

**Dr. Jürg M. Stettbacher**

Neugutstrasse 54

CH-8600 Dübendorf

---

Telefon: +41 43 299 57 23

E-Mail: [dsp@stettbacher.ch](mailto:dsp@stettbacher.ch)

# **Quellencodierung**

## **Grundlagen der Datenkompression**

Version 1.00  
2018-02-19

Zusammenfassung: Ausgehend von den Grundzügen der Informationstheorie werden verschiedene Verfahren der Quellencodierung behandelt. Dabei geht es um die Kompression von Daten.

# Inhaltsverzeichnis

<b>1 Zweck</b>	<b>3</b>
<b>2 Einleitung</b>	<b>3</b>
<b>3 Quellencodierungstheorem</b>	<b>4</b>
3.1 Quellencode . . . . .	5
3.2 Codewortlänge . . . . .	6
3.3 Redundanz . . . . .	7
3.4 Theorem zur Quellencodierung . . . . .	9
<b>4 Lauflängencodierung</b>	<b>9</b>
<b>5 Huffman Codes</b>	<b>13</b>
5.1 Einführendes Beispiel . . . . .	13
5.2 Verfahren . . . . .	15
5.3 Anwendungen . . . . .	15
<b>6 Lempel Ziv Codes</b>	<b>20</b>
<b>7 Arithmetische Codes</b>	<b>20</b>
<b>8 JPEG</b>	<b>20</b>
<b>9 Ende</b>	<b>20</b>

Früher produzierten wir Automobile in Massen,  
heute Information.

*John Naisbitt*

# **1 Zweck**

Ziel dieses Dokuments ist es,

- Die Absicht der *Quellencodierung* aufzueigen,
- mit Hilfe der *Informationstheorie* ihre Grenzen abzustecken,
- sowie verschiedene Verfahren der Quellencodierung anhand von Beispielen<sup>1</sup> als exemplarische Vertreter vorzustellen.

# **2 Einleitung**

Als Überblick betrachten wir die folgende bekannte Darstellung (siehe Abbildung 1). Eine Datenquelle<sup>2</sup> produziert einen Datenstrom. Wir nehmen vorläufig an, dass die einzelnen Datenwerte einen zufälligen Charakter haben. Diese Daten sollen nun übertragen werden. Aber der Datenkanal ist nicht perfekt, nicht sicher und die Übertragung ist nicht kostenlos. Das heisst, es entstehen Übertragungsfehler und Kosten. Dem Aspekt des Datenschutzes wollen wir an dieser Stelle nicht weiter nachgehen. Um unnötige Kosten und Übertragungsfehler zu verhindern werden die Daten der Quelle zuerst aufbereitet, bevor sie übertragen werden:

- Um Kosten zu sparen werden sie komprimiert.
- Zur Verhinderung von Übertragungsfehlern verwendet man einen speziellen Fehlerschutzcode.

Im Folgenden betrachten wir das Thema der Datenkompression und in dem Kontext stellen sich sofort einige Fragen, zum Beispiel diese:

---

<sup>1</sup> In diesem Dokument sind Beispiele zur Kennzeichnung von den Symbolen ▼ und ▲ umschlossen.

<sup>2</sup> Ein Beispiel für eine Datenquellen könnte etwa eine Textdatei sein, wenn sie ausgelesen wird, oder das Mikrofon in einem Telefon, das eine Abfolge von digitalisierten Schalldruckwerten liefert, oder ein physikalischer Sensor, der über ein Netzwerk periodisch Wetterdaten verschickt.

## Quellencodierung

*Drausser rechts ist kein code als ein Bit mehr oder weniger*

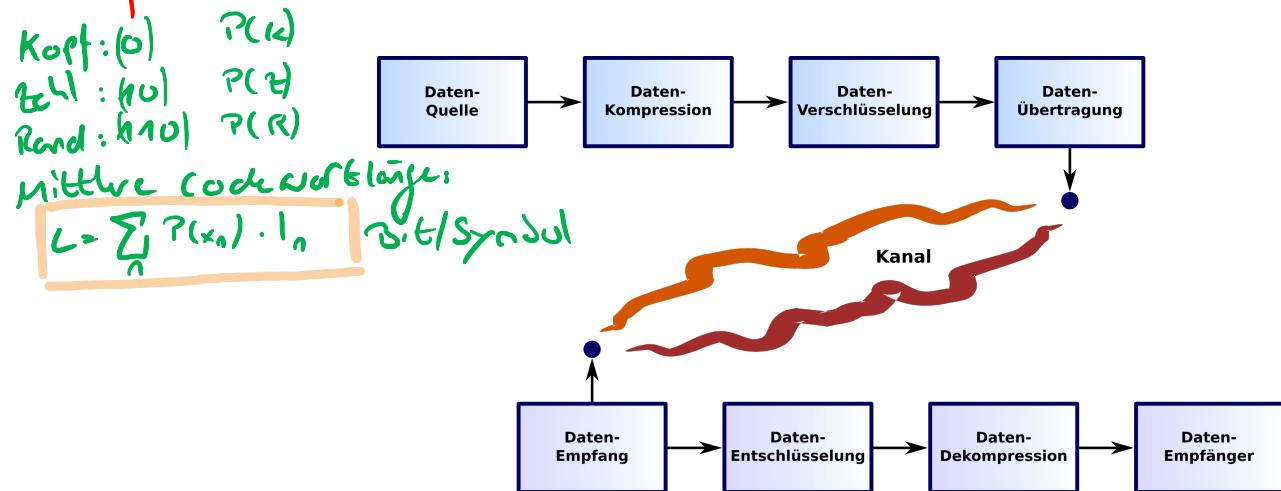


Abbildung 1: Schema eines allgemeinen Kommunikationssystems.

Wie oft muss man eine Textdatei durch das Zip-Programm<sup>3</sup> schicken bis die Datei nur noch 1 Bit gross ist? Haben Sie sich diese Frage auch schon gestellt? Die Informationstheorie gibt darauf eine sehr einfache Antwort. Sie sagt nämlich, dass das in der Regel<sup>4</sup> gar nicht möglich ist.

Das sogenannte Quellencodierungstheorem nennt eine klare Grenze der Komprimierbarkeit und ist somit von zentraler Bedeutung wenn es zum Beispiel um die Beurteilung von Kompressionsverfahren geht. Im Folgenden wird das Theorem behandelt, bevor wir damit beginnen, einige bekannte Quellen-codierungsverfahren zu erläutern. Bei den behandelten Beispielen verzichten wir auf die Betrachtung aller Details und beschränken uns auf die hauptsächlichen Prinzipien.

