



Kap. 4: Grundlagen der Codierung

4.1 Einführung

4.2 Blockcodes

4.3 Codes variierender Länge

4.4 Komprimierende Codes

4.5 Fehlererkennende und -korrigierende Codes



Quellen

- **M. Broy: "Informatik - Eine grundlegende Einführung", Teil II, Springer-Verlag, 1992 (Kap. 1)**
- **U. Rembold, P. Levi: "Einführung in die Informatik für Naturwissenschaftler und Ingenieure", 3. Auflage, Hanser-Verlag, 1999 (Kap. 2.1.)**
- **D. Werner u.a.: "Taschenbuch der Informatik", Fachbuchverlag Leipzig, 1995 (Kap. 2.2)**
- **F. Mayer-Lindenberg: "Konstruktion digitaler Systeme", Vieweg-Verlag, 1998 (Kap. 2)**
- **J. Plate: "Modem-Technik", Skript FH München, 1998**
- **H. Dispert, H.-G. Heuck: "Einführung in die Technische Informatik und Digitaltechnik", Vorlesungsskript FH Kiel, © 1995 (Kap. 8.7), http://www.e-technik.fh-kiel.de/~dispert/digital/digital/dig0_00.htm**
- **W. Görke: "Fehlertolerante Rechensysteme", Oldenbourg Verlag, 1989 (Kap. 4.1)**



4.1 Einführung

Wiederholung (Kap. 2.4):

- **Codes** oder **Codierungen** sind Abbildungen $c:A \rightarrow B$ oder $c:A^* \rightarrow B^*$ zwischen Zeichenvorräten A und B oder zwischen Wörtern über Zeichenvorräten.
- Die Bildmenge $\{b \in B \mid b=c(a), a \in A\}$ unter c, d.h. die Menge der Codewörter von c, wird ebenfalls **Code** genannt.
- Die Ausgangszeichen heißen auch **Klarzeichen**, die Zielelemente **Codezeichen** und **Codewörter**.
- Besteht der Definitionsbereich einer Codierung aus Einzelzeichen, so heißt die Codierung auch **Chiffrierung** und die Bildmenge auch **Chiffren**.
- Eine Codierung erlaubt für dieselbe betrachtete Information den Übergang von Zeichen und Wörtern eines gegebenen Repräsentationssystems zu Zeichen und Wörtern eines neuen Repräsentationssystems.



Weitere Begriffe

- In der Regel ist die Abbildung eines Codes injektiv, d.h. verschiedene Zeichen oder Wörter werden auf verschiedene Codewörter abgebildet. Damit ist die auf der Bildmenge umkehrbare Codierung beschrieben durch eine Abbildung
$$d: \{b \in B \mid b=c(a), a \in A\} \rightarrow A,$$
die **Decodierung** genannt wird.
- Codierung wird auch als **Verschlüsselung**, Decodierung als **Entschlüsselung** bezeichnet.
- Für die Informationsdarstellung in Rechensystemen werden fast ausschließlich **Binär-Codierungen** (**Binär-Codes**) **von Alphabeten** betrachtet. Dies sind Codierungen der Form
$$c: A \rightarrow \{0,1\}^*,$$
wobei A ein vorgegebenes Alphabet ist.
- Beispiele (vgl. Kap. 3):
Codierung ganzer Zahlen, Gleitkommazahlen, Text/Zeichenketten



Ziele von Codierungen

- **Funktionalität**
 - Repräsentationssysteme zur Speicherung, Verarbeitung und Übertragung von Information
 - Umkehrbarkeit (Decodierbarkeit)
 - Ordnungserhaltung nach Codierung (z.B. für Sortierung)
 - Änderung an nur einer Stelle beim Übergang zum nächsten Zahlenwert (Korrektheit bei Messwerterfassung)
- **Effizienz**
 - übersichtliche und wenig aufwändige Codierungsfunktion
 - einfache und wirtschaftliche Verarbeitung in der neuen Repräsentierung
 - einfache Komplementbildung (für Arithmetik)
 - einfache Realisierung arithmetischer Operationen
 - möglichst kurze Codewörter; Reduktion von Speicherbedarf, Übertragungszeit, Energiekosten usw.



Ziele von Codierungen (2)

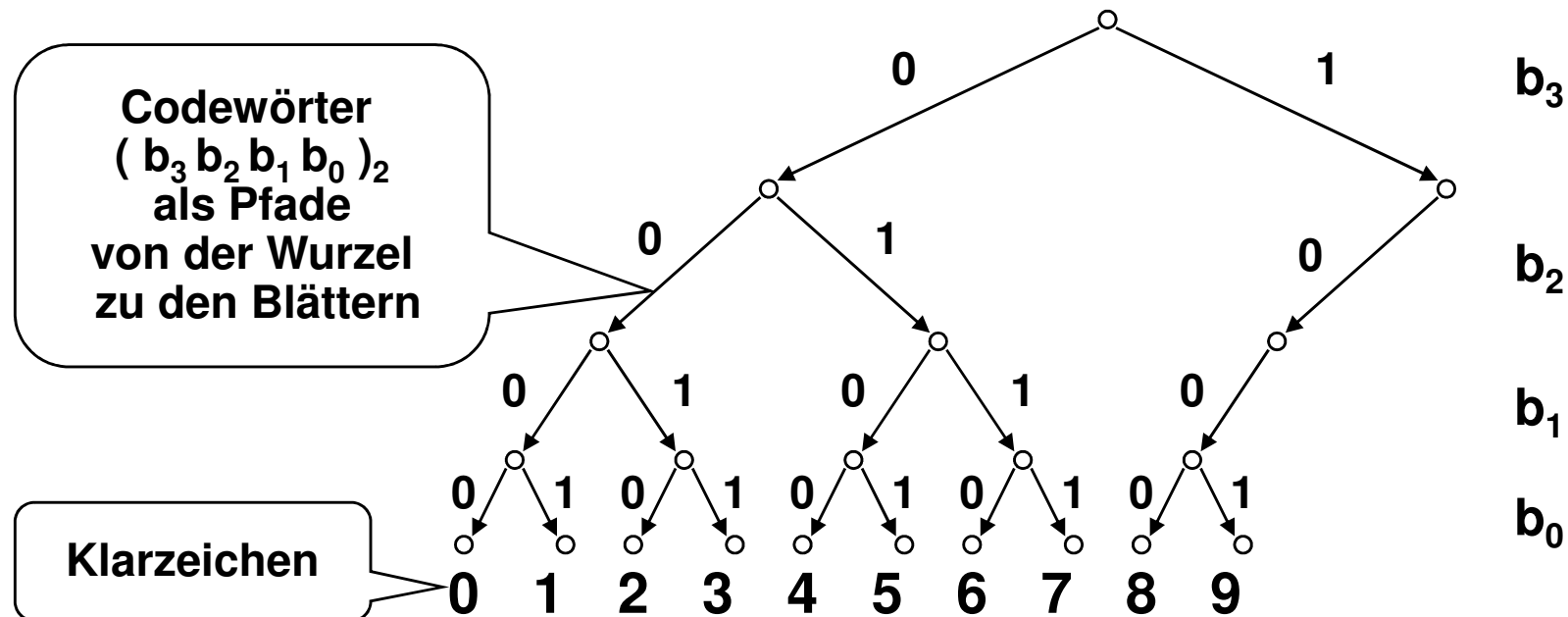
- **Effizienz (Forts.)**
 - **Beispiele:**
 - Codierung und Arithmetik ganzer Zahlen (vgl. Kap. 3.3)
 - UTF-8-Codierung von Unicode-Zeichen (vgl. Kap. 3.4.6)
- **Sicherung gegen Verfälschung**
 - Fehler können zu Veränderungen der Repräsentierung von Information während der Speicherung und Übertragung führen (Störung) !
 - Erkennen von Fehlern in "geringfügig" gestörten Codewörtern
 - Erkennen von Verarbeitungsfehlern
 - automatische Korrektur fehlerhafter Codewörter ohne Informationsverlust (korrekte Decodierung)
 - Maßnahmen konkurrieren mit Effizienz
- **Geheimhaltung von Information**
 - Verschlüsselung (kryptographische Methoden)



Codebaum

Def

- Jeder Binär-Code kann graphisch durch einen **binären Codebaum** dargestellt werden:
 - jeder Stelle im Codewort wird eine Schicht im Baum zugeordnet
 - jedem Binärwert wird ein linker und ein rechter Unterbaum zugeordnet
- Beispiel (Codebaum des BCD-Codes):





4.2 Blockcodes

Def

- Ein Code $c:A \rightarrow B^n$, dessen Codewörter alle die gleiche Länge n besitzen, heißt (*n-stelliger*) **Blockcode**.
- Die Anzahl der möglichen Codewörter eines Blockcodes $c:A \rightarrow B^n$ beträgt $|B|^n$.
- Ein n -stelliger Blockcode $c:A \rightarrow B^n$ heißt **dicht**, wenn c surjektiv ist, d.h. wenn alle $b \in B^n$ Codewörter unter c darstellen.
- Binäre Blockcodes $c:A \rightarrow \{0,1\}^n$ sind für Rechensysteme besonders geeignet, da die Codewörter n -Bit-Maschinenwörtern entsprechen.
- Ist $|A|=m$, d.h. A besteht aus m Zeichen, so benötigt man für einen binären Blockcode $c:A \rightarrow \{0,1\}^n$ mindestens $n = \lceil \log_2 m \rceil$ Stellen. ($\lceil x \rceil$ bedeutet "kleinste ganze Zahl größer gleich x ".)



Beispiele

- Die Codierung ganzer Zahlen in 2er-Komplement-Darstellung in n-Bit-Maschinenwörtern (vgl. Kap. 3.3.3) stellt einen binären, dichten Blockcode dar.
- Der ASCII-Code ist ein dichter 7-Bit Blockcode.
- Die BCD-Codierung von Dezimalziffern in 4-Bit Nibbles (vgl. Kap. 3.3.4) ist nicht dicht.
- Die Anzahl der möglichen Codewörter in einem n-Bit-Maschinenwort beträgt 2^n .
- Zur Codierung der 10 Dezimalziffern in einem binären Blockcode benötigt man mindestens $n = \lceil \log_2 10 \rceil = 4$ Stellen.



Gewichtete Codes

Def

- Ein binärer Blockcode $c:A \rightarrow \{0,1\}^n$ zur Codierung von Zahlen heißt **gewichtet** oder **bewertbar**, wenn den Stellen der Codewörter Gewichte W_i zugeordnet sind und sich der Wert der dargestellten Zahl z ergibt zu

$$z = \sum_{i=1}^n b_i W_i ,$$

wobei $b_i \in \{0,1\}$, $i=1,\dots,n$ die den Gewichten in der Codierung von z zugeordneten binären Ziffern entsprechen.

Andernfalls heißt der Code **Anordnungscode**.



Beispiele: gewichtete BCD-Codes

	üblicher BCD-Code	51111- Code
W_i	$2^3 2^2 2^1 2^0$	5 1 1 1 1
0	0 0 0 0	0 0 0 0 0
1	0 0 0 1	0 0 0 0 1
2	0 0 1 0	0 0 0 1 1
3	0 0 1 1	0 0 1 1 1
4	0 1 0 0	0 1 1 1 1
5	0 1 0 1	1 0 0 0 0
6	0 1 1 0	1 1 0 0 0
7	0 1 1 1	1 1 1 0 0
8	1 0 0 0	1 1 1 1 0
9	1 0 0 1	1 1 1 1 1



Beispiele: gewichtete BCD-Codes (2)

	Aiken- Code				
W_i	2	4	2	1	
0	0	0	0	0	— Pseudotetraden liegen in der Mitte
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	— selbstkomplementierend (Vertauschen 0-1 ergibt Komplement)
5	1	0	1	1	
6	1	1	0	0	— monoton wachsend
7	1	1	0	1	
8	1	1	1	0	— Rundungserkennung (≥ 5 , < 5) am vordersten Bit
9	1	1	1	1	

- Übertrag stimmt mit Dezimalübertrag überein



Beispiele: gewichtete BCD-Codes (3)

	2-aus-5-Code				
W_i	7	4	2	1	0
0	1	1	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	1	0	0	0	1
8	1	0	0	1	0
9	1	0	1	0	0

- bis auf die Null monoton wachsend
- fehlererkennend (für alle 1-Bit-Fehler)
- Einsatz: Strichcode (5 Striche: 3 schmal, 2 breit) (Postleitzahlencodierung)



Beispiele: gewichtete BCD-Codes (4)

	1-aus-10- Ring-Code	
W_i	9876543210	
0	0000000001	— monoton wachsend — sehr übersichtlich — großer Aufwand — Einsatz: Anzeigen, numerische Tastaturen
1	0000000010	
2	0000000100	
3	0000001000	
4	0000010000	
5	0000100000	
6	0001000000	
7	0010000000	
8	0100000000	
9	1000000000	



Gray-Code

Def

- Sei A ein Alphabet. (Damit ist eine lineare Ordnung auf A definiert.) Ein binärer Blockcode $c:A \rightarrow \{0,1\}^n$ heißt **Gray-Code** oder **einschrittiger Code**, wenn sich die Codierungen zweier in der Ordnung aufeinander folgender Zeichen in A stets in genau einer Bit-Stelle unterscheiden.
- Unterscheiden sich die Kodierung des ersten und des letzten Zeichens von A ebenfalls nur in einer Stelle, so spricht man von einem **zyklischen Gray-Code**.
- Gray-Codes besitzen wegen ihrer Einschrittigkeit eine hohe Bedeutung bei der Messwerterfassung, z.B. bei der Analog/Digital-Wandlung in Drehwinkelgebern, da sie inkorrekte Messergebnisse vermeiden.



