Politechnika Wrocławska

Projekt 3 – Struktury Danych
„Tablice mieszające”

Wydział Informatyki i Telekomunikacji

Informatyczne Systemy Automatyki

Aleksander Żołnowski, 272536

Franciszek Jeziorowski, 272526

Spis treści

1 Wstęp

1.1 Tablice mieszające

Tablica mieszająca, znana również jako tablica z haszowaniem (ang. hash table), to rodzaj struktury danych używanej do implementacji tablic asocjacyjnych. Tablica asocjacyjna to abstrakcyjny typ danych, który pozwala na przechowywanie par klucz-wartość w taki sposób, aby dostęp do tych danych był bardzo szybki. Dzięki tablicy mieszającej można także sprawnie porównywać dane, takie jak fragmenty tekstów czy plików. Innymi słowy, tablica mieszająca pozwala na efektywne przechowywanie i szybkie wyszukiwanie informacji.

1.2 Separate Chaining

Separate chaining to metoda rozwiązywania kolizji w tablicach mieszających, polegająca na tworzeniu listy powiązanej dla każdego indeksu tablicy. Elementy o kolizyjnych kluczach są łączone w jedną listę, co umożliwia ich szybkie odnajdywanie poprzez przeglądanie listy powiązanej pod odpowiednim indeksem. Jest to powszechne rozwiązanie stosowane w implementacji tablic mieszających.

1.3 Open addressing

Open addressing to kolejna metoda rozwiązywania kolizji, w której każdy rekord jest przechowywany bezpośrednio w tablicy wiaderek (ang. bucket array), a rozwiązanie kolizji odbywa się poprzez sondowanie (ang. probing). Gdy nowy rekord ma zostać wstawiony, wiadra są badane, zaczynając od wyznaczonego przez funkcję haszującą indeksu i kontynuując w określonej sekwencji sondowania, aż do znalezienia niezajętego slotu. Podczas wyszukiwania rekordu, wiadra są przeszukiwane w tej samej sekwencji sondowania, aż do znalezienia albo rekordu docelowego, albo wolnego slotu w tablicy, co oznacza niepowodzenie w znalezieniu szukanego rekordu.

1.4 Cuckoo hashing

Cuckoo hashing to metoda rozwiązywania kolizji w otwartej adresacji, która gwarantuje złożoność O(1) w najgorszym przypadku podczas wyszukiwania oraz stały czas amortyzowany dla operacji wstawiania. Kolizje są rozwiązywane poprzez utrzymanie dwóch tablic mieszających, z których każda ma swoją własną funkcję haszującą. W przypadku kolizji, zajęty slot zostaje zastąpiony nowym elementem, a poprzedni element slotu jest przenoszony do drugiej tablicy mieszającej. Proces ten trwa do momentu, aż każdy klucz znajdzie swoje miejsce. Jeśli procedura wpadnie w nieskończoną pętlę, co jest identyfikowane poprzez utrzymanie licznika pętli, obie tablice zostają przehaszowane.

2 Implementacja

Istnieje wiele sposobów rozwiązania problemu występowania kolizji w tablicach mieszających. W tym projekcie porównano ze sobą 3 różne implementacje: Separate Chaining, Open addressing oraz Cuckoo hashing.

Wszystkie z zaimplementowanych rozwiązań zostały napisane obiektowo i dziedziczą po abstrakcyjnej klasie HashTable, która jest interfejsem. Podczas implementowania klas przedstawiających sposoby rozwiązywania kolizji zaimplementowano dla każdej z nich następujące podstawowe metody, zgodnie z interfejsem:

- Insert – metoda odpowiadająca za dodanie elementu (klucz i wartość)
- Find – metoda odpowiadająca za odnalezienie elementu w tablicy
- GetSize – metoda zwracająca zmienną, która reprezentuje ilość elementów w tablicy.

Dla każdej z klas zaimplementowane zostały również metody pomocnicze takie jak:

- Rehashing – niezbędna metoda odpowiadająca za zwiększenie rozmiaru tablicy i ponowne przehashowanie elementów.
- Funkcje mieszające – niezbędna metoda odpowiadające za wyliczenie indeksu w tablicy dla klucza elementu
- Display – metoda pomocnicza wykorzystywana w celu wyświetlenia elementów znajdujących się w tablicy.

