Struktury Danych i Złożoność Obliczeniowa

Zadanie projektowe nr 1: Badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w różnych strukturach danych.

1.Wstęp teoretyczny

Czasową złożoność obliczeniową określamy jako ilość czasu niezbędnego do rozwiązania problemu w zależności od liczby danych wejściowych. Złożoność czasową oznaczamy jako O(n).

1.1 Tablica

a) usuwanie i dodawanie

- początek tablicy

Usunięcie elementu z początku tablicy wymaga przesunięcia każdego elementu tablicy na poprzednie miejsce. Natomiast dodawanie polega na przesunięcie każdego elementu na następne miejsce i wstawienie nowego na początek. Złożoność czasowa takiego algorytmu to O(n) – zależy wprost proporcjonalnie od ilości elementów. Dodatkową ilość czasu może zająć relokacja pamięci w celu jej zaoszczędzenia.

- koniec tablicy

Zarówno czas usunięcia elementu z końca tablicy, jak i dodania elementu na koniec jest stały – złożoność obliczeniowa O(1). Jednak dodatkowy czas może zająć relokacja pamięci, której złożoność zależy od ilości elementów.

- dowolne miejsce w tablicy

Czas wykonania operacji zależy od położenia modyfikowanego elementu w tablicy. W założeniu pesymistycznym złożoność obliczeniowa wynosi O(n).

b) wyszukiwanie

W najgorszym wypadku (jeśli szukany element jest na końcu) algorytm wyszukiwania elementu o podanej wartości w tablicy ma złożoność czasową O(n).

1.2. Lista dwukierunkowa

a) usuwanie i dodawanie

- początek/koniec listy

Czas usunięcia lub dodania elementu będzie stały – złożoność obliczeniowa O(1).

- dowolne miejsce na liście

Czas wykonania danej operacji zależy od położenia modyfikowanego elementu. Aby go znaleźć, należy przejść listę do poprzednika elementu (co w najgorszym wypadku będzie miało złożoność obliczeniową O(n)), a następnie dokonać usunięcia lub wstawienia.

b) wyszukiwanie

Aby znaleźć element o podanym kluczu, należy przejść listę do poprzednika elementu, co w najgorszym wypadku będzie miało złożoność obliczeniową O(n).

1.3. Kopiec

a) usuwanie i dodawanie elementu o podanym kluczu

Zarówno dodanie, jak i usunięcie elementu może wymagać odtworzenia struktury kopca w przypadku naruszenia warunku kopca. Zatem w pesymistycznym przypadku złożoność obliczeniowa zależy od liczby poziomów drzewa i wyniesie $O(n*\log_2 n)$. Jeśli warunek kopca nie zostałby naruszony, wtedy złożoność czasowa jest stała – element zostaje dodany do tablicy reprezentującej kopiec na ostatnim miejscu jako ostatni liść.

b) wyszukiwanie

Wyszukiwanie elementu w kopcu odbywa się tak samo jak wyszukiwanie w tablicy – ma złożoność obliczeniową najwyżej O(n).

1.4. Drzewo czerwono – czarne

Dzięki zrównoważonej strukturze, która gwarantuje, że drzewo o n wewnętrznych węzłach nigdy nie będzie miało więcej poziomów niż $2*\lg_2(n-1)$, wszystkie operacje wykonywane na drzewie mają pesymistyczną złożoność czasową $O(\lg_2 n)$.

2. Plan eksperymentu

- tablica oraz tablica reprezentująca kopiec są dynamiczne i relokowane przy usuwaniu i dodawaniu elementów, w niektórych przypadkach w eksperymencie nie brano pod uwagę czasu relokacji w celu uzyskania dokładniejszych wyników
- każda operacja jest testowana na strukturze zawierającej 20 000 lub mniej elementów. Dla każdego pomiaru generowana jest nowa populacja. Wyniki z 20 pomiarów są uśredniane
- elementem struktur jest liczba typu *integer* generowana losowo z zakresu <-1000, 1000> za pomocą funkcji *rand()*
- pomiarów czasu dokonano za pomocą funkcji
  *BOOL QueryPerformanceCounter (__out LARGE_INTEGER *lpPerformanceCount);*

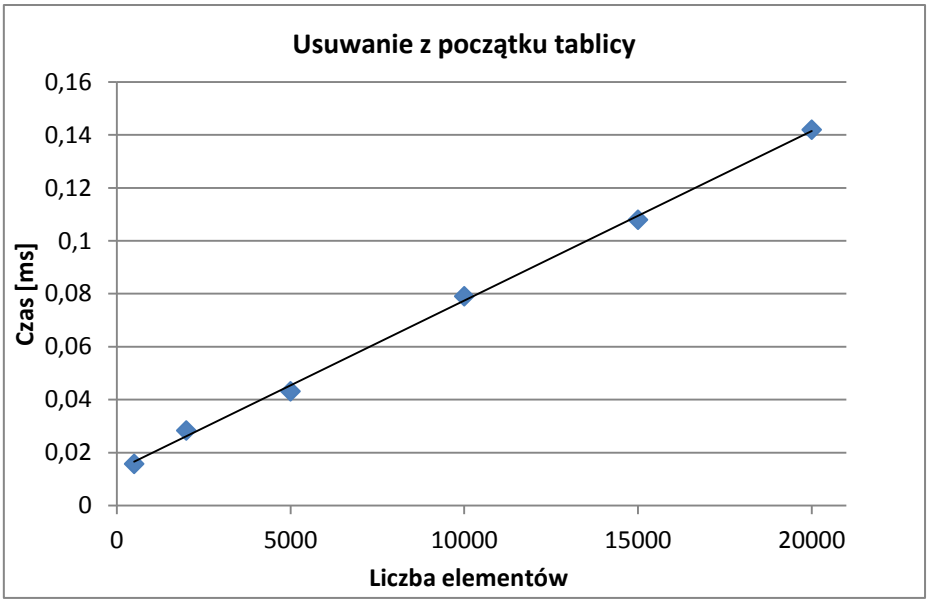
3. Wyniki eksperymentu

3.1 Tablica

a) usuwanie z początku tablicy

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,0158
2.	2000	0,0284
3.	5000	0,0432
4.	10000	0,0791
5.	15000	0,1080
6.	20000	0,1420

Tabela 1 – uśrednione czasy usuwania elementu z początku tablicy wraz z relokacją pamięci

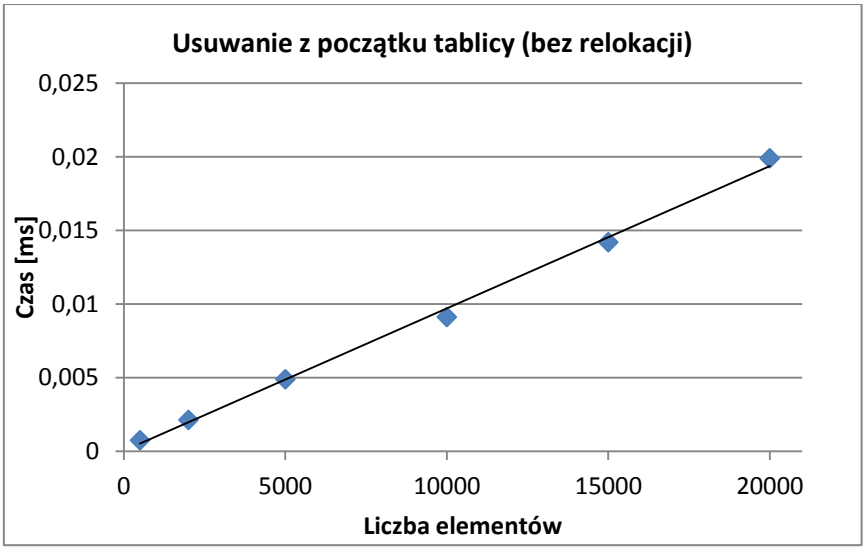


Wykres 1 – zależność czasu usuwania elementu z początku tablicy od liczby elementów (wraz z relokacją pamięci)

