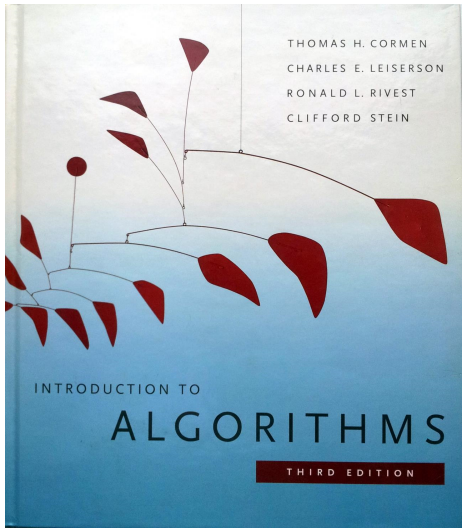
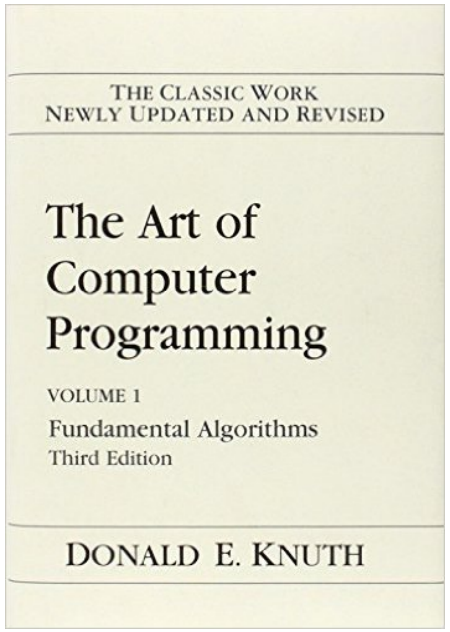
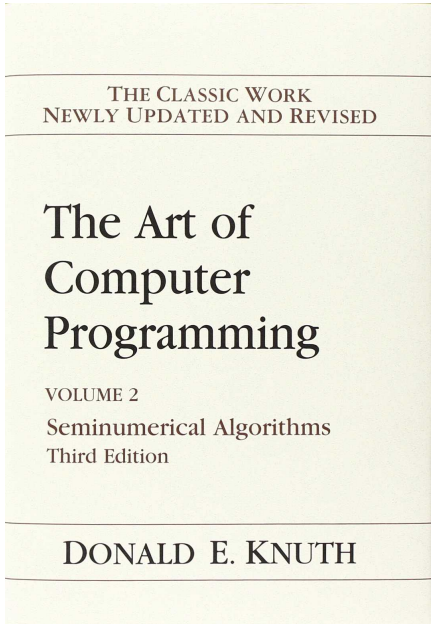
Multimedia II

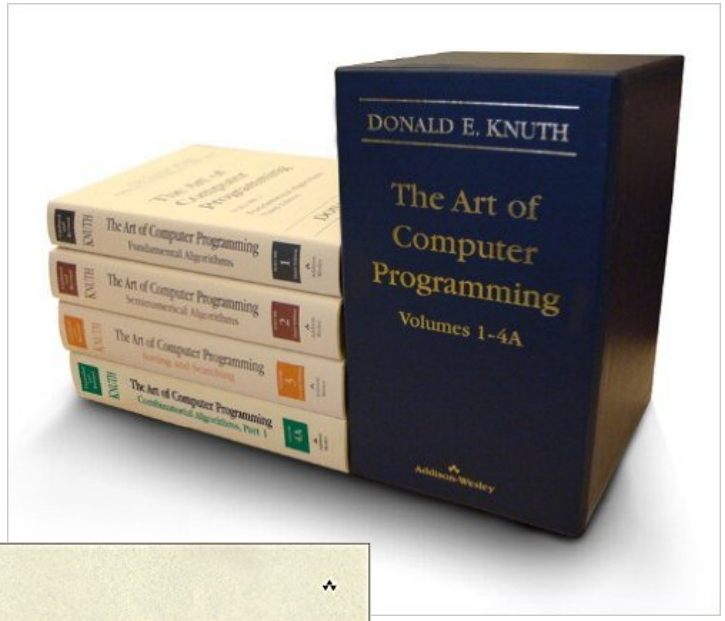
Informationsquellen

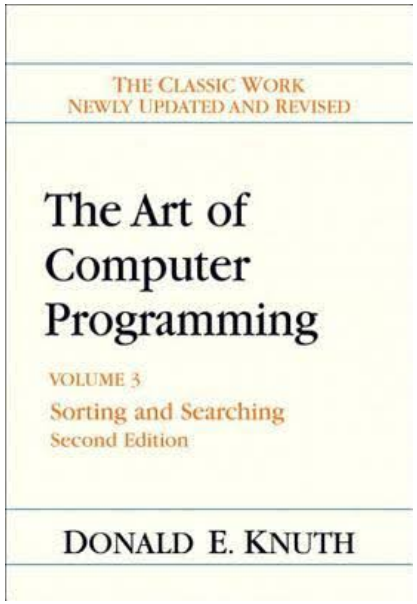


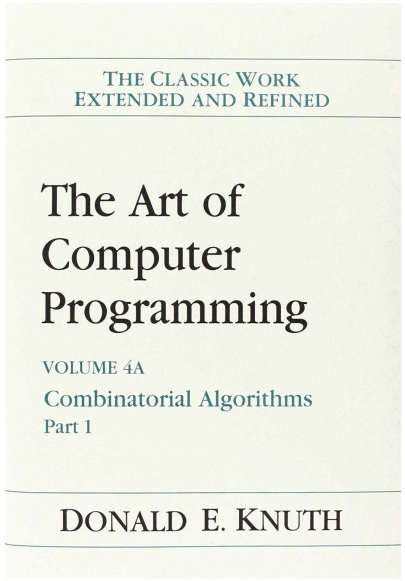
- Helmut Herold, Bruno Lurz und Jürgen Wohlrab:
  Grundlagen der Informatik. München. Pearson Studium 2007
- Christian Horn, Immo Kerner und Peter Forbig:
  Lehr- und Übungsbuch Informatik. Fachbuchverlag Leipzig, (3.Auflage) 2003
- D.E.Knuth: The Art of Computer Programming. Vol.1-4. Addison Wesley 1998 (TAOCP)
- Peter Rechenberg und Gustav Pomberger: Informatik Handbuch. Hanser Verlag, (3.Auflage) 2002
- Uwe Schneider: Taschenbuch der Informatik. Fachbuchverlag Leipzig

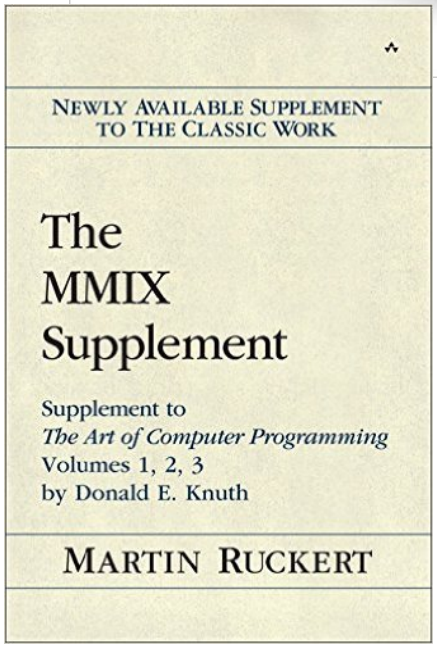

THE CLASSIC WORK
NEWLY UPDATED AND REVISED
The Art of Computer Programming
VOLUME 1
Fundamental Algorithms
Third Edition
DONALD E. KNUTH


THE CLASSIC WORK
NEWLY UPDATED AND REVISED
The Art of Computer Programming
VOLUME 2
Seminumerical Algorithms
Third Edition
DONALD E. KNUTH


DONALD E. KNUTH
The Art of Computer Programming
Volumes 1-4A


THE CLASSIC WORK
NEWLY UPDATED AND REVISED
The Art of Computer Programming
VOLUME 3
Sorting and Searching
Second Edition
DONALD E. KNUTH

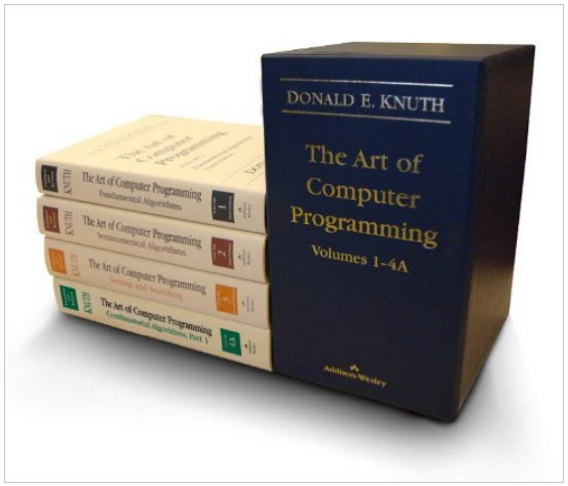

THE CLASSIC WORK
EXTENDED AND REFINED
The Art of Computer Programming
VOLUME 4A
Combinatorial Algorithms
Part 1
DONALD E. KNUTH


NEWLY AVAILABLE SUPPLEMENT
TO THE CLASSIC WORK
The MMIX Supplement
Supplement to
The Art of Computer Programming
Volumes 1, 2, 3
by Donald E. Knuth
MARTIN RUCKERT

Donald Ervin Knuth: The Art of Computer Programming: Volume 1. Fundamental Algorithms

Donald Ervin Knuth: The Art of Computer Programming, Volume 2. Seminumerical Algorithms (3rd Edition)

Donald Ervin Knuth: The Art of Computer Programming: Volume 3. Sorting and Searching



Donald Ervin Knuth: The Art of Computer Programming, Volume 4, Fascicle 0: Introduction to Combinatorial Algorithms and Boolean Functions

Donald Ervin Knuth: The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams

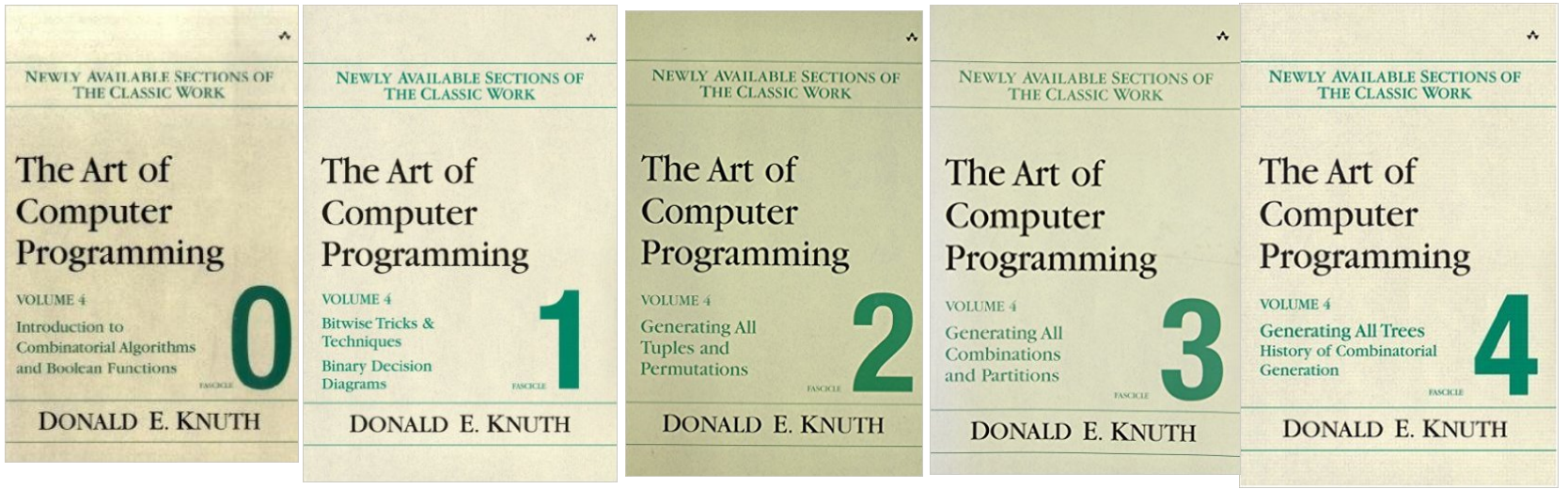
Donald Ervin Knuth: The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations

