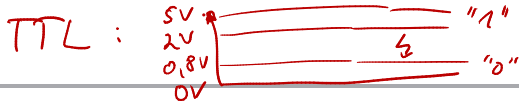


Motivation



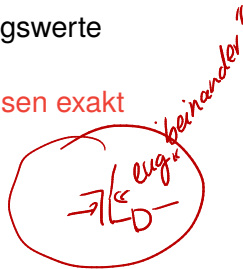
- Ein Rechner speichert, verarbeitet und produziert Informationen.
- Alle Ergebnisse müssen als Funktion der Anfangswerte exakt reproduzierbar sein.

→ Informationsspeicherung und Verarbeitung müssen exakt sein.

schön noise

- Probleme: Noise, Crosstalk, Abschwächung

→ Es gibt keine exakte Datenübertragung oder Datenspeicherung.



→ Ziel: Quantisierung der Informationsspeicherung mit Signal groß gegenüber maximaler Störung

- Binär-Codierung (nur zwei Zustände) ist die einfachste (und sicherste) Signal-Quantisierung.
- BIT (0, 1) als grundlegende Informationseinheit

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

- Wie werden im Rechner Zeichen dargestellt ?
- Codes fester Länge
- “Längenoptimale Kodierungen” von Zeichen:
Häufigkeitscodes (Bsp.: Huffman-Code)

Kapitel 2 – Kodierung

1. Kodierung von Zeichen

2. Kodierung von Zahlen

3. Anwendung: ReTI *→ Kodierung von Befehlen*

Albert-Ludwigs-Universität Freiburg

Dr. Tobias Schubert, Dr. Ralf Wimmer

Professur für Rechnerarchitektur

WS 2016/17

Definition

Eine nichtleere Menge $A = \{a_1, \dots, a_m\}$ heißt (endliches)
Alphabet der Größe m .

a_1, \dots, a_m heißen Zeichen des Alphabets.

- $A^* = \{\underline{w} \mid \underline{w} = b_1 \dots b_n \text{ mit } n \in \mathbb{N}, \forall i \text{ mit } 1 \leq i \leq n : \underline{b_i} \in A\}$ ist die Menge aller Wörter über dem Alphabet A .
- $|\underline{b_1 \dots b_n}| := n$ heißt Länge des Wortes $b_1 \dots b_n$.
- Das Wort der Länge 0 wird mit ε bezeichnet.

Beispiel:

Sei $A = \{\underline{a}, \underline{b}, \underline{c}, \underline{d}\}$.

Dann ist $bcada$ ein Wort der Länge 5 über A .

ε A^*

$$a_1 \in A \Rightarrow c(a_1) = 01001$$

Sei $A = \{a_1, \dots, a_m\}$ ein endliches Alphabet der Größe m .

- Eine Abbildung $c: A \rightarrow \{0, 1\}^*$ oder $c: A \rightarrow \{0, 1\}^n$ heißt Code, falls c injektiv ist.

Codewörter können unterschiedliche Länge haben „ $c: A \rightarrow \{0, 1\}^n$ “ mit $n! = 1, 2, 3, \dots$

↳ jedes Codewort hat Länge n !

- 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:
 $n \geq \lceil \log_2 m \rceil$.

- Ist $n = \lceil \log_2 m \rceil + r$ mit $r > 0$, so können die r zusätzlichen Bits zum Test auf Übertragungsfehler verwendet werden (siehe Kap. 6).

↳ Linkseindeutig, d.h. für jedes Codewort gibt es genau ein $a_i \in A$.

- Die Kodierung eines jeden Zeichens besteht aus n Bits.
 - ASCII (American Standard Code for Information Interchange): 7 Bits (es gibt Erweiterungen mit 8 Bits)
 - EBCDIC: 8 Bits
 - Unicode: 16 Bits
 - Vgl. „Rechnerarchitektur“
- Diese Kodierungen sind recht einfach zu behandeln. Unter Umständen wird für sie aber mehr Speicherplatz gebraucht als unbedingt nötig.

