

Morse Code

A	.-	P	..-	1
B	...-	Q	---	2
C	..-.	R	.-.	3	...-
D	..-	S	...	4
E	.	T	-	5
F	..-.	U	..-	6
G	--.	V	...-	7
H	W	..-	8
I	..	X	..-	9
J	.-..	Y	..-	0
K	-.-	Z	...-		
L	.-..	,	--..		
M	--	.	..--		
N	-.	?	..--		
O	---	-		

ASCII-Tabelle

Bits 6543	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
	Bits 210	NUL	BS	DLE	CAN	(0	8	@	H	P	X	`	h	p	x
000	SOH	TAB	DC1	EM	!)	1	9	A	I	Q	Y	a	i	q	y
001	STX	LF	DC2	SUB	"	*	2	:	B	J	R	Z	b	j	r	z
010	ETX	VT	DC3	ESC	#	+	3	;	C	K	S	[c	k	s	{
011	EOT	FF	DC4	FS	\$,	4	<	D	L	T	\	d	l	t	
100	ENQ	CR	NAK	GS	%	-	5	=	E	M	U]	e	m	u	}
101	ACK	SO	SYN	RS	&	.	6	>	F	N	V	^	f	n	v	~
110	BEL	SI	ETB	US	'	/	7	?	G	O	W	_	g	o	w	←
111																

BS=Backspace
CR=Carriage Return
EM=End of Media

ESC=Escape
FF=Form Feed
BEL=Piepston

Coding

Zwei Probleme:

- Übertragung (oder Speicher) kann unzuverlässig sein (verrauscht)
 - fehlererkennende Codes => z.B. Paritätsbit
wenn falsch, dann wiederholen
 - fehlerkorrigierende Codes => z.B. alles dreimal senden
- Wie Übertragung von kontinuierlichen Signalen statt Symbolen
 - => Vektorquantisierung

Coding

Fehlererkennende Codes:

Idee: feststellen, ob die Übertragung fehlerhaft war;
wenn ja, dann Übertragung wiederholen bis fehlerfrei

Mögliche Fehlerprüfung mit Paritätsbit:

- Zähle Anzahl n der Einsen in den Übertragenen Daten
- Füge zusätzliches Bit b an mit $b = \begin{cases} 0 & \text{wenn } n \text{ gerade} \\ 1 & \text{wenn } n \text{ ungerade} \end{cases}$
=> Summe aller Einsen immer gerade (*even parity*)
- wenn Empfänger ungerade Zahl Einsen empfängt
ist ein Fehler aufgetreten

Coding

Problem: Gesendet: 010101 1
Empfangen: 100101 1
Daten Paritätsbit

Wenn 2 Bits (oder beliebige gerade Anzahl Bits) fehlerhaft sind
stimmt die Parität
=> Fehler wird nicht erkannt

Wenn p die Wahrscheinlichkeit für das „Kippen“ eines Bits ist,
ist

$$\begin{aligned} \text{Wahrscheinlichkeit, daß 0 Bit falsch} &= (1-p)^n \\ k \text{ Bit falsch} &= \binom{n}{k} p^k (1-p)^{n-k} \end{aligned}$$

Coding

gerade oder ungerade Parität?

- Aufwand für berechnung ist gleich
- Vorteil ungerade Parität: Leitung „tot“ => Empfang 000...0 wird
als Fehler erkannt

Berechnung:

- meist in Hardware (einfache Schaltkreise)

- als $O(\log n)$ -Algorithmus:

$$\begin{aligned} &\text{parität}(x_1, x_2, \dots, x_n) \\ &= \text{parität}(x_1, x_2, \dots, x_{n/2}) \text{ XOR } \text{parität}(x_{n/2+1}, \dots, x_n) \end{aligned}$$

Coding

Wann sollte Parität berechnet werden, d.h. für welche Blöcke?

