Universität Bremen

Algebraische Datentypen
Praktische Informatik 3 WS 2019/20

Thomas Barkowsky
Tobias Brandt
Yannic Block
Franklin K. Kenfack
Till Schlechtweg
M. Tarek Soliman

5. Übungsblatt

Ausgabe: 11.11.2019
Abgabe: 22.11.2019

In diesem Übungsblatt geht es um binäre Suchbäume. Das Ziel ist, die Funktionen für einen sortierten Suchbaum gemäß dem unten definierten Datentyp zu implementieren. In diesen Bäumen enthalten die Knoten Schlüssel (Werte), und die Schlüssel des linken Teilbaums (des linken Kindes) eines Knotens sind immer kleiner, die des rechten Teilbaums (des rechten Kindes) immer größer als der Schlüssel des jeweiligen Knotens. Das betrifft auch alle Teilbäume.

Es gibt unterschiedliche Variante binärer Suchbäume. In unserem Fall sollen Knoten nur sortiert hinzufügt werden, wobei bereits vorhandene Knoten nicht noch einmal hinzugefügt werden dürfen (keine Duplikate).

Unser Datentyp ist wie folgt definiert:

```
data Tree a = Node a (Tree a) (Tree a) | Leaf a | Empty
```

Hinweis: Ihr dürft bei der Implementierung einer Funktion die zuvor schon von euch implementierten Funktionen verwenden.

5.1 *isEmpty, isLeaf, isNode, makeUnhollow, leftChild, rightChild, nodeCount, min, max* *5 Punkte*

Implementiert die folgenden Funktionen:

```
isEmpty :: Tree a → Bool
```

isEmpty untersucht, ob ein Baum/Teilbaum leer ist, und gibt *True* zurück, wenn er leer ist.

```
isLeaf :: Tree a → Bool
```

isLeaf untersucht, ob ein Baum/Teilbaum ein Blatt ist (keine Kinder hat), und gibt in diesem Fall *True* zurück.

```
isNode :: Tree a → Bool
```

isNode untersucht, ob ein Baum/Teilbaum ein Knoten ist (Kinder hat), und gibt *True* zurück, wenn das der Fall ist.

```
makeUnhollow :: Tree a → Tree a
```

makeUnhollow wandelt einen Knoten, der keine Kinder hat, in ein Blatt um:

```
makeUnhollow (Node 5 Empty Empty)     ⇝ Leaf 5
makeUnhollow (Leaf 5)                 ⇝ Leaf 5
makeUnhollow (Node 5 (Leaf 4) Empty)  ⇝ (Node 5 (Leaf 4) Empty)
```

```
leftChild :: Node a → Maybe(Node a)
```

leftChild gibt das linke Kind als *Maybe*-Datatyp zurück, wenn es existiert (wenn es nicht *Empty* ist), sonst *Nothing*.

```
rightChild :: Node a → Maybe(Node a)
```

rightChild arbeitet analog zu *leftChild*

```
nodeCount :: Tree a → Int
```

nodeCount gibt die Anzahl der nicht-leeren Knoten (inklusive Blätter) zurück:

```
nodeCount Empty                    ⇝ 0
nodeCount (Leaf 5)                 ⇝ 1
nodeCount (Node 5 (Leaf 4) Empty)  ⇝ 2

min :: Tree a → Maybe a
```

gibt den Knoten/das Blatt mit dem kleinsten Schlüssel als *Maybe*-Datentyp zurück. Wenn der Baum leer ist, entsprechend *Nothing*.

```
max :: Tree a → Maybe a
```

funktioniert parallel zu *min*.

5.2 *insert, fulfillTree, filterTreeToList* — *3 Punkte*

Implementiert die folgenden Funktionen:

```
insert :: (Ord a) ⇒ Tree a → a → Tree a
```

insert fügt einen neuen Knoten hinzu. Wenn der Knoten bereits im Baum vorhanden ist, wird er nicht erneut hinzugefügt. Beachtet dabei, dass der Baum sortiert bleiben soll. Das heißt, dass jedes linke Kind einen kleineren Schlüssel als den Eltern-Knoten hat, und jedes rechtes Kind einen größeren.

```
fulfillTree :: (a → Bool) → Tree a → Bool
```

fulfillTree überprüft, ob für alle Knoten im Baum das angegebene Kriterium zutrifft:

```
fulfillTree (odd) (Node 5 Empty (Leaf 7))  ⇝ True
fulfillTree (≥6) (Node 5 Empty (Leaf 7))   ⇝ False
fulfillTree (==10) (Empty)                 ⇝ True

filterTreeToList :: (a → Bool) → Tree a → [a]
```

filterTreeToList gibt eine Liste aller Elemente des Baums, die das angegebene Kriterium erfüllen:

```
filterTreeToList (even) (Node 5 (Leaf 4) (Leaf 6))  ⇝ [4,6]
filterTreeToList (odd) (Node 5 (Leaf 4) (Leaf 6))   ⇝ [5]
filterTreeToList (>6) (Node 5 (Leaf 4) (Leaf 6))    ⇝ []
```

5.3 *countElem, hasDuplicates* — *2 Punkte*

Implementiert die folgenden Funktionen:

```
countElem :: (Eq a) ⇒ Tree a → a → Int
```

countElem gibt die Anzahl der Knoten/Blätter zurück, die den angegebenen Schlüssel haben:

```
countElem (Node 5 (Leaf 4) (Node 6 (Leaf 5) Empty)) 5 ⇝ 2

hasDuplicates :: (Eq a) ⇒ Tree a → Bool
```

hasDuplicates gibt *True* zurück, wenn der Baum Duplikate hat.

Wir stellen euch die Funktion printTree zur Verfügung, um Bäume in ASCII-Darstellung auf der Konsole anzuzeigen.