



3 Kompression

3.1 Einführung

Motivation - Datenvolumen



- Text 1 Seite mit 80 Zeichen/Zeile,
64 Zeilen/Seite, 1 Byte/Zeichen =
 $80 * 64 * 1 * 8 = 40 \text{ kBit/Seite}$
- Standbild 24 Bit/Pixel, $512 * 512$ Pixel/Bild =
 $512 * 512 * 24 = 8 \text{ MBit/Bild}$
- Audio CD-Qualität, Abtastrate 44,1 KHz,
16 Bit pro Abtastwert =
 $44,1 * 16 = 706 \text{ kBit/s}$
Stereo: 1,412 MBit/s
- Video Vollbild mit $1024 * 1024$ Pixel/Bild,
24 Bit/Pixel, 30 Bilder/s =
 $1024 * 1024 * 24 * 30 =$
720 MBit/s



Folie 190

Motivation



- Sowohl zur Speicherung als auch zur Übertragung ist eine Datenkompression erforderlich



Folie 191

Kompression



- Fast alle Daten enthalten Redundanz
 - „eins, zwei, drei“ vs. „1, 2, 3“
- Unterdrückung von Redundanz erhöht die Informationsrate
 - Geringerer Speicherungsbedarf
 - Weniger Übertragungskapazität



Folie 192

Kompression



- Eventuell Einführung von nicht wahrnehmbarem Verlust
- Probleme
 - Rechenaufwand
 - Fehleranfälligkeit
 - Bitfehler, Paketverlust, etc.
 - Eventuell Informationsverlust



Folie 193

Kompressionsarten I



- Verlustfreie Kompression (lossless compression)
 - Das Original kann vollständig wiedergewonnen werden
 - Kompressionsraten von 2:1 bis ca. 50:1



Folie 194

Kompressionsarten II



- Verlustbehaftete Kompression (lossy compression)
 - Unterschied zwischen Originalobjekt und dekodiertem Objekt
 - Physiologische und wahrnehmungspsychologische Eigenschaften des Auges und des Ohres werden ausgenutzt
 - Höhere Kompressionsraten als bei verlustfreier Kompression möglich (50:1 bis über 100:1)



Folie 195

Kompressionsarten III



- Entropie-Kodierung
 - Korrelationen finden und ausnutzen
 - Nullunterdrückung, Lauflängen-Kodierung
 - Beispiele
 - Huffman Coding
 - Lempel-Ziv (LZW, etc.)
 - Verlustlos
 - Weiterverarbeitung möglich



Folie 196

Kompressionsarten IV



- Differenzen-Kodierung
 - in Raum und Zeit
 - $d_i = |\text{val}_i - \text{val}_{i+1}| < e$
 - Delta-Modulation
 - Vorhersage



Folie 197

Kompressionsarten V



- Truncation Coding
 - Grenzen der menschlichen Wahrnehmung
 - z.B. Runden: Gleitpunkt-Zahlen
 - Audio, Video
 - Verlustbehaftet
- Transformationskodierung
 - Daten aus einem anderen Blickwinkel betrachten
 - Koordinatentransformation
 - für Bilder



Folie 198

Kompressionsarten VI



- Modellbasierte Kodierung
 - Decoder = eine Maschine die einen Medienstrom erzeugt
 - Coder findet die richtigen Eingaben für diese Maschine
 - Audio, Video mit sehr niedriger Bitrate



Folie 199



3 Kompression

3.2 Entropie-Kodierung

Lauf­längen­kodierung



- run-length encoding
- Prinzip
 - Ersetze Wiederholungen desselben Zeichens im Text („run“) jeweils durch einen Zähler und das Zeichen



Folie 201

Lauf­längen­kodierung



- Beispiel
 - Text:
AAAABBBBAABBBBBCCCCCCCCDABCBAABBBB
 - Kodierung:
4A3B2A5B8C1D1A1B1C1B2A4B
- Es ist nur dann eine gute Kompressionsrate zu erwarten, wenn es häufig lange „runs“ gibt:
 - Folgen von Leerzeichen
 - Führende Nullen
 - identische Graustufen in Bildern