- kleine Blöcke => höhere Genauigkeit
- große Blöcke => höhere Effizienz

also Kompromiß zwischen Genauigkeit und Effizienz, je nach Aufgabe

Andere Fehlererkennende Codes:

- Telegramm beginnt mit Anzahl aller Wörter
- Summe aller Übertragenen Zahlen (*checksum*)

Coding

Annahme, daß jedes Bit (Symbol) gleich wahrscheinlich kippt, ist nicht immer angebracht:

- fehlerhafte Leitung für ein bestimmtes Bit
- typisch menschliche Fehler (Vertauschungen)
„Beispiel“ => „Beispiel“

Verwendung von gewichteten Codes:

berechne laufende Summe der Summen

w	w	w
wx	w+x	2w+x
wxy	w+x+y	3w+2x+y
wxyz	w+x+y+z	4w+3x+2y+z

erkennt auch Vertauschungen

Coding

Fehlerkorrigierende Codes:

Idee: Versenden zusätzlicher Information, die - falls Fehler erkannt - genügt, um Fehler zu beheben.

Einfache Methode: sende alles drei mal

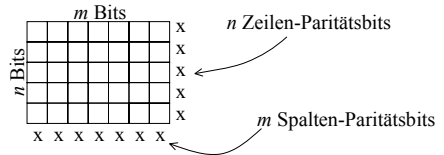
Fehler, wenn nicht alle drei Kopien identisch;
wenn nur zwei Kopien identisch, werden sie
als korrekt angesehen

Problem: Vervielfachung der zu übertragenden Daten

Coding

Fehlerkorrigierende Codes:

Rechteck-Codes:

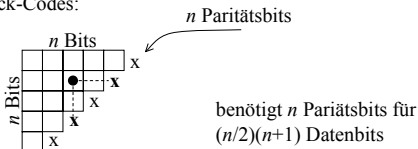


Wenn Bit (i,j) „kippt“ stimmt das i -te Zeilenparitätsbit
und das j -te Spaltenparitätsbit nicht.

Coding

Fehlerkorrigierende Codes:

Dreieck-Codes:



benötigt n Paritätsbits für
 $(n/2)(n+1)$ Datenbits

Idee läßt sich erweitern (z.B. mehrdimensionale Anordnung
der Datenbits statt zweidimensionale) => kubische Codes

Kompression

Komplexität nicht nur für Algorithmen und Probleme,
sondern auch für Daten (Kolmogorov).

z.B. die Daten 0110100111011010010001101110100101
sind komplexer
als die Daten 0100100010000100000100000010000000

Kolmogorov-Komplexität beschäftigt sich mit der Frage:
„Welches ist das kleinste Programm,
das als Ausgabe eine gegebene Bitfolge (=Zahl) ausgibt?“

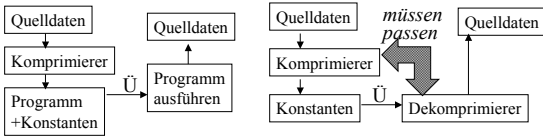
- Meist enthalten solche Programme viele Konstanten.
- Oft werden nur die Konstanten als Kompression angesehen,
- dann muß aber das Kompressionsprogramm bekannt sein.

Wozu Kompression?

Kompression

- spart Speicherplatz auf Datenträgern
- spart sogar Speicherplatz im RAM (RAM-Doubler, etc.)
- verkürzt Zeiten für Datentransport
- verkürzt manchmal auch Ausführungszeiten von Programmen
- belastet Rechnernetzwerke weniger
- erhöht Kapazität von Übertragungsleitungen niedriger Bandbreite

Zwei Varianten:



Kompression

Eine einfache Kompression: Run-Length Encoding

Gegeben sei z.B. die Datenfolge

13 13 13 13 5 5 5 5 5 5 5 5 5 5 26 26 26 26 12 7 7 7 7 7

bestehend aus 26 Zeichen.

