

Prof. Dr.-Ing. Dr. h. c. J. Becker

becker@kit.edu

Karlsruher Institut für Technologie (KIT)

Institut für Technik der Informationsverarbeitung (ITIV)

Digitaltechnik

Nachrichtenübertragung und Optimale Codes

Übertragung von Nachrichten und Daten

- Spezielle Anforderungen an Codierung:
 - Schutz gegen **Verfälschungen** + **typische Fehlerfälle**
 - **Darstellungsaufwand minimieren** (Kosten, Performanz)
- Weiteres Problem: Annahme des **Einzelfehlerfalls** trifft häufig in **Übertragungssystemen nicht zu**
 - **zeitlich konzentrierte Fehler**
(Störung über einen Zeitraum)
 - **Bündelstörungen** treten auf
- Also: **dedizierte zusätzliche Maßnahmen notwendig**

Bündelstörung

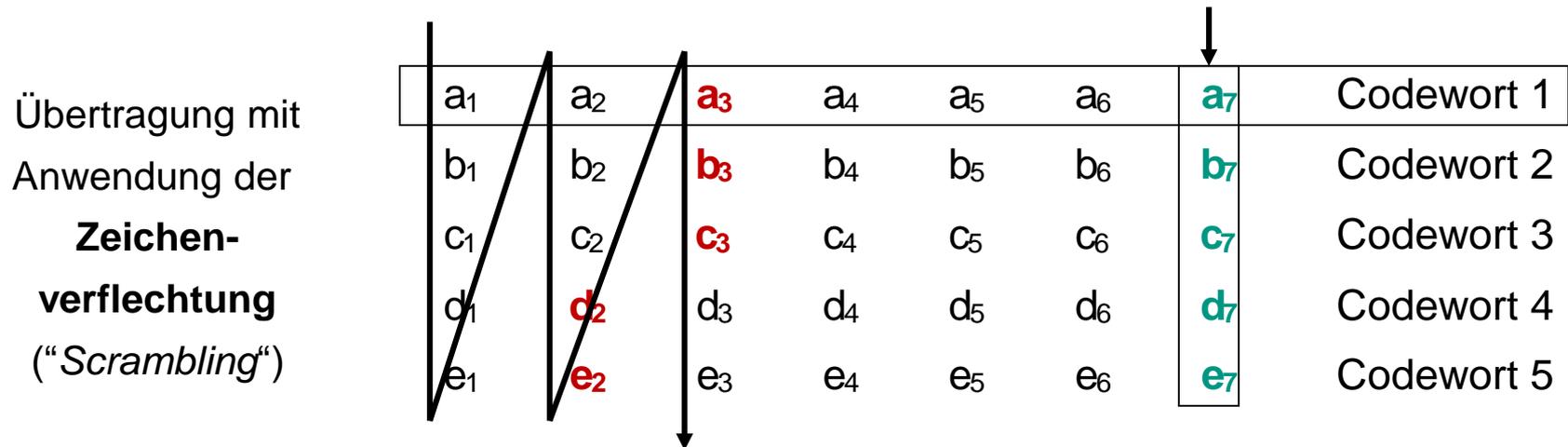
- **Problem:** eine längere Auswirkung einer Störung auf ein Signal
- Dadurch können **mehrere Bits** nacheinander **gestört** werden
- **Bündelfehler:** bisher besprochene Codes sind nicht geeignet

- **Lösung:** Bei der **Zeichenverflechtung** werden die **Binärstellen** eines Zeichens (Binärwort) **nicht** nacheinander übertragen
→ **Verwürfelung** oder **Scrambling** der Stellen

- **Also:** länger **anhaltender Störeinfluss** (Bündelstörung) bestimmter Länge **verfälscht** nur **Binärstellen** von **unterschiedlichen** Codewörtern

