Huffman Codierung

Vortrag von Martin Uhl

Wintersemester 1996/97

Inhalt

1 Überblick

Im allgemeinen wird vom Computer ein Code fester Länge benutzt z.B. ASCII. Wenn wir die Anzahl der Bits reduzieren wollen, die erforderlich ist eine Nachricht zu codieren, brauchen wir unterschiedlich lange Codewörter für verschiedene Zeichen. Wenn wir weniger Bits verwenden für Codewörter, die häufiger auftreten, verwenden wir im Durchschnitt weniger Bits pro Zeichen. Der Morsecode ist ein Beispiel für so einen Code: z.B. das Codewort für E ist „·“, während das Codewort für Z „--··“ ist.

Die durschschnittliche Länge ist allerdings nicht das einzig wichtige für einen „guten Code“.

1.1 Beispiele

$P(a_1)= 1/2$, $P(a_2)= 1/4$, $P(a_3)= P(a_4)= 1/8$ (a_i steht für ein codiertes Zeichen)
Entropie 1,75 Bit/Zeichen

Zeichen	Code 1	Code 2	Code 3	Code 4
a_1	0	0	0	0
a_2	0	1	10	01
a_3	1	00	110	011
a_4	10	11	111	0111
⌀ Länge	1,125	1,25	1,75	1,875

Formel für durchschnittliche Länge: $l=\sum_{i=1}^{4} P(a_i)n(a_i)$ ($n(a_i)$ ist die Anzahl der Bits für Zeichen a_i)

Code 1: Da a_1 und a_2 gleich sind, ist der Code mehrdeutig und somit ungeeignet

Code 2: Zwar hat jedes Zeichen ein eigenes Codewort, jedoch gibt es beim Decodieren Probleme z.B. 100 kann sowohl a_2 a_3 als auch a_2 a_1 a_1 sein
→ Wir fordern eindeutige Decodierbarkeit.

Code 3 und 4 sind eindeutig decodierbar.

Code 3 ist zudem Präfix Code, d.h. kein Codewort ist Präfix eines anderen.
→ Fano-Bedingung[1] ist erfüllt

---

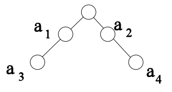
[1] Fano-Bedingung: kein Codewort darf Präfix eines anderen Codeworts sein. (Alle Huffman Codes erfüllen die Fano-Bedingung.)

Basierend auf den oben angegebenen Codes, kann man Codebäume erstellen:

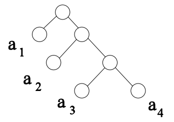
Kennzeichen für einen Präfix Code:
Nur an den Blättern im Codebaum sind Zeichen.

Ein Präfix Code ist eindeutig decodierbar, aber:
Ein eindeutig decodierbarer Code muß kein Präfix Code sein: (z. B.) Code 4.

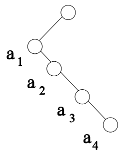
links 0 **rechts 1**



Code 2



Code 3



Code 4

Code 3 besitzt eine weitere Besonderheit: seine durchschnittliche Länge entspricht exakt der Entropie.
Solche Codes findet man allerdings nur, wenn die Wahrscheinlichkeit der Form 2^{-k} (1/2, 1/4...) entspricht. ($\bar{l} = H(S)$ für $P(a_i)=2^{-k}$ $k \in \mathbb{N}$)
Aber es gibt eine Technik, mit der ein Code erstellt werden kann, dessen durchschnittliche Länge maximal ein Bit von der Entropie abweicht.

Diese Technik nennt man die

2 Huffman Codierung.

Sie wurde von David Huffman entwickelt und trägt seinen Namen.
Huffman Codes sind Präfix Codes und das Optimum für ein gegebenes Modell (also gegebene Wahrscheinlichkeiten).

Die (binäre) Huffman Codierung basiert auf folgenden Beobachtungen hinsichtlich optimaler Codes:

1. In einem optimalen Code haben Zeichen, die öfters vorkommen (höhere Auftrittswahrscheinlichkeit) kürzere Codewörter, als Zeichen, die seltener auftreten.