Wenn dasselbe Zeichen mehrmals hintereinander kommt, bietet sich an, nur das Zeichen und seine Anzahl abzuspeichern:

5 13 11 5 4 26 1 12 5 7

Jetzt nur noch 10 Zeichen. 16 Zeichen gespart, Kompression = 62%

Kompression

Run-Length Encoding funktioniert nicht immer:

z.B. 2 2 2 3 2 2 2 5 7 5 2 2 => 3 2 1 3 3 2 1 5 1 7 1 5 2 2

vorher 12 Zeichen nachher 14 Zeichen

Worst-Case: Komprimierte Datei ist doppelt so groß wie Original

Typische Einsatzgebiete für RLE:

- Grafiken (oft viele gleichartige Pixel hintereinander, v.a. schwarz-weiß)

- Übertragung von Faxen

Kompression

Sei z.B. folgender Bit-Code für Symbole gegeben:

leer	000
E	001
I	010
N	011
S	100
T	101
-	110
.	111

Der Text „EINE SEE-ENTE“

würde durch folgende 39 Bit dargestellt:

001 010 011 001 000 100 001 001 110 001 011 101 001

Wir sehen: E kommt 6 mal vor, S nur einmal.

Idee: verwende im Code weniger Bits für oft vorkommende Zeichen, dafür mehr für seltene Zeichen.

Kompression

Was wäre z.B. mit folgendem Code:

leer	000
E	1
I	010
N	011
S	100
T	101
-	110
.	111

statt

leer	000
E	001
I	010
N	011
S	100
T	101
-	110
.	111

Unser Text wird zu

1 010 011 1 000 100 1 1 110 1 011 101 1

nur noch 27 Bits

Aber Dekompression geht nicht mehr:

1 010 011 1 000 ...
E I N E

oder 1 010 011 1 000 ...
T ?

Dieser Code ist also ambig.

Kompression

Bedingung an Code:

Der Code keines Symbols darf Präfix des Codes eines anderen Symbols sein.

leer	00000
E	1
I	001
N	011
S	010
T	00001
-	00010
.	00011

Der Text „EINE SEE-ENTE“

würde dann **eindeutig** dargestellt als

1 001 011 1 00000 010 1 1 00010 1 011 00001 1

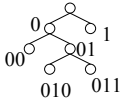
Das sind 33 Bit (6 weniger als 39, Ersparnis 15%).

Frage: gibt es einen Code, der weniger als 33 Bit benötigt?

Kompression

Die „Präfixfreiheit“ läßt sich beschreiben mit Hilfe der Blätter eines Baumes:

Numerierung der Knoten: Wurzel = leer
 linker Nachfolger von xxx = xxx0
 rechter Nachfolger von xxx = xxx1



Ein Knoten A ist genau dann ein Präfix des Knotens B, wenn B ein Nachkomme von A ist.

Weil Blätter keine Nachkommen haben, ist ein Code, der nur die Baumnumerierung der Blätter benutzt, präfixfrei.

Kompression

Formale Problemdefinition

gegeben: Text, der n verschiedene Symbole C_1, C_2, \dots, C_n enthält (Reihenfolge der Symbole ist egal)
 Symbol C_i kommt insgesamt f_i mal vor.

gesucht: Codierung $C_i \rightarrow S_i$ der Symbole durch präfixfreie Bitfolgen S_1, S_2, \dots, S_n der Längen s_1, s_2, \dots, s_n , so daß $\sum_{i=1}^n s_i f_i$ minimiert wird.

Kompression

Kompression: klar: $C_i \rightarrow S_i$

Dekompression:

bitfolge = leer
index = 0
 solange *index* < *länge(text)*
 hänge *text(index)* an *bitfolge* an
 falls $\exists i: \textit{bitfolge} = S_i$
 dann gib C_i aus
 bitfolge = leer
 index++

1001011100000010...
 1=E
 001011100000010...
 0=?
 001011100000010...
 00=?
 001011100000010...
 001=I
 011100000010...
 0=?
 011100000010...
 01=?
 011100000010...
 011=N
 100000010...
 1=E
 ...

Kompression

Induktive Konstruktion des Huffman-Codes

Behauptung: Wenn $f_i > f_j$ dann muß $s_i \leq s_j$ sein.

Beweis: wäre $s_i > s_j$, dann könnte durch Vertauschen von S_i und S_j der Wert von $\sum_{i=1}^n s_i f_i$ verkleinert werden.

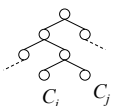
Konstruktionsidee:

Wir bauen induktiv einen Baum auf, dessen Blattnumerierung der gesuchte Huffman-Code ist.