Z wyników przedstawionych powyżej (tabela 1, wykres 1) wynika, że czas usuwania elementu z początku tablicy jest wprost proporcjonalny do liczby elementów w strukturze. Ponieważ w przypadku wykorzystania relokacji pamięci oczywistym było, że czas będzie rósł wprost proporcjonalnie w stosunku do liczby elementów, pomiary powtórzono na algorytmie nie relokującym pamięci.

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,00075
2.	2000	0,00213
3.	5000	0,00489
4.	10000	0,00912
5.	15000	0,01420
6.	20000	0,01990

Tabela 2 - uśrednione czasy usuwania elementu z początku tablicy bez relokacji pamięci

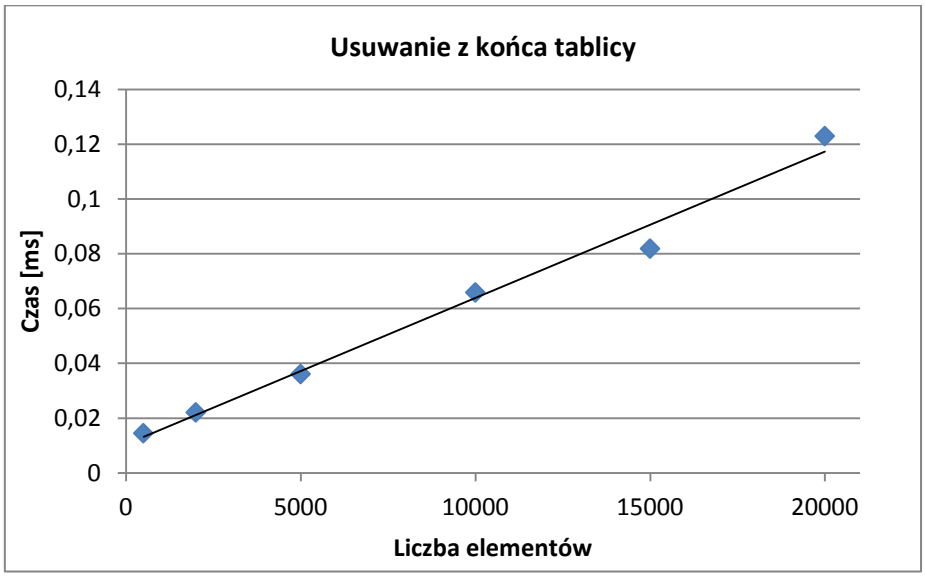


Wykres 2 – zależność czasu usuwania elementu od liczby elementów

b) usuwanie z końca tablicy

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,0145
2.	2000	0,0221
3.	5000	0,0361
4.	10000	0,0659
5.	15000	0,0819
6.	20000	0,1230

Tabela 3 - uśrednione czasy usuwania elementu z końca tablicy wraz z relokacją pamięci

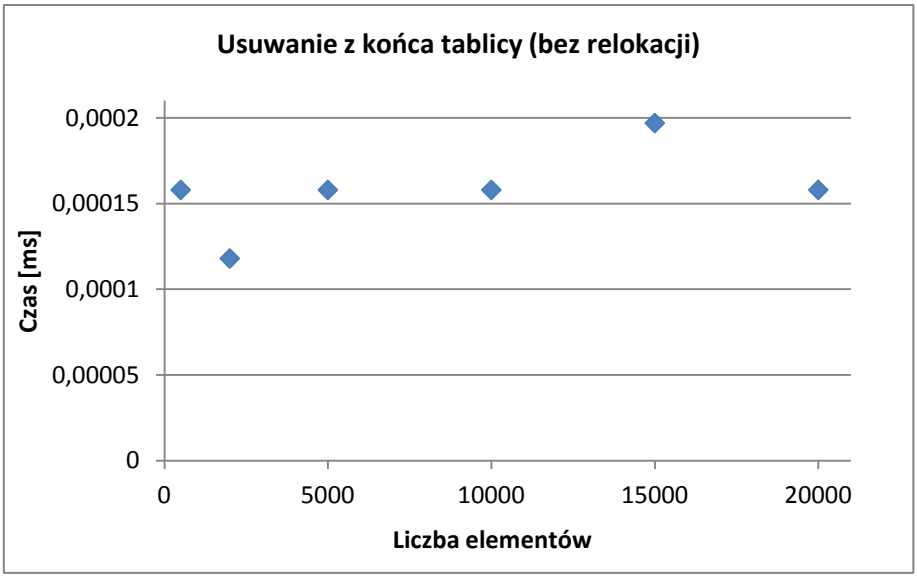


Wykres 3 - zależność czasu usuwania elementu z końca tablicy od liczby elementów (wraz z relokacją pamięci)

Ponieważ na podstawie powyższych pomiarów (tabela 3, wykres 3) ze względu na używanie relokacji pamięci nie można jednoznacznie ustalić zależności między czasem wykonania operacji usuwania, a liczbą elementów, pomiary powtórzono nie biorąc pod uwagę relokacji pamięci.

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,000158
2.	2000	0,000118
3.	5000	0,000158
4.	10000	0,000158
5.	15000	0,001970
6.	20000	0,000158

Tabela 4 - uśrednione czasy usuwania elementu z końca tablicy bez relokacji pamięci



Wykres 4 - zależność czasu usuwania elementu z końca tablicy od liczby elementów

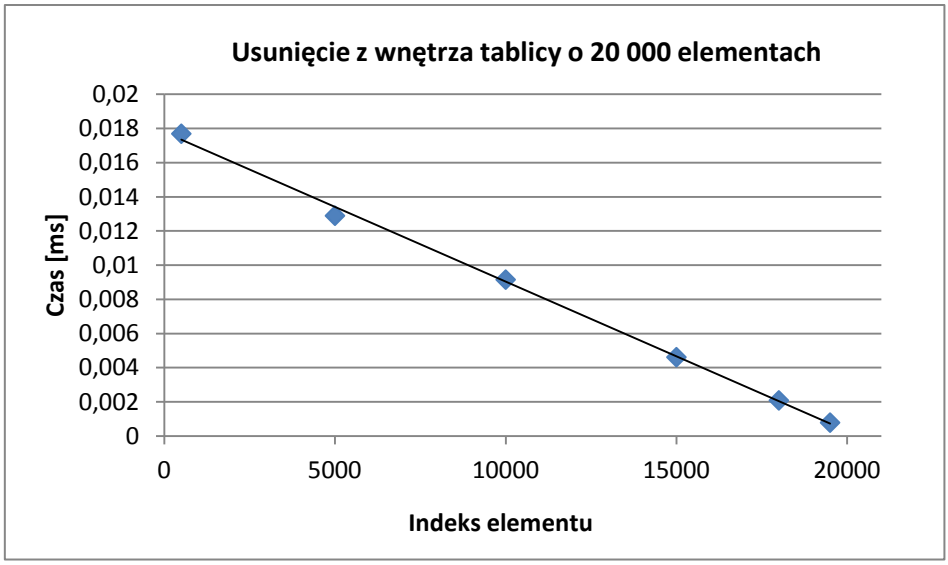
Na podstawie pomiarów czasu usuwania elementu z końca tablicy bez relokowania pamięci (wykres 4) można stwierdzić, że czas nie zależy od ilości elementów w strukturze i złożoność obliczeniowa tej operacji wynosi O(1).

