Algorithmen und Datenstrukturen

Teil 14: Codierung

DHBW Stuttgart Campus Horb
Fakultät Technik
Studiengang Informatik
Dozent: Olaf Herden
Stand: 06/2023

Gliederung

- Grundbegriffe
- Code-Erzeugung
- Code-Sicherung
- Lineare Codes
- Nicht-binäre Codes

Definitionen (I)

- Seien $A = \{a_1, \ldots, an\}$ Menge elementarer Zeichen
- A heißt Alphabet
- Aneinanderreihung von Zeichen aus Alphabet A heißen Wort w
- A^* := Menge aller Wörter über A
- Beispiel:
  - Sei $A = \{a, b, c\}$
  - Beispiele für Wörter über A: $a, b, abba, ccbbaa, \ldots$
  - Gegenbeispiele (keine Wörter über A): $ae, xa, \ldots$
  - $A* = \{a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc,$
    $aaa, aab, aac, aba, abb, abc, aca, acb, acc, \ldots\}$
- Anzahl der Zeichen eines Wortes w: Länge von w, Notation $|w|$

Definitionen (II)

- Spezialfall:
  - $A = \{0,1\}$
  - Worte heißen Binärwort
  - A^* Menge der Binärwörter
- Gegeben sei ein Binärwort $b_{n-1}b_{n-2}$ ... b_0 der Länge n
- Dann:
  - b_{n-1} MSB (Most significant bit)
  - b_0 LSB (Least significant bit)

Definitionen (III)

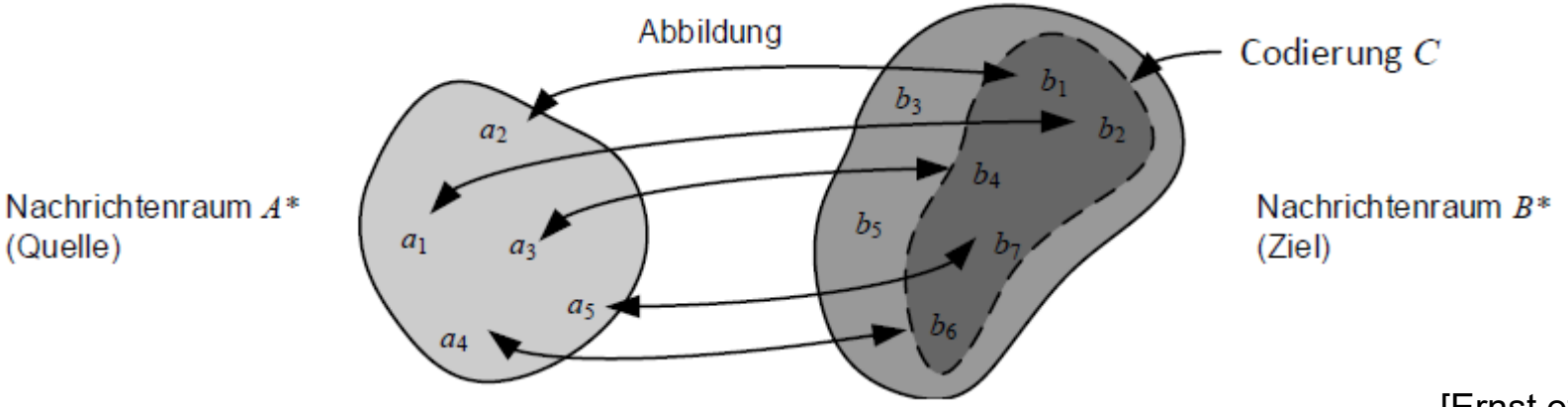
- Entropie: Mittlerer Informationsgehalt pro Zeichen einer Quelle
- Formaler:
  - Gegeben Quelle $X = \{x_1, \ldots, x_n\}$, Wahrscheinlichkeit $p(x)$ für alle $x \in X$
  - Entropie H (eigentlich großes Eta) definiert als

$$H(X) = \sum_{i=1}^{n} p(x_i) \cdot log_2 \left(\frac{1}{p(x_i)}\right) = -\sum_{i=1}^{n} p(x_i) \cdot log_2 \left(p(x_i)\right)$$

  - Dabei gilt: $0 \cdot log_2 0 := 0$ und $a \cdot log_2 \left(\frac{a}{0}\right) := \infty$
- Anmerkungen:
  - Es gilt immer $H(X) \geq 0$
  - Entropie wird maximal, wenn alle Zeichen gleich wahrscheinlich $(H_0 = \log_2(n))$
  - Entropie deutsche Sprache ca. $4{,}1$
  - Maximaler Wert bei 26 Buchstaben $\log_2(26) \approx 4{,}7$

Codierung

- Gegeben:
  - Nachrichtenraum A^* über Alphabet $A = \{a_1, \ldots, an\}$ (Quelle)
  - Nachrichtenraum B^* über Alphabet $B = \{b_1, \ldots, b_m\}$ (Ziel)
- Codierung C ist bijektive Abbildung $A^* \rightarrow B^*$
- Beachte: $C \subseteq B^*$

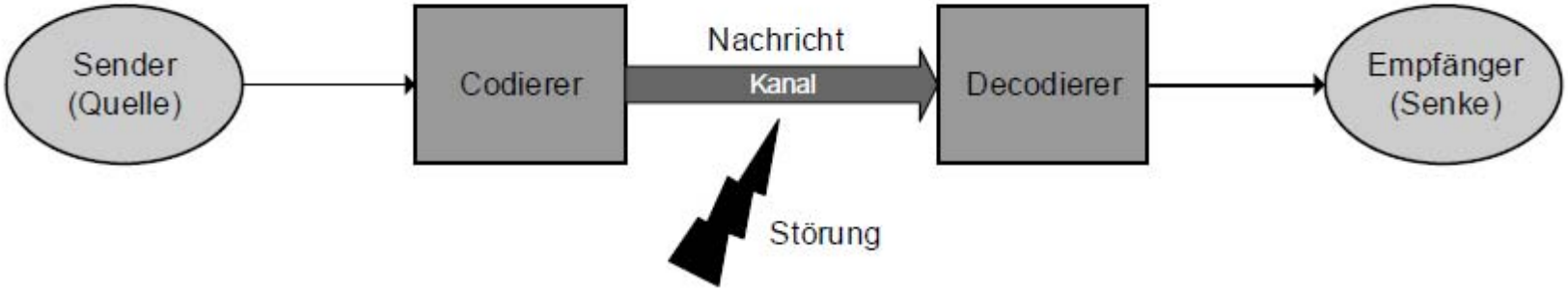


[Ernst et. al. 2016]

- Ist $B = \{0,1\}$, dann heißt C Binärcodierung

Übertragung/Anforderungen

- Übertragung:



[Ernst et. al. 2016]

- Anforderungen an Codierung:
  - Möglichst kompakte Darstellung
  - Unempfindlich gegen Störungen
  - Maschinell leicht zu verarbeiten

Mittlere Wortlänge

- Def.: Mittlere Wortlänge L
- Gegeben:
  - Zeichen $X = \{x_1, \ldots, x_n\}$
  - l_i Wortlänge von Zeichen x_i
  - p_i Auftretenswahrscheinlichkeit von Zeichen x_i
- Dann:

$$L = \sum_{i=1}^{n} p_i \cdot l_i$$

- Shannonsches Codierungstheorem:

  $H \leq L$ („Entropie ist immer kleiner gleich der mittleren Wortlänge“)
- Begründung: Entropie maximal, wenn alle Zeichen gleich wahrscheinlich

Code-Redundanz

- Redundanz R eines Codes ist definiert als

$$R = L - H \; [Bit \; pro \; Zeichen]$$

- Anschaulich: Anteil einer Nachricht, der im statistischen Sinn keine Information enthält
- Geringe Redundanz:
  - Schnelle Übertragung
  - Platzsparend
- Andererseits:
  - Redundanz kann Störsicherheit bringen
  - Fehlererkennung, Fehlerbehebung

Quellen-Redundanz

- Quellen-Redundanz R_Q ist definiert als

$$R_Q = H_0 - H \; [Bit \; pro \; Zeichen]$$

- Anschaulich: Abweichung mittlerer Informationsgehalt von Q vom maximal möglichen Informationsgehalt bei gleichem Alphabet, aber anderer Auftretenswahrscheinlichkeit der Zeichen

Beispiel (I): BCD-Code (I)

- BCD: Binary Coded Decimal
- Dezimalsystem nach BCD:
  - Dezimalziffer wird zu 4 Binärzeichen (Tetradencode)
  - Umwandlung jeder Dezimalziffer separat
- Beispiel: Erste 15 Zahlen

Dezimal	Binär	BCD
0	0000	0000 0000
1	0001	0000 0001
2	0010	0000 0010
3	0011	0000 0011
4	0100	0000 0100
5	0101	0000 0101
6	0110	0000 0110
7	0111	0000 0111

Dezimal	Binär	BCD
8	1000	0000 1000
9	1001	0000 1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

- Pseudotetraden: In BCD-Code nicht verwendete Tetraden (1010 bis 1111)

Beispiel (II): BCD-Code (II)

