Marcin Całkowski
nr 218191
Śr. 13.15

Struktury danych i złożoność obliczeniowa

Zadanie projektowe nr 1

Badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w różnych strukturach danych

Prowadzący: Zbigniew Buchalski

Politechnika Wrocławska

1. Wstęp

Tematem projektu jest przedstawienie własnoręcznie napisanych struktur danych, oraz przedstawienie wyników czasów działania podczas operacji takich jak dodawanie, usuwanie oraz wyszukiwanie. Struktury w projekcie poddane testom to kolejno tablica, lista oraz kopiec binarny. Wyniki mają przedstawić zalety oraz wady zarówno samych struktur, jak i różnic w technice implementacji wspomnianych algorytmów.

2. Podstawowe zagadnienia teoretyczne

Struktura danych jest miejscem, w którym przechowywane są nasze dane. Można go nazwać pojemnikiem, w którym umieszczamy ważne dla nas informacje, w celu usprawnienia ich porządkowania a także odnajdywania tych, które są nam potrzebne. Struktur danych jest wiele, i w zależności od naszych potrzeb powinniśmy wybrać strukturę najbardziej optymalną do naszego zadania. Każda struktura jest zdefiniowana w odpowiedni sposób, który zwykle charakteryzuje się pewnymi wadami i zaletami w procesie ich dodawania, usuwania czy też porządkowania.

Tablicą nazywamy strukturę, w której nasze zmienne przechowywane są kolejno w pamięci w miejscach („szufladkach") wielkości jednego elementu naszej struktury. Rozróżniamy dwa typy tablic: Tablice statyczne oraz dynamiczne. Różnią się one właściwie tylko miejscem, w którym jest tablica alokowana (w przypadku tablicy statycznej jest ona automatycznie przypisana na początku programu, w przypadku dynamicznej w dowolnym miejscu w programie). Tablica posiada zwykle rząd złożoności obliczeniowej około O(n) ze względu na konieczność przekopiowania całej zawartości do nowej, większej lub mniejszej w zależności czy dodawaliśmy czy usuwaliśmy element. Odczytanie wartości z indeksu jako zwykłe odczytanie wartości miejsca tablicy będzie miało złożoność O(1), gdyż odczytany jest tylko wskazany przez nas element.

Lista jest strukturą opartą na wskaźnikach. Jej podstawową własnością jest brak bezpośredniego indeksowania kolejnych komórek, a do kolejnych elementów dostajemy się poprzez wskaźniki innych elementów. Istnieją dwa rodzaje list: jedno oraz dwukierunkowe, które różnią się ilością wskaźników. Lista jednokierunkowa posiada wskaźnik tylko na następny element, a dwukierunkowa oprócz tego posiada również wskaźnik na poprzedni. Oprócz tego listy można podzielić również na cykliczne i niecykliczne. Różnią się one tym, że wskaźnik ostatniego elementu w listach cyklicznych wskazuje na pierwszy element listy, oraz (w przypadku listy dwukierunkowej) wskaźnik wstecz pierwszego elementu wskazuje na ostatni w liście. Przybliżone złożoności listy to O(1) dla dodawania, usuwania, czy odczytywania wartości na krańcach listy (początek i koniec). Przeszukiwanie reszty tablicy, jak i dodawanie elementów w jej wnętrzu ma złożoność O(n) gdyż trzeba przechodzić po kolejnych wskaźnikach elementów szukając odpowiedniej wartości.

Kopcem binarnym nazwiemy strukturę hierarchiczną zbudowaną na wzór drzewa posiadającego taką własność, że jego największy lub najmniejszy element (w zależności od typu kopca) znajduje się na samej górze. Hierarchia kopca opiera się na nadrzędności innych elementów nad innymi. Elementy nadrzędne nazywa się rodzicami, a podrzędne potomkami. W kopcu binarnym każdy rodzic posiada dokładnie dwóch potomków z których każdy jest mniejszy (kopiec typu MAX) od swojego rodzica. Utrzymanie hierarchii pozwala na łatwy dostęp do elementu największego (z rzędem złożoności obliczeniowej O(1)). Operacje dodawania i usuwania mogą mieć złożoności rzędów O(log2(n)) albo O(log(n)) w zależności od potrzeby reorganizacji struktury po każdym takim zabiegu. Jest to konieczne, ponieważ aby hierarchia w kopcu została zachowana potrzebne jest odpowiednie posortowanie danych, które może okazać się czasochłonne szczególnie wykonywane rekurencyjnie dla dużej liczby danych.

