Algorithmieren und Programmieren

Sommersemester 2023

Lehrstuhl Programmiersprachen und Compilerbau
Prof. Dr. rer. nat. habil. Petra Hofstedt
Ilja Becker
Andreas Eberle, Jan Robert Meyer, Oliver Pflug

Brandenburgische
Technische Universität
Cottbus - Senftenberg

Übungsblatt 10

Abgabedatum: 09.07.2023

Hinweise

- Beachten Sie, dass die Tutoren unkommentierte Programme nicht als Lösung akzeptieren, selbst wenn die Programme richtig funktionieren. Zu einer richtigen Lösung gehören immer sinnvolle Kommentare, deren Umfang der Komplexität des Programms angemessen ist.
- **Halten Sie sich an die in den Übungsblättern vorgegebenen Namen von Funktionen und Dateien, Funktionstypen (Typsignaturen), Reihenfolge der Parameter und verwenden Sie — sofern vorhanden — die Vorgaben!**
- Auf den Übungsblättern finden Sie einige Haskell-Quelltextfragmente. Diese sind, der besseren Lesbarkeit wegen, unter Nutzung einiger mathematischer Sonderzeichen wiedergegeben.
- Für diese Veranstaltung wird die Verwendung des ***Glasgow Haskell Compiler (GHC)*** (`https://www.haskell.org/`) empfohlen. Weitere Informationen finden Sie im Software-Reiter auf Moodle.
- Als **Tutoriumsaufgabe** markierte (Teil-)aufgaben werden in den Übungen ausführlicher besprochen. Die schriftlichen bzw. elektronischen Lösungen müssen jedoch trotzdem mit abgegeben werden. Bitte schauen Sie Sich diese Aufgaben im Vorfeld der Übung an und bereiten Sie Sich darauf vor.
- Die Abgabe Ihrer Lösungen erfolgt vor Ablauf der Abgabefrist digital über die Moodle-Plattform an Ihren Tutor. Erstellen Sie dazu ein PDF-Dokument, das die Lösungen Ihrer schriftlichen Aufgaben enthält. Laden Sie dieses PDF-Dokument und die erzeugten Hakell-Dateien, mit den in den Aufgaben vorgegebenen Namen, bei Moodle hoch.

Informationsquellen

- Sie finden unter `http://haskell.org/` sehr viele Informationen über die Programmiersprache Haskell. Von besonderem Interesse sind dabei sicherlich die Übersicht über zahlreiche online verfügbare Haskell-Tutorials (`https://wiki.haskell.org/Tutorials`) sowie die Suchmaschine Hoogle (`http://hoogle.haskell.org/`) für die Haskell-API, die Ihnen mit zunehmender Haskell-Erfahrung wertvolle Dienste leisten wird.

Sie können maximal **(10 Punkte)** mit diesem Übungsblatt erreichen.

Aufgabe 1 (AVL-Bäume) 4 Punkte

Für jede Teilaufgabe gibt es einen Punkt.

1. Geben Sie den AVL-Baum an, der durch sukzessives Einfügen der folgenden Werte in einen anfänglich leeren AVL-Baum entsteht (unter Annahme der üblichen Ordnung von Werten des Typs Int):

   `1 3 5 7 8 6 4 2`

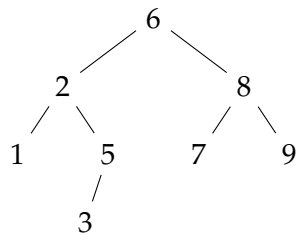
   Geben Sie auch den nach jedem Einfügeschritt entstandenen Zwischenbaum an.

2. Geben Sie den AVL-Baum an, der durch sukzessives Einfügen der folgenden Werte in einen anfänglich leeren AVL-Baum entsteht (unter Annahme der üblichen Ordnung von Werten des Typs Int):

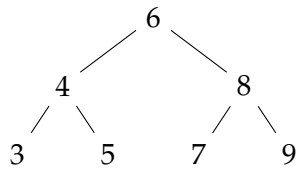
   `6, 4, 8, 2, 1, 3, 7, 5`

   Geben Sie auch den nach jedem Einfügeschritt entstandenen Zwischenbaum an.

