Datenkompression

- Übersicht
- Literatur
- Klassifikation:

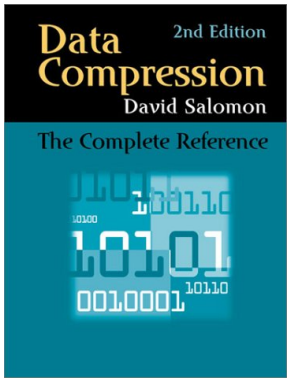
ausgewählte Verfahren:

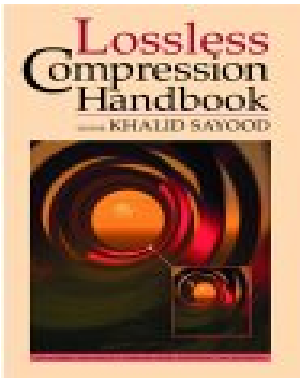
- Lauflängenkodierung
- Huffman Kodierung, arithmetische Kodierung
- Lempel-Ziv Kodierung und Varianten
- Burrows-Wheeler Transformation

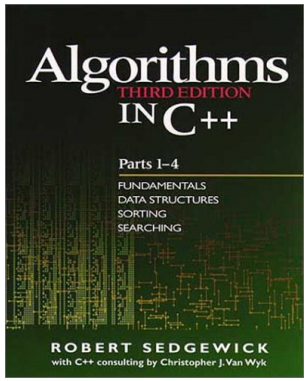
- Wavelets

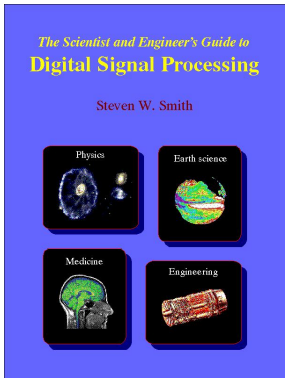
- weitere Verfahren später (Audio, Image, Video)

Datenkompression: Literatur









David Salomon, Data Compression - The Complete Reference, Springer 2000

Khalid Sayood, Ed., Lossless Compression Handbook, Elsevier, 2003

Robert Sedgewick, Algorithms in C++, Addison-Wesley, 1998

Stephen W. Smith, Guide to Digital Signal Processing, California Technical Publ., 1997

online als PDF verfügbar: www.dspguide.com

Mark Nelson, Data Compression with the Burrows-Wheeler Transform,

Dr. Dobbs Journal, 9/1996

F. Bauernöppel, Verfahren und Techniken zur Datenkompression, c't 10/1991

M. Tamm, Packen wie noch nie - BWT-Algorithmus, c't 16/2000, 194

Ad-Hoc Kodierung: ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

Ad-Hoc Kodierung: Morse, Braille, ...

A	.-	N	-.	1	.----	Period	.-.-.-
B	-...	O	---	2	..---	Comma	--..--
C	-.-.	P	.--.	3	...--	Colon	---...
Ch	----	Q	--.-	4	....-	Question mark	..--..
D	-..	R	.-.	5	.....	Apostrophe	.----.
E	.	S	...	6	-....	Hyphen	-....-
F	..-.	T	-	7	--...	Dash	-..-.
G	--.	U	..-	8	---..	Parentheses	-.--.-
H	....	V	...-	9	----.	Quotation marks	.-..-.
I	..	W	.--	0	-----		
J	.---	X	-..-				
K	-.-	Y	-.--				
L	.-..	Z	--..				
M	--						

If the duration of a dot is taken to be one unit, then that of a dash is three units. The space between the dots and dashes of one character is one unit, between characters is three units, and between words six units (five for automatic transmission). To indicate that a mistake has been made and for the receiver to delete the last word, send "........" (eight dots).

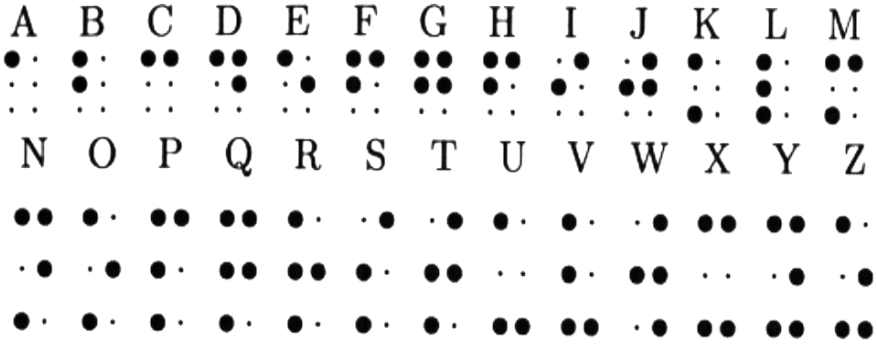


Table 1.1: The 26 Braille Letters.

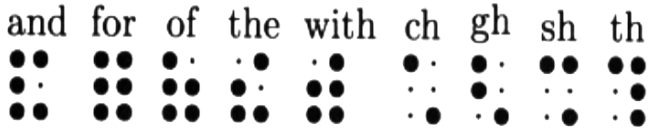


Table 1.2: Some Words and Strings in Braille.

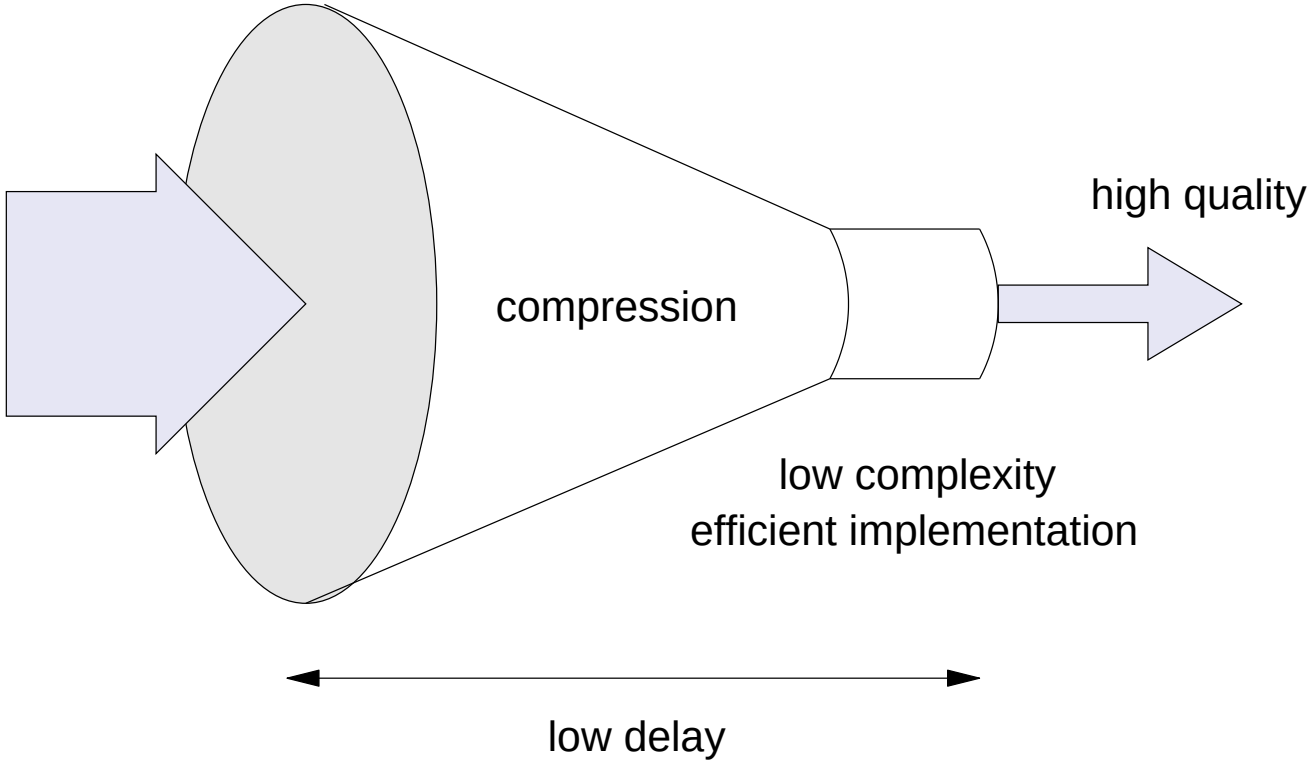
- zwei wichtige Beispiele für "frühe" Codes
- Morse: häufige Symbole ("e") mit kurzen Codes
- Braille: alle 64 Codes verwendet, keine Fehlererkennung

(Salomon)

Multimedia: Datenmengen

Medium:	Datenmenge:	Komprimierung:	
Text	80 x 60 Zeichen 4.8 KB	Huffman 1:2 PKZIP 1:3	
Bild	640x480x3 = 900 KByte	Fax 1:10 JPEG 1:15	
Grafik	500 Linien		
Sprache	1 Kanal, 8Khz 64 Kbps, 480 KB/min	GSM 1:8	
MIDI	15 KB/min		
Musik	stereo, 44.1 KHz 1.44 Mbps, 10 MB/min	MP3 1:10	
Video	640x480x3x25 1.3 GB/min	MPEG2 1:60	

Datenkompression: Kriterien



- gute Kompressionsrate, gute Qualität
- geringe Latenz (Verzögerung)
- geringer Bedarf an Rechenleistung und Speicherbedarf

Klassifikation:

"direct coding"		PCM
"entropy coding"	repetitive sequence suppresion	zero suppresion
		run-length encoding
	statistical encoding	pattern substitution
		Huffman encoding
"source coding" (e.g. speech)	transform encoding	FFT
		DCT
		. . .
	differential encoding	DPCM
		delta modulation
		ADPCM
	vector quantization	general / fractal / . . ..
"channel coding"		HDLC, ...

