

04 – Codierungs- und Informationstheorie

Grundzüge digitaler Systeme (192.134)

Vortrag von: Stefan Neumann

■ Bisher:

- Haben uns mit Zahlendarstellung im Computer beschäftigt
- Wie drückt man Dezimalzahlen mit Bits aus?
- Bits konnten „perfekt“ gespeichert werden

■ Jetzt:

- Darstellung von Buchstaben mit Bits
- Was können wir tun, wenn einige Bits „fehlerhaft“ sind?
- Wie komprimiert man Daten effektiv?

Codierungs- und Informationstheorie – Übersicht

- 1 Zeichencodierungen
- 2 Luhn-Algorithmus, ISBN-10 und ISBN-13
- 3 Codierungstheorie
- 4 Fehlererkennende und fehlerkorrigierende Codes
 - Hamming-Distanz
 - Fehlerkorrigierende Codes
 - Hamming-Code
- 5 Arten von Codes
- 6 Informationstheorie und Kompression
 - Informationstheorie nach Shannon
 - Huffman-Code

Zeichencodierungen

- ASCII
- Unicode
- UTF
- ISO-Latin-1 (ISO 8859-1)

- Binärcodierung von Zeichen
- American Standard Code for Information Interchange (ASCII)
 - Vom American National Standards Institute (ANSI) festgelegt
 - 7 Bits zur Codierung (in der ursprünglichen Version)
 - Es lassen sich also $2^7 = 128$ Zeichen darstellen
- Seit 1963: US-ASCII-Code
 - Einführung von Kleinbuchstaben
 - 8. Bit als Prüfbit
 - Beschränkung auf 128 Zeichen – Probleme bei Sonderzeichen!

US-ASCII-Code-Tabelle (nicht prüfungsrelevant)

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

- Erst Spalte, dann Zeile auslesen
- Beispiele:
 - Q wird dargestellt als $(51)_{16} = (0101\ 0001)_2$
 - m wird dargestellt als $(6D)_{16} = (0110\ 1101)_2$
 - 9 wird dargestellt als $(39)_{16} = (0011\ 1001)_2$

- 1991: Vereinheitlichung aller Zeichen im **Unicode 1.0**
 - 16-Bit-Code
 - Darstellung von $2^{16} = 65\,536$ Zeichen
 - US-ASCII-Code gleich zu Beginn des Coderaums
- Internationaler Standard
 - Zur Zusammenfassung bekannter Textzeichen in einem Zeichensatz
 - Ursprünglich als 2-Byte-Code konzipiert
 - Inzwischen auch 4-Byte-Variante
 - Mit der Möglichkeit, die Codierung weiterer Zeichen zu standardisieren
- Beispiel: „☎“
 - Hex: 260E
 - Binär: 0010 0110 0000 1110
- <http://unicode-table.com/de/>

UTF-Codierung

- Unicode Transformation Format (UTF)
 - Verfahren zur Abbildung von Unicode-Zeichen auf Byte-Folgen
 - De-facto-Standard für Zeichencodierung im Internet (98.8% Stand Okt. 2025¹)
- UTF-8
 - Je nach Zeichen 1, 2, 3 oder 4 Bytes (variable Länge!)
 - Ausgefeiltes Verfahren, um Texte (aus einem lateinischen Alphabet) mit möglichst wenig Bytes darzustellen
 - Mit 1 Byte: Alle ASCII-Zeichen wie in erweiterter ASCII-Code-Tabelle
 - Erste 128 Zeichen des Unicode ident mit ASCII
 - Mit 2 Byte: Andere Sonderzeichen wie Umlaute
 - Mit bis zu 4 Byte: Kyrillische, fernöstliche und afrikanische Sprachen
 - Standard-Codierung Unix/Linux
- UTF-16
 - Je nach Zeichen 2 oder 4 Bytes (variable Länge!)
 - Standard-Codierung Windows, Java

¹https://w3techs.com/technologies/history_overview/character_encoding/ms/y

UTF-Codierung (nicht prüfungsrelevant)

Unicode-Bereich (hexadezimal)	UTF-8-Codierung (binär)	Bemerkungen	Möglichkeiten (theoretisch)	
0000 0000 – 0000 007F	0xxxxxxx	In diesem Bereich (128 Zeichen) entspricht UTF-8 genau dem US-ASCII -Code: Das höchste Bit ist 0, die restliche 7-Bit-Kombination ist das ASCII-Zeichen.	2^7	128
0000 0080 – 0000 07FF	110xxxxx 10xxxxxx	Das erste Byte beginnt immer mit 11, die folgenden Bytes mit 10. Die xxxxx stehen für die Bits des Unicode-Zeichenwerts. Dabei wird das niederwertigste Bit des Zeichenwerts auf das rechte x im letzten Byte abgebildet, die höherwertigen Bits fortschreitend <i>von rechts nach links</i> . Die Anzahl der Einsen vor der ersten 0 im ersten Byte ist gleich der Gesamtzahl der Bytes für das Zeichen. (In Klammern jeweils die theoretisch maximal möglichen.)	$2^{11} - 2^7$ (2^{11})	1920 (2048)
0000 0800 – 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx		$2^{16} - 2^{11}$ (2^{16})	63.488 (65.536)
0001 0000 – 0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx		2^{20} (2^{21})	1.048.576 (2.097.152)

Quelle: Wikipedia

ISO-Latin-1 (ISO 8859-1)

- Zweithäufigster Zeichensatz im Web (1,0% Stand Okt. 2025¹)
- Versucht, möglichst viele Zeichen westeuropäischer Sprachen abzudecken
- 8-Bit Zeichencodierung
- Führende 0 + ASCII
- Führende 1 + 96 weitere darstellbare Zeichen
- Ähnlich: Windows-1252 Westeuropäisch
- Unterschiede zu Unicode und Windows-1252
 - Ein paar Sonderzeichen
 - Steuerzeichen
 - Kein Euro-Zeichen €

¹https://w3techs.com/technologies/history_overview/character_encoding/ms/y

Tabelle für ISO/IEC 8859-1 (nicht prüfungsrelevant)

Code	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	nicht belegt															
1...	nicht belegt															
2...	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3...	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4...	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6...	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8...	nicht belegt															
9...	nicht belegt															
A...	NBSP	ı	€	£	¤	¥	¦	§	¨	©	ª	«	¬	SHY	®	¯
B...	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C...	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D...	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E...	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F...	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Quelle: Wikipedia

SP ... space, NBSP ... non-breaking space, SHY ... soft hyphen

Codierungs- und Informationstheorie – Übersicht

- 1 Zeichencodierungen
- 2 Luhn-Algorithmus, ISBN-10 und ISBN-13
- 3 Codierungstheorie
- 4 Fehlererkennende und fehlerkorrigierende Codes
 - Hamming-Distanz
 - Fehlerkorrigierende Codes
 - Hamming-Code
- 5 Arten von Codes
- 6 Informationstheorie und Kompression
 - Informationstheorie nach Shannon
 - Huffman-Code

Luhn-Algorithmus

- Einfache Dezimal-Prüfziffer zur Fehlererkennung
 - Typischerweise an letzter Stelle angehängt
 - Wird zum Beispiel bei Kreditkarten verwendet
- **Validierung (Regel):**
 - Von rechts Zahlen durchlaufen und addieren, dabei jede zweite Ziffer doppelt zählen
 - Falls beim Verdoppeln eine Zahl > 9 herauskommt, anschließend 9 subtrahieren ($2 \times 7 - 9$)
 - Falls Gesamtsumme $\equiv 0 \pmod{10}$: gib "valide" aus, sonst "nicht valide"
- **Prüfziffer berechnen:** Gesamtsumme genauso berechnen wie zuvor, aber ohne die letzte Ziffer. Anschließend die Differenz bis zum nächsten Vielfachen von 10 addieren.
- **Beispiel:** 3709451274518151 \Rightarrow ursprünglicher Code 370945127451815, Prüfziffer 1
 - Valide, denn $6 + 7 + 0 + 9 + 8 + 5 + 2 + 2 + 5 + 4 + 1 + 1 + 7 + 1 + 1 + 1 \equiv 0 \pmod{10}$
- Erkennt alle Einzelfehler und viele Nachbarvertauschungen, aber nicht alle
- Grenzen: keine Fehlerkorrektur; kein Schutz gegen absichtliche Manipulation

3	7	0	9	4	5	1	2	7	4	5	1	8	1	5	1
$\times 2$	$\times 1$	$\times 2$	$\times 1$	$\times 2$	$\times 1$	$\times 2$	$\times 1$	$\times 2$	$\times 1$	$\times 2$	$\times 1$	$\times 2$	$\times 1$	$\times 2$	$\times 1$
6	7	0	9	8	5	2	2	5	4	1	1	7	1	1	1