Folie 202

Lauf­längen­kodierung für Binärdateien



- Bei Binärdateien folgt jedem Run von Einsen immer ein Run von Nullen und umgekehrt
- Es genügt deshalb, nur die Länge der Runs zu speichern



Folie 203

Lauf­längen­kodierung für Binärdateien



- Beispiel

000000000001111111111111111100000000	12	14	9	
000000000011111111111111111110000000	10	18	7	
0000000111111111111111111111110000	7	24	4	
0000001111111111111111111111111000	6	26	3	
000011111111111111111111111111110	4	30	1	
00011111110000000000000000000111111	3	7	18	7
00011111000000000000000000000011111	3	5	22	5
0001110000000000000000000000000111	3	3	26	3
0001110000000000000000000000000111	3	3	26	3
0001110000000000000000000000000111	3	3	26	3
0001110000000000000000000000000111	3	3	26	3
00001111000000000000000000000001110	4	4	23	3 1
00000011100000000000000000000111000	4	3	20	3 3
00000001111111111111111111111110000	7	24	4	



Folie 204

Kodierung mit variabler Länge



- klassische Zeichencodes verwenden gleich viele Bits für jedes Zeichen
- wenn verschiedene Zeichen mit unterschiedlichen Häufigkeiten vorkommen
 - Verwende für häufige Zeichen wenige Bits
 - und für seltene Zeichen mehr Bits



Folie 205

Kodierung mit variabler Länge



- Code 1: A B C D E ...
 1 2 3 4 5 ... (binär)
 - Kodierung von ABRACADABRA mit konstanter Zeichengröße (5 Bit):
0000100010100100000100011000010010000
001000101001000001
- Code 2: A B R C D
 0 1 01 10 11
 - kodierter Text:
0 1 01 0 10 0 11 0 1 01 0



Folie 206

Kodierung ohne explizite Begrenzer



- Code 2 kann nur eindeutig dekodiert werden, wenn Zeichenbegrenzer mitgespeichert werden
 - vergrößert die Datenmenge erheblich
- Idee
 - Keine Kodierung eines Zeichens darf mit dem Anfang der Kodierung eines anderen Zeichens übereinstimmen
 - D.h. kein Codewort ist Präfix eines anderen Codeworts
 - Dann kann auf Begrenzer verzichtet werden



Folie 207

Kodierung ohne explizite Begrenzer



- Code 3:
 - A 11
 - B 00
 - R 011
 - C 010
 - D 10
- kodierter Text:
1100011110101110110001111



Folie 208

Darstellung als Trie



- jeder beliebige Trie mit M äußeren Knoten kann jede beliebige Zeichenfolge mit M verschiedenen Zeichen kodieren
- Der Code für jedes Zeichen wird durch den Pfad von der Wurzel zum Zeichen bestimmt
 - mit 0 für „nach links gehen“ und 1 für „nach rechts gehen“

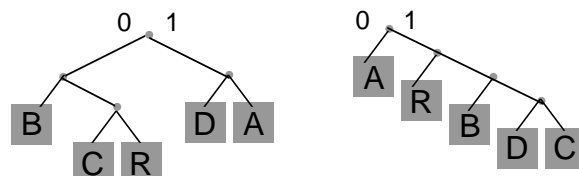


Folie 209

Darstellung als Trie



- Die Trierdarstellung garantiert, dass kein Code für ein Zeichen mit dem Anfang eines anderen übereinstimmt
- die Zeichenfolge lässt sich auf eindeutige Weise dekodieren



Folie 210

Huffman Code



- Frage
 - wie findet man für eine Kodierung mit variabler Länge für gegebene Zeichenwahrscheinlichkeiten die optimale Bitkodierung ?



Folie 211

Algorithmus von D.Huffman (1952)



1. Bestimme die Auftrittshäufigkeiten der Zeichen und schreibe sie an die Blattknoten eines aufzubauenden Binärbaums
2. Nimm die bisher unerledigten zwei Knoten mit den geringsten Häufigkeiten und berechne deren Summe
3. Erzeuge einen Elternknoten für diese beiden und beschrifte ihn mit der Summe. Markiere die Verzweigung zum linken Sohn mit 0, die zum rechten Sohn mit 1
4. Markiere die beiden bearbeiteten Knoten als erledigt. Wenn es nur noch einen nicht erledigten Knoten gibt, terminiere. Sonst weiter mit 2.



