

# 04 – Codierungs- und Informationstheorie

Grundzüge digitaler Systeme

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 Linearer Barcode, EAN-13, andere Codierungen
- 3 Codierungstheorie
- 4 Fehlererkennende und fehlerkorrigierende Codes
  - Hamming-Distanz
  - Fehlerkorrigierende Codes
  - Hamming-Code
  - Polynomcodes
- 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

Quelle: [ascii-table.com](http://ascii-table.com)

- 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/>

- Unicode Transformation Format (UTF)
  - Verfahren zur Abbildung von Unicode-Zeichen auf Byte-Folgen
  - De-facto-Standard für Zeichencodierung im Internet (ca. 98%<sup>1</sup>)
- 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
- Windows und Java: UTF-16

---

<sup>1</sup>[https://w3techs.com/technologies/history\\_overview/character\\_encoding/ms/y](https://w3techs.com/technologies/history_overview/character_encoding/ms/y) (Stand: Okt. 2023)



# 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 <b>US-ASCII</b> -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 (ca. 1,4%<sup>1</sup>)
- 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 €

---

<sup>1</sup>[https://w3techs.com/technologies/history\\_overview/character\\_encoding/ms/y](https://w3techs.com/technologies/history_overview/character_encoding/ms/y) (Stand: Okt. 2023)

# 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	¡	¢	£	¤	¥	¦	§	¨	©	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 Linearer Barcode, EAN-13, andere Codierungen
- 3 Codierungstheorie
- 4 Fehlererkennende und fehlerkorrigierende Codes
  - Hamming-Distanz
  - Fehlerkorrigierende Codes
  - Hamming-Code
  - Polynomcodes
- 5 Arten von Codes
- 6 Informationstheorie und Kompression
  - Informationstheorie nach Shannon
  - Huffman-Code

# Andere gängige Codierungen

- Barcodes



- Hier nicht betrachtet:

- Audiosignale
- Video
- etc.

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



# Linearer Barcode

---

- Strichcode, Balkencode, Streifencode oder Barcode
  - Opto-elektronisch lesbare Schrift
  - Aus verschieden breiten, parallelen Strichen und Lücken
- Maschinell eingelesen und elektronisch weiterverarbeitet
  - Mit optischen Lesegeräten, wie z.B. Barcodelesegeräten (Scanner) oder Kameras
- Patent aus 1952, in Wien seit 1979 verwendet
- Viele Varianten, z.B. EAN-13 (Internationale Norm ISO/IEC 15420)
  
- EAN-13 codiert 12 Stellen plus eine Prüfziffer
  - Zeichenvorrat: 0, . . . , 9
  - Meist im Klartext darunter
  - Erste implizite Ziffer links daneben
- ISBN und ISSN werden im EAN-13-Code codiert



# Darstellung von EAN-13



Ziffer	0	1	2	3	4	5	6	7	8	9
Code A										
Code B										
Code C										

Randzeichen

Trennzeichen

Quelle: [www.tinohempel.de](http://www.tinohempel.de)

- Einzelne Ziffer codiert mit je:
  - Zwei dunklen und zwei hellen Streifen
  - Unterschiedlicher Breite
- Insgesamt 32 Symbole
  - Ziffern 0, ..., 9 mit jeweils 3 Symbolen (Codierung A, B, C),
  - Ein Randsymbol als erstes und letztes Zeichen
  - Ein Trennsymbol in der Mitte

# Aufbau von EAN-13

- Die Prüfziffer  $p \in \{0, 1, \dots, 9\}$  (13. Ziffer) wird so gewählt, dass gilt:

$$\underbrace{(z_1)}_{\text{implizit}} + \underbrace{3z_2 + z_3 + 3z_4 + z_5 + 3z_6 + z_7}_{\text{linker Teil}} + \underbrace{3z_8 + z_9 + 3z_{10} + z_{11} + 3z_{12} + p}_{\text{rechter Teil}} \equiv 0 \pmod{10}$$

- Codierung der Ziffern

- Erste Ziffer implizit (impliziert durch linke sechs Zahlen; siehe Tabelle)
- Linker Teil: jede Ziffer entweder codiert mit Code A oder B (abhängig von erster Ziffer)
- Rechter Teil: Code C

1. Ziffer	Codemuster linke Seite
0	AAAAAA
1	AABABB
2	AABBAB
3	AABBBA
4	ABAABB
5	ABBAAB
6	ABBBAA
7	ABABAB
8	ABABBA
9	ABBABA





