Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik VI
Prof. Dr.-Ing. Hermann Ney

Proseminar Datenkompression im WS 2004/2005

Huffman-Kodierung

Viktor Engelmann
Martin Zimmermann

Matrikelnummer 251 372
Matrikelnummer 250 702

16. November 2004

Betreuerin: Nicola Ueffing

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Ein Kompressionsalgorithmus ist eine Abbildung φ, die eine Zeichenfolge $A = (a_1, a_2, \ldots, a_n)$ auf eine Zeichenfolge $B = (b_1, b_2, \ldots, b_m)$ abbildet, wobei $m < n$ gelten soll:

$$\varphi(A) = B$$

Ist φ injektiv, existiert die Umkehrabbildung φ^{-1} und aus B kann A vollständig rekonstruiert werden, das heißt:

$$\varphi^{-1}(B) = \varphi^{-1}(\varphi(A)) = A$$

φ heißt dann **verlustlos**. Ist φ nicht injektiv, es existiert aber eine Abbildung ψ die

$$\psi(B) = \psi(\varphi(A)) \simeq A$$

erfüllt, so heißt φ **verlustbehaftet**.

Verlustlose Kompressionsalgorithmen finden dann Anwendung, wenn zum Beispiel Texte oder Zahlenfolgen komprimiert werden sollen, bei denen die Folge nach dem Dekomprimieren mit der Ausgangsfolge identisch sein muss. Werden dagegen Bilder komprimiert, können auch verlustbehaftete Algorithmen benutzt werden, die dann Farbunterschiede, die das menschliche Auge nicht mehr wahrnehmen kann, nicht mehr berücksichtigen. Das Bild kann stärker komprimiert werden, ohne dass für den Betrachter sichtbare Verluste entstehen.

Trotz Breitbandtechnik und stetig wachsender Datenträgerkapazitäten ist Kompression notwendig, wie folgende Beispiele zeigen:

Beispiel 1: Unkomprimierte Bilder:
Mit einer Auflösung von 1024 x 768 Pixeln und 24 Bit Farbtiefe benötigen Bilder immer

$$1024 \cdot 768 \cdot 24 \text{ Bits} = 18.874.368 \text{ Bits} = 2,25 \text{ Megabyte.}$$

Beispiel 2: Vergleich zwischen Audio-CD und MP3:
Eine CD hat eine Datenrate von 176 Kilobyte/Sekunde, ein MP3 in CD-Qualität eine Datenrate von 16 Kilobyte/Sekunde. Das entspricht einer (verlustbehafteten) Kompression um 91%.

Beispiel 3: Unkomprimierte Videos:
Mit einer Auflösung von 720 x 576 Pixeln, 25 Bildern pro Sekunde und 24 Bit Farbtiefe, wie bei PAL-DVDs üblich, passen auf eine DVD mit einer Kapazität von $8,5$ Gigabyte

$$\frac{8,5 \cdot 8.388.608 \text{ Bit}}{24 \cdot 25 \cdot 720 \cdot 576 \text{ Bit/sec}} = 293,45 \text{ sec}$$

Film ohne Ton, also weniger als 5 Minuten.

1.2 Aufbau der Ausarbeitung

Thema dieser Ausarbeitung ist die Huffman-Kodierung zur Kompression von Zeichenfolgen, die bei zeichenweiser Kodierung optimal ist.

In Kapitel 2 werden wichtige Begriffe, die an verschiedenen Stellen gebraucht werden, definiert (2.1) und Grundlagen aus der Informationstheorie eingeführt (2.3), mit denen später die Optimalität der Huffman-Kodierung bewiesen wird. Außerdem werden zwei Beispiele für Vorgänger der Huffman-Kodierung gezeigt, der nicht-präfixfreie Morse-Kode (2.4.1) und die nicht-optimale Shannon-Fano-Kodierung (2.4.2).

Im Kapitel 3 wird dann der Huffman-Kode eingeführt, die Funktionsweise erläutert (3.1) und an einem Beispiel gezeigt (3.2). Danach wird die Laufzeit- und Speicherplatzkomplexität bestimmt (3.3), die Optimalität bewiesen (3.5) und Anwendungen beschrieben (3.7).

Das Kapitel 4 beschäftigt sich mit Modifikationen des Algorithmus: Es wird die Verwendung nicht-binärer Bäume (4.1) und das Einlesen des Textes in Blöcken von Zeichen statt in einzelnen Zeichen (4.2) gezeigt, die adaptive Huffman-Kodierung erklärt (4.3) und an einem Beispiel der Aufbau des dynamischen Baumes erläutert (4.4).

2 Grundlagen

2.1 Definitionen

Ein **Alphabet** $\Sigma = \{s_1, s_2, \ldots, s_k\}$ ist eine endliche, nicht-leere Menge. $s \in \Sigma$ heißt **Symbol** oder **Zeichen**.

Ein **Wort** $A = (a_1, a_2, \ldots, a_n)$ ist eine endliche Folge aus Σ^n. Die **Länge** $|A|$ des Wortes ist hier n.
Für ein Alphabet Σ ist $\Sigma^+ := \bigcup_{n \in \mathbb{N}} \Sigma^n$ die Menge aller Wörter über Σ.

Ein **Ereignis** ist in diesem Zusammenhang das Auftreten eines bestimmten Symbols innerhalb eines Wortes.

Seien Σ und Y Alphabete. Eine **Kodierung** g ist eine injektive Abbildung

$$g : \Sigma \to Y^+$$

$$s_j \mapsto (y_{j_1}, y_{j_2}, \ldots, y_{j_{l_j}}), \quad j = 1, 2, \ldots, |\Sigma|$$

Dabei ist Σ das **Quellalphabet** und Y das **Kodealphabet**. $l_j = |g(s_j)|$ ist die Länge des Kodewortes für das Zeichen s_j.
Die **Kodewortmenge** der Kodierung ist definiert als $g(\Sigma) = \{g(s_1), g(s_2), \ldots, g(s_k)\}$.

Eine Kodierung heißt **eindeutig dekodierbar**, wenn

$$G : \Sigma^+ \to Y^+$$

$$(a_1, a_2, \ldots, a_n) \mapsto (g(a_1), g(a_2), \ldots, g(a_n))$$

eine injektive Abbildung ist.

Seien $b = (b_1, b_2, \ldots, b_r)$ und $c = (c_1, c_2, \ldots, c_s) \in Y^+$ zwei Kodewörter. b heißt **Präfix** von c, wenn $d = (d_1, d_2, \ldots, d_{s-r}) \in Y^+$ existiert mit $c = (b_1, b_2, \ldots, b_r, d_1, d_2, \ldots, d_{s-r})$.
Eine Kodierung heißt **präfixfrei**, wenn kein Kodewort $s \in g(\Sigma)$ Präfix eines $s' \in g(\Sigma)$ ist.

Ein Beispiel für die Definitionen ist die Speicherung von Text auf einem Datenträger. Hier ist das Quellalphabet Σ die Menge der $ASCII$-Zeichen, das Kodealphabet Y die Menge $\{0, 1\}^8$ und die Kodierung g weist jedem $ASCII$-Zeichen eindeutig eine achtstellige Binärzahl zu. Die Textdatei ist ein Wort über Σ und das Auftreten eines bestimmten Buchstabens ist ein Ereignis.

Eine Kodierung auf eindeutige Dekodierbarkeit zu prüfen ist sehr schwer - diese Frage ist äquivalent zur „Post'schen Korrespondenz“, einem Problem, das schwerer als die NP-vollständigen Probleme ist. Es existieren aber Spezialfälle, die einfach zu prüfen sind, zu denen die präfixfreien Kodierungen gehören.

Lemma

Präfixfreie Kodierungen sind eindeutig dekodierbar.

Beweis Es sei $(g_1, g_2, \ldots, g_l) := (g(a_1), g(a_2), \ldots, g(a_n)) = (g(b_1), g(b_2), \ldots, g(b_m)) \in Y^+$ und die Kodierung $g : X \to Y^+$ injektiv. Zu zeigen: $(a_1, a_2, \ldots, a_n) = (b_1, b_2, \ldots, b_m) \in X^+$ $(\Rightarrow n = m \in \mathbb{N}))$.

Angenommen, es wäre $|g(a_1)| < |g(b_1)|$, also

$$g(a_1) = (g_1, g_2, \ldots, g_{|g(a_1)|})$$
$$g(b_1) = (g_1, g_2, \ldots, g_{|g(a_1)|}, \ldots, g_{|g(b_1)|})$$

Dann wäre $g(a_1)$ Präfix von $g(b_1)$, was ein Widerspruch zur vorausgesetzten Präfixfreiheit ist.

Analog ist $|g(a_1)| \not> |g(b_1)|$, also $g(a_1) = g(b_1)$ und wegen der Injektivität von g ist $a_1 = b_1$. Da also $a_1 = b_1$, ist auch $(g(a_2), g(a_3), \ldots, g(a_n)) = (g(b_2), g(b_3), \ldots, g(b_m))$ usw., also folgt

$$(a_1, a_2, \ldots, a_n) = (b_1, b_2, \ldots, b_m) \quad \text{und} \quad m = n$$

2.2 Darstellung einer Kodierung als Baum

Eine Kodierung g von Symbolen eines Alphabets Σ durch Symbole eines zweiten Alphabets Y mit $|Y| = d$ kann als Baum betrachtet werden, in dem jeder Knoten d Nachfolger hat. Dabei steht jede Kante zu einem Sohn eines Knotens für ein Symbol aus Y. Ein $s \in \Sigma$ wird in den Knoten geschrieben, der mit dem Weg erreicht wird, der durch $g(s) = (y_1, y_2, \ldots, y_{|g(s)|})$ beschrieben ist. Da g injektiv ist, kann in einem Knoten höchstens ein Symbol aus Σ stehen. Dabei werden alle Knoten, die kein $s \in \Sigma$ enthalten und deren Nachfolger ebenfalls kein $s \in \Sigma$ enthalten, aus dem Baum entfernt. Stehen alle $s \in \Sigma$ in Blättern des Baumes, so ist die Kodierung präfixfrei, da sonst ein Symbol s_i auf dem Weg zu einem $s_j \neq s_i$ liegen müsste. Da aber s_i und s_j Blätter sind, muss $s_i = s_j$ gelten, was ein Widerspruch ist.

Umgekehrt kann aus einem beliebigen Baum, in dessen Knoten die Symbole aus Σ stehen, eine Kodierung abgelesen werden. Dabei muss $|Y|$ mindestens so groß wie der Maximalgrad der Knoten des Baumes sein, da sonst ein Knoten existiert, der nicht mit Y kodiert werden kann. Jede Kante des Baumes wird mit einem Symbol aus Y bezeichnet, wobei die Kanten, die vom gleichen Knoten ausgehen, paarweise verschiedene Bezeichnungen zugewiesen bekommen müssen, da die Kodierung g sonst nicht injektiv wäre. $(y_1, y_2, \ldots, y_l)$ beschreibt dann den Weg von der Wurzel zu einem Knoten, in dem ein $s \in \Sigma$ gespeichert ist und es gilt $g(s) = (y_1, y_2, \ldots, y_l)$. Die entstehende Kodierung ist genau dann präfixfrei, wenn alle Symbole aus Σ in Blättern liegen.

2.3 Informationstheorie

Die Informationstheorie ist ein Teilgebiet der Mathematik, das sich unter anderem mit den Themen Information, Informationsübertragung, Datenkompression und Kodierung

beschäftigt. Dabei wird Kommunikation auf eine theoretische Weise betrachtet.

Claude Elwood Shannon lieferte 1948 in seiner fundamentalen Arbeit „A Mathematical Theory of Communication“ eine Definition, mit der er eigentlich nur eine Formel zur Berechnung der nötigen Bandbreite für Informationsübertragungen angeben wollte. Sie lieferte aber auch die lange gesuchte Beschreibung der Quantität von Informationen, die sogenannte **Eigeninformation**. Damit begründete er das Gebiet der Informationstheorie als eigenständige Wissenschaft:

Sei e ein Ereignis und $P(e)$ die Wahrscheinlichkeit für das unabhängige Eintreten des Ereignisses e, dann ist die Eigeninformation $i(e)$ des Ereignisses e definiert durch

$$i(e) := \log_d \left(\frac{1}{P(e)} \right) = -\log_d \left(P(e) \right)$$

Man beachte, dass

$$-\log_d(1) = 0 \quad \text{und} \quad \lim_{y \downarrow 0} -\log_d(y) = \infty$$

gilt. Das heißt, je seltener ein Ereignis auftritt, desto informationsreicher ist sein Auftreten. Ein Beispiel ist, dass das Bellen eines Hundes, der selten bellt, mehr Information enthält, als das Bellen eines Hundes, der häufig bellt.

