Praktische Informatik 1 WiSe 98/99

Prof. Bernd Krieg-Brückner
Shi Hui
Kolyang
Axel Lankenau
Christoph Lüth
Lutz Zegartowski

# 3. Übungsblatt

Ausgabe: 19.11.98

Abgabe: Woche 07.12. – 11.12.98

---

## 1 *Auf die Zerlegung kommt es an …* *Wichtung: 20%*

Jede natürliche Zahl $n$ mit $n > 1$ läßt sich eindeutig in ihre Primfaktoren zerlegen, so ist z.B. $42 = 2 \cdot 3 \cdot 7$. Schreibe eine Haskell–Funktion `pfaktor :: Int -> [Int]`, die für eine natürliche Zahl die Liste ihrer Primfaktoren liefert.

## 2 *Send more money …* *Wichtung: 20%*

Der Informatik-Student im 39. Semester, Bill G., kam mal wieder mit dem „Büchergeld“ nicht hin. Da sein Vater und Förderer ein Faible für Denksportaufgaben hatte, schickte Bill G. eine Bettelpostkarte mit den Worten

```
  S E N D
+ M O R E
---------      (gleicher Buchstabe, gleiche Ziffer;
M O N E Y       verschiedene Buchstaben, verschiedene Ziffern)
```

Der Vater hatte in seinem eigenen Grundstudium die *list comprehension* Darstellung in funktionalen Programmiersprachen kennengelernt und ließ die Gleichung von einem kleinen Haskell–Programm lösen. Zur Überraschung von Bill G. schickte sein Vater wesentlich weniger Geld, als er eigentlich erwartet hatte.

Implementiert eine Funktion `money :: -> [[Int]]`, die alle möglichen Kombinationen errechnet, wobei jedes Element einer inneren Liste für einen Buchstaben steht. Welchen Betrag überwies der Vater und welche Summe erwartete Bill G.?

## 3 *Die spinnen, die Römer …* *Wichtung: 30%*

Weil Cäsars Kuriere auf dem Weg ins befestigte Lager Brimborium zu häufig von den cleveren Galliern aus dem kleinen Dorf, das immer noch Widerstand leistete, abgefangen wurden, beauftragte der große Julius seinen Chef–Kryptologen Schlussimbus mit der Entwicklung eines Chiffrierverfahrens für geheime Nachrichten. Schlussimbus ersann folgende Geheimschrift: Im Originaltext wird jeder Buchstabe nach einer festen Substitutionsvorschrift durch einen anderen ersetzt, aus allen ‘a‘’s werden zum Beispiel ‘k‘’s, aus ‘b‘’s werden ‘u‘’s usw. Schlussimbus schrieb auf seinem Abakus II, 450 MHz eine Haskell–Funktion `caesar :: String -> [(Char,Char)] -> String`, die eine Nachricht und eine Kodierungstabelle (als Liste von Buchstabenpaaren) übergeben bekommt und die verschlüsselte Nachricht als Ergebnis liefert.
So wird z. B. `caesar "obelix" [('b','d'),('l','f'),('o','i')]` ausgewertet zu `"idefix"`.

1. Implementiere die Funktion `caesar` in Haskell! Diskutiere ihre Schwachpunkte und beschreibe welche Bedingungen die Kodierungstabelle erfüllen muß, damit der Empfänger die Nachricht dekodieren kann.

2. Dem gallischen Spion Nullnullsix ist es bisher nicht gelungen, eine der verschlüsselten Nachrichten zu entziffern. Während einer Orgie zu Ehren des nordischen Wagenlenkers Møkkå

Håkkønen überredet er daher Schlussimbus, das Verfahren dahingehend zu „verbessern“, daß statt der Kodierungstabelle nur eine natürliche Zahl als Parameter dient, die angibt, um wieviel Stellen im Alphabet jedes Zeichen verschoben wird. So wird z.B. `caesarNT "asterix" 3` ausgewertet zu `"dvwhula"`. Schlussimbus ist einverstanden und implementiert die Funktion `caesarNT :: String -> Int -> String` in Haskell. Tue es ihm nach!