## Beispiel EAN-13

Prüfziffer  $p$  und EAN-13 für: 123456789012

$$z_1 + 3z_2 + z_3 + 3z_4 + z_5 + 3z_6 + z_7 + 3z_8 + z_9 + 3z_{10} + z_{11} + 3z_{12} + p \equiv 0 \pmod{10}$$

$$1 + 3 \cdot 2 + 3 + 3 \cdot 4 + 5 + 3 \cdot 6 + 7 + 3 \cdot 8 + 9 + 3 \cdot 0 + 1 + 3 \cdot 2 + p \equiv 0 \pmod{10}$$

$$1 + 6 + 3 + 12 + 5 + 18 + 7 + 24 + 9 + 0 + 1 + 6 + p \equiv 0 \pmod{10}$$

$$92 + p \equiv 0 \pmod{10}$$

$$p = 8$$



# Codierungs- und Informationstheorie — Übersicht

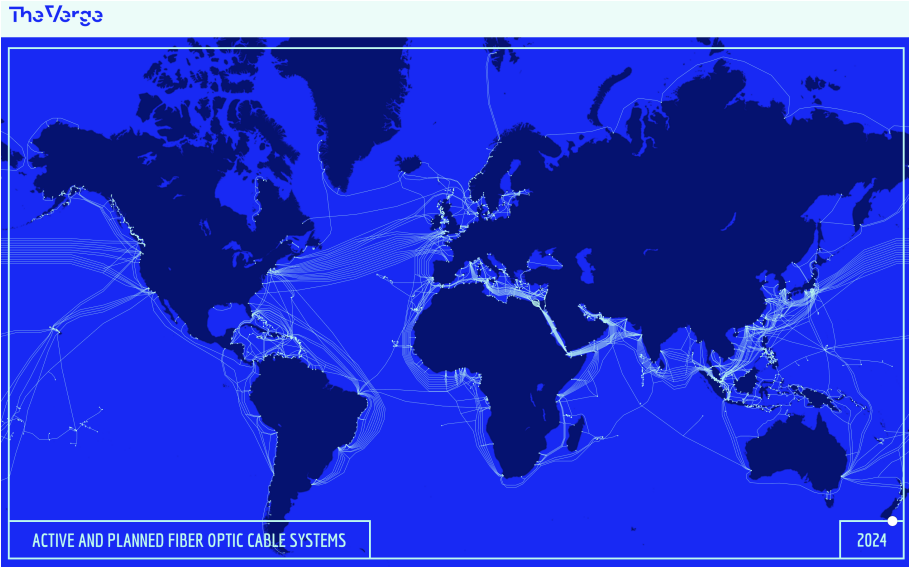
---

- 1 Zeichencodierungen
- 2 Linearer Barcode, EAN-13, andere Codierungen
- 3 **Codierungstheorie**
- 4 Fehlererkennende und fehlerkorrigierende Codes
  - Hamming-Distanz
  - Fehlerkorrigierende Codes
  - Hamming-Code
  - Polynomcodes
- 5 Arten von Codes
- 6 Informationstheorie und Kompression
  - Informationstheorie nach Shannon
  - Huffman-Code

Wie werden (Internet-)Daten weltweit  
übertragen?



# Datenübertragung weltweit<sup>1</sup>

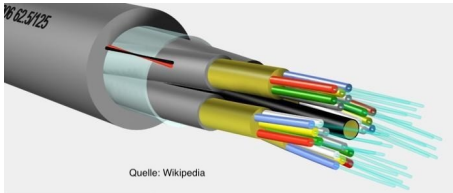


Credit: TeleGeography

<sup>1</sup><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



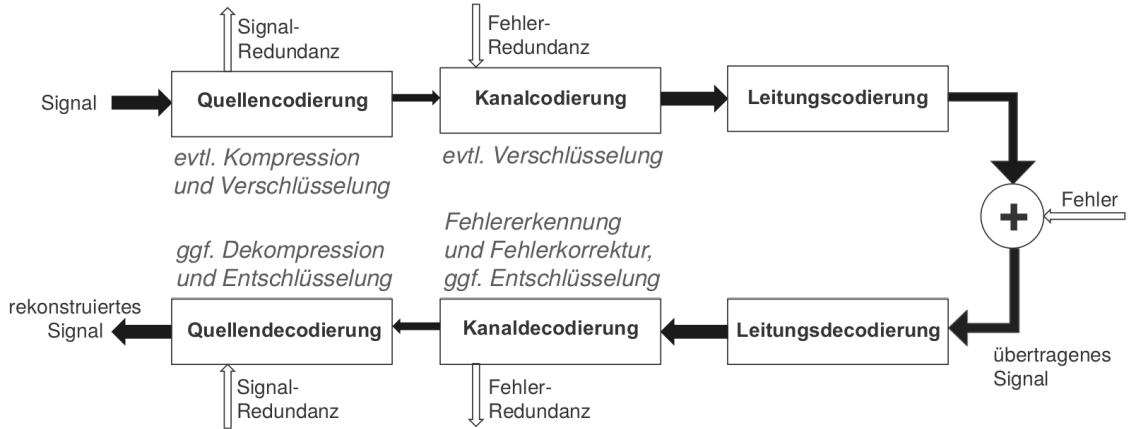
- **Codierungstheorie** ist die Theorie der fehlererkennenden und -korrigierenden Codes
- **Ziel:** Schutz vor Fehlern bei Übertragung oder Speicherung
- Verwendet Ideen aus Algebra, Kombinatorik, Zahlentheorie und Geometrie
- Begründer: Golay und Hamming, ca. 1950