Kompression

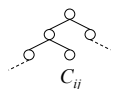
Seien f_i und f_j die beiden kleinsten Häufigkeiten.

Dann ist offensichtlich, daß der gesuchte Baum so aussehen muß:



Alle Blätter sind höchstens so tief wie C_i und C_j .
 Dabei haben C_i und C_j den selben Vorgänger.

Induktion: Ersetzen wir alle C_i und C_j durch ein neues Symbol C_{ij} im Text. Dann muß der Baum so aussehen:



Es ist $f_i + f_j \leq f_k + f_l$ für alle anderen Paare k, l

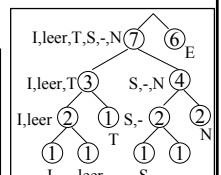
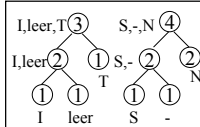
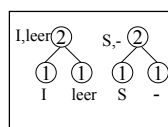
Kompression

Algorithmus für Erzeugung des Huffman-Codes:

solange noch 2 uncodierte Symbole in Menge
 ersetze C_i und C_j mit kleinsten f_i und f_j durch C_{ij} mit $f_i + f_j$
 definiere C_i und C_j als Nachfolgeknoten von C_{ij} .

Beispiel: „EINE SEE-ENTE“

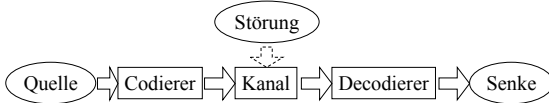
$f_E=6, f_I=1, f_N=2, f_{leer}=1, f_S=1, f_T=1, f_{-}=1$



Also: $S_E=1, S_I=0000, S_N=011, S_{leer}=0001, S_S=0100, S_T=0101, S_{-}=001$
 „EINE SEE-ENTE“ = 1000001110001010011010110110011 (31 Bits)

Informationstheorie

Ein typisches Modell für ein Signalübertragungssystem:



- An der Quelle fallen Daten in irgendeiner Form an.
- Der Codierer übersetzt die Daten in diskrete Symbole (meist 0/1).
- Die Symbole werden über den Kanal geschickt und evtl. gestört.
- Der Decodierer transformiert den empfangenen Code zurück zu den ursprünglichen Daten
- An der Senke sollen möglichst die gleichen Daten ankommen, wie sie an der Quelle entstanden sind.

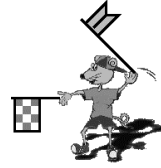
Informationstheorie

Coding

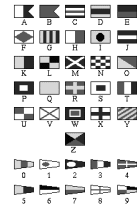


Fragestellung beim Coding:

Wie stellt man am besten Information in Form von Symbolen S_1, S_2, \dots, S_q dar?



Die Problematik gab es schon sehr früh, als Kommunikation mit Flaggen



Informationstheorie

Was ist Information?

Ähnlich wie bei der Komplexität eines Textes gilt auch für die Information, daß z.B.

ACBCACBBAABCACBBCCA

mehr Information enthält als

AAAAAABBAAAAACAAA

Wir wollen Information eines Symbols S definieren als

Welchen Wissenszuwachs hat ein Empfänger, wenn er das Symbol S empfängt.

Informationstheorie

Was ist Information?

Welchen Wissenszuwachs hat ein Empfänger, wenn er das Symbol S empfängt.

In anderen Worten: Wie groß ist die **Ungewissheit** über das zu empfangende Symbol.
Wie groß ist die **Überraschung**, wenn es kommt.

z.B. Empfänger ist 100% sicher: „als nächstes Symbol kommt ein A.“
Also kommt ein A.
Empfänger weiß auch nicht mehr als er vorher schon wußte.

Informationstheorie

Was ist Information?

Die Ungewißheit über ein zu empfangendes Symbol hängt also mit der Wahrscheinlichkeit dieses Symbols zusammen.