- Beispiele:
  - 745 -> 0111 0100 0101
  - 63,25 -> 0110 0011, 0010 0101
  - 0,1 -> 0000, 0001
- Bewertung:
  - Vorteile:
    - Anwendungen mit Rechengenauigkeit
    - Stellengenaue Ergebnisse möglich
    - Binärsystem hingegen (z.B. 0,1 nur näherungsweise darstellbar)
  - Nachteile:
    - Mehr Platzbedarf
    - Geringere Rechengeschwindigkeit

- Begriffe (nicht nur bei BCD-Code):
  - Nutzwort: Code, der tatsächlich genutzt wird
  - Fehlerwort: Code, der nicht genutzt wird (hier: Pseudotetraden)

Beispiel (III): ASCII-Code (I)

- ASCII: American Standard Code for Information Interchange
- Erste Standardisierung 1968: ANSI X3.4 bzw. ISO 8859/1.2
- Enthält alle Buchstaben, Ziffern und Steuerzeichen
- Ursprünglich 7-Bit-Code mit 128 Zeichen

BIT 5–7 1–4	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0/ 0000	NUL	DLE	SP	0	@	P	‘	p
1/ 0001	SOH	DC1	!	1	A	Q	a	q
2/ 0010	STX	DC2	“	2	B	R	b	r
3/ 0011	ETX	DC3	#	3	C	S	c	s
4/ 0100	EOT	DC4	$	4	D	T	d	t
5/ 0101	ENQ	NAK	%	5	E	U	e	u
6/ 0110	ACK	SYN	&	6	F	V	f	v
7/ 0111	BEL	ETB	’	7	G	W	g	w
8/ 1000	BS	CAN	(	8	H	X	h	x
9/ 1001	HT	EM	)	9	I	Y	i	y
A/ 1010	LF	SUB	*	:	J	Z	j	z
B/ 1011	VT	ESC	+	;	K	[	k	{
C/ 1100	FF	FS	,	<	L	\	l	|
D/ 1101	CR	GS	–	=	M	]	m	}
E/ 1110	SO	RS	.	>	N	^	n	~
F/ 1111	SI	US	/	?	O	_	o	DEL

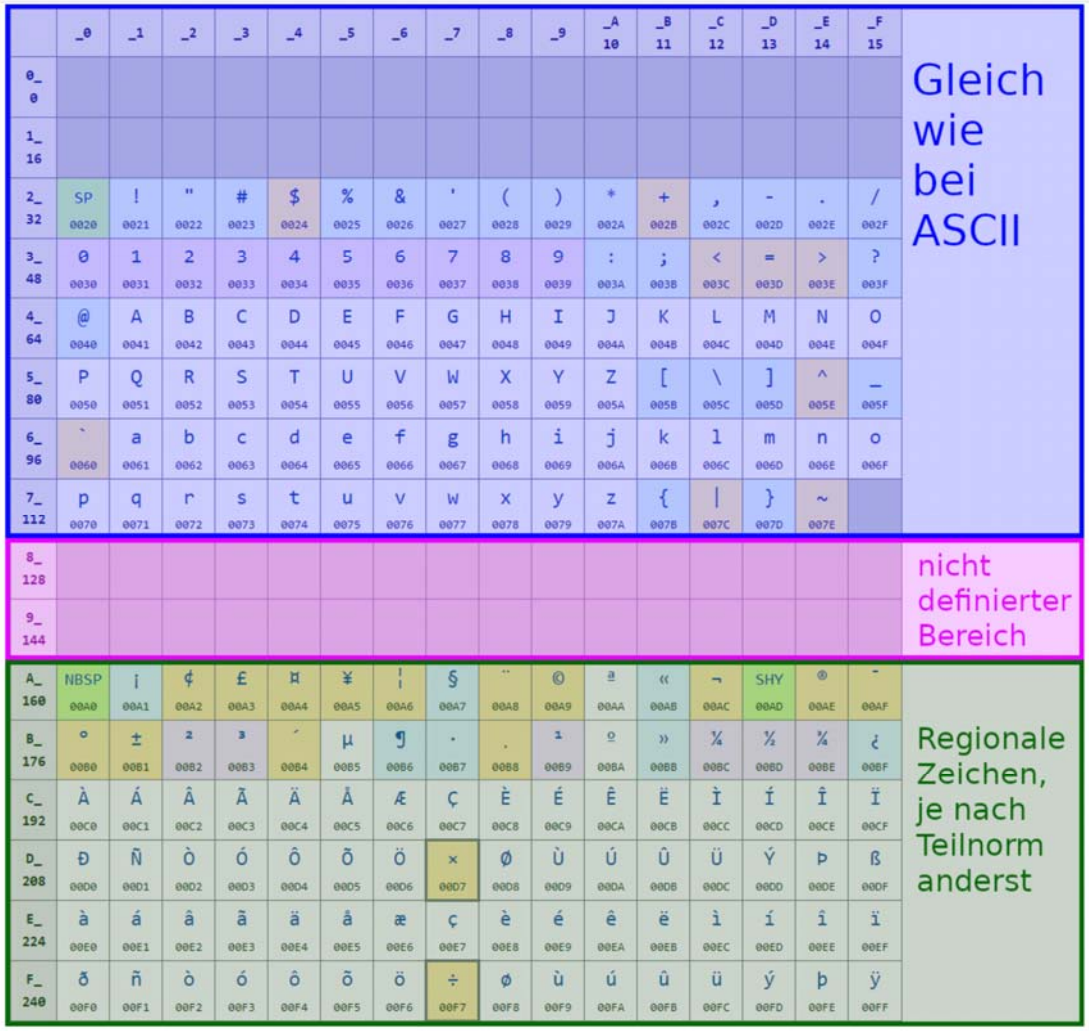
[https://www.cfd.tu-berlin.de/Lehre/EDV1/skripte/alles/node77.html]

ASCII-Code (II)

- (De)codierung erfolgt durch „Ablesen“ in Tabelle
- Beispiele:
  - 100 0111 -> G
  - ? -> 011 1111
- Nachteil: Wesentliche Zeichen europäischer Sprachen fehlen
- Daher:
  - Nutzung achtes Bit
  - Neben Sonderzeichen auch mathematische Zeichen und grafische Zeichen
- Trotz allem: Vielzahl an (verschiedenen) Sonderzeichen nicht handhabbar
- Ausweg ISO 8859:
  - 8 Bit ASCII Zeichensatz
  - Erste 128 Zeichen gleich (entsprechen 7 Bit ASCII)
  - Zweite 128 Zeichen variabel zur Anpassung an Sprachraum

ASCII-Code (III)

- Aufbau und normierte ISO 8859 Zeichensätze:



ISO 8859	
-1	*Latin-1*, Westeuropäisch
-2	*Latin-2*, Mitteleuropäisch
-3	*Latin-3*, Südeuropäisch
-4	*Latin-4*, Nordeuropäisch
-5	Kyrillisch
-6	Arabisch
-7	Griechisch
-8	Hebräisch
-9	*Latin-5*, Türkisch
-10	*Latin-6*, Nordisch
-11	Thai
~~-12~~	(existiert nicht)
-13	*Latin-7*, Baltisch
-14	*Latin-8*, Keltisch
-15	*Latin-9*, Westeuropäisch
-16	*Latin-10*, Südosteuropäisch

  - Für Deutschland: ISO-8859-1 (Latin-1)

[https://www.marcelwicki-softwaredesign.ch/2018/08/08/zeichencodierungen/]

Unicode (I)

- Idee: Alle (Sonder-)Zeichen aller Sprachen in einem Code
- Erweiterung von 8 auf 32 Bit
- D.h. $2^{32} \approx 4{,}29$ Milliarden Zeichen
- Bezeichnung eines 16 Bit codierten Unicodezeichens:
  - U+XXXX
  - Jedes X hexadezimale Ziffer
- Kompatibilität zu ASCII-Code:
  - ASCII: U+0000 bis U+007F
  - ISO 8859-1: U+0080 bis U+00FF

Unicode (II)

- Unicode weist Zeichen Code zu
- Keine Aussage über Länge der Codierung
- UTF: Unicode Transformation Format
  - UTF-32:
    - Immer 32 Bit
    - Sehr speicherplatzintensiv
  - UTF-16 (Windows, OS-X):
    - Für „gebräuchliche“ Zeichen 2 Byte pro Zeichen
    - 4 Byte für „exoterische“ Zeichen
  - UTF-8 (www, E-Mail, Unix):
    - Zwischen einem und vier Byte pro Zeichen
    - Vorteil:
      - 1 Byte-Codierung entspricht 7 Bit ASCII
      - Sehr effizient bei lateinischen Zeichen (gegenüber 2 Byte in UTF-16)
      - Bei anderen Schriften typischerweise 3 Byte (gegenüber 2 Byte in UTF-16)

Übersicht

- Grundbegriffe
- Code-Erzeugung
- Code-Sicherung
- Lineare Codes
- Nicht-binäre Codes

Codeerzeugungsproblem

- Gegeben:
  - Alphabet mit Zeichen
  - Auftretenshäufigkeiten der Zeichen
- Gesucht:
  - Code mit minimaler mittlerer Codewortlänge

