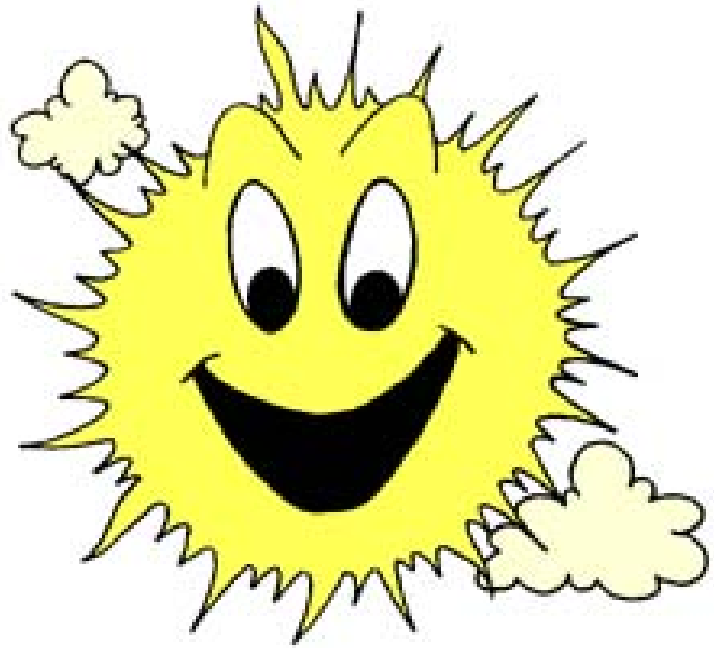
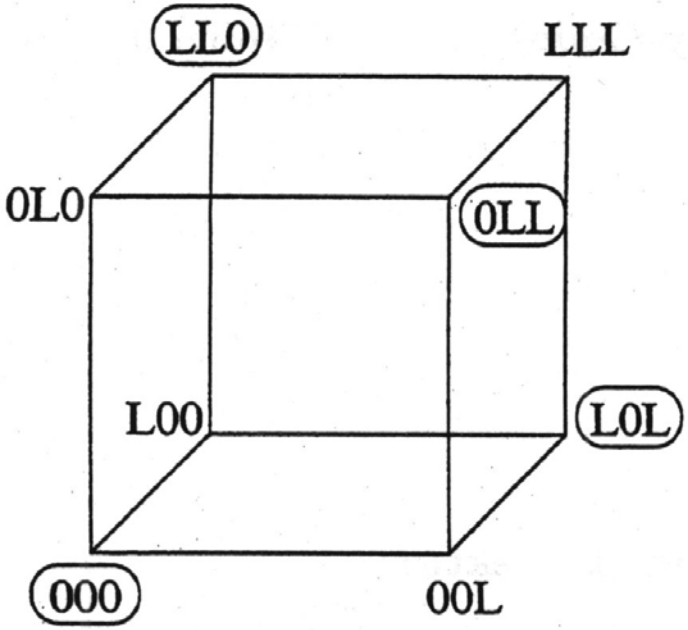
Wiederholung der 10. Vorlesung



17.3 Möglichkeiten der Fehlererkennung

- Man nutzt *nicht* den gesamten zur Verfügung stehenden Wortraum für gültige Codewörter
- Man lässt **Abstand** zwischen gültigen Codewörtern
- Einzelne Bitfehler erzeugen illegalen Codewörter



In diesem Beispiel:

- Einzelbitfehler werden erkannt
- Stelle des Fehlers nicht erkennbar
- daher keine Korrektur möglich

= wird benutzt

17.4 Stellen-Distanz – Hamming-Distanz

- Abstand zwischen zwei bestimmten gültigen Wörtern bezeichnet man als Stellendistanz

Beispiel:

A: 00000
B: 01011
C: 10101
D: 11110

Die Stellendistanz zwischen:

A und B = 3
A und C = 3
A und D = 4
B und C = 4
B und D = 3
C und D = 3

- Hamming-Distanz **h** oder Hamming-Abstand ist der **kleinste Anzahl** der (Bit-)Stellen, in denen sich zwei gültige Wörter voneinander unterscheiden

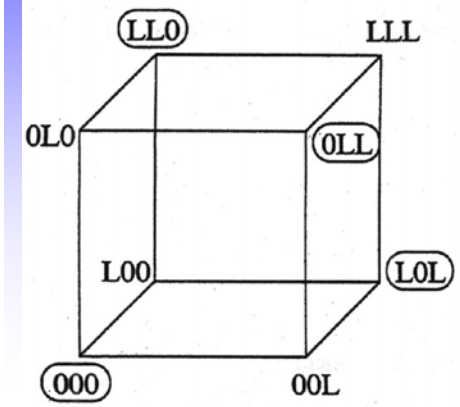
Im Beispiel:

h = 3

17.4 Codierungslänge - Redundanz

- Entropie: H = ld(N), wobei N die Anzahl der Nutzwörter bezeichnet.
- Mittlere Codierungslänge L
- Redundanz: R = L – H, wobei L die mittlere Länge der Codewörter bezeichnet.

