Übung 03: Binärbaum / Rot-Schwarz-Baum

Abgabetermin: 20.04.2010 10:15

Name: _____ **Matrikelnummer**: _____

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

Aufgabe	Punkte	gelöst	abzugeben schriftlich	abzugeben elektronisch	Korr.	Pkte
Aufgabe 03.1	19	❑	Java-Programm Testfälle und Ergebnisse	Java-Programm	❑	
Aufgabe 03.2	5	❑	Rot-Schwarz-Bäume	-	❑	

Aufgabe 03.1: Stichwortverzeichnis als binärer Suchbaum (19 Punkte, Teilaufgaben: a=15, b=4)

a) Implementieren Sie das Stichwortverzeichnis aus Übung 2 als binären Suchbaum. Ersetzen Sie dazu die sortierte Ringliste mit den Knoten *WordNode* durch einen binären Suchbaum. Für die Knoten mit den Absatznummern (*ParagraphNode*) können Sie weiterhin die Ringeliste verwenden.

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 *WordIndexTree* 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

b) Testen Sie die Effizienz Ihrer Lösung mit Hilfe des Programms *TestEfficiency.java* aus der Vorgabedatei. Diskutieren Sie schriftlich die Gründe für die Performanzunterschiede zwischen sortierter Ringliste und binärem Suchbaum, unter anderem:

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

Implementierungshinweise:

- Zur Implementierung des Iterators im Binärbaum dürfen Sie die Klasse *java.util.Stack* aus der Java-Bibliothek verwenden.
- Ihre Implementierung der Klassen *WordDataImpl*, *WordNode* und *WordIndexImpl* aus Übung 2 wird in dieser Übung zum Vergleich der Effizienz benötigt. Übernehmen Sie die Klassen aus Ihrer Ausarbeitung für Übung 2 und benennen Sie sie auf *WordDataList*, *WordNodeList* und *WordIndexLis*t um.

Abzugeben ist: Java-Programm, Testergebnisse

Aufgabe 03.2: Rot-Schwarz-Baum (5 Punkte)

Zeigen Sie mit allen Zwischenschritten die Rot-Schwarz-Bäume die beim Einfügen der Buchstaben DASNIBELUNGENLIED entstehen.