2. In einem optimalen Code haben die 2 Zeichen, die am seltensten vorkommen, ein Codewort derselben Länge

Der erste Punkt ist klar: wenn die häufigsten Zeichen längere Codes hätten als seltenere, wäre der Code nicht so optimal wie bei einem, bei dem es andersherum ist. Folglich kann so ein Code nicht optimal sein.

Den zweiten Punkt kann man folgendermaßen beweisen:

Nehmen wir an, wir haben einen optimalen Code, in dem die beiden Codewörter für die beiden seltensten Zeichen unterschiedliche Länge haben.
Nehmen wir weiter an, das längere Codewort der beiden ist um k Bits länger als das kürzere.
Da der Code ein Präfix Code ist, kann das kürzere Codewort kein Präfix des längeren sein.

Also, wenn wir die letzten k Bits des längeren Codeworts wegfallen lassen würden, wären die beiden Codewörter immer noch unterschiedlich. Und weil diese beiden Codewörter für die beiden seltensten Zeichen stehen, besteht keine Gefahr darin, die letzten k Bits abzuschneiden. Dadurch, daß wir diese letzten k Bits abschneiden, erhalten wir einen neuen Code, der eine kleinere durchschnittliche Länge hat. Dies paßt allerdings nicht mit der Anfangsannahme zusammen, daß der erste Code optimal ist, somit ist bewiesen, daß der zweite Punkt zutrifft.

Für das Huffman Verfahren fügt man noch eine Voraussetzung zu diesen beiden Beobachtungen hinzu: Die Codewörter, die zu den beiden seltensten Zeichen gehören, unterscheiden sich nur im letzten Bit.
Diese Voraussetzung verträgt sich mit den beiden Beobachtungen und für zu einem einfachen Codierungsverfahren.

2.1 Konstruktion eines Huffman Codes

Wir haben eine Quelle mit Alphabet $A = \{a_1,a_2,a_3,a_4,a_5\}$
Die Auftrittswahrscheinlichkeiten sind:
$P(a_1) = P(a_3) = 0{,}2$ $P(a_2) = 0{,}4$ $P(a_4) = P(a_5) = 0{,}1$
Entropie = 2,122 Bits/Zeichen.

Wir sortieren das Alphabet nach absteigender Wahrscheinlichkeit:

Zeichen	Wahrsch.	Codewort
a_2	0,4	$c(a_2)$
a_1	0,2	$c(a_1)$
a_3	0,2	$c(a_3)$
a_4	0,1	$c(a_4)$
a_5	0,1	$c(a_5)$

a_4 und a_5 sind die haben die geringste Wahrscheinlichkeit,
also definieren wir ihre Codewörter wiefolgt:

$$c(a_4) = \alpha_1 * 0$$
$$c(a_5) = \alpha_1 * 1$$

Wobei α ein Binärstring ist und * für Konkatenation steht.

Als nächstes definieren wir ein neues Alphabet A‘ mit $A' = \{a_1,a_2,a_3,a'_4\}$,
wobei sich a'_4 aus a_4 und a_5 zusammensetzt, und die Wahrscheinlichkeit
$P(a'_4) = P(a_4) + P(a_5) = 0{,}2$ ist.
Wir sortieren dieses neue Alphabet wieder nach absteigender Wahrscheinlichkeit:

Zeichen	Wahrsch.	Codewort
a_2	0,4	$c(a_2)$
a_1	0,2	$c(a_1)$
a_3	0,2	$c(a_3)$
a'_4	0,2	α_1

In diesem Alphabet sind a_3 und a'_4 die zwei Zeichen am Ende der sortierten Liste, wir weisen ihnen folgende Codeworte zu:

$c(a_3) = \alpha_2 * 0$
$c(a'_4) = \alpha_2 * 1$

Aber da $c(a'_4) = \alpha_1 \Rightarrow \alpha_1 = \alpha_2 * 1$
können wir einsetzen:
$c(a_4) = \alpha_2 * 10$
$c(a_5) = \alpha_2 * 11$

