Struktury danych	
Kierunek *Informatyczne Systemy Automatyki*	**Termin** *środa TP* $15^{15} - 16^{55}$
Imię, nazwisko, numer albumu *Piotr Brajer 272538*	**Data** *04.06.2024*
Link do Projektu https://github.com/ElemkayZ/strukturyDanych/tree/main/projekt3-HashTables	

Sprawozdanie – Projekt 3 – Tablice Mieszające

1) Otwarte adresowanie z kwadratowym próbkowaniem

Funkcja Haszująca: (klucz) mod (rozmiar tablicy)
Rozwiązywanie konfliktów: wykonywane próbkowanie jako kwadrat kolejnych liczb naturalnych np.: dla kolejnych próbek 1,2,3,… przesunięcie wynikowe to 1,4,9,…
Dodawanie:

1. Sprawdź czy nie przekroczono load limit, jeśli tak rozszerz tablice
2. Wykonaj hash klucza
3. Sprawdź czy miejsce w tablicy wolne, jeśli nie to wykonuj próbkowanie do momentu wolnego miejsca

Usuwanie:

1. Wykonaj hash klucza
2. Sprawdź czy to odpowiedni klucz na miejscu tablicy, jeśli nie to wykonuj próbkowanie do momentu znalezienia

2) Oddzielne łączenie - Separate Chaining

Funkcja Haszująca: (klucz) % (rozmiar tablicy)
Rozwiązywanie konfliktów: na zajętym miejscu w tablicy tworzona jest lista kierunkowa
Dodawanie:

1. Sprawdź czy nie przekroczono load limit, jeśli tak rozszerz tablice
2. Wykonaj hash klucza
3. Nowy element dodaj na końcu listy kierunkowej na index tablicy

Usuwanie:

1. Wykonaj Hash klucza
2. Przejdź po tablicy do momentu znalezienia u usuń

2) Podwójne mieszanie – Double Hashing

Funkcje Haszujące:

1. (klucz)mod(rozmiar tablicy)
2. ([(klucz)*0,6180339887]mod1)* (rozmiar tablicy)

Porównaj wyniki i jako wynik funkcji podaj hash na miejscu z mniejszą listą kierunkową

Rozwiązywanie konfliktów: na zajętym miejscu w tablicy tworzona jest lista kierunkowa

Dodawanie:

4. Sprawdź czy nie przekroczono load limit, jeśli tak rozszerz tablice
5. Wykonaj hash klucza
6. Nowy element dodaj na końcu listy kierunkowej na index tablicy

Usuwanie:

3. Wykonaj Hash klucza 1 funkcji
4. Przejdź po liście do momentu znalezienia, po czym wykonaj 7.
5. jeśli nie znaleziono wykonaj 6.
6. Wykonaj Hash klucza 2 funkcji i wykonaj 4.
7. Usuń element

3) Złożoności obliczeniowe struktur:

struktura	insert		remove	
	avg	worst	avg	worst
open addressing	O(1)	O(n)	O(1)	O(n)
separate chaining	O(1)	O(n)	O(1)	O(n)
double hashing	O(1)	O(n)	O(1)	O(n)

4) Założenia projektowe:

a) Wielkość struktur:

i) Każda struktura została badana na: 250 000,400 000, 500 000, 600 000, 750 000, 1 000 000, 1 500 000 ilości elementów

b) Sposób generowania elementów do struktur:

i) Do wygenerowania liczb wykorzystałem mt19937, na przedziałach 0-10 000 000

```
std::mt19937 rng(time(0));
std::uniform_int_distribution<unsigned long> rNum(0,10000000);
```

```
for (int i = 0; i < dataBaseSize ; ++i) {
    oHash.insert(*new Pair(rNum(rng),rNum(rng)));
    dHash.insert(rNum(rng),rNum(rng));
    chHash.insert(rNum(rng),rNum(rng));
}
```

c) Sposób Pomiaru czasu funkcji struktury:
   i) Wygenerowano 100 przypadkowo generowanych zbiorów danych dla każdej struktury
   ii) Każda funkcja została mierzona pojedynczo 100 razy
   iii) W sumie każdą funkcje mierzono 10 000 razy a średnia pomiarów została podana w tabelach i wykresach

```
for (int i = 0; i < 100; i++)
{
    auto start = std::chrono::high_resolution_clock::now();
    chHash.insert(rNum(rng),rNum(rng));
    auto stop = std::chrono::high_resolution_clock::now();
    d = stop - start;
    chHashInsert += d.count();
}
```

