2 Kompressionsverfahren für Multimedia

2.1 Grundlagen der Kompressionsverfahren

2.2 Kompressionsverfahren für Standbilder

2.3 Kompressionsverfahren für Video

2.4 Audio-Kompression

2.1 Grundlagen der Kompressionsverfahren

Motivation: Datenvolumen

Text

1 Seite mit 80 Zeichen/Zeile und 64 Zeilen/Seite und 1 Byte/Zeichen ergibt 80 * 64 * 1 * 8 = 40 kbit/Seite

Standbild

24 Bit/Pixel, 512 × 512 Pixel/Bild ergibt 512 × 512 × 24 = **8 MBit/Bild**

Audio

CD-Qualität, Abtastrate 44,1 KHz, 16 Bit pro Abtastwert ergibt 44,1 × 16 = 706 kBit/s Stereo: **1,412 MBit/s**

Video

Vollbild mit 1024 × 1024 Pixel/Bild, 24 Bit/Pixel, 30 Bilder/s ergibt 1024 × 1024 × 24 × 30 = **720 MBit/s.**
Realistischer: 360 × 240 Pixel/Bild, 360 × 240 × 24 × 30 = **60 MBit/s**
=>
Sowohl zur Speicherung als auch zur Übertragung ist eine Datenkompression erforderlich.

Prinzip der Datenkompression

1. Verlustfreie Kompression (lossless compression)

- Das Original kann perfekt wiedergewonnen werden
- Kompressionsraten von 2:1 bis ca. 50:1 möglich
- Beispiel: Huffman-Kodierung

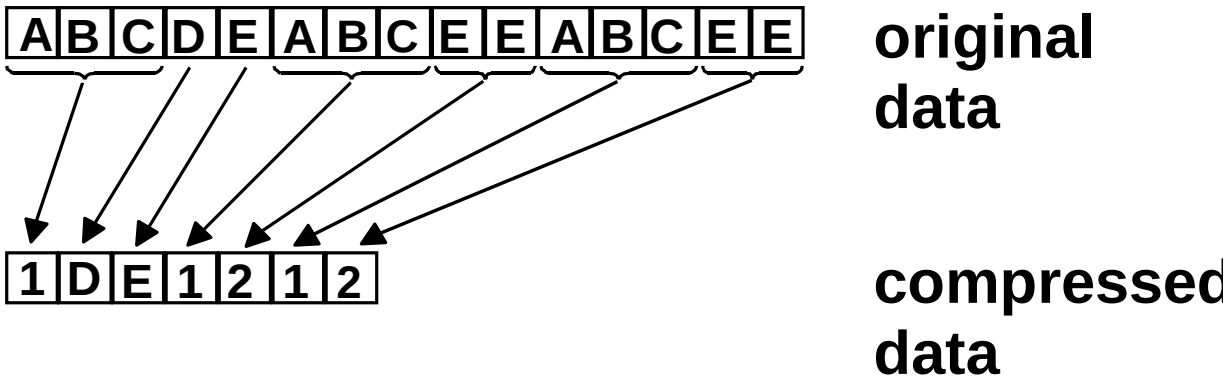
2. Verlustbehaftete Kompression (lossy compression)

- Es gibt einen Unterschied zwischen Originalobjekt und dekodiertem Objekt
- Physiologische und wahrnehmungspsychologische Eigenschaften des Auges und des Ohres werden ausgenutzt
- Höhere Kompressionsraten als bei der verlustfreien Kompression möglich (über 100:1; typisch: 50:1)