c) usuwanie elementu z wnętrza tablicy

Pomiary czasu przeprowadzono dla struktury o 20 000 elementach. Nie relokowano pamięci. Usuwano elementy położone w różnych miejscach tablicy, indeksy dobrano tak, aby czasy można było porównać z czasami usuwania elementów z początku tablicy (tabela 2, wykres 2), to znaczy : usunięcie pierwszego elementu z tablicy o 5000 elementach powinno zająć w przybliżeniu tyle samo czasu, co usunięcie elementu o indeksie 15 000 z tablicy o 20 000 elementach (w obu przypadkach 4 999 elementów trzeba przesunąć na poprzednie miejsce).

L.p.	Indeks elementu	Średni czas z 20 pomiarów [ms]
1.	19500	0,00079
2.	18000	0,00209
3.	15000	0,00462
4.	10000	0,00916
5.	5000	0,01290
6.	500	0,01770

Tabela 5 – usuwanie elementu o danym indeksie z tablicy zawierającej 20 000 elementów, bez relokacji pamięci

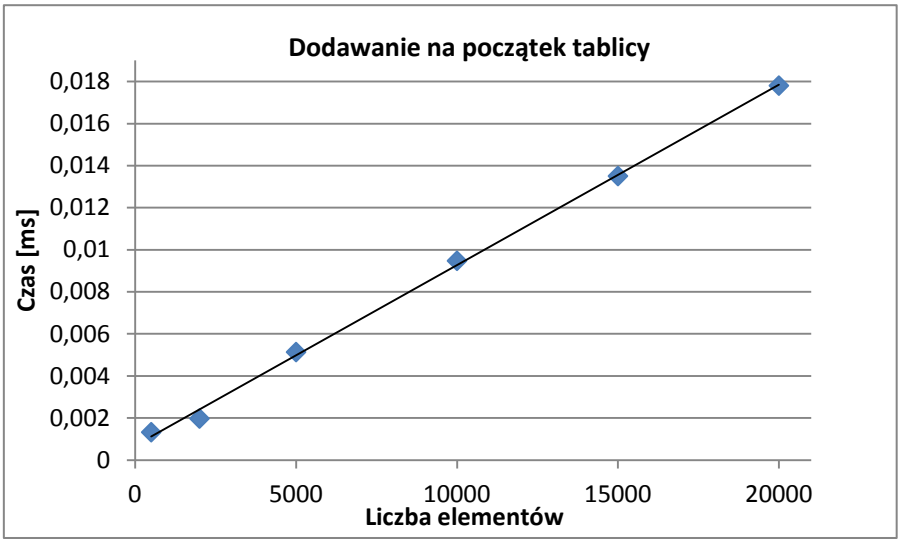


Wykres 5 - zależność czasu usuwania elementu z wnętrza tablicy od położenia tego elementu w tablicy, bez relokacji pamięci

d) dodawanie na początek tablicy

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,00132
2.	2000	0,00197
3.	5000	0,00513
4.	10000	0,00947
5.	15000	0,01350
6.	20000	0,01780

Tabela 6 – dodawanie elementu na początek tablicy bez relokacji pamięci

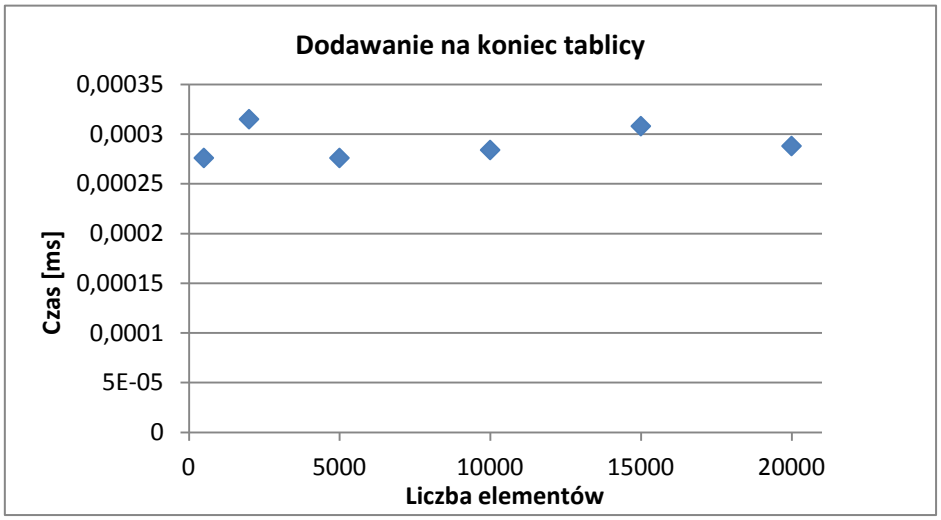


Wykres 6 – zależność czasu dodawania elementu na początek tablicy od liczby elementów w strukturze

e) dodawanie na koniec tablicy

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,000276
2.	2000	0,000315
3.	5000	0,000276
4.	10000	0,000284
5.	15000	0,000308
6.	20000	0,000288

Tabela 7 - dodawanie elementu na koniec tablicy bez relokacji pamięci

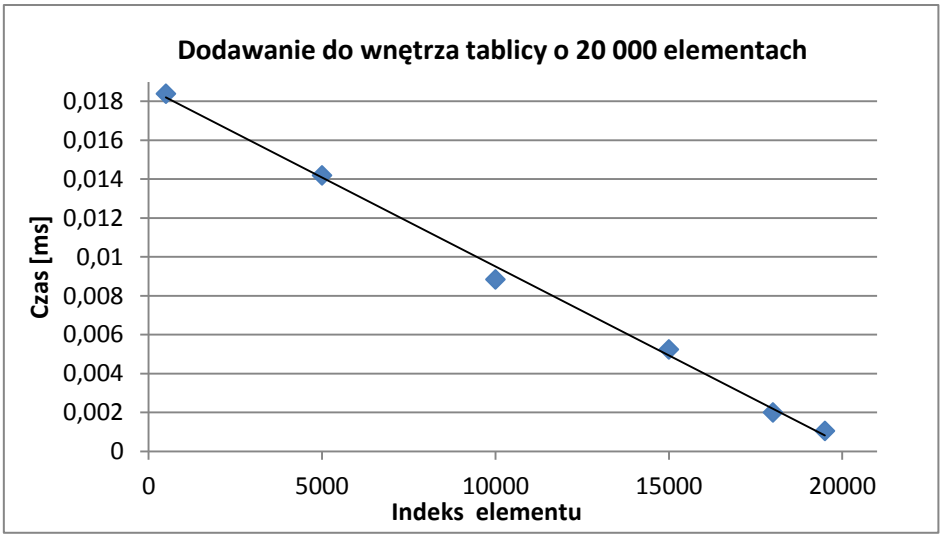


Wykres 7 - zależność czasu dodawania elementu na koniec tablicy od liczby elementów w strukturze

f) dodawanie do wnętrza tablicy

L.p.	Indeks elementu	Średni czas z 20 pomiarów [ms]
1.	19500	0,00106
2.	18000	0,00201
3.	15000	0,00525
4.	10000	0,00885
5.	5000	0,01420
6.	500	0,01840