Donald Ervin Knuth: The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions

Donald Ervin Knuth: The Art of Computer Programming, Volume 4, Fascicle 4: Generating All Trees--History of Combinatorial Generation

Donald Ervin Knuth: The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1

(Addison-Wesley)



Bits & Bytes

https://de.wikipedia.org/wiki/Bit

https://de.wikipedia.org/wiki/Byte

https://en.wikipedia.org/wiki/Byte

Zeichencodierung

https://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange

https://en.wikipedia.org/wiki/ASCII

Unicode (Unicode 9.0.0 - 21.Juni 2016)

https://de.wikipedia.org/wiki/Unicode

http://unicode.org/

http://www.unicode.org/versions/Unicode9.0.0/

UTF-8

https://de.wikipedia.org/wiki/UTF-8

http://www.utf8-zeichentabelle.de/

UTF-16

https://de.wikipedia.org/wiki/UTF-16

ASN.1

Beschreibungssprache zur Definition von Datenstrukturen (ITU,ISO)

https://de.wikipedia.org/wiki/Abstract_Syntax_Notation_One

http://www.itu.int/rec/T-REC-X.680-200811-I/en

ITU-T: X.680, X.681, X.682, X.683, X.690-695

BER - Basic Encoding Rules (X.690)

Punycode

https://de.wikipedia.org/wiki/Punycode

Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)

https://tools.ietf.org/html/rfc3492

BASE-64

https://de.wikipedia.org/wiki/Base64

https://en.wikipedia.org/wiki/Base64

https://www.base64decode.org/

JSON

https://de.wikipedia.org/wiki/JavaScript_Object_Notation

http://www.json.org/

The JavaScript Object Notation (JSON) Data Interchange Format

https://tools.ietf.org/html/rfc7159

ECMA International: The JSON Data Interchange Format. 2013

http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Von-Neumann Rechnerarchitektur

https://de.wikipedia.org/wiki/Von-Neumann-Architektur

http://www.itwissen.info/definition/lexikon/Von-Neumann-Rechner-von-Neumann-architecture.html

http://www.elektronik-kompendium.de/sites/com/1309261.htm

Intel x86 Prozessorarchitektur

https://de.wikipedia.org/wiki/X86-Prozessor

Virtualisierung

https://de.wikipedia.org/wiki/Virtualisierung_(Informatik)

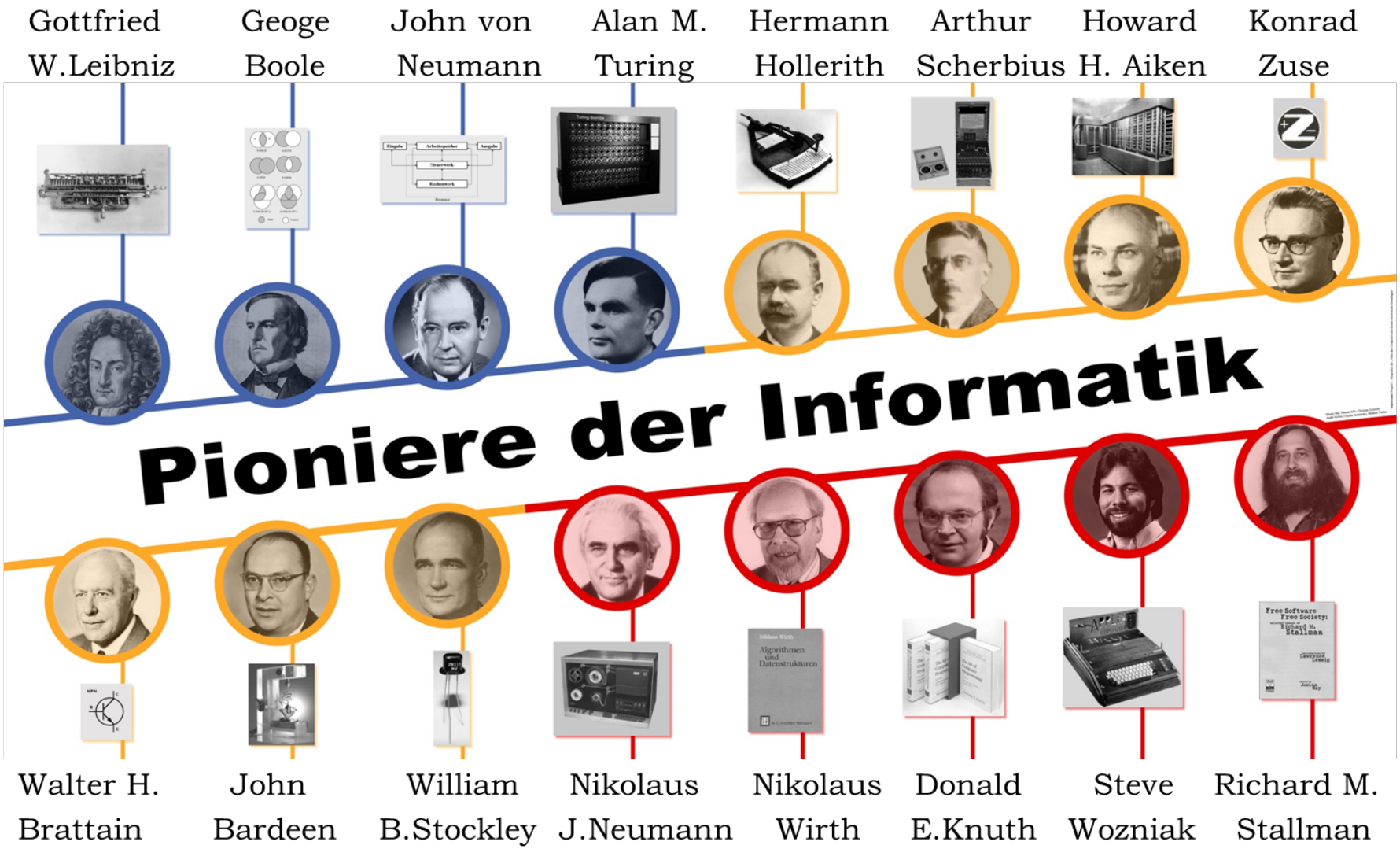
http://www.vmware.com/de/solutions/virtualization.html

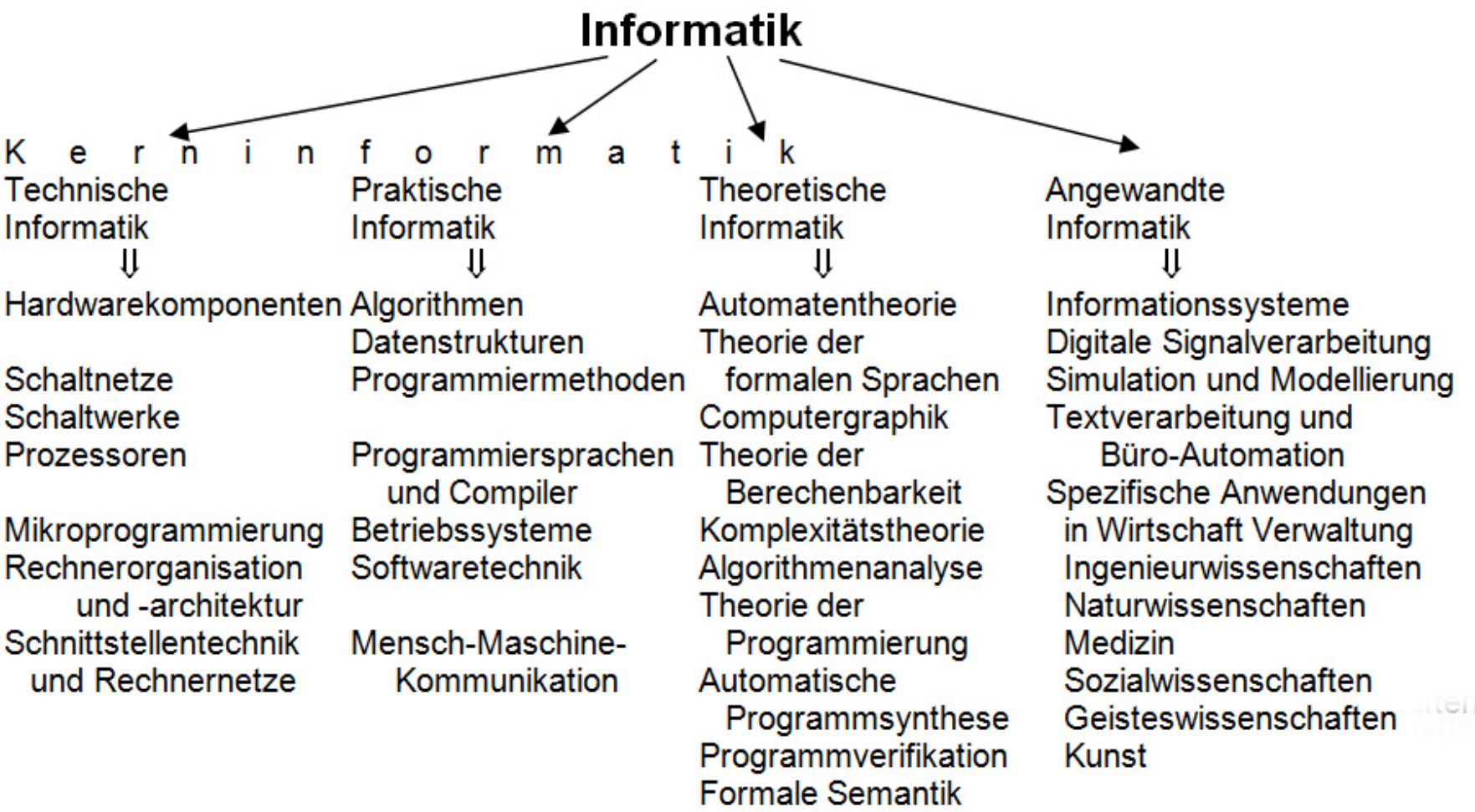
Betriebssysteme

https://de.wikipedia.org/wiki/Betriebssystem

https://de.wikipedia.org/wiki/Liste_von_Betriebssystemen

https://en.wikipedia.org/wiki/Operating_system


Gottfried W.Leibniz
Geoge Boole
John von Neumann
Alan M. Turing
Hermann Hollerith
Arthur Scherbius
Howard H. Aiken
Konrad Zuse
Pioniere der Informatik
Walter H. Brattain
John Bardeen
William B.Stockley
Nikolaus J.Neumann
Nikolaus Wirth
Donald E.Knuth
Steve Wozniak
Richard M. Stallman


Informatik
K e r n i n f o r m a t i k
Technische Informatik
Praktische Informatik
Theoretische Informatik
Angewandte Informatik
Hardwarekomponenten
Schaltnetze
Schaltwerke
Prozessoren
Mikroprogrammierung
Rechnerorganisation und -architektur
Schnittstellentechnik und Rechnernetze
Algorithmen
Datenstrukturen
Programmiermethoden
Programmiersprachen und Compiler
Betriebssysteme
Softwaretechnik
Mensch-Maschine-Kommunikation
Automatentheorie
Theorie der formalen Sprachen
Computergraphik
Theorie der Berechenbarkeit
Komplexitätstheorie
Algorithmenanalyse
Theorie der Programmierung
Automatische Programmsynthese
Programmverifikation
Formale Semantik
Informationssysteme
Digitale Signalverarbeitung
Simulation und Modellierung
Textverarbeitung und Büro-Automation
Spezifische Anwendungen in Wirtschaft Verwaltung
Ingenieurwissenschaften
Naturwissenschaften
Medizin
Sozialwissenschaften
Geisteswissenschaften
Kunst

1. Bits & Bytes Informationstheorie minimale (binäre) Codes

- BOM (BE/LE, MSBF/LSBF)
- Informationsentropie/mittlere Codelänge
- minimale Codes - Entropie-Codierung
  (Huffman, Shannon-Fano, Tunstall, arith.Codierung)
- Daten-Kompression (RLE, LZW ...)