Nun definieren wir wieder ein neues Alphabet A'' mit $A'' = \{a_1,a_2,a'_3\}$,
wobei sich a'_3 aus a_3 und a'_4 zusammensetzt, und die Wahrscheinlichkeit
$P(a'_3) = P(a_3) + P(a'_4) = 0{,}4$ ist.
Wir sortieren dieses neue Alphabet wieder nach absteigender Wahrscheinlichkeit:

Zeichen	Wahrsch.	Codewort
a_2	0,4	$c(a_2)$
a'_3	0,4	α_2
a_1	0,2	$c(a_1)$

Die unwahrscheinlichsten Zeichen hier sind a_1 und a'_3. Deshalb.

$c(a'_3) = \alpha_3 * 0$
$c(a_1) = \alpha_3 * 1$

Aber da $c(a'_3) = \alpha_2 \Rightarrow \alpha_2 = \alpha_3 * 0$
kann man wiederum einsetzen:
$c(a_3) = \alpha_3 * 00$
$c(a_4) = \alpha_3 * 010$
$c(a_5) = \alpha_3 * 011$
Ein weiteres Mal definieren wir ein neues, diesmal zweistelliges Alphabet A'''
$A''' = \{a''_3,a_2\}$,
wobei sich a''_3 aus a_1 und a'_3 zusammensetzt, und die Wahrscheinlichkeit

$P(a''_3) = P(a_1) + P(a'_3) = 0{,}6$ ist.
Wir sortieren dieses neue Alphabet wieder nach absteigender Wahrscheinlichkeit:

Zeichen	Wahrsch.	Codewort
a''_3	0,6	α_3
a_2	0,4	$c(a_2)$

Bei nur noch 2 Zeichen ist es jetzt klar:
$c(a''_3) = 0$
$c(a_2) = 1$

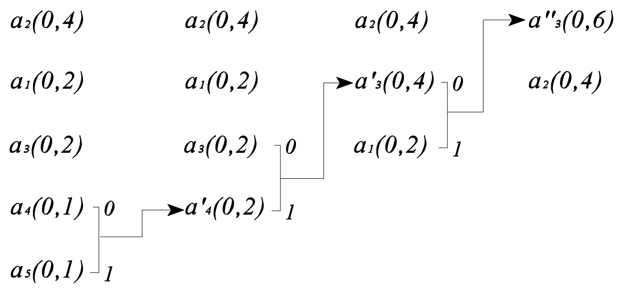
Was bedeutet, daß $\alpha_3 = 0$, welches wiederum bedeutet, daß:

$c(a_1) = 01$
$c(a_3) = 000$
$c(a_4) = 0010$
$c(a_5) = 0011$

Wir können nun den endgültigen Huffman Code aufschreiben:

Zeichen	Wahrsch.	Codewort
a_2	0,4	1
a_1	0,2	01
a_3	0,2	000
a_4	0,1	0010
a_5	0,1	0011

Schema:

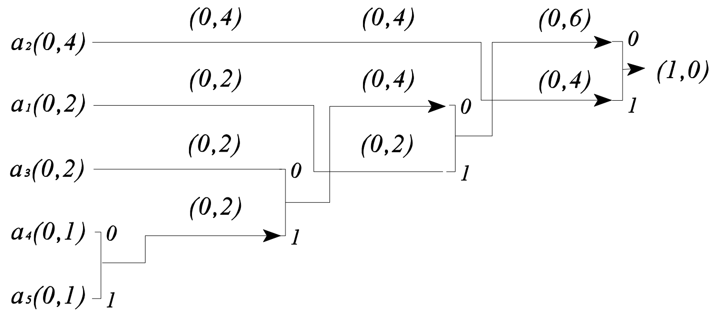


Die durchschnittliche Länge dieses Codes ist:

$\bar{l} = 0{,}4 \times 1 + 0{,}2 \times 2 + 0{,}2 \times 3 + 0{,}1 \times 4 + 0{,}1 \times 4 = 2{,}2$ Bits/Zeichen

Eine Meßart für die Effizienz dieses Codes ist die Redundanz, der Unterschied zwischen der Entropie und der durchschnittlichen Länge.
In diesem Fall ist die Redundanz 0.078 Bits/Zeichen.