Entropie- vs. Quellenkodierung

Entropiekodierung:

- Eigenschaften der Datenquelle werden ignoriert
- Signalwiederholungen entfernen
- statistische Verfahren, z.B. Huffman-Kodierung
- verlustfrei und reversibel
- geringe Kompressionsfaktoren (z.B. ca. 2 für Audiodaten)

Quellenkodierung (source encoding):

- Eigenheiten der Datenquelle / -senke berücksichtigen
- z.B. Frequenzgang / Maskierung / Rauschschwellen des Ohrs
- verlustfrei
- verlustbehaftet für bessere Kompression (z.B. MP3 bis ca 10:1)

Codec

- verlustlose Datenkompression:

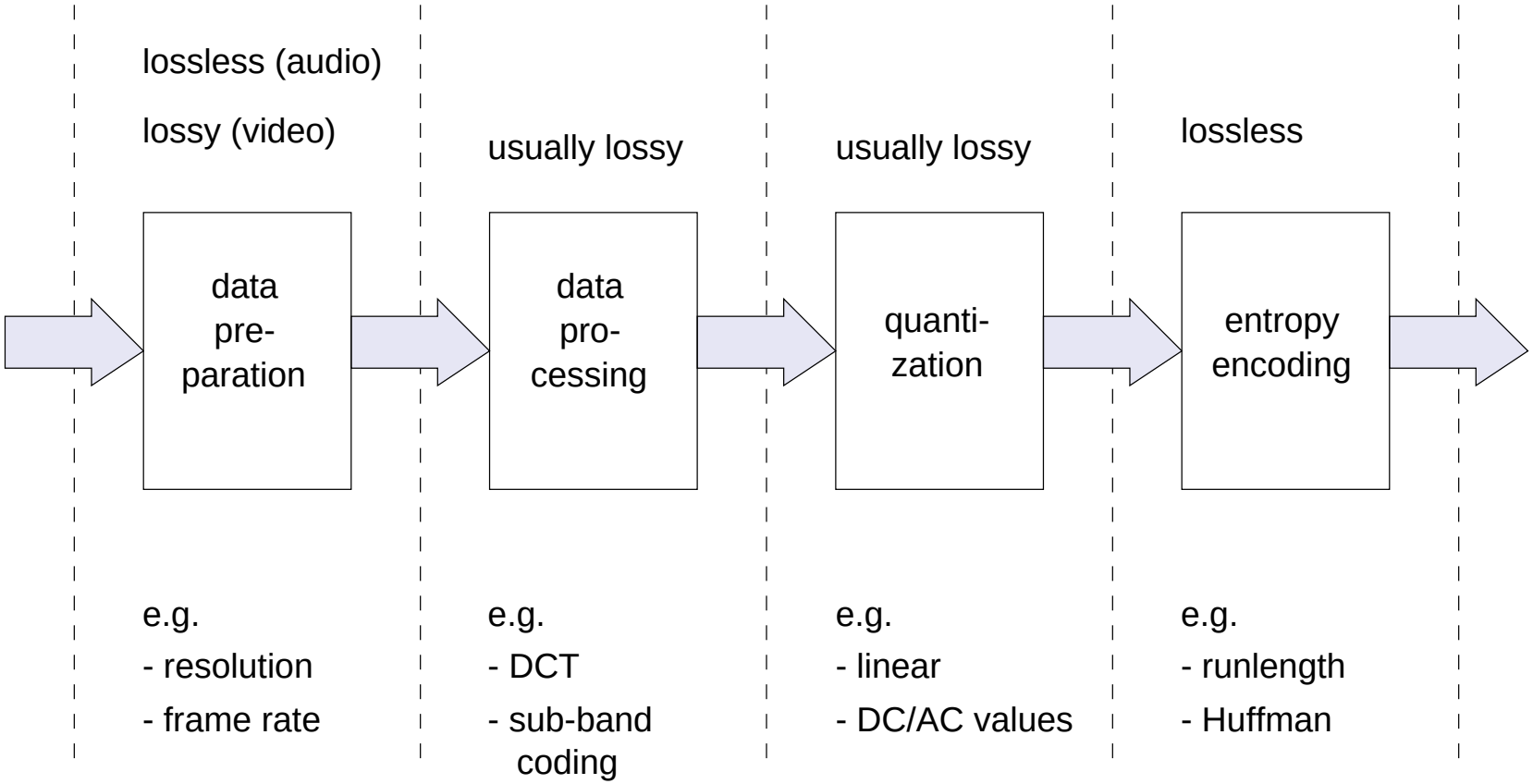
  decode(encode(x)) = x

- jedes Verfahren benötigt Paar aus Coder und Decoder
  := "CODEC"

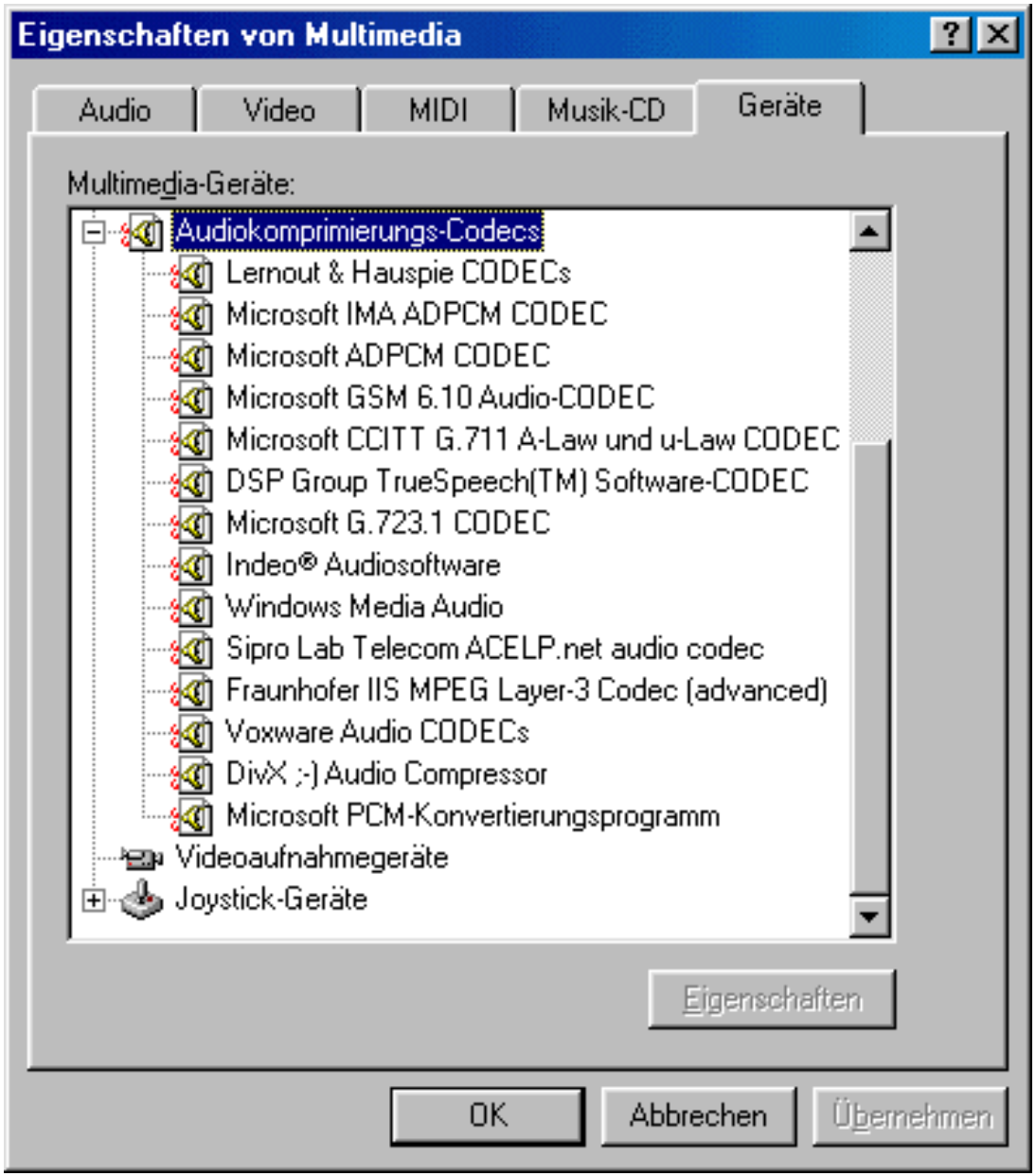
vollständige Abstraktion und Kapselung möglich:

- decode1(decode2(kanal(encode2(encode1(x))))) = x
- äußere / innere Schichten brauchen nicht vom Codec zu wissen
- beliebig tiefe Schachtelung
- Beispiele: Windows / Java Media Framework

CODECs: Verkettung



Codecs: Beispiel Windows 9x



Windows-Systemsteuerung:

- Liste installierter Codecs (*.ax Dateien)
- Audio, Video, MIDI

- je nach installierter SW
- hier: 14 Audio-Codecs
- Sprache vs. Musik

Datenkompression: mehrfach?