Die Überraschung, wenn Symbol S_i empfangen wird, ist umso größer, je unwahrscheinlicher das Symbol ist:

$$\text{Überraschung} = \frac{1}{\text{Wahrscheinlichkeit}(S_i)} = \frac{1}{p_i}$$

$p_i=1 \Rightarrow$ keine Überraschung, kein Informationsgewinn.

Informationstheorie

Bedingungen an ein Informationsmaß $I(p_i)$:

- $p_i=1$, keine Überraschung $I(1) = 0$
- p_i sehr klein, große Überraschung $I(\epsilon) \gg 0$
- Überraschung ist nie negativ $I(p_i) \geq 0$
- Kontinuität in p
- Überraschung zweier unabhängiger Ereignisse ist die Summe der Einzelüberraschungen $I(p_1 p_2) = I(p_1) + I(p_2)$
- Überraschung n unabhängiger Ereignisse gleicher Wahrscheinlichkeit p ist n -fache Einzelüberraschung $I(p^n) = nI(p)$

Funktion, die alle Bedingungen erfüllt: $I(p) = k \cdot \log_b(p)$

üblich: $k = -1, b=2$, also $I(p) = -\log_2(p) = \log_2(1/p)$

Informationstheorie

Was ist Information?

Vor einem Ereignis: Ungewissheit
 Beim Ereignis: Überraschung
 Nach dem Ereignis: Information

Ungewissheit \equiv Überraschung = Information

Der Informationsgewinn durch den Empfang eines Symbols S_i , das mit der Wahrscheinlichkeit p_i ankommt ist

$$I(p_i) = -\log_2(p_i) \quad \text{Einheit der Information ist das „Bit“}$$

D.h. z.B.: Kommt ein Symbol mit Wahrscheinlichkeit 25% an, dann ist der Informationsgewinn 2 Bit.

Informationstheorie

Was ist Information?

Wir empfangen nun mehrere verschiedene Symbole

S_1, S_2, \dots, S_q mit den Wahrscheinlichkeiten p_1, p_2, \dots, p_q

Was ist deren durchschnittlicher Informationsgehalt?

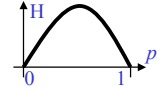
Symbol trägt zum Gesamtdurchschnitt bei: $p_i \cdot I(S_i) = p_i \cdot \log_2(1/p_i)$

Der Gesamtdurchschnitt ist

$$\lim_{p_i \rightarrow 0} H = \lim_{p_i \rightarrow 1} H = 0 \quad H = \sum_{i=1}^q p_i \cdot \log_2\left(\frac{1}{p_i}\right) = -\sum_{i=1}^q p_i \cdot \log_2 p_i$$

Entropie

$q=2$
 $(p_2=1-p_1)$



Kompression

Huffman-Code ignoriert Reihenfolge der Symbole \Rightarrow suboptimal

Beispiel: Text = ABC-DEF ABC-DEF ABC-DEF

Keine Einsparung durch Huffman (bei Codierung mit 3 Bits/Symbol)

Aber offensichtlich steckt im Text nicht mehr Information als 3 mal ABC-DEF.

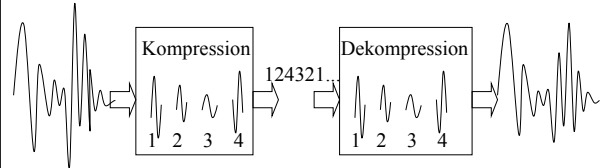
Idee: Fasse mehrere Symbole zu einem Vektor zusammen, betrachte Vektoren als neue Symbole.

Aber: Was wenn Symbole nicht diskret sondern $\in \mathbf{R}$?

Kompression

Wenn Symbole reelle Zahlen/Vektoren sind, ist es meist nicht wichtig, die exakt gleichen Werte nach der Dekompression zu erhalten.

z.B. MiniDisc:



Die Ausgabe ist nicht identisch mit der Eingabe. Aber ähnlich.

