7. Übungsaufgabe zu Funktionale Programmierung
Themen: Graphen, binäre Bäume und Typklassen
ausgegeben: 10.12.2003, fällig: 17.12.2003

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe7.hs` in Ihrem Homeverzeichnis ablegen, wie immer auf oberstem Niveau. Kommentieren Sie Ihre Programme wieder aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

- Gerichtete Graphen können in Haskell als Listen von Paaren repräsentiert werden. Die erste Komponente eines Paares ist dabei eine Knotenmarkierung, die den markierten Knoten eindeutig identifiziert, die zweite Komponente eine Liste von Nachfolgerknoten des Knotens der ersten Komponente, die durch die Markierungen dieser Nachfolgerknoten gegeben ist. Als Knotenmarkierungen sollen dabei ausschließlich Typen aus der Klasse `Num` zugelassen sein.

  1. Schreiben Sie eine (polymorphe) Haskell-Rechenvorschrift `insert`, die ein Paar aus Knoten und Nachfolgerknoten als erstes Argument und einen (dazu typkompatiblen) Graphen als zweites Argument übernimmt und als Ergebnis den um das erste Argument vergrößerten Graphen zurückgibt.
  2. Schreiben Sie eine (polymorphe) Haskell-Rechenvorschrift `reachable`, die einen Knoten k (gegeben durch seine Markierung) und einen (dazu typkompatiblen) Graphen G übernimmt und als Ergebnis eine aufsteigend sortierte Liste aller von k aus in G erreichbaren Knoten (einschließlich k selbst) ausgibt. Kein Knoten soll in der Ausgabeliste mehrfach auftreten.
  3. Schreiben Sie eine (polymorphe) Haskell-Rechenvorschrift `isCyclic`, die überprüft, ob ein als Argument übergebener Graph einen Kreis enthält und einen entsprechenden Wahrheitswert ausgibt.
  4. Schreiben Sie eine (polymorphe) Haskell-Rechenvorschrift `isConnected`, die überprüft, ob ein als Argument übergebener Graph zusammenhängend ist und einen entsprechenden Wahrheitswert ausgibt. Ein Graph heißt *zusammenhängend*, wenn es zwischen je zwei Knoten einen Pfad gibt.

- In dieser Aufgabe betrachten wir eine Variante binärer Bäume, die durch den wie folgt definierten polymorphen Datentyp `Tree` gegeben ist.

```
data Tree a = Leave a |
              INode a (Tree a) (Tree a)
```

  - Schreiben Sie eine Haskell-Rechenvorschrift `leaves`, die angewandt auf einen Baum die Liste aller Blätter von links nach rechts ausgibt, d.h. der Wert des "linkesten" Blatts ist das erste Element in der Ergebnisliste, der Wert des "rechtesten" Blatts das letzte Element.
  - Schreiben Sie eine Haskell-Rechenvorschrift `nodesOnLevel`, die angewandt auf eine natürliche Zahl i als erstes und einen Baum t als zweites Argument die Liste aller Knoten der i-ten Schicht von t ausgibt. Wie bei der Funktion `leaves` soll dies von links nach rechts geschehen. Die i-te Schicht eines Baums ist dabei durch die Menge seiner Knoten mit Tiefe i gegeben. Die Wurzel des Baumes bildet somit die 0-te Schicht eines Baums.

- Führen Sie die polymorphe Typklasse `Sim` ein, die die boolesche Funktion `similar` unterstützt und Werte der Elemente der Typklasse auf Ähnlichkeit überprüft.

  Die in den ersten beiden Teilaufgaben betrachteten Graphen und Bäume sollen Instanzen der Typklasse `Sim` sein. Dabei sollen zwei Graphen ähnlich heißen, wenn ihre Knotenmengen übereinstimmen und die Kantenmenge des einen Graphen eine Teilmenge derjenigen des anderen ist. Zwei Bäume sollen ähnlich heißen, wenn es eine natürliche Zahl i gibt, so dass die Knoten der i-ten Schicht beider Bäume in Anzahl und Markierung (wieder von links nach rechts) übereinstimmen. Geben Sie entsprechende Implementierungen der Funktion `similar` für Graphen und Bäume an.