Seien $\Sigma = \{s_1, s_2, \ldots, s_k\}$ ein endliches Alphabet, $A = (a_1, a_2, \ldots, a_n)$ ein Wort über Σ und p_i, $i = 1 \ldots k$ die relative Häufigkeit von s_i in A. $i(s_i)$ ist auch die Länge einer optimalen Kodierung von s_i (Beweis im folgenden Absatz) und $n \cdot p_i$ ist die absolute Häufigkeit des Auftretens von s_i. Daher ist $n \cdot p_i \cdot i(s_i)$ die Länge einer optimalen Kodierung für alle Auftreten von s_i. Wird nun die Länge der Kodierungen der Ereignisse für alle $s_i \in \Sigma$, $i = 1, \ldots, k$ summiert, ergibt sich die Länge m der optimalen Kodierung $(y_1, y_2, \ldots, y_m)$ von A über dem Kode-Alphabet Y mit $|Y| = d$.

$$m = \sum_{i=1}^{k} n \cdot p_i \cdot (-\log_d(p_i))$$

$$= -n \cdot \sum_{i=1}^{k} p_i \cdot \log_d(p_i)$$

Geteilt durch die Länge n des Wortes A ergibt sich die durchschnittliche Länge der Kodewörter

$$H(A) := -\sum_{i=1}^{k} p_i \cdot \log_d(p_i)$$

Diese wird als **Entropie** der Folge bezeichnet und beschreibt die erreichbare Kompressionsrate [5]. Hat die Folge eine Entropie von annähernd 1, sind die Ereignisse ungefähr gleich wahrscheinlich, werden daher als zufällig angesehen und können kaum komprimiert werden. Der russische Mathematiker Andrej Kolmogorow folgerte hieraus, dass ein Wort dann Information enthält, wenn man es komprimieren kann. Zu beachten ist, dass die Umkehrung nicht gilt: Ein unkomprimierbares Wort, zum Beispiel „OK“ , kann trotzdem Information enthalten. Hat die Folge eine sehr kleine Entropie, existieren in der Folge

Regelmäßigkeiten, auch Redundanzen genannt. Solche Folgen kann man sehr stark komprimieren.

Der Begriff Entropie stammt ursprünglich aus der Thermodynamik und beschreibt die wachsende Unordnung im Universum. Zum Beispiel zerspringt eine zu Boden fallende Tasse in Scherben, zu Boden fallende Scherben werden sich aber niemals zu einer Tasse zusammensetzen, dort würde die Unordnung sinken. Das Universum strebt gegen eine Entropie von 1. Dies ist im zweiten Hauptsatz der Thermodynamik formuliert.

Zum Beweis der Optimalität einer Kodierung des Wortes $A = (a_1, a_2, \ldots, a_n)$ mit der Länge $H(A) \cdot n$ werden die folgenden beiden Sätze benötigt.

Satz (a) McMillan (1959), (b) Kraft (1949)

a) Für alle eindeutig dekodierbaren Kodierungen über einem Kodierungsalphabet mit d Symbolen mit Kodewortlängen $(n_1, n_2, ..., n_k)$ gilt

$$\sum_{i=1}^{k} d^{-n_i} \leq 1$$

b) Gilt $\sum_{i=1}^{k} d^{-n_i} \leq 1$ für natürliche Zahlen $(n_1, n_2, ..., n_k)$, so existiert ein präfixfreier (also eindeutig dekodierbarer) Kode mit Kodewortlängen $(n_1, n_2, ..., n_k)$.

Beweise in [4].

Eine Interpretation des Satzes von McMillan ist, dass man in einem Baum mit d Nachfolgern pro Knoten in der i-ten Ebene nicht mehr als d^i Symbole setzen kann, wobei ein Symbol in der Ebene i auch durch bis zu d Symbole in der Ebene $i+1$ ersetzt werden kann, weil dann gilt, dass

$$\sum_{j=1}^{d} \frac{1}{d^{i+1}} = \frac{d}{d^{i+1}} = \frac{1}{d^i}$$

ist. Diese d Symbole tragen also zusammen nicht mehr zu dieser Summe bei, als ein Symbol in der Ebene i. Dies lässt sich rekursiv fortführen.

Der Satz von Kraft besagt, dass man unter Berücksichtigung der Regeln des Satzes von McMillan einen Kodebaum konstruieren kann, der eine präfixfreie Kodierung beschreibt.

Beispiel mit $d = 2$, $k = 6$, $\Sigma = \{x_1, x_2, \ldots, x_6\}$ mit den Kodewortlängen

$$n_1 = n_2 = 2, \quad n_3 = n_4 = n_5 = 3, \quad n_6 = 4$$

$$\sum_{i=1}^{6} 2^{-n_i} = \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{16} = \frac{15}{16} \leq 1$$

Daher existiert ein Binärbaum mit diesen Blatttiefen, der eine präfixfreie Kodierung beschreibt. Ein solcher Baum lässt sich konstruieren, indem die Tiefen sortiert werden, ein

vollständiger Binärbaum aufgebaut wird und dann der jeweils rechteste Knoten der Tiefe n_i mit dem i-ten Element belegt und seine Nachfolger gelöscht werden. Der für das Beispiel resultierende Kode-Baum ist in Abbildung 1 dargestellt.

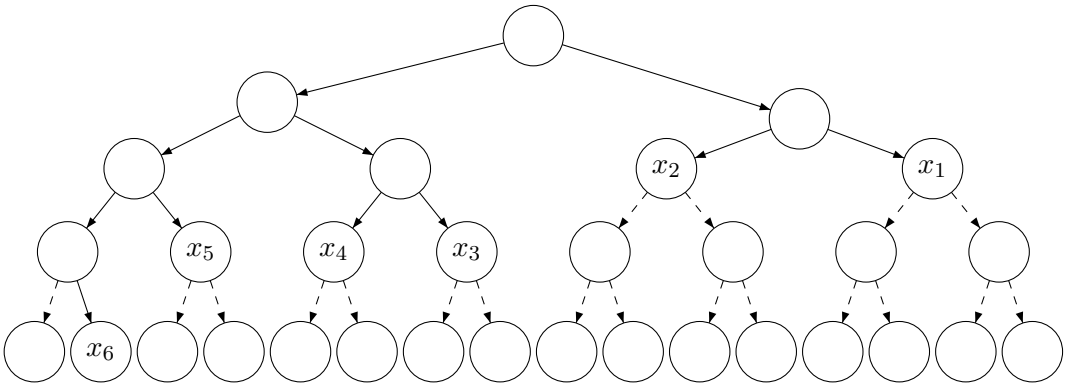

Abbildung 1: Beispiel zum Satz von Kraft

Satz von Shannon (1948): Noiseless Coding Theorem

Sei $A = (a_1, a_2, \ldots, a_n)$ ein Wort über $\Sigma = \{s_1, s_2, \ldots, s_k\}$. Die Länge einer optimalen Kodierung über einem Kodealphabet mit d Symbolen von A hat die durchschnittliche Kodewortlänge

$$H(A) \leq -\sum_{i=1}^{k} p_i \cdot \log_d(p_i) < H(A) + 1$$

Beweis

Sei dazu p_i, $i = 1 \ldots k$ die relative Häufigkeit des Symbols s_i in A und l_i für $i = 1 \ldots k$ die Länge des Kodewortes für s_i.

Hinrichtung: $H(A) \leq \underbrace{\sum_{i=1}^{k} l_i \cdot p_i}_{\text{Länge eines Kodes}} \quad \Leftrightarrow H(A) - \sum_{i=1}^{k} l_i \cdot p_i \leq 0$

$$H(A) - \sum_{i=1}^{k} l_i \cdot p_i = \left(\sum_{i=1}^{k} p_i \cdot \log_d\left(\frac{1}{p_i}\right)\right) - \left(\sum_{i=1}^{k} l_i \cdot p_i\right) \cdot \underbrace{\log_d d}_{=1}$$

$$= \sum_{i=1}^{k} p_i \cdot \left(\log_d\left(\frac{1}{p_i}\right) - l_i \cdot \log_d d\right)$$

$$= \sum_{i=1}^{k} p_i \cdot \log_d\left(\frac{d^{-l_i}}{p_i}\right)$$

$$= \frac{1}{\ln d} \cdot \sum_{i=1}^{k} p_i \cdot \ln\left(\frac{d^{-l_i}}{p_i}\right)$$

Bemerkung: $\ln(z) \leq z - 1 \quad$ für $z > 0$

$$\leq \frac{1}{\ln d} \cdot \sum_{i=1}^{k} p_i \cdot \left(\frac{d^{-l_i}}{p_i} - 1 \right)$$

$$= \frac{1}{\ln d} \cdot \left(\left(\sum_{i=1}^{k} d^{-l_i} \right) - \underbrace{\left(\sum_{i=1}^{k} p_i \right)}_{=1} \right)$$

$$= \frac{1}{\ln d} \left(\left(\sum_{i=1}^{k} d^{-l_i} \right) - 1 \right) \leq 0 \quad \Leftrightarrow \quad \sum_{i=1}^{k} d^{-l_i} \leq 1$$

Nach dem Satz von McMillan gilt diese Behauptung.

Rückrichtung: Zu zeigen: Es existiert ein präfixfreier Kode g Kodewortlängen l_i, so dass

$$H(A) + 1 > \underbrace{\sum_{i=1}^{k} p_i \cdot l_i}_{\text{Länge des Kodes g}}$$

Wähle dazu die l_i so, dass

$$-\log_d(p_i) \leq l_i < -\log_d(p_i) + 1$$

$$\Leftrightarrow d^{-l_i} \leq p_i < d^{-l_i+1}$$

$$\Rightarrow \sum_{i=1}^{k} d^{-l_i} \leq \sum_{i=1}^{k} p_i = 1$$

$\Rightarrow$ Satz von Kraft: Es existiert ein Kode g mit $|g(s_i)| = l_i$

Es bleibt zu zeigen: g erfüllt die Bedingung $H(A) + 1 > \sum_{i=1}^{k} p_i \cdot l_i$.
Nach Konstruktion gilt

$$p_i < d^{-l_i+1} \Leftrightarrow \log_d(p_i) < (-l_i + 1)$$

$$\Leftrightarrow \sum_{i=1}^{k} p_i \cdot \log_d(p_i) < \sum_{i=1}^{k} p_i \cdot (-l_i + 1)$$

$$\Leftrightarrow -H(A) < \sum_{i=1}^{k} p_i \cdot (-l_i + 1)$$

$$\Leftrightarrow H(A) > -\sum_{i=1}^{k} p_i \cdot (-l_i + 1)$$

$$\Leftrightarrow H(A) > -\left(-\left(\sum_{i=1}^{k} p_i \cdot l_i\right) + \underbrace{\sum_{i=1}^{k} p_i}_{1}\right)$$

$$\Leftrightarrow H(A) > \left(\sum_{i=1}^{k} p_i \cdot l_i\right) - 1$$

$$\Leftrightarrow H(A) + 1 > \underbrace{\sum_{i=1}^{k} p_i \cdot l_i}_{\text{Länge des Kodes g}}$$

2.4 Vorgänger der Huffman-Kodierung

Ein Ziel der Informationstheorie war es, einen Algorithmus zu finden, der mit Hilfe eines Kodebaumes eine im Sinne der Entropie optimale Kodierung konstruiert.

2.4.1 Morsebäume

Der Morse-Kode wurde bereits Mitte des 18. Jahrhunderts unter Berücksichtigung der Buchstabenhäufigkeiten bei der Anordnung der Buchstaben entwickelt. Der Morse-Kode wurde zum Telegraphieren benutzt und wird auch heute noch zur Kommunikation verwendet. Als Kodierung zur Kompression wird er aber nicht verwendet, da er nicht präfixfrei ist und deswegen drei Zustände zur Beschreibung benötigt werden: *lang*, *kurz* und *aus*. Dabei steht *kurz* für den linken Sohn, *lang* für den rechten Sohn und *aus* für die Pause zwischen *lang* und *kurz*. Außerdem muss mit *aus*, *aus*, *aus* das Pfadende im Baum angezeigt werden, da der Baum nicht präfixfrei ist. Zum Beispiel wird im deutschen Morsebaum (Abbildung 2) ein A als $kurz, aus, lang, aus, aus, aus$ gemorst.

