WS 2020/2021

Funktionale Programmierung

5. Übungsblatt

Prof. Dr. Margarita Esponda

Ziel: Arbeiten mit Funktionen höherer Ordnung, endrekursiven Funktionen und Komplexitätsanalyse von Funktionen.

1. Aufgabe (3 Punkte)

Im Haskell-Prelude ist die **iterate**-Funktion wie folgt definiert:

```
iterate      :: (a -> a) -> a -> [a]
iterate f x  =  x : iterate f ( f x )
```

Definieren Sie unter Verwendung der **iterate**-Funktion Funktionen, die folgende unendlichen Listen darstellen:

[1, -1, 1, -1, 1, -1, …]
[0, 1, 3, 7, 15, 31, 63, …]
[(0,1), (1,1), (1,2), (2,3), (3,5), (5,8), …]

Lösung:

```
list1 = iterate (*(-1)) 1
list2 = iterate ((+1).(*2)) 0

nextFib :: (Int,Int) -> (Int,Int)
nextFib (a,b) = (b, a+b)

list3 = iterate  nextFib (0,1)
```

2. Aufgabe (4 Punkte)

Analysieren Sie die Komplexität folgender zwei Multiplikationsfunktionen:

```
mult :: Integer -> Integer -> Integer
mult n 0 = 0
mult n m = mult n (m-1) + n

russMult :: Integer -> Integer -> Integer
russMult n 0 = 0
russMult n m | (mod m 2)==0  = russMult (n+n) (div m 2)
             | otherwise     = russMult (n+n) (div m 2) + n
```

Lösung:

Die **Eingabegröße** ist: **m** = das 2. Argument der **mult** Funktion

In der ***mult*** Funktion findet die Rekursion über das zweite Argument von (m-1) bis 0 statt. D.h. die maximale Anzahl von rekursiven Aufrufen ist gleich **m**. Zum Schluss wird **m**-Mal **n** zusammenaddiert.

$$T(m) = 2 \cdot m \in O(m)$$

Die **Eingabegröße** ist: **m** = 2. Argument der **russMult** Funktion

In der ***russMult*** Funktion wird die Rekursion nur so lange durchgeführt, bis **m** durch **2** ungleich 0 ist. D.h. so lange **m** noch ganzzahlig teilbar ist. Das entspricht dem Logarithmus Basis 2 von m.

$$T(m) = c \cdot \log_2(m) = O(\log_2 m) \qquad mit\ c = Berechnungsaufwand\ der\ (+),\ div\ und\ \mathrm{mod}\ Funktionen.$$

3. Aufgabe (6 Punkte)

a) Schreiben Sie eine Haskell Funktion, das einen Text als Eingabe bekommt und alle Worte, die sich mit den letzen drei Buchstaben reimen, in eine Liste von Gruppenworten klassifiziert.

Anwendungsbeispiel:

classifyRhymeWords "Nikolaus baut ein Haus aus Holz und klaut dabei ein Bauhaus." =>

[[“klaut“,"baut"],["Nikolaus","Bauhaus","Haus","aus"],["ein","ein"],["dabei"],["und"],["Holz"]]

Lösung:

```
delimiter :: Char -> Bool
delimiter c = elem c [',', '.', '?', '!', ' ', '\n', '\t']

text2words :: [Char] -> [[Char]]
text2words [] = []
text2words ws = text2words' [] [] ws
            where
            text2words' acc [] []     = acc
            text2words' acc word []   = word:acc
            text2words' acc word (x:rest) | delimiter x  = text2words' (word:acc) [] rest
                                          | otherwise    = text2words' acc (word++[x]) rest

groupSamePrefix :: [[Char]] -> [[[Char]]]
groupSamePrefix  []   = []
groupSamePrefix [word] = [[word]]
groupSamePrefix  ws   = gPrefix [] [] ws
                where
                gPrefix xs []  []  = xs
                gPrefix xs rws []  = rws:xs
                gPrefix xs []  (w:ws) = gPrefix xs [w] ws
                gPrefix xs (rw:rws) (w:ws) | samePrefix rw w = gPrefix xs (w:rw:rws) ws
                                           | otherwise       = gPrefix ((rw:rws):xs) [w] ws

                samePrefix (x:y:z:rest) (x':y':z':rest') = (x,y,z) == (x',y',z')
                samePrefix a b = a==b

classifyRhymeWords :: [Char] -> [[[Char]]]
classifyRhymeWords text = map (map reverse) (groupSamePrefix (startMergeSort (text2words
                                                                    (reverse text))))
```

b) Analysieren Sie die Komplexität der Funktion.

