Technische Informatik 1

Prof. Dr. Rolf Drechsler

Christina Plump

Überblick

Teil 1: Der Rechneraufbau (Kapitel 2-5)

- Rechner im Überblick
- Pipelining
- Speicher
- Parallelverarbeitung

Teil 2: Der Funktionalitätsaufbau (Kapitel 6-12)

- **Kodierung**
  - **Zeichen**
  - **Zahlen**
- Grundbegriffe, Boolesche Funktionen
- Darstellungsmöglichkeiten
- Schaltkreise, Synthese, spezielle Schaltkreise

Kapitel 6: Kodierung

Kodierung von Zeichen

Kodierung von Zahlen

Lernziele

- Darstellungsmöglichkeiten von Zeichen im Rechner kennenlernen
- Formalisierte Darstellung von Codes kennenlernen und verstehen
- Fehlererkennende und fehlerkorrigierende Codes als Kategorien kennenlernen und ihre Merkmale verstehen
- Häufigkeitsunabhängige und –abhängige Code kennen lernen
- Hamming-Code kennenlernen und anwenden können
- Huffman-Code kennenlernen und anwenden können

Motivation

- Traditionelle Fähigkeit eines Rechners
  - Verarbeitung von **Zeichen** (Textverarbeitung)
  - Rechen mit **Zahlen**
  - Darstellung von Bildern

- Algorithmus operiert mit abstrakten Objekten
- Repräsentation im Rechner durch Folgen von Bits geschehen

Codes und ihre Begrifflichkeiten

Sei $A = \{a_1, a_2, a_3, \dots, a_k\}$ ein endliches Alphabet der Größe k.

- **Code:** Abbildung $c: A \to \{0,1\}^*$ oder $c: A \to \{0,1\}^n$, falls c injektiv
  - **Code fester Länge:** $c: A \to \{0,1\}^n$
  - Es gilt: $n \geq \lceil \log_2 k \rceil$, d.h. $n = \lceil \log_2 k \rceil + r, r > 0$
  - r zusätzliche Bits genutzt für Test auf Übertragungsfehler
- **Codewörter**: Menge $c(A) := \{ w \in \{0,1\}^* \mid \exists\, a \in A : c(a) = w \}$

American Standard Code for Information Interchange (ASCII)

- 7-Bit Zeichenkodierung (33 nicht-druckbare und 95 druckbare Zeichen)
- Standard seit den 1970er Jahren
- Früher 7-bit Kodierung für Zeichen (128 Bit)
- Bit 8 häufig für Paritätscheck genutzt (siehe später) oder für nationale Erweiterungen des Zeichensatzes

ASCII		ASCII		ASCII		ASCII		ASCII		ASCII		ASCII		ASCII	
0	Nul	16	DLE	32	space	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34		50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYM	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	`	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	/	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	del

Abspeicherungs- und Übertragungsfehler

- **Distanz** (Hamming-Distanz) zweier Bitfolgen v, w: $dist(v,w) = \left|\{j \in \{1,\dots,n\} | v_j \neq w_j\}\right|$
  - Beispiel: $dist(001100, 111000) = |\{1,2,4\}| = 3$
  - Beispiel: $dist(111000, 111000) = |\{\}| = 0$
- **Übertragungsfehler (Abspeicherungsfehler)**: gesendete Bitfolge v verschieden von empfangende Bitfolge w
  - „Kippen von Bitstellen“ $(0 \to 1, 1 \to 0)$
  - erhöhen die Distanz zwischen gesendeter Bitfolge v und empfangener Bitfolge w
  - bei $dist(v,w) = 1$: **einfacher** Übertragungsfehler
- Entwicklung fehlererkennender und –korrigierender Codes

Fehlererkennender Code

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

- **Distanz des Codes c**: $dist(c) := \min\left\{dist\left(c(a_i), c(a_j)\right) \middle| a_i, a_j \in A, a_i \neq a_j\right\}$

  entspricht der kürzesten Distanz zweier Codewörter.

- **k-fehlererkennend**: 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 (Fehlererkennung):
Ein Code c von fester Länge n ist genau dann k-fehlererkennend, wenn $dist(c) \geq k + 1$

1-fehlererkennender Code: Beispiel

- **Paritätstest:** Hat Bitfolge $w \in \{0,1\}^n$ eine gerade Anzahl an gesetzten Bitstellen?