3. Plan eksperymentu

Eksperyment został wykonany w moim własnym programie którego kod został oparty na algorytmach znalezionych w materiałach z zajęć lub w sieci. Podstawą jego wykonania jest zbadanie czasu przeprowadzenia odpowiedniej operacji dla wszystkich struktur oraz porównanie ich wyników, podczas przeprowadzania operacji dodawania, usuwania oraz wyszukiwania danych. Struktury zgodnie z poleceniem przechowują liczby całkowite typu **int**. Dane do testów czasowych zostały każdorazowo wylosowane aby dać pewność średnich wyników. Dla każdej struktury test przeprowadzany jest **100 razy**, uzyskany wynik czasowy jest średnią z wszystkich uzyskanych wyników.

Czas został zmierzony przy pomocy biblioteki **chrono** (dostępnej od standardu c++11). Pozwala ona na uzyskanie dobrych wyników czasu bez dużych problemów, co znacząco ułatwia testowanie wszystkich struktur. Wszystkie wyniki czasowe przedstawione są w **milisekundach**. Maszyna testowa wyposażona jest w procesor **Intel i7** z taktowaniem **2.4 GHz** oraz **8GB** pamięci RAM, a program uruchamiany był w wersji 64 bitowej.

Testy wykonane zostały dla następujących wielkości: Dla tablicy i listy kolejno dla **100 000, 200 000, 500 000, 1 000 000, 2 000 000**. Dla kopca binarnego ze względu na sortowanie rekurencyjne użyłem wartości **5 000, 10 000, 15 000, 20 000, 25 000**. Ze względu na zastosowane przeze mnie sortowanie rekurencyjne struktura ta ma problemy z operacjami na większej ilości danych. Dla tych wszystkich wielkości przedstawię poniżej wyniki czasowe operacji takich jak wczytanie z pliku, dodawanie, usuwanie czy wyświetlanie wartości.