4. Aufgabe (6 Punkte)

Der Selectionsort Algorithmus sucht das kleinste/größte Element in der zu sortierenden Liste, platziert dieses am Anfang der Liste und wiederholt das Verfahren mit dem Rest der Liste.

a) Definieren Sie eine polymorphe Funktion, die unter Verwendung des Selectionsort-Algorithmus die Elemente einer gegebenen Liste sortiert. Ein zweites Argument, das eine Vergleichsoperation sein soll, entscheidet, ob die Liste in absteigender oder ansteigender Reihenfolge sortiert wird. Verwenden Sie in Ihrer Definition eine lokale Hilfsfunktion **calculateFirst**, die je nach angegebener Vergleichsoperation das kleinste oder das größte Element der Liste findet, und eine zweite lokale Funktion **deleteElem**, die das gefundene Element aus der Restliste entfernt.

Anwendungsbeispiele:

selectionSort (<) [2,1,5,0,4,3,7] => [0,1,2,3,4,5,7]

selectionSort (>) [2,1,5,0,4,3,7] => [7,5,4,3,2,1,0]

Lösung:

```
selectSort :: (Ord a) => (a->a->Bool) -> [a] -> [a]
selectSort p []  = []
selectSort p [x] = [x]
selectSort p xs  = first: (selectSort p ys)
            where
            first = calculateFirst p xs
            ys = deleteFirst first xs

            calculateFirst p [x]    = x
            calculateFirst p (x:xs) = calculateFirst' p x xs
                        where
                        calculateFirst' p x [] = x
                        calculateFirst' p x (y:ys) | p y x     = calculateFirst' p y ys
                                                   | otherwise = calculateFirst' p x ys
            deleteElem x [] = []
            deleteElem x (y:ys) | x==y = ys
                                | otherwise = y: (deleteElem x ys)
```

b) Analysieren Sie die Komplexität der Funktion.

Lösung:

Die Eingabegröße ist **n** = Länge der zu sortierenden Liste.

Um das kleinste/größte Element der Liste zu finden, muss die Liste zuerst komplett durchlaufen werden. Ein zweiter Durchlauf, der im schlimmsten Fall bis Ende der Liste geht, muss stattfinden, um das gefundene kleinste/größte Element zu löschen. Das verursacht insgesamt ***2*n*** Reduktionen. Die **selectSort** Funktion wird dann rekursiv mit der Restliste angewendet, die (***n-1***) Elemente beinhaltet. Die Rekursion geht solange, bis die Eingabeliste nur noch ein Element hat. Zum Schluss müssen ***n*** (:) Operationen durchgeführt werden.

Anzahl der Reduktionen

$$selectSort\ p\ [x_1,x_2,...,x_n] \Rightarrow x_n : selectSort\ p\ [x_1,x_2,...,x_{n-1}]$$ 1 + 2*n

$$\Rightarrow x_i : x_k : selectSort\ p\ [x_1,x_2,...,x_{n-2}]$$ 1 + 2*(n-1)

$$\Rightarrow x_i : x_k : x_j : selectSort\ p\ [x_1,x_2,...,x_{n-3}]$$ 1 + 2*(n-2)

. . .

$$\Rightarrow x_i : x_k : x_j : ... : x_m\ selectSort\ p\ [x_1]$$ 1 + 2*1

$$T(n) = n + 2 \cdot \sum_{i=n}^{1} i = n + \frac{n \cdot (n+1)}{2} + n = (2 + \frac{1}{2}) \cdot n + \frac{1}{2} \cdot (n^2) = a \cdot n + b \cdot n^2 \in O(n^2)$$

5. Aufgabe (3 Punkte + 3 Bonuspunkte)

a) (3 Punkte) Definieren Sie eine Funktion **pyth_tripels** die bei Eingabe einer natürlichen Zahl **n**, die Liste aller (a, b, c) Pythagoras-Zahlentripel mit $0 < a \le b \le c \le n$ ohne Wiederholungen berechnet.

Anwendungsbeispiel:

pyth_tripels 16 => [(3,4,5),(5,12,13),(6,8,10),(9,12,15)]

Lösung 1): Hier wird **pyth_tripels_until** anstatt **pyth_tripels** verwendet

```
pyth_tripel :: Int -> Int -> Int -> Bool
pyth_tripel a b c  = a>0 && b>0 && c>0 && (a^2 + b^2 + c^2 - 2*(hypo^2) == 0)
                where
                hypo = max a (max b c)

pythagoras_all :: Int -> [[Int]]
pythagoras_all n = [[a,b,c] | a <- [1..n], b <- [1..n], c <- [1..n], pyth_tripel a b c]


pyth_tripels_until :: Int -> [[Int]]
pyth_tripels_until n = makeSet (sort (map sort (pythagoras_all n)))
```

b) (3 Bonuspunkte) Analysieren Sie die Komplexität der **pyth_tripels_until** Funktion.

