Praca domowa 02 – hashing

Termin zwrotu : 27 października godz 23.00
Zadanie uznaje się za zaliczone, gdy praca oceniona zostanie na co najmniej 6 pkt.

Należy zaproponować rozwiązanie przechowujące słowa w tablicy haszującej. Można użyć obiektów typu String. Można użyć dowolnej funkcji skrótu dla wyznaczania lokalizacji słowa w tablicy. Rozmiar tablicy haszującej winien być podawany w momencie jej tworzenia (konstrukcji – patrz dalej, parametr `<hash_size>` programu). W przypadku wystąpienia kolizji należy użyć techniki łańcuchowania (chaining).

Klasa hash musi udostępniać trzy metody :

```
void insert (String word)
boolean find(String Word)
void remove (String word)
```

Dodatkowo (poza klasą realizującą obsługę tablicy haszującej) należy napisać kod umożliwiający wczytanie zbioru słów z pliku wejściowego (wskazywanego przez parametr `<input_file>` programu) zawierającego w kolejnych liniach pojedyncze słowa, i wstawienie tych słów do utworzonej tablicy. Słowa zawierają wyłącznie duże litery alfabetu angielskiego (bez znaków narodowych). Porządek słów w tablicy nie jest istotny. Program winien wyznaczyć i wyprowadzić informację o ilości kolizji, które wystąpiły podczas wstawiania danych wejściowych do tablicy haszującej.

Sposób obliczania ilości kolizji : w trakcie wstawiania do tablicy z użyciem metody `insert(String word)` jeżeli w wyznaczonej pozycji tablicy haszującej istnieje już wpis, należy licznik kolizji zwiększyć o jeden w każdym momencie, w którym określona pozycja łańcucha zapisów tablicy haszującej porównywana jest ze słowem wejściowym. W poniższym przykładzie pewna pozycja tablicy haszującej zawiera już listę trzech słów :

```
ONE -> TWO -> THREE
```

`insert ("TWO")` zwiększy licznik kolizji o jeden (porównanie ze słowem `"ONE"` zakończy się wynikiem negatywnym), `insert ("THREE")` zwiększy licznik kolizji o dwa, a `insert ("FOUR")` zwiększy licznik kolizji o trzy.

Sam sposób doboru funkcji skrótu nie podlega ocenie, lecz rozwiązanie, które zapewni możliwość przetworzenia największego testowego pliku wejściowego z najmniejszą średnią ilością kolizji (obliczoną dla różnych wartości rozmiaru tablicy haszującej) będzie dodatkowo premiowane.

Program ma być zapisany wyłącznie w dwóch plikach `Hash.java` – zawierającym implementację mechanizmu haszowania, oraz `Hashing.java` – zawierającym program główny. Program nie może korzystać z jakichkolwiek bibliotek zewnętrznych.

Proces kompilacji musi być możliwy z użyciem komendy

```
javac –Xlint Hashing.java Hash.java
```

Uruchomienie programu winno być możliwe z użyciem komendy

```
java hashing <input_file> <hash_size>
```

Przykładowy plik wejściowy :

```
A
B
D
D
F
E
D
G
```

Przykładowy wynik końcowy :

```
Ilość kolizji : 4
```

Wymagania :

- Klasa implementująca mechanizm haszowania (wraz z ze strukturą danych) winna zostać zdefiniowana w pliku `Hash.java`
- Program główny (metoda main) winien być zdefiniowany w pliku `Hashing.java`
- W pliku README winien być zawarty szczegółowy opis przyjętej (zaimplementowanej) funkcji skrótu (metody haszowania), uzasadnienie wyboru wykorzystanej funkcji, oraz szczegółowy opis przyjętej metody łańcuchowania (obsługi konfliktów).

Sposób oceny **:**

- 1 pkt – **Kompilacja** : każdy z plików winien być kompilowany bez jakichkolwiek błędów lub ostrzeżeń (w sposób omówiony wyżej)
- 1 pkt – **Wykonanie** : program powinien wykonywać się bez jakichkolwiek błędów i ostrzeżeń (dla pliku danych wejściowych zgodnych z wyżej zamieszczoną specyfikacją) z wykorzystaniem omówionych wyżej parametrów linii komend
- 2 pkt – **README** : plik README.pdf dokumentuje w sposób kompletny i właściwy struktury danych, funkcję haszującą oraz mechanizm rozwiązywania konfliktów.
- 1 pkt – **Komentarze wewnętrzne** : czy program jest skomentowany w sposób zapewniający zrozumienie jego działania, oraz wyjaśniający warunki, które muszą zachodzić przed i po wykonaniu każdej z funkcji.
- 1 pkt – **Styl kodowania** : czy funkcji i zmienne posiadają samo-wyjaśniające nazwy ? Czy podział na funkcje ułatwia czytelność i zrozumiałość kodu ? Czy funkcje eliminują (redukują) powtarzające się bloki kodu ? Czy wcięcia, odstępy, wykorzystanie nawiasów itp. (formatowanie kodu) są spójne i sensowne ?
- 4 pkt – **Poprawność algorytmu** : czy funkcje `insert`, `find` oraz `remove` zostały zaimplementowane poprawnie , przy czym za funkcję `insert` można otrzymać dwa punkty, a za dwie pozostałe łącznie dwa.