Sprawozdanie
Projektowanie aglorytmów i metod sztucznej inteligencji
Laboratorium 6

Michał Wieczorek
226284

17 maja 2017

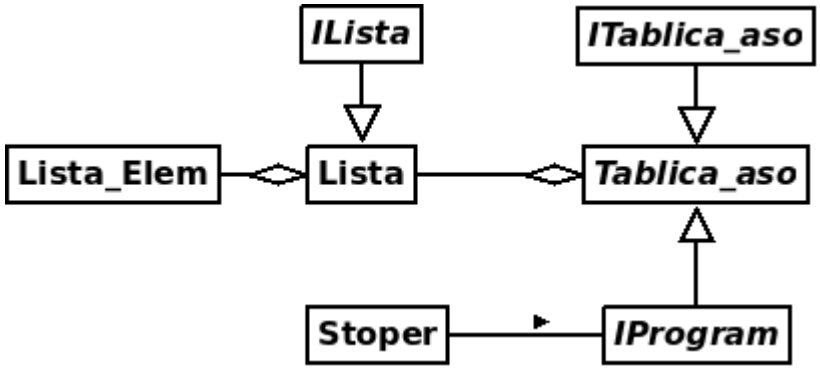
1 Cel i przebieg laboratorium

Omówienie działania tablicy asocjacyjnej. Dyskusja na temat algorytmów haszujących. Jak dobór funkcji haszującej wpływa na szybkość wyszukiwania elementu w tablicy asocjacyjnej. Złożoność obliczeniowa funkcji wyszukującej.
Implementacja prototypów funkcji tablicy asocjacyjnej. Budowa szkieletu programu.

2 Struktura i działanie algorytmu

2.1 Struktura kodu

W dotychczasowym programie pojawiła się nowa klasa - *Tablica-aso*, dziedziczy ona bezpośrednio z interfejsu *ITablica-aso* oraz *IProgram*. Ponieważ każdy element tablicy asocjacyjnej jest listą, to również i z *Lista*.



2.2 Tablica asocjacyjna

Interfejs tablicy asocjacyjnej składa się z 3 podstawowych metod:

- Dodawanie elementu o podanym kluczu i wartości
- Usunięcie elementu o podanym kluczu
- Wyszukanie elementu o podanym kluczu i zwrócenie jego wartości

Tak jak wspomniano wcześniej Tablica asocjacyjna składa się ze zwykłego arraya i listy. W każdej komórce tablicy znajduje się element listy. Z kolei element listy składa się z klucza, wartości i oczywiście wskaźnika na kolejny element.

2.3 Funkcje haszujące

Idealnym algorytmem haszowania jest funkcja, która równomiernie rozkłada elementy w tablicy bez zostawiania wolnych miejsc, czy przedłużania list. Złożość obliczeniowa wynosi wtedy 0(1). Jeżeli jednak zaistnieją dwa różne klucze (zjawisko kolizji), ale o tych samych haszach, wtedy lista zostaje nadpisana i złożoność wyszukania tego elementu wzrasta do $O(n)$.

Zaimplementowano 2 funkcje haszujące:

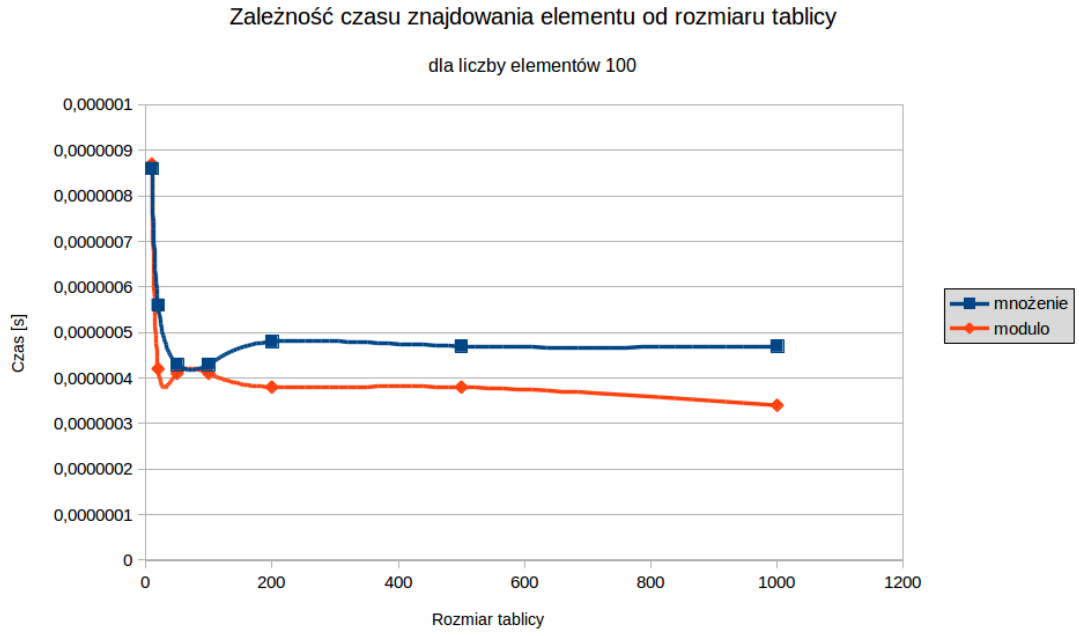
- Haszowanie modularne - polega na sumowaniu znaków ASCII z których składa się klucz, następnie z wyniku brane jest modulo o parametrze rozmiaru tablicy, $h(k) = kmod(m)$
- Haszowanie przez mnożenie - bardziej skomplikowane, dobieramy pewną stałą A (ja przyjąłem zaproponowaną przez książkę Cormena: $A = (\sqrt{5} - 1)/2)$ z przedziału $0 < A < 1$ a następnie mnożymy przez sume znaków ASCII klucza i wyznaczamy część ułamkową tej liczby. Ostatnim krokiem jest przemnożenie liczby przez rozmiar tablicy i wyciągniecię części całkowitej z całości, $h(k) = \lfloor m(kAmod(1) \rfloor$

