Teil 2
Kapitel 6

Kodierung

6.1 Kodierung von Zeichen

6.2 Kodierung von Zahlen

Motivation

- Ein Rechner kann traditionell
  - Zeichen verarbeiten (Textverarbeitung),
  - mit Zahlen rechnen,
  - Bilder darstellen.
- Ein Algorithmus kann zwar prinzipiell mit abstrakten Objekten verschiedener Art operieren, aber diese müssen im Rechner letztendlich als Folgen von Bits repräsentiert werden.

6.1 Kodierung von Zeichen

- Wie werden im Rechner Zeichen dargestellt ?
- fehlererkennende und fehlerkorrigierende Codes (Bsp.: Parity Check, Hamming-Code)
- Häufigkeitscodes (Bsp.: Huffman-Code)

Code

Sei $A=\{a_1,\ldots,a_k\}$ ein endliches Alphabet der Grösse k.

- Eine Abbildung $c : A \rightarrow \{0,1\}^*$ oder $c : A \rightarrow \{0,1\}^n$ heißt **Code**, falls c injektiv ist.
- Die Menge $c(A) := \{ w \in \{0,1\}^* \mid \exists\, a \in A : c(a)=w \}$ heißt Menge der **Codewörter**
- Ein Code $c : A \rightarrow \{0,1\}^n$ heißt **Code fester Länge.**
- Für einen Code $c : A \rightarrow \{0,1\}^n$ fester Länge gilt: $n \geq \lceil \log_2 k \rceil$.

  Ist $n = \lceil \log_2 k \rceil + r$ mit $r > 0$, so können die r zusätzlichen Bits zum Test auf **Übertragungsfehler** verwendet werden (siehe später).

American Standard Code for Information Interchange

(7-Bit) Code zur Darstellung von Zeichen in Rechnern

ASCII value	Character	ASCII value	Character	ASCII value	Character	ASCII value	Character	ASCII value	Character	ASCII value	Character
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	91	O	97	a	113	q
34	´´	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	´	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	v
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	.	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95		111	o	127	del

Abspeicherungs- und Übertragungsfehler

- Ein **Übertragungsfehler** (Abspeicherungsfehler) eines Wortes liegt vor, wenn die empfangene Bitfolge verschieden von der gesendeten Bitfolge ist.
- Übertragungsfehler = **Kippen von Bitstellen** ($0 \rightarrow 1$, $1 \rightarrow 0$)
- Übertragungsfehler erhöhen die Distanz dist(v,w) zwischen der gesendeten Bitfolge v und der empfangenen Bitfolge w,

  wobei man unter der **Distanz zweier Bitfolgen** die Anzahl der Stellen versteht, an denen sich die beiden Bitfolgen unterscheiden.
- Ein Übertragungsfehler heißt **einfach**, wenn dist(v,w)=1 gilt.
- **Beispiel**
  - dist(00001101,10001100) = 2
  - dist(00001101,00001101) = 0

Fehlererkennender Code

Sei $c : A \to \{0,1\}^n$ ein Code fester Länge von A.

- Der kürzeste Abstand
  $$\mathrm{dist}(c) := \min\{\ \mathrm{dist}(c(a_i),c(a_j)); a_i,a_j \in A \text{ mit } a_i \neq a_j\ \}$$
  zwischen zwei verschiedenen Codewörtern heißt **Distanz des Codes** c .
- Der Code c heißt **k-fehlererkennend**, wenn der Empfänger in jedem Fall entscheiden kann, ob ein gesendetes Codewort durch Kippen von bis zu k Bits verfälscht wurde.

Lemma

Ein Code c von fester Länge ist genau dann k-fehlererkennend, wenn $\mathrm{dist}(c) \geq k+1$ gilt.

1-fehlererkennender Code: Beispiel

Sprechweise

Eine Bitfolge $w \in \{0,1\}^n$ besteht den **Paritätstest** (engl. **Parity-Check**), wenn die Anzahl der gesetzten Bitstellen, d.h. die Anzahl der auf 1 gesetzten Bitstellen, gerade ist.

