Universität Bremen

Praktische Informatik 3 WS 2023/24

Thomas Barkowsky
Tede von Knorre
Yaroslav Purgin
Tim Sperling

3. Übungsblatt

Ausgabe: 06.11.2023
Abgabe: 13.11.2023

Achtung: In diesem Übungsblatt dürfen nur Funktionen der Standardbibliothek (*standard prelude*) verwendet werden. Es dürfen also keine weiteren Bibliotheken importiert werden.

3.1 *Spaß mit Listen* — *5 Punkte*

Im ersten Teil dieses Übungsblatts wollen wir ein wenig mit Listen herumspielen. Die Funktion

```
containsDoubleElem :: Eq a ⇒ [a] → Bool
```

soll für uns entscheiden, ob in einer Liste aufeinanderfolgend zwei gleiche Elemente enthalten sind:

```
containsDoubleElem "Hallo!" ⇝ True
containsDoubleElem [1,2,1] ⇝ False
```

Nun wäre es natürlich nett, wenn wir aufeinanderfolgende doppelte Elemente aus einer Liste entfernen könnten. Das soll die Funktion

```
removeDoubleElem :: Eq a ⇒ [a] → [a]
```

tun:

```
removeDoubleElem "Hallo!␣Huhu!" ⇝ "Halo!␣Huhu!"
```

Was war noch mal ein *Palindrom*? Ach ja, das ist eine Zeichenkette, die vorwärts und rückwärts gelesen die gleiche Zeichenfolge darstellt:

```
isPalindrome "ABBA" ⇝ True
isPalindrome "PAPA" ⇝ False
```

Ein leerer String und eine Zeichenkette, die aus nur einem Zeichen besteht, sind selbstverständlich auch Palindrome...

Und schließlich noch eine Funktion für Leute, die nicht so gut rechnen können: neulich las ich im Zusammenhang einer Empfehlung, wieviel Trinkgeld man (z.B. beim Frisör) geben sollte: "Wenn Sie nicht wissen, wieviel 10% sind, dann runden sie einfach auf den nächsten Zehnerbetrag auf"(im Ernst, ohne Witz!). Die Welt braucht also eine Funktion, der eine Liste von (Euro-)Beträgen übergeben bekommt und zu jedem Preis den passenden zu zahlenden Betrag (also zuzüglich 10 Prozent Trinkgeld, gerundet auf ganze Eurobeträge) zurückgibt:

```
computeTip :: [Float] → [(Float, Int)]
```

Beispiel:

```
computeTip [79.00, 19.20, 102.50] ⇝ [(79.0,87),(19.2,21),(102.5,113)]
```

Fürs Runden eignet sich übrigens die Funktion round.

3.2 *Noch einmal Primzahlen...* — *5 Punkte*

Gemäß der sogenannten *Goldbachschen Vermutung* lässt sich jede gerade Zahl größer als 2 als Summe zweier Primzahlen darstellen. Diese Vermutung ist unbewiesen, wobei bisher auch kein Gegenbeispiel gefunden werden konnte.

Wir schreiben zunächst eine Funktion

```
nextPrime :: Integer → Integer ,
```

die zu einer gegebenen Zahl die nächsthöhere Primzahl ermittelt:

```
nextPrime 19 ⇝ 23
nextPrime 1000 ⇝ 1009
```

Weiterhin ist es möglicherweise nützlich, eine Funktion

```
primesUpTo :: Integer → [Integer]
```

zu haben, die alle Primzahlen bis zu einer gegeben Obergrenze in einer Liste zusammenfasst:

```
primesUpTo 7  ⇝  [2,3,5,7]
primesUpTo 50  ⇝  [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]
```

Nun können wir die Funktion

```
goldbach :: Integer → [(Integer, Integer)]
```

schreiben, die zu einer gegebenen geraden Zahl alle Paare primer Summanden aufzählt. Jedes Paar soll hierbei nur einmal – mit der kleineren Zahl zuerst – aufgeführt werden. Beispiel:

```
goldbach 30  ⇝  [(7,23),(11,19),(13,17)]
```

Nun benötigen wir allerdings nicht **alle** Primzahlpaare für eine gegebene Zahl. Es reicht ja aus, **ein** Beispiel für eine Zerlegung zu finden (das spart eine Menge Rechenzeit und wir können uns schneller der nächsten zu prüfenden geraden Zahl zuwenden). Hierfür definieren wir die Funktion

```
goldbach2 :: Integer → (Integer, Integer, Integer) ,
```

die zu einer vorgegebenen geraden Zahl größer als 2 ein Tripel zurückgibt, bestehend aus der Zahl selbst und ihren beiden primen Summanden. Der erste der beiden Summanden soll immer die **kleinstmögliche** Zahl für eine Zerlegung sein. Beispiel:

```
goldbach2 1002  ⇝  (1002,5,997)
```

Hier angekommen sind wir beispielsweise in der Lage, mögliche Zerlegungen der ersten drei geraden Zahlen ab einer Milliarde zu berechnen:

```
[[goldbach2 a | a←[10^9,10^9+2..10^9+4]]  ⇝
[(1000000000,71,999999929),(1000000002,73,999999929),(1000000004,67,999999937)]
```