- komprimierte Daten beliebig weiter komprimierbar?
- diverse Patentanmeldungen dazu :-)

- aber nicht möglich (sonst jede Datei auf 1 Bit komprimierbar...)
- nur mit "Tricks" (z.B. Daten in Dateinamen speichern)

=> jeder Algorithmus expandiert bestimmte Dateien

Achtung: Verschlüsselung beseitigt Redundanzen der Daten:

- Verschlüsselung nur als letzter Schritt
- nach allen Kodierungsschritten
- verschlüsselte Daten nicht komprimierbar

Compression Challenge

- komprimierte Daten beliebig weiter komprimierbar?
- Mike Goldman's Angebot:

> I will attach a prize of $5,000 to anyone who successfully meets this challenge. First, the contestant will tell me HOW LONG of a data file to generate. Second, I will generate the data file, and send it to the contestant. Last, the contestant will send me a decompressor and a compressed file, which will together total in size less than the original data file, and which will be able to restore the compressed file to the original state.
>
> With this offer, you can tune your algorithm to my data. You tell me the parameters of size in advance. All I get to do is arrange the bits within my file according to the dictates of my whim. As a processing fee, I will require an advance deposit of $100 from any contestant. This deposit is 100% refundable if you meet the challenge.

http://www.faqs.org/faqs/compression-faq/part1/section-8.html

Compression Challenge

- komprimierte Daten beliebig weiter komprimierbar?
- mittlerweile verbesserte Version der Challenge:

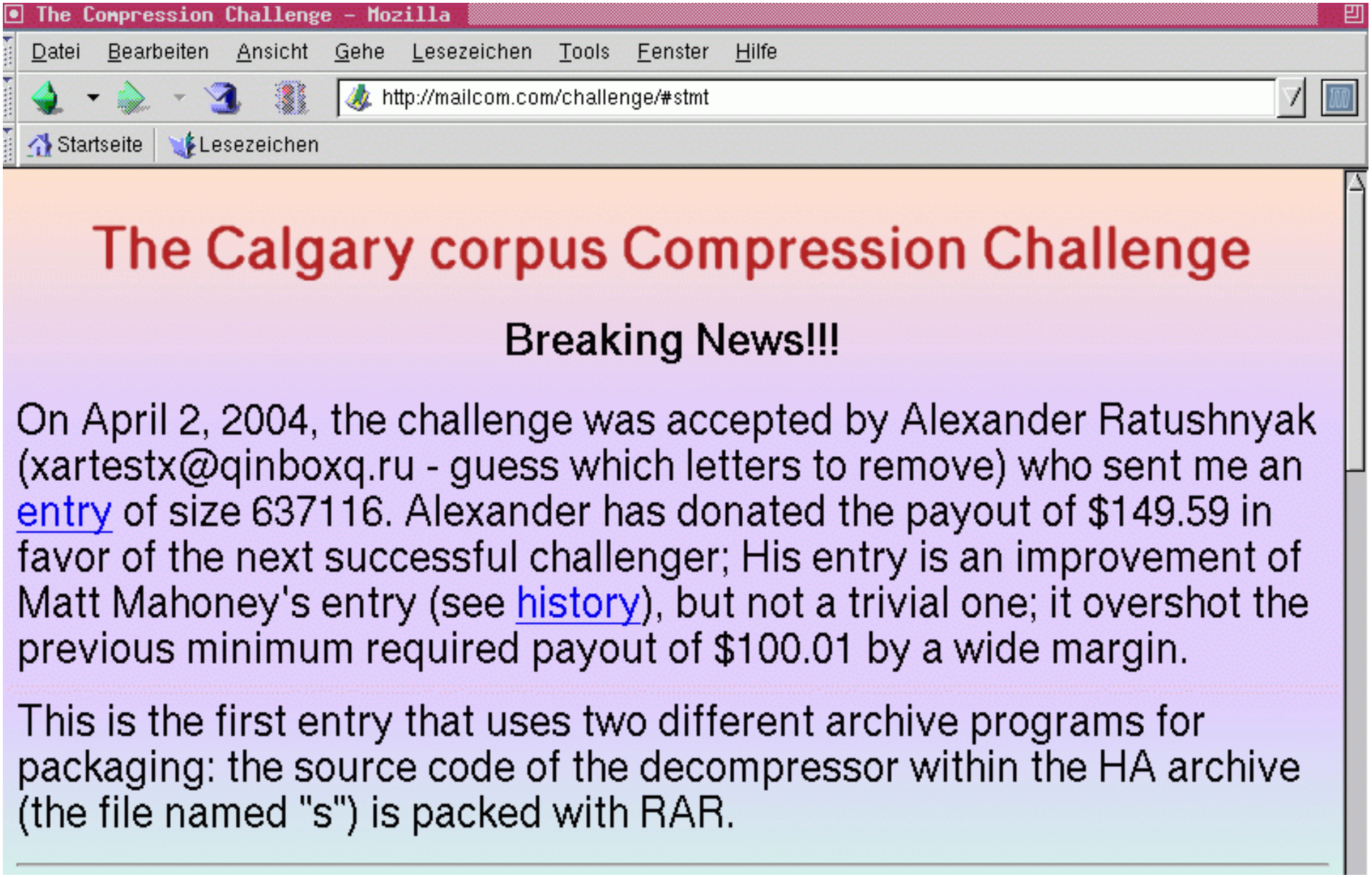
Aufgabe:

- Erstellen eines Programms (Linux binary / sources)
- Ausführen des Programms erzeugt die 14 Dateien
  aus dem "calgary corpus" (1107225 bytes als Zip-Archiv)
- möglichst geringe Größe des Programms
- Gewinnsumme abhängig von der erzielten Verbesserung

  derzeitiger Wert: 637116 Bytes
  "corpus"-Dateien mit zwei Algorithmen komprimiert,
  Packprogramm selbst komprimiert

(www.mailcom.com/challenge)

Compression Challenge



Datenkompression: Szenarien

Backup, Archivierung:

- möglichst gute Kompression, schneller Encoder
- Komplexität und Geschwindigkeit des Decoders egal

Software-Archive (JAR, Zip):

- möglichst schneller Decoder, gute Kompression
- Encoder wird selten benutzt,

Datenübertragung (Modem):

- Echtzeitanforderungen, möglichst gute Performance
- Notwendigkeit zum Umgang mit bereits komprimierten Daten

CD/DVD, weak-signal Kommunikation:

- möglichst gute Fehlererkennung und -korrektur

Transformationen

- Sinuston, Darstellung im Zeitbereich und Frequenzbereich
- Darstellung im Frequenzbereich offenbar weit effizienter

wichtige Beispiele für Transformationen:

- Burrows-Wheeler (für Textdateien)
- Fouriertransformation (Audio, Bilder)
- Cosinustransformation
- Wavelet-Transformation

Informationsgehalt, Redundanz

The entropy of the data depends on the individual probabilities P_i, and is smallest when all n probabilities are equal. This fact is used to define the redundancy R in the data. It is defined as the difference between the entropy and the smallest entropy. Thus

$$R = \left(-\sum_{1}^{n} P_i \log_2 P_i \right) - \log_2(1/P) = -\sum_{1}^{n} P_i \log_2 P_i + \log_2 n.$$

The test for fully compressed data (no redundancy) is thus $\sum_1^n P_i \log_2 P_i = \log_2 n$.