- Siehe Kapitel 4:
  - Huffman-Algorithmus
  - Kurze Wiederholung (wenn nicht mehr bekannt):
    - Präfixcode
    - (Minimale) mittlere Codewortlänge
    - Ablauf Algorithmus

Algorithmus von Shannon und Fano

- Gegeben:
  - Menge von Zeichen mit Auftretenshäufigkeiten
- Vorgehen:
  - Sortiere Zeichen nach ihrer Häufigkeit
  - Teile Zeichen in zwei Gruppen, so dass Summe der Häufigkeiten in Teilgruppen möglichst gleich
  - Erweitere Code einer Gruppe um 0, der anderen um 1 (bzw. trage Gruppen in Binärbaum ein)
  - Enthält Teilgruppe mehr als 1 Zeichen, fahre rekursiv fort

Beispiel (I)

- Gegeben:

Zeichen	A	N	B	T	E	H
Häufigkeit	16	20	8	6	42	8

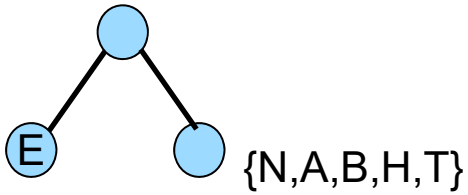
- Ordne nach Häufigkeit:

Zeichen	Häufigkeit	
E	42	0
N	20	1
A	16	1
B	8	1
H	8	1
T	6	1

Summe = 100

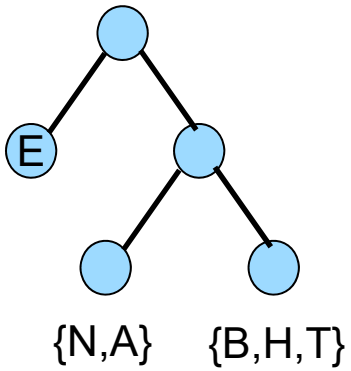
An nächsten an Hälfte: 42

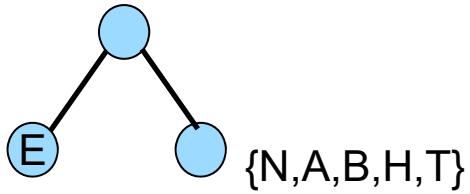
Also: In {E} und {N,A,B,H,T} unterteilen



Beispiel (II)

Zeichen	Häufigkeit	
E	42	0
N	20	1 0
A	16	1 0
B	8	1 1
H	8	1 1
T	6	1 1





Summe = 58

Unterteilungen:

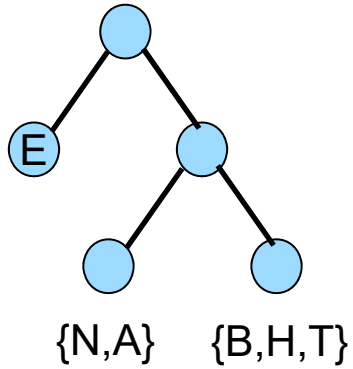
- {N} und {A,B,H,T} : 20 und 38
- {N,A} und {B,H,T} : 36 und 22

An nächsten an Hälfte: 36

Also: In {N,A} und {B,H,T} unterteilen

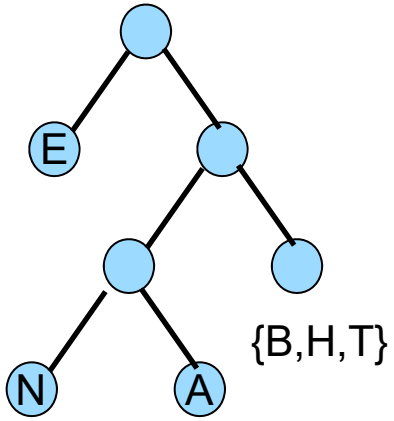
Beispiel (III)

Zeichen	Häufigkeit	
E	42	0
N	20	1 0 0
A	16	1 0 1
B	8	1 1
H	8	1 1
T	6	1 1



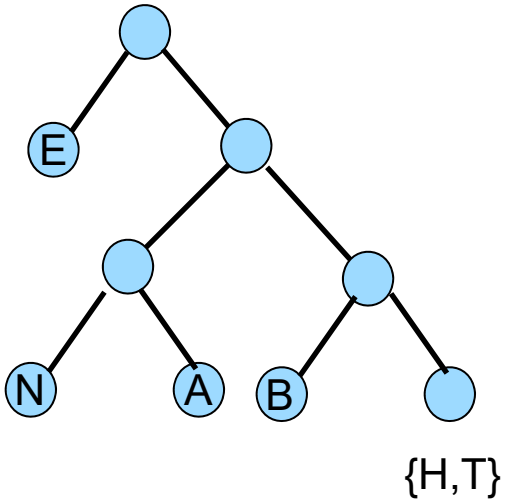
Gruppe {N,A}

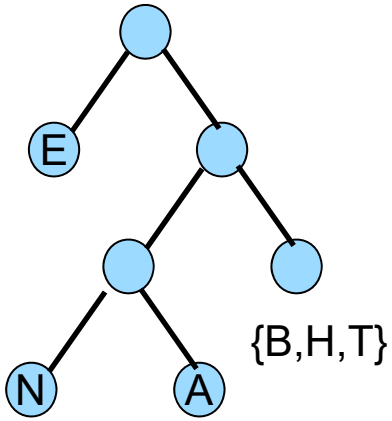
Nur eine Unterteilung möglich



Beispiel (IV)

Zeichen	Häufigkeit	
E	42	0
N	20	1 0 0
A	16	1 0 1
B	8	1 1 0
H	8	1 1 1
T	6	1 1 1





Unterteile {B,H,T}, Summe = 22

Unterteilungen:

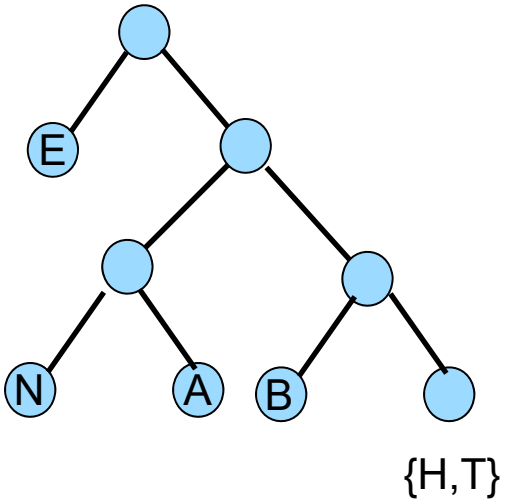
- {B} und {H,T} : 8 und 14
- {B,H} und {T} : 16 und 6

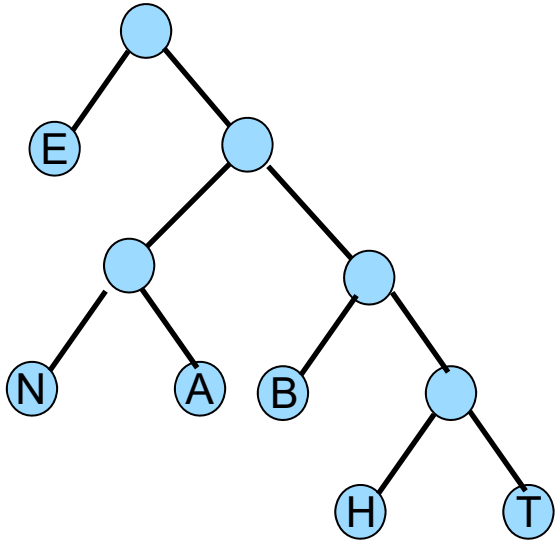
An nächsten an Hälfte: 8 und 14

Also: In {B} und {H,T} unterteilen

Beispiel (V)

Zeichen	Häufigkeit	
E	42	0
N	20	1 0 0
A	16	1 0 1
B	8	1 1 0
H	8	1 1 1 0
T	6	1 1 1 1





Gruppe {H,T}

Nur eine Unterteilung möglich

Bezug Baum - Code:

- Linke Kante mit 0 beschriften
- Rechte Kante mit 1 beschriften
- Pfad Wurzel-Blatt ergibt Code

Übersicht

- Grundbegriffe
- Code-Erzeugung
- Code-Sicherung
- Lineare Codes
- Nicht-binäre Codes

Stellen- und Hamming-Distanz

- Seien x und y Binärwörter gleicher Länge
- Stellen-Distanz oder Hamming-Distanz $d(x, y)$:= Anzahl Stellen, an denen sich x und y unterscheiden
- Berechnung: $x\ XOR\ y$, dann Anzahl Einsen zählen
- Bei korrekter Übertragung ist $x = y$ und $d(x, y) = 0$
- Hamming-Distanz $h(C)$ eines Codes C: Minimale Stellendistanz eines Codes

Beispiel

- Gegeben:

Ziffer	Code C_1
1	001
2	010
3	011
4	100

- Wie groß ist die Hamming-Distanz des Code und was bedeutet dies für die Fehlererkennung?