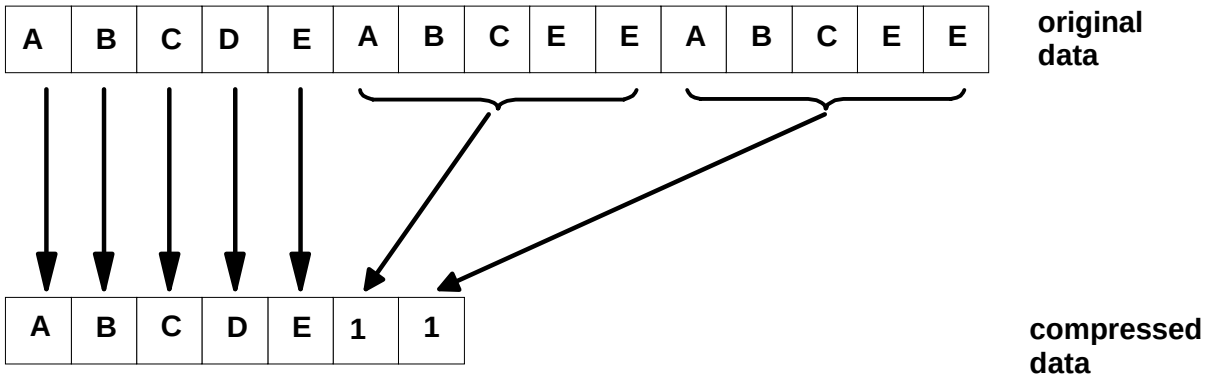
Einfache verlustfreie Kompressionsverfahren

Musterersetzung (pattern substitution)

Beispiel 1



Beispiel 2



Man beachte: Beide Ersetzungen erreichen dieselbe Kompressionsrate.

Lauflängenkodierung

(Englisch: run-length encoding)

Prinzip

Ersetze Wiederholungen desselben Zeichens im Text („runs“) jeweils durch einen Zähler und das Zeichen.

Beispiel

Text:
AAAABBBAABBBBBCCCCCCCCCDABCBAABBBBCCD

Kodierung: 4A3B2A5B8C1D1A1B1C1B2A4B3C1D

Wie wir sehen, ist mit einer guten Kompressionsrate nur zu rechnen, wenn es häufig lange „runs“ gibt, zum Beispiel lange Folgen von Leerzeichen oder führenden Nullen oder auch identischen Graustufen in Bildern.

Lauflängenkodierung für Binärdateien

Bei Binärdateien sind wir sicher, dass ein Run von Einsen stets von einem Run von Nullen gefolgt wird und umgekehrt. Es genügt deshalb, nur die Länge der Runs zu speichern!

Beispiel

```
000000000000000000000000001111111111111000000000  28 14 9
000000000000000000000000011111111111111110000000  26 18 7
000000000000000000000001111111111111111111110000  23 24 4
000000000000000000000011111111111111111111111000  22 26 3
000000000000000000001111111111111111111111111110  20 30 1
000000000000000000011111110000000000000000001111111  19  7 18 7
000000000000000000011111000000000000000000000011111  19  5 22 5
000000000000000000011100000000000000000000000000111  19  3 26 3
000000000000000000011100000000000000000000000000111  19  3 26 3
000000000000000000011100000000000000000000000000111  19  3 26 3
000000000000000000011100000000000000000000000000111  19  3 26 3
000000000000000000011110000000000000000000000001110  20  4 23 3 1
000000000000000000011100000000000000000000000111000  22  3 20 3 3
011111111111111111111111111111111111111111111111111   1 50
011111111111111111111111111111111111111111111111111   1 50
011111111111111111111111111111111111111111111111111   1 50
011111111111111111111111111111111111111111111111111   1 50
011111111111111111111111111111111111111111111111111   1 50
011000000000000000000000000000000000000000000000011   1  2 46 2
```

Kodierung mit variabler Länge

Die klassischen Zeichencodes verwenden gleich viele Bits für jedes Zeichen. Wenn nun die verschiedenen Zeichen mit unterschiedlichen Häufigkeiten vorkommen, so kann man für häufige Zeichen wenige Bits und für seltene Zeichen mehr Bits vorsehen.

Beispiel

Code 1:	A	B	C	D	E	...	
	1	2	3	4	5	...	(binär)

Kodierung von ABRACADABRA mit konstanter Zeichengröße (=5 Bits)

00001000101001000001000110000100100000010001010010000001

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

Kodierung ohne explizite Begrenzer

Code 2 kann nur eindeutig dekodiert werden, wenn die Zeichenbegrenzer mitgespeichert werden. Das vergrößert die Datenmenge erheblich.