- Beispiel: Parity-Check Code
  Sei $c: A \to \{0,1\}^n$ ein Code fester Länge von A.
  Betrachte $C: A \to \{0,1\}^{n+1}$, mit $C(a) = (c(a), w_{n+1})$ mit $w_{n+1} = 1$, wenn $P(a) = false$, 0 sonst.

C ist 1-fehlererkennend

Fehlerkorrigierender Code

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

- **k-fehlerkorrigierend**: 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 wiederherstellen kann.

Lemma (Fehlerkorrektur):
Ein Code c von fester Länge n ist genau dann k-fehlererkennend, wenn $dist(c) \geq 2k + 1$

Lemma (Fehlerkorrektur): Beweis

- Betrachtung neuer Struktur: $M(c(a_i), k) := \{w \in \{0,1\}^n | dist(c(a_i), w) \leq k\}$
  - M heißt Kugel um $c(a_i)$ mit Radius k
  - Enthält alle Wörter mit maximaler Distanz k vom Codewort $c(a_i)$
- Jetzt gilt: c ist k-fehlerkorrigierend $\Leftrightarrow \forall a_i, a_j, i \neq j\ gilt: M(c(a_i), k) \cap M\big(c\big(a_j\big), k\big) = \emptyset$
- Beweis reduziert sich zu:

$$\forall a_i, a_j, i \neq j\ gilt: M(c(a_i), k) \cap M\big(c\big(a_j\big), k\big) = \emptyset \Leftrightarrow dist(c) \geq 2k + 1$$

Erinnerung:
$(A \Leftrightarrow B \wedge B \Leftrightarrow C\ ) \Leftrightarrow (A \Leftrightarrow C)$

Lemma (Fehlerkorrektur): Beweis

zu zeigen: $\forall a_i, a_j, i \neq j \; gilt: M(c(a_i), k) \cap M\left(c\left(a_j\right), k\right) = \emptyset \Leftrightarrow dist(c) \geq 2k + 1$

- „Hinrichtung“:
  - Annahme: $dist(c) < 2k + 1$
  - $\exists a_i, a_j \in A: dist\left(c(a_i), c\left(a_j\right)\right) = m < 2k + 1$
  - es lässt sich also eine Folge von Codewörtern konstruieren: $c(a_i) = w_0, w_1, \dots, w_k, \dots, w_{2k} = c(a_j)$ mit
    - $dist(w_i, w_{i+1}) = 0$ oder
    - $dist(w_i, w_{i+1}) = 1 \; \forall i \in \{0, \dots, 2k\}$
  - dann gilt $w_k \in M(c(a_i), k) \wedge w_k \in M(c\left(a_j\right), k)$

Erinnerung:
$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$

Lemma (Fehlerkorrektur): Beweis

zu zeigen: $\forall a_i, a_j, i \neq j \; gilt: M(c(a_i), k) \cap M\big(c\big(a_j\big), k\big) = \emptyset \Leftrightarrow dist(c) \geq 2k + 1$

- „Rückrichtung“:
  - Annahme: $M(c(a_i), k) \cap M\big(c\big(a_j\big), k\big) \neq \emptyset$
  - $\exists w \in M(c(a_i), k) \cap M\big(c\big(a_j\big), k\big): dist(c(a_i), b) \leq k \wedge dist\big(c\big(a_j\big), b\big) \leq k$
  - dann können wir schreiben: $k + k \geq dist(c(a_i), b) + dist\big(c\big(a_j\big), b\big) = dist(c(a_i), b) + dist\left(b, c\big(a_j\big)\right) \geq dist\left(c(a_i), c\big(a_j\big)\right) \geq dist(c)$
  - dann gilt $dist(c) \leq 2k$
  - dann gilt $dist(c) < 2k + 1$

Erinnerung:
$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$

1-fehlerkorrigierender Code: Eigenschaft

Satz (1-fehlerkorrigierender Code):
Für einen 1-fehlerkorrigierenden Code c: $A \rightarrow \{0,1\}^{m+r}$ eines Alphabets A, mit $|A| = 2^m$ und $m \geq 3$, gilt der folgende Zusammenhang: $\boldsymbol{r \geq 1 + \lfloor \log_2 m \rfloor}$

Satz [1-fehlerkorrigierender Code]: Beweis

- Betrachte Umgebungskugel: $M_1(a) = \{w \in \{0,1\}^{m+r} | dist(w, c(a)) \leq 1\}$
  - es gilt: $|M_1(a)| = m + r + 1 \; \forall a \in A$
- Eigenschaft 1-fehlerkorrigierend heißt: $M_1(a_i) \cap M_1(a_j) = \emptyset, \forall a_i, a_j, \in A, i \neq j$
- Dann gilt: $2^m(m + r + 1) \leq 2^{m+r}$
- Lässt sich vereinfachen zu: $m + r + 1 \leq 2^r$