## 3 Quellencodierungstheorem

Aus der Informationstheorie kennen wir die folgenden zentralen Begriffe:

- Information (in Bit)

<sup>3</sup> Zip ist eine Familie von Datei-Kompressionsprogrammen. Unter Linux heisst das Programm einfach *zip*. Unter Windows heissen die bekanntesten Versionen PKZIP, WinZip, 7-Zip, etc.

<sup>4</sup> Mit *in der Regel* sei die Aussage vorerst auf eine normale Datei beschränkt, die zum Beispiel einen gewöhnlichen Satz in deutscher Sprache enthält.

- Entropie (in Bit/Symbol)

Zur Erinnerung: Die Information, resp. der Informationsgehalt  $I(x_n)$  bezieht sich auf ein Symbol  $x_n$ , das von einer Quelle  $Q$  erzeugt wird. Die Entropie  $H(Q)$  ist der Erwartungswert<sup>5</sup> der Information  $I(x_n)$  aller Symbole  $x_n$  der Quelle  $Q$  und bezieht sich damit auf die Quelle selbst. Wir sagen auch: Die Entropie ist die mittlere Information der Quelle pro Symbol.

Für die weiteren Berachtungen benötigen wir nun einige neue Begriffe.

### 3.1 Quellencode

Wir setzen voraus, dass die Symbole  $x_n$  der Quelle  $Q$  schon in der Form von irgendwelchen binären Mustern daher kommen. Wir wollen gerade zwei Beispiele dazu ansehen.

▼ Die Quelle *Münzwurf* liefert drei Symbole  $x_n$  mit  $n = 0 \dots 2$ . Dabei steht  $x_0$  für *Kopf*,  $x_1$  für *Zahl* und  $x_2$  wird verwendet für den Fall, wo die Münze auf dem Rand stehen bleibt oder vom Tisch fällt. Die Ereignisse werden von der Quelle gemäss der folgenden Tabelle codiert:

Symbol	Code
$x_0$	$\underline{c}_0 = (10)$
$x_1$	$\underline{c}_1 = (110)$
$x_2$	$\underline{c}_2 = (1110)$

Beachte, dass die Codeworte unterschiedlich lang sind, was absolut zulässig ist. Werden die Codeworte allerdings seriell als Bitstrom übertragen, was oft der Fall ist, so muss darauf geachtet werden, dass kein Codewort eine Vorsilbe eines anderen Codewortes ist. Wäre zum Beispiel bei einem anderen Code das Codewort  $q_0 = (101)$  und das Codewort  $q_1 = (1011)$ , so könnte der Empfänger nach Empfangen der ersten drei Bits nicht entscheiden, ob er nun  $q_0$  bekommen hat, oder den Anfang von  $q_1$ . Man Codes, bei denen kein Codewort Vorsilbe eines anderen Codewortes ist, *präfixfreie Codes*. ▲

▼ Wir können das Werfen eines Würfels als Datenquelle betrachten, die Zufallszahlen  $y_m$  zwischen eins und sechs liefert. Mit je drei Bits können allen Symbole dargestellt werden:

---

<sup>5</sup> Der Erwartungswert ist ein gewichteter Mittelwert. Das heisst im konkreten Fall der Entropie, dass der Informationsgehalt von jedem Symbol bei der Mittelwertbildung mit seiner Auftretenswahrscheinlichkeit gewichtet wird. Seltene Symbole tragen folglich wenig, häufige Symbole tragen viel zur Entropie bei.

Symbol	Code
$y_1 = 1$	$\underline{c}_1 = (001)$
$y_2 = 2$	$\underline{c}_2 = (010)$
...	...
$y_6 = 6$	$\underline{c}_2 = (110)$

Wir haben also die Symbole  $y_m$  mit  $m = 1 \dots 6$  und  $m$  entspricht gerade auch der betreffenden Augenzahl des Würfels. Zudem ist jedem Symbol  $x_m$  ein Codewort  $c_m$  zugeordnet, das aus dem Binärwert der Augenzahl besteht. Beachte, dass die Codeworte (000) und (111) im vorliegenden Beispiel nicht definiert sind und daher nie auftreten dürfen. ▲

## 3.2 Codewortlänge

Wir gehen weiterhin davon aus, dass wir es mit binären Codes zu tun haben. Die Codewortlänge  $\ell_n$  bezieht sich auf das Codewort  $c_n$  eines bestimmten Symbols  $x_n$  und gibt an, aus wievielen Bits das Codewort besteht. Wir nehmen die Beispiele von oben nochmals auf.

▼ Die Codeworte  $c_n$  mit  $n = 0 \dots 2$  beim Münzwurf haben die folgenden Längen:

Symbol	Code	Codewortlänge
$x_0$	$\underline{c}_0 = (10)$	$\ell_0 = 2$ Bit
$x_1$	$\underline{c}_1 = (110)$	$\ell_1 = 3$ Bit
$x_2$	$\underline{c}_2 = (1110)$	$\ell_2 = 4$ Bit

▲

▼ Die Codeworte der Würfel-Quelle mit den Ereignissen  $y_m$  mit  $m = 1 \dots 6$  haben alle eine Länge von drei Bit. Also ist in diesem Fall  $\ell_m = 3$  Bit für alle Codeworte  $c_m$ . ▲

Zum Schluss wollen wir die mittlere Codewortlänge  $L$  einer Quelle angeben, welche die Symbole  $x_n$  mit  $n = 0 \dots N - 1$  liefert. Dabei müssen wir berücksichtigen, dass die Codeworte  $c_n$  allenfalls unterschiedlich oft auftreten. Das tun man, indem man  $L$  als Erwartungswert der Codewortlängen  $\ell_n$  bildet:

$$L = \sum_{n=0}^{N-1} P(x_n) \cdot \ell_n \quad (\text{Bit/Symbol}) \quad (1)$$

Beachte, dass wir die Einheit *Bit/Symbol* verwenden um anzugeben, dass es sich bei dieser Grösse um einen Mittelwert pro Symbol handelt. Wir setzen unsere beiden Beispiele fort:

- ▼ Für den Münzwurf müssen wir uns zuerst die Wahrscheinlichkeiten  $P(x_n)$  beschaffen. Zu diesem Zweck werfen wir die Münze sehr oft (zum Beispiel 10'000 mal) und zählen aus, wie häufig jedes Ereignis auftritt. Wenn wir dann noch durch die gesamte Anzahl Würfe dividieren, erhalten wir eine Approximation von  $P(x_n)$ .

