Codierung

Quellencodierung für diskrete gedächtnislose Quellen

Quellencodierungstheorem 1

Im vorigen Abschnitt wurde die Entropie als der mittlere Informationsgehalt einer diskreten gedächtnislosen Quelle eingeführt. Darüber hinaus wird gezeigt, dass die von einer Quelle abgegebenen Zeichen ohne Informationsverlust durch binäre Auswahlvorgänge, also Ja/Nein- Entscheidungen, rekonstruiert werden können. Damit kann der Zeichenvorrat einer diskreten Quelle stets ohne Informationsverlust durch die Binärzeichen eines Codes dargestellt werden. Bei einem eindeutig umkehrbaren Code kann keine Information verloren gehen, denn die Zeichen können aus den Codewörtern eindeutig wiedergewonnen werden. Interessanter ist die Frage, wie viele Binärzeichen (Bit) man im Mittel benötigt, um die Information einer Quelle darzustellen. Also um beispielsweise für eine Datei die Frage zu beantworten, wie viel Speicherplatz auf der Festplatte benötigt wird oder wie lange die Modemübertragung dauert.
Eine Antwort darauf gibt das Quellencodierungstheorem von Shannon. Zu dessen Vorbereitung betrachten wir das Beispiel einer Binärcodierung mit Hilfe des Codebaumes im Bild. Angefangen bei der Wurzel werden den Kanten die Codeziffern ”0” und ”1” bei einer Verzweigung nach rechts bzw. links zugeordnet. Bis zu den Endknoten werden vier Kanten durchlaufen, so dass sich im Beispiel das Codewort 0110 ergibt.

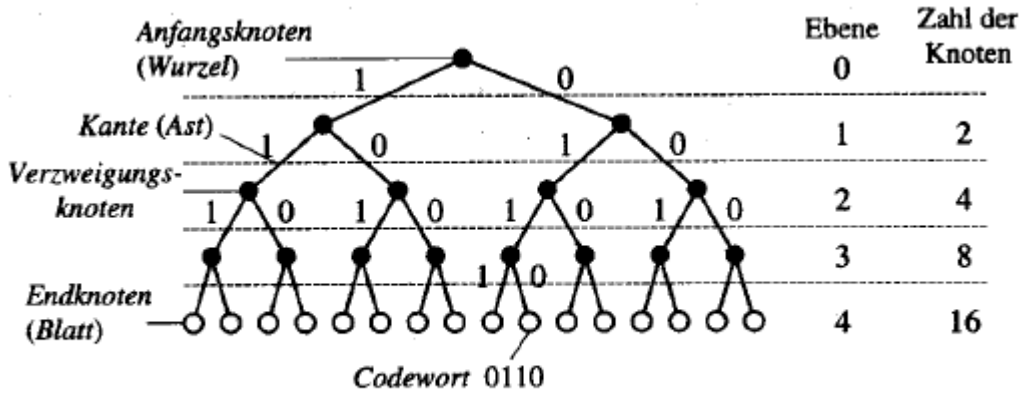


Bild Codebaum

Geht man allgemein von *N* Ebenen aus, so resultieren 2^N Endknoten mit Codewörtern der Länge $ld2^N$ bit = *N* bit. Damit lassen sich genau 2^N Zeichen eindeutig codieren. Nimmt man weiter an, dass diese durch eine Quelle mit genau 2^N gleichwahrscheinlichen Zeichen geliefert werden, so steht dem mittleren

Codierungsaufwand von *N* bit pro Zeichen eine Entropie von ebenfalls
$H(X) = H_0 = N$ bit gegenüber.
Der sich im Beispiel andeutende Zusammenhang zwischen mittlerer Codewortlänge und Entropie der Quelle wird im ***Quellencodierungstheorem*** von Shannon verallgemeinert.

Satz *Quellencodierungstheorem* l
Für jede diskrete gedächtnislose Quelle X mit endlichem Zeichenvorrat und Entropie H(X) existiert ein Präfix-Code mit einem Codealphabet aus r Zeichen so, dass für die mittlere Codewortlänge n gilt

$$\frac{H(X)}{ld(r)} \le \overline{n} \le \frac{H(X)}{ld(r)} + 1$$

Mit r wollen wir die Anzahl der Zustände eines Codesymbols bezeichnen, dann heißt ein Code

- **Idealer Code,** wenn $\overline{n} = H(X)/ld(r)$ gilt
- **Optimaler oder kompakter Code,** zwar $\overline{n} > H(X)/ld(r)$ ist, aber mit dem gegebenen Codealphabet keine kürzere Codierung möglich ist.

Der Ausdruck Präfix-Code bedeutet, dass kein Codewort Anfang eines anderen Codewortes ist. Dann ist eine eindeutige Codierung eines Zeichenstromes ohne Trennzeichen möglich. Im Falle $r = 2$ wird ein binärer Code verwendet. Wird die Entropie in bit angegeben, so ist die Dimension der mittleren Codewortlänge ebenfalls bit und als Angabe in Binärzeichen zu verstehen.
Das Quellencodierungstheorem sagt, dass die mittlere Codewortlänge die Entropie der Quelle nicht unterschreitet.

$$\overline{n} > H(X)/ld(r)$$

Durch Blockbildung der Zeichen, kann jedoch eine Quellencodierung durchgeführt werden, deren mittlere Codewortlänge beliebig nahe an die Entropie heran kommt. Damit wird die Bedeutung der Entropie nochmals hervorgehoben. Sie ist das Maß für den minimalen mittleren Aufwand beim Codieren. Eine praktische Realisierung liefert der ***Huffman-Code***.