ISBN-10

- Einfache Mod-11-Prüfziffer zur Fehlererkennung
 - Letzte Stelle ist Prüfziffer: 0–9 oder X (=10)
 - Verwendet bei älteren ISBNs (heute meist ISBN-13)
- **Validierung (Regel):**
 - Ziffern mit Gewichten 10, 9, ..., 1 multiplizieren und summieren
 - X als 10 werten
 - Falls Gesamtsumme $\equiv 0 \pmod{11}$: “valide”, sonst “nicht valide”
- **Prüfziffer berechnen:** Gesamtsumme genauso berechnen wie zuvor, aber ohne die letzte Ziffer. Anschließend die Differenz bis zum nächsten Vielfachen von 11 addieren.
- **Beispiel:** 0306406152
 - Valide, denn $0 + 27 + 0 + 42 + 24 + 0 + 24 + 3 + 10 + 2 \equiv 0 \pmod{11}$
- Erkennt alle Einzelfehler und alle Nachbarvertauschungen
- Grenzen: keine Fehlerkorrektur; X als Sonderfall

0	3	0	6	4	0	6	1	5	2
×10	×9	×8	×7	×6	×5	×4	×3	×2	×1
0	27	0	42	24	0	24	3	10	2

ISBN-13

- Dezimal-Prüfziffer (Mod-10) nach EAN-13/GS1
 - Letzte Stelle ist Prüfziffer
 - Standard bei aktuellen ISBNs
- **Validierung (Regel):**
 - Von rechts Zahlen durchlaufen und addieren, dabei jede zweite Ziffer dreifach zählen
 - Falls Gesamtsumme $\equiv 0 \pmod{10}$: gib "valide" aus, sonst "nicht valide"
- **Prüfziffer berechnen:** Gesamtsumme genauso berechnen wie zuvor, aber ohne die letzte Ziffer. Anschließend die Differenz bis zum nächsten Vielfachen von 10 addieren.
- **Beispiel:** 9780306406157
 - Ursprünglicher Code 978030640615, Prüfziffer 7
 - Valide, denn $9 + 21 + 8 + 0 + 3 + 0 + 6 + 12 + 0 + 18 + 1 + 15 + 7 \equiv 0 \pmod{10}$
- Erkennt alle Einzelfehler und viele Nachbarvertauschungen (Ausnahmen bei Differenz 5)
- Grenzen: keine Fehlerkorrektur; kein Schutz gegen Manipulation

9	7	8	0	3	0	6	4	0	6	1	5	7
×1	×3	×1	×3	×1	×3	×1	×3	×1	×3	×1	×3	×1
9	21	8	0	3	0	6	12	0	18	1	15	7

Luhn vs. ISBN-10 vs. ISBN-13

■ Gemeinsamkeiten

- Prüfziffer am Ende, lineare Gewichtssumme
- Dient reiner Fehlererkennung, nicht -korrektur
- Schnell berechenbar, für manuelle Eingaben geeignet

■ Unterschiede

- **Luhn:** Mod-10, jede 2. Ziffer verdoppeln;
erkennt alle Einzelfehler, viele Nachbarvertauschungen
- **ISBN-10:** Mod-11, Gewichte 10, ..., 1, Prüfziffer 0–9 oder X;
erkennt alle Einzelfehler *und* alle Nachbarvertauschungen
- **ISBN-13:** Mod-10, Gewichte 1 und 3 alternierend;
erkennt alle Einzelfehler, viele Nachbarvertauschungen
- Ist ISBN-13 nicht „schlechter“ als ISBN-10?
 - Kompatibilität mit EAN-13/GS1-Barcodes (gleiche Mod-10/1-3-Regel, 13 Stellen)
 - Rein numerische Prüfziffer erforderlich — Mod-11 bräuchte teils X
 - Einheitliche Infrastruktur (Scanner, Datenbanken) wichtiger als maximale Fehlererkennungsleistung

Codierungs- und Informationstheorie – Übersicht

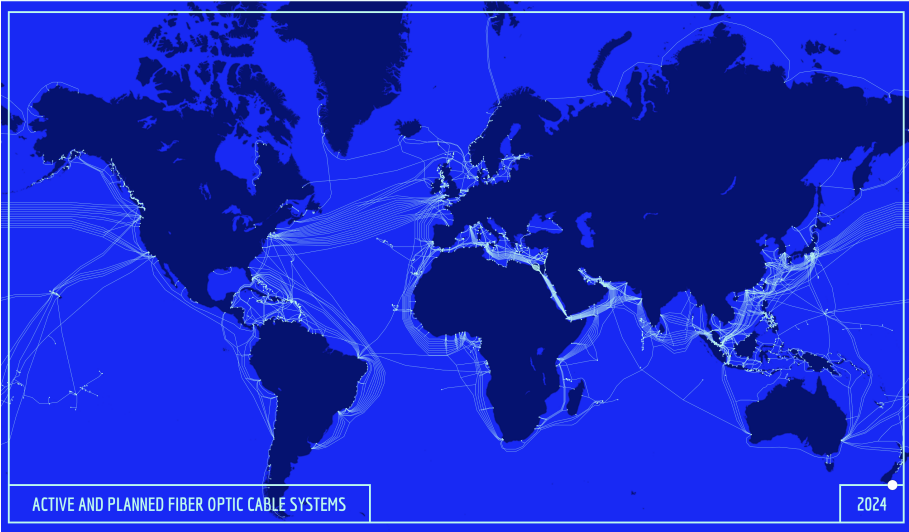
- 1 Zeichencodierungen
- 2 Luhn-Algorithmus, ISBN-10 und ISBN-13
- 3 Codierungstheorie**
- 4 Fehlererkennende und fehlerkorrigierende Codes
 - Hamming-Distanz
 - Fehlerkorrigierende Codes
 - Hamming-Code
- 5 Arten von Codes
- 6 Informationstheorie und Kompression
 - Informationstheorie nach Shannon
 - Huffman-Code

Wie werden (Internet-)Daten weltweit
übertragen?



Datenübertragung weltweit

The Verge

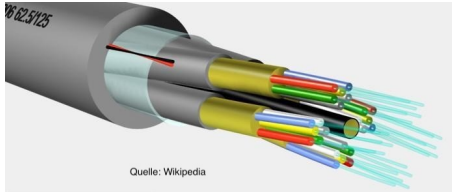


Credit: [TeleGeography](#)

<https://www.theverge.com/c/24070570/internet-cables-undersea-deep-repair-ships>

Datenübertragung weltweit

- Weltweit sind Kabel zur Datenübertragung verlegt
 - Zwischen verschiedenen Kontinenten, Städten, etc.
 - Aber auch in Datacenters, zwischen einzelnen Rechnern, etc.
- Je nach Verwendungszweck verschiedene Kabeltypen im Einsatz
- Übertragung ist analog und unterliegt Gesetzen der Physik
 - ⇒ Kann teilweise zu Fehlern in Übertragung kommen
 - ⇒ Müssen in der Lage sein, diese Fehler zu erkennen und zu korrigieren
 - ⇒ Wollen vorhandene Ressourcen möglichst effizient nutzen

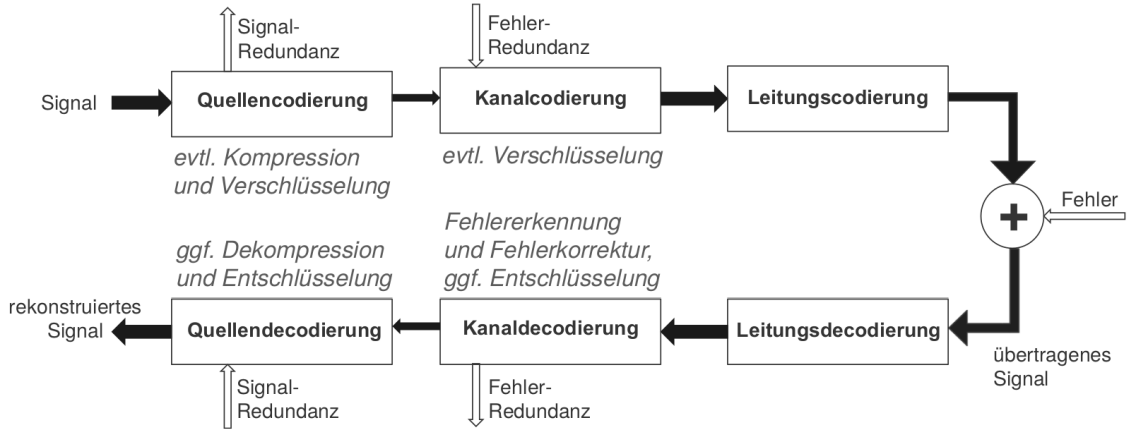


- **Codierungstheorie** ist die Theorie der fehlererkennenden und -korrigierenden Codes
- **Ziel:** Schutz vor Fehlern bei Übertragung oder Speicherung
- Verwendet Ideen aus Algebra, Kombinatorik, Zahlentheorie und Geometrie
- Begründer: Golay und Hamming, ca. 1950