Bits & Bytes

Bit = Binary Digit $\in \{0,1\}$

1560-1621 Thomas Harriot (England) gilt als Erfinder der Dyadik (binäre Arithmetik)

1550-1617 John Neper (Napier) (Zuordnung von Buchstaben den Zweierpotenzen und „Arithmetica localis“)

1646-1716 Gottfried Wilhelm Leibniz

1701 „Essay d‘une nouvelle Science Nombres“

(Abhandlung über eine neue Wissenschaft der Zahlen)

Bits & Bytes

$2^7 = 128$

1 Byte = 8 Bits : $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$ (MSBF = Most Signifikant Bit First)

oder $b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7$ (LSBF = Least Signifikant Bit First)

$2^0 = 1$

mit den Stellenwertigkeiten:

128	64	32	16	8	4	2	1
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
2^7	2^6						2^0

SI-Symbole

1 KB = 1000Byte

1 MB = 1000KByte

(Dezimalpräfixe)

IEC-Symbol

1 KiB = 1024 B

1 Mi = 1024 KiB

(Binärpräfixe)

Inf. MSBF $b_7\ b_6 \ldots\ldots b_1\ b_0$
$2^7\ 2^6 \qquad\qquad 2^1\ 2^0$

ISO / ITU-T CCITT $b_8\ b_7 \ldots\ldots b_2\ b_1$
$2^7\ 2^6 \qquad\qquad 2^1\ 2^0$
MSBF
ASN.1
BER
X.680 / X.690

aber INTERNET RFC 791 ietf.org $b_0\ b_1 \ldots\ldots b_6\ b_7$
$2^7\ 2^6 \ldots\ldots 2^1\ 2^0$
MSBF

Bsp: 1101 0101

D=13 5

MSBF $128+64+16+4+1 = 213$

$13 \cdot 16^1 + 5 \cdot 16^0 = 213$

Horner-Schema

	1	1	0	1	0	1	0	1
2*	↓	2	6	12	26	52	106	212
	1	3	6	13	26	53	106	213

LSBF

$10^3\ 10^2\ 10^1\ 10^0$

2018

40% 256 16 1

$16^3\ 16^2\ 16^1\ 16^0$

1AF6H

$1 \cdot 16^3 + 10 \cdot 16^2 + 15 \cdot 16^1 + 6 \cdot 16^0$

Tetrade
Nibble
Halbbyte

Bsp: 1101 0101

13=D 5

MSBF 128+64+16 + 4 + 1 = 213

16^1 16^0

13·16 + 5 = 208+5 = 213

LSBF A B

1 0 1 0 1 0 1 1

128 +32 +8 +2+1 = 171

1 0 1 0 1 0 1 1

2 2 4 10 20 42 84 170

1 2 5 10 21 42 85 171

aber NICHT die Halbbytes/Nibbles drehen

10·16+11=171

5·16+13=93

Bits & Bytes

1 Byte = 8 Bits : $b_7\ b_6\ b_5\ b_4$ $b_3\ b_2\ b_1\ b_0$

8 4 2 1 8 4 2 1

$2^3\ 2^2\ 2^1\ 2^0$ $2^3\ 2^2\ 2^1\ 2^0$

Hexziffer Є {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

Beispiele: 00H = 0000 0000 = 0
80H = 1000 0000 = 128
AAH = 1010 1010 = 170
55H = 0101 0101 = 85
FFH = 1111 1111 = 255

Bits & Bytes

	B = Bytes	b = bits
Kilo = 1024	1 KB = 1024 Bytes	1 Kb = 1024 bit (unüblich)
kilo = 1000	1 kB = 1000 Bytes (unüblich)	1 kb = 1000 bit (z.B. 9,6 kbps = 9600 bps) 1 Kbps (üblich für 1000 Bit/s = 1 Kb/s)
	bei Festplatten: 120 MB =	120 000 000 Bytes = 114,44 Mega Bytes (effektiv)
Mega = 1048576	1 MB	1 Mb (oft als Mbps aber mbps=1 Mill. Bit/s sind gemeint)
mega = 1 Mill.	1 mB (unüblich)	1 mbps

Bits & Bytes

1 Byte = 8 Bit (Codierung von 2^8 = 256 Zeichen)

1 Byte = 2 Hexziffern = $b_7b_6b_5b_4b_3b_2b_1b_0$

1 Byte = 2 Hexziffern = 2 Halbbytes = 2 Nibbles = 2 Tetraden

1 KBit = 1024 Bit = 2^{10} Bit = 128 Byte = 2^7 Byte
→ 1 KByte = 2^{10} Bit

1 MByte = 1024 KByte = 1024*1024 Byte = 1 048 576 Byte = 2^{20} Byte
→ 1.000.000 Bytes = 0,9765625 KB

1 GByte = 1 Giga Byte = 1024 MByte = 2^{20} KByte = 2^{30} Byte = 1.073.741.824 Byte

1 TByte = 1 Tera Byte = 1024 GByte = 2^{20} MByte = 2^{30} KByte = 2^{40} Byte
= 1.099.511.627.776 Byte

1 PByte = 1 Peta Byte = 1024 TByte = 2^{30} MByte = 2^{40} KByte = 2^{50} Byte

1 EByte = 1 Exa Byte = 1024 PByte = 2^{40} MByte = 2^{50} KByte = 2^{60} Byte

1 ZByte = 1 Zetta Byte = 1024 EByte = 2^{50} MByte = 2^{60} KByte = 2^{70} Byte

1 YByte = 1 Yotta Byte = 1024 ZByte = 2^{60} MByte = 2^{70} KByte = 2^{80} Byte

Bits & Bytes - Maßeinheiten

Dezimalpräfixe 10^3,10^6,10^9,...

SI-Präfixe

k,M,G,T,...

für Bitraten, HD-Video, DVD

Binärpräfixe 2^{10},2^{20},2^{30},...

IEC 60027-2 (Nov.2000)

Kibi, Mebi, Gibi, ...

für RAM, HDD, USB-Sticks, SSD

SI = International System of Units
IEC = International Electrotechnical Commission

Zusammenführung von SI- und IEC-Präfixen in: IEC/ISO 80000

Bits / Bytes / Oktetts

Informatik

MSBF
(Most Signifikant Bit First)

$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$

LSBF
(Least Signifikant Bit First)

$b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7$

ITU-T / ISO
(X.680 / X.690) - MSBF

$b_8\ b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1$

INTERNET / IETF.org
(IP - RFC 791) - MSBF

$b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7$

Byteorder

LE = Little Endian (das niederwertige Byte steht zuerst)

II typisch für INTEL-, NEC-V800-, Alpha-, Atmel-AVR- und PICmicro-Plattform (80 10 = 16*256+128 = 4224)

256^0 256^1

BE = Big Endian (das höherwertige Byte steht zuerst)

MM typisch für Motorola-, SUN-Sparc-, MIPS- und Power-PC-Plattform (80 10 = 128*256 + 16 = 32784)

