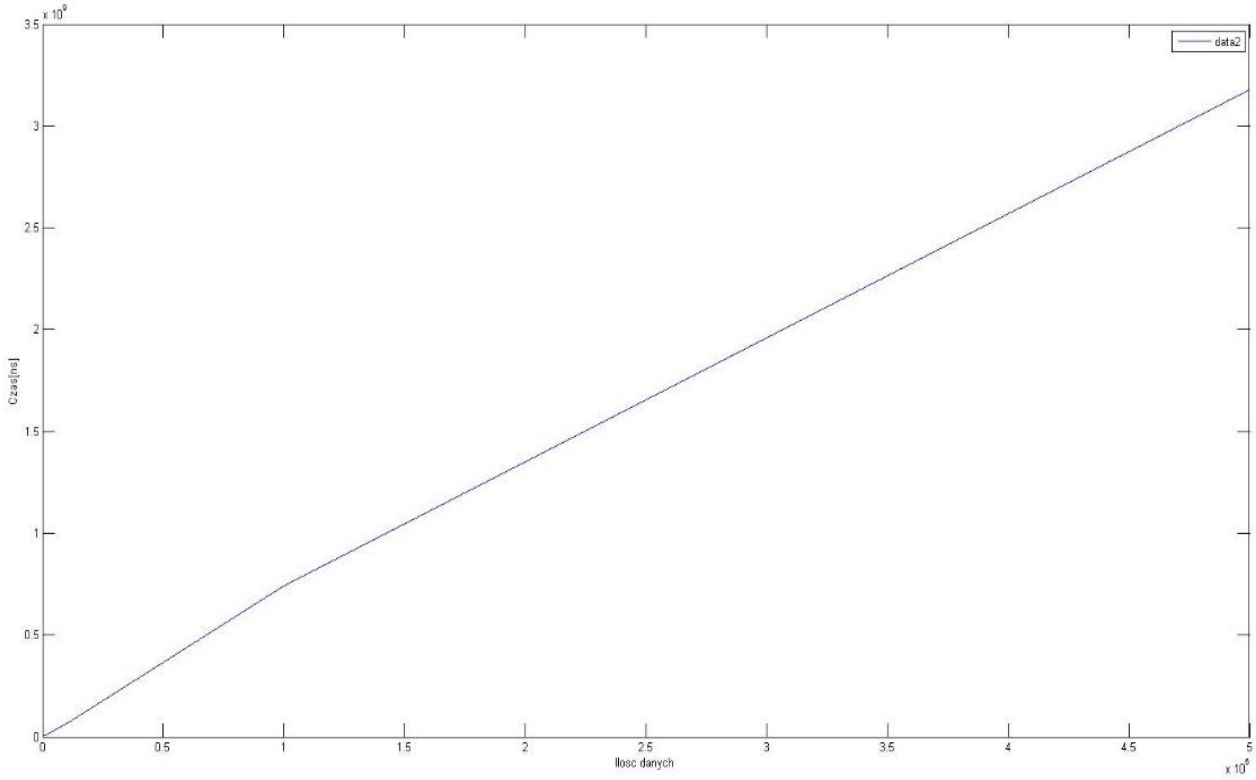
Sprawozdanie z laboratorium nr 5

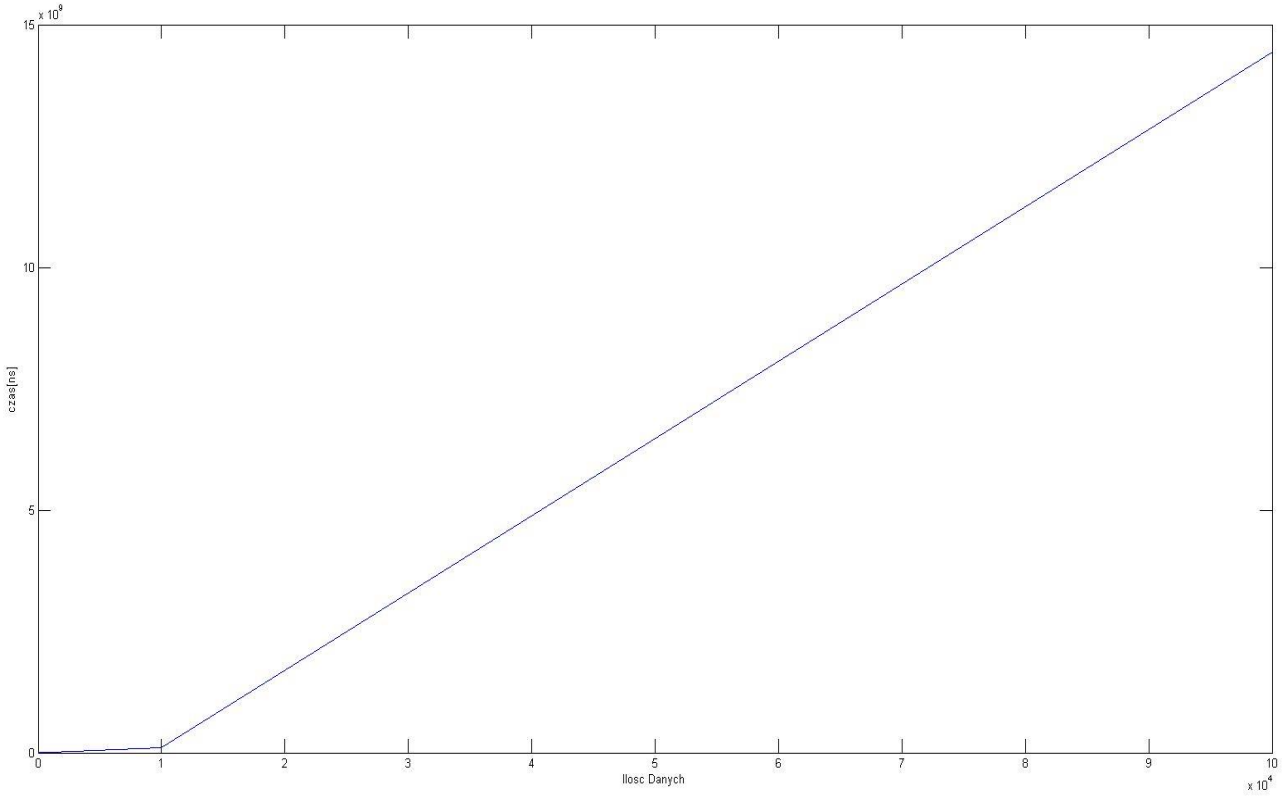
„Haszowanie i wykrywanie kolizji”



Powyżej znajduje się wykres złożoności obliczeniowej mojej funkcji haszującej. Złożoność obliczeniowa dla n operacji wynosi O(n) czyli wstawienie pojedynczego elementu wynosi O(I). Problem haszowania został rozwiązany za pomocą porcjonowania. Poniżej znajdują się dane liczbowe w (ns) potwierdzające złożoność pojedynczej operacji.

1.0e+03 *[6.2069; 1.1448 ;0.6142 ;0.5718 ;0.6292;0.7395; 0.6355] ns.

W moim programie wykorzystałem tablice dynamiczną składającą się z list<string>. Kolizyjność w moim programie jest rozwiązana, za pomocą list tzn. element posiadający tą samą wartość haszująca(indeks tablicy) co już inny istniejący, zostaje wstawiony jako kolejny element listy.