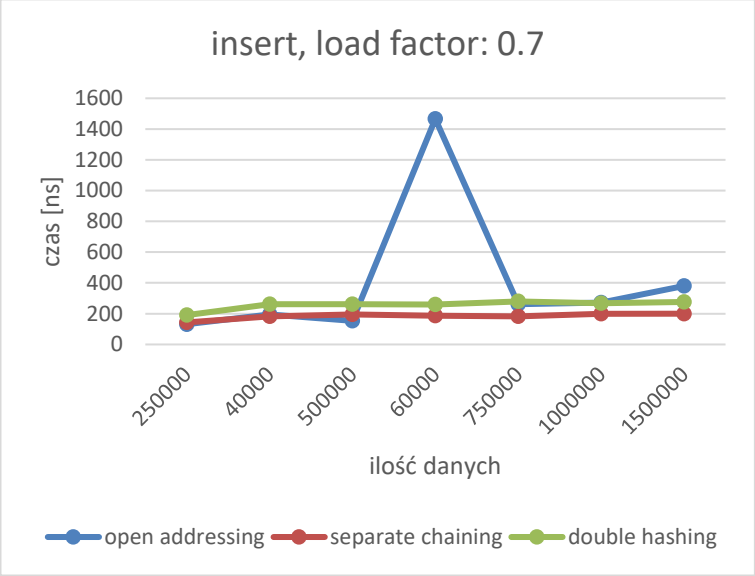
Badania:

Tabela 1 : średni czas wykonywania funkcji[ns], load factor: 0.7

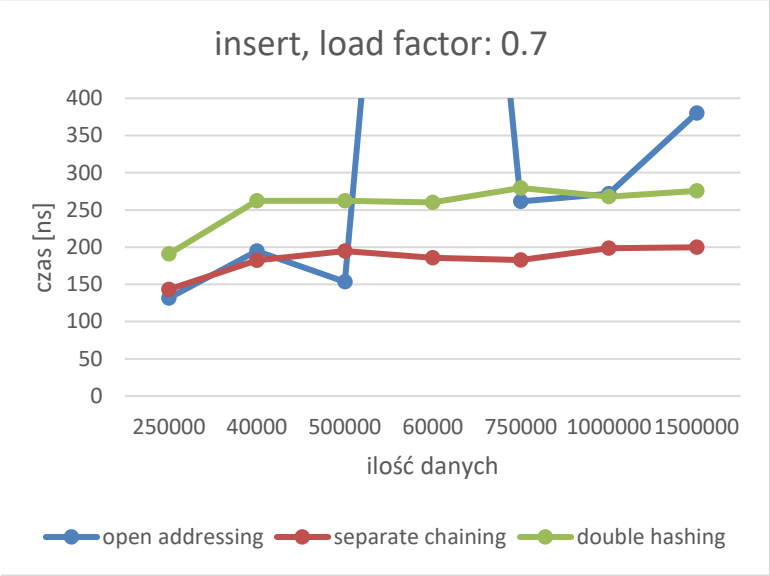
funkcja	struktura	250000	400000	500000	600000	750000	1000000	1500000
insert	open addressing	131,55	194,97	153,62	1465,91	261,59	271,95	380,16
	separate chaining	143,23	182,22	194,8	185,73	182,89	198,45	199,84
	double hashing	190,77	262,51	262,35	260,31	279,52	267,87	275,71
remove	open addressing	81,57	115,31	93,95	89,51	135,73	127,68	139,95
	separate chaining	108,89	140,48	146,86	168,23	146,04	179,06	186,99
	double hashing	167,69	189,04	197,72	222,4	213,49	231,47	225,2

Tabela 2 : średni czas wykonywania funkcji[ns], load factor: 0.95

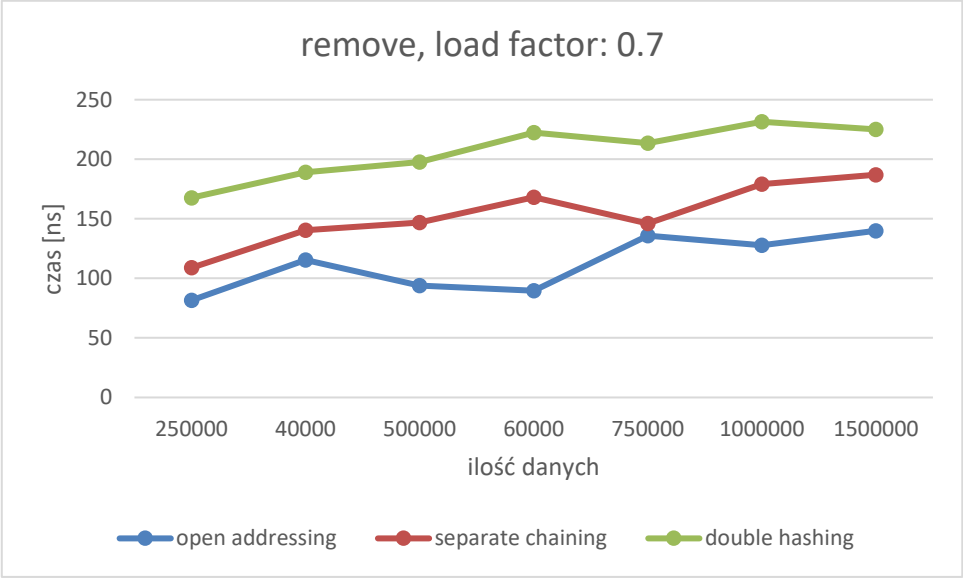
funkcja	struktura	250000	400000	500000	600000	750000	1000000	1500000
insert	open addressing	250,89	210,42	292,41	205,13	221,52	223,13	266,83
	separate chaining	193,52	190,49	729,94	194,51	225,59	209,31	270,39
	double hashing	322,64	216,46	812,55	251,85	337,51	269,63	449,63
remove	open addressing	92,64	95,07	131,79	94,57	92,87	112,77	121,1
	separate chaining	116,31	140,54	148,76	160,79	170,85	178,44	204,13
	double hashing	181,64	202,16	198,73	212,6	226,02	219,58	261,55