- kann bei bekannter Nachricht leicht berechnet werden
- oder aus bisherigen Symbolen schätzen

- gute Kodierung erreicht R ~= 0

(Shannon)

Run-Length Kodierung

2. all is too well
2. a2l is t2o we2l
2. a@2l is t@2o we@2l

"Lauflängenkodierung"

- Ersetzen von wiederholten Symbolen durch Marke+Anzahl+Symbol
- lohnt erst ab Lauflänge 3
- diverse Varianten zur Kodierung des Zählers

- sehr einfach, sehr schnell
- nur für bestimmte Daten geeignet
- z.B. für S/W-Bilder
- aber nicht für normalen Text, Audio, Grauwert- oder Farbbilder

Lauflängenkodierung: FAX (group 3)

Run length	White code-word	Black code-word	Run length	White code-word	Black code-word
0	00110101	0000110111	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	000000100100
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

(a)

Run length	White code-word	Black code-word	Run length	White code-word	Black code-word
64	11011	0000001111	1344	011011010	0000001010011
128	10010	000011001000	1408	011011011	0000001010100
192	010111	000011001001	1472	010011000	0000001010101
256	0110111	000001011011	1536	010011001	0000001011010
320	00110110	000000110011	1600	010011010	0000001011011
384	00110111	000000110100	1664	011000	0000001100100
448	01100100	000000110101	1728	010011011	0000001100101
512	01100101	0000001101100	1792	00000001000	same as
576	01101000	0000001101101	1856	00000001100	white
640	01100111	0000001001010	1920	00000001101	from this
704	011001100	0000001001011	1984	000000010010	point
768	011001101	0000001001100	2048	000000010011	
832	011010010	0000001001101	2112	000000010100	
896	011010011	0000001110010	2176	000000010101	
960	011010100	0000001110011	2240	000000010110	
1024	011010101	0000001110100	2304	000000010111	
1088	011010110	0000001110101	2368	000000011100	
1152	011010111	0000001110110	2432	000000011101	
1216	011011000	0000001110111	2496	000000011110	
1280	011011001	0000001010010	2560	000000011111	

(b)

Table 2.30: Group 3 and 4 Fax Codes: (a) Termination Codes. (b) Make-Up Codes.

Auflösung:

- horizontal: 8.5 Pixel/mm
- vertikal: 3.85 / 7.7 / 15.4 Zeilen/mm
- (etwa 2200x1700 Pixel pro Seite A4)

Analyse typischer Briefe / Dokumente:

- häufig 2, 3, 4 schwarze Pixel
- oder 2 .. 7 weisse Pixel
- oder ganz weisse Zeilen ...

- ungeeignet für Bilder / Graustufen
- dafür besserer Code ("group 4")

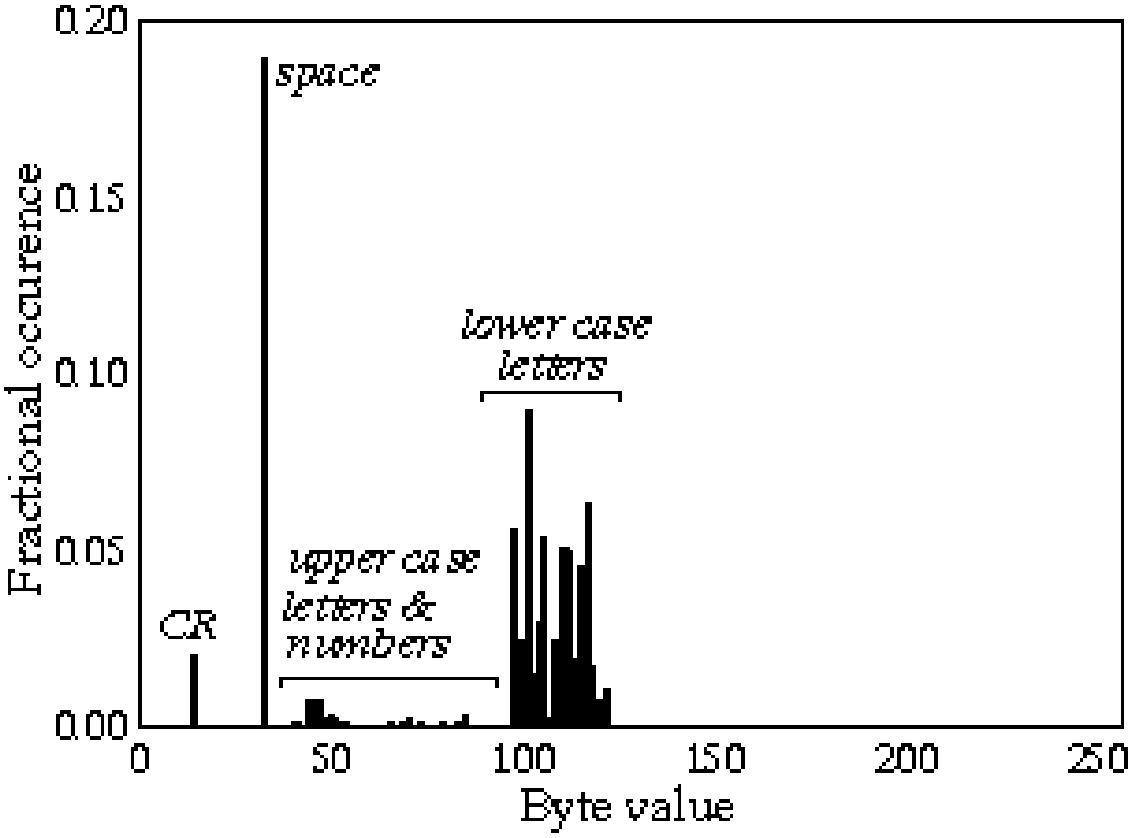
Buchstabenhäufigkeiten . . .

```
53++!305))6*;4826)4+.)4+);806*;48!8'60))85;]8*:+*8!83(88)5*!;
46(;88*96*?;8)*+(;485);5*!2:*+(;4956*2(5*)8'8*;4069285);)6
!8)4++;1(+9;48081;8:8+1;48!85;4)485!528806*81(+9;48;(88;4(+?3
4;48)4+;161;:188;+?;
```

"Here, then, we have, in the very beginning, the groundwork for something more than a mere guess. The general use which may be made of the table is obvious - but, in this particular cipher, we shall only very partially require its aid. As our predominant character is 8, we will commence by assuming it as the "e" of the natural alphabet. To verify the supposition, let us observe if the 8 be seen often in couples - for "e" is doubled with great frequency in English - in such works, for example, as "meet", "fleet", "speed", "seen", "been", "agree", etc. In the present instance we see it double no less than five times, although the cyrptograph is brief.

- Edgar Allen Poe, The Gold Bug

Beispiel: Buchstabenhäufigkeiten



- Einzelhäufigkeiten in English: "ENATOIN SHRDLU ..."
- entsprechende Modelle auch für Paare ("qu"), Triplets, ...
- Grundidee: häufige Symbole mit kurzen Bitstrings kodieren

(Histogramm: DSP Guide)

Huffman-Kodierung

Symbole	Häufigkeit	Codewort
A	0.3	11
B	0.3	10
C	0.1	011
D	0.15	010
E	0.15	00

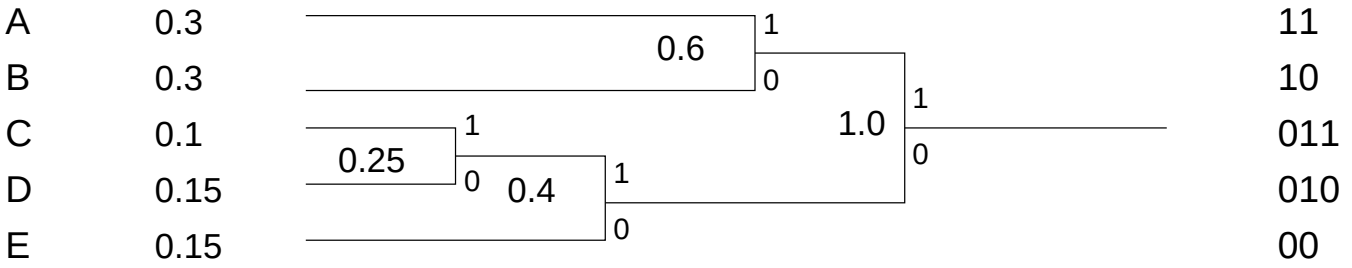
A	B	D	C	A	A	E	C	D
11	10	010	011	11	11	00	011	010

Ausnutzen der statistischen Eigenschaften der Daten:

- häufige Zeichen mit kurzen Bitfolgen kodieren
- seltene Symbole mit längeren (evtl. sehr langen)

