5. Aufgabenblatt zu Funktionale Programmierung vom 06.11.2007. Fällig: 13.11.2007 / 20.11.2007 (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 Haskell durch den schon vom letzten Aufgabenblatt bekannten folgenden algebraischen Haskell-Typ repräsentieren:

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

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 Integer-Komponente bezeichnen wir in der Folge auch als Benennung eines Knoten.

1. Schreiben Sie eine Haskell-Rechenvorschrift `schicht` mit der Signatur `schicht :: Tree -> Int -> [Int]`, die angewendet auf einen Baum B und eine nichtnegative ganze Zahl t die Benennungen der t-ten Schicht von B in Form einer Liste ganzer Zahlen ausgibt. Die Elemente dieser Liste sollen dabei genauso von links nach rechts angeordnet sein, wie in der t-ten Schicht von B. Wird ein negativer Wert für die gewünschte Schicht eingegeben oder ist die gewünschte Schicht größer als die maximal Tiefe des Baums, so wird als Resultat die leere Liste zurückgeliefert.

2. Schreiben Sie eine Haskell-Rechenvorschrift `abschneiden` mit der Signatur `abschneiden :: Tree -> Int -> Tree`. Angewendet auf einen Baum B und eine nichtnegative ganze Zahl t, liefert die Funktion `abschneiden` einen Baum zurück, der aus B dadurch entsteht, dass alle Knoten in Schichten s mit $s > t$ gestrichen sind. Bis einschließlich zur Schichttiefe t stimmt der Resultat- mit dem Argumentbaum in Struktur und Benennungen überein. Ist das ganzzahlige Argument negativ, so wird als Resultat der leere Baum zurückgeliefert.

3. Schreiben Sie eine Haskell-Rechenvorschrift `ausschneiden` mit der Signatur `ausschneiden :: Tree -> Int -> Int -> [[Int]]`. Angewendet auf einen Baum B und zwei ganzzahlige Argumente m und n liefert die Funktion `ausschneiden` folgendes Resultat: Für $m > n$ liefert die Funktion als Resultat die leere Liste zurück. Anderenfalls, d.h. für $m \leq n$ liefert die Funktion eine Liste von Listen als Resultat zurück, wobei die i-te Liste in der Resultatliste die Knotenwerte der i-ten Knotenschicht in derselben Reihenfolge enthält, wie sie im Argumentbaum vorkommen. Ist dabei m kleiner als 0 oder/und n größer als max_B, so wird die Resultatliste entsprechend mit leeren Listen aufgefüllt.

4. Schreiben Sie eine Haskell-Rechenvorschrift `summe` mit der Signatur `summe :: Tree -> Int -> Integer`. Angewendet auf einen Baum B und ein ganzzahliges Argument t berechnet die Funktion `summe` die Summe aller Knotenmarkierungen in B bis einschließlich zur t-ten Schicht. Ist t kleiner als 0, so ist das Resultat der Funktionsanwendung 0.