Ein Binärbaum ist eine andere Möglichkeit den Huffman Code darzustellen: Er ähnelt sehr dem obigen Schema.



2.2 Huffman Codes mit minimaler Varianz

Wenn wir die Sortierung im vorhergehenden Beispiel etwas anders gemacht hätten, hätten wir einen anderen Huffman Code gefunden. Beim ersten neu-sortieren hätten wir das a'_4 höher plazieren können:

Zeichen	Wahrsch.	Codewort
a_2	0,4	$c(a_2)$
a'_4	0,2	α_1
a_1	0,2	$c(a_1)$
a_3	0,2	$c(a_3)$

Jetzt kombinieren wir a_1 und a_3 zu a'_1, welches die Wahrscheinlichkeit 0.4 hat, sortieren das Alphabet a_2 a'_4 und a'_1 und setzen das a'_1 so hoch wie möglich in die Liste:

Zeichen	Wahrsch.	Codewort
a'_1	0,4	α_2
a_2	0,4	$c(a_2)$
a'_4	0,2	α_1

Dann kombinieren wir a_2 und a'_4:

Zeichen	Wahrsch.	Codewort
a‘$_2$	0,6	α_3
a‘$_1$	0,4	α_2

Wenn wir das nun auflösen, erhalten wir folgenden Code:

Zeichen	Wahrsch.	Codewort
a_1	0,2	10
a_2	0,4	00
a_3	0,2	11
a_4	0,1	010
a_5	0,1	011

Die durchschnittliche Länge hiervon ist:

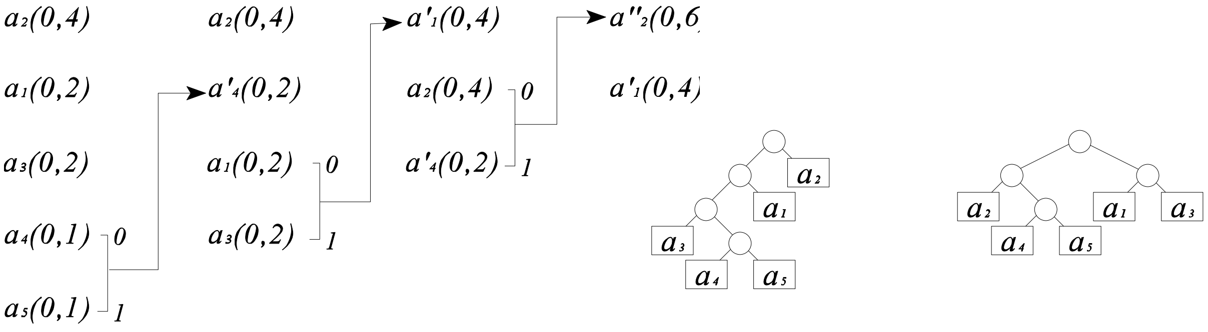
$\bar{l} = 0.4 \times 2 + 0,2 \times 2 + 0,2 \times 2 + 0,1 \times 3 + 0,1 \times 3 = 2,2$ Bits/Zeichen
Redundanz: 0.078 Bits/Zeichen.

Die zwei Codes sind identisch in bezug auf ihre Redundanz, jedoch die Varianz der Länge der Codewörter ist signifikant unterschiedlich.