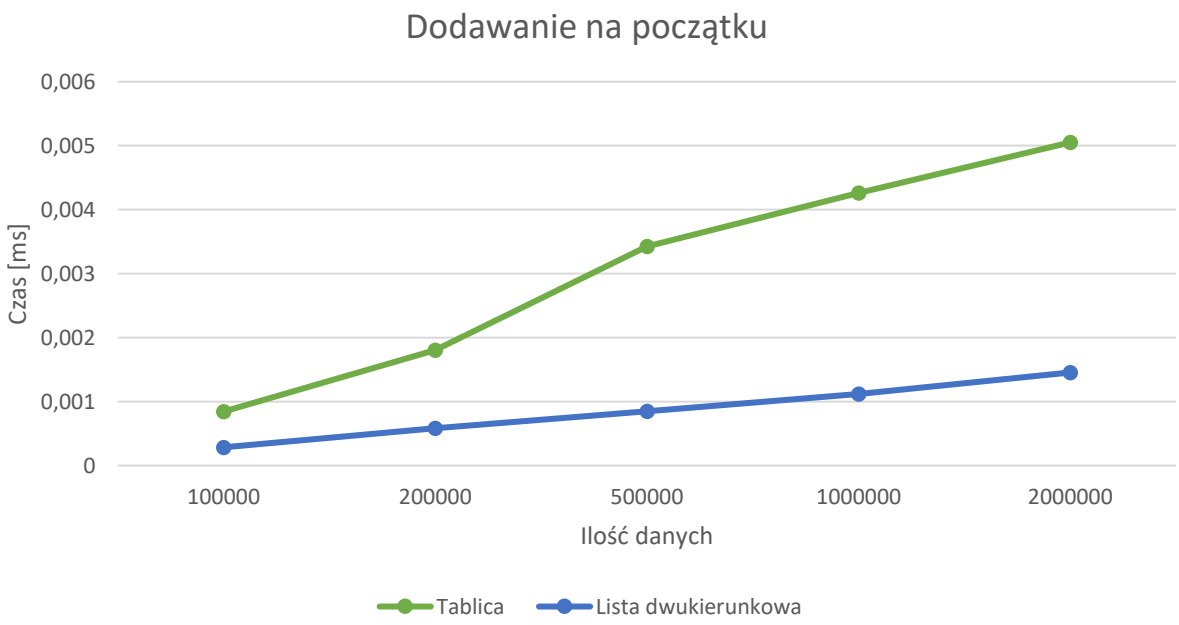
4. Przedstawienie wyników pomiarów

Poniżej przedstawiono w tabelach wyniki uzyskane podczas badań, kolejno dla dodawania, usuwania oraz wyszukiwania elementów w strukturach. Zaznaczyć trzeba, że ze względu na rozbieżność testowanych grup pomiędzy kopcem, a listą i tablicą wyniki te nie będą bezpośrednio porównywane. Wyniki konkretnego typu zostały porównane ze sobą na wykresach.

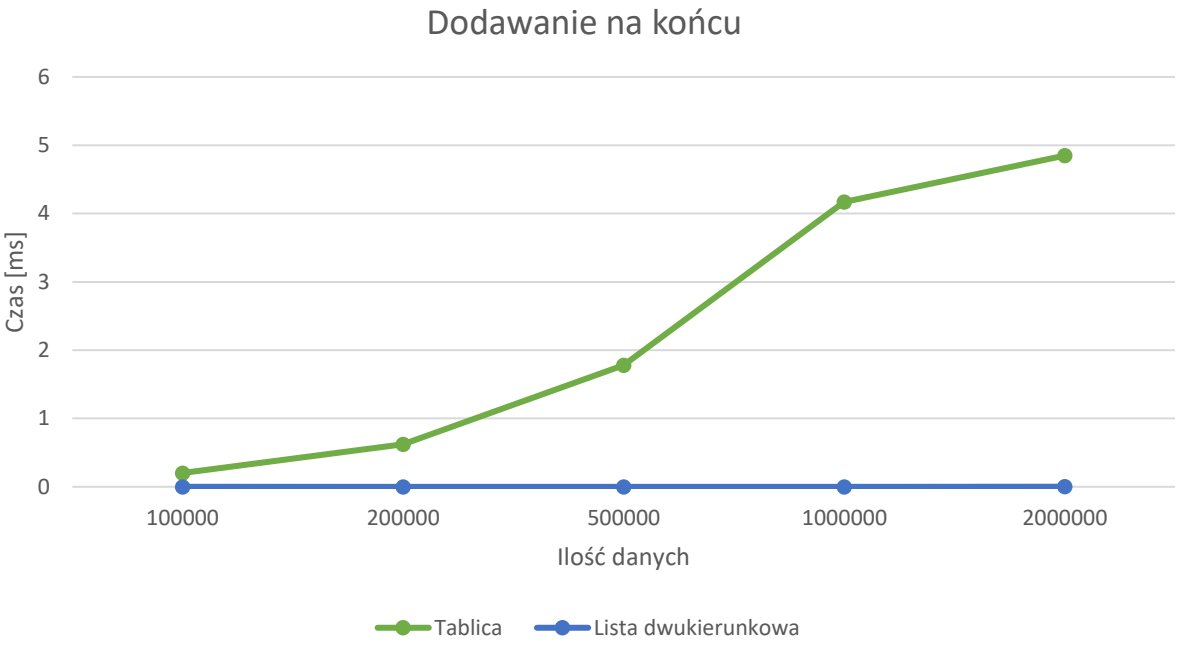
1. Dodawanie elementów do struktury

	Ilość elementów:		100 000	200 000	500 000	1 000 000	2 000 000
Czas [ms]	Na początku	Tablica	0,000844	0,001802	0,003425	0,004263	0,005051
		Lista dwukierunkowa	0,000285	0,000585	0,000847	0,001121	0,001455
	Na końcu	Tablica	0,202346	0,62162	1,7797	4,16821	4,84882
		Lista dwukierunkowa	0,00024	0,000523	0,000815	0,001077	0,001363
	Losowe miejsce	Tablica	0,212656	0,646115	1,86944	4,21243	4,97136
		Lista dwukierunkowa	0,279837	0,513369	0,745663	0,948431	1,16275