Metody pomocnicze różniły się ilością i zastosowaniem, ponieważ był dotowane specjalnie pod implementowany sposób rozwiązywania kolizji, dlatego nie zostały zawarte w klasie reprezentującej interfejs. Przykładowo Cuckoo Hashing posiadał, aż 3 funkcje mieszające, które zmieniały się między sobą podczas wywołania rehashingu. Zabieg ten pomagał uniknąć wystąpienie cyklu kolizji.

```cpp
template<typename K, typename V>
int Cuckoo<K,V>::current_hash1(K key)
{
    switch(cycle % 3)
    {
        case 0: return hash1(key);
        case 1: return hash2(key);
        case 2: return hash3(key);
        default: return hash1(key);
    }
}

template<typename K, typename V>
int Cuckoo<K,V>::current_hash2(K key)
{
    switch(cycle % 3)
    {
        case 0: return hash2(key);
        case 1: return hash3(key);
        case 2: return hash1(key);
        default: return hash2(key);
    }
}
```

Rysunek 1 Metody odpowiadające za wybór funkcji mieszających dla tablic w Cuckoo Hashing

W projekcie zaimplementowano, również klasę Timer, aby porównać wydajność operacji insert, find oraz getSize dla przedstawionych sposobów rozwiązywania kolizji. Klasa ta była pomocnicza, ponieważ usprawniła przebieg badań, wykorzystując wcześniej zaimplementowany interfejs.

Pełna implementacja znajduję się w repozytorium:
https://github.com/AbaturDev/Struktury-Danych/tree/main/Projekt3

3 Badania

W celu zbadania wydajności zaimplementowanych rozwiązań przeprowadzono dla każdego z nich serię testów operacji dodawania, przeszukania oraz zwrócenia rozmiaru w zależności od ilości elementów w tablicy mieszającej. Testy zostały przeprowadzone na różnych rozmiarach tablicy, które zwiększały się logarytmicznie o 10-krotność dla każdego kroku. Zakres wielkości tablicy wynosił od 10 do 1 000 000 elementów. Wydajność badanych operacji została zmierzona w nanosekundach(ns).

3.1 Separate Chaining

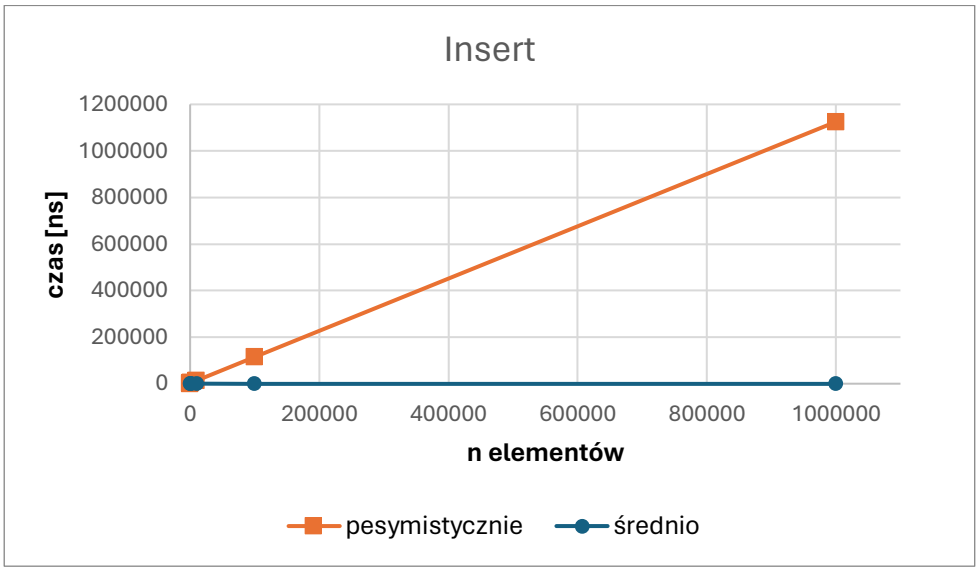
3.1.1 Insert

Badanie wydajności operacji dodawania elementu do tablicy mieszającej. Przebadano przypadek średni, czyli standardowe dodanie elementu do tablicy

mieszającej oraz pesymistyczny, gdy dodajemy wszystkie elementy na jeden klucz.