Bündelstörung: Zeichenverflechtung

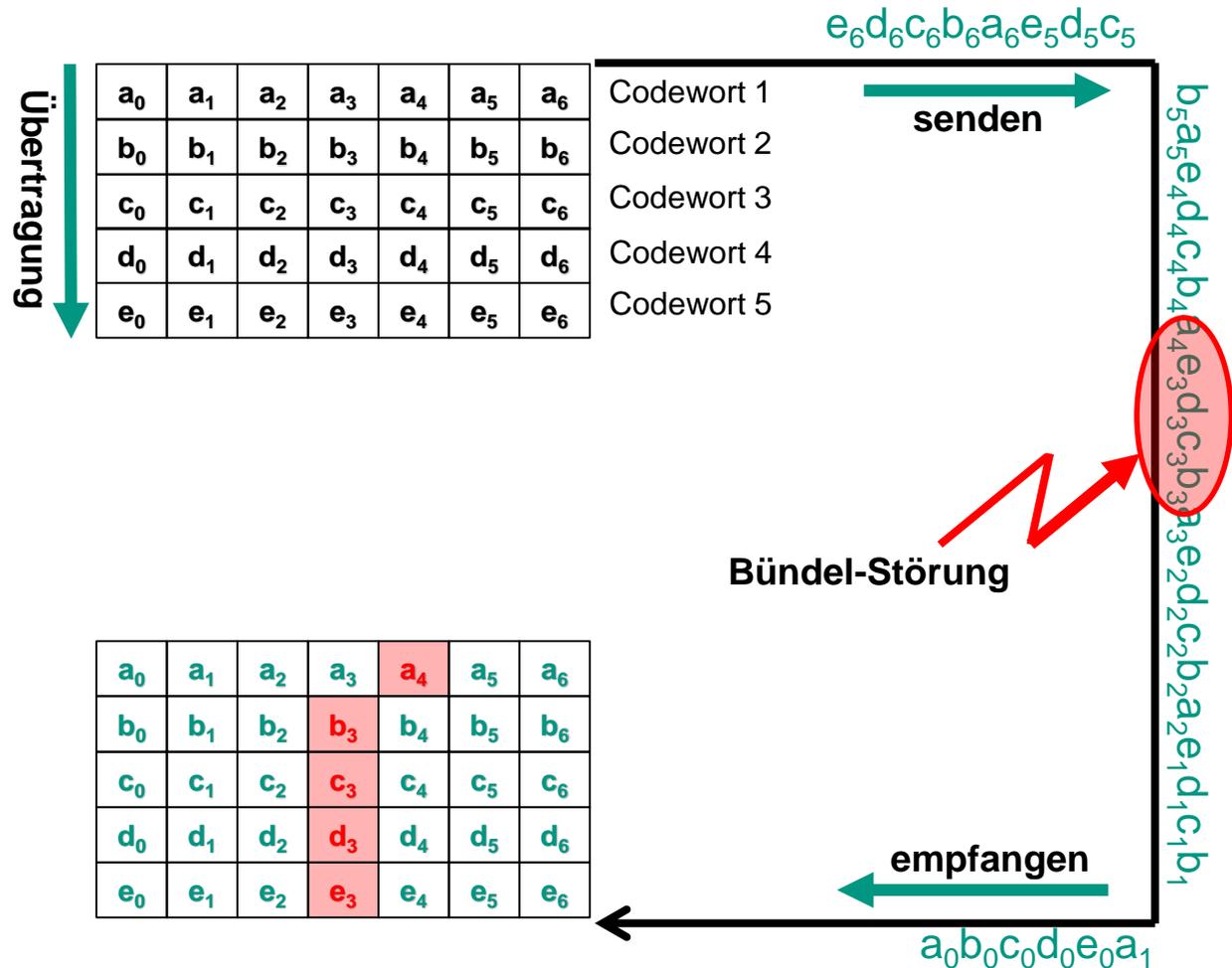
Geänderte Reihenfolge der Übertragung:



- Eine **Störung** von maximal **5 aufeinanderfolgenden Bits** ist damit **erkennbar**
- **Trotz andauernder Störung:**
 - pro Codewort nur **1 Bit verfälscht**
 - **Paritätsbit (7. Bit)** detektiert Einzelfehlerfälle

Bündelstörung: Fehlererkennung/Korrektur

Beispiel für eine Übertragung mit Bündelstörung:



Allgemein gilt:

erlaubt der benutzte Code Fehlererkennung bzw. Fehlerkorrektur → gesamte Bündelfehler ist erkennbar bzw. korrigierbar

- Neben der **Fehlersicherheit** spielt bei der Übertragung auch die **Länge einer Nachricht** eine Rolle
- Der **Morse-Code** hat schon früh die **statistischen Eigenschaften** der englischen Sprache ausgenutzt, um eine möglichst **kurze Darstellung** der **Nachricht** zu erzielen
- **Häufig auftretende Buchstaben** werden möglichst **kurzen Codewörtern** zugeordnet, während für seltene Buchstaben längere Codewörter gewählt werden
- **Beispiel Morse-Code:**

häufige Buchstaben

e: · t: -
i: .. a: ·-

seltene Buchstaben

j: ·--- q: ---
y: -... ö: ----

- **Optimale Codes** versuchen die im Mittel auftretende **durchschnittliche Codewortlänge m** zu **minimieren**
- Der **minimal erreichbare Idealwert** ist dabei der **durchschnittliche Informationsgehalt H** des Zeichenvorrats -> **Entropie** der Sendequelle
- Für einen Zeichenvorrat $\{x_1, x_2, x_3, \dots, x_n\}$ gilt:

∅ **Informationsgehalt H :**
$$H = \sum_{i=1}^n p(x_i) \cdot H_e(x_i) = \sum_{i=1}^n p(x_i) \cdot \log_2 \frac{1}{p(x_i)}$$

∅ **Codewortlänge m :**
 $m(x_i)$: Länge CW für X_i

$$\bar{m} = \sum_{i=1}^n p(x_i) \cdot m(x_i)$$

■ Konstruktionsvorschrift:

- **Zeichenvorrat** nach **aufsteigender Wahrscheinlichkeit** linear sortieren → **lineare Ordnung**
- **Zwei Teilmengen** so konstruieren, dass die **Summen-wahrscheinlichkeiten** der beiden Teilmengen möglichst **gleich groß** sind, d.h. die addierten Auftrittswahrscheinlichkeiten der in beiden Teilmengen enthaltenen Zeichen ergeben möglichst die gleiche Summe
- Die **erste Teilmenge** erhält das **Codierungszeichen "0"** und die **zweite Teilmenge eine "1"**
- Mit den jeweils **resultierenden Teilmengen rekursiv** in **gleicher Weise** fortfahren
→ bis nur noch 1 Zeichen in resultierenden Teilmengen enthalten ist