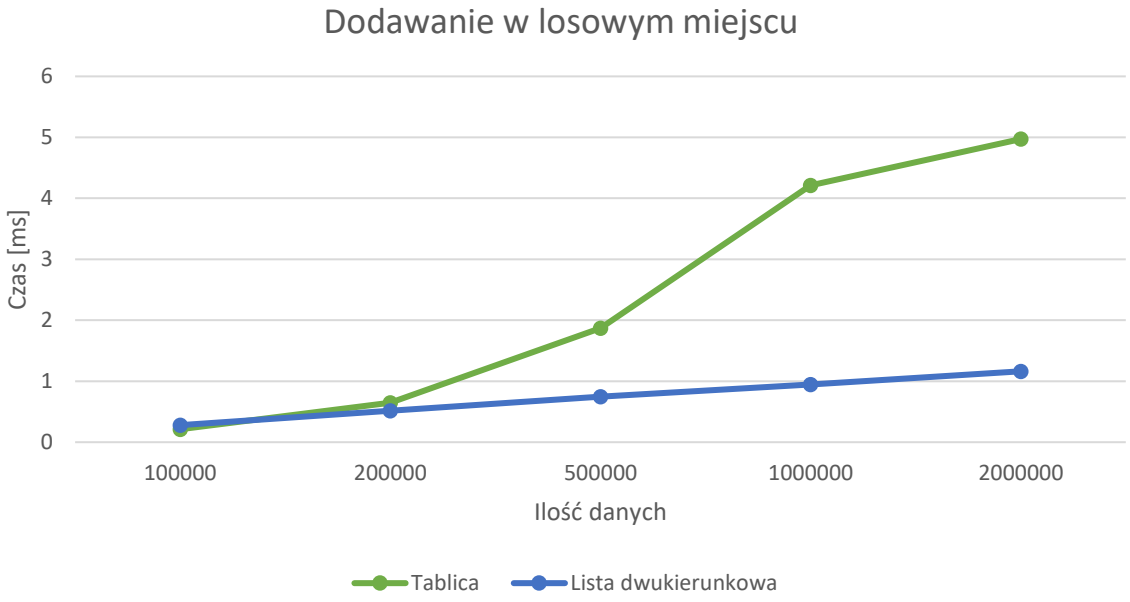
Tabela 1. Wyniki czasowe dodawania na początku, końcu oraz w losowym miejscu dla tablicy oraz listy.



Wykres 1. Porównanie dodawania na początku tablicy oraz listy dwukierunkowej.



Wykres 2. Porównanie dodawania na końcu tablicy oraz listy dwukierunkowej.



Wykres 3. Porównanie dodawania w losowym miejscu tablicy oraz listy dwukierunkowej.

Dodawanie elementu dla kopca binarnego:

Ilość elementów	**5 000**	**10 000**	**15 000**	**20 000**	**25 000**
Czas [ms]	0,017274	0,077323	0,143805	0,19946	1,0824

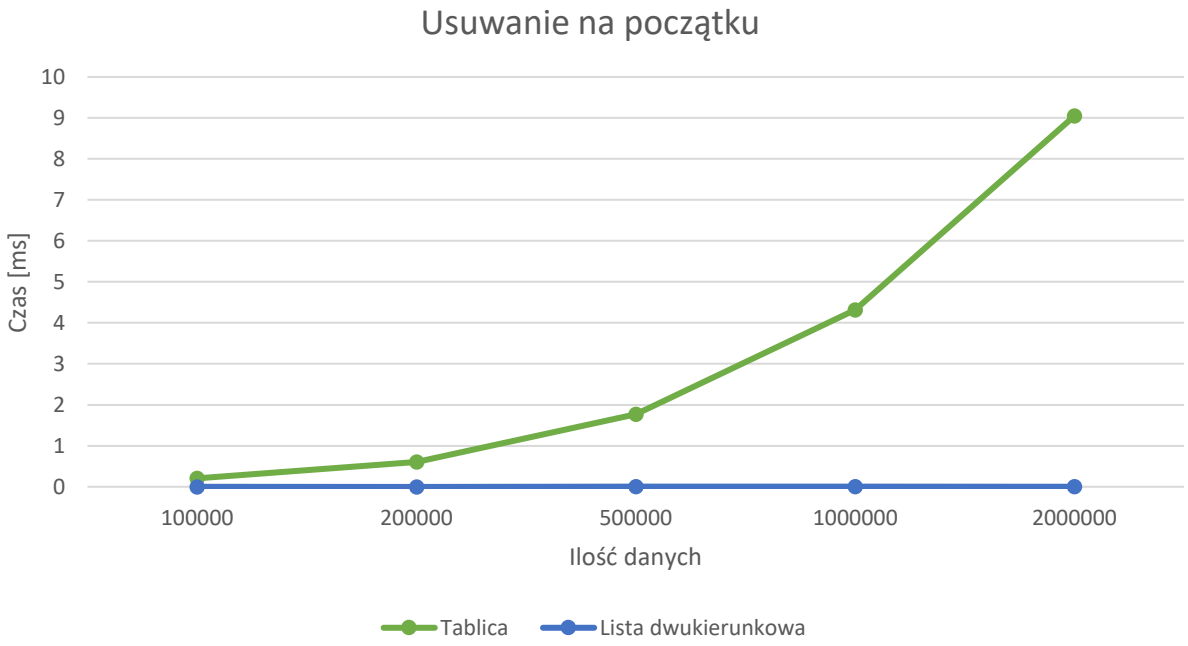
Tabela 2. Czas dodawania elementów do kopca binarnego

