Sprawozdanie z laboratorium nr 7

226543
Tomasz Kaliciak

24 maja 2017

1 Wstęp

Celem ćwiczenia było zaimpementowanie tablicy z haszowaniem. Dla dwóch wybranych funkcji skrótu należało porównać złożność obliczeniową odczytu dla róznych ilości bucketów.
Tablica z haszowaniem przechowuje pary (klucz, wartość) i umożliwia szybki dostęp do wartości poprzez podanie klucza. Struktura danych charakteryzuje się małą złożonością obliczeniową w przypadku operacji przeszukiwania, wstawiania i usuwania elementów (oczekiwana złożoność tych operacji to $O(1)$, a w najgorszym przypadku $O(n)$).

2 Wybrane funkcje skrótu

Pierwsza funkcja skrótu zamienia dwa pierwsze znaki klucza na kod ASCII i sumuje ich wartości. Funkcja zwaraca resztę z dzielenia obliczonej sumy przez liczbę bucketów.
Druga to funkcja haszująca stworzona przez Glenna Fowler, Landona Curta Noll oraz Kiema-Phong Vo. Inicjujemy zmienną hash wartością FNV offset basis. Dla każdego bajtu klucza mnożymy zmienną hash przez stałą nazywaną FNV prime, następnie wykonujemy operację XOR z danym bajtem napisu. Funkcja zwaraca resztę z dzielenia wartości zmiennej hash przez liczbę bucketów. Pseudo kod wygląda następująco:

```
hash = FNV_offset_basis
for each byte_of_data to be hashed
     hash = hash x FNV_prime
     hash = hash XOR byte_of_data
return hash % numberOfBuckets
```

Jest to bardzo szybka i prosta funkcja haszująca, która generuje mało kolizji.

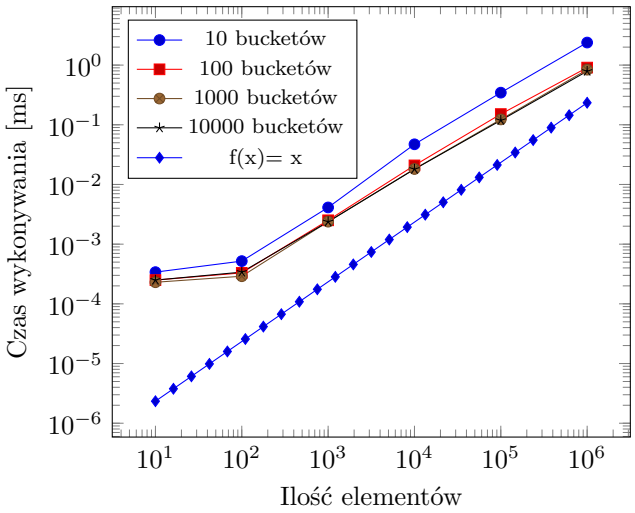
3 Pomiary

Badając złożoność obliczeniową przeszukiwania tablicy wykorzystano plik names.csv, który zawiera wygenerowane pary login-hasło. Dodatkowo na potrzeby pomiarów stworzono nową klasę Tester. Podczas dodawania elementów z pliku wybierany jest losowy klucz. Po ukończeniu tego procesu rozpoczynają się pomiary złożoności obliczeniowej, mierzony jest czas wyszukiwania wybranego wcześniej losowego elementu. Dla każdej konfiguracji liczba elementów - liczba bucketów wykonano pomiary 200 razy.

3.1 Funkcja haszująca nr 1

Tabela 1: Zestawienie czasów przeszukiwania tablicy

	Czas [ms]			
Ilość elementów	10 bucketów	100 bucketów	1000 bucketów	10000 bucketów
10^1	0.00034	0.00025	0.00023	0.00025
10^2	0.00052	0.00033	0.00029	0.00034
10^3	0.00412	0.00251	0.00238	0.00237
10^4	0.0471	0.02087	0.01815	0.01804
10^5	0.3452	0.15106	0.12295	0.11865
10^6	2.38602	0.90776	0.8374	0.7899



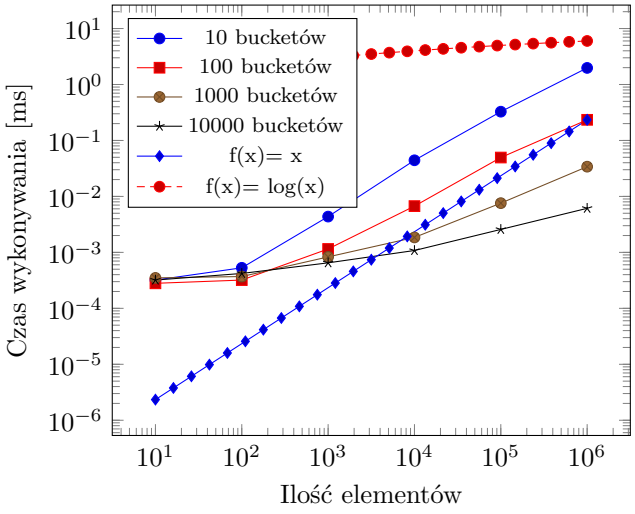
Wykres 1: Czas wykonywania od ilości elementów

W przypadku 10-100 elementów złożoności obliczeniowe wynoszą w przybliżeniu $O(1)$, jednak po przekroczeniu pewnej ilości elementów zaczynają przypominać $O(n)$.

3.2 Funkcja haszująca nr 2

Tabela 2: Zestawienie czasów przeszukiwania tablicy

	Czas [ms]			
Ilość elementów	10 bucketów	100 bucketów	1000 bucketów	10000 bucketów
e 10^1	0.00032	0.00028	0.00035	0.00032
10^2	0.00053	0.00032	0.00037	0.00042
10^3	0.00436	0.00115	0.00083	0.00065
10^4	0.04423	0.00671	0.00185	0.00108
10^5	0.32668	0.04952	0.00758	0.00257
10^6	1.979	0.233825	0.034	0.006125



Wykres 2: Czas wykonywania od ilości elementów

Podobnie jak w wcześniej przy małych ilościach danych złożności obliczeniowe wydają się być stałe, przy ponad 1000 elementów zaczynają rosnąć. Złożoność przy 10 bucketach to $O(n)$, a w pozostałych przypadkach jest mniejsza, ale nie mniejsze niż $O(log(n))$.

4 Wnioski

- Stałą złożoność obliczeniową udało się uzyskać tylko dla niewielkich ilości danych i dużej ilości bucketów. W pozostałych przypadków złożoność obliczeniowa przypomina bardziej $O(n)$
- Powodem nieuzyskania stałej złożoności obliczeniowej może być, sposób wybierania wyszukiwanego elementu. Przy 200 powtórzeniach pomiarów uzyskanie kilku pesymistycznych przypadków znacznie wydłużało czas pracy programu, co doprowadza do zwiększenia średniego czasu. Gdyby przeprowadzono znacznie więcej pomiarów to badanie byłoby bardziej miarodajne
- Można zaobserować, że zwiększenie liczby bucketów prowadzi do zmniejszenia złożoności obliczeniowej, ponieważ występuje wtedy mniej kolizji
- Pierwsza funkcja skrótu okała się być gorsza. Opiera się za zbyt prostej operacji przez co jej wyniki nie są zbytnio zróżnicowane