- Je nach Ziel unterschiedliche Datenmodifikationen (Codierungen)
 - Datenkompression: Reduktion der Datenmenge
 - Codierungstheorie: Absicherung gegen Übertragungsfehler (durch erhöhte Redundanz)
 - Kryptographie: Absicherung gegen ungewollte Empfänger und Sender
- Oft werden auch mehrere Codierungen kombiniert
 - Erst Kompression, dann Verschlüsselung, dann Absicherung gegen Übertragungsfehler
- Kompression hilfreich für statistische Gleichverteilung der Zeichen
 - ⇒ Bessere Absicherung gegen Übertragungsfehler

- „Eine Informationseinheit ist dann redundant, wenn sie ohne Informationsverlust weggelassen werden kann“ [Wikipedia]
- „Redundant ist der Teil einer Nachricht, der keine Information enthält“ [Wikipedia]
- Gezielter Einsatz zur Fehlererkennung und -korrektur
- Steigerung der Qualität (weniger Fehler)
auf Kosten der Quantität (niedrigere Nutzdatenrate)
- Stärke der Redundanz: Anwendungsabhängig
 - Sicherheitskritische Systeme (viel)
 - Telefonie (wenig)

Datenübertragung mit mehrstufiger Codierung

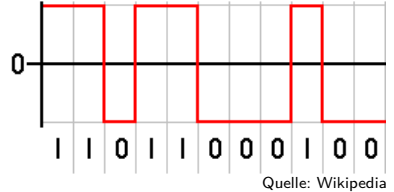


Quelle: nach Wikipedia

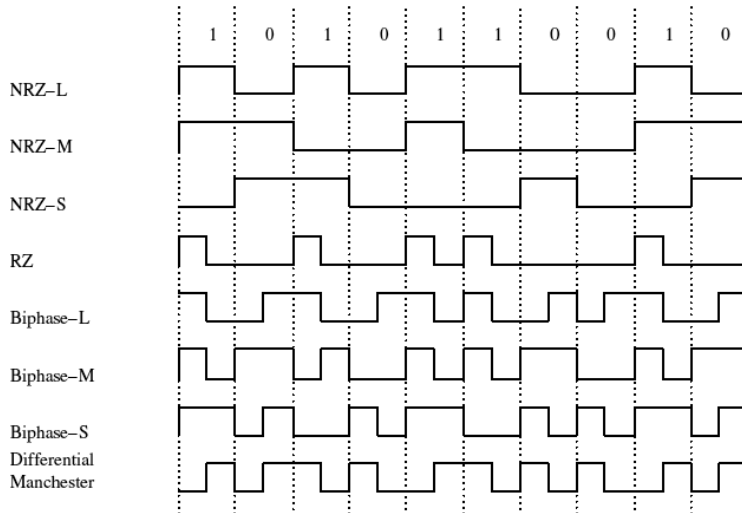
- Quellencodierung
 - Originaldaten werden binär codiert
 - Kompression üblich, Verschlüsselung möglich
- Kanalcodierung
 - Umcodierung der Binärdaten (evtl. komprimiert und verschlüsselt), d.h. Einfügen zusätzlicher Bits, um Übertragung sicherer zu gestalten (Hinzufügen von Redundanz)
 - Fehlerkorrigierende Codes üblich, Verschlüsselung möglich
- Leitungscodierung
 - Nochmaliges Umcodieren der Daten, um sie auf Übertragungsmedium zu optimieren
 - „Dabei werden bestimmte Pegelfolgen, etwa Lichtintensitäten auf Glasfasern oder Spannungen oder Ströme auf elektrischen Leitungen, binären Bitsequenzen im Datenstrom zugeordnet“ [Wikipedia: Leitungscodierung]

Leitungscode NRZ

- Non Return to Zero (NRZ):
 - Zwei Pegelzustände (high / low)
 - Jeder Leitungszustand trägt Information
- Zuordnung je eines Leitungspegels für die logischen Zustände 0 und 1
 - Ein Leitungspegel kann auch 0 Volt sein
- Separates Taktsignal nötig
- Verwendung, wenn in Nutzdaten keine langen konstanten Folgen
 - Beispiel: ASCII-codierte Texte
 - Grenze für „lang“ vom Medium abhängig
- Bei magnetischer Datenaufzeichnung problematisch



Leitungscode – Varianten von NRZ



Quelle: Wikibooks

- NRZ-L codiert 1 mit „high“, 0 mit „low“
- NRZ-M bewirkt einen Pegelwechsel bei 1
- NRZ-S bewirkt einen Pegelwechsel bei 0
- RZ: unipolarer Return-to-Zero-Code zweite Takthälfte immer „low“
- Biphase: für jedes Bit zwei Zustände
- Differential Manchester: mind. eine Flanke pro Bit, Takt „mitcodiert“, Wechsel zu Beginn codiert 0

Codierungs- und Informationstheorie – Übersicht

- 1 Zeichencodierungen
- 2 Luhn-Algorithmus, ISBN-10 und ISBN-13
- 3 Codierungstheorie
- 4 Fehlererkennende und fehlerkorrigierende Codes
 - Hamming-Distanz
 - Fehlerkorrigierende Codes
 - Hamming-Code
- 5 Arten von Codes
- 6 Informationstheorie und Kompression
 - Informationstheorie nach Shannon
 - Huffman-Code

Fehlererkennende und fehlerkorrigierende Codes

- Bei der Übermittlung von Information treten unweigerlich Störungen auf, z.B. bei Telefon
- Problem bei redundanzarmen Codierungen
 - Bei „Umfallen“ eines Bits von 0 auf 1 oder von 1 auf 0
 - ▶ Inhalt der übertragenen Information zerstört oder geändert

Zwei Arten von Übertragungsfehlern:

- Einzelbit-Fehler
 - Einzelne, unabhängige Bits gestört
- Fehlerbündel (engl. burst)
 - Mehrere aufeinanderfolgende Bits gestört

Gestörte Übertragung

- Gesucht: Maß für Störungsanfälligkeit
„Wie sicher ist ein Code gegenüber Störungen?“
- Beispiel: Alphabet $\{a, e, i, o\}$ mit folgender Codierung

a 00

e 01

i 10

o 11

- Ein einziges Bit gestört \rightarrow falsche Nachricht wird empfangen
 - Zu übertragende Nachricht „a“ (00)
 - Falls erstes Bit gestört (10 statt 00) \rightarrow „i“ empfangen
 - Falls zweites Bit gestört (01 statt 00) \rightarrow „e“ empfangen

Gestörte Übertragung

- Wählt man folgende Codierung, ist es durch Ändern eines einzigen Bits nicht mehr möglich, ein anderes gültiges Codewort zu erhalten

a 000

e 011

i 101

o 110

- Zu übertragende Nachricht: „a“ (000)
 - Falls erstes Bit gestört (100 statt 000) → ungültiges Codewort
 - Falls zweites Bit gestört (010 statt 000) → ungültiges Codewort
 - Falls drittes Bit gestört (001 statt 000) → ungültiges Codewort
- Fehlererkennung: Ja
 - Ungültiges Codewort → Übertragungsfehler aufgetreten
- Fehlerkorrektur: Nein
 - Beispiel: 010 empfangen → sollte „a“ oder „e“ oder „o“ übertragen werden?

⇒ Wie kann man diese Prozesse formal verstehen?