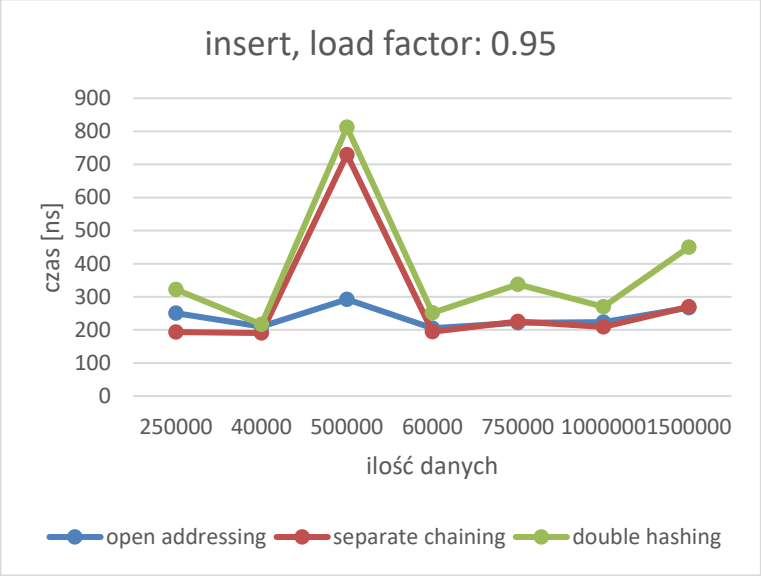
WYKRES 1: ŚREDNI CZAS WYKONYWANIA INSERT WSZYSTKICH TABLIC Z LOAD FACTOR: 0.7



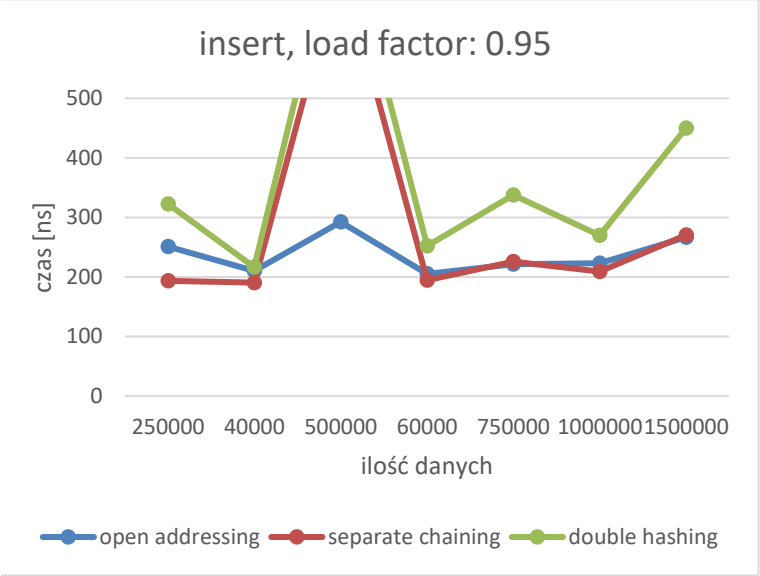
WYKRES 2: ŚREDNI CZAS WYKONYWANIA INSERT WSZYSTKICH TABLIC Z LOAD FACTOR: 0.7



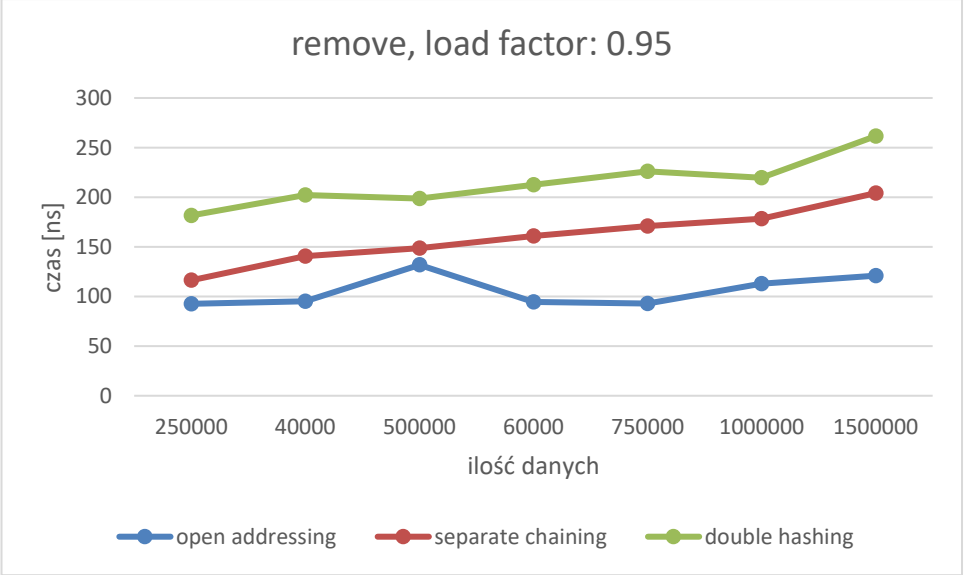
WYKRES 3: ŚREDNI CZAS WYKONYWANIA REMOVE WSZYSTKICH TABLIC Z LOAD FACTOR: 0.7



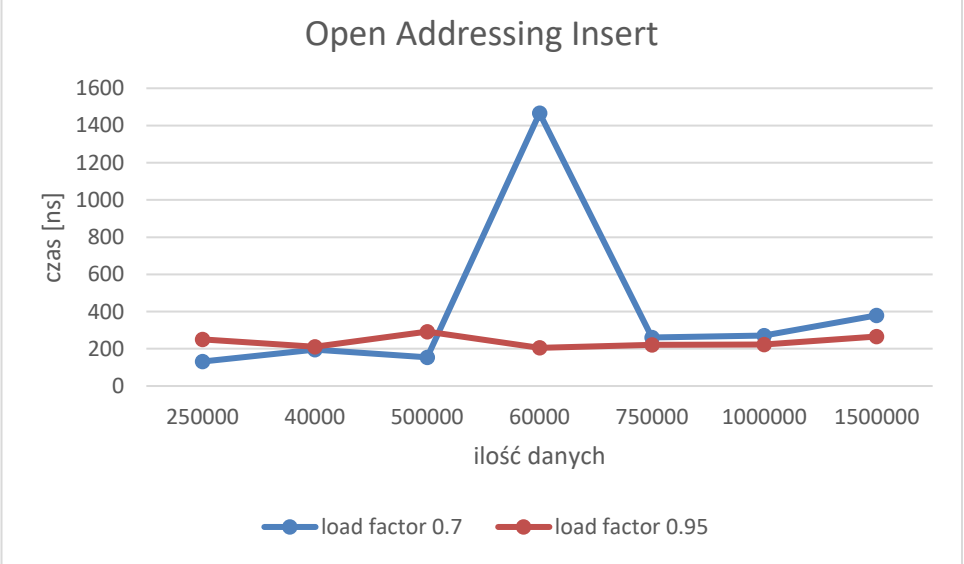
WYKRES 4: ŚREDNI CZAS WYKONYWANIA INSERT WSZYSTKICH TABLIC Z LOAD FACTOR: 0.95