Symbol	Code	Codewortlänge	Wahrscheinlichkeit
$x_0$	$\underline{c}_0 = (10)$	$\ell_0 = 2$ Bit	$P(x_0) = 0.45$
$x_1$	$\underline{c}_1 = (110)$	$\ell_1 = 3$ Bit	$P(x_1) = 0.47$
$x_2$	$\underline{c}_2 = (1110)$	$\ell_2 = 4$ Bit	$P(x_2) = 0.08$

Beachte, dass die Summe aller Wahrscheinlichkeiten eins sein muss. Wir können jetzt die mittlere Codewortlänge ausrechnen:

$$L = P(x_0) \cdot \ell_0 + P(x_1) \cdot \ell_1 + P(x_2) \cdot \ell_2 = 2.65 \text{ Bit/Symbol}$$

Würde man also die Quelle sehr lange beobachten und schliesslich die mittlere Länge über alle gesehenen Codeworte bilden, so würde das selbe Resultat von 2.65 Bit/Symbol heraus kommen. ▲

- ▼ Beim Würfelexperiment waren von Anfang an alle Codeworte gleich lang, nämlich 3 Bit. Unabhängig von den Wahrscheinlichkeiten wird demnach auch die mittlere Codewortlänge  $L = 3$  Bit/Symbol.

▲

### 3.3 Redundanz

Genau genommen gibt es zwei Definitionen der Redundanz:

- Die Redundanz einer Quelle<sup>6</sup>.
- Die Redundanz eines Codes.

---

<sup>6</sup> Die Redundanz  $R$  einer Quelle ist definiert als  $R_Q = \log_2(N) - H$ , wenn  $N$  Anzahl verschiedener Symbole der Quelle ist, und  $H$  die Entropie der Quelle. In unserem Kontext hat diese Form der Redundanz jedoch keine Bedeutung.

Da wir nur Quellen betrachten, die bereits einen Code ausgeben, interessieren wir uns im Folgenden nur für die Redundanz von Codes. Die Definition dieser Redundanz  $R$  lautet:

$$R = L - H$$

(Bit/Symbol)

*Die Bits welche keine  
Information transportieren  
(2) so überflüssig*

Die Redundanz ist also die Differenz von mittlerer Codewortlänge  $L$  und Entropie  $H$  einer Quelle.

Zur Erinnerung: Die Entropie  $H$  ist die mittlere Information pro Symbol einer Quelle. Wir geben sie in Bit/Symbol an, genauso wie die mittlere Codewortlänge. Die Entropie sagt uns also, wieviele Bits pro Codewort (minimal) notwendig sind, um die Information der Quelle zu codieren.

Die mittlere Codewortlänge gibt an, wieviele Bits tatsächlich pro Codewort vorhanden sind. Ist also  $R > 0$ , so umfasst der Code mehr Bits als notwendig sind, um die Information wiederzugeben. Ist  $R < 0$ , so reicht die Anzahl Bits pro Codewort nicht aus, um die Information darzustellen. Die Information der Quelle lässt sich demnach mit einem derartigen Code nicht transportieren. Ein Teil der Information geht zwangsläufig verloren.

Wir möchten nochmals die Beispiele von oben fortsetzen:

▼ Für den *Münzwurf* haben wir oben bereits die Wahrscheinlichkeiten der Symbole  $x_n$  mit  $n = 0 \dots 2$  beziffert.

Symbol	Code	Codewortlänge	Wahrscheinlichkeit
$x_0$	$c_0 = (10)$	$\ell_0 = 2$ Bit	$P(x_0) = 0.45$
$x_1$	$c_1 = (110)$	$\ell_1 = 3$ Bit	$P(x_1) = 0.47$
$x_2$	$c_2 = (1110)$	$\ell_2 = 4$ Bit	$P(x_2) = 0.08$

Damit folgt sofort die Entropie  $H$ :

$$H = P(x_0) \cdot \log_2 \frac{1}{P(x_0)} + P(x_1) \cdot \log_2 \frac{1}{P(x_1)} + P(x_2) \cdot \log_2 \frac{1}{P(x_2)} = 1.32 \text{ Bit/Symbol}$$

Die mittlere Codewortlänge  $L$  haben wir oben schon berechnet:

$$L = 2.65 \text{ Bit/Symbol}$$

Damit folgt die Redundanz  $R$ :

$$R = L - H = 1.33 \text{ Bit/Symbol}$$

Man erkennt, dass bei diesem Code rund die Hälfte aller Bits keine informationstragende Funktion haben. ▲

## Quellencodierung

▼ Beim Würfeln hat jede der sechs Augenzahlen dieselbe Auftretenswahrscheinlichkeit, nämlich  $P(y_m) = 1/6$  mit  $m = 1 \dots 6$ . Damit folgt die Entropie  $H$ :

$$H = \sum_{m=1}^6 P(y_m) \cdot \log_2 \frac{1}{P(y_m)} = \log_2(6) = 2.59 \text{ Bit/Symbol}$$

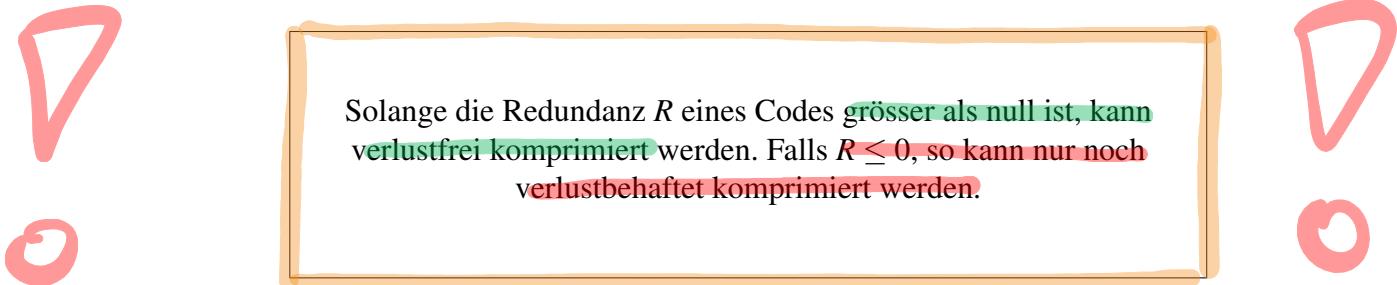
Mit der oben schon gefundenen mittlere Codewortlnge  $L = 3$  Bit/Symbol folgt die Redundanz:

$$R = L - H = 0.41 \text{ Bit/Symbol}$$



## 3.4 Theorem zur Quellencodierung

Das Theorem betrachtet eine Quelle, resp. einen Code und geht aus von der Redundanz  $R$ :



Ausgedrückt durch die mittlere Codewortlänge  $L$  und die Entropie  $H$ : Man kann verlustfrei komprimieren, solange  $L > H$  ist. Im Idealfall der verlustfreien Kompression wird demnach  $L = H$ . Im letzteren Fall lässt sich nur noch verlustbehaftet weiter komprimieren.