- Berechne Stellendistanzen von je zwei Codewörtern
- Am übersichtlichsten in Matrix eintragen:

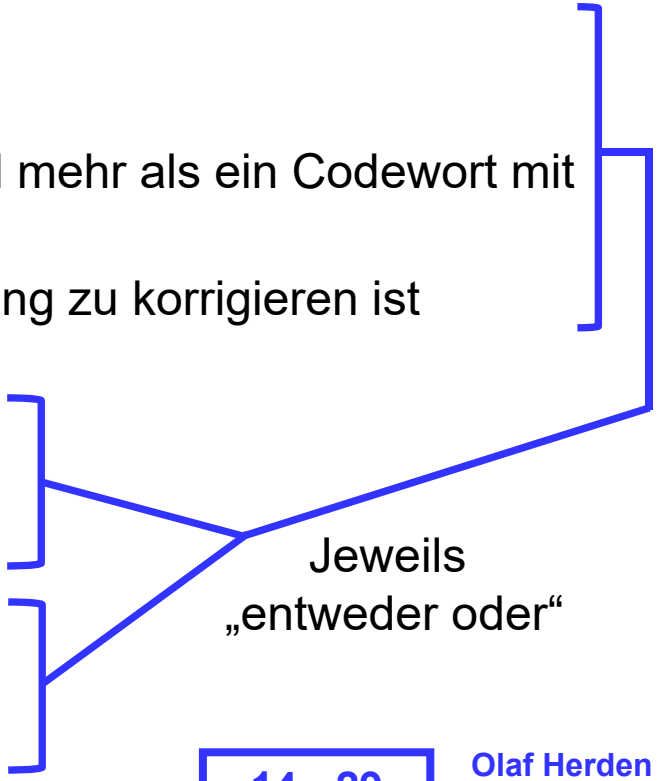
	001	010	011	100
001	-	-	-	-
010	2	-	-	-
011	1	1	-	-
100	2	2	3	-

- $h(C_1) = 1$

  $\Rightarrow$ Fehler lassen sich nicht in jedem Fall erkennen (es gibt Nutzwörter zwischen denen keine Fehlerwörter liegen)

Fehlererkennung und –korrektur (I)

- Es gilt:
  - Sind max. $h-1$ Bit in Wort fehlerhaft $\Rightarrow$ Erkennung möglich
  - Sind max. $(h-1)/2$ Bit in Wort fehlerhaft $\Rightarrow$ Korrektur möglich
- $h = 1$:
  - Keine Fehlererkennung, da Fehler wieder zu gültigem Codewort führen
- $h = 2$:
  - Fehlererkennung (von 1 Bit-Fehlern)
  - Keine Korrektur möglich, da
    - ungültiges Codewort, das in mind. einem Fall mehr als ein Codewort mit Stellendistanz 1 als Nachbarn hat
    - Damit: Keine Entscheidung, in welche Richtung zu korrigieren ist
- $h = 3$ oder $h = 4$:
  - 1-Bit-Fehler können korrigiert werden
  - 2-Bit- bzw. 3-Bit-Fehler können erkannt werden
- $h = 5$ oder $h = 6$:
  - 2-Bit-Fehler können korrigiert werden
  - 4-Bit- bzw. 5-Bit-Fehler können erkannt werden



Fehlererkennung und –korrektur (II)

- Beispiel („entweder oder“):
  - Betrachte Codewörter $x = 10101110$ und $y = 00101011$
  - Code habe $h = 3$
  - x und y unterscheiden sich an genau drei Stellen
  - Annahme: Codewort x wurde gesendet, Fehler bei Übertragung
  - Fall 1: Einzelnes Bit falsch:
    - Durch Kippen des MSB wird aus x $x' = 00101110$
    - Das nächstliegende gültige Codewort zu x‘ ist x mit $d(x, x') = 1 \Rightarrow$ Fehler wird korrigiert
  - Fall 2: Zwei Bits bei Übertragung verändert
    - Übertragung ergibt $x'' = 00101111$
    - Fehlererkennung, da kein gültiges Codewort (nächstes gültiges muss Mindestabstand 3 haben)
    - Keine Fehlerkorrektur möglich
      - da x'' auch aus y entstehen könnte (durch Fehler im dritten Bit von rechts)
      - Auf Empfängerseite nicht unterscheidbar

Fehlererkennung und –korrektur (III)

- Beispiel (Ursprünglicher Code C_1 und alternativer 3-Bit-Code C_2):

Ziffer	Code C_1	Code C_2
1	001	000
2	010	011
3	011	101
4	100	110

- $h(C_2) = 2$, denn:

	000	011	101	110
000	-	-	-	-
011	2	-	-	-
101	2	2	-	-
110	2	2	2	-

- 1-Bit-Fehler können in Code C_2 erkannt werden

Fehlererkennung und –korrektur (IV)

- Codeüberdeckungsproblem: Wie kann optimaler Code mit vorgegebener Hamming-Distanz generiert werden?
- „Optimal“ := Möglichst kurze Codewörter

m-aus-n-Codes (I)

- Blockcode mit Wortlänge n
- Jedes Wort hat
  - genau m Einsen und
  - $n-m$ Nullen
- Beispiele:

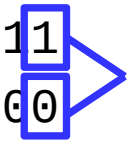
Ziffer	2-aus-5-Code	1-aus-10-Code
0	00011	0000000001
1	00101	0000000010
2	00110	0000000100
3	01001	0000001000
4	01010	0000010000
5	01100	0000100000
6	10001	0001000000
7	10010	0010000000
8	10100	0100000000
9	11000	1000000000

m-aus-n-Codes (II)

- m-aus-n-Codes haben Hamming-Distanz 2:
  - Jedes Codewort gleich viele Einsen $\Rightarrow$ Zwei Codewörter müssen sich in mindestens zwei Stellen unterscheiden
- Gegeben $n, m \Rightarrow$ Anzahl Codewörter $\binom{n}{m}$

Einfacher Paritätscode

- Füge an Codewort Paritätsbit (synonym: Prüfbit) als LSB an, so dass Anzahl Einsen jedes Codeworts gerade (gerade Parität, even parity) oder ungerade (ungerade Parität, odd parity) ist
- Beispiel:
  - Gerade Parität
  - `10110101` → `101101011`
  - `00100100` → `001001000`

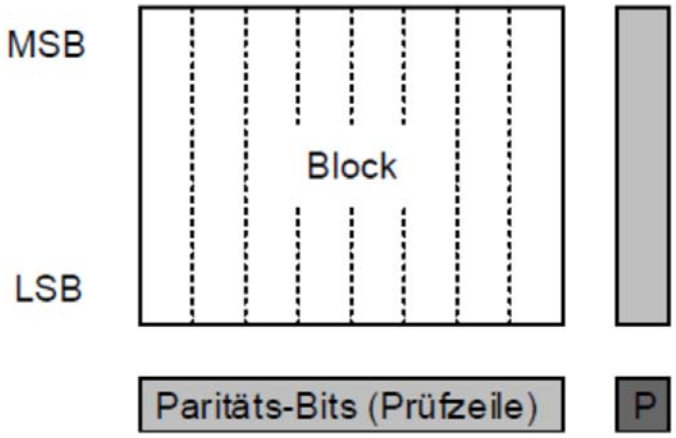


Paritätsbit

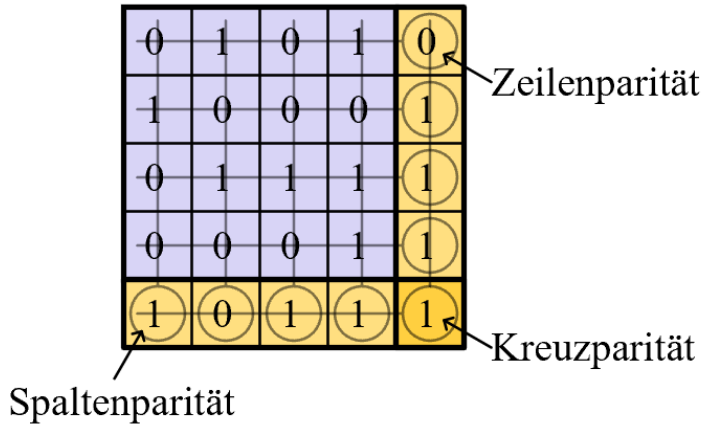
- Erkennung von 1-Bit-Fehlern
- Keine Korrektur möglich