Shannon-Fano-Code

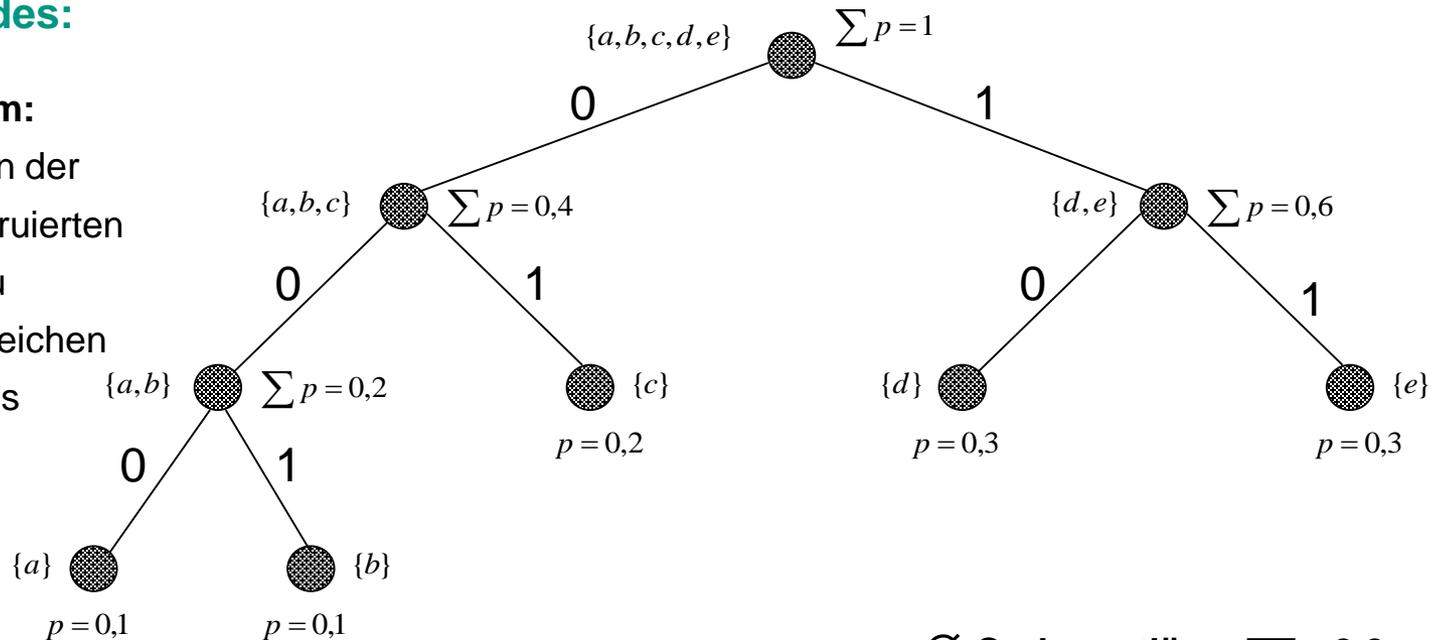
Beispiel:

Zeichen	a	b	c	d	e
Wahrscheinlichkeit	0,1	0,1	0,2	0,3	0,3

Konstruktion des optimalen Codes:

Codierungsbaum:

→ Repräsentation der rekursiv konstruierten Teilmengen zu codierender Zeichen und ihre Codes



∅ Codewortlänge $\bar{m} = 2,2$

Codierung:

Zeichen	a	b	c	d	e
Kodierung	000	001	01	10	11

Beschreibung der Eigenschaften und Konstruktion

- Effiziente Codierungstechnik für Datenkompression (*JPEG, MPEG*)
- Codierung mit variabler Bitlänge der Codewörter
 - **effizienter als Codierung mit fester Bitlänge**
- Präfixfreie-Codierung
 - **kein Kodewort ist Präfix eines anderen Codewortes**
- Vorgehensweise graphisch darstellbar (*binärer Baum*, siehe auch Graphentheorie in Kap. 5.3 im Buch)
 - **Blätter repräsentieren Codewörter**
 - **der Weg dorthin das eigentliche Codewort**
- Vorgehensweise Huffman-Codierung: *Greedy-Algorithmus* (*greedy* = “gierig”)
→ **nutzt Liste mit Auftrittswahrscheinlichkeiten der zu codierenden Zeichen**

Huffman's Codierungs-Algorithmus (HKA) (1)

