WiSe 2015/2016

Funktionale Programmierung

5. Übungsblatt (Abgabe: Mi., den 25.11. um 10:10 Uhr)

Prof. Dr. Margarita Esponda

Ziel: Auseinandersetzung mit Such- und Sortieralgorithmen, Funktionen höherer Ordnung und O-Notation.

1. 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 aus der Vorlesung 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:

```
selectSort (<) [2,1,5,0,4,3]   =>   [0,1,2,3,4,5]
selectSort (>) [2,1,0,4,3]     =>   [5,4,3,2,1,0]
```

b) Analysieren Sie die Laufzeitkomplexität Ihrer Funktion.

2. Aufgabe (3 Punkte)

Definieren Sie eine polymorphe Funktion **isSorted**, die bei Eingabe einer Liste von beliebigen sortierbaren Objekten entscheiden kann, ob diese sortiert sind oder nicht.

a) Schreiben Sie zuerst Ihre Definition mit Hilfe expliziter Rekursion (1 Punkt).

b) Definieren Sie in einer Zeile die **isSorted** Funktion unter sinnvoller Verwendung einer Funktion höherer Ordnung.

3. Aufgabe (5 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
```

4. Aufgabe (10 Punkte)

Definieren Sie eine polymorphe Funktion **maxLengthRepSeq**, die die längste sich wiederholende Objektsequenz aus einer Objekt-Liste findet (siehe Vorlesungsfolien).

Anwendungsbeispiele:

maxLengthRepSeq "abtdahdtdahko" => "tdah"

maxLengthRepSeq [1,1,0,1,1,0,0,1,0,1,1,0] => [1,0,1,1,0]

Vorgehensweise:

a) Programmieren Sie zuerst folgende drei Hilfsfunktionen:

i) Eine Funktion **allSuffixes**, die alle verschiedenen Suffixe einer Zeichenkette in einer Liste als Ergebnisse zurückgibt.

Anwendungsbeispiel: **allSuffixes** "xyzab" => ["xyzab","yzab","zab","ab","b"]

ii) Definieren Sie eine Funktion **prefix**, die zwei Zeichenketten bekommt und das längste gemeinsame Prefix, falls es eines gibt, berechnet.

Anwendungsbeispiel: **prefix** "abcde" "abacde" => "ab"

iii) Definieren Sie eine Funktion **largestPrefix**, die eine Liste von Zeichenketten durchgeht und das längste Prefix aus zwei benachbarten Zeichenketten der Liste in einem Tupel als Ergebnis zurückgibt. Das erste Element des Tupels ist die Länge des Prefixes.

Anwendungsbeispiel:

largestPrefix ["a","abca","bca","bcadabca","ca","cdabca"] => (3,"bca")

b) Programmieren Sie dann unter Verwendung Ihrer Funktionen in i), ii), iii) und eines geeigneten Sortieralgorithmus die **maxLengthRepSeq** Funktion.

c) Analysieren Sie die Komplexität alle definierten Funktionen.

5. Aufgabe (8 Punkte)

Schreiben Sie ein Haskell-Programm, das einen Text als Eingabe bekommt und alle Worte, die sich in 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"]]

6. Aufgabe (2 Punkte)

Benutzen Sie die **foldr**-Funktion, um eine eigene **myMin**-Funktion zu definieren, die das kleinste Element einer Liste berechnet.

Wichtige Hinweise:

3) **Zur jeder Funktion sollen Testfunktionen geschrieben werden.**

4) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Semantik der Funktionen wiedergeben.

5) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.

6) Kommentieren Sie Ihre Programme.

7) Verwenden Sie geeignete lokale Funktionen und Hilfsfunktionen in Ihren Funktionsdefinitionen.

5) Schreiben Sie in allen Funktionen die entsprechende Signatur.