3. Geben Sie den AVL-Baum an, der durch Löschen der Werte 7 und 8 (in dieser Reihenfolge) aus dem im Folgenden angegebenen AVL-Baum entsteht. Geben Sie die Zwischenbäume und eventuell notwendige Rotationsschritte an.



4. Geben Sie den AVL-Baum an, der durch Löschen des Wertes 6 aus dem im Folgenden angegebenen AVL-Baum entsteht. Geben Sie die Zwischenbäume eventuell notwendige Rotationsschritte an. Sollten mehrere Lösungen in Frage kommen, so geben Sie alle Lösungen an.



Aufgabe 2 (AVL-Bäume) 6 Punkte

Verwenden Sie für die Abgabe den Dateinamen: `Avltree.hs`

Wir betrachten den folgenden Datentyp für AVL-Bäume:

```
data AVLBaum el = AVLBlatt
    | AVLKnoten Int (AVLBaum el) el (AVLBaum el)
    deriving Show
```

Knoten eines AVL-Baums enthalten im Vergleich mit binären Suchbäumen zusätzlich eine ganze Zahl, die die Höhe des Baumes angibt. Wir speichern die Höhe statt der Balance, da sich die Höhe bei Änderungen des Baumes einfacher aktualisieren lässt. Die Balance lässt sich aus den Höhen der Unterbäume berechnen.

Auf die Aufgaben 2.1-2.4 gibt es insgesamt **(2 Pkt.)**

Auf die Aufgabe 2.5 gibt es **(2 Pkt.)**

Auf die Aufgabe 2.6 gibt es **(2 Pkt.)**

1. **(Tutoriumsaufgabe)** Programmieren Sie eine Haskell-Funktion

   ```
   hoehe :: AVLBaum el →Int
   ```

   welche die Höhe eines AVL-Baums liefert.

2. **(Tutoriumsaufgabe)** Programmieren Sie eine Haskell-Funktion

   ```
   balance :: AVLBaum el →AVLBaum el →Int
   ```

   welche aus dem linken und dem rechten Unterbaum eines nicht-leeren AVL-Baums die Balance des Baumes berechnet.

3. **(Tutoriumsaufgabe)** Programmieren Sie eine Haskell-Funktion

   ```
   avlKnoten :: AVLBaum el →el →AVLBaum el →AVLBaum el
   ```

   welche aus einem linken Unterbaum, einer Wurzel und einem rechten Unterbaum einen Wert vom Typ `AVLBaum` konstruiert. Der entstandene Baum muss nicht unbedingt balanciert sein.

4. **(Tutoriumsaufgabe)** Programmieren Sie eine Haskell-Funktion

   ```
   verbinden :: AVLBaum el →el →AVLBaum el →AVLBaum el
   ```

   welche aus zwei AVL-Bäumen und einem Element einen neuen AVL-Baum konstruiert. Dabei wird vorausgesetzt, dass die Elemente des ersten AVL-Baums kleiner und die des zweiten AVL-Baums größer als das direkt übergebene Element sind und sich die Höhen der Bäume um maximal zwei unterscheiden. Normalerweise sollen die beiden AVL-Bäume als linker und rechter Unterbaum und das Element als Wurzel des neuen Baums verwendet werden. Wenn dadurch allerdings ein unbalancierter Baum entstehen würde, soll mittels Rotation bzw. Doppelrotation ein entsprechender balancierter Baum erzeugt werden.

5. Programmieren Sie eine Haskell-Funktion

   ```
   avlEinfuegen :: (Ord el) ⇒el →AVLBaum el →AVLBaum el
   ```

   welche ein Element in einen AVL-Baum einfügt. Wenn der angegebene Wert bereits im AVL-Baum enthalten ist, soll der Baum nicht geändert werden.

6. Programmieren Sie eine Haskell-Funktion

   ```
   avlLoeschen :: (Ord el) ⇒el →AVLBaum el →AVLBaum el
   ```

   welche ein Element aus einem AVL-Baum löscht. Wenn der angegebene Wert nicht im AVL-Baum enthalten ist, soll der Baum nicht geändert werden.