3. Nullnullsix hatte natürlich auf nichts anderes gewartet und entwickelte eine Haskell–Funktion `crack :: String -> [String]`, die es ihm ermöglichte, alle weiteren römischen Nachrichten bequem zu entziffern. Wie das?

$\boxed{4}$ *Blubb, blubb, blubb, . . .* *Wichtung: 30%*

Der BubbleSort-Algorithmus zur Sortierung einer Liste von vergleichbaren Elementen in aufsteigender Reihenfolge funktioniert wie folgt:

- Betrachte, beginnend vom Anfang der Liste ($i = 0$), jeweils zwei benachbarte Elemente $a_i, a_{i+1}$ und vergleiche sie: Ist $a_i > a_{i+1}$, so ist die Vorsortierung falsch, da $a_i$ zu Unrecht vor $a_{i+1}$ in der Liste stand. Daher werden die beiden Elemente vertauscht und danach der Index $i$ um eins erhöht, d.h. im nächsten Schritt wird das größere der beiden gerade betrachteten Elemente mit dem nächstfolgenden verglichen. Falls $a_i \leq a_{i+1}$, so wird *nicht* getauscht sondern nur der Index $i$ erhöht.
- Der vorige Punkt wird wiederholt, bis die letzten beiden Elemente der Liste betrachtet worden sind. Nach einem eventuellen Tausch ist garantiert das größte Element der Liste bis an das Ende „geblubbert“ (daher der Name).
- Für eine $n$–elementige Liste, wiederhole die vorigen Punkte $n - 1$ Mal.

Man kann sich einige Optimierungen überlegen, so muß z.B. nicht immer bis zum Ende der Liste verglichen werden, wenn die größten $k$ Elemente schon an ihre korrekte Position „geblubbert“ sind.

1. Implementiere zunächst eine Funktion `bubble :: [Int] -> [Int]` die das schrittweise Vergleichen und den Vertauschungsvorgang implementiert. Der Ausdruck `bubble [5,4,7,3,1]` wird z.B. ausgewertet zu `[4,5,3,1,7]`.
2. Verwende die Funktion `bubble`, um die Sortierfunktion `bSort :: [Int] -> [Int]` zu implementieren. Wie unten gezeigt, hat `bSort [5,4,7,3,1]` das Ergebnis `[1,3,4,5,7]`.

**Sortierbeispiel:**
Die Liste `[5,4,7,3,1]` ist zu sortieren, wobei die oben erwähnte optimierte Fassung dargestellt ist:

| | |
|---|---|
| $[\boxed{5,4},7,3,1]$ | Vertauschung, da $5 > 4$ |
| $[4,\boxed{5,7},3,1]$ | keine Vertauschung, da $5 < 7$ |
| $[4,5,\boxed{7,3},1]$ | Vertauschung |
| $[4,5,3,\boxed{7,1}]$ | Vertauschung, danach 7 am richtigen Platz, neue Runde |
| $[\boxed{4,5},3,1,\mathit{7}]$ | keine Vertauschung |
| $[4,\boxed{5,3},1,\mathit{7}]$ | Vertauschung |
| $[4,3,\boxed{5,1},\mathit{7}]$ | Vertauschung, danach 5 am richtigen Platz, neue Runde |
| $[\boxed{4,3},1,\mathit{5},\mathit{7}]$ | Vertauschung |
| $[3,\boxed{4,1},\mathit{5},\mathit{7}]$ | Vertauschung, danach 4 am richtigen Platz, neue Runde |
| $[\boxed{3,1},\mathit{4},\mathit{5},\mathit{7}]$ | Vertauschung, danach 3 am richtigen Platz |
| $[\mathit{1},\mathit{3},\mathit{4},\mathit{5},\mathit{7}]$ | Fertig! |