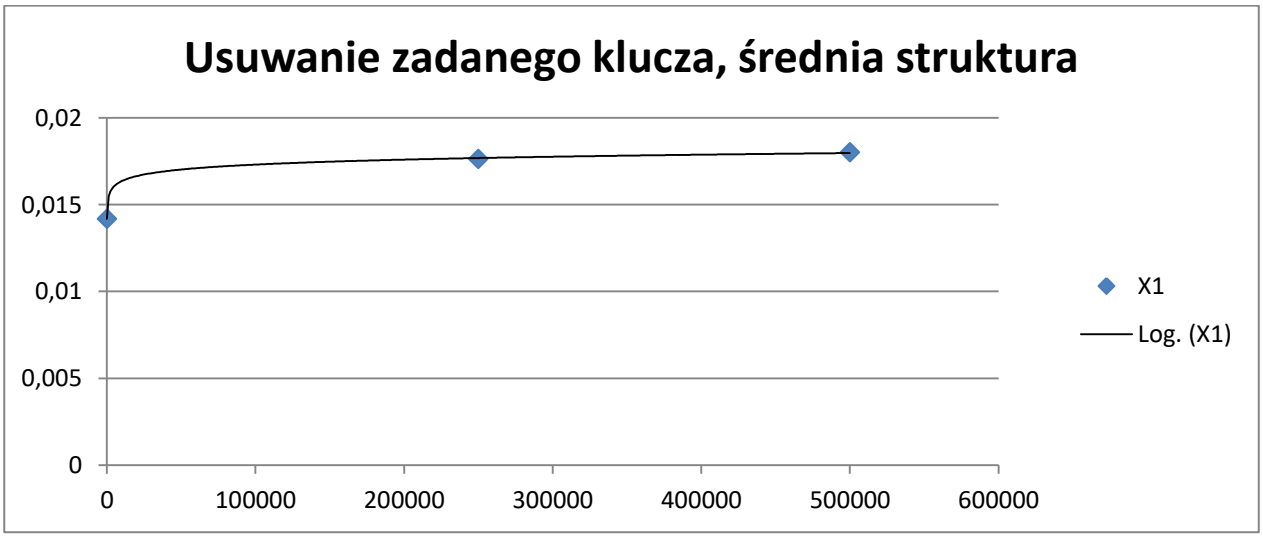
Usuwanie:

Usuwanie				
lp.	Liczba elementów	Wielkość liczb	Średnia czasu	Średnia OLE
1	50	M	0,0113829	0,011693907
2	50	S	0,01184942	
3	50	D	0,0118494	
4	250 000	M	0,014175956	0,016605852
5	250 000	S	0,0176342	
6	250 000	D	0,0180074	
7	500 000	M	0,0136222	0,015115013
8	500 000	S	0,0147418	
9	500 000	D	0,01698104	

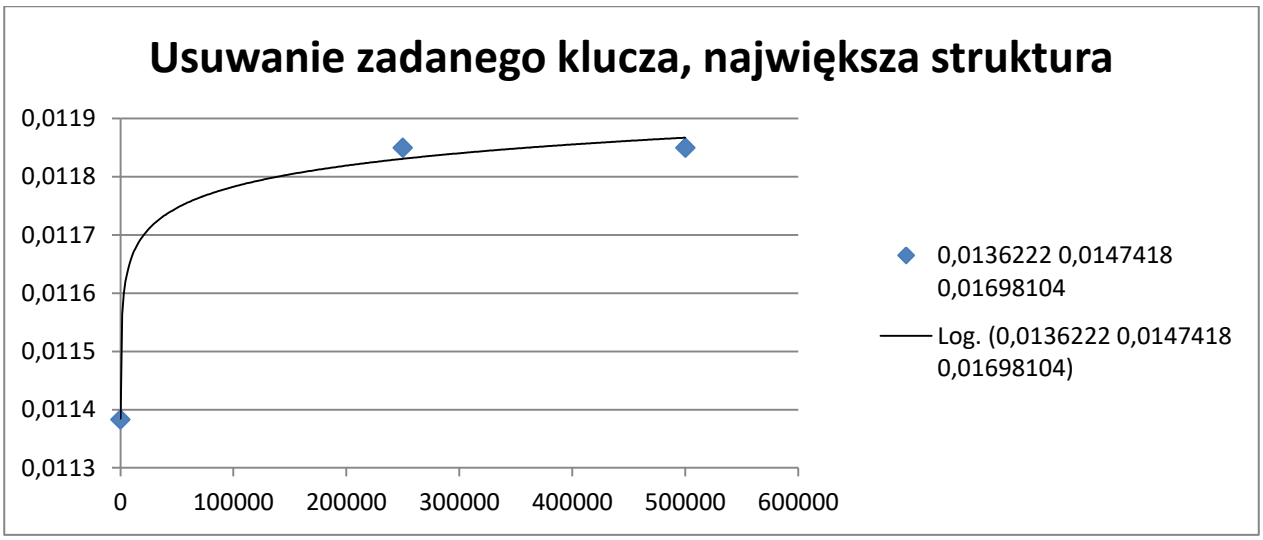
Tabela 5 Usuwanie zadanej wartości z drzewa BST



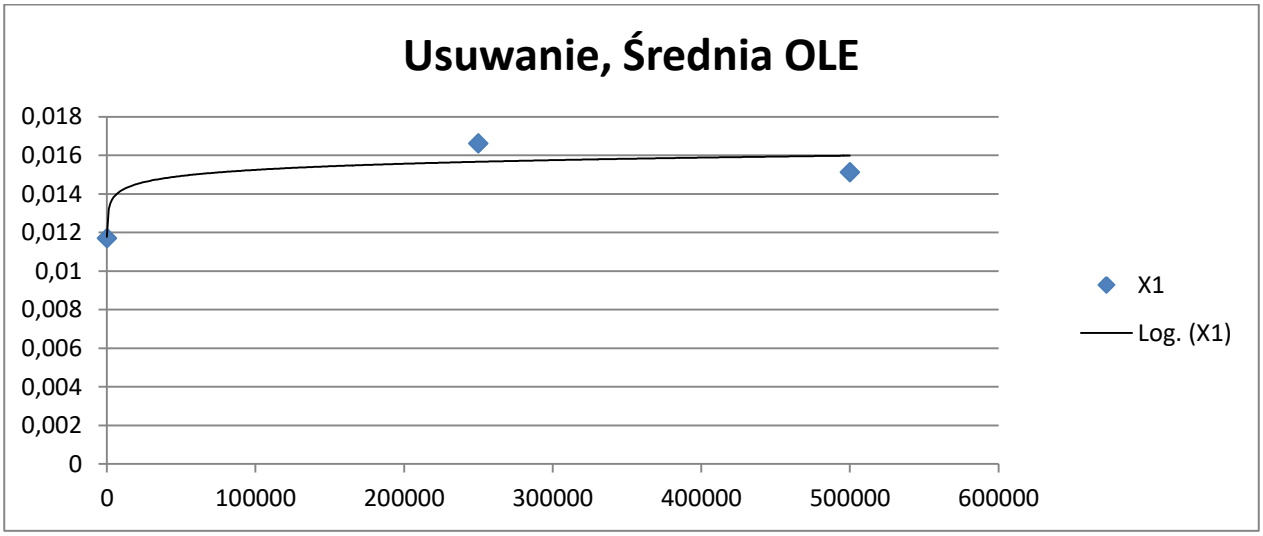
Wykres 17 Usuwanie z drzewa BST, dla struktury danych liczącej 50 elementów



Wykres 18 Usuwanie z drzewa BST, dla struktury danych liczącej 250 000 elementów



Wykres 19 Usuwanie z drzewa BST, dla struktury danych liczącej 500 000 elementów

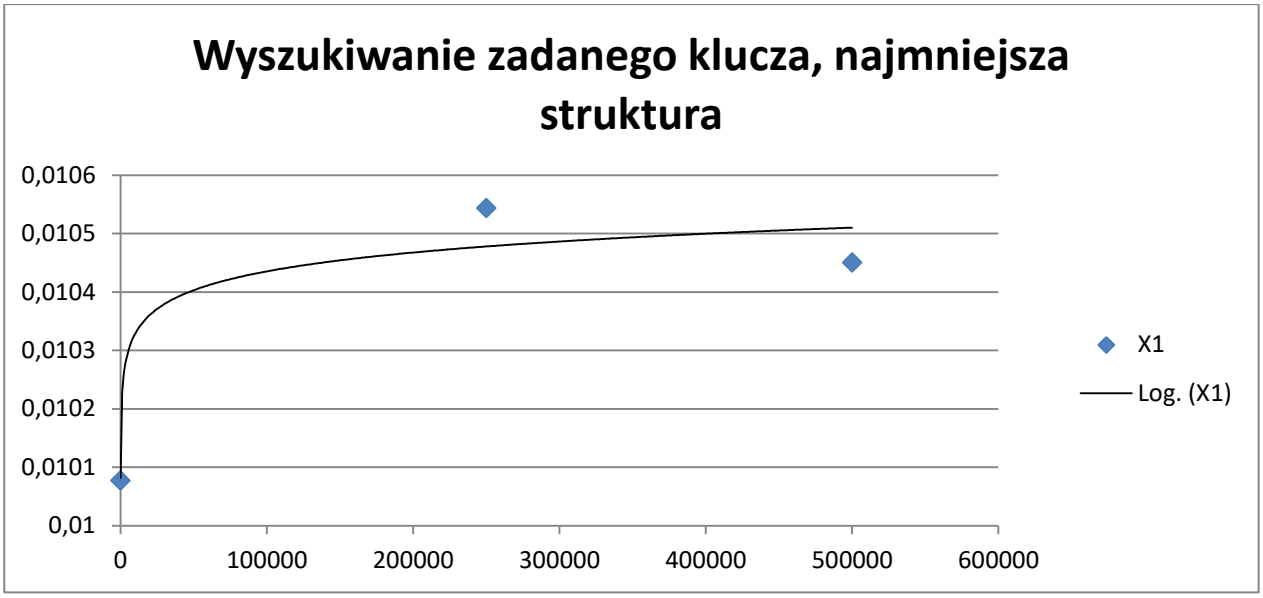


Wykres 20 Wykres przedstawia zależność czasu operacji usuwania dla średniej (pomiaru czasu) od liczby elementów.