- 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

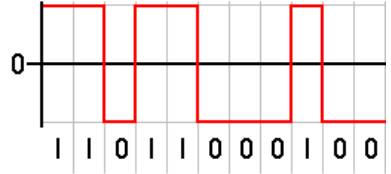


Quelle: nach Wikipedia

- 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]

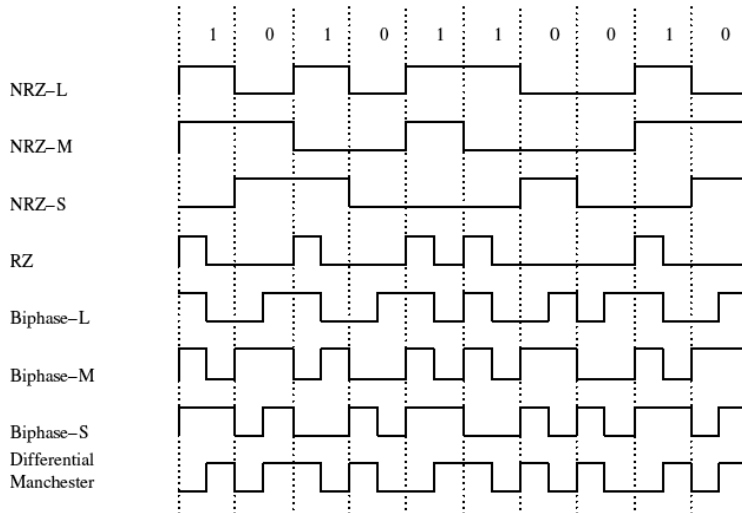
# Leitungscode NRZ

- 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



Quelle: Wikipedia

# 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 Linearer Barcode, EAN-13, andere Codierungen
- 3 Codierungstheorie
- 4 Fehlererkennende und fehlerkorrigierende Codes
  - Hamming-Distanz
  - Fehlerkorrigierende Codes
  - Hamming-Code
  - Polynomcodes
- 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 → falsche Nachricht wird empfangen
  - Zu übertragende Nachricht „a“ (00)
  - Falls erstes Bit gestört (10 statt 00) → „i“ empfangen
  - Falls zweites Bit gestört (01 statt 00) → „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

- a mit 00 und e mit 01 unterscheiden sich an einer Stelle

⇒ Hamming-Distanz  $D(00, 01) = 1$

# Hamming-Distanz

- Abstandsbestimmung von einem ganzen Code:
  - In einer Matrix für jedes Paar von Codewörtern (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  $D$  vom Code  $\mathcal{C}$  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	00
e	01
i	10
o	11

	a	e	i	o
a	-	1	1	2
e	1	-	2	1
i	1	2	-	1
o	2	1	1	-

- Zweiter Fall: Code mit  $D = 2$

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

a	000
e	011
i	101
o	110

	a	e	i	o
a	-	2	2	2
e		-	2	2
i			-	2
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 Fall von  $D = 2$  haben wir den Code vom ersten Fall genommen und entsprechend die 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 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: 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 Prozedere, 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

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

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 Index (als Codebit) in dieser Darstellung vorkommt
  - Beispiel:  $11 = 8 + 2 + 1$  oder  $29 = 16 + 8 + 4 + 1$
  - ⇒ Codebit 11 liefert Beitrag zu Prüfbits an den Stellen 1, 2 und 8

# 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 werden kann
- Beispiel: Die Prüfbits  $c_1, c_2$  und  $c_8$  waren nicht richtig, so muss das Bit 11 korrigiert (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

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

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

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

$$0 \oplus 0 = 0, \quad 1 \oplus 1 = 0$$

$$0 \oplus 1 = 1, \quad 0 \oplus 1 = 1$$

$$\oplus \dots \text{XOR}$$

# 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:

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

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

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

Zu übertragendes Wort: „**1100**110“

- Angenommen das Bit  $c_5$  wird bei der Übertragung gestört und daher das Wort „1100**0**10“ 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 „1100**0**10“ ergeben die Berechnungen:

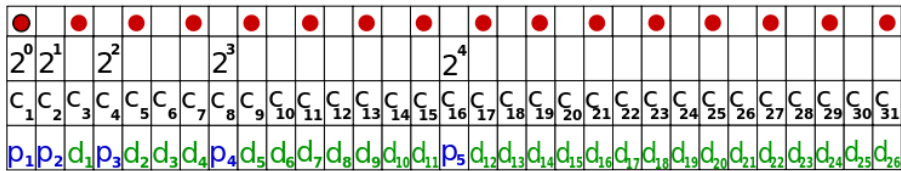
$$p_1 = c_1 = (c_3 + c_5 + c_7) \bmod 2 = c_3 \oplus c_5 \oplus c_7 = 0 \oplus 0 \oplus 0 = 0 \neq 1$$

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

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

- Da die Berechnungen für die Prüfbits  $p_1 = c_1$  und  $p_3 = c_4$  einen anderen Wert ergeben als übertragen wurde, weiß man, dass der Korrekturindikator den Wert  $5 = 1 + 4$  hat und somit das gestörte Bit  $c_5$  ist (unter der Annahme, dass nur ein Bit gestört wurde)

## Hamming-Code – Gleichung für Prüfbits



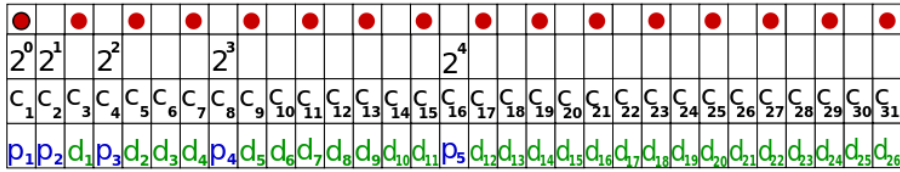
Quelle: Wikipedia

- 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  $d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8$ 
  - 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),  $\lfloor \lg(n) \rfloor + 1$  Prüfbits (blau)
- $n = k + \lfloor \lg(n) \rfloor + 1$  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

# Hamming-Code – Linearität

---

- Hatten 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 von einem **linearen Code**
  - **Definition:** Für alle Datenwörter  $m_1$  und  $m_2$  mit Codes  $c_1$  und  $c_2$  gilt: Wenn  $m = m_1 + m_2$  (bzgl. Modulo2-Arithmetik) dann ist auch  $c = c_1 + c_2$  (bzgl. Modulo2-Arithmetik)



# 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 also 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  $n = 2^\ell, \ell \in \mathbb{N}$ , Codebits
  - In diesem Fall wäre  $c_n = p_\ell$
  - Außerdem gäbe es für  $p_\ell$  keine gültige Prüfgleichung, weil die Codebits, auf die sich  $p_\ell$  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  $\lfloor \lg(n) \rfloor + 1$  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?

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

$m$ Datenbits	$r$ 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  $r \geq \lg(n + 1)$
- ⇒ Hamming-Code ist fast optimal
- Die obige Konstruktion (“untere Schranke”) gilt für jeglichen Code

- **Polynomcodes** sind Codes mit mehreren Prüfbits

- Erlauben Fehlererkennung
- Spezielle Polynomcodes erlauben ebenfalls Fehlerkorrektur (behandeln wir aber nicht)

- **Idee:**

- Bitfolgen, die codiert werden sollen, werden als Polynome interpretiert, deren Koeffizienten nur aus 0en und 1en bestehen
  - Wir erstellen ein **Message-Polynom**, das auf der ursprünglichen Bitfolge basiert
  - Außerdem einigen sich Encoder und Decoder auf ein sogenanntes **Generator-Polynom**
  - Daraus lässt sich ein **Transmissionspolynom** ableiten, dessen (Nicht-)Teilbarkeit durch  $G(x)$  Fehler aufdecken kann
- ⇒ Falls wir einen Rest erhalten, gab es einen Fehler bei der Übertragung

# Polynomcodes – Bitfolgen als Polynome

- Interpretation einer Bitfolge  $(a_{k-1} \cdots a_1 a_0)$  der Länge  $k$  als Koeffizienten eines Polynoms vom Grad  $k - 1$

$$a_{k-1}x^{k-1} + \dots + a_1x^1 + a_0x^0$$

- Dieses Polynom wird als **Message-Polynom** bezeichnet
- Beispiel: Bitfolge „110001“ entspricht dem Message-Polynom  $M(x) = x^5 + x^4 + x^0$
- „Grad“ eines Polynoms  $\rightarrow$  höchste vorkommende Potenz von  $x$
- Beispiel:  $x^5 + x^4 + 1$ 
  - $\Rightarrow$  Polynom fünften Grades
- **Achtung!** Wir rechnen mit diesen Polynomen immer modulo 2
  - Es gilt:

$$1 + 1 \equiv 0 \pmod{2}, \quad -1 \equiv +1 \pmod{2}$$

# Rechnen mit Polynomen modulo 2

---

## ■ Beispiel: Addition

$$(x^5 + x^4 + 1) + (x^5 + x^3 + x) \equiv x^4 + x^3 + x + 1 \pmod{2}$$

$$\text{da } 1 \cdot x^5 + 1 \cdot x^5 \equiv 0 \cdot x^5 \equiv 0 \pmod{2}$$

## ■ Beispiel: Multiplikation

$$(x + 1) \cdot (x + 1) \equiv x^2 + 1 \pmod{2}$$

$$\text{da } 1 \cdot x + 1 \cdot x \equiv 0 \cdot x \equiv 0 \pmod{2}$$

# Rechnen mit Polynomen modulo 2

## ■ Beispiel: Division

$$(x^5 + x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x + 1) : (x^2 + 1) = x^3 + x^2 + x + 1$$

$$\begin{array}{r} x^5 \quad \quad + \quad x^3 \\ \hline x^4 + \quad x^3 + 0 \\ x^4 + \quad \quad \quad x^2 \\ \hline \quad x^3 + x^2 + 0 \\ \quad x^3 + \quad \quad x \\ \hline \quad \quad x^2 + x + 1 \\ \quad \quad x^2 + \quad 1 \\ \hline \quad \quad \quad x \end{array}$$

Rest:  $R(x) = x$

# Polynomcodes – Überblick

---

- Generator-Polynom  $G(x)$  zur Berechnung der Prüfsumme wird vorab festgelegt
  - Ist sowohl Encoder als auch Decoder bekannt
  - Beispiel:  $G(x) = x^{16} + x^{12} + x^5 + 1$
- Basierend auf dem Message-Polynom und dem Generator-Polynom finden wir eine Prüfsumme (die ebenfalls ein Polynom ist)
  - Prüfsumme so wählen, dass das übertragene Wort durch  $G(x)$  ohne Rest teilbar ist
  - Anhängen dieser Prüfsumme an die Nachricht, d.h. an das Message-Polynom  $M(x)$
- Übertragen beider Teile (Message und Prüfsumme) als ein Codewort
- Ergibt das erhaltene Wort bei Division durch  $G(x)$  einen Rest ungleich 0, so liegt ein Übertragungsfehler vor



# Algorithmus zur Berechnung der Prüfsumme

$r$  ... Grad von  $G(x)$

$m$  ... Nachrichtenwortlänge

- 1 Hänge am Ende vom Nachrichtenwort  $r$  Nullen an (für Prüfsumme verwenden)
  - 2 Für Nachrichtenpolynom  $M(x)$  das erweiterte Nachrichtenpolynom  $x^r \cdot M(x)$  berechnen  
⇒ Das resultierende Codewort hat  $m + r$  Bit
  - 3 Berechne Rest  $R(x)$  von Modulo2-Division  $\frac{x^r \cdot M(x)}{G(x)}$
  - 4 Berechne das **Transmissionspolynom**  $T(x) = x^r \cdot M(x) - R(x)$ , das übertragen wird
- Folgende Eigenschaften gelten:
- $\frac{T(x)}{G(x)}$  hat Rest 0
  - Modulo-2-Subtraktion verändert nur die  $r$  angehängten Bits für die Prüfsumme, da der Grad von  $R(x)$  kleiner als  $r$  ist

# Polynomcode – Beispiel

- **Gegeben:** Nachrichtenwort „0110“ und Generator-Polynom  $G(x) = x^3 + x^2 + 1$ ;
- **Gesucht:** Zu übertragende Bitfolge
- $M(x) = 0 + x^2 + x + 0 = x^2 + x$
- Grad von  $G(x)$ :  $r = 3$
- Das Polynom  $x^r \cdot M(x) = x^3 \cdot (x^2 + x) = x^5 + x^4$  muss nun durch  $G(x)$  dividiert werden  
 $(x^5 + x^4) : (x^3 + x^2 + 1) = x^2$   
 $\underline{x^5 + x^4 + x^2}$   
       $x^2$  Rest
- Der Rest  $R(x)$  muss nun von  $x^r \cdot M(x)$  abgezogen werden:

$$T(x) \equiv x^r \cdot M(x) - R(x) \equiv x^5 + x^4 - x^2 \equiv x^5 + x^4 + x^2 \pmod{2}$$

