

Huffman Codes und Datenkompression

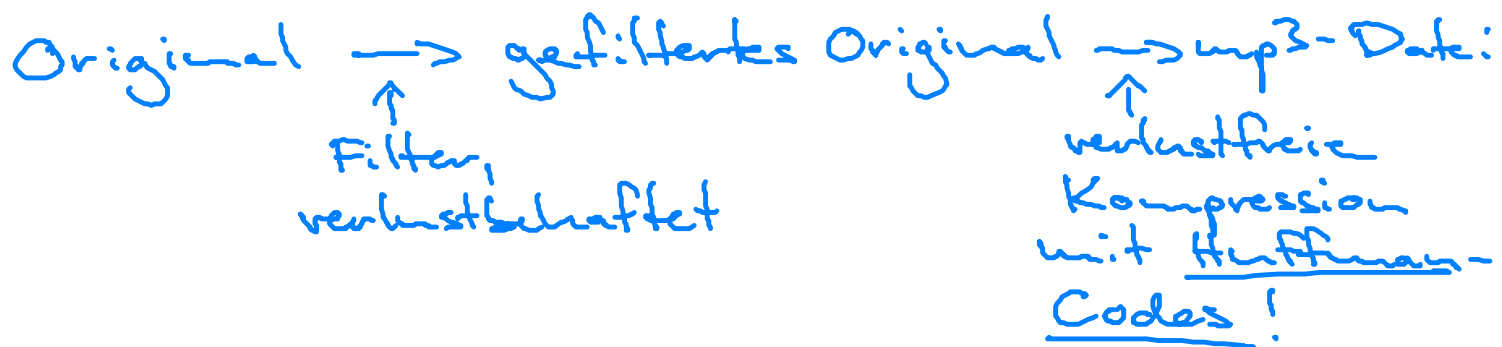
Ziel: Speichere Daten unter
Verwendung von möglichst
wenig Speicherplatz

Originalformat $\xrightarrow{\quad}$ komprimiertes Format
 \uparrow

Kompression

- verlustfreie Datenkompression, d.h. Originaldaten können vollständig aus komprimierten Daten zurückgewonnen werden.
- verlustbehaftete Datenkompression:
z.B. Audiodaten (z.B. mp3-Format)
Videodaten

zum mp3-Format



Ersparnis c.a. 20%!

In Folgenden: verlustfreie Kompression

intuitive Vorstellung: Komprimierung von Textdateien

Annahme: Datei besteht aus einzelnen

Zeichen c aus Alphabet (Zeichensatz) C ,

z.B.: $C = \text{ASCII-Zeichensatz}$

Unicode - -

Java - -

(reservierte Worte
 $\hat{=}$ 1 Zeichen)

Gängige Zeichensätze verwenden sogen.

Blockcodes (d.h. Repräsentation von
Zeichen als 0-1-Strings fester Länge)

z.B. ASCII : Länge 8

Unicode : Länge 16

weiteres Beispiel (vereinfacht):

a	→	000
b	→	001
c	→	010
d	→	011
e	→	100
f	→	101
g	→	110
-	→	111

Blockcodes erlauben sehr einfache

Codierung und Decodierung

Bsp: f a u g a g a w ...

101 000 111 ...

kodierte Datei:

f a - g a g a - g a f f - f e g e
101 000 111 110 000 110 000 111 110 000 101 101 111 101 100 110 100

Decodieren: je 3 Bits $\hat{=}$ Zeichen $\in C$

101 000 111 110 000 110 000 111 110 000 101 101 111 101 100 110 100

Grundidee der Kompression:

- Häufigkeitsanalyse der Zeichen c in der Originaldatei:
- Konstruiere Code variabler Länge, in dem häufige Zeichen kurze Codewörter bekommen und seltene Zeichen lange Codewörter

Bsp: ein Code variabler Länge

a	→	0
b	→	1
c	→	10
d	→	11
e	→	100
f	→	101
g	→	110
_	→	111

Codewörter: jedem Zeichen aus C ist ein Codewort zugeordnet.