Das Ziel der Quellencodierung besteht normalerweise darin, die Redundanz mit vertretbarem Aufwand zu entfernen oder mindestens zu reduzieren. Zu diesem Zweck wird der vorliegende Code (resp. die betreffenden Symbole) in einen neuen Code übersetzt, der bezüglich Redundanz bessere Eigenschaften hat. Die folgenden Kapitel behandeln entsprechende Verfahren.

## 4 Lauflängencodierung

Die Lauflängencodierung (englisch *Run Length Encoding*, RLE) ist eine lose Sammlung von ähnlichen Methoden zur Komprimierung von Sequenzen (englisch *Runs*) identischer Symbole. Es werden im folgenden zwei typische Anwendungen gezeigt.

A: Aufstarten  
P: Produktivität  
R: Reload  
E: Fehler

möglicher Protokoll: A A A PPP... P R R R R PPP... ? EEE EEE P

Seite 9

Marker = das Symbol an dem es am weitesten gedreht wird.

## Quellencodierung

**Strings von Zeichen** Betrachten wir beispielsweise den folgenden String, der nur aus binär codierten Grossbuchstaben besteht:

... TERRRRRRRRMAUIIIIIIIIIIIIIWQCSSSSSSSSSL ...

Man erkennt sofort, dass sich immer wieder einzelne Symbole mehrfach wiederholen. Diese Wiederholungen könnte man zusammenfassen, zum Beispiel so:

... TE (9xR) MAU (17xI) WQC (10xS) L ...

Offensichtlich ist der String jetzt kürzer. Wir haben jeden Run zur Markierung in Klammern gesetzt und in der Klammer angegeben, wie oft welches Zeichen vorkommt. Wir nennen das auch einen *Token*. Der Empfänger erkennt mit Hilfe der Klammer den Run und kann ihn wieder expandieren. Aber es gibt ein Problem: Wir verwenden nun Klammerzeichen um die Token zu markieren, aber Klammern kommen im ursprünglichen Zeichensatz gar nicht vor. Nebenbei sei noch erwähnt, dass das kleine  $x$  im ursprünglichen Zeichensatz genauso wenig vorkommt. Statt den Klammern ist also ein anderer Marker gesucht, einer, der im Zeichensatz vorkommt, aber selten gebraucht wird. Wir wollen nun annehmen, dass die Analyse einer längeren Sequenz ergibt, dass A das seltenste Zeichen ist. Damit erhält man:

... TEA09RMA01AUA17IWQCA10SL ...

Zur Verbesserung der Sichtbarkeit wurden die zusammengefassten Runs unterstrichen. Betrachten wir den ersten Token A09R, so ist A der Marker, der einen Run einleitet. 09 symbolisiert den Zähler. Beachte: Ziffern kommen im Zeichensatz ebenfalls nicht vor. Aber wir können den Zähler ohne Weiteres binär mit einer fixen Breite einsetzen, zum Beispiel mit 6 Bit. Damit könnten Runs bis zur Länge 63 codiert werden. Das kleine  $x$  wurde fallen gelassen. Statt dessen folgt gleich das Zeichen, das im Run wiederholt wird. Der Empfänger muss das Format selbstverständlich kennen. Dann aber kann er nach jedem Marker einfach die nächsten 6 Bit lesen, als Zahl, resp. Zähler interpretieren, dann das nächste Zeichen lesen und entsprechend dem Zähler wiederholen.

Wir wollen noch den zweiten Token A01A ansehen, denn dieser kam im Vorschlag mit Klammern noch gar nicht vor. Im ursprünglichen String steht an dieser Stelle allein der Buchstabe A, der uns nun als Marker dient. Wenn der Empfänger das A liest, wird er es als Marker verstehen und die folgenden Bits entsprechend decodieren. Aus diesem Grund müssen wir zwingend bei jedem Auftreten des Markers im ursprünglichen Text, ihn durch einen Run der Länge eins ersetzen. Das wurde hier gemacht.

Zusammenfassend lautet die Anleitung für das eben beschriebene Verfahren so:

- Definiere vorgängig einen Marker, resp. ein gültiges Zeichen, das in den zu verarbeitenden Texten selten vorkommt.
- Definiere vorgängig eine Zählerbreite (in Bits), so dass Runs der typischen Länge damit erfasst werden können.

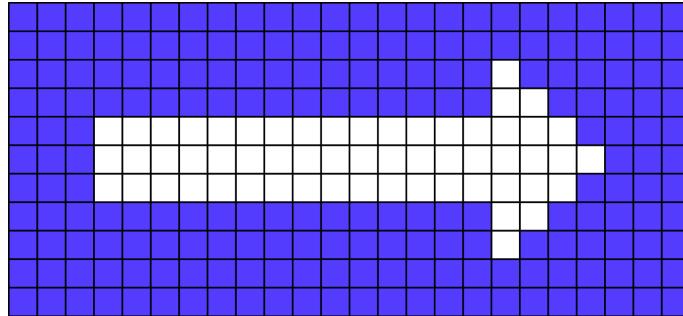


Abbildung 2: Bild mit einem Bit pro Pixel.

- Ersetze im Urtext jeden Run durch einen Token. Ist der Run länger, als was der Token abbilden kann, so bilde mehrere Token hintereinander.
- Ersetze im Urtext alle verbleibenden Marker-Zeichen durch einen Token mit der Run-Länge eins.

Der Decoder sucht lediglich nach Markern und expandiert die betreffenden Token.

**Bilder (serialisiert)** In einfachen Bildern mit Flächen derselben Farbe lässt sich ein ähnliches Verfahren anwenden. Als Beispiel dient Abbildung 2. Jeder Pixel besteht aus einem Bit und kann nur schwarz (0) oder weiss (1) sein. Das Bild ist 24 Pixel breit und 11 Pixel hoch. Total umfasst das Bild also 264 Pixel an je ein Bit, also 33 Byte, wenn das Bild unkomprimiert gespeichert wird.

Mit Hilfe der Lauflängencodierung lassen sich nun Pixel identischer Farbe zusammenfassen. Zu diesem Zweck wird das Bild zuerst serialisiert. Das heisst, wir denken uns die einzelnen Zeilen alle aneinander aufgereiht. Selbstverständlich wäre das mit Spalten genauso möglich, aber wir wählen hier die zeilenweise Serialisierung.