Beispiele

zyklischer,
4-stelliger
Gray-Code

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

zyklischer Gray-Code
für BCD-Ziffern
(Glixon-Code)

	2^3	2^2	2^1	2^0
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				

0  1 

Darstellung wird auch
Code-Lineal genannt

Vergleich:
Dual-Darstellung

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

Ablese-
Einheit

7→8: Alle Stellen
ändern gleichzeitig
ihren Wert.
Fehleranfällig !

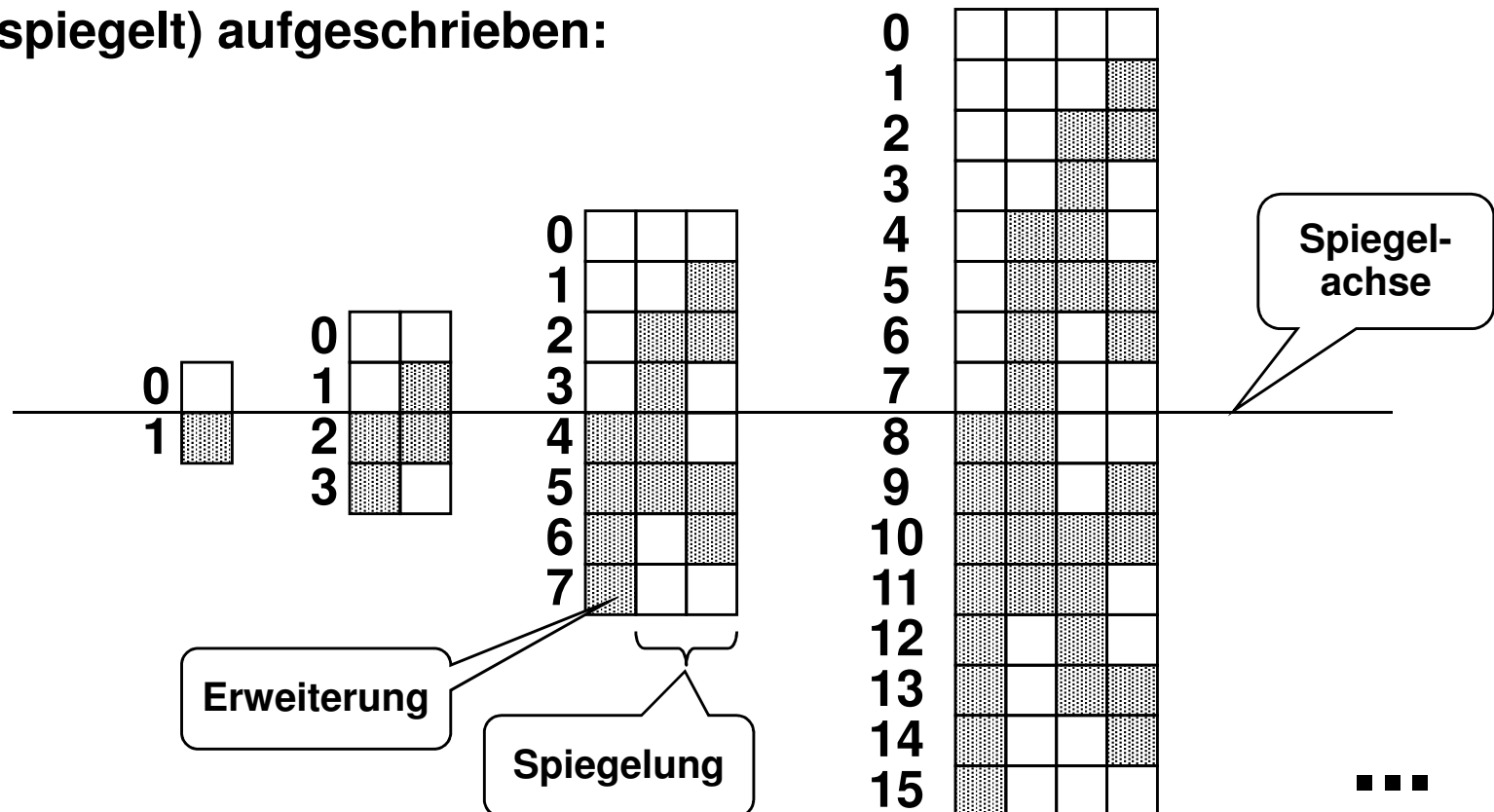


Bildung des n-stelligen Gray-Codes

- **Bildungsgesetz (rekursiv):**

- 0 und 1 werden mit den Dualzahlen 0 und 1 codiert.
- Wenn eine neue Stelle gebraucht wird, wird sie mit 1 besetzt.

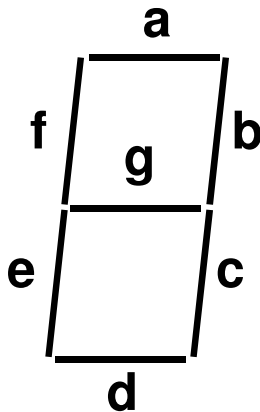
In den folgenden Stellen wird die bisherige Codierung rückwärts (gespiegelt) aufgeschrieben:





Blockcodes für spezielle Aufgaben

- Beispiel: 7-Segment-Code für Dezimalziffern-Anzeigen:



	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1



4.3 Codes variierender Länge

Def

- Ein Code $c:A \rightarrow B^*$, dessen Codewörter verschiedene Längen besitzen können, heißt *variabel langer Code* oder *Code variierender Länge*.
- Beispiel (Impulswahlverfahren des Fernsprechsystems):
 $c:\{0, \dots, 9\} \rightarrow \{O, L\}^*$

Ziffer	Code
1	LO
2	LLO
3	LLLO
4	LLLLO
5	LLLLLO
6	LLLLLLO
7	LLLLLLLO
8	LLLLLLLLLO
9	LLLLLLLLLLO
0	LLLLLLLLLLLO

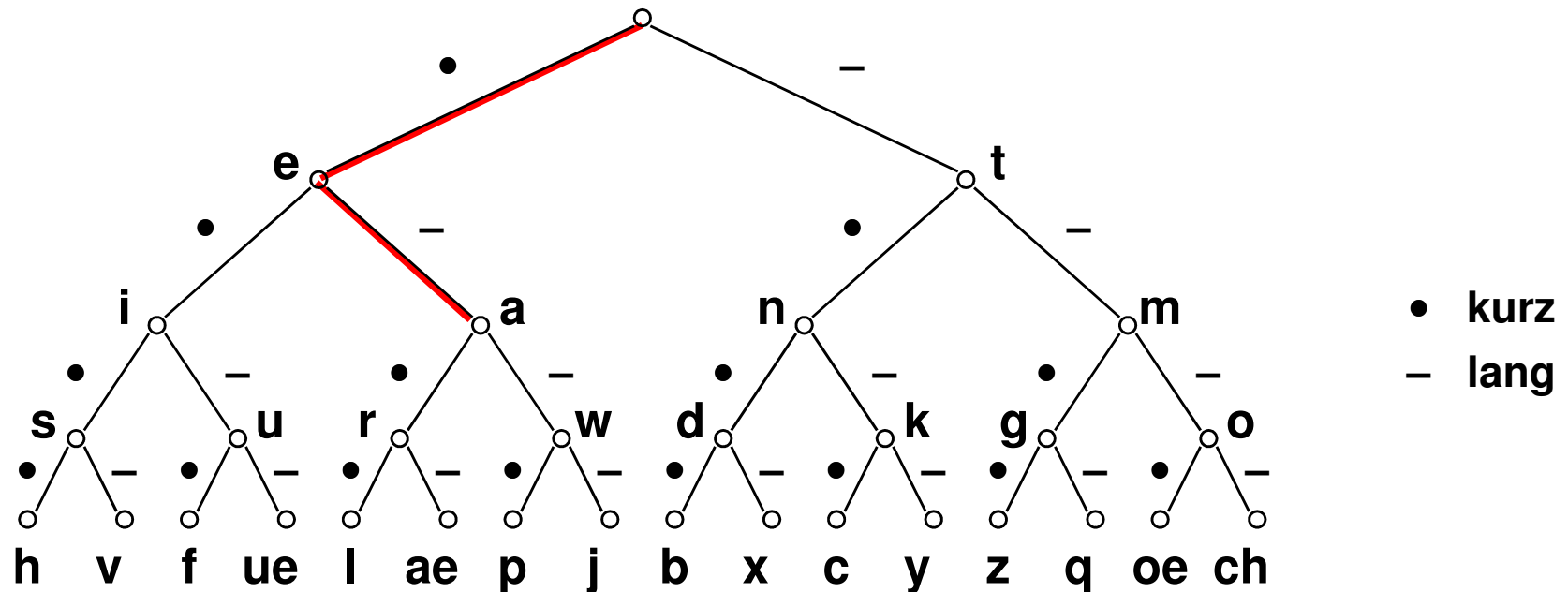
Codes variierender Länge in Rechensystemen

- **In heutigen Rechensystemen sind Codes variierender Länge für elementare Datentypen aufgrund der Maschinenwort-Struktur nicht sehr gebräuchlich.**
- **Beispiele:**
 - **Codierung von Maschinenbefehlen**
 - z.B. 2, 4 oder 6 Bytes lange Befehle in IBM /360-Architektur
 - Bit-variabel lange Befehle im Prozessor Intel iAPX 432
 - **UTF-8-Codierung von Unicode-Zeichen in 1, 2, ..., 6 Bytes (vgl. Kap. 3.4.6)**
 - **Bitstream-Codierung in Abstract Syntax Notation One (ASN.1)**
 - **ASN.1: Eine formale Sprache zur abstrakten Beschreibung von Nachrichten für den Austausch zwischen Anwendungen.**
 - **Siehe auch <http://www.itu.int/ITU-T/asn1/>**
 - **Plattform-unabhängig, kompakt, schnell – und grundlegend.**



Beispiel: Morse-Code

- Ausschnitt als Codebaum:



⇒ Häufig vorkommende Schriftzeichen besitzen kurze Codewörter.
Zur korrekten Decodierung wurde zur Trennung der Codewörter ein drittes Codezeichen "Pause" (□) eingeführt.

Beispiel: • – : a • □ – : e t • □ • • □ – – □ • □ • – • □ : e i m e r



Fano-Bedingung

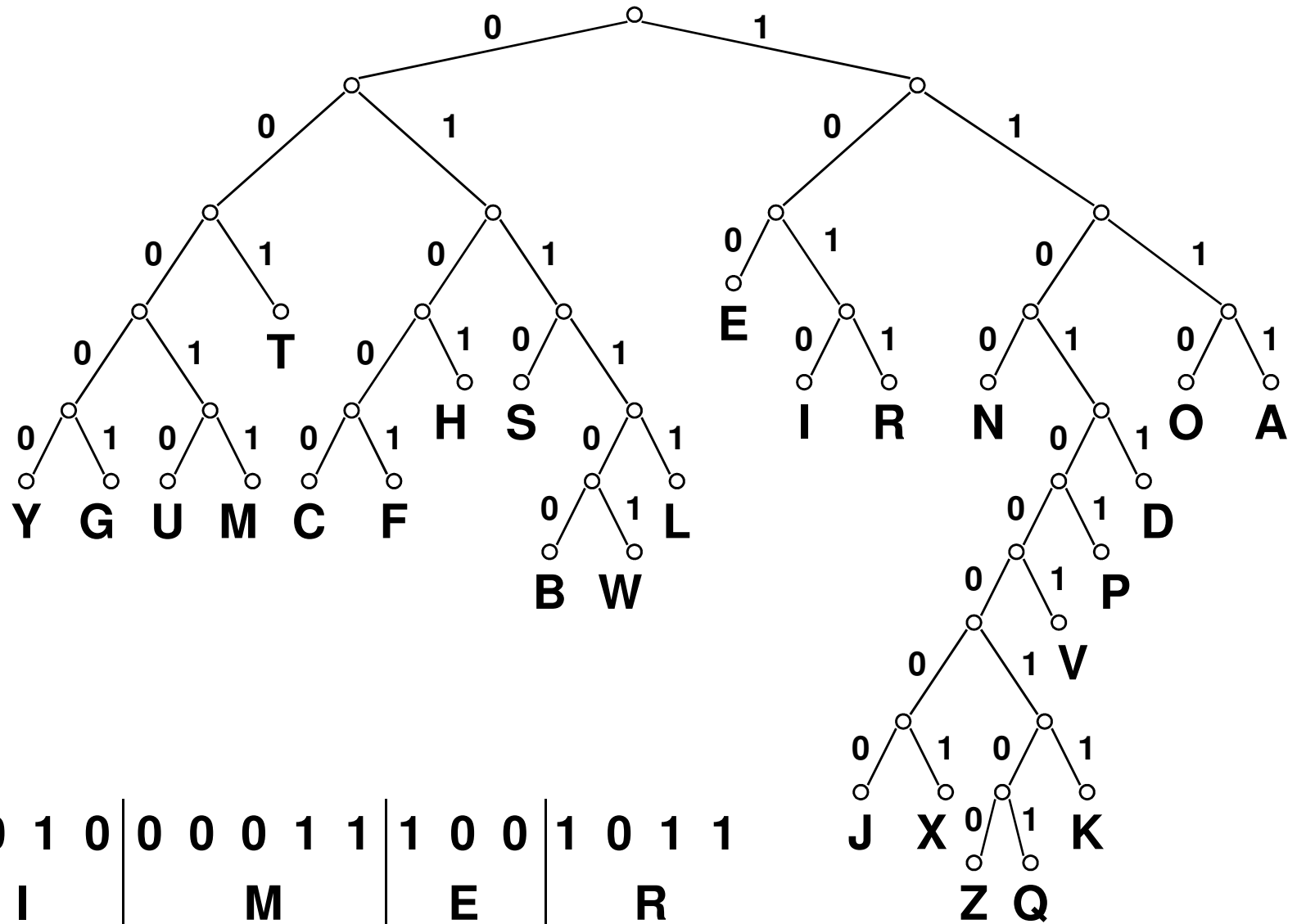
- Vorteil von Codes variierender Länge:
Häufig vorkommende Zeichen aus A können durch kurze Codewörter dargestellt werden.
- Problem:
Trennen der Codewörter voneinander ist schwieriger.
- Die Eindeutigkeit der Decodierung eines Codes variierender Länge ist gegeben, wenn der Code die sogenannte **Fano-Bedingung** erfüllt: Kein Codewort ist Präfix (Anfangsstück) eines anderen Codewortes.
- Bemerkungen:
 - Die Fano-Bedingung ist gleichbedeutend damit, dass im Codebaum zu codierende Zeichen nur als Blätter des Baumes auftreten.
 - Codes mit erfüllter Fano-Bedingung heißen auch **präfixfrei**.