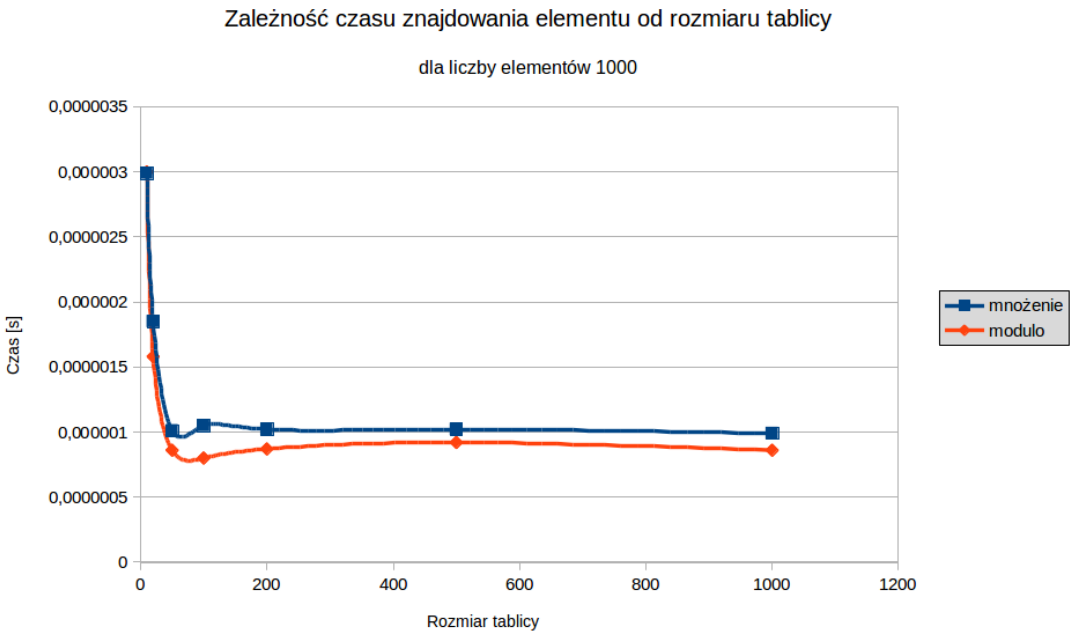
3 Testy programu

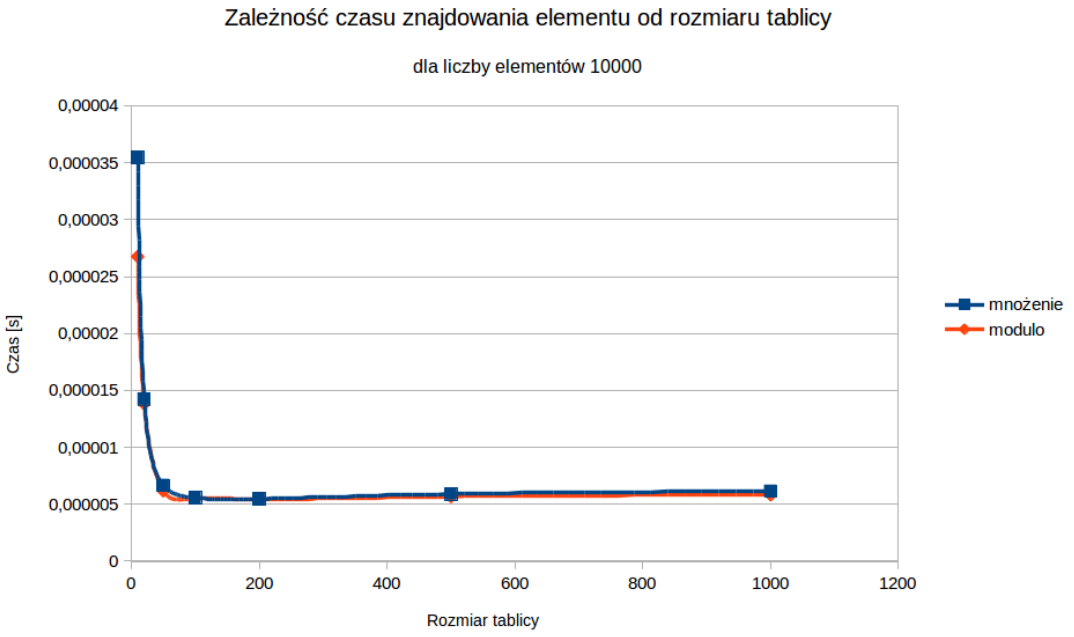
Mierzenie czasu wyszukiwania elementu nie jest prostym zadaniem. Problemem jest to, że każda lista w tablicy haszującej ma przeważnie inną długość. W poniższych testach wyraz do wyszukania jest losowany przed wypełnieniem całej tablicy, a następnie umieszczany na odpowiednim polu (poprzez algorytm haszujący). Następnie dodawane są pozostałe wyrazy, dzięki temu zabiegowi, nasz element będzie zawsze umieszczony na końcu, którejś listy.
Aby wyniki były w miarę wiarygodne, program liczył średnią czasów 1000 pomiarów. Niestety w przypadku tych krótszych czasów dokładność pomiaru stoperem nie jest wystarczająca, dlatego należy pominąć nieznaczne zmiany wartości.

3.1 Ze zmianą rozmiaru tablicy

W pierwszej części testów mierzone są czasy wyszukiwania dla stałej liczby elementów w tablicy, zmienia się jedynie rozmiar tablicy haszującej. Przyjąłem 3 warianty liczby elementów 10, 100, 1000.