Idee

Keine Kodierung eines Zeichens darf mit dem Anfang der Kodierung eines anderen Zeichens übereinstimmen. Kein Codewort ist der Prefix eines anderen Codeworts. Dann können wir auf Begrenzer verzichten.

Code 3:

A	11
B	00
R	011
C	010
D	10

kodierter Text: 11000111101011101100011111

Darstellung als Trie

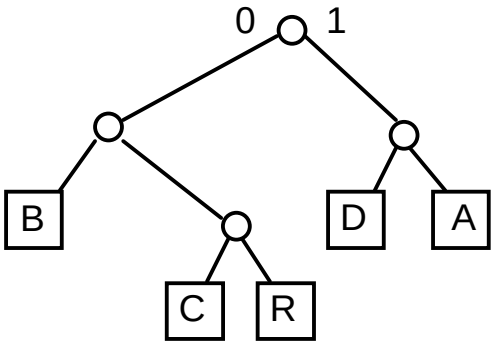
Eine einfache Methode zur Darstellung des Codes ist die Verwendung eines Trie. Tatsächlich kann jeder beliebige Trie mit M äußeren Knoten benutzt werden, um jede beliebige Zeichenfolge mit M verschiedenen Zeichen zu kodieren. Als Beispiel zeigt die folgende Abbildung zwei Codes, die für ABRACADABRA verwendet werden könnten. Der Code für jedes Zeichen wird durch den Pfad von der Wurzel zu diesem Zeichen bestimmt, mit 0 für „nach links gehen“ und 1 für „nach rechts gehen“, wie in einem Trie üblich. Demzufolge entspricht der links dargestellte Trie dem oben angegebenen Code, und der rechts dargestellte Trie entspricht einem Code, der die Zeichenfolge

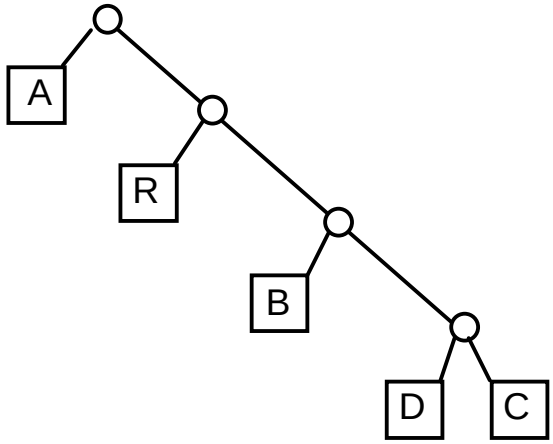
01101001111011100110100

erzeugt, die um zwei Bits kürzer ist.

Trie für unser Beispiel

Die Darstellung als Trie garantiert, dass kein Code für ein Zeichen mit dem Anfang eines anderen übereinstimmt, so dass sich die Zeichenfolge unter Benutzung des Trie auf eindeutige Weise dekodieren lässt.