Def



Beispiel

Huffman-
Code für das
engl.
Alphabet als
präfixfreier
Binär-Code
variierender
Länge





Beispiel (2)

- Jeder Blockcode der Länge n erfüllt die Fano-Bedingung automatisch. Zur Decodierung werden jeweils Blöcke von n Codezeichen gebildet und decodiert.

Beispiel: BCD-Codierung der dezimalen Ziffern in Tetraden.

- Der Morse-Code erfüllt die Fano-Bedingung nicht. Zur sicheren Decodierung verwendet er ein Trennzeichen zwischen Codewörtern.



4.4 Komprimierende Codes

- **Ziele komprimierender Codes**
 - Reduktion der Länge der Repräsentierung von Information durch Kompression
 - Kostenersparnis
- **Anwendungsbereich:**
Speicherung und Übertragung von Information.
- **Hier besprochen:**
Verlustfreie Codierungen, die eine vollständige und korrekte Decodierung ermöglichen (z.B. für Text, Programme usw. notwendig)
- **Überblick**
 - Lauflängenkodierung (*Run Length Encoding, RLE*)
 - Wörterbuch-Kompression
 - Huffman-Codierung
 - Shannon/Fano-Codierung



Lauf­längen­kodierung

- (engl.: *Run Length Encoding*, RLE)
- Viele Daten enthalten Läufe, d.h. Folgen identischer Zeichen.
- Idee:
Folge identischer Zeichen durch (Anzahl, Zeichen) codieren.
- Problem:
Unterscheidung des Zählers von Daten gleicher Repräsentierung
- RLE codiert Läufe beliebiger Zeichen typischerweise durch
 - (Zeichen, Marker, Anzahl)
 - Marker in Daten codiert durch (Marker, Marker)



LaufLängenkodierung (2)

- **Beispiel:**
 - Marker: #
 - "ABBBB BBBCDEEEEEEEEEEEF#34777777" (31 Zeichen)
 - komprimiert: "AB#7CDE#11F##347#6" (18 Zeichen)
- Vereinfachung in Sonderfällen möglich,
z.B. bei 7-Bit-Zeichen \Rightarrow MSB als Markierung des Zählers
- Anwendungsspezialfall: Null-Unterdrückung



Wörterbuchkompression (Lempel-Ziv)

- Wörterbuch mit Tupeln (Phrase, Codewort) wird schrittweise erzeugt
 - Phrase: Folgen von Eingabezeichen
 - erzeugte Codewörter enthalten Verweise in das Wörterbuch
- Vorteile
 - adaptiv (selbstanpassend)
 - optimal, wenn Tabelle beliebig groß und Eingabe unendlich lang sind
- Problem: Datenstruktur für das Wörterbuch
 - Tabelle, Baum, Hash-Funktion (vgl. 2. Semester: Datenstrukturen)
- Verfahren:
 - LZ77 (Lempel, Ziv, 1977), neuere Varianten `pkzip`, `gzip`
 - LZ78 (Lempel, Ziv, 1978), Baum statt Tabelle als Basis
 - LZW (Lempel, Ziv, Welch, 1984), komplexeres Tabellenverfahren, Basis von `compress` und dem Bildformat `gif`



Beispiel

- Eingabe: AAABAABAABAABB...

① A A A B A A B A A B A A B B ...



Wörterbuch und erzeugter Code:

Eintrag #	1
Phrase	A
Codewort	(0,A)

② A A A B A A B A A B A A B B ...



Wörterbuch und erzeugter Code:

Eintrag #	1	2
Phrase	A	AA
Codewort	(0,A)	(1,A)



Beispiel (2)

③ A A A B A A B A A B A A B B ...
 ↑

Wörterbuch und erzeugter Code:

Eintrag #	1	2	3
Phrase	A	AA	B
Codewort	(0,A)	(1,A)	(0,B)

④ A A A B A A B A A B A A B B ...
 ↑

Wörterbuch und erzeugter Code:

Eintrag #	1	2	3	4
Phrase	A	AA	B	AAB
Codewort	(0,A)	(1,A)	(0,B)	(2,B)



Beispiel (3)

⑤ A A A B A A B A A B A A B B ...
↑

Wörterbuch und erzeugter Code:

Eintrag #	1	2	3	4	5
Phrase	A	AA	B	AAB	AABA
Codewort	(0,A)	(1,A)	(0,B)	(2,B)	(4,A)

⑥ A A A B A A B A A B A A B B ...
↑

Wörterbuch und erzeugter Code:

Eintrag #	1	2	3	4	5	6
Phrase	A	AA	B	AAB	AABA	AABB
Codewort	(0,A)	(1,A)	(0,B)	(2,B)	(4,A)	(4,B)

...



Häufigkeitsabhängige Codierungen

- Ein Code $c:A \rightarrow B^*$ mit variierender Länge kann ebenfalls zur Komprimierung von Worten $w \in A^*$ genutzt werden.
- Annahmen:
 - $A = \{ a_1, a_2, \dots, a_m \}$.
 - Die **relative Häufigkeit** p_i jedes Zeichens $a_i \in A$ in w sei bekannt.
 - Abhängigkeiten zwischen Zeichen werden nicht betrachtet, d.h.: zu jedem Zeitpunkt entspricht die Wahrscheinlichkeit für ein neu zu kodierendes Zeichen genau der relativen Häufigkeit p_i des Zeichens (in der Informationstheorie als **stochastische Quelle** bezeichnet).
 - Es darf durch die Codierung kein Informationsverlust auftreten.



Häufigkeitsabhängige Codierungen (2)

- **Aufgabe:**
Gesucht ist ein Binärcode $c:A \rightarrow \{0,1\}^*$ zur Codierung von $w \in A^*$, sodass die Gesamtlänge $L = |c(w)|$ des codierten Wortes minimal ist.
- **Idee:**
 - Häufiger vorkommende Zeichen werden durch kurze Codewörter dargestellt, weniger wahrscheinliche durch längere.
 - ⇒ Die Gesamtlänge zur Repräsentation verkleinert sich.



Anmerkung

- Die Aufgabenstellung entspricht einem Optimierungsproblem:

Sei l_i die Länge des unbekannten Codeworts für $a_i \in A$,
 N_i die Anzahl der Vorkommen von a_i in w , $N = |w|$

Dann ist:

$$L = |c(w)| = N_1 l_1 + N_2 l_2 + \dots + N_m l_m \stackrel{!}{=} \min$$

Division durch N liefert:

$$dl = L/N = \sum_{i=1}^m p_i l_i \stackrel{!}{=} \min$$

ist zu
minimieren

Die Aufgabenstellung ist also gleichbedeutend mit der
Minimierung der durchschnittlichen Länge dl der Codewörter.



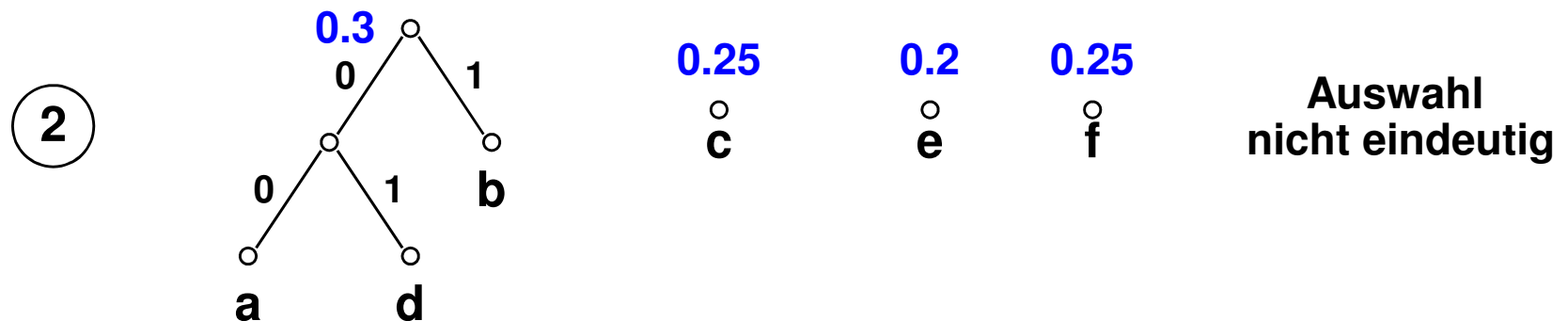
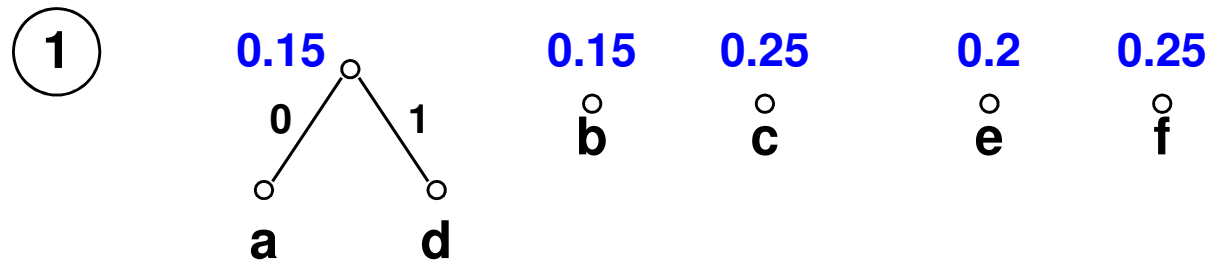
Huffman-Codierung

- Die Huffman-Codierung generiert anhand der relativen Häufigkeiten einen optimalen Code, der die mittlere Codewortlänge minimiert.
- Verfahren zur Konstruktion des Codebaums:
 - (1) Ordne jedem Zeichen einen isolierten Knoten mit dem Gewicht der relativen Häufigkeit des Zeichens zu.
 - (2) Suche die beiden Zeichen/Teilbäume mit dem geringsten Gewicht.
 - (3) Gruppierung:
 - Bilde einen binären Teilbaum mit diesen Zeichen/Teilbäumen.
 - Ordne den beiden neuen Kanten die Codierungen 0 und 1 frei zu.
 - Ordne dem Teilbaum die Summe der Gewichte der beiden Zeichen/Teilbäume als Gewicht zu.
 - (4) Wiederhole (2) und (3) so lange, bis ein einziger binärer Baum mit dem Gewicht 1 existiert.



