

Informatik I - Gruppe 1 - Übungsblatt 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.

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

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.

```
1  --Vorlesungsbeispiel für den ADT A
2  --Signatur:
3  class ADT a where
4      a, b :: a
5      op1  :: a -> a
6      op2, op3 :: a -> a -> a
7  --Semantik
8      op1 a = b
9      op2 a x = x
10     op3 b x = x
11
12 --Beispiel 1: ADT Color
13 --Signatur:
14 class Color c where
15     red, green, blue, yellow :: c
16     kompleColor              :: c -> c
17     mixColor                  :: c -> Float -> c
18     lightenColor              :: c -> c -> c
19 --Semantik
20     kompleColor red = green
21     mixColor blue yellow = green
22     lightenColor c f = c
23
24 --Beispiel 2: ADT Fish
25 --Signatur:
26 class Fish f where
27     shark, trout :: f
28     saltwaterFish :: f -> Bool
29     isLarger, isStronger :: f -> f -> f
30 --Semantik
31     saltwaterFish f = True
32     isLarger f g = f
33     isStronger f g = g
```

Informatik I - Gruppe 1 - Übungsblatt 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:

				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	
1	7	21	35	35	21	7	1		

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.

```
1 -- Die Funktion nimmt n über k entgegen und gibt den Binomialkoeffizienten aus.
2 -- Beispiel für n = 3 und k = 1 gibt 3 aus.
3 -- Diese Funktion ist eine direkte rekursive Funktion da sie (pascal) sich selbst wieder
  ↳ aufruft (pascal).
4 pascal :: Int -> Int -> Int
5 pascal n k
6     | k == 0 = 1
7     | n == k = 1
8     | otherwise = if k>n
9         then 0
10        else
11            pascal (n-1) (k-1) + pascal (n-1) k
```

