Struktury Danych	
Kierunek *Informatyczne Systemy Automatyki*	**Termin** *środa TN 15:15 – 16:55*
Imię, nazwisko, numer albumu *Jakub Wojtala 272542, Mateusz Marko 273168*	**Data** *5.06.2024r.*
Link do projektu https://github.com/kubikal7/HashTables/	

Sprawozdanie – miniprojekt nr 3

W niniejszym projekcie skupiamy się na implementacji oraz analizie trzech wariantów tablic mieszających: Separate Chaining, adresowanie zamknięte z kubełkami ze zbalansowanym BST oraz Cuckoo hashing. Naszym celem jest zarówno praktyczna implementacja tych struktur, jak i dokładne zbadanie ich złożoności czasowej przy wykonywaniu podstawowych operacji, takich jak wstawianie czy usuwanie danych. Analiza ta pozwoli nam określić, w jakich scenariuszach każda z tych metod haszowania wykazuje lepszą wydajność, co jest kluczowe dla optymalizacji algorytmów oraz aplikacji korzystających z tablic haszujących. Projekt ten, będący w istocie implementacją słownika opartego na tablicy mieszającej, umożliwi nam lepsze zrozumienie tych mechanizmów oraz bardziej efektywne wykorzystywanie konkretnych metod.

Spis treści

1. Wstęp

• Wstęp teoretyczny

➢ **Słownik (Dictionary)** - abstrakcyjna struktura danych służąca do operowania na parach klucz-wartość. Tablica mieszająca (hash table) to struktura danych używana do implementacji słownika, często błędnie nazywana „tablicą haszującą”. Główną ideą jest zamiana klucza na indeks tablicy przy użyciu funkcji mieszającej (hash function). Ponieważ liczba możliwych kluczy zwykle przewyższa liczbę indeksów tablicy, występują kolizje - sytuacje, w których dwa różne klucze są przypisane do tego samego indeksu. Istnieje kilka metod radzenia sobie z kolizjami, takich jak separate chaining czy cuckoo hashing. Każda z tych metod ma swoje unikalne zalety i zastosowania i nieco inaczej zadziała w przypadku kolizji, w zależności od specyfiki przechowywanych danych i wymagań dotyczących wydajności.

• Rozwiązywanie kolizji

➢ **Metoda łańcuchowa (separate chaining)** – Jest to metoda, w której każdy kubełek tablicy mieszającej przechowuje listę wszystkich elementów, które mapują się do tego samego indeksu. Dzięki temu, gdy wystąpi kolizja, nowy element jest po prostu dodawany do listy powiązanej z danym kubełkiem.

Operacje i złożoność

- ***insert***(k, v) - Wstawianie klucza k z wartością v. Złożoność $O(1)$ w najlepszym przypadku, gdy kubełek do którego mapuje się klucz jest pusty. Złożoność najgorsza $O(n)$, gdy wszystkie elementy są zmapowane do tego samego kubełka tworząc długą listę.
- ***remove***(k) - Usunięcie klucza k wraz z jego wartością. Złożoność $O(1)$ w najlepszym przypadku, gdy element znajduje się na początku listy w kubełku i w najgorszym $O(n)$, gdy element znajduje się na końcu długiej listy w kubełku.

➢ **Adresowanie zamknięte (closed addressing)** – w tej metodzie każdy kubełek tablicy mieszającej może przechowywać wiele elementów, które są organizowane w strukturę danych, taką jak zbalansowane drzewo BST. Gdy wystąpi kolizja, nowy element jest dodawany do odpowiedniego kubełka, a struktura wewnątrz kubełka zarządza wieloma elementami.

Operacje i złożoność

- ***insert***(k, v) - Wstawianie klucza k z wartością v. Złożoność $O(1)$ w najlepszym przypadku, gdy kubełek do którego mapuje się klucz jest pusty. Złożoność najgorsza $O(log\ n)$, gdy kubełek do którego mapuje się klucz, zawiera wiele elementów i wymaga zrównoważenia drzewa BST.
- ***remove***(k) - Usunięcie klucza k wraz z jego wartością. Złożoność $O(1)$ w najlepszym przypadku, gdy element znajduje się na początku struktury w kubełku i w najgorszym $O(log\ \ n)$, gdy element znajduje się głęboko w zbalansowanym drzewie BST i wymaga przeszukania oraz zrównoważenia drzewa po usunięciu.

- **Haszowanie kukułcze (Cuckoo hashing)** - metoda haszowania, w której używane są dwie lub trzy tablice z osobnymi funkcjami haszującymi. Każdy klucz ma dwie możliwe lokalizacje. Gdy wystąpi kolizja, nowy element przemieszcza już istniejący element do jego alternatywnej lokalizacji, co może wywołać serię przemieszczeń.