insert			
średnio		pesymistycznie	
n	czas [ns]	n	czas [ns]
10	250	10	250
100	291	100	1625
1000	292	1000	3459
10000	250	10000	12167
100000	263	100000	115291
1000000	275	1000000	1125083

Tabela 1 Wyniki operacji insert



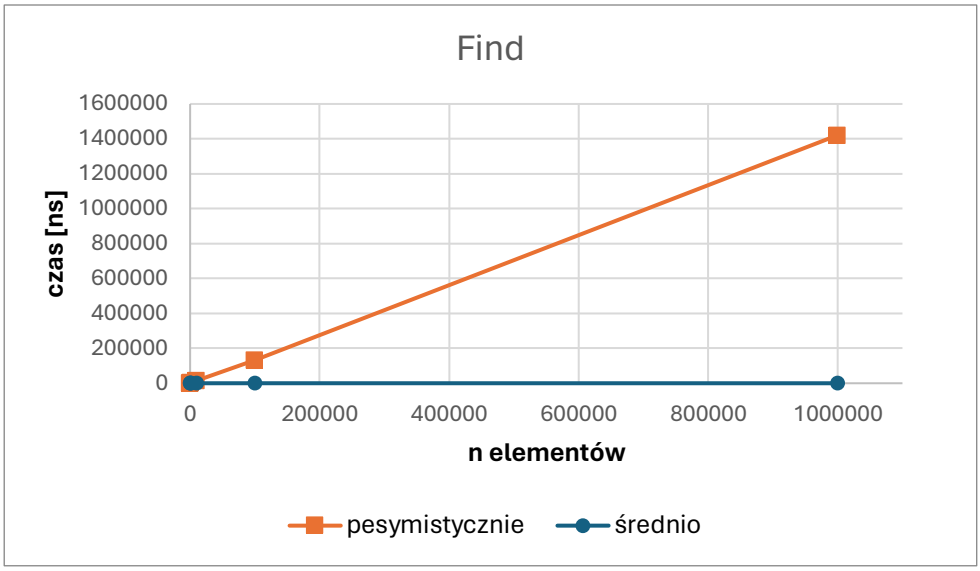
Rysunek 2 Charakterystyka złożoności czasowej dla operacji insert

3.1.2 Find

Badanie wydajności operacji znalezienia elementu w tablicy mieszającej. Przebadano przypadek pesymistyczny, wszystkie elementy na jednym kluczu oraz przypadek średni, czyli standardowe rozłożenie elementów w tablicy.

Find			
średnio		pesymistycznie	
n	czas [ns]	n	czas [ns]
10	208	10	208
100	250	100	791
1000	208	1000	4083
10000	221	10000	16541
100000	208	100000	132125
1000000	241	1000000	1421000

Tabela 2 Wyniki dla operacji find



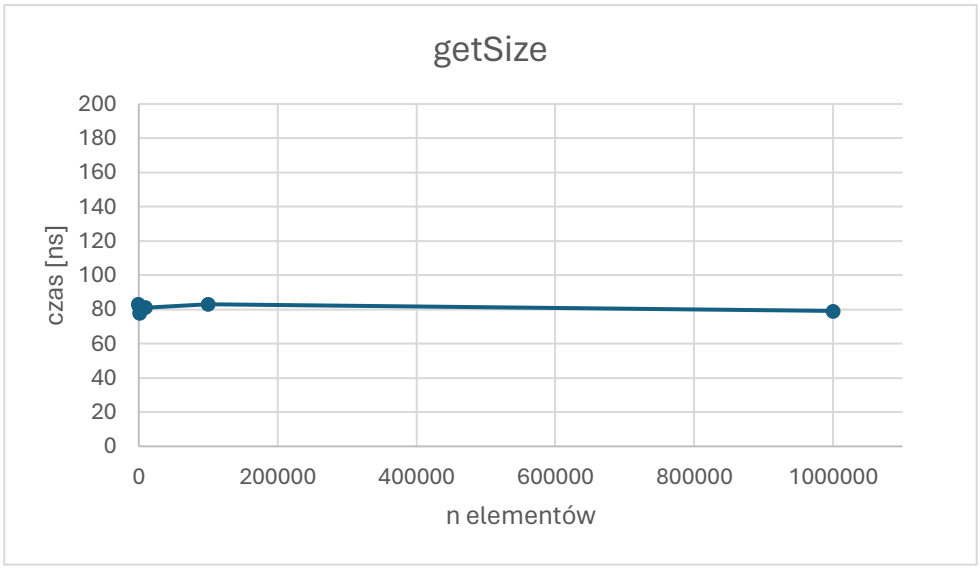
Rysunek 3 Charakterystyka złożoności czasowej dla operacji Find