Beginnend mit schwarzen Pixeln wird nun im serialisierten Bild gezählt, wieviele solche Pixel auftreten bis zum ersten weissen Pixel. Dann werden die weissen Pixel gezählt, bis wieder ein schwarzer auftritt, und so weiter. Wir erhalten die folgende Zahlenreihe (die Klammern, Kommas und Abstände dienen nur der Darstellung und würden im binären String selbstverständlich nicht vorkommen):

$$( 65, 1, 23, 2, 8, 17, 7, 18, 6, 17, 21, 2, 22, 1, 54 )$$

Die Summe all dieser Zahlen muss wieder 264 ergeben, die Gesamtzahl der Pixel im Bild. Da aus der Zahlenreihe die Dimension des Bildes (24 x 11 Pixel) nicht mehr ersichtlich ist, muss dies der Zahlenreihe noch voran gestellt sein. Somit erhalten wir als komprimierte Variante des Bildes diesen Zahlenvektor:

$$( 24, 11, 65, 1, 23, 2, 8, 17, 7, 18, 6, 17, 21, 2, 22, 1, 54 )$$

Offen ist noch die Frage, nach der Anzahl Bits, mit der jede Zahl dargestellt wird. Selbstverständlich müssen wir jede Zahl in gleicher Weise darstellen. Ein konservativer Vorschlag wäre 9 Bit pro Zahl. Damit kann man bis 511 zählen. Würde das Bild also nur aus schwarzen Pixeln bestehen, so könnte das gesamte Bild mit nur einer Zahl (plus Dimension) dargestellt werden. Unser Vektor besteht aus 17 Zahlen, total also aus 153 Bit. Damit wird das Bild nicht ganz auf die halbe Grösse komprimiert.

Da Bilder mit nur einer Farbe nicht allzu spannend sind und in der Praxis vermutlich nicht allzu oft auftreten, könnte man sich auch darauf einigen, beispielsweise nur 6 Bit pro Pixel vorzusehen. Damit kann man Runs bis zur Länge 63 zählen. Damit lässt sich allerdings die dritte Zahl in unserem Vektor nicht mehr darstellen. Darum muss man die 65 ersetzen durch 63, 0, 2. Das bedeutet, dass zuerst 63 schwarze Pixel kommen, dann null weisse und nochmals 2 schwarze, total also 65 schwarze Pixel nacheinander. Wir erhalten den neuen Zahlenvektor:

$$(24, 11, 63, 0, 2, 1, 23, 2, 8, 17, 7, 18, 6, 17, 21, 2, 22, 1, 54)$$

Es resultieren folglich 19 Werte an je 6 Bit, total also 114 Bit. Wir geben noch die Kompressionsrate<sup>7</sup>  $R$  an als Quotient von komprimierten Bits durch originale Bits:

$$R = \frac{114}{264} \approx 0.43$$

Zusammenfassend lässt sich das Verfahren so beschreiben:

- Definiere vorgängig die Bitbreite des Pixelzählers<sup>8</sup>.
- Definiere vorgängig die Pixelfarbe mit der begonnen wird.
- Serialisierung des Bildes.
- Falls der erste Pixel nicht die vordefinierte Farbe hat, setze die erste Pixelzahl auf null.
- Zähle jeweils alle Pixel gleicher Farbe bis zum nächsten Farbwechsel.
- Sollte eine Pixelzahl grösser werden als das Maximum des Zählers, so teile die Zahl auf, wie oben gezeigt.

---

<sup>7</sup> Die Kompressionsrate wird zwar oft mit dem Symbol  $R$  bezeichnet, wie die Redundanz. Die beiden Grössen haben aber nichts miteinander zu tun und dürfen nicht verwechselt werden.

<sup>8</sup> Die Bitbreite sollte so gewählt werden, dass damit auch die Bilddimension wiedergegeben werden kann. Alternativ kann für die Bilddimension (die ersten beiden Werte im Vektor) eine separate Bitbreite definiert werden.

**Bilder (zeilenweise)** Statt die Zeilen eines Bildes zuerst aneinander zu reihen, um dann zusätzlich die Bilddimension angeben zu müssen, könnte man das Bild auch zeilenweise komprimieren. Das lohnt sich vor allem bei grösseren Bildern, bedingt aber, dass ein bestimmter Zahlwert (in der Regel der Maximalwert) reserviert wird, um anzugeben, wann eine Zeile fertig ist. Im Übrigen wäre das Verfahren identisch mit dem zuvor beschriebenen.

Als Beispiel wird das Bild in Abbildung 2 nochmals aber zeilenweise komprimiert. Dabei sei der Pixelzähler 5 Bit breit und wir beginnen wieder mit schwarzen Pixeln. Mit 5 Bit können wir 0 bis 31 Pixel zählen, wobei wir den Wert 31 für das Zeilenende reservieren. Damit folgt der Zahlenvektor (Bildzeilen getrennt):

$$( 24, 31, \\ 24, 31, \\ 17, 1, 6, 31, \\ 17, 2, 5, 31, \\ 3, 17, 4, 31, \\ 3, 18, 3, 31 \\ 3, 17, 4, 31, \\ 17, 2, 5, 31, \\ 17, 1, 6, 31, \\ 24, 31, \\ 24, 31 )$$

Der Vektor besteht nun aus 36 Einträgen mit total 180 Bit. Beachte, dass die Bilddimension nicht mehr angegeben werden muss, da jede Zeile genau 24 Pixel (Bildbreite) umfasst und sich zusammen mit der Anzahl Zeilen (Anzahl des Wertes 31) die Bilddimension automatisch ergibt. Die Kompressionsrate  $R$  beträgt:

$$R = \frac{180}{264} \approx 0.68$$

## 5 Huffman Codes

### 5.1 Einführendes Beispiel

Es sei gleich ein Beispiel voran gestellt: Eine Quelle  $Q$  liefere vier verschiedene Symbole, nämlich  $A, B, C$  und  $D$ . Es ist naheliegend für die Symbole die folgenden Codeworte zu wählen:

häufige Symbole  $\rightarrow$  Kurze Codeworte  
seltene Symbole  $\rightarrow$  lange Codeworte  
zus.: Präfixfrei

Symbol	Nummerncode
A	(00)
B	(01)
C	(10)
D	(11)

Es wurden die Symbole also einfach binär nummeriert. Daher nennen wir einen solchen Code auch einen *Nummerncode*. Im Folgenden werden wir zeigen, dass dies oft nicht die beste Lösung ist. Betrachten wir nun die Wahrscheinlichkeiten der Symbole:

$$P(A) = 0.60 \quad P(B) = 0.30 \quad P(C) = 0.05 \quad P(D) = 0.05$$

Damit folgt die Entropie  $H(Q)$  der Quelle  $Q$ :

$$H(Q) = \sum_{x=A}^D P(x) \cdot \log_2 \frac{1}{P(x)} = 1.40 \text{ Bit/Symbol}$$

Da der oben vorgeschlagene Nummerncode die mittlere Codewortlänge  $L_N = 2$  Bit hat, erhalten wir eine Redundanz von  $R_N = 0.60$  Bit. Das Quellencodierungstheorem besagt, dass man einen Code finden können, der ohne Informationsverlust die Symbole A bis D mit durchschnittlich  $L_{opt} = H(Q) = 1.40$  Bit/Symbol codiert.  $L_{opt}$  bezeichnet dabei die optimale mittlere Codewortlänge. Die Frage ist: Wie findet man einen solchen Code?

