5. Aufgabenblatt zu Funktionale Programmierung vom 17.11.2009. Fällig: 24.11.2009 / 01.12.2009 (jeweils 15:00 Uhr)

Themen: *Funktionen auf algebraischen Datentypen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe5.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts 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.

Wir betrachten für dieses Aufgabenblatt wieder Binärbäume, die wir in ähnlicher Form schon auf dem letzten Aufgabenblatt kennengelernt haben:

```
data Tree a = Nil |
              Node a (Tree a) (Tree a) deriving (Eq,Ord,Show)
```

Unter der *Tiefe* eines Knotens k in einem Binärbaum verstehen wir die Anzahl der Vorgängerknoten von k bis zur Wurzel des Baums. Die Wurzel selbst hat dabei Tiefe 0. Die maximale Tiefe eines Baums B definieren wir als die maximale Tiefe eines seiner Knoten und bezeichnen sie mit max_B. Die Menge aller Knoten eines Binärsbaums B gleicher Tiefe t bezeichnen wir als die t-te *Schicht* von Knoten von B. Die "a"-Komponente bezeichnen wir in der Folge auch als Benennung eines Knoten.

1. Ein Baum heißt *Suchbaum*, wenn für alle Knoten `k` des Baums gilt, dass die Benennungen im linken Teilbaum, so vorhanden, echt kleiner und die im rechten Teilbaum, so vorhanden, echt größer als die Benennung von `k` sind. Schreiben Sie eine Wahrheitswertfunktion `isSearchTree` mit der Signatur `isSearchTree :: (Ord a, Show a) => (Tree a) -> Bool`, die überprüft, ob ein Argumentbaum ein Suchbaum ist.

2. Schreiben Sie eine Haskell-Rechenvorschrift `makeSearchTree` mit der Signatur `makeSearchTree :: (Ord a, Show a) => (Tree a) -> (Tree a)`, die angewendet auf einen beliebigen Baum einen Suchbaum als Ergebnis zurückliefert, in dem alle im Argumentbaum vorkommenden Knotenbenennungen (unter Streichung von Duplikaten) im Resultbaum vorkommen und jede im Resultbaum vorkommende Benennung mindestens einmal im Argumentbaum vorkommt (beachten Sie, dass der Resultbaum i.a. nicht eindeutig festgelegt ist).

3. Schreiben Sie eine Haskell-Rechenvorschrift `layerToSearchTree` mit der Signatur `layerToSearchTree :: (Eq a, Show a) => (Tree a) -> Int -> (Tree a)`, die angewendet auf einen Baum B und eine nichtnegative ganze Zahl t die Benennungen der t-ten Schicht von B in Form eines Suchbaums ausgibt. Durchläuft man den Resultatbaum in Infixordnung, sollen die Benennungen genau in der Reihenfolge gefunden werden wie sie in der t-ten Schicht von B von links nach rechts angeordnet sind. Wird ein negativer Wert für die gewünschte Schicht eingegeben oder ist die gewünschte Schicht größer als die maximale Tiefe des Baums oder enthält die t-te Schicht von B Benennungen mehr als einmal oder ist nicht aufsteigend sortiert, so wird als Resultat der leere Baum `Nil` zurückgeliefert (beachten Sie auch hier, dass der Resultatbaum i.a. nicht eindeutig festgelegt ist).

4. Sei B ein Binärbaum, seien $B_1, \ldots, B_k$ alle Teilbäume von B, die zugleich Suchbäume sind, seien $s_1, \ldots, s_k$ die Summe der Knotenbenennungen der Teilbäume $B_1, \ldots, B_k$, und sei max_B das Maximum der Werte $0, s_1, \ldots, s_k$. Schreiben Sie eine Haskell-Rechenvorschrift `maxSearchTreeSum` mit der Signatur `maxSearchTreeSum :: Num a => (Tree a) -> a`, die angewendet auf einen Binärbaum B den Wert max_B zurückliefert.