Tabelle 3-1 Buchstaben, Morsezeichen [Obe82] und relative Häufigkeiten in der deutschen Schriftsprache [Küp54]

Buchstabe	Morse-zeichen	rel. Häufig-keiten	Buchstabe	Morse-zeichen	rel. Häufig-keiten
A	● ■	0,0651	N	■ ●	0,0992
B	■ ● ● ●	0,0257	O	■ ■ ■	0,0229
C	■ ● ■ ●	0,0284	P	● ■ ■ ●	0,0094
D	■ ● ●	0,0541	Q	■ ■ ● ■	0,0007
E	●	0,1669	R	● ■ ●	0,0654
F	● ● ■ ●	0,0204	S	● ● ●	0,0678
G	■ ■ ●	0,0365	T	■	0,0674
H	● ● ● ●	0,0406	U	● ● ■	0,0370
I	● ●	0,0782	V	● ● ● ■	0,0107
J	● ■ ■ ■	0,0019	W	● ■ ■	0,0140
K	■ ● ■	0,0188	X	■ ● ● ■	0,0002
L	● ■ ● ●	0,0283	Y	■ ● ■ ■	0,0003
M	■ ■	0,0301	Z	■ ■ ● ●	0,0100

Die Huffman-Codierung geschieht in drei Schritten. Wir veranschaulichen sie anschließend mit einem kleinen Beispiel.

Huffman-Codierung

l. Ordnen: Ordne die Zeichen nach fallenden Wahrscheinlichkeiten.

2. Reduzieren: Kombiniere die beiden Zeichen mit den kleinsten Wahrscheinlichkeiten zu einem neuen ”zusammengesetzten Zeichen”; ordne die Liste neu wie in Schritt l und fahre fort bis alle Zeichen zusammengefasst sind.

3. Codieren: Beginne bei der letzten Zusammenfassung; ordne jeder ersten Ziffer des Codeworts eines Zeichens der ersten Komponente des ”zusammengesetzten Zeichens” eine ”0” und der zweiten Komponente eine ”1” zu. Fahre sinngemäß fort, bis alle Zeichen codiert sind.

Im Falle mehrerer „Zeichen” mit derselben Wahrscheinlichkeit werden die, „Zeichen” kombiniert, die am wenigsten bereits zusammengefasste Zeichen beinhalten. Damit erreicht man bei gleicher mittlerer Codewortlänge eine in der Übertragungstechnik günstigere, weil gleichmäßigere Verteilung der Codewortlängen.

Am Beispiel der Quelle in der Tabelle 3-2 wird das Verfahren vorgeführt. Dabei vereinfachen wir es etwas, indem wir auf das explizite Umordnen verzichten. Bei den von Hand durchgeführten kleinen Beispielen kann man auf das explizite Sortieren verzichten. Man erhält ein einfacheres Verfahren, dessen Code allerdings

nicht mehr bitkompatibel zu dem Code mit Umsortieren sein muss.

Tabelle 3-2 Diskrete gedächtnislose Quelle mit dem Zeichenvorrat X = {a,b,c,d,e,f} = $\{x_1,\ldots,x_6\}$, den Wahrscheinlichkeiten p_i, den Informationsgehalten $I(p_i)$ und der Entropie $H(X)$

x_i	a	b	c	d	e	f
p_i	0,05	0,15	0,05	0,4	0,2	0,15
$I(p_i)$ / bit	4,32	2,74	4,32	1,32	2,32	2,74
$H(X)$ / bit	≈ 2,25					

Im Beispiel erhält man im ersten Schritt die in Bild 3-3 angegebene Reihenfolge der Zeichen in der ersten Spalte mit den Wahrscheinlichkeiten *p;* in der zweiten Spalte.
Im zweiten Schritt werden die beiden Zeichen mit kleinsten Wahrscheinlichkeiten ”c” und ”a” kombiniert. Die neuen beiden, „Zeichen” mit den kleinsten Wahrscheinlichkeiten, ”ca” und ”e” werden jetzt zusammengefasst. Für das zusammengesetzte Zeichen erhält man die Wahrscheinlichkeit 0,25. Damit ist sie größer als die Wahrscheinlichkeiten für ”b” und ”f”. Letztere sind nun die beiden kleinsten Wahrscheinlichkeiten. Es werden ”b” und ”f” zusammengefasst. Die zugehörige Wahrscheinlichkeit hat den Wert 0,35. Die nunmehr beiden kleinsten Wahrscheinlichkeiten, für ”bf’ und ”cae”, ergeben zusammen die Wahrscheinlichkeit 0,6. Die beiden verbleibenden Wahrscheinlichkeit für ”d” und ”caebf” müssen zusammen den Wert eins ergeben.