Huffmans<sup>9</sup> Idee ist es nun, unterschiedlich lange Codeworte zu verwenden. Häufige Symbole erhalten kurze Codeworte, seltene Symbole lange Codeworte. In unserem Beispiel wären dies die folgenden (wir werden gleich zeigen, wie man darauf kommt):

Symbol	Huffman Code
A	(1)
B	(01)
C	(001)
D	(000)

Beachte, dass Huffman Codes - sollen sie seriell übertragen werden - präfixfrei sein müssen. Was was bedeutet wurde weiter oben in Kapitel 3.1 schon erläutert. Man erhält nun die mittlere Codewortlänge  $L_H$  des Huffman Codes:

$$L_H = \sum_{x=A}^D P(x) \cdot \ell(x) = 1.50 \text{ Bit/Symbol}$$

Das bedeutet, wir erreichen die perfekte Lösung  $L_{opt} = H(Q) = 1.40$  Bit/Symbol nicht ganz, sind aber schon recht nahe.

---

<sup>9</sup> David Albert Huffman (1925 bis 1999), amerikanischer Pionier der Computerwissenschaften.

## 5.2 Verfahren

Einen Huffman Code findet man mit dem folgenden grafischen Verfahren. Es handelt sich dabei um einen sogenannten Huffman-Baum, der oben Blätter hat, darunter Äste und schliesslich zuunterst einen Stamm<sup>10</sup>.

1. Ordne alle Symbole nach aufsteigenen Auftretenswahrscheinlichkeiten auf einer Zeile. Dies sind die Blätter des Huffman-Baums.  
Gibt es Symbole mit gleichen Wahrscheinlichkeiten, so spielt die Reihenfolge unter ihnen keine Rolle.
2. Notiere unter jedes Blatt seine Wahrscheinlichkeit.
3. Schliesse die beiden Blätter mit der kleinsten Wahrscheinlichkeit an einer gemeinsamen Astgabel an. Ordne dem Ast die Summe der Wahrscheinlichkeiten der beiden Blätter zu.  
Gibt es mehrere mögliche Kombinationen von Blättern mit den kleinsten Wahrscheinlichkeiten, so spielt es keine Rolle, welche man davon auswählt.
4. Wiederhole Punkt 3 mit Blättern und Ästen so lange, bis nur noch der Stamm des Baums übrig bleibt.
5. Nun wird festgelegt, ob bei jeder Astgabel der linke Zweig eine 0 oder eine 1 erhält. Der rechte Zweig erhält dann das Komplement.
6. Nun werden auf dem Pfad vom Stamm zu jedem Blatt die Nullen und Einsen ausgelesen und von links nach rechts nebeneinander geschrieben. Dies sind die Huffman-Codeworte.

## 5.3 Anwendungen $\rightarrow$ Es muss a priori die Informationsdokumente sein

Anhand von Beispielen werden einige Anwendungen erläutert.

▼ Gegeben sei eine Quelle  $\Psi$ , welche die Symbole  $M, N, O, R$  und  $S$  produziert. Die folgenden Wahrscheinlichkeiten sind gegeben:

$$P(M) = 0.35 \quad P(N) = 0.20 \quad P(O) = 0.25 \quad P(R) = 0.05 \quad P(S) = 0.15$$

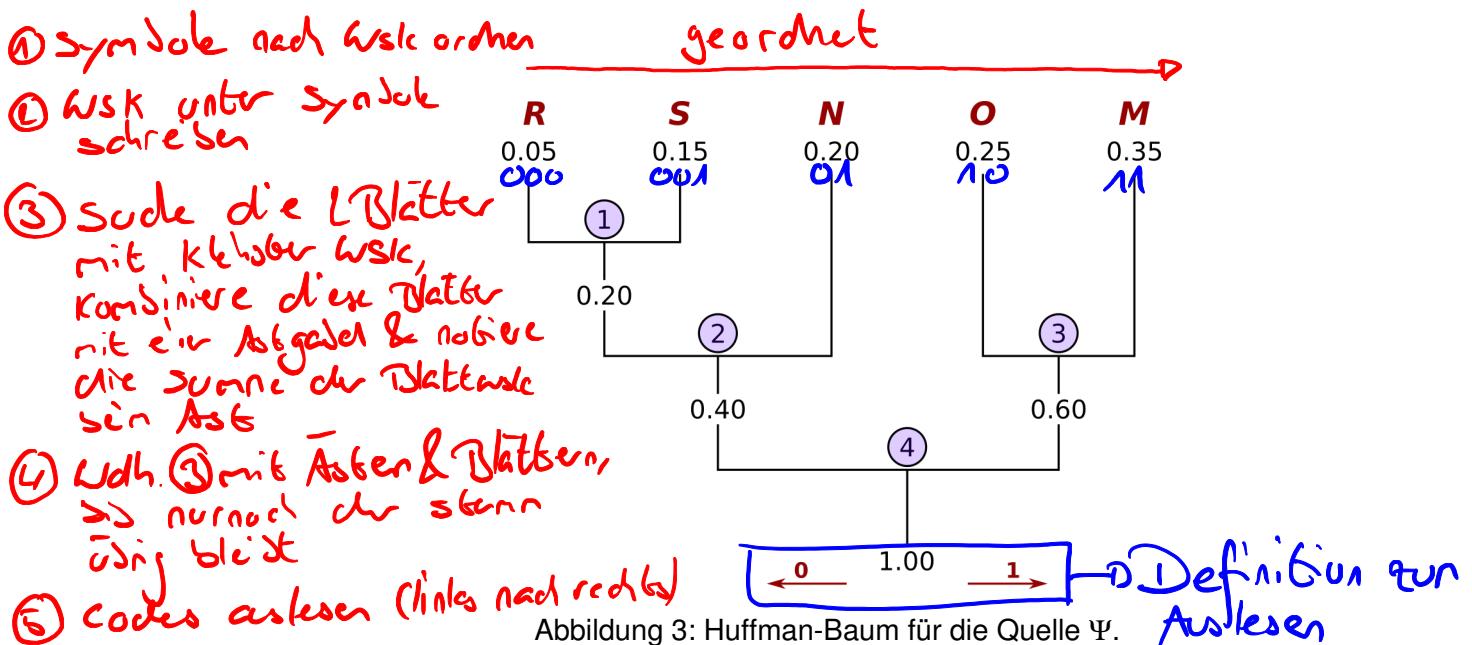
$H: 1.12 \text{ Bits/Symbol}$

Wir überprüfen noch kurz, ob nichts fehlt:

$$P(M) + P(N) + P(O) + P(R) + P(S) = \underline{\underline{1.00}}$$

---

<sup>10</sup> Selbstverständlich kann man den Baum auch liegend zeichnen. In der Regel sind die Blätter dann links und der Stamm rechts.