3.1.3 getSize

Badanie wydajności operacji zwrócenia wartości, która wyraża rozmiar tablicy mieszającej – czyli ilość elementów, które się w niej znajdują.

getSize	
n	czas [ns]
10	83
100	82
1000	78
10000	81
100000	83
1000000	79

Tabela 3 Wyniki operacji getSize



Rysunek 4 Charakterystyka złożoności czasowej dla operacji getSize

3.2 Open addressing

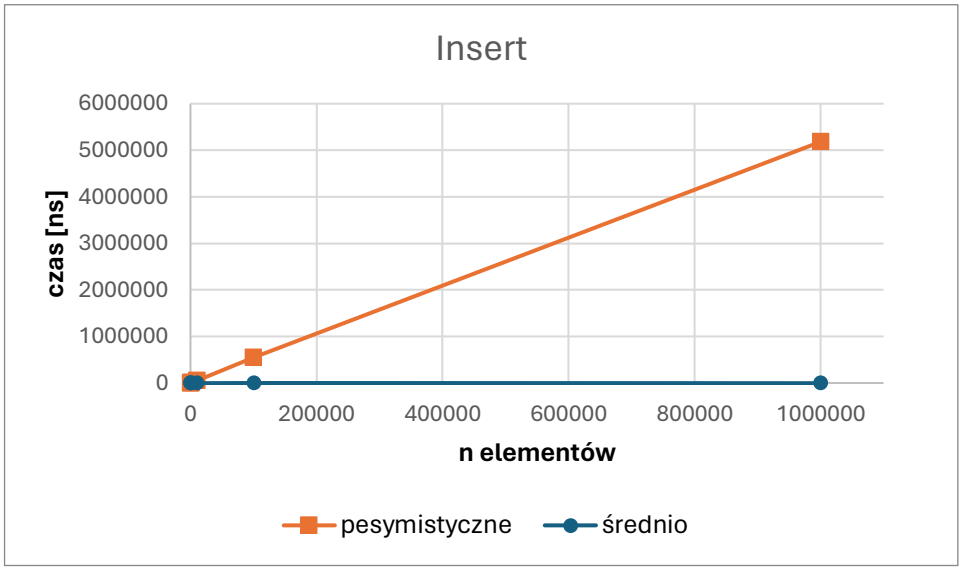
Badania przypadków średniego oraz pesymistycznego dla badanych operacji przeprowadzono, tak jak w powyższych opisach dla Separate chaining.

3.2.1 Insert

Badanie wydajności operacji dodawania elementu do tablicy mieszającej.

insert			
średnio		pesymistycznie	
n	czas [ns]	n	czas [ns]
10	329	10	325
100	375	100	2250
1000	342	1000	12833
10000	354	10000	54042
100000	375	100000	543000
1000000	333	1000000	5180375

