Algorithmieren und Programmieren

Sommersemester 2019

Prof. Dr. rer. nat. habil. Petra Hofstedt
Sven Löffler M. Sc.
Sonja Breuß, Deborah Buckenauer, Johannes Kuhn, Julius Schöning, Carlo Bückert

Brandenburgische
Technische Universität
Cottbus - Senftenberg

Übungsblatt 8

Abgabedatum: 12.06.2019

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 der ***Haskell-Plattform*** (`https://www.haskell.org/platform/`) empfohlen.
- Als **Tutoriumsaufgabe** markierte (Teil-)aufgaben werden in den Übungen ausführlicher besprochen. Die schriftliche 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-Datein, 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 (`http://haskell.org/haskellwiki/Tutorials`) sowie die Suchmaschine Hoogle (`http://haskell.org/hoogle/`) für die Haskell-API, die Ihnen mit zunehmender Haskell-Erfahrung wertvolle Dienste leisten wird.

Sie können maximal **(7 Punkte)** mit diesem Übungsblatt erreichen.

Aufgabe 1 (Sortieren mit variabler Ordnung) 2 Punkte

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

Es soll eine Haskell-Funktion `quickSort` programmiert werden, welche nach verschiedenen Kriterien sortieren kann. Dazu soll ihr neben der zu sortierenden Liste eine Vergleichsfunktion als Argument übergeben werden. Die Vergleichsfunktion soll eine irreflexive Ordnung darstellen.

1. **(Tutoriumsaufgabe)** Welchen Typ hat eine Vergleichsfunktion für einen Typ `el`? Welchen Typ hat dann die Funktion `quickSort`, die als zusätzliches Argument die zu verwendende Vergleichsfunktion erhält?
2. Schreiben Sie eine enstprechend angepasste Quicksort-Implementierung.
3. Betrachten Sie den folgenden Datentyp zur Repräsentation von Studenten:

```
data Studiengang = Informatik | Ebusiness | IMT deriving (Show, Ord, Eq)
data Datum = Datum {tag::Int, monat::Int, jahr :: Int} deriving Show
data Student = Student {
    name :: String,
    vorname :: String,
    studiengang :: Studiengang,
    geburtsdatum :: Datum
    }
    deriving (Show)
```

   Programmieren Sie die Vergleichsfunktionen

   `nameVergleich`, `vornameVergleich`, `studiengangVergleich` und `datumVergleich`

   die von der angepassten `quickSort`-Implementierung verwendet werden können, um eine Liste von Studenten nach Name, Vorname, Studiengang bzw. Geburtsdatum zu sortieren. So soll z.B.

```
quicksort nameVergleich studentenListe
```

   eine Liste mit Studenten nach dem Nachnamen sortieren.

4. Programmieren Sie eine Haskell-Funktion `und` zur Komposition von Vergleichsfunktionen. Es seien `v1` und `v2` zwei Vergleichsfunktionen. Dann soll `v1 und v2` eine Vergleichsfunktion sein, mittels der zuerst nach dem durch `v1` dargestellten Kriterium und bei äquivalenten Einträgen nach dem durch `v2` dargestellten Kriterium sortiert wird. So soll z.B.

```
quicksort (und studiengangVergleich nameVergleich) studentenListe
```

   Studenten zunächst nach Studiengang und bei gleichem Studiengang anschließend nach dem Namen sortieren.

Aufgabe 2 (Binäre Bäume und Suchbäume) 3 Punkte

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

In dieser Aufgabe wollen wir binäre Bäume zum Speichern von Elementen und als Suchbäume verwenden.

Es gibt auf die Aufgaben 1-5, 6-10 sowie 11-13 jeweils einen Punkt.

1. **(Tutoriumsaufgabe)** Geben Sie einen Datentyp zur Repräsentation binärer Bäume an.

2. **(Tutoriumsaufgabe)** Programmieren Sie eine Haskell-Funktion `makeEmptyBinTree`, welche einen leeren binären Baum erstellt. Geben Sie den Typ der Funktion mit an.

3. **(Tutoriumsaufgabe)** Programmieren Sie eine Haskell-Funktion `makeBinTree`, welche aus einem Element einen binären Baum erstellt. Der entstandene Baum enthält das Element als Wurzel-Element und zwei leere Unterbäume. Geben Sie den Typ der Funktion mit an.

