

## Übung zur Vorlesung *Einführung in die Informatik 2 für Ingenieure (MSE)*

Alexander van Renen (renen@in.tum.de)

<http://db.in.tum.de/teaching/ss16/ei2/>

### Lösungen zu Blatt 6

#### Aufgabe 1: Hashing with Chaining

In dieser Aufgabe wollen wir unsere eigene generische Hashtabelle implementieren, mit der man Werte unter Schlüssel ablegen kann. In der Vorlesung haben Sie Double Hashing, Hashing with Chaining und Hashing with Linear Probing kennen gelernt. Wir werden in dieser Aufgabe *Hashing with Chaining* für unsere Hashtabelle verwenden.

#### Lösung

Die folgende Klasse implementiert die Hashtabelle mit Hashing with Chaining. Sie hat eine interne Klasse `Entry` für die Einträge in der Hashtabelle bestehend aus Schlüssel und Wert.

```
1 import java.util.ArrayList;
2 import java.util.LinkedList;
3
4 class HashingChaining<K, V> {
5     // Interne Klasse fuer die Elemente in der Hashtabelle
6     class Entry {
7         K key;
8         V value;
9         public Entry(K key, V value) {
10             this.key = key;
11             this.value = value;
12         }
13     }
14
15     // Die Groesse der Hashtabelle
16     int size;
17     // Die Hash-Tabelle besteht aus einer ArrayList von verketteten
18     // Listen
19     ArrayList<LinkedList<Entry>> table;
20
21     public HashingChaining(int size) {
22         this.size = size;
23         // Erzeuge alle verketteten Listen
24         table = new ArrayList<LinkedList<Entry>>(size);
25         for (int i = 0; i < size; i++) {
26             table.add(i, new LinkedList<Entry>());
27         }
28     }
29 }
```

```

28
29 void put(K key, V value) {
30     // Berechne die Einfuegeposition
31     int hashCode = key.hashCode() % size;
32     Entry newEntry = new Entry(key, value);
33     table.get(hashCode).add(newEntry);
34 }
35
36 V get(K key) {
37     // Nehme die passende verkettete Liste fuer den Hashwert
38     int hashCode = key.hashCode() % size;
39     LinkedList<Entry> list = table.get(hashCode);
40
41     // Suche in dieser Liste nach dem Element mit dem passenden
42     // Schluessel
43     for (Entry element : list) {
44         if (element.key.equals(key)) {
45             return element.value; // Gefunden
46         }
47     }
48     return null; // Nicht gefunden
49 }
50
51 public static void main(String[] args) {
52     // Hashtabelle anlegen
53     HashingChaining<String, Integer> hashtable =
54         new HashingChaining<String, Integer>(16);
55     // Werte einfuegen
56     hashtable.put("ALL", 1);
57     hashtable.put("YOUR", 2);
58     hashtable.put("BASE", 3);
59     hashtable.put("ARE", 4);
60     hashtable.put("BELONG", 5);
61     hashtable.put("TO", 6);
62     hashtable.put("US", 7);
63     // Schluessel abfragen
64     System.out.println(hashtable.get("US"));
65 }

```

## Aufgabe 2: Komplexitätsangaben

Sie haben in der Vorlesung und Übung Komplexitätsangaben in der Landau-Notation (z.B.  $\mathcal{O}(n)$ ) kennen gelernt. Diese geben das asymptotische Laufzeitverhalten von Funktionen an. In dieser Aufgabe wollen wir feststellen, was dies in der Praxis bedeutet. Dafür messen wir die Laufzeit für das Nachschlagen in `HashMap` und `TreeMap`. Welche Laufzeitkomplexität erwarten Sie jeweils in Abhängigkeit von der Eingabegröße und können Sie diese mit Ihren Messergebnissen nachweisen?

## Lösung

Abbildung 1 zeigt die Laufzeiten für das Nachschlagen in `HashMap` und `TreeMap` mit logarithmischer x-Achse. Dadurch erkennt man sehr gut die logarithmische Laufzeit der `TreeMap` als Gerade. Die konstante Laufzeit der `HashMap` ist ebenfalls leicht erkennbar, da sie mit größerer Eingabe gleich bleibt.

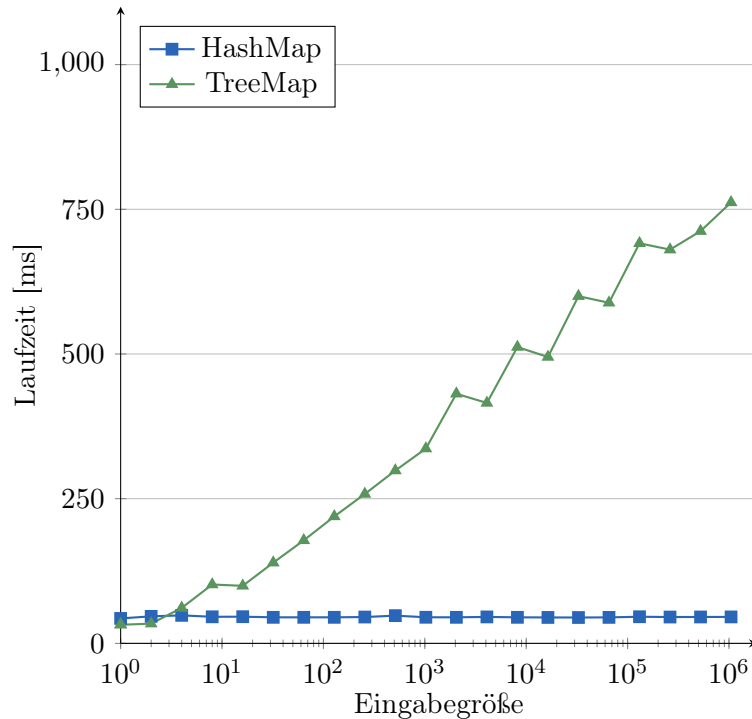


Abbildung 1: Laufzeiten für `HashMap` und `TreeMap` in logarithmischer Darstellung