Kodierung:

$\overset{f}{101} \overset{a}{0} \overset{_}{111} \overset{g}{110} \overset{a}{0} \overset{g}{110} \overset{a}{0} \overset{_}{111} \overset{g}{110} \overset{a}{0} \overset{f}{101} \overset{f}{101} \overset{_}{111} \overset{f}{101} \overset{e}{100} \overset{g}{110} \overset{e}{100}$
 $\underbrace{\quad\quad}_{bd?}$

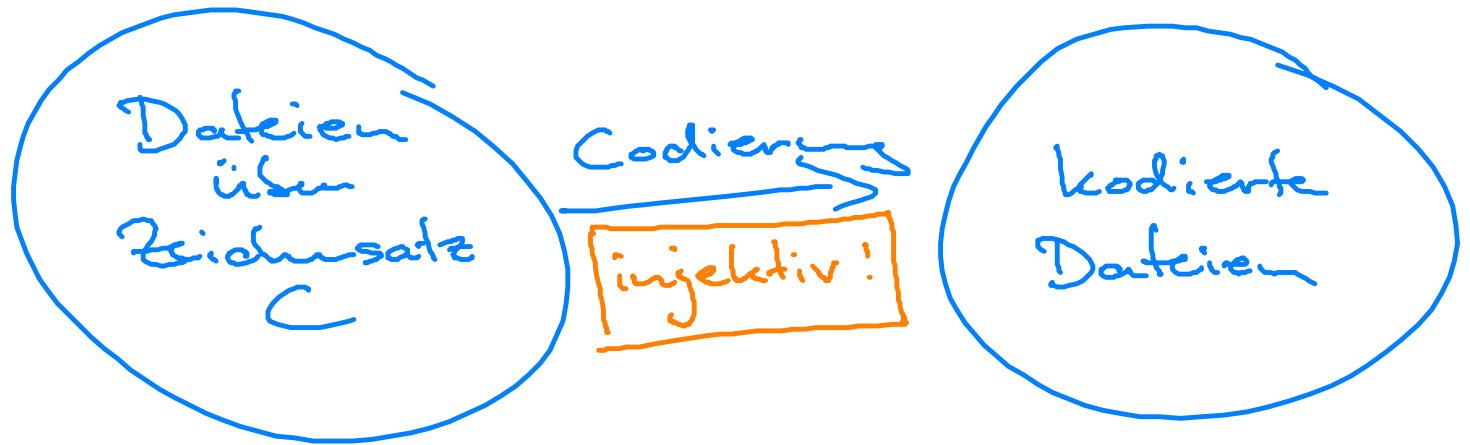
weitere Beispiel:

abba \neq ag
0110 = 0110

Problem: nicht
eindeutig
dekodierbar!

Codes variabler Länge sind u.U.
nicht eindeutig dekodierbar.

Def.: Ein Code heißt eindeutig-dekodierbar : \leftrightarrow verschiedene Originaldateien führen zu verschiedenen kodierten Dateien



Eine wichtige Klasse von eindeutig dekodierbaren Codes sind die sogen. Präfixcodes:

Präfixcodes:

Def: Ein Code heißt Präfixcode, wenn kein Codewort als Präfix eines anderen Codeworts auftritt.

Bsp:

a	→	0
b	→	1
c	→	10
d	→	11
e	→	100
f	→	101
g	→	110
h	→	111

kein Präfixcode, da 1 (Codewort für b) Präfix ist von 10 (Codewort für c)

b) Jeder Blockcode ist ein Präfixcode

Lemma: Präfixcodes sind eindeutig
dekodierbar.

Aber: Präfixcodes sind nicht die einzigen
eindeutig dekodierbaren Codes

Bsp:

a 1

b 10

c 00

10000 11000000 10010001
a c c a b c c a c b c a

Im folgenden betrachten wir jedoch nur
Präfixcodes, da diese eine besonders
schöne Struktur haben und sehr einfach
dekodiert werden können.

Lemma: Präfixcodes lassen sich mit binären
Bäumen identifizieren:

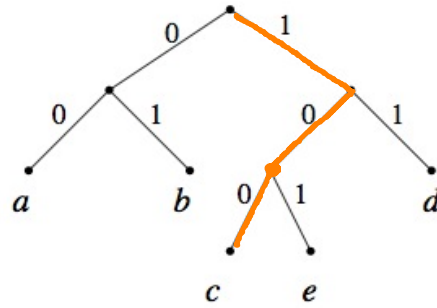
Zeichen $c \in C \iff$ Weg in Baum von
Wurzel bis Blatt ($\hat{=}$ Zeichen)

