3. Aufgabenblatt zu Funktionale Programmierung vom 30.10.2006. Fällig: 07.11.2006 / 14.11.2006 (jeweils 15:00 Uhr)

Themen: *Funktionen auf Zeichenreihen, ganzen Zahlen und Listen von Listen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe3.hs` ablegen. Wie für die Lösung zum ersten Aufgabenblatt sollen Sie dieses Mal also wieder ein "gewöhnliches" Haskell-Skript schreiben. Versehen Sie wieder wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten. Im einzelnen sollen Sie die im folgenden beschriebenen Problemstellungen bearbeiten.

Denken Sie bei der Kommentierung daran, dass Ihre Lösungen auch Grundlage für das Abgabegespräch am Semesterende sind.

1. In dieser Aufgabe beschäftigen wir uns mit b-adischen Zahlensystemen und der Konvertierung b-adischer Zahlen. Den Schlüssel dazu liefert der folgende Satz: Für $b > 1$, $b \in \mathbb{N}$, läßt sich jede Zahl $x \in \mathbb{N}$ auf genau eine Weise in der Form

$$x = \sum_{i=0}^{n} d_i * b^i$$

darstellen, wobei $n \in \mathbb{N} \cup \{0\}$, $0 \leq d_i < b$, $d_n \neq 0$. Für $b = 10$ erhalten wir den Spezialfall der Dezimalzahlen, für $b = 2$ den der Binärzahlen und für $b = 16$ den der Hexadezimalzahlen. In der Folge betrachten wir den Ziffernvorrat $\{0, 1, 2, \ldots, 9, A, B, C, D, E, F\}$. Mit diesem Ziffernvorrat können wir b-adische Zahlen für Basen b mit $1 < b \leq 16$ darstellen.

Um auch negative Zahlen darstellen zu können, vereinbaren wir, dass negative Zahlen durch ein vorangestelltes "-" Zeichen ausgezeichnet werden. Im Zehnersystem etwa stellen wir die Zahl -17 durch die Zeichenreihe `-17` dar. Auch für negative Zahlen sind keine führenden Nullen zulässig. Eine Ausnahme machen wir lediglich für die Zahl 0, die wir durch die Zeichenreihe `0` darstellen. `-0` ist nicht zulässig, ebensowenig wie `-0017`. In gleicher Weise treffen wir diese Vereinbarungen auch für die anderen b-adischen Zahlensysteme. Auf diese Weise haben wir für jedes b-adische Zahlensystem mit $1 < b \leq 16$ für jede ganze Zahl eine eindeutige Darstellung ohne führende Nullen.

Schreiben Sie eine Haskell-Rechenvorschrift `convert` zur Konversion einer b′-adischen Zahl in eine b″-adische Zahl mit $1 <$ b′,b″ ≤ 16. Dabei repräsentieren wir b-adische Zahlen als Zeichenreihen über dem Ziffernvorrat des jeweiligen b-adischen Systems. Im Hexadezimalsystem ist das der Ziffernvorrat $\{0, 1, 2, \ldots, 9, A, B, C, D, E, F\}$. Die Funktion `convert` hat also die Signatur `convert :: (String,Int) -> Int -> String`. Dabei bezeichnet die erste Komponente des ersten Arguments die zu konvertierende Zahl, die zweite Komponente des ersten Arguments die Basis des b′-adischen Systems, in dem die erste Komponente des ersten Arguments vorliegt, und das zweite Argument die Basis des b″-adischen Systems, in das die erste Komponente des ersten Arguments konvertiert werden soll. Das Ein-/Ausgabeverhalten ist dann wie folgt festgelegt: Haben die beiden Zahlargumente einen Wert zwischen 2 und 16 und ist das Zeichenreihenargument gültige Repräsentation einer Zahl des durch das erste Zahlargument spezifizierten b′-adischen Systems, so wird dieses Argument in die Darstellung des durch das zweite Zahlargument bestimmten b″-adischen Systems konvertiert; ansonsten ist das Ergebnis die Zeichenreihe `Ungueltige Eingabe`.