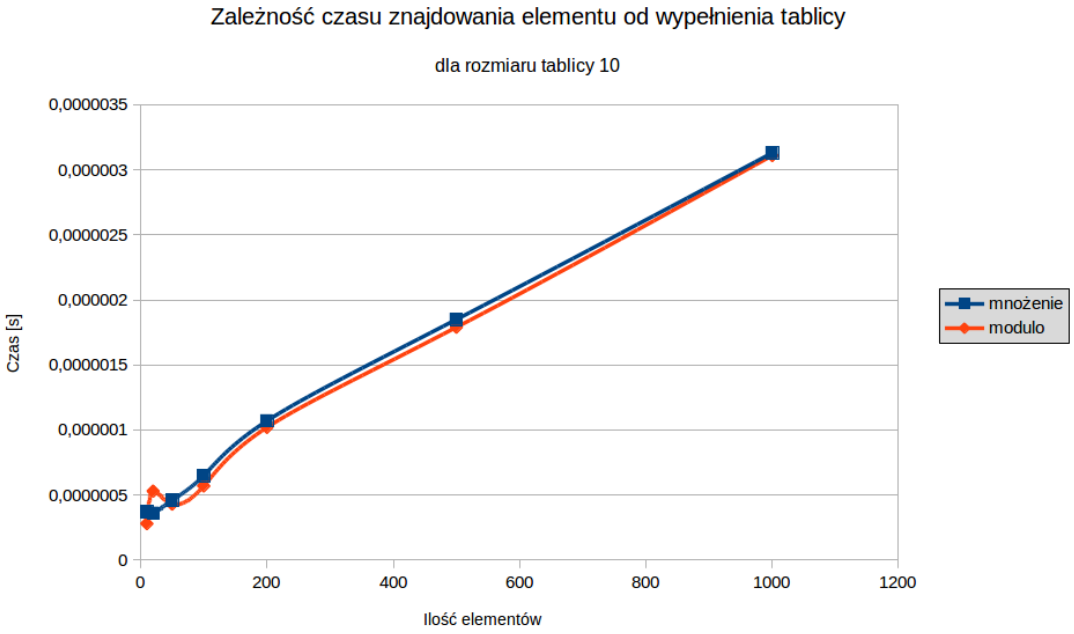


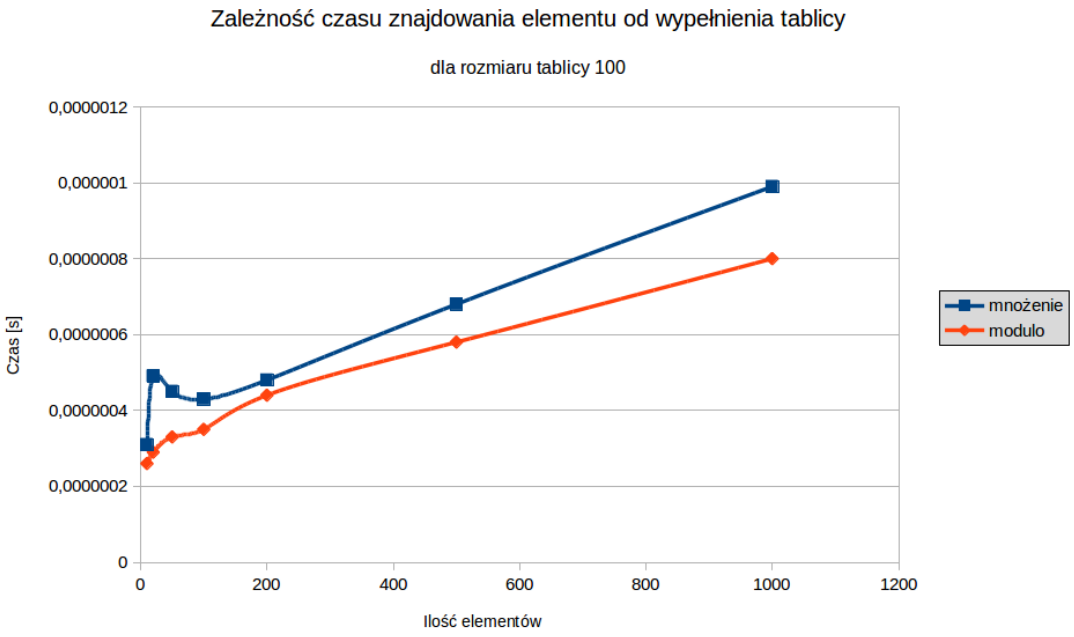

Zależność czasu znajdowania elementu od rozmiaru tablicy
dla liczby elementów 1000
Czas [s]
0,0000035
0,000003
0,0000025
0,000002
0,0000015
0,000001
0,0000005
0
0
200
400
600
800
1000
1200
Rozmiar tablicy
mnożenie
modulo