Die Komplexität der **pyth_tripels_until** Funktion hängt von der Länge der Ergebnisliste, die die **pythagoras_all** Funktion berechnet, ab. Es gibt noch keine direkte Möglichkeit bei Eingabe eines beliebigen **n** die Länge direkt zu berechnen, aber die Anzahl der pythagoräischen Tripel kann nicht größer als **n^2** sein, weil in jedem pythagoräischen Tripel (a,b,c) die Gleichung $c^2 = a^2 + b^2$) gilt. D.h. Der Wert von c kann immer aus a und b berechnet werden und es kann nur maximal n^2 verschiedene Werte für c geben, die auch alle verschiedenen Kombinationen von a und b entsprechen. Dann, wenn wir einen Sortieralgorithmus mit O(n(log(n))) Komplexität verwenden, sieht die Komplexität der pyth_tripels_until Funktion im schlimmsten Fall wie folgt aus:

Lösung 2):

```
pyth_tripels_until :: Int -> [(Int, Int, Int)]
pyth_tripels_until n = [(a,b,c) | a <- [1..n], b <- [a..n], c <- [b..n], pyth_tripel a b c]
```

Komplexitätsanalyse:

$$
\begin{aligned}
T_{pyth\_tripels\_until}(n) &= \sum_{i=1}^{n}\sum_{k=1}^{n} k = \sum_{i=1}^{n}\left(\tfrac{n\cdot(n+1)}{2}\right) \\
&= \frac{1}{2}\sum_{i=1}^{n}(n^2+n) = \frac{1}{2}\sum_{i=1}^{n} n^2 + \frac{1}{2}\sum_{i=1}^{n} n \\
&= \frac{1}{2}\sum_{i=1}^{n} n^2 + \frac{1}{2}\left(\frac{n\cdot(n+1)}{2}\right) \\
&= \frac{n\cdot(n+1)\cdot(2n+1)}{6} + \frac{1}{4}(n^2+n) \\
&= \frac{2n^3+n^2+n^2+n}{6} + \frac{1}{4}(n^2+n) \\
&= c_1\cdot n^3 + c_2\cdot n^2 + c_3\cdot n \\
&= O(n^3)
\end{aligned}
$$

6. Aufgabe (3 Punkte)

In der Bildverarbeitung werden für die Darstellung von Farben unterschiedliche Formate verwendet.

Es gibt z.B. das RGB-Format, in dem mit Hilfe von drei ganzen Zahlen zwischen 0 und 255 (Rot, Grün und Blau-Werte) die Farben kodiert werden.

Im Zeitschriften und Büchern wird das CMYK-Format verwendet, in dem Bilder aus einer Kombination der Farben Cyan, Magenta, Yellow und Black (CMYK) gedruckt werden.

Definieren Sie einen algebraischen Datentyp **Color** indem Sie Farben in verschiedenen Formaten darstellen, und definieren Sie damit die **rgb2cmyk** Funktion, die unter Verwendung folgender Formel und nach Eingabe der RGB-Werte die entsprechenden CMYK-Werte berechnet.

Wenn RGB = (0,0,0), dann ist CMYK = (0,0,0,1),

sonst werden die Werte nach folgenden Formeln berechnet:

$$
\begin{aligned}
w &= \max(R/255, G/255, B/255) \\
C &= (w-(R/255))/w \\
M &= (w-(G/255))/w \\
Y &= (w-(B/255))/w \\
K &= 1-w
\end{aligned}
$$

Lösung:

```
data Color = RGB Int Int Int | CMYK Double Double Double Double
                                                    deriving Show
rgb2cmyk :: Color -> Color
rgb2cmyk (RGB 0 0 0)  = CMYK 0 0 0 1
rgb2cmyk (RGB r g b)  = CMYK c m y k
                                  where
                                    rd = (fromIntegral r)/255
                                    gd = (fromIntegral g)/255
                                    bd = (fromIntegral b)/255
                                    w = max (max rd gd) bd
                                    c  = w - rd/w
                                    m = w - gd/w
                                    y  = w - bd/w
                                    k  = 1 - w
```

7. Aufgabe (5 Punkte)

Nehmen wir an, wir müssen Berechnungen realisieren mit Zahlen, die folgende Form haben:

$a+b\sqrt{2}$, wobei ***a*** und ***b*** ganze Zahlen sind.