Powyżej znajduje się wykres złożoności obliczeniowej mojej funkcji wyszukującej element w tablicy haszowanej. Jak można zauważyć następuje wzrost nakładu czasowego na pojedynczą operację wyszukiwania, wynika to z „porcjonowania” danych tzn. listy na tablicy dynamicznej robią się dłuższe co spowalnia znacznie proces wyszukiwania. Algorytm ma złożoność O(m), gdzie m jest rozmiarem porcji na listach.

Inne rozwiązania problemu kolizyjności

a) Próbkowanie liniowe – polega na wstawianiu stringa na pozycji wyliczonej przez funkcje haszującą i jeżeli miejsce w tablicy jest zajęte, to przesuwamy się na najbliższy wolny indeks tablicy. Wyszukiwanie działa analogicznie. Złożoność operacji wstawiania i wyszukiwania w tym algorytmie w najgorszym wypadku wynosi O(n), gdy wszystkie komórki są już zajęte, a gdy nie ma kolizji O(I).
b) Haszowanie kukułcze – polega na zastosowaniu dwóch tablic i odpowiadających im funkcji haszujących. Elementy są dodawane do pierwszej tablicy według pierwszej funkcji haszującej dopóki nie nastąpi kolizja. Jeżeli nastąpi, to wyliczany jest indeks w drugiej tablicy przez drugą funkcję haszującą.