Algorithmen & Datenstrukturen

Blatt 7

Dr. Matthias Thimm
Tina Walber, Leon Kastler, Martin Leinberger und Maximilian Strauch
Fachbereich Informatik, Universität Koblenz-Landau

7. Januar 2014

1 Multiple Choice zu AVL Bäumen(3 Punkte)

Kreuzen Sie an, ob die Aussage innerhalb eines Themas **wahr** (W) oder **falsch** (F) ist. Für jedes falsche oder fehlende Kreuzchen werden 0,5 Punkte abgezogen. Die Mindestpunktzahl für diese Aufgabe ist insgesamt 0.

Gegeben ist ein AVL Baum:

◯ W ◯ F In jedem Knoten ist die Differenz der Höhe des linken und rechten Teilbaums maximal 1.

◯ W ◯ F Ausschließlich in der Wurzel darf die Differenz der Höhe des linken und rechten Teilbaums maximal 1 sein.

◯ W ◯ F Für jeden Knoten gilt: Die Anzahl der Knoten im rechten und im linken Teilbaum darf sich maximal um 1 unterscheiden.

◯ W ◯ F Im rechten Teilbaum der Wurzel können mehr als doppelt so viel Knoten wie im Linken sein.

Gegeben ist ein Rot-Schwarz-Baum:

◯ W ◯ F Für jeden Knoten k gilt: Jeder Pfad von k zu einem Blatt enthält die gleiche Anzahl an roten Knoten.

◯ W ◯ F Kein Pfad von der Wurzel zu einem Blatt kann mehr rote als schwarze Knoten enthalten.

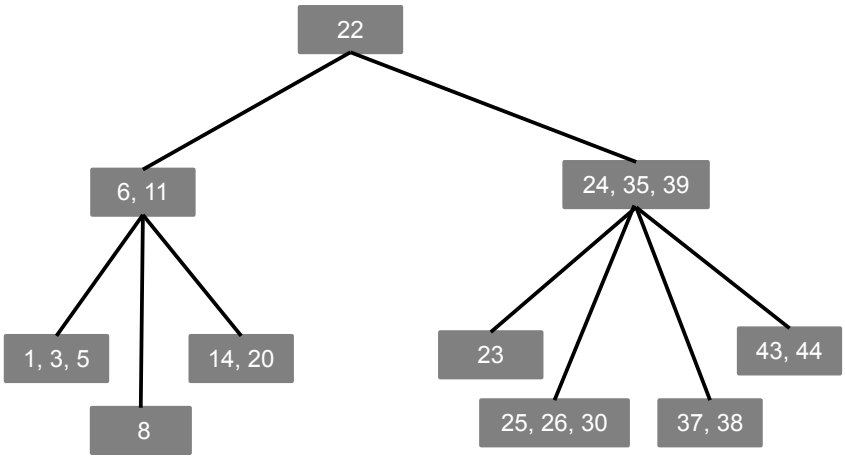
2 AVL-Baum und Rot-Schwarz-Baum (4 Punkte)

Fügen Sie folgende Zahlen in einen initial leeren (i) AVL-Baum und (ii) Rot-Schwarz-Baum ein: 7, 9, 20, 8, 4, 2, 3, 12, 11

Zeichen Sie den jeweiligen Baum nach jeder Einfügeoperation und nach jeder erforderlichen Ausgleichsoperation. Geben Sie beim AVL-Baum für jeden Knoten das Balance-Kriterium an, bzw. beim Rot-Schwarz-Baum für jeden Knoten die Farbe. Geben Sie an, welche Schritte Sie zum Ausgleich des Baums vornehmen. Je 2 Punkte für (i) und (ii).

3 Rot-Schwarz- und 2-3-4-Bäume (4 Punkte)

In der Vorlesung wurden sowohl Rot-Schwarz Bäume, als auch 2-3-4 Bäume besprochen. 2-3-4 Bäume werden zumeist durch Rot-Schwarz Bäume implementiert. Gegeben ist folgender 2-3-4 Baum:



a) Erstellen Sie den zugehörigen Rot-Schwarz Baum aus dem gegebenen 2-3-4 Baum (1 Punkt).

b) Fügen Sie folgende Zahlen in den Rot-Schwarz Baum ein und nehmen Sie die notwendigen Transformationen vor. Zeichnen Sie den Baum nach jeder Einfügeoperation und jedem Ausgleich. Geben Sie an, welche Schritte Sie zum Ausgleich des Baums vornehmen. (2 Punkte).

```
[21, 42, 2]
```

c) Erstellen Sie den korrespondierenden 2-3-4 Baum aus dem zuletzt erhaltenen Rot-Schwarz Baum (1 Punkt).