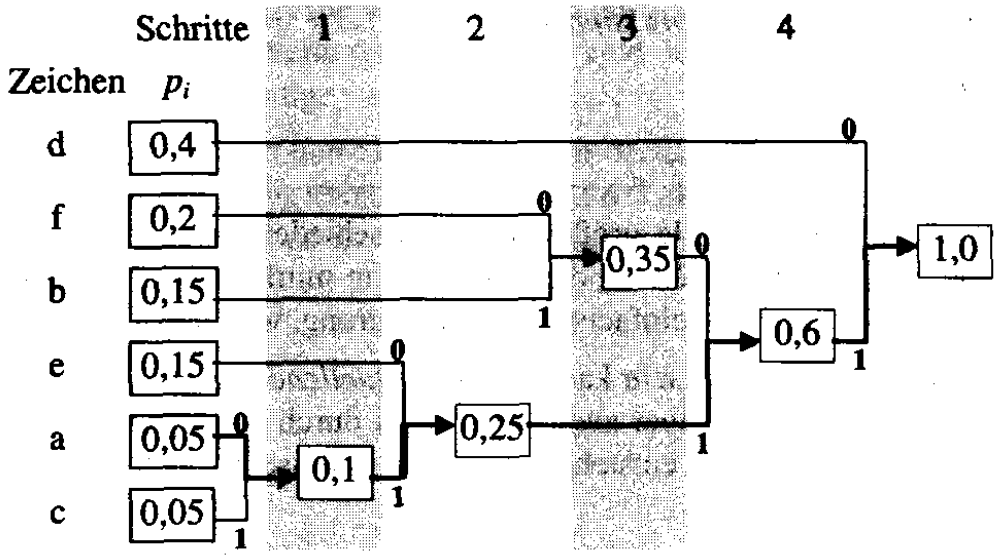


Bild 3-3 Huffman-Codierung

Im dritten Schritt werden den Zeichen die Codewörter zugewiesen. Hierbei beginnt man ganz rechts und schreitet nach links fort. Bei jeder Weggabelung (Zusammenfassung von Zeichen) wird dem Pfad nach oben die ”0” und dem Pfad nach unten die ”l” (oder jeweils umgekehrt) zugewiesen. Der Pfad für die Codezuteilung für das Zeichen ”a” ist in Bild 3-3 fett gedruckt. Man erhält schließlich den in Tabelle 3-3 zusammengestellten Huffman-Code.

Tabelle 3-3 Huffman-Code zu Bild 3-3

Zeichen x_i	d	f	b	e	a	c
Wahrscheinlichkeit p_i	0,4	0,2	0,15	0,15	0,05	0,05
Codewort	0	100	101	110	1110	1111
Codewortlänge n_i in bit	1	3	3	3	4	4

Ein Code ist um so effizienter je kürzer seine mittlere Codewortlänge ist.
Die ***mittlere Codewortlänge*** bestimmt sich aus der Länge der einzelnen Codewörter *n;* gewichtet mit der Wahrscheinlichkeit der jeweiligen Codewörter (Zeichen) *p;*.

$$\overline{n} = \sum_{i=1}^{N} p_i n_i$$

Im Beispiel ist die mittlere Codewortlänge n = 2,3 bit nahe an der Entropie H(X) = 2,25 bit. Eine wichtige Kenngröße ist das Verhältnis von Entropie zu mittlerer Codewortlänge, die ***Effizienz des Codes*** oder auch Datenkompressionsfaktor genannt. Sie erreicht im Beispiel den Wert h = 0,976.

Effizienz des Codes, Datenkompressionsfaktor

$$h = \frac{H(X)}{\bar{n}}$$

Aus dem Beispiel wird deutlich: Je größer die Unterschiede zwischen den Wahrscheinlichkeiten der Zeichen sind, desto größer ist die Ersparnis an mittlerer Wortlänge durch die Huffmann-Codierung im Vergleich zur einfachen Blockcodierung, wie die BCD-Codierung.
Wie effizient der Code bestenfalls sein kann gibt das ***Quellencodierungstheorem*** von Shannon an. Die Informationstheorie zeigt aber auch, dass dabei durch Kombination der Zeichen unter Umständen sehr lange Codewörter entstehen, die einer praktischen Umsetzung des Quellencodierungstheorems entgegenstehen.
Die Decodiervorschrift des Huffman-Codes folgt unmittelbar aus Bild 3-3. Durch den Verzicht auf das Umordnen, kann der *Codebaum* in Bild 3-4 direkt aus dem Bild 3-3 abgelesen werden. Er liefert die anschauliche Interpretation der Codier-vorschrift. Für jedes neue Codewort beginnt die Decodierung am ***Anfangsknoten***, auch ***Wurzel*** genannt.
Wird eine ”0” empfangen so schreitet man auf der mit ”0” gewichteten Kante wie auf einem Zweig nach oben. Im Beispiel erreicht man den Endknoten, das Blatt, ”d”. Das gesendete Symbol ist demzufolge ”d” und man beginnt mit dem nächsten Bit von neuem an der Wurzel.
Wird eine ”l” empfangen wählt man den Zweig nach unten. Man erreicht im Beispiel einen Verzweigungsknoten. Das nächste Bit wählt einen der beiden folgenden Zweige aus. So verfährt man, bis man ein Blatt erreicht. Danach beginnt die Decodierung für das nächste Zeichen wieder an der Wurzel.