Kreuzparität (I): Konzept

- Idee: Prüfzeilen und –spalten



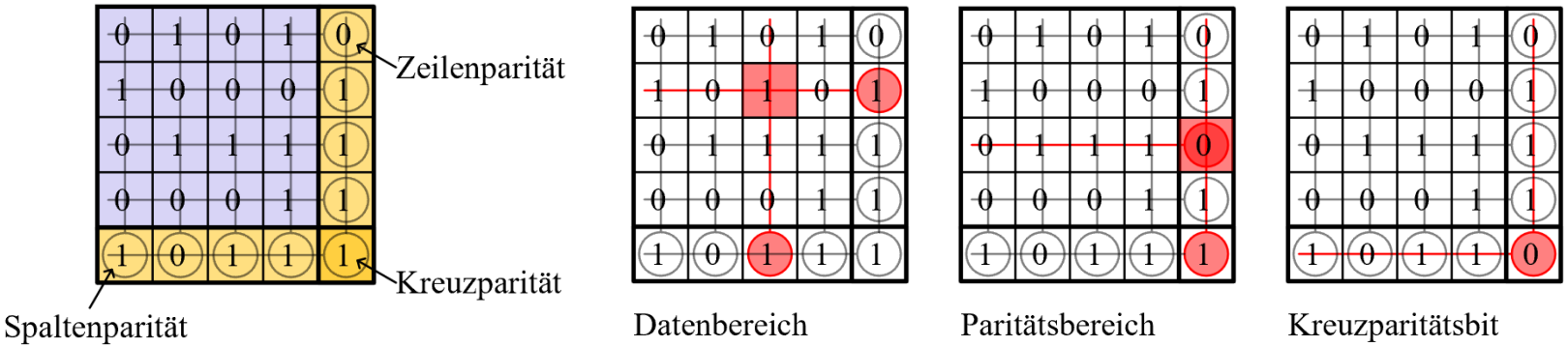
Längsprüfwort
(Prüfspalte)



- Gebe jedem Wort Paritätsbit
- Fasse Paritätsbits von Wörtern zu Prüfzeile zusammen (LRC, Longitudinal Redundancy Check)
- Nach Block von k Wörtern: Bilde Parität der Zeilen des Blocks, fasse diese zu Prüfspalte zusammen (VRC, Vertical Redundancy Check, Längsprüfwort)
- Bilde aus Prüfzeile und –spalte Paritätsbit P (ergänzt Anzahl Einsen im Block auf gerade Anzahl)

Kreuzparität (II): 1-Bit-Fehler

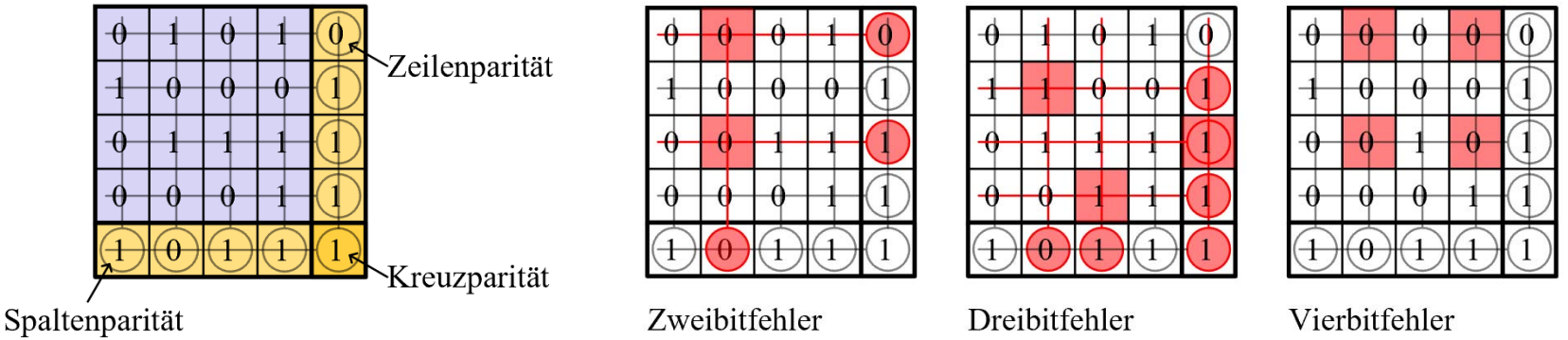
- Erkennung und Korrektur von 1-Bit-Fehlern



- Fall 1: Fehler im Datenbereich
  - Sowohl in Paritätszeile wie auch –spalte je 1 Bit verletzt Parität
  - Positionen dieser beiden Bits definiert eindeutig Position im Block
  - Invertiere dieses Bit
- Fall 2: Fehler in Prüfwort, nicht in P
  - Paritätsverletzung in Prüfzeile oder –spalte, nicht in beiden
  - Keine Korrektur notwendig
- Fall 3: Fehler in P
  - Sowohl Prüfzeile wie auch –spalte in Ordnung
  - Keine Korrektur notwendig

Kreuzparität (III): 2-/3-/4-Bit-Fehler

- Erkennung von 2- und 3-Bit-Fehlern



- 2-Bit-Fehler (beide in Daten):
  - Paritätszeile oder –spalte verletzt Parität
  - Sind beide Fehler in einer Zeile/Spalte ⇒ Fehler hebt sich auf
- 3-Bit-Fehler (zwei in Daten, einer in Prüfspalte)
  - Paritätsverletzung in Prüfzeile und –spalte
  - Ungünstige Fälle denkbar: sehen aus wie 1-Bit-Fehler
- 4-Bit-Fehler (alle in Daten bei (un)günstiger Lage):
  - Sowohl Prüfzeile wie auch –spalte in Ordnung
  - Fehler werden nicht erkannt

Kreuzparität (IV): Eigenschaften

- Redundanz, abhängig von Wortlänge s und Anzahl k Worte pro Block:

$$R = \frac{k+s+1}{k} \quad [Bit\ pro\ Wort]$$

- Hamming-Distanz $h = 4$:
  - Entweder 1-Bit-Fehler korrigieren
  - oder 2-Bit-Fehler und 3-Bit-Fehler erkennen

Kreuzparität (V): Beispiel (I)

Bsp. 3.8 Absicherung des Wortes INFORMATIK mit Kreuzparitätskontrolle

Zur binären Codierung des Wortes INFORMATIK wird der ASCII-Code (siehe Tabelle 3.2) benützt, wobei die Anzahl der Einsen zu einer geraden Zahl in einem Paritätsbit und nach jedem vierten Wort in einem Längsprüfwort ergänzt wird. Bei der Übertragung seien 1-Bit Fehler aufgetreten, so dass das Wort ANFORMAPIK empfangen wird. Wie oben beschrieben, lassen sich diese beiden Übertragungsfehler erkennen und korrigieren, das korrekte Wort INFORMATIK lässt sich also wieder restaurieren. Der Decodiervorgang ist in Tabelle 3.6 dargestellt.

Tabelle 3.6 Das im ASCII-Zeichensatz codierte Wort INFORMATIK wird mit Paritätsbits und Längsprüfwörtern in gerader Parität seriell übertragen. Die Übertragung erfolgte in Blöcken, wobei die beiden ersten Blöcke je vier und der dritte Block nur noch zwei Zeichen enthält, er wurde daher mit Nullen zur vollen Länge von vier Zeichen aufgefüllt. Es sind zwei Fehler (A statt I in Spalte 1 und P statt T in Spalte 8) aufgetreten. Diese lassen sich lokalisieren und korrigieren. Die zur Fehleridentifikation führenden Paritätsbits sowie die entsprechenden Bits der Längsprüfworte sind durch Pfeile markiert, die fehlerhaft übertragenen Bits sind grau unterlegt

	empfangene Daten				Längs-prüfwort	empfangene Daten				Längs-prüfwort	empfangene Daten				Längs-prüfwort	
MSB	1	1	1	1	0	1	1	1	1	0	1	1	0	0	0	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	
	0	1	0	1	1 ⇐	0	1	0	0	1	1	1	0	0	0	
	0	1	1	1	1	0	1	0	0	0 ⇐	0	0	0	0	0	
	0	1	1	1	1	1	0	0	0	1	0	1	0	0	1	
LSB	1	0	0	1	0	0	1	1	0	0	1	0	0	0	1	
	1	0	1	1	1	1	0	0	1	0	1	1	0	0	0	Paritätsbits
	⇑								⇑							
	A	N	F	O		R	M	A	P		I	K				empfangener Text
	⇑								⇑							
	I								T							Korrekturen

[Ernst et. al. 2016]

Kreuzparität (VI): Beispiel (II)

Bsp. 3.9 Kreuzparitätskontrolle hat eine Hamming-Distanz von vier

Dies soll am ersten Datenblock von Bsp. 3.8 erläutert werden, bei dem also nur der Wortteil INFO im ASCII-Code übertragen wurde. Beispiele für Einzel-, Doppel-, Dreifach- und Vierfachfehler sind in Tabelle 3.7 gezeigt. Am Dreifach-Fehler in Tabelle 3.7c mit nur zwei falschen Paritätsbits erkennt man die mögliche Verwechslung mit einem 1-Bit Fehler; der 3-Bit Fehler würde also bei Einsatz als korrigierender Code als 1-Bit Fehler interpretiert und falsch korrigiert. Ein vierfacher Fehler ist wie in 3.7d zu sehen nicht in jedem Fall erkennbar.

