Fakultät IV – Elektrotechnik und Informatik
MPGI 1 WS 2013/14
Pepper / Höger / Reicherdt / Zuber / Tutoren

Aufgabenblatt 13

letzte Aktualisierung: 23. 01, 10:22

Ausgabe: 22.01.2014
Abgabe: 30./31.01.2014

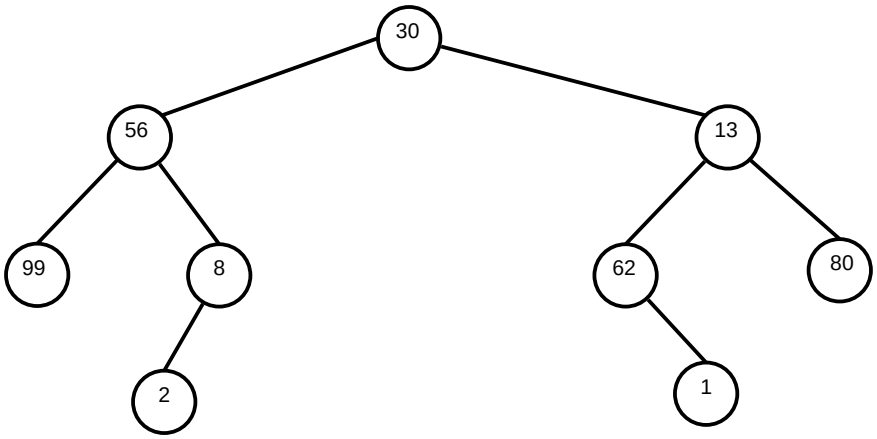
Thema: *Binärbäume, Suchbäume, Baumfunktionale*

Achtung: In diesem Aufgabenblatt erwarten wir von euch ein **Testmodul**, das Testfunktionen enthält. Es soll **alle** eure Module der Hausaufgabe importieren und deren Funktionen mit **geeigneten** Testwerten füttern und **nachvollziehbare** Ausgaben liefern.

1. Aufgabe: Einführung in Binärbäume

In dieser Aufgabe wollen wir grundlegende Begriffe für Binärbäume klären.

1.1. (Tut) Grundbegriffe Was bedeuten folgende Begriffe? Ihr könnt die Begriffe an folgendem Baum erklären:



- Wurzel
- Knoten
- Blatt
- Kante
- Pfad
- Höhe eines Baumes
- Tiefe eines Blattes
- Schicht
- Unterbaum

1.2. (Tut) Baumtraversierung Wir wollen nun den Baum traversieren, um eine Liste zu generieren. Dazu betrachten wir zunächst die Struktur eines Binärbaumes (`BinTree.sign`):

```
SIGNATURE BinTree[alpha]
SORT alpha
TYPE binTree == nil
                node(data: alpha, left: binTree, right: binTree)
```

Um nun aus einem Baum eine Sequenz zu erzeugen gibt es drei gängige Methoden, Preorder, Inorder und Postorder. Welche Sequenz würde man erhalten, wenn der oben dargestellte Baum einmal Preorder, einmal Inorder und einmal Postorder traversiert wird?

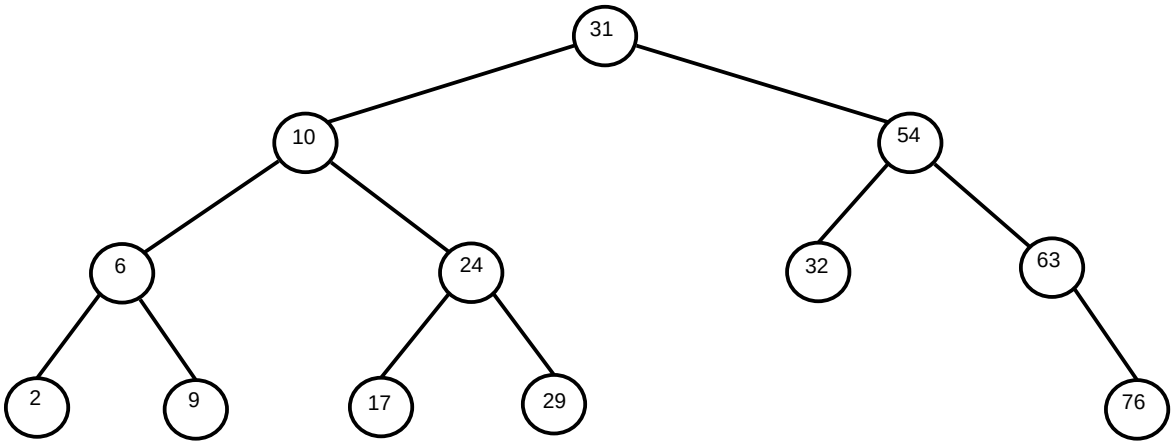
1.3. (Pflicht-Hausaufgabe) Baumtraversierung Erweitere die Struktur `BinTree` um die Funktionen `preorder` (Datum, links, rechts), `inorder` (links, Datum, rechts), `postorder` (links, rechts, Datum), welche die Daten eines Binärbaums zu einer Liste zusammenfassen.

