Technische Universität München
Fakultät für Informatik
M. Bauer, D. Golias, S. Riesner,
Dr. M. Schneider

WS 2002/2003
Lösungsvorschläge zu Blatt 4
14. November 2002

Übungen zu Einführung in die Informatik I

Aufgabe 14 (Lösungsvorschlag)

Die ersten 15 Kontonummern lauten:

```
1)  1000  5)   0333  12)  1113
2)  0011  6)   0444  13)  1231
3)  0102  7)   ...   14)  1322
4)  0220  11)  0999  15)  1455
```

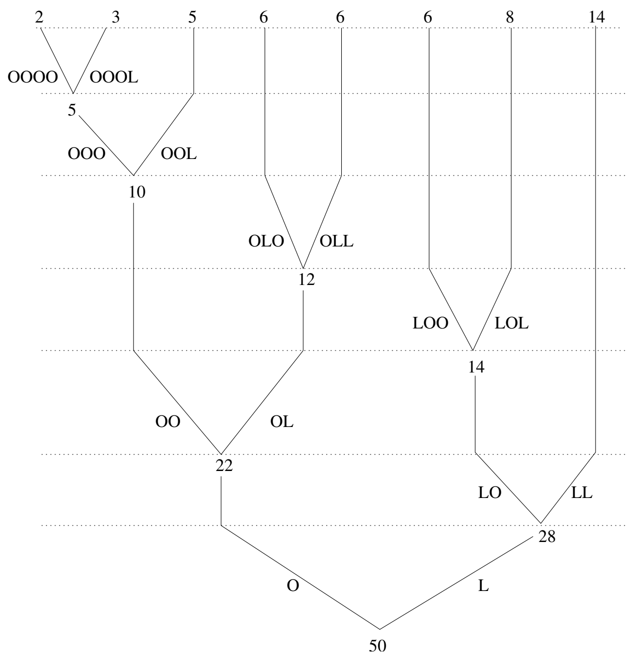
Ein Beispiel für eine falsche Kontonummer: `1327`; Korrektur: `1322`

Aufgabe 15 Huffman - Algorithmus (Lösungsvorschlag)

a) Die Ableitung der Codierung des Alphabet Z mit der Wahrscheinlichkeitsverteilung

z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
2	3	5	6	6	6	8	14

veranschaulichen wir mit Hilfe des folgenden Baumdiagramms:



Vorgehensweise:

- Aufbau des Baumdiagramms von oben nach unten, wobei jeweils die Zeichen mit minimalem $p_{z'} + p_{z''}$ zu einem neuen Zeichen z_{neu} mit der Wahrscheinlichkeit p_{neu} zusammengefaßt werden. Die gestrichelten Linien symbolisieren dabei die verschiedenen rekursiven Schritte.
- Ist die Wurzel des Baumes erreicht, so durchläuft man den Baum von unten nach oben.
- Hierbei annotieren wir die Kanten des Graphen mit der Codierung, der im jeweiligen Rekursionsschritt vorhandenen Zeichen;
- Tritt bei Traversieren des Baumes ein Verzweigung auf, so hängen wir im linken Ast an das Ende der bisherigen Codierung ein `O` (bzw. ein `L`) und im rechten Ast ein `L` (bzw. ein `O`) an.

Somit ergibt sich unter anderem folgende Codierung:

$c(z_1)$	$c(z_2)$	$c(z_3)$	$c(z_4)$	$c(z_5)$	$c(z_6)$	$c(z_7)$	$c(z_8)$
OOOO	OOOL	OOL	OLO	OLL	LOO	LOL	LL

b) Die Fano-Bedingung besagt bekanntlich, dass keine Codierung eines Zeichens z ein Präfix der Codierung eines Zeichens $z' \neq z$ ist; (Man nennt ein Wort $\mathtt{s'}$ Präfix eines Wortes $\mathtt{s}$, wenn ein Wort $\mathtt{x}$ existiert, so dass $\mathtt{s} = \mathtt{x} \circ \mathtt{s}'$.) Man erkennt bereits an der Definition des Huffman-Algorithmus, dass die Zeichenmenge in jedem Rekrusionsschritt in drei Teilmengen aufgeteilt wird und diese Teilmengen jeweils von rechts nach links unterschiedlich codiert werden. Somit kann keine Codierung Präfix einer anderen Codierung sein.

c) Die mittlere Länge L der Codierung eines Zeichens ist bekanntlich durch

$$L = \sum_{z \in Z} (p_z \cdot |l(z)|)$$

gegeben, wobei $l(z)$ die Länge der Codierung des Zeichens z bezeichnet. Für obiges Alphabet erhalten wir:

$c(z_1)$	$c(z_2)$	$c(z_3)$	$c(z_4)$	$c(z_5)$	$c(z_6)$	$c(z_7)$	$c(z_8)$
OOOO	OOOL	OOL	OLO	OLL	LOO	LOL	LL
4	4	3	3	3	3	3	2

Somit ergibt sich L = ???