Hamming-Distanz

- Für zwei Codewörter $x_1, x_2 \in \{0, 1\}^n$ ist die **Hamming-Distanz** gegeben durch

$$D(x_1, x_2) = \sum_{i=1}^n |x_1(i) - x_2(i)|$$

⇒ Wir zählen, an wie vielen Stellen sich die Codewörter unterscheiden

- Kann mit XOR-Operation implementiert werden

- Beispiel:

a 00

e 01

i 10

o 11

- Die Codewörter 00 (für a) und 01 (für e) unterscheiden sich an einer Stelle
⇒ Hamming-Distanz $D(00, 01) = 1$

Hamming-Distanz

- Abstandsbestimmung von einem ganzen Code:
 - Matrix für jedes Paar von Codewörter (siehe nächster Slide)
 - Minimaler Abstand zweier Codewörter legt Hamming-Abstand des Codes fest
 - Formal: Für einen Code \mathcal{C} mit Codewörtern $\mathcal{C} = \{x_1, \dots, x_m\}$ ist der **Hamming-Abstand D des Codes \mathcal{C}** gegeben durch

$$D = \min_{i \neq j} D(x_i, x_j)$$

- Beispiel:

a 0000 0000

e 0000 1111

i 1111 0000

o 1111 1000

⇒ Hamming-Distanz vom gesamten Code ist $D = 1$, da Codes für i und o Hamming-Distanz 1 haben

Hamming-Distanz

- Erster Fall: Code mit $D = 1$

		a	e	i	o
a	00				
e	01	a	-	1	1
i	10	e	1	-	2
o	11	i	1	2	-
		o	2	1	1

- Zweiter Fall: Code mit $D = 2$

- Da die Matrix symmetrisch ist, können wir die untere linke Hälfte weglassen

		a	e	i	o
a	000				
e	011	a	-	2	2
i	101	e		-	2
o	110	i			-
		o			

Parity-Bit (Paritätsbit)

- Erhöhung der Hamming-Distanz eines Codes von 1 auf 2 mittels Paritätsbits
 - Parität: „[bildungssprachlich] Gleichsetzung, -stellung, [zahlenmäßige] Gleichheit“
- Dient der Erkennung fehlerhaft übertragener Informationswörter
- Analog zu Prüfziffer in EAN-Code
- Erfolgt durch Anhängen eines sogenannten Parity-Bits
 - Jedem Codewort wird ein Bit hinzugefügt
 - Even parity: Parity-Bit wird so gesetzt, dass Anzahl der 1en (inkl. Paritätsbit) in einem Codewort gerade ist
 - Odd parity: Parity-Bit wird so gesetzt, dass Anzahl der 1en (inkl. Paritätsbit) in einem Codewort ungerade ist
 - Beispiel: Codewort 1010 0110. Even parity: 1010 0110 0. Odd parity: 1010 0110 1.
 - Beispiel: 0110 0011 1 mit even parity Bit erhalten. \Rightarrow Es muss einen Fehler gegeben haben.
- Beispiel von vorherigem Slide:
 - Beim zweiten Fall ($D = 2$) haben wir den Code vom ersten Fall genommen und Parity-Bits für even parity eingefügt
- Kann Fehler nur erkennen, nicht korrigieren
- Mehrere Paritätsbits nach obigem Prinzip?
 - Funktioniert nicht (es werden nur 0en angehängt), braucht weitere Ideen

Prüfstellen-Codes (systematische Codes)

- Ein Paritätsbit
- Mehrere Paritätsbits
 - Je ein Paritätsbit für unterschiedliche Teile des Informationsworts
Beispiel: **Hamming-Code**
 - Mehrdimensionale Paritätsverfahren
Beispiel: Kreuz- oder Blockparität
- Prüfsumme
 - Beispiel: EAN (nicht binär!)
 - **Polynomcodes**
 - Zyklische Codes, z.B. **CRC-Codes**













Fehlerkorrigierende Codes

- Bei Codes mit Hamming-Abstand D gilt:
 - Bis zu $D - 1$ Bit-Fehler sind erkennbar
 - Bei k Bit-Fehlern mit $k < \frac{D}{2}$ können wir Fehler sogar korrigieren
- Beispiel: Code mit den vier Codewörtern
00000 00000, 00000 11111, 11111 00000 und 11111 11111
 - Hamming-Distanz $D = 5$
 - ⇒ Maximal zwei Fehler können korrigiert werden
 - Empfangene Nachricht: 00000 00111
 - Angenommen wir wissen, dass nur zwei Fehler aufgetreten sind
 - ⇒ Original 00000 11111
 - Wenn drei oder vier Fehler aufgetreten sind, etwa 00000 00011 oder 00000 00001
 - ⇒ Original 00000 00000 oder 00000 11111?
 - ⇒ Fehler kann nicht korrigiert werden, aber er wird erkannt

Hamming-Code

- Populäre Bildungsvorschrift, um Codes zu erhalten: [Hamming-Code](#)
 - Von Richard Hamming entwickelt
 - Linearer fehlerkorrigierender Blockcode
 - ⇒ Allgemeines Verfahren, um binäre Strings ohne Fehlerkorrektur auf (längere) Codewörter mit Fehlerkorrektur abzubilden
- Verwendung mehrerer Paritätsbits
 - Die Paritätsbits fügen Redundanz ein, die zur Fehlererkennung genutzt wird
 - Die Paritätsbits werden anhand der Eingabebits gesetzt (analog zur Prüfziffer beim EAN-Code)
- Der Hamming-Code, den wir konstruieren, hat Hamming-Abstand $D = 3$
 - Kann bis zu zwei Bitfehler in einem Codewort erkennen
 - Kann einen Bitfehler korrigieren


















Hamming-Code

																																																																																																																																																																																																																																														
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Quelle: Wikipedia

- Die Bits des Codewortes werden mit 1 beginnend von links nach rechts durchnummeriert:
 c_1, c_2, c_3, \dots
- Jene Bits, die Potenzen von 2 sind, also 1, 2, 4, 8, 16, usw. sind Prüfbits:
 $p_1, p_2, p_3, p_4, p_5, \dots$
- Die restlichen Bits ($c_3, c_5, c_6, c_7, c_9, \dots$) sind mit den informationstragenden Datenbits gefüllt: $d_1, d_2, d_3, d_4, d_5, \dots$
- Jedes Prüfbit ist ein Parity-Bit für eine bestimmte Menge von Bits
- Ein Bit kann in die Berechnung verschiedener Prüfbits involviert sein

Hamming-Code

																																
2^0	2^1			2^2				2^3							2^4																	
C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}	C_{20}	C_{21}	C_{22}	C_{23}	C_{24}	C_{25}	C_{26}	C_{27}	C_{28}	C_{29}	C_{30}	C_{31}		
p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	p_5	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}	d_{17}	d_{18}	d_{19}	d_{20}	d_{21}	d_{22}	d_{23}	d_{24}	d_{25}	d_{26}		

Quelle: Wikipedia

- Feststellen, zu welchen Prüfbits das Codebit mit Index k beiträgt:
 - k als Summe von Zweierpotenzen darstellen
- Ein Codebit liefert zu all jenen Prüfbits einen Beitrag, deren Codebit-Index eine Zweierpotenz in dieser Darstellung ist
 - Beispiel: $11 = 8 + 2 + 1$
 - ⇒ Codebit 11 liefert Beitrag zu Prüfbits an den Stellen 1, 2 und 8
 - Beispiel: $29 = 16 + 8 + 4 + 1$
 - ⇒ Codebit 29 liefert Beitrag zu Prüfbits an den Stellen 1, 4, 8 und 16

Hamming-Code – Prüfung

- Wenn ein bestimmtes Wort empfangen wird, setzt man einen Korrekturindikator auf 0
- Dann kontrolliert man jedes Prüfbit c_k ($k = 1, 2, 4, 8, \dots$), um zu sehen, ob der im empfangenen Codewort stehende Wert mit dem neu berechneten Prüfwert übereinstimmt
- Für alle $k = 1, 2, 4, 8, \dots$, an denen der neu berechnete Prüfwert von c_k abweicht: Addiere k zum Korrekturindikator
- Ist der Korrekturindikator nach Bearbeitung aller Prüfbits gleich 0, so akzeptiert man das Wort als korrektes Codewort
- Andernfalls enthält der Indikator die Nummer des gestörten Bits, das somit leicht korrigiert (= invertiert) werden kann
- Beispiel: Wenn die Prüfbits c_1 , c_2 und c_8 nicht richtig sind, muss Bit 11 invertiert werden, da $1 + 2 + 8 = 11$

Hamming-Code – Beispiel

- Hamming-Code mit vier Datenbits und drei Prüfbits
- Datenbits c_3, c_5, c_6, c_7 und Prüfbits p_1, p_2, p_3

2^0	2^1		2^2			
c_1	c_2	c_3	c_4	c_5	c_6	c_7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

