Informatik I - Gruppe 1 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

Aufgabe T6.1: Rekursion (3 Punkte)

Gegeben sei eine Funktion f, definiert durch:

$$f(n) = \begin{cases} 2 \cdot f(n-1) + 1 & \text{für } n \geq 1 \\ 1 & \text{sonst} \end{cases}$$

(a) Geben Sie eine Wertetabelle an, die für $n = 0, 1, 2, 3, 4, 5, 6$ das Ergebnis $f(n)$ angibt.

n	0	1	2	3	4	5	6
$f(n)$	1	3	7	15	31	63	127

(b) Geben Sie den schrittweisen Ablauf der rekursiven Funktion als Expansion und Kontraktion für den Wert $n = 4$ an.

$$\begin{aligned} f(4) &= 2 \cdot f(3) + 1 \\ &= 2 \cdot (2 \cdot f(2) + 1) + 1 \\ &= 2 \cdot (2 \cdot (2 \cdot f(1) + 1) + 1) + 1 \\ &= 2 \cdot (2 \cdot (2 \cdot (2 \cdot f(0) + 1) + 1) + 1) + 1 \\ &= 2 \cdot (2 \cdot (2 \cdot (2 \cdot 1 + 1) + 1) + 1) + 1 \\ &= 2 \cdot (2 \cdot (2 \cdot (2 + 1) + 1) + 1) + 1 \\ &= 2 \cdot (2 \cdot (2 \cdot (3) + 1) + 1) + 1 \\ &= 2 \cdot (2 \cdot (6 + 1) + 1) + 1 \\ &= 2 \cdot (2 \cdot (7) + 1) + 1 \\ &= 2 \cdot (14 + 1) + 1 \\ &= 2 \cdot (15) + 1 \\ &= 30 + 1 \\ &= 31 \end{aligned}$$

(c) In welchem Verhältnis oder mathematischen Zusammenhang stehen $n \geq 0$ und der Funktionswert $f(n)$? Geben Sie die Funktion $f(n)$ in einer nicht-rekursiven Form an.

$$f(n) = 2^{(n+1)} - 1$$

Informatik I - Gruppe 1 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

Aufgabe T6.2: ADT Spezifikation (4 Punkte)

In der Vorlesung haben Sie abstrakte Datentypen (ADT) kennengelernt. Dabei wurde in der Vorlesung die Spezifikation eines Datentyps A vorgestellt (Folie 291). Sie sollen nun 2 Realisierungen des Datentyps A angeben, die nicht bereits in der Vorlesung vorgestellt wurden (Bool, Nat0 und anderen Datentypen aus Num) und welche die Signatur des Datentyps A realisieren. Überprüfen Sie für beide Realisierungen, ob diese die Spezifikation erfüllen.

```haskell
--Vorlesungsbeispiel für den ADT A
--Signatur:
class ADT a where
    a, b :: a
    op1   :: a -> a
    op2, op3  :: a -> a -> a
--Semantik
    op1 a = b
    op2 a x = x
    op3 b x = x

--Beispiel 1: ADT Color
--Signatur:
class Color c where
    red, green, blue, yellow    :: c
    kompleColor                 :: c -> c
    mixColor                    :: c -> Float -> c
    lightenColor                :: c -> c -> c
--Semantik
    kompleColor red = green
    mixColor blue yelllow = green
    lightenColor c f = c

--Beispiel 2: ADT Fish
--Signatur:
class Fish f where
    shark, trout :: f
    saltwaterFish   :: f -> Bool
    isLarger, isStronger  :: f -> f -> f
--Semantik
    saltwaterFish f = True
    isLarger f g = f
    isStronger f g = g
```

Informatik I - Gruppe 1 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

Aufgabe P6.3: Binomialkoeffizienten / Pascalsches Dreieck (2 Punkte)

Zur Ermittlung der Binomialkoeffizienten $\binom{n}{k}$ kann das Pascalsche Dreieck zu Hilfe gezogen werden. Den passenden Koeffizienten findet man in der n-ten Zeile an der k-ten Stelle:

$$
\begin{array}{ccccccccccccc}
 & & & & & & 1 & & & & & & \\
 & & & & & 1 & & 1 & & & & & \\
 & & & & 1 & & 2 & & 1 & & & & \\
 & & & 1 & & 3 & & 3 & & 1 & & & \\
 & & 1 & & 4 & & 6 & & 4 & & 1 & & \\
 & 1 & & 5 & & 10 & & 10 & & 5 & & 1 & \\
1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1 \\
 & 7 & & 21 & & 35 & & 35 & & 21 & & 7 & \\
\end{array}
$$

Dabei kann dies mathematisch durch folgende Funktion f definiert werden:

$$
\begin{aligned}
f(n,0) &= 1 \\
f(n,n) &= 1 \\
f(n,k) &= \begin{cases} 0 & \text{für } k > n \\ f(n-1,k-1) + f(n-1,k) & \text{sonst} \end{cases}
\end{aligned}
$$