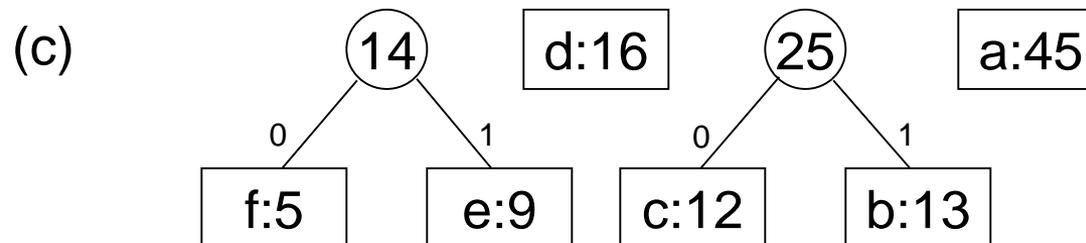
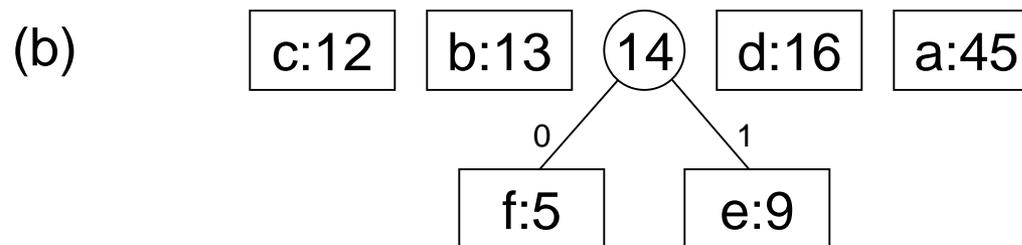
- HKA ist ein *Greedy-Algorithmus*
- HKA konstruiert einen *Baum* (= azyklischer zusammenhängender Graph) T , der die *Codierung* der einzelnen Zeichen repräsentiert
- *Sortierte Liste Q* → jeweils die beiden Zeichen in Q mit niedrigsten Auftrittswahrscheinlichkeiten (AWS) finden
- Zwei gefundene Zeichen werden zu neuem *Element z* “verschmolzen”, dessen AWS die Summe der beiden Einzelwahrscheinlichkeiten zugeordnet wird
 - *Einsortierung von z in Q entsprechend seiner AWS*
- *Abbruch des HKA:*
nur noch 1 Objekt in Q übrig → *Codierungsbaum T*

Huffman's Codierungs-Algorithmus (HKA) (2)

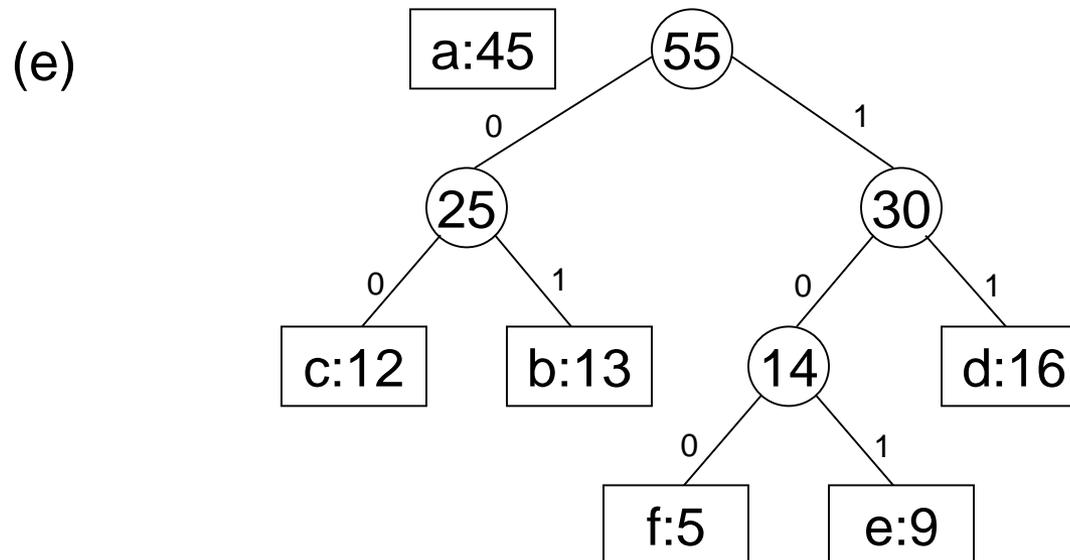
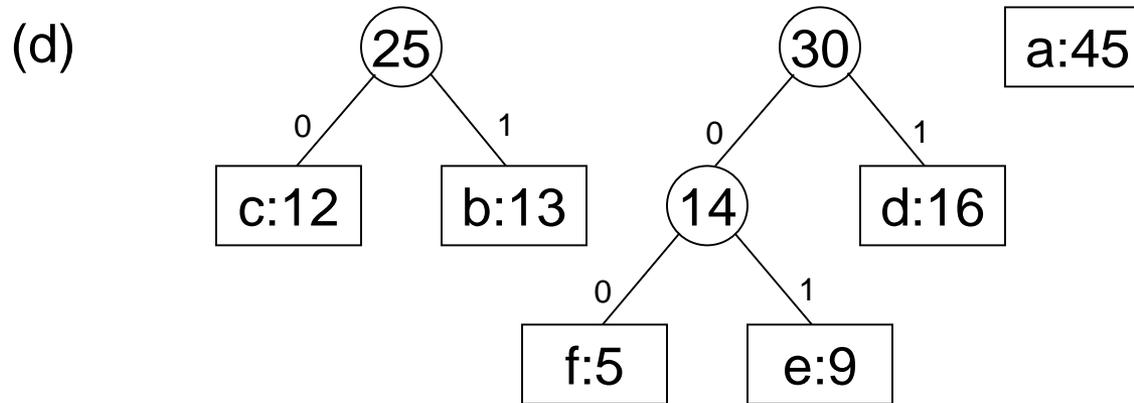
```
begin
n ← |C|;           // C ist sortierte Menge der zu codierenden Elemente
Q ← C;           // Q ist sortierte Liste (-> verwendete Datenstruktur)
for i ← 1 to n-1 do
    z ← ALLOCATE-NODE(); // Erstellung eines neuen Knotens z im Codierbaum
    x ← EXTRACT-MIN(Q); // Bestimme Knoten minimaler Auftrittswahrscheinlichkeit (AWS)
                        // in Q, entferne diesen aus Q und speichere ihn in x
    y ← EXTRACT-MIN(Q); // Bestimme Knoten minimaler AWS in Q, entferne diesen aus
                        // Q und speichere ihn in Y
    left[z] ← x;       // Knoten x wird linker Nachfolgeknoten von z im Codierbaum
    right[z] ← y;      // Knoten y wird rechter Nachfolgeknoten von z im Codierbaum
    f[z] ← f[x] + f[y]; // Bestimmung der AWS des vereinten Knotens z
    INSERT(Q, z);      // neuer Knoten z wird in Q (= Codierbaum) entsprechend seiner AWS
end;
return EXTRACT-MIN(Q); // letztes verbleibendes Listenelement in Q (= Wurzel des Codierbaums)
                        // wird als Endergebnis ausgegeben
end
```

