Prof. Dr. Gerd Stumme, Folke Eisterlehner, Dominik Benz
Fachgebiet Wissensverarbeitung

29.06.2009

12. Hausübung „Algorithmen und Datenstrukturen“

Sommersemester 2009

Abgabetermin: Montag, 06.07.2009, 10:00 Uhr

1 (2,4)-Bäume

Aufgabe 1 (minimale Höhe von 2,4-Bäumen)

Welche Höhe hat ein 2,4-Baum mit n Schlüsseln mindestens? Zeichnen Sie einen 2,4-Baum mit $n = 15$ Schlüsseln mit minimaler Höhe. Zur Abgabe der Zeichnung können Sie entweder den Baum mit einem Zeichenprogramm ihrer Wahl erstellen und in das PDF-Format konvertieren, oder eine “ASCII-Repräsentation” in folgendem Stil anfertigen (bei der Abgabe von .txt-Dateien):

```
     [4 6 8]
    /   |   \
[1]  [5]  [9 10]
```

(4 Punkte)

Aufgabe 2 (Anzahl strukturell verschiedener Bäume)

Wieviele strukturell verschiedene (2,4)-Bäume der Höhe 3 gibt es? Symmetrische Varianten sollen doppelt gezählt werden. Geben Sie die exakte Zahl an, eine Begründung ist nicht gefordert.

(4 Punkte)

2 AVL-Bäume

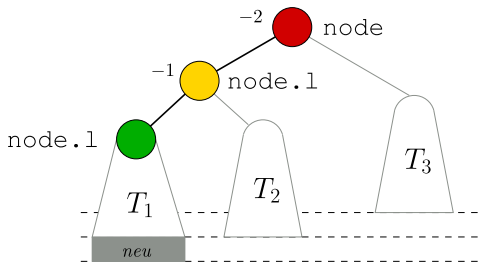
Aufgabe 3 (Einfügeoperation in AVL-Bäumen)

In der Vorlesung wurde beispielhaft beschrieben, wie neue Schlüsselwerte in einen AVL-Baum eingefügt werden können, so dass der resultierende Baum weiterhin gesichert höhenbalanciert ist. Auf der Webseite zur Vorlesung finden Sie die generische Klasse **AbstractAVLTree<K>**, welche das Einfügen eines Schlüssels **key** in einen AVL-Baum mit Wurzel **node** rekursiv implementiert:

```
boolean recursiveInsert(AVLNode<K> node, K key) {
   // 1. Fall: node ist ein Blatt :
```

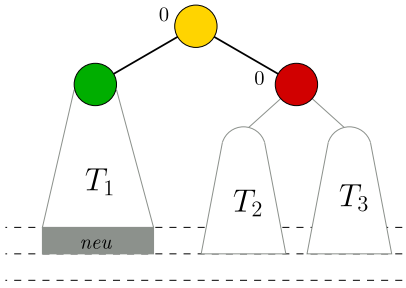
```
   //    mache node zum inneren Knoten mit Schluessel key
   //    (beachte, dass Blaetter durch Knoten mit leerem
   //     Schluesselwert (null) dargestellt werden)
   if( node.getKey()==null ) {
      node.setKey(key);
      node.setLeft(new AVLNode<K>(null));
      node.setRight(new AVLNode<K>(null));
      return true;
   }
   // 2. Fall: node.key = key: nichts zu tun
   else if( node.getKey().compareTo(key)==0 ) {
      return false;
   }
   // 3. Fall: node.key < key: fuege rekursiv in rechten Teilbaum ein
   else if( node.getKey().compareTo(key)<0 ) {
      boolean higher = recursiveInsert(node.getRight(), key);
      return rebalanceRightAfterInsertion(node, higher);
   }
   // 4. Fall: node.key > key: fuege rekursiv in linken Teilbaum ein
   else {
      boolean higher = recursiveInsert(node.getLeft(), key);
      return rebalanceLeftAfterInsertion(node, higher);
   }
}
```

Es kann natürlich sein, dass durch das Einfügen des Schlüssels in einen Teilbaum, die Balancebedingung für den Wurzelknoten **node** verletzt wird, wie in folgender Grafik für den Fall **node.key>key** mit den entsprechenden Balancewerten dargestellt (zur besseren Lesbarkeit haben wir **left** durch **l** ersetzt):



Entsprechend liefert die Methode **true** zurück, falls der Baum durch die Einfügeoperation gewachsen ist und **false** sonst. Falls die Balancebedingung verletzt wurde, muss der Baum *rebalanciert* werden. Hierzu werden die Methoden **rebalanceLeftAfterInsertion(...)** bzw. **rebalanceRightAfterInsertion(...)** aufgerufen.

Für die im obigen Beispiel dargestellte Situation wird z.B. eine *Rechtsrotation* um **node** durchgeführt:



Insgesamt sind jeweils fünf verschiedene Situationen nach dem Einfügen eines Schlüssels in den linken bzw. rechten Teilbaum zu beachten. Diese sind für den linken Teilbaum im Kommentar zur Methode **rebalanceLeftAfterInsertion(...)** beschrieben.

Die Methode **rebalanceRightAfterInsertion(...)** ist abstrakt. Sie zu Implementieren ist Teil dieser Aufgabe.

a) Leiten Sie eine Klasse **AVLTree** von **AbstractAVLTree** ab, welche die fehlende Methode implementiert. Dabei erfolgt die Implementierung der Rebalancierungsmethode **rebalanceRightAfterInsertion(...)** symmetrisch zu der von **rebalanceLeftAfterInsertion(...)**. Fangen Sie also am besten damit an, sich die unterschiedlichen Fälle zu verdeutlichen und durch entsprechende Zeichnungen bildhaft zu machen.

(12 Punkte)

b) Implementieren Sie ferner eine private rekursive Methode **isAVLTree(AVLNode<K> t)**, welche **true** zurückliefert, falls **t** Wurzel eines AVL-Baums ist und **false** sonst.

Stellen Sie zusätzlich eine öffentliche parameterfreie Variante **isAVLTree()** zur Verfügung, welche obige rekursive Methode mit dem Wurzelknoten des Baums aufruft.

(8 Punkte)

c) Schreiben Sie eine Klasse **AVLTest**, in deren **main**-Methode ein AVL-Baum für Integers erzeugt wird und in einer Schleife mit 10000 Zufallszahlen gefüllt wird (verwenden Sie die Klasse **DataSource3** von der Vorlesungsseite zum Erzeugen der Zufallszahlen). Nach jeder Einfügeoperation soll mittels eines Aufrufs der Methode **isAVLTree()** überprüft werden, ob der Baum ein AVL-Baum ist. Sobald die Methode **false** zurückliefert soll der Einfügevorgang mit einer Fehlermeldung abgebrochen werden. Ansonsten soll die mittels eines Aufrufs von **getInternalPathLength()** ermittelte *innere Pfadlänge* ausgeben werden. (5 Punkte)

d) *Zusatz:* Vergleichen Sie die innere Pfadlänge mit der des binären (natürlichen) Suchbaums vom letzten Aufgabenblatt, wenn dieser mit den selben Zahlen in der selben Reihenfolge gefüllt wurde.

Viel Spaß und viel Erfolg!