⇒ Zu übertragende Bitfolge: „0110**100**“

# Übertragungsfehler – Erkennung

---

- Angenommen bei der Übertragung tritt eine Störung auf
- Es kommt also nicht das Polynom  $T(x)$  an, sondern das gestörte Polynom  $T(x) + E(x)$ , wobei  $E(x)$  ein **Error-Polynom** ist
- Bei der Decodierung wird das empfangene Polynom durch  $G(x)$  dividiert:

$$\frac{T(x) + E(x)}{G(x)}$$

- Da  $T(x)$  ohne Rest durch  $G(x)$  dividierbar ist, bleibt als Rest nur der Rest von  $\frac{E(x)}{G(x)}$
- ⇒ Störungen, die ein Error-Polynom  $E(x)$  erzeugen, das durch  $G(x)$  teilbar ist, können nicht erkannt werden, alle anderen sehr wohl

# Übertragungsfehler – Beispiel

---

- Bei der Übertragung wird das Wort „0110100“ gestört, es wird stattdessen „**1**110100“ empfangen
  - Diese Bitfolge entspricht dem Polynom  $x^6 + x^5 + x^4 + x^2$
  - Bei der Division dieses Polynoms durch  $G(x)$  ergibt sich als Resultat  $x^3 + x$  und als Rest  $x^2 + x$
- ⇒ Der Rest ist ungleich 0, der Übertragungsfehler wird somit erkannt
- ⇒ Zur Fehlerkorrektur kann ggf. der Rest in einer Tabelle nachgeschlagen werden (nicht hier behandelt)

# CRC-Codes (Cyclic Redundancy Check)

---

- **Cyclic Redundancy Check (CRC)** Codes sind spezielle Polynomcodes
  - Zyklische Polynomcodes (siehe einige Slides später)
  - Basieren auf speziellen Generator-Polynomen mit guten Eigenschaften
- 1961 von Peterson entwickelt
- Einfach in Hardware zu implementieren
- Einfach zu analysieren
- Dienen der Fehlererkennung
- Gut für Fehlererkennung bei verrauschten Kanälen
  - Beispiel: Telekommunikation, Netzwerke

# CRC-Standard-Polynome

---

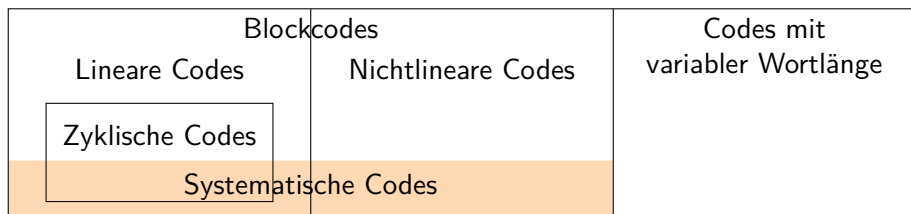
- Generator-Polynome, die zum Standard erhoben wurden:
  - $\text{CRC-12} = x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
  - $\text{CRC-16} = x^{16} + x^{15} + x^2 + 1$
  - $\text{CRC-CCITT} = x^{16} + x^{12} + x^5 + 1$
- CRC-12 wird für Bitfolgen der Länge 6, die beiden anderen werden für Folgen der Länge 8 verwendet
- Aus wie vielen Bits besteht ein Codewort bei CRC-12?
  - 6 Datenbits
  - 12 Absicherungsbits
  - ⇒ 18 Bits insgesamt

# Codierungs- und Informationstheorie — Übersicht

---

- 1 Zeichencodierungen
- 2 Linearer Barcode, EAN-13, andere Codierungen
- 3 Codierungstheorie
- 4 Fehlererkennende und fehlerkorrigierende Codes
  - Hamming-Distanz
  - Fehlerkorrigierende Codes
  - Hamming-Code
  - Polynomcodes
- 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



- Spezieller Blockcode
- Linear-Kombination von zwei Codewörtern ebenfalls Codewort
  - D.h. die Summe bzw. Vielfaches
  - **Definition:** Für alle Datenwörter  $m_1$  und  $m_2$  mit Codes  $c_1$  und  $c_2$  gilt: Wenn  $m = m_1 + m_2$  (bzgl. Modulo2-Arithmetik) dann ist auch  $c = c_1 + c_2$  (bzgl. Modulo2-Arithmetik)
- Modulo-2-Arithmetik!  $1 + 1 = 0$
- 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:**

Ein **zyklischer Code**  $C$  ist ein linearer Code, der zusätzlich folgende Eigenschaft hat:

Ist  $(a_0, a_1, \dots, a_{n-1})$  ein Codewort von  $C$ ,

dann ist auch  $(a_1, \dots, a_{n-1}, a_0)$  ein Codewort von  $C$ .