Tabelle 3.7 Die zur Fehleridentifikation führenden Paritätsbits sowie die entsprechenden Bits der Längsprüfworte sind durch Pfeile markiert, die fehlerhaft übertragenen Bits sind grau unterlegt

(a) 2-Bit Fehler mit zwei falschen Paritätsbits

1	1	1	1	0
0	0	0	0	0
0	0	0	0	0
0	1	0	0	1
0	1	1	1	1
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
⇑			⇑	

(b) 2-Bit Fehler mit vier falschen Paritätsbits

1	1	1	1	0
0	0	0	0	0
0	0	0	1	0 ⇐
0	1	0	1	1 ⇐
0	1	1	1	1
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
⇑			⇑	

(c) 3-Bit Fehler mit zwei falschen Paritätsbits

1	1	1	1	0
1	0	0	0	0 ⇐
0	0	0	0	0
0	1	0	0	1
0	1	1	1	1
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
			⇑	

(d) 4-Bit Fehler: alle Paritätsbits korrekt

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

[Ernst et. al. 2016]

Tetraden mit drei Paritätsbits (I)

- Ordne Codewort mehr als 1 Paritätsbit zu
- Vorteil: Jedes Wort kann für sich geprüft werden
- Beispiel: Tetraden mit 3 Paritätsbits

$b_6 b_5 b_4 b_3 b_2 b_1 b_0$

Code Paritätsbit

Ziffer	Code
0	0000111
1	0001100
2	0010010
3	0011001
4	0100001
5	0101010
6	0110100
7	0111111
8	1000000
9	1001011

- Regeln Bildung der Paritätsbits:
  - $b_2 = 1$, wenn Anzahl Einsen in b_6, b_5 und b_4 gerade
  - $b_1 = 1$, wenn Anzahl Einsen in b_6, b_5 und b_3 gerade
  - $b_0 = 1$, wenn Anzahl Einsen in b_6, b_4 und b_3 gerade

Tetraden mit drei Paritätsbits (II)

- Drei Paritätsbits $\Rightarrow 2^3 = 8$ Zustände („Kein Fehler“ und „Fehler in einer der sieben Stellen“)
- D.h. Code kann 1-Bit-Fehler erkennen und korrigieren
- Ermittlung fehlerhafter Stelle: (r = richtig, f = falsch)

Fehlerhaftes Bit	b_2	b_1	b_0
0	r	r	f
1	r	f	r
2	f	r	r
3	r	f	f
4	f	r	f
5	f	f	r
6	f	f	f

Verletzt ein Prüfbit p die Parität
$\Rightarrow p$ selbst fehlerhaft

Tetraden mit drei Paritätsbits (III)

- Entropie:
  - Annahme: Alle 10 Ziffern gleiche Wahrscheinlichkeit $p = 0{,}1$
  - $H = log_2\left(\frac{1}{p}\right) = log_2(10) \approx 3{,}3219$
- Redundanz:
  - $R = L - H \approx 7 - 3{,}3219 = 3{,}6781\ Bit/Zeichen$
  - Setzt sich zusammen aus 3 Prüfbits
  - Und 0,6781 Bit/Zeichen Informationsbits (nur 10 von 16 möglichen verwendet)
- Hamming-Distanz:
  - Für Informationsbits: $h = 1$
  - Zusammen mit Prüfbits: $h = 3$
  - Damit: Entweder
    - Erkennung und Korrektur von 1-Bit-Fehlern
  - Oder
    - Erkennung (aber keine Korrektur) von 2-Bit-Fehlern

Übersicht

- Grundbegriffe
- Code-Erzeugung
- Code-Sicherung
- Lineare Codes
- Nicht-binäre Codes

Exkurs: Körper

- Def.: Körper

  Körper ist Menge K mit zwei inneren zweistelligen Verknüpfungen „+“ und „·“ (Addition und Multiplikation) mit:

  - Existenz neutrales Element 0, d.h. $\forall k \in K: k + 0 = k$
  - Existenz neutrales Element 1, d.h. $\forall k \in K \backslash \{0\}: k \cdot 1 = k$
  - Distributivgesetzen:

    $a \cdot (b + c) = a \cdot b + a \cdot c$ für alle $a, b, c \in K$.

    $(a + b) \cdot c = a \cdot c + b \cdot c$ für alle $a, b, c \in K$.

- Beipiele:
  - Menge der rationalen, reellen und komplexen Zahlen
  - Boolescher Körper:
    - $K = \{0,1\}$
    - Konjunktion AND als Multiplikation
    - Antivalenz XOR als Addition

Exkurs: Vektorraum

- Sei V Menge, K Körper, $\oplus: VxV \rightarrow V$ innere zweistellige Verknüpfung (Vektoraddition) und $\otimes: KxV \rightarrow V$ äußere zweistellige Verknüpfung (Skalarmultiplikation)
- $(V, \oplus, \otimes)$ heißt Vektorraum (über Körper K), wenn folgendes gilt:

Vektoraddition:

V1: $u \oplus (v \oplus w) = (u \oplus v) \oplus w$ (Assoziativgesetz)

V2: Existenz eines neutralen Elements $0_V \in V$ mit $v \oplus 0_V = 0_V \oplus v = v$

- V3: Existenz eines zu $v \in V$ inversen Elements $-v \in V$ mit $v \oplus (-v) = (-v) \oplus v = 0_V$

V4: $v \oplus u = u \oplus v$ (Kommutativgesetz)

Skalarmultiplikation:

S1: $\alpha \odot (u \oplus v) = (\alpha \odot u) \oplus (\alpha \odot v)$ (Distributivgesetz)

S2: $(\alpha + \beta) \odot v = (\alpha \odot v) \oplus (\beta \odot v)$

S3: $(\alpha \cdot \beta) \odot v = \alpha \odot (\beta \odot v)$

S4: Neutralität des Einselements $1 \in K$, also $1 \odot v = v$

- Unterraum: Teilmenge, die die Eigenschaften „mitnimmt“

Definitionen (I)

- Def.: Linearer Code (bzw. linearer (s,r)-Code))
  - Sei B^s Vektorraum über booleschem Körper B
  - Sei $C \subseteq B^s$ Code mit $n = 2^r$ Codewörtern der Länge s
  - C heißt linearer Code g.d.w C ist Unterraum von B^s

- Def.: Gewicht eines Codeworts
  - Sei C linearer Code, $x \in C$ Codewort
  - Gewicht $g(x) := Anzahl\ Einsen\ in\ x$
- Satz: „Gewicht = Stellendistanz zum Nullvektor“
  - Sei **0** das nur aus Nullen bestehende Codewort
  - Dann: $g(x) = d(x, \mathbf{0})$

- Def.: Minimales Gewicht eines Codes
  - Sei C linearer Code
  - Dann: $g_{min} := \min\{g(x) \mid x \in C, x \neq \mathbf{0}\}$

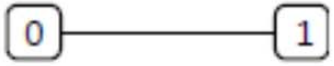
Definitionen (II)

- Satz:
  - Sei C linearer Code
  - Hamming-Distanz h = Minimalgewicht von C ($h = g_{min}$)

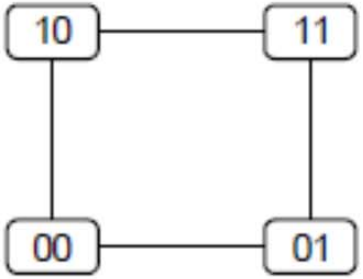
- Anwendung:
  - Reduktion Berechnungsaufwand Hamming-Distanz
  - Für Codes mit n Codewörtern allgemein: Ermittlung aller Stellendistanzen $\frac{(n^2-n)}{2}$ Vergleiche
  - Für lineare Codes: Berechne n Stellendistanzen zu **0**, bilde Minimum

Geometrische Interpretation (I)

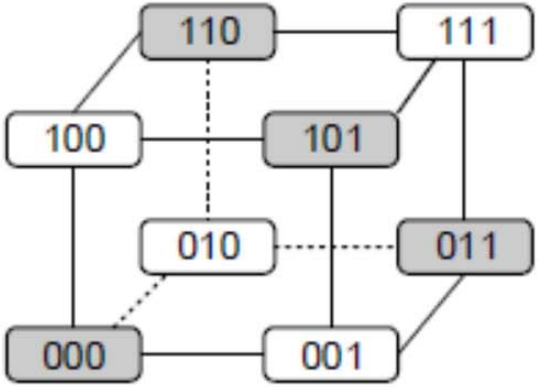
- Anordnung $n = 2^r$ Codewörter als Ecken r-dimensionalen Würfels/Kugel
- Anm.: Zugrundeliegender Raum ist diskret $\Rightarrow$ Würfel = Kugel
- Zeichne Kante, wenn sich Codewörter um genau $1\ Bit$ unterscheiden
- Beispiele:



(a) Für $r = 1$ enthält der Code $2^1 = 2$ Codewörter. Der zugehörige eindimensionale Würfel ist eine Gerade



(b) Für $r = 2$ enthält der Code $2^2 = 4$ Codewörter. Der zugehörige zweidimensionale Würfel ist ein Quadrat