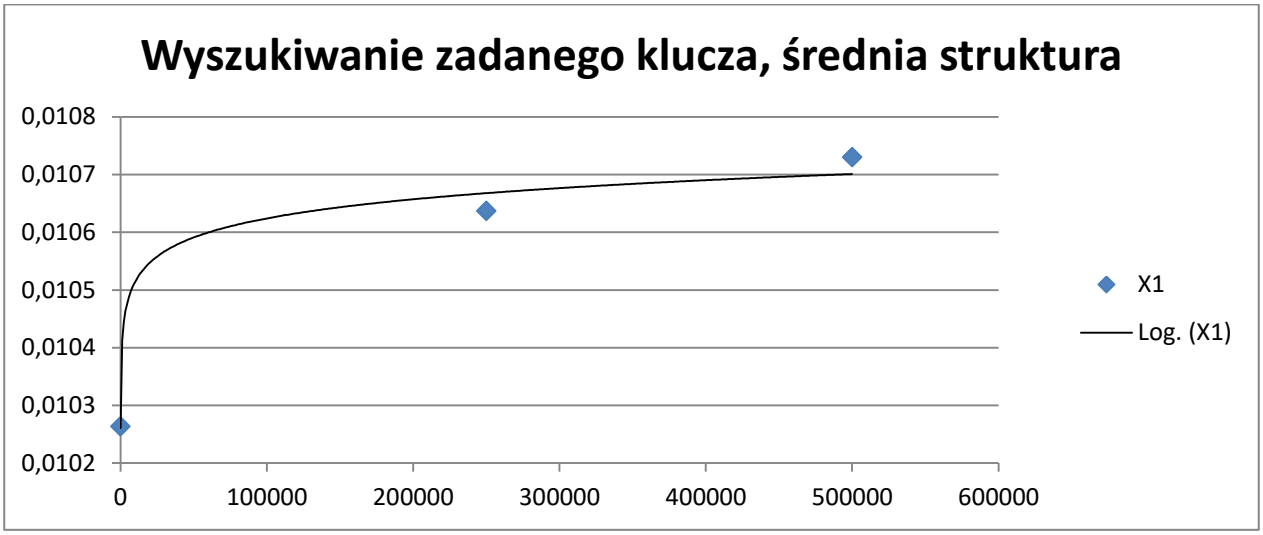
Wyszukiwanie:

WYSZUKAJ				
lp.	Liczba elementów	Wielkość liczb	Średnia czasu	Średnia OLE
1	50	M	0,010076678	0,010356593
2	50	S	0,0105432	
3	50	D	0,0104499	
4	250 000	M	0,01026329	0,010543191
5	250 000	S	0,0106365	
6	250 000	D	0,010729782	
7	500 000	M	0,01026329	0,010636494
8	500 000	S	0,010729782	
9	500 000	D	0,01091641	

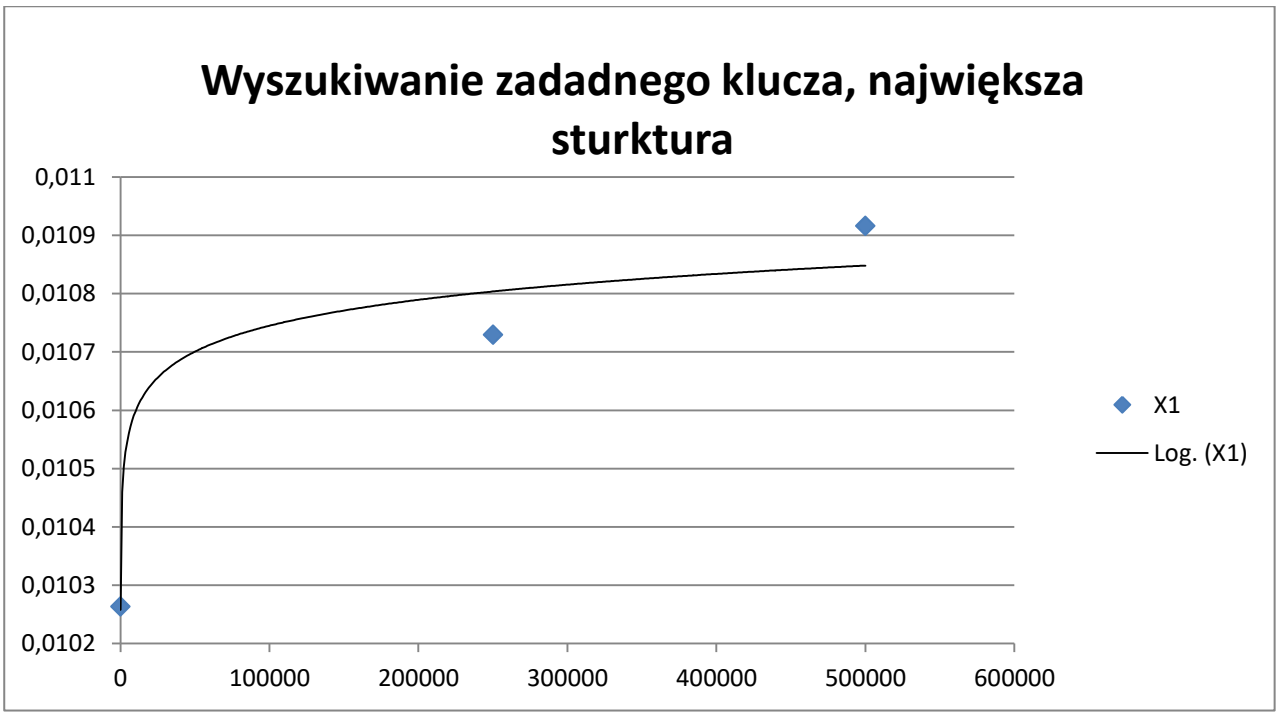
Tabela 6 Wyszukiwanie w drzewie BST



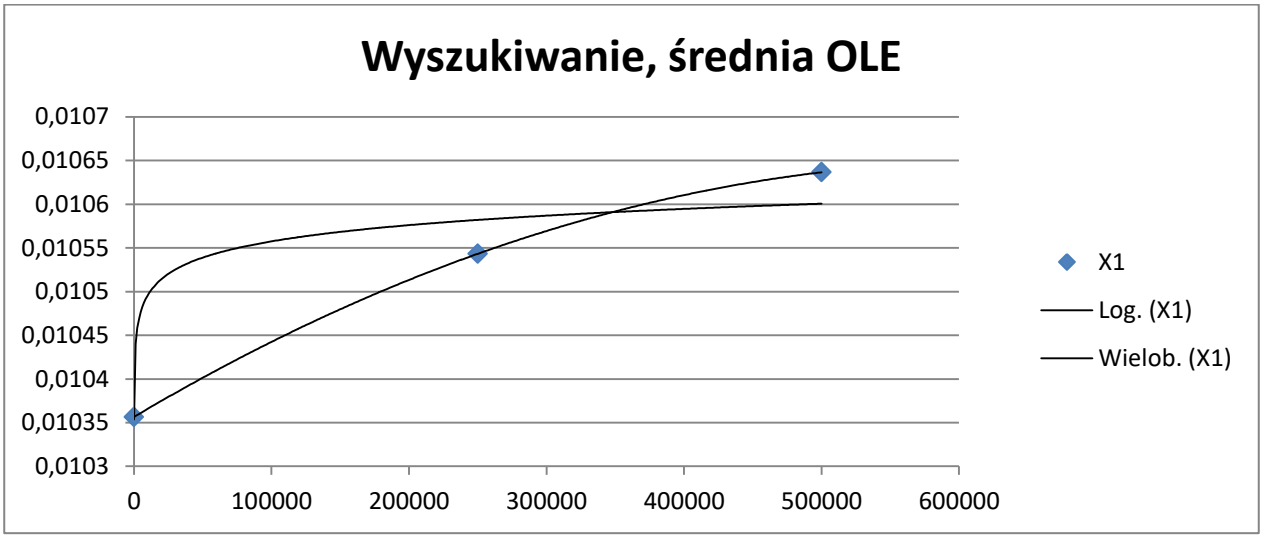
Wykres 21 Wyszukiwanie w drzewie BST, dla struktury danych liczącej 50 elementów



Wykres 22 Wyszukiwanie w drzewie BST, dla struktury danych liczącej 250 000 elementów



Wykres 23 Wyszukiwanie w drzewie BST, dla struktury danych liczącej 500 000 elementów



Wykres 24 Wykres przedstawia zależność czasu operacji wyszukiwania dla średniej (pomiaru czasu) od liczby elementów.

5. Drzewo czerwono-czarne

5.1 Wstęp teoretyczny

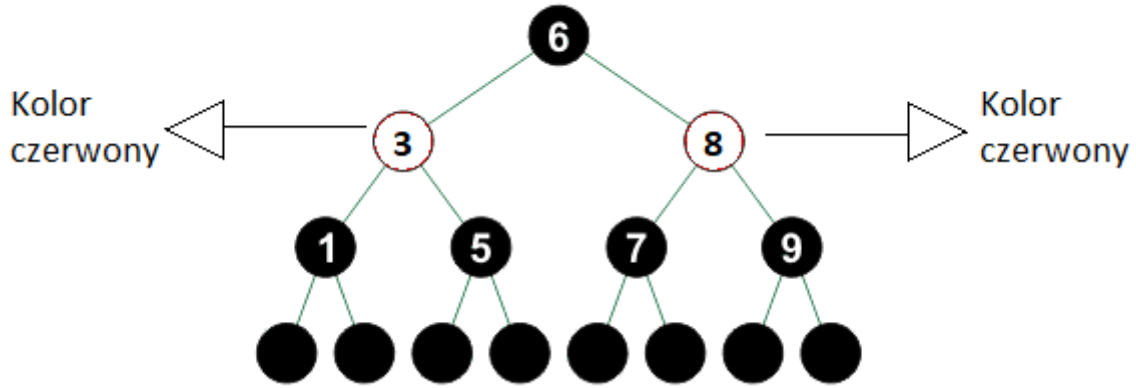
Drzewa czerwono-czarne są odmianą samoorganizujących się binarnych drzew poszukiwań, którą wynalazł niemiecki informatyk Rudolf Bayer w roku 1972 i którą następnie opisali w 1978 Leonidas J. Guibas i Robert Sedgewick. W liściach drzew czerwono-czarnych nie przechowuje się danych. Liście te nawet nie muszą znajdować się bezpośrednio w pamięci komputera. Puste wskazanie w polu syna węzła może być interpretowane jako liść. Jednakże używanie rzeczywistych liści upraszcza niektóre algorytmy na drzewach czerwono-czarnych. W celu zaoszczędzenia pamięci często wybiera się pojedynczy **węzeł-strażnika**, który pełni rolę wszystkich liści w drzewie. W takim przypadku węzły wewnętrzne w drzewie czerwono-czarnym w polach synów-liści przechowują wskazania do tego węzła strażnika. Puste drzewo zawiera jedynie węzeł strażnika. Drzewa czerwono-czarne są odmianą samoorganizujących się drzew BST.