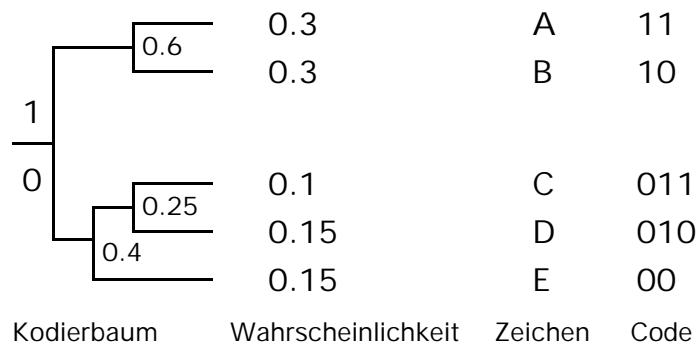
Folie 212

Beispiel Huffman Code



- Wahrscheinlichkeiten der Zeichen:

- $p(A) = 0.3$; $p(B) = 0.3$; $p(C) = 0.1$;
 $p(D) = 0.15$; $p(E) = 0.15$



Folie 213

Optimalität des Huffman - Code



- Zeichen mit großen Häufigkeiten sind näher an der Wurzel des Baumes
 - Sie haben kürzere Codewortlänge
- Ergibt dies einen optimalen Code ?



Folie 214

Optimalität des Huffman - Code



- Die Länge einer kodierten Zeichenfolge ist gleich der gewichteten äußeren Pfadlänge des Huffman-Baumes
- Die gewichtete äußere Pfadlänge eines Baumes ist gleich der über alle äußeren Knoten gebildeten Summe der Produkte des Gewichts mit der Entfernung von der Wurzel
- Kein Baum mit den gleichen Häufigkeiten bei den äußeren Knoten hat eine kleinere gewichtete äußere Pfadlänge als der Huffman-Baum



Folie 215

Beweisidee



- Mit Hilfe des gleichen Prozesses kann ein beliebiger anderer Binärbaum konstruiert werden, doch ohne bei jedem Schritt unbedingt die zwei Knoten mit dem kleinsten Gewicht auszuwählen
- Mittels Induktion lässt sich beweisen, dass keine Strategie zu einem besseren Ergebnis führen kann als die, bei der jeweils zuerst die beiden kleinsten Gewichte ausgewählt werden



Folie 216

Dekodierung von Huffman Codes I



- Dekodierung unter Verwendung des Tries:
 1. Lies den Eingabestrom sequenziell und traversiere den Trie, bis ein Blattknoten erreicht ist
 2. Gib bei Erreichen des Blattknotens das erkannte Zeichen aus
- Beobachtung
 - die Eingabe-Bitrate ist konstant
 - die Ausgabe-Zeichenrate ist variabel



Folie 217

Dekodierung von Huffman Codes II



- Verwendung einer Dekodiertabelle
- Erzeugung der Tabelle
 - Hat das längste Codewort L Bits, dann hat die Tabelle 2^L Einträge
 - Sei c_i das Codewort für Zeichen s_i . c_i habe l_i Bits. Wir erzeugen 2^{L-l_i} Einträge in der Tabelle, bei denen jeweils die ersten l_i Bits = c_i sind und die verbleibenden $L-l_i$ Bits alle möglichen Kombinationen annehmen
 - An all diesen Adressen wird s_i als erkanntes Zeichen eingetragen und l_i als Codewortlänge gemerkt



Folie 218

Dekodierung von Huffman Codes III



- Einsatz der Tabelle zur Dekodierung
 1. Lies L Bits aus dem Eingabestrom in einen Puffer
 2. Benutze den Puffer als Adresse in der Tabelle und gib das erkannte Zeichen s_i aus
 3. Entferne die ersten l_i Bits aus dem Puffer und ziehe weitere l_i Bits aus dem Eingabestrom nach
 4. Weiter mit Schritt 2
- Beobachtung
 - das Tabellenverfahren ist schnell
 - die Ausgabe-Zeichenrate ist konstant
 - die Eingabe-Bitrate ist variabel