Niestety rozbieżność w ilości danych spowodowana rekurencyjnością dodawania w kopcu nie pozwala bezpośrednio porównać go z tablicą oraz listą.

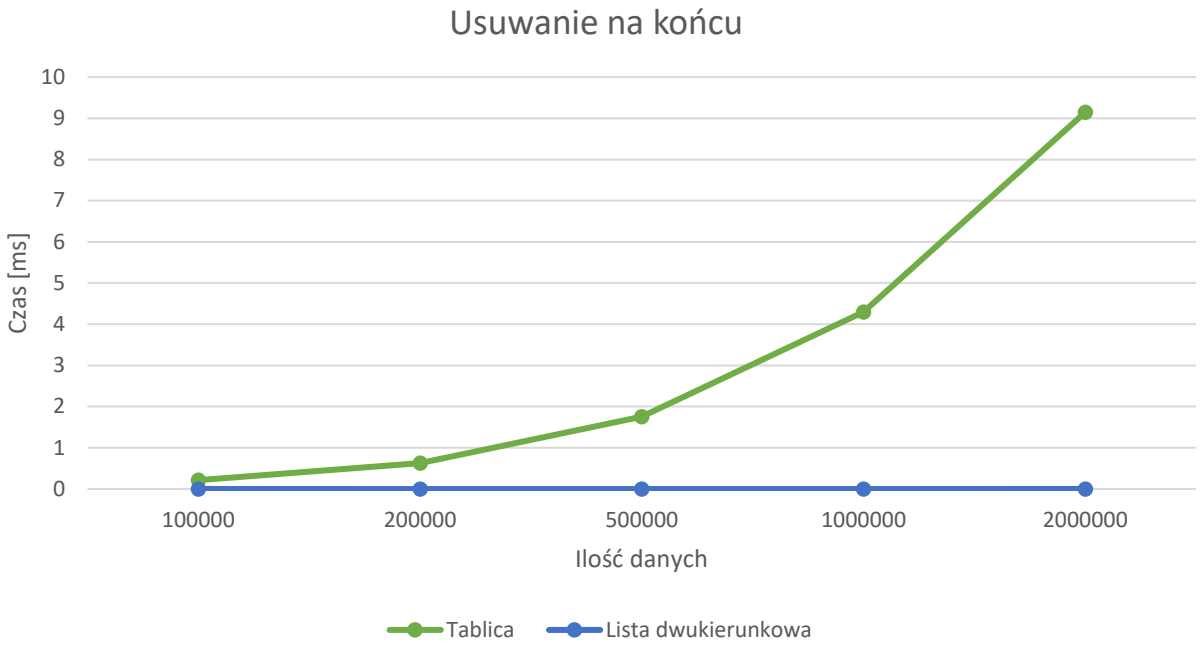
2. Usuwanie elementów ze struktury

	Ilość elementów:		100 000	200 000	500 000	1 000 000	2 000 000
Czas [ms]	Na początku	Tablica	0,206575	0,604875	1,77247	4,31889	9,04601
		Lista dwukierunkowa	0,000213	0,000435	0,000643	0,000873	0,001085
	Na końcu	Tablica	0,213534	0,628077	1,75726	4,29468	9,14463
		Lista dwukierunkowa	0,000078	0,00016	0,000247	0,000313	0,000355
	Losowe miejsce	Tablica	0,222984	0,61196	1,77795	4,27494	9,09689
		Lista dwukierunkowa	0,223276	0,477374	0,684752	0,891692	1,10827

