HASKELL 3: Kurze Programmieraufgaben zu Bäumen in Haskell

panitz

Zusammenfassung

Dieser Kurs besteht aus 10 kleinen Programmieraufgaben in der Programmiersprache Haskell. In diesem Kurs finden sich Aufgaben zu Funktionen auf einer allgemeinen Baumstruktur.

Frage: Größe des Baumes

Schreiben Sie eine Funktion, die die Elemente, die in einem Baum enthalten sind, zählt. Haskell

```haskell
      module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

groesse::Num t => Baum t1 -> t
groesse  _ = 0
```

Haskell

Lösung

Haskell

```haskell
        module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

groesse::Num t => Baum t1 -> t
groesse E = 0
groesse (B _ cs)
  = 1+(sum$map groesse cs)
```

Haskell

Erläuterung

Der leere Baum enthält kein Element. Ansonsten wird die Summe dder Kindgrößen berechnet und 1 hinzu addiert.

Frage: Test auf Element im Baum

Schreiben Sie eine Funktion, die testet, ob ein bestimmtes Element im Baum gespeichert ist. Haskell

```haskell
      module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

enthaelt::Eq t => t -> Baum t -> Bool
enthaelt _ _ = False
```

Haskell

Lösung

Haskell

```haskell
        module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

enthaelt::Eq t => t -> Baum t -> Bool
enthaelt _ E = False
enthaelt x (B e xs)
 |x==e  = True
 |otherwise = or$map (enthaelt x) xs
```

Haskell

Erläuterung

Der leere Baum enthält keine Elemente. Ansonsten wird das Wurzelelement verglichen. Ist das Element nihct das Wurzelelement, so wird in den Kindern rekursiv gesucht.

Frage: Maximale Baumtiefe

Schreiben Sie eine Funktion, die die maximale Baumtiefe Berechnet. Haskell

```haskell
      module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

maxDepth::(Num t, Ord t)=> Baum a -> t
maxDepth _ = 0
```

Haskell

Lösung

Haskell

```haskell
        module Baum where

data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

maxDepth::(Num t, Ord t)=> Baum a -> t
maxDepth E = 0
maxDepth (B _ xs)
 = 1+(foldl max 0$map maxDepth xs)
```

Haskell

Erläuterung

Der leere Baum hat die Tiefe 0. Ansonsten wird der Maximum der Baumtiefen der Kinder genommen und 1 hinzu addiert.

Frage: Maximale Kinderanzahl

Schreiben Sie eine Funktion, die die maximale Anzahl der Kinder der Knoten innerhalb eines Baumes berechnet. Haskell

```haskell
      module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

maxWidth :: Baum t -> Int
maxWidth _ = 0
```

Haskell

Lösung

Haskell

```haskell
        module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

maxWidth :: Baum t -> Int
maxWidth E = 0
maxWidth (B _ xs)
 = foldl max (length xs)
    $map maxWidth xs
```

Haskell

Erläuterung

Beim leeren Baum, gibt es nie Kinder, also ist die maximale Länge 0. Ansonsten wird das Maximum der rekursiven Aufrufe auf der Kinder genommen und mit der Anzahl der Kinder verglichen.

Frage: Alle Baumelemente

Schreiben Sie eine Funktion, die einer Liste aller im Baum gespeicherten Elemente selektiert. Die Elemente seien in Präorder sortiert. Haskell

```haskell
      module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

elemente :: Baum t -> [t]
elemente _ = []
```

Haskell

Lösung

Haskell

```haskell
        module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

elemente :: Baum t -> [t]
elemente E = []
elemente (B x xs)
 = x:(concat$map elemente xs)
```

Haskell

Erläuterung

Das Wurzelelement wird vorne an die Konkatination aller Kinderlemente angefügt.

Frage: Blattelemente eines Baumes

Haskell

```haskell
      module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

fringe :: Baum t -> [t]
fringe _ = []
```

Haskell

Lösung

Haskell

```haskell
        module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

fringe :: Baum t -> [t]
fringe E = []
fringe (B x []) = [x]
fringe (B _ xs)
 = concat$map fringe xs
```

Haskell

Erläuterung

Blattknoten ergeben einelementige Ergebnislisten. Ansonsten sind die Ergebnisse der rekursiven Aufrufe zu konkatenieren.

