WS 2017/2018

Funktionale Programmierung

5. Übungsblatt

Prof. Dr. Margarita Esponda

(Abgabetermin: Mi. den 29. Nov., um 9:55)

Ziel: Auseinandersetzung mit Funktionen höherer Ordnung, Sortieralgorithmen und Zeitkomplexität.

1. Aufgabe (5 Punkte)

a) Definieren Sie unter sinnvoller Verwendung der **foldl** Funktion eine Funktion **toDecFrom**, die eine Basis **b** (mit **1<b<10**) und eine Zahl als Liste der einzelnen Ziffern bekommt und die Dezimaldarstellung der Zahl berechnet.

Anwendungsbeispiel: **toDecFrom** 4 [1, 2, 3, 0] => 108

b) Analysieren Sie die Zeitkomplexität der Funktion.

2. Aufgabe (9 Punkte)

Betrachten Sie folgende zwei Definitionen der Funktion **flatten** :: [[a]] -> [a], welche eine Liste von Listen zu einer Liste kombiniert.

flatten_r xss = foldr (++) [] xss

flatten_l xss = foldl (++) [] xss

a) Analysieren Sie die Zeitkomplexität beider Funktionen.

b) Definieren Sie die **flatten** Funktion unter sinnvoller Verwendung von Listengeneratoren.

3. Aufgabe (7 Punkte)

Betrachten Sie folgende Potenz-Sequenzen, die aus allen Kombinationen der Zahlen **b** und **e** entstehen können, wenn **b** als Basis und **e** als Exponent für die Berechnungen verwendet werden und $2 \le b \le 4$ und $2 \le e \le 6$ sind.

$$2^2 = 4, \quad 2^3 = 8, \quad 2^4 = 16, \quad 2^5 = 32, \quad 2^6 = 64$$
$$3^2 = 9, \quad 3^3 = 27, \quad 3^4 = 81, \quad 3^5 = 243, \quad 3^6 = 729$$
$$4^2 = 16, \quad 4^3 = 64, \quad 4^4 = 256, \quad 4^5 = 1024, \quad 4^6 = 4096$$

Wenn wir alle Zahlen ohne Wiederholungen sortiert auflisten, haben wir folgende Zahlenliste:

4, 8, 9, 16, 27, 32, 64, 81, 243, 256, 729, 1024, 4096

a) Definieren Sie eine Haskell-Funktion, die bei Eingabe von **n** und **m** die Liste aller paarweise verschiedenen Potenzen berechnet, mit $2 \le b \le n$ und $2 \le e \le m$.

b) Analysieren Sie die Komplexität Ihre Funktion.

4. Aufgabe (6 Punkte)

a) Definieren Sie eine Funktion, die aus einer beliebigen Liste natürlicher Zahlen die kleinste natürliche Zahl findet, die nicht in der Liste vorkommt.

Anwendungsbeispiel: **minNatNotIn** [3, 5, 2, 7, 6, 10, 0, 1, 4, 12] => 8

b) Analysieren Sie die Komplexität Ihrer **minNatNotIn** Funktion.

5. Aufgabe (4 Punkte)

Analysieren Sie die Zeitkomplexität folgender Funktion aus der Vorlesungsfolien:

```
binSearch :: (Ord a) => a -> [a] -> Bool
binSearch b [] = False
binSearch b [x] = b==x
binSearch b xs | b<mitte   = binSearch b links
               | b>mitte   = binSearch b rechts
               | b==mitte = True
                            where
                             half = (length xs) `div` 2
                             (links, rechts) = (take half xs, drop half xs)
                             mitte = head rechts
```

Wichtige Hinweise:

1. Schreiben Sie in alle Funktionen die entsprechende Signatur.
2. Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Semantik der Funktionen wiedergeben.
3. Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.
4. Kommentieren Sie Ihre Programme.
5. Verwenden Sie geeignete lokale Funktionen und Hilfsfunktionen in Ihren Funktionsdefinitionen.