3. Aufgabenblatt zu Funktionale Programmierung vom 19.10.2010. Fällig: 26.10.2010 / 02.11.2010 (jeweils 15:00 Uhr)

Themen: *Funktionen auf ganzen Zahlen, Tupeln, Listen und Matrizen*

Für dieses Aufgabenblatt sollen Sie die zur Lösung der unten angegebenen Aufgabenstellungen zu entwickelnden Haskell-Rechenvorschriften in einer Datei namens `Aufgabe3.lhs` im home-Verzeichnis Ihres Gruppenaccounts auf der Maschine g0 ablegen. **Anders** als bei der Lösung zu den ersten beiden Aufgabenblättern sollen Sie dieses Mal also ein "**literate Script**" schreiben. Versehen Sie 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. Schreiben Sie eine Haskell-Rechenvorschrift `dreiNplusEins` mit der Signatur `dreiNplusEins :: Integer -> [Integer]`. Die Funktion `dreiNplusEins` angewendet auf ein positives Argument n, $n > 0$, konstruiert die Elemente der Ergebnisliste in folgender Weise: Ist n gerade, wird n durch 2 geteilt; ist n ungerade, so wird n mit 3 multipliziert und auf dieses Zwischenresultat 1 aufaddiert. Dieser Prozess wird so lange wiederholt, bis die Funktion `dreiNplusEins` auf 1 angewendet wird. Angewendet auf das Argument 22 liefert die Funktion `dreiNplusEins` die Resultatliste $[22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]$. Die Länge dieser Resultatliste nennen wir die *Zykluslänge von n*. Die Zykluslänge von 22 ist also 16.

   Bemerkung: Man vermutet, dass obiges Verfahren stets mit 1 endet; ein Beweis dafür ist noch nicht gefunden worden. Für Argumente kleiner oder gleich 1 Million ist Terminierung jedenfalls sichergestellt

2. Gegeben seien folgende Typsynonyme:

   ```
   type UntereGrenze = Integer
   type ObereGrenze  = Integer
   type MaxZykLaenge = Integer
   ```

   Schreiben Sie eine Haskell-Rechenvorschrift `maxZyklus` mit der Signatur `maxZyklus :: UntereGrenze -> ObereGrenze -> (UntereGrenze,ObereGrenze,MaxZykLaenge)`. Angewendet auf zwei Argumente $0 < m, n < 1.000.000$, bestimmt die Funktion `maxZyklus` die maximale Zykluslänge für alle p aus dem Intervall von m bis n einschließlich der Intervallgrenzen m und n. Ist das Intervall leer, d.h. $m > n$, so ist die maximale Zykluslänge 0.

3. Schreiben Sie eine Haskell-Rechenvorschrift `anzNachbarn` mit der Signatur `anzNachbarn :: [[Bool]] -> (Integer,Integer) -> Integer`, die angewendet auf eine $M \times N$-Wahrheitswertmatrix und ein Indexpaar (m, n), das auf eine Komponente innerhalb dieser Matrix zeigt, die Anzahl der mit `True` benannten Nachbarfelder angibt. Dabei hat eine Komponente bis zu 8 Nachbarn (oben, unten, rechts, links, diagonal). Zeigt das Paar nicht auf eine Komponente innerhalb Matrix, so ist das Ergebnis der Funktionsanwendung -1. Angewendet auf die 3×3-Matrix `[[True,False,False],[True,False,False], [False,True,False]]` und $(0, 1)$, liefert die Funktion `anzNachbarn` das Resultat 2, angewendet auf die selbe Matrix und das Paar $(5, -2)$ das Resultat -1. (Die Komponentenlisten beschreiben Matrixzeilen, nicht Matrixspalten.)

4. Schreiben Sie eine Haskell-Rechenvorschrift `transform` mit der Signatur `transform :: [[Bool]] -> [[Integer]]`, die angewendet auf eine $M \times N$-Wahrheitswertmatrix eine $M \times N$-Matrix ganzer Zahlen ausgibt, wobei jede Komponente der Ergebnismatrix die Anzahl der mit `True` benannten Nachbarfelder der entsprechenden Komponente der Argumentmatrix angibt. Angewendet auf `[[True,False,False],[True,False,False],[False,True,False]]` liefert die Funktion `transform` die Ergebnismatrix `[[1,2,0],[2,3,1],[2,1,1]]`.

Sie können davon ausgehen, dass alle Funktionen nur mit "passenden" Argumenten aufgerufen werden, insbesondere mit Matrizen in Form von Listen von Listen, bei denen alle Zeilen von gleicher Länge sind.

Denken Sie bitte daran, dass Sie für die Lösung dieses Aufgabenblatts ein "literate" Haskell-Skript schreiben sollen!

Haskell Live

Am Freitag, den 22.10.2010, werden wir uns in *Haskell Live* u.a. mit der Aufgabe *"Krypto Kracker!"* beschäftigen.

Krypto Kracker!

Eine ebenso populäre wie einfache und unsichere Methode zur Verschlüsselung von Texten besteht darin, eine Permutation des Alphabets zu verwenden. Bei dieser Methode wird jeder Buchstabe des Alphabets einheitlich durch einen anderen Buchstaben ersetzt, wobei keine zwei Buchstaben durch denselben Buchstaben ersetzt werden. Das stellt sicher, dass verschlüsselte Texte auch wieder eindeutig entschlüsselt werden können.

Eine Standardmethode zur Entschlüsselung nach obiger Methode verschlüsselter Texte ist als "blanker Textangriff" bekannt. Diese Angriffsmethode beruht darauf, dass der Angreifer den Klartext einer Textphrase kennt, von der er weiß, dass sie in verschlüsselter Form im Geheimtext vorkommt. Durch den Vergleich von Klartext- und verschlüsselter Phrase wird auf die Verschlüsselung geschlossen, d.h. auf die verwendete Permutation des Alphabets. In unserem Fall wissen wir, dass der Geheimtext die Verschlüsselung der Klartextphrase

`the quick brown fox jumps over the lazy dog`

enthält.

Ihre Aufgabe ist nun, eine Liste von Geheimtextphrasen, von denen eine die obige Klartextphrase codiert, zu entschlüsseln und die entsprechenden Klartextphrasen auszugeben. Kommt mehr als eine Geheimtextphrase als Verschlüsselung obiger Klartextphrase in Frage, geben Sie alle möglichen Entschlüsselungen der Geheimtextphrasen an. Im Geheimtext kommen dabei neben Leerzeichen ausschließlich Kleinbuchstaben vor, also weder Ziffern noch sonstige Sonderzeichen.

Schreiben Sie ein Programm in Haskell oder in irgendeiner anderen Programmiersprache ihrer Wahl, das diese Entschlüsselung für eine Liste von Geheimtextphrasen vornimmt.

Angewendet auf den aus drei Geheimtextphrasen bestehenden Geheimtext (der in Form einer Haskell-Liste von Zeichenreihen vorliegt)

```
["vtz ud xnm xugm itr pyy jttk gmv xt otgm xt xnm puk ti xnm fprxq",
 "xnm ceuob lrtzv ita hegfd tsmr xnm ypwq ktj",
 "frtjrpgguvj otvxmdxd prm iev prmvx xnmq"]
```

sollte Ihre Entschlüsselungsfunktion folgende Klartextphrasen liefern (ebenfalls wieder in Form einer Haskell-Liste von Zeichenreihen):

```
["now is the time for all good men to come to the aid of the party",
 "the quick brown fox jumps over the lazy dog",
 "programming contests are fun arent they"]
```