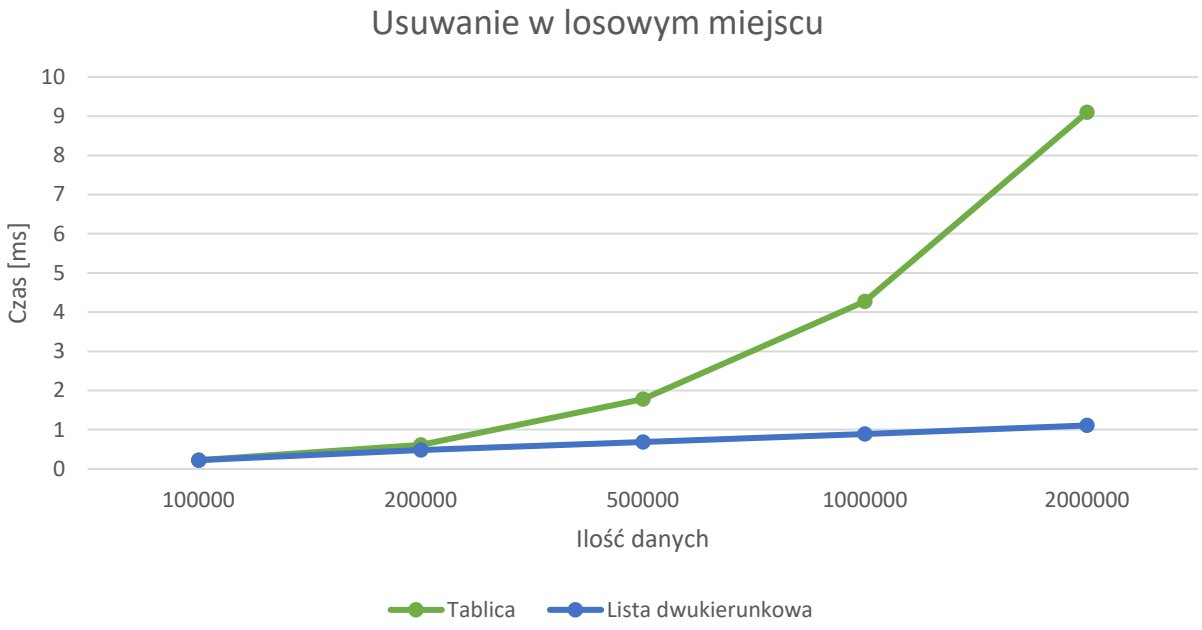
Tabela 3. Wyniki czasowe usuwania na początku, końcu oraz w losowym miejscu dla tablicy oraz listy.



Wykres 4. Porównanie usuwania na początku tablicy oraz listy dwukierunkowej.



Wykres 5. Porównanie usuwania na końcu tablicy oraz listy dwukierunkowej.