4. **(Tutoriumsaufgabe)** Wir wollen den binären Baum als Suchbaum verwenden. Bei einem binären Suchbaum sind die Elemente im linken Unterbaum kleiner als der Knoten und im rechten Unterbaum größer. Diese Ordnung muss beim Einfügen neuer Elemente beachtet werden.

   Programmieren Sie eine Funktion **`insert`**, welche ein neues Element in einen schon vorhandenen Suchbaum einfügt. Wenn ein Element bereits in dem Baum enthalten ist, soll es nicht noch einmal eingefügt werden. Geben Sie den Typ der Funktion mit an.

5. Wir wollen jetzt, dass alle Elemente einer Liste in einen Suchbaum eingefügt werden können. Schreiben Sie dazu eine Haskell-Funktion `createBinTree`, welche aus einer Liste einen binären Suchbaum erzeugt. Fügen Sie dabei ausgehend von einem leeren Baum schrittweise jedes Element aus der Liste in den Baum ein. Geben Sie den Typ der Funktion mit an.

6. Wir wollen jetzt die in einem binären Baum gespeicherten Elemente wieder auslesen. Schreiben Sie dazu eine Haskell-Funktion `nodesInOrder`, welche einen binären Baum erhält und alle in ihm gespeicherten Elemente als Liste zurück gibt. Dabei sollen für jeden Knoten zuerst die Elemente des linken Unterbaumes, dann das Element am Knoten selbst und schließlich die Elemente des rechten Unterbaumes ausgegeben werden. Geben Sie den Typ der Funktion mit an.

7. Schreiben Sie eine Funktion `nodesPreOrder`, welche die Elemente so ausliest, dass das Knotenelement jeweils vor den Elementen der Unterbäume ausgelesen wird. Geben Sie den Typ der Funktion mit an.

8. Schreiben Sie eine Funktion `nodesPostOrder`, welche die Elemente so ausliest, dass das Knotenelement nach den Elementen der Unterbäume ausgelesen wird. Geben Sie den Typ der Funktion mit an.

9. Schreiben Sie eine Funktion `nodesLevelOrder`, welche die Elemente des binären Baumes in der Reihenfolge einer Breitensuche ausliest. Geben Sie den Typ der Funktion mit an.

10. Schreiben Sie eine Haskell-Funktion `leaves`, welche alle Blätter eines binären Baumes ausliest. Geben Sie den Typ der Funktion mit an.

11. Die Funktion **`map`** lässt sich auf Bäume verallgemeinern. Schreiben Sie eine Funktion `binTreeMap`, welche eine übergebene Funktion f auf alle Elemente eines Baumes anwendet. Geben Sie den Typ der Funktion mit an.

12. Auch die Funktion **`foldr`** lässt sich auf Bäume verallgemeinern. Schreiben Sie eine Funktion `binTreeFold :: (acc →el →acc →acc) →acc →BinTree el →acc`, welche einen Baum mit Hilfe der übergebenen Funktion f von unten nach oben faltet.

13. Programmieren Sie mit Hilfe der Funktion `binTreeFold` jeweils eine Version von `nodesInOrder`, `nodesPreOrder, nodesPostOrder` und `leaves`.

Aufgabe 3 (Aufwand) 2 Punkte

Geben Sie für die im Folgenden genannten Funktionen aus Aufgabe 2 die Aufwände T^{worst} und T^{best} an. Gehen Sie bei der Bestimmung der Aufwände analog zu den in der Vorlesungen präsentierten Beispielen vor. Geben Sie ebenfalls die Aufwandklasse an.

1. Ermittlen Sie den Aufwand für die **`insert`** Funktion aus Aufgabe 2. Schildern Sie wie der Baum und das Element aussehen müssen, um den worst bzw. best case für den nicht trivialen Fall zu erreichen.
2. Ermitteln Sie den Aufwand für die `createBinTree` Funktion aus Aufgabe 2. Schildern Sie wie der Baum und die Eingabeliste aussehen müssen, um den worst bzw. best case für den nicht trivialen Fall zu erreichen.
3. Ermittlen Sie den Aufwand für die `leaves` Funktion aus Aufgabe 2.