Beispiel:



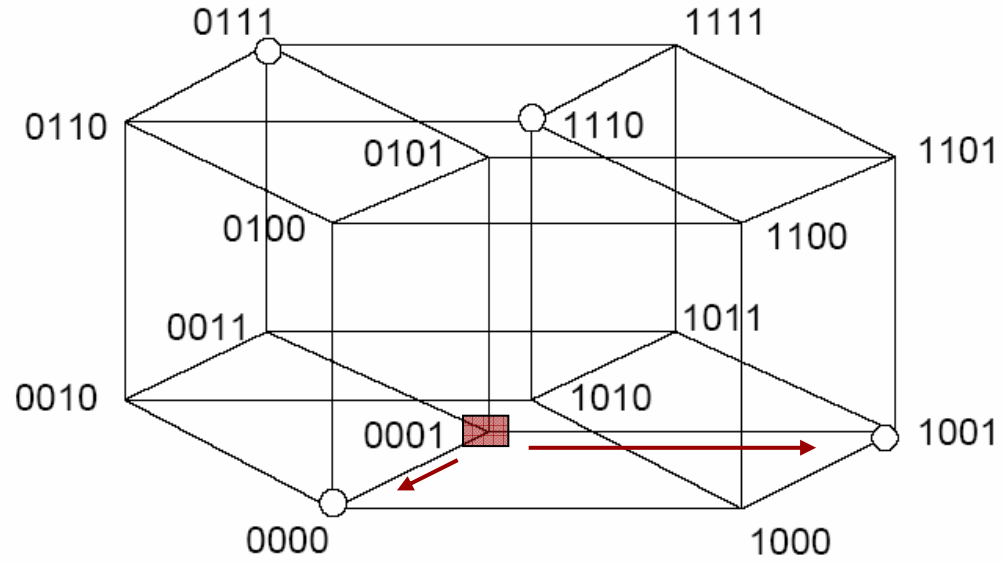
Codierungslänge L = 3

Entropie H = ld(4) = 2

Redundanz R = L – H = 1

Hamming Distanz h = 1

17.4 Hamming-Distanz – Hyperkubus



Kreise sind gültige Worte (4):

Codierungslänge **L** = 4 (bit), Entropie **H** = ld(4) = 2

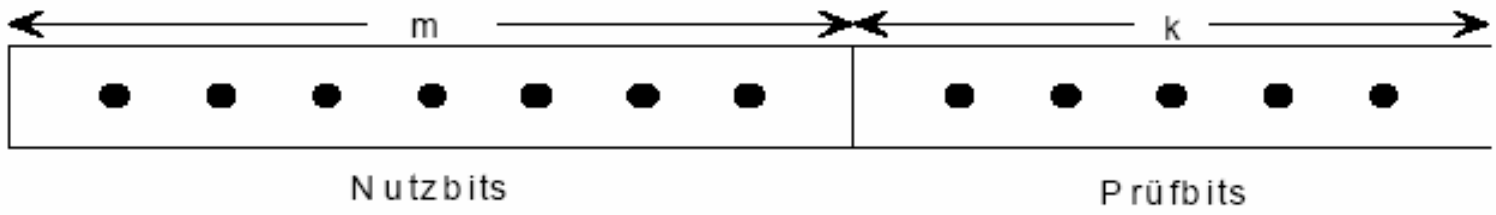
- Redundanz R = L – H = 2
- Eine Hammingdistanz h = 2 kann realisiert werden:
- 1- und 2-Bit-Fehler können erkannt werden.
- Es ist hier nicht ratsam, 1-Bit Fehler zu korrigieren, da dies zu falschen Ergebnissen führen kann.

17.5 Fehlererkennung - Fehlerkorrektur

- Fehlertoleranz basiert auf Redundanz
- Vorrat an möglichen Codewörtern wird nicht vollständig ausgeschöpft:
  - benutzte Codewörter: Nutzwörter
  - nicht benutzte : Pseudowörter
- die redundanten Bits heißen Prüfbits

- Mit s zusätzlichen Bits:
  - Wieviele Fehler können erkannt?
  - Wieviele Fehler können korrigiert werden?