Aufgabe 16 Morse-Code (Lösungsvorschlag)

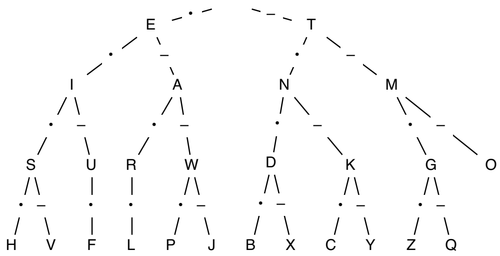
Im Jahr 1884 sendete Samuel Morse eine Nachricht mit einem Fernschreiber von Washington nach Baltimore. Er verwendete dazu einen Code der heute als *Morse-Code* bekannt ist. Dieser Code besteht aus einer Folge von langen und kurzen Tönen.

a) Suchen Sie im Internet nach einer Code-Tabelle für den Morse-Code.

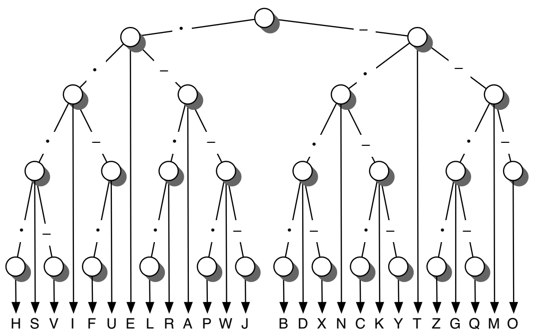
b) Was ist das Alphabet beim Morse-Code?

Das Alphabet ist $\{-, \cdot\}$, also Punkt und Strich; eigentlich kommt aber noch ein drittes Zeihe dazu, das Leerzeichen zwischen zwei Buchstaben

c) Untersuchen Sie den Morse-Code auf Einhaltung der Fano-Bedingung. Erstellen Sie dazu den kompletten Code-Baum für den Morse-Code.



Der Morse-Code erfüllt die Fano-Bedingung nicht. Beispiel: `e` entspricht ·, `r` entspricht · — ·. Nimmt man jedoch das Leerzeichen als Alphabetsymbol hinzu, so ergibt sich folgender Code-Baum:



In diesem Code-Baum ist die Fano-Bedingung erfüllt, damit ist der Morse-Code eindeutig decodierbar.

d) Warum sind die Code-Sequenzen unterschiedlich lang?

Kurze Sequenzen entsprechen häufigen Buchstaben, lange Sequenzen entsprechen selteneren Buchstaben, dadurch wird die Länge des codierten Textes kurz. Samuel Morse ging in eine Schriftsetzerei in Philadelphia und zählte dort die Buchstaben in der Schublade eines Schriftsetzers. Er fand dort 12000 mal den Buchstaben `e`, 9000 mal `t`, 8000 mal die Buchstaben `a`, `i`, `n` und `m`. Dies spiegelt sich in der Länge der Sequenzen für die einzelnen Buchstaben wieder.

Aufgabe 17 Lispbäume in OCaml (Lösungsvorschlag)

a) Der wesentliche Unterschied besteht darin, dass die inneren Knoten des Baumes keine Informationen tragen.

b) Geben Sie eine rekursive `OCaml` Daten-Typ-Deklaration für Lispbäume an. Die Menge M soll hierbei polymorph gehalten werden.

```
# type 'm lisptree = Atom of 'm | Node of 'm lisptree * 'm lisptree;;
type 'a lisptree = Atom of 'a | Node of 'a lisptree * 'a lisptree
```

c) Implementieren Sie eine OCaml-Funktion `occurrences`, welche ermittelt, wie oft eine Markierung m in einem Baum z vorkommt.

```
let rec occurrences m z = match z with
        | Atom(x) -> if (x = m) then 1 else 0
        | Node(l,r) -> (occurrences m l) + (occurrences m r)
;;
```

d) **(H)** Implementieren Sie eine OCaml-Funktion `isBalanced`, welche ermittelt, ob sich ein gegebener Lispbaum in Balance befindet. Dabei gilt: Ein Baum ist ausbalanziert, wenn die Differenz zwischen größtem und kleinstem vorkommendem Knoten-Niveau nicht mehr als 1 beträgt.

Zunächst eine Funktion füer das maximale Niveau:

```
let rec max_niveau lt = match lt with
        | Atom(m) -> 0
        | Node(x,y) -> 1 + max (max_niveau x) (max_niveau y)
;;
```

Dann eine für das minimale Niveau:

```
let rec min_niveau lt = match lt with
        | Atom(m) -> 0
        | Node(x,y) -> 1 + min (min_niveau x) (min_niveau y)
;;
```

Und schließlich die isBalanced Funktion:

```
let isBalanced lt =
if (max_niveau(lt)-min_niveau(lt)) <= 1 then true else false;;
```

e) **(H)** Testen Sie die vorstehenden Funktionen an wenigstens einem selbstgewählten Beispiel-Lispbaum.

Do it yourself ...