Algorytmy i Struktury Danych II
ZESTAW 02

Autor: Marcin Wolski

(A) Funkcje hashujące

Funkcje hashujące implementujące przyporządkowanie dla zbiorów:

a) liczb całkowitych $\mathbf{n}, \mathbf{n+1}, \mathbf{n+2}, ..., \mathbf{m}$ gdzie n < m

```
int generateIndex (int number, int first) {
    return number - first;
}
```

b) liczb całkowitych $\mathbf{n}, \mathbf{n+2}, \mathbf{n+4}, ..., \mathbf{m}$ gdzie n < m

```
int generateIndexEven (int number, int first) {
    return (number - first)/2;
}
```

c) liter $\mathbf{a}, \mathbf{b}, \mathbf{c}, ..., \mathbf{z}$ bez polskich znaków ęóąśłżźć

```
int generateIndexLetter (char letter) {
    return letter - 'a';
}
```

W C++ zmienna znakowa przechowuje wartość ASCII. Na przykład wartość ASCII znaku ”a” wynosi 97, ”b” wynosi 98 itd. Wykorzystujemy ten fakt do przypisywania miejsca w tablicy hashującej.

d) dwuliterowych napisów, gdzie każda litera jest z zakresu $\mathbf{a} - \mathbf{z}$ bez polskich znaków ęóąśłżźć

```
int generateIndexLetterPair
                    (char letter1, char letter2) {
    return (letter1 - 'a')*26 + (letter2 - 'a');
}
```

(B) Typ danych setHashed

Typ danych setHashed, wykorzystujący haszowanie otwarte, reprezentuje matematyczny zbiór oraz operacje które dla dwóch zbiorów realizują:

- sumę zbiorów
- część wspólną zbiorów
- różnicę zbiorów
- sprawdzanie identyczności zbiorów

oraz dla elementu zbioru realizują:

- wstawanie elementu do zbioru
- usuwanie elementu ze zbioru
- sprawdzanie czy element należy do zbioru

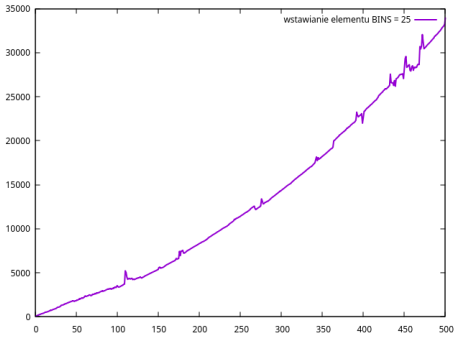
Badanie złożoności operacji

Czas wykonania operacji zmierzony został dla zbiorów o rozmiarach [1, 500]. Dla każdej wielkości obliczona została średnia z 1000 powtórzeń działania. Wygenerowane zostały pliki z danymi które zwizualizowane zostały za pomocą programu gnuplot.

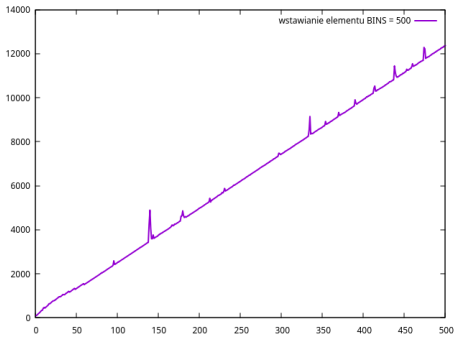
Wykresy

Pionowa oś - czas w nanosekundach
Pozioma oś - rozmiar problemu

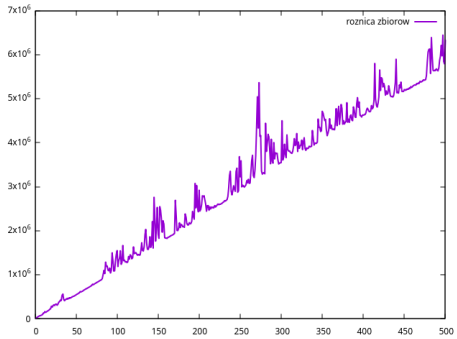
Wstawianie elementu, mala ilosc binów (25)



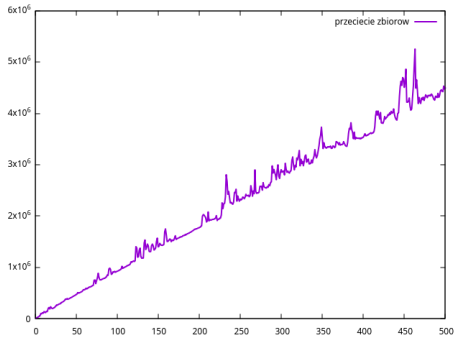
Wstawianie elementu, duza ilosc binów (500)



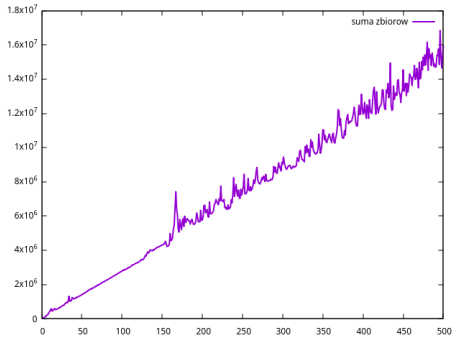
Różnica dwóch zbiorów



Przecięcie dwóch zbiorów



Suma dwóch zbiorów



(C) Porównanie implementacji

Porównanie implementacji zbioru z zadań A i B z zestawu 1 i zbioru haszującego z zadania B z obecnego zestawu.

Wartości zbioru z zadania A z pierwszego zestawu nie są związane z konkretnymi kluczami. Są poindeksowane sekwencyjnie w tablicy. Randomowy dostęp do elementu wynosi O(1), a do konkretnego elementu (musi przeszukać tablice po kolei) wynosi O(n).

W zbiorze z zadanie A z pierwszego zestawu wartości również nie są powiązane z kluczami. Dostęp jest sekwencyjny (przeszukiwanie od pierwszego węzła).

Zbiór hashujący, dzięki cechom słownika(tablicy asocjacyjnej), pozwala na szybkie wstawianie, wyszukiwanie i usuwanie. Przy każdej operacji odwołujemy się do indeksu zwróconego przez funkcję haszującą. Zbiór haszujący zaimplementowany został na bazie tablicy list wiązanych.
Przy przeszukiwaniu, w najgorszym przypadku wszystkie klucze wskazują na jeden slot tablicy i struktura staje się porównywalna do listy wiązanej. To oznacza że operacje mają złożoność O(n).
Zakładając jednak, że klucze są rozmieszczone równomiernie, można oczekiwać złożoności wynoszącej O(1).

Której implementacji najlepiej użyć?

W przypadku gdy w naszym zbiorze dane posiadają charakterystyczny klucz który je definiuje, lepiej wykorzystać zbiór haszujący. Dostęp do konkretnego elementu będzie wynosił O(1) przy równomiernie rozłożonych wartościach.

Jeśli zależy nam bardziej na dostępie do randomowego elementu w zbiorze, lepiej wykorzystać setSimple, złożoność O(1).

Jeśli randomowy dostęp nie jest istotny i wartości nie są powiązanie z kluczami,

a zależy nam na dynamicznym zbiorze (łatwe usuwanie, dodawanie), możemy wykorzystać setLinked.