Drzewa czerwono-czarne gwarantują, iż ich wysokość nie przekroczy dwukrotnej wartości wysokości minimalnej. Dokonywane jest to przez kolorowanie węzłów na czerwono lub czarno i stosowanie po każdej operacji wstawiania lub usuwania odpowiedniej procedury równoważącej drzewo tak, aby były spełnione następujące warunki:

1. Każdy węzeł drzewa jest albo czerwony, albo czarny.
2. Każdy liść drzewa (węzeł pusty nil) jest zawsze czarny.
3. Korzeń drzewa jest zawsze czarny.
4. Jeśli węzeł jest czerwony, to obaj jego synowie są czarni – innymi słowy, w drzewie nie mogą występować dwa kolejne czerwone węzły, ojciec i syn.
5. Każda prosta ścieżka od danego węzła do dowolnego z jego liści potomnych zawiera tę samą liczbę węzłów czarnych.

Wskaźniki równe NIL będziemy traktować jako wskazania na zewnętrzne węzły drzewa poszukiwań binarnych (liście), a zwyczajne węzły drzewa zawierające klucze będą pełniły funkcję wewnętrznych węzłów drzewa .

Przykład drzewa czerwono-czarnego:



Rysunek 3 Przykład drzewa czerwono-czarnego

Ze względu na druk czarno-biały węzły czerwone zostały osobno opisane.

5.2 Operacje dodawania, usuwania, wyszukiwania w drzewie czerwono-czarnym

Dodawanie:

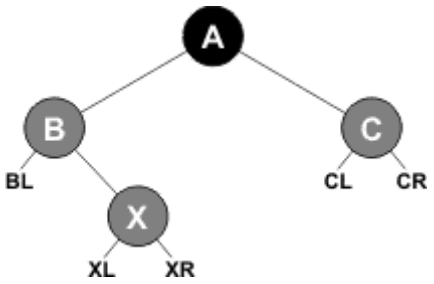
Operacja wstawiania węzła do drzewa czerwono-czarnego składa się z następujących etapów:

Tworzymy nowy węzeł, inicjujemy go danymi, po czym wstawiamy do drzewa czerwono-czarnego za pomocą zwykłej procedury wstawiania węzła do drzewa binarnego (ponieważ drzewo czerwono-czarne wciąż jest drzewem binarnym). Gdy węzeł znajdzie się w drzewie, kolorujemy go na czerwono i sprawdzamy czy nie zostały zaburzone warunki drzewa czerwono-czarnego.

- Jeśli nowy węzeł jest korzeniem, to zmieniamy jego kolor na czarny. Wstawianie jest zakończone.
- Jeśli ojciec nowego węzła jest czarny, to również kończymy, ponieważ warunki drzewa nie zostały zaburzone.
- W innym razie mamy dwa kolejne węzły czerwone i musimy przywrócić warunki drzewa czerwono-czarnego. W tym celu należy przeanalizować trzy przypadki, które występują w lustrzanych postaciach.

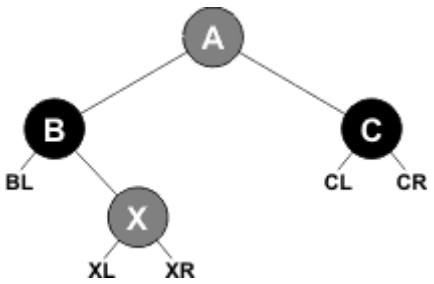
Przypadek 1:

Wujek **C** (brat ojca **B,** czyli drugi syn węzła **A**) wstawionego węzła **X** jest czerwony.



Rysunek 4 Dodawanie Drzewo Czerwono-Czarne , Przypadek 1 stan początkowy

Ojca B i wujka C kolorujemy na czarno. Dziadka A kolorujemy na czerwono.

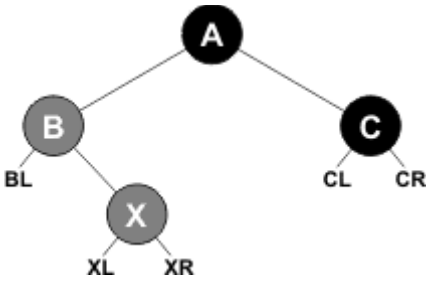


Rysunek 5 Dodawanie Drzewo Czerwono-Czarne , Przypadek 1 stan po zmianie kolorów ojca B i wujka C na czarno, dziadka A na czerwono

Jeśli węzeł A jest korzeniem drzewa, to zmieniamy jego kolor na czarny (ponieważ korzeń drzewa musi być czarny) i kończymy. W przeciwnym razie za nowe X przyjmujemy A i sprawdzamy od początku kolejne przypadki na wyższym poziomie drzewa.

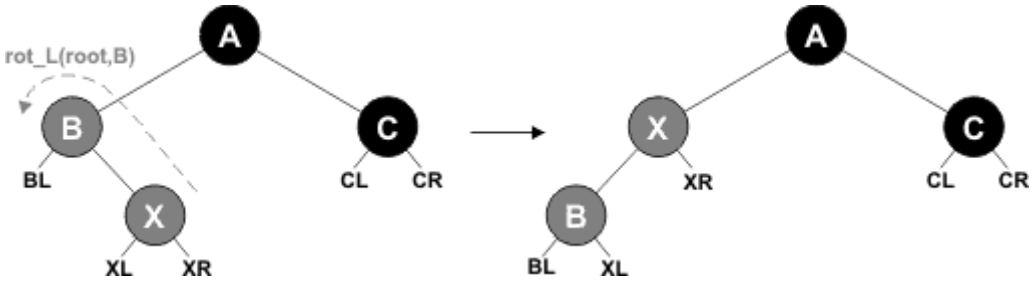
Przypadek 2:

Wujek **C** jest czarny, a węzeł **X** jest prawym dzieckiem węzła **B** (przypadek lustrzany: wujek **B** jest czarny, a **X** jest lewym dzieckiem węzła **C**).



Rysunek 6 Dodawanie Drzewo Czerwono-Czarne , Przypadek 2 stan początkowy

Wykonujemy rotację w lewo względem węzła B (w przypadku lustrzanym wykonujemy rotację w prawo względem C).

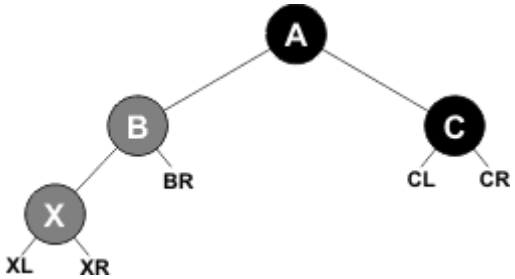


Rysunek 7 Dodawanie Drzewo Czerwono-Czarne , Przypadek 1 stan po rotacji drzewa w lewo względem węzła B

Za X przyjmujemy węzeł B i przechodzimy do przypadku 3.

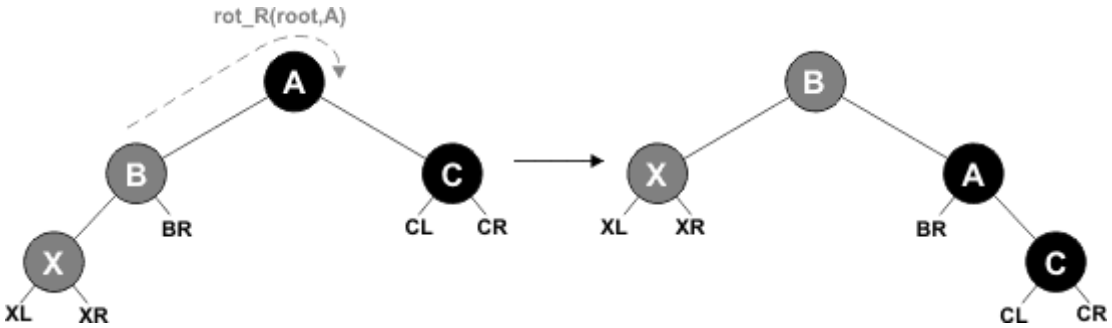
Przypadek 3:

Wujek **C** jest czarny, a węzeł **X** jest lewym dzieckiem węzła **B** (przypadek lustrzany: wujek B jest czarny, a X jest prawym dzieckiem węzła C):



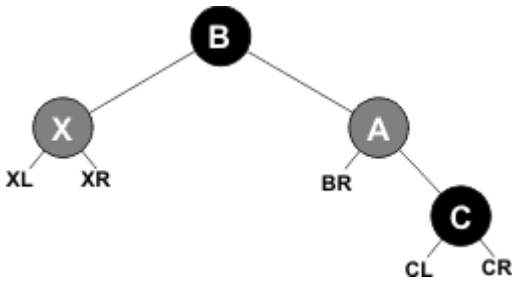
Rysunek 8 Dodawanie Drzewo Czerwono-Czarne , Przypadek 3 stan początkowy

Wykonujemy rotację w prawo względem węzła A (w przypadku lustrzanym wykonujemy rotację w lewo względem A):



Rysunek 9 Dodawanie Drzewo Czerwono-Czarne , Przypadek 3 Rotacja w prawo względem węzła A

Zmieniamy kolory węzłów A i B na przeciwne:



Rysunek 10 Dodawanie Drzewo Czerwono-Czarne , Przypadek 3 zmiana koloru węzła A i B na przeciwne (A – z czarnego na czerwony, B – z czerwonego na czarny, ponieważ węzeł B stał się korzeniem drzewa)

Operacja wstawiania zostaje zakończona, ponieważ węzeł B ma taki sam kolor jak poprzednio węzeł A. Zatem operacja rotacji przywróciła strukturę drzewa czerwono-czarnego.

Wstawianie nowego węzła do drzewa czerwono-czarnego o n węzłach można wykonać w czasie O(log n)

Usuwanie:

Usuwanie węzła z drzewa czerwono-czarnego jest trudniejsze od wstawiania, ponieważ należy rozpatrzyć więcej różnych przypadków. Operacja wykonywana jest dwuetapowo. Najpierw należy usunąć wybrany węzeł tak, jak dla drzew BST (możliwe są 4 przypadki). Ze względu na to, że ze ścieżek, które wcześniej zawierały usunięty węzeł, zniknął jeden węzeł czarny, to potrzeba ten kolor dodać do węzła znajdującego się na tych ścieżkach, czyli do prawego syna następnika. Powstaje ciąg czterech symetrycznych przypadków, które są lustrzanymi odbiciami, które należy uwzględniać przy usuwaniu elementu z drzewa czerwono-czarnego. Węzeł posiadający dodatkowy kolor czarny będziemy oznaczali gwiazdką. Kolor szary oznacza, że węzeł może być czerwony lub czarny.