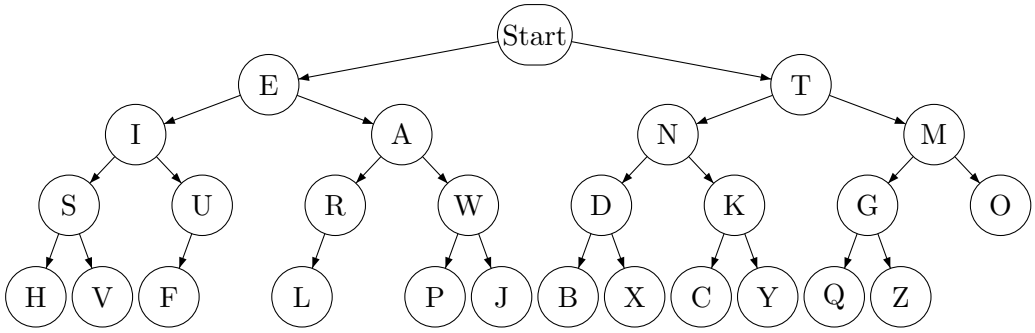

Abbildung 2: Der Morsebaum

Wird der Morse-Kode mit nur zwei Zuständen (0 und 1) realisiert, ergibt sich folgendes Problem:

- Wird $1 = kurz$ definiert, kann $0 = lang$ sein. Dann ist 01 der Weg im Baum zum Knoten A. Wird 01 übertragen, gibt es keine Möglichkeit, anzuzeigen, ob das nächste Bit zum nächsten Buchstaben gehört oder ein R beziehungsweise W übertragen werden soll. Das Problem ließe sich lösen, indem vor jedes Bit, dass weiter nach unten im Baum führen soll, eine 1 gesetzt wird und eine 0 das Ende eines Pfades anzeigt. Damit entstehen die Kodes $11 = kurz$, $10 = lang$ und $0 = Ende$, also genau drei Zustände.

- Wird $1 = an$ und $0 = aus$ definiert, müssen $kurz$ und $lang$ unterschieden werden, wofür wir 1 als $kurz$ und 11 als $lang$ setzen können. Das hat den Vorteil, dass hier $kurz$ mit einem Bit kodiert werden kann. Allerdings muss die 0 dahinter, die das Ende der Richtungsbeschreibung anzeigt, mitgezählt werden. Hier ist $110 = lang$, $10 = kurz$ und $0 = Ende$.
  Wird stattdessen $11 = lang$ definiert, weil bei der zweiten 1 klar ist, dass $lang$ gemeint ist, ist die Kodierung des Zustandes $lang$ ein Bit kürzer. Dies ist aber der gleiche Kode wie zuvor.

Da der Kode für A gleich dem Anfang des Kodes für R, W, L, P und J ist, ist es also notwendig, zusätzliche Informationen zu übertragen. Daher kann A wie oben beschreiben übertragen werden als

- 10110: weitergehen: links, weitergehen: rechts, Ende
- 101100: $(an, aus) \widehat{=}$ kurz, $(an, an, aus) \widehat{=}$ lang, $(aus) \widehat{=}$ Ende
- 1011111000: In der Praxis ist $lang$ ein fünfmal so langes Zeichen wie $kurz$ und $Ende$ wird durch zwei weitere Pausen angezeigt.

2.4.2 Shannon-Fano-Kodierung

Professor Shannon, der Begründer der Informationstheorie, und Professor Mario Fano, ein Kollege am Massachusetts Institute of Technology, versuchten gemeinsam, eine im Sinne der Entropie optimale Kodierung mit Hilfe eines präfixfreien Baumes zu erzeugen. Sie bauten diesen Baum „von oben nach unten“ auf, scheiterten aber mit diesem Prinzip, da die so entstandene Kodierung nicht immer optimal war. Erst Fanos Student David Huffman konnte einen Algorithmus entwickeln, der immer einen optimalen Baum erzeugt.

Funktionsweise der Shannon-Fano-Kodierung:

1. Für jedes im Text vorkommende Symbol wird ein Knoten gebildet und die absolute Häufigkeit des Symbols im Knoten gespeichert.
2. Die Knoten werden sortiert, als Vergleichsschlüssel dient die Häufigkeit.
3. Ein neuer Knoten wird gebildet. Die Menge der Knoten wird in in zwei Gruppen geteilt, so dass die Summe der Häufigkeiten in beiden Gruppen möglichst gleich groß ist. Eine Gruppe wird der linke, die andere der rechte Teilbaum des neuen Knotens. Dieses Verfahren wird mit den neu entstandenen Gruppen wiederholt, bis jede Gruppe nur noch einen Knoten enthält. Damit ist der Baum aufgebaut.

Der linke Sohn eines Knotens wird mit 0 bezeichnet, der rechte mit 1 und die Kodierung eines Symbols ist der Pfad zu dem Knoten, in dem das Symbol liegt. Nach Konstruktion des Baumes enthalten nur die Blätter Symbole. Die Kodierung ist daher präfixfrei.

2.4.3 Beispiel zur Shannon-Fano-Kodierung

Es seien fünf Symbole mit den folgenden Häufigkeiten gegeben: A: 15, B: 7, C: 6, D: 6, E: 5. Die Knoten sind bereits sortiert und liegen zu Beginn (Abbildung 3) alle in derselben Gruppe.

Abbildung 3: Beispiel zur Shannon-Fano-Kodierung 1

Diese Gruppe wird nun in zwei Gruppen geteilt, so dass die Summe der Häufigkeiten möglichst gleich groß ist. Die optimale Teilung ergibt die Gruppen (A, B) und (C, D, E). Die erste Gruppe wird der linke Teilbaum des neuen Knoten, die zweite der rechte Teilbaum (Abbildung 4).

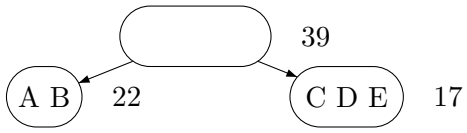

Abbildung 4: Beispiel zur Shannon-Fano-Kodierung 2

Jetzt werden die beiden Gruppen wieder geteilt. In der linken Gruppe werden die beiden Symbole A und B als Söhne an den neuen Vaterknoten angefügt. Die zweite Gruppe wird in einen linken Sohn C und eine Gruppe mit den Symbolen D und E geteilt und an den neuen Vaterknoten angehängt (Abbildung 5).

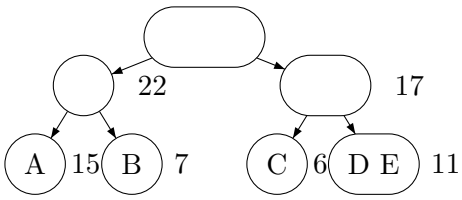

Abbildung 5: Beispiel zur Shannon-Fano-Kodierung 3

Im letzten Schritt wird die Gruppe mit den Symbolen D und E geteilt und wieder als Söhne an einen neuen Knoten angehängt (Abbildung 6).

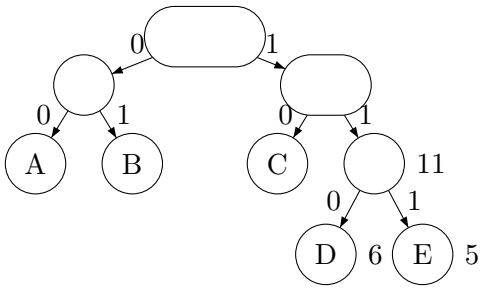

Abbildung 6: Beispiel zur Shannon-Fano-Kodierung 4

Da es nun keine Gruppen mehr gibt, ist der Baum fertig aufgebaut und aus Abbildung 6 können die Kodierungen für die Zeichen abgelesen werden: A$\widehat{=}$00, B$\widehat{=}$01, C$\widehat{=}$10, D$\widehat{=}$110, E$\widehat{=}$111.
Damit ergibt sich eine durchschnittliche Kodewortlänge von

$$\frac{(15+7+6)\cdot 2 \text{ Bit } + (6+5)\cdot 3 \text{ Bit}}{39} \approx 2,28 \text{ Bits}$$

Im Beispiel zur Huffman-Kodierung in Kapitel 3.2 wird mit der Huffman-Kodierung eine kleinere durchschnittliche Kodewortlänge für die hier verwendeten Häufigkeiten erreicht.

3 Huffman-Kodierung

1951 stellte Professor Robert Mario Fano während einer Elektrotechnik Vorlesung am MIT seine Studenten vor die Wahl, eine Abschlussklausur zu schreiben oder ersatzweise einen Algorithmus zu entwickeln, der den optimalen Kodebaum erzeugt. Unter den Studenten war auch David Huffman, der sich für die Entwicklung des Algorithmus entschied. Dies hätte er nach eigener Aussage nicht getan, wenn er gewusst hätte, dass Fano, sein Professor und Shannon, der Begründer der Informationstheorie, an dieser Aufgabe gescheitert waren. Nachdem Huffman mehrere Monate verschiedene Ansätze entwickelt hatte, aber nie deren Optimalität nachweisen konnte, wollte er anfangen, für die Klausur zu lernen. In dem Moment, in dem er seine Unterlagen wegwarf, kam ihm die Idee.

3.1 Funktionsweise

Sei $\Sigma = \{s_1, s_2, \ldots, s_k\}$ ein endliches Alphabet und $A := (a_1, a_2, \ldots, a_n)$ ein Wort über Σ und Y das Kodealphabet mit $|Y| = d$. Beispielsweise sei Σ die Menge aller $ASCII$-Zeichen, $Y = \{0, 1\}$ und A ein Text. Analog zur Kodierung mit Morsebäumen und der Shannon-Fano-Kodierung werden die Kodierungen als Bäume dargestellt. Ein Baumknoten besteht aus einem Symbol aus Σ, einem Integer für die absolute Häufigkeit seines Symbols und d Zeigern auf seine Nachfolger.

1. Ein Array der Länge k von Pointern auf Baumknoten, einen für jedes Symbol $s \in \Sigma$, wird mit 0 für die absolute Häufigkeit der Array-Elemente initialisiert.

2. Die absoluten Häufigkeiten für jedes $s \in \Sigma$ in A werden bestimmt und jeweils in den Knoten gespeichert.

3. Die Einträge des Arrays werden sortiert. Als Vergleichsschlüssel dient die Häufigkeit. Alle Pointer, deren Häufigkeit 0 ist, also solche, deren Symbol in der Datei nicht vorkommt, werden gelöscht, da diese Symbole nicht kodiert werden müssen. Die Anzahl der verbleibenden Knoten sei k^*.

4. Da der Baum von unten nach oben aufgebaut wird, muss sichergestellt werden, dass der Baum nur auf der untersten Ebene nicht vollständig ist:
   Ist $k^* \notin \{(d-1) \cdot \alpha + 1 \mid \alpha \in \mathbb{N}\}$, müssen die $k^* - ((d-1) \cdot \alpha + 1)$ Knoten mit den geringsten Häufigkeiten als Söhne eines neuen Knotens gesetzt werden und dieser Anhand der addierten Häufigkeiten seiner Söhne wieder im Array einsortiert werden (Notwendigkeit siehe Lemma 4, Kapitel 3.5).

5. Wieder wird ein neuer Baumknoten erstellt. Als seine d Nachfolger werden die Array-Elemente mit den d geringsten Häufigkeiten gewählt. Als Häufigkeit für ihn wird die Summe der Häufigkeiten seiner Söhne gesetzt. Nun werden seine Nachfolger aus dem Array entfernt und der neue Knoten in das Array einsortiert. Dieser Schritt wird $\left\lfloor \frac{k^*-1}{d-1} \right\rfloor$ mal wiederholt. Danach bleibt nur ein einziger Baum übrig, in dem alle $s \in \Sigma$ in den Blättern liegen. Dieser Baum heißt **Huffman-Baum**.

Die Nachfolger werden mit den Symbolen aus Y bezeichnet. Dann ist die Kodierung eines Zeichens, die die Huffman-Kodierung liefert, der Weg zum Blatt, in dem es liegt. Im Fall $Y = \{0, 1\}$ wird der rechte Nachfolger eines Knoten mit 1 bezeichnet, der linke mit 0.

3.2 Beispiel

Es werden dieselben Symbole und Häufigkeiten wie im Beispiel zur Shannon-Fano-Kodierung (siehe Kapitel 2.4.3) benutzt, die wiederum bereits sortiert und in das Array eingetragen sind: A: 15, B: 7, C: 6, D: 6, E: 5 und $d = 2$. (Abbildung 7)

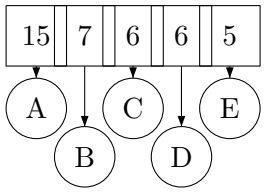

Abbildung 7: Beispiel zur Huffman-Kodierung 1

Im ersten Schritt werden die beiden Knoten mit den geringsten Häufigkeiten, D und E, mit einem neuen Vaterknoten zu einem Baum vereint, der anhand der addierten Häufigkeit 11 an zweiter Stelle in das Array einsortiert wird (Abbildung 8). Die Symbole liegen in den Blättern des Baumes.

