Übungsblatt 11 Daniel Funk IT-S 23.01.2019

Aufgabe 11.1

a) Huffman-Codierung:

0.3	0.2	0.15	0.15	0.1	0.1
a	b	c	d	e	f

Abb.1 Codebaum (auf Seite 2)

a	b	c	d	e	f	Zeichen
0.3	0.2	0.15	0.15	0.1	0.1	rel. Häufigkeit p_i
1	3	3	3	4	4	Länge l_i

->0.3*1+0.2*3+0.15*3+0.15*3+0.1*4+0.1*4= 2.6 Bit

b) badecfa -> 001 1 011 0000 010 0001 1

c) Shannon-Fano-Codierung:

1. Abb.2 Codebaum (auf Seite 3)
2.

a	b	c	d	e	f	Zeichen
0.3	0.2	0.15	0.15	0.1	0.1	rel. Häufigkeit
2	3	3	2	3	3	Länge

-> 0.3*2+0.2*2+0.15*3+0.15*2+0.3*1+0.3*1= 2.55 Bit

3. badecfa -> 010 00 10 110 001 111 00

Commented [DF1]: Keine Mengenzeichen {} , da Zahlen doppelt vorkommen. Besser: Tuppel (a, b, c, d, e, f) (0.3, 0.2, 0.15, 0.15, 0.1, 0.1)

Aufgabe 11.2

a) Code-Redundanz:
Bedeutet, dass in einer Nachricht zusätzliche überflüssige Elemente mitgeschickt werden, die keine weiter Information enthalten, um die eigentliche Nachricht zu unterstützen. Theoretisch könnte dies so aussehen, dass man jedes Bit seiner Nachricht ein zweites Mal mitschickt, falls Informationen unterwegs verloren gehen, die Nachricht trotzdem sicher ankommt.

Hamming-Abstand:
Der Hamming-Abstand gibt an um wie viel Bits sich zwei gleichlange Bitwörter unterscheiden. Bei einem Hamming-Abstand von d=2 unterscheiden sich bei zwei gleichlangen Bitwörtern 2 Bits, z.B.: 00110 -> 10100
Man weiß somit dann, dass z.B. 00110 -> 11100 bei einem Abstand von d=2 nicht korrekt sein kann, sich die beiden Bitwörter um 3 Bits unterscheiden.

b) Um einen 1-Bit-Fehler zu erkennen ist ein Hamming-Abstand von d=2 erforderlich. Um einen 3-Bit-Fehler zu erkennen wäre deshalb ein Hamming-Abstand von mindesten d=6 nötig.

c) Um 3-Bit-Fehler sicher beheben zu können ist nach d=2*k+1 (k =Bits) ein Hamming-Abstand von d=7 notwendig: d=2*3+1=7. Bei einem 1-Bit-Fehler d=3.

Aufgabe 11.3

a) Eine 1-Bit-Fehlererkennung ist möglich, da bei einer geraden Anzahl gesetzter Bit ein Paritätsbit gesetzt wird und man somit eine ungerade Anzahl von gesetzten Bits erhält, bekommt man jedoch eine grade Anzahl von Bits, obwohl das Paritätsbit nicht gesetzt ist, so liegt ein 1-Bit-Fehler vor.
2-Bit-Fehler sind nicht zu erkennen, da wenn man eine bei gerade Anzahl an gesetzter Bits 2-Bit-Fehler hat, diese wieder zu einer geraden Anzahl von gesetzten Bits führt. Bsp: 01011100**1** ->11000000**1** Das Paritätsbit unterscheidet nur zwischen gerader und ungerader Parität.

b) 0010010**1,** 1111111**0**, 1010101**0**, 0001000**1**

c) 00100101: eine gerade Anzahl an Bits könnte gesetzt sein (2, 4, 6 Bits).
   11111111: Paritätsbit könnte für ungerade Parität gesetzt werden.
d) 00100101, 11111111: Wenn wir das Paritätsbit für gerade Parität haben, dann liegt bei dem ersten Codewort kein Fehler vor, aber bei dem zweiten Codewort wäre das Paritätsbit falsch gesetzt. Wenn das Paritätsbit bei ungerader Parität gesetzt wird wäre der Fehler umgekehrt.

Aufgabe 11.4

a) ISBN 3-528-05783-6:
   3*10+5*9+2*8+8*7+0*6+5*5+7*4+8*3+3*2+6=236 236:11=10 Rest 5 ->ungültig
   ISBN 3-528-05738-6:
   3*10+5*9+2*8+8*7+0*6+5*5+7*4+3*3+8*2+6=231 231:11=11 Rest 0 ->gültig

b) 281234554321x:

   281234554321x
   1313131313131 Gewichte
   241632554923x Produkte Modulo 10

   2+4+1+6+3+2+5+5+4+9+2+3+x=42+x (42+x):10 muss Rest 0 ergeben, damit die GTIN gültig ist, deshalb muss die Prüfziffer x=8 sein: (42+8):10=5 Rest 0 ->gültig

Abbildung 1 Huffman-Codebaum

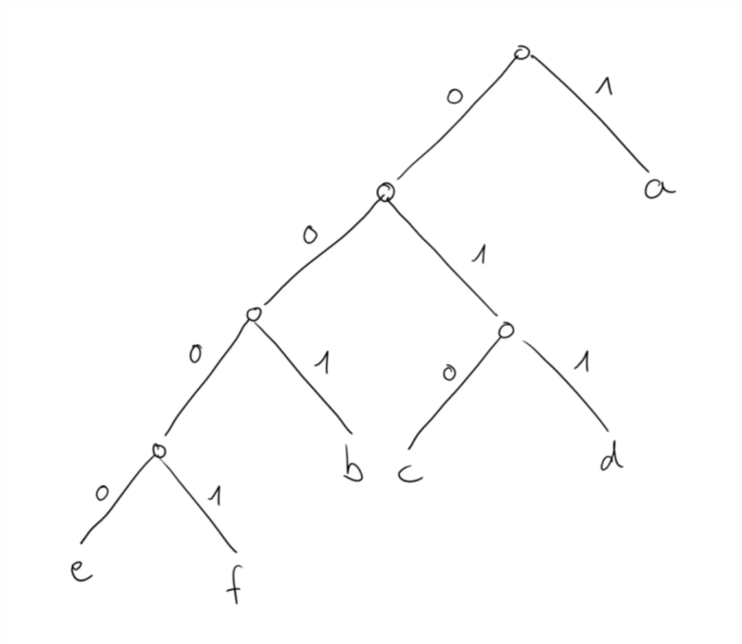

Abbildung 2 Shannon-Fano-Codebaum