Satz [1-fehlerkorrigierender Code]: Beweis

zu zeigen: $(m + r + 1) \leq 2^r \Rightarrow r \geq 1 + \lfloor \log_2 m \rfloor$

- Annahme: $m = 2^k + l$ und $k, l \in \mathbb{N}$ mit $2^k > l \geq 0, k\ maximal$
- Dann gilt: $(m + r + 1) \leq 2^r \Leftrightarrow 2^k + l + r + 1 \leq 2^r$
  - $2^k + l + r + 1 \leq 2^r \Rightarrow k < r$
  - $k < r \Leftrightarrow 1 + k \leq r$
  - $1 + k \leq r \Rightarrow 1 + \lfloor \log_2 m \rfloor \leq r$

Hamming-Code

- 1-fehlerkorrigierend
  - Erweiterung um r Bitstellen
  - r minimal mit $m + r \leq 2^r - 1$ (entspricht exakt Bedingung aus Satz über die minimale Länge)
  - Alphabet A mit $|A| = 2^m, m = 2^k$ ergibt $m + 1 + \lfloor \log_2 m \rfloor$ Bitstellen
- Benutzung der Bitstellen an Zweierpotenzstellen als Prüfbits $(2^0, 2^1, 2^2, \dots)$
  - Bitstelle 2^j überprüft die Bitstellen, deren Position in Binärdarstellung an der j-ten Stelle eine 1 haben
  - Belegung der Bitstelle 2^j ergibt sich aus der Parität der dortigen Belegungen

Hamming-Code: Beispiel

- Annahme: Wort ist 0111 0101 0000 1111

- Algorithmus:
  - Bestimme Parameter: $m = 16, r = 5$

Hamming-Code: Beispiel

- Annahme: Wort ist 0 1110 _101 0000 _111 _1_ _
- Algorithmus:
  - Bestimme Parameter: $m = 16, r = 5$
  - Für jede Position: Bestimme relevante Prüfbits und fülle sie mit Wortbelegung an dieser Position

Position	2^4	2^3	2^2	2^1	2^0	Wort
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
Prüfbit						

Hamming-Code: Beispiel

- Annahme: Wort ist 0 1110 _101 0000 _111 _1_ _
- Algorithmus:
  - Bestimme Parameter: $m = 16, r = 5$
  - Für jede Position: Bestimme relevante Prüfbits und fülle sie mit Wortbelegung an dieser Position
  - Für jedes Prüfbit: Bestimme Parität $(\sum_{i=1} x_i)\ mod\ 2$

Position	2^4	2^3	2^2	2^1	2^0	Wort
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
Prüfbit						

Hamming-Code: Beispiel

- Annahme: Wort ist 0 1110 _101 0000 _111 _1_ _
- Algorithmus:
  - Bestimme Parameter: $m = 16, r = 5$
  - Für jede Position: Bestimme relevante Prüfbits und fülle sie mit Wortbelegung an dieser Position
  - Für jedes Prüfbit: Bestimme Parität $(\sum_{i=1} x_i)\ mod\ 2$
  - Füge Prüfbits in Wort ein
- Ergebnis: 0111 01101 0000 01110100

Position	2^4	2^3	2^2	2^1	2^0	Wort
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
Prüfbit	**1**	**0**	**0**	**0**	**0**	

Hamming-Code: Fehlererkennung/-korrektur

- Annahme: Wort war 0111 01101 0000 01110100

Hamming-Code: Fehlererkennung/-korrektur

- Annahme: Wort war 0111 01100 0000 01110100

Hamming-Code: Fehlererkennung/-korrektur

- Annahme: Wort war 0111 01100 0000 01110100

Fehler

- Algorithmus:
  - Bestimme Parameter: $m = 16, r = 5$
  - Für jede Position: Bestimme relevante Prüfbits und fülle sie mit Wortbelegung an dieser Position

Position	2^4	2^3	2^2	2^1	2^0	Wort
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
Prüfbit	1	0	0	0	0	

Hamming-Code: Fehlererkennung/-korrektur

- Annahme: Wort war 0111 01100 0000 01110100

Fehler

- Algorithmus:
  - Bestimme Parameter: $m = 16, r = 5$
  - Für jede Position: Bestimme relevante Prüfbits und fülle sie mit Wortbelegung an dieser Position
  - Für jedes Prüfbit: Bestimme Parität $(\sum_{i=1} x_i)\ mod\ 2$