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 Programm-entwurf 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 Zahl 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.

```
1 import Distribution.Simple.Flag (Flag)
2 -- Funktion zum Anzeigen von rationalen Zahlen
3 -- Die Notation wird später in der Vorlesung erläutert
4 instance Show Ratio where
5     show (MkRatio nom den) = (show nom) ++ "/" ++ (show den)
6 -- Eine Hilfsfunktion die den größten gemeinsamen teiler zweier Zahlen a und b bestimmt indem
7 -- sie a zurückgibt falls b = 0 ist. Ansonsten soll der größte gemeinsame Teiler von a Modulo
8 -- b bestimmt werden.
9 --Beispiel ggT 4 6 sollte 2 ausgeben.
10 ggT :: Integer -> Integer -> Integer
11 ggT a b = if b == 0 then a
12         else ggT b (mod a b)
13 -- Funktionen erstellen Rationale Zahlen
14 myRatio :: Ratio
15 myRatio = MkRatio 2 3
16 myRatio2 :: Ratio
17 myRatio2 = MkRatio 4 6
18 -- Funktion die eine rationale Zahl vollständig kürzt.
19 --Beispiel reduceFraction MyRatio2 sollte 2 / 3 ausgeben.
20 reduceFraction :: Ratio -> Ratio
21 reduceFraction (MkRatio nom den) = MkRatio (div nom (ggT nom den)) (div den (ggT nom den))
22
23 fromRatio :: Ratio -> Float
24 fromRatio (MkRatio nom den) = roundFive (fromIntegral nom / fromIntegral den)
25 -- Funktion die eine Gleitkommazahl auf fünf nachkommastellen rundet
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

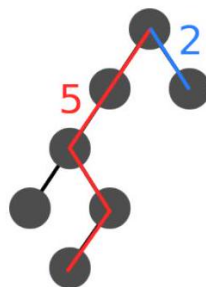
```
25 -- Beispiel roundFive 2.1234567 gibt 2.12346 aus.
26 roundFive :: Float -> Float
27 roundFive f = fromIntegral(round (f*10^5))/(10^5)
28 -- Funktion die eine Gleitkommazahl die auf die fünfte Stelle gerundet wird in eine Ganzzahl
   ↳ umwandelt, mit der Besonderheit das die Zahlen nach dem Komma nun zur ganzen Zahl gehören.
   ↳
29 -- Beispiel decimalPlace 2.1234567 gibt 212346 aus.
30 decimalPlace :: Float -> Integer
31 decimalPlace f = truncate (roundFive f*10^5)
32 -- Funktion die eine Gleitkommazahl in eine Rationale Zahl umwandelt mit der Besonderheit das
   ↳ die Gleitkommazahl vor der umwandlung auf die 5. nachkommastelle gerundet wird.
33 -- Beispiel toRatio 2.1234567 sollte 106173 / 50000 ausgeben.
34 toRatio :: Float -> Ratio
35 toRatio f = reduceFraction (MkRatio (decimalPlace f) 100000)
```

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 konkreten Datentypen **BtreeFloat** implementieren, der nur Elemente vom Typ **Float** abspeichert. Folgen Sie dabei dem Programmmentwurf 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 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
1 {-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
2 {-# HLINT ignore "Use camelCase" #-}
3 -- Die Datentyp BtreeFloat besteht aus einem leerem Knoten Nil oder ein Knoten Node mit einer
4   ↳ Gleitkommazahl (Float) als Wert und einem linken Kind (einem Binärbaum) und einem rechten
5   ↳ Kind (einem Binärbaum).
6 data BtreeFloat = Nil | Node Float BtreeFloat BtreeFloat
7   deriving (Eq, Show, Ord)
8 -- Die Funktion fügt eine Gleitkommazahl in den Binärbaum ein. Diese Funktion fügt eine
9   ↳ Gleitkommazahl in den Binärbaum ein. Für jeden Knoten wird der Wert s mit dem Wert des
10  ↳ aktuellen Knotens s^i verglichen. Ist s < s^i dann wird der Wert in den linken Binärbaum
11  ↳ eingefügt, ist s > s^i dann wird der Wert in den rechten Binärbaum eingefügt. Sind beide
12  ↳ Werte gleich wird der Wert nicht eingefügt. Der Wert s wird soweit nach unten
13  ↳ durchgereicht bis ein Blatt erreicht wird. Dort wird dann ein neuer Knoten mit dem Wert s
14  ↳ und zwei leeren Kindern (Binärbäumen) eingefügt
15 --Beispiel insert tree 2 sollte Node 1.0 (Node 2.0 (Node 3.0 (Node 4.0 Nil Nil) (Node 5.0
16   ↳ (Node 6.0 Nil Nil) Nil)) Nil (Node 7.0 (Node 2.0 Nil Nil) Nil) ausgeben.
17 insert :: BtreeFloat -> Float -> BtreeFloat
18 insert Nil flt = Node flt Nil Nil
19 insert (Node val left right) flt
20   | flt < val = Node val (insert left flt) right
21   | otherwise = Node val left (insert right flt)
22 -- Die Funktion gibt die Tiefe des Binärbaumes an. Die Tiefe ist dabei der maximal Längste
23   ↳ Pfad von der Wurzel bis zu einem Blatt.
24 -- Beispiel depth tree sollte 5 ausgeben
25 depth :: BtreeFloat -> Int
26 depth Nil = 0
27 depth (Node val left right)
28   | depth left > depth right = 1 + depth left
29   | depth left < depth right = 1 + depth right
30   | depth left == depth right = 1 + depth right
31   | otherwise = 1 + depth right
32 -- Die Funktion gibt den größten Knotenwert innerhalb eines Binärbaumes zurück, im Falle eines
33   ↳ leeren Knotens ist die Ausgabe 0.
34 -- Beispiel max_node tree ist 7.0
35 max_node :: BtreeFloat -> Float
36 max_node Nil = 0
37 max_node (Node val Nil Nil) = val
38 max_node (Node val left right) = max_node right
39 -- Die Funktion gibt die Differenz zwischen dem kürzesten und dem längsten Pfad im Binärbaum
40   ↳ zurück. Ein Pfad sei dabei die Anzahl der Knoten vom Wurzelknoten (dem Startknoten) zu
41   ↳ einem Blattknoten.
42 -- Beispiel path_diff tree sollte 2 ausgeben
43 path_diff :: BtreeFloat -> Int
44 path_diff Nil = depth Nil - shortest_path Nil
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
32 path_diff (Node left val right) = depth (Node left val right) - shortest_path (Node left val
    ↪ right)
33 -- Hilfsfunktion um die Funktion path_diff implementieren zu können
34 shortest_path :: BtreeFloat -> Int
35 shortest_path Nil = 1
36 shortest_path (Node val left right)
37   | shortest_path left > shortest_path right = 1 + shortest_path right
38   | shortest_path left < shortest_path right = 1 + shortest_path left
39   | shortest_path left == shortest_path right = 1 + shortest_path left
40   | otherwise = 1 + shortest_path left
41 -- Konstruktor für einen Binärbaum BtreeFloat
42 tree :: BtreeFloat
43 tree = Node 1 (Node 2 (Node 3 (Node 4 Nil Nil) (Node 5 (Node 6 Nil Nil) Nil)) Nil) (Node 7 Nil
    ↪ Nil)
```