- aber: nur wenn Häufigkeiten bekannt sind und konstant bleiben

Huffman: Konstruktion



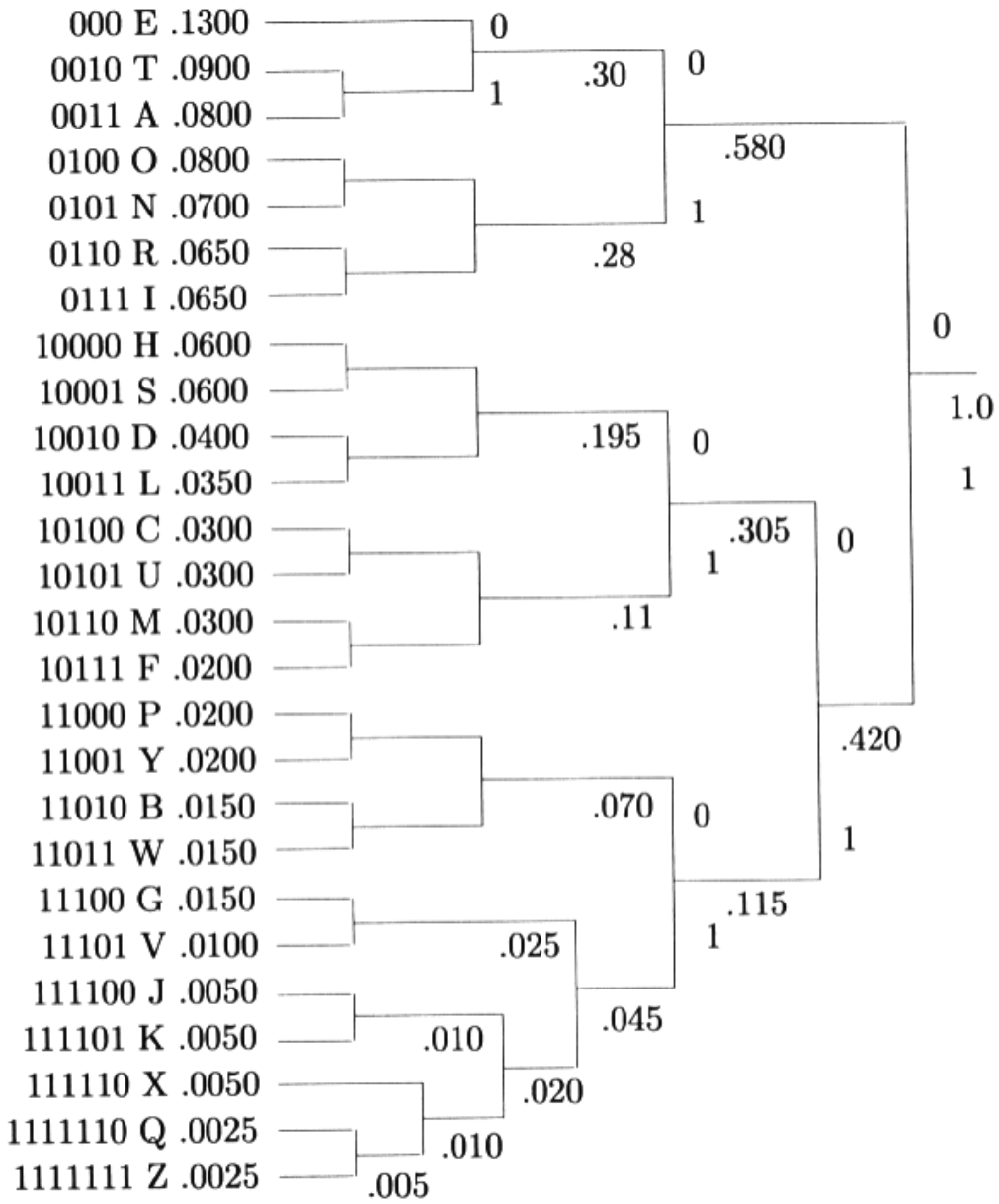
1. sortiere die Symbole nach ihrer Häufigkeit
2. kombiniere die zwei Symbole mit geringster Häufigkeit
   (bei gleichen Wahrscheinlichkeiten: zufällige Auswahl)
3. wiederhole die Konstruktion, bis alle Symbole kombiniert sind

- "bottom-up" Verfahren
- erfordert a-priori Kenntnis der Häufigkeiten

- "semi-adaptive": 2-pass Verfahren, sehr langsam
- "adaptive": Anpassen des Baums an geänderte Häufigkeiten

(siehe Salomon 2.9)

Huffman: Beispiel



(Salomon, 2.12)

Shannon-Fano Kodierung

	Prob.	Steps					Final
1.	0.25	1	1				:11
2.	0.20	1	0				:10
3.	0.15	0		1	1		:011
4.	0.15	0		1	0		:010
5.	0.10	0		0		1	:001
6.	0.10	0		0		0	1 :0001
7.	0.05	0		0		0	0 :0000

Table 2.9: Shannon-Fano Example.

	Prob.	Steps			Final
1.	0.25	1	1		:11
2.	0.25	1	0		:10
3.	0.125	0	1	1	:011
4.	0.125	0	1	0	:010
5.	0.125	0	0	1	:001
6.	0.125	0	0	0	:000

2.10: Shannon-Fano Balanced Example.

- Symbole in möglichst gleichgroße Mengen einteilen
- linke (im Bild: obere) Menge bekommt Bit 1, andere Menge Bit 0
- für alle Teilmengen mit mehr als 2 Elementen wiederholen

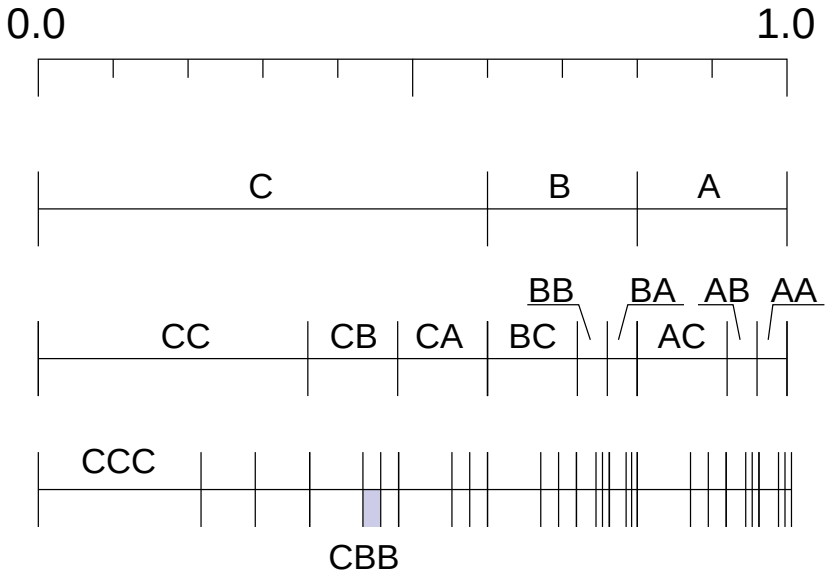
- Verfahren ähnlich zur Huffman-Konstruktion
- aber nur optimal, wenn in jedem Schritt gleichgroße Mengen

Arithmetic Coding

- Huffman-Kodierung ist nur in Spezialfällen optimal
- weil jedem einzelnen Symbol ein Code zugeordnet wird

arithmetische Kodierung:

- Zuordnung von Zeichen zu arith. Intervallen
- entsprechend der Zeichenhäufigkeiten
- rekursiv für Zeichenfolgen (plus zusätzliches Ende-Symbol)
- Verfahren patentiert und lizenzpflichtig



p(C) = 0.6 p(B) = 0.2 p(A) = 0.2

Beispiel-Intervall für Folge "cbb":