Quelle: Wikipedia

$$c_1 := p_1 = (c_3 + c_5 + c_7) \bmod 2 = c_3 \oplus c_5 \oplus c_7$$

$$c_2 := p_2 = (c_3 + c_6 + c_7) \bmod 2 = c_3 \oplus c_6 \oplus c_7$$

$$c_4 := p_3 = (c_5 + c_6 + c_7) \bmod 2 = c_5 \oplus c_6 \oplus c_7$$

$$0 \oplus 0 = 0 \qquad 1 \oplus 1 = 0 \qquad \oplus \dots \text{ XOR}$$

$$0 \oplus 1 = 1 \qquad 0 \oplus 1 = 1$$

Hamming-Code – Beispiel

- Datenwort: 0110

2^0	2^1		2^2			
c_1	c_2	c_3	c_4	c_5	c_6	c_7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

Quelle: Wikipedia

- Berechne Prüfbits:

$$c_1 := p_1 = (c_3 + c_5 + c_7) \bmod 2 = c_3 \oplus c_5 \oplus c_7 = 0 \oplus 1 \oplus 0 = 1$$

$$c_2 := p_2 = (c_3 + c_6 + c_7) \bmod 2 = c_3 \oplus c_6 \oplus c_7 = 0 \oplus 1 \oplus 0 = 1$$

$$c_4 := p_3 = (c_5 + c_6 + c_7) \bmod 2 = c_5 \oplus c_6 \oplus c_7 = 1 \oplus 1 \oplus 0 = 0$$

Zu übertragendes Wort: 1100110

- Angenommen das Bit c_5 wird bei der Übertragung gestört und daher das Wort 1100010 empfangen

Hamming-Code – Beispiel

2^0	2^1		2^2			
c_1	c_2	c_3	c_4	c_5	c_6	c_7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

Quelle: Wikipedia

- Für 1100010 ergeben die Berechnungen:

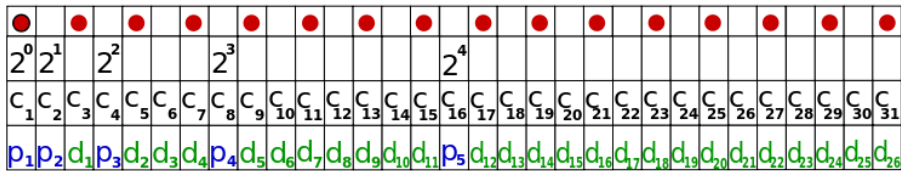
$$p_1 = (c_3 + c_5 + c_7) \bmod 2 = c_3 \oplus c_5 \oplus c_7 = 0 \oplus 0 \oplus 0 = 0 \neq c_1$$

$$p_2 = (c_3 + c_6 + c_7) \bmod 2 = c_3 \oplus c_6 \oplus c_7 = 0 \oplus 1 \oplus 0 = 1 = c_2$$

$$p_3 = (c_5 + c_6 + c_7) \bmod 2 = c_5 \oplus c_6 \oplus c_7 = 0 \oplus 1 \oplus 0 = 1 \neq c_4$$

- Da die Prüfbits p_1 und p_3 einen anderen Wert als die übertragenen Codebits c_1 und c_4 ergeben, besitzt der Korrekturindikator den Wert $5 = 1 + 4$.
Das gestörte Bit ist somit c_5 (unter der Annahme, dass nur ein Bit gestört wurde)

Hamming-Code – Gleichung für Prüfbits



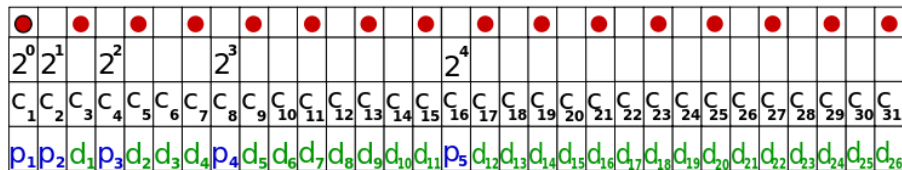
Quelle: Wikipedia

- Das Prüfbit p_i wird über alle Stellen c_j des Codeworts berechnet, in denen an der i -ten Stelle der Binärcodierung des Index j eine logische 1 steht (außer diese Stelle ist ein Prüfbit)
- Beispiel: Datenwort $d_1d_2d_3d_4d_5d_6d_7d_8$
 - 8 Datenbits \Rightarrow 4 Prüfbits, also 12 Bit insgesamt
 - $p_1 \dots$ alle „ungeraden“ Bits
 - $\Rightarrow p_1 = (c_3 + c_5 + \dots + c_{11}) \bmod 2$
 - $p_2 \dots$ jene c_j , wo in der Spalte 2 eine 1 gesetzt ist, außer c_2 selbst
 - $\Rightarrow p_2 = (c_3 + c_6 + c_7 + c_{10} + c_{11}) \bmod 2$
- Prüfbits treten nie rechts in der Gleichung auf

Blockbildung

		Prüfbitindex			
		4	3	2	1
Codebitindex	0	0	0	0	0
	1	0	0	0	1
	2	0	0	1	0
	3	0	0	1	1
	4	0	1	0	0
	5	0	1	0	1
	6	0	1	1	0
	7	0	1	1	1
	8	1	0	0	0
	9	1	0	0	1
	10	1	0	1	0
	11	1	0	1	1
	12	1	1	0	0
	13	1	1	0	1
	14	1	1	1	0
15	1	1	1	1	

Hamming-Code – Gleichungen für Prüfbits



Quelle: Wikipedia

Wie lauten die Prüfgleichungen für einen Hamming-Code mit k Datenbits?

- k Datenbits (grün), $\lceil \lg(n+1) \rceil$ Prüfbits (blau)
- $n = k + \lceil \lg(n+1) \rceil$ Codebits (schwarz)

$$p_1 = c_3 + c_5 + \dots + c_{n-1+(n \bmod 2)}$$

$$p_2 = c_3 + c_6 + c_7 + c_{10} + c_{11} + \dots$$

$$p_3 = c_5 + c_6 + c_7 + c_{12} + c_{13} + c_{14} + c_{15} + \dots$$

$$p_4 = c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} + c_{15} + \dots$$

$$p_5 = \dots$$

Blockgröße

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

Blockbildung

		Prüfbitindex			
		4	3	2	1
Codebitindex	0	0	0	0	0
	1	0	0	0	1
	2	0	0	1	0
	3	0	0	1	1
	4	0	1	0	0
	5	0	1	0	1
	6	0	1	1	0
	7	0	1	1	1
	8	1	0	0	0
	9	1	0	0	1
	10	1	0	1	0
	11	1	0	1	1
	12	1	1	0	0
	13	1	1	0	1
	14	1	1	1	0
	15	1	1	1	1

- Haben bereits gesehen: Das Datenwort 0110 hat den Hamming-Code 1100110
- Analog erhält man: Das Datenwort 1100 hat den Hamming-Code 0111100
- Welchen Code hat das Datenwort 1010?
 - Wenn man selbst rechnet: Code ist 1011010
 - 1010 ergibt sich aus 0110 und 1100 bzgl. Modulo2-Addition
 - Andere Art es zu erhalten: Die Modulo2-Addition von 1100110 und 0111100 ist 1011010
- Dies ist die Eigenschaft einer **linearen Codierung**
 - **Definition „lineare Codierung“:** Für alle Datenwörter m_1 und m_2 mit Codes c_1 und c_2 gilt: Das Datenwort $m_1 + m_2$ besitzt das Codewort $c_1 + c_2$ (+ ... bitweise Modulo2-Addition)

Hamming-Code – Mehrfachfehler

- Auch beim Auftreten von mehreren gestörten Stellen kann der Indikator den Wert 0 haben oder aber einen Wert, der größer ist als die Länge der Codewörter
- Falls der letztere Fall eintritt, kann man mit Sicherheit auf das Vorhandensein mehrerer gestörter Stellen schließen
- Der hier vorgestellte Hamming-Code kann nur einfache Fehler korrigieren
- Für Fehlerbündel kann man eine Folge von k aufeinanderfolgenden Codewörtern in Form einer Matrix anordnen, und zwar ein Codewort pro Zeile
- Prüfsummen werden dann für Zeilen und Spalten gebildet, gesendet wird zeilenweise

Hamming-Code – Einschränkungen

- Es gibt **keinen** Hamming-Code mit $n = 2^\ell$ Codebits ($\ell \in \mathbb{N}$)
 - In diesem Fall wäre $c_n = p_\ell$
 - Es gäbe für p_ℓ keine gültige Prüfgleichung, weil die Codebits, auf die sich p_ℓ beziehen würde, einen Index $> n$ hätten
 - Ist das ein Problem?
 - Nein, denn für k Datenbits können wir trotzdem immer einen Hamming-Code entwerfen