Beispiel

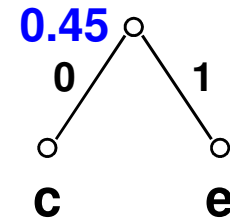
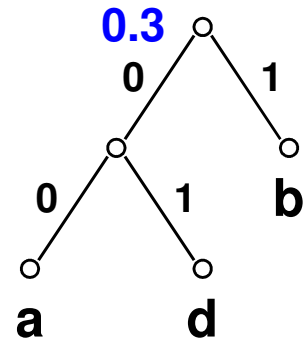
- $A = \{ a, b, c, d, e, f \}$
- Gegebene relative Häufigkeiten:
(0.1, 0.15, 0.25, 0.05, 0.2, 0.25)
- Entwicklung des Codebaums:



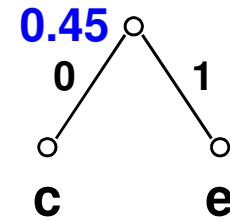
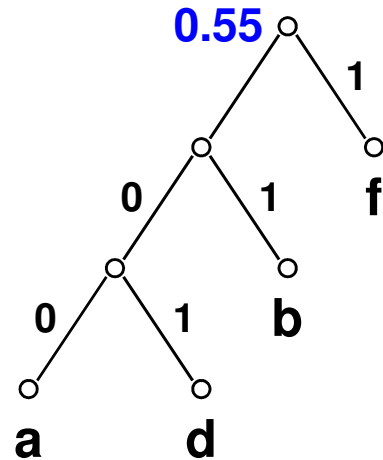


Beispiel (2)

3

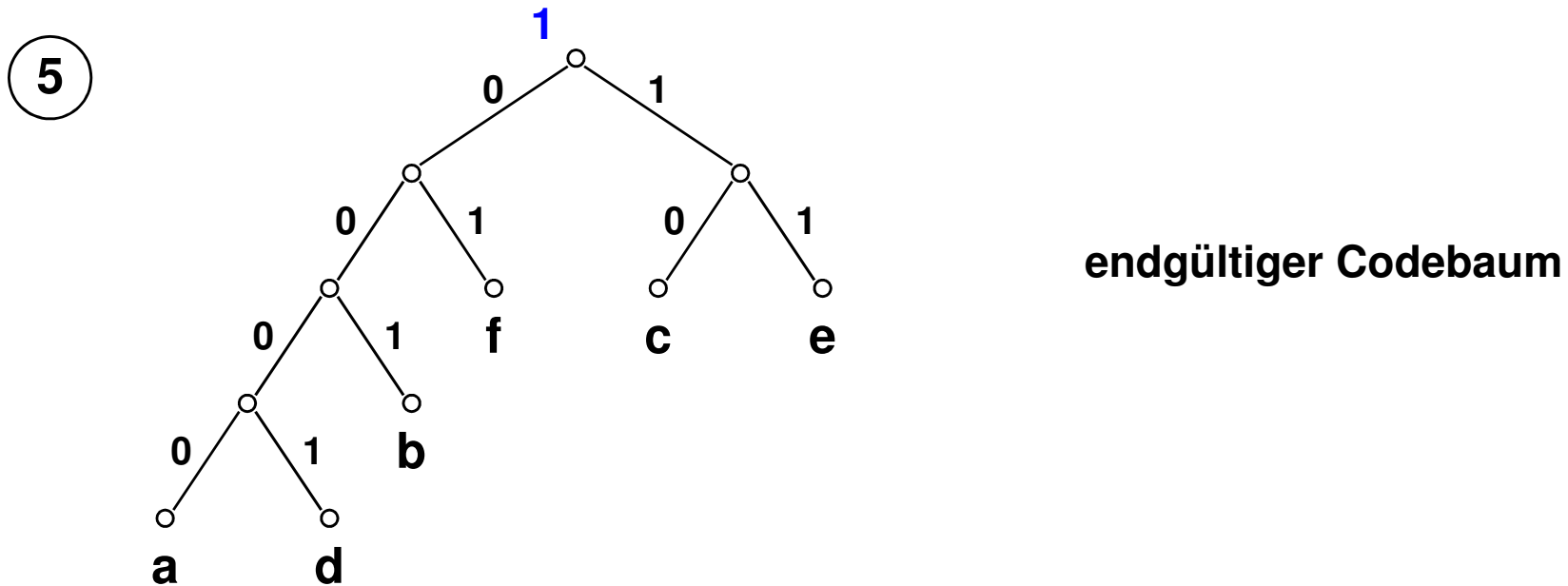


4





Beispiel (3)



- Bestimmung der mittleren Codewortlänge:

Zeichen	a	b	c	d	e	f
rel. Häufigkeit p_i	0.1	0.15	0.25	0.05	0.2	0.25
Länge l_i	4	3	2	4	2	2

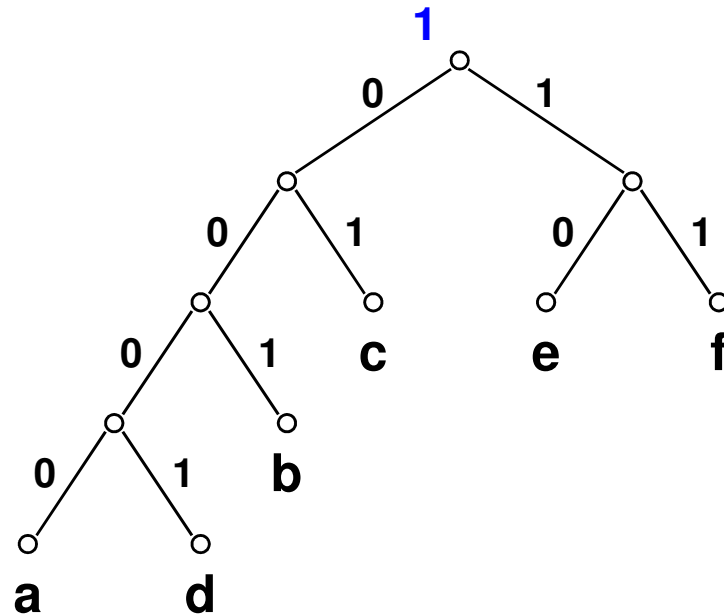
⇒ mittl. Codewortlänge $dl = 0.4 + 0.45 + 0.5 + 0.2 + 0.4 + 0.5 = 2.45$ Bit

Unter Verwendung eines Blockcodes wären $\lceil \log_2 6 \rceil = 3$ Bit notwendig



Beispiel (4)

- Bei Wahl der anderen Alternative am Ende von Schritt 2 hätte sich der folgende Codebaum ergeben :



... und damit die gleiche mittlere Codewortlänge.



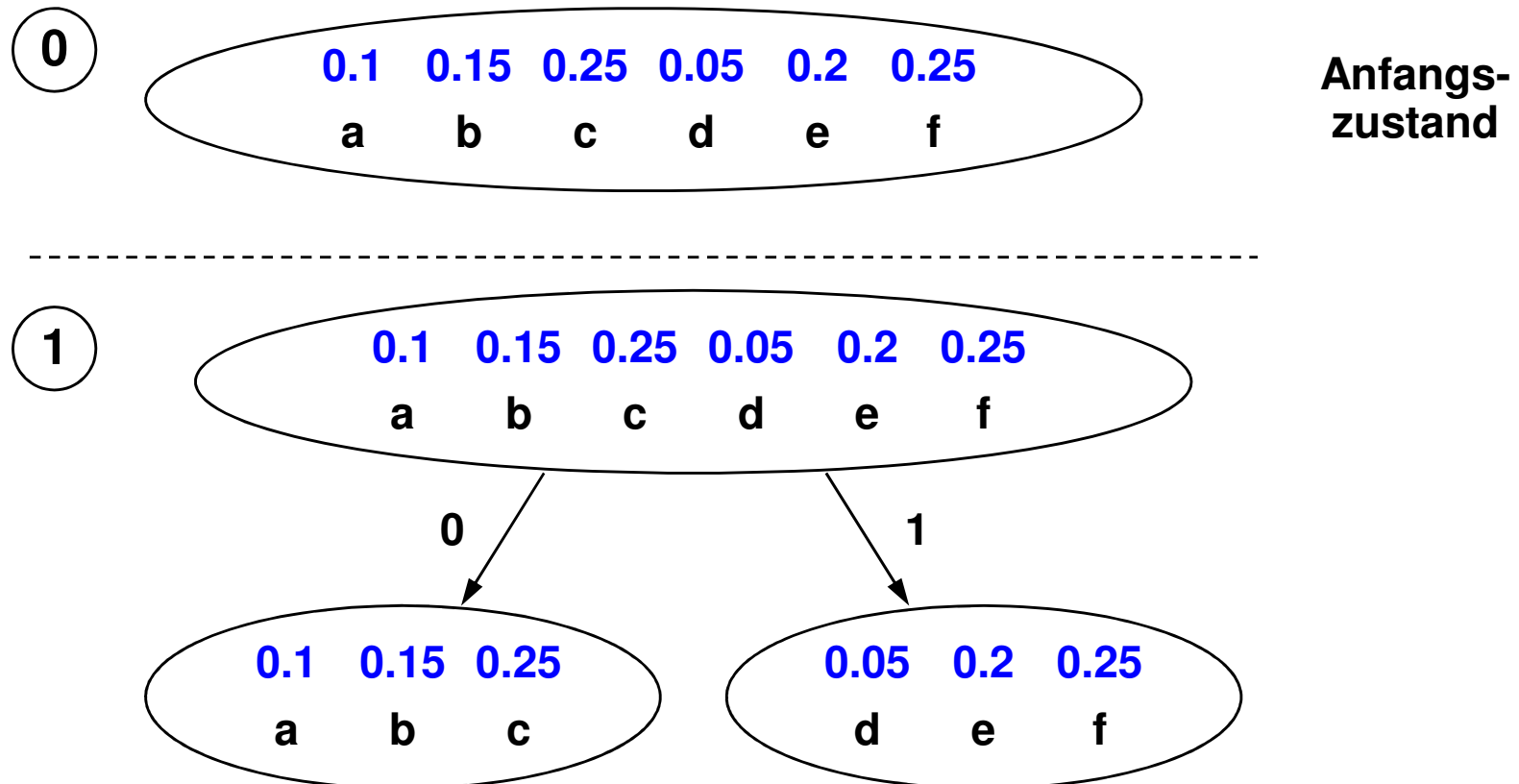
Shannon/Fano-Codierung

- Die Shannon/Fano-Codierung entwickelt den Codebaum im Gegensatz zur Huffman-Codierung top-down.
- Verfahren zur Konstruktion des Codebaums:
 - (1) Bilde die Wurzel des Baumes bestehend aus der Menge aller Zeichen und dem Gewicht aus der Summe aller relativen Häufigkeiten (1).
 - (2) Wähle ein Blatt des Baumes, dessen zugeordnete Menge M von Zeichen nicht einelementig ist.
 - (3) Teilung:
 - Teile M in zwei möglichst gleichgewichtige Teilmengen M_0 und M_1 .
 - Ordne M als linkes und rechtes Kind M_0 und M_1 zu sowie den neuen Kanten die Codierungen 0 und 1 zu.
 - (4) Wiederhole (2) und (3) so lange, bis alle Blätter des Baumes einelementig sind.



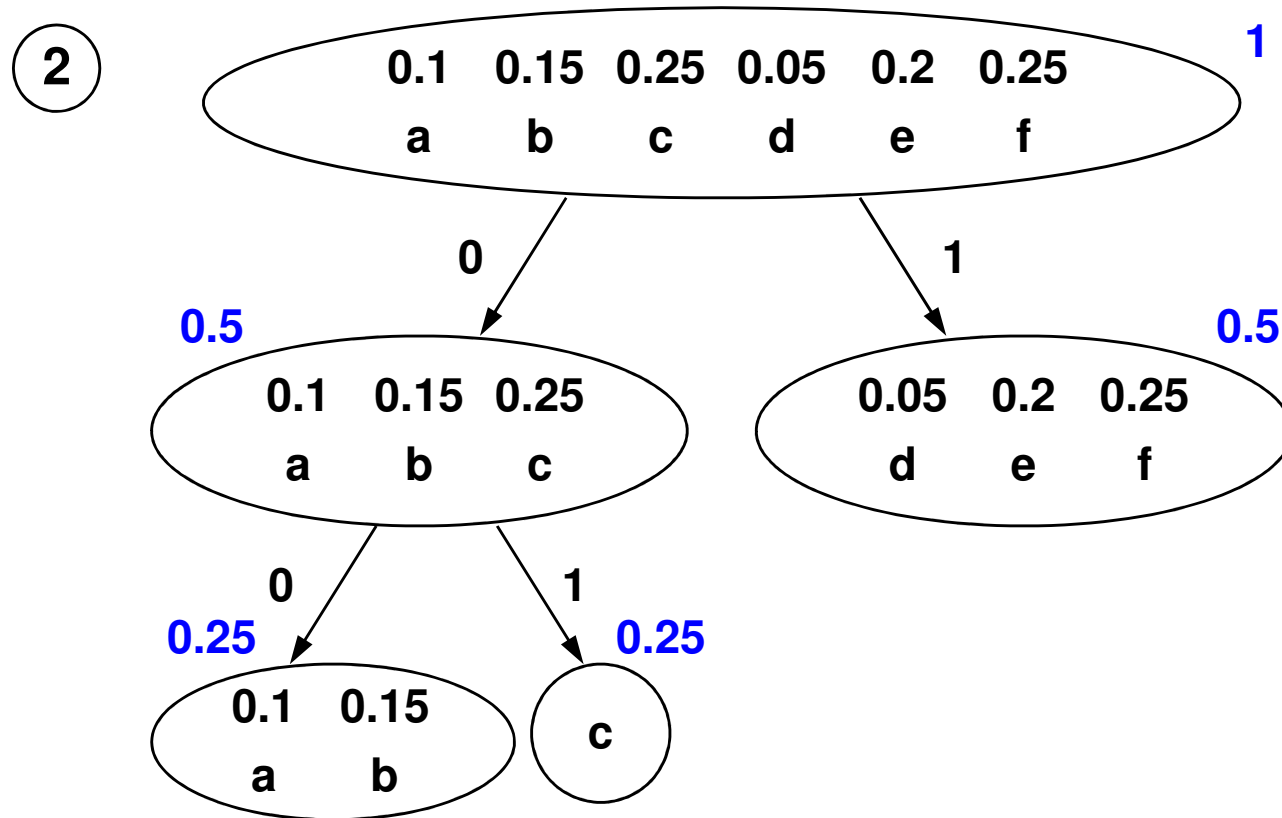
Beispiel

- $A = \{ a, b, c, d, e, f \}$
- Gegebene relative Häufigkeiten: (0.1, 0.15, 0.25, 0.05, 0.2, 0.25)
- Entwicklung des Codebaums:



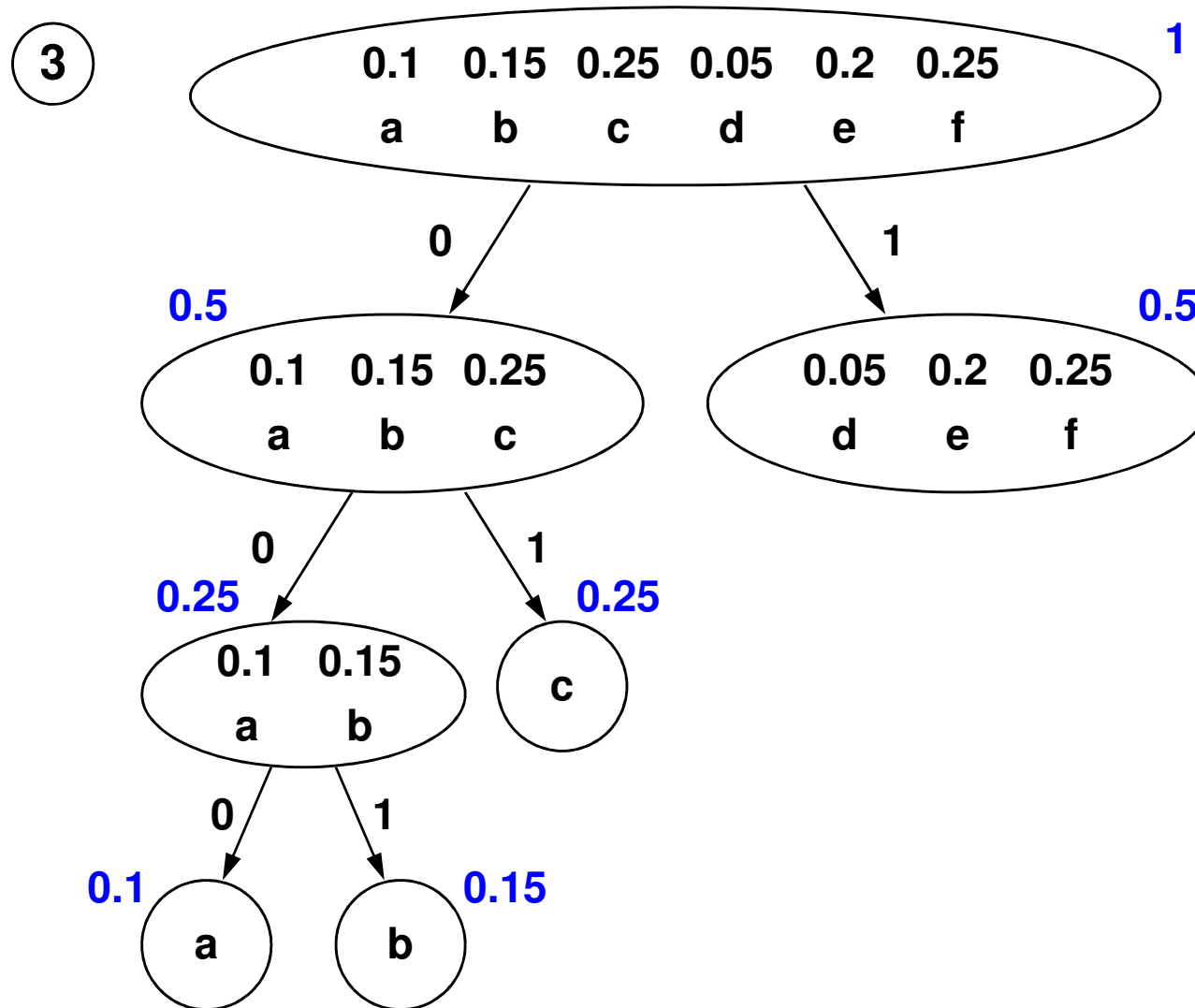


Beispiel (2)



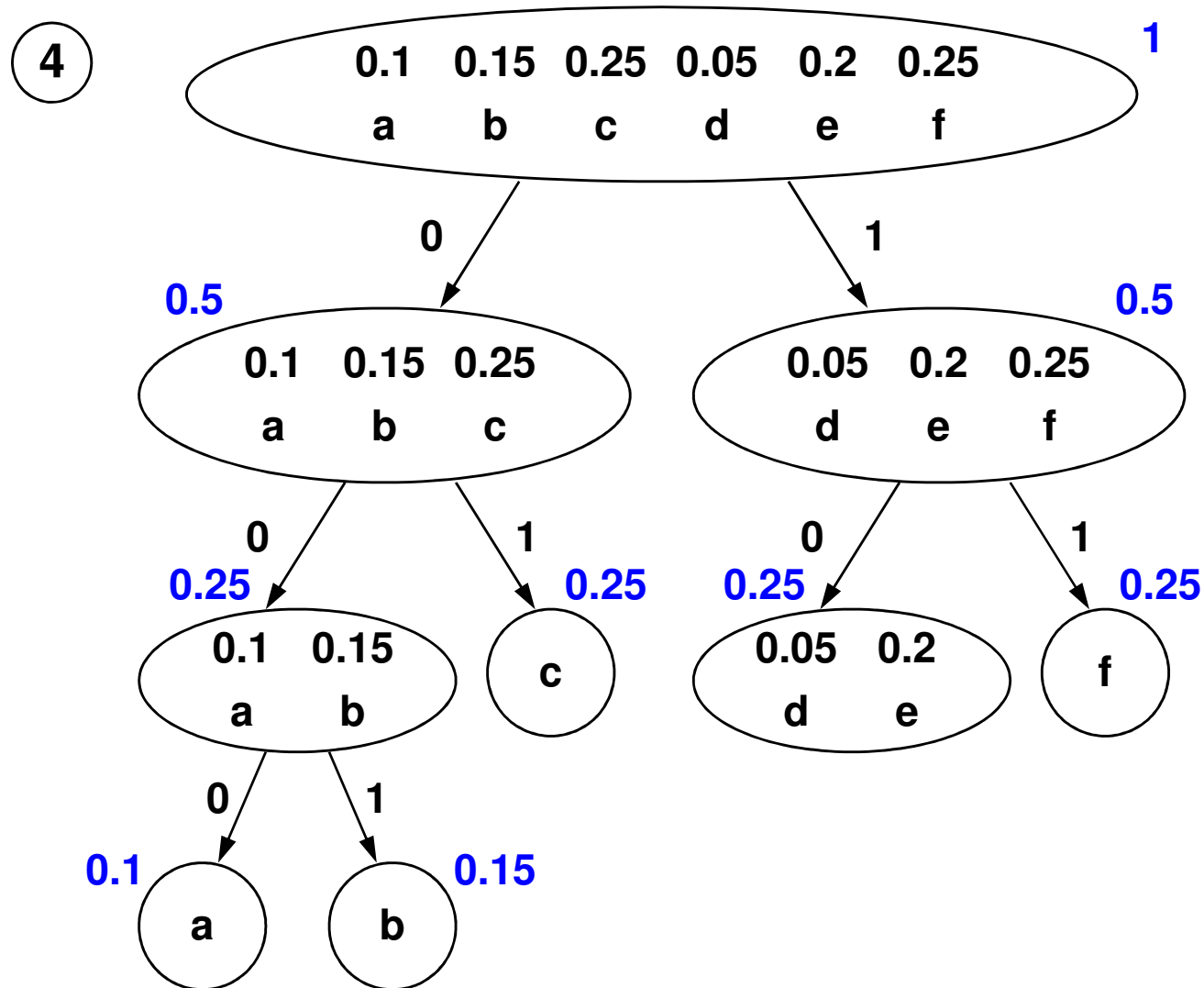


Beispiel (3)



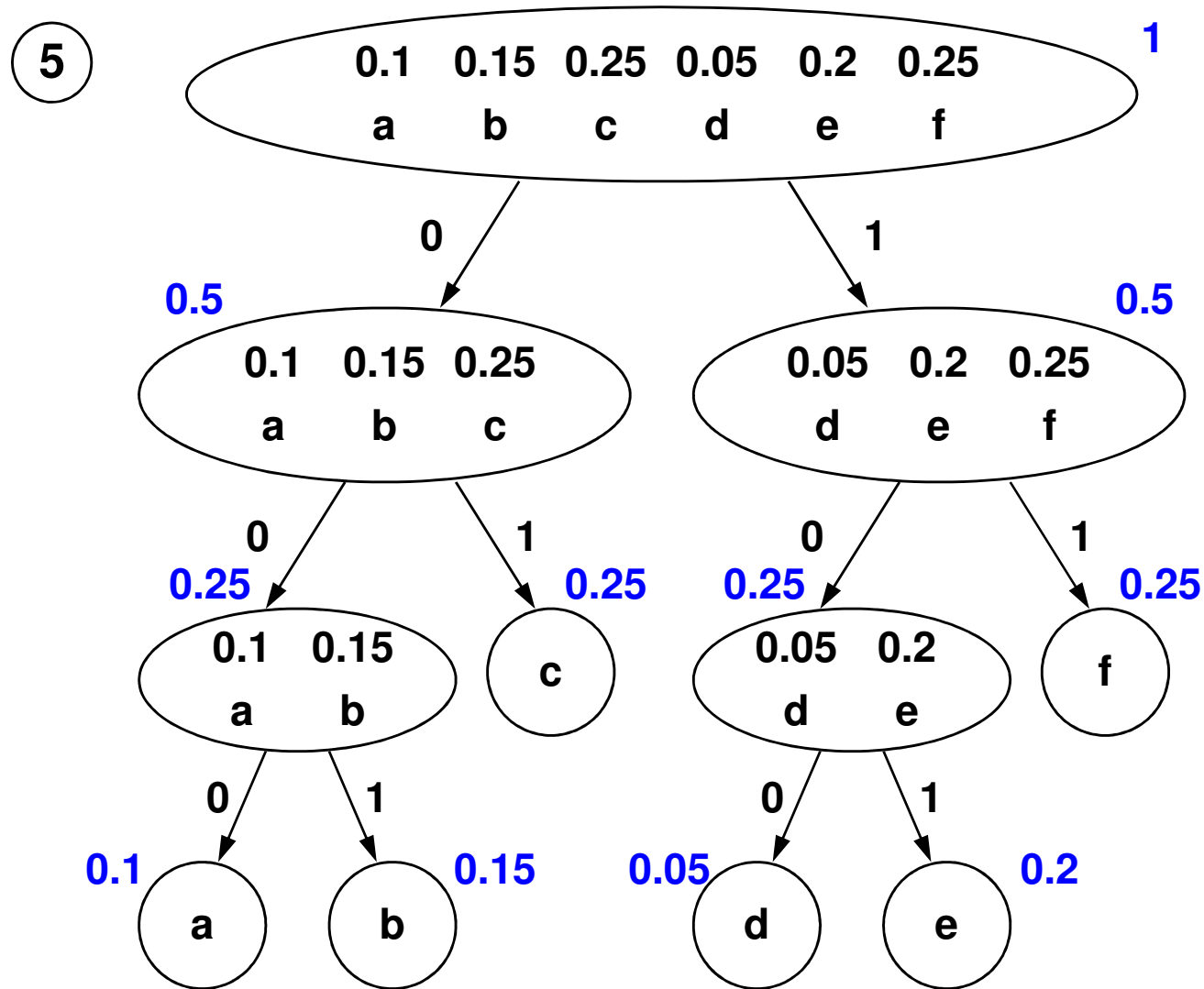


Beispiel (4)





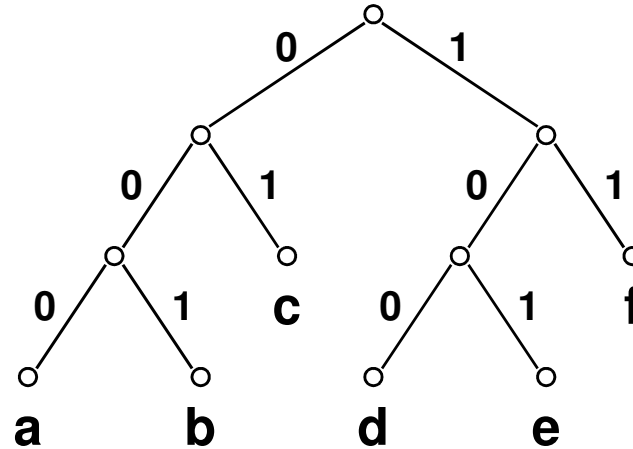
Beispiel (5)





Beispiel (6)

- **Endgültiger Codebaum:**



- **Bestimmung der mittleren Codewortlänge:**

Zeichen	a	b	c	d	e	f
rel. Häufigkeit p_i	0.1	0.15	0.25	0.05	0.2	0.25
Länge l_i	3	3	2	3	3	2

⇒ **mittl. Codewortlänge $dl = 0.3 + 0.45 + 0.5 + 0.15 + 0.6 + 0.5 = 2.5$ Bit**



Anmerkungen

- Die Suche nach optimalen Codes wird im Rahmen der Vorlesung "Informations- und Systemtheorie" (2. Semester) auf eine informationstheoretische Basis gestellt.
- Das dort besprochene **Codierungstheorem von Shannon** besagt,
 - dass es eine untere Grenze für die mittlere Codewortlänge dl gibt, die durch den mittleren "Informationsgehalt" der Nachrichtenquelle bestimmt ist. Diese wird als **Entropie H** bezeichnet und in Bit gemessen.
 H bestimmt sich für die hier betrachtete gedächtnislose Quelle mit den Zeichenwahrscheinlichkeiten p_1, p_2, \dots, p_m zu
$$H = - \sum_{i=1}^m p_i \cdot \log_2 p_i$$

Formal kann dann der Begriff der **Code-Redundanz R** als $R = dl - H$ bzw. der relativen Code-Redundanz r als $r = R / dl = 1 - H/dl$ eingeführt werden.
 - dass sich jede Nachrichtenquelle so codieren lässt, dass die Redundanz des Codes beliebig klein wird.
- Komprimierende Codes verkleinern also die Code-Redundanz.