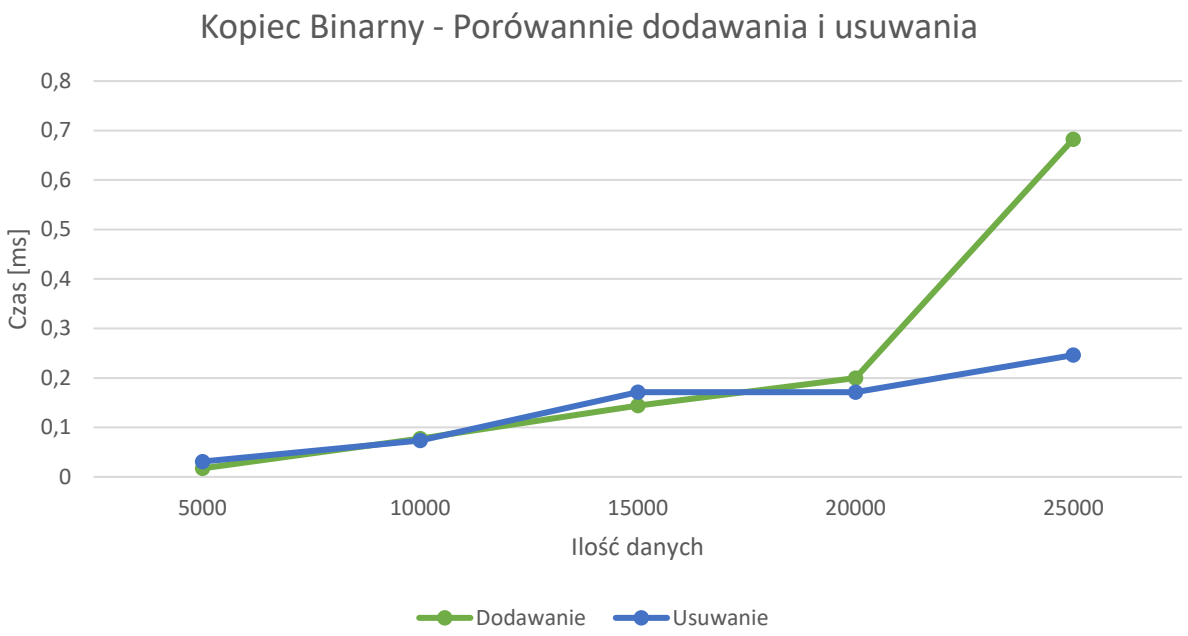
Wykres 6. Porównanie usuwania losowym miejscu tablicy oraz listy dwukierunkowej.

Usuwanie elementu kopca binarnego:

Ilość elementów	5 000	10 000	15 000	20 000	25 000
Czas [ms]	0,031026	0,073595	0,17095	0,17107	0,24608

Tabela 4. Czas usuwania elementów do kopca binarnego.

Podobnie jak poprzednio rozbieżność w ilości nie pozwala bezpośrednio porównać wyników pomiarów. Poniżej wykres porównujący operacje dodawania i usuwania elementów w kopcu binarnym:



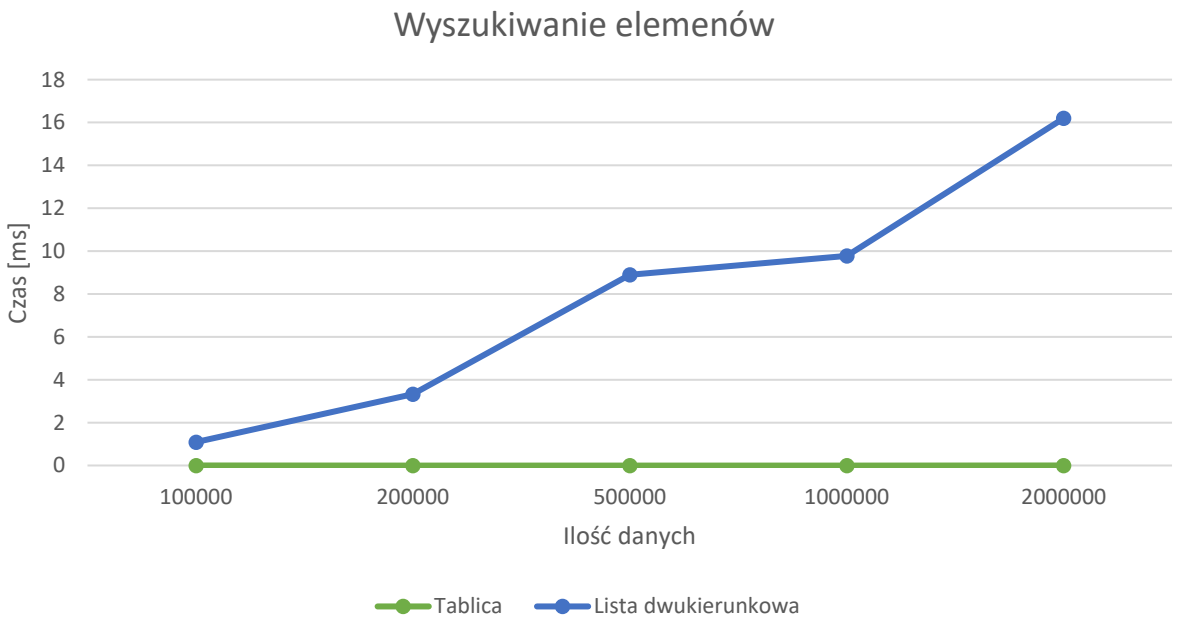
Wykres 7. Porównanie operacji dodawania oraz usuwania dla kopca binarnego