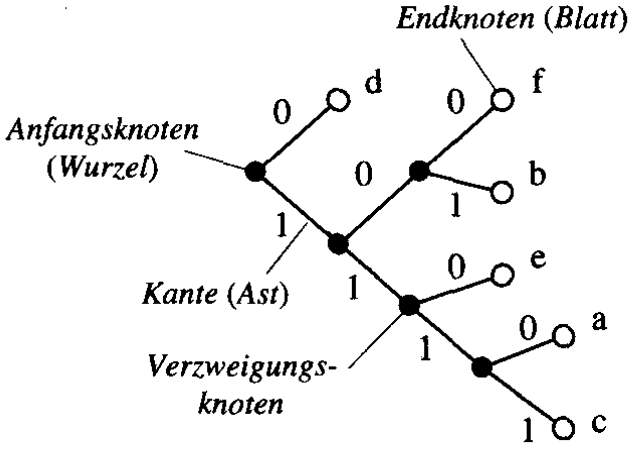
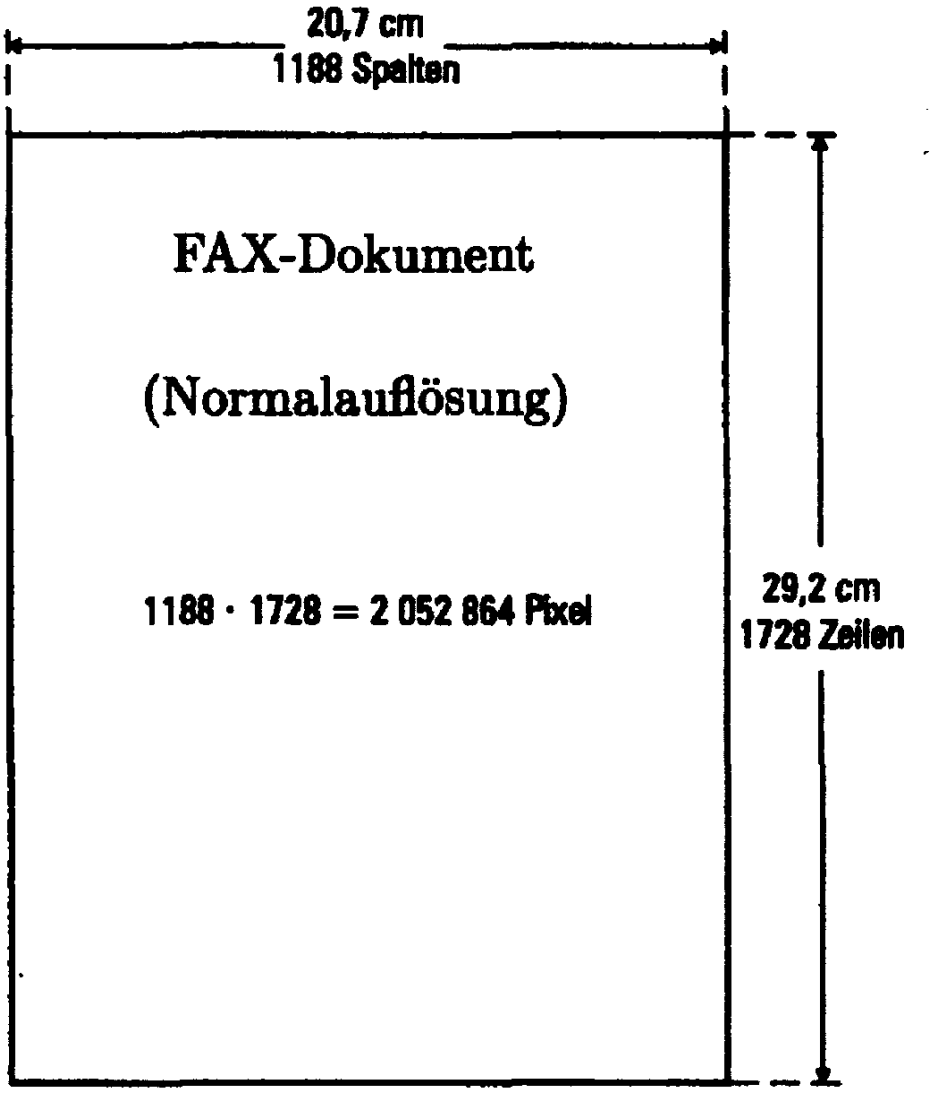


Bild 3-4 Codebaum zum Huffman-Code in Bild 3-3

Die Huffman-Codierung hat sich in vielen speziellen Anwendungen bewehrt. Als abschließendes Beispiel wollen wir uns die Runlängencodierung bei der Faksimileübertragung ansehen.

Anwendung: Quellencodierung bei FAX

Per FAX wird ein zweidimensionales Dokument als Folge von Zeilen übertragen. Das Dokument wird dabei als Bild angesehen, das Abbildungen und Texte enthält. Es wird in Zeilen und Spalten quantisiert, so daß ein Gitter von Bildelementen, die als Pixel bezeichnet werden, entsteht.



Ein Standard DIN A4 Dokument ist 20, 7 cm breit und 29, 2 cm hoch. Die Normalauflösung beim FAX ist *1188 Pixel/Zeile* bei *1728 Zeilen*. Es gibt auch einen hochauflösenden Standard mit *2376 Pixel/Zeile* bei ebenfalls 1728 Zeilen. Im Normalfall erfolgt also eine Auflösung einer DIN A4 Seite in *2052864 Pixel*. Die Auflösung beim (allerdings farbigen) PAL-Fernsehstandard beträgt 720Pixel/Zeile mit 625 Zeilen = 450000 Pixel. D.h. die Anzahl der Pixel im FAX beträgt 4, 56 mal die Anzahl der Pixel im Fernsehbild.