256^1 256^0

BOM (Byte Order Mark) = U+FEFF = FEFFH

Unicode CodePoint CP

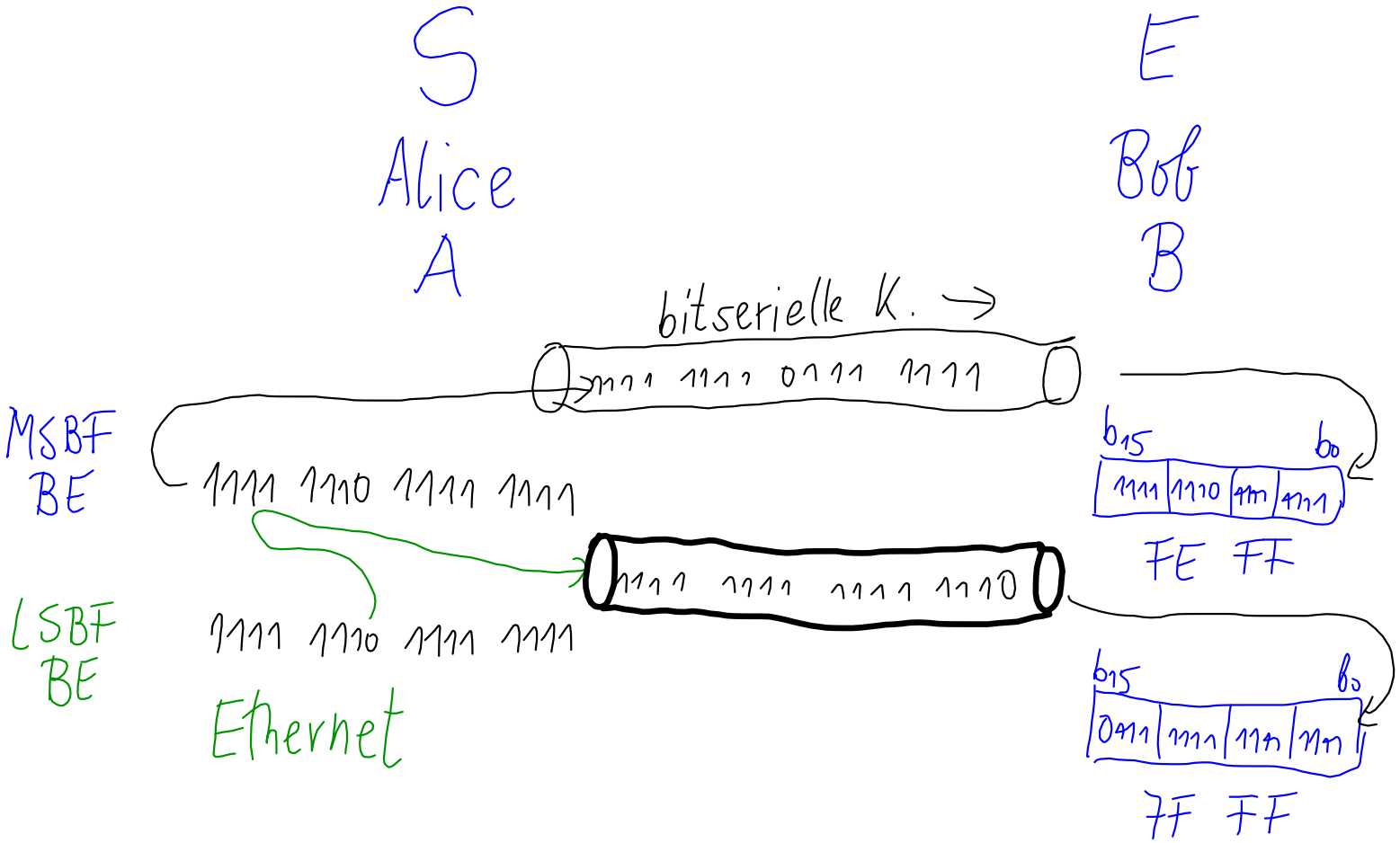
0x FEFF

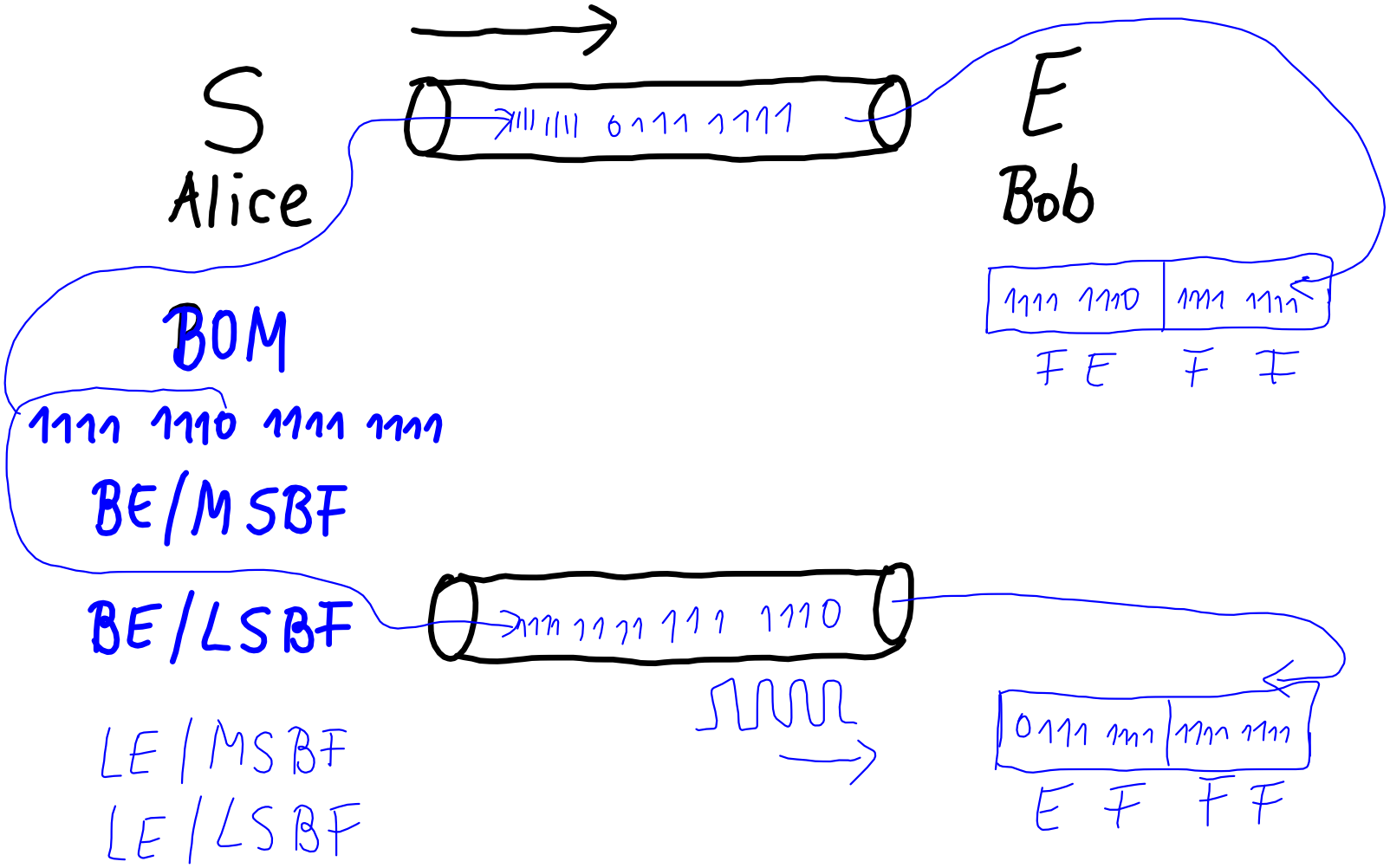
NUXI - Problem

UNIX	1 2 3 4	BE	für 4-Byte Worte
XINU	4 3 2 1	LE	
NUXI	2 1 4 3	Middle Endian	
IXUN	3 4 1 2	Mixed Endian	

Hex Fun:

BAADFØØD
BADCAB1E
CAFE
AFFE


S
Alice
A
E
Bob
B
bitserielle K. →
MSBF
BE
1111 1110 1111 1111
LSBF
BE
1111 1110 1111 1111
Ethernet
b15
b0
FE FF
b15
b0
7F FF


S
Alice
E
Bob
1111 1110 1111 1111
BOM
1111 1110 1111 1111
BE/MSBF
BE/LSBF
LE/MSBF
LE/LSBF
1111 1110 1111 1111
0111 1111 1111 1111
FE FF
EF FF

Die allgemeine Zahldarstellung einer Zahl z in einem Stellenwertsystem zur Basis B mit den Ziffern $z_i \in \{0,1,2,3,...,B-1\}$ lautet:

$$(z)_B = (z_n z_{n-1} \ldots z_0, z_{-1} \ldots z_{-m})_B = \sum_{i=-m}^{n} z_i B^i \qquad \text{für } (B \geq 2)$$

$$= \underbrace{(\ldots((z_n B + z_{n-1})B + z_{n-2})B + \ldots + z_1)B + z_0}_{\text{ganzzahliger Anteil}} + \underbrace{(\ldots((z_{-m} B^{-1} + z_{-m+1})B^{-1} + z_{-m+2})B^{-1} + \ldots + z_{-1})B^{-1}}_{\text{gebrochener Anteil}}$$

Hornerschema zur Berechnung des Wertes von z:

BOM (Byte Order Mark)

zur Signalisierung der Bit- und Byte-Order

BOM = U+FEFF = 1111 1110 1111 1111

BOM (binär)	MSBF	LSBF
BE	1111 1110 1111 1111	0111 1111 1111 1111
LE	1111 1111 1111 1110	1111 1111 0111 1111

BOM (hexadez.)	MSBF	LSBF
BE	FE FF	7F FF
LE	FF FE	FF 7F

UTF-8 Codierung am Bsp. des BOM

BOM für BE/MSBF: FEFF = 1111 1110 1111 1111 (RFC 3629)

UTF-8-Code-Templates

1 Byte	0xxxxxxx	(2^7 Zeichen)
2 Bytes	110xxxxx 10xxxxxx	(2^{11} Zeichen)
3 Bytes	1110xxxx 10xxxxxx 10xxxxxx	(2^{16} Zeichen)
4 Bytes	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	(2^{21} Zeichen)

16 Bit ⟶ 3 Bytes 1110xxxx 10xxxxxx 10xxxxxx

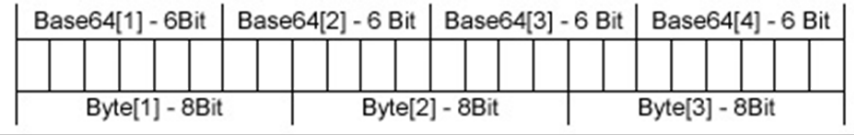
1110**1111** 10**111011** 10**111111**

EF **BB** **BF**

BOM in UTF-8	MSBF	LSBF
BE	EF BB BF	
LE		

BASE-64-Codierung am Bsp. des UTF-8-codierten BOM

(RFC 4648)



EF BB BF

3·8 = 24 Bits

4·6 = 24 Bits

0..63

$2^6 = 64$

Wert	Zeichen	Wert	Zeichen	Wert	Zeichen	Wert	Zeichen
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

16 32 8 4 2 1

111011 | 11 1011 | 1011 10 | 11 1111

59 59 46 63

"7" "7" "u" "/"

Zeichen des BASE-64-Alphabet

"77u/" BASE-64-String

3737752FH = 0x3737752F (4 Bytes → 33% Overhead)

BOM in UTF-8 in BASE-64	MSBF	LSBF
BE	"77u/" 0x37377752F	
LE		

Zusammenfassung

Bitorder: MSBF = Most Signifikant Bit First

LSBF = Least Signifikant Bit First

Byteorder: LE = Little Endian (= INTEL Byteorder)

BE = Big Endian

auch: Middle Endian, Mixed Endian, Bi-Endian

BOM = Byteorder Mark: FEFFH = 0xFEFF

(zur Signalisierung der Bit- und Byteorder) in UTF-8: EF BB BF

in UTF-8/BASE64: "77u/" = 0x3737752F

Informatik & Telekommunikation

- Informatik ist Wissenschaft, Technik und Anwendung der maschinellen Erfassung, Verarbeitung und Übertragung von Informationen. (Gesellschaft für Informatik e.V.)
- Computer:
  Das Wesen der Computer besteht darin, dass sie Informationen verschiedenster Art verarbeiten und ineinander umwandeln.
  Das zentrale Wirkprinzip eines Computers ist seine Programmierbarkeit.
- Telekommunikation (TKG §3 (22,23) 25.6.2004)
  „Telekommunikation“ ist der technische Vorgang des Aussendens, Übermittelns und Empfangens von Signalen mittels Telekommunikations-anlagen“

Informationsebenen

A) Statistik (Zeichenmenge, Auftrittswahrscheinlichkeit, Häufigkeit):
Der statistische Informationsgehalt einer Zeichenkette
(gemessen in Bits: BIT=binary digit)

B) Lexik (griech. lexikón (biblíon) = Wörterbuch):
Zeichensatz/Wortschatz einer Sprache zur Informationsdarstellung
(Alphabet/Wörter=Lexeme/Zeichencodierung)
Lexikologie = Wortlehre, Wortkunde, Wortschatzuntersuchung

C) Syntax (griech. syntaksis=Zusammen-Ordnung):
Regeln der Zusammenstellung von Zeichen und Wörtern / Beziehungen
der Zeichen untereinander (Sinn-Sätze als Träger semantischer
Information/Grammatik)

D) Semantik: (griech: σημαίνειν sēmainein „bezeichnen“)
Bedeutung der Zeichen/Wörter für ...? (Bedeutungslehre)
(Aussage, Sinn, Botschaft)

E) Pragmatik (griech. pragmatike =Handlungsaspekt, die Kunst richtig zu handeln):
beabsichtigte/ausgeführte/ausgelöste Handlung bzw. Tat

F) Apobetik (griech. αποβαίνοντα = Ergebnis, Erfolg):
beabsichtigtes/erreichtes Ziel bzw. Ergebnis

siehe auch: Semiotik, Syntaktik, Sigmatik

Informationskriterien

- Effektivität (Relevanz und Angemessenheit, Konsistenz, Richtigkeit, Verwendbarkeit)
- Effizient (Bereitstellung mit wirtschaftlichen Ressourcen)
- Vertraulichkeit
- Integrität
- Verfügbarkeit
- Complianz (Einhaltung externer und interner Regeln)
- Verlässlichkeit

Information ist zusätzliches zweckorientiertes Wissen.

Information hängt vom Wissensstand einer Person ab.

Informationstheorie

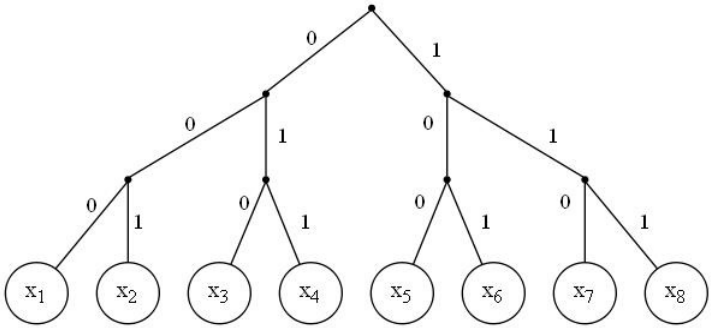
Begründer: Claude Shannon (1916-2001) in 1948

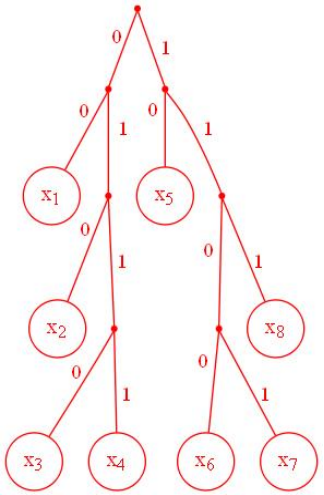
Ziel: quantitative Beschreibung des Informationsgehaltes einer Nachricht

Nachricht: besteht aus Zeichen $x_i \in X$ eines Alphabetes X

Die Wahrscheinlichkeit des Auftretens eines Zeichens x_i in der Nachricht des Senders sei $p_i=p(x_i)$ (Auftrittswahrscheinlichkeit)

Codebaum: mit Zeichen in den Blättern des Baumes





Präfix-Eigenschaft (Links-Fano-Bedingung): keine Codefolge ist selbst Anfang (Präfix) einer anderen Codefolge

010101101111... ?

Def.: Die Anzahl der Entscheidungsschritte H im Codebaum zur eindeutigen Identifizierung eines x_i wird als Entscheidungsgehalt des Zeichens x_i bezeichnet (H in Bit).

In einem balancierten Codebaum gilt: $H \leq \lceil ld\, n \rceil$ (n = Anzahl der Zeichen in X)

Def.: Entscheidungsgehalt einer Nachricht aus N Zeichen $H_N = N * ld\ n$

Def.: Informationsgehalt eines Zeichens $x_i \in X$

$$I(x_i) = ld\left(\frac{1}{p(x_i)}\right) = -ld\left(p(x_i)\right)$$ (in Bits pro Zeichen)

also: eine sichere Auftrittswahrscheinlichkeit bedeutet Null Information !

kleine $p_i = p(x_i)$ bedeutet viel Information!

