Siódme laboratorium

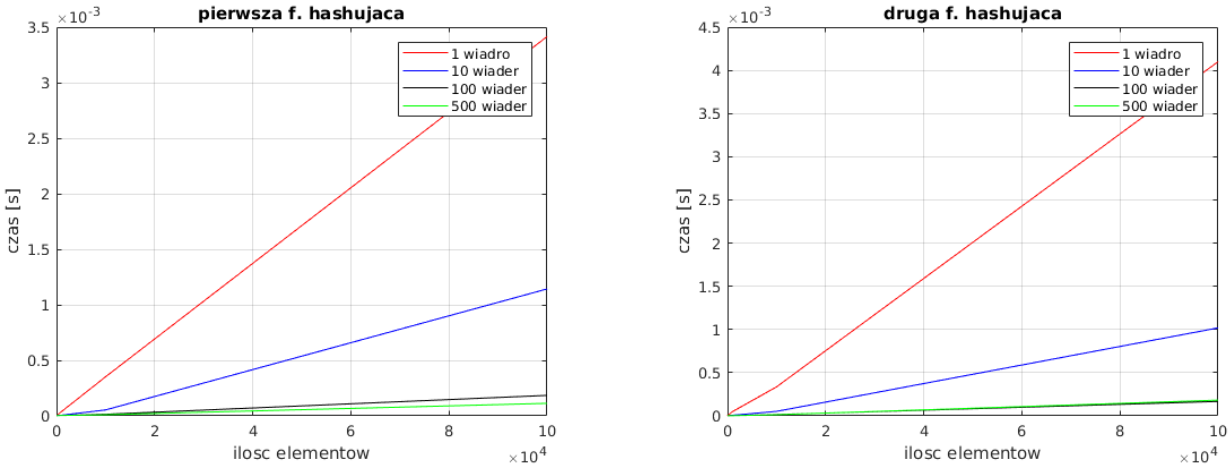
Aleksandra Nycz 226270

26.04.2017

1 Wyniki

Pomiary przeprowadzono dla dwóch różnych funkcji hashujących – jedna polegała na dodaniu kodów ASCII kolejnych liter klucza i obliczenia z nich modulo liczby "wiader", druga na hashowaniu przez mnożenie, została wybrana na podstawie literatury[1]. Drugim parametrem była różna ilość wiader. Wyniki przedstawia poniższa tabela. Pomiary zostały wykonane dla 10 powtórzeń dla ilości danych do stu tysięcy, ponieważ później czas zapisu był zbyt długi. Klucze były generowane losowo jako wyrazy pięcioliterowe. Wyszukiwany był element o ostatnim generowanym kluczu, co pozwalało na losowy rozkład umiejscowienia elementu.

	ilość wiader							
	pierwsza f. hash.				druga f. hash.			
ilość	1	10	100	500	1	10	100	500
10^1	0.0000011	0.0000016	0.0000015	0.0000009	0.0000017	0.0000016	0.0000009	0.00000065
10^2	0.00000455	0.0000049	0.0000014	0.0000017	0.0000082	0.000002	0.0000021	0.0000031
10^3	0.0000375	0.0000055	0.0000038	0.0000038	0.0000485	0.0000065	0.0000023	0.000002
10^4	0.00035175	0.0000534	0.0000141	0.00001	0.0003358	0.000052	0.0000152	0.0000135
10^5	0.003415	0.0011428	0.0001844	0.0001128	0.0041	0.001018	0.000167	0.000182



Rysunek 1: Wykresy

2 Problemy

- Wskaźnik na następny element w liście musiał być polem publicznym, ponieważ pojawił się probelm z funkcją *readPointer* – nie zwracała wskaźnika.
- Pojawił sie problem z funkcją *doAlgorithm*, więc do odczytu elementu podczas testowania algorytmu odwołano się bezpośrednio.

[1] "Wprowadzenie do algorytmów" Cormen, rozdz. 11.3

- Implementacja nie zapobiega duplikowaniu się elementów z tym samym kluczem.

3 Wnioski

- Na podstawie danych widać, że im większa liczba wiader tym szybciej algorytm działa.
- Dla ilości wiader w okolicach $100 - 500$ różnica w czasie jest niewielka lub prawie rzadna.
- Pierwsza funkcja hashująca wypadła lepiej niż druga. Według literatury są to funkcje porównywalne.
- Złożoność odczytu elementu wynosi ok. $O(n)$.