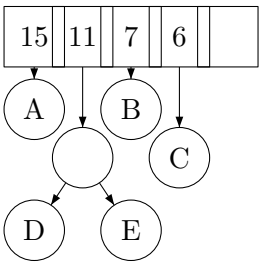

Abbildung 8: Beispiel zur Huffman-Kodierung 2

Im zweiten Schritt werden wieder die beiden Knoten mit den geringsten Häufigkeiten B und C zu einem neuen Baum mit der Häufigkeit 13 für die Wurzel zusammengefasst und an zweiter Position einsortiert. Der Baum, der im ersten Schritt entstanden ist, wird an die dritte Stelle sortiert, da seine addierte Häufigkeit kleiner als die des gerade entstandenen Baumes ist (Abbildung 9).

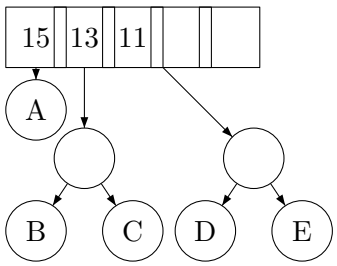

Abbildung 9: Beispiel zur Huffman-Kodierung 3

Im dritten Schritt sind die beiden Knoten mit den geringsten Häufigkeiten die Wurzeln der beiden Bäume. Diese werden zu einem neuen Baum zusammengefügt, der mit der addierten Häufigkeit 24 an die erste Position im Array gesetzt wird (Abbildung 10).

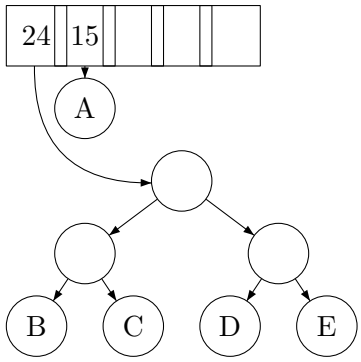

Abbildung 10: Beispiel zur Huffman-Kodierung 4

Im letzten Schritt wird das A mit dem bereits bestehenden Baum zum Huffman-Baum zusammengefasst (Abbildung 11).

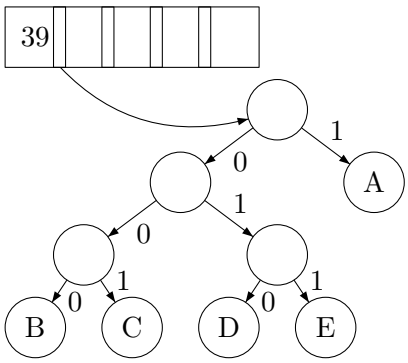

Abbildung 11: Beispiel zur Huffman-Kodierung 5

Aus diesem Baum können nun die im Sinne der Entropie optimalen Kodes abgelesen werden: A$\widehat{=}$1, B$\widehat{=}$000, C$\widehat{=}$001, D$\widehat{=}$010, E$\widehat{=}$011. Die durchschnittliche Kodewortlänge ist

$$\frac{15 \cdot 1 \text{ Bit } + (7+6+6+5) \cdot 3 \text{ Bit}}{39} \approx 2,23 \text{ Bits}$$

Dieser Wert ist kleiner als der der Shannon-Fano-Kodierung und entspricht der Entropie der Folge.

3.3 Komplexität

Es sei n die Anzahl der Zeichen im Text, der komprimiert werden soll, k die Größe des Quellalphabets und d die Größe des Kodierungsalphabets.

Lemma: Ein Huffman-Baum für ein Quellalphabet mit k Symbolen hat $\frac{k-1}{d-1}$ innere Knoten.

Beweis durch vollständige Induktion über k:

Induktionsanfang: $k = d$. Alle d Symbole werden im ersten Schritt als Söhne desselben Vaterknotens gesetzt.

$$\frac{k-1}{d-1} = \frac{d-1}{d-1} = 1$$

Induktionsannahme: Die Behauptung gelte für Quellalphabete mit $k \in \{(d-1) \cdot \alpha + 1 \mid \alpha \in \mathbb{N}\}$ Symbolen. Ist k nicht aus dieser Menge, so ergänze Σ um Symbole mit der Häufigkeit 0 zu Σ', so dass $|\Sigma'| \in \{(d-1) \cdot \alpha + 1 \mid \alpha \in \mathbb{N}\}$ ist. Diese Ergänzung ist zulässig, siehe Kapitel 3.5.
Induktionsschritt: $k \to k + d - 1$. Im ersten Schritt werden die d Symbole mit den geringsten Häufigkeiten entfernt und in einen Baum ω mit d Söhnen, die alle Blätter sind, einsortiert. Nach einem Schritt sind im Array also $(k + d - 1) - d + 1 = k$ Bäume, die nach Induktionsannahme durch $\frac{k-1}{d-1}$ innere Knoten verbunden werden. ω ist ebenfalls ein innerer Knoten, also hat ein Huffman-Baum für ein Quellalphabet mit $k+d-1$ Elementen $1 + \frac{k-1}{d-1} = \frac{(k+d-1)-1}{d-1}$ innere Knoten.

Induktionsschluss: Nach dem Prinzip der vollständigen Induktion hat ein Huffman-Baum für ein Quellalphabet mit $k \in \{(d-1) \cdot \alpha + 1 \mid \alpha \in \mathbb{N}\}$ Symbolen $\frac{k-1}{d-1}$ innere Knoten.

3.3.1 Laufzeitkomplexität

Das Einlesen der Datei zum Zählen der Häufigkeit der verschiedenen Zeichen ist in $O(n)$, denn jedes Zeichen muss einmal gelesen und im Array eingetragen werden. Das Sortieren des Arrays, welches k Einträge hat, hat mit Merge-Sort einen Aufwand von $k \cdot ld(k)$ Schritten. Der Huffman-Baum wird generiert, indem $\frac{k-1}{d-1}$ mal die seltensten d Elemente zusammengeführt werden, wodurch jeweils $d-1$ Elemente weniger vorhanden sind. Nach i Schritten sind dann noch $k - i \cdot (d-1)$ Elemente übrig. Im worst-case haben die seltensten d Symbole zusammen ein größeres Vorkommen als das erste Element, also den größtmöglichen Abstand zu ihrer neuen Position. Mit Insertition-Sort hat dann die Wiederherstellung der Sortierung eine Laufzeitkomplexität von $i \cdot (d-1) - 1$ Schritten, da das neue Element an jedem übrigen Element (bis auf sich selbst) vorbei geführt werden muss. Insgesamt ergeben sich

$$\sum_{i=1}^{\frac{k-1}{d-1}} k - (i \cdot (d-1) - 1)$$

$$= \sum_{i=1}^{\frac{k-1}{d-1}} k - (d-1) \cdot \sum_{i=1}^{\frac{k-1}{d-1}} i + \sum_{i=1}^{\frac{k-1}{d-1}} 1$$

$$= k \cdot \frac{k-1}{d-1} - (d-1) \cdot \frac{\frac{k-1}{d-1} \cdot \left(\frac{k-1}{d-1} - 1\right)}{2} + \frac{k-1}{d-1}$$

$$= \frac{k^2 + (2-d)k + d - 3}{2d-2}$$

Schritte. Also ist die Generierung des Huffman-Baumes aus $O\left(\frac{k^2+(2-d)k+d-3}{2d-2}\right)$.

Zum Auslesen der Kodierungen aus dem Huffman-Baum muss jeder der $k + \frac{k-1}{d-1}$ Knoten einmal besucht werden. An den k Blättern muss der Weg dorthin als Bitvektor ausgegeben werden. Dieser hat maximal die Länge k. Daher hat das Auslesen der Kodierung eine Laufzeit in $O\left(k^2 + k + \frac{k-1}{d-1}\right)$.

Die Größe der komprimierten Daten ist $n \cdot H(A)$ Bit, daher ist die Ausgabe in $O(n \cdot H(A))$, wobei die Datei noch ein weiteres Mal durchlaufen wird, also zusätzlich n Schritte benötigt werden. Die gesamte Kompression ist damit aus

$$O\left(\underbrace{n}_{\text{einlesen}} + \underbrace{k \cdot ld(k)}_{\text{sortieren}} + \underbrace{\frac{k^2 + (2-d) \cdot k + d - 3}{2 \cdot d - 2}}_{\text{Baum generieren}} + \underbrace{k^2 + k + \frac{k-1}{d-1}}_{\text{Kode auslesen}} + \underbrace{n \cdot H(A) + n}_{\text{ausgeben}}\right)$$

3.3.2 Speicherplatzkomplexität

Wie gezeigt werden $\frac{k-1}{d-1}$ innere Knoten benötigt und für jedes der k Zeichen einen Blattknoten, also hat die Huffman Kodierung eine Speicherplatzkomplexität aus

$$O\left(\frac{k-1}{d-1} + k\right)$$

3.4 Beweis der Präfixfreiheit

Da, wie in Kapitel 2.1 gezeigt, präfixfreie Kodierungen eindeutig dekodierbar sind, ist eine Huffman-Kodierung eindeutig dekodierbar, weil sie präfixfrei ist.

Beweis

Angenommen, die aus dem Huffman-Baum resultierende Kodierung ist nicht präfixfrei, das heißt, die Kodierung eines Symbols s_i ist der Anfang der Kodierung eines anderen Zeichens s_j $(i \neq j)$. Die Kodierungen sind aber Wege in dem Baum, also muss s_i auf dem Weg nach s_j liegen. Somit ist s_i kein Blatt.

Zu zeigen: Nach jedem Schritt der Konstruktion eines Huffman-Baumes gilt: Jedes Zeichen liegt in einem Blatt und in jedem Blatt liegt ein Zeichen.

Beweis durch vollständige Induktion über die Schritte der Baumgenerierung r:

Induktionsanfang: $r = 0$: Nach Konstruktion liegt zu Beginn ein Array von k Blättern vor. Ein einzelnes Blatt ist ein Baum der Höhe 1, in jedem Blatt liegt ein Zeichen und umgekehrt liegt jedes Zeichen in einem Blatt.

Induktionsannahme: $r \in \mathbb{N}_0$ beliebig, aber fest: Nach r Schritten gilt die Behauptung für alle Bäume im Array.

Induktionsschritt: $r \to r+1$: ω sei der neue Knoten, $a_{k-d+1} \dots a_k$ seien die Bäume mit geringster Häufigkeit. N sei der Baum, der aus ω und $a_{k-d+1} \dots a_k$ besteht (siehe Abbildung 12).

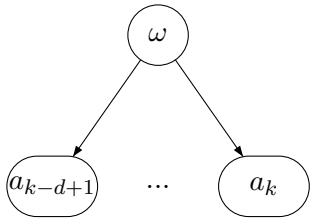

Abbildung 12: Grafik zum Beweis der Präfixfreiheit

Da kein Knoten an einen Knoten von $a_{k-d+1} \dots a_k$ angehängt wurde, sind die Blätter von $a_{k-d+1} \dots a_k$ auch Blätter in N, also liegt weiterhin jedes Zeichen in einem Blatt. Es ist nur ω als Knoten hinzugekommen, ω ist kein Blatt und enthält kein Zeichen, daher existiert in N kein Blatt, das kein Zeichen enthält.

Induktionsschluss: Nach dem Prinzip der vollständigen Induktion gilt die Behauptung für alle Huffman-Bäume.

3.5 Beweis der Optimalität

Eine Huffman-Kodierung ist immer eine optimale Kodierung, dass heißt, es existiert keine Kodierung mit einer kürzeren durchscnittlichen Kodewortlänge. Also muss eine Huffman-Kodierung optimal im Sinne der Entropie sein.

Für diesen Beweis werden einige weitere Lemmata benötigt:

Sei T ein Baum, X die Menge aller Blätter von T, die ein Symbol speichern, l_i die Pfadlänge zum Knoten $x_i \in X$ und p_i die Häufigkeit des Symbols, das in $x_i \in X$ gespeichert. ist Dann sei

$$B(T) := \sum_{x_i \in X} l_i \cdot p_i$$

$B(T)$ beschreibt die durchschnittliche Pfadlänge zu den Blättern in einem Kodebaum T.

Bemerkung 1

Ein Baum T ist genau dann Kodebaum eines optimalen Kodes für ein Wort A, wenn $B(T) = H(A)$ gilt.

Bemerkung 2

Aus $p_i \leq p_j$ folgt in einer optimalen Kodierung $l_i \geq l_j$. Daher können die Ereignisse $a_1 \dots a_k$ so nummeriert werden, dass gilt:

$$p_1 \geq p_2 \geq \dots \geq p_{k-1} \geq p_k$$

und

$$l_1 \leq l_2 \leq \dots \leq l_{k-1} \leq l_k$$

Lemma 1

Sind $a_{k-d+1} \dots a_k$ die Ereignisse mit der geringsten Häufigkeit, dann existiert ein optimaler Kodebaum T, in dem sie Nachfolger desselben Vorgängers, also Brüder, sind.