17.5 Fehlererkennung – Fehlerkorrektur (2)



- Code-Redundanz r bezeichnet das Verhältnis **r = k / m**

Es gilt:

- Wenn Codewörter einer Codierung aus *m* Nutzbits und *k* Prüfbits aufgebaut sind, können 1-Bit-Fehler erkannt werden, wenn
  $m + k < 2^k$ gilt.
- Eine Codierung mit einem Hammingabstand *d* erlaubt die Erkennung von *d - 1* Bitfehlern und
  (d-1)/2 Korrekturen, wenn *d* ungerade ist, bzw.
  d/2 – 1 Korrekturen, wenn *d* gerade ist.

17.5 Fehlererkennung – Fehlerkorrektur (3)

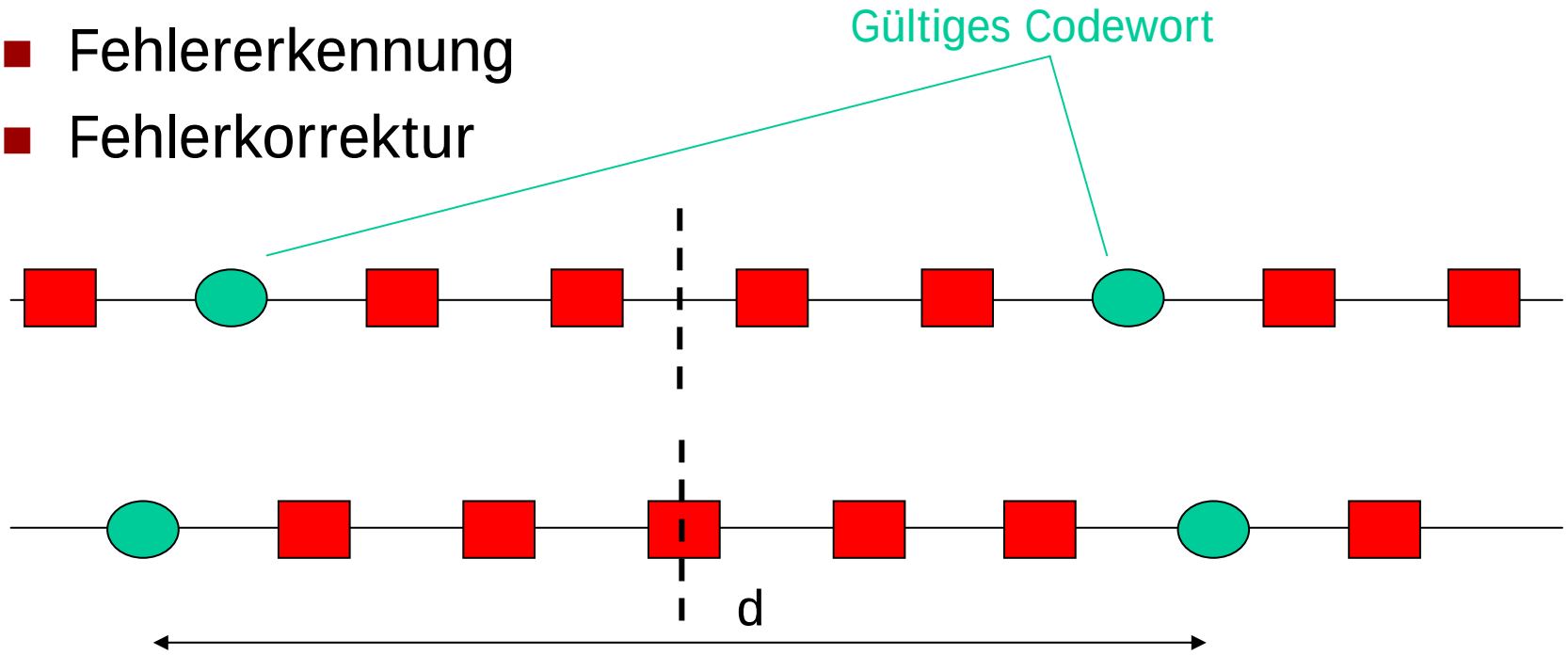
- Bedingungen werden beispielsweise durch das Tripletten-Schema erfüllt

m	4	8	11	16	26
k	3	4	4	5	5
r	0,75	0,50	0,36	0,32	0,19

- SEC (single error correction)
- DED (double error detection)
- Beim SEC/DED Code lassen sich Einzel- und Doppelbitfehler erkennen, aber nur Einzelbitfehler sicher korrigieren

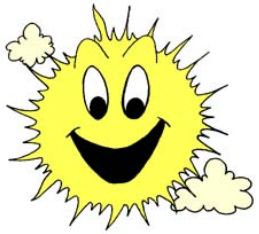
17.5 Fehlererkennung – Fehlerkorrektur (4)

- Fehlererkennung
- Fehlerkorrektur



- Wenn d ungerade:
  - Erkennen Bitfehler d-1,
  - Korrektur (d-1)/2
- Wenn d gerade:
  - Erkennen Bitfehler d-1,
  - Korrektur d/2-1