  Operacje i złożoność

  - ***insert*(k, v)** - Wstawianie klucza k z wartością v. Złożoność $O(1)$ w najlepszym przypadku, gdy oba możliwe kubełki dla nowego elementu są puste. Złożoność najgorsza $O(n)$, gdy wystąpi cykl przemieszczeń, co wymaga ponownego haszowania tablic.
  - ***remove*(k)** - Usunięcie klucza k wraz z jego wartością. Cuckoo hashing gwarantuje, że element jest zawsze w jednym z dwóch miejsc, więc złożoność usunięcia wynosi $O(1)$.

• Założenia projektowe

- Projekt zakłada implementację słownika opartego na tablicy mieszającej z wykorzystaniem trzech metod rozwiązywania kolizji: adresowania zamkniętego z kubełkami opartymi na zbalansowanych drzewach BST, cuckoo hashing oraz metodę łańcuchową. Głównym celem jest porównanie tych metod pod kątem złożoności czasowej ich podstawowych operacji, takich jak wstawianie i usuwanie elementów. Projekt ma na celu analizę efektywności każdej z tych metod w różnych scenariuszach, co jest kluczowe dla optymalizacji algorytmów korzystających z tablic mieszających.

- Eksperymenty przeprowadzone zostały na czterech populacjach po 25 prób, a nastepnie została z nich wyciągnięta średnia. Same badania z kolei wykonane zostały na stacjonarnym komputerze wyposażonym w procesor ośmiordzeniowy AMD Ryzen 7 2700, dysk SSD i 16 GB pamięci RAM . Oprogramowanie użyte do implementacji i testów to środowisko programistyczne Visual Studio, z wykorzystaniem języka C++ bez dostępu do gotowych bibliotek takich jak STL, co wymagało samodzielnej implementacji wszystkich struktur danych oraz skorzystania z utworzonych wcześniej struktur w ramach poprzedniego projektu.

• Funkcje mieszające

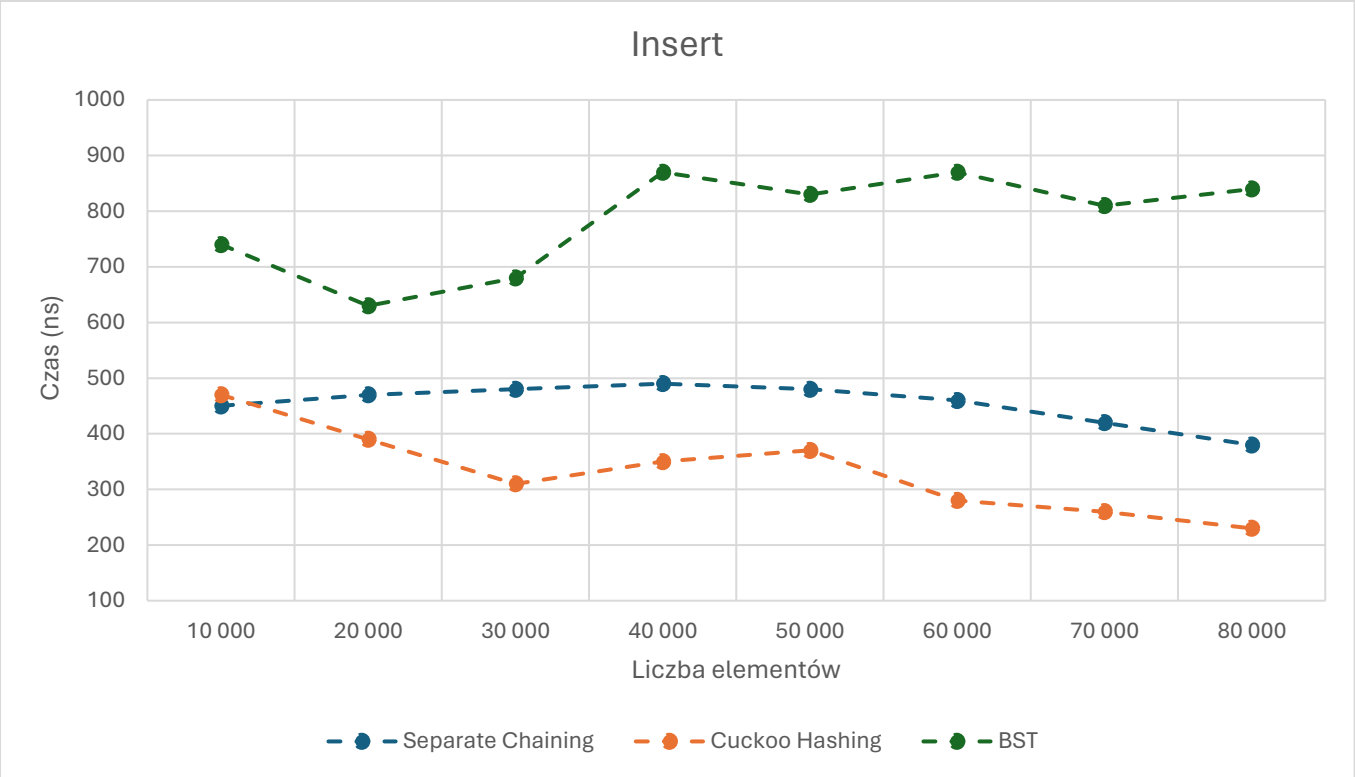
- Dla **Separate chaining**: $h(x) = x \ mod \ m$
- Dla **kubełków ze zbalansowanym BST**: $h(x) = x \ mod \ m$
- Dla **Cuckoo Hashing**:
  1. $h(x) = x \ mod \ m$
  2. $h(x) = (x * 500) \ mod \ m$
  3. $h(x) = (x + 100\,000) \ mod \ m$