3. Wyszukiwanie elementów

Średnie wyniki czasu wyszukania elementów o podanej wartości w tablicy i liście dwukierunkowej:

Wartość	100 000	200 000	500 000	1 000 000	1 500 000
Tablica	0,000089	0,000197	0,000293	0,000393	0,000469
Lista Dwukierunkowa	1,09099	3,32865	8,90302	9,7748	16,19301

Tabela 5. Czas wyszukiwania elementów w tablicy oraz w liście dwukierunkowej.



Wykres 8. Porównanie wyszukiwania tablicy oraz listy dwukierunkowej.

Wyniki wyszukiwania elementów dla kopca binarnego:

Ilość elementów	5 000	10 000	15 000	20 000	25 000
Czas [ms]	0,00014	0,00009	0,0007	0,00059	0,00051

Tabela 6. Czas wyszukiwania elementów w kopcu binarnym.

5. Wnioski z eksperymentu

Z przeprowadzonego eksperymentu pierwszym wyciągniętym wnioskiem jest wyraźna różnica pomiędzy listą a tablicą. W tablicy mamy bardzo szybki dostęp do danych, jednak poprzez konieczność przepisania całej tablicy w przypadku wykonywania operacji dodawania lub usuwania powoduje zwiększenie czasu wykonywania operacji wraz z wzrostem rozmiaru tablicy zgodnie z teorią (wyniki bliskie złożoności O(n), najlepiej zobrazowane na wykresie 1). Lista z kolei mimo bardzo szybkiego dodawania i usuwania elementów, szczególnie na krańcach listy gdzie proces ten wykonywany błyskawicznie, miał o wiele słabsze wyniki podczas szukania elementu (wykres 8), spowodowanego koniecznością sprawdzania element po elemencie korzystając ze wskaźników. Widać więc bardzo dobrze zależność: tablica jest użyteczna, gdy często czytamy zawartość elementów, natomiast lista w przypadku częstego ich dodawania.

Ciekawym przypadkiem podczas testowania tablicy jest fakt, iż podczas dodawania elementów do niej na początku algorytm działał błyskawicznie, jednak w przypadku dodawania na koniec bądź w losowym miejscu algorytm potrzebował już do kilku milisekund na wykonanie (w zależności od wielkości struktury, wykresy 1-3). Podejrzewam iż jest to wina optymalizowania kodu przez kompilator i w tym pierwszym przypadku wstawianiu zmiennej bezpośrednio przed tablicę, jednak nie mniej jest to ciekawe zjawisko.

Niestety problematyczna okazała się struktura kopca, która ze względu na konieczność kopcowania elementów w celu zachowania odpowiedniej kolejności liczb nie była w stanie pomieścić tylu danych co inne struktury. Algorytm wykonałem zgodnie z teorią, jednak użycie jej w sposób rekurencyjny nie pozwala na kopcowanie dużej liczby elementów. Problemy pojawiały się już dla instancji większych niż 25 000, czyli dużo mniej niż w przypadku pozostałych dwóch struktur.

Struktura kopca ma jednak zaletę którą jest szybkie wyszukiwanie liczb, szczególnie dużych (w przypadku zastosowanego przeze mnie kopca MAX). Dzięki kopcowaniu szybko możemy się dowiedzieć, czy szukana przez nas liczba znajduje się w danej gałęzi drzewa kopca. Dzięki temu możemy zmniejszać czas wyszukiwania, wykluczając ścieżki, w których dany rodzić ma mniejszą wartość od liczby przez nas szukanej. Wykorzystanie tej zalety w odpowiednich okolicznościach może nam znacząco przyśpieszyć poszukiwanie danych w strukturze.