Tabela 4 Wyniki operacji insert



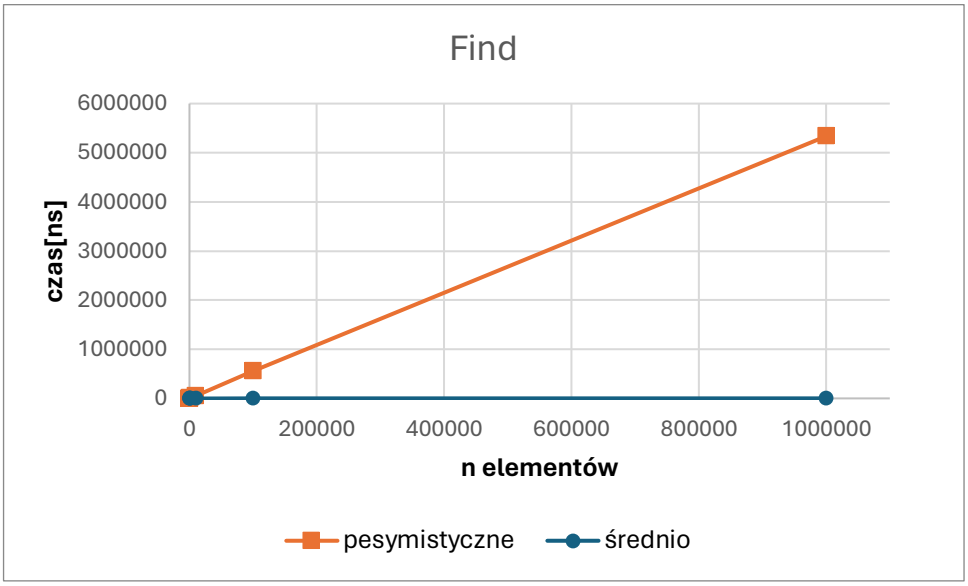
Rysunek 5 Charakterystyka złożoności czasowej dla operacji Insert

3.2.2 Find

Badanie wydajności operacji znalezienia elementu w tablicy mieszającej.

find			
średnio		pesymistycznie	
n	czas [ns]	n	czas [ns]
10	209	10	375
100	250	100	2334
1000	208	1000	13708
10000	243	10000	52250
100000	208	100000	563042
1000000	250	1000000	5335833

Tabela 5 Wyniki operacji Find



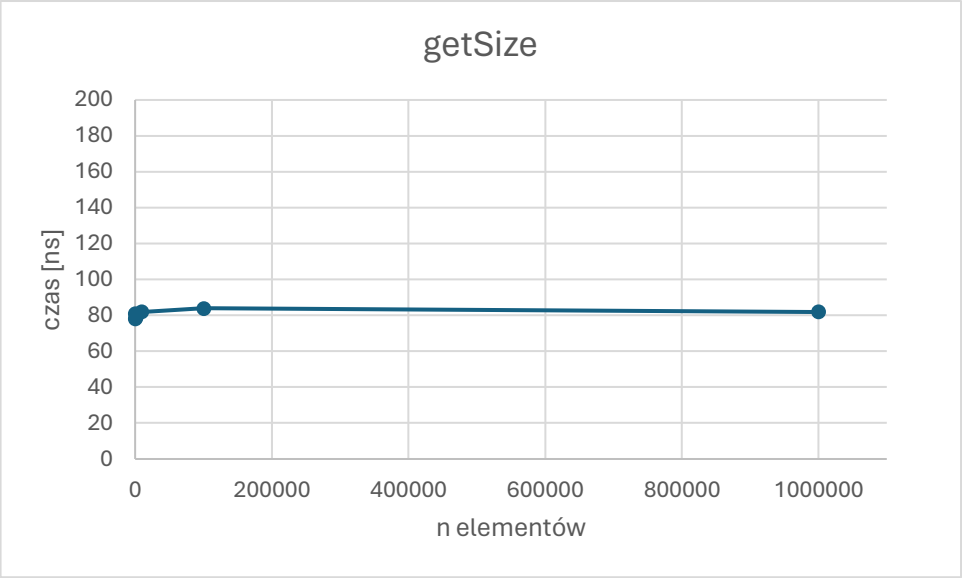
Rysunek 6 Charakterystyka złożoności czasowej dla operacji Find

3.2.3 getSize

Badanie operacji zwrócenia rozmiaru tablicy mieszającej.