2. Badania

a) Operacja wstawiania (insert)

Tabela 1 Wstawianie - czas operacji (ns) dla poszczególnej ilości elementów i wybranych implementacji tablicy mieszającej

Czas (ns)	Liczba elementów							
	10 000	20 000	30 000	40 000	50 000	60 000	70 000	80 000
Separate Chaining	450	470	480	490	480	460	420	380
Cuckoo Hashing	470	390	310	350	370	280	260	230
BST	740	630	680	870	830	870	810	840



Rysunek 1 Wykres zależności czasu (ns) od liczby elementów dla wybranych implementacji tablicy mieszającej – operacja wstawiania

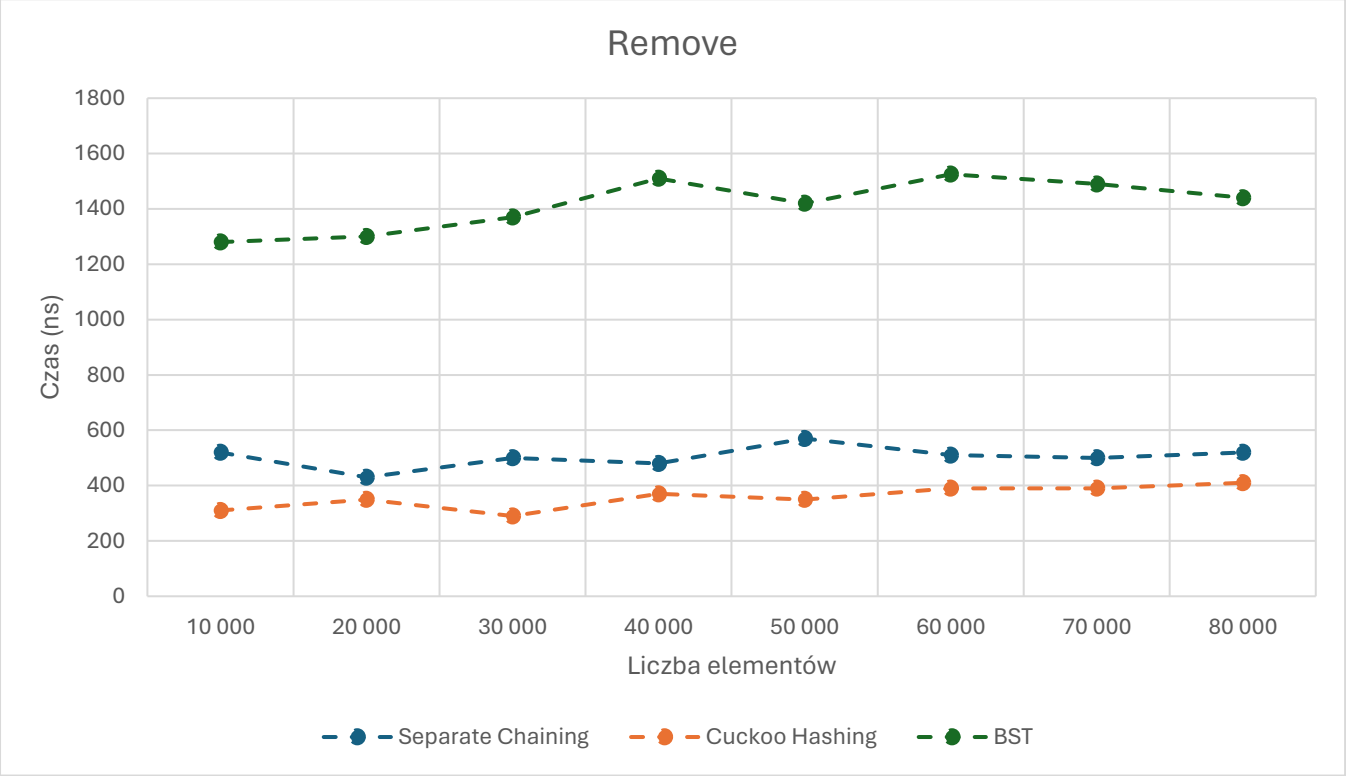
Jak możemy zauważyć czasy wstawiania elementów różnią się. Metoda cuckoo hashing ma najkrótsze czasy wstawiania, utrzymując się poniżej 300 ns niezależnie od liczby elementów. Separate chaining również radzi sobie dobrze, z czasami w okolicach 400 ns. Natomiast metoda Adresowania zamkniętego z BST ma najwyższe czasy wstawiania, sięgające nawet 900 ns. Ogólnie rzecz biorąc, Cuckoo Hashing jest najwydajniejsza dla operacji wstawiania.