Beweis

Angenommen sie sind nicht Brüder, dann sei T_k ein Teilbaum mit mehr als einem Knoten, dessen Wurzel Sohn des Vaters von a_k ist. Dann haben die Ereignisse aus T_k eine längere Kodierung als a_k, was ein Widerspruch zur Bemerkung 2 ist. Daher müssen die Brüder von a_k Blätter sein. a_i sei ein Bruder von a_k mit $p_i \geq p_{k-d+1}$ und p_j nicht Bruder von a_k aber $p_{k-d+1} \geq p_j \geq p_k$, woraus $p_i \geq p_j$ folgt. T^* sei der Baum der durch Vertauschung von a_i und a_j entsteht.

Zu zeigen:

$$B(T^*) \leq B(T) \Leftrightarrow B(T^*) - B(T) \leq 0$$

$l_k - l_j$ Entspricht der Differenz der Länge der Kodierung von a_i und a_j, weil $l_k = l_i$, da a_i und a_k Brüder sind.
$(l_k - l_j) \cdot p_j$ ist die Verlängerung der Kodierungen von a_j bei der Vertauschung.
$(l_k - l_j) \cdot p_i$ ist die Verkürzung der Kodierungen von a_i bei der Vertauschung.

$$\Rightarrow B(T^*) - B(T) = (l_k - l_j) \cdot (p_j - p_i)$$

Daher ist zu zeigen

$$(l_k - l_j) \cdot (p_j - p_i) \leq 0$$
$$\Leftrightarrow l_k \cdot (p_j - p_i) - l_j \cdot (p_j - p_i) \leq 0$$
$$\Leftrightarrow l_k \cdot (p_j - p_i) \leq l_j \cdot (p_j - p_i)$$

1. Fall: $p_j < p_i \Rightarrow p_j - p_i < 0$

$$\Rightarrow l_k \geq l_j$$

Dies gilt nach Voraussetzung.

2. Fall: $p_j = p_i \Rightarrow p_j - p_i = 0$

$$\Rightarrow (l_k - l_j) \cdot (p_j - p_i) = (l_k - l_j) \cdot 0 = 0 \leq 0$$

Dies ist eine wahre Aussage und damit gilt die Behauptung. Das bedeutet, dass entweder T nicht optimal wäre oder (wenn $p_i = p_j$) eine Nummerierung der Ereignisse existiert, in der a_i und a_j vertauscht sind, so dass T ein Baum ist, der für diese Nummerierung der Behauptung entspricht.

Für einen optimalen Baum T wurde gezeigt, dass $a_{k-d+1} \dots a_k$ Brüder sind und in einem Huffman-Baum sind sie nach Konstruktion Brüder. T sei ein optimaler Baum für das Wort A über dem Alphabet Σ und die seltensten Ereignisse $a_{k-d+1} \dots a_k$ seien niedrigste Blätter und Brüder. T' bezeichne den Baum, der durch Löschung von $a_{k-d+1} \dots a_k$ entsteht. Der Vater dieser Knoten wird ein Blatt, dem ein Ereignis $\omega \notin \Sigma$ mit $P(\omega) := \sum_{i=k-d+1}^{k} p_i$ zugewiesen wird. T' ist ein Kodebaum für das modifizierte Wort A', das aus A entsteht indem alle $a_{k-d+1} \dots a_k$ durch ω ersetzt werden und A' ist ein Wort über dem modifizierten Alphabet $\Sigma' := \Sigma \setminus \{a_{k-d+1} \dots a_k\} \cup \{\omega\}$.
Berechne $B(T')$:

$$B(T) = \sum_{i=1}^{k} p_i \cdot l_i$$

$$B(T') = \sum_{i=1}^{k} p_i \cdot l_i - \sum_{i=k-d+1}^{k} p_i \cdot l_i + \sum_{i=k-d+1}^{k} p_i \cdot (l_i - 1)$$

$$= B(T) + \sum_{i=k-d+1}^{k} p_i \cdot (l_i - 1) - p_i \cdot l_i$$

$$= B(T) - \sum_{i=k-d+1}^{k} p_i$$

Lemma 2

Ist T ein optimaler Baum für ein Wort A über einem Alphabet Σ, so ist T' ein optimaler Baum für A' über Σ'.

Beweis

Angenommen T' wäre kein optimaler Baum für A', dann existiert nach Shannon ein optimaler Baum $T^{*'}$ mit

$$B(T^{*'}) < B(T') = B(T) - \sum_{i=k-d+1}^{k} p_i$$

Wird ω in $T^{*'}$ wieder durch $a_{k-d+1} \dots a_k$ ersetzt, so ergibt sich ein Baum T^* für den gilt:

$$B(T^*) = B(T^{*'}) + \sum_{i=k-d+1}^{k} p_i < B(T') + \sum_{i=k-d+1}^{k} p_i = B(T) + \sum_{i=k-d+1}^{k} p_i - \sum_{i=k-d+1}^{k} p_i$$

$$\Rightarrow B(T^*) < B(T)$$

Das heißt, T ist nicht optimal, was ein Widerspruch zur Voraussetzung ist.

Lemma 3

Ist T ein Huffman-Baum für ein Wort A über einem Alphabet Σ, so ist T' ein Huffman-Baum für A' über Σ'.

Beweis

T' ist ein Huffman-Baum, wenn gilt, dass aus

$$\sum_{i=k-d+1}^{k} p_i = P(\omega)$$

folgt, dass

$$i(a_{k-d+1}) > i(\omega) \geq i(a_{k-d+1}) - 1 \wedge i(a_k) - 1 \geq i(\omega) > i(a_k) - 2$$

also dass ω in T' in einer Ebene liegt, in der es in einem Huffman-Baum für A' liegen kann. Es gilt, dass

$$d \cdot p_k \leq \sum_{i=k-d+1}^{k} p_i = P(\omega) \leq d \cdot p_{k-d+1}$$

weil $p_k = min\{a_{k-d+1} \dots a_k\}$ und $p_{k-d+1} = max\{a_{k-d+1} \dots a_k\}$ ist.

$$\Leftrightarrow \log_d\left(\frac{1}{d \cdot p_k}\right) \geq \log_d\left(\frac{1}{P(\omega)}\right) \geq \log_d\left(\frac{1}{d \cdot p_{k-d+1}}\right)$$

$$\Leftrightarrow \log_d\left(\frac{1}{p_k}\right) - \log_d(d) \geq \log_d\left(\frac{1}{P(\omega)}\right) \geq \log_d\left(\frac{1}{p_{k-d+1}}\right) - \log_d(d)$$

$$\Leftrightarrow i(a_k) - 1 \geq i(\omega) \geq i(a_{k-d+1}) - 1$$

da aber $a_{k-d+1} \dots a_k$ Brüder sind, ist

$$i(a_{k-d+1}) + 1 > i(a_k)$$

$$\Rightarrow i(a_{k-d+1}) > i(a_k) - 1 \geq i(\omega) \geq i(a_{k-d+1}) - 1$$

so dass der erste Teil der Behauptung erfüllt ist.
Außerdem gilt mit $i(a_{k-d+1}) > i(a_k) - 1$, dass

$$i(a_k) - 1 \geq i(\omega) \geq i(a_{k-d+1}) - 1$$

$$\Rightarrow i(a_k) - 1 \geq i(\omega) \geq i(a_{k-d+1}) - 1 > i(a_k) - 2$$

$$\Rightarrow i(a_k) - 1 \geq i(\omega) > i(a_k) - 2$$

Also ist die Behauptung bewiesen.

Lemma 4

Hat ein Kodebaum T eine Höhe h und hat ein innerer Knoten k auf einer Höhe $l_k < h - 1$ weniger als d Söhne, so beschreibt T keinen optimalen Kode.

Beweis

Sei T^* der Baum, der entsteht, indem ein niedrigstes Blatt b (auf der Höhe h) als Nachfolger von k gesetzt wird, dann ist

$$B(T^*) = B(T) - P(b) \cdot h + P(b) \cdot (l_k + 1)$$

$$= B(T) + P(b) \cdot (l_k + 1 - h)$$

nach Voraussetzung gilt $l_k + 1 - h < 0$

$$\Rightarrow B(T^*) < B(T)$$

also ist T nicht optimal.

Beweis der Optimalität einer Huffmankodierung mit
$k = |\Sigma| \in \{(d-1) \cdot \alpha + 1 \mid \alpha \in \mathbb{N}\}$ durch vollständige Induktion über k:

Induktionsanfang: $k = d$: Trivial, da der Algorithmus einen Baum mit einem Vaterknoten und dessen $d = k$ Söhnen generiert. Es existieren für $d = k$ $d!$ äquivalente Kodebäume einer präfixfreien Kodierung, bei denen jeder innere Knoten d Nachfolger hat. Nach Shannon existiert aber mindestens ein optimaler, präfixfreier Kode (dessen innere Knoten wegen der Optimalität genau d Söhne haben müssen - zu beachten ist hierbei, dass obwohl die Wurzel Vorgänger der niedrigsten Blätter ist, sie d Nachfolger haben muss, weil sie sonst nicht mehr Vorgänger der niedrigsten Blätter wäre und dann Lemma 4 verletzt wäre). Daher muss dieses Ergebnis des Algorithmus optimal sein.

Induktionsannahme: Sei $k \in \{(d-1) \cdot \alpha + 1 \mid \alpha \in \mathbb{N}\} \geq 2$ beliebig aber fest. Für $|\Sigma| = k$ gilt die Behauptung.

Induktionsschritt: $k \to k + d - 1$: Σ habe $k + d - 1$ Elemente. Angenommen ein Huffman-Baum T für das Wort A über Σ ist nicht optimal. Nach Shannon existiert ein optimaler Baum T^* mit $B(T^*) < B(T)$.
In beiden Bäumen stimmt die Häufigkeit überein, daher folgt aus $B(T^*) < B(T)$ mit dem Zusammenfassen der seltensten Blätter (T', siehe Lemma 1), dass

$$B(T^{*'}) = B(T^*) - \sum_{i=k-d+1}^{k} p_i < B(T) - \sum_{i=k-d+1}^{k} p_i = B(T')$$

mit übereinstimmenden $\sum_{i=k-d+1}^{k} p_i$.

$$\Rightarrow B(T^{*'}) < B(T')$$

Da aber T' ein Huffman-Baum mit k Blättern ist, ist er nach Induktionsannahme optimal und damit B(T') minimal. Widerspruch, also ist die Annahme, dass T nicht optimal ist, falsch.

Induktionsschluss: Nach dem Prinzip der vollständigen Induktion beschreibt jeder Huffman-Baum einen präfixfreien, also eindeutig dekodierbaren Kode.

Beweis der Optimalität einer Huffmankodierung mit $k = |\Sigma| \in \mathbb{N} \geq 2$

Es sei A ein Wort über einem Alphabet Σ mit $k = |\Sigma| \in \mathbb{N} \backslash (d-1) \cdot \mathbb{N} + \{1\}$. Σ^* sei ein Alphabet, das aus Σ durch Ergänzung von Symbolen, die nicht in Σ vorkommen, entsteht, wobei $k^* = |\Sigma^*| \in (d-1) \cdot \mathbb{N} + \{1\}$ gelten soll. T^* sei der Huffman-Baum für A über Σ^*. T sei nun der Baum, der durch Löschung der ergänzten $k^* - k$ Symbole aus dem Baum T^* entsteht. Wie bewiesen ist T^* ein optimaler Kodebaum für A. Daher bleibt zu zeigen, dass $B(T) = B(T^*)$ ist:

$$B(T) = \sum_{i=1}^{k} p_i \cdot l_i$$

$$B(T^*) = \sum_{i=1}^{k^*} p_i \cdot l_i$$

$$= \sum_{i=1}^{k} p_i \cdot l_i + \sum_{i=k+1}^{k^*} \underbrace{p_i}_{=0} \cdot \underbrace{l_i}_{<\infty}$$

$$= \sum_{i=1}^{k} p_i \cdot l_i + 0 = B(T)$$

3.6 Entartete Huffman-Bäume

Eine Huffman Kodierung ist für gegebene Häufigkeiten immer optimal, der Baum kann aber entarten: Haben die Symbole a_i für $i = 0, 1, 2, \ldots, n$ die Häufigkeit $p_i = 2^i$, ergibt sich folgende Situation: Im ersten Schritt werden die letzten beiden Knoten mit den Häufigkeiten 1 und 2 zusammengefasst und wieder als letztes Element einsortiert. Nun wird dieser Baum als rechter Teilbaum mit dem Knoten mit der Häufigkeit 4 zusammengefügt und mit einer addierten Häufigkeit von 7 wieder als letzter Knoten in das Array eingefügt. Allgemein gilt: Nach dem j-ten Schritt hat der Baum die Tiefe $j-1$ und das Gewicht $\sum_{i=0}^{j-1} 2^i = 2^j - 1 < 2^j$, das heißt der neue Baum wird immer als letzter Knoten in das Array einsortiert. Nach n Schritten hat der Huffman-Baum die Tiefe n, beschreibt aber immer noch die beste Kodierung für die gegebenen Häufigkeiten.