Optimale Codes: Huffman-Code

Beispiel:

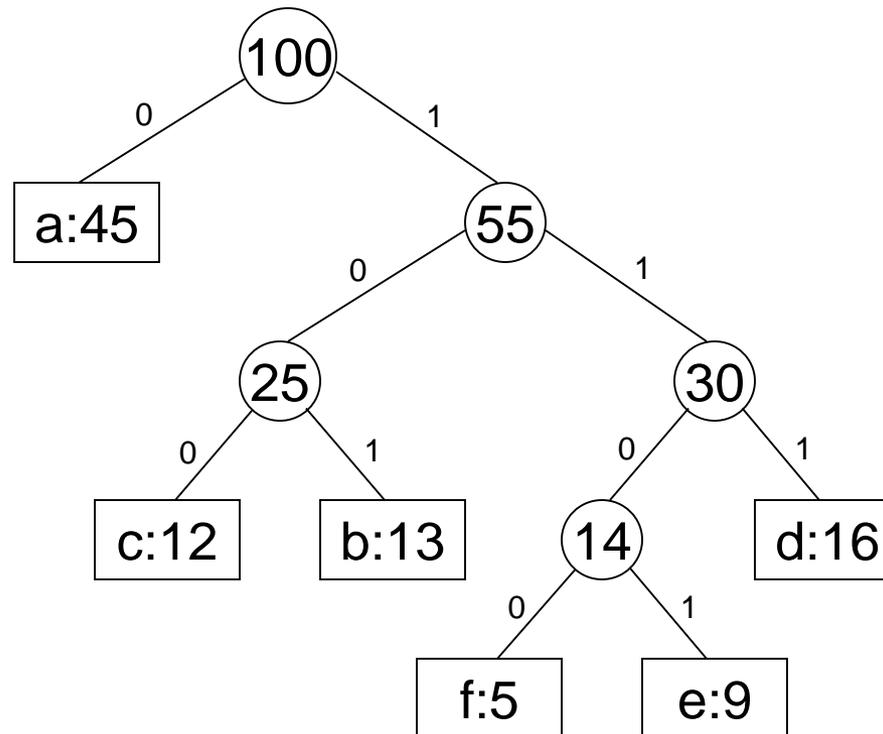


Optimale Codes: Huffman-Code



Optimale Codes: Huffman-Code

(f)



Anwendung im JPEG-Format:

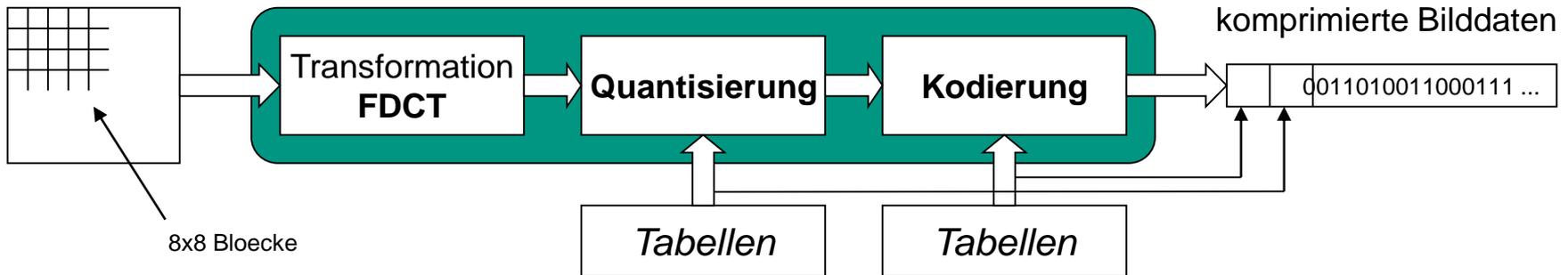
- **JPEG: Joint Photographic Experts Group**
- **Ziel:** allgemeiner **Standard** für alle Arten von **Bildern** (Color oder Graustufen), die dadurch “portabel” sind
- Ordentliche **Kompressionsraten**
- Dafür **leichter Verlust** an **Bildinformation**
- Verlustfreie Kompression möglich

Optimale Codes: JPEG-Verfahren

Übersicht des Komprimierungsverfahrens JPEG:

JPEG Komprimierer

Eingabe: Ursprungsbild



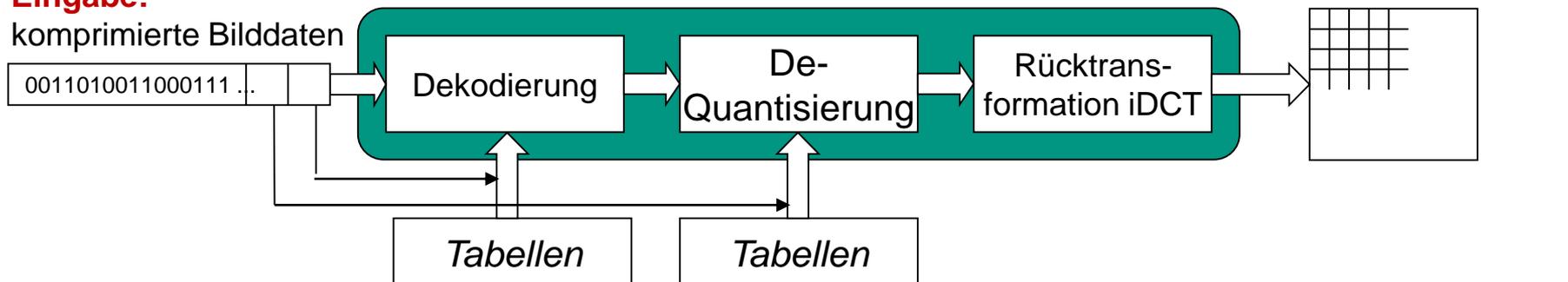
Ausgabe:

komprimierte Bilddaten

JPEG Dekomprimierer

Eingabe:

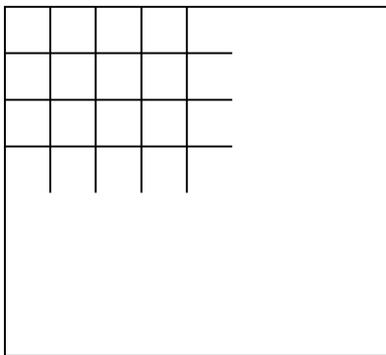
komprimierte Bilddaten



Ausgabe: Ursprungsbild

- **Aufteilung des zu komprimierenden Bildes in 8x8-Pixelblöcke**
 - **Geeignete Blockgröße** für die **FDCT** um **digitaltechnischen Realisierungsaufwand zu optimieren und** den Gesamtaufwand nicht zu hoch zu treiben
 - Bei größeren Blöcken steigt der Rechen- bzw. Kostenaufwand quadratisch an

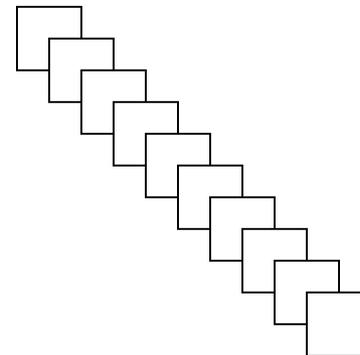
Ursprungsbild



Aufteilung



8x8 Pixelblöcke



Optimale Codes: JPEG-Verfahren

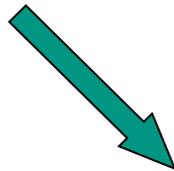
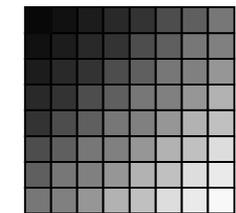
- **Schnelle diskrete Cosinustransformation (FDCT)** liefert per Block eine Koeffizientenmatrix zurück

- FDCT transformiert Daten aus **Ortsbereich $f(i,j)$** in **Frequenzbereich $F(u,v)$**

- **Formel für die FDCT:**

$$F(u, v) = \frac{1}{4} F_u F_v \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos \frac{(2y+1)u\pi}{16} \cos \frac{(2x+1)v\pi}{16}$$

$$F_u, F_v = \frac{1}{\sqrt{2}}, \text{ für } u, v = 0 \quad F_u, F_v = 1, \text{ für } u, v = 1 \dots 7$$



$f(x, y)$

-84	-75	-66	-57	-49	-40	-31	-22
-75	-65	-54	-43	-33	-22	-11	-1
-66	-54	-54	-29	-17	-4	8	21
-57	-43	-41	-15	-1	14	28	42
-49	-32	-29	-1	15	31	41	63
-40	-22	-17	14	31	49	67	85
-31	-11	8	28	47	67	87	106
-22	-1	21	42	63	85	106	127

0,0 7,7



$F(u, v)$

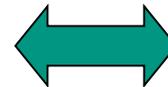
-1	-275	0	-29	0	-7	0	-2
-275	74	0	7	0	2	0	0
0	0	0	0	0	0	0	0
-29	7	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-7	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-2	0	0	0	0	0	0	0