Merke: Information = Grad der Unsicherheit

Def.: Der mittlere Informationsgehalt (= Entropie) einer Quelle wird definiert als:

$$H(x) = \sum_{i=1}^{n}\left(p(x_i) * I(x_i)\right) = \sum_{i=1}^{n}\left(p(x_i) * ld\left(\frac{1}{p(x_i)}\right)\right) = -\sum_{i=1}^{n}\left(p(x_i) * ld\left(p(x_i)\right)\right)$$

Def.: Sei l_i die Länge eines Codes des i-ten Zeichens der Quelle, dann ist die mittlere Codelänge L der Quelle definiert als:

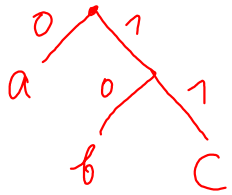
$$L = \sum_{i=1}^{n}(p(x_i) * l_i)$$

Bsp:

X	p_i	p_i	$I(x_i)$	$H(X)$	p_i	$I(x_i)$	$H(X)$	p_i
a	50%	$= \frac{1}{2} = 2^{-1}$	1	$\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1$	25% $= \frac{1}{4}$	2	0,81	100%
b	50%	$= \frac{1}{2} = 2^{-1}$	1	= 1	75% $= \frac{3}{4}$	2-ld3		0% ...

	p_i	$I(x_i)$	$H(X)$	p_i	$I(x_i)$	$H(X)$	Code	l_i
a	33%	ld3	$\frac{1}{3} \cdot$ ld3	50%	1		0	1
b	33%	ld3	$\frac{1}{3}$ ld3 +	25%	2	1,5	10	2
c	33%	ld3	$\frac{1}{3}$ ld3 +	25%	2		01	2
			= ld3					

$1{,}5 = H(X) = \sum p_i \cdot l_i = 1{,}5 = L$



x(i)	p(i)	I(x(i))	H(x)	Code	I(i)	L
a	50%	1	0,5*1+0,5*1=1	0	1	0,5*1+0,5*1=1
b	50%	1		1	1	

a	25%	2	0,25*2+0,75(2-ld 3)=0,81	0	1	0,25*1+0,75*1=1
b	75%	2-ld3		1	1	

a	33%	ld 3	0,333*ld 3+0,333*ld 3*0,333*ld 3=ld 3	0	1	0,33*1+0,33*2+0,33*2=1,66666
b	33%	ld 3		10	2	
c	33%	ld 3		11	2	

a	25%	2	0,25*2+0,25*2+0,5*1=1,5	11	2	0,25*2+0,25*2+0,5*1=1,5
b	25%	2		10	2	
c	50%	1		0	1	

a	25%	2	0,25*2+0,25*2+0,25*2+0,25*2=2	00	2	0,25*2+0,25*2+0,25*2+0,25*2=2
b	25%	2		01	2	
c	25%	2		10	2	
d	25%	2		11	2	

a	12,5%	3	0,125*3+1,125*3+0,25*2+0,5*1=1,75	111	3	0,125*3+1,125*3+0,25*2+0,5*1=1,75
b	12,5%	3		110	3	
c	25%	2		10	2	
d	50%	1		0	1	

a	20%	ld 5	0,2*ld 5+0,2*ld 5+0,2*ld 5+0,2*ld 5+0,2*ld 5=ld 5=2,32193	111	3	0,2*3+0,2*3+0,2*3+0,2*3+0,2*1=2,6
b	20%	ld 5		110	3	
c	20%	ld 5		101	3	
d	20%	ld 5		100	3	
e	20%	ld 5		0	1	

a	6,25%	4	0,0625*4+0,0625*4+0,125*3+0,25*2+0,5*1=1,875	1111	4	0,0625*4+0,0625*4+0,125*3+0,25*2+0,5*1=1,875
b	6,25%	4		1110	4	
c	12,5%	3		110	3	
d	25%	2		10	2	
e	50%	1		0	1	

Nach Shannon gilt: $H(x) \leq L \leq H(x) + 1$

d.h. die mittlere Codelänge einer optimalen Codierung
ist stets $\geq$ der Entropie der zu codierenden Nachricht.

Problem: Kann evtl. eine (minimale) Codierung gefunden werden mit H(x) = L ?

Codierungen:

- https://de.wikipedia.org/wiki/Shannon-Fano-Kodierung
- https://de.wikipedia.org/wiki/Arithmetisches_Kodieren
- https://de.wikipedia.org/wiki/Tunstall-Kodierung
- https://en.wikipedia.org/wiki/Tunstall_coding

Lösung: Huffman Codierung von David A.Huffman (1929-99/USA) in 1952

- https://de.wikipedia.org/wiki/Huffman-Kodierung

Coderedundanz:= R(X) = L(X) - H(X)

z.B. n=10, x_i mit je 4 Bit codiert

$(\ell_i = 4)$

$\rightarrow$ R(X) = 4 - ld 10 ~ 0,68

3,32

relative Coderedundanz:=

R(X) = [L(X) - H(X)] / L(X) = 1 - H(X)/L(X)

$= 1 - \frac{3,32}{4} \sim 17\%$

Dt. Schriftsprache 26 Buchstaben

a) Gleichverteilung der Buchstabenhäufigkeit $\Rightarrow$ Entropie = 4,7 Bit/Sym

b) unter Berücksichtigung der realen Buchstabenhäufigkeit
$\Rightarrow$ Entropie = 4,1 Bit/symbol

c) bei Beachtung von Silben mit einer mittleren Symbollänge von 3
$\Rightarrow$ Entropie = 2,8 Bit/symbol

d) Grenzwert der Entropie deutscher Texte $\sim$ 1,3 Bit/symbol (siehe Küpfmüller)
(aber Taschenbuch der TK S.41) 1,6 Bit/symbole

Küpfmüller et.al.:Einführung in die Theoretische Elektrotechnik. Springer

absolute Redundanz: $R = L - E$ = mittlere Codelänge - Entropie z.B. $L = \mathrm{ld}\, N$ (N: Anzahl der Quellsymbole)

relative Redundanz: $r_{rel} = \frac{L-E}{L} = 1 - \frac{E}{L}$

r_{rel} (dt. Sprache) $= 1 - \frac{1,3}{4,7} = \frac{3,4}{4,7} = 73\%$

$(= 0,66 = 66\%)$ nach Taschenbuch der TK

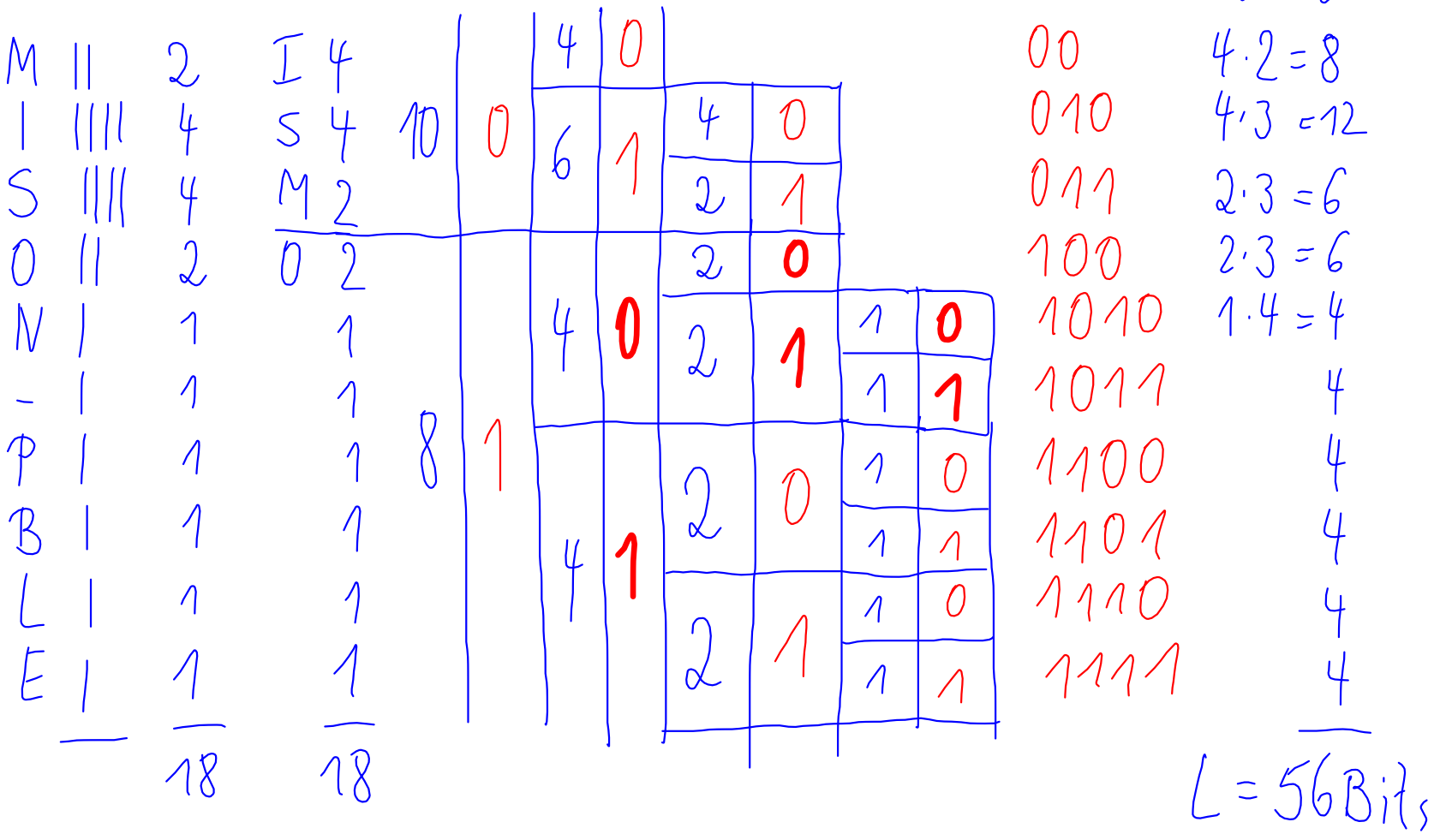
Übung

Bsp.: Shannon-Fano-Codierung
"MISSISSIPIHIPPIE"

Bsp.: Huffman-Codierung
"MISSISSIPIHIPPIE"

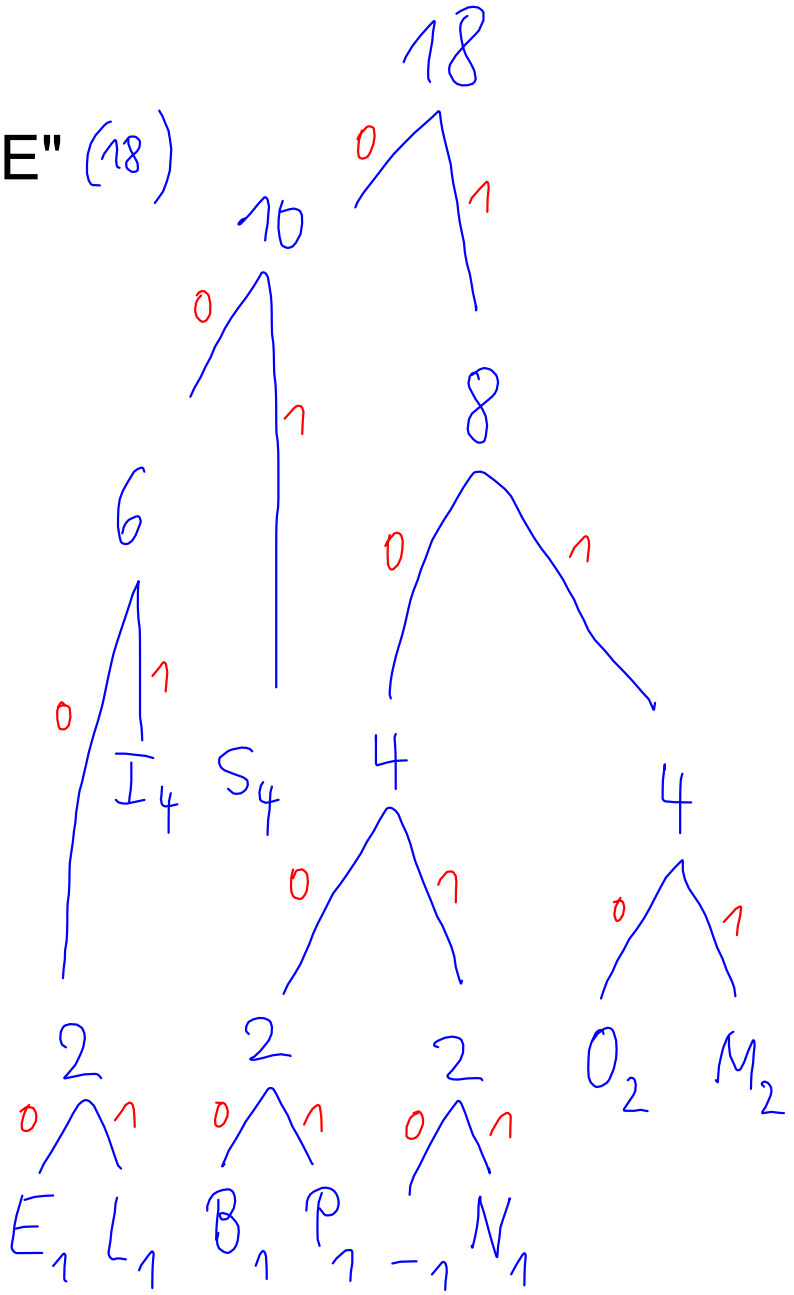
Bsp.: Tunstall-Codierung
"MISSISSIPIHIPPIE"