Würden wir zuerst die Wurzeln ausrechnen und dann die Summen machen, hätten wir Rundungsfehler, die im Laufe der Berechnungen größer werden können. Die Rundungsfehler können minimiert werden, wenn wir zuerst nur die ganzzahligen Operationen realisieren und das Ausrechnen der Wurzeln ans Ende verschieben. Beispiel:

$$\left(3+2\sqrt{2}\right)\cdot\left(2+\sqrt{2}\right)=10+7\sqrt{2}$$

a) Definieren Sie einen algebraischen Datentyp **RootNum**, der unsere Zahlen mit Hilfe der Koeffizienten **a** und **b** darstellt, und definieren Sie die Additions-, Subtraktions- und Multiplikationsoperation für diesen Datentyp.

b) Zum Testen definieren Sie eine Funktion **getValue,** die eine **RootNum** Variable in einer Gleitkommazahl ausrechnet.

Lösung:

```
data RootNum = RN Integer Integer  deriving (Show, Eq)

add :: RootNum -> RootNum -> RootNum
add (RN a b) (RN c d) = RN (a+c) (b+d)

sub :: RootNum -> RootNum -> RootNum
sub (RN a b) (RN c d) = RN (a-c) (b-d)
```

```
mult :: RootNum -> RootNum -> RootNum
mult (RN a b) (RN c d) = RN (a*c+2*b*d) (a*d+b*c)

getValue :: RootNum -> Double
getValue (RN a b) = (fromIntegral a) + (fromIntegral b)*sqrt(2)
```

8. Aufgabe (6 Punkte)

Ein Element einer Liste von ***n*** Objekten stellt die absolute Mehrheit der Liste dar, wenn das Element mindestens $\left(\frac{n}{2}+1\right)$-mal in der Liste vorkommt.

a) Definieren Sie eine **majority** Funktion, die das Majority-Element der Liste findet, wenn eines existiert oder sonst **Nothing** zurückgibt.

Die Signatur der Funktion soll wie folgt aussehen:

majority :: (Eq a) => [a] -> **Maybe** a

b) Analysieren Sie die Komplexität der Funktionsdefinition.

O(n) Lösung:

```
majority :: (Eq a) => [a] -> Maybe a
majority []   = Nothing
majority [x]  = Just x
majority xs | (freq l_maj xs) > half = (Just l_maj)
            | otherwise = Nothing
              where
              (c, l_maj) = local_maj (1, head xs) (tail xs)
              local_maj (n, m) [] = (n, m)
              local_maj (n, m) (e:es) | e==m = local_maj (n+1,m) es
                                      | (e/=m && n==0) = local_maj (1, e) es
                                      | otherwise = local_maj (n-1,m) es
              half = div (length xs) 2

freq :: Eq a => a -> [a] -> Int
freq elem xs = sum [ 1 | x<-xs, x==elem ]
```

Begründung:

$$
\begin{aligned}
T_{majority}(n) &= T_{local\_maj}(n) + T_{freq}(n) + T_{half}(n) \\
&= c_1*n + c_2*n + c_3 \\
&= (c_1 + c_2)*n + c_3 \\
&= O(n)
\end{aligned}
$$

$O(n^2)$ Lösungsbeispiele:

```
majority' :: (Eq a) => [a] -> Maybe a
majority'  [] = Nothing
majority'  xs = if mll==[] then Nothing else Just (head mll)
           where
            mll = [e|e<-xs, length [m|m<-xs, m==e]>len]
            len = (div (length xs) 2)

majority' :: (Eq a) => [a] -> Maybe a
majority' []   = Nothing
majority' [x] = Just x
majority' xs  = maj xs xs
   where
      maj  []    _  = Nothing
      maj (x:xs) ys | (length (filter (==x) ys)) > len = Just x
                    | otherwise = maj xs ys
      len = (div (length xs) 2)
```

Wichtige Hinweise:

1) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Semantik der Funktionen wiedergeben.
2) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.
3) Kommentieren Sie Ihre Programme.
4) Verwenden Sie geeignete lokale Funktionen und Hilfsfunktionen in Ihren Funktionsdefinitionen.
5) Geben Sie für alle Funktionen die entsprechende Signatur an.
6) Schreiben Sie getrennte Test-Funktionen für alle Aufgaben.
7) Die Lösungen sollen elektronisch (nur Whiteboard-Upload) abgegeben werden. **Keine verspätete Abgabe per Email ist erlaubt.**