Der Inhalt eines FAX-Pixels ist eine Binärzahl *S* für Schwarz, *W* für Weiß. Eine Zeile besteht also aus einer Abfolge von 1188 Binärwerten. Dabei ist offensichtlich, daß häufig längere Serien derselben binären Wertigkeit im FAX auftreten. Das von der CCITT (ITU- T) standardisierte Lauf1ängen-Codierverfahren dient der Informationskomprimierung. Es basiert auf einem modifiziertem Huffman-Code, der in Tabelle 4.2-1 wiedergegeben ist. Der Code unterscheidet Serien von *S* und Serien von W. Die Länge jeder Serie wird durch ein zweigeteiltes Codewort wiedergegeben. Der erste Teil wird als ***Makeup Codeword*** bezeichnet, es enthält die wichtigeren Bits (MSB, most signifcant bit). Der zweite Teil heißt ***Terminating Codeword***, in dem die weniger wichtigen Bits (LSB, least significant bit) stehen. Jeder Serie der Länge zwischen 0 und 63 wird eindeutig ein Huffman-Codewort zugeordnet, genauso jeder Serie der Länge $64 \cdot k$, $1 < k < 27$, ($64 \cdot 27 = 1728$). Der Code enthält zusätzlich ein eindeutiges ***END OF LINE (EOL)*** Zeichen, das gleichzeitig anzeigt, daß auf dieser Zeile kein schwarzes Pixel mehr folgt.

Beispiel:
Die aus 1188 Pixeln bestehende Zeile mit den Serien

200 W, 10 *S*, 10 W, 84 *S*, 884 W

soll nach Tabelle 4.2-1 codiert werden. Es ergibt sich

010111	10011	0000100	00111	0000001111	00001101000	000000000001
192 W	8 W	10 *S*	10 W	64 *S*	20 *S*	EOL

Zur Übertragung der 1188bit der ursprünglichen Zeile werden nur 56bit benötigt, der Kompressionsfaktor ist also 21, 2.

Run length	White	Black	Run length	White	Black
		Makeup codewords			
64	11011	0000001111	960	011010100	0000001110011
128	10010	000011001000	1024	011010101	0000001110100
192	010111	000011001001	1088	011010110	0000001110101
256	0110111	000001011011	1152	011010111	0000001110110
320	00110110	000000110011	1216	011011000	0000001110111
384	00110111	000000110100	1216	011011000	0000001110111
448	01100100	000000110101	1280	011011001	0000001010010
512	01100101	0000001101100	1408	011011011	0000001010100
576	01101000	0000001101101	1472	010011000	0000001010101
640	01100111	0000001001010	1536	010011001	0000001011010
704	011001100	00000010010121	1600	010011010	0000001011011
768	011001101	0000001001100	1664	011000	0000001100100
832	011010010	0000001001101	1728	010011011	0000001100101
896	011010011	0000001110010	EOL	000000000001	000000000001
Run length	White	Black	Run length	White	Black
		Terminating Codewords			
0	00110101	000110111	32	00011011	000001101010
1	000111	010	33	00010010	000001101011
2	0111	11	34	00010011	000011010010
3	1000	10	35	00010100	000011010011
4	1011	011	36	00010101	000011010100
5	1100	0011	37	00010110	000011010101
6	1110	0010	38	00010111	000011010110
7	1111	00011	39	00101000	000011010111
8	10011	000101	40	00101001	000001101100
9	10100	000100	41	00101010	000001101101
10	00111	0000100	42	00101011	000011011010
11	01000	0000101	43	00101100	000011011011
12	001000	0000111	44	00101101	000001010100
13	000011	00000100	45	00000100	000001010101
14	110100	00000111	46	00000101	000001010110
15	110101	000011000	47	00001010	000001010111
16	101010	0000010111	48	00001011	000001100100
17	101011	0000011000	49	01010010	000001100101
18	0100111	0000001000	50	01010011	000001010010
19	0001100	00001100111	51	01010100	000001010011
20	0001000	00001101000	52	01010101	000001000100
21	0010111	00001101100	53	00100100	000000110111
22	0000011	00000110111	54	00100101	000000111000
23	0000100	00000101000	55	01011000	000000100111
24	0101000	00000010111	56	01011001	000000101000
25	0101011	00000011000	57	01011010	000001011000
26	0010011	000011001010	58	01011011	000001011001
27	0100100	000011001011	59	01001010	000000101011
28	0011000	000011001100	60	01001011	000000101100
29	00000010	000011001101	61	00110010	000001011010
30	00000011	000001101000	62	00110011	000001100110
31	00011010	000001101001	63	00110100	000001100111

Tabelle: Huffman-Code für die FAX-Übertragung nach CCITT

So einfach die Huffman-Codierung und Decodierung ist, sie besitzt drei Nachteile:

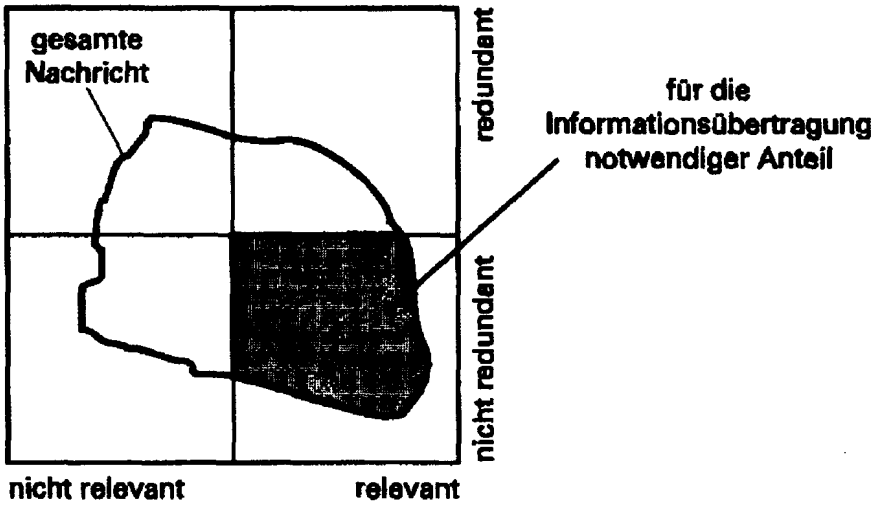
- Die unterschiedlichen Codewortlängen führen zu einer ungleichmäßigen Bitrate und Decodierverzögerung.
- Datenkompressionsverfahren reduzieren die Redundanz und erhöhen deshalb die Fehleranfälligkeit. Im Falle der Huffman-Codierung bedeutet das, dass durch ein falsch erkanntes Bit gegebenenfalls alle nachfolgenden Zeichen falsch detektiert werden können.
- Die Huffman-Codierung setzt die Kenntnis der Wahrscheinlichkeiten der Zeichen oder zumindest geeigneter Schätzwerte voraus. Diese sind jedoch oft nicht bekannt bzw. ihre Schätzung ist relativ aufwendig.

Für die Komprimierung von großen Dateien werden deshalb heute oft ***universelle Codierverfahren*** wie der ***Lempel-Ziv-Algorithmus*** eingesetzt. Sie beginnen die Komprimierung ohne a priori Wissen über die Statistik der Daten.

Datenkompression

Einführung

Die Aufgabe der *Datenkompression* ist es, den technischen Aufwand bei der Übertragung oder Speicherung von Information möglichst klein zu halten, indem der Informationsfluss der Quelle mit einer möglichst geringen Bitrate codiert wird. Natürliche Nachrichtenquellen enthalten neben einer gewissen Weitschweifigkeit (Redundanz) auch Teilinformationen, die Für den Empfänger ohne Belang sind. Diese Bestandteile bezeichnet man als ***Irrelevanz*** der Quelle.



Die Nachrichtenebene

Entsprechend wird bei der Codierung zwischen zwei Konzepten unterschieden:

- Unter der ***Irrelevanz*** versteht man vom Empfänger der Nachricht nicht benötigte Information, wie beispielsweise in der herkömmlichen Telephonie die nicht übertragenen Spektralanteile über 3,4 kHz. Dabei geht streng genommen Information verloren. Man spricht von einer ***verlustbehafteten Codierung***. Das ursprüngliche Sprachsignal kann nicht mehr rekonstruiert werden.
- Mit der ***Redundanz*** bezeichnet man die im Signal, „mehrfach" vorhandene Information. Sie kann ohne Informationsverlust beseitigt werden, wie beispielsweise durch die Huffman-Codierung. Man spricht dann von einer ***verlustlosen Codierung***.

Wichtige Anwendungsbeispiele findet man im digitalen Hörrundfunk (Digital Audio Broadcast, DAB) und digitalen Fernsehen (Digital Video Broadcast, DVB) . Beide Übertragungssysteme arbeiten auf der Grundlage der Audio- und Video-Codierung nach dem MPEG (Motion Pictures Experts Group) -Standard. Die Audio-Codierung fußt auf einem psychoakustischen Modell mit spektralen und zeitlichen Verdeckungseffekten. Die in einem Signalblock jeweils nicht hörbaren Anteile (Irrelevanz) werden weggelassen. Ähnliches gilt auch für die Video-Codierung, wobei sich durch die prädiktive Codierung auf der Basis einer Bewegungsschätzung von Bildinhalten ein hoher Komprimierungsgrad erreichen lässt.
Der mit Datenkompressionsverfahren erreichbare ***Kompressionsgrad,*** mit dem Aufwand ohne Kompression k_0 und dem Aufwand nach der Kompression k_m

$$G_K = \frac{k_0 - k_m}{k_0}$$

hängt vom Verfahren und den Eigenschaften der Quelle ab. Einige Beispiele für praktisch erzielbare Kompressionsgrade sind: bis zu 80% bei Textdateien (z.B. Word97 mit ZIP)
87,5% beim Übergang von der PCM-Telefonie mit 64kbit/s auf den vom Höreindruck vergleichbaren ITU Standard G.729 mit 8 kbit/s ca. 90% bei der Codierung von Audio-CD mit 2 ·16 bit ·44 kHz = 1408 kbit/s durch den MPEG-Standard AAC (Advanced Audio Coding) mit 112 kbit/s bei etwa gleicher Hörqualität.

Ein weiteres Beispiel ist die Entropie der deutschen Schriftsprache. Eine Häufigkeitsanalyse und Auswertung liefert den in Bild 6-1 skizzierten Zusammenhang. Betrachtet man die Zeichen isoliert, ergibt sich eine Entropie von etwa 4,7 bit/Symbol. Fasst man mehrere Zeichen zusammen, so werden Bindungen zwischen den Zeichen sichtbar, wie die Kombination von "qu" oder Silben und

Wörter. Für sehr lange Blöcke kann ein asymptotisches Verhalten mit einer Grenzentropie von etwa 1,6 bit/Symbol erwartet werden.