getSize	
n	czas [ns]
10	81
100	78
1000	79
10000	82
100000	84
1000000	82

Tabela 6 Wyniki operacji getSize



Rysunek 7 Charakterystyka złożoności czasowej dla operacji getSize

3.3 Cuckoo hashing

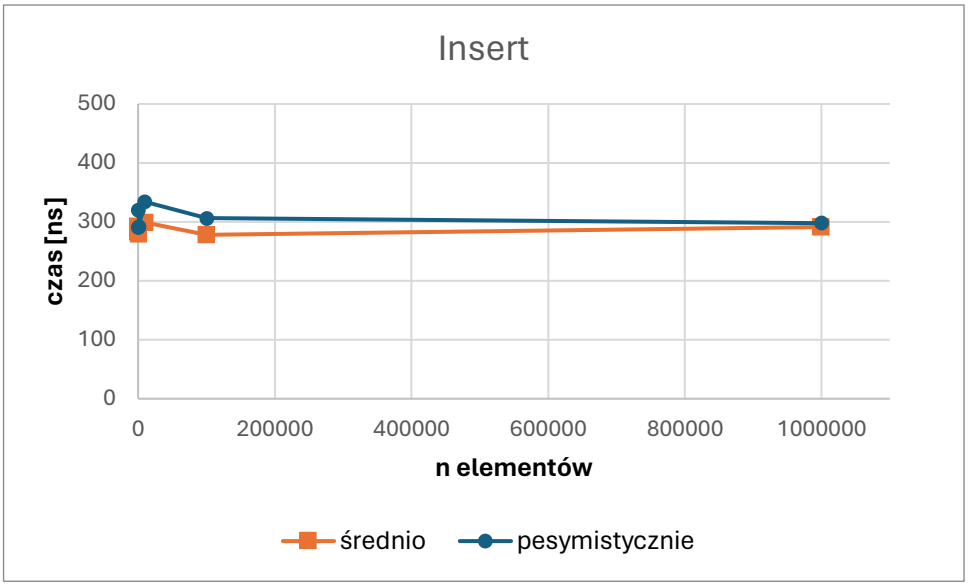
Badania przypadków średniego oraz pesymistycznego dla badanych operacji przeprowadzono, tak jak w powyższych opisach dla Separate chaining.

3.3.1 Insert

Badanie wydajności operacji dodawania elementu do tablicy mieszającej.

insert			
średnio		pesymistycznie	
n	czas [ns]	n	czas [ns]
10	284	10	320
100	292	100	291
1000	279	1000	292
10000	299	10000	334
100000	278	100000	306
1000000	291	1000000	298

Tabela 7 Wyniki operacji insert



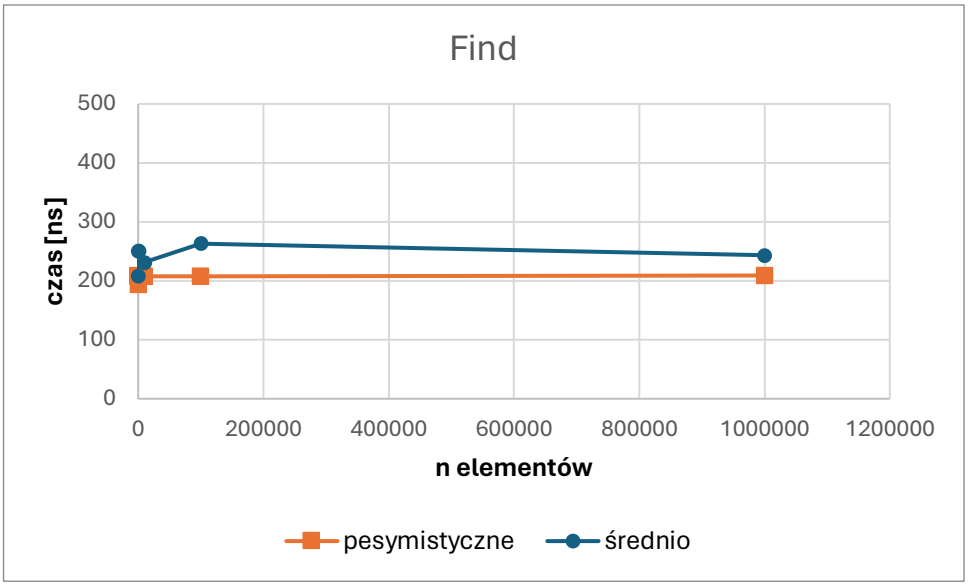
Rysunek 8 Charakterystyka złożoności czasowej dla operacji Insert

3.3.2 Find

Badanie wydajności operacji znalezienia elementu w tablicy mieszającej.

find			
średnio		pesymistycznie	
n	czas [ns]	n	czas [ns]
10	208	10	208
100	250	100	209
1000	250	1000	194
10000	231	10000	208
100000	263	100000	208
1000000	243	1000000	209

