Motivation

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

$\rightarrow$ Informationsspeicherung und Verarbeitung müssen exakt sein.

- Probleme: Noise, Crosstalk, Abschwächung

$\rightarrow$ Es gibt keine exakte Datenübertragung oder Datenspeicherung.

$\rightarrow$ 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

Motivation

- 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.

$\rightarrow$ Kodierung!

Kapitel 2.1 - Kodierung von Zeichen

- 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

Albert-Ludwigs-Universität Freiburg

Dr. Tobias Schubert, Dr. Ralf Wimmer

Professur für Rechnerarchitektur
WS 2016/17

UNI FREIBURG

Alphabete und Wörter

Definition

Eine nichtleere Menge $A = \{a_1, \ldots, a_m\}$ heißt (endliches) Alphabet der Größe m.
$a_1, \ldots, a_m$ heißen Zeichen des Alphabets.

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

Beispiel:
Sei $A = \{a, b, c, d\}$.
Dann ist *bcada* ein Wort der Länge 5 über A.

Code

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

- Eine Abbildung $c : A \to \{0,1\}^*$ oder $c : A \to \{0,1\}^n$ 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 $c : A \to \{0,1\}^n$ heißt Code fester Länge.
- Für einen Code $c : A \to \{0,1\}^n$ 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).

Codes fester Länge

- 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

	000	001	010	011	100	101	110	111
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

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**:

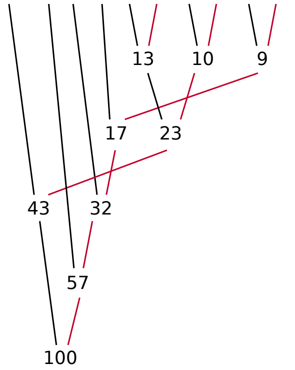
Häufiges Zeichen	$\rightarrow$	kurzer Code
Seltenes Zeichen	$\rightarrow$	langer Code

- **Voraussetzungen**:
  - Häufigkeitsverteilung ist bekannt $\rightarrow$ statische Kompression
  - Häufigkeitsverteilung ist nicht bekannt $\rightarrow$ dynamische Kompression

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

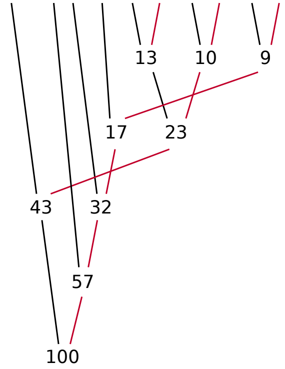


Baue binären Baum, indem die beiden kleinsten Häufigkeiten jeweils zu einem neuen Knoten addiert werden.

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

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

Präfixcodes

Definition

Sei A ein Alphabet der Größe m.

- $a_1 \ldots a_p \in A^*$ heißt Präfix von $b_1 \ldots b_l \in A^*$, falls $p \leq l$ und $a_i = b_i \; \forall i,\ 1 \leq i \leq p$.
- Ein Code $c : A \to \{0,1\}^*$ heißt Präfixcode, falls es kein Paar $i, j \in \{1, \ldots, m\}$ 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.

Weitere Verfahren

- Es gibt zahlreiche Ansätze zur Datenkompression. (Beispiel: Lempel-Ziv-Welch.)
- In Programmtexten gibt es häufig viele Leerzeichen, gleiche Schlüsselwörter und so weiter.

$\rightarrow$ 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.