0.6*0.6 + 0.8*(0.48 - 0.36) = 0.456

0.6*0.6 + 0.6*(0.48 - 0.36) = 0.432

0.432 = 0.0110 0.456 = 0.0111

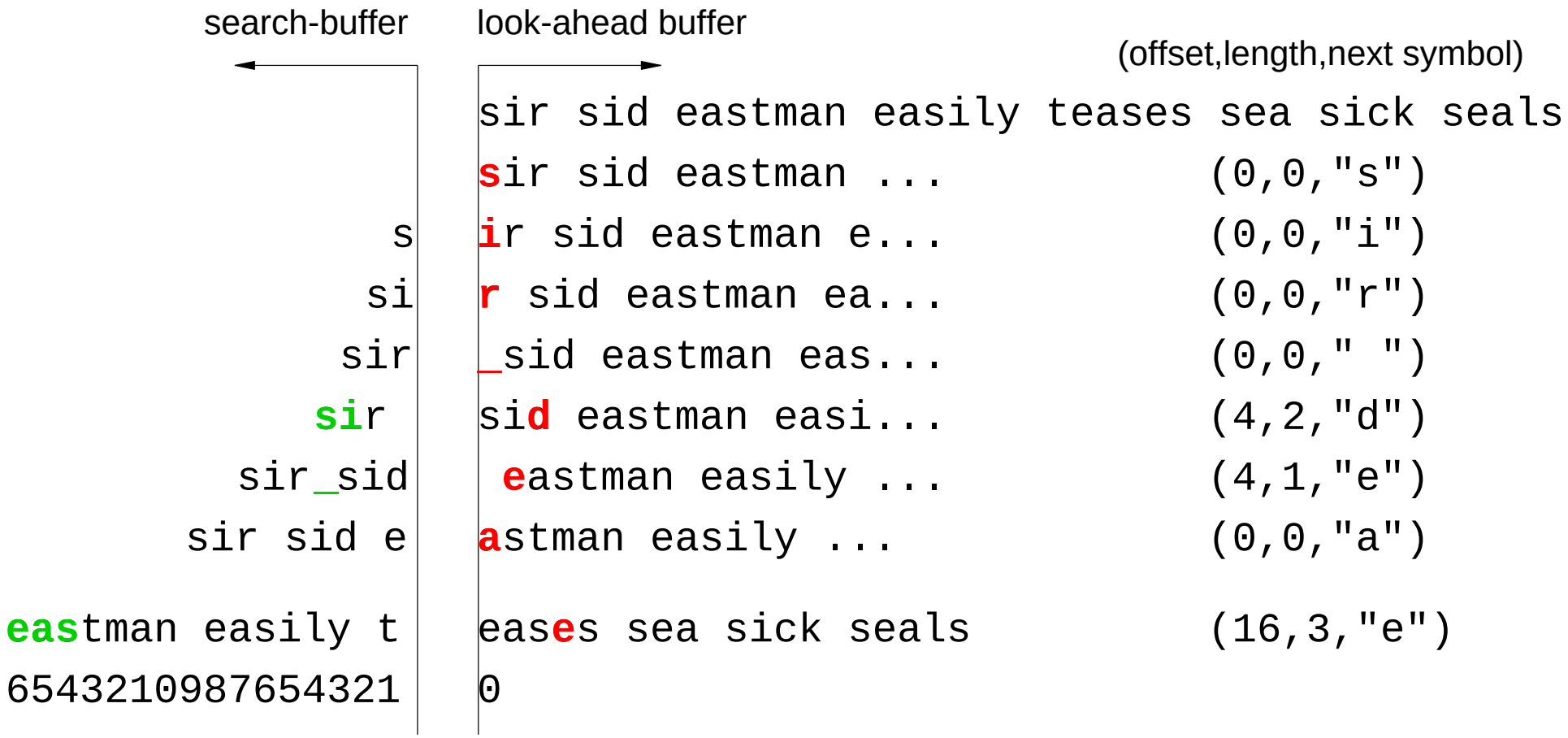
Binärcode für "cbb" = 0111

Dictionary-Methoden

- warum Beschränkung auf Einzelsymbole?
- Verzeichnis mit häufigen Worten (Symbolfolgen) aufbauen
- Index in dieses Verzeichnis kodieren
- unbekannte Symbolfolgen in das Verzeichnis aufnehmen

- Lempel-Ziv Verfahren, Dutzende Varianten
- universell anwendbar: Texte, Programme, Bilder, ...
- sehr gute Kompressionsraten (für verlustfreie Algorithmen)

LZ77



"sliding window" Methode:

- "search buffer" (links) als Dictionary, einige KByte
- Tokens mit Index ins Dictionary und nächstes Symbol schreiben

LZW-Verfahren

- Welsh, 1984
- Dictionary mit Einzelsymbolen (z.B. 256 Bytes) initialisieren
- unbekannte Strings in Dictionary aufnehmen
- dabei Strings maximaler Länge versuchen

- was passiert bei vollem Dictionary?
- diverse Varianten: Dictionary löschen, LRU, ...
- Einsatz u.a. für ZIP, gzip, GIF, TIFF; ...

LZW-Algorithmus

```
for (i=0; i < 255; i++) {
  append i as a 1-symbol to the dictionary
}
append null to the dictionary;
di = dictionary index of null;
repeat
  read(ch);
  if <<<di,ch>> is in the dictionary then
    di = dictionary index of <<<di,ch>>;
  else
    output(di);
    append <<di,ch>> to the dictionary;
    di = dictionary index of ch;
  endif;
until end-of-input;
```

LZW: Beispiel

0	NULL	256	si
1	SOH	257	ir
	...	258	r_
32	space	259	_s
	...	260	sid
97	a	261	d_
98	b	262	_e
99	c	263	ea
		...	
255	255	4095	

- Strings wachsen nur langsam (max. 1 Zeichen)

BWT: Burrows-Wheeler Transformation

- Eingangsdaten in (großen) Blöcken bearbeiten
- z.B. bzip2: 100 .. 900 KByte Blockgröße

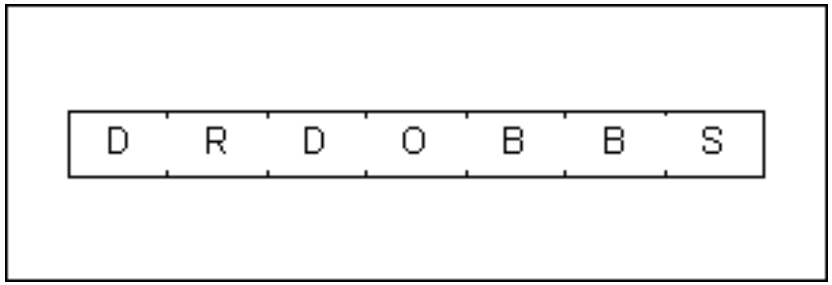
- Daten so umordnen daß ähnliche Strings entstehen
- anschließend normale statistische Kodierung
- völlig neuartiges Prinzip, ursprüngliche Idee (Wheeler 1983)

- adaptiert automatisch an statistische Eigenschaften der Daten
- Algorithmus funktioniert für Texte, Programme, beliebige Daten

gute Beschreibungen:
M. Nelson, Data compression with the BWT, Dr. Dobbs. 9/1996
M. Tamm, Packen wie noch nie, c't 16/2000

(Burrows, Wheeler: A block sorting lossless data compression algorith, DEC report 124, 1994)

BWT: Aufbau der Matrix

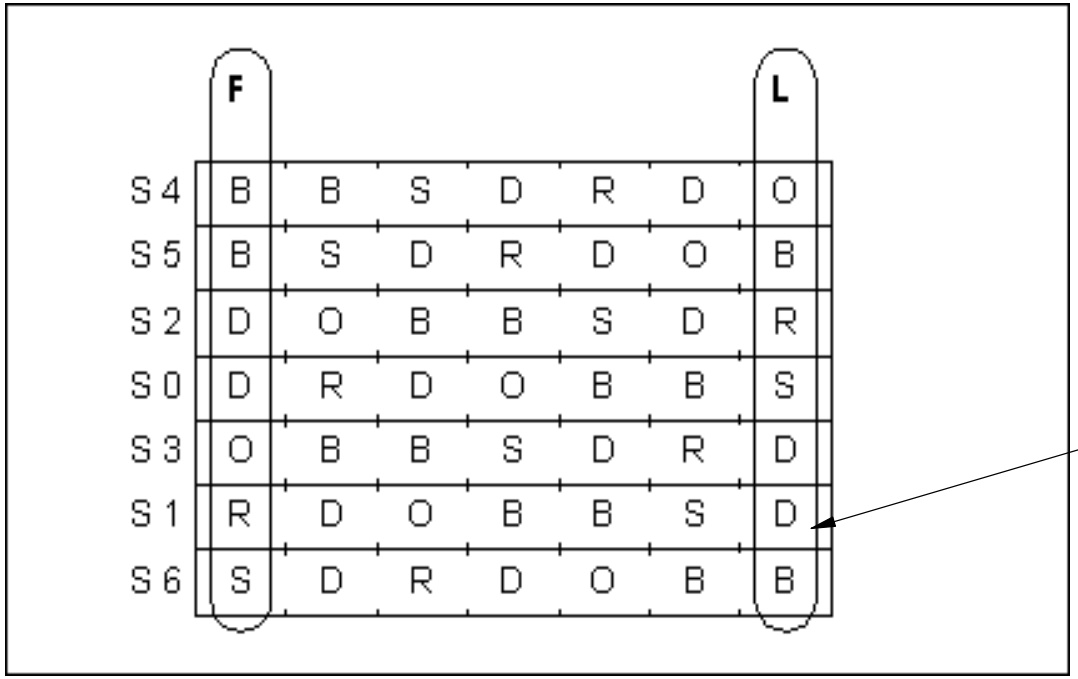


- Eingabedaten zu einer Matrix anordnen
- jede Zeile der Matrix um ein Zeichen rotiert

String 0	D	R	D	O	B	B	S
S 1	R	D	O	B	B	S	D
S 2	D	O	B	B	S	D	R
S 3	O	B	B	S	D	R	D
S 4	B	B	S	D	R	D	O
S 5	B	S	D	R	D	O	B
S 6	S	D	R	D	O	B	B

(aber 100000 x 100000 Matrix ist unhandlich)

BWT: Puffer, Rotation



- Sortieren der Matrix, Aufbau eines Indexvektors
- geeignete Vergleichsfunktion wie strcmp() mit wrap-around
- markierte Spalten F (first) und L (last) der Originaldaten
- Zeichen in L sind jeweils die Vorgänger der Zeichen in F
- Ausgabe der BWT: Spalte L und "Primärindex" (hier 5)