Beispiel: ASCII-Tabelle

erste 3 Bits

letzte 4 Bits

	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1
0000	nul	dle		0	@	P	.	p
0001	soh	dc1	!	1	A	Q	a	q
0010	sfx	dc2	"	2	B	R	b	r
0011	etx	dc3	#	3	C	S	c	s
0100	eot	dc4	\$	4	D	T	d	t
0101	enq	nak	%	5	E	U	e	u
0110	ack	syn	&	6	F	V	f	v
0111	bel	etb	'	7	G	W	g	w
1000	bs	can	(8	H	X	h	x
1001	ht	em)	9	I	Y	i	y
1010	lf	sub	*	:	J	Z	j	z
1011	vt	esc	+	;	K	[k	{
1100	ff	fs	,	<	L	\	l	
1101	cr	qs	-	=	M]	m	}
1110	so	rs	.	>	N	^	n	*
1111	si	us	/	?	O	_	o	del

Steuerzeichen

Schriftzeichen

Codierung dieser 10 Zeichen je einmal je 7 Bit

Carrage Return

Häufigkeitsabhängige Codes

- **Ziel:** Reduktion der Länge einer Nachricht durch Wahl **verschieden langer Codewörter** für die verschiedenen Zeichen eines Alphabets (also kein Code fester Länge!)
- **Idee:**
 - Häufiges Zeichen → kurzer Code
 - Seltenes Zeichen → langer Code
- **Voraussetzungen:**
 - Häufigkeitsverteilung ist bekannt → **statische Kompression**
 - Häufigkeitsverteilung ist nicht bekannt → **dynamische Kompression**
 - 1) *Stream alle Zeichen, zähle die Häufigkeiten*
 - 2) *kodieren*
 - 3) *2) Verteilung dem „dekodierer“ mit-schicken.*

Huffman-Code

- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

- 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.

Huffman-Code

- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

■ Beispiel:

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

9


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

Huffman-Code

- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

■ 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.

Huffman-Code

- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

■ Beispiel:

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

13 10 9

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

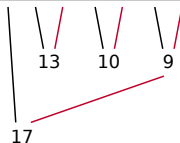
Huffman-Code

- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

■ 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.



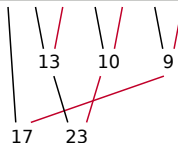
Huffman-Code

- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

■ 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.



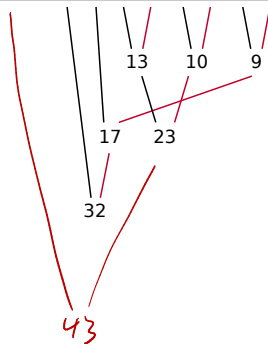
Huffman-Code

- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

■ 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.

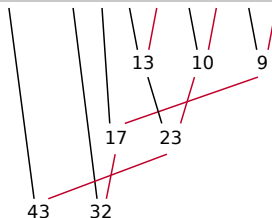


Huffman-Code

- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

■ 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.

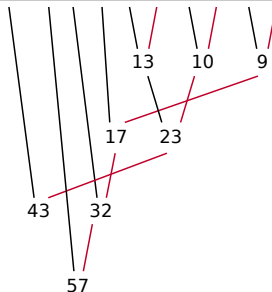
Huffman-Code

- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

■ 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.



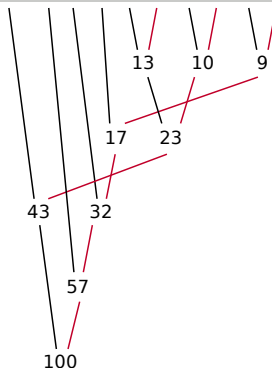
Huffman-Code

- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

■ 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.



Huffman-Code

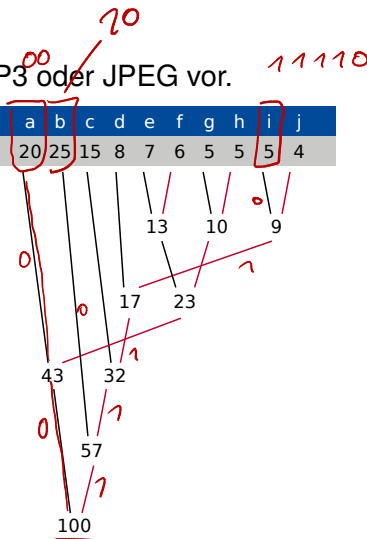
- Der **Huffman-Code** ist der bekannteste häufigkeitsabhängige Code.
- Kommt als Teilschritt z.B. in MP3 oder JPEG vor.