Huffman - Code

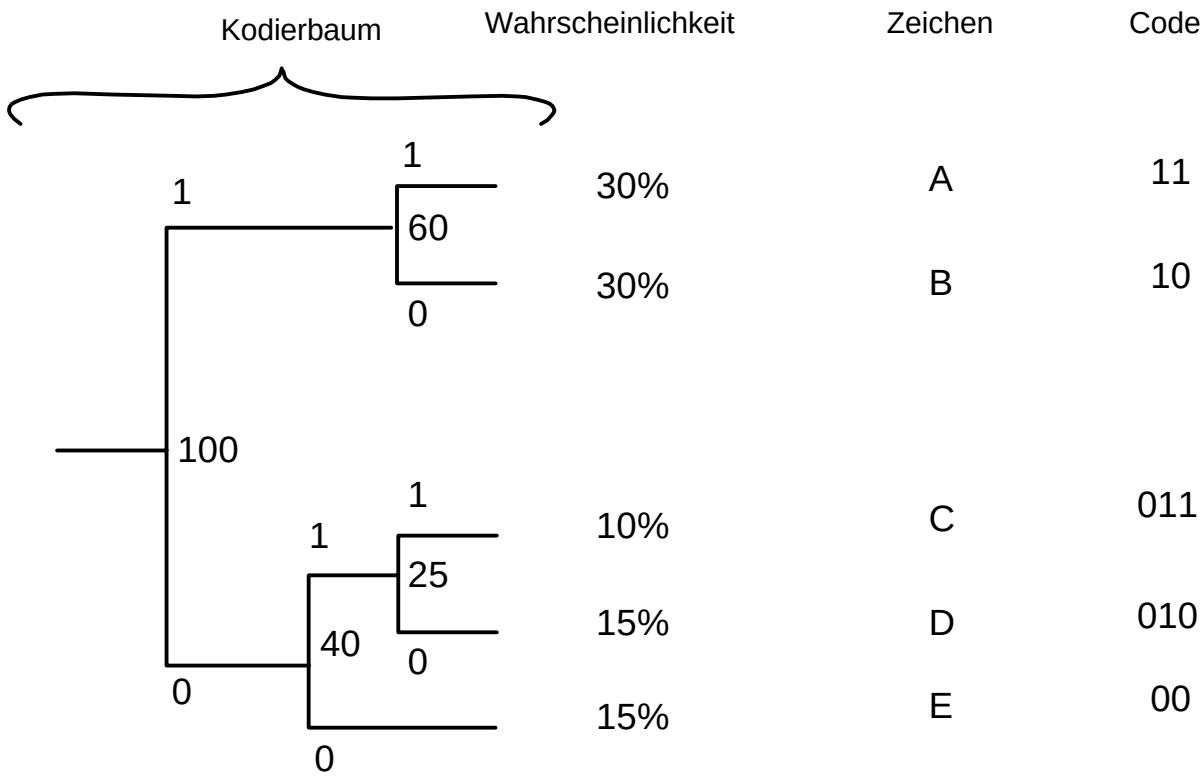
Die Frage ist nun, wie man für eine Kodierung mit variabler Länge für gegebene Zeichenhäufigkeiten (oder Zeichenwahrscheinlichkeiten) im Text die optimale Bitkodierung findet. Der Algorithmus dazu wurde von D. Huffman (1952) angegeben.

Algorithmus Erzeuge-Huffman-Code

1. Bestimme die Auftrittshäufigkeiten der Zeichen und schreibe sie an die Blattknoten eines aufzubauenden Binärbaums an.
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, sind wir fertig. Sonst weiter mit Schritt 2.

Huffman - Code, Beispiel

Wahrscheinlichkeiten der Zeichen:
p(A) = 0.3; p(B) = 0.3; p(C) = 0.1; p(D) = 0.15; p(E) = 0.15



Huffman - Code, Optimalität

Die Zeichen mit großen Häufigkeiten sind näher an der Wurzel des Baumes und haben somit eine kürzere Codewortlänge; deshalb ist es ein guter Code. Es ist sogar der bestmögliche Code!

Denn:

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“ (zugehöriger Häufigkeitszähler) mit der Entfernung von der Wurzel. Dies ist offensichtlich eine Möglichkeit, die Länge der kodierten Zeichenfolge zu berechnen; sie ist äquivalent zu der über alle Buchstaben gebildeten Summe der Produkte der Häufigkeit des Auftretens eines Buchstaben mit der Anzahl der Bits bei jedem Auftreten.

Kein Baum mit den gleichen Häufigkeiten bei den äußeren Knoten hat eine kleinere gewichtete äußere Pfadlänge als der Huffman-Baum.

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.

Dekodierung von Huffman-Codes (1)

Nahe liegend ist eine 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, aber die Ausgabe-Zeichenrate ist variabel!

Dekodierung von Huffman-Codes (2)

Als Alternative bietet sich die Verwendung einer *Dekodiertabelle* an.

Erzeugung der Tabelle

- Hat das längste Codewort L Bits, so 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-li} 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, zugleich wird l_i als Codewortlänge vermerkt.

Dekodierung von Huffman-Codes (3)

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, aber die Eingabe-Bitrate ist variabel!

Huffman - Code, Kommentare

- Ein sehr guter Code für viele praktische Zwecke.
- Allerdings 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.
- Ein (durchaus berechenbarer) Verlust entsteht dadurch, dass jedes Zeichen mit einer ganzen Zahl von Bits kodiert werden muss und somit die Codelänge den Häufigkeiten nicht ganz (theoretisch optimal) angepasst werden kann. Verbesserung: arithmetische Kodierung.

