technische universität dortmund

Fakultät für Informatik
Lehrstühle 2, 11

Prof. Dr. C. Sohler, A. Krivošija
H. Blom, N. Kriege, H. Timm, B. Zey
http://tiny.cc/dap2p

Sommersemester 2014

DAP2 Praktikum – Blatt 10

Ausgabe: 6. Juni — Abgabe: 16.–20. Juni

Hinweis zu den Langaufgaben

Für diese Aufgaben stehen Ihnen ca. zwei Wochen Bearbeitungszeit zur Verfügung. In den Praktika dieser Woche können (und sollen) Sie mit der Bearbeitung beginnen. Den Rest müssen Sie außerhalb des Praktikums erledigen. Neben den Lernräumen können Sie dafür auch die Poolräume (z.B. in einer Freistunde) nutzen, wenn dort gerade keine Veranstaltungen stattfinden. Im Praktikum in zwei Wochen präsentieren Sie Ihrem Tutor dann Ihre Lösungen auf Ihrem Praktikumsrechner im Pool und bekommen Punkte dafür (nach Absprache mit dem Tutor ist es auch möglich, die Lösung auf Ihrem eigenen Rechner zu präsentieren).

Wichtig: Sie müssen die Aufgaben **vor** Beginn des Praktikums (in dem Sie die Lösungen präsentieren) fertig haben.

Ausserdem Wichtig: Der Quellcode ist natürlich (wie immer) mit sinnvollen Kommentaren zu versehen.

Langaufgabe 10.1 (4 Punkte)

Lernziel: Hashing – Hash-Tabelle

Laden Sie die Datei `vorlagenBlatt10.zip` von der Praktikumsseite runter und entpacken Sie sie in Ihrem Verzeichnis für Blatt 10.

Programmieren Sie eine Hashtabelle, die HashItems (wie unten definiert) aufnehmen kann. Kollisionen sollen dabei durch Listen aufgelöst werden. Ein Teil der Dateien ist dabei schon vorgegeben, unter anderem befinden sich fertige Hashfunktionen in der Datei `GeneralHashFunctionLibrary.java`. Sie alle bilden Strings auf ganzzahlige Hashwerte ab. Standardmäßig soll hier die Hash-Funktion `RSHash` benutzt werden. Achten Sie darauf, dass auch negative Rückgabewerte möglich sind, weshalb der Betrag verwendet werden sollte. Denken Sie außerdem daran, dass der zurückgegebene Wert immer durch den Modulo-Operator auf die Größe der Hashtabelle begrenzt werden muss.

Eine Hashtabelle ist hier ein Array von Listen, bei dem die Listen wiederum HashItems enthalten. Für die Listen können Sie die Java-Implementierung `LinkedList` verwenden. Diese Listen sollen HashItems aufnehmen, wobei ein HashItem aus dem Schlüssel `key` vom Typ `String` und dem Wert `info` vom Typ `int` besteht.

Schreiben Sie eine Klasse `HashItem`, die die oben beschriebenen Attribute, einen Konstruktor und notwendige get- und set-Methoden besitzt.

Implementieren Sie eine Klasse HashTable mit folgenden Methoden:

- Der Konstruktor bekommt die `hashSize` übergeben und erzeugt eine Hashtabelle dieser Größe. Er legt also ein Array an, sowie an jeder Stelle dieses Arrays eine Liste für die Kollisionen.
- Die Methode `put(String key)` speichert ein Element in der Hashtabelle. Dazu wird zunächst der Hashwert gebildet, dann die entsprechende Liste ausgewählt und dann das Element mit `info`=1 dort abgelegt. Beachten Sie, dass das Element bereits vorhanden sein kann und fügen es dann nicht noch einmal ein.
- Die Funktion `get(String key)` liest ein Element aus der Hashtabelle und gibt es zurück. Wenn das Element nicht vorhanden ist, soll `null` zurück gegeben werden.
- Die Funktion `clear()` löscht alle Elemente aus der Hashtabelle. Der Zustand nach dem Aufruf dieser Funktion ist also der gleiche Zustand wie nach dem Aufruf des Konstruktors.
- Die Funktion `numberOfCollisions()` gibt die Anzahl der aufgetretenen Kollisionen zurück. Eine Kollision liegt immer dann vor, wenn ein neues Element (also mit einem noch nicht vorhandenen Key) in die Hashtabelle eingefügt wird, es jedoch zum entsprechenden Hashwert schon ein Element gibt.
- Die Funktion `printHashTable()` gibt die gesamte Hashtabelle auf der Konsole aus. Die Ausgabe soll dabei sortiert nach Hashwerten erfolgen. Es werden also nacheinander alle Listen mit den entsprechenden Key-Info-Paaren ausgegeben. Außerdem soll die Anzahl der Kollisionen ausgegeben werden, die in `numberOfCollisions()` berechnet wird. Die Ausgabe sollte aussehen wie in der Datei `output.txt`. Diese Ausgabe wurde erzeugt durch das Einlesen von 20 Zahlen (die als Strings interpretiert wurden) in eine Hashtabelle der Größe 10.

Wenn Sie nur die Aufgabe 10.1 bearbeiten: Schreiben Sie eine Klasse `HashTest`, welche Zahlen aus der Datei `zahlen.txt` einliest, diese als Strings interpretiert und anschließend in die Hashtabelle einfügt. Die Größe der Hashtabelle soll als erster Kommandozeilenparameter angegeben werden, der zweite Parameter bestimmt die Anzahl der einzulesenden und einzutragenden Zahlen. **Sonst:** Implementieren Sie Aufgabe 10.2 und testen Ihre Hashtabelle mit der Anwendung `CountWords.java`.

Langaufgabe 10.2 **(4 Punkte)**

Lernziel: Hashing – Wörter zählen

Benutzen Sie die Hashtabelle aus der ersten Aufgabe, um folgendes Problem zu lösen: Sie haben eine größere Textdatei und möchten untersuchen, wie oft ein Wort aus einer fest gegebenen Menge interessierender Wörter in diesem Text vorkommt. In der Datei `CountWords.java` befinden sich bereits die Wörter, die uns interessieren. Der restliche Code soll in diese Datei eingefügt werden.

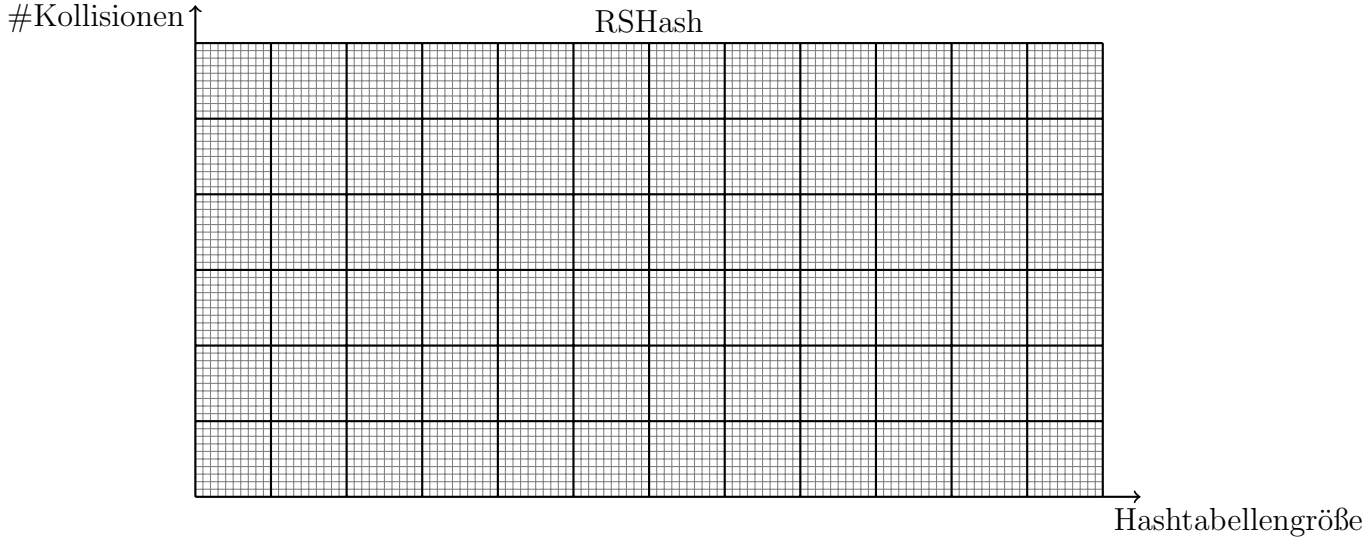
- Das Programm soll ein bis drei Parameter per Kommandozeile entgegennehmen. Der erste Parameter soll dabei immer die Textdatei sein, deren Text analysiert werden soll. Der zweite und dritte Parameter geben an, wie groß die Hashtabelle sein soll und welche der beiden Hashfunktionen `RSHash` und `JSHash` benutzt werden soll. Wenn keine Größe angegeben wird, ist die Größe 10 zu wählen, wenn keine Hashfunktion angegeben wird, ist `RSHash` zu wählen. Es soll auch möglich sein, die Hashfunktion anzugeben und die Größe wegzulassen. Die Aufrufsyntax soll wie folgt aussehen:

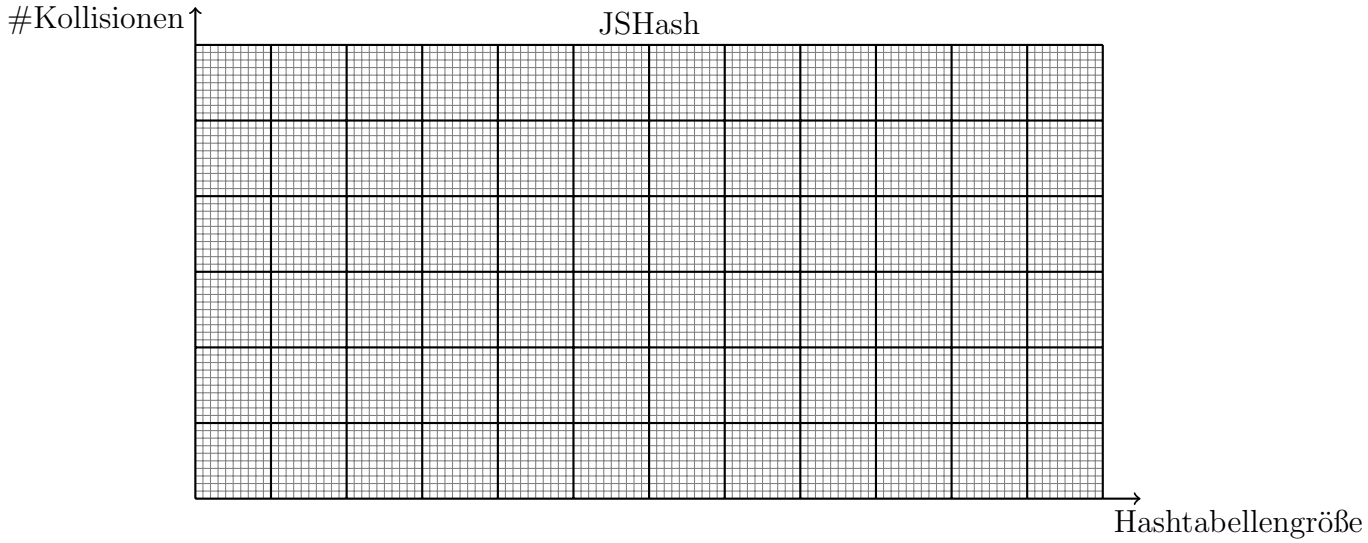
```
java CountWords textFileName [hashSize] [RSHash|JSHash]
```

- Aus der angegebenen Datei sollen alle Wörter eingelesen und in die Hashtabelle eingefügt werden. Beachten Sie hierzu die Hinweise am Ende sowie, dass Wörter durch verschiedene Trennzeichen (Punkt, Leerzeichen, Komma, Klammern, Ausrufezeichen, etc.) getrennt werden können.
- Wenn für das Wort schon ein Eintrag in der Tabelle vorhanden ist, so soll der `info`-Wert des Hashitems um 1 erhöht werden.
- Nach dem Zählen soll die Statistik für die vorgegebenen Wörter ausgegeben werden.