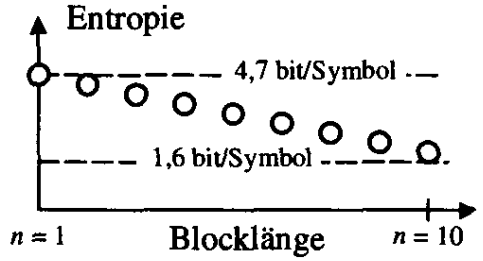


Bild 6-1 Entropie der deutschen Schriftsprache als Funktion der Blocklänge

Die Verfahren zur Datenkompression teilt man in drei Gruppen:

- statische Verfahren, wie z. B. die *Hugman-Codierung* für deutsche Sprache. Im Vergleich zur ASCII-Codierung kann ein Kompressionsgrad von etwa 50% erreicht werden.
- adaptive Verfahren, wie beispielsweise eine Huffman-Codierung, wobei der Codierung eine gemessene Häufigkeitsverteilung zugrundegelegt wird.
- dynamische Verfahren, wie z. B. die Codierung nach ITU Standard V42.bis.

Ein grundsätzliches Problem der *Entropiecodierung* ist, dass sie die Kenntnis der Wahrscheinlichkeitsverteilung der Zeichen voraussetzt. Oftmals sind die Wahrscheinlichkeiten vorab nicht bekannt und müssen erst per Häufigkeitsanalyse geschätzt werden. Abhilfe schaffen hier die ***universellen Codierverfahren:***

i. universelle Algorithmen zur Datenkompression sind adaptiv und benutzen keine a priori Wahrscheinlichkeitsangaben → die Codierung geschieht sofort!

ii. es existieren aufwandsgünstige Algorithmen → die Codierung ist schnell!

iii. je nach Vorlagen sind hohe Kompressionsfaktoren erreichbar → die Codierung ist effizient! Im Weiteren wird ein interessantes und wichtiges Verfahren exemplarisch vorgestellt. Das LZ77-Verfahren von ***Lempel*** und ***Ziv*** aus dem Jahr 1977. Von Welch wurde das Verfahren 1984 verbessert und hat als ***LZW-Verfahren*** Eingang in den ITU-Standard V42.bis gefunden.

Lempel-Ziv-Codierung

Das Codierungsverfahren LZ77 nach Lempel und Ziv beruht auf dem Prinzip eines dynamischen Wörterbuches. Wir stellen kurz das Konzept vor und veranschaulichen es anhand eines einfachen Beispieles. Für das Codierungsverfahren LZ77 sind vier Überlegungen wichtig, s. a. Bild 6-3:

i. Die zu codierende Zeichenfolge wird so in bereits codierte Phrasen (Teilfolgen) zergliedert (***Parsing***), dass alle Phrasen verschieden sind.
ii. Bereits codierte Phrasen dienen als Wörterbuch und werden im ***Phrasenspeicher*** gespeichert.
iii. Die Codierung geschieht mit Hilfe von Ersatzzeichen, die auf die Phrasen im Phrasenspeicher verweisen
iv. Die Codierung ist ein dynamischer blockorientierter Vorgang. Ein über die zu codierende Zeichenfolge ***gleitendes Fenster*** aus Phrasenspeicher und ***Look-ahead-Buffer*** stellt einen lokalen und somit dynamischen Kontextbezug her.

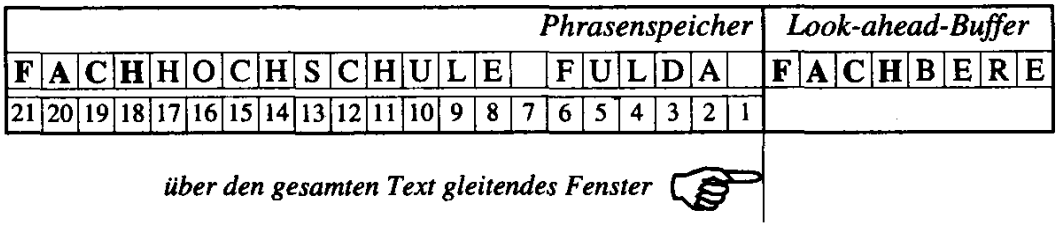


Bild 6-3 Gleitendes Fenster des Codierverfahrens nach LZ77 mit der Übereinstimmung „FACH"

Bei der Codierung wird der Text durch ***Ersatzzeichen*** substituiert, s. Bild 6-4. In Bild 6-3 ergibt sich für die Zeichenkette „FACH" das Ersatzzeichen [21,4,B].

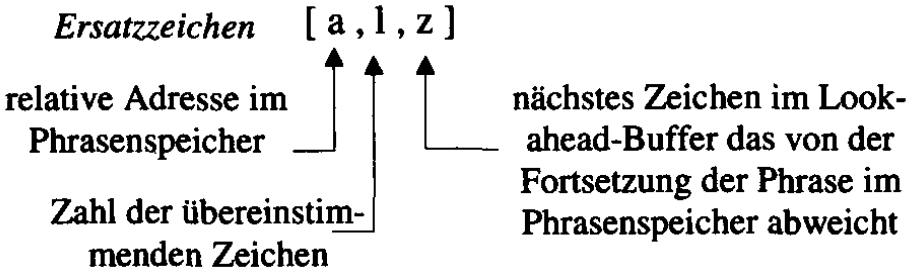


Bild 6-4 Definition der Ersatzzeichen

Ein kurzes Beispiel in Bild 6-5 zeigt ein spezielles Probleme und dessen Lösung auf. Im sechsten Codierungsschritt wird keine Entsprechung für das Zeichen „g" im Phrasenspeicher gefunden. Es muss eine sogenannte ***Nullphrase [O,O,Zeichen]*** eingefügt werden. Anhand der beiden Nullen wird sie bei der Decodierung eindeutig erkannt.