2.2.1 Anwendungsbeispiel

Nehmen wir an, wir haben eine Datenquelle mit 10000 Zeichen/Sec, d.h. wir brauchen einen Datenkanal mit 22000 Bit/sec welcher einen konstanten Datenstrom verlangt. Damit es nicht zu Datenverlusten kommt. Um nun einen konstanten Datenstrom zu gewährleisten wird noch ein Puffer dazwischengeschaltet, der die Variationen ausgleichen soll. Dieser Puffer ist jedoch nur von beschränkter Größe, und je größer die Varianz der Codewörter ist, desto schwieriger ist der Entwurf eines solchen Puffers.
Nehmen wir an, die Datenquelle generiert für mehrere Sekunden lang a_4s und a_5s. Wenn wir den ersten Code benutzen, bedeutet das, daß wir Bits mit 40000 Bit/sec generieren. Pro Sekunde muß der Puffer also 18000 Bits speichern.
Andererseits, wenn wir den zweiten Code benutzen, generieren wir 30000 Bit/sec, und der Puffer muß nur 8000 Bit/sec speichern.
Beachten wir nun den anderen Fall: Wenn wir nur a_1s bekommen statt der a_4s und a_5s würde der erste Code 10000 Bit/sec produzieren. Da der Datenkanal aber weiterhin 22000 Bit/sec erwartet, muß der Puffer nun ein Defizit von 12000 Bit/sec kompensieren. Benutzen wir den zweiten Code muß der Puffer nur 2000 Bits/sec kompensieren.
Dies legt nahe, daß wir in diesem Fall den zweiten Code dem ersten vorziehen würden.

Um einen Huffman Code mit minimaler Varianz zu erzeugen, müssen wir nur das kombinierte Zeichen so hoch wie möglich in die Liste setzen.



Code mit minimaler Varianz

Vergleich der beiden Codes

2.3 Länge von Huffman Codes

Wir haben bisher gesagt, daß das Huffman Verfahren, einen optimalen Code produziert, aber wir haben noch nicht gesagt, welches die durchschnittliche Länge eines optimalen Codes ist. Die Länge hängt von einigen Dingen ab: dem Alphabet und den Wahrscheinlichkeiten der einzelnen Zeichen.
Wir werden jetzt zeigen, daß der optimale Code für eine Quelle S, also der Huffman Code für die Quelle S eine durchschnittliche Codelänge $\bar{l}$ hat, die nach unten von der Entropie und nach oben von der Entropie plus einem Bit abgegrenzt wird

$$H(S) \leq \bar{l} \leq H(S)+1$$

Um dies zu beweisen Brauchen wir eine Folgerung, bekannt als Kraft-McMillansche Ungleichung
Der erste Teil dieser Folgerung besagt (McMillan zufolge), daß wenn wir einen eindeutig decodierbaren Code C mit K Codeworten der Länge $\{l_1,l_2,\ldots,l_K\}$ haben, die folgende Ungleichung gilt:

$$\sum_{i=1}^{K} 2^{-l_i} \leq 1$$

Beispiel
Die Längen der Codeworte sind: {1,2,3,4,4}. Wenn wir dies in die linke Seite der Formel einsetzen, erhalten wir:

$$2^{-1}+2^{-2}+2^{-3}+2^{-4}+2^{-4}=1$$

Was die Kraft McMillan Ungleichung erfüllt.

Wenn wir den minimalen Varianzcode benutzen, dessen Längen {2,2,2,3,3} waren, ergibt sich folgendes:

$$2^{-2}+2^{-2}+2^{-2}+2^{-3}+2^{-3}=1$$

Was ebenfalls die Ungleichung erfüllt.

Der zweite Teil dieser Folgerung (Kraft zufolge) legt fest daß, wenn wir eine Reihe von positiven ganzen Zahlen $\{l_1,l_2,...,l_K\}$ haben, die die obige Ungleichung erfüllt, dann gibt es einen eindeutig decodierbaren Code, dessen Codewortlängen von dieser Reihe angegeben werden.