0,0 7,7

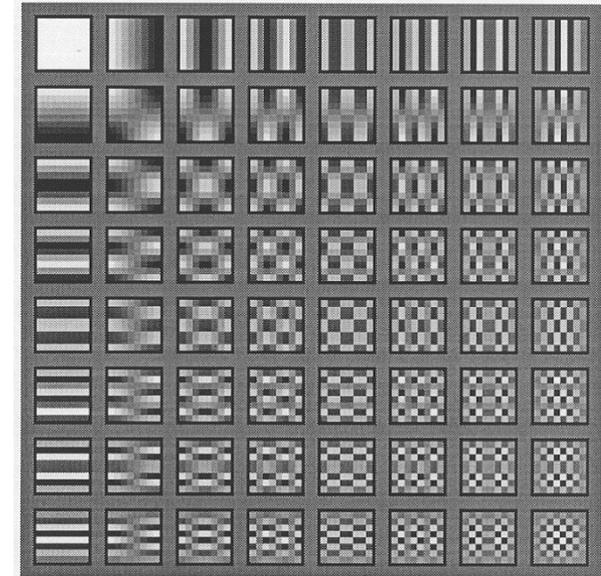
JPEG-Verfahren: FDCT

$F(u, v)$

	0,0							0,7
	-1	-275	0	-29	0	-7	0	-2
	-275	74	0	7	0	2	0	0
	0	0	0	0	0	0	0	0
	-29	7	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	-7	2	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	-2	0	0	0	0	0	0	0
	7,0							7,7



64 Basisfunktionen



Einfluss der Frequenzen eines Blocks auf die transformierte Matrix

- Einfluss der **Bildveränderlichkeit** (-> **Frequenzen!**) auf **einzelne Koeffizienten** der transformierten Matrix
- **Obere linke Koeffizient** (DC-Anteil) wird **größer**, je **monotoner** die **Farbverteilung** ist
- **Untere rechte Koeffizient** wird durch eine **möglichst ungleichmäßige Verteilung** der Werte der zu transformierenden **Pixelmatrix beeinflusst**

Optimale Codes: JPEG-Verfahren

- **Quantisierungsmatrix:** möglichst **viele "Nullen"** in **Transformationsmatrix**
- **Quantisierung:** eigentliche **verlustbehaftete Vorgang** in der JPEG-Kodierung
- **Verlust:** durch **Rundung** nach Division durch **Quantisierungsfaktor Q(u,v)**

JPEG-Verfahren:
Quantisierung

$$\text{Formel: } F^Q(u, v) = \text{round}\left(\frac{F(u, v)}{Q(u, v)}\right), F^Q(u, v) = F^Q(u, v) * Q(u, v)$$

Quantisierungsmatrix:

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	58	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Bsp.: $F(3,0) = -29, Q(3,0) = 16$

$$\Rightarrow F^Q(3,0) = \text{round}\left(\frac{-29}{16}\right) = \text{round}(-1,8125) = \underline{-2}$$

nach der De-Quantisierung : $F^Q(3,0) = -2 \cdot 16 = \underline{-32}$

-1	-275	0	-29	0	-7	0	-2
-275	74	0	7	0	2	0	0
0	0	0	0	0	0	0	0
-29	7	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-7	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-2	0	0	0	0	0	0	0

Quantisierung



0	-25	0	-2	0	0	0	0
-23	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

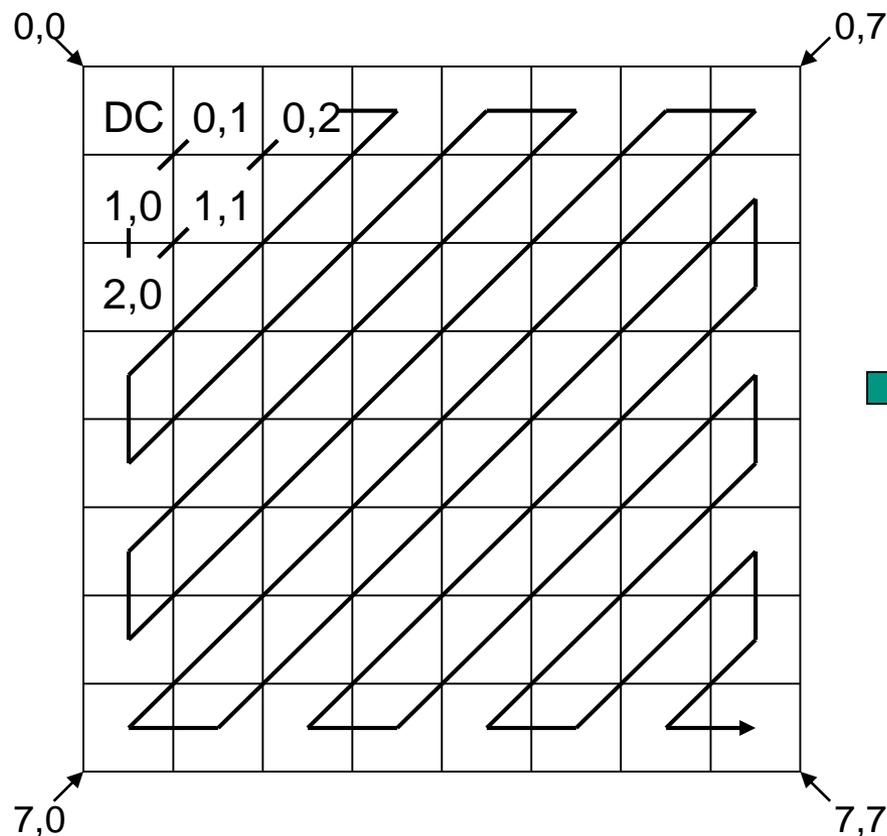
De-Quantisierung



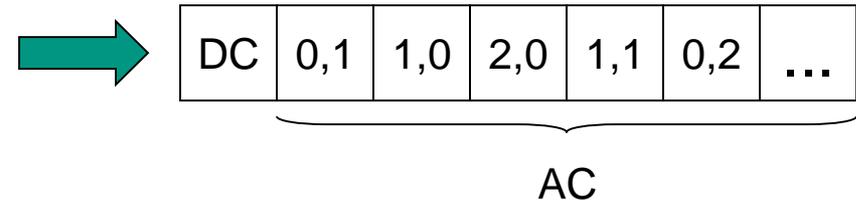
0	-275	0	-32	0	0	0	0
-276	72	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-28	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