Bei einer solchen Konstruktion hat das n-te Symbol eine relative Häufigkeit von $50\% = \frac{1}{2^1}$, das $(n-1)$-te Symbol eine eine Häufigkeit von $25\% = \frac{1}{2^2}$ und das i-te eine Häufigkeit von $\frac{1}{2^{n-i}}$ etc. In diesem Fall ist die Entropie

$$\sum_{i=0}^{n} \frac{1}{2^{n-i+1}} \cdot \log_2 \left(\frac{1}{\frac{1}{2^{n-i+1}}} \right)$$

$$= -2 \cdot \left(\frac{1}{2}\right)^{n+1} \cdot (n+1) - 2 \cdot \left(\frac{1}{2}\right)^{n+1} + 2$$

$$\lim_{n \to \infty} -2 \cdot \left(\frac{1}{2}\right)^{n+1} \cdot (n+1) - 2 \cdot \left(\frac{1}{2}\right)^{n+1} + 2 = 2$$

Das heißt, dass eine Datei mit m Bytes, die eine solche Häufigkeit haben, auf $2 \cdot m$ Bits, also mit dem Faktor $0,25$ komprimiert werden.

3.7 Anwendungen

Nahezu jeder heute benutzte Kompressionsalgorithmus verwendet die Huffman-Kodierung, um die Daten, die er - eventuell auch verlustbehaftet - produziert hat, nachzukomprimieren ohne dabei weitere Verluste zu bewirken. Bekannte Beispiele sind MP3, JPEG und MPEG4 (DivX, OpenDivX, XviD). Aber auch außerhalb der Anwendung zur Speicherung von komprimierten Daten findet die Huffman-Kodierung Anwendung, insbesondere bei der Datenübertragung. Zum Beispiel hat die NASA-Sonde Viking 1976 die ersten Bilder von der Marsoberfläche mittels Huffman-Kodierung komprimiert und übertragen [2]. Außerdem wird die Kodierung bei Video-Streams, Faxgeräten, digitalem Fernsehen und Show-View verwendet.

4 Modifikationen

4.1 Nicht-binäre Bäume

Werden statt binärer Bäume ternäre Bäume als Huffman-Bäume verwendet, muss informationstheoretisch ein Alphabet mit drei Symbolen zur Kodierung benutzt werden und die Entropie mit dem Logarithmus zur Basis 3 berechnet werden.

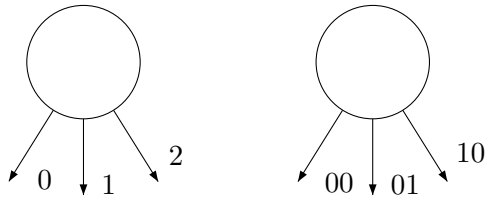

Abbildung 13: Tertiäre Bäume

In der Praxis ergibt sich dabei folgendes Problem: Um Wege mit mehreren verschiedenen Richtungen zu beschreiben, werden mehr als ein Bit gebraucht, im Fall von ternären Bäumen zwei Bits für die Richtungen 00, 01 und 10 (Abbildung 13). Allerdings kann auch noch 11 dargestellt werden, es wäre daher sinnvoller, quartiäre Bäume zu verwenden, wie in Abbildung 14 dargestellt. Allgemein gilt: für d Zweige werden $\lceil ld(d) \rceil$ Bits benötigt, womit $2^{\lceil ld(d) \rceil}$ Richtungen beschrieben werden können. Daher ist es sinnvoll, d als Zweierpotenz zu wählen, da dann wegen $2^{\lceil ld(d) \rceil} = d$ genauso viele Zweige d existieren, wie mit $\lceil ld(d) \rceil$ Bits beschrieben werden können, also keine möglichen Richtungen ungenutzt bleiben.

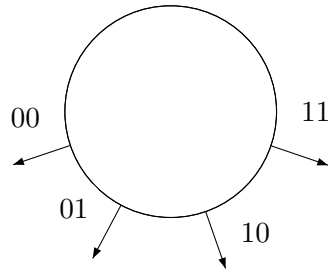

Abbildung 14: Quartiäre Bäume

Werden $2^{ld(d)}$ Zweige benutzt, ist es die effizienteste Methode, den Nachfolger anhand des Bitvektors der Länge $ld(d)$ auszuwählen, indem der Knoten einen vollständigen Binärbaum der Höhe $ld(d)$ enthält, der anhand des Bitvektors traversiert wird (Abbildung 15). Daraus ergeben sich aber keine Vorteile: Knoten eines binären Baumes werden in Binärbäume in den Knoten eines nicht-binären Baumes verlegt. Deshalb wird der nicht-binäre Baum durch Binärbäume dargestellt.

Letztendlich müssen die Symbole mit Bits kodiert werden, daher muss bei der Berechnung der Entropie zwangsläufig der Logarithmus dualis verwendet werden, was bewirkt, dass die optimale Kodierung mit binären Huffman-Bäumen erreicht wird.

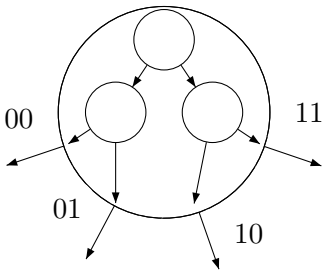

Abbildung 15: Quartiäre Bäume mit Pfadangabe

4.2 Größere Wortlänge

Wird nicht jedes Symbol einer Zeichenfolge der Länge n einzeln kodiert, sondern Symbolblöcke der festen Länge q aus Σ^q, besteht die Kodewortfolge nur noch aus $\left\lceil \frac{n}{q} \right\rceil$ Worten aus Σ^q statt n Worten aus Σ. Allerdings muss die Häufigkeit für alle möglichen Blöcke bestimmt werden: Ist die Zeichenfolge ein Wort aus dem Alphabet Σ mit $|\Sigma| = k$, so gibt es k^q verschiedene Blöcke. Diese Zahl ist aber durch die Länge der Zeichenfolge beschränkt: Es kann höchstens $\left\lfloor \frac{n}{q} \right\rfloor$ verschiedene Blöcke geben. Die Anzahl der Blätter des Huffman-Baumes ist also durch $\left\lfloor \frac{n}{q} \right\rfloor \leq k^q$ beschränkt. Im Vergleich dazu kann der Huffman-Baum bei Kodierung der einzelnen Zeichen höchstens k Blätter haben.

An dem eigentlichen Algorithmus sind keine Änderungen nötig: Nach dem die Häufigkeiten der in der Zeichenfolge vorkommenden Blöcke bestimmt worden sind, wird der Huffman-Baum generiert, der Text anhand der Kodierung, die durch den Baum gegeben ist, blockweise kodiert und mit dem Baum gespeichert.

Die Kodelänge wird bei jedem q kleiner, allerdings wird bei großem q die Größe des Baumes zum Problem: Es werden weniger Blöcke mehrfach in der Zeichenfolge vorkommen, somit müssen viele verschiedenen Blöcke als Bestandteil des Baumes gespeichert beziehungsweise übertragen werden. Damit werden die Einsparungen durch die Kodierung der Daten schnell zunichte gemacht. Im worst-case kommt kein Block mehrfach in der Zeichenfolge vor. Dann muss die gesamte Zeichenfolge als Inhalt der Blätter übermittelt werden und wird zusätzlich komprimiert übertragen. In diesem Fall wäre die kodierte Datenmenge größer als die unkodierte Zeichenfolge.

4.3 Adaptive Huffman-Kodierung

In der Praxis ist oftmals bei der Kodierung noch nicht die komplette zu kodierende Zeichenfolge bekannt, so zum Beispiel bei einer stetigen Datenübertragung, die sofort kodiert und übertragen werden soll. In einem solchen Fall kann der Huffman-Baum nicht generiert werden, da die Häufigkeiten erst bestimmt werden können, wenn die gesamte Zeichenfolge bekannt ist. Somit kann die Huffman-Kodierung, die für gegebene Häufigkeitsverteilungen eine optimalen Kodierung erzeugt, nicht angewendet werden.

Eine Möglichkeit, dieses Problem zu lösen, ist die Verwendung eines statischen Huffman-Baumes, der vor der Übertragung dem Kodierer und Dekodierer bekannt ist: Anhand von Statistiken oder Vorüberlegungen werden Häufigkeiten für die Symbole bestimmt und mit

diesen ein Huffman-Baum erzeugt. Mit dem Kode, der durch diesen, unabhängig von den tatsächlich zu kodieren Daten, erzeugten Baum bestimmt ist, wird dann die Zeichenfolge kodiert. Da der Baum mit Statistiken früherer Daten oder anhand theoretischer Überlegungen erstellt worden ist, ist er nicht optimal für die aktuelle Zeichenfolge.

Bei der zweiten Möglichkeit, der adaptiven Kodierung, wird ein dynamischen Huffman-Baum benutzt, der sowohl beim Kodierer als auch beim Dekodierer parallel erzeugt wird und deswegen nicht übertragen werden muss. Dieser wird bei jedem einzelnen Zeichen, das kodiert wird, den neuen Häufigkeiten angepasst, und bei Zeichen, die zum ersten Mal kodiert werden müssen, erweitert.

Da im Baum die Häufigkeiten geändert werden, muss nach jeder dieser Änderungen überprüft werden, ob die Ordnung des Baumes verletzt wurde. Damit wird die Optimalität der Kodierung sichergestellt. Zwei Regeln müssen für jeden Knoten im Baum erfüllt sein:

- Der rechte Sohn eines Knoten muss im Vergleich zum linken Sohn immer die höhere Häufigkeit haben. Ist diese Bedingung nicht erfüllt, müssen die beiden Söhne mit den Teilbäumen, deren Wurzel sie sind, getauscht werden.
- Ein Knoten muss eine geringere Häufigkeit als alle Knoten, die in höheren Ebenen des Baumes liegen, haben. Ist diese Bedingung nicht erfüllt, müssen die Knoten, die diese verletzen mit ihren Teilbäumen getauscht werden.

Bei der Kodierung starten Kodierer und Dekodierer mit einem Baum, der nur aus einer Wurzel besteht. Diese enthält den Eintrag „Not yet transmitted“(NYT) und hat die Häufigkeit 0. Soll nun ein Symbol kodiert werden, sucht der Kodierer in seinem Baum nach diesem Symbol. Dabei können zwei Fälle auftreten:

- Das Symbol ist bereits im Baum enthalten: Dann wird seine Häufigkeit, die in seinem Blatt gespeichert ist, und die Häufigkeiten der inneren Knoten einschließlich der Wurzel, die auf dem Pfad zu diesem Blatt liegen, um 1 erhöht. Ist nun die Ordnung des Huffman-Baumes verletzt, muss diese wiederhergestellt werden. Dazu müssen zwei Bedingungen erfüllt sein: Zum Schluss wird der Kode des Symbols an den Dekodierer übertragen. Dort wird mit Hilfe der Kodierung das Symbol dekodiert, ausgegeben und äquivalent zum Vorgehen im Baum des Kodierers der Baum des Dekodierer geändert, so dass beide immer identisch sind.
- Das Symbol ist noch nicht im Baum enthalten: Der Huffman-Baum wird um einen inneren Knoten erweitert, der an die Stelle des NYT-Knoten gesetzt wird. Der alte NYT-Knoten wird der linke Sohn des neuen Knotens. Das neue Symbol wird in einem zweiten neuen Knoten als rechter Sohn gespeichert und die Häufigkeiten auf dem Weg zu diesem erhöht. Falls die Ordnung des Baumes dabei verletzt wird, muss sie, wie oben bereits beschrieben, wiederhergestellt werden. Der NYT-Knoten rutscht bei jedem neuen Zeichen im Baum eine Ebene tiefer. Da aber bei einer Übertragung oder Kodierung wenige Zeichen in den Baum eingefügt werden, wird das Kodewort für den NYT-Knoten selten übertragen werden.
  Dann wird der alte Pfad zum NYT-Knoten oder eine vorher zwischen Kodierer und Dekodierer vereinbarte *Escape*-Sequenz übertragen, um anzuzeigen, dass ein neues Symbol in den Baum eingetragen werden muss. Dieses wird dann unkodiert

verschickt, beim Dekodierer ausgegeben und in den Baum eingefügt, der dann gegebenenfalls wieder so geändert werden muss, so dass beide Bäume wieder identisch sind.