2. In dieser Aufgabe beschäftigen wir uns mit der Darstellung und Konvertierung von Zahlen im (-3)-adischen Zahlsystem. Den Schlüssel dazu liefert uns der folgende Satz: Jede ganze Zahl $z \in \mathbb{Z} \setminus \{0\}$ läßt sich auf genau eine Weise in der Form

$$z = \sum_{i=0}^{n} d_i * (-3)^i$$

darstellen, wobei $n \in \mathbb{N} \cup \{0\}$, $d_i \in \{0, 1, 2\}$, $d_n \neq 0$. Mit der Zusatzvereinbarung die Zahl 0 auch im (-3)-adischen System durch 0 darzustellen, besitzt jede ganze Zahl eine eindeutige Darstellung ohne führende Nullen im (-3)-adischen System (auch hier mit Ausnahme der 0 selbst). So entspricht z.B. die Dezimalzahl 3 der (-3)-adischen Zahl 120, die Dezimalzahl -5 der (-3)-adischen Zahl 21.

Schreiben Sie eine Haskell-Rechenvorschrift `convMin3` mit der Signatur `convMin3 :: String -> Int -> String`, die eine (-3)-adische in eine b-adische Zahl mit $1 < b \leq 16$ konvertiert. Dabei gibt das erste Argument die zu konvertierende Zahl und das zweite Argument die Basis des b-adischen Systems, in das das erste Argument konvertiert werden soll. Das Ein-/Ausgabeverhalten ist dann wie folgt festgelegt: Hat das zweite Argument einen Wert zwischen 2 und 16 und ist das erste Argument gültige Repräsentation einer (-3)-adischen Zahl, so wird das Argument in die Darstellung des durch das zweite Argument bestimmten b-adischen Systems konvertiert; ansonsten ist das Ergebnis die Zeichenreihe `Ungueltige Eingabe`.

3. Für die Lösung vieler Probleme ist es nötig zu überprüfen, ob eine vorgegebene Menge von Elementen Duplikate enthält. In dieser Aufgabe wollen wir diese Frage für Listen ganzer Zahlen entscheiden. Schreiben Sie also eine Wahrheitswertfunktion `containsDuplicates :: [Int] -> Bool`, die angewendet auf eine Argumentliste `lst` den Wahrheitswert `True` liefert, wenn `lst` Duplikate enthält, `False` sonst. Überlegen Sie sich, dass sich die Frage nach dem Vorhandensein von Duplikaten (vergleichsweise) einfach beantworten lässt, wenn die Argumentliste sortiert vorliegt.

4. Ein Graph G ist ein Paar (N, E), wobei N eine Menge von Knoten und E eine Menge von Kanten bezeichnet mit $E \subseteq N \times N$. Ein Graph G heißt *ungerichtet*, wenn aus $(m, n) \in E$ stets folgt, dass auch $(n, m) \in E$ gilt.

   Für viele Anwendungen sind als Darstellungen von Graphen sog. Adjazenzmatrizen besonders geeignet. Adjazenzmatrizen sind quadratische Matrizen über der Menge der Knoten eines Graphen. Der Eintrag 1 an der Position (m, n) gibt dabei an, dass $(m, n) \in E$ ist, der Eintrag 0 an der entsprechenden Stelle, dass $(m, n) \notin E$ ist.

   In Haskell lässt sich dies wie folgt realisieren:

   ```
   type Graph = [[Int]]
   ```

   Die k-te Liste in einem Wert vom Typ `Graph` gibt dabei die k-te Zeile der intendierten Adjazenzmatrix an.

   Schreiben Sie eine Wahrheitswertfunktion `istUngerichtet` mit der Funktionalität `istUngerichtet :: Graph -> Bool`, die angewendet auf einen Graphen `g` den Wahrheitswert `True` liefert, falls `g` Repräsentation eines ungerichteten Graphen ist, ansonsten den Wahrheitswert `False`. Beachten Sie, dass (entsprechend unserer Vereinbarung) nur quadratische Matrizen über den Einträgen 0 und 1 Repräsentation von Graphen sein können. Ist die Argumentmatrix keine Repräsentation eines Graphen, so liefert die Funktion `istUngerichtet` ebenfalls den Wahrheitswert `False`.