Kolor Szary: 

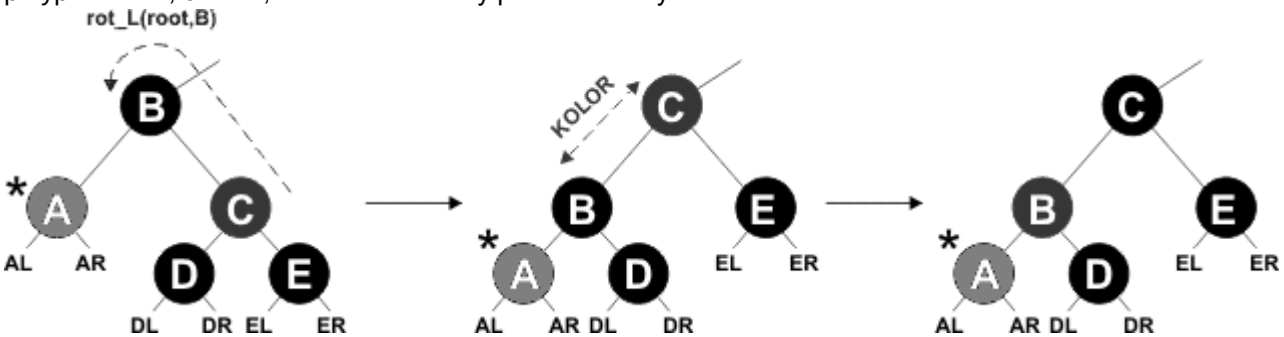
Kolor Czerwony: 

Kolor Czarny: 

Przypadek 1:

Brat węzła A* (węzeł C) jest czerwony.

Wykonujemy rotację w lewo względem B (ojciec węzła A*). Zamieniamy kolory ojca (węzeł B) i dziadka (węzeł C). W ten sposób brat A* (teraz węzeł D) staje się węzłem czarnym. Otrzymujemy jeden z przypadków 2, 3 lub 4, które rozróżniamy po kolorach synów brata A*.

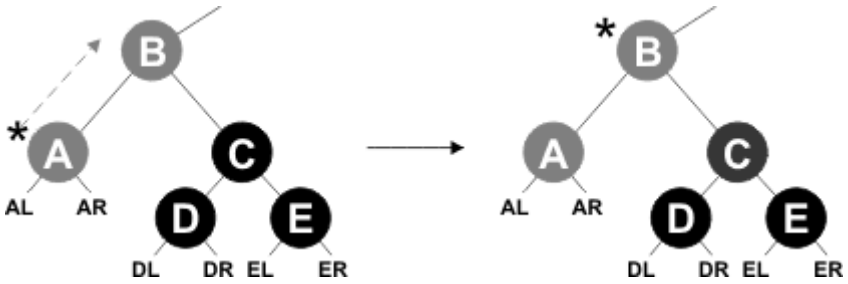


Rysunek 11 Usuwanie z drzewa RB przypadek 1

Przypadek 2. Brat węzła A* (węzeł C) jest czarny i posiada czarnych synów (węzły D i E).

Zabieramy jeden kolor czarny z węzłów A i C, przenosząc go w górę do węzła B. W efekcie węzeł A nie ma już nadmiarowego koloru czarnego, staje się normalnym węzłem czerwonym lub czarnym. Z kolei zabranie koloru czarnego z węzła C skutkuje zmianą koloru na czerwony. Ponieważ obaj synowie D i E są czarni, zmiana ta nie powoduje naruszenia warunku drzewa czerwono-czarnego.

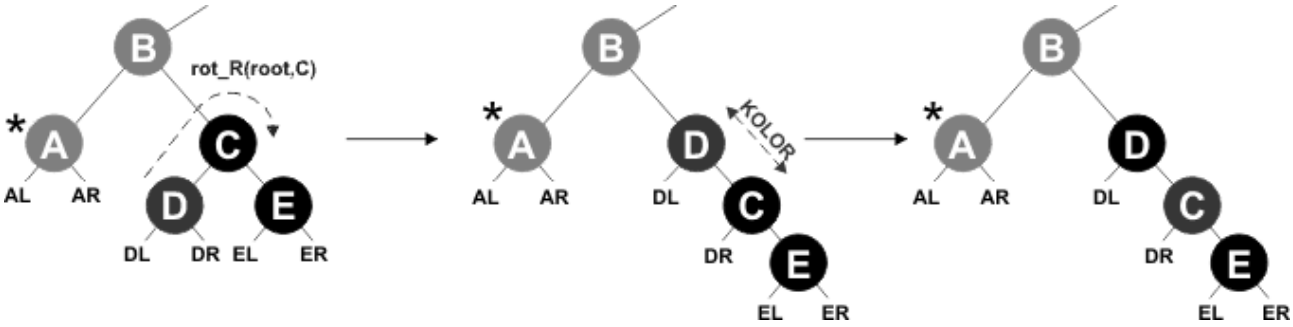
Ponieważ problem dodatkowego koloru czarnego przeniósł się w górę drzewa do węzła B (ojca A), to wracamy do rozpatrywania przypadków z nowym węzłem A*, który teraz jest ojcem poprzedniego węzła A, czyli węzłem B*.



Rysunek 12 Usuwanie z drzewa RB, przypadek 2

Przypadek 3. Brat węzła A* (węzeł C) jest czarny, lewy syn węzła C jest czerwony, prawy syn węzła C jest czarny.

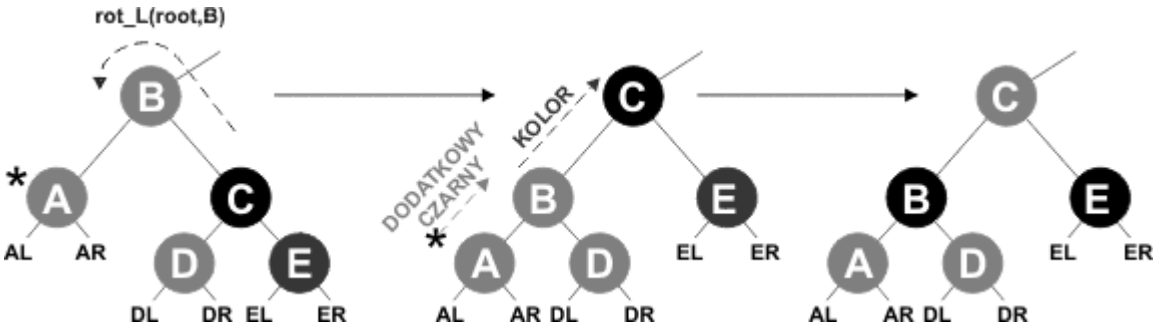
Wykonujemy rotację w prawo względem węzła C, po czym zamieniamy kolory pomiędzy węzłami C i D. Otrzymujemy w ten sposób przypadek 4.



Rysunek 13 Usuwanie z drzewa RB, przypadek 3

Przypadek 4. Brat węzła A* (węzeł C) jest czarny, a jego prawy syn jest czerwony.

Wykonujemy rotację względem węzła B (ojciec A*). Przenosimy kolor z węzła B do C. W takim przypadku B traci kolor i możemy w tym węźle umieścić dodatkowy kolor czarny z węzła A. Dzięki temu pozbywamy się dodatkowej czerni. Na koniec węzeł E kolorujemy na czarno.



Rysunek 14 Usuwanie z drzewa RB, przypadek 4

Ta transformacja przywraca strukturę drzewa czerwono-czarnego, zatem kończy operację usuwania węzła.

Podobnie jak pozostałe elementarne operacje na drzewie czerwono-czarnym o n węzłach, usuwanie elementu jest wykonywane w czasie **O(log n).**

Jednak ze względu na to, że w badanych zestawach danych występują struktury, które liczą 250 000 i 500 000 elementów z zakresu liczb -100 do 100, to z łatwością można wywnioskować, że prawdopodobieństwo powtarzania się liczb jest bliskie 1. Oznacza to, że np. liczba 33 może wystąpić 20 tysięcy razy dla struktury zawierającej 500 000 elementów. Z tego też względu usuwanie takiego elementu zajmie więcej czasu, ze względu na zasadę działania drzewa czerwono-czarnego (samoorganizację struktury oraz zasadę usuwania elementu z drzewa)

Wyszukiwanie:

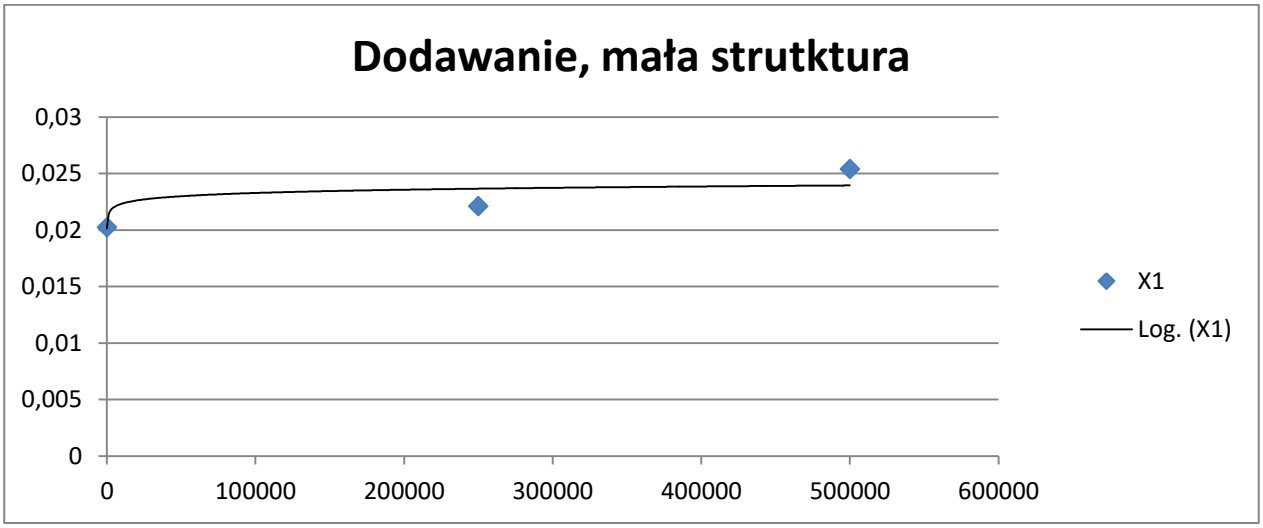
Drzewa czerwono-czarne mają podobny algorytm wyszukiwania jak drzewa BST, przyjmujemy pesymistyczną złożoność czasową takiej operacji czyli: **O(log n).**