Ende der Wiederholung



17.6 Paritätsbit

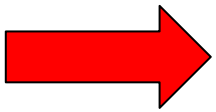
- Die Nutzwörter werden um ein Paritätsbit ergänzt, so dass – in Abhängigkeit vom Verfahren – die Codewörter immer eine gerade oder ungerade Anzahl von 1-Bits enthalten.
  - gerade Parität: Anzahl der 1-bits im erweiterten Codewort ist gerade
  - ungerade Parität: Anzahl der 1-bits im erweiterten Codewort ist ungerade
- Der Einsatz von Paritätsbits wird auch als Zeichen- oder Querparität bezeichnet.
- Eingesetzt wird das Paritätsbit bei der seriellen Übertragung, wobei noch gerade oder ungerade Parität festgesetzt wird.

17.7 Blocksicherung

Daten werden für die Übertragung zu einem Block zusammengefasst.

Beispiel:
ungerade Zahl an 1-Bits für Zeilen und Spalten

Codewort	D3	D2	D1	D0	PB
CW1	0	1	0	1	**1**
CW2	1	1	0	0	**1**
CW3	1	0	0	0	**0**
CW4	1	1	**1**	1	**0**
CW5	0	0	0	0	**1**
Prüfwort	**0**	**0**	**1**	**1**	**0**



Einzelne Bitfehler können detektiert und korrigiert werden.

17.7 Blocksicherung

Codewort	D3	D2	D1	D0	PB
CW1	0	1	0	1	**1**
CW2	1	1	0	0	**1**
CW3	1	0	**1**	0	**0**
CW4	1	1	**1**	1	**0**
CW5	0	0	0	0	**1**
Prüfwort	**0**	**0**	**1**	**1**	**0**

Codewort	D3	D2	D1	D0	PB
CW1	0	1	0	1	**1**
CW2	1	1	0	0	**1**
CW3	1	**1**	0	0	**0**
CW4	1	1	**1**	1	**0**
CW5	0	0	0	0	**1**
Prüfwort	**0**	**0**	**1**	**1**	**0**

Zwei Bitfehler können i.a. nur detektiert und nicht sicher korrigiert wer-den.

17.8 *Zyklischer Redundanzcode (CRC)*

- Zyklischer Redundanzcode, Polynomcode

- Für jede Nachricht N wir eine „**Prüfsumme**“ P berechnet und mit N zu N^* zusammengefasst.
- P wird mit einem Generatorpolynom G der Ordnung g berechnet.
- N wird mit g 0-Bits zu N* ergänzt.
- N* wird durch G geteilt.
- Der Rest der Division wird von N* subtrahiert

17.8 Zyklischer Redundanzcode (CRC) (2)

- N^* ist nun durch G ohne Rest teilbar.
- Auf der Empfängerseite kann dies über-prüft werden, da auch hier G bekannt ist.
- Mit der Entfernung von P erhält der Empfänger die ursprüngliche Nachricht N.
- Mit dem CRC-Verfahren können Fehler detektiert aber nicht korrigiert werden.
- Wird das Verfahren mit Paritätsbits er-gänzt kann auch eine Korrektur durch-geführt werden. der Division wird von N* subtrahiert

17.8 Zyklischer Redundanzcode (CRC) (3)

- Berechnung erfolgt ohne Berücksichtigung von Überträgen (algebraische Feldtheorie Modulo-2).
- Berechnung kann einfach in Hardware realisiert werden.
- Generatorpolynome werden so konstruiert, dass bestimmte Fehlerklassen detektiert werden können.

Bezeichnung	Polynom	Anwendung
CRC-12	$x^{12}+x^{11}+x^3+x^2+x^1+1$	6 Bit Werte
CRC-16	$x^{16}+x^{15}+x^2+1$	8 Bit Werte
CRC-CCITT	$x^{16}+x^{12}+x^5+1$	8 Bit Werte
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$	Internet

17.9 Huffman-Codierung

- Situation: Die Zeichen eines Zeichenvorrats werden mit unterschiedlicher Wahrscheinlichkeit verwendet.
- Gesucht: Codierung des Zeichenvorrats, die eine minimale mittlere Länge der Codewörter liefert.
- Ziel: Minimierung der zu übertragenden bzw. zu speichernden Datenmenge.