⇒ Daraus folgt, dass auch  $(a_2, a_3, \dots, a_{n-1}, a_0, a_1)$ ,  
 $(a_3, a_4, \dots, a_{n-1}, a_0, a_1, a_2), \dots, (a_{n-1}, a_0, a_1, \dots, a_{n-2})$   
Codewörter von  $C$  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

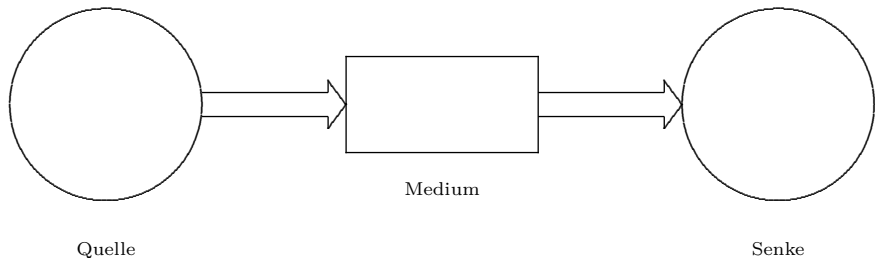
# Codierungs- und Informationstheorie — Übersicht

---

- 1 Zeichencodierungen
- 2 Linearer Barcode, EAN-13, andere Codierungen
- 3 Codierungstheorie
- 4 Fehlererkennende und fehlerkorrigierende Codes
  - Hamming-Distanz
  - Fehlerkorrigierende Codes
  - Hamming-Code
  - Polynomcodes
- 5 Arten von Codes
- 6 Informationstheorie und Kompression
  - Informationstheorie nach Shannon
  - Huffman-Code

# Das Modell der Informationsübertragung

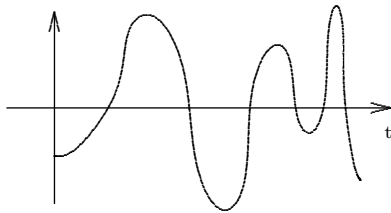
---



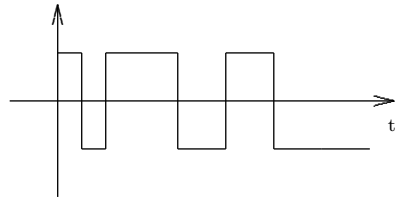
- Information soll von der Quelle (Sender) über ein Übertragungsmedium (Nachrichtenkanal) zu einer Senke (Empfänger) übermittelt werden
- Beispiel:
  - Sender und Empfänger sind Menschen
  - Der Sender spricht mit dem Empfänger, wobei als Übertragungsmedium Schallwellen fungieren

# 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

- 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.  $\{0 \mid 1\}$  oder  $\{\bullet \mid -\}$

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

# 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  $p$ :** Ein Zeichen  $x$  kommt  $n_x$  Mal in einem Wort der Länge  $n$  vor, dann ist  $p = \frac{n_x}{n}$
  - Beispiel: 0001 0100 0100, die relative Häufigkeit von 0 ist  $\frac{3}{4}$  und von 1 ist  $\frac{1}{4}$
- Der Informationsgehalt einer aus mehreren (voneinander unabhängigen) Zeichen bestehenden Nachricht soll gleich der Summe der Informationsgehalte der einzelnen Zeichen sein
  - Für Zeichen  $x$  und  $y$  sollten gelten:

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

wobei  $h(\cdot)$  der Informationsgehalt von  $x$  ist

- Informationsgehalt  $h$ :

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

$\text{ld} \dots$  *logarithmus dualis* ( $\log_2$ ); Basisumrechnung:  $\log_b x = \frac{\log_a x}{\log_a b}$  für  $a \neq 1$ ,  $b \neq 1$   
 $p \dots$  relative Häufigkeit

- 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
  - 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 x, y, z

	$p_i$	$h_i$
x	0.50	1
y	0.25	2
z	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:

x	1
y	01
z	00

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

# Mittlere Wortlänge und Redundanz

- Für einen gegebenen Code, sei  $\ell_i$  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  $\ell_i$  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):

$$r = R/L$$

	$p_i$	$h_i$	$p_i \cdot h_i$	$\ell_i$	$p_i \cdot \ell_i$
x	0.7	0.515	0.360	1	0.7
y	0.2	2.322	0.464	2	0.4
z	0.1	3.322	0.332	2	0.2

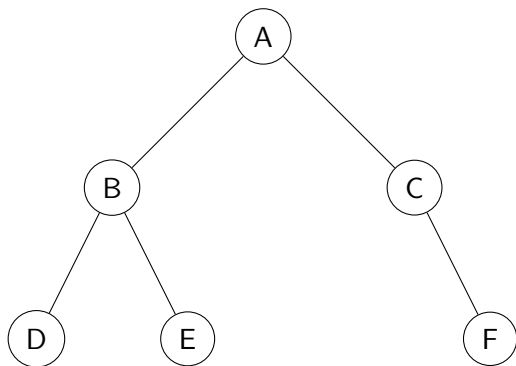
- $H = 0.7 \cdot 0.515 + 0.2 \cdot 2.322 + 0.1 \cdot 3.322 \text{ Bit} = 1.156 \text{ Bit}$
- $L = 0.7 \cdot 1 + 0.2 \cdot 2 + 0.1 \cdot 2 \text{ Bit} = 1.3 \text{ Bit}$
- $R = L - H = 1.3 - 1.156 = 0.144 \text{ Bit}$
- $r = R/L = 0.144/1.3 \approx 0.111 \text{ (dimensionslos)}$



