PHILIPPS-UNIVERSITÄT MARBURG D-35032 Marburg
Fachbereich Mathematik und Informatik Hans Meerwein Straße
Prof. Dr. R. Loogen Lahnberge

Übungen zur „Praktischen Informatik III“, WS 2005/06

Nr. 4, Abgabe: 22. November 2005 vor der Vorlesung

Importieren Sie das auf der Vorlesungsseite zur Verfügung gestellte Modul `SearchTree.hs` durch die Angabe von `import SearchTree` am Anfang Ihres Programms. Der folgende Auszug aus der Spezifikation zeigt, welche Operationen aus dem Modul exportiert werden. Nur diese können in den Aufgaben zur Baumverarbeitung verwendet werden.

```
module SearchTree
  (STree,
   nil,       -- STree a
   node,      -- a -> STree a -> STree a -> STree a
   isNil,     -- STree a -> Bool
   isNode,    -- STree a -> Bool
   leftSub,   -- STree a -> STree a
   rightSub,  -- STree a -> STree a
   rootVal,   -- STree a -> a
   insTree,   -- Ord a => a -> STree a -> STree a
   delete,    -- Ord a => a -> STree a -> STree a
   minTree,   -- Ord a => STree a -> a
   showTree   -- STree a -> IO ()
  )
where ...
```

A. Hausaufgaben

Die Lösungen sollten grundsätzlich schriftlich, Programme zusätzlich auf Diskette oder per E-Mail an Ihre Tutorin abgegeben werden. Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

15. Baum-Balancierung — 6 Punkte

Ein binärer Baum heißt *balanciert*, wenn sich die Anzahl der Knoten im linken und rechten Teilbaum um höchstens eins unterscheiden und wenn beide Teilbäume balanciert sind. Der leere Baum ist per definitionem balanciert.

(a) Definieren Sie eine Funktion `size :: STree a -> Int` zur Bestimmung der Anzahl der Knoten in einem Suchbaum. / 1

(b) Schreiben Sie eine Funktion `isBalanced :: STree a -> Bool` die testet, ob ein gegebenener Baum balanciert ist. / 2

(c) Entwickeln Sie eine Funktion `balance :: Ord a => STree a -> STree a` die einen beliebigen Suchbaum in einen balancierten Suchbaum mit denselben Einträgen umwandelt. / 3

16. Bäume mit Größenangabe — 4 Punkte

Zur effizienteren Bestimmung der Größe von Suchbäumen soll die Implementierung des `searchTree`-Moduls so abgeändert werden, dass in den Knoten neben dem Wert auch die Größe des entsprechenden Teilbaums gespeichert wird. Zur Implementierung von Suchbäumen soll die folgende Datenstruktur eingesetzt werden:

```
data STree a = Nil | Node a Int (STree a) (STree a)
```

Passen Sie die Implementierung des Moduls `SearchTree` an die geänderte Datenstrukturdeklaration an.

17. Listeninduktion — 2 Punkte

Beweisen Sie die folgenden Aussagen für beliebige endliche Listen `xs, xs1, xs2` über einem Elementtyp `a` mittels Listeninduktion:

(a) `xs ++ [] = xs`

(b) `reverse (xs1 ++ xs2) = (reverse xs2) ++ (reverse xs1)`

B. Mündliche Aufgaben

18. Allgemeine Baumverschmelzung

Zur Definition der `delete`-Funktion ist im Modul `SearchTree` eine spezielle `join`-Funktion definiert, bei der vorausgesetzt wird, dass alle Einträge im ersten Argumentbaum kleiner als alle Einträge des zweiten Argumentbaums sind. Definieren Sie eine allgemeine Funktion `join :: Ord a => STree a -> STree a -> STree a`, die beliebige Suchbäume zu einem Suchbaum verschmilzt.

19. Programmtransformation

Gegeben seien die folgenden Funktionen

```
my_unlines     :: [String] -> String
my_unlines css =  concat (addnewline ccs)

addnewline          :: [String] -> [String]
addnewline []       =  []
addnewline (cs:css) =  (cs ++ "\n") : addnewline ccs

concat          :: [[a]] -> [a]
concat []       =  []
concat (xs:xss) =  xs ++ concat xss
```

(a) Analysieren Sie den Zeitaufwand der Funktion `my_unlines`.

(b) Entwickeln Sie eine zu `my_unlines` äquivalente Funktion, die effizienter arbeitet, d.h. mit weniger Schritten dasselbe Ergebnis bestimmt.

(c) Zeigen Sie die Äquivalenz Ihrer Funktion zu `my_unlines`.