Bsp.: Shannon-Fano-Coding
"MISSION_IMPOSSIBLE" (18)



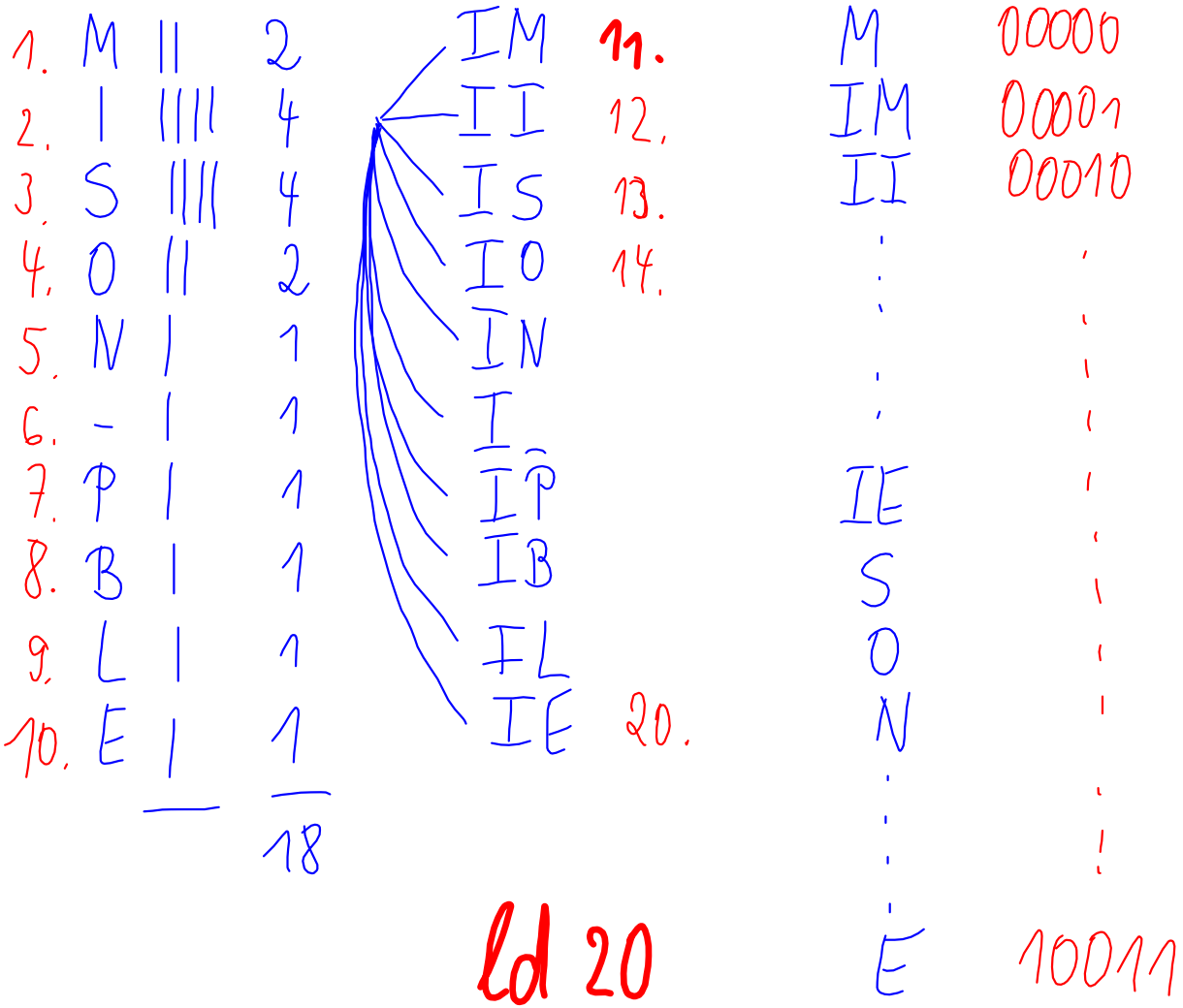
Bsp.: Huffman-Coding
"MISSION_IMPOSSIBLE" (18)

				$h_i * l_i$
M	II	2	111	2·3 = 6
I	IIII	4	001	4·3 = 12
S	IIII	4	01	4·2 = 8
O	II	2	110	2·3 = 6
N	I	1	1011	4
-	I	1	1010	4
P	I	1	1001	4
B	I	1	1000	4
L	I	1	0001	4
E	I	1	0000	4
	18			L = 56 Bits



Bsp.: Tunstall-Coding (äquidistante Codelänge)

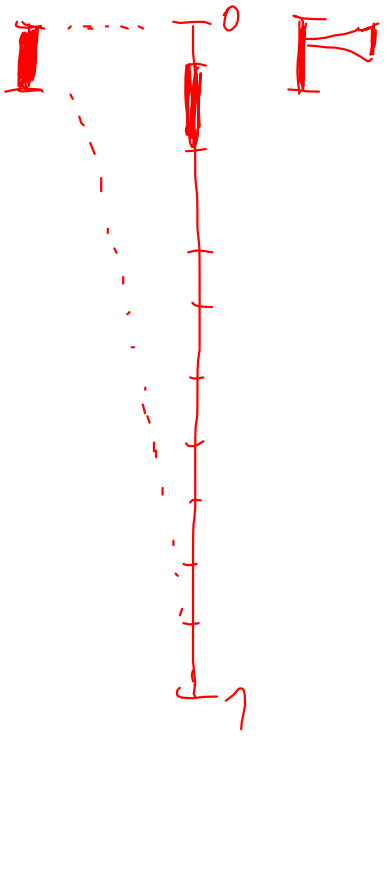
"MISSION_IMPOSSIBLE" (18)



Bsp.: arithmetische Coding

"MISSION_IMPOSSIBLE"

M	II	2	1/9
I	IIII	4	2/9
S	IIII	4	2/9
O	II	2	1/9
N	I	1	1/18
-	I	1	
P	I	1	
B	I	1	
L	I	1	
E	I	1	1/18
		18	

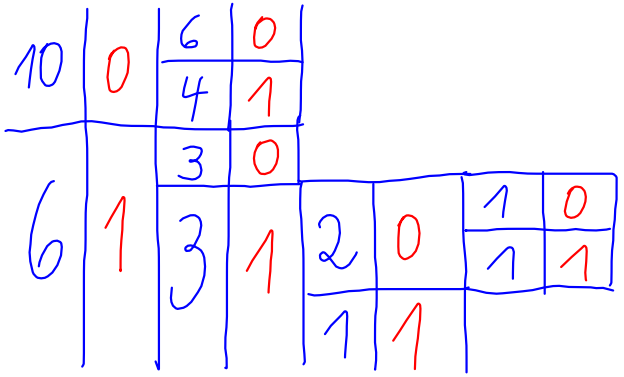


I ← jede Zahl aus dem Intervall codiert die Nachricht als Ganzes

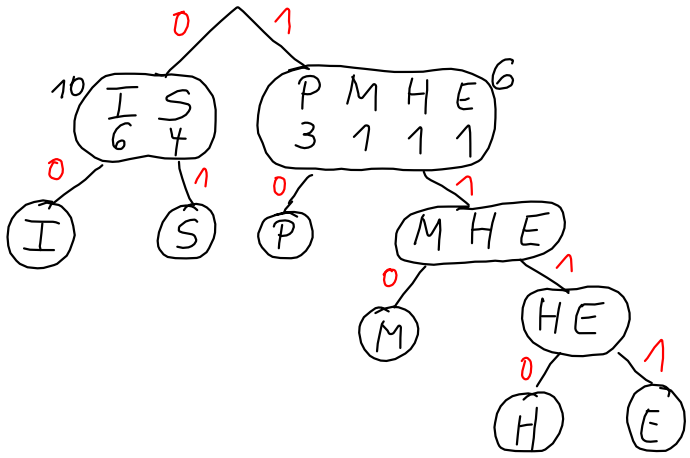
Bsp.: Shannon-Fano-Codierung "MISSISSIPIHIPPIE"

M	I	1
I	### I	6
S	IIII	4
P	III	3
H	I	1
E	I	1
		16

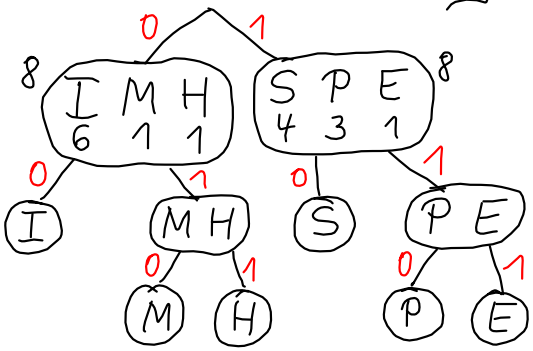
I	6
S	4
P	3
M	1
H	1
E	1
	16



		$h_i * l_i$
I 6	00	12
S 4	01	8
P 3	10	6
M 1	1100	4
H 1	1101	4
E 1	111	3
		37 Bits



mit anderer Partitionierung

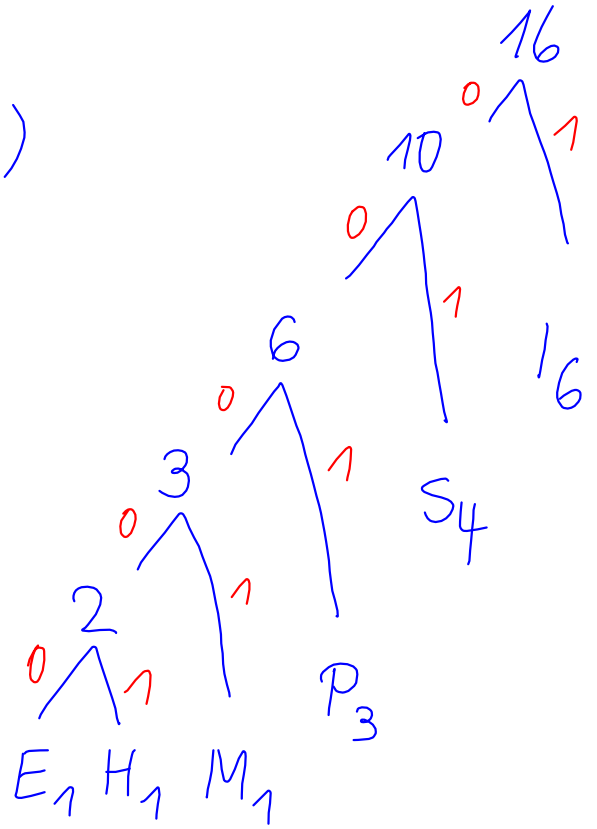


		$h_i * l_i$
I	2	12
S	2	8
P	3	9
M	3	3
H	3	3
E	3	3
		38

Bsp.: Huffman-Codierung

"MISSISSIPIHIPPIE" (16)

		h_i	l_i	$h_i * l_i$
M	I	1	0001	4
I	HH I	6	1	6
S	IIII	4	01	8
P	III	3	001	9
H	I	1	00000	5
E	I	1	00001	5
		16		37 Bits