5.3 Pomiary

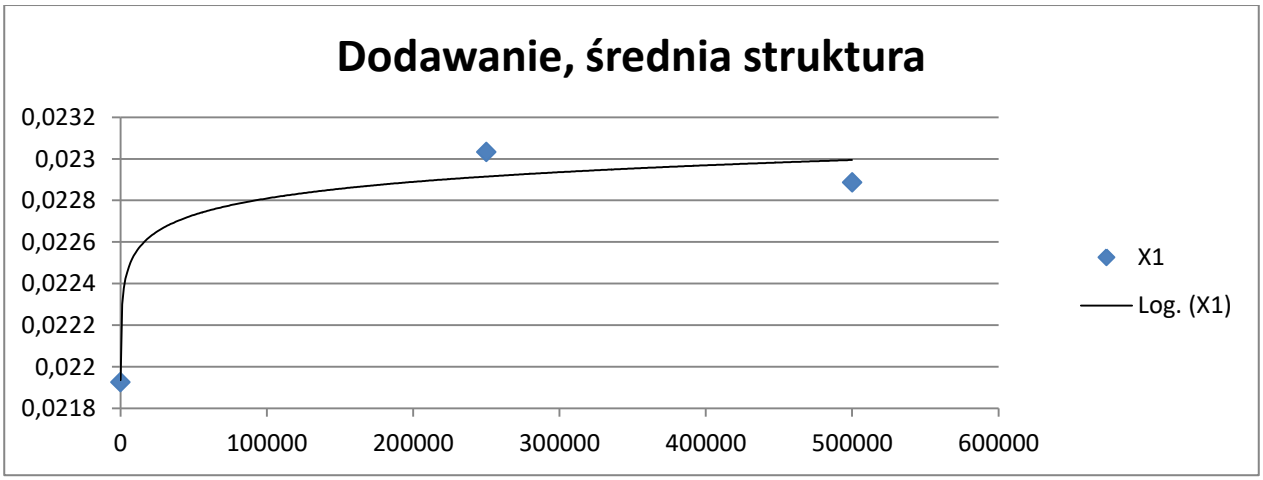
Dodawanie do drzewa RB:

DODAWANIE				
lp.	Liczba elementów	Wielkość liczb	Średnia czasu	Średnia OLE
1	50	M	0,02024664	0,022579213
2	50	S	0,0221127	
3	50	D	0,0253783	
4	250 000	M	0,0219261	0,022614773
5	250 000	S	0,02303232	
6	250 000	D	0,0228859	
7	500 000	M	0,02743092	0,02830174
8	500 000	S	0,02883044	
9	500 000	D	0,02864386	

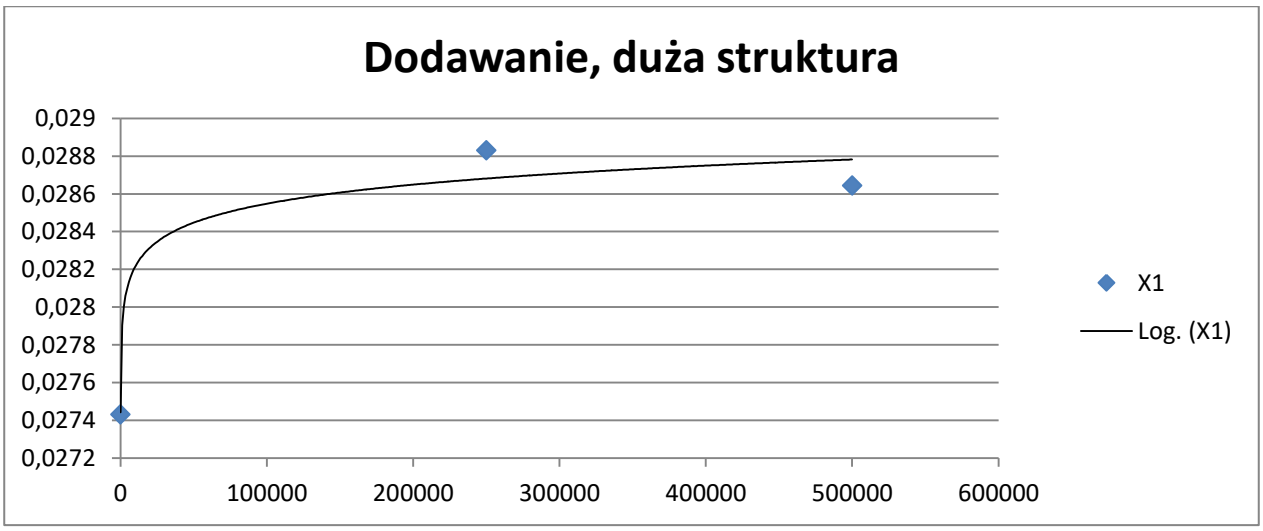
Tabela 7 Dodawanie nowego elementu do drzewa Czerwono-czarnego



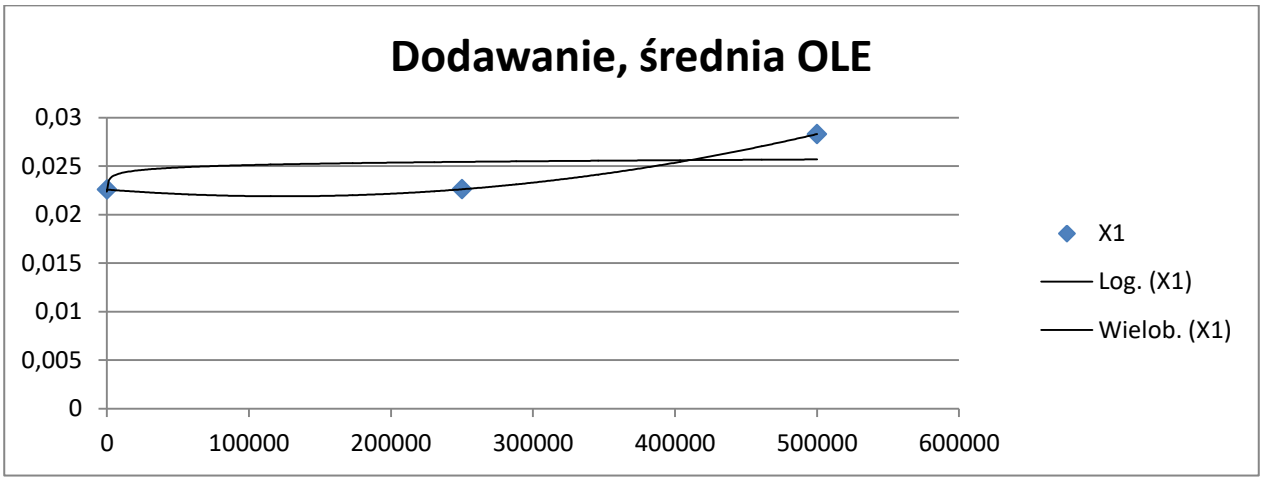
Wykres 25 Dodawanie do drzewa RB dla liczby elementów: 50



Wykres 26 Dodawanie do drzewa RB dla liczby elementów: 250 000



Wykres 27 Dodawanie do drzewa RB dla liczby elementów: 500 000

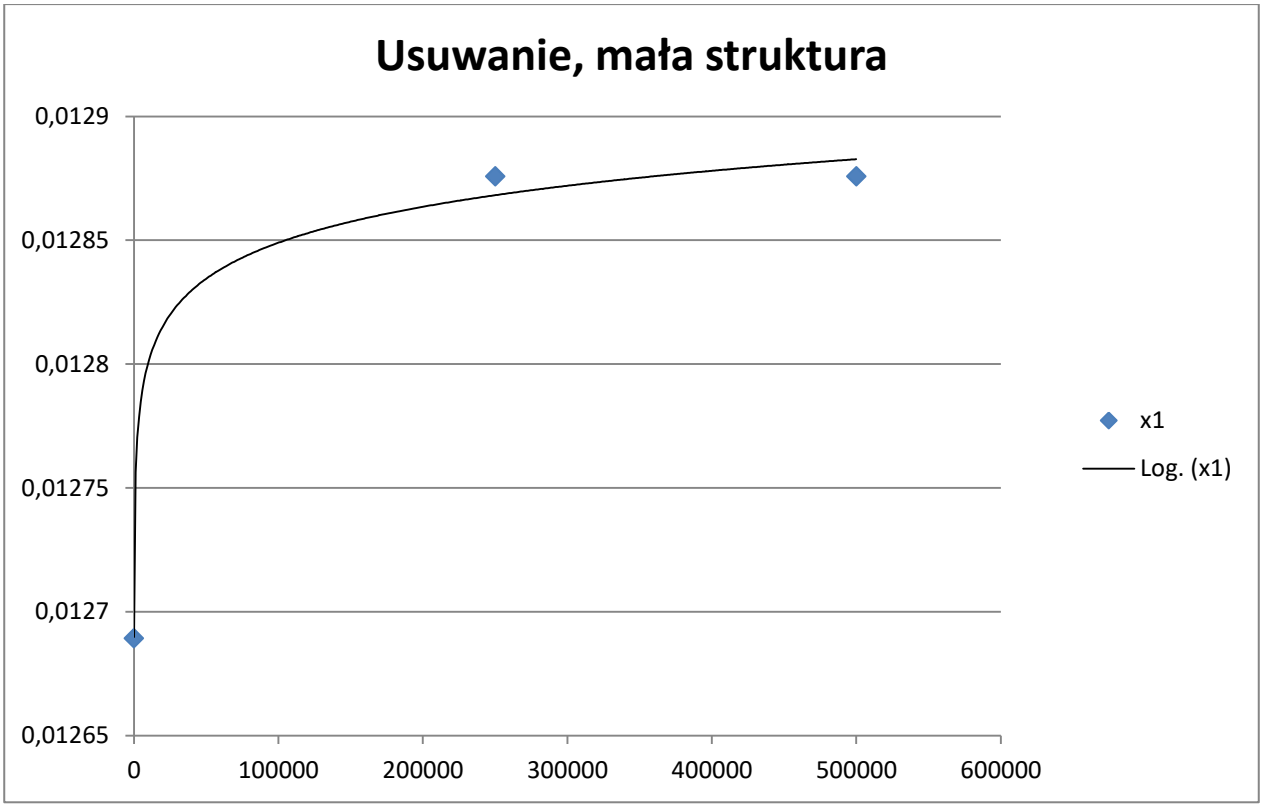


Wykres 28 Wykres przedstawia zależność czasu operacji dodawania dla średniej (pomiaru czasu) od liczby elementów, z naniesioną logarytmiczną oraz wielomianową linią trendu