Tabela 8 Wyniki operacji find



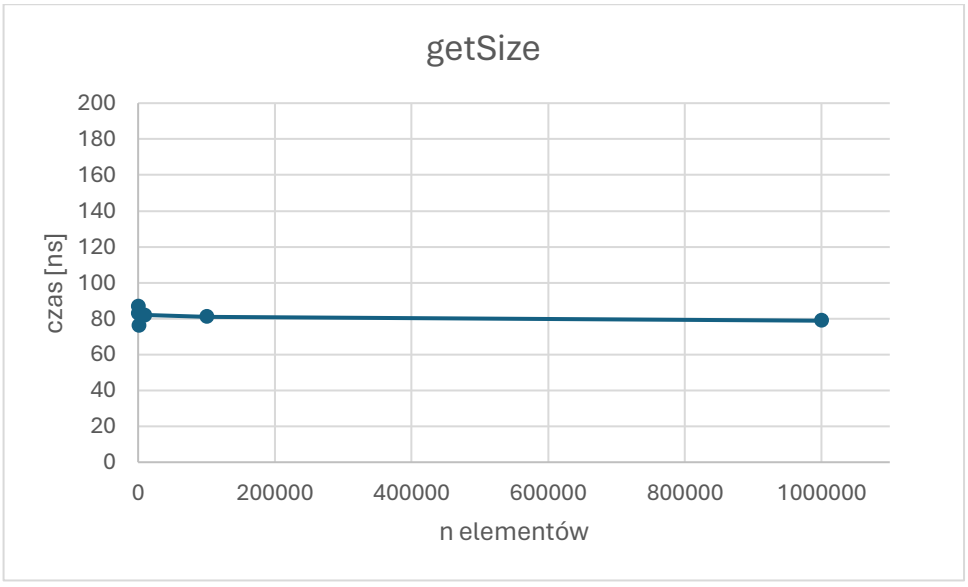
Rysunek 9 Charakterystyka złożoności czasowej dla operacji Find

3.3.3 getSize

Badanie operacji zwrócenia rozmiaru tablicy mieszającej.

getSize	
n	czas [ns]
10	83
100	87
1000	76
10000	82
100000	81
1000000	79

Tabela 9 Wyniki operacji getSize



Rysunek 10 Charakterystyka złożoności czasowej dla operacji getSize

3.4 Złożoność obliczeniowa

Następnie przeanalizowano złożoność wyników zgodnie z notacją dużego O.

Struktura	Insert		Find		getSize
	Średnio	Pesymistyczny	Średnio	Pesymistyczny	
Separate	O(1)	O(n)	O(1)	O(n)	O(1)
Open	O(1)	O(n)	O(1)	O(n)	O(1)
Cuckoo	O(1)	O(1)	O(1)	O(1)	O(1)

Tabela 10 Analiza wyników operacji zgodnie z notacja dużego O

Wyniki otrzymane po przebadaniu i przeanalizowaniu zostały porównane z wynikami z literatury naukowej.

Struktura	find()/remove()		insert()		Uporządk.
	Avg.	Worst	Avg.	Worst	
Hash table[2]	$O(1)$	$O(n)$	$O(1)$	$O(n)$	nie
Hash table[3]	$O(1)$	$O(\log n)$	$O(1)$	$O(\log n)$	nie
Hash table[4]	$O(1)$	$O(1)$	$O(1)$	$O(1)$[5]	nie
AVL/black-red tree	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	tak
BST	$O(\log n)$	$O(n)$	$O(\log n)$	$O(n)$	tak
Lista[6]	$O(n)$	$O(n)$	$O(1)/O(n)$	$O(1)/O(n)$	„tak"

[2] Adresowanie otwarte lub kubełki z listą
[3] Adresowanie zamknięte plus kubełki ze zbalansowanym BST
[4] Cuckoo hashing
[5] Koszt zamortyzowany
[6] Dwa warianty: dodawanie na koniec lub dodawanie w pozycji posortowanej

Rysunek 11 Wyniki przedstawione na wykładzie

Nasze wyniki potwierdzają oczekiwania teoretyczne i są zgodne z wynikami przedstawionymi na wykładzie, które traktujemy jako wyniki z literatury.