Kompression

Wir sprechen von

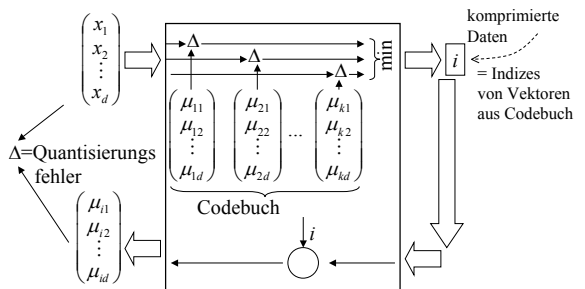
verlustbehafteter Kompression (lossy compression)
 wenn durch das Komprimieren/Dekomprimieren die ursprünglichen Daten nicht mehr rekonstruiert werden können.

Einsatzgebiete für verlustbehaftete Kompression:

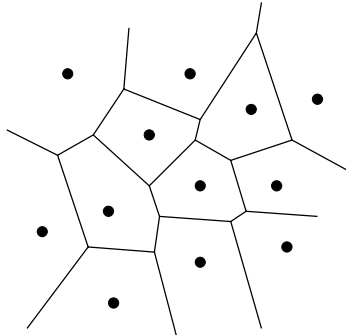
- Audio- und Video-Signale (μ -law, MPEG)
- Bilder (JPEG)
- physikalische Meßdaten

Kompression

Verlustbehafteter Kompression durch Vektorquantisierung



Kompression



Vektorquantisierung teilt den zu komprimierenden Raum in Voronoi-Regionen ein.

Kompression

Problemstellung:

gegeben: n Vektoren v_1, v_2, \dots, v_n aus \mathbf{R}^d

gesucht: k Vektoren $\mu_1, \mu_2, \dots, \mu_k$ aus \mathbf{R}^d ,

so daß eine Abbildung $f: \{1, \dots, n\} \rightarrow \{1, \dots, k\}$ gefunden werden kann, bei der der gesamte Quantisierungsfehler

$$\sum_{i=1}^n |v_i - \mu_{f(i)}| \text{ minimiert wird.}$$

Globales Minimum ist für sehr große n sehr aufwendig zu finden, => iterative Näherungslösungen

Kompression

Der k -Mittelwerte-Algorithmus

Idee:

- Beginne mit irgendwelchen $\mu_1, \mu_2, \dots, \mu_k$
- berechne Quantisierungsfunktion
- verschiebe jedes μ_j , so daß Quantisierungsfehler kleiner wird
- iteriere so lange bis Quantisierungsfehler
 - nicht mehr kleiner wird, oder
 - klein genug ist, oder
 - keine Zeit mehr zum Weitermachen

Kompression

Der k -Mittelwerte-Algorithmus

Die Quantisierungsfunktion sei $f(v_i) = \operatorname{argmin}_j |\mu_j - v_i|$ (der v_i am nächsten liegende Codevektor)

Der Quantisierungsfehler durch Codevektor j ist dann

$$\sum_{i: f(i)=j} |\mu_j - v_i|$$

Es gilt nun: Der Quantisierungsfehler ist minimal, wenn

$$\mu_j = \frac{1}{r} \sum_{i: f(i)=j} v_i \quad \text{mit } r = |\{i: f(i)=j\}|$$

Wir ersetzen also μ_j durch den Mittelwert aller ihm zugeordneten Vektoren.

Kompression

Der k -Mittelwerte-Algorithmus im Überblick:

initialisiere $\mu_i = v_i$
solange Quantisierungsfehler zu groß für $i=0, \dots, n$
berechne $f(v_i) = \operatorname{argmin}_j |\mu_j - v_i|$
für $j=0, \dots, k$
ersetze μ_j durch $\frac{1}{r} \sum_{i: f(i)=j} v_i$

$v_1, \dots, v_5 = 4, 2, 7, 9, 4$

$\mu_1 = 4, \mu_2 = 2$

v_i	4	2	7	9	4
$f(v_i)$	1	2	1	1	1

$$\mu_1 = 1/4(4+7+9+4) = 6,$$

$\mu_2 = 2$

v_i	4	2	7	9	4
$f(v_i)$	2	2	1	1	2

$$\mu_1 = 1/2(7+9) = 8,$$

$$\mu_2 = 1/3(4+2+4) = 10/3$$

v_i	4	2	7	9	4
$f(v_i)$	2	2	1	1	2

fertig