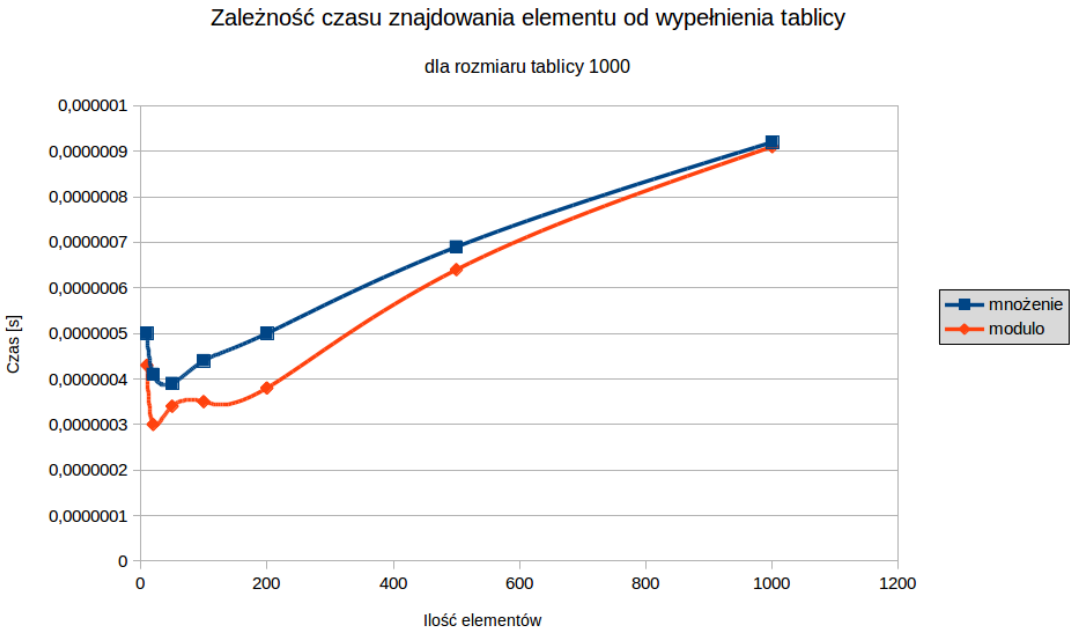

Zależność czasu znajdowania elementu od rozmiaru tablicy
dla liczby elementów 10000
Czas [s]
0,00004
0,000035
0,00003
0,000025
0,00002
0,000015
0,00001
0,000005
0
0
200
400
600
800
1000
1200
Rozmiar tablicy
mnożenie
modulo

3.2 Ze zmianą ilości elementów w tablicy

W drugiej części testów mierzone są czasy wyszukiwania dla stałego rozmiaru tablicy, zmienia się ilość elementów. Również tutaj są 3 warianty: 10, 100, 1000.




Zależność czasu znajdowania elementu od wypełnienia tablicy
dla rozmiaru tablicy 100
0,0000012
0,000001
0,0000008
0,0000006
0,0000004
0,0000002
0
Czas [s]
0
200
400
600
800
1000
1200
Ilość elementów
mnożenie
modulo


Zależność czasu znajdowania elementu od wypełnienia tablicy
dla rozmiaru tablicy 1000
0,000001
0,0000009
0,0000008
0,0000007
0,0000006
0,0000005
0,0000004
0,0000003
0,0000002
0,0000001
0
Czas [s]
0
200
400
600
800
1000
1200
Ilość elementów
mnożenie
modulo

4 Wnioski

Ponieważ wykresy różniły się znacząco od siebie, opiszę tylko te najbardziej wiarygodne. Dla rosnącego rozmiaru tablicy haszującej weźmy wykres 3 - dla 10000 elementów. Ponieważ elementów jest dużo więcej od ilości miejsca w tablicach, są one przerzucane na koniec listy. Z wykresu możemy więc zauważyć iż algorytm haszowania modulo sprawdza się minimalnie lepiej na całym przedziale. Przy rozmiarze tablicy 100 czas wyszukiwania stabilizuje się, możemy więc przyjąć, że od tego momentu złożoność wynosi $O(1)$. Dla mniejszych rozmiarów tablicy czas wyszukiwania wydłuża się znacząco.
Co do rosnącej liczby elementów w tablicy, do opisu wybrałem wykres pierwszy. Bardzo ładnie widać tu złożoność $O(n)$ algorytmu wyszukiwania. Haszowanie modularne sprawdza się nieco lepiej. Przy mniejszej liczbie elementów są małe zakłócenia, jednak jest to spowodowane szybkim odczytem danych.

Podsumowując, tablice haszujące z listami nadają się idealnie do przechowywania par danych i szybkiego odczytu ich wartości. Odpowiednie dobranie algorytmu haszującego zapewni jeszcze lepszą płynność działania tej struktury. Oprócz mierzenia czasów, porównałem ułożenie danych w tablicy po haszowaniu modularnym i przez mnożenie. Okazało się, że dane po haszowaniu modularnym skupiały się przeważnie w jednym miejscu, pozostawiając większą część tablicy pustą. Funkcja haszowania przez mnożenie rozkładała elementy bardziej równomiernie, nadal były puste miejsca w tablicy, jednak elementy były w niej bardziej porozrzucane.