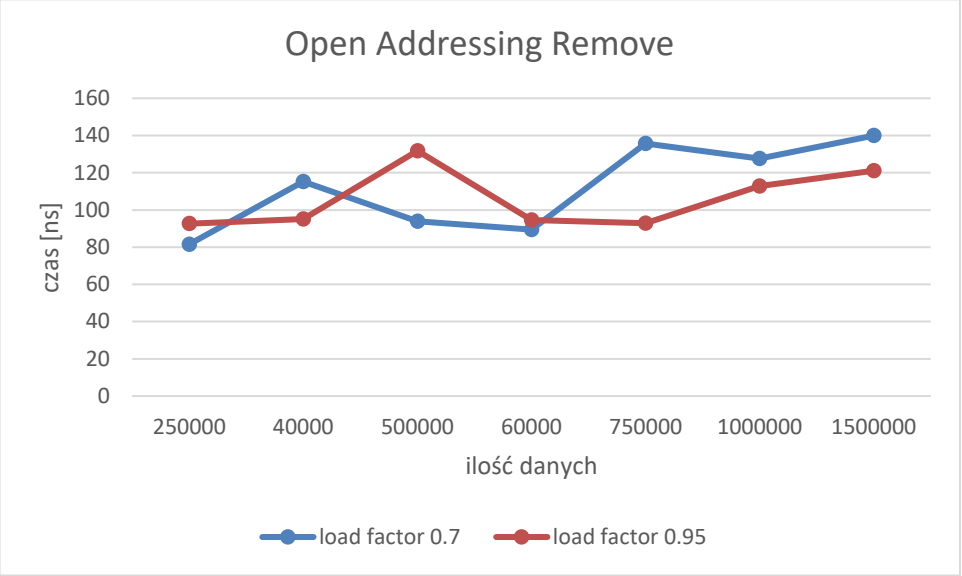
WYKRES 5: ŚREDNI CZAS WYKONYWANIA INSERT WSZYSTKICH TABLIC Z LOAD FACTOR: 0.95



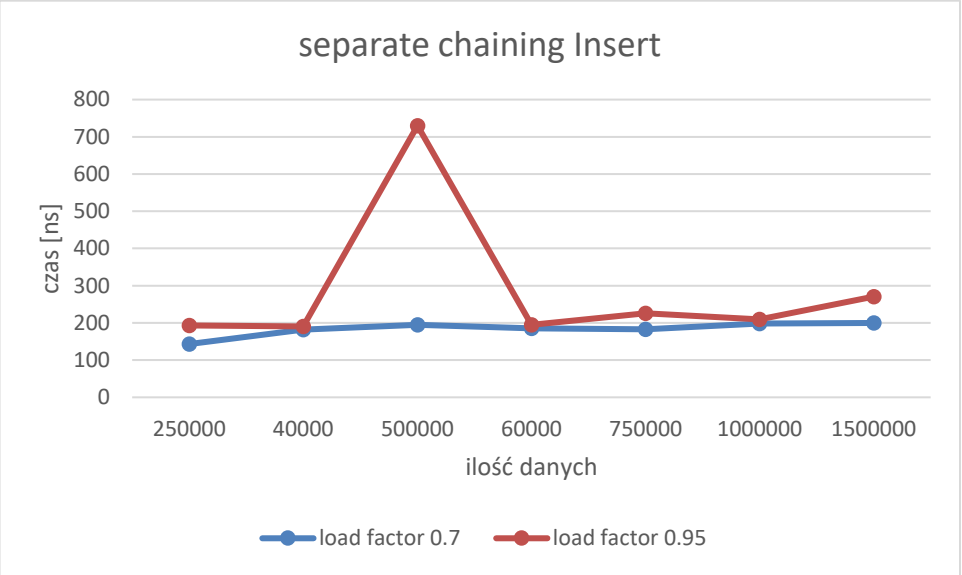
WYKRES 6: ŚREDNI CZAS WYKONYWANIA REMOVE WSZYSTKICH TABLIC Z LOAD FACTOR: 0.95



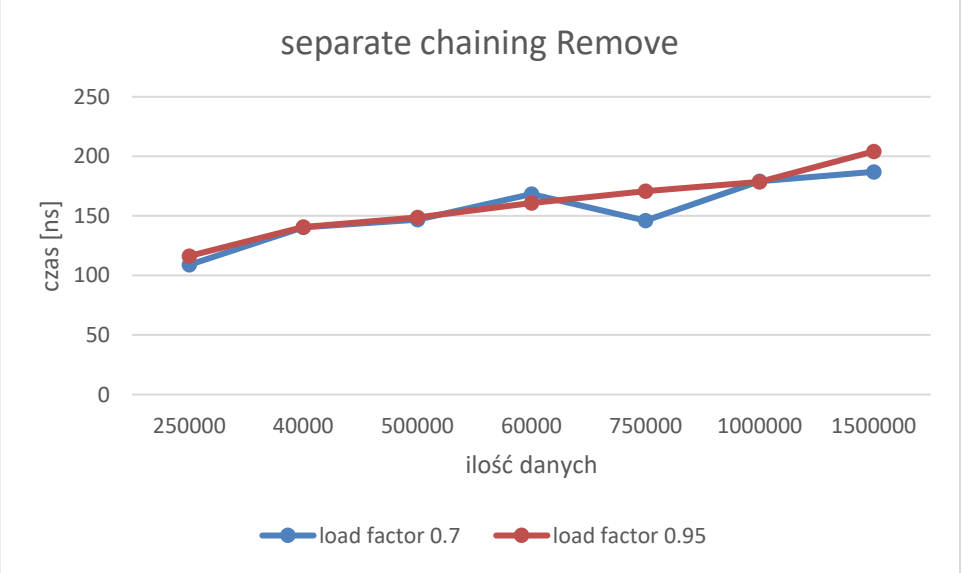
WYKRES 7:PORÓWNANIE ŚREDNICH CZASÓW FUNKCJI INSERT TABLICY OPEN ADDRESSING



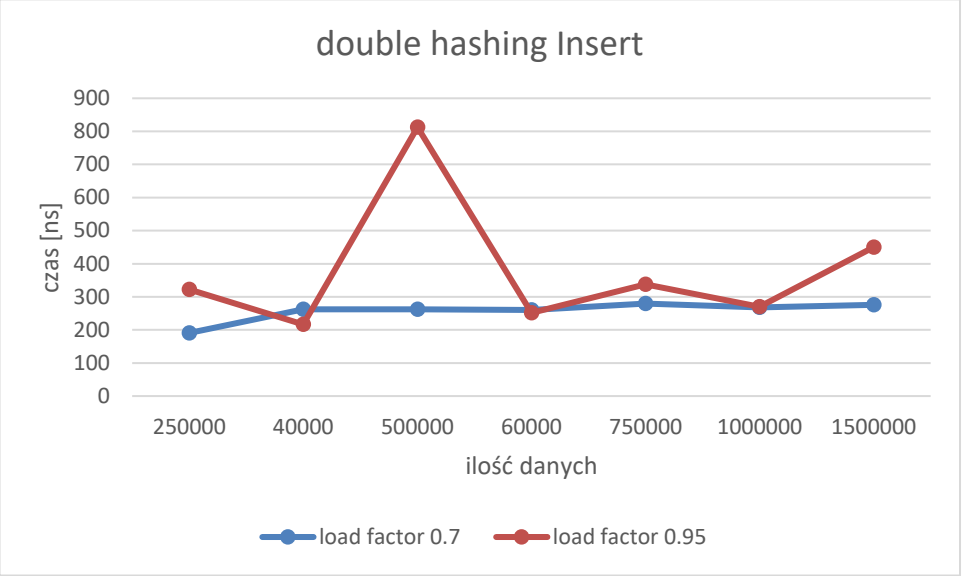
WYKRES 8: PORÓWNANIE ŚREDNICH CZASÓW FUNKCJI REMOVE TABLICY OPEN ADDRESSING



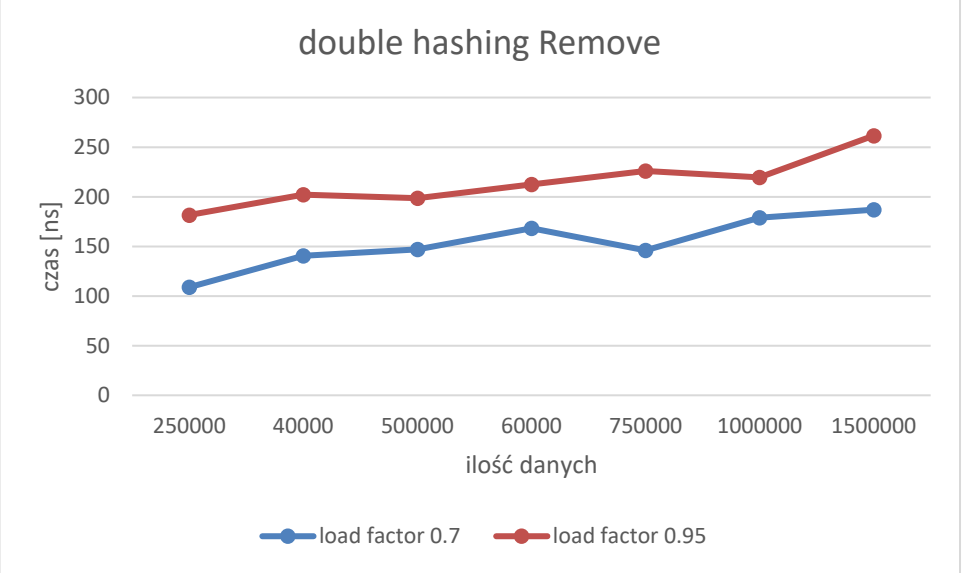