Lempel-Ziv-Kodierung

Gehört zu der großen Gruppe der auf einem Wörterbuch basierenden Kodierungstechniken.

Wörterbuch:
Tabelle von Zeichenketten, auf die bei der Kodierung Bezug genommen wird.

Beispiel
„Vorlesung“ stehe im Wörterbuch auf Seite x_3, Zeile y_3.
Damit kann es durch (x_3,y_3) referenziert werden.
Ein Satz wie z.B. „Heute ist Vorlesung“ wird dann beispielsweise kodiert als Sequenz von Referenzen: (x_1,y_1) (x_2,y_2) (x_3,y_3).

Wörterbuch-basierte Kodierungstechniken

Statische Verfahren:

Das Wörterbuch wird vor der Kodierung festgelegt und nicht mehr geändert, weder beim Kodieren noch beim Dekodieren.

Dynamische Verfahren:

Das Wörterbuch wird basierend auf der zu übertragenden Nachricht beim Sender bzw. Empfänger aufgebaut.

Die Kodierung nach **Lempel und Ziv** (1977) ist eine dynamische wörterbuchbasierte Kodierungstechnik. Eine Weiterentwicklung des Algorithmus wird in vielen heute gebräuchlichen Kompressionsverfahren eingesetzt, z.B. Lempel-Ziv-Welch (LZW) im Compress-Befehl von Unix.

Das bekannte Bildkomprimierungsformat TIFF (Tag Image File Format) basiert auch auf Lempel-Ziv.

Ziv-Lempel-Kodierung, Prinzip

Idee

Der aktuelle Teil einer Nachricht kann durch eine Referenz auf einen bereits vorgekommenen identischen Teil der Nachricht durch dynamisches Anpassen des Wörterbuch ersetzt werden.

LZW-Algorithmus

```
InitializeStringTable();
WriteCode(ClearCode);
 ω= the empty string;
for each character in string {
  K = GetNextCharacter();
  if  ω+ K is in the string table {
    ω=ω+K /* String concatenation*/
  } else {
    WriteCode(CodeFromString(ω));
    AddTableEntry(ω+K );
    ω=K
  }
}
WriteCode(CodeFromString(ω));
```

LZW, Beispiel 1, Kodierung

Alphabet: X = {A, B, C}

Nachricht: ABABCBABAB

Wörterbuch: Initialzustand

Index	Eintrag
0	
1	A
2	B
3	C

Kodierung: 1 2 4 3 5 8

LZW-Algorithmus: Dekodierung (1)

Man beachte, dass auch bei der Dekodierung das Wörterbuch dynamisch aufgebaut werden muss!

```
While((Code=GetNextCode() != EofCode){
  if (Code == ClearCode)
    {
      InitializeTable();
      Code = GetNextCode();
      if (Code==EofCode)
        break;
      WriteString(StringFromCode(Code));
      OldCode = Code;
    } /* end of ClearCode case */
  else
    {
     if (IsInTable(Code))
       {
         WriteString(StringFromCode));
         AddStringToTable(StringFromCode(OldCode)+
           FirstChar(StringFromCode(Code)));
         OldCode = Code;
       }
```

LZW-Algorithmus: Dekodierung (2)

```
    else
      {/* codes in not in table */
        OutString = StringFromCode(OldCode +
          FirstChar(StringFromCode(OldCode)));
        WriteString(OutString);
        AddStringToTable(OutString);
        OldCode = Code;
      }
  }
}
```

LZW, Beispiel 2, Codierung

Unser Alphabet ist {A,B,C,D}. Wir kodieren den String ABACABA. Im ersten Schritt initialisieren wir die Code-Tabelle:

#1 = A
#2 = B
#3 = C
#4 = D

Wir lesen A aus der Eingabe. Wir finden A in der Tabelle und behalten A im Puffer. Wir lesen B aus der Eingabe und betrachten nun AB. AB ist nicht in der Tabelle, wir fügen AB als #5 an die Tabelle an und schreiben #1 für das erste A in die Ausgabe. Der Puffer enthält jetzt nur noch B. Wir lesen als nächstes A, betrachten BA, fügen es als neuen Eintrag #6 an die Tabelle an und schreiben #2 für B in die Ausgabe.