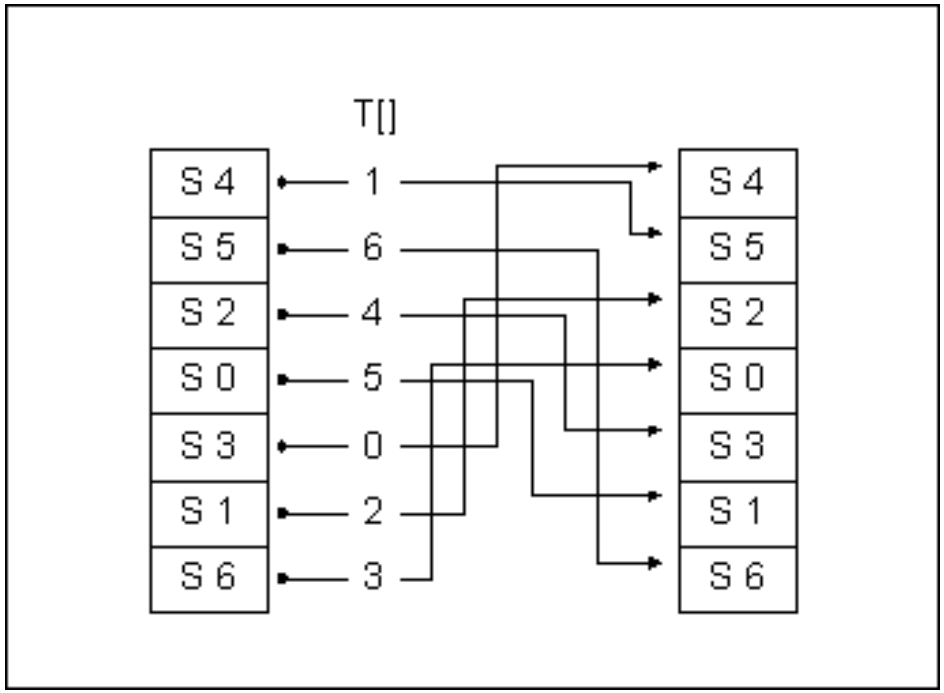
BWT: Puffer, sortiert

```
L: F.........................

t: hat redistributors of a free
t: hat refer to this License an
t: hat system in reliance on co
w: hat the Program does.\n\n  1
t: hat there is no warranty (or
t: hat there is no warranty for
w: hat they have is not the ori
t: hat they, too, receive or ca
t: hat users may redistribute t
t: hat version or of any later
t: hat what they have is not th
t: hat work are not derived fro
```

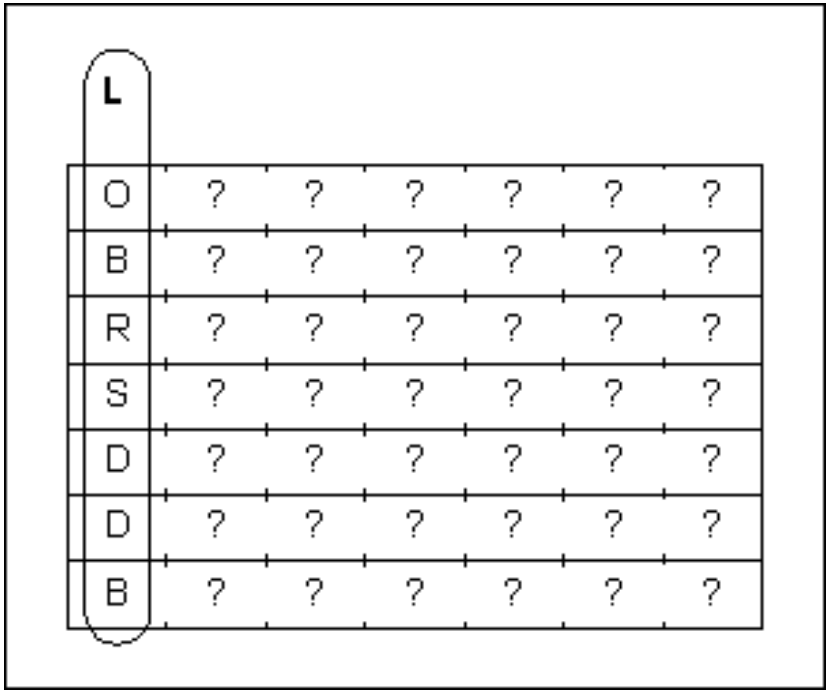
- (BWT eines Ausschnitts aus der GPL)

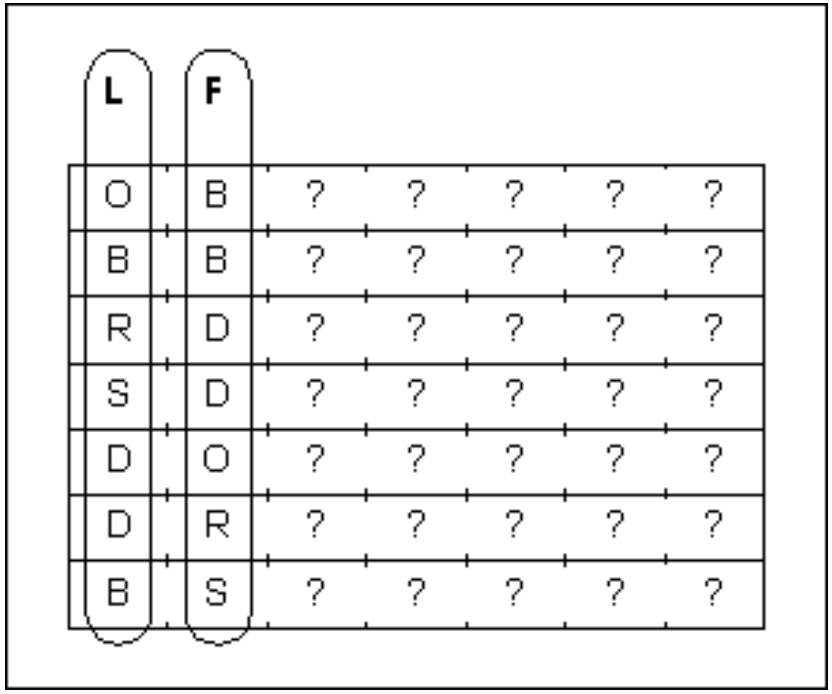
BWT: Transformationsvektor



- T[i] enthält die Zeile, die S[i+1] enthält
- Zeile 3 enthält S0, Zeile 5 enthält S1, also T[3] = 5
- Zeile 2 enthält S2, also T[5] = 2, usw.
- im Beispiel also T = { 1, 6, 4, 5, 0, 2, 3 }
- aber: Berechnung von T aus L ?

BWT: Rekonstruktion von T





- Wiederherstellung von F:
- L ist gegeben, F war sortiert: einfach die Zeichen in L sortieren
- Zeichen "O": offenbar gilt T[4] = 0
- F sortiert: T[1] = 0, T[6] = 1, usw.

BWT: Move-to-Front Kodierung

- verwaltet Liste mit 256 Werten
- Ausgabe eines Zeichens:
  Index des Zeichens in der Liste ausgeben
  und Zeichen nach vorne (Index 0) verschieben
- Beispiel: "tttWtwttt" (aus sortierter GPL)
  Ausgabe: { 116, 0, 0, 88, 1, 119, 1, 0, 0 }
- anschließend noch Huffman- oder arithmetische Kodierung

BWT: Gesamtablauf

```
Compressing a file using the demo programs:

RLE input-file | BWT | MTF | RLE | ARI > output-file

A brief description of each of the programs follows:

RLE.CPP
            This program implements a simple run-length encoder. If the input
            file has many long runs of identical characters, the sorting
            procedure in the BWT can be degraded dramatically.
BWT.CPP
            The standard Burrows-Wheeler transform is done here.  This
            program outputs repeated blocks consisting of a block size integer,
            a copy of L, the primary index, and a special last character index.
            This is repeated until BWT.EXE runs out of input data.
MTF.CPP
            The Move to Front encoder.
RLE.CPP
            The fact that the output file is top-heavy with runs
            containing zeros means that applying another RLE pass
            to the output can improve overall compression.
ARI.CPP
            This is an order-0 adaptive arithmetic encoder,
            directly derived from the code published by Witten and Cleary
            in their 1987 CACM article.
```

BWT: Vergleich mit anderen Verfahren

Performance Figures

The table below shows obtained bit/byte for some different universal coding utilities and files. I have chosen pretty large files to lessen the boundary–effects at the start.

File × Utility –> bit/byte	bzip	dmc	gzip	zip	arj	lha	lharc	compress
bible.tar (5140480 bytes)	1.80	1.83	2.53	2.53	2.60	2.80	2.97	2.89
netscape (4501956 bytes)	3.29	3.45	3.51	3.51	3.51	3.58	3.72	4.89
gcc–2.7.2.1.tar (7090289 bytes)	1.49	1.61	2.02	2.01	2.02	N/A	N/A	2.99

The file *bible.tar* consists of selected texts from the bible (In swedish). See Project Runeberg. Notice the superior performance of bzip and dmc for this kind of source! The perhaps not so well–known coder *dmc* uses dynamic markov modelling and arithmetic coding and was written by Gordon V. Cormack. It performs almost as good as bzip but is slower.

The file *netscape* is a stripped binary. bzip performs best also here but the difference is not so big, althogh note the lousy performance of the standard UNIX utility compress. (Uses LZW technique).

The file *gcc–2.7.2.1.tar* consists of lot of C–code. Also here bzip is outstanding. UNIX compress has to use double as many bits as bzip...