4 Rechtschreibprüfung (9 Punkte)

Ihre Aufgabe besteht darin, die Grundlage für eine Rechtschreibprüfung zu implementieren, einmal auf der Grundlage der Java Klasse TreeSet[1] und zum anderen mit Hilfe von ternären Bäumen[2].

Für eine Rechtschreibprüfung ist es wichtig eine Datenstruktur zu haben, mir der man schnell herausfinden kann, ob ein Wort richtig geschrieben ist oder nicht. Zudem ist es wichtig herauszufinden, welche Worte ähnlich einem falsch geschriebenen Wort sind, was wir aber bei dieser Aufgaben nicht beachten. Ternäre Bäume sind eine dynamische Datenstruktur, die für diese Zwecke eingesetzt wird. Diese Bäume besitzen, neben ihren Werten, jeweils bis drei Nachfolgeknoten. Als Werte enthält ein Knoten zum einen einen Buchstaben und eine Marke, die das Ende eines Wortes anzeigt. Um Worte in den Baum einzufügen wird der erste Buchstaben des Wortes mit dem Wert der Wurzel des Baums verglichen:

a) Sind sie gleich, geht man auf den mittleren Nachfolgeknoten und vergleicht dessen Wert mit dem nächsten Buchstaben des Wortes.

b) Sind sie unterschiedlich folgen wir dem linken/rechten Nachfolgenoten, falls der aktuelle Buchstabe des Wortes im Alphabet vor/nach dem Buchstaben im Knoten kommt.

c) Existiert der Knoten nicht, wird er erstellt, sein Wert mit dem Buchstaben belegt und wiederholt diese Prozedur für alle folgenden Buchstaben des Wortes.

Beim letzten Buchstaben des Wortes wird der Knoten der den Buchstaben enthält markiert, damit wir wissen dass das Wort zu Ende ist. Die Abbildung 1 zeigt wie ein Baum aus den Worten *alpha*, *bravo*, *all* und *brav* aufgebaut wird.

Um nun zu erkennen, ob ein Wort in unserem Baum enthalten ist, führen wir folgenden Algorithmus durch: Wir beginnen mit dem ersten Buchstaben des Wortes und vergleichen ihn mit dem Werten der Wurzel.

a) Sind sie gleich, vergleichen wir den nächsten Buchstaben des Wortes mit dem Wert des mittleren Nachfolgeknotens.

b) Sind sie unterschiedlich gehen wir zum linken/rechten Nachfolgenoten, falls der Buchstabe des Wortes vor/nach dem Wert des aktuellen Knotens kommt, und vergleichen dessen Wert mit dem aktuellen Buchstaben des Wortes.

c) Existiert der Knoten nicht, ist das Wort nicht in unserem Baum.

d) Hat das Wort keine weiteren Buchstaben, betrachten wir uns die Markierung des aktuellen Knotens. Ist diese gesetzt, ist das Wort in unserem Baum, sonst nicht.

[1] http://docs.oracle.com/javase/6/docs/api/java/util/TreeSet.html
[2] http://en.wikipedia.org/wiki/Ternary_tree