Position	2^4	2^3	2^2	2^1	2^0	Wort
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
Prüfbit	1/1	0/1	0/1	0/0	0/1	

Hamming-Code: Fehlererkennung/-korrektur

- Annahme: Wort war 0111 01100 0000 01110100

Fehler

- Algorithmus:
  - Bestimme Parameter: $m = 16, r = 5$
  - Für jede Position: Bestimme relevante Prüfbits und fülle sie mit Wortbelegung an dieser Position
  - Für jedes Prüfbit: Bestimme Parität $(\sum_{i=1} x_i)\ mod\ 2$
  - Überprüfe Prüfbits
- Ergebnis: Fehler bei $2^3 + 2^2 + 2^0 = 13$

Position	2^4	2^3	2^2	2^1	2^0	Wort
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
Prüfbit	1/1	0/1	0/1	0/0	0/1	

Häufigkeitsabhängige Codes

- unterschiedliche Häufigkeit von Zeichen zur Längenreduktion nutzen
  - häufig vorkommende Zeichen: kurzes Code-Wort
  - selten vorkommende Zeichen: langes Code-Wort
- unterschiedliche Formen je nach Voraussetzung
  - bekannte Häufigkeitsverteilung: **statische Kompression**
  - unbekannte Häufigkeitsverteilung: **dynamische Kompression**
- bekanntester Häufigkeitsabhängiger Code: **Huffman-Code**
  - baut binären Häufigkeitsbaum entsprechend der Häufigkeit der Zeichen (klein nach groß)
  - Belegung der Kindkanten mit 0 und 1
  - Pfad von Wurzel zu Zeichen ergibt Code-Wort

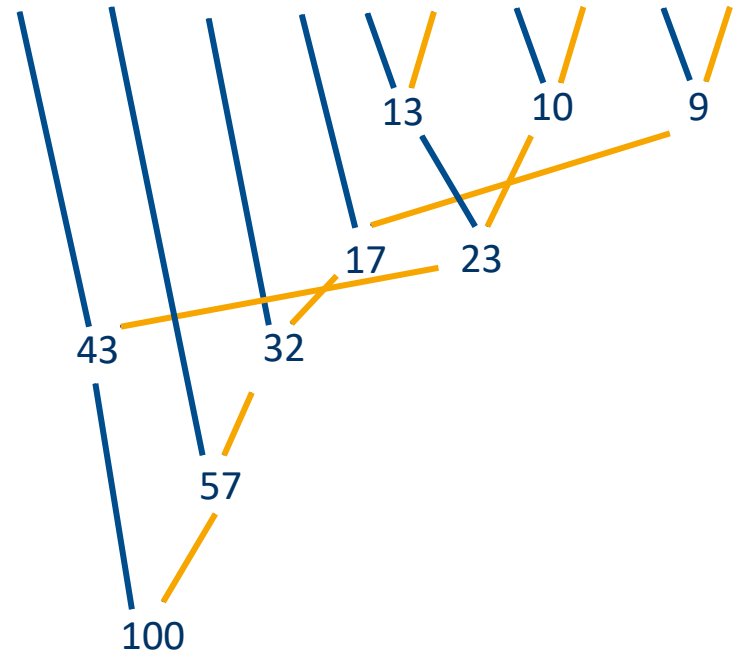
Huffman-Code: Beispiel

Annahme: Häufigkeitsverteilung wie in Tabelle

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

Algorithmus:

- Baue binären Baum durch Verknüpfung der kleinsten Häufigkeiten zu einem neuen Knoten
- Markiere linke Kanten mit **0**, rechte mit **1**



Ergebnis:

Zeichen	a	b	c	d	e	f	g	h	i	j
Häufigkeit [%]	20	25	15	8	7	6	5	5	5	4
Code	00	10	110	1110	0100	0101	0110	0111	11110	11111

Zusammenfassung

- Grundlegende Definitionen für Codes
- Fehlererkennung, Fehlerkorrektur
- Häufigkeitsunabhänigige Codes:
  - Parity Check
  - **Hamming-Code**
  - viele weitere Verfahren:
    - zweidimensionaler Parity Check
    - CRC (Cyclic Redundancy Check): Standard bei Übertragungen über lokale Netzwerke / das Internet
- Häufigkeitsabhängige Codes: **Huffman-Code**