Tabela 8 – dodawanie elementu do tablicy na zadanym miejscu



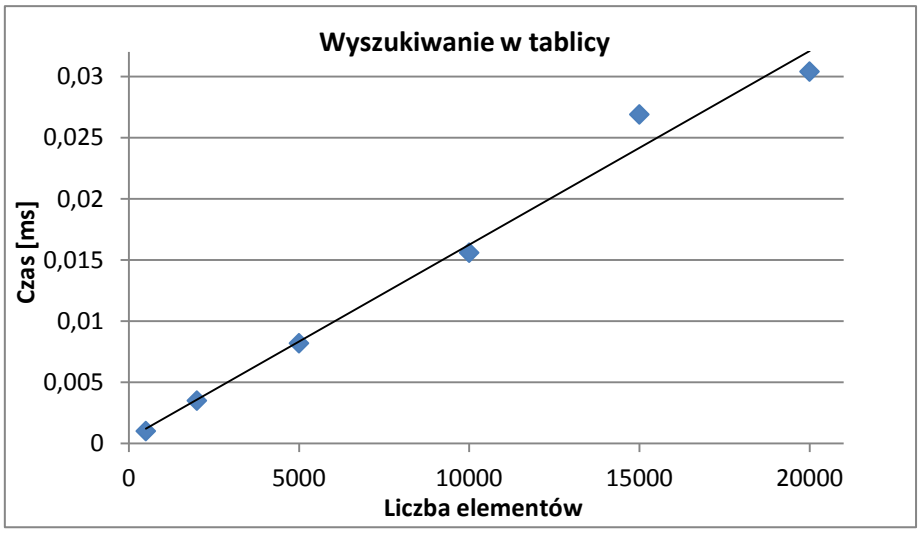
Wykres 8 – czas dodania elementu do tablicy w zależności od indeksu wstawianego elementu

g) wyszukiwanie

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,00102
2.	2000	0,00351
3.	5000	0,00820
4.	10000	0,01560
5.	15000	0,02690
6.	20000	0,03040

Tabela 9 - wyszukiwanie elementu w tablicy

W testach zadbano, aby połowa zadanych do wyszukania elementów mogła zostać znaleziona w strukturze, a połowa nie. Pesymistyczne założenie złożoności obliczeniowej O(n) jest prawdziwe, jeśli szukany element znajduje się na końcu struktury lub nie znajduje się w niej.



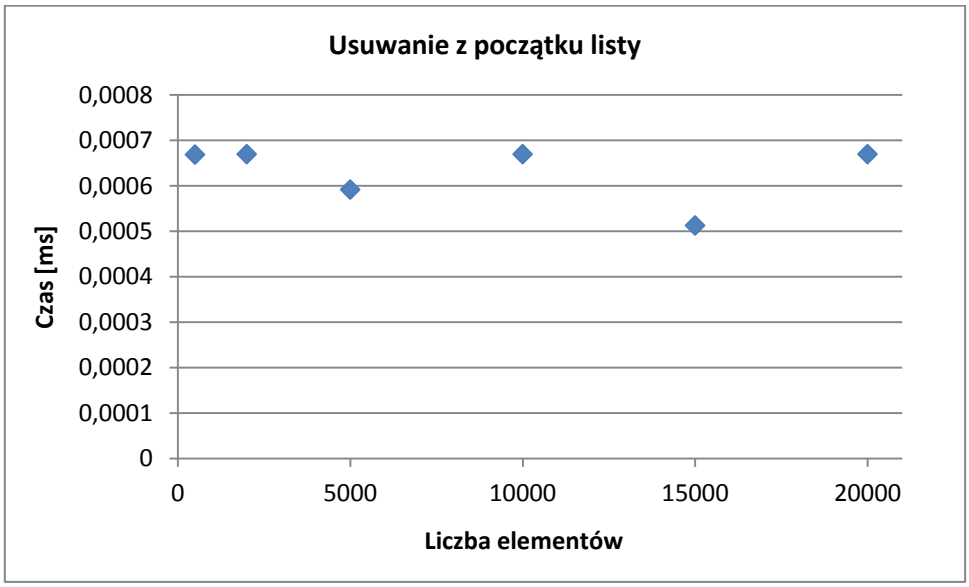
Wykres 9 – czas wyszukiwania elementu w tablicy w zależności od liczby elementów

3.2 Lista dwukierunkowa

a) usuwanie z początku listy

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,000669
2.	2000	0,000670
3.	5000	0,000592
4.	10000	0,000670
5.	15000	0,000513
6.	20000	0,000670

Tabela 10 – usuwanie elementu z początku tablicy

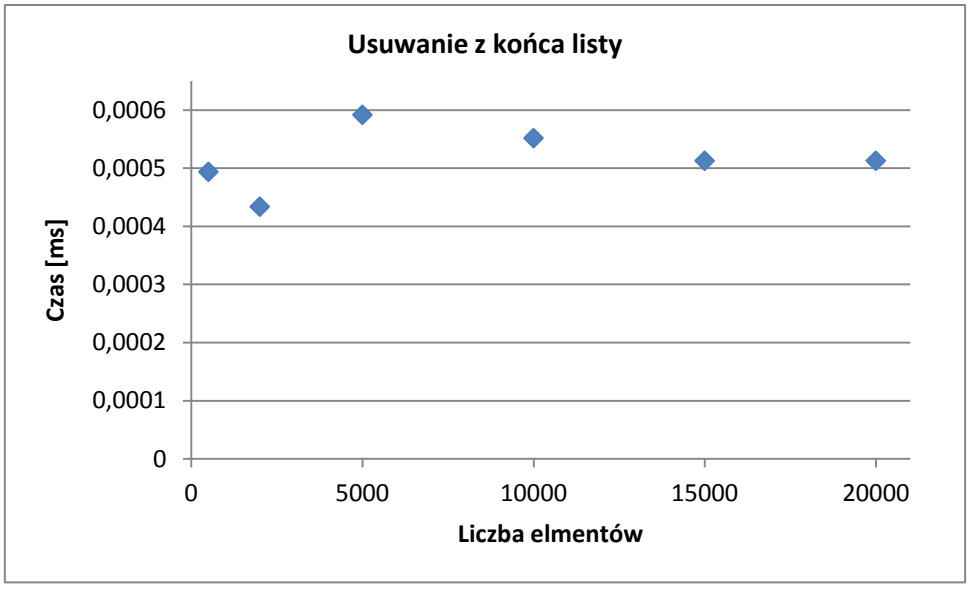


Wykres 10 – zależność czasu usuwania elementu z początku listy od liczby elementów w liście

b) usuwanie z końca listy

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,000494
2.	2000	0,000434
3.	5000	0,000592
4.	10000	0,000552
5.	15000	0,000513
6.	20000	0,000513