Es ist also alles in Ordnung. Nun wenden wir das oben beschriebene Huffman-Verfahren an, siehe Abbildung 3. Oben sind die nach Wahrscheinlichkeiten sortierten Blätter aufgetragen. Die Zahlen im Kreis geben die Reihenfolge an, in der die Äste gezeichnet werden. In jedem neuen Ast ist seine Wahrscheinlichkeit eingetragen, der Stamm hat folglich die Summe aller Wahrscheinlichkeiten, also eins. Beim Stamm sind mit den horizontalen Pfeilen noch die Ausleserichtungen angegeben. Suchen wir also beispielsweise das Codewort für das Symbol  $N$ , so beginnen wir mit Auslesen unten und fahren zuerst den Stamm hoch. Dann, beim Punkt Nr. 4, fahren wir nach links. Dies ergibt ein Null im Codewort. Dann fahren wir den Ast weiter nach oben. Beim Punkt Nr. 2 geht es nach rechts, was ein Eins im Codewort ergibt. Auf diesem Weg erreicht man schliesslich das Blatt  $N$ . Das Codewort dieses Blattes ist also (01). In analoger Weise findet man alle Codeworte mit den betreffenden Codewortlängen.

Symbol	Huffman Code	Codewortlänge
$M$	(11)	$\ell(M) = 2 \text{ Bit}$
$O$	(10)	$\ell(O) = 2 \text{ Bit}$
$N$	(01)	$\ell(N) = 2 \text{ Bit}$
$S$	(001)	$\ell(S) = 3 \text{ Bit}$
$R$	(000)	$\ell(R) = 3 \text{ Bit}$

$$\begin{aligned}
 L &= 0.33 \cdot 2 + 0.15 \cdot 1 + \\
 &\quad 0.1 \cdot 1 + 0.15 \cdot 3 + \\
 &\quad 0.05 \cdot 3 \\
 &= 0.7 + 0.5 + 0.4 + 0.45 \\
 &\quad + 0.15 = \underline{\underline{2.20 \text{ Bit/Symbol}}}
 \end{aligned}$$

Nun soll überprüft werden, wie erfolgreich dieser Code ist: Auf Grund der Wahrscheinlichkeiten folgt die Entropie  $H(\Psi) = 2.12 \text{ Bit/Symbol}$ . Die mittlere Codewortlänge resultiert zu  $L = 2.20 \text{ Bit/Symbol}$ . Es verbleibt also eine Redundanz von  $R = 0.08 \text{ Bit/Symbol}$ , was weniger als 4 % der mittleren Codewortlänge entspricht. Das Potenzial für Verbesserungen ist folglich nicht sehr gross. ▲

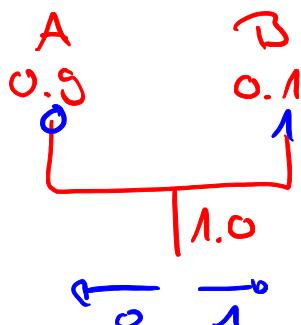
$$A \rightarrow P(A) = 0.9$$

$$B \rightarrow P(B) = 0.1$$

$$H: 0.47 \text{ Bit/Symbol} = 0.9 \log_2 \frac{1}{0.9} + 0.1 \log_2 \frac{1}{0.1}$$

$$L: 1.00 \text{ Bit/Symbol}$$

$$R: 0.53 \text{ Bit/Symbol}$$



$$AA \rightarrow P(AA) = P(A) \cdot P(A) = 0.81$$

Total: 1

$$AB \rightarrow P(AB) = P(A) \cdot P(B) = 0.09$$

$$BA \rightarrow P(BA) = P(B) \cdot P(A) = 0.09$$

$$BB \rightarrow P(BB) = P(B) \cdot P(B) = 0.01$$



$$H_2 = H_A + H_B = 0.99 \text{ Bit/Symbol}$$

somit statisch unabhängig

$$L_2 = 1.99 \text{ Bit/Symbol} \rightarrow L_1 = \frac{L_2}{2} = 0.69 \text{ Bit/Symbol}$$

$$R_2 = L_2 - H_2 = 0.35 \text{ Bit/Symbol} \rightarrow R_1 = \frac{R_2}{2} = 0.17 \text{ Bit/Symbol}$$

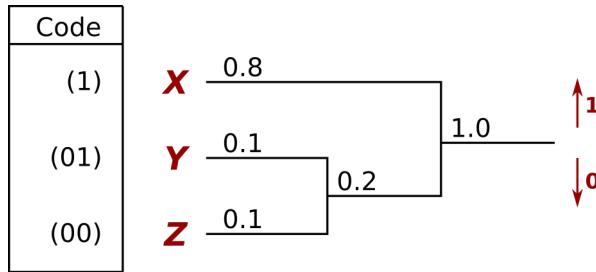


Abbildung 4: Huffman-Baum für die Quelle  $\Omega$ .

▼ Eine Quelle  $\Omega$  liefert statistisch unabhängige Symbole  $X$ ,  $Y$  und  $Z$  mit den Wahrscheinlichkeiten:

$$P(X) = 0.80 \quad P(Y) = 0.10 \quad P(Z) = 0.10$$

Den Huffman-Baum zeichnen wir diesmal horizontal, siehe Abbildung 4. Die Analyse liefert das Folgende:

$$\begin{aligned} H_1(\Omega) &= 0.922 \text{ Bit/Symbol} \\ L_1 &= 1.200 \text{ Bit/Symbol} \\ R_1 &= 0.278 \text{ Bit/Symbol} \end{aligned}$$

Wir werden anschliessend gleich sehen, warum die Grössen hier mit dem Index 1 versehen sind.

Fazit: Nahezu ein Viertel des Codes ist immer noch redundant. Das ist kein wirklich befriedigendes Resultat. Es fragt sich nur, Wie man das verbessern könnte, denn mit den drei Symbolen und ihren Wahrscheinlichkeiten ist kein anderer Huffman-Code möglich.