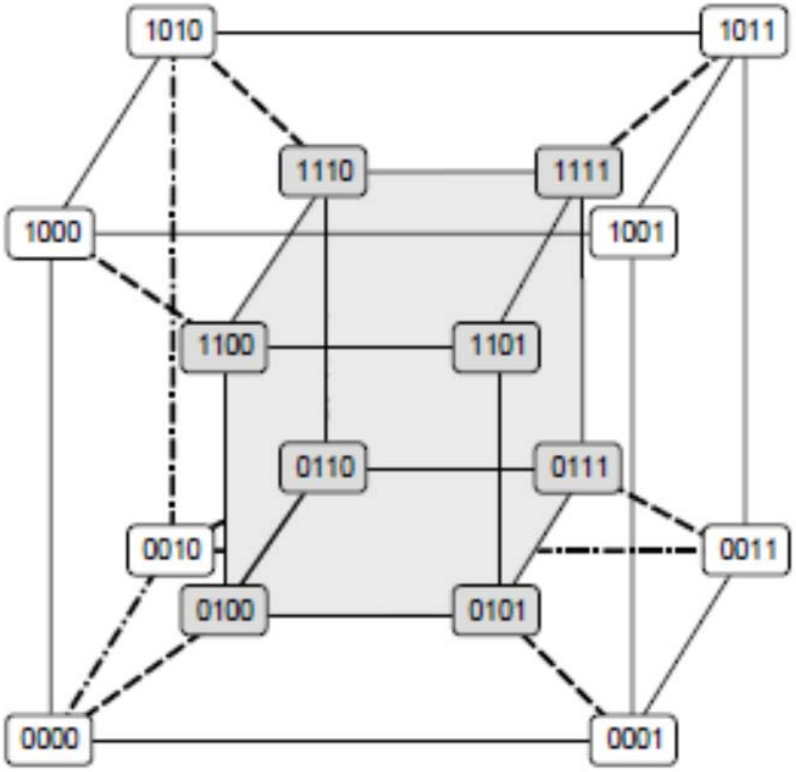


(c) Für $r = 3$ enthält der Code $2^3 = 8$ Codewörter. Die geometrische Anordnung aller aus drei Bit bildbaren Wörter führt dann zu einem dreidimensionalen Würfel. Die grau unterlegten Wörter bilden einen Code mit Hamming-Distanz $h = 2$, ebenso die hell unterlegten Wörter

- Hamming-Distanz h: Kürzester Weg von einem Nutzwort zu anderem Nutzwort führt über mindestens h Kanten

Geometrische Interpretation (II)

- Idee Anordnung auch für $r > 3$ (Hyperwürfel, Hypercube)
- Beispiel:



Perfekte Codes (I)

- Ziel: Konstruktion Code, bei dem bis zu x-Bit-Fehler korrigierbar sein sollen
- Basis: Geometrische Interpretation
- Ordne Codewörter so an,
  - dass jedes Codewort Mittelpunkt einer Kugel mit Radius x ist
  - und sich diese Kugeln nicht überlappen
  - Dann: $h = 2x + 1$
- Code mit 1-Bit-Fehler korrigieren:
  - $h = 3$ notwendig
  - Kugeln müssen mindestens Radius $x = 1$ haben
  - Obergrenze Anzahl Codewörter: $n_x \leq \frac{V_{ges}}{V_x}$
    - V_{ges}: Gesamtvolumen (=Anzahl Codewörter)
    - V_x: Volumen (=Anzahl Codewörter) Kugel mit Radius x

Perfekte Codes (II)

- Gesamter Raum B^s hat Volumen 2^s
- Volumen Kugel mit Radius 1:
  - Umfasst Mittelpunkt und alle über eine Kante erreichbaren Codewörter
  - D.h. $V_1 = 1 + s$
  - Damit gilt: $n_1 \leq \frac{2^s}{V_1} = \frac{2^s}{1+s}$
- Verallgemeinerung:
  - Bei x pro Codewort zu korrigierenden Fehlern: Setze an Stelle V_1 das Volumen V_x
  - Abzählen aller Codewörter, die von betrachtetem Punkt aus über maximal x Kanten erreichbar sind: $V_x = 1 + \sum_{i=1}^{x} \binom{s}{i}$
- Damit: Obere Grenze n_x s-stelliger Codewörter eines Codes mit Korrigierbarkeit von x Fehlern: $n_x \leq \frac{2^s}{V_x} = \frac{2^s}{1+\sum_{i=1}^{x}\binom{s}{i}}$

Perfekte Codes (III)

- Formel
  - besagt: n_x ist Obergrenze Anzahl Codewörter
  - Sagt nichts aus, ob n_x erreichbar ist und wie diese Codes konstruiert werden können
- Def.: Perfekter Code
  - Sei C linearer Code mit Fehlerkorrigierbarkeit x
  - C heißt perfekter Code g.d.w. n_x wird erreicht
- Beispiel:
  - Für $s = 3, x = 1$ ergibt sich $n_1 \leq \frac{2^3}{(1+3)} = 2$
  - Für Code $\{000,111\}$ ist $x = 1$ und $h = 3$
  - Code ist perfekt

Perfekte Codes (IV)

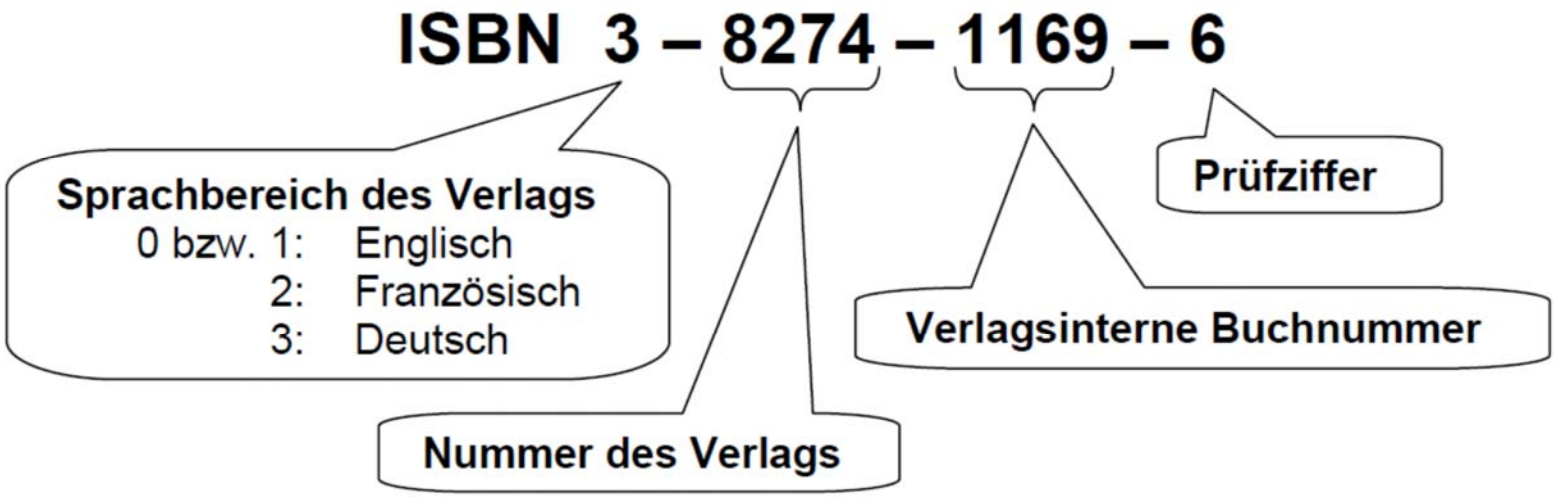
- Beispiel:
  - Für $s = 8, x = 1, h = 5$
  - $n_2 \leq \frac{2^8}{\left(1+\sum_{i=1}^{2}\binom{8}{i}\right)} = \frac{256}{(1+8+28)} \approx 6{,}9$
  - Für $s = 8$ gibt es einige Code mit $h = 5$
  - Diese umfassen aber maximal 4 Codewörter, sind nicht linear
  - 8-stelliger Code mit vier Codewörtern $\Rightarrow$ bestenfalls Hamming-Distanz 4 möglich

Übersicht

- Grundbegriffe
- Code-Erzeugung
- Code-Sicherung
- Lineare Codes
- Nicht-binäre Codes

ISBN (I)

- Ursprünglich: 10-stelliger Code



- Berechnung Prüfziffer P:
  - Jeder der ersten neun Ziffern wird Multiplikator zugeordnet (von 10 abwärts)
  - Ziffer wird mit Multiplikator multipliziert und Produkte addiert
  - P Zahl, die zu Summe addiert werden muss, so dass nächstes ganzes Vielfaches von 11 erreicht wird
  - $P \in \{0,1, \ldots, 10\} \Rightarrow$ Schreibe X für 10

ISBN (II)

- Beispiel:

ISBN	3	8	2	7	4	1	1	6	9
Multiplikator	10	9	8	7	6	5	4	3	2
Produkt	30	72	16	49	24	5	4	18	18
Summe	30	102	118	167	191	196	200	218	236

30 + 72 + 16 + 49 + 24 + 5 + 4 + 18 + 18 = 236

236 : 11 = 21 Rest 5 ➔ 11 – 5 = 6 ➔ 6 ist die berechnete Prüfziffer.

