Codierung u. Kompression

Codierung: bestimmtes **Format / Protokoll**, was zur Übertragung und Speicherung von Daten verwendet wird

Kompression: **Verringerung** der Dateigröße

verlustfrei	verlustbehaftet
alle Daten können wiederhergestellt werden	manche Daten gehen verloren
⊕ kein Datenverlust	⊕ höhere Kompressionsrate
z.B. Huffmann	z.B. JPEG

Kompressionsrate: $\frac{\text{unkomprimiert}}{\text{komprimiert}}$ %

Kompressionsfaktor: $\frac{\text{komprimiert}}{\text{unkomprimiert}}$

codierte Daten bestehen aus **Codewörtern**

haben gleiche Länge?
Dann ist es ein **Blockcode**

Block aus verfügbaren Zeichen
↓ Wortlänge ↓ Zeichenvorrat

kein Codewort ist Anfang eines anderen Codeworts?
Dann ist es ein **Präfixcode**
↓
immer eindeutig decodierbar

Alle Codewörter können im
ein Code { • Codebaum oder • Codetabelle } dargestellt werden

Lauflängencodierung (Run-Length-Encoding, RLE)

- Bei mehreren gleichen hintereinander vorkommenden Zeichen / Zahlen / Symbolen
- Übertragung des Zeichens einmal + dessen Anzahl
- z.B. "AAAA" → "A4"

⊕
- einfach
- schnell
- verlustfrei

⊖
- vergleichsweise ineffizient
- nur einsetzbar, wenn gleiche Zeichen oft hintereinander vorkommen

Huffman-Codierung

💡 kürzerer Code für häufig vorkommende Zeichen

! Vorkommenswkts müssen bekannt sein, Optimierung auf einen bestimmten Fall

⊕ verlustfrei, kürzestes präfixfreies Code

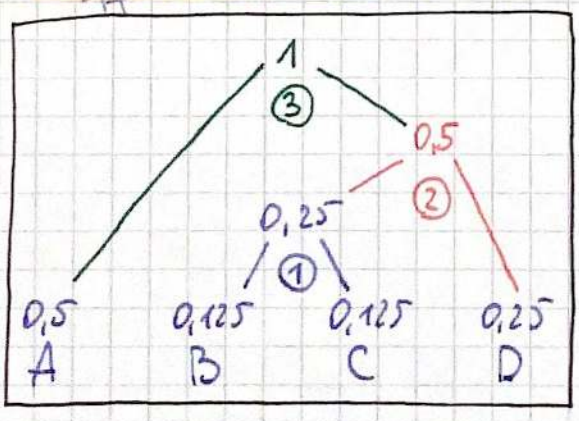
Vorgehen:

① "AABCADAD"

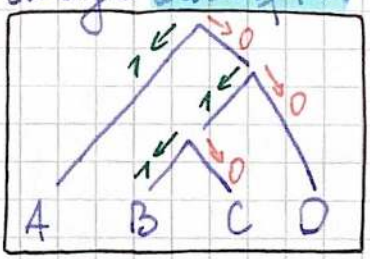
② Wkts ermitteln

A:	0,5
B:	0,125
C:	0,125
D:	0,25

③ Huffman-Baum erstellen



④ Zweige beschriften



1 links
0 rechts
} konsistent!

⑤ Codes ablesen und in Tabelle eintragen

A:	1
B:	011
C:	010
D:	00

(Pfade ablaufen)

die beiden kleinsten Wkts jeweils zusammenführen, bis der Baum vollständig ist

Fehlererkennung: Repetitionscodes

- Bei der Übertragung können Fehler passieren
- Idee: wir übertragen jedes Bit zwei- oder dreimal
- Problem: ineffizient

Fehlererkennung: Paritätsbit

- Parität: # der Einser-Bits
- Idee: an jedes Codewort / Block einen Paritätsbit anhängen

Codewort:	Parität:	Paritätsbit:
1000100	gerade	0
0110111	ungerade	1

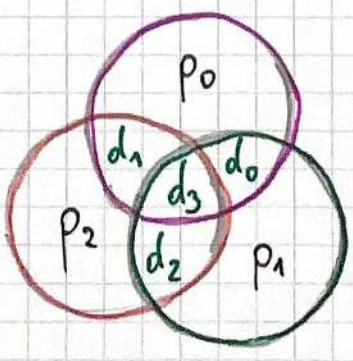
Fehlerkorrektur: (7,4)-Hamming-Code

- 4 Datenbits + 3 fehlerkorrigierende Bits = 1 Fehler kann korrigiert werden

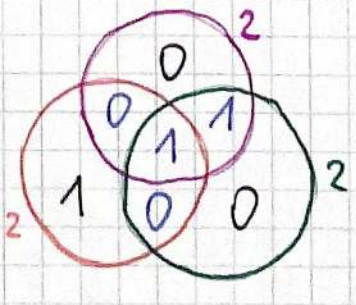
① 1001

② Reihenfolge: $p_0\, p_1\, d_0\, p_2\, d_1\, d_2\, d_3$ → $p_0\, p_1\, 1\, p_2\, 001$

③ p_0, p_1, p_2 so bestimmen, dass



→ Summe der Bits in jedem Kreis soll gerade sein



④ Ergebnis: 0011001