Fehlerkorrigierende Codes – Anzahl der Prüfbits

- Für Datenwörter mit m Bits verwendet der Hamming-Code $\lceil \lg(n) + 1 \rceil$ Prüfbits.
Können wir das verbessern?
- Angenommen wir haben n Codebits, davon m Datenbits, r Prüfbits, $n = m + r$
- Gewünscht: Korrektur von einem fehlerhaften Bit
- Wie viele Prüfbits r braucht man mindestens?

$$m + r = n \text{ Codebits}$$

$$m \text{ Datenbits} \quad r \text{ Prüfbits}$$

- Anzahl der möglichen Bitmuster in den Prüfbits: 2^r
- Um einen Fehler an allen n Stellen korrigieren zu können, müssen in den r Prüfbits mindestens folgende Bitmuster codiert sein:
 - 1 Bitmuster für fehlerfrei
 - Je 1 Bitmuster für jede mögliche Fehlerposition, d.h. n weitere Bitmuster
- ▶ Also muss r erfüllen:

$$2^r \geq n + 1 = m + r + 1$$

⇒ Es muss gelten $r \geq \lceil \lg(n + 1) \rceil$

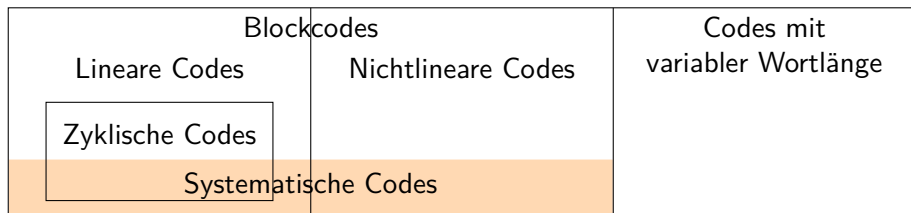
⇒ Hamming-Code ist optimal

- Die obige Abschätzung („untere Schranke“) gilt für jeglichen Code

Codierungs- und Informationstheorie – Übersicht

- 1 Zeichencodierungen
- 2 Luhn-Algorithmus, ISBN-10 und ISBN-13
- 3 Codierungstheorie
- 4 Fehlererkennende und fehlerkorrigierende Codes
 - Hamming-Distanz
 - Fehlerkorrigierende Codes
 - Hamming-Code
- 5 Arten von Codes
- 6 Informationstheorie und Kompression
 - Informationstheorie nach Shannon
 - Huffman-Code

Arten von Binärcodes



- **Blockcodes:** Konstante Codewortlänge
 - **Lineare Codes:** Summe von Codewörtern ist auch gültiges Codewort (Modulo-2-Arithmetik!)
 - **Zyklische Codes:** Durch bitweises Rotieren (Shiften) eines Codeworts entsteht wieder ein gültiges Codewort
 - **Systematische Codes:** Codewörter der Länge n bestehen aus m Informationsbits und $r = (n - m)$ Prüfbits (Prüfstellen)
- **Codes mit variabler Wortlänge:** Wörter eines Codes sind unterschiedlich lang

- Spezielle Art von Blockcode
- Linearkombination von zwei Codewörtern ist ebenfalls ein Codewort
 - Da wir modulo 2 rechnen, ist Linearkombination gleichbedeutend mit Summe
 - **Definition „Linearcode“:** Für alle Codewörter c_1 und c_2 gilt:
 $c_1 + c_2$ ist ebenfalls ein Codewort (+ ... bitweise Modulo2-Addition)
- Vorteil
 - Verwendung von Methoden der Linearen Algebra
 - ▶ Einfach zu codieren und decodieren
- Viele wichtige Codes sind linear
 - Hamming-Code
 - Alle zyklischen Codes

- Spezieller linearer Blockcode
- Wichtiger Kanalcode
- Anwendung bei
 - Übertragungskanälen
 - Datenspeichern

■ **Definition „zyklischer Code“:**

Ein zyklischer Code \mathcal{C} ist ein linearer Code, der zusätzlich folgende Eigenschaft besitzt:
Ist $(a_0 a_1 \cdots a_{n-1})$ ein Codewort in \mathcal{C} , dann ist auch $(a_1 \cdots a_{n-1} a_0)$ ein Codewort von \mathcal{C} .

⇒ Daraus folgt, dass auch

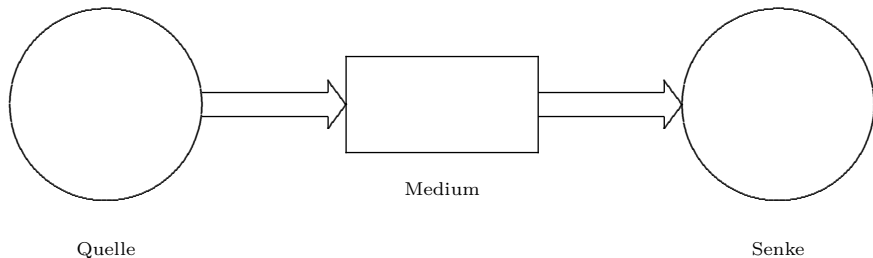
$(a_2 a_3 \cdots a_{n-1} a_0 a_1), (a_3 a_4 \cdots a_{n-1} a_0 a_1 a_2), \dots, (a_{n-1} a_0 a_1 \cdots a_{n-2})$
Codewörter von \mathcal{C} sind.

- Mehrfache Codierung
 - Quelle, Kanal, Leitung
 - Codierung, Verschlüsselung, Kompression
- Zugang zur Codierung:
 - Mathematische Grundlagen, um gewünschte Eigenschaften eines Codes zu gewährleisten
 - Technische Implementierung
 - Anpassung an das verwendete physikalische Medium:
 - Speichermedium: optisch, magnetisch, elektrisch
 - Übertragungsmedium: elektrische Leitung, Funk, Glasfaser

Codierungs- und Informationstheorie – Übersicht

- 1 Zeichencodierungen
- 2 Luhn-Algorithmus, ISBN-10 und ISBN-13
- 3 Codierungstheorie
- 4 Fehlererkennende und fehlerkorrigierende Codes
 - Hamming-Distanz
 - Fehlerkorrigierende Codes
 - Hamming-Code
- 5 Arten von Codes
- 6 Informationstheorie und Kompression
 - Informationstheorie nach Shannon
 - Huffman-Code

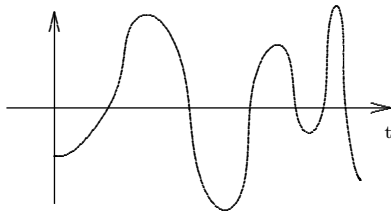
Das Modell der Informationsübertragung



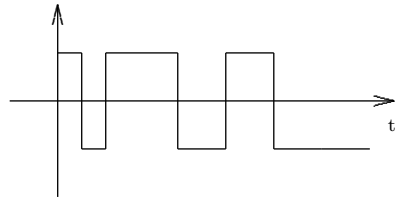
- Information soll von der Quelle (Sender) über ein Übertragungsmedium (Nachrichtenkanal) zu einer Senke (Empfänger) übermittelt werden
- Beispiel:
 - Sender und Empfänger sind Menschen
 - Der Sender spricht mit dem Empfänger, wobei als Übertragungsmedium Schallwellen fungieren

Was verstehen wir unter Information

- Information ist die Neuheit, welche durch Nachrichten übermittelt wird
- Nachrichten werden durch Signale übertragen
- Man unterscheidet zwischen analogen und diskreten Signalen
- Oftmals müssen analoge Signale zur digitalen Verarbeitung auf diskrete abgebildet werden



zeit- und wertkontinuierliches



wertdiskretes und zeitkontinuierliches

Signal

Definitionen – Sprache

- Die Regeln einer Sprache werden als *Grammatik* bezeichnet
- Die *Grammatik* beschreibt die *Syntax* der Sprache
- Man benötigt Kenntnis über die Regeln, nach denen eine Nachricht aufgebaut ist, bevor man die Information aus der Nachricht herauslesen kann
- Eine Nachricht ist aus Zeichen zusammengesetzt
- Die Menge aller unterschiedlichen Zeichen einer Sprache ist das *Alphabet*

Syntax vs. Semantik

- *Syntax* beschreibt, wie das Alphabet in Nachrichten verwendet wird
- *Semantik* beschreibt die Bedeutung von Nachrichten
- *Metasprache*: Eine Sprache wird mit einer anderen Sprache erklärt
 - Beispiele:
 - Legende von Karte
 - Beschreibung, wie Nachrichten aufgebaut sind

Beispiele für künstliche Sprachen

- Mathematische Zeichensprache:

$(1 + 1) = 2$ syntaktisch und semantisch richtig

$(1 + 3) < 2$ syntaktisch richtig, aber semantisch falsch

$1 + 2) < 5$ syntaktisch falsch

- Notenschrift



- Chemische Formelsprache (z.B. $C_6H_{12}O_6$)

- Programmiersprachen:

`if (a) then { b; } else { c; }`

`ITE(a, b, c)` andere Syntax, gleiche Semantik

- *Binäralphabet*: Besteht aus nur zwei Zeichen, z.B. $\{0, 1\}$ oder $\{\bullet, -\}$

Anmerkung: „“, „{“ und „}“ sind Metasprache

Definitionen – Code

- Einfachere Übertragungsmöglichkeiten, einfachere Verarbeitung oder Erhöhung der Störsicherheit bei der Übertragung
- *Wort*: Zeichenfolge aus dem Vorrat eines Alphabets
- *Code*: Abbildung von Wörter zwischen Sprachen
- *Codewort*: Wort in der Zielsprache, das durch den Code „getroffen“ wird
- Abbildung muss umkehrbar eindeutig sein
 - ⇒ Zu jedem Wort gibt es ein eindeutiges Codewort und umgekehrt
- *Feste oder variable Wortlänge*
 - Variable Wortlänge: Wörter sind unterschiedlich lang
 - Beispiel: Zeichen, die öfter vorkommen, werden auf kürzere Wörter abgebildet

Mehrdeutigkeit des Begriffs „Code“

- Kodierungstheorie, algebraische Sichtweise: ein Code ist die Menge der Codewörter
 - Ein Code (= die Menge der Codewörter) kann linear, zyklisch, ... sein.
 - Codierung(sfunktion): Abbildung von Daten- auf Codewörter
- Informationstheorie: ein Code besteht aus
 - einer Menge von Datenwörtern
 - einer Menge von Codewörtern
 - einer Codierungsfunktion von den Daten- auf die Codewörter
 - einer Decodierungsfunktion von den Code- auf die Datenwörter

Die zentrale Komponente ist die Codierungsfunktion, sodass „Code“ gelegentlich nur die Codierungsfunktion meint.

- Auch die Codierungsfunktion kann linear sein.
- Siehe Hammingcode: Sowohl der Code (im Sinne der Kodierungstheorie) als auch die Codierungsfunktion sind linear.
- „Code“ kann also je nach Kontext bedeuten:
 - Menge der Codewörter
 - Codierungsfunktion von den Daten- auf die Codewörter
 - das gesamte Codierungssystem inklusive Decodierungsfunktion

Informationsgehalt

- Der Informationsgehalt von einem Zeichen hängt von der Häufigkeit ab, mit der das Zeichen gesendet wird
- Häufig gesendete Zeichen \Rightarrow niedriger Informationsgehalt
 - Für ein Zeichen mit relativer Häufigkeit p sollte der Informationsgehalt also proportional sein zu $1/p$
 - **Relative Häufigkeit p_x** : Wenn ein Zeichen x in einem Wort mit n Zeichen n_x -mal vorkommt, dann ist $p_x = \frac{n_x}{n}$ die relative Häufigkeit von x im Wort.
 - Beispiel: Wort 0001 0100 0100
Die relative Häufigkeit von 0 und 1 im Wort ist $p_0 = \frac{3}{4}$ und $p_1 = \frac{1}{4}$
- Der Informationsgehalt einer aus mehreren (voneinander unabhängigen) Zeichen bestehenden Nachricht soll gleich der Summe der Informationsgehalte der einzelnen Zeichen sein
 - Für Zeichen x und y sollte gelten:

$$h(p_x) + h(p_y) = h(p_x \cdot p_y),$$

wobei $h(p_x)$ den Informationsgehalt von x bezeichnet

Informationsgehalt

■ Informationsgehalt h :

$$h = \text{ld}(1/p) = \text{ld}(p^{-1}) = -\text{ld } p$$

p ... relative Häufigkeit

ld ... *logarithmus dualis* (\log_2)

Basisumrechnung: $\log_b c = \frac{\log_a c}{\log_a b}$ für $a \neq 1$, $b \neq 1$, $c > 0$

■ Die Einheit des Informationsgehaltes wird **Bit** genannt

■ Warum?

■ Angenommen wir wollen 2^n verschiedene Zahlen darstellen

■ Angenommen alle Zahlen haben die gleiche relative Häufigkeit $p = \frac{1}{2^n}$

■ Informationsgehalt einer einzelnen Zahl ist $h = \text{ld}(1/2^n) = n$

⇒ Brauchen n Bit, um jede der Zahlen darzustellen

⇒ Wir hatten das gleiche Ergebnis bereits in Zahlendarstellung erhalten
(um Zahlen $0, 1, \dots, 2^n - 1$ darzustellen, benötigen wir n Bits)

■ Interpretation: Informationsgehalt gibt die minimale Anzahl an Bits an, die ein Codewort haben muss, um ein Zeichen mit relativer Häufigkeit p zu übertragen

■ Idealisiert, da h auch fraktionale Werte annehmen kann

Mittlerer Informationsgehalt (Entropie)

- Sei \mathcal{I} die Menge unserer Zeichen (das Alphabet)
 - Sei p_i die relative Häufigkeit und $h_i = \text{ld}(1/p_i)$ der Informationsgehalt von $i \in \mathcal{I}$
- Mittlerer Informationsgehalt H der Zeichen in \mathcal{I} :

$$H = \sum_{i \in \mathcal{I}} p_i \cdot h_i = \sum_{i \in \mathcal{I}} p_i \cdot \text{ld}(1/p_i) = - \sum_{i \in \mathcal{I}} p_i \cdot \text{ld } p_i$$

- Häufig auch als Entropie bezeichnet, Einheit Bit
 - Bildet den gewichteten Durchschnitt über den Informationsgehalt der einzelnen Zeichen
- ⇒ „Wenn ich ein zufälliges Zeichen erhalte, das anhand der relativen Häufigkeit ausgewählt wurde, wie viel Informationsgehalt sollte ich erwarten?“

Beispiel

- Alphabet mit a, e, i

	p_i	h_i
a	0.50	1
e	0.25	2
i	0.25	2

- $H = 0.5 \cdot 1 + 0.25 \cdot 2 + 0.25 \cdot 2 = 1.5$ Bit
- Möglicher Code für das obige Beispiel:

a	1
e	01
i	00

- Code ist umkehrbar eindeutig
 - Beispiel: 011100011 entspricht eaaiea
 - 11 ist kein Codewort

Mittlere Wortlänge und Redundanz

- Für einen gegebenen Code, sei ℓ_i Länge des i -ten Codewortes

- Mittlere Wortlänge L :

$$L = \sum_i p_i \cdot \ell_i \quad [\text{Bit}]$$

- Bildet den gewichteten Durchschnitt über die Wortlänge der Codewörter
- Vergleich zum mittleren Informationsgehalt:
 - Wortlänge ℓ_i statt Informationsgehalt h_i verwendet
 - Mittlerer Informationsgehalt \sim idealisierte Codierung, mittlere Wortlänge \sim konkrete Codierung

- Redundanz R :

$$R = L - H \quad [\text{Bit}] \text{ mit } R \geq 0$$

- Relative Redundanz r (dimensionslos):

$$r = R/L$$

Beispiel

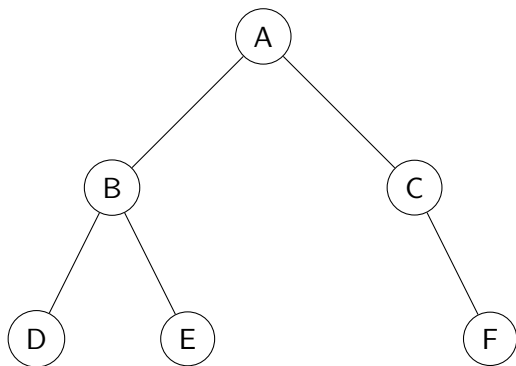
	p_i	h_i	$p_i \cdot h_i$	ℓ_i	$p_i \cdot \ell_i$
a	0.7	0.515	0.360	1	0.7
e	0.2	2.322	0.464	2	0.4
i	0.1	3.322	0.332	2	0.2

- $H = 0.7 \cdot 0.515 + 0.2 \cdot 2.322 + 0.1 \cdot 3.322 \text{ Bit} = 1.156 \text{ Bit}$
- $L = 0.7 \cdot 1 + 0.2 \cdot 2 + 0.1 \cdot 2 \text{ Bit} = 1.3 \text{ Bit}$
- $R = L - H = 1.3 - 1.156 = 0.144 \text{ Bit}$
- $r = R/L = 0.144/1.3 \approx 0.111 \text{ (dimensionslos)}$