Folie 219

Bewertung des Huffman Code



- sehr guter Code für viele praktische Zwecke
- nur geeignet, wenn die Häufigkeiten der Zeichen a priori bekannt und immer gleich oder ähnlich sind
- Variante
 - Ermittle die Häufigkeiten für jeden gegebenen Text neu und speichere/übertrage den Code mit den Daten



Folie 220

Bewertung des Huffman Code



- Ein Verlust entsteht dadurch, dass jedes Zeichen mit einer ganzzahligen Anzahl Bits kodiert werden muss
 - die Codelänge kann den Häufigkeiten nicht theoretisch optimal angepasst werden
- Verbesserung
 - arithmetische Kodierung (kommt später)



Folie 221

Wörterbuch-basierte Kodierung



- Wörterbuch
 - Tabelle von Zeichenketten, auf die bei der Kodierung Bezug genommen wird
- Beispiel
 - „Vorlesung“ stehe im Wörterbuch auf Seite x_3 , Zeile y_3
 - Referenzierung durch (x_3, y_3)
 - „Heute ist Vorlesung“ wird dann z.B. kodiert als Sequenz von Referenzen

$(x_1, y_1) (x_2, y_2) (x_3, y_3)$



Folie 222

Wörterbuch-basierte Kodierung



- Statische Verfahren
 - Das Wörterbuch wird vor der Kodierung festgelegt und nicht mehr geändert



Folie 223

Wörterbuch-basierte Kodierung



- Dynamische Verfahren
 - Das Wörterbuch wird basierend auf der zu übertragenden Nachricht beim Sender bzw. Empfänger aufgebaut
 - Wichtige Verfahren beruhen auf zwei Algorithmen von Abraham Lempel und Jacob Ziv (1977/78)
 - LZSS Lempel-Ziv-Storer-Szymanski (z.B. gzip, ZIP)
 - LZW Lempel-Ziv-Welch (LZW, 1984)
 - LZC Lempel-Ziv Compress
 - LZMW Unix Compress,
GIF (Graphics Interchange Format)



Folie 224

LZW Kodierung



- Startet mit einem Wörterbuch, das die Einzelzeichen enthält
- Kodierungs-Algorithmus

```
w = NIL
while (Z = readchar()) {
  if wZ in Wörterbuch
    w = wZ
  else {
    gib Code für w aus
    füge wZ ins Wörterbuch ein
    w = Z }
}
gib Code für w aus
```



Folie 225

LZW Beispiel



- Eingabe: abababcabac

	w	Z	Neuer Eintrag in Wörterbuch	Ausgang
			Eintrag	Index
a		a		
b	a	b	ab	256
a	b	a	ba	257
b	a	b		
a	ab	a	aba	258
b	a	b		
a	ab	c	abc	259
a	c	a	ca	260
b	a	b		
a	ab	a		
b	aba	c	abac	261
c				

LZW Dekodierung



- Startet mit dem Wörterbuch der Einzelzeichen
- Algorithmus:

```
v = Wörterbuch(readcode())
output v
while (C = readcode()) {
  if C in Wörterbuch
    w = Wörterbuch(C)
  else {
    K = v[0]    -- v[0] = erstes Zeichen in v
    w = vK
  }
  output w
  Z = w[0]
  füge vK ins Wörterbuch ein
  v = w
}
```



Folie 227

LZW Eigenschaften



- die Code-Tabelle (Wörterbuch) muss nicht übertragen werden
- In der Praxis wird die Länge der Tabelle begrenzt
 - Trade-Off zwischen Geschwindigkeit und Kompressionsrate
- LZW passt sich dynamisch an die Eigenschaften der Zeichenkette an



Folie 228

LZW Eigenschaften



- Komplexität
 - Kodierung: $O(N)$,
N = Länge der nichtkodierten Nachricht
 - Dekodierung: $O(M)$,
M = Länge der kodierten Nachricht
 - Da $M \leq N$ ist die Dekodierung sehr effizient
- Typische Beispiele für Dateigrößen im Vergleich zum Original

Art	Huffman	Lempel-Ziv
C-Dateien	65 %	45 %
Maschinenkode	80 %	55 %
Text	50 %	30 %