Geben Sie nach der Benutzung der Hashtabelle ebenfalls aus, zu wie vielen Kollisionen es gekommen ist. Untersuchen und dokumentieren Sie, wie diese Anzahl variiert, wenn die Größe der Hashtabelle 10, 100, 1000 und 2000 entspricht. Überprüfen und dokumentieren Sie ebenfalls, wie bei diesen Größen die Kollisionszahl ist, wenn man anstatt der Hashfunktion `RSHash` die alternative Hashfunktion `JSHash` benutzt. Benutzen Sie für die statistische Analyse den Text `matrix.txt`.

Hinweis: Sie sollten also acht Messergebnisse erhalten. Sie können für ihre Messergebnisse die folgenden Raster benutzen:





Beachten Sie die Hinweise und Tipps auf den folgenden Seiten.

Hinweise und Tipps

10.1 Einlesen einer Datei

Um in Java eine Text-Datei einzulesen, kann die Klasse `RandomAccessFile` benutzt werden. Zuerst erstellen Sie mittels des Dateipfades ein Objekt der Klasse `RandomAccessFile`. Der Konstruktor der Klasse benötigt zusätzlich einen zweiten String-Parameter, der die Zugriffsrechte auf die Datei definiert. Mit der Methode `readLine()` können Sie sich jeweils die nächste Zeile aus der Datei herausziehen. Bei den vorgegebenen Dateien im Ordner testCount handelt es sich um natürlichsprachige Texte, deren Wörter durch Leerzeichen, Satzzeichen, Anführungszeichen oder Klammern getrennt sein können. Um aus einer Zeile alle Wörter zu parsen, kann die Klasse `StringTokenizer` benutzt werden. Im Konstruktor erwartet die Klasse als erstes den zu trennenden String und als zweites einen String mit den Zeichen, welche für die Trennung sorgen. Mit der Funktion `nextToken()` kann der jeweils nächste Teilstring rausgeholt werden. Beachten Sie die möglichen Exceptions, die hierbei geworfen werden können. Diese sollen abgefangen werden und durch eine geeignete Ausgabe beschrieben werden. Für detaillierte Informationen zu den Klassen besuchen Sie bitte die Java-API:
`http://download.oracle.com/javase/6/docs/api/java/io/RandomAccessFile.html`
`http://download.oracle.com/javase/6/docs/api/java/util/StringTokenizer.html`

Beispiel:

```
import java.io.RandomAccessFile;
import java.util.StringTokenizer;
...
RandomAccessFile file = new RandomAccessFile(path, "r");
...
String zeile = file.readLine();
...
StringTokenizer st = new StringTokenizer(zeile, "Hier Trennzeichen festlegen");
while (st.hasMoreTokens()) {
    //Hier einzelne Tokens (= Wörter) verwenden
}
```

10.2 Array von Listen

Die Klasse `LinkedList` wird mittels `import java.util.LinkedList;` für Sie verfügbar. Die wichtigsten Methoden sind:

- `add(E e)` zum Einfügen eines Elements,
- `isEmpty()` zur Überprüfung, ob die Liste leer ist,
- `indexOf(Object o)` liefert den Index eines Elements,
- `get(int index)` um das Element an Position `index` der Liste zu bekommen,
- `clear()` um die Liste zu leeren.

Für detaillierte Informationen schauen Sie bitte in der Java-API nach:
`http://download.oracle.com/javase/6/docs/api/java/util/LinkedList.html`

Um nun ein Array von Listen anzulegen, nutzen Sie folgende Anweisung:

```
LinkedList<HashItem>[] array = new LinkedList[HashSize];
```