17.9 Huffman-Codierung (2)

- Huffman Baum (H-Baum) liefert eine Lösung für die Minimierung der mittleren Länge der Codierung der Codewörter.
- Der H-Baum ist ein Binärbaum, der iterativ erstellt werden kann.
- Die Blätter des Baums stellen die Code-wörter dar.
- Die Codierung ergibt sich aus der Kantenbeschriftung von der Wurzel zu den Blättern

17.9 mittlere Länge für die Codierung

$$L = \sum_{i=1}^{n} p_i l_i$$

L: mittlere Länge für Codierung eines Zeichens

n: Anzahl der Zeichen des Zeichenvorrats

p_i: Wahrscheinlichkeit für das Auftreten des Zeichens i

l_i: Länge der Codierung für Zeichen i

17.9 Konstruktion des Huffman-Baumes

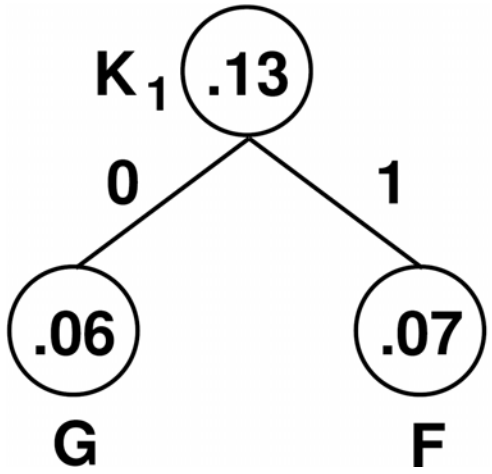
1. Schritt: Suche zwei Zeichen x_i und x_j der Informationsquelle mit kleinsten Wahrscheinlichkeiten
2. Schritt: Bilde einen Knoten K_{ij} des Codebaums; ordne ihm die Wahrscheinlichkeit $p(K_{ij}) = p(x_i) + p(x_j)$ zu. Verbinde K_{ij} mit x_i und x_j.
3. Schritt: Entferne x_i und x_j aus der Informationsquelle und füge ihr statt dessen K_{ij} hinzu.
4. Schritt: Gehe zu Schritt 1, falls die Informationsquelle noch mehr als ein Zeichen enthält.
5. Schritt: Füge (letztes) Zeichen als Wurzel zum Codebaum. Beschrifte die Kanten wie oben erläutert.

17.9 Beispiel H-Baumes

x_i	A	B	C	D	E	F	G
p_i	.25	.21	.18	.14	.09	.07	.06

Auswahl: F (.07), G (.06)