Politechnika Wrocławska

b) Operacja usuwania klucza (remove)

Tabela 2 Usuwanie - czas operacji (ns) dla poszczególnej ilości elementów i wybranych implementacji tablicy mieszającej

Czas (ns)	Liczba elementów							
	10 000	20 000	30 000	40 000	50 000	60 000	70 000	80 000
Separate Chaining	520	430	500	480	570	510	500	520
Cuckoo Hashing	310	350	290	370	350	390	390	410
BST	1280	1300	1370	1510	1420	1525	1490	1440



Rysunek 2 Wykres zależności czasu (ns) od liczby elementów dla wybranych implementacji tablicy mieszającej – operacja usunięcia

Tutaj widzimy że metoda separate chaining ponownie pokazuje jedne z najniższych czasy operacji około 500 ns. Cuckoo hashing wciąż jednak sprawuje się lepiej, dalej ma stabilne czasy usuwania w granicach 400 ns. I znów metoda adresowania zamkniętego z BST wykazuje najwyższe czasy, przekraczając nawet 1000 ns. Cuckoo hashing zdaje się być także najefektywniejszym wyborem dla operacji usuwania.

3. Wnioski

- Metoda łańcuchowa wykazuje stabilność zarówno w operacjach wstawiania, jak i usuwania, niezależnie od liczby elementów. Jest skuteczna przy dużym obciążeniu, ponieważ każda operacja ogranicza się do przeszukiwania pojedynczych kubełków. Jednakże, przy dużej liczbie kolizji, długość list w kubełkach może wpłynąć na wydajność, co wydłuży czasy przeszukiwania kubełka pod konkretnym indeksem.

- Adresowanie zamknięte z kubełkami opartymi na zbalansowanych drzewach BST wykazało najgorsze czasy zarówno dla wstawiania, jak i usuwania. Teoretycznie w najlepszym przypadku ma złożoność $O(1)$, jednakże w najgorszym, gdy drzewo jest niezbalansowane, złożoność może wzrosnąć nawet do $O(\log n)$. BST jest więc najbardziej efektywna w warunkach, gdzie dane są w miarę równomiernie rozłożone i można utrzymać zrównoważoną strukturę drzewa. Praktycznie w testach dla tablicy mieszającej z rozwiązaniem kolizji poprzez drzewo BST zarówno wstawianie jak i usuwanie okazało złożoność $O(1)$, co świadczy o tym, że dane w tablicy były rozłożone równomiernie, jednakże testy pokazały, że czasowo było gorsze niż rozwiązanie z Cuckoo Hashing czy Separate Chaining.

- Haszowanie kukułcze, dzięki zastosowaniu dwóch lub trzech funkcji haszujących zapewnia zazwyczaj bardzo dobre czasy zarówno dla operacji wstawiania jak i usuwania, ale może być bardziej czasochłonne w przypadku wystąpienia cykli, co czasami wymaga przetasowania tablic. Jest więc idealne dla aplikacji, które wymagają szybkiego wstawiania i częstych operacji dodawania bądź usuwania, ale mogą tolerować okazjonalne przetasowania.

4. Bibliografia

- http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/files/sd/w8.pdf
- https://www.geeksforgeeks.org/cuckoo-hashing/
- https://geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/
- https://www.baeldung.com/cs/hashing-separate-chaining