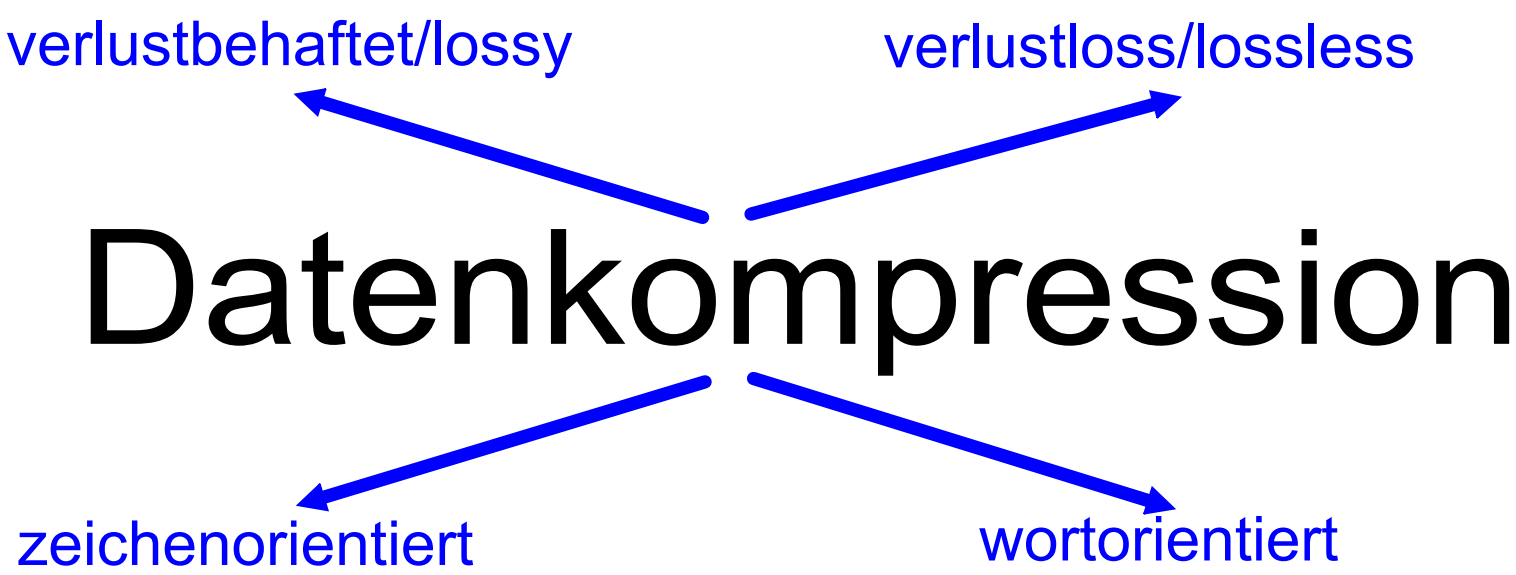
Algorithmus: Shannon Fano Codierung

(top down)

- Zeichenhäufigkeiten bestimmen und von groß nach klein sortieren
- Teile die Menge der auftretenden Zeichen bzgl. der Summe der Häufigkeiten in zwei möglichst gleichgroße Teil-Mengen (Partitionierung). Diese Partitionierung erfolgt gemäß der Reihenfolge der sortierten Häufigkeiten, d.h. es werden beginnend bei dem am häufigsten auftetenden Zeichen solange Zeichen in die erste der beiden Teilmengen aufgenommen bis die aggregierte Zeichenhäufigkeit 50% der Gesamtzeichenhäufigkeit erreicht bzw. überschreitet. Die restlichen Zeichen werden jeweils in die zweite Teilmenge aufgenommen.
- Solange noch keine einelementige Menge resultiert unterteile diese Teilmengen weiter nach dem gleichen Kriterien
- Dadurch entsteht ein Codebaum mit den Zeichen in den Blätter (Prefix-Eigenschaft ist automatisch erfüllt)

Algorithmus: Huffman Codierung
(buttom up)

- Zeichenhäufigkeiten bestimmen und von groß nach klein sortieren
- Ordne jeweils den Zeichen einen Vaterknoten zu, die jeweils die minimalen Häufigkeiten von Zeichen bzw. der vorherig entstandenen Vaterknoten zugeordneten aggregierte Häufigkeiten (also die Summe der Häufigkeiten der beiden Sohnknoten) aufweisen
- Dadurch entsteht ein bittom-up ein binärer Codebaum mit den Zeichen in den Blätter (Prefix-Eigenschaft ist automatisch erfüllt)



Shannon Fano Encoding
Huffman Encoding
Arithmetic Encoding
RLE = Run Length Encoding / RLC = Run Length Coding
LZ = Lempel Ziv Kompression 1977 (LZ78)
LZW = Lempel Ziv Welch Kompression 1984
DCT = Discrete Cosinus Transformation

RLE (Run-length encoding)
RLC (Run-length coding)

Bsp.: a=(1,1,0,0,0,1,0,0,1,1,1,1,0,0,0,0,0,1,1,0)

0 2 3 1 2 4 5 2 1
0 1 0 1 0 1 0 1 0

Konvention: die erste Lauflänge ist eine Nullen-Lauflänge

Lauflängenvektor = (0, 2, 3, 1, 2, 4, 5, 2, 1) = LL
000 010 011 001 010 100 101 010 001
LL_1 ..... LL_m

IN: n, a_i i=1(1)n
l=1; m=0;
IF (a_1==1) THEN {m++; LL_m=0}
FOR (i=2; i≤n; i++) DO
  IF (a_{i-1} == a_i) THEN {l++;}
  ELSE {m++; LL_m=l; l=1;}
m++; LL_m=l;
OUT: LL_j j=1(1)m

i++ ≙ i=i+1

```
E: n, a_i mit i=1(1)n
L=1; m=0
IF (a_1 == 1) THEN m++; LL_m=0;
FOR (i=2; i<=n; i++) DO
   IF (a_i-1 == a_i) THEN L++
   ELSE {m++; LL_m=L; L=1;}
m++; LL_m = L
A: LL_i mit i=1(1)m
```

CCITT F3-Faxcodierung

MIME-Typ image/g3fax mit Dateiendung .g3
ITU-T-Empfehlungen: T.30
T.37 für asynchrone Übertragung
T.38 für synchrone Übertragung

```
/* Tabelle der schwarzen Lauflängen */
char *ccitt_bla[] =
{
   "0 0000110111",
   "1 010",
   "2 11",
   "3 10",
   "4 011",
   "5 0011",
   "6 0010",
   "7 00011",
   "8 000101",
   "9 000100",
   "10 0000100",
   "11 0000101",
   "12 0000111",
   "13 00000100",
   "14 00000111",
   "15 000011000",
   "16 0000010111",
   "17 0000011000",
   "18 0000001000",
   "19 00001100111",
   "20 00001101000",
   "21 00001101100",
   "22 00000110111",
   "23 00000101000",
   "24 00000010111",
   "25 00000011000",
   "26 000011001010",
   "27 000011001011",
   "28 000011001100",
   "29 000011001101",
   "30 000001101000",
   "31 000001101001",
   "32 000001101010",
   "33 000001101011",
   "34 000011010010",
   "35 000011010011",
   "36 000011010100",
   "37 000011010101",
   "38 000011010110",
   "39 000011010111",
   "40 000001101100",
   "41 000001101101",
   "42 000011011010",
   "43 000011011011",
   "44 000001010100",
   "45 000001010101",
   "46 000001010110",
   "47 000001010111",
   "48 000001100100",
   "49 000001100101",
   "50 000001010010",
   "51 000001010011",
   "52 000000100100",
   "53 000000110111",
   "54 000000111000",
   "55 000000100111",
   "56 000000101000",
   "57 000001011000",
   "58 000001011001",
   "59 000000101011",
   "60 000000101100",
   "61 000001011010",
   "62 000001100110",
   "63 000001100111",
   "64 0000001111",
   "128 000011001000",
   "192 000011001001",
   "256 000001011011",
   "320 000000110011",
   "384 000000110100",
   "448 000000110101",
   "512 0000001101100",
   "576 0000001101101",
   "640 0000001001010",
   "704 0000001001011",
   "768 0000001001100",
   "832 0000001001101",
   "896 0000001110010",
   "960 0000001110011",
   "1024 0000001110100",
   "1088 0000001110101",
   "1152 0000001110110",
   "1216 0000001110111",
   "1280 0000001010010",
   "1344 0000001010011",
   "1408 0000001010100",
   "1472 0000001010101",
   "1536 0000001011011",
   "1600 0000001011011",
   "1664 0000001100100",
   "1728 0000001100101",
   "1792 00000001000",
   "1856 00000001100",
   "1920 00000001101",
   "1984 000000010010",
   "2048 000000010011",
   "2112 000000010100",
   "2176 000000010101",
   "2240 000000010110",
   "2304 000000010111",
   "2368 000000011100",
   "2432 000000011101",
   "2496 000000011110",
   "2560 000000011111",
   "9999 000000000001",
   NULL
};
```

```
/* Tabelle der weissen Lauflaengen */
char *ccitt_whi[] =
{
   "0 00110101",
   "1 000111",
   "2 0111",
   "3 1000",
   "4 1011",
   "5 1100",
   "6 1110",
   "7 1111",
   "8 10011",
   "9 10100",
   "10 00111",
   "11 01000",
   "12 001000",
   "13 000011",
   "14 110100",
   "15 110101",
   "16 101010",
   "17 101011",
   "18 0100111",
   "19 0001100",
   "20 0001000",
   "21 0010111",
   "22 0000011",
   "23 0000100",
   "24 0101000",
   "25 0101011",
   "26 0010011",
   "27 0100100",
   "28 0011000",
   "29 00000010",
   "30 00000011",
   "31 00011010",
   "32 00011011",
   "33 00010010",
   "34 00010011",
   "35 00010100",
   "36 00010101",
   "37 00010110",
   "38 00010111",
   "39 00101000",
   "40 00101001",
   "41 00101010",
   "42 00101011",
   "43 00101100",
   "44 00101101",
   "45 00000100",
   "46 00000101",
   "47 00001010",
   "48 00001011",
   "49 01010010",
   "50 01010011",
   "51 01010100",
   "52 01010101",
   "53 00100100",
   "54 00100101",
   "55 01011000",
   "56 01011001",
   "57 01011010",
   "58 01011011",
   "59 01001010",
   "60 01001011",
   "61 00110010",
   "62 00110011",
   "63 00110100",
   "64 11011",
   "128 10010",
   "192 010111",
   "256 0110111",
   "320 00110110",
   "384 00110111",
   "448 01100100",
   "512 01100101",
   "576 01101000",
   "640 01100111",
   "704 011001100",
   "768 011001101",
   "832 011010010",
   "896 011010011",
   "960 011010100",
   "1024 011010101",
   "1088 011010110",
   "1152 011010111",
   "1216 011011000",
   "1280 011011001",
   "1344 011011010",
   "1408 011011011",
   "1472 010011000",
   "1536 010011001",
   "1600 010011010",
   "1664 011000",
   "1728 010011011",
   "1792 00000001000",
   "1856 00000001100",
   "1920 00000001101",
   "1984 000000010010",
   "2048 000000010011",
   "2112 000000010100",
   "2176 000000010101",
   "2240 000000010110",
   "2304 000000010111",
   "2368 000000011100",
   "2432 000000011101",
   "2496 000000011110",
   "2560 000000011111",
   "9999 000000000001",
    NULL
};
```

LZ-Kompression (Lempel-Ziv)

Codier-Prinzip:

- Aufbau eines Wörterbuches mit Teilzeichenketten variabler Länge
- Codiere diese Teilzeichenketten mit dem Index ihres Wörterbuchplatzes

1) sequentielles parsen der Quellzeichenkette und zerlegen in Teilstrings, die bisher noch nicht aufgetreten sind, d.h. solange ein Teilstring und das folgende Symbol noch nicht im Wörterbuch stehen, erweitere den Teilstring um dieses Zeichen und trage das Resultat in das Wörterbuch ein

2) codiere die Quellzeichenkette mittels der Nummer/Index eines im Wörterbuch gefundenen Teilstrings erweitert um den Code des nachfolgenden Zeichens.

https://de.wikipedia.org/wiki/Lempel-Ziv-Welch-Algorithmus

http://www.ufg.ac.at/zid/public_html/funk/medientechnik/material/LZW.html

LZW-Kompression

Erzeugung von k-Bit langen Codes mit k>8 (üblich ist k=12). $2^{12} = 4096$

Die Codes 0..255 entsprechen dabei den üblichen Zeichencodes und den Codes 256..(2^k-1) werden ganze Zeichenketten zugeordnet.

```
FUNCTION LZW_Kompression();
      BEGIN String:= erstes Quellzeichen;
            WHILE  noch ein Quellzeichen vorhanden  DO
            BEGIN Zeichen:= nächstes Quellzeichen;
                  IF String+Zeichen in der Zeichenkettentabelle THEN
                     String:=String + Zeichen
                  ELSE
                   BEGIN Zielzeichen:=LZW_Code des Strings;
                         lege String+Zeichen in der Tabelle ab;
                         String:=Zeichen;
                   END;
                  Zielzeichen:=LZW_Code des Strings;
            END
      END;
```

Die Zeichenkettentabelle braucht nicht mit in die Zieldatei übernommen werden, sondern wird automatisch bei der Dekompression generiert.