Anmerkungen (2)

- Eine weitere Verkleinerung der mittleren Codewortlänge d_l gegenüber der Huffman- oder der Shannon/Fano-Codierung kann erreicht werden, indem nicht einzelne Zeichen aus A codiert werden, sondern Gruppen von k aufeinander folgenden Zeichen.

(In der Informationstheorie lässt sich zeigen, dass mit wachsendem k die mittlere Codewortlänge bezogen auf ein Zeichen gegen die Entropie konvergiert).



Anwendungsbeispiel

- **Fax-Komprimierung CCITT T4**

- Das Scannen einer Seite führt zu Zeilen aus 1728 einzelnen schwarzen und weißen Punkten (Pixeln) (bei 3,85 oder 7,7 Zeilen/mm (fein)).
- Für jede Bildzeile werden die Längen der Folgen von schwarzen und weißen Pixeln bestimmt (Laufängenbestimmung).
- Die Zahlenfolge wird durch speziellen Huffman-Code komprimiert:

weiß:	Länge	Codewort	schwarz:	Länge	Codewort
	0	00110101		0	0000110111
	1	000111		1	010
	2	0111		2	11
	3	1000		3	10
	4	1011		4	011
	5	1100		5	0011
	6	1110		6	0010
	7	1111		7	00011
	8	10011		8	000101
	9	10100		9	000100
	10	00111		10	0000100
	11	01000		11	0000101
	12	001000		12	0000111
	



Nicht verlustfreie Codierungen

- Nicht verlustfreie Codierungen nehmen einen Informationsverlust in Kauf, um einen höheren Komprimierungsgrad zu erreichen.
- Typische Anwendungsbereiche:
 - Audio-Daten
 - Bilddaten
- Typische Kompressionsverfahren:
 - MPEG-Audio
 - Verlustminimierung auf Basis eines **psycho-akustischen Modells**
 - Lauflängenkodierung
 - Huffman-Codierung der Lauflängenbeschreibung
 - MP3 für Audio-Daten
 - JPEG (Joint Photographic Expert Group) für Bilddaten
 - dem Auge angepasste Bildtransformationen (Informationsverlust)
 - Lauflängenkodierung von quantisierten Werten
 - Huffman-Codierung der Lauflängenbeschreibung
 - JPEG2000: Basis „Wavelet“-Transformation
 - MPEG/MPEG2 Videostrom-Kompression

4.5 Fehlererkennende und -korrigierende Codes

Überblick

1. Einführung
2. Fehlererkennende Codes
3. Fehlerkorrigierende Codes
4. Zyklische Codes zur Fehlererkennung



4.5.1 Einführung

- Bei der Eingabe, Verarbeitung, Übertragung und Speicherung von Informationen können Fehler auftreten, die zu Störungen in der Repräsentierung führen.
- Ein **Bitfehler** eines binären Signals ist seine Umkehrung ($0 \rightarrow 1$, $1 \rightarrow 0$).
 - typisches Maß: Bitfehler-Wahrscheinlichkeit
 - Beispiel: Für ISDN semipermanente Verbindungen wird eine Bitfehler-Wahrscheinlichkeit von 10^{-7} angegeben.
- Störungen führen zur **Verfälschung** von Codewörtern.
- Fehlererkennung ist nur möglich, wenn die durch Bitfehler entstehenden Binärwörter keine gültigen Codewörter sind.
- Bitfehler, die ein Codewort in ein anderes gültiges Codewort verfälschen, sind nicht erkennbar.
- **Codesicherung** beinhaltet alle Maßnahmen der Erkennung oder Korrektur von Bitfehlern in Codewörtern oder Blöcken von Codewörtern.

Def



Beispiele

- bei Eingabe durch den Menschen:
 - vertauschte Zeichen ("Zahlendreher")
 - ausgelassene oder verdoppelte Zeichen
- bei Übertragung:
 - Übertragungsstörung kann zu einzelnen oder mehreren aufeinander folgenden fehlerhaften Bits (**Burst-Fehler** oder *Bündel-Fehler*) führen.
- bei Speicherung:
 - "Umkippen" von einzelnen Datenbits eines Maschinenworts im Hauptspeicher durch fehlerhaften Speicherchip oder Strahlung
 - DRAM: Entladung des Speicherkondensators durch ionisierende Strahlung, insb. Alphateilchen.
 - Mängel in der Magnetisierung einer Plattenoberfläche können zu *Burst-Fehlern* führen.



Redundanz

- Unter **Code-Redundanz** soll im folgenden jeglicher Zusatzaufwand in einem Code verstanden werden, der über die reine Darstellung der gewünschten Codewörter hinausgeht.
- **Beispiele:**
 - Im BCD-Code stellen nur 10 der 16 möglichen Tetraden gültige Codewörter dar.
 - Die gesprochene deutsche Sprache enthält etwa 80% Redundanz.
- Das Vorhandensein von Redundanz kann benutzt werden, um aufgetretene Fehler zu erkennen und evtl. sogar zu korrigieren.
- Zusätzliche Redundanz entsteht z.B., wenn einem Code zusätzliche Bitstellen hinzugefügt werden.
- Kompression und Fehlererkennung konkurrieren miteinander !
Bei der Festlegung von Codierungen ist zu prüfen, inwieweit Redundanz wünschenswert oder erforderlich ist.



Konsequenzen bei der Decodierung

- Je nach Umfang der Störung und der vorhandenen Redundanz sind unterschiedliche Fälle bei der Decodierung möglich:
 - keine Störung \Rightarrow fehlerfreie Decodierung
 - "geringe" Störung \Rightarrow Decodierung der ursprünglichen Nachricht ist möglich, der aufgetretene Fehler wird *maskiert* (d.h. tritt nach außen nicht in Erscheinung).
 - "stärkere" Störung \Rightarrow Decodierung der ursprünglichen Nachricht ist nicht möglich, aber Vorhandensein eines Fehlers wird erkannt.
 - "sehr starke" Störung \Rightarrow Decodierung führt zu einer fehlerhaften ursprünglichen Nachricht (Katastrophe).



Hamming-Gewicht, Hamming-Abstand

Def

- Sei $c:A \rightarrow \{0,1\}^*$ ein binärer Code. Das **Hamming-Gewicht** $g(w)$ eines Codewortes $w \in \{0,1\}^*$ ist die Anzahl der Stellen des Codeworts mit dem Wert "1".
 - Beispiel: $g(01000101) = 3$
- Seien $a, b \in \{0,1\}^n$ zwei n -stellige Codewörter. Der **Hamming-Abstand** oder die **Hamming-Distanz** $h(a,b)$ von a und b gibt die Anzahl der Stellen an, in denen sich die Codewörter a und b unterscheiden.
 - Beispiel: $h(01000101, 00010111) = 3$
- Sei $c:A \rightarrow \{0,1\}^n$ ein binärer Blockcode. Der **Hamming-Abstand des Codes c** ist als der kleinste Hamming-Abstand $h(a,b)$ zwischen zwei verschiedenen Codewörtern a und b definiert.

01000101
00010111



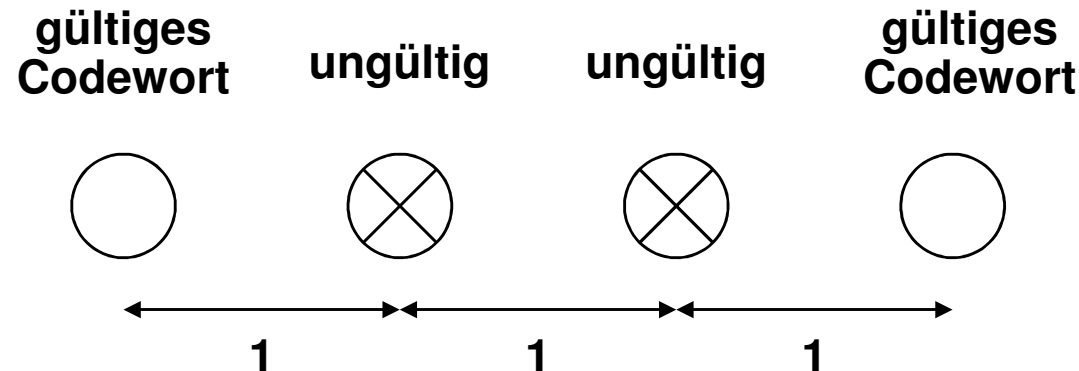
Hamming-Abstand von Codes

- **Beispiele (Hamming-Abstand von Codes):**
 - **ASCII-Code:** 1
 - **jeder dichte Code** 1
 - **BCD-Code** 1 , auch wenn Redundanz vorhanden.
 - **2-aus-5-Code** 2



4.5.2 Fehlererkennende Codes

- **Satz:**
Hat ein Code den Hamming-Abstand d , so können alle Störungen, die höchstens $d-1$ Bits betreffen, sicher erkannt werden.
- **Beispiel: $d = 3$**



- ⇒ Dichte Codes mit Hamming-Abstand 1 können keine Fehler erkennen.
- ⇒ Zur Erkennung von 1-Bit-Fehlern ist mindestens ein Hamming-Abstand von $d=2$ erforderlich.



Beispiel: 2-aus-5-Code

- Hamming-Abstand: $d=2$

	7	4	2	1	0						
0	1	1	0	0	0	2	0	0	1	0	1
1	0	0	0	1	1						↕
2	0	0	1	0	1						
3	0	0	1	1	0	X	0	0	1	0	0
4	0	1	0	0	1						↕
5	0	1	0	1	0						
6	0	1	1	0	0						
7	1	0	0	0	1	3	0	0	1	1	0
8	1	0	0	1	0						
9	1	0	1	0	0						

⇒ 1-Bit-Fehler werden sicher erkannt



Paritätsbit

- Ein ungesicherter Code ($d=1$) kann durch die Hinzunahme eines Prüfbits (**Paritätsbit**, *parity bit*) auf $d=2$ erweitert werden. Diese Erweiterung eines Codewortes wird auch **Querparität** oder **Zeichenparität** genannt.

⇒ 1-Bit-Fehler werden erkannt.
- Alternativen zur Festlegung des Paritätsbits:
 - gerade Parität (*even parity*): Das Codewort wird auf ein gerades Gewicht (gerade Anzahl von 1-Bits) erweitert.
 - ungerade Parität (*odd parity*): Das Codewort wird auf ein ungerades Gewicht (ungerade Anzahl von 1-Bits) erweitert.



Paritätsbit (2)

- Fehlererkennung durch Paritätsprüfung auf Empfängerseite (*parity check*):
 - Bildung der Quersumme modulo 2 über das gesamte Datenwort einschließlich Paritätsbit.
 - Dabei Addition modulo 2 (Restklassenaddition):
 $0+0=0$, $0+1=1$, $1+0=1$, $1+1=0$ (XOR)
- ⇒ Jede ungeradzahlige Anzahl von 1-Bit-Fehlern wird erkannt, keine geradzahlige Anzahl von Bitfehlern wird erkannt.



Beispiele:

- **ASCII-Code (vgl. Kap. 3.4.2)**

- 7-Bit-Zeichen
- MSB zur Speicherung des Paritätsbits
- gerade oder ungerade Parität möglich
- Beispiel (gerade Parität):

▪	Codewort:	"A": 1000001	"W": 1010111
	Paritätsbit:	0	1
	Erweit. Codewort:	01000001	11010111

- **Arbeitsspeicher mit Parität**

- je Byte ein Parity-Bit
- gerade oder ungerade Parität möglich
- bei PCs für den Massenmarkt aus Kostengründen häufig weggelassen, Fehlererkennung als nebensächlich angesehen.



Allgemeine Prüfziffern

- Praxisproblem: Ziffernvertauschungen bei Eingabe können durch einfache Quersummenbildung nicht erkannt werden.
- Abhilfe: Quersummenbildung mit gewichtetem Code
- Beispiel (Internationale ISBN-Buchnummern):



Werner et al:
Taschenbuch der Informatik,
Fachbuchverlag Leipzig

— Prüfziffernbestimmung:

- Wichtung der Stellen von rechts beginnend mit 1, 2, 3, ...
- Prüfziffer: gewichtete Quersumme modulo 11 = 0
- Der mögliche Rest 10 wird codiert durch die Prüfziffer X.