Folie 229

Arithmetische Kodierung I



- Informationstheoretisch gesehen ist die Huffman-Kodierung nicht ganz optimal
 - einem Datenwort wird immer eine ganzzahlige Anzahl Bits als Codewort zugewiesen
- Idee der arithmetischen Kodierung
 - Eine Nachricht wird als Gleitkommazahl aus dem Intervall $[0, 1)$ kodiert
 - das Intervall wird nach der Wahrscheinlichkeit der einzelnen Symbole aufgeteilt
 - Jedes Intervall repräsentiert ein Zeichen



Folie 230

Kodierungs - Algorithmus



1. Zerlege das Intervall $[0,1)$ in mehrere Teilintervalle. Für jedes Zeichen des Alphabets wird ein Teilintervall verwendet. Die Teilung erfolgt nach den Zeichenwahrscheinlichkeiten.
2. Das nächste zu kodierende Zeichen bestimmt das nächste Basisintervall. Teile dieses Intervall erneut nach den Zeichenwahrscheinlichkeiten. Führe dies so lange fort, bis ein Endesymbol erreicht wird oder eine bestimmte Anzahl Zeichen kodiert ist.
3. Wähle eine Zahl aus dem zuletzt entstandenen Intervall wird als Repräsentant für das Intervall. Wähle eine Zahl, die sich mit möglichst wenigen Bits darstellen lässt.



Folie 231

Dekodierungs - Algorithmus



1. Das Intervall $[0,1)$ wird wie beim Kodierungsprozess unterteilt.
2. Der Code bestimmt das nächste zu unterteilende Basisintervall nach seinem Wert. Dieses Intervall steht für ein bestimmtes Zeichen, das damit dekodiert ist. Dies wird so lange fortgeführt, bis das Endesymbol dekodiert worden ist oder eine bestimmte vorab festgelegte Anzahl von Zeichen.

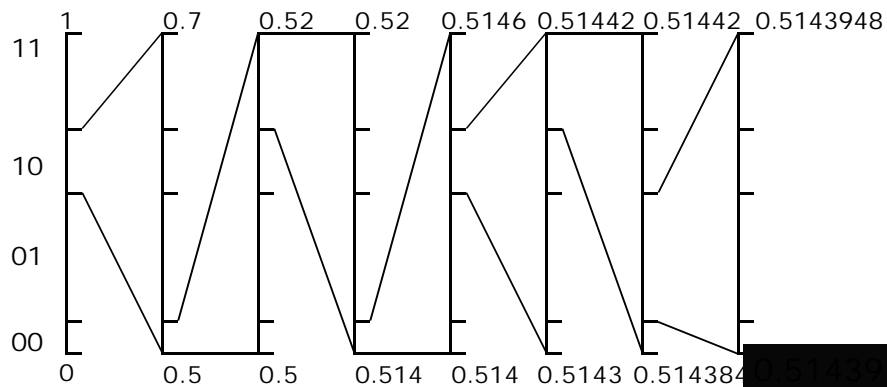


Folie 232

Statische arithmetische Kodierung - Beispiel

Zeichen, Alphabet	00	01	10	11
Wahrscheinlichkeiten	0.1	0.4	0.2	0.3
Erste Kodierungsintervalle	[0,0.1)	[0.1,0.5)	[0.5,0.7)	[0.7,1)

Nachricht: 10 00 11 00 10 11 01



Dekodierung des Beispiels

Schritt	Intervall	Zeichen	Dekodierungsentscheidung
1	[0.5, 0.7)	10	0.51439 befindet sich in [0.5, 0.7)
2	[0.5, 0.52)	00	0.51439 befindet sich im 1. Zehntel des Intervalls [0.5, 0.7)
3	[0.514, 0.52)	11	0.51439 befindet sich im 7. Zehntel des Intervalls [0.5, 0.52)
4	[0.514, 0.5146)	00	0.51439 befindet sich im 1. Zehntel des Intervalls [0.514, 0.52)
5	[0.5143, 0.51442)	10	0.51439 befindet sich im 5. Zehntel des Intervalls [0.514, 0.5146)
6	[0.514384, 0.51442)	11	0.51439 befindet sich im 7. Zehntel des Intervalls [0.5143, 0.51442)
7	[0.51439, 0.5143948)	01	0.51439 befindet sich im 4. Zehntel des Intervalls [0.514384, 0.51442)