Implementieren Sie die Funktion `pascal::Int → Int → Int`, die die Parameter n und k entgegennimmt und den Binomialkoeffizienten zurückliefert. Implementieren Sie dies als rekursive Funktion. Geben Sie die Berechnung für `pascal (3,1)` als Kommentar an. Geben Sie außerdem an, ob es sich um eine direkte oder indirekte Rekursion handelt.

```haskell
pascal :: Int -> Int -> Int
pascal n k
    | k == 0 = 1
    | n == k = 1
    | otherwise = if k>n
       then 0
       else
           pascal (n-1) (k-1) + pascal (n-1) k
```

Informatik I - Gruppe 1 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

Aufgabe P6.4: Rationale Zahlen (3 + 3 = 6 Punkte)

In der Vorlesung haben Sie eine Haskell-Implementierung der rationalen Zahlen kennengelernt. In dieser Aufgaben soll die Implementierung des Datentyps `Ratio` erweitert werden. Folgen Sie dabei dem Programmentwurf der Vorlesung und erstellen Sie (1) Funktionsköpfe, (2) Beispiele und (3) Funktionsrümpfe sowie (4) eine Überprüfung anhand der generierten Beispiele.

(a) Entwickeln Sie eine Funktion `reduceFraction`, die eine rationale Zahle entgegen nimmt und die vollständig gekürzte rationale Zahl zurück gibt. Implementieren Sie dazu die Hilfsfunktion `ggT`, die den größten gemeinsamen Teiler zweier Zahlen a und b bestimmt, indem sie a zurück gibt, falls $b = 0$. Ansonsten soll der größte gemeinsame Teiler von b und $a\%b$ bestimmt werden.

(b) Entwickeln Sie Funktionen `fromRatio` und `toRatio`, die eine Konvertierung zwischen dem Datentyp Float und den rationalen Zahlen vornehmen. Die rationale Zahl soll dazu in ihrer vollständig gekürzten Form zurück gegeben werden. Nicht jede rationale Zahl kann als Gleitkommazahl exakt dargestellt werden. Runden Sie die Gleitkommazahlen dazu im Kontext dieser Aufgabe auf 5 Nachkommastellen.

Hinweis: Sie können die Haskell-Datei aus dem Learnweb als Startpunkt nutzen.

```haskell
import Distribution.Simple.Flag (Flag)
-- Datentyp für rationale Zahlen repräsentiert durch zwei Ganzzahlen für Zähler und Nenner
data Ratio = MkRatio Integer Integer


-- Gleichheit für rationale Zahlen
instance Eq Ratio where
  (MkRatio nom1 den1) == (MkRatio nom2 den2) = (nom1 * den2 == nom2 * den1)

-- Ordnung für rationale Zahlen
instance Ord Ratio where
  (MkRatio nom1 den1) <= (MkRatio nom2 den2) =
    if ((signum den1) * (signum den2) == 1)
      then (nom1 * den2 <= nom2 * den1)
      else (nom2 * den1 <= nom1 * den2)

-- Rechenoperationen für rationale Zahlen inklusive Addition, Multiplikation, Negierung,
↪  Betragsfunktion, Vorzeichenfunktion und Konvertierung von Integer
instance Num Ratio where
  (+) (MkRatio nom1 den1) (MkRatio nom2 den2) = (MkRatio (nom1 * den2 + nom2 * den1) (den1 *
  ↪  den2))
  (*) (MkRatio nom1 den1) (MkRatio nom2 den2) = (MkRatio (nom1 * nom2) (den1 * den2))
  negate (MkRatio nom den) = (MkRatio (- nom) den)
  abs (MkRatio nom den) = (MkRatio (abs nom) (abs den))
  signum (MkRatio nom den) = (MkRatio ((signum nom) * (signum den)) 1)
  fromInteger i = (MkRatio i 1)
```

Informatik I - Gruppe 1 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell
-- Funktion zum Anzeigen von rationalen Zahlen
-- Die Notation wird später in der Vorlesung erläutert
instance Show Ratio where
  show (MkRatio nom den) = (show nom) ++ "/" ++ (show den)

ggT :: Integer -> Integer -> Integer
ggT a b = if b == 0 then a
    else ggT b (mod a b)

myRatio :: Ratio
myRatio = MkRatio 2 3

reduceFraction :: Ratio -> Ratio
reduceFraction (MkRatio nom den) = MkRatio (div nom (ggT nom den)) (div den (ggT nom den))

fromRatio :: Ratio -> Float
fromRatio (MkRatio nom den) = roundFive (fromIntegral nom / fromIntegral den)

roundFive :: Float -> Float
roundFive f = fromIntegral(round (f*10^5))/(10^5)

decimalPlace :: Float -> Integer
decimalPlace f = truncate (roundFive f*10^5)