WYKRES 9:PORÓWNANIE ŚREDNICH CZASÓW FUNKCJI INSERT TABLICY SEPARATE CHAINING



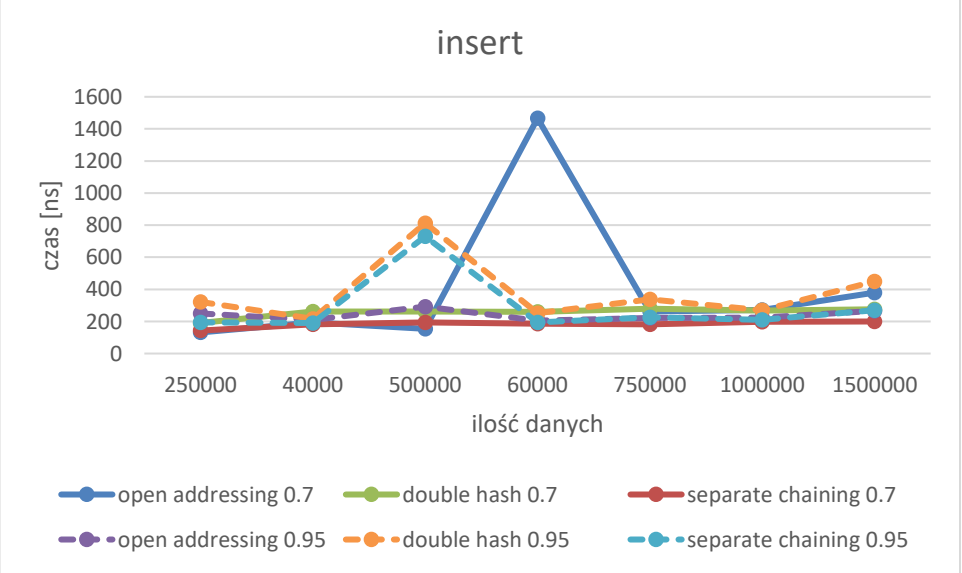
WYKRES 10: PORÓWNANIE ŚREDNICH CZASÓW FUNKCJI REMOVE TABLICY SEPARATE CHAINING



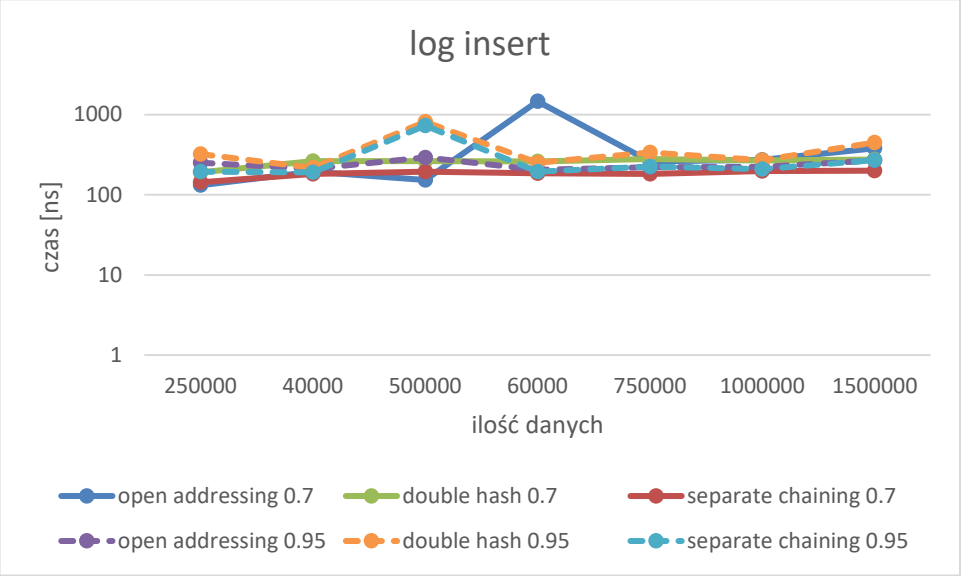
WYKRES 11:PORÓWNANIE ŚREDNICH CZASÓW FUNKCJI INSERT TABLICY DOUBLE HASHING



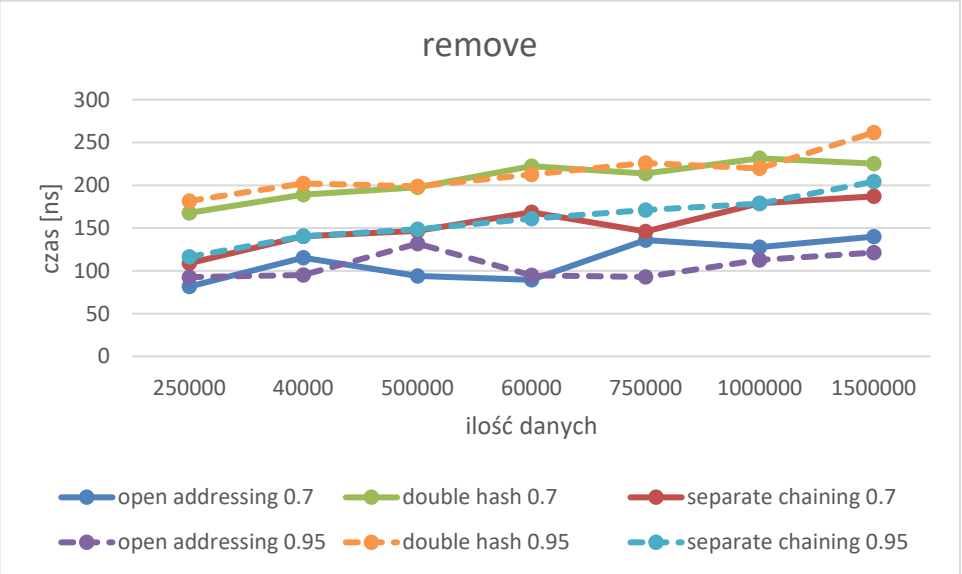
WYKRES 12: PORÓWNANIE ŚREDNICH CZASÓW FUNKCJI REMOVE TABLICY DOUBLE HASHING



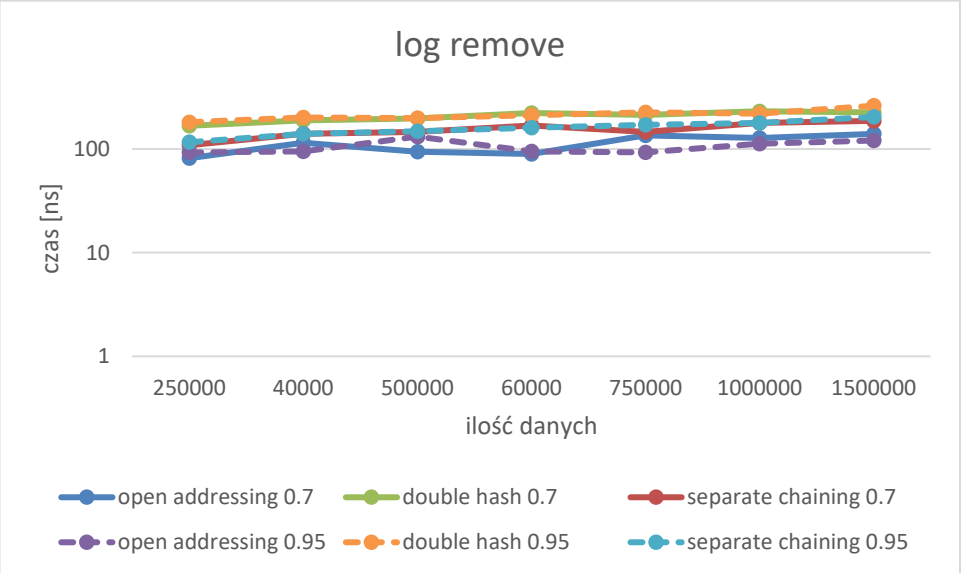