Beispiel:

Eingabetext: '/WED/WE/WEE/WEB/WET' (Länge=19 Bytes)

Lösung: Zielcodes=('/','W','E','D',256,'E',260,261,257,'B', 260,'T')

(bei 10 Bit- Code:12*10 Bit, d.h.15 Bytes)

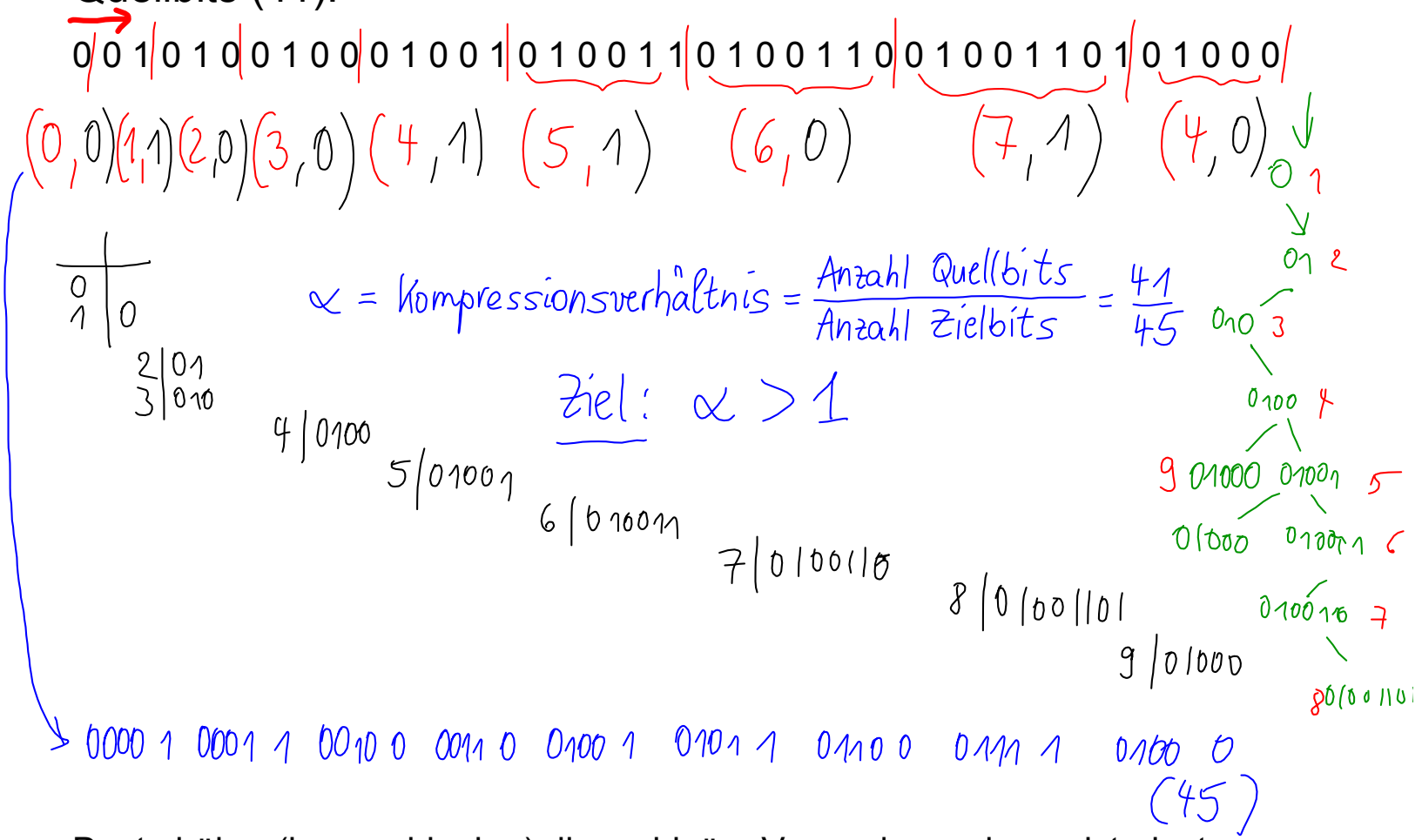
mit 256='/W',257='WE',258='ED',259='D/',260='/WE',261='E/',

262='/WEE',263='E/W',264='WEB',265='B/',266='/WET

```
FUNCTION LZW_Dekompression();
      BEGIN Old_Code:=erstes Zeichen;
            Zielzeichen:=Old_Code;
            WHILE noch ein Quellzeichen vorhanden DO
            BEGIN New_Code:=nächstes Quellzeichen;
                  IF New_Code nicht in der Zeichenkettentabelle THEN
                   BEGIN String:=Übersetzung von Old_Code;
                         String:=String+Zeichen;
                   END
                  ELSE String:=Übersetzung von New_Code;
                  Zielzeichen:=String;
                  Zeichen:=erstes Zeichen von String;
                  nimm Old_Code+Zeichen in der Tabelle auf;
                  Old_Code:=New_Code;
            END
      END;
```

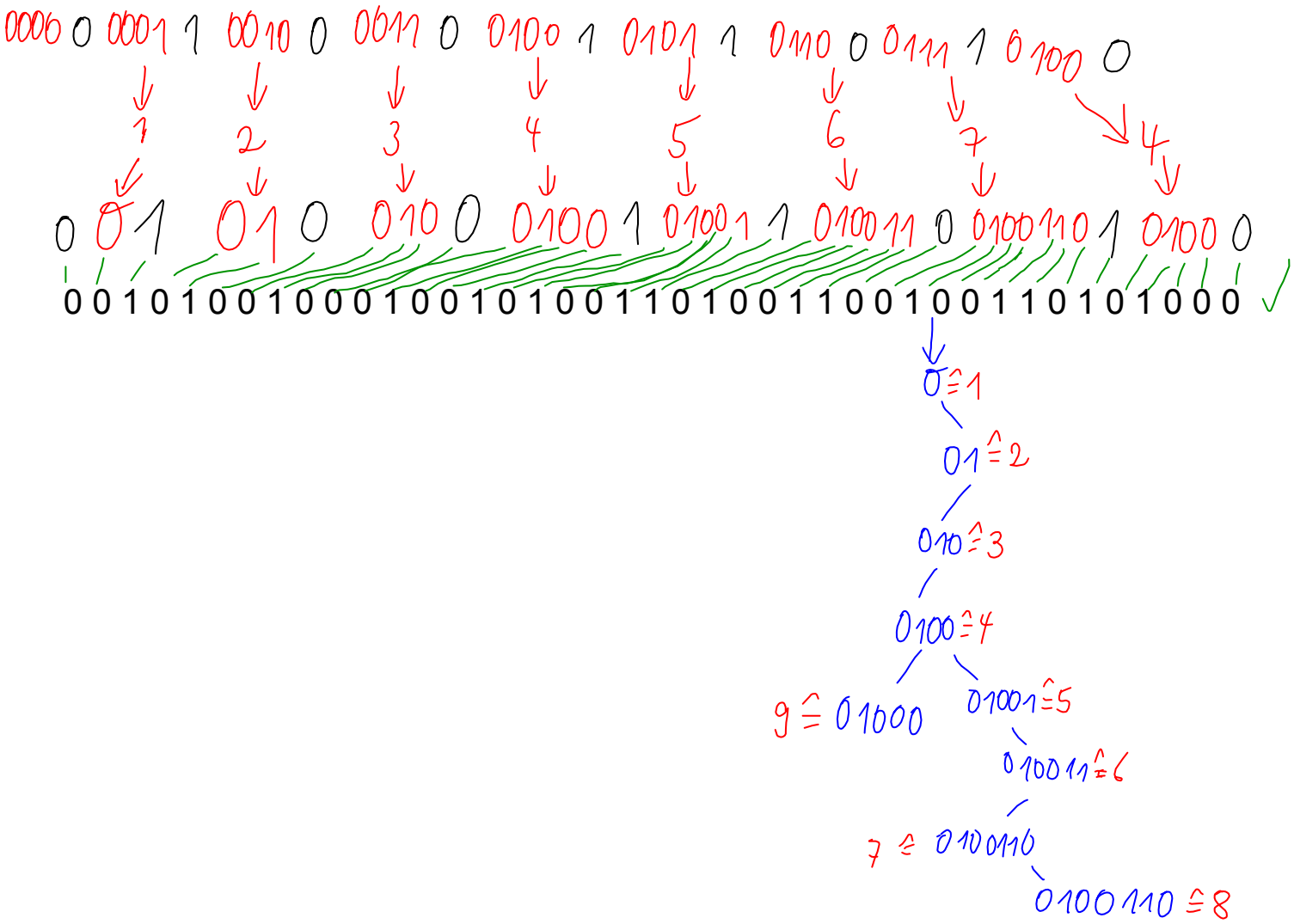
LZ-Kompression (Lempel-Ziv)

Quellbits (41):



Desto höher (bzw. schlanker) dieser binäre Verzweigungsbaum ist, desto besser ist die Kompression. Beim Entkomprimieren wird glz. der Baum aus der komprimierten Folge erstellt.

Dekompression (beim Empfänger)



Bsp.: 29 Quellbits

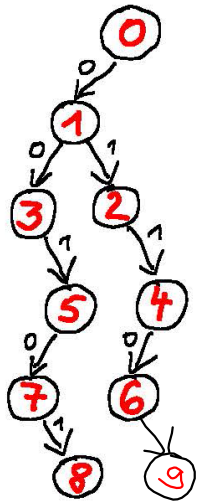
0 0 1 0 0 0 1 1 0 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1

Quellfolge: 0/0 1/0 0/0 1 1/0 0 1/0110/0010/00101/01101

(29 Bits) (0,0)(1,1)(1,0)(2,1)(3,1)(4,0)(5,0)(7,1)(6,1)

1 2 3 4 5 6 7 8 9

36 Bits: 000 0 001 1 001 0 010 1 011 1 100 0 101 0 111 1 110 1



Da Knoten 8 und 9 nicht wieder verwendet werden, reichen 3 Bit zur Codierung der Knoten 0..7.

Eine Verbesserung des Algorithmus kann durch die Speicherung der Tabelle als Hash-Tabelle mit Primzahllänge erreicht werden. Üblich ist z.B. ein 12-Bit LZW-Code. Hierbei sind die Codes 256 bis 4095 für Zeichenketten reserviert.

Die LZW-Kompression führt unter günstigen Umständen bis zu 90% Platzersparnis.

Informationsquellen:

Terry Welch: A Technique for High-Performance Data Compression. COMPUTER, Juni 1984

Ziv/A.Lempel: A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory, Mai 1977

https://de.wikipedia.org/wiki/Lempel-Ziv-Welch-Algorithmus

Abkürzungen (Bits & Bytes)

- ACE = ASCII compatible encoding
- ASCII = American Standard Code of Information Interchange
- ANSI = American National Standard Institute
- BASE = Basic Encoding
- BE = Big Endian
- BER = Basic Encoding Rules (nach ITU-T X.690)
- BIT = Binary Digit
- BOM = Byte Order Mark (=U+FEFF)
- CP = Codepoint
- IDN = Internet Domain Name
- LAN = Local Area Network
- LE = Little Endian
- LSBF = Least Significant Bit First
- MIME = Multimedia Internet Mail Extension (RFC 1341)
- MSBF = Most Significant Bit First
- NOR = NOT OR
- NAND = NOT AND
- PCP = Postsches Korrespondenzproblem
- RFC = Request for Comments (Internet Standards)
- IEC = International Electrotechnical Commission
- UCS = Universal Character Set (nach ISO 10646)
- UTF = Unicode Transformation Format
- WAN = Wide Area Network