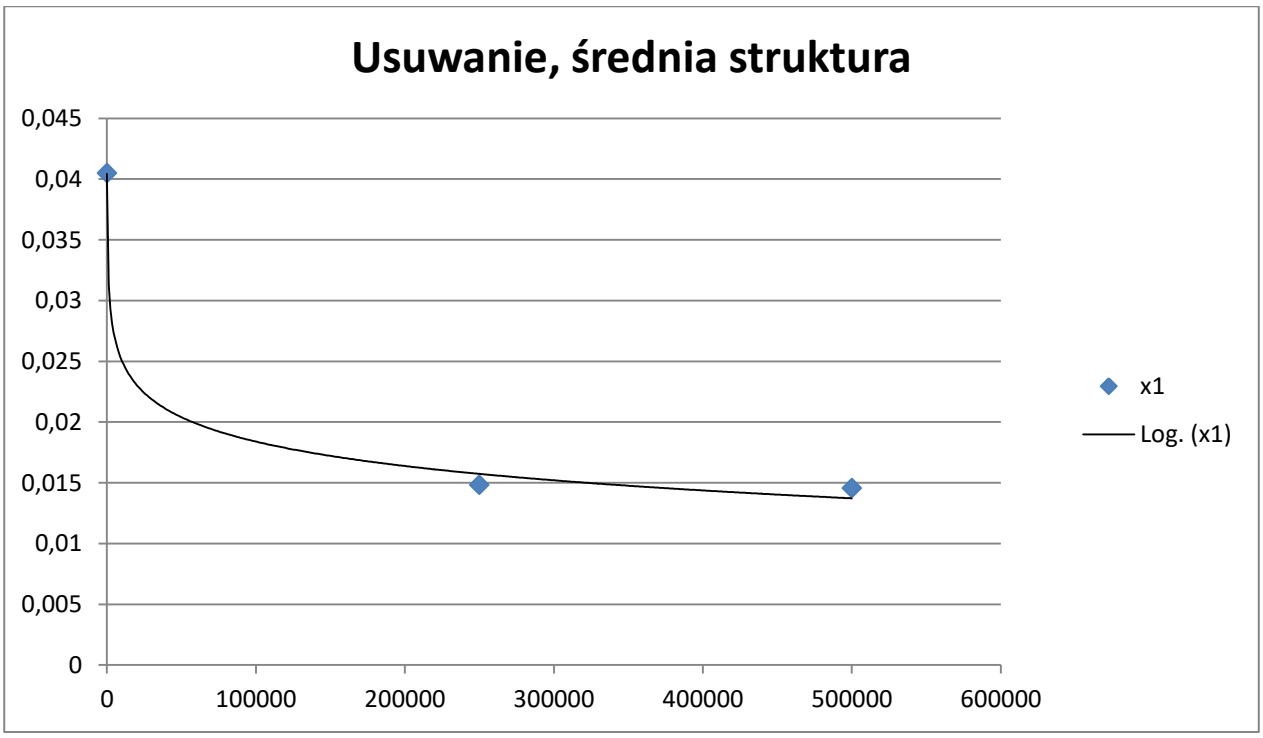
Usuwanie:

USUWANIE				
lp.	Liczba elementów	Wielkość liczb	Średnia czasu	Średnia OLE
1	50	M	0,01268914	0,01281354
2	50	S	0,01287574	
3	50	D	0,01287574	
4	250 000	M	0,0404933	0,023294533
5	250 000	S	0,0148351	
6	250 000	D	0,0145552	
7	500 000	M	0,0436656	0,024445267
8	500 000	S	0,0150217	
9	500 000	D	0,0146485	

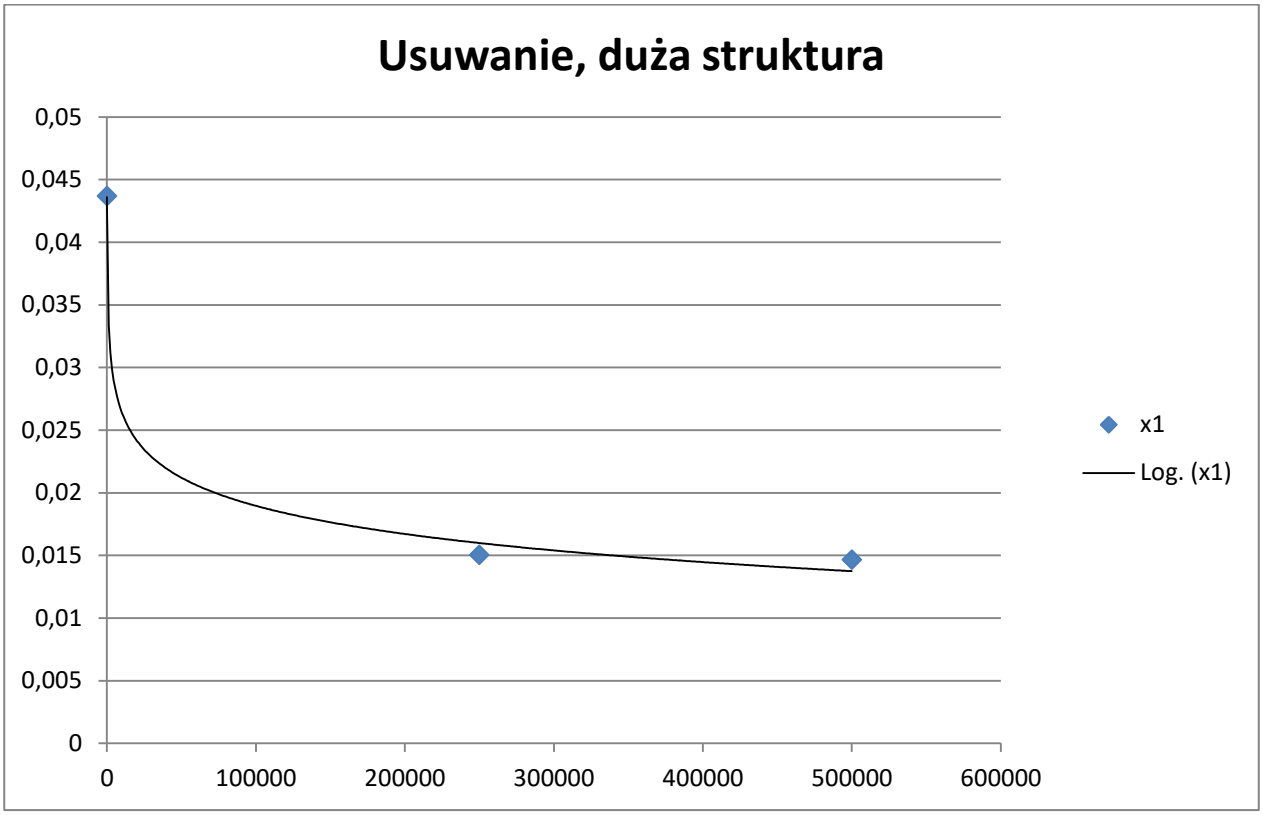
Tabela 8 Usuwanie z z drzewa RB



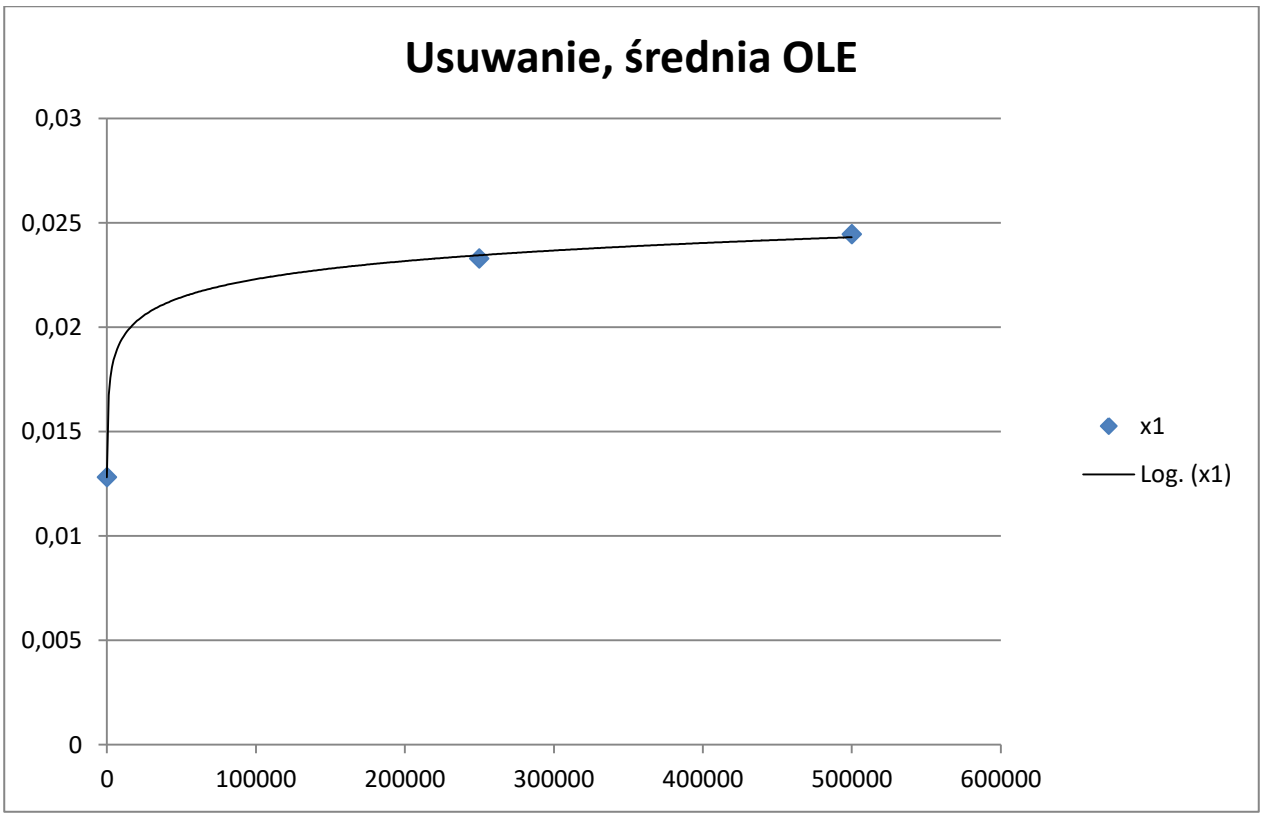
Wykres 29 Usuwanie z drzewa RB dla liczby elementów: 50