LZW, Beispiel (2)

Am Ende enthält die Code-Tabelle

#1 = A
#2 = B
#3 = C
#4 = D
#5 = AB
#6 = BA
#7 = AC
#8 = CA
#9 = ABA.

Der ausgegebene Datenstrom ist #1 #2 #1 #3 #5 #1.

Man beachte, dass die Code-Tabelle nicht übertragen werden muss! Übertragen wird nur die initiale Tabelle, die volle Dekodier-Tabelle entsteht dynamisch.

Für praktische Anwendungen wir die Länge der Code-Tabelle begrenzt. Die Größe der Tabelle stellt einen Trade-Off zwischen Geschwindigkeit und Kompressionsrate dar.

LZW, Eigenschaften

- Das Wörterbuch enthält die am häufigsten vorkommenden Zeichenketten.
- Es wird automatisch beim Dekodieren aufgebaut und muss nicht zusätzlich übertragen werden.
- Es paßt sich dynamisch an die Eigenschaften der Zeichenkette an.
- Die Kodierung wird in O(N), N = Länge der nichtkodierten Nachricht, Dekodierung in O(M), M = Länge der kodierten Nachricht, durchgeführt. Damit ist die Dekodierung sehr effizient. Da mehrere Symbole in eine Kodierungseinheit kombiniert werden, ist M <= N und Lempel-Ziv in der Regel effizienter als Huffman.

Ergebnisse im Vergleich

Typische Beispiele für Dateigrößen nach der Kompression in Prozent der Ursprungsgröße

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

Arithmetische Kodierung

Informationstheoretisch gesehen ist die Huffman-Kodierung in der Regel nicht ganz optimal, da einem Datenwort stets eine ganzzahlige Anzahl von Bits als Codewort zugewiesen wird. Abhilfe bietet die arithmetische Kodierung.

Idee

Eine Nachricht wird als *Gleitkommazahl* aus dem Intervall [0,1) kodiert. Dazu wird das Intervall aufgeteilt nach der Wahrscheinlichkeit der einzelnen Symbole. Jedes Intervall repräsentiert ein Zeichen.

Arithmetische Kodierung, Algorithmus

Kodierung

1. Das Intervall [0,1) wird in mehrere Teilintervalle zerlegt, wobei jeweils ein Teilintervall für jedes Zeichen des Alphabets verwendet wird. Die Teilung erfolgt nach den Zeichenwahrscheinlichkeiten.
2. Das nächste zu kodierende Zeichen bestimmt das nächste Basisintervall. Dieses Intervall wird wiederum nach den Zeichenwahrscheinlichkeiten unterteilt. Dies wird so lange fortgeführt, bis ein Endesymbol erreicht wird oder eine bestimmte Anzahl Zeichen kodiert ist.
3. Eine Zahl aus dem zuletzt entstandenen Intervall wird als Repräsentant für das Intervall gewählt. Am besten wählt man eine Zahl, die sich mit möglichst wenigen Bits darstellen lässt.