— Probe:

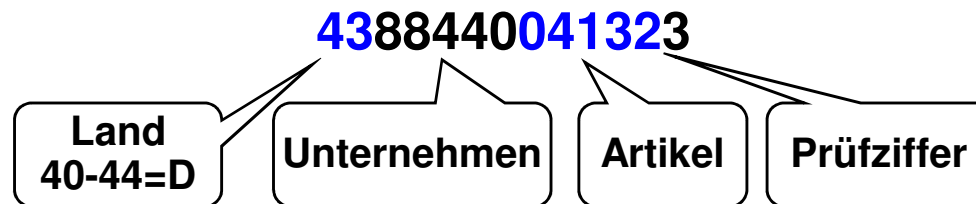
3	3	4	3	0	0	8	9	2	3	
10	9	8	7	6	5	4	3	2	1	Gewichte

$$3 \cdot 10 + 3 \cdot 9 + 4 \cdot 8 + 3 \cdot 7 + 0 \cdot 6 + 0 \cdot 5 + 8 \cdot 4 + 9 \cdot 3 + 2 \cdot 2 + 3 = 176 \quad 176 : 11 = 16 \text{ Rest } 0 \Rightarrow \text{gültig!}$$



Allgemeine Prüfziffern (2)

- Beispiel GTIN (Global Trade Item Number, früher „EAN“):



Quelle: EAN / GS1

— Prüfziffernbestimmung:

- Wichtung der Stellen von rechts beginnend mit 1, 3, 1, 3, 1, ...
- Prüfziffer: gewichtete Quersumme modulo 10 = 0

— Probe:	4	3	8	8	4	4	0	0	4	1	3	2	3	
	1	3	1	3	1	3	1	3	1	3	1	3	1	Gewichte
	4	9	8	4	4	2	0	0	4	3	3	6	3	Produkte modulo 10

$$4 + 9 + 8 + 4 + 4 + 2 + 0 + 0 + 4 + 3 + 3 + 6 + 3 = 50. \quad 50:10 = 5 \text{ Rest } 0 \Rightarrow \text{gültig!}$$



Codewort-Verdopplung

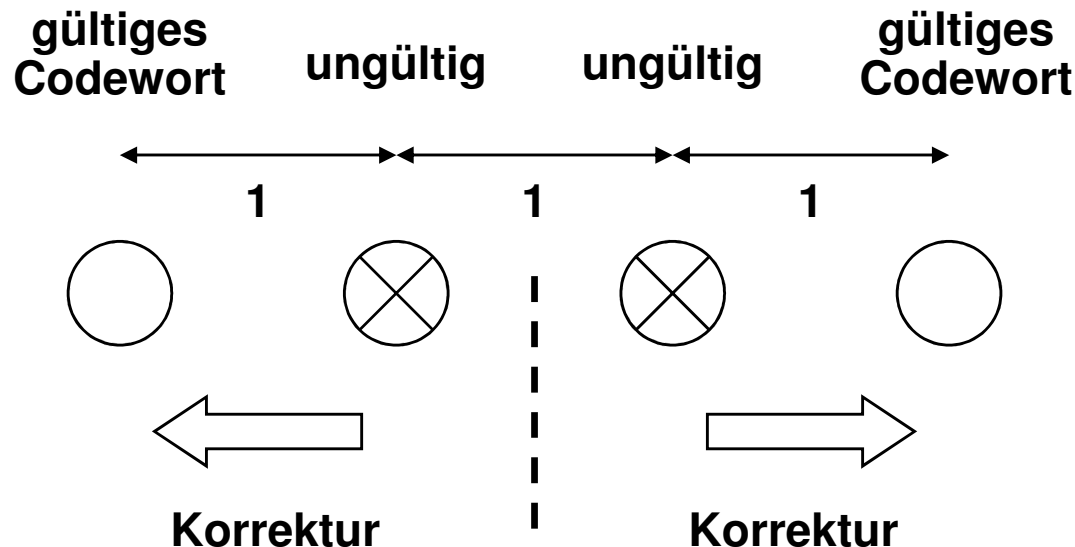
- Ein Code mit Hamming-Abstand d wird durch Verdoppeln der Codewörter ($w \rightarrow w||w$) zu einem Code mit Hamming-Abstand $2 \cdot d$.
 \Rightarrow Für $d=1$ werden damit 1-Bit-Fehler erkannt.
- Bemerkungen:
 - Anwendungsbeispiel: Wiederholung von Zahlen in Telegrammen.
 - In Rechensystemen relativ unüblich, bei der Verarbeitung von Information als Zeitredundanz (zweimalige Nacheinanderausführung) vorkommend.
 - Einfach, aber u.U. verschwenderisch



4.5.3 Fehlerkorrigierende Codes

- **Satz:**
Hat ein Code den Hamming-Abstand $d = 2 \cdot k + 1$, so können alle Störungen, die höchstens k Bits betreffen, sicher korrigiert werden.

- **Beispiel:**
 $d = 3 \Rightarrow k = 1$



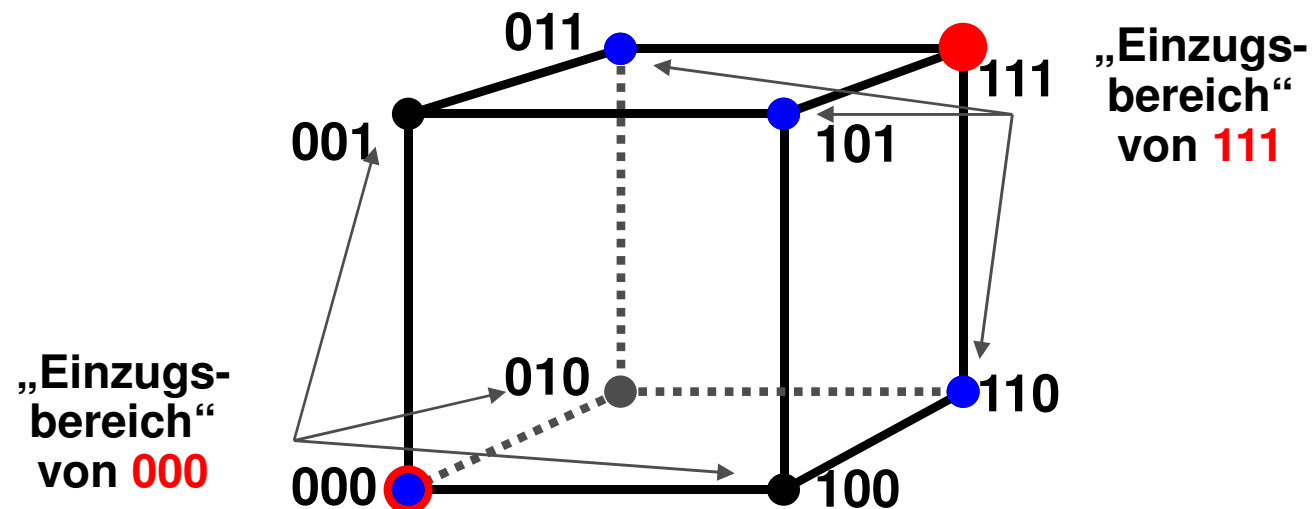
\Rightarrow Zur Korrektur von 1-Bit-Fehlern ist ein Hamming-Abstand von $d=3$ notwendig.

- Der Hamming-Abstand $2 \cdot k + 1$ ist minimal zur Korrektur von k -Bit-Fehlern.



Fehlerkorrigierende Codes (2)

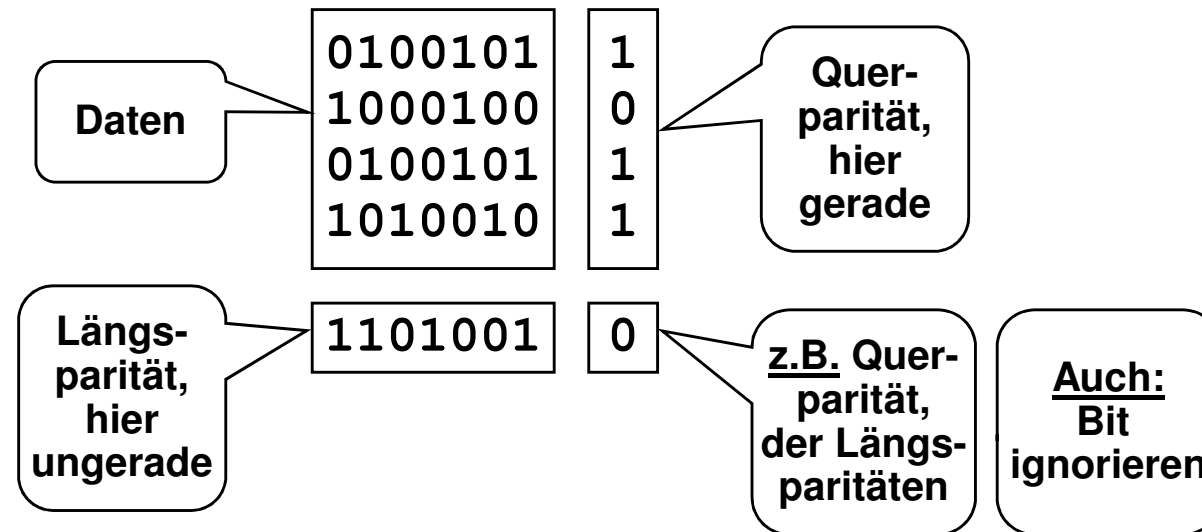
- Visualisierung am n-Würfel
 - Beispiel: Die Code-Wörter des Code $c:A \rightarrow \{0,1\}^n$ für $n=3$ entsprechen den Ecken eines Würfels.
 - Liegt zwischen je zwei Codewörtern (**blau**) jeweils mindestens eine „ungenutzte“ Ecke, so gilt $d=2$ und $(d-1=)$ 1-Bit-Fehler lassen sich erkennen.
 - Liegen mindestens zwei ungenutzte Ecken zwischen den Codewörtern (**rot**), gilt $d=3 = 2 \cdot k + 1$ für $k=1$, also: 1-Bit-Fehler lassen sich korrigieren:





Binärer Rechteck-Code

- Rechteck-Code:
 - Binärer Block-Code als Ausgangsbasis
 - Paritätsbit je Codewort (Querparität) wie bisher
 - Paritätsbit je Spalte für einen Block von Codewörtern (z.B. 16 oder 64) (Längsparität)



- Hamming-Abstand für einen Codewort-Block wird auf 3 erhöht.
⇒ 1-Bit-Fehler für den Codewort-Block können korrigiert werden.



Beispiel

① Gesendet werde:

0100101	1
1000100	0
0100101	1
1010010	1
1101001	0

② Empfangen werde:

0100101	1
1000100	0
0110101	1
1010010	1
1101001	0

③ Kontrollbestimmung der Quer- und Längsparität:

0100101	1
1000100	0
0110101	0
1010010	1
1111001	1

verschiedener Wert

verschiedener Wert

③ Vergleich: gesendet - empfangen

④ Rückschluss auf verfälschte Bitstelle und Korrektur:

0100101	1
1000100	0
0110101	0
1010010	1
1111001	1

verfälschte Stelle: 1→0



Beispiel (2)

① Gesendet werde:

0100101	1
1000100	0
0100101	1
1010010	1
1101001	0

② Empfangen werde:

0100101	1
1000100	0
0100101	0
1010010	1
1101001	0

③ Kontrollbestimmung der Quer- und Längsparität:

0100101	1
1000100	0
0100101	1
1010010	1
1101001	0

verschiedener Wert

Alle Längsparitäten stimmen!

③ Vergleich: gesendet - empfangen

④ Rückschluss auf verfälschte Bitstelle und Korrektur:

0100101	1
1000100	0
0100101	0
1010010	1
1101001	0

verfälschte Stelle: 0→1



Beispiel (3)

① Gesendet werde:

0100101	1
1000100	0
0100101	1
1010010	1
1101001	0

② Empfangen werde:

0100101	1
1000100	0
0100101	1
1010010	1
1101011	0

③ Kontrollbestimmung der Quer- und Längsparität:

0100101	1
1000100	0
0100101	1
1010010	1
1101001	1

Alle Querparitäten stimmen

Verschiedene Werte

③ Vergleich: gesendet - empfangen

④ Rückschluss auf verfälschte Bitstelle und Korrektur:

0100101	1
1000100	0
0100101	1
1010010	1
1101011	0

verfälschte Stelle: 1→0



Lineare Codes, systematische Codes

Def

- Es seien ausschließlich Binär-Codes betrachtet. Ein **linearer** Code oder **Gruppencode** ist ein Blockcode der Länge n , der 2^m , $m \leq n$, Codewörter besitzt.
- Lineare Algebra als Math. Grundlage
 - hier naiverer Zugang
- **Systematische lineare Codes** sind solche, bei denen jede der n Codewortstellen eindeutig als eine der m **Informationsstellen** oder als eine der $r := n - m$ **Prüfstellen** identifiziert werden kann.
- Systematische lineare Codes werden auch als **(n, m, d) -Codes** mit d als Hamming-Distanz des Codes bezeichnet.



Hamming-Code

Def

- Ein **Hamming-Code** ist ein systematischer linearer Code, der k Bit-Fehler bei minimaler Redundanz korrigieren kann. Im engeren Sinne sind Hamming-Codes solche, die 1-Bit-Fehler korrigieren können.
- Grundidee:
 - Den m Datenbits eines Codeworts werden r Prüfbits zugeordnet.
 \Rightarrow Codewortlänge: $n=m+r$ Bits. Bedingung für r : $2^{r-1} < m+r < 2^r$
 - Prüfbits werden an den Positionen $1=2^0, 2=2^1, 4=2^2, \dots, 2^{r-1}$ angenommen, Datenbits dazwischen und danach.
 - Jedes Prüfbit enthält damit in der dualen Darstellung seiner Position genau eine 1.
 - Jedes Prüfbit ist das Paritätsbit (gerade Parität, Addition modulo 2) für alle Datenbits, deren Position an dieser Stelle eine 1 besitzt.
- Vorteil:
 - Aufwand für Prüfbits wächst nur logarithmisch!