The following programs and flags were used to generate the different codes:

- *bzip –9* ver. 0.21
- *dmc 50000000* (That's a large buffer in case you wondered) ver 0.0.0
- *gzip –9* ver 1.2.4
- *zip –9* ver 2.0.1 for UNIX (performs better than *pkzip –ex*)
- *arj –em* ver 2.30
- *lha* LHa ver. 1.00 for UNIX
- *lha o* LHarc ver. 1.02 for UNIX
- *compress* UNIX (N)compress ver. 4.2.4

Wavelet-Transformation

- Wavelets := "kleine Wellen"
- "lokalisierte", skalierbare Basisfunktionen

$$\Phi_{(s,l)}(x) = 2^{-\frac{s}{2}} \Phi\left(2^{-s}x - l\right)$$

- diverse Basisfunktionen möglich
- Eigenschaften können "maßgeschneidert" werden:
- lineare Transformation:

$$W(x) = \sum_{k=-1}^{N-2} (-1)^k c_{k+1} \Phi(2x + k)$$

- Normierungsbedingung:

$$\sum_{k=0}^{N-1} c_k = 2, \quad \sum_{k=0}^{N-1} c_k c_l = 2\delta_{l,0}$$

Wavelets: Beispielfunktionen

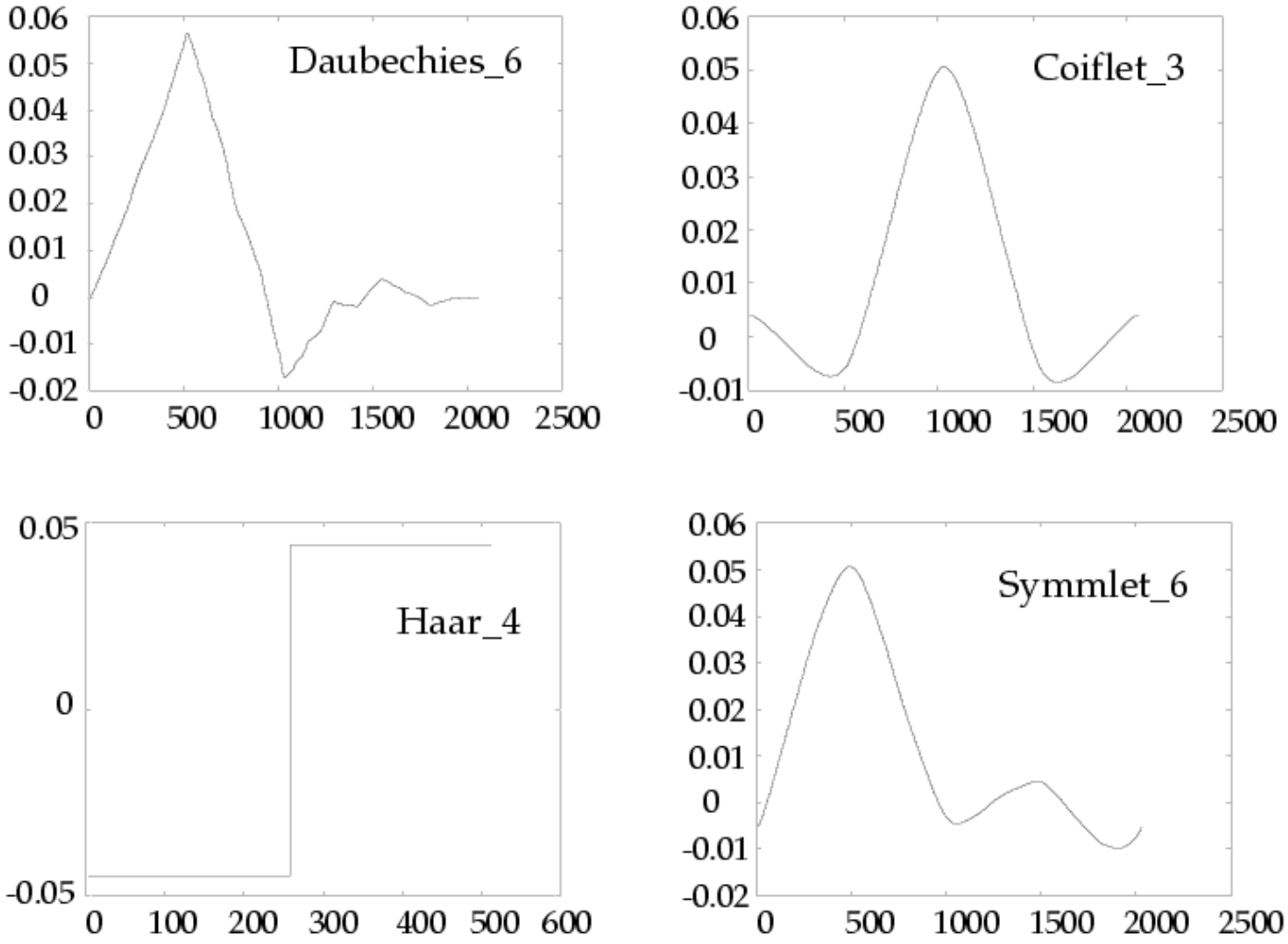
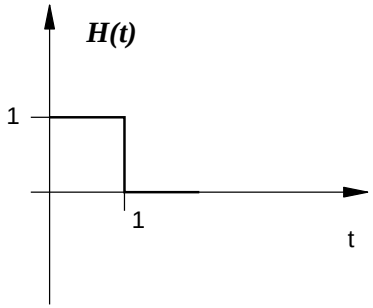
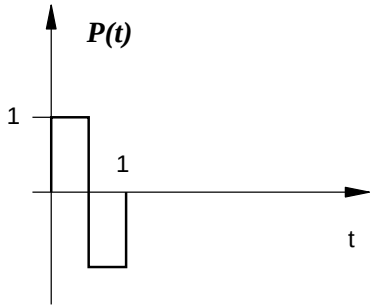


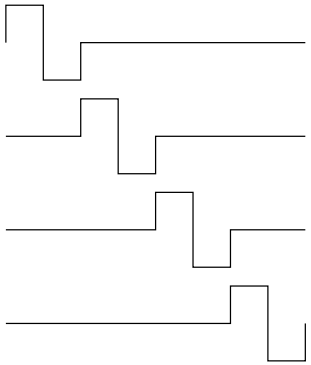
Fig. 4. Several different families of wavelets. The number next to the wavelet name represents the number of vanishing moments (A stringent mathematical definition related to the number of wavelet coefficients) for the subclass of wavelet. These figures were generated using WaveLab.

(Amara Graph, Introduction to Wavelets)

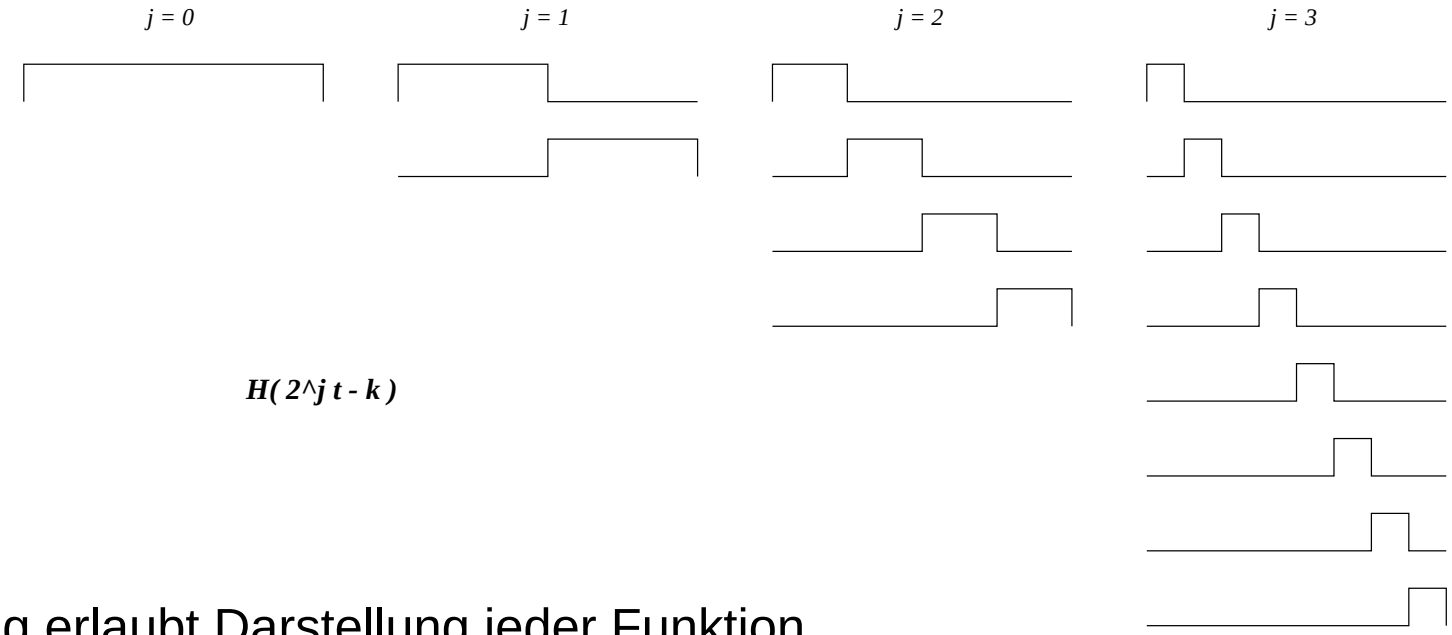
Wavelets: Haar-Wavelet







P(4 t - k)



H(2^j t - k)

- Überlagerung erlaubt Darstellung jeder Funktion

(s. Salomon 5.6)