$K_1 = .13$

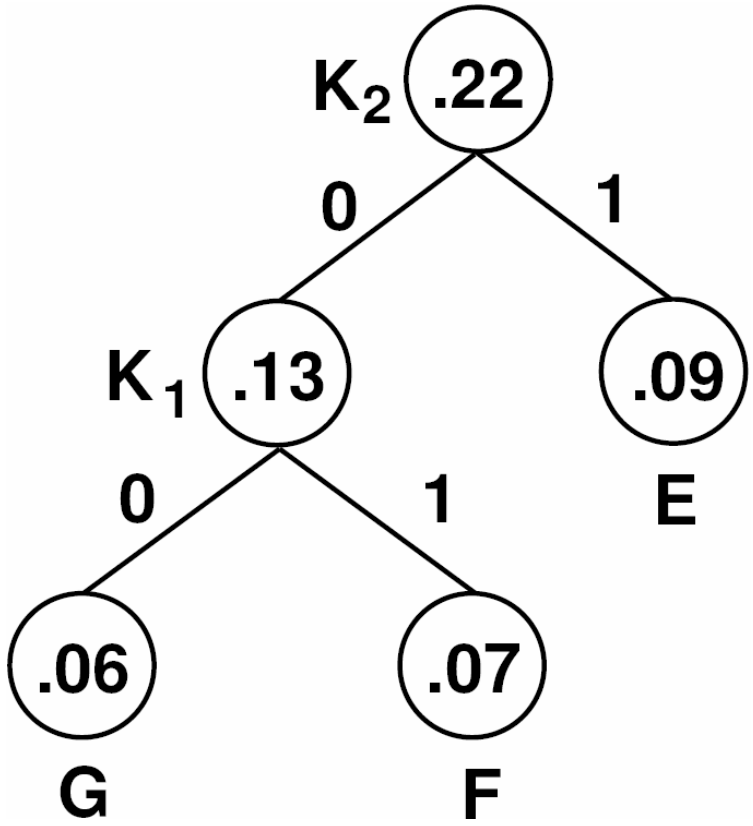


17.9 Beispiel H-Baumes (2)

x_i	A	B	C	D	K_1	E
p_i	.25	.21	.18	.14	.13	.09

Auswahl: E, K_1

K_2 = .22

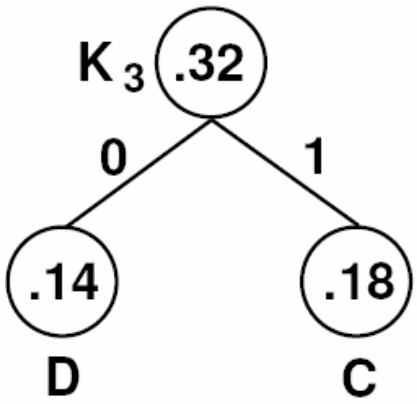


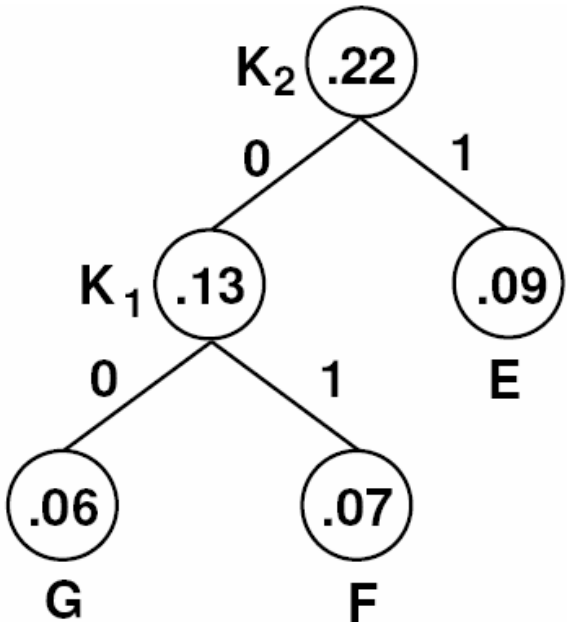
17.9 Beispiel H-Baumes (3)

x_i	A	K_2	B	C	D
p_i	.25	.22	.21	.18	.14

Auswahl: C, D

K_3 = .32



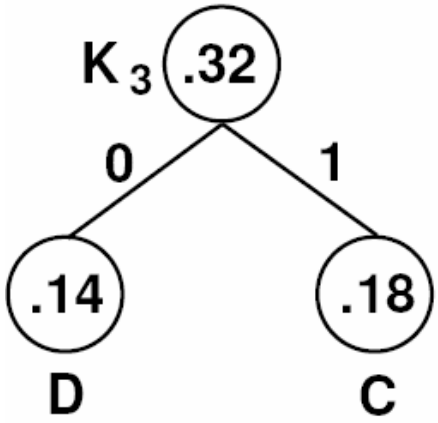


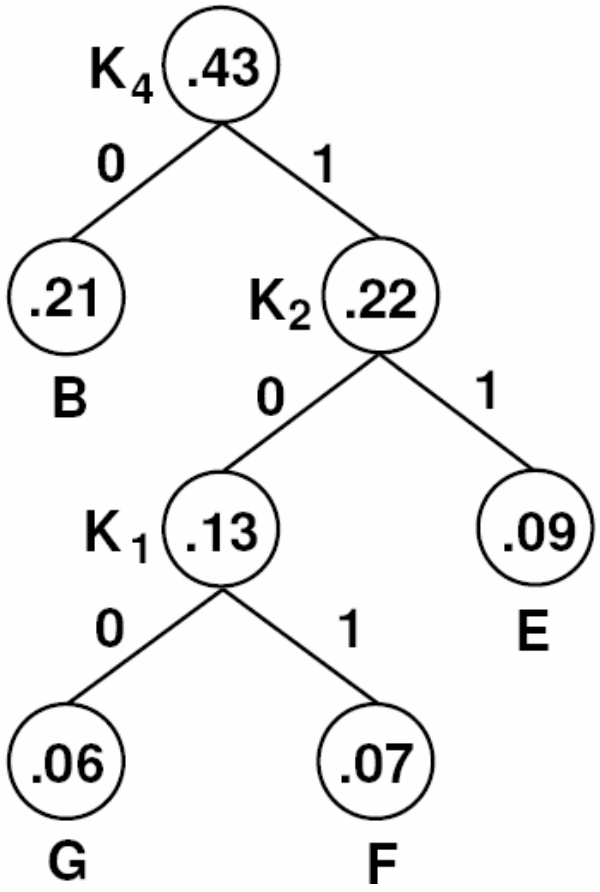
17.9 Beispiel H-Baumes (4)