Arithmetische Dekodierung, Algorithmus

Dekodierung

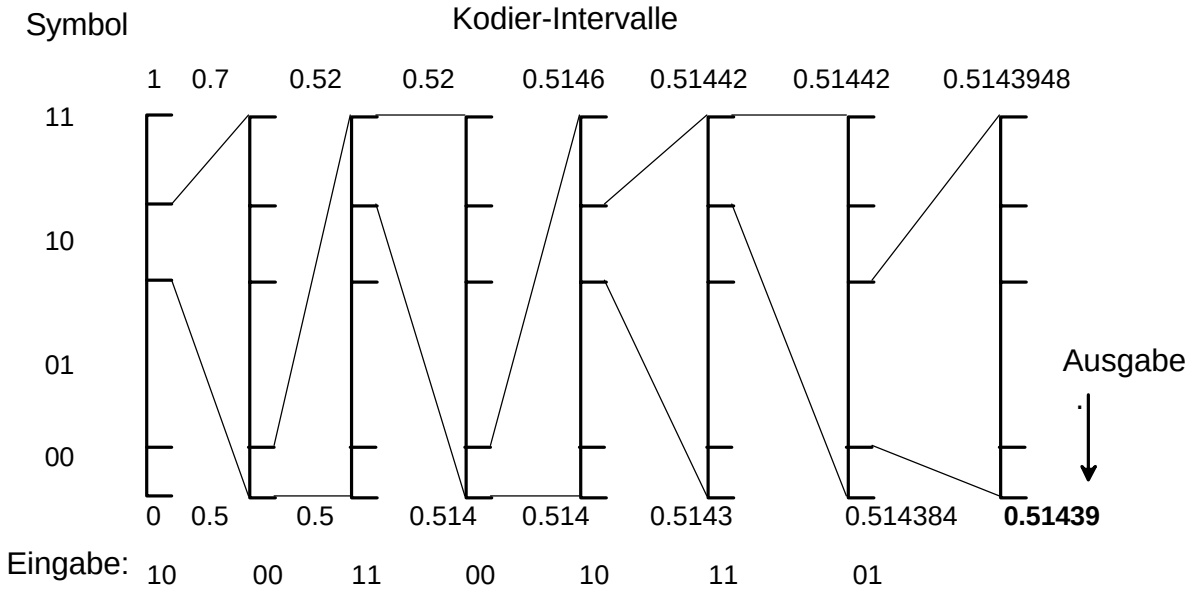
1. Das Intervall [0,1) wird wie beim Kodierungsprozess unterteilt.
2. Der Code bestimmt das nächste zu unterteilende Basisintervall nach seinem Wert. Dieses Intervall steht für ein bestimmtes Zeichen, das damit dekodiert ist. Dies wird so lange fortgeführt, bis das Endesymbol dekodiert worden ist oder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

Statische Arithmetische Kodierung, Beispiel

Alphabet = {00,01,10, 11}
Wahrscheinlichkeiten = {0,1;0,4;0,2;0,3}

Zeichen	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



Kodierung

Schritt	Eingabe Zeichen	Kodierungs-intervall	Kodierungsentscheidung
1	10	[0.5, 0.7)	Das Zeichenintervall ist [0.5, 0.7)
2	00	[0.5, 0.52)	das erste Zehntel von [0.5, 0.7)
3	11	[0.514, 0.52)	die letzten drei Zehntel von [0.5, 0.52)
4	00	[0.514, 0.5146)	das erste Zehntel von [0.514, 0.52)
5	10	[0.5143, 0.51442)	Beginnend mit dem fünften Zehntel, zwei Zehntel von [0.514, 0.5146)
6	11	[0.514384, 0.51442)	die letzten drei Zehntel von [0.5143, 0.51442), beginnend mit dem zweiten Zehntel
7	01	[0.51439, 0.5143948)	Vier Zehntel von [0.514384, 0.51442), beginnend mit dem zweiten Zehntel
8	Wähle eine Zahl aus dem Intervall [0.51439, 0.5143948) als Ausgabe: 0.51439		

Übertragung: 0.51439

Dekodierung

Schritt	Intervall	deko-diertes 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 ersten 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 ersten 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)
8	Die dekodierte Nachricht ist 10 00 11 00 10 11 01		

