Übung 06: Hashing

Abgabetermin: 15.06.2010 10:15

Name: _____ **Matrikelnummer**: _____

Gruppe: ❒ G1 (Wolfinger) ❒ G2 (Wolfinger)

Aufgabe	Punkte	gelöst	abzugeben schriftlich	abzugeben elektronisch	Korr.	Pkte
Aufgabe 06.1	24	❑	Java-Programm Testfälle und Ergebnisse	Java-Programm	❑	

Aufgabe 06.1: Stichwortverzeichnis als Hashtabelle (24 Punkte)

a) Implementieren Sie das Stichwortverzeichnis aus Übung 2 und 3 als Hashtabelle. Ersetzen Sie dazu die Ringliste bzw. den binären Suchbaum durch eine Hashtabelle. Um die Absatznummern (*Paragraphs*) in WordData zu speichern, können Sie die Ringeliste aus Übung 1 weiterverwenden.

Die Schnittstellen *WordData* und *WordIndex* bleiben unverändert:

```
interface WordData {
  String getWord();
  int getFrequency();
  int[] getParagraphs();
}
```

- *String getWord()* liefert das Wort
- *int getFrequency()* liefert die Häufigkeit für ein Wort
- *int[] getParagraphs()* liefert ein Array mit den Nummern der Absätze in denen das Wort vorkommt

Implementieren Sie die Klasse *WordIndexHashMap* mit folgender Schnittstelle:

```
interface WordIndex extends Iterable<WordData> {
  void insert(String word, int paragraph);
  int getFrequency(String word);
  int getDifferentWordCount();
  int getTotalWordCount();
  double getMeanWordFrequency();
  String[] getWordsWithMinFrequency(int frequency);
  String[] getWordsStartingWith(String prefix);
  Iterator<WordData> iterator();
}
```

- *void insert(String word, int paragraph)* fügt ein neues Wort ein
- *int getFrequency(String word)* ermittelt die Häufigkeit für ein Wort
- *int getDifferentWordCount()* liefert die Anzahl der verschiedenen Wörter
- *void getTotalWordCount()* liefert die Summe der Häufigkeiten aller Wörter
- *double getMeanWordFrequency()* liefert die durchschnittliche Häufigkeit aller Wörter
- *String[] getWordsWithMinFrequency(int frequency)* liefert alle Wörter mit einer gegebenen Mindesthäufigkeit
- *String[] getWordsStartingWith(String prefix)* liefert alle Wörter die mit einem gegebenen Prefix beginnen.
- *Iterator<WordData> iterator()* liefert einen Iterator für alle verschiedenen Wörter. Der Iterator soll die Wörter in genau der Reihenfolge liefern, in der die Wörter in der Hashtabelle gespeichert sind. Der Iterator soll nicht sortieren.

Implementieren Sie die Hashtabelle in drei Varianten mit jeweils verschiedener Kollisionsstrategie: a) mit linearem Probieren, b) mit quadratischem Hashen, und c) mit *Separate Chaining*. Ein Parameter im Konstruktur soll die Kollisionsstrategie einstellen:

```
class WordIndexHashMap implements WordIndex {
  public enum CollisionResolution {
    LINEAR_PROBING, QUADRATIC_PROBING, SEPARATE_CHAINING };
```

```
    public WordIndexHashMap(CollisionResolution cr) { ... }
    ...
}
```

Implementierungshinweis: Wählen Sie die initiale Größe der Hashtabelle so, dass *tabSize = 4 * j + 3*. Beginnen Sie mit *j = 4*, d.h. mit einer *tabSize = 19*. Führen Sie ein *rehash* durch, sobald die Hashtabelle zu 50% gefüllt ist. Vergrößern Sie dabei die Hashtabelle auf eine neue *tabSize* die sich durch ein Verdoppeln von *j* ergibt.

b) Testen Sie die Funktion Ihrer Lösung mit Hilfe des Programms *TestWordIndex.java* aus der Vorgabedatei. Ihre Lösung soll die gleiche Ausgabe erzeugen wie in der Vorgabedatei *Uebung6.Testausgabe.txt*.

c) Testen Sie die Effizienz Ihrer Lösung mit Hilfe des Programms *TestEfficiency.java* aus der Vorgabedatei. Fassen Sie die Ergebnisse in einer Tabelle zusammen. Diskutieren Sie schriftlich die Gründe für die Performanzunterschiede zwischen den drei Varianten der Hashtabelle, sowie im Vergleich zum binärem Suchbaum aus Übung 3. Gehen Sie dabei unter anderem auf folgende Fragen ein:

- Welche Methoden arbeiten in der Hashtabelle schneller als im Binärbaum? Warum?
- Welche Methoden arbeiten in der Hashtabelle gleich schnell oder langsamer als im Binärbaum? Warum?

Abzugeben ist:

- Das Java-Programm
- Die Testausgabe von *TestWordIndex.java*
- Die schriftlichen Ausführungen zur Effizienz