7. Aufgabenblatt zu Funktionale Programmierung vom 02.12.2015.
Fällig: 09.12.2015 / 16.12.2015 (jeweils 15:00 Uhr)

Themen: *Aufgaben auf Graphen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe7.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also 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. Wir betrachten noch einmal die Datenstruktur zur Darstellung ungerichteter Graphen $G = (N, E)$ mit Knotenmenge N und Kantenmenge $E \subseteq N \times N$ aus Aufgabenteil 4 von Aufgabenblatt 6 sowie die Datentypen für Zeichen, Ziffern und natürliche Zahlen von Aufgabenblatt 5 und 6 zusammen mit den dort getroffenen Festlegungen:

```
type Node     = Integer
type Edge     = (Node,Node)
type Source   = Node
type Sink     = Node
newtype Graph = Gph [(Source,[Sink])] deriving (Eq,Show)

data Zeichen  = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P
                | Q | R | S | T | U | V | W | X | Y | Z
                deriving (Eq,Ord,Show,Read,Enum,Bounded)
                       -- wie auf Aufgabenblatt 6
data Ziffer   = Null | Eins | Zwei | Drei | Vier | Fuenf | Sechs | Sieben
                | Acht | Neun deriving (Eq,Ord,Show,Read,Enum,Bounded)
                      -- wie auf Aufgabenblatt 6
newtype Nat   = Nat ((Zeichen,Zeichen,Zeichen),(Ziffer,Ziffer,Ziffer))
                        deriving (Eq,Ord,Show) -- wie auf Aufgabenblatt 5
```

   Sei $G = (N, E)$ ein Graph, $k \in IN$ eine natürliche Zahl und $M \subseteq N$ eine Teilmenge der Knotenmenge N von G. M heißt *k-unabhängig*, wenn M unabhängig ist und die Mächtigkeit k hat.

   Schreiben Sie eine Haskell-Rechenvorschrift

   (a) `kIndependent :: Graph -> Nat -> [[Node]]`

   Angewendet auf einen Graphen G und eine natürliche Zahl k bestimmt die Funktion `kIndependent` alle Teilmengen der Knotenmenge von G, die k-unabhängig sind. Jede Knotenliste in der Resultatliste soll dabei aufsteigend sortiert sein. Gibt es mehr als eine k-unabhängige Teilmenge von Knoten, sollen die zugehörigen Knotenlisten in der Resultatliste lexikographisch aufsteigend angeordnet sein. Gibt es keine k-unabhängige Teilmenge, liefert die Funktion die leere Liste als Resultat.

   Eine Folge $w = (e_1, e_2, \ldots, e_k)$ von Kanten eines Graphen $G = (N, E)$ heißt *Weg* vom Quellknoten der Kante e_1 zum Zielknoten der Kante e_k, wenn für alle $1 \leq i < k$ Ziel- und Quellknoten der Kanten e_i und e_{i+1} übereinstimmen. Ein Weg $w = (e_1, e_2, \ldots, e_k)$ heißt *Kreis*, wenn Quell- und Zielknoten der Kanten e_1 und e_k übereinstimmen. G heißt *zusammenhängend*, wenn je zwei Knoten m und n von G durch einen Weg von m nach n verbunden sind. G heißt *kreisfrei*, wenn es keine Kantenfolge von G gibt, die ein Kreis ist. G heißt *Baum*, wenn G zusammenhängend und kreisfrei ist.

   Schreiben Sie zwei Haskell-Wahrheitswertfunktionen

   (b) `isConnected :: Graph -> Bool`

   (c) `isTree :: Graph -> Bool`

   Angewendet auf einen Graphen G liefern die Wahrheitswertfunktionen `isConnected` und `isTree` den Wahrheitswert `True`, wenn G zusammenhängend bzw. ein Baum ist, ansonsten den Wahrheitswert $False$.