Hinweis: In Den Vorgaben findet ihr die Struktur `PrintTree`, die eine gecurryte Backtick-Funktion für Bäume bereitstellt.

2. Aufgabe: Binäre Suchbäume

Als speziellen Binärbaum werden wir den binären Suchbaum betrachten. Er hat die Eigenschaft, dass alle Werte im linken Unterbaum kleiner, und alle Werte im rechten Unterbaum größer als der Wert im Elternknoten sind.

2.1. (Tut) Operationen auf binären Suchbäumen Gegeben sei der folgende Suchbaum mit Integer-Werten als Daten:



Demonstriert den Algorithmus der Suche mit dem Beispiel nach der Suche(10) und nochmals mit Suche(60).

Nun wollen wir in den Baum die 60 einfügen. Wie geht der Algorithmus vor?

Als dritten Algorithmus gibt es noch den Löschalgorithmus. Wie funktioniert er, wenn in unserem Suchbaum nacheinander die 32, die 54 und die 10 gelöscht werden sollen?

2.2. (Tut) find Erweitert die Struktur `BinSearchTree` um die Funktion `find`, welches ein übergebenes Element vom Typ `alpha` aus einem Suchbaum heraussucht. Die Rückgabe soll vom Typ `option[alpha]` sein.

Hinweis:
```
SIGNATURE Option [data]
          TYPE option  == nil
                          avail( cont: data )
```

2.3. (Tut) insert Erweitere die Struktur `BinSearchTree` um die Funktion `insert`. Diese soll ein übergebenes Element an der richtigen Stelle des Suchbaumes einfügen.

Hinweis: Zur Kontrolle könnt ihr wieder die Struktur `PrintTree` verwenden.

2.4. (Pflicht-Hausaufgabe) delete Erweitere die Struktur `BinSearchTree` um die Funktion `delete`. Diese soll ein übergebenes Element aus dem Suchbaum entfernen. Sollte das Element nicht im Baum enthalten sein soll diese Funktion keinen Effekt haben.

3. Aufgabe: Baumfunktionale

3.1. (Tut) Map und Reduce für Bäume Analog zu den Listenfunktionalen sollen `map` und `reduce` für Bäume implementiert werden. Legt für jedes Funktional eine eigene Struktur an.

Haltet euch an folgende Signaturen:

```
-- aus BinTreeMap.sign
FUN map: (alpha -> beta) -> (binTree[alpha] -> binTree[beta])

-- aus BinTreeReduce.sign
FUN reduce: (alpha**beta**beta -> beta) -> beta -> (binTree[alpha] -> beta)
```

3.2. (Pflicht-Hausaufgabe) Sortierte Studentenliste In dieser Aufgabe betrachten wir eine Studentendatenbank, die zur Speicherung der Studenten einen Suchbaum benutzt. Gegeben ist hierzu die Signatur der Struktur `StudentDB`:

StudentDB.sign:

```
SIGNATURE StudentDB

IMPORT Nat                          ONLY nat
IMPORT Real                         ONLY real
IMPORT BinSearchTree[student,<]     ONLY binTree[student]
IMPORT Seq[student]                 ONLY seq[student]

-- datatypes
TYPE student == student(matric: nat, name:denotation, surname: denotation, sex:se
TYPE sex == m f

-- order
FUN < : student ** student -> bool

FUN ‘ : student -> denotation
```

Erweitere die Struktur `StudentDB` um die Funktion `FUN studentList: binTree[student] -> se` Die Funktion soll die Studenten eines Suchbaumes als Liste in aufsteigender Reihenfolge nach Matrikelnummern sortiert zurückliefern. Nutzt dabei keine Rekursion, sondern Baumfunktionale. Welcher Traversierungsart entspricht dein Algorithmus?

Hinweis: Die Ordnungsrelation $<$ ist vorgegeben:

```
DEF student(id1,_,_) < student(id2,_,_) == id1 < id2
```