3. Aufgabenblatt zu Funktionale Programmierung vom 21.10.2008. Fällig: 28.10.2008 / 04.11.2008 (jeweils 15:00 Uhr)

Themen: *Funktionen auf Zeichen, 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` im home-Verzeichnis Ihres Gruppenaccounts 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 mit einer funktionalen Umsetzung der unter dem Namen "Türme von Hanoi" bekannten Sage. Schreiben Sie dazu eine Haskell-Rechenvorschrift `tvh` mit der Signatur `tvh :: Int -> Char -> Char -> Char -> [(Char,Char)]`. Das erste Argument gibt die Anzahl der zu verlegenden Scheiben an, das zweite Argument den Namen des Ausgangsstapels, das dritte Argument den Namen des Zielstapels und das vierte Argument den Namen des Hilfsstapels. Angewendet auf Argumente n, a, b und c, liefert die Funktion `tvh` als Resultat die Scheibenbewegungen, die erforderlich sind, um n Scheiben vom Stapel a unter Verwendung des Hilfsstapels c bei kleinstmöglicher Zahl von Scheibenbewegungen auf den Stapel b zu versetzen. Jede Bewegung ist dabei in der Resultatliste als ein Paar von Zeichen (d, e) codiert mit der Bedeutung, dass die aktuell oberste Scheibe vom Stapel d entfernt und zuoberst auf dem Stapel e abgelegt wird. Das i-te Element in der Resultatliste beschreibt dabei die i-te Scheibenbewegung. Wir gehen davon aus, dass die Scheiben auf dem Ausgangsstapel der Größe nach abnehmend gestapelt sind. Während des Umstapelns darf zu keinem Zeitpunkt auf einem der drei Stapel eine größere auf einer kleineren Scheibe zu liegen kommen. Soll eine nichtpositive Anzahl von Scheiben verlegt werden oder ist einer der Stapelnamen verschieden von `'A'`, `'B'` und `'C'` oder taucht mehrfach als Argument auf, so liefert die Funktion `tvh` die leere Liste als Resultat.

2. Ein gerichteter 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 Element $(m, n) \in E$ bezeichnet dann eine von m nach n gerichtete Kante in G.

   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 am Schnittpunkt der m-ten Zeile und der n-ten Spalte 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; die Zahl der Zeilen gibt die Anzahl der Knoten des Graphen an.

   Schreiben Sie eine Haskell-Rechenvorschrift `erreichbarVon` mit der Signatur `erreichbarVon :: Graph -> Int -> [Int]`, die angewendet auf einen Graphen `g` und einen Knoten `m` aus `g` alle von `m` aus in `g` über eine oder mehrere gerichtete Kanten, d.h. über eine nichtleere Folge von Kanten, erreichbare Knoten bestimmt. Die Resultatliste soll dabei absteigend geordnet sein. 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 oder ist `m` kein Knoten von `g`, d.h. ist die Bedingung $1 \leq m \leq dim(g)$ verletzt mit $dim(g)$ Zahl der Zeilen der `g` repräsentierenden Matrix, so soll als Resultatwert die leere Liste geliefert werden.

3. In dieser Teilaufgabe betrachten wir noch einmal gerichtete Graphen und dieselbe Datenstruktur wie in der vorigen Teilaufgabe zu ihrer Implementierung, sowie folgende Typaliase:

```
type Knoten    = Int
type AnzKanten = Int
```

   Schreiben Sie eine Wahrheitswertfunktion `istErreichbar` mit der Signatur `istErreichbar :: Graph -> Knoten -> Knoten -> AnzKanten -> Bool`, die angewendet auf einen Graphen `g`, zwei Knoten `m` und `n` aus `g` und eine ganze Zahl j den Wahrheitswert `True` liefert, falls `n` in `g` von `m` aus über eine gerichtete nichtleere Kantenfolge mit höchstens j Kanten zu erreichen ist, ansonsten den Wahrheitswert `False`. Beachten Sie wieder, 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 oder sind `m` oder `n` keine Knoten von `g`, d.h. ist die Bedingung $1 \leq m \leq n \leq dim(g)$ verletzt mit $dim(g)$ Zahl der Zeilen der `g` repräsentierenden Matrix, so ist der Resultatwert ebenfalls `False`.

4. Eine Zeichenreihe `s` heißt Palindrom, wenn `s` sich in gleicher Weise von links nach rechts wie von rechts nach links liest. So ist z.B. die Zeichenreihe `otto` ein Palindrom der Länge 4 (die Zeichenreihe `Otto` ist kein Palindrom!). Schreiben Sie eine Haskell-Rechenvorschrift `allMaxPals` mit der Signatur `allMaxPals :: String -> [String]`, die angesetzt auf eine Zeichenreihe `t` alle Teilzeichenreihen maximaler Länge von `t` bestimmt, die ein Palindrom sind. Je weiter links eine solche Teilzeichenreihe in `t` vorkommt, desto kleiner soll ihre Position in der Ergebnisliste sein.