Wykres 30 Usuwanie z drzewa RB dla liczby elementów: 250 000



Wykres 31 Usuwanie z drzewa RB dla liczby elementów: 500 000

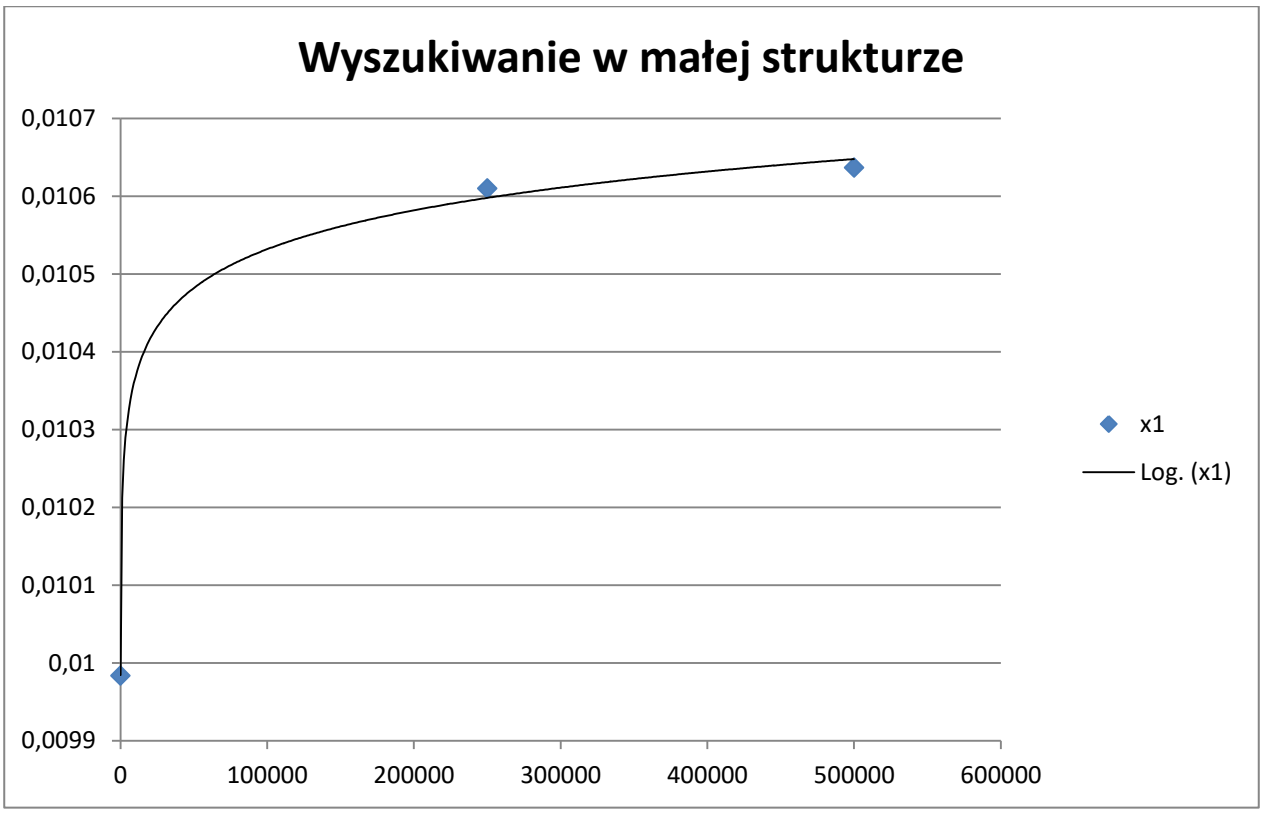


Wykres 32 Wykres przedstawia zależność czasu operacji usuwania dla średniej (pomiaru czasu) od liczby elementów

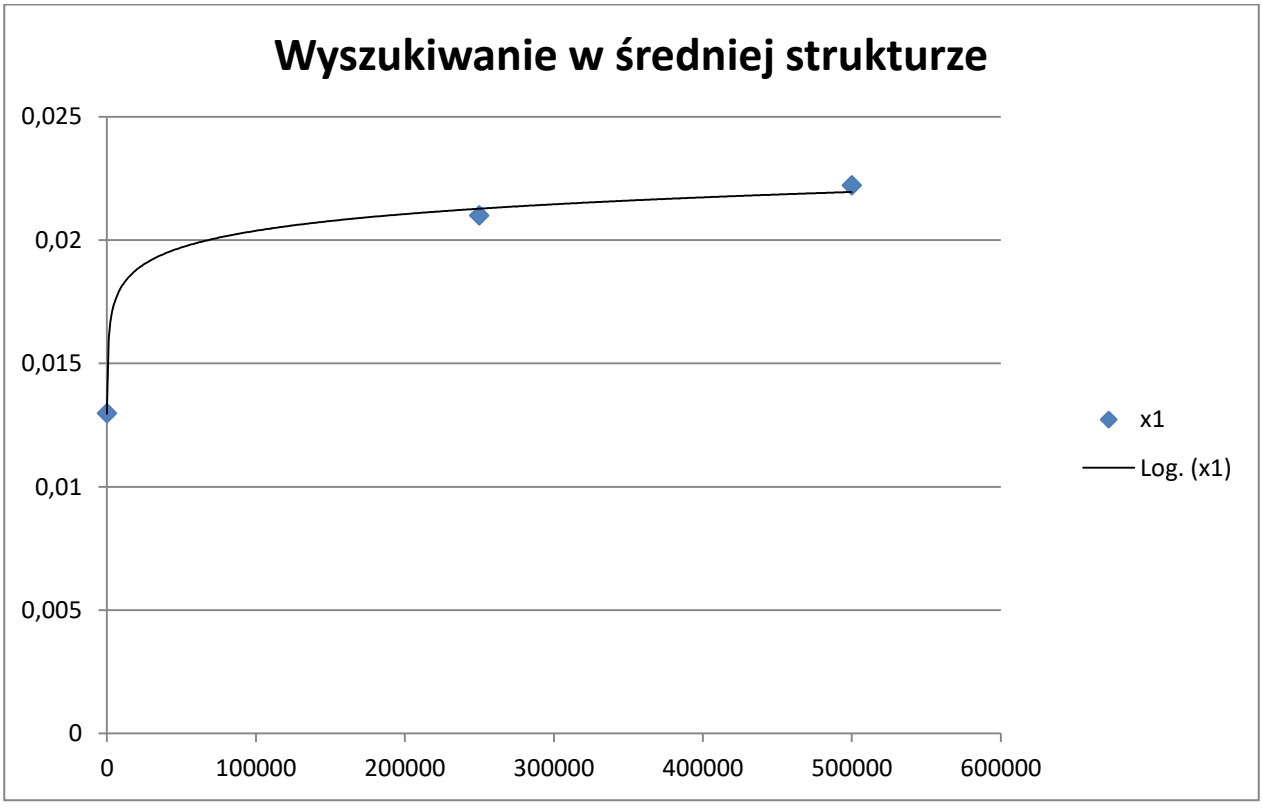
Wyszukiwanie:

WYSZUKAJ				
lp.	Liczba elementów	Wielkość liczb	Średnia czasu	Średnia OLE
1	50	M	0,00998338	0,010409915
2	50	S	0,010609874	
3	50	D	0,01063649	
4	250 000	M	0,01296908	0,018722714
5	250 000	S	0,020993062	
6	250 000	D	0,022206	
7	500 000	M	0,037414312	0,042981348
8	500 000	S	0,044411962	
9	500 000	D	0,04711777	

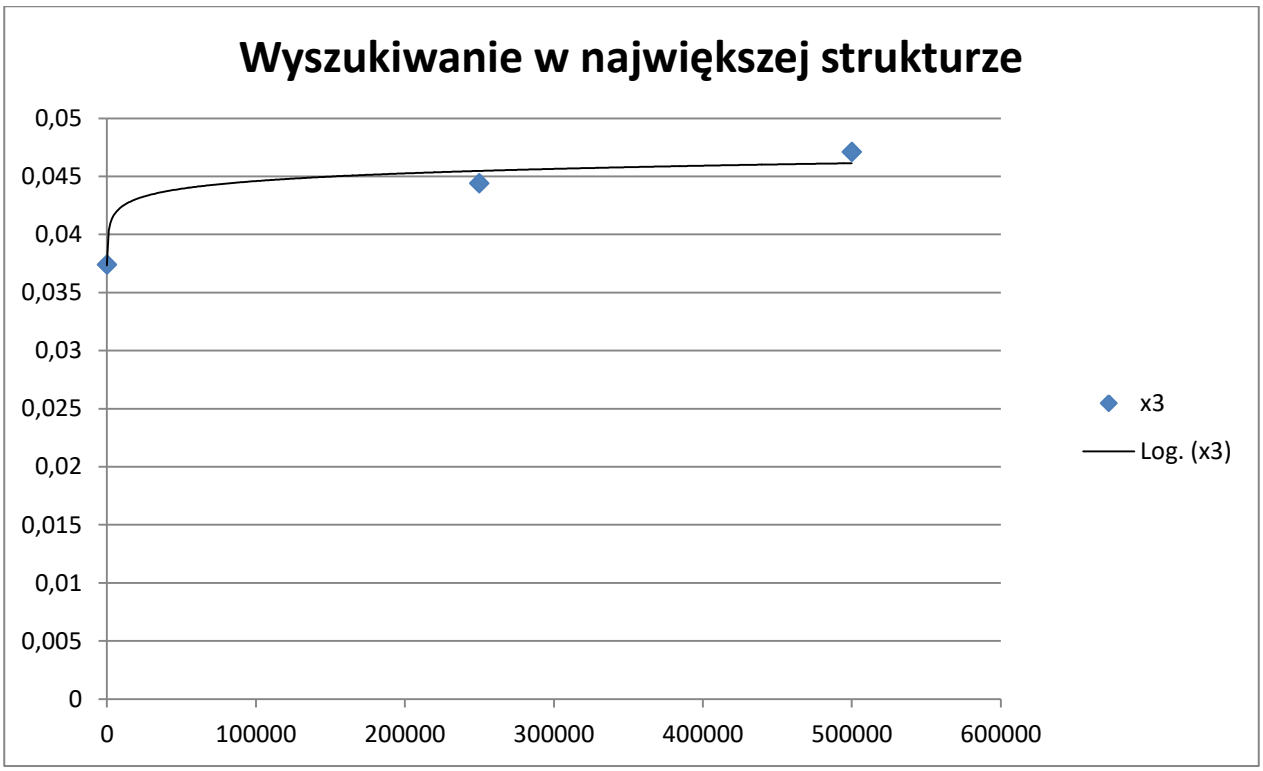
Tabela 9 Wyszukiwanie elementu w drzewie RB