Beispiel

Sei $c : A \to \{0,1\}^n$ ein Code fester Länge von A.

Betrachte den Code $C : A \to \{0,1\}^{n+1}$, der aus Code c entsteht, in dem eine Bitstelle an jedes Codewort c(a) hinten (oder vorne) angefügt wird und so gesetzt wird, dass der neue Code C(a) den Gleichheitstest besteht.

$\Rightarrow$ Code C ist 1-fehlererkennend !

Fehlerkorrigierender Code

Sei $c : A \rightarrow \{0,1\}^n$ ein Code fester Länge von A.

- Der Code c heißt **k-fehlerkorrigierend**, wenn der Empfänger in jedem Fall entscheiden kann, ob ein gesendetes Codewort durch Kippen von bis zu k Bits verfälscht wurde, und das gesendete Codewort aus der empfangenen Bitfolge wieder restaurieren kann.

Lemma

Ein Code c von fester Länge ist genau dann k-fehlerkorrigierend, wenn $dist(c) \geq 2k+1$ gilt.

Beweis Lemma [Fehlerkorrektur]

Sei $M(c(a_i), k) := \{w \in \{0,1\}^n \mid dist(c(a_i),w) \leq k\}$

die Kugel um $c(a_i)$ mit Radius k

Dann gilt:

c ist k-fehlerkorrigierend $\Leftrightarrow$

$\forall\, a_i, a_j \;\; i \neq j$ gilt: $M(c(a_i),k) \cap M(c(a_j),k) = \emptyset$

Für den Beweis ist also zu zeigen:

$[\forall\, a_i, a_j \;\; i \neq j$ gilt: $M(c(a_i),k) \cap M(c(a_j),k) = \emptyset]$

$\Leftrightarrow dist(c) \geq 2k+1$

Beweis Lemma [Fehlerkorrektur]

„⇒"

Annahme: $dist(c) < 2k+1$

d.h. $\exists\, a_i, a_j$ mit $dist(c(a_i),c(a_j)) = l$ mit $l < 2k+1$;

also gibt es eine Folge: $c(a_i) = b_0, b_1, \ldots, b_{k-1}, b_k, b_{k+1}, \ldots b_{2k} = c(a_j)$

mit $dist(b_i, b_{i+1}) = 0$ oder $dist(b_i, b_{i+1}) = 1$ $(i=0,\ldots,2k-1)$,

also $b_k \in M(c(a_i),k) \cap M(c(a_j),k)$

Beweis Lemma [Fehlerkorrektur]

„⇐"

Annahme: $M(c(a_i),k) \cap M(c(a_j),k) \neq \emptyset$

Es gibt also b im Durchschnitt mit:

$dist(c) \leq dist(c(a_i),c(a_j)) \leq dist(c(a_i),b) + dist(b,c(a_j)) \leq k + k$

1-fehlerkorrigierender Code

Sei $c : A \to \{0,1\}^{m+r}$ ein 1-fehlerkorrigierender Code fester Länge von A mit

- $|A|=2^m$
- $m \geq 3$.

Satz

Es gilt $r \geq 1+\lfloor \log_2 m \rfloor$

Beweis

$M_1(a):=\{b \in \{0,1\}^{m+r} : b$ entsteht durch Kippen von bis zu einem Bit aus $c(a)\}$

Es muss gelten $M_1(a_1) \cap M_1(a_2)=\emptyset$ für alle $a_1,a_2 \in A$.

Es gilt $|M_1(a)|=m+r+1$ für alle $a \in A$.