x_i	K_3	A	K_2	B
p_i	.32	.25	.22	.21

Auswahl: K_2, B

K_4 = .43



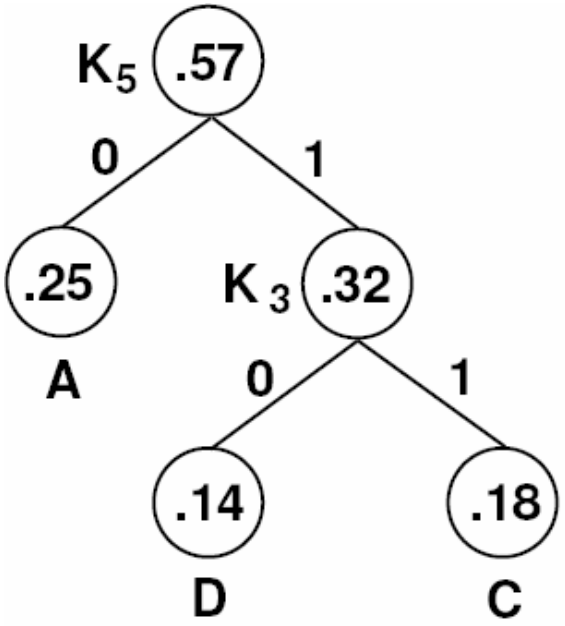


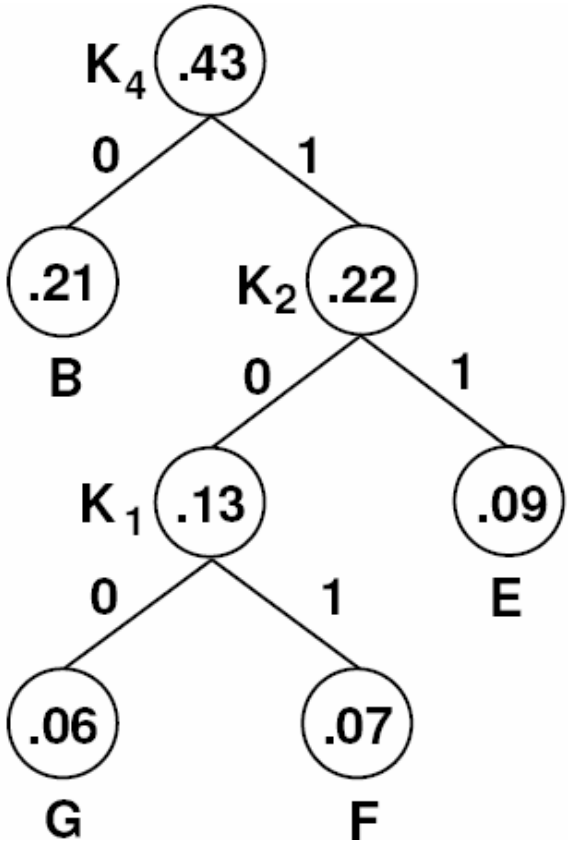
17.9 Beispiel H-Baumes (5)

