4. Aufgabenblatt zu Funktionale Programmierung vom 30.10.2006. Fällig: 14.11.2006 / 21.11.2006 (jeweils 15:00 Uhr)

Themen: *Funktionen auf Zeichenreihen, ganzen Zahlen und Graphen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe4.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.

1. In dieser Aufgabe beschäftigen wir uns noch einmal mit der Darstellung und Konvertierung (-3)-adischer Zahlen, die wir auf Aufgabenblatt 3 eingeführt hatten. Insbesondere gelten für die Darstellung (-3)-adischer Zahlen die dort getroffenen Vereinbarungen.

   (a) Schreiben Sie eine Haskell-Rechenvorschrift `convToMin3` mit der Signatur `convToMin3 :: Int -> String`, die eine durch das Argument gegebene Dezimalzahl in ihre (-3)-adische Darstellung konvertiert.

   (b) Schreiben Sie eine Haskell-Rechenvorschrift `incMin3` mit der Signatur `incMin3 :: String -> String`, die eine durch das Argument gegebene (-3)-adische Zahl inkrementiert, also um 1 erhöht. Ist die Argumentzeichenreihe keine gültige Darstellung einer (-3)-adischen Zahl, liefert die Funktion `incMin3` als Resultat die Zeichenreihe `Ungueltige Eingabe`.

2. Für die Lösung vieler Probleme ist es hilfreich, die Elemente einer Menge entsprechend ihrer Häufigkeit zu sortieren. In dieser Aufgabe wollen wir eine solche Sortierung für Listen ganzer Zahlen vornehmen. Schreiben Sie dazu eine Haskell-Rechenvorschrift `haeufSort` mit der Signatur `haeufSort :: [Int] -> [Int]`. In der Resultatliste sollen dabei die Elemente entsprechend der Häufigkeit ihres Vorkommens abnehmend angeordnet sein. Treten verschiedene Elemente gleich häufig auf, so werden diese Elemente ebenfalls absteigend angeordnet. Angewendet auf die Argumentliste `[2,1,3,2,3,3]` soll die Funktion `haeufSort` das Resultat `[3,3,3,2,2,1]` liefern, angewendet auf die Argumentliste `[1,3,2,3,1,3,2,2,1,1]` das Resultat `[1,1,1,1,3,3,3,2,2,2]`

3. In dieser Aufgabe betrachten wir noch einmal die auf Aufgabenblatt 3 eingeführten ungerichteten Graphen und ihre Realisierung in Haskell mittels des Typsynonyms `Graph`:

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

   Unter dem *Grad* eines Knotens eines ungerichteten Graphen verstehen wir die Anzahl der von ihm ausgehenden Kanten. Schreiben Sie eine Haskell-Rechenvorschrift `minMaxGrad` mit der Funktionalität `minMaxGrad :: Graph -> (Int,Int)`, die angewendet auf die Repräsentation eines ungerichteten Graphen `g` ein Paar ganzer Zahlen (m, n) liefert, wobei n den minimalen und m den maximalen Grad eines Knoten in g angibt. Ist das Argument `g` nicht Repräsentation eines ungerichteten Graphen, so liefert die Funktion `minMaxGrad` das Resultatpaar $(-1, -1)$.