WYKRES 13: PORÓWNANIE ŚREDNICH CZASÓW FUNKCJI INSERT



WYKRES 14: PORÓWNANIE ŚREDNICH CZASÓW FUNKCJI INSERT LOGARYTM



WYKRES 15: PORÓWNANIE ŚREDNICH CZASÓW FUNKCJI REMOVE



WYKRES 16: PORÓWNANIE ŚREDNICH CZASÓW FUNKCJI REMOVE LOGARYTM

5) Wnioski:

Na wykresach funkcji insert można zauważyć 3 skoki pasujące do najgorszych wcześniej wyliczonych złożoności obliczeniowych. One mogą być wynikiem grupowań wtórnych, lecz większym prawdopodobieństwem tych nieprawidłowości jest przypadkowe wykonanie rozszerzenia tablic wymagających całkowitego nowego hashowania elementów. To stwierdzenie można potwierdzić tym, że w tych samych momentach skoków czasu dodawania elementów funkcja usuwania wykonuje się w odpowiednim czasie co było by mało prawdopodobne podczas istnienia grupowań wtórnych. Kolejnym potwierdzenie jest fakt że skoki powstały na tablicach bazujących na metodzie łańcuchowej dla, której grupowania są nie szkodliwe.

Za pomocą nieregularności można potwierdzić złożoności obliczeniowe w najgorszych przypadkach.

Za pomocą wykresu 15 można zauważyć że double hash jest wolniejszy od separate Chaining przez wykorzystanie 2 funkcji mieszającej, która przez wykonywanie badań na niskim współczynniku zajętości jedyne pogarsza działanie tablicy za miast przyspieszać.

Za pomocą wykresu 7 można zauważyć że nawet podczas wysokiego współczynnika zajętości (0.95) czasy wykonywania są bardzo podobne dzięki wykorzystaniu jeden z rodzajów próbkowania kwadratowego podczas rozwiązywania kolizji elementów.