Dynamische arithmetische Kodierung



- Idee
 - Übertragung in Blöcken
 - Neuberechnung der Wahrscheinlichkeiten nach jedem Block
- Beispiel
 - Alphabet = {A,B,C}
 - Anfangswahrscheinlichkeiten = {0,2; 0,3; 0,5}
 - Nachricht: ACB AAB (in 3er-Blöcken)

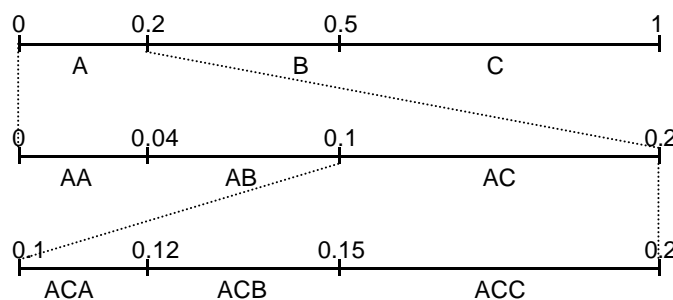


Folie 235

Beispiel dynamische arithm. Kodierung



- 1. Block



- Übertragung [0.12, 0.15), z.B. 0.12

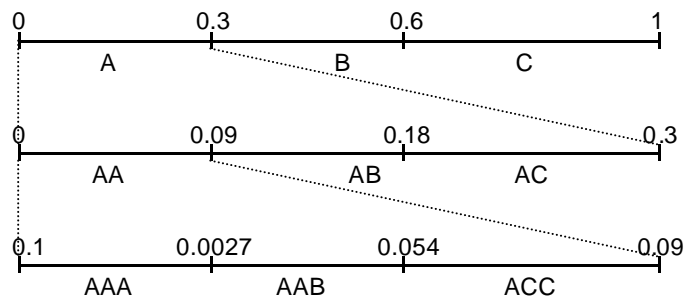


Folie 236

Beispiel Fortsetzung



- Aktualisierung der Wahrscheinlichkeiten
 - $P_{\text{neu}} = (P_{\text{alt}} + P_{\text{aktuell}}) / 2$
 - Neue Wahrscheinlichkeiten = $\{0,3; 0,3; 0,4\}$
- Kodierung des zweiten Blocks



- Übertragung $[0.0027, 0.054)$, z.B. 0.05

Folie 237

Eigenschaften der arithm. Kodierung



- Die Kodierung hängt von der Zeichenwahrscheinlichkeit und damit vom Kontext ab
- Die Wahrscheinlichkeit der Symbole bestimmt die Kompressionseffizienz
 - wahrscheinlichere Symbole werden in größeren Intervallen kodiert und brauchen weniger Präzisionsbits

Folie 238

Eigenschaften der arithm. Kodierung



- Die Code-Länge ist theoretisch optimal
 - die Anzahl Bits pro Zeichen kann nicht-ganzzahlig sein und der Zeichenwahrscheinlichkeit genau angepasst werden
- Die Terminierung des Verfahrens beim Dekodieren ist auf mehrere Arten möglich
 - Die Nachrichtenlänge ist Sender und Empfänger bekannt
 - Es gibt nur eine bestimmte Anzahl von Nachkommastellen (auch dies ist Sender und Empfänger bekannt zu geben)



Folie 239

Probleme der arithm. Kodierung



- Präzision von Maschinen ungenau
 - „overflow“ oder „underflow“ kann vorkommen
- Die Dekodierung ist erst nach vollständigem Erhalt der (Teil-)Nachricht möglich
 - Diese kann aus vielen Nachkommastellen bestehen
- Sehr fehleranfällig
 - ein Bitfehler zerstört die ganze (Teil-)Nachricht
- Exakte Wahrscheinlichkeiten sind oft nicht erhältlich
 - die maximale theoretische Code-Effizienz kann praktisch kaum erreicht werden



Folie 240