- **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)

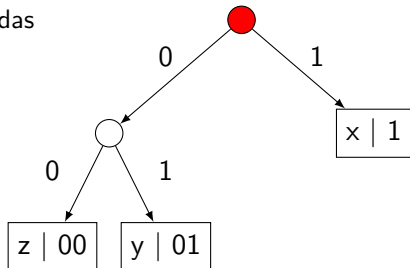
- Ein (ungerichteter, ungewichteter) Graph  $G$  ist gegeben durch ein Tupel  $(V, E)$ 
  - $V$  = die Knoten des Graphen
  - $E \subseteq V \times V$  ist die Menge der Kanten
    - Eine Kante ist ein Paar von Knoten
    - Im Beispiel:  
 $V = \{A, B, C, D, E, F\}$   
 $E = \{(A, B), (B, D), (B, E), (A, C), (C, F)\}$
- 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:

x	1
y	01
z	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

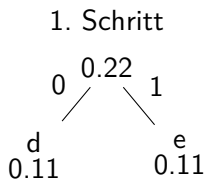
---

- Alphabet besteht aus Zeichen  $\{a \mid b \mid c \mid d \mid e\}$

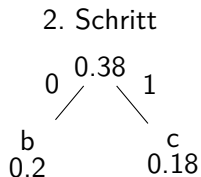
	$p$
a	0.40
b	0.20
c	0.18
d	0.11
e	0.11

# Beispiel

	$p$
a	0.40
de	0.22
b	0.20
c	0.18



	$p$
a	0.40
bc	0.38
de	0.22

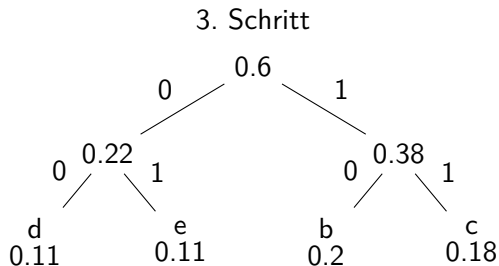


# Beispiel

	$p$
a	0.40
de	0.22
b	0.20
c	0.18

	$p$
a	0.40
bc	0.38
de	0.22

	$p$
bcde	0.60
a	0.40



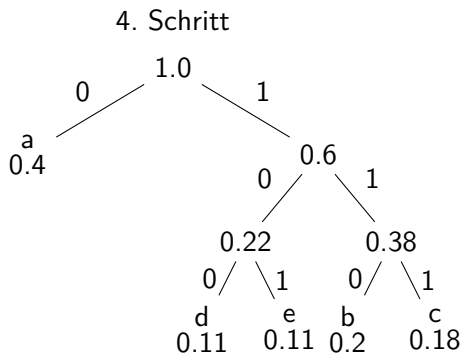
# Beispiel

	$p$
a	0.40
de	0.22
b	0.20
c	0.18

	$p$
a	0.40
bc	0.38
de	0.22

	$p$
bcde	0.60
a	0.40

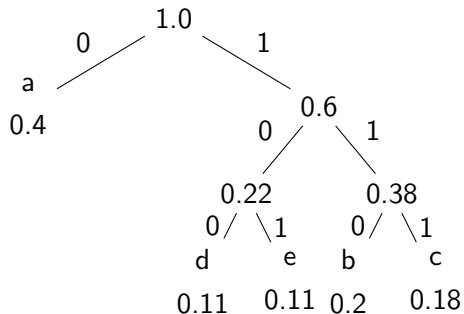
	$p$
abcde	1.0





# Beispiel

4. Schritt



	$p$	Codew.	$l$	$p \cdot l$
a	0.40	0	1	0.40
b	0.20	110	3	0.60
c	0.18	111	3	0.54
d	0.11	100	3	0.33
e	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  $\{a \mid b\}$

	$p$	Codewort
a	0.8	0
b	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

# Beispiel

---

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

	$p$	Codewort	$\ell$	$p \cdot \ell$
aa	0.64	0	1	0.64
ab	0.16	10	2	0.32
ba	0.16	110	3	0.48
bb	0.04	111	3	0.12

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

# Anmerkungen (nicht prüfungsrelevant)

---

- Die oben beschriebene Methode kann bei einer ungünstigeren 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 die 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