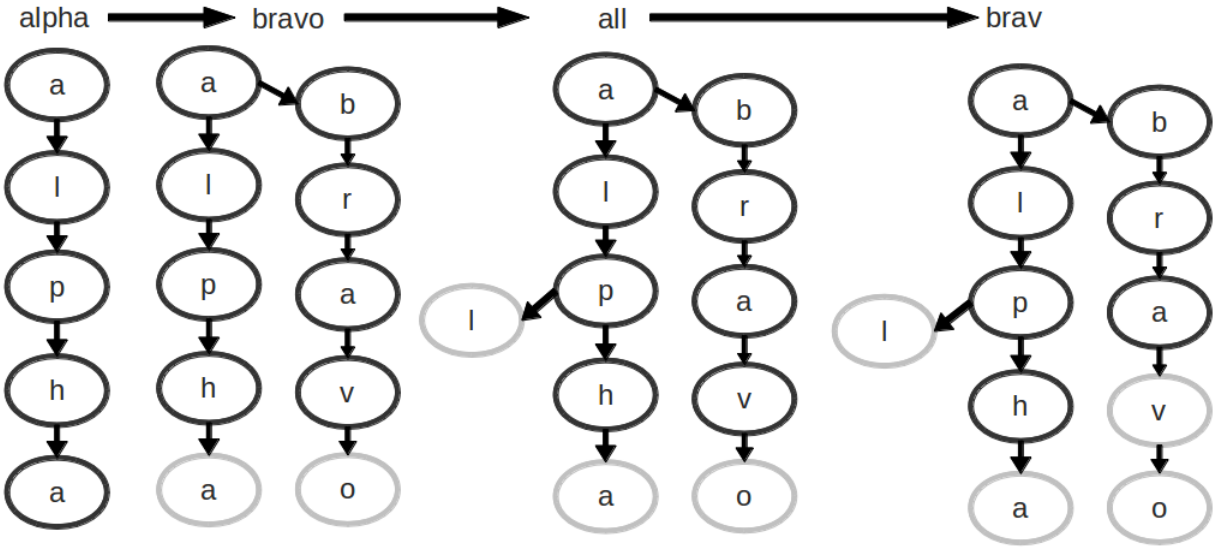


Abbildung 1: Ternärbaum aufbauen, Wortenden sind grau markiert

Aufgaben

1) Kopieren Sie den Inhalt des Ordners `TernaryTree` in Ihren Workspace. Im Unterordner `src` finden Sie die Java Dateien, die Sie für die Implementierung nutzen sollen. Lesen Sie auch die Kommentare im Quellcode. Eine einfache Implementierung mit `java.util.ArrayList`, welche nicht besonders effizient ist, ist Ihnen vorgegeben. Sie finden im Ordner `data` die Datei `wordsEn.txt` mit einer Liste aller englischen Worte. Diese können Sie nutzen, um Ihre jeweilige Datenstruktur aufzubauen. (0 Punkte)

2) Implementieren Sie den die Rechtschreibprüfung, analog zu der gegebenen Implementation, mit Hilfe der Java Klasse `java.util.TreeSet`. Hierzu müssen Sie die Schnittstelle `aud.SpellCheck` und deren Methoden implementieren. (2 Punkte)

3) Implementieren Sie die Rechtschreibprüfung mit Hilfe von ternären Bäumen. Hierzu müssen Sie die Grundstruktur des ternären Baums definieren und die Methoden der Schnittstelle `aud.SpellCheck` implementieren. Dabei gilt es speziell zu beachten, dass ternäre Bäume nicht die einzelnen Schlüsselworte zusammen mit den Werten in den Knoten speichern, sondern die einzelnen Buchstaben in den Knoten enthalten sind. (4 Punkte)

4) Überlegen Sie sich einen Ansatz wie Sie Ihre Implementation evaluieren wollen. Implementieren Sie diesen Ansatz und vergleichen Sie Ihre Implementation des ternären Baums mit der ArrayList-Variante und Ihrer Variante mit TreeSets. Nutzen Sie hierzu die Liste aller englischen Worte (`wordsEn.txt` im Ordner `data`) aufzubauen. Diskutieren Sie das Ergebnis Ihrer Evaluation. (3 Punkte)

Hinweis

Ihre Implementierung der ternären Bäume wird für den Anwendungsfall nicht unbedingt schneller sein als die anderen beiden Varianten.

Weihnachtsmann-Aufgabe (freiwillig!)

Der Weihnachtsmann ist ziemlich genervt, da in den letzten Jahren eine abnehmende Qualität in Puncto Rechtschreibung bekommen haben. Zudem hat die Anzahl von Zeichengebilden wie *rofl* und *YOLO* stark zugenommen. Erstellen Sie ein Programm, dass die Wunschzettel in Form von Textdateien einliest und falsche oder unbekannte Worte markiert bzw. direkt korrigieren kann. Diese Aufgabe ist freiwillig, Sie können aber Ihren Ansatz uns gerne zuschicken.