## Aufgabe 3: AVL-Bäume

Fügen Sie in einen AVL-Baum nacheinander die folgenden Elemente ein und führen Sie dabei die notwendigen Rotationen durch: 4, 8, 16, 12, 14, 3, 2, 6, 5

## Lösung

Abbildung 2 zeigt die Einfügevorgänge im AVL-Baum zusammen mit den notwendigen Rotationen. Die kleinen Zahlen neben den Knoten zeigen jeweils den Balancierungsfaktor. Rote Zahlen stehen dabei für eine Verletzung des AVL-Kriteriums.

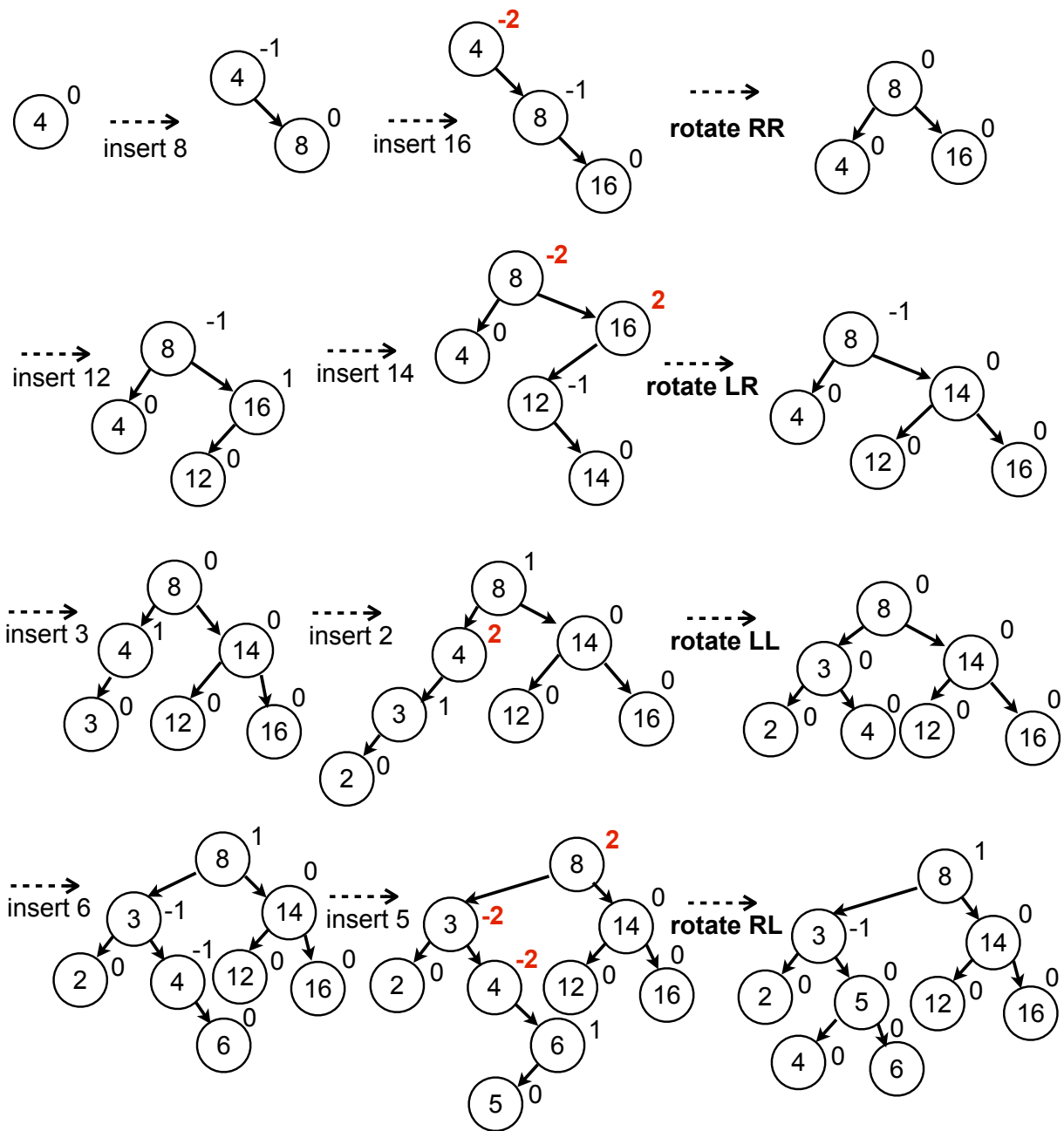


Abbildung 2: Einfügesequenz für den AVL-Baum mit Rotationen