x_i	K_4	K_3	A
p_i	.43	.32	.25

Auswahl: A, K_3

K_5 = .57



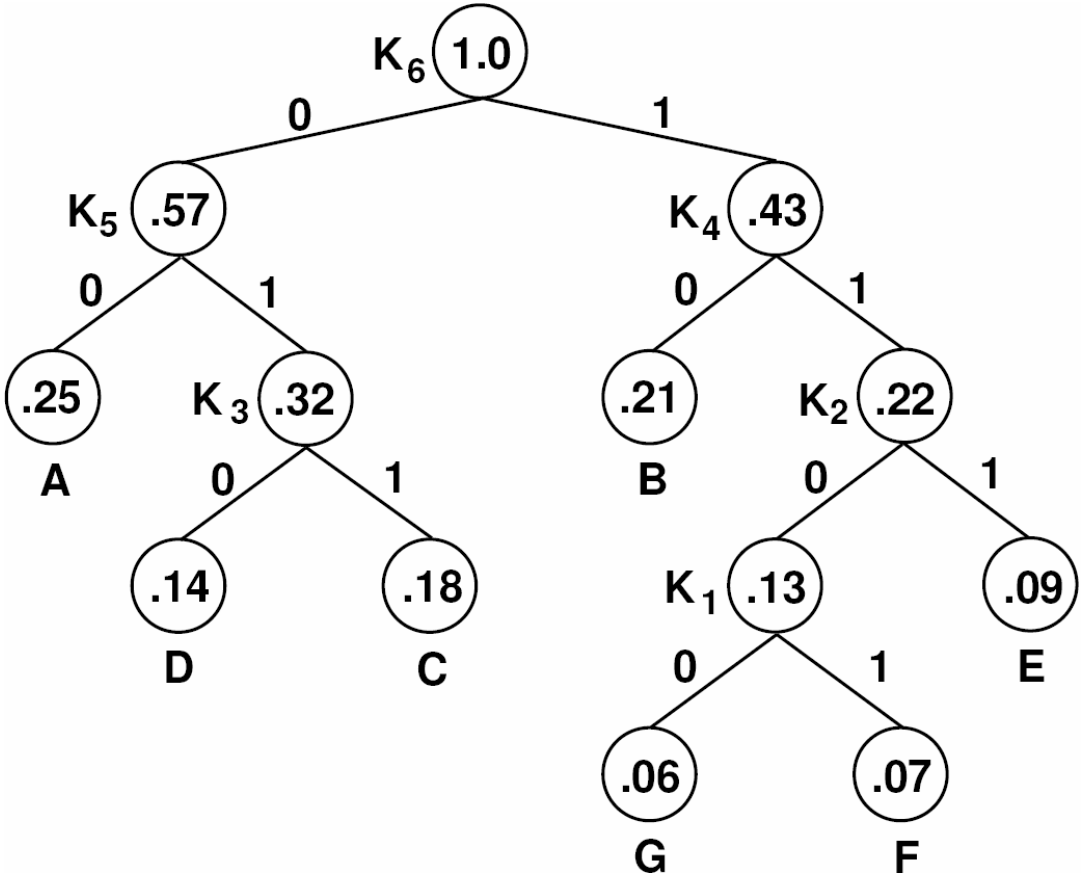


17.9 Beispiel H-Baumes (6)

x_i	K_5	K_4
p_i	.57	.43

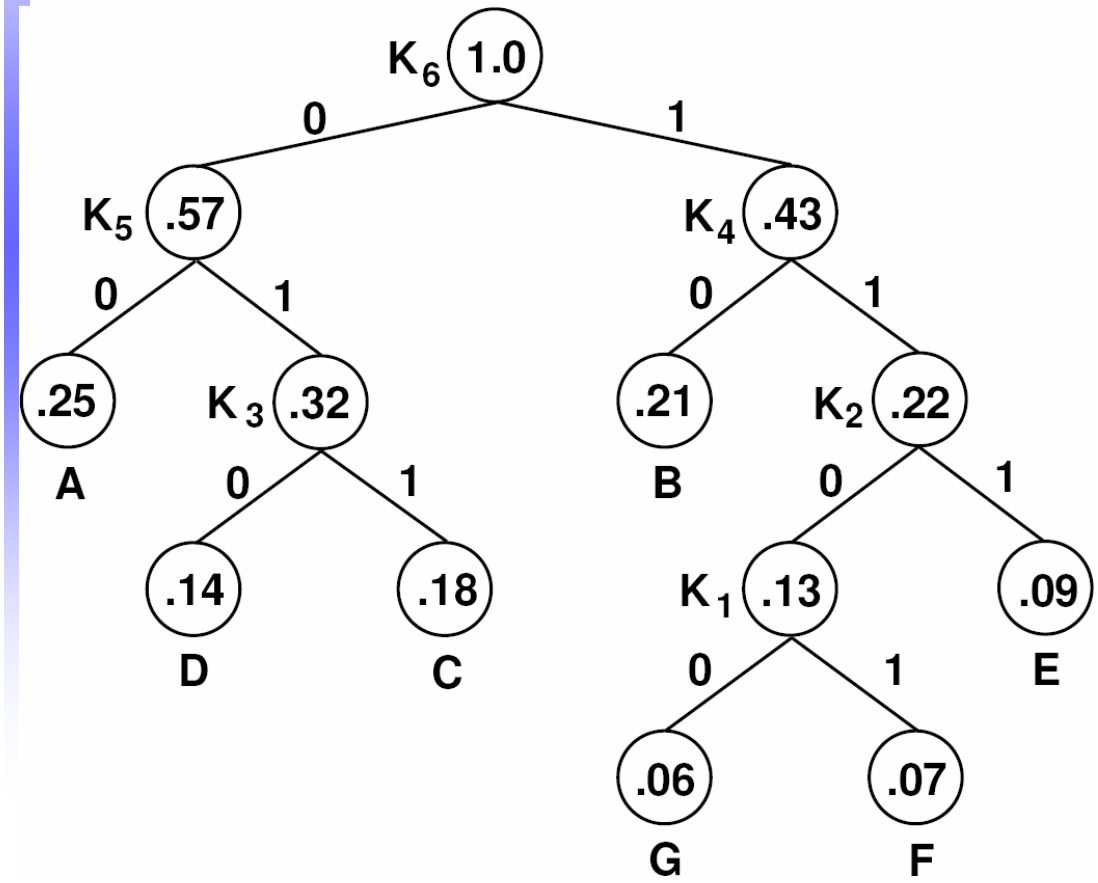
Auswahl: K_4, K_5

$K_6 = 1.0$



17.9 Beispiel H-Baumes (7)

x_i	A	B	C	D	E	F	G
p_i	.25	.21	.18	.14	.09	.07	.06



A: 00

G: 1100

0011011101111 **?**

➡ **A F F E**

Präfixfreier Code

17.9 Fazit

7 Zeichen

➡ L = **3** bit/Zeichen bei Codierung gleicher Länge

Huffman:

x_i	A	B	C	D	E	F	G
p_i	.25	.21	.18	.14	.09	.07	.06
l_i	2	2	3	3	3	4	4

➡ L = **2,67** bit/Zeichen