## Informatik A, WS 2016/17 — 15. Übungsblatt

Bonuspunkte, freiwillige Abgabe bis Freitag, 10. Februar 2017, 12:00 Uhr

87. Wörterbuchoperationen in einem binärem Suchbaum, 10 Bonuspunkte

    (a) Fügen Sie die Schlüssel I, N, F, O, R, M, A, T, K, Z in dieser Reihenfolge in einen anfangs leeren binären Suchbaum ein. Zeichnen Sie den Baum nach jedem Einfügevorgang.

    (b) Wenn wir nun nach U suchen, mit welchen Schlüsseln wird U dabei verglichen?

    (c) Löschen Sie die Schlüssel Z, O, N. Zeichnen Sie wieder den Baum nach jedem Löschvorgang.

    (d) Angenommen, wir suchen den Schlüssel 363 in einem binären Suchbaum. Bestimmen Sie für jede der folgenden Schlüsselfolgen, ob es einen Suchbaum gibt, wo sie als Folge der Vergleichsschlüssel auf dem Suchpfad nach 363 auftreten kann. Begründen Sie jeweils Ihre Antwort.

        i. 2, 399, 387, 219, 266, 382, 381, 278, 363.

        ii. 2, 252, 401, 398, 330, 344, 397, 363.

        iii. 935, 278, 347, 621, 299, 392, 358, 363.

        iv. 925, 202, 911, 240, 912, 245, 363.

        v. 924, 220, 911, 244, 898, 258, 362, 363.

88. Binäre Suchbäume, Programmieraufgabe, 5 Bonuspunkte

    In der Vorlesung wurde der algebraische Datentyp `Suchbaum` definiert. Implementieren Sie folgende Funktionen.

    (a) (5 Punkte) Bestimme die Höhe eines Binärbaums.

    ```
    höhe:: Suchbaum s w -> Int
    ```

    Ein Baum mit einem einzigen Knoten hat die Höhe 0, und der Baum `Nil` soll die Höhe $-1$ haben.

    (b) (0 Punkte) Teste, ob ein gegebener Binärbaum die Suchbaumeigenschaft hat.

    ```
    istSuchbaum:: Ord s => Suchbaum s w -> Bool
    ```

    Variante i. Verwenden Sie die Funktion `toList` aus der Vorlesung.

    Variante ii. Kommen Sie ohne `toList` oder eine vergleichbare Funktion aus.

89. Binärbäume, 5 Bonuspunkte

    Beweisen Sie mit vollständiger Induktion: Jeder binäre Suchbaum mit $n$ „echten Knoten“ (Datensätze mit dem Konstruktor `Knoten`) hat $n + 1$ `Nil`-Knoten.

90. Wörterbuchoperationen mit Listen, 0 Punkte

    Implementieren Sie die Wörterbuchoperationen durch Listen, analog zur Vorlesung.[1]

```
class Wörterbuch wb where
  einfügen :: ...
type ListenWörterbuch s w = [(s,w)]
instance Wörterbuch ListenWörterbuch where
  einfügen x wert ...
```

[1] `http://www.inf.fu-berlin.de/lehre/WS16/INFA/Woerterbuch.hs` zum Abspeichern (kodiert mit UTF-8) oder `Wooerterbuch.hs` zum Lesen im Browser.

91. Traversieren von binären Suchbäumen, Programmieraufgabe, 0 Punkte

Eine *Traversierung* eines binären Suchbaums $T$ ist eine Funktion, welche alle Knoten von $T$ systematisch besucht. Es gibt vier Arten der Traversierung:

- **Preorder**: Besuche erst die Wurzel, dann rekursiv den linken Teilbaum, gefolgt von dem rechten Teilbaum.
- **Inorder** (symmetrische Reihenfolge): Besuche erst rekursiv den linken Teilbaum, danach die Wurzel, dann den rechten Teilbaum.
- **Postorder**: Besuche erst rekursiv den linken Teilbaum, dann den rechten Teilbaum und schließlich die Wurzel.
- **Levelorder**: Besuche die Knoten nach aufsteigender Tiefe und alle Knoten mit der gleichen Tiefe von links nach rechts.

(a) Schreiben Sie Funktionen

```
preorder, inorder, postorder, levelorder:: Suchbaum s w -> [(s,w)]
```

die die Einträge eines Suchbaumes in der Reihenfolge, wie sie besucht werden, als Liste zurückgeben.

Versuchen Sie dabei, die `++`-Operation zu vermeiden und lineare Laufzeit zu erreichen.

(b) Die Preorder-Traversierung eines binären Suchbaums mit 10 Einträgen ergibt die folgende Schlüsselfolge: M B A F D P N O Z R. Die Postorder-Traversierung liefert A D F B O N R Z P M. Wie sieht der Baum aus?