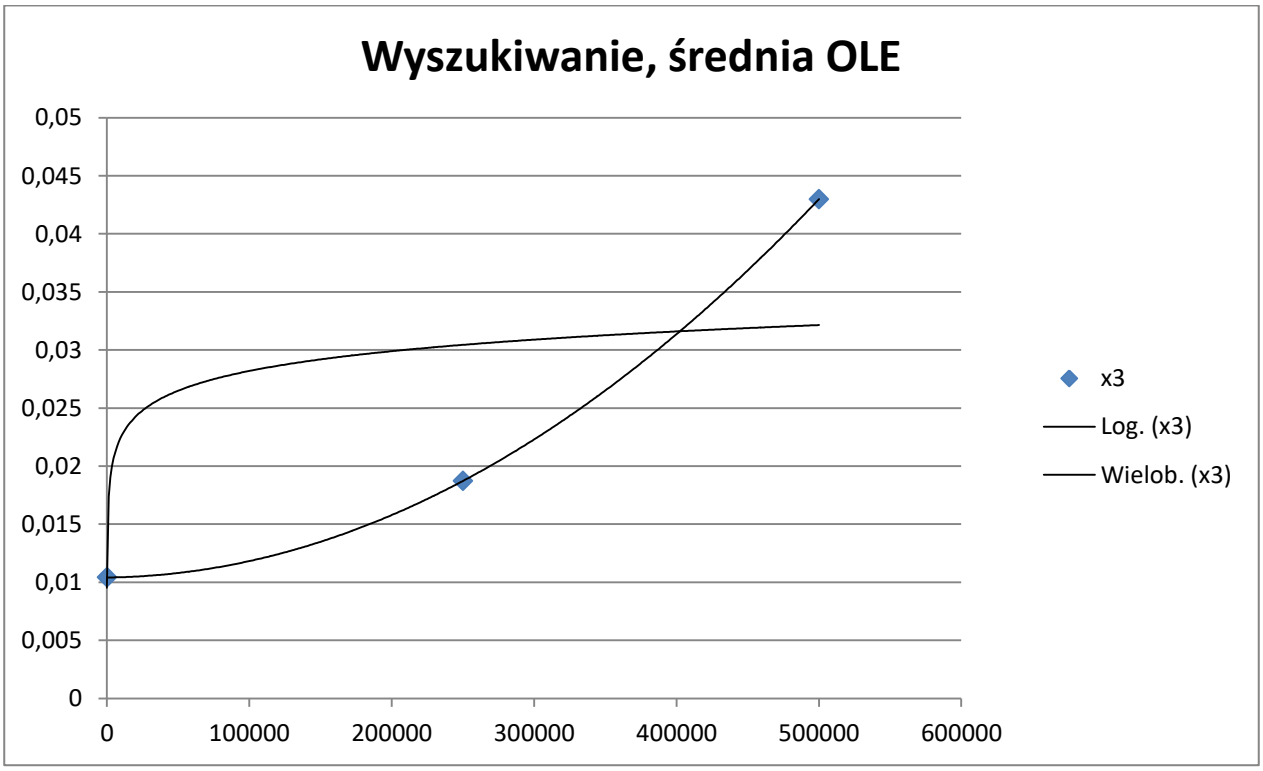
Wykres 33 Wyszukiwanie w drzewie RB dla liczby elementów: 50



Wykres 34 Wyszukiwanie w drzewie RB dla liczby elementów: 250 000



Wykres 35 Wyszukiwanie w drzewie RB dla liczby elementów: 500 000



Wykres 36 Wykres przedstawia zależność czasu operacji Wyszukiwania dla średniej (pomiaru czasu) od liczby elementów, z naniesioną logarytmiczną oraz wielomianową linią trendu

6. Wnioski

Kopiec Binarny:

Przy dodawaniu w kopcu binarnym dla małej struktury, której liczba elementów wynosi 50 elementów (wykres 1), zauważyłem, że linia trendu naniesiona na ten wykres przypomina funkcję liniową i przypomina złożoność czasową O(n). Może to wynikać ze zbyt małej ilości elementów i w takim wypadku struktura zachowuje się jak przy złożoności czasowej przyjętej dla tablic, czyli O(n), gdyż struktura kopca oparta jest na tablicach. Dla większych struktur, przechowujących większą liczbę elementów, operacja dodawania (wykresy 2 i 3), dzięki linii trendu naniesionej na wykres można oszacować, że rzeczywiście złożoność czasowa dla tej struktury wynosi *O(logn),* tak jak przyjęto teoretycznie. Podobnie również można wywnioskować obserwując wykres średniej od liczby elementów dla dodawania, czyli wykresu nr. 4, operacja dodawania w strukturze kopca binarnego przyjmuje złożoność czasową O(logn).

Usuwanie z kopca binarnego – wykresy 5,6,7 zachowują się zgodnie z przewidywaniami i można oszacować, że złożoność czasowa takiej operacji wynosi tak jak przyjęto we wstępie teoretycznym czyli O(logn). Natomiast, jeżeli spojrzymy na wykres usuwania z kopca binarnego – wykres 8, w zależności od wielkości struktury widać, że operacje te cechuje złożoność czasowa O(n), a nie O(logn), co doskonale widać, dzięki naniesionej linii trendu.

Podczas badania wyszukiwania w kopcu binarnym – wykres 9,10,11,12 – wszystkie te wykresy wykazują, że złożoność czasowa takiej operacji wynosi O(n), podobnie jak w przypadku tablic. Niewielkie odchylenia od linii trendu na wykresach mogą wynikać z pozycji klucza, który był wyszukiwany, gdyż podobnie jak przy tablicach, aby odnaleźć dany element, musimy przesunąć się po całej tablicy zaczynając od początku, jeżeli element znajdował się na końcu to operacja ta zajmie kilka mikrosekund więcej.

Badając tą strukturę, można było zaobserwować, że złożoności czasowe przyjęte w opracowaniu teoretycznym zgadzały się z otrzymanymi wynikami pomiarów.

Drzewo BST:

Podczas dodawania do drzewa BST, można zauważyć dla mniejszej struktury – wykres 13, że operacja dodawania zajęła więcej czasu dla najmniejszej wielkości liczb w zestawie danych, a mniej dla liczb z zakresu -1000000 – 1000000 oraz -2000000 – 2000000. Próby powtórzyłem wielokrotnie, aby zminimalizować ryzyko przypadkowego błędu pomiaru, lecz za każdym razem wykres wyglądał tak samo. Takie zachowanie, mogę jedynie wytłumaczyć, tym że czas takich operacji to ok. 0,0127 ms., a gdy w zestawie danych dany klucz powtarza się to zgodnie z zasadą dodawania nowego elementu do drzewa procedura wydłuża się tzn. przechodzimy pomiędzy prawym lub lewym synem do momentu, aż dany syn nie istnieje. Jeżeli struktura liczby 50 elementów i liczby z zakresu -2000000 – 2000000 to prawdopodobieństwo powtórzenia się danego klucza jest bardzo małe. Z tego powodu operacja zajmuje bardzo niewiele, lecz mniej czasu.

Podczas dodawania dla większych struktur danych – wykresy 14,15 oraz wykres 16 dla średniej OLE można zauważyć, że struktura zachowuje się zgodnie z założeniami i złożoność czasowa operacji dodawania do drzewa BST jest O(log n).

Różnice w czasie dodawania dla najmniejszego i największego zestawu danych są bardzo niewielkie, co świadczy o wydajności i efektywności tego algorytmu.

Usuwanie z drzewa BST dla wykresów 17,18,19,20 – na wszystkich tych wykresach linia trendu wykazuje, że złożoność czasowa tych operacji wynosi O(log n), tak jak przyjęto teoretycznie. Na wykresie 20, dla średniej OLE widać niewielkie różnice względem linii trendu.

Wyszukiwanie w drzewie BST, dla wykresów 21 oraz 23 można zauważyć, że wartości elementów nieco różnią się względem linii trendu, lecz gdy przyjrzymy się z jakimi czasami zostały zbadane dane struktury okazuje się, że są to różnice nanosekund, a więc bardzo niewielkie różnice, wręcz pomijalne, a złożoność czasowa wynosi O(log n). Podobnie, sprawa wygląda dla wykresu 23, czyli największej badanej struktury oraz wykresu 24 przedstawiającego zależność czasową operacji wyszukiwania dla średniej OLE. Naniesiona logarytmiczna linia trendu nie pokrywa się idealnie ze zmierzonymi czasami, naniosłem więc dla porównania wielomianową linię trendu, która idealnie łączy wszystkie punkty. Ze względu jednak na to, że różnica w wyszukiwaniu pomiędzy najmniejszą a największą strukturą wynosi mniej niż 0,0003ms. przyjmuje, że złożoność czasowa dla tej operacji wynosi O(log n) i jest to bardzo wydajny i efektywny algorytm.

Drzewo Czerwono-Czarne

Podczas badania tej struktury pod względem dodawania do niej nowego klucza można zauważyć na wykresach 25,26,27, że złożoność czasowa tej operacji wynosi O(log n), również naniesione linie trendu, które niemal idealnie pokrywają się z naniesionymi punktami pomiarowymi. Wyniki zgadzają się z tymi przyjętymi teoretycznie, jednak gdy spojrzymy na wykres 28 dla średniej OLE widzimy, że czas ten narasta w sposób wielomianowy, a nie logarytmiczny. Oznacza to, że wraz ze wzrostem liczby elementów w strukturze czas nie rośnie logarytmicznie, może to jednak wynikać ze zbyt małej ilości danych pomiarowych.

Usuwanie w drzewie czerwono-czarnym jest operacją bardziej złożoną od dodawania. Składa jest z wielu warunków, które musimy sprawdzić w przypadku usunięcia danego klucza. Widzimy na wykresie 29, że dla małej struktury czas ten narasta logarytmicznie, natomiast dla wykresów 30 i 31 zmniejsza się logarytmicznie. Czas operacji dla zestawu danych zawierającego 250 000 i 500 000 elementów oraz liczby z zakresu -100 – 100 widać, że czas ten jest większy niż dla pozostałych wielkości liczb. Może to wynikać z równoważenia się drzewa oraz tym, że usuwany klucz w tych zestawach danych powtarza się wielokrotnie co wydłuża czas operacji, natomiast gdy zakres liczb się zwiększa zmniejsza się prawdopodobieństwo powtórzenia tego samego klucza, przez co drzewo składa się z bardziej unikalnych kluczy przez co może wykonywać mniejszą liczbę kroków w celu równoważenia się drzewa. Na wykresie 32, dla średniej OLE widać, że logarytmiczna linia trendu pokrywa się niemal idealnie z punktami pomiarowymi. Wynika z tego, że zgodnie z teorią, pesymistycznie przyjęta złożoność czasowa dla tej operacji, czyli O(log n) jest zgodna z otrzymanymi wynikami.

Podczas wyszukiwania zadanego klucza w drzewie czerwono-czarnym można zauważyć, że wykresy 33,34,35 wykazują, że złożoność czasowa takiej operacji wynosi O(log n).

Podobnie również jak dla drzewa BST, na wykresie 36 widać, że wraz ze wzrostem wielkości struktury pod względem ilości elementów czas operacji zwiększa się wielomianowo, a nie logarytmicznie.

Zauważyłem również, że czas wyszukiwania w drzewie czerwono-czarnym był większy od tego zmierzonego podczas badania drzewa BST.

Funkcja pomiaru czasu użyta w zadaniu laboratoryjnym, sprawia że czas pomiaru zależy również od wydajności sprzętu, na którym przeprowadzono badania. Mogło to mieć wpływ, na niektóre uzyskane wyniki.

7. Spis tabel i wykresów

Bibliografia:

Cormen Thomas G., Leiserson Charles E., Rivest Ronald L., *Wprowadzenie do Algorytmów,* wyd. 4, Warszawa: WNT, 1997

Wykaz stron internetowych:

http://www.algorytm.org/

http://cpp0x.pl/

http://www.p-programowanie.pl/

http://eduinf.waw.pl/inf/alg/001_search/index.php