Durch diese Folgerung werden wir folgendes zeigen:

1. Die durschschnittliche Codewortlänge $\bar{l}$ eines optimalen Codes für eine Quelle S ist größer oder gleich als H(S)
2. Die durschschnittliche Codewortlänge $\bar{l}$ eines optimalen Codes für eine Quelle S ist kleiner H(S)+1

Für eine Quelle S, mit Alphabet $A = \{a_1, a_2, ..., a_K\}$ und dem Wahrscheinlichkeitsmodell $\{P(a1), P(a_2),...,P(a_K)\}$ ist die durchschnittliche Codewortlänge gegeben durch

$$\bar{l}=\sum_{i=1}^{K} P(a_i)l_i$$

deswegen können wir den Unterschied zwischen der Entropie der Quelle S und der der durchschnittlichen Länge folgendermaßen schreiben:

$$\begin{aligned}H(S) - \bar{l} &= -\sum_{i=1}^{K} P(a_i)\log_2 P(a_i) - \sum_{i=1}^{K} P(a_i)l_i\\ &=\sum_{i=1}^{K} P(a_i)\left(\log_2\left(\frac{1}{P(a_i)}\right)-\log_2(2^{l_i})\right)\\ &=\sum_{i=1}^{K} P(a_i)\log_2\left(\frac{2^{l_i}}{P(a_i)}\right)\\ &\le\log_2\left(\sum_{i=1}^{K} 2^{-l_i}\right)\end{aligned}$$

Die letzte Ungleichung bekommt man über die Jensen Ungleichung[2]. Da der Code ein optimaler Code ist, gilt:

$$H(S) - \bar{l} \le 0$$

Wir werden die obere Grenze dadurch zeigen, daß es einen eindeutig decodierbaren Code mit durchschnittlicher Codewortlänge H(S) + 1 gibt. Deshalb, wenn wir einen optimalen Code

---

[2] Jensens Ungleichung: Wenn f(x) eine konkave Funktion ist, dann ist $E[f(X)] \le f(E[X])$. Die log Funktion ist eine konkave Funktion (konkav ≡ die Steigung der Fkt wird immer kleiner)

haben, muß dieser Code eine durchschnittliche Länge haben, die weniger oder gleich wie H(S) + 1 ist.

Gegeben sind eine Quelle, ein Alphabet, und Wahrscheinlichkeitsmodell wie zuvor, wir definieren:

$$l_i = \left\lceil \log_2 \frac{1}{P(a_i)} \right\rceil$$

Wobei $\lceil x \rceil$ die kleinste ganze Zahl größer oder gleich x ist. Deswegen gilt:

$$\lceil x \rceil = x + \epsilon \qquad \text{für} \quad 0 \le \epsilon \le 1$$

und deswegen gilt:

$$\log_2 \frac{1}{P(a_i)} \le l_i \le \log_2 \frac{1}{P(a_i)} + 1 \qquad *$$

Von der linken Ungleichung können wir sehen, daß:

$$2^{-l_i} \le P(a_i)$$

Deswegen gilt:

$$\sum_{i=1}^{K} 2^{-l_i} \le \sum_{i=1}^{K} P(a_i) = 1$$

Wegen der Kraft McMillan Ungleichung existiert ein eindeutig decodierbarer Code mit den Codewortlängen $\{l_i\}$. Die durchschnittliche Länge dieses Codes kann nach oben abgegrenzt werden, indem man die rechte Ungleichung von * nimmt.

$$\bar{l} = \sum_{i=1}^{K} P(a_i) - l_i < \sum_{i=1}^{K} P(a_i) \left(\log_2 \frac{1}{P(a_i)} + 1 \right)$$

oder $\bar{l} < H(S) + 1$

Wir können an der Art an der die obere Grenze hergeleitet wurde, sehen, daß dies eine ziemlich lockere obere Grenze ist. Tatsächlich kann man zeigen, daß wenn p_{max} die größte Wahrscheinlichkeit im Wahrscheinlichkeitsmodell ist, dann für $p_{max} < 0{,}5$ die obere Grenze für den Huffman Code $H(S) + p_{max}$ ist, während für $p_{max} \ge 0{,}5$ die obere Grenze $H(S) + p_{max} + 0{,}086$. Offensichtlich ist dies eine engere Abgrenzung als die, die wir oben hergeleitet haben. Die Herleitung dieses Codes dauert allerdings einige Zeit.

2.4 Erweiterte Huffman Codes

Bei normalen Anwendungen, bei denen das Alphabet groß ist, ist p_{max} im allgemeinen ziemlich klein, und die Abweichung von der Entropie ist ebenfalls gering.