- Aktueller ISBN-Code 13-stellig
- Bisherige Ziffern bleiben
- Zusätzlich: Präfix vorangestellt, je nach Buch 978 oder 979
- Prüfziffernberechnung wie bei EAN-Code

EAN

- EAN: Europäische Artikel Nummer
  - 13-stellig
  - Letzte Ziffer Kontrollziffer
  - Berechnung:
    - Ordne ersten 12 Ziffern alternierend Faktoren 1 und 3 zu
    - Bilde Produkte und Summe der Produkte
    - Prüfziffer ist Zahl, die zur Summe addiert werden muss, um nächstes gnazes Vielfaches von 10 zu erhalten
- Beispiel:

EAN	9	7	8	3	8	2	7	4	1	1	6	9
Multiplikator	1	3	1	3	1	3	1	3	1	3	1	3
Produkt	9	21	8	9	8	6	7	12	1	3	6	27

Die Summe der Produkte beträgt 117, die Ergänzung zur nächsten Zehnerzahl ist 3.

IBAN (I)

- IBAN: International Bank Account Number (https://www.iban.de/)
  - Bis zu 34 Zeichen
  - In Deutschland 22 Zeichen: $DEp_1p_2b_1b_2b_3b_4b_5b_6b_7b_8k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}$ mit
    - k_1 bis k_{10} und mit b_1 bis b_8 ehemalige Kontonummer und BLZ
    - p_1 und p_2 Prüfziffern
  - Berechnung Prüfziffern:
    - Stelle Länderkürzel und auf 0 gesetzte Prüfziffern ganz nach rechts
    - Setze Buchstaben der Länderkennung auf Position im Alphabet $+9$
    - Berechne für diese Zahl Rest der Division durch 97
    - Prüfziffern: $98 - Rest$
  - Prüfung:
    - Stelle Länderkürzel und Prüfziffern ganz nach rechts
    - Setze Buchstaben der Länderkennung auf Position im Alphabet $+9$
    - $Zahl\ modulo\ 97$ muss 1 ergeben

IBAN (II)

- Beispiel:

  Für die deutsche Bankleitzahl 711 500 00 und Kontonummer 215 632 soll die IBAN berechnet werden. Kombination der beiden Zahlen mit rechts angehängtem Länderkürzel und Prüfziffern Null lautet: 711 500 00 0000 215 632 DE 00.
  Nach dem Ersetzen des Kürzels DE mit den Positionen im Alphabet plus 9 (D=13, E=14) erhält man: 711 500 00 0000 215 632 131400.
  Der Rest bei Division durch 97 liefert: $711\,500\,00\,0000\,215\,632\,131400 \bmod 97 = 49$.
  Die gesuchten Prüfziffern lauten also $98 - 49 = 49$, die gesamte IBAN ergibt sich zu DE 49 711 500 00 0000 215 632.

  Eine Validierung der IBAN liefert $711\,500\,00\,0000\,215\,632\,131449 \bmod 97 = 1$, diese ist also korrekt.

- Gewährung Sicherheit:
  - Arithmetik mit bis zu 36-stelligen Zahlen notwendig
  - Mit gängigen Programmiersprachen (Datentypen bis 64 Bit) nicht möglich

Zusammenfassung (I)

- Grundbegriffe:
  - Definitionen
  - Entropie
  - Codierung
  - Code-Redundanz
  - Quellen-Redundanz
  - Beispiele: BCD, ASCII, Unicode

- Code-Erzeugung:
  - Code-Erzeugungsproblem
  - (Huffmann-Algorithmus)
  - Algorithmus von Fano und Shannon

Zusammenfassung (II)

- Code-Sicherung:
  - Stellen- und Hamming-Distanz
  - Fehlererkennung- und –korrektur
  - m-aus-n-Codes
  - Paritätscodes: Bit und Kreuzparität
  - Tetraden mit drei Paritätsbits

- Lineare Codes:
  - Linearer Code
  - Gewicht Codewort
  - Minimales Gewicht eines Codes
  - Geometrische Interpretation
  - Perfekte Codes

- Nicht-binäre Codes:
  - ISBN, EAN, IBAN

Übungen (I)

- **Aufgabe 1**

  Finden Sie im folgenden Rätsel fünf Begriffe aus diesem Vorlesungsabschnitt!

```
Q Y B C P V D S D F H W L F O R L Y E L
F D J U B G W H S M S P L D T Q T C T U
H E M M Z L E A C T K Q M P E T H P U X
R I N F R V K N T E A U E P S F G B H P
S P S Y L D Z N R Y Q F V R W Q G T M Z
F O H T M B W O O W J D O J D P W P L N
O R W P O B C N S M N K M Z V G T V W P
O T L E P F L N Z N T K W N M P O I H B
C N X R J P O L P T E M B G G K Q W D O
Q E D F H Z N Y Q H K F D R U K X F P E
T D Q E A W U R Y O I B F A I N W K M Y
J N I K M T S O N I Z D G R H G P P Y W
L Q O T R T E A T I R A P Z U E R K V D
L Z H E N Q V N W M N J R X T B K U X E
T C N R R X H A M M I N G D I S T A N Z
Z E K C J Y H H U H I F T O J M P O L M
G G W O J M U E Z N M Y T O R D Q Z E F
J K K D H I F R P O D R Q C K F F X W R
S B G E H S D S L T G G E A J Y F T D K
C R P M P M S D Q P G S D B T F R D Y F
```

Übungen (II)

- **Aufgabe 2**
  - Was ist ein Blockcode?
  - Was ist der BCD-Code?
  - Was sind Pseudotetraden?
  - Was ist Entropie und wie wird diese berechnet?
  - Was ist Redundanz und wie wird diese berechnet?
  - Was ist Quellenredundanz?
  - Was bedeutet UTF?
  - Was ist die Hammingdistanz?
  - Was ist ein Paritätscode?
  - Wie ist Kreuzparität aufgebaut?
  - Was ist ein linearer Code?
  - Was ist ein perfekter Code?
  - Wie funktioniert die Prüfziffernberechnung beim ISBN?
  - Wie wird die Kontrollziffer im EAN berechnet?
  - Wie ermittelt man die Prüfziffern beim IBAN?

Übungen (II)

- **Aufgabe 3**

  Geben Sie für die Umwandlung von Klein- in Großbuchstaben im ASCII-Code eine einfache logische Operation mit Bit-Maske an!

- **Aufgabe 4**

  Das Alphabet $A = \{a, e, i, o, u\}$ mit den Auftrittswahrscheinlichkeiten $P(a) = 0{,}25$, $P(e) = 0{,}2$, $P(i) = 0{,}1$, $P(o) = 0{,}3$, $P(u) = 0{,}15$ sei mit $\{11{,}10{,}010{,}00{,}011\}$ binär codiert.

  Berechnen Sie die mittlere Codewortlänge und die Redundanz des Codes!

- **Aufgabe 5**

  Bestimmen Sie die Stellendistanzen und die Hamming-Distanzen für die folgenden Codes:

  a) $\{110101{,}101011{,}010011{,}101100\}$

  b) $\{00101011{,}01001010{,}01111000{,}10101001\}$

Übungen (IV)

- **Aufgabe 6**

  Bei einer seriellen Datenübermittlung werden mit 7 Bit codierte ASCII-Zeichen mit einem zusätzlichen Paritätsbit und einem Längsprüfwort nach jeweils 8 Zeichen gesendet. Es gilt gerade Parität.

  Folgende Nachricht wird empfangen:

MSB	1 0 1 1 1 1 1 1	0	
	0 1 1 1 1 1 1 1	1	
	0 1 0 0 0 0 0 1	0	
Datenbits	0 0 0 1 0 1 0 0	0	Prüfspalte
	1 0 1 0 0 0 1 0	1	
	1 1 0 0 1 0 0 1	0	
LSB	0 0 1 1 0 1 1 0	0	
Paritätsbits	1 0 0 0 1 0 0 0	0	

  a) Wie lautet die empfangene Nachricht?

  b) Sind Übertragungsfehler aufgetreten? Wenn ja, wie lautet die korrekte Nachricht?

  c) Bestimmen Sie die durch die Paritätsbits bedingte zusätzliche Redundanz!

Übungen (V)

- **Aufgabe 7**

  Wie viele verschiedene Codewörter kann ein 2-aus-6-Code und wie viele ein 1-aus-15-Code enthalten? Geben Sie für die beiden Codes die Hamming-Distanz und die Redundanzen an! Dabei kann für alle Codewörter die gleiche Auftretenswahrscheinlichkeit angenommen werden.

- **Aufgabe 8**

  Gegeben seien die folgenden Zeichen und ihre Auftretenshäufigkeiten:

Zeichen	A	B	C	D	E	F
Häufigkeit	25	25	20	15	10	5

  a) Berechnen Sie die Codierung mit dem Fano-Shannon-Algorithmus!

  b) Geben Sie die mittlere Codewortlänge und die Entropie an!

Übungen (VI)

- **Aufgabe 9**
  a) Berechnen Sie den Prüfcode zur 10-stelligen ISBN 3-8689-4379
  b) Ist der EAN 5702016604137 korrekt?
  c) Geben Sie die korrekte EAN-Prüfziffer für 978-386894379
  d) Ist die IBAN DE14200800000816170700 korrekt?
  e) Ermitteln Sie die IBAN für die Kontonummer 1212557188 und die BLZ 10020099!