2. Wir betrachten Streckennetze verschiedener Fluglinien, die Verbindungen zwischen verschiedenen Flughäfen anbieten und Allianzen mit anderen Fluggesellschaften eingegangen sind.

```
data Airports   = AMS | AUC | DUS | FRA | HAJ | HAM | JFK | LAX | MUC | SFO
                | TXL | VIE deriving (Eq,Ord,Bounded,Enum,Show)
data Airlines   = Aeroflot | AirBerlin | AirFrance | AUA | BritishAirways |
                  Delta | GermanWings | KLM | Lufthansa | Swiss
                              deriving (Eq,Ord,Bounded,Enum,Show)
type Fare       = Nat  -- Ticketpreis einer Direktverbindung
type TotalFare  = Fare -- Gesamtticketpreis einer Umsteigeverbindung
type Networks   = Airlines -> Airports -> [(Airports,Fare)]
             -- Eine Abb. vom Typ Networks gibt fuer jede Fluglinie und jeden
             -- Flughafen an, welche Zielflughaefen sie von dort aus
             -- zu welchem Ticketpreis anfliegt. Taucht ein Paar aus
             -- Zielflughafen und Ticketpreis mehrfach in einer Liste auf,
             -- ist dies fuer diese Aufgabe redundant und bedeutungslos.
             -- Taucht ein Zielflughafen mit verschiedenen Ticketpreisen
             -- auf, so bietet diese Gesellschaft guenstigere und weniger
             -- guenstige Fluege auf dieser Strecke an. Eine Fluglinie kann
             -- auch einen Rundflug ab einem Flughafen anbieten. In diesem Fall
             -- ist der Startflughafen in der Liste der von dort erreichbaren
             -- Zielflughaefen enthalten.
type Alliances  = Airlines -> [Airlines]
             -- Eine Abb. vom Typ Alliances gibt fuer jede Fluglinie an, mit
             -- welchen Fluglinien sie sich in einer Allianz befindet.
             -- Mehrfachvorkommen einer Fluglinie in einer Allianzliste
             -- sind moeglich, aber fuer diese Aufgabe redundant und
             -- bedeutungslos. Der Einfachheit halber koennen Allianzen
             -- unsymmetrisch sein: Taucht Fluglinie B in der Allianzliste
             -- von Fluglinie A auf, muss dies umgekehrt nicht gelten.
type Origin      = Airports
type Destination = Airports
type Relation    = (Origin,Airlines,Destination)
             -- Ein Tripel vom Typ Relation gibt an, dass die angegebene
             -- Fluglinie einen Flug vom Start- zum Zielflughafen anbietet.
type Connection  = [Relation]
             -- Eine Liste vom Typ Connection beschreibt eine Umsteigeverbindung
             -- vom Startflughafen der ersten Relation zum Zielflughafen der
             -- letzten Relation.
```

Schreiben Sie Haskell-Rechenvorschriften zur Beantwortung der Frage, wie man mit den wenigsten Umstiegen vom Start- zum Zielflughafen kommt unter ausschließlicher Benutzung einer einzigen Airline (Teilaufgabe d)), wie man mit den wenigsten Umstiegen vom Start- zum Zielflughafen kommt unter ausschließlicher Benutzung einer einzigen Airline und der mit ihr in einer Allianz befindlichen Airlines (Teilaufgabe e)), wie man zum günstigsten Gesamtpreis vom Start- zum Zielflughafen kommt, wenn mit allen Airlines geflogen werden darf (Teilaufgabe f)). Gibt es keine Verbindung vom Start- zum Zielflughafen unter den jeweiligen Einschränkungen, ist die Ergebnisliste leer. Gibt es mehr als eine Verbindung, ist es egal, in welcher Reihenfolge die Verbindungen in der Ergebnisliste aufgeführt sind. Stimmen Start- und Zielflughafen überein, geht es um die Frage, wie man mit den wenigsten Zwischenstopps (möglicherweise ohne Zwischenstopp mit einem Rundflug) bzw. zum günstigsten Preis wieder zum Ausgangsflughafen zurückkommt.

d) `airlineConnections ::`
   `Origin -> Destination -> Networks -> Airlines -> [Connection]`

e) `allianceConnections ::`
   `Origin -> Destination -> Networks -> Alliances -> Airlines -> [Connection]`

f) `cheapestConnections ::`
   `Origin -> Destination -> Networks -> [(Connection,TotalFare)]`

Wichtig: Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen für dieses oder spätere Aufgabenblätter wieder verwenden möchten, so kopieren Sie diese in die neue Abgabedatei ein. Ein `import` schlägt für die Auswertung durch das Abgabeskript fehl, weil Ihre frühere Lösung, aus der importiert wird, nicht mit abgesammelt wird. Deshalb: Kopieren statt importieren zur Wiederwendung!

Haskell Live

Der nächste *Haskell Live*-Termin ist am Freitag, den 04.12.2015.