- Beispiel:

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

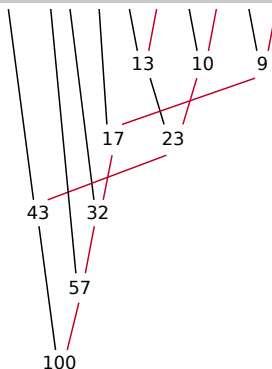
Baue binären Baum, indem die beiden kleinsten Häufigkeiten jeweils zu einem neuen Knoten addiert werden.

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



Erzeugte Huffman-Kodierung

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



Erzeugte Kodierung:

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

Huffman-Code: Dekodierung

Erzeugte Kodierung:

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

Problem:

a = 01
b = 011

- 1 Lesen des Bitstromes bis Symbol erkannt wurde.
- 2 Erkanntes Symbol ausgeben und weiter mit 1.

↑
Schnitt

Bitstrom:
011.....

ist
das "a" mit 1
oder ist das "b"?

Definition

Sei A ein Alphabet der Größe m .

*"01" ist Präfix
von "011"*

- $a_1 \dots a_p \in A^*$ heißt **Präfix** von $b_1 \dots b_l \in A^*$, falls $p \leq l$ und $a_i = b_i \forall i, 1 \leq i \leq p$. *variabler Länge*

- Ein Code $c : A \rightarrow \{0, 1\}^*$ heißt **Präfixcode**, falls es kein Paar $i, j \in \{1, \dots, m\}$ gibt, so dass $c(a_i)$ Präfix von $c(a_j)$.

- Der Huffman-Code ist ein Präfixcode.
- Bei Präfixcodes können Wörter über $\{0, 1\}$ eindeutig dekodiert werden. (Sie entsprechen Binärbäumen mit Codewörtern an den Blättern.)
- Huffman-Code ist ein bzgl. mittlerer Codelänge optimaler Präfixcode (unter Voraussetzung einer bekannten Häufigkeitsverteilung) - ohne Beweis.

Definition

Sei A ein Alphabet der Größe m .

- $a_1 \dots a_p \in A^*$ heißt **Präfix** von $b_1 \dots b_l \in A^*$, falls $p \leq l$ und $a_i = b_i \forall i, 1 \leq i \leq p$.
- Ein Code $c : A \rightarrow \{0, 1\}^*$ heißt **Präfixcode**, falls es kein Paar $i, j \in \{1, \dots, m\}$ gibt, so dass $c(a_i)$ Präfix von $c(a_j)$.
 - Der Huffman-Code ist ein Präfixcode.
 - Bei Präfixcodes können Wörter über $\{0, 1\}$ eindeutig dekodiert werden. (Sie entsprechen Binärbäumen mit Codewörtern an den Blättern.)
 - Huffman-Code ist ein bzgl. mittlerer Codelänge optimaler Präfixcode (unter Voraussetzung einer bekannten Häufigkeitsverteilung) - ohne Beweis.



Frage: Welche dieser Codes sind Präfixcodes

a. $c('A') = \underline{01}$, $c('B') = \underline{110}$, $c('C') = \underline{011}$ → Kein!

b. $c('A') = \underline{01}$, $c('B') = \underline{110}$, $c('C') = \underline{111}$ → JA

c. $c('1') = xz$, $c('2') = xy$, $c('3') = yz$ → JA

d. Keiner der Obigen. ~ Aussage kann nicht stimmen!

- Es gibt zahlreiche Ansätze zur **Datenkompression**.
(Beispiel: Lempel-Ziv-Welch.)
- In Programmtexten gibt es häufig viele Leerzeichen, gleiche Schlüsselwörter und so weiter.
- Kodiere Folgen von Leerzeichen bzw. Schlüsselwörter durch kurze Codes.
- Das wird z.B. bei GIF und TIFF genutzt.
- Das soll auch funktionieren, wenn man noch nicht weiß, welche Zeichenketten häufig vorkommen.