Das Rezept dafür ist das Folgende: Man bildet alle möglichen Doppelsymbole, das heisst, man fasst immer zwei aufeinander folgende Symbole der Quelle zusammen und codiert sie als Paar. Auf diese Weise vervielfacht sich die Anzahl Symbole, was zu einer feineren Granularität des Huffman-Codes führt und in der Folge eine bessere Annäherung an den Wert der Entropie zulässt. Falls das Resultat immer noch nicht zufriedenstellend sein sollte, so kann man Dreifachsymbole oder noch grössere Symbol-Gruppen bilden. Sind die betreffenden Symbole statistisch unabhängig voneinander, so lassen sich die Wahrscheinlichkeiten der Mehrfachsymbole (Verbundwahrscheinlichkeiten) als Produkt der Wahrscheinlichkeiten der involvierten Einzelsymbole berechnen. Besteht eine statistische Abhängigkeit zwischen Symbolen, so müssen die Verbundwahrscheinlichkeiten anderweitig ermittelt werden, zum Beispiel experimentell.

Wir wollen nun alle möglichen Doppelsymbole der Quelle  $\Omega$  mit den dazu gehörigen Wahrscheinlichkeiten auflisten. Von den drei Symbolen  $X$ ,  $Y$ ,  $Z$  kann jedes Symbol an erster Stelle und an zweiter Stelle auftreten. Folglich gibt es  $3 \cdot 3 = 9$  mögliche Doppelsymbole.

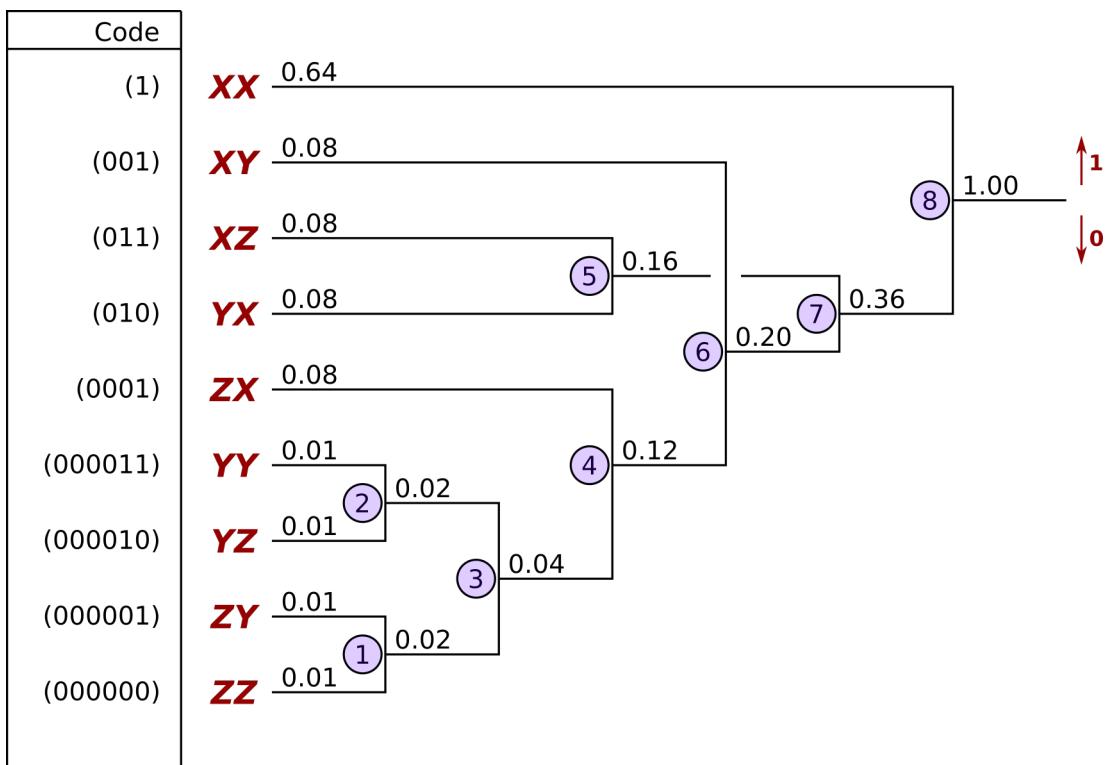


Abbildung 5: Huffman-Baum für Doppelsymbole der Quelle  $\Omega$ .

$$\begin{array}{lll}
 XX: P(XX) = 0.64 & YX: P(YX) = 0.08 & ZX: P(ZX) = 0.08 \\
 XY: P(XY) = 0.08 & YY: P(YY) = 0.01 & ZY: P(ZY) = 0.01 \\
 XZ: P(XZ) = 0.08 & YZ: P(YZ) = 0.01 & ZZ: P(ZZ) = 0.01
 \end{array}$$

Damit können wir nun einen neuen Huffman-Baum zeichnen, siehe Abbildung 5. In der Abbildung sind als didaktische Hilfe wieder die Entstehungsschritte des Baumes eingezeichnet (Zahlen in Kreisen). Im Normalfall kann man diese Nummern weglassen. Man erkennt sofort, dass der Huffman-Baum jetzt sehr viel grösser ist als jener mit Einfachsymbolen. Die Analyse liefert folgende Resultate:

$$\begin{aligned}
 H_2(\Omega) &= 1.844 \text{ Bit / 2 Symbole} \\
 L_2 &= 1.920 \text{ Bit / 2 Symbole} \\
 R_2 &= 0.076 \text{ Bit / 2 Symbole}
 \end{aligned}$$

Beachte: Alle diese Grössen beziehen sich jetzt auf Doppelsymbole. Das wird mit dem Index 2 angezeigt. Ferner ist die Einheit dieser Grössen jetzt nicht mehr Bit pro Quellsymbol, sondern Bit pro *zwei* Symbole. Wollen wir also diese Resultate mit dem Huffman-Code von Einfachsymbolen vergleichen, so müssen diesem Umstand Rechnung tragen, und zwar so:

$$\begin{array}{ccc}
 L_1 = 1.200 \text{ Bit / 1 Symbol} & \iff & \frac{L_2}{2} = 0.960 \text{ Bit / 1 Symbol} \\
 R_1 = 0.278 \text{ Bit / 1 Symbol} & \iff & \frac{R_2}{2} = 0.038 \text{ Bit / 1 Symbol}
 \end{array}$$

Was auf den ersten Blick erstaunen mag, ist dass mit dem zweiten Code die mittlere Codewortlänge  $L_2/2$  weniger als ein Bit pro Symbol beträgt. Das ist darum möglich, weil jedes Codewort (das kürzeste davon ist nur ein Bit lang) zwei Quellsymbole transportiert. Die Redundanz ist nun nur noch knapp 4 % der mittleren Codewortlänge. ▲

## **6 Lempel Ziv Codes**

## **7 Arithmetische Codes**

## **8 JPEG**

## **9 Ende**

If you understand everything,  
you must be misinformed.

*Sprichwort, angeblich aus Japan*