$\Rightarrow 2^m(m+r+1) \leq 2^{m+r}$, woraus die Behauptung (nach kleiner Rechnung) folgt

Beweis Satz [1-fehlerkorrigierender Code]

z.z. $(m+r+1) \leq 2^r \Rightarrow r \geq 1+\lfloor \log_2 m \rfloor$

Sei dazu $m= 2^k + l$ mit l,k natürliche Zahlen, $l \geq 0$ und k maximal

dann gilt

$(m+r+1) \leq 2^r \Leftrightarrow$

$2^k + l + r +1 \leq 2^r \Rightarrow$

$k < r \Leftrightarrow$

$k+1 \leq r \Leftrightarrow$

$1+\lfloor \log_2 m \rfloor \leq r$

Hamming Code

- ist 1-fehlerkorrigierender Code
- erweitert nicht fehlerkorrigierenden Code um r Bitstellen; dabei wird die Anzahl r der Zusatzbits so gewählt, dass r minimal ist unter der Bedingung $m + r \leq 2^r - 1$ und entspricht somit exakt der Bedingung aus dem Beweis des letzten Satzes für die minimale Länge eines 1-fehlerkorrigierenden Codes!
- ist also optimal im Hinblick auf die Zahl der ergänzten Stellen
- es werden z.B. für die Codierung eines Alphabets A mit $|A|=2^m$ und $m = 2^k$ insgesamt

$$m + 1 + \lfloor \log_2 m \rfloor$$

Bitstellen benötigt.

Hamming Code: Idee

- Erweitere nicht fehlerkorrigierenden Code um r Bitstellen.
- Benutze die Bitstellen $2^0, 2^1, \ldots 2^{r-1}$ als Überprüfungsbits, wobei die Bitstelle 2^j die Bitstellen überprüft, deren Binärdarstellungen an der j-ten Stelle eine 1 haben.
- Die Bitstelle 2^j wird so belegt, dass gerade viele Bitstellen, deren Binärdarstellungen an der j-ten Stelle eine 1 haben, gesetzt sind.

Hamming Code an einem Beispiel

- Uncodiertes Zeichen: 0111 0101 0000 1111
- m = 16, r = 5

Wie sieht nun der Hamming Code dieses Zeichens aus ?

- Der Code wird auf 21 Bitstellen verlängert
- Die "Zweierpotenz"-Bitstellen werden als Überprüfungsbits benutzt

0 1110 _ 101 0000 _ 111 _ 1 _ _

wobei die Bitstelle 2^j die Bitstellen überprüft, deren Binärdarstellungen an der j-ten Stelle eine 1 haben.
Die Bitstelle 2^j wird so belegt, dass gerade viele Bitstellen, deren Binärdarstellungen an der j-ten Stelle eine 1 haben, gesetzt sind.

Hamming Code an einem Beispiel

	2^4	2^3	2^2	2^1	2^0	zu codierende Bitfolge
3						1
5						1
6						1
7						1
9						0
10						0
11						0
12						0
13						1
14						0
15						1
17						0
18						1
19						1
20						1
21						0

Das Überprüfungsbit 2^j überprüft die Bitstellen, die in ihrer Binärdarstellung an der j-ten Stelle eine 1 haben.

Hamming Code an einem Beispiel

	2^4	2^3	2^2	2^1	2^0	zu codierende Bitfolge
3				_	_	1
5			_		_	1
6			_	_		1
7			_	_	_	1
9		_			_	0
10		_		_		0
11		_		_	_	0
12		_	_			0
13		_	_		_	1
14		_	_	_		0
15		_	_	_	_	1
17	_				_	0
18	_			_		1
19	_			_	_	1
20	_		_			1
21	_		_		_	0

Das Überprüfungsbit 2^j überprüft die Bitstellen, die in ihrer Binärdarstellung an der j-ten Stelle eine 1 haben.

Hamming Code an einem Beispiel

	2^4	2^3	2^2	2^1	2^0	zu codierende Bitfolge
3				1	1	1
5			1		1	1
6			1	1		1
7			1	1	1	1
9		0			0	0
10		0		0		0
11		0		0	0	0
12		0	0			0
13		1	1		1	1
14		0	0	0		0
15		1	1	1	1	1
17	0				0	0
18	1			1		1
19	1			1	1	1
20	1		1			1
21	0		0		0	0

1	0	0	0	0

Der Prüfbitwert ergibt sich aus der Summe modulo 2 der jeweiligen Spalte

Das Überprüfungsbit 2^j überprüft die Bitstellen, die in ihrer Binärdarstellung an der j-ten Stelle eine 1 haben.