Phrasenspeicher															*Look-ahead-Buffer*							
I	N	F	O	R	M	A	T	I	O	N	-	I	S	-	I	N	F	O	R	M	A	T
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	

ION-NOT-MATTER-NOR-ENERGY

[15,7,T]

Phrasenspeicher															*Look-ahead-Buffer*							
I	O	N	-	I	S	-	I	N	F	O	R	M	A	T	I	O	N	-	N	O	T	-
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	

MATTER-NOR-ENERGY

[15,4,N]

Phrasenspeicher															*Look-ahead-Buffer*							
S	-	I	N	F	O	R	M	A	T	I	O	N	-	N	O	T	-	M	A	T	T	E
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	

R-NOR-ENERGY

[10,1,T]

Phrasenspeicher															*Look-ahead-Buffer*							
I	N	F	O	R	M	A	T	I	O	N	-	N	O	T	-	M	A	T	T	E	R	-
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	

NOR-ENERGY

[4,1,M]

Phrasenspeicher															*Look-ahead-Buffer*							
F	O	R	M	A	T	I	O	N	-	N	O	T	-	M	A	T	T	E	R	-	N	O
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	

R-ENERGY

[11,2,T]

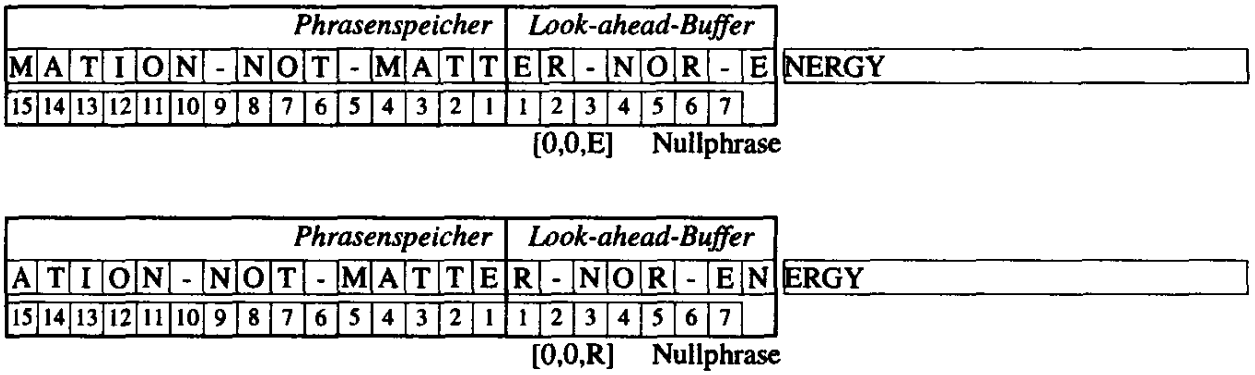


Bild 6-5 Beispiel für die Codierung mit dem LZ77-Verfahren mit Nullphrase und "Character Run"

Ein weiterer interessanter Fall sind Zeichenwiederholungen, sogenannte *"Character Runs"*. Diese werden durch zwei Ersatzzeichen codiert. An der führenden "0" wird wieder ein Sonderfall angezeigt. Durch die "1" und das folgende Zeichen wird das Zeichen angezeigt. Im nachfolgenden Ersatzzeichen wird die Zahl der Wiederholungen und das erste nachfolgende Zeichen codiert. Die Codierung nach Lempel und Ziv führt zu einer Datenkompression, wenn der Codierungsaufwand in Binärzeichen des Ersatzzeichens geringer ist als der Aufwand für die direkte Codierung, wie z. B. im ASCII-Code mit 8 Bit pro Zeichen. Der Codierungsaufwand für ein Ersatzzeichen lässt sich mit der Fensterlänge des Phrasenspeichers w_P und des Look-ahead-Buffers w_L und der Zeichencodierung mit z. B. 8 Bit angeben.

$$\frac{k_E}{bit} = ldw_p + ldw_L + 8$$

Für den typischen Fall von $w_P = 2^{12} = 4096$ und $w_L = 2^4 = 16$ ergeben sich 24 Bits für ein Ersatzzeichen. Bei Phrasen mit drei übereinstimmenden Zeichen werden mit 24 Bits für das Ersatzzeichen zu 32 Bits für vier ASCII-Zeichen bereits 25% eingespart.

Für die Lempel-Ziv-Codierung ist festzustellen:

- häufig auftretende Zeichenkette werden effektiv codiert.
- selten auftretende Zeichen werden mit der Zeit aus dem Phrasenspeicher entfernt.
- Zeichenwiederholungen (Character Runs) werden sehr effektiv codiert.
- Nullphrasen müssen mit relativ vielen Bits dargestellt werden.
- informationstheoretische Überlegungen zeigen, dass die Lempel-Ziv-Codierung asymptotisch optimal in dem Sinne ist, dass für sehr lange Texte die Redundanz fast vollständig beseitigt wird.
- praktisch erreichbare Komprimierungsgrade liegen bei langen Texten bei 50 ... 60%.