Kanten auf dem Weg:
nach links $\hat{=} 0$
" rechts $\hat{=} 1$

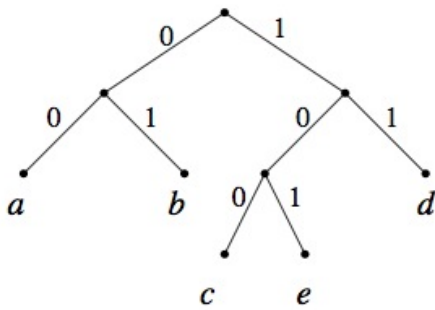
Bsp:

a	→	00
b	→	01
c	→	100
d	→	11
e	→	101

Baum:



Präfix garantiert, dass die Zeichen bei Konstruktion des binären Baumes nur in Blättern auftauchen.



a : 00
b : 01
c : 100
d : 11
e : 101

Speicherplatz (#bits) der codierten Datei, die mit Präfixcode T (binärer Baum) codiert wurde:

$$B(T) = \sum_{c \in C} f(c) \cdot h_T(c)$$

Häufigkeit von c im Originaltext

#bits für Codierung von c

Suche nach „besten“ Präfixcode führt zu dem folgenden Optimierungsproblem:

Um eine Datei mit Zeichensatz C und Häufigkeiten $f(c)$, $c \in C$, optional mit einem Präfixcode zu codieren,

suchen wir einen binären Baum T , dessen Blätter den Zeichensatz C entsprechen, der die Größe

$$B(T) = \sum_{c \in C} f(c) \cdot \lg_T(c)$$

minimiert, d.h. $B(T) \leq B(T')$ für alle Präfixcodes T' .

! Anwendungsproblem

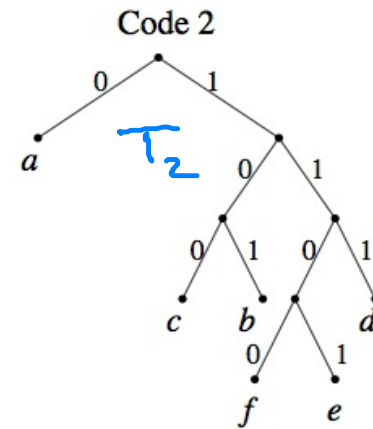
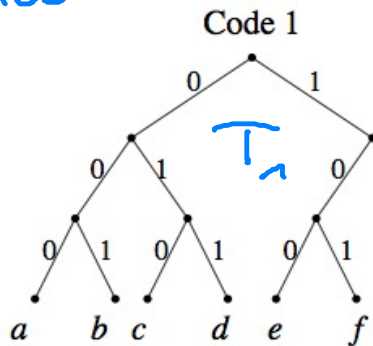
! mathematische Formulierung des Problems

Dieses T heißt dann optimaler Präfixcode.

Beispiel für verschiedene Präfixcodes:

Zeichen c	$f(c)$ in 1000	Code 1	Code 2
a	45	000	0
b	13	001	101
c	12	010	100
d	16	011	111
e	9	100	1101
f	5	101	1100

$\Sigma 100$



$$B(T_1) = 300.000 \text{ bits}$$

$$B(T_2) = 1000 \cdot (45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \text{ bits} = 224.000 \text{ bits}$$

Huffman Algorithmus

Input: Zeichensatz C , Häufigkeiten $f(c)$

Output: Optimaler Präfixcode T

Algorithmus 3.1 (Huffman Algorithmus)

1. Fasse jedes Zeichen $c \in C$ als einelementigen Baum auf und füge es in eine Priority Queue Q ein, wobei die Häufigkeit $f(c)$ als Schlüsselwert dient.
2. Solange Q mehr als einen Baum enthält:
 - Wähle die beiden Bäume T_1 und T_2 mit den kleinsten Häufigkeiten (muss nicht eindeutig sein).
 - Entferne sie aus Q .
 - Konstruiere einen neuen Baum aus den T_1, T_2 als Teilbäume unter einer neuen Wurzel und gebe ihm die Häufigkeit $f(T_1) + f(T_2)$.
 - Füge diesen Baum in Q ein.
3. Gebe (den einzig übrig gebliebenen Baum) T in Q zurück. Dieser Baum (der so genannte Huffman Baum oder Huffman Code) ist ein optimaler Präfixcode.