Hamming Code von

0111 0101 0000 1111

ergibt sich zu

0 1110 1 101 0000 0 111 0 1 0 0

Wie findet man einen Fehler ?

Hamming Code von 0111 0101 0000 1111

0 1110 1 101 0000 0 111 0 1 0 0

Nehme an, es gibt einen Fehler an Position 13 !

Wie findet man einen Fehler ?

Hamming Code von 0111 0101 0000 1111

0 1110 1 100 0000 0 111 0 1 0 0

Nehme an, es gibt einen Fehler an Position 13 !

Wie findet man den Fehler ?

Wie findet man einen Fehler ?

	2^4	2^3	2^2	2^1	2^0	zu codierende Bitfolge
3				1	1	1
5			1		1	1
6			1	1		1
7			1	1	1	1
9		0			0	0
10		0		0		0
11		0		0	0	0
12		0	0			0
13		0	0		0	0
14		0	0	0		0
15		1	1	1	1	1
17	0				0	0
18	1			1		1
19	1			1	1	1
20	1		1			1
21	0		0		0	0
	1	0	0	0	0	

Fehler muss sich in Zeile 8+4+1=13 befinden !

Die Spalten 8, 4 und 1 bestehen den Gleichheitstest nicht !

Häufigkeitsabhängige Codes

Ziel

Reduktion der Länge einer Nachricht durch Wahl verschieden langer Codewörter für die verschiedenen Zeichen eines Alphabets

Idee

Häufiges Zeichen → kurzer Code
Seltenes Zeichen → langer Code

Voraussetzungen

Häufigkeitsverteilung ist bekannt → **statische Kompression**
Häufigkeitsverteilung nicht bekannt → **dynamische Kompression**

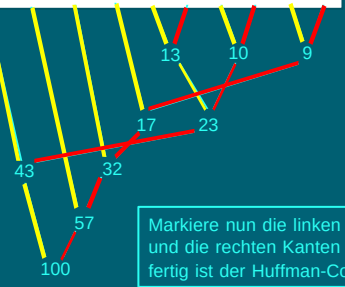
Häufigkeitsabhängige Codes: Huffman

bekanntester häufigkeitsabhängiger Code

Beispiel

Zeichen	a	b	c	d	e	f	g	h	i	j
Häufigkeit [%]	20	25	15	8	7	6	5	5	5	4

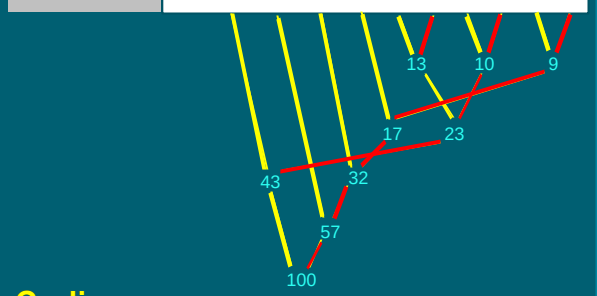
Baue binären Baum, indem die beiden kleinsten Häufigkeiten jeweils zu einem neuen Knoten addiert werden



Markiere nun die linken Kanten mit 0 und die rechten Kanten mit 1, fertig ist der Huffman-Code!

Huffman-Code

Zeichen	a	b	c	d	e	f	g	h	i	j
Häufigkeit [%]	20	25	15	8	7	6	5	5	5	4



Erzeugte Codierung

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
00	10	110	1110	0100	0101	0110	0111	11110	11111

Zusammenfassung

- Grundlegende Definitionen für Codes
- Fehlererkennung, Fehlerkorrektur
- Beispiel: Parity Check, Hamming Code

  Viele weitere Verfahren:
  - Zweidimensionaler Parity Check
  - CRC (Cyclic Redundancy Check):
    Standard bei Übertragungen über das Netz
- Häufigkeitsabhängige Codes: Huffman