Jedoch, in Fällen, in denen das Alphabet klein ist, und die Auftrittswahrscheinlichkeit der verschiedenen Zeichen ungleichmäßig verteilt ist, kann der Wert von p_{max} ziemlich groß sein und der Huffman Code kann recht uneffizient, verglichen mit der Entropie, sein.

Beispiel:

Ein Code mit dem Alphabet $A = \{a_1, a_2, a_3\}$ und dem Wahrscheinlichkeitsmodell $P(a_1)=0{,}8$, $P(a_2)=0{,}02$, $P(a_3)=0{,}18$. Die Entropie für diese Quelle ist 0.816 Bits/Zeichen.

Huffman Code für diese Quelle:

Zeichen	Codewort
a_1	0
a_2	11
a_3	10

Die durchschnittliche Länge für diesen Code ist 1,2 Bits/Zeichen. Der Unterschied zwischen der durchschnittlichen Codelänge und der Entropie, die Redundanz, ist für diesen Code 0.384 Bits/Zeichen, was 47% der Entropie ist. Dies bedeutet: Um diese Sequenz zu codieren bräuchten wir 47% mehr Bits, wie minimal erforderlich wären.

Dies kann man manchmal dadurch verringern, daß mehrere Zeichen zusammengefaßt werden. Betrachten wir eine Quelle S, die eine Reihe von Zeichen aus dem Alphabet $A=\{a_1,a_2,\dots,a_m\}$ ausgibt. Jedes Element dieser Menge wird unabhängig von den anderen generiert. Die Entropie dieser Quelle ist gegeben durch:

$$H(S)=-\sum_{i=1}^{K} P(a_i)\log_2 P(a_i)$$

Wir wissen, daß wir einen Huffman Code für diese Quelle mit der durchschnittlichen Länge R so generieren könnten, daß folgendes gilt

$$H(S) \leq R \leq H(S) + 1 \qquad **$$

(Wir haben hier die lockerere Abschätzung verwendet, Man kann das gleiche auch mit der engeren Abschätzung machen)
Nehmen wir an, wir codieren die Reihe indem wir ein Codewort für alle n Symbole erstellen. Da es m^n Kombinationen von n Symbolen gibt, brauchen wir m^n Codewörter in unserem Huffman Code. Wir könnten diesen Code generieren, indem wir alle m^n Symbole als Zeichen eines erweiterten Alphabets sehen.

$$A^{(n)}=\{a_1a_1\dots a_1(\text{n-mal}),a_1a_1\dots a_2,\dots,a_1a_1\dots a_m.a_1a_1\dots a_2a_1,\dots,a_ma_m\dots a_m\}$$

Welches von einer erweiterten Quelle $S^{(n)}$ stammt. Bezeichnen wir die durchschnittliche Codewortlänge dieser Quelle als $R^{(n)}$. Dann wissen wir, daß

$$H(S^{(n)}) \leq R^{(n)} \leq H(S^{(n)}) + 1 \qquad ***$$

$R^{(n)}$ ist die Anzahl der erforderlichen Bits, die es braucht, um n Symbole zu codieren. Deswegen ist die Anzahl der Bits pro Symbol, R:

$$R = \frac{1}{n} R^{(n)}$$

Die Anzahl von Bits pro Symbol kann abgeschränkt werden als:

$$\frac{H(S^{(n)})}{n} \leq R < \frac{H(S^{(n)})}{n} + \frac{1}{n}$$

Um dies mit ** zu vergleichen und den Vorteil zu sehen, den wir bekommen, wenn wir Symbole in Blöcken anstatt einzeln codieren, sollten wir $H(S^{(n)})$ in $H(S)$ ausdrücken.