toRatio :: Float -> Ratio
toRatio f = reduceFraction (MkRatio (decimalPlace f) 100000)
```

Informatik I - Gruppe 1 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

Aufgabe P6.5: Rekursive Datentypen $(1+3+1, 5+1, 5+3 = 10$ Punkte)

In der Vorlesung haben Sie den ADT Binärbaum `Btree` kennengelernt. Sie sollen nun einen konkrete Datentypen `BtreeFloat` implementieren, der nur Elemente vom Typ `Float` abspeichert. Folgen Sie dabei dem Programmentwurf der Vorlesung und erstellen Sie (1) Funktionsköpfe, (2) Beispiele und (3) Funktionsrümpfe sowie (4) eine Überprüfung anhand der generierten Beispiele. Bearbeiten Sie dafür folgende Aufgaben:

(a) Implementieren Sie nun den Binärbaum `BtreeFloat`. Ein Binärbaum ist entweder ein leerer Knoten `Nil` oder ein Knoten `Node` mit einer Gleitkommazahl `Float` als Wert und einem linken Kind (einem Binärbaum) und einem rechten Kind (einem Binärbaum).

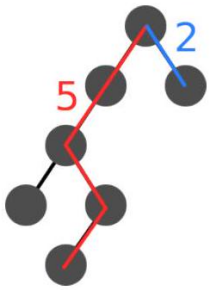
(b) Implementieren Sie die Funktion `insert :: BtreeFloat -> Float -> BtreeFloat` für den Datentyp `BtreeFloat`. Diese Funktion fügt eine Gleitkommazahl in den Binärbaum ein. Für jeden Knoten wird der Wert s mit dem Wert des aktuellen Knoten s_i verglichen. Ist $s < s_i$ dann wird der Wert in den linken Binärbaum eingefügt, ist $s > s_i$ dann wird der Wert in den rechten Binärbaum eingefügt. Sind beide Werte gleich wird der Wert nicht eingefügt. Der Wert s wird soweit nach unten durchgereicht bis ein Blatt erreicht wird. Dort wird dann ein neuer Knoten mit dem Wert s und zwei leeren Kindern (Binärbäumen) eingefügt.

(c) Implementieren Sie die Funktion `depth : : BtreeFloat-> Int` für den Datentypen `BtreeFloat`. Diese Funktion gibt die Tiefe eines Binärbaumes an. Dabei ist die Tiefe beschrieben durch den maximal längsten Pfad von der Wurzel bis zu einem Blatt.

(d) Implementieren Sie die Funktion `max_node :: BtreeFloat -> Float` für den Datentypen `BtreeFloat`. Diese Funktion gibt den größten Knotenwert innerhalb des Binärbaumes zurück. Sollte die Eingabe nur ein leerer Knoten sein, dann soll `max_node` den Wert 0 zurückgeben. *Hinweis: Sie können davon ausgehen, dass der Binärbaum mit der Funktion `insert` `erstellt` wurde.*

(e) Implementieren Sie die Funktion `path_diff :: BtreeFloat -> Int` für den Datentypen BtreeFloat. Diese Funktion gibt die Differenz zwischen dem kürzesten und dem längsten Pfad im Binärbaum zurück. Ein Pfad sei dabei die Anzahl der Knoten vom Wurzelknoten (dem Startknoten) zu einem Blattknoten.

Beispiel:



In dem Beispiel oben hat der angegebene Binärbaum den kürzesten Pfad der Länge 2 und der längste Pfad sei dabei 5. In diesem Fall würde die Funktion 3 zurückgeben.

Informatik I - Gruppe 1 - Übungblatt 6

Ausgabe: 14.11.2022
Abgabe: 21.11.1022
Tutor: Tim Völker

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

```haskell

{-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
{-# HLINT ignore "Use camelCase" #-}
data BtreeFloat = Nil | Node Float BtreeFloat BtreeFloat
  deriving (Eq, Show, Ord)

insert :: BtreeFloat -> Float -> BtreeFloat
insert Nil flt = Node flt Nil Nil
insert (Node val left right) flt
  |flt < val = Node val (insert left flt)  right
  |otherwise = Node val left (insert right flt)

depth :: BtreeFloat -> Int
depth Nil = 0
depth (Node val left right)
  |depth left > depth right = 1 + depth left
  |depth left < depth right = 1 + depth right
  |depth left == depth right = 1 + depth right
  |otherwise = 1 + depth right

max_node :: BtreeFloat -> Float
max_node Nil = 0
max_node (Node val Nil Nil) = val
max_node (Node  val  left right) = max_node right

path_diff :: BtreeFloat -> Int
path_diff Nil = depth Nil - shortest_path Nil
path_diff (Node left val right) = depth (Node left val right) - shortest_path (Node left val
↪  right)

shortest_path :: BtreeFloat -> Int
shortest_path Nil = 1
shortest_path (Node val left right)
  |shortest_path left > shortest_path right = 1 + shortest_path right
  |shortest_path left < shortest_path right = 1 + shortest_path left
  |shortest_path left == shortest_path right = 1 + shortest_path left
  |otherwise = 1 + shortest_path left

tree :: BtreeFloat
tree = Node 1 (Node 2 (Node 3 (Node 4 Nil Nil) (Node 5 (Node 6 Nil Nil) Nil)) Nil) (Node 7 Nil
↪  Nil)
```