- **Ziel:** Code mit möglichst wenig Redundanz
- Eine Möglichkeit zur Datenverdichtung: **Huffman-Code**
 - Verfahren zur verlustfreien Kompression
 - Erstellt einen sogenannten Codebaum basierend auf der Häufigkeit der einzelnen Zeichen
 - Wird weiterhin als Teil vom Zip-Dateiformat eingesetzt

Bäume (im Sinne der Graphentheorie)

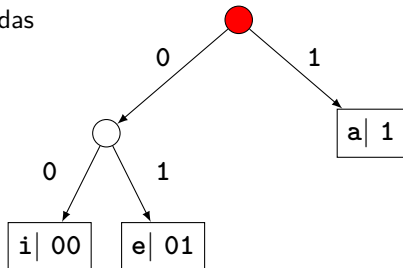
- Ein (ungerichteter, ungewichteter) **Graph G** ist gegeben durch ein Tupel (V, E)
 - V = die **Knoten** des Graphen
 - $E \subseteq V \times V$ ist die Menge der **Kanten**
 - Eine Kante ist ein Paar von Knoten
 - Im Beispiel:
 $V = \{A, B, C, D, E, F\}$
 $E = \{(A, B), (B, D), (B, E), (A, C), (C, F)\}$
- Wir nennen G einen **Baum**, wenn G zusammenhängend ist und keine Kreise enthält
 - „Zusammenhängend“: Jeder Knoten kann jeden anderen erreichen
 - „Kreis“: Eine Sequenz von Kanten, bei der man von einem Knoten ausgeht, über mehrere Kanten läuft und wieder zum selben Knoten zurückkehrt, ohne dabei eine Kante mehr als einmal zu benutzen



Codebäume

- Man kann Codes auch als Baum darstellen
 - Ein spezieller Knoten ist die **Wurzel** (im Bild rot gefüllt)
 - Jede Kante ist mit einem Bit 0/1 beschriftet
 - Die Knoten ohne ausgehenden Kanten heißen **Blätter**
- Man erhält Codewörter wie folgt:
 - Man beginnt bei der Wurzel
 - Am Anfang startet man mit einem leeren Codewort
 - Für jede Kante über die man läuft, hängt man das entsprechende Zeichen an das Codewort an
- Beispiel:

a 1
e 01
i 00



- Algorithmus zur Berechnung vom Huffman-Code:
 - Schrittweises Aufbauen eines Codebaums durch Zusammenfassen der beiden Zeichen mit der geringsten relativen Häufigkeit
 - Diese werden im weiteren Verlauf wie ein einzelnes Zeichen behandelt, dessen relative Häufigkeit gleich der Summe der beiden einzelnen relativen Häufigkeiten ist
 - Wird solange fortgesetzt, bis nur noch ein Zeichen übrig ist, und damit der Codebaum fertig erstellt ist

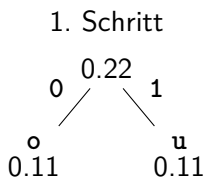
Beispiel

- Alphabet besteht aus Zeichen $\{\mathbf{a}, \mathbf{e}, \mathbf{i}, \mathbf{o}, \mathbf{u}\}$

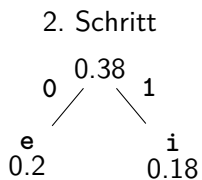
	p
a	0.40
e	0.20
i	0.18
o	0.11
u	0.11

Beispiel

	p
a	0.40
ou	0.22
e	0.20
i	0.18



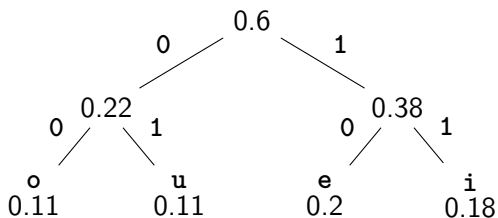
	p
a	0.40
ei	0.38
ou	0.22



Beispiel

3. Schritt

	p
a	0.40
ou	0.22
e	0.20
i	0.18



	p
a	0.40
ei	0.38
ou	0.22

	p
eiou	0.60

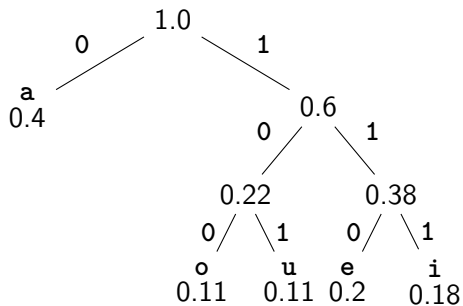
Beispiel

	p
a	0.40
ou	0.22
e	0.20
i	0.18

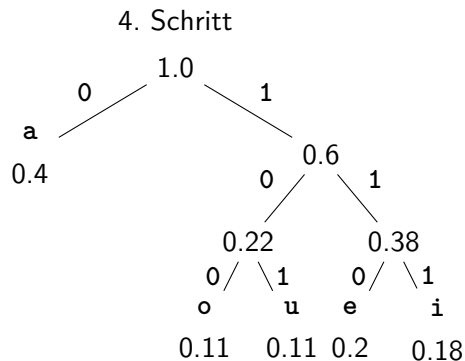
	p
a	0.40
ei	0.38
ou	0.22

	p
eiou	0.60
a	0.40

4. Schritt



Beispiel



	p	Codew.	l	$p \cdot l$
a	0.40	0	1	0.40
e	0.20	110	3	0.60
i	0.18	111	3	0.54
o	0.11	100	3	0.33
u	0.11	101	3	0.33

- Es ergibt sich eine mittlere Wortlänge von $L = 2.20$ Bit und eine Redundanz von $R = 0.06$ Bit

Huffman-Code

- Wenn man statt Einzelzeichen ganze Zeichenfolgen codiert, kann man die Redundanz beliebig klein machen
- Beispiel: Alphabet $\{\mathbf{a}, \mathbf{e}\}$

	p	Codewort
a	0.8	0
e	0.2	1

- Mittlerer Informationsgehalt eines Zeichens: $H = 0.722$ Bit
- Codiert man jedes Zeichen getrennt, so ergibt sich die Redundanz $R = 0.278$ Bit

- Codiert man jeweils zwei aufeinanderfolgende Zeichen durch ein einziges Codewort variabler Länge, so erhält man folgende Tabelle:

	p	Codewort	ℓ	$p \cdot \ell$
aa	0.64	0	1	0.64
ae	0.16	10	2	0.32
ea	0.16	110	3	0.48
ee	0.04	111	3	0.12

- Damit sinkt die Redundanz auf $R = 0.116$ Bit

- Die oben beschriebene Methode kann bei einer ungünstigen Verteilung der relativen Häufigkeiten, beim Zusammenfassen von Paaren von Einzelzeichen, eine größere Redundanz liefern als der zu den Einzelzeichen gehörige Code
 - Indem man jedoch das Verfahren sukzessive fortsetzt, d.h. drei, vier, fünf, usw. Einzelzeichen zusammenfasst, wird man irgendwann einen Code erhalten, dessen Redundanz kleiner ist als jene des aus den Einzelzeichen bestehenden Codes
 - Die relative Redundanz aber wird mit jedem der angewandten Schritte kleiner
 - Solche Verfahren haben natürlich einen (potentiell deutlich) erhöhten Rechenaufwand
- ⇒ In Praxis Tradeoffs zwischen bestmöglicher Kompression und Zeit für die Kompression

Ein adaptiver Huffman-Code

- Huffman-Code setzt voraus, dass sich (bei mehrfacher Anwendung) die relativen Häufigkeiten der einzelnen Zeichen nicht ändern
 - Das kann bei manchen zu übertragenden Bitfolgen dazu führen, dass die mittlere Länge der Codewörter länger ist als notwendig, weil diese Bitfolgen zufällig nur relativ unwahrscheinliche Zeichen repräsentieren
 - Außerdem möglich, dass sich Häufigkeitsverteilung der Zeichen mit der Zeit ändert
- Abhilfe: **Adaptive Codes**
 - Bei adaptiven Codes führen sowohl der Sender als auch der Empfänger für jedes Zeichen einen Zähler mit
 - Die relative Häufigkeit wird dann regelmäßig neu berechnet
 - Theoretisch ist es möglich, nach jedem übertragenen Zeichen den Code zu adaptieren (in der Praxis zu aufwändig)
 - Stattdessen einigt man sich darauf, nach jeweils N übertragenen Zeichen den Code zu adaptieren
 - Sender und Empfänger müssen denselben Wert für N verwenden