Jetzt kann das nächste Zeichen kodiert werden.

4.4 Beispiel zur adaptiven Huffman-Kodierung

Die Zeichenfolge $HALLO$ soll übertragen werden: Vor der Übetragung des ersten Zeichen enthält der Kodebaum des Kodierers und des Dekodierers (Abbildung 16) jeweils nur den NYT-Knoten, der die Häufigkeit (in den Abbildungen rechts neben den Knoten) 0 hat.

Abbildung 16: Beispiel zur adaptiven Huffman-Kodierung 1

Das erste Zeichen H soll nun übertragen werden. Da der Kodebaum nur den NYT-Knoten enthält, findet der Kodierer das Zeichen H nicht. Also wird der Pfad zum NYT-Knoten gefolgt vom unkodierten H an den Dekodierer gesendet, der das H in seinen Baum einsortiert. Es wird als rechter Sohn einer neuen Wurzel mit der Häufigkeit 1 gesetzt, der NYT-Knoten wird als linker Sohn gesetzt (Abbildung 17). Der Kodierer speichert das H analog in seinem Baum.

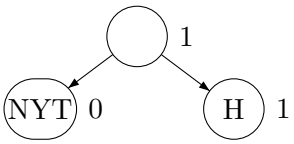

Abbildung 17: Beispiel zur adaptiven Huffman-Kodierung 2

Das nächste Zeichen der Folge ist ebenfalls noch nicht vorgekommen, deswegen wird der Pfad zum NYT-Knoten 0 gefolgt vom unkodierten A übertragen. Dann muss das Zeichen wieder in einem Knoten als rechter Sohn des alten NYT-Knoten gesetzt werden (Abbildung 18). Die Häufigkeiten der Knoten, die auf dem Pfad zu dem Blatt, das A repräsentiert, werden um 1 erhöht.

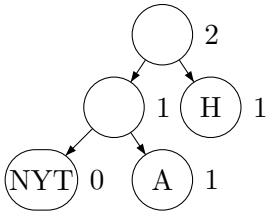

Abbildung 18: Beispiel zur adaptiven Huffman-Kodierung 3

Als drittes Zeichen wird das L übertragen, welches auch noch nicht im Baum gespeichert ist. Dazu wird wieder der Pfad zum NYT-Knoten 00 und das Zeichen zum Dekodierer übertragen und danach in beiden Bäumen die neuen Knoten gebildet. Nachdem ein Knoten für das L als Sohn des alten NYT-Knoten gesetzt worden ist, müssen die beiden Teilbäume der Wurzel vertauscht werden (Abbildung 19), da der linke Sohn die addierte Häufigkeit 2 hat, der rechte hat aber nur die Häufigkeit 1. Dies verletzt die Forderung, dass der rechte Sohn immer die größere Häufigkeit haben muss.

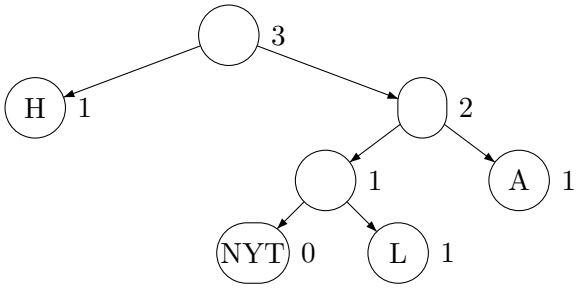

Abbildung 19: Beispiel zur adaptiven Huffman-Kodierung 4

Das zweite L der Folge ist das erste zu übertragende Zeichen, das bereits im Baum enthalten ist. Der Kodierer erhöht die Häufigkeiten auf dem Weg zum Blatt, welches das L repräsentiert, und übertragt die Bitfolge 101, die den Pfad beschreibt, an den Dekodierer. Dieser erhöht die Häufigkeiten auf diesem Pfad ebenfalls und gibt das Zeichen, das er in seinem Baum erreicht, aus. Damit hat das L, dessen Blatt in der dritten Ebene des Baumes ist, die Häufigkeit 2. Das H, dessen Blatt in der ersten Ebene liegt, hat aber nur die Häufigkeit 1. Dies verletzt aber die Bedingung, dass die Häufigkeit eines Zeichen mit einem Blatt in einer niedrigeren Ebene immer kleiner oder gleich der Häufigkeit aller Zeichen mit Blättern in den höheren Ebenen sein muss. Also müssen die Blätter, die das H beziehungsweise das L repräsentieren, getauscht werden (Abbildung 20).

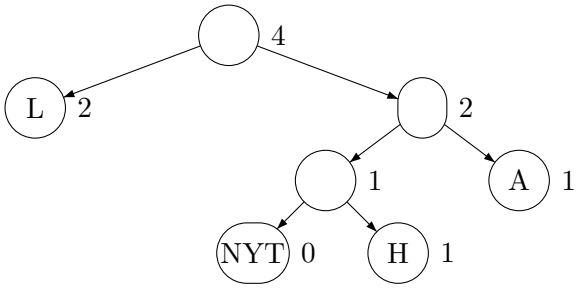

Abbildung 20: Beispiel zur adaptiven Huffman-Kodierung 5

Das letzte Zeichen ist noch nicht im Baum enthalten, deswegen wird wieder der Pfad zum NYT-Knoten 100 und das unkodierte O übertragen, die neuen Knoten gebildet und die Häufigkeiten erhöht. Danach ist es wieder nötig, das Blatt, welches das A repräsentiert, mit seinem Bruder zu tauschen, damit der rechte Sohn die höhere Häufigkeit hat (Abbildung 21).

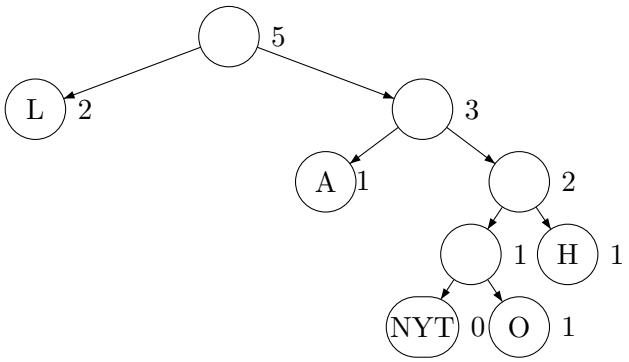

Abbildung 21: Beispiel zur adaptiven Huffman-Kodierung 6

4.5 Optimierter Morse-Kode

Auf Basis der prozentualen Verteilung der Buchstaben (siehe Tabelle 1, aus [1]) in deutschen Texten kann mit Hilfe der Huffman-Kodierung ein neuer, präfixfreier Morsebaum erzeugt werden (Abbildung 22).

Tabelle 1: Durchschnittliche Buchstabenhäufigkeit in deutschen Texten

A	0,0651	H	0,0476	O	0,0251	V	0,0067
B	0,0189	I	0,0755	P	0,0079	W	0,0189
C	0,0306	J	0,0027	Q	0,0002	X	0,0003
D	0,0508	K	0,0121	R	0,0700	Y	0,0004
E	0,1740	L	0,0344	S	0,0727	Z	0,0113
F	0,0166	M	0,0253	T	0,0615		
G	0,0301	N	0,0978	U	0,0435		

Die zweite Spalte der Tabelle 2 enthält den Morse-Kode des Zeichens der ersten Spalte, die dritte Spalte den mit Huffman-Kodierung optimierten Morse-Kode. Die Summen der letzten beiden Spalten der Tabelle beschreiben die benötigte Übertragungsdauer in Zeiteinheiten eines durchschnittlichen deutschen Textes mit 100 Zeichen. Der optimierte Morse-Kode benötigt $39,01\%$ weniger Zeiteinheiten im Vergleich zum Morse-Kode. Dieser Kode findet aber keine Anwendung, weil die Zeiteinheiten hierzu groß gewählt werden müssten, damit ein Mensch in der Lage wäre, ein fortlaufendes Signal in einzelne Bits einteilen zu können. Er müsste dazu die Zeiteinheiten genau abgrenzen können.

Tabelle 2: Vergleich der Morse-Kodes unter Berücksichtigung der Häufigkeit

	Morse-Kode	optimierter Kode	Häufigkeit in Prozent	Kodelänge·Häufigk. Morse-Kode	Kodelänge·Häufigk. Optimierter Kode
A	101100	0110	6,51	39,06	26,04
B	1101010100	010000	1,89	18,9	11,34
C	11010110100	01111	3,06	33,66	15,3
D	11010100	1010	5,08	40,64	20,32
E	100	000	17,40	52,2	52,2
F	1010110100	010011	1,66	16,6	9,96
G	110110100	10010	3,01	27,09	15,05
H	101010100	1110	4,76	42,84	19,04
I	10100	0010	7,55	37,75	30,2
J	101101101100	010010010	0,27	3,24	2,43
K	110101100	101110	1,21	10,89	7,26
L	1011010100	01110	3,44	34,4	17,2
M	1101100	10011	2,53	17,71	12,65
N	110100	110	9,78	58,68	29,34
O	1101101100	10110	2,51	25,1	12,55
P	10110110100	0100101	0,79	8,69	5,53
Q	11011010100	01001001101	0,02	0,22	0,22
R	10110100	0101	7,00	56	28
S	1010100	0011	7,27	50,89	29,08
T	1100	1000	6,15	24,6	24,6
U	10101100	1111	4,35	34,8	17,4
V	1010101100	01001000	0,67	6,7	5,36
W	101101100	010001	1,89	17,01	11,34
X	11010101100	01001001100	0,03	0,33	0,33
Y	110101101100	0100100111	0,04	0,48	0,4
Z	110110101100	101111	1,13	13,56	6,78
		Σ	100	672,04	409,92

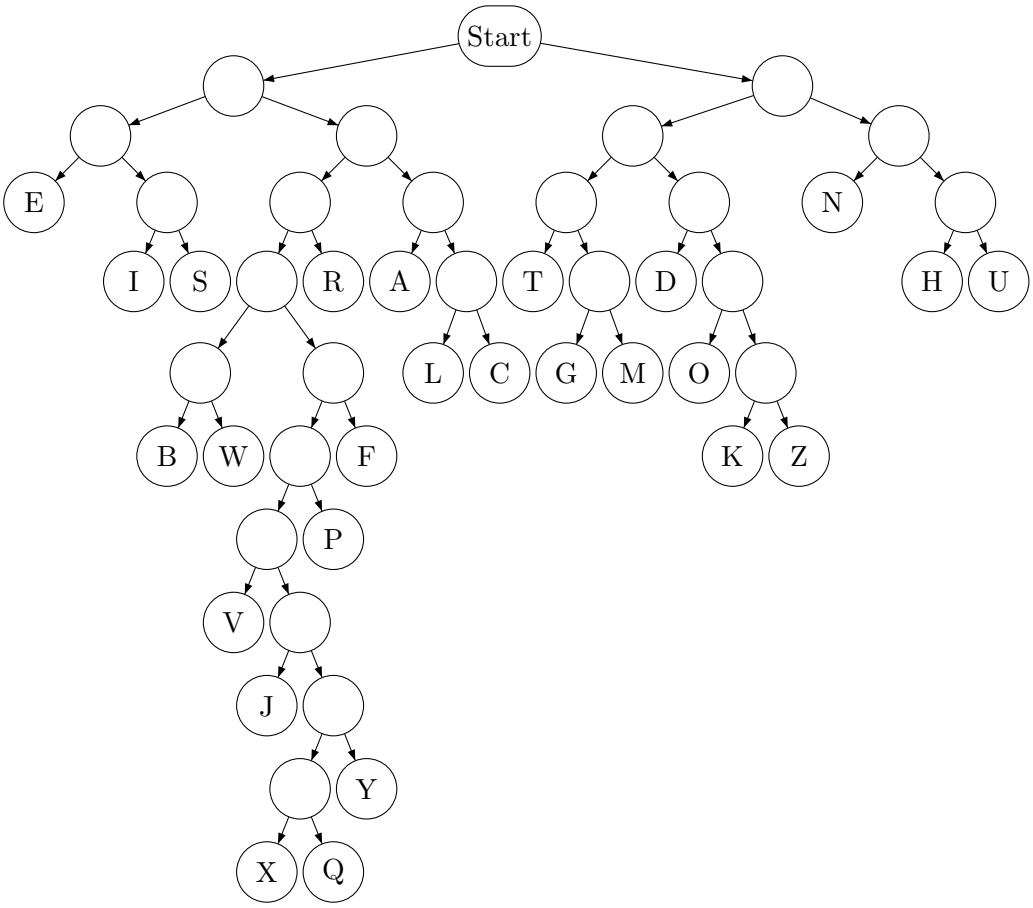

Abbildung 22: Der optimierte Morsebaum

5 Schlusswort

In diesem Seminar über die Huffman-Kodierung wurde die Funktionsweise des Algorithmus beschrieben: Die seltensten Knoten werden zu einem neuen Baum zusammengefasst und dieser wird wieder in die Menge der Knoten eingefügt. Sind alle Knoten in einem Baum zusammengefasst, ist der Huffman-Baum generiert, aus dem die Kodierungen ermittelt werden können. Mit den Grundlagen der Informationstheorie wurde dann bewiesen, dass sich aus dem Huffman-Baum immer die im Sinne der Entropie optimale, das heißt kürzeste, Kodierung ablesen lässt. Es wurde auch bewiesen, dass diese Kodierung immer präfixfrei ist. Zum Schluss wurden einige Modifikationen des Algorithmus besprochen und gezeigt, dass die Benutzung nicht-binärer Bäume nicht sinnvoll ist und die Kodierung mit einer größeren Wortlängen nur bei großen zu kodierenden Zeichenfolgen Sinn macht. Mit der adaptiven Huffman-Kodierung wurde eine Variante des Algorithmus gezeigt, mit der die Zeichenfolge bereits kodiert werden kann, bevor die gesamte Folge, und damit auch die eigentlich benötigten Häufigkeiten, bekannt ist. Damit können dann zum Beispiel Datenübertragungen komprimiert werden.