Frage: Baum abbilden

Screiben Sie eine Funktion, die einen neuen Baum erzeugt, indem ein Funktionsparameter auf jedes Element des Baumes angewendet wird.

Haskell

```haskell
      module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

abbilden :: (t -> a) -> Baum t -> Baum a
abbilden _ _ = E
```

Haskell

Lösung

Haskell

```haskell
        module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

abbilden :: (t -> a) -> Baum t -> Baum a
abbilden _ E = E
abbilden f (B x xs)
 = B (f x)$map (abbilden f) xs
```

Haskell

Erläuterung

Es handelt sich um die klassische Funktion `fmap` der Typklasse Funktor. Es wird die Funktion `f` auf das Wurzelelement angewendet und die gesamte Funktion rekursiv auf alle Kinder angewendet.

Frage: Kinderlisten Splitten

Schreiben Sie eine Funktion, die einem Baum so umwandelt, dass er keinen Knoten mit mehr als 5 Kindern mehr enthält. Hierzu sind bei einem Knoten mit mehr als 5 Kindern, die Kinder 2 bis 6 dem ersten Kind als neue Kinder vorne anzustellen. Sie werden also zu Enkelkindern. Dieses ist rekursiv so lange durchzuführen, bis kein Knoten mehr mehr als 5 Kinder hat. Haskell

```haskell
      module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

splitten :: Baum a -> Baum a
splitten _ = E
```

Haskell

Lösung

Haskell

```haskell
        module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

splitten :: Baum a -> Baum a
splitten E = E
splitten (B x ((B y ys):c2:c3:c4:c5:c6:cs))
 = splitten
     (B x (B y (ys++[c2,c3,c4,c5,c6]):cs))
splitten (B x cs)
 = B x (map splitten cs)
```

Haskell

Erläuterung

Per *Pattern Matching* werden Knoten mit mehr als 5 Kindern identifiziert und transformiert. Anschließend wird di Rekursion auf den transformierten Baum ausgeführt. Hat der Knoten maximal 5 Kinder, so wird rekursiv jedes Kind traversiert.

Frage: Längster Pfad

Schreiben Sie eine Funktion, die die Liste der Elemente auf dem längsten Pfad im Baum berechnet. Gibt es mehrere gleichlange Pfade ist der linkeste zu nehmen. Haskell

```haskell
      module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

longestPath :: Baum a -> [a]
logestPath _ = []
```

Haskell

Lösung

Haskell

```haskell
        module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

longestPath :: Baum a -> [a]
longestPath E = []
longestPath (B x []) = [x]
longestPath (B x cs)
  = x:(foldl
        (\xs ys->if length xs>=length ys
                 then xs
                 else ys)
        []
        $map longestPath cs)
```

Haskell

Erläuterung

Es wird der längste Pfad der Kinder selektiert und auf diesen das Wurzelelement noch vorne dran gehängt.

Frage: Faltung auf Bäumen

Schreiben Sie eine Funktion, die alle Elemente eines Baumes mit Hilfe einer Operatorfunktion in Präorder auf ein Startelement aufrechnet. Haskell

```haskell
      module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

faltung:: (b->a->b) -> b -> Baum a -> b
faltung f start _ = start
```

Haskell

Lösung

Haskell

```haskell
        module Baum where
data Baum a = B a [Baum a]|E
  deriving (Show,Eq)

faltung:: (b->a->b) -> b -> Baum a -> b
faltung f start E = start
faltung f start (B x cs)
  = foldl
      (\s c-> faltung f s c)
      (f start x)
      cs
```

Haskell

Erläuterung

Zunächst wird das Wurzelelement mit dem Startelement verknüpft. das Ergbnis wird für die Standardfaltung auf Listen über das Ergebnis der rekursiven Kindfaltungen als neues Startelement verwendet.