JPEG-Verfahren: Kodierung (1) – ZigZag Kodierung

- Die **8x8 Koeffizientenmatrix** (Frequenzbereich) **nach Cosinus-Transformation** mittels des **Zig-Zag-Verfahrens** in "Stream" von **64 Elementen** umformen
-> Dadurch wird **Sequenz** mit möglichst **langen Nullfolgen** gebildet



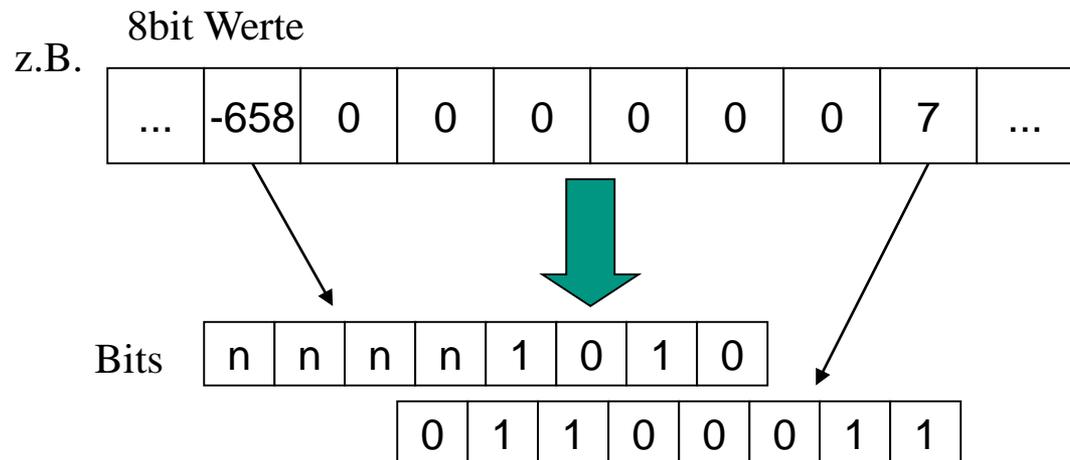
JPEG – ZigZag-Scan



JPEG-Verfahren: Kodierung (2) – Run-Length Kodierung

- aus ZigZag-Scan entstandene Streams werden nun komprimiert
- **zuerst: Run-Length-Encoding**, um lange Nullfolgen platzsparend zu kodieren
- ZigZag-geordnete AC-Koeffizienten werden so kodiert, dass nur **AC-Koeffizienten ungleich Null** in Binärvektorform (**RRRRSSSS**) dargestellt werden, **wobei:**
 - **RRRR** (4-Bit) -> **Anzahl der Nullen vor dem aktuellen Koeffizienten** angibt
 - **SSSS** (4-Bit) -> **Wertebereiche der Koeffizienten** gemäß unterer Tabelle repräsentiert
- der **SSSS-Subvektor** bestimmt **höherwertige Bits** der **AC-Koeffizienten**, die **niederwertigen Bits** werden **unkodiert** an das **RS-Symbol** angehängt

SSSS	AC-Koeffizienten
1	-1,1
2	-3...-2,2...3
3	-7...-4,4...7
usw.	
10	-1023...-512,512...1023



JPEG-Verfahren: Kodierung (3) – Huffman Kodierung

- **letzte Schritt:** gebildete RS-Elemente werden **Huffman-kodiert**
- **Betrachtet:** **Häufigkeit** des **Auftretens** der **RS-Werte**
- **häufig vorkommende Elemente:** **kurz kodiert**
selten vorkommende Elemente: **lang kodiert**
- **entweder:** ein **Huffman-Kode** aus **gegebenen Daten (AWS)** ermittelt,
oder: eine **Standardtabelle** verwendet
(kleiner Ausschnitt s. Tabelle)
- **Huffman-Codierung** ist **verlustfrei**
- Mit **verschiedenen Tabellen** werden unterschiedliche **unterschiedliche Kompressionsraten** erreicht
- **niederwertige Bits** der **AC-Koeffizienten** werden **unkodiert** belassen

Kodierungstabelle

RRRR	SSSS	Huffman-Code-Wort
0000	0000	1010 (EOB)
0000	0001	00
0000	0010	01
0000	0011	100
0000	0100	1011
0000	0101	11010
0000	0110	1111000
0000	0111	11111000
0000	1000	1111110110
0000	1001	1111111110000010
0000	1010	1111111110000011



Originalbild



25x Komprimiert



75x Komprimiert

**JPEG – Kompression
Beispiele**

Optimale Codes: JPEG-Verfahren



25x Komprimiert



50x Komprimiert



99x Komprimiert