6 Anhang

6.1 Lebensgeschichte von David A. Huffman

David Albert Huffman wurde am 09. August 1925 in Chicago, Illinois geboren. Er entwickelte sich nur sehr langsam, beim Sprechen lernen war er anderen Kindern um zwei Jahre hinterher. Um ihm zu helfen, wurde seine Mutter Mathematiklehrerin an einer Schule für Problemkinder, um ihm dort einen Platz zu verschaffen. Aber Tests zeigten schnell, dass er gleichaltrigen Kindern eigentlich weit voraus war. Er übersprang mehrere Klassen und machte schon mit 18 Jahren einen Bachelor-Abschluss in Elektrotechnik an der Ohio State University. Danach diente er im Zweiten Weltkrieg bei der US-Navy auf einem Zerstörer, der Minen vor Japan und China räumte.

1949 machte er seinen Master-Abschluss, ebenfalls in Elektrotechnik, an der Ohio State University und 1953 am Michigan Institute of Technology (MIT) den Doktortitel. Seine Doktorarbeit war die erste formale Auseinandersetzung mit „switching circuits“, einem Bereich der Rechnerstrukturen, die ihm auch eine Fakultätsposition am MIT verschaffte, wo er eine Vorlesung über „switching circuits“ hielt.

1967 wurde er der erste Fachbereichsleiter des Fachbereichs Informatik an der University of California in Santa Cruz.

Anfang der siebziger Jahre fing er an, optische Täuschungen zu sammeln und veröffentlichte 1971 Methoden, 2-dimensionale Bilder in Y, V und T Formen zu zerlegen und beschrieb eine „Bilder-Grammatik“ mittels derer man die Möglichkeit ihrer 3-dimensionalen Existenz prüfen kann. Die Abhandlung darüber nannte er „Impossible Objects as Nonsense Sentences“. Diese Methoden finden bis heute Anwendung in der Bilderkennung und damit in der Robotik. Er befasste sich außerdem mit der Form von Radarwellen und Boolschen Algebren und lieferte auch wichtige Beiträge zur Theorie der Endlichen Automaten und zur Informationstheorie.

Für seine Arbeit bekam er viele Auszeichnungen

- 1999: Richard W. Hamming Medaille vom Institute of Electrical and Electronics Engineers (IEEE) für seine außergewöhnlichen Beiträge zur Informationstheorie
- Louis E. Levy Medaille vom Franklin Institute für seine Doktorarbeit über sequentielle „switching circuits“
- Alumnus Auszeichung der Ohio State University
- W. Wallace McDowell Award
- Urkunde Computer Pioneers, Auszeichnung von der IEEE Computer Gesellschaft
- Golden Jubilee Auszeichnung für technische Innovation von der IEEE Gesellschaft für Informationstheorie

Eines seiner Hobbys war die Beschreibung mathematischer Probleme als Papierfaltungen.

1994 trat er in den Ruhestand, hielt aber weiterhin Vorlesungen zu den Themen Informationstheorie und Signal-Analyse. Am 07. Oktober 1999 starb er im Alter von 74 Jahren

in Santa Cruz an einem Krebsleiden. Er wurde überlebt von seiner Frau Marilyn Huffman aus Santa Cruz, seiner ersten Frau Jane Ayres Huffman und ihren drei Kindern Elise, Linda und Stephen Huffman [3].

6.2 Quellcode-Ausschnitte

Der folgende C++ Quellcode ist ein Beispiel für die Erstellung des Huffman Baumes. Die StreamBit Routine ist wichtig für die Ausgabe des kodierten Textes in eine Datei, die hier nicht näher erläutert wird

```
class huffm {                       // Klasse fuer den Baum
   public:
      char zeichen;
      int haeufigkeit;
      huffm* zero;
      huffm* one;
      huffm(){                      // Konstruktor
         zero=NULL;
         one=NULL;
      }
};

huffm huffman[256];

// die folgende Prozedur vereint den i-ten und (i-1)ten Knoten
void Join(int i){
   huffm *temp = new(huffm);        // erstelle neuen Knoten
   temp->one = huffman[i];          // setze seine Nachfolger
   temp->zero = huffman[i-1];
                                    // setze seine Häufigkeit
   temp->haeufigkeit = huffman[i]->haeufigkeit + huffman[i-1]->haeufigkeit;
   huffman[i] = NULL;
   huffman[i-1] = temp;
   InsertionSortOnce(huffman, i-1); // einsortieren des neuen Knotens
}

// schreibt einzelne Bits (gepuffert) in eine Datei
void StreamBit(bool bit, FILE *stream, bool flush = false)
   {
   static int pufferchar = 0;          // Puffer
   static int exponent = 128;          // beschreibt die Position des nächsten zu
                                       //schreibenden Bits im Puffer

   if(flush)                           // falls der Puffer nur geleert werden soll
                                        //(wenn nach der Kodierung des letzten
      {                                //
      fputc(pufferchar, stream);       // gepuffertes Zeichen in die Datei ausgeben
      exponent = 128;                  // Position des nächsten zu schreibenden Bits
                                       // zurücksetzen
```

```
      pufferchar = 0;                  // Puffer löschen
      return;
      }

   pufferchar += exponent * bit;       // Bit in den Puffer schreiben
   exponent /= 2;                      // Position des nächsten zu schreibenden Bits
                                        // weitersetzen
   if(exponent == 0)                   // wenn der Puffer voll ist
      {
      fputc(pufferchar, stream);       // gepuffertes Zeichen in die Datei ausgeben
      exponent = 128;                  // Position des nächsten zu schreibenden Bits
                                       // zurücksetzen
      pufferchar = 0;                  // puffer löschen
      }
   }

void main() {
   for(int i = 0; i < 256; i++)
      huffman[i] = new(huffm);
   fill_array();                                  // Haeufigkeit ermitteln(z.B. Datei ein
   SelectionSort(huffman, 256);                   // Array sortieren
   int lastpos=256;
   while(huffman[--lastpos]->haeufigkeit == 0) // Knoten mit Haeufigkeit 0 löschen
      delete(huffman[lastpos]);

   while(lastpos > 0)                             // Huffmanbaum aufbauen
      Join(lastpos--);

   Output();                                      // Ergebnis ausgeben
}
```

Bei der Implementierung der Huffman-Kodierung ist zu beachten, dass in der Praxis Dateien nur byteweise geschrieben werden können. Ein Byte ist 8 Bit groß, aber die Anzahl der Bits mit der ein Text kodiert wird, muss kein Vielfaches von 8 sein, daher können bis zu 7 Bits zu viel in der komprimierten Datei stehen, die bei der Dekompression dazu führen würden, dass bis zu 7 Byte zu viel in die dekomprimierte Datei ausgegeben werden. Die Anzahl der überflüssigen Bits kann vor oder hinter dem kodierten Text gespeichert werden. Wird sie hinter dem kodierten Text gespeichert, müssen nach der Dekompression bereits dekomprimierte Bits wieder gelöscht werden. Hierzu einige mögliche Konzepte mit Vor- und Nachteilen:

- Dekompression in den RAM würde eine zu große Speicherkomplexität nach sich ziehen
- Kopieren (bis auf die letzten Bytes) der dekomprimierten Datei würde temporär den doppelten Festplattenspeicher beanspruchen, der evtl. nicht zur Verfügung steht
- Ein 7 Byte großer FIFO Puffer, aus dem die letzten Bytes nicht mehr auf die Festplatte geschrieben werden, würde den Arbeits- und Festplattenspeicher entlasten, wäre aber schwerer zu implementieren

Diese Ansätze haben einen gemeinsamen Nachteil: Aus der Information, wie viele Bits am Ende zu viel waren, muss geschlossen werden, wie viele Bytes überflüssig waren. Ein Konzept, das diesen Nachteil nicht hat, ist:

- Ein Puffer für die Eingabedaten, aus dem am Ende die letzten Bits nicht mehr dekomprimiert werden

Wenn die Anzahl der überflüssigen Bits bereits vor dem kodierten Text gespeichert wird, fällt der Puffer für die Eingabedaten weg. Zur Bestimmung der Anzahl der überflüssigen Bits bei Verwendung dieser Methode einige mögliche Konzepte:

- Die Kodierung zweimal durchzuführen würde einen inakzeptablen Laufzeitverlust mit sich bringen
- Berechnung mit Mitteln der Informationstheorie: $8-(H(A)\cdot n \bmod 8)$ Bits sind überflüssig, aber diese Berechnung kann Rundungsfehler oder Zahlbereichsüberschreitungen beinhalten
- Wenn während der Kompression an den Blättern die absolute Häufigkeit $P_a(s)$ des Symbols s gespeichert wird, ist die Anzahl der Bits, die im letzten Byte noch benötigt werden:

  $$\left(\sum_{s\in\Sigma}(|g(s)|\cdot P_a(s)) \bmod 8\right) \bmod 8$$

  Diese Methode ist leicht zu implementieren, hat einen minimalen Laufzeitverlust bei der Kompression (der zusätzliche Aufwand ist aus $O(2\cdot k)$) und ist numerisch stabil.

6.3 Effiziente Ausgabe des Huffman-Baumes

Ein intuitiver Algorithmus zur Speicherung des Baums ist eine Präfix-Ausgabe, bei der ein beliebiges Symbol c einen inneren Knoten anzeigt. Dann muss im Falle des Auftretens von c im Text ein weiteres Symbol hinzugefügt werden. In der Praxis wird hierfür „\“ gewählt. Also zeigt $\backslash c$ das Symbol an und c einen inneren Knoten. Wird nun aber ein innerer Knoten nach dem Symbol „\“ ausgegeben, wird dies beim Dekodieren als das Symbol c interpretiert. Wird das Symbol „\“ als „\\“ kodiert, ist diese Kodierung präfixfrei, also eindeutig dekodierbar; der Baum aus dem Beispiel zur Huffman-Kodierung (Abbildung 11) würde hier kodiert als

$$CCCB\backslash CCDEA$$

Es gibt $k-1$ innere Knoten und k Symbole, von denen zwei mit je 2 Byte kodiert werden können, der Baum benötigt hier also zwischen $2\cdot k-1$ und $2\cdot k+1$ Byte.

Ein weniger intuitiver Algorithmus ist die Verwendung der streambit Routine, die bei den Code Ausschnitten in Kapitel 6.2 vorkommt, um zuerst die Struktur des Baumes zu kodieren und anschließend die Blätter von links nach rechts ausgeben. Hier fallen auch die zusätzlichen Bytes zur Unterscheidung der Symbole von Strukturen weg, da diese getrennt behandelt werden. Bei diesem Verfahren werden $k-1$ innere Knoten zum Beispiel mit 0 kodiert, k Bits um Blätter anzuzeigen, gefolgt von k Byte für die Symbole, also

$$\left\lceil\frac{2\cdot k-1}{8}\right\rceil + k \text{ Byte.}$$

Hier würde der Beispielbaum kodiert als

$(00011011)_2(10000000)_2$ B C D E A, also □ € B C D E A

Literatur

[1] *de.wikipedia.org.*

[2] *www.bernd-leitenberger.de.*

[3] *www.huffmancoding.com.*

[4] Rudolf Mathar. *Informationstheorie.* Verlag der Augustinus Buchhandlung, 1993.

[5] Khalid Sayood. *Introduction to data compression.* Morgan Kaufmann, 2000.