4 Porównanie

Porównanie wyników i złożoności obliczeniowych dla zaimplementowanych sposobów rozwiązywania kolizji

Struktura	Insert		Find		getSize
	Średnio	Pesymistyczny	Średnio	Pesymistyczny	
Separate	O(1)	O(n)	O(1)	O(n)	O(1)
Open	O(1)	O(n)	O(1)	O(n)	O(1)
Cuckoo	O(1)	O(1)	O(1)	O(1)	O(1)

Tabela 11 Porównanie złożoności obliczeniowych operacji dla różnych implementacji

- Wyniki dla operacji getSize, która odpowiada za zwrócenie ilości elementów w tablicy mieszającej, wypadają tak samo dla wszystkich implementacji. Złożoność obliczeniowa dla każdego ze sposobów jest O(1).
- Operacja Insert, która odpowiada za dodanie elementu do tablicy mieszającej, średnio dla każdego ze sposobów wypada tak samo charakteryzując się stałą złożonością O(1). Natomiast w pesymistycznym przypadku zdecydowanie lepiej wypada Cuckoo hashing, które ma złożoność obliczeniowa O(1), gdy Open addressing oraz Separate Chaining mają liniową złożoność obliczeniowa O(n).
- Dla operacji find, odpowiadającej za znalezienie elementu w tablicy mieszającej, najlepiej wypada Cuckoo hashing, które zarówno dla przypadku średniego jak i pesymistycznego charakteryzuje się złożonością obliczeniowa O(1). Separate Chaining i Open addressing średnio, również mają stałą złożoność obliczeniowa O(1), natomiast w pesymistycznym przypadku złożoność obliczeniowa operacji find dla tych implementacji wynosi O(n).

5 Wnioski

- Wybór implementacji: Wyniki przeprowadzonych badań potwierdzają, że wybór odpowiedniego sposobu rozwiązywania kolizji jest podyktowany wymaganiami i charakterystyką danych na jakich będziemy pracować. Dla uniwersalnych przypadków każda z implementacji wypada tak samo i wybór zależy od nas. Natomiast w przypadku gdy nie wiemy z jakimi danymi będziemy mieć do czynienia lub gdy jest duże prawdopodobieństwo wystąpienia kolizji najlepsze będzie Cuckoo Hashing, które zapewnia stałą złożoność dla wszystkich operacji.

- Wydajność w przypadku pesymistycznym: Warto zauważyć, że Cuckoo Hashing wypada znacznie lepiej niż Separate Chaining i Open Addressing w przypadku pesymistycznym. Podczas gdy te ostatnie dwie metody mają liniową złożoność obliczeniową O(n) dla operacji Find i Insert, Cuckoo Hashing zachowuje stałą złożoność O(1), co oznacza, że niezależnie od ilości elementów w tablicy, czas wykonania operacji pozostaje taki sam.

- Zarządzanie pamięcią: Separate chaining wymaga przechowywania list wiązanych dla kolidujących elementów, natomiast pozwala na najwyższy load factor co sprawia, że marnuje się najmniej pustych indeksów w tablicy. Open addressing oraz Cuckoo hashing nie zużywają miejsca na dodatkową

strukturę za to wymagają niższego poziomu load factor, co wiąże się z nieefektywnym zarządzaniem pamięcią – występowanie pustych indeksów.

- Implementacja: Separate Chaining i Open Addressing są relatywnie prostsze do zaimplementowania w porównaniu do Cuckoo Hashing. Implementacja Cuckoo Hashing wymaga bardziej skomplikowanego zarządzania dwoma współzależnymi tablicami mieszającymi i mechanizmami przenoszenia elementów, co czyni ją bardziej wymagającą pod względem implementacyjnym.

Podsumowując wybór konkretnego sposobu rozwiązywania kolizji zależy od konkretnej sytuacji, charakterystyki danych i wymagań programu.

6 Bibliografia

- Data Structures and Algorithms in C++, 2nd Edition Michael T. Goodrich, Roberto Tamassia, David M. Mount ISBN: 978-0-470-38327-8
- Wykład 8 - Tablice mieszające, Jarosław Rudy http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/files/sd/w8.pdf
- https://www.baeldung.com/cs/cuckoo-hashing
- https://en.wikipedia.org/wiki/Hash_table