$$H(S^{(n)}) = -\sum_{i_1=1}^{m} \sum_{i_2=1}^{m} \cdots \sum_{i_n=1}^{m} P(a_{i_1}, a_{i_2}, \ldots, a_{i_n}) \; \log(P(a_{i_1}, a_{i_2}, \ldots, a_{i_n}))$$

$$= -\sum_{i_1=1}^{m} \sum_{i_2=1}^{m} \cdots \sum_{i_n=1}^{m} P(a_{i_1}), P(a_{i_2}), \ldots, P(a_{i_n}) \; \log(P(a_{i_1}), P(a_{i_2}), \ldots, P(a_{i_n}))$$

$$= -\sum_{i_1=1}^{m} \sum_{i_2=1}^{m} \cdots \sum_{i_n=1}^{m} P(a_{i_1}), P(a_{i_2}), \ldots, P(a_{i_n}) \; \sum_{j=1}^{n} \log(P(a_{i_j}))$$

$$= -\sum_{i_1=1}^{m} P(a_{i_1}) \log(P(a_{i_1})) \left\{ \sum_{i_2=1}^{m} \cdots \sum_{i_n=1}^{m} P(a_{i_2}), \ldots, P(a_{i_n}) \right\}$$

$$-\sum_{i_2=1}^{m} P(a_{i_2}) \log(P(a_{i_2})) \left\{ \sum_{i_1=1}^{m} \sum_{i_2=1}^{m} \cdots \sum_{i_n=1}^{m} P(a_{i_1}), P(a_{i_3}), \ldots, P(a_{i_n}) \right\}$$

$$\vdots$$

$$-\sum_{i_n=1}^{m} P(a_{i_n}) \log(P(a_{i_n})) \left\{ \sum_{i_1=1}^{m} \sum_{i_2=1}^{m} \cdots \sum_{i_{n-1}=1}^{m} P(a_{i_1}), P(a_{i_2}), \ldots, P(a_{i_{n-1}}) \right\}$$

Die n-1 Aufsummierungen in den Klammern summieren sich auf eine auf. Deshalb:

$$H(S^{(n)}) = -\sum_{i_1=1}^{m} P(a_{i_1}) \log(P(a_{i_1})) - \sum_{i_2=1}^{m} P(a_{i_2}) \log(P(a_{i_2})) \cdots - \sum_{i_n=1}^{m} P(a_{i_n}) \log(P(a_{i_n}))$$

$$= nH(S)$$

und wir können *** schreiben als:

$$H(S) \leq R \leq H(S) + \frac{1}{n}$$

wenn wir dies mit ** vergleichen, sehen wir, daß wir durch das codieren der Quelle in längeren Zeichenblöcken, eine durchschnittliche Codelänge erhalten, die näher an der Entropie liegt.

Beispiel:

Wir werden die vorherige Quelle hernehmen, und für eine Kombination von jeweils 2 Zeichen codieren.

Zeichen	Wahrsch.	Codewort
a_1a_1	0,64	0
a_1a_2	0,016	10101
a_1a_3	0,144	11
a_2a_1	0,016	101000
a_2a_2	0,0004	10100101
a_2a_3	0,0036	1010011
a_3a_1	0,1440	100
a_3a_2	0,0036	10100100
a_3a_3	0,0324	1011

Die durchschnittliche Codewortlänge für diesen erweiterten Code ist 1,7516 Bits/Zeichen, Jedoch bezieht sich jedes Zeichen im erweiterten Alphabet auf 2 Zeichen im Originalalphabet. Deswegen, bezogen auf das Originalalphabet ist die durchschnittliche Codewortlänge 0,8758 Bits/Zeichen. Die Redundanz ist ungefähr 0.06 Bits/Zeichen, was ein wenig mehr als 7% der Entropie ist.

2.5 Nichtbinäre Huffman Codes

Das binäre Huffman Codierungsverfahren kann auch zu einem nichtbinären erweitert werden, in welchen die Codeelemente aus einem m-ären Alphabet kommen, wobei m ungleich zwei ist. Erinnern wir uns daran, daß wir den Huffman Algorithmus aus folgenden Beobachtungen für einen optimalen binären Präfix Code bekommen haben:

1. In einem optimalen Code haben Zeichen, die öfters vorkommen (