Tabela 11 – usuwanie elementu z końca listy

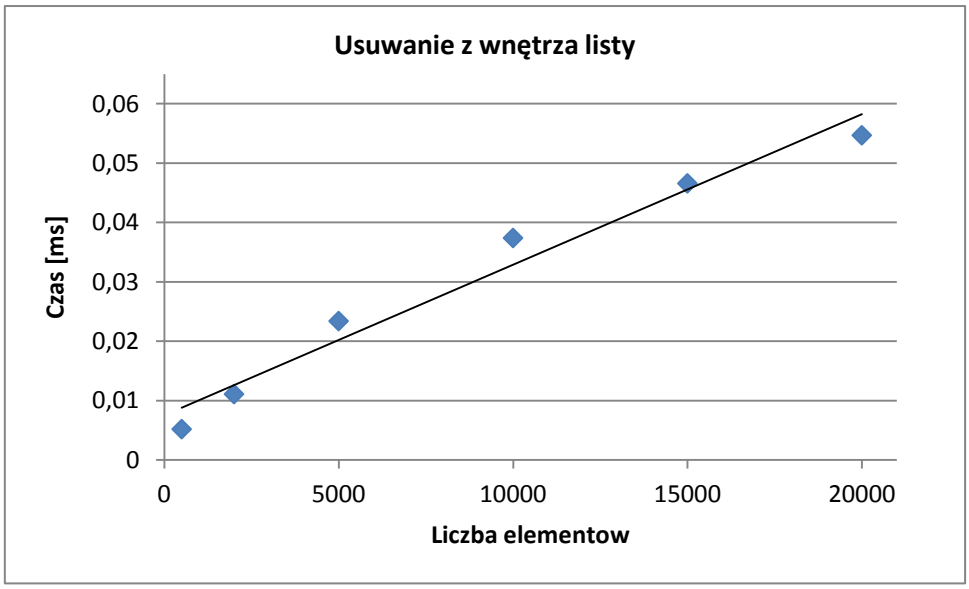


Wykres 11 - zależność czasu usuwania elementu z końca listy od liczby elementów w liście

c) usuwanie elementu o podanym kluczu

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,052
2.	2000	0,0111
3.	5000	0,0234
4.	10000	0,0374
5.	15000	0,0466
6.	20000	0,0547

Tabela 12 – usuwanie elementu z wnętrza listy

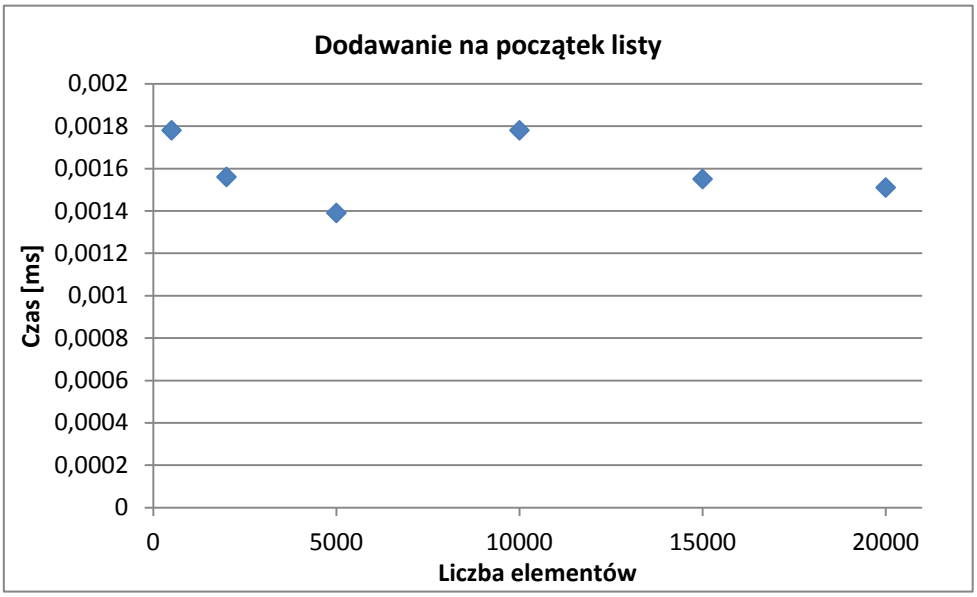


Wykres 12 – czas usuwania elementu z wnętrza listy w zależność od liczby elementów w niej przechowywanych

d) dodawanie na początek listy

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,00398
2.	2000	0,00406
3.	5000	0,00339
4.	10000	0,00178
5.	15000	0,00355
6.	20000	0,00351

Tabela 13 – dodawanie elementu na początek listy

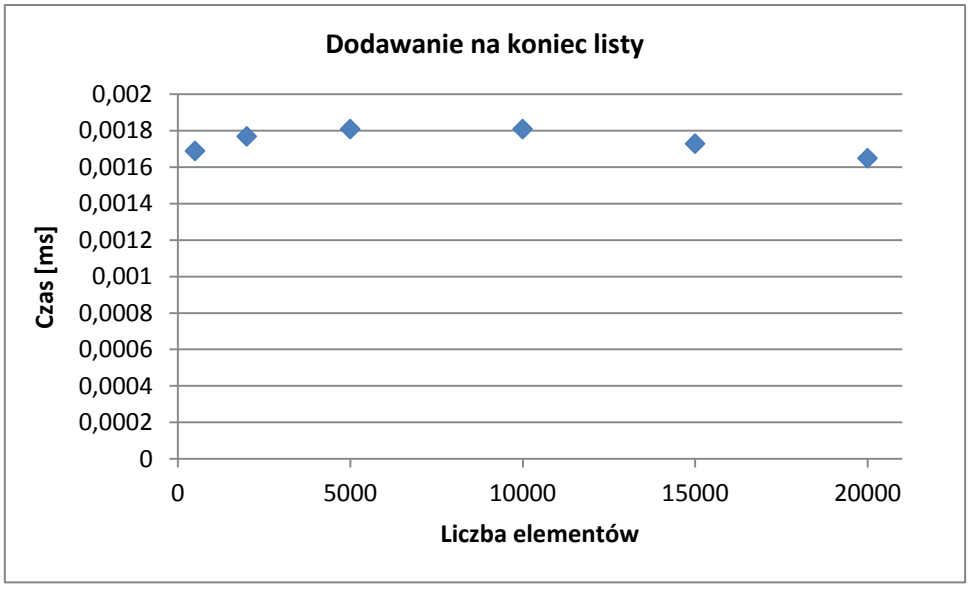


Wykres 13 – czas dodawania elementu na początek listy w zależności od liczby elementów w niej przechowywanych

e) dodawanie na koniec listy

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,00169
2.	2000	0,00177
3.	5000	0,00181
4.	10000	0,00181
5.	15000	0,00173
6.	20000	0,00165