92. Kodierung, 0 Punkte

(a) Welche der folgenden Binärcodes sind eindeutig dekodierbar? Welche sind präfixfrei? Warum?

$$C_1 = \{0110, 010, 0\}, \quad C_2 = \{010, 00, 001, 01\}, \quad C_3 = \{1, 001, 0100, 011\}.$$

(b) Konstruieren Sie für die folgenden Kodewortlängen einen binären präfixfreien Kode und zeichnen Sie den zugehörigen Kodebaum.

$$n_1 = 1,\ n_2 = 2,\ n_3 = 3,\ n_4 = n_5 = 5,\ n_6 = n_7 = n_8 = 6,\ n_9 = 7.$$

93. Kreuzsicherungscode, Programmieraufgabe, 10 Bonuspunkte

Der *Kreuzsicherungscode* wandelt ein 0-1-Wort der Länge $n^2$ in ein 0-1-Wort der Länge $n^2 + 2n$ um, indem er die Eingabe als $(n \times n)$-Matrix auffasst und für jede der $n$ Zeilen und $n$ Spalten jeweils ein Paritätsbit anhängt.

(a) Schreiben Sie eine Kodierungsfunktion `encode::Int -> [Int] -> [Int]` für den Kreuzsicherungscode. Dabei soll der erste Parameter die Dimension $n$ der Code-Matrix angeben, und der zweite Parameter ist eine 0-1-Folge, deren Länge ein Vielfaches von $n^2$ ist.

(b) Schreiben Sie einer entsprechende Decodierfunktion `decode::Int -> [Int] -> [Int]`, die einen Fehler korrigieren kann: Für jede 0-1-Folge $x$ passender Länge und jede Funktion `kippeEinBit::[Int] -> [Int]`, die die Identitätsfunktion ist oder die höchstens ein einzelnes Bit einer Bitfolge verändert, muss gelten:

```
(decode n . kippeEinBit . encode n) x = x
```

94. Alle verschieden, 0 Punkte

- Eingabe: Eine Liste `xs::Ord t => [t]`.
- Ausgabe: `True`, falls alle Elemente in `xs` paarweise verschieden sind.

Im folgenden sehen Sie drei Programme, die dieses Problem lösen sollen.[2] Welche Programme sind korrekt? Wenn ein Programm nicht korrekt ist, geben Sie ein Gegenbeispiel. Bewerten Sie von jedem Programm (egal ob es korrekt ist) die Effizienz, indem Sie die Anzahl von Vergleichen abschätzen, die es im schlimmsten Fall benötigt, in Abhängigkeit von der Listenlänge $n$.

```
-- erstes Programm
alle_verschieden_1::Ord t => [t] -> Bool
alle_verschieden_1 []  = True
alle_verschieden_1 [x] = True
alle_verschieden_1 (x1:x2:xs) | x1 /= x2  = alle_verschieden_1 (x2:xs)
                              | otherwise = False

-- zweites Programm
alle_verschieden_2::Ord t => [t] -> Bool
alle_verschieden_2 []     = True
alle_verschieden_2 (x:xs) = eindeutig && alle_verschieden_2 xs
  where eindeutig = filter (==x) xs == []

-- drittes Programm
alle_verschieden_3::Ord t => [t] -> Bool
alle_verschieden_3 = alle_verschieden_1 . sort

-- Sortieren durch Verschmelzen
sort :: Ord t => [t] -> [t]
sort [] = []
sort l@[_] = l -- Liste mit 1 Element
sort l = verschmelze (sort l1) (sort l2)
   where (l1,l2) = splitAt halb l
         halb = length l `div` 2
verschmelze :: Ord t => [t] -> [t] -> [t]
verschmelze l1 [] = l1
verschmelze [] l2 = l2
verschmelze l1@(x:xs) l2@(y:ys)  | x <= y    = x: verschmelze xs l2
                                 | otherwise = y: verschmelze l1 ys
```

95. Die Conway-Folge, Programmieraufgabe, 0 Punkte

Diese Folge entsteht, indem man in einer Zahlenfolge gleiche Zahlen zusammenfasst, in Gruppen laut vorliest und dann die ausgesprochenen Zahlen hintereinanderschreibt. Zum Beispiel würde man 33114555 als „zwei Dreien, zwei Einsen, eine Vier, drei Fünfen“ lesen und als 23 21 14 35 schreiben. Wenn man diese Operation wiederholt anwendet, erhält man immer längere Ziffernfolgen[3], zum Beispiel

$$1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, 31131211131221, \ldots$$

(a) Schreiben Sie eine Funktion `sprich:: [Int] -> [Int]` für diese Operation. Beispiel: `sprich [3,3,1,1,4,5] = [2,3,2,1,1,4,1,5]`.

(b) Schreiben Sie eine Funktion `istGültig:: [Int] -> Bool`, die feststellt, ob eine Zahlenfolge im Wertebereich der Funktion `sprich` ist.

[2] `http://www.inf.fu-berlin.de/lehre/WS16/INFA/alleverschieden.hs`
[3] `http://mathworld.wolfram.com/LookandSaySequence.html`