Details am Beispiel

- **(7,4,3)-Hamming Code**
 - 4 Daten- und 3 Prüfbits, Hamming-Abstand = 3
 - betrachtetes Datenwort sei 1011

P	P		P			
001	010	011	100	101	110	111
		1		0	1	1

- **Bestimmung der Prüfbits**
 - $P_{100} = D_{101} + D_{110} + D_{111} = 0+1+1 = 0$
 - $P_{010} = D_{011} + D_{110} + D_{111} = 1+1+1 = 1$
 - $P_{001} = D_{011} + D_{101} + D_{111} = 1+0+1 = 0$

P	P		P			
001	010	011	100	101	110	111
0	1	1	0	0	1	1



Details am Beispiel (2)

- Störung führe zu verfälschtem Codewort (Fehler in Datenbit)
 - 1-Bit-Fehler an Stelle 110

P	P		P			
001	010	011	100	101	110	111
0	1	1	0	0	0	1
					1	

- Bestimmung des **Fehlersyndroms** aus Prüfbit und zugehörigen Datenbits (Quersumme modulo 2) nach Auslesen/Übertragung
 - $S_{100} = P_{100} + D_{101} + D_{110} + D_{111} = 0 + 0 + 0 + 1 = 1$
 - $S_{010} = P_{010} + D_{011} + D_{110} + D_{111} = 1 + 1 + 0 + 1 = 1$
 - $S_{001} = P_{001} + D_{011} + D_{101} + D_{111} = 0 + 1 + 0 + 1 = 0$
- Entscheidung:
 - Ist das Fehlersyndrom $(S_{100}S_{010}S_{001})=000$, liegt kein Fehler vor.
 - Andernfalls gibt $(S_{100}S_{010}S_{001})_2$ als Dualzahl die verfälschte Stelle an.
 - Hier: 110_2 ist die verfälschte Stelle.



Details am Beispiel (3)

- Störung führe zu verfälschtem Codewort (Fehler in Prüfbit)
 - 1-Bit-Fehler an Stelle 010

P	P		P			
001	010	011	100	101	110	111
0	0	1	0	0	1	1
	1					

- Bestimmung des *Fehlersyndroms*
 - $S_{100} = P_{100} + D_{101} + D_{110} + D_{111} = 0 + 0 + 1 + 1 = 0$
 - $S_{010} = P_{010} + D_{011} + D_{110} + D_{111} = 0 + 1 + 1 + 1 = 1$
 - $S_{001} = P_{001} + D_{011} + D_{101} + D_{111} = 0 + 1 + 0 + 1 = 0$
- Entscheidung:
 - Gleiches Verfahren.
 - 010_2 ist die verfälschte Stelle.



Optimalität des Hamming-Codes

- **Satz:** Für einen Block-Code mit m Datenbits und r Prüfbits und Hamming-Distanz $d \geq 3$ gilt: $m+r+1 \leq 2^r$.
- Da der Hamming-Code die Bedingung erfüllt ($m+r < 2^r$), stellt er einen optimalen Code mit minimal notwendiger Anzahl von Prüfbits für die Korrektur von 1-Bit-Fehlern dar.
- Satz ist einsichtig, denn:
 - Codewortlänge: $n=m+r$ Bits
 - 2^m Datenwörter, eingebettet in 2^{m+r} Codewörter
 - Jedes Datenwort hat $m+r$ Nachbarn mit Hamming-Abstand 1. Keiner dieser Nachbarn kann Nachbar eines anderen Datenworts sein (sonst gäbe es Codeworte mit Hamming-Distanz kleiner als 3).
 - Also belegt jedes Datenwort mit seinen Nachbarn $m+r+1$ Codeworte.
 - Also insgesamt für die Anzahl der Datenwörter:
 $2^m (m+r+1) \leq 2^{m+r}$ und damit $m+r+1 \leq 2^r$



SEC/DED-Codes

- **SEC/DED: Single Error Correction / Double Error Detection**
- **Erweiterung von Hamming-Codes auf Hamming-Distanz $d=4$ durch weiteres Paritätsbit P über alle Stellen.**
- ⇒ **Korrektur aller 1-Bit-Fehler und Erkennung aller 2-Bit-Fehler.**
- **Unterscheidung in der Decodierung**
 - **Syndrom und P korrekt ⇒ kein Fehler**
 - **Syndrom und P falsch ⇒ 1-Bit-Fehler korrigierbar**
 - **Syndrom falsch und P korrekt ⇒ 2-Bit-Fehler erkannt, nicht korrigieren.**
- **Anwendung:**
 - **Basis für betriebssichere Arbeitsspeicher von Rechensystemen**



Maintenance (M)-Codes

- Varianten von SEC/DED-Hamming-Codes
- zusätzliche Ziele:
 - möglichst geringer technischer Aufwand, z.B. möglichst wenige Stellen nehmen an der Quersummenprüfung teil.
 - Überprüfung der korrekten Funktion der Erzeugung der Prüfbits.
 - Erweiterbarkeit auf verschiedene Maschinenwortlängen (Kaskadierbarkeit der Logik-Schaltkreise).
- Basis des käuflichen **ECC Memory** (Error Correcting Code) für hochverfügbare Rechensysteme
- Typische M-Codes (n,m,d) für die üblichen Maschinenwortlängen:
 - (13,8,4)
 - (22,16,4)
 - (39,32,4)
 - (72,64,4)

Prüfen Sie die
Bedingung
 $2^{r-1} < m+r < 2^r$
an diesen Werten nach!

Überführung von k-Bit-Fehler in 1-Bit-Fehler

- Hamming-Codes sind unbrauchbar zur Behandlung von Burst-Fehlern, wie sie z.B. bei der Datenübertragung auftreten.
- Idee: Vertauschung der Übertragungsreihenfolge
 - k Worte werden mit Hamming-Codierung für 1-Bit-Fehler-Korrektur zeilenweise in eine Tabelle geschrieben.
 - Die Tabelle wird spaltenweise gesendet.
 - Tritt während der Übertragung ein k-Burst-Fehler auf (der k aufeinander folgende Bits betrifft), so liegt in jeder Zeile nur ein 1-Bit-Fehler vor.
 - Aufgrund des Hamming-Codes ist eine vollständige Korrektur aller k Worte beim Empfänger möglich.



4.5.4 Zyklische Codes

- Die in 4.5.3 besprochenen Codes erkennen und korrigieren Einzelfehler oder eine geringe Anzahl von Fehlern (z.B. SEC/DED).
- In bestimmten Anwendungsbereichen, wie
 - Datenübertragung durch serielle Bitströme (z.B. Modem, LAN),
 - serielle Datenspeicherung auf magnetisierten Oberflächen (z.B. Diskette, Festplatte),

treten im Falle einer Störung aber Burst-Fehler, d.h. Folgen verfälschter Bitstellen, auf.

- Zur Sicherung gegenüber solchen Fehlern werden **zyklische Codes** eingesetzt (auch **Polynom-Codes** genannt).
- Zyklische Codes sind die Basis der zyklischen Blocksicherung **CRC (Cyclic Redundancy Check)**.
- Einfache technische Realisierung mittels Schieberegisterschaltungen (vgl. LV Digitaltechnik) möglich.



Grundlage: Polynomdivision

- Die n Bits eines Datenblocks B der Länge n werden als Koeffizienten eines Polynoms $P(x)$ vom Grad $\leq n-1$ interpretiert.

- Beispiel:

$$B=1011 \quad P(x) = 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0 = x^3 + x + 1$$

- Polynom-Addition modulo 2

- Addition der Koeffizienten gleicher Exponenten modulo 2
- Rechenregeln: $0+0=0$, $0+1=1$, $1+0=1$, $1+1=0$ (XOR)
- Subtraktion entspricht hier Addition (!)

- Beispiel:

$$B=1011 \quad P(x) = 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0 = x^3 + x + 1$$

$$C=1101 \quad Q(x) = 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 = x^3 + x^2 + 1$$

$$\Sigma = 0110$$

$$P(x) + Q(x) = 0 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x^1 + 0 \cdot x^0 = x^2 + x$$



Grundlage: Polynomdivision (2)

- Polynom-Division modulo 2
 - $P(x) = D(x) * Q(x) + R(x)$
 - Restpolynom $R(x)$ besitzt einen Grad, der kleiner als der von $Q(x)$ ist
 - Für CRC interessiert nur $R(x)$, nicht $D(x)$

- Beispiel:

$$P(x) = 1 * x^5 + 0 * x^4 + 1 * x^3 + 1 * x^2 + 0 * x^1 + 1 * x^0 = x^5 + x^3 + x^2 + 1$$

$$Q(x) = 1 * x^3 + 1 * x^2 + 0 * x^1 + 1 * x^0 = x^3 + x^2 + 1$$

$$\begin{array}{r}
 P(x) : Q(x) = (x^5 + x^3 + x^2 + 1) : (x^3 + x^2 + 1) = x^2 + x \\
 \begin{array}{r}
 x^5 + x^4 + x^2 \\
 \hline
 x^4 + x^3 \\
 x^4 + x^3 + x \\
 \hline
 x + 1
 \end{array}
 \end{array}$$

$$\Rightarrow D(x) = x^2 + x, \quad R(x) = x + 1$$



Vorgehensweise

- Ein zyklischer Code mit m Datenbits und r CRC-Prüfbits wird durch ein **Generator-Polynom** $G(x)$ vom Grad r festgelegt.
- Codierung:
 - Der Nutzdatenblock definiert ein Polynom $M(x)$ vom Grad $\leq m-1$
 - An die Nutzinformation werden r Nullbits angehängt. Die Nachricht einschließlich CRC-Feld ist dann $n=m+r$ Bits lang und entspricht dem Polynom $x^r * M(x)$.
 - Dieses Polynom wird durch das Generator-Polynom $G(x)$ dividiert. Es entsteht ein Restpolynom $R(x)$ vom Grad $\leq r-1$, das die Gleichung
$$x^r * M(x) = D(x) * G(x) + R(x) \quad \text{erfüllt.}$$
 - Die Koeffizienten von $R(x)$ werden in das CRC-Feld eingetragen. Das gesendete Codewort aus Nutzdatenblock und CRC-Feld entspricht damit dem Polynom $P(x) = x^r * M(x) + R(x)$.
 - $P(x)$ ist nach Konstruktion durch $G(x)$ ohne Rest teilbar!
(Beachte: $x^r * M(x) + R(x) = x^r * M(x) - R(x)$ wg. Addition modulo 2).



Vorgehensweise (2)

- **Überprüfung/Fehlererkennung:**
 - Auf Empfängerseite wird $P(x)$ wieder durch $G(x)$ mit Rest $R'(x)$ dividiert.
 - Fehlerfreiheit \Leftrightarrow Restpolynom $R'(x)=0$



Beispiel

- Generator-Polynom $G(x)=x^3+1$ vom Grad $r=3$
- Datenblock mit $m=4$: 0110, d.h. $M(x)=x^2+x$

Daten				CRC		
0	1	1	0			

- $x^r * M(x) = x^5+x^4$
- $x^r * M(x) : G(x)$

$$\begin{array}{r}
 (x^5+x^4) : (x^3+1) = x^2+x \\
 \begin{array}{r}
 x^5 \qquad \qquad +x^2 \\
 \hline
 x^4 \qquad \qquad +x^2 \\
 x^4 \qquad \qquad +x \\
 \hline
 \qquad \qquad x^2+x
 \end{array}
 \end{array}$$

$$\Rightarrow R(x) = x^2+x$$

Daten				CRC		
0	1	1	0	1	1	0

versendet



Beispiel (2)

- Überprüfung beim Empfänger: (a) ohne Verfälschung

Daten				CRC		
0	1	1	0	1	1	0

- $P(x) : G(x)$

$$\begin{array}{r} (x^5 + x^4 + x^2 + x) : (x^3 + 1) = x^2 + x \\ \underline{x^5 + x^2} \\ x^4 + x \\ \underline{x^4 + x} \\ 0 \end{array}$$

- $R(x) = 0 \Rightarrow$ Daten unverfälscht.



- | Daten | | | | CRC | | |
|-------|---|---|---|-----|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| | | 1 | | | | |

- $$\begin{array}{r} \mathbf{x^5} \qquad \qquad \mathbf{+x^2+x} \qquad \qquad \mathbf{:} \quad \mathbf{x^3} \qquad \qquad \mathbf{+1} \quad \mathbf{=} \quad \mathbf{x^2} \\ \mathbf{x^5} \qquad \qquad \mathbf{+x^2} \\ \hline \qquad \qquad \qquad \mathbf{x} \end{array}$$

- 4 - 88



Anmerkungen

- **Erkannte Fehler:**
 - Sei $F(x)$ ein Fehlerpolynom, und $P'(x) = P(x) + F(x)$ werde empfangen. Der zyklische Code mit Generator-Polynom $G(x)$ erkennt einen Fehler genau dann, wenn $G(x)$ das Fehlerwort $F(x)$ nicht ohne Rest teilt.
 - Wenn der Grad des Fehlerpolynoms kleiner ist als der Grad r des erzeugenden Polynoms $G(x)$, ist $G(x)$ kein Teiler von $F(x)$.
- ⇒ Eine beliebige Störung im CRC-Feld wird erkannt, da der Grad des Restpolynoms $R(x) \leq r-1$ ist.
- ⇒ Bei einem r -Bit-CRC werden $(r-1)$ -Burst-Fehler immer erkannt, längere Fehler können bei geeigneter Wahl des Generator-Polynoms mit hoher Wahrscheinlichkeit erkannt werden.



Standardisierte Generator-Polynome

- **CRC-16:** $x^{16}+x^{15}+x^2+1$
- **CRC-CCITT:** $x^{16}+x^{12}+x^5+1$ (auch ISO 2111, ISO 3309)
- **CRC-32:** $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$
(Ethernet)