Tabela 14 – dodawanie elementu na koniec listy

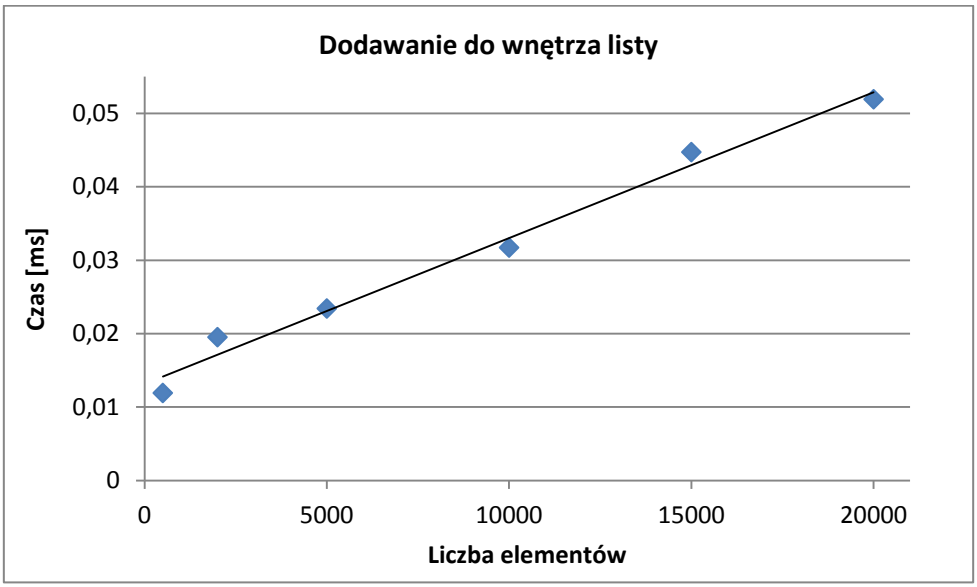


Wykres 14 - zależność czasu dodawania elementu na koniec listy od liczby elementów w strukturze

f) dodawanie za wybranym elementem

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,0119
2.	2000	0,0195
3.	5000	0,0234
4.	10000	0,0317
5.	15000	0,0447
6.	20000	0,0519

Tabela 15 – dodawanie elementu do wnętrza listy

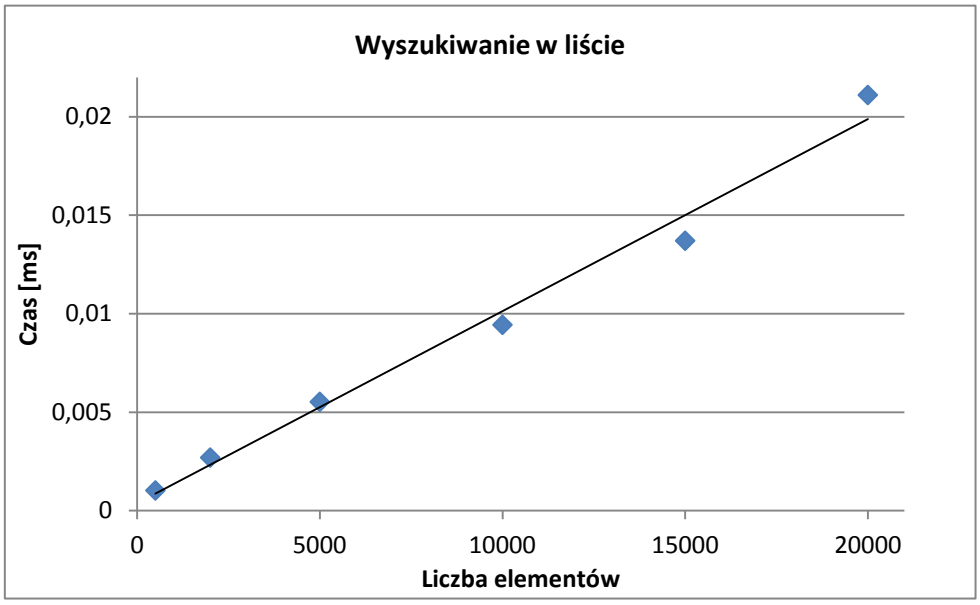


Wykres 15 - zależność czasu dodawania elementu do wnętrza listy od liczby elementów w strukturze

g) wyszukiwanie

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,00102
2.	2000	0,00269
3.	5000	0,00552
4.	10000	0,00943
5.	15000	0,0137
6.	20000	0,0211

Tabela 16 – wyszukiwanie elementu w liście



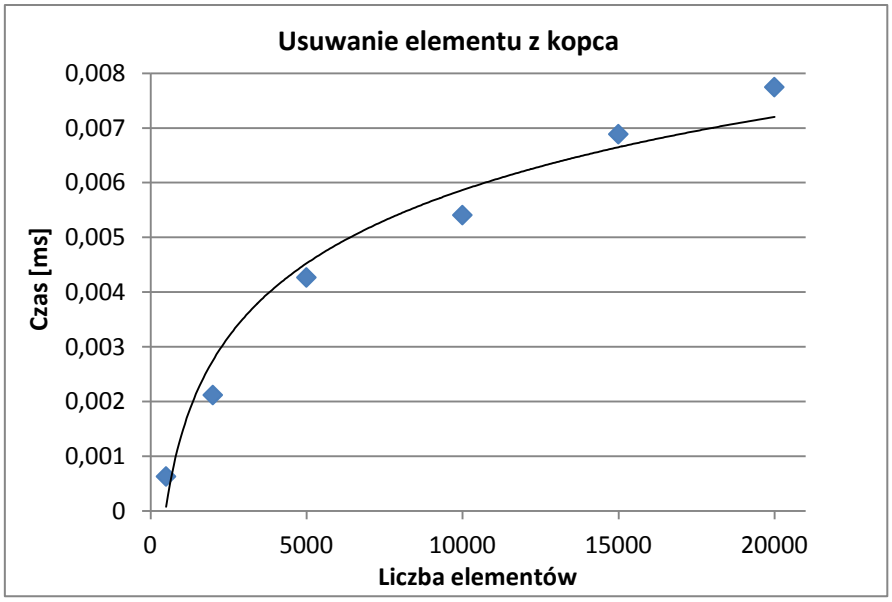
Wykres 16 – czas wyszukiwania elementu w liście w zależności od liczby elementów w niej przechowywanych

3.3 Kopiec

a) usuwanie elementu o podanym kluczu

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,000631
2.	2000	0,00212
3.	5000	0,00427
4.	10000	0,00541
5.	15000	0,00689
6.	20000	0,00775

Tabela 17 – usuwanie z kopca elementu o zadanym kluczu

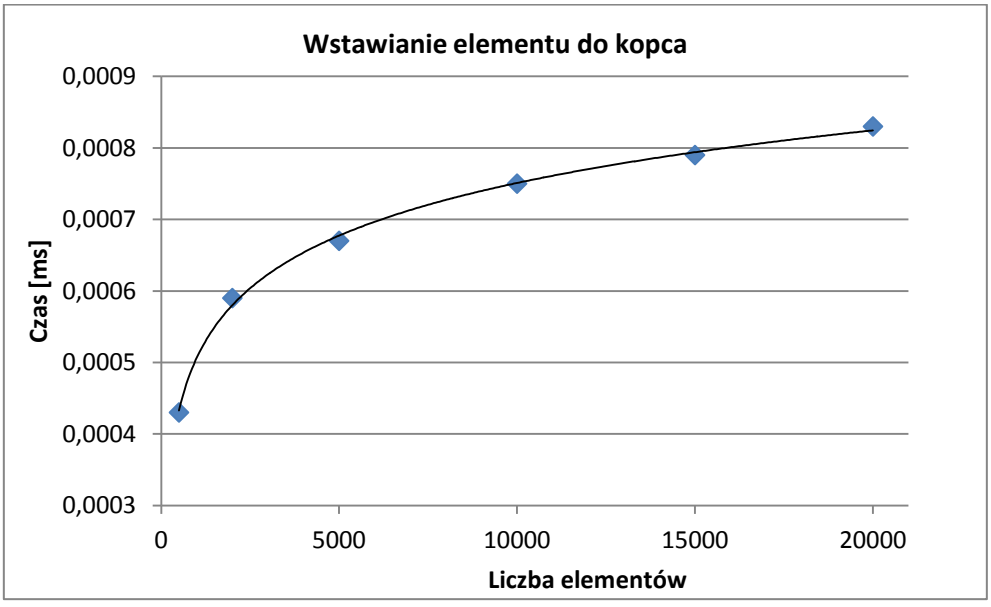


Wykres 17 – czas usuwania elementu z kopca w zależności od liczby elementów w kopcu