Dynamische arithmetische Kodierung, Beispiel

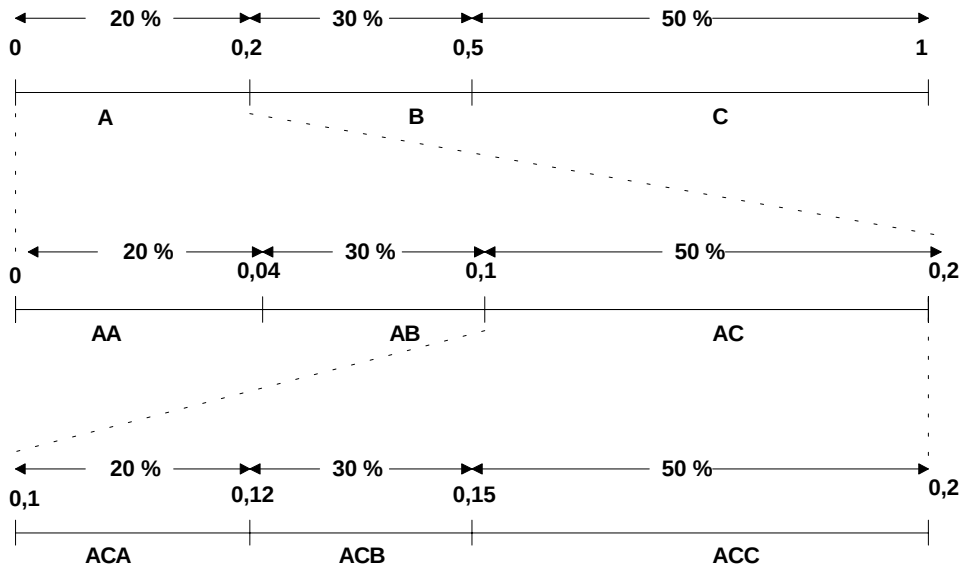
Idee:

Übertragung in Blöcken, Neuberechnung der Wahrscheinlichkeiten nach jedem Block (beim Kodieren und beim Dekodieren).

Alphabet = {A,B,C}
Anfangswahrscheinlichkeiten = {0,2; 0,3; 0,5}

Nachricht: ACB AAB (in 3er-Blöcken)

Kodierung des ersten Blocks:



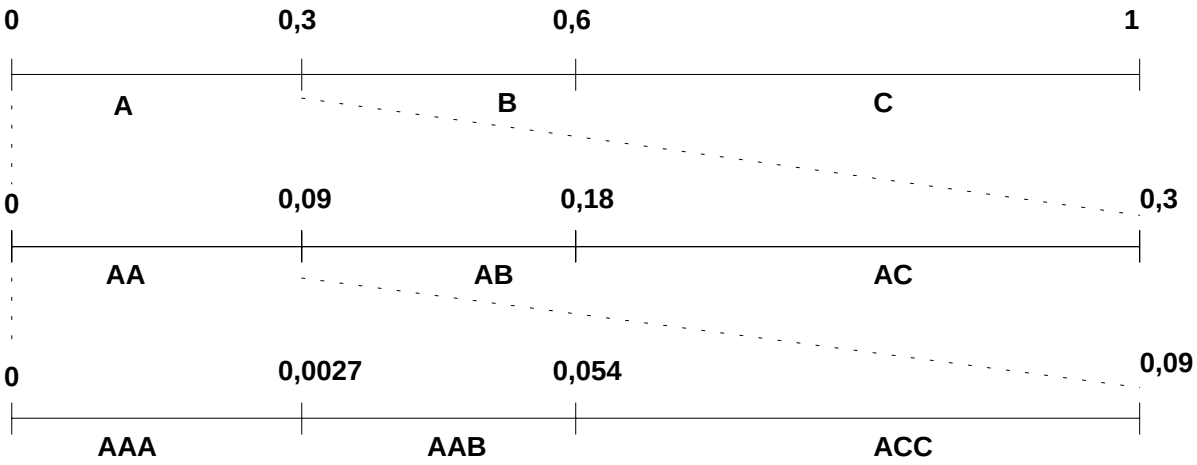
Übertragung [0,12; 0,15) z.B. 0, 12

Aktualisierung der Wahrscheinlichkeiten

$$P_{neu} = \frac{P_{alt} + P_{aktuell}}{2}$$

Neue Wahrscheinlichkeiten = {0,3; 0,3; 0,4}

Kodierung des zweiten Blocks



Übertragung [0,0027; 0,054) z.B. 0,05

Arithmetische Kodierung, Eigenschaften

- 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 so weniger Präzisionsbits.
- Die Code-Länge ist theoretisch optimal: die Anzahl von Bits pro Zeichen kann nicht-ganzzahlig sein und somit der Zeichenwahrscheinlichkeit genauer angepasst werden als bei der Huffman-Kodierung.
- 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)

Arithmetische Kodierung, Probleme:

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