b) dodawanie elementu

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,000434
2.	2000	0,000591
3.	5000	0,00671
4.	10000	0,000749
5.	15000	0,000789
6.	20000	0,000829

Tabela 18 – dodawanie elementu do kopca

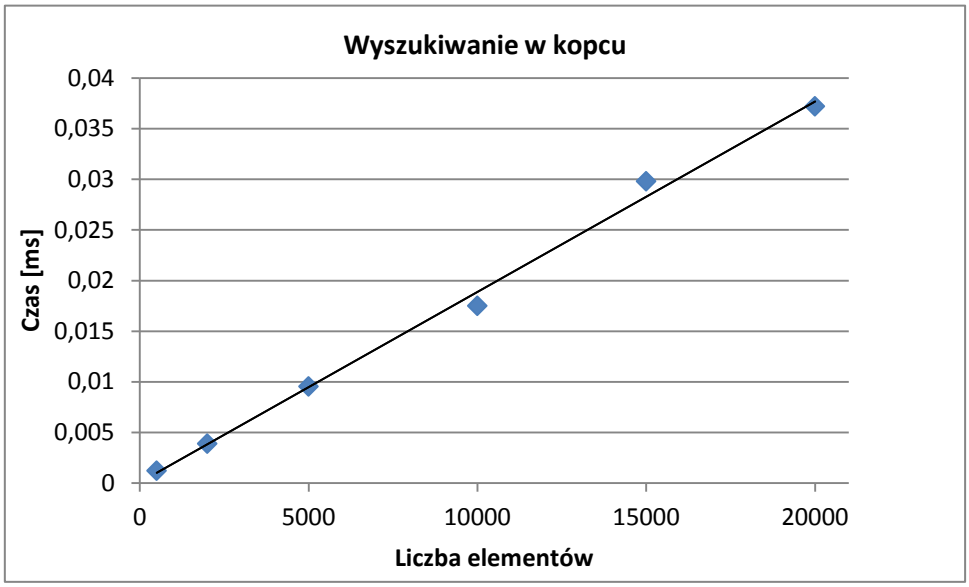


Wykres 18 – zależność czasu wstawienia elementu od liczby elementów w kopcu

c) wyszukiwanie

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,00122
2.	2000	0,00389
3.	5000	0,00954
4.	10000	0,0175
5.	15000	0,0298
6.	20000	0,0372

Tabela 19 – wyszukiwanie elementu w kopcu



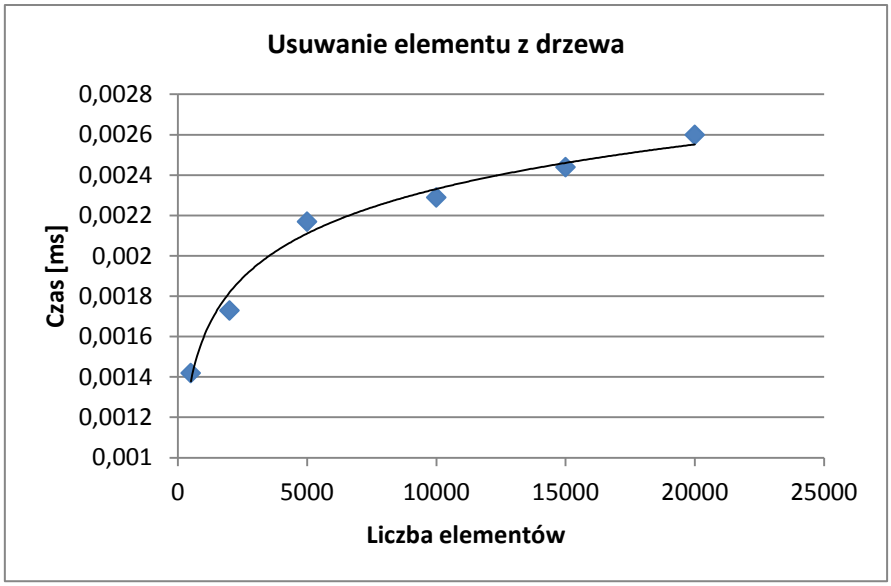
Wykres 19 – czas wyszukiwania elementu w kopcu w zależności od liczby elementów w strukturze

3.4 Drzewo czerwono – czarne

a) usuwanie elementu o podanym kluczu

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,00142
2.	2000	0,00173
3.	5000	0,00217
4.	10000	0,00229
5.	15000	0,00244
6.	20000	0,00260

Tabela 20 – usuwanie elementu o podanym kluczu z drzewa

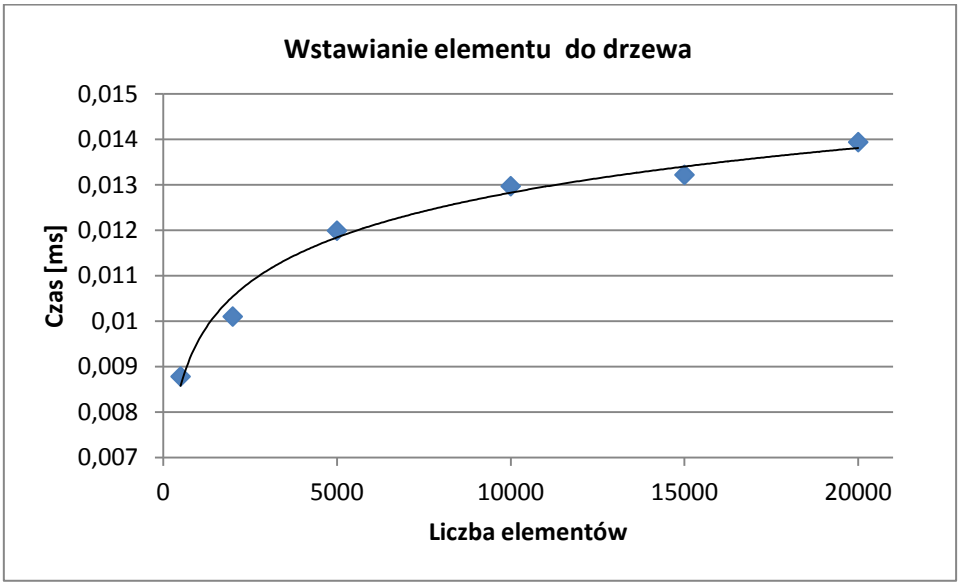


Wykres 20 – czas usuwania elementu z drzewa w zależności od liczby elementów w drzewie

b) dodawanie elementu

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,00878
2.	2000	0,0101
3.	5000	0,0119
4.	10000	0,0129
5.	15000	0,0132
6.	20000	0,0139

Tabela 21 – wstawianie elementu do drzewa

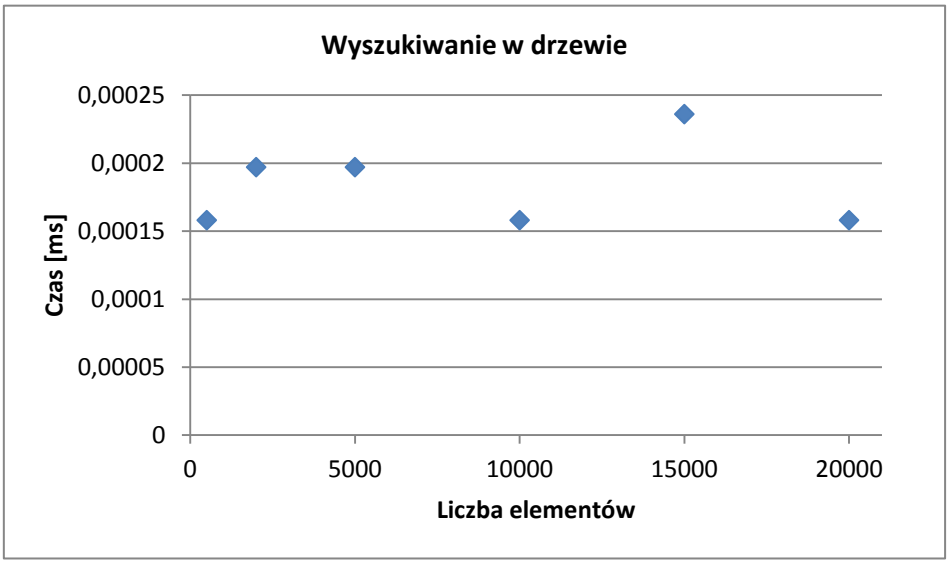


Wykres 21 – zależność czasu dodawania elementu do drzewa od liczby elementów w drzewie

c) wyszukiwanie

L.p.	Liczba elementów	Średni czas z 20 pomiarów [ms]
1.	500	0,000158
2.	2000	0,000197
3.	5000	0,000197
4.	10000	0,000158
5.	15000	0,000236
6.	20000	0,000158

Tabela 22 - wyszukiwanie elementu w drzewie



4. Wnioski

Rzędy złożoności obliczeniowych testowanych struktur zasadniczo zgadzają się z danymi w literaturze pod względem zależności czasu wykonywania operacji od liczby elementów przechowywanych w strukturze.

- Czasy usunięcia z początku, dodania na początek i wyszukiwania elementu w tablicy rosną wprost proporcjonalnie do liczby elementów, co ilustrują wykresy 2, 6 i 9. Operacje usuwania i dodawania na koniec tablicy przeprowadzane są w czasie stałym (wykres 4 i 7). Natomiast czas usunięcia lub wstawienia elementu do wnętrza tablicy jest wprost proporcjonalny do liczby elementów, które leżą na dalszych pozycjach od usuwanego elementu, bądź od miejsca, na które chcemy dodać element (wykresy 5 i 8).
- Operacje na elementach krańcowych listy (dodawanie i usuwanie z końca lub początku) mają stałe czasy, co można zaobserwować na wykresach 10,11,13 i 14. Czas usunięcia lub wstawienia elementu do wnętrza listy uzależniony jest od położenia modyfikowanego elementu na liście. Generalnie średnie czasy operacji (usuwania, dodawania, wyszukiwania) na elementach wewnątrz listy rosną wraz z liczbą elementów przechowywanych w liście (wykresy 12, 15,16).

- Logarytmiczną złożoność obliczeniową mają pewne algorytmy używane w kopcu (usuwanie, wstawianie – wykresy 17, 18) oraz wszystkie operacje na drzewach czerwono – czarnych. Badanie operacji wstawiania i usuwania z drzewa potwierdziło tą wiedzę (wykresy 20, 21), jednak wykonując operację wyszukania elementu w drzewie, nie zaobserwowano różnic podczas testów struktur o różnej liczbie elementów. Otrzymywano tylko i wyłącznie wyniki pomiarów 0 milisekund oraz 0,000395 milisekundy, które uśredniono (tabela 22, wykres 22). Wynika to zapewne z bardzo szybkiego algorytmu, dzięki któremu różnice w czasie nie są widoczne przy takich ilościach elementów (500-20 000).

Wartość elementu sama w sobie ma znikomy lub zerowy wpływ na czas jego modyfikacji w strukturze. Natomiast w przypadku struktur takich, jak kopiec i drzewo czerwono – czarne wartości przechowywanych oraz dodawanych elementów mają znaczenie. W zależności od wartości dodanego elementu, może lub nie musi być konieczne przywrócenie porządku struktury. Podczas testów starano się jednak usuwać i dodawać elementy o bardzo różnych wartościach, aby uśrednione wyniki pokazały faktyczną zależność czasu od liczby elementów.

Dynamiczna alokacja pamięci w przypadku struktur tablicowych jest istotna w prawidłowym działaniu programu. Nie korzystanie z niej skutkuje niestabilnością programu, który po wykonaniu wielu operacji może zakończyć swoje działanie w wyniku błędu pamięci. Jednak korzystanie z relokacji po każdym usunięciu/dodaniu elementu wydłuża czas operacji, a dodatkowo uniemożliwia zaobserwowanie rzeczywistej efektywności danego algorytmu. Można to zauważyć, porównując ze sobą wykresy 3 i 4 – na wykresie 3 czas usunięcia elementu z końca tablicy rośnie wraz z liczbą elementów, natomiast wykres 4 obrazuje eksperyment, w którym pominięto czas relokacji pamięci. Z porównania wynika, że relokacja pamięci wpłynęła na wynik testów, ukazując fałszywą zależność między czasem operacji a liczbą elementów w strukturze. W przypadku operacji usuwania elementu z początku tablicy również dokonano pomiarów dla algorytmu wykorzystującego relokację pamięci i nie korzystającego z niej. Zależność czasu od liczby elementów w obu sytuacjach była taka sama (wykresy 1 i 2), ale poszczególne czasy były około 10 razy dłuższe w przypadku algorytmu z relokacją pamięci (tabele 1 i 2).

Badane w projekcie struktury mają różne zastosowania w zależności od ich specyficznych cech. Tablice dają się łatwo przetwarzać dzięki bezpośredniemu dostępowi do konkretnego elementu poprzez indeks. Listy służą do reprezentacji zbiorów dynamicznych, takich jak stosy i kolejki, ponieważ nie wymagają ręcznej alokacji pamięci. Kopca używa się przede wszystkim w algorytmie sortowania przez kopcowanie. Największą zaletą drzew czerwono - czarnych jest zrównoważona oraz uporządkowana struktura, pozwalająca na szybką modyfikację elementów i efektywną implementację algorytmu wyszukiwania.

5. Bibliografia

1. Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., *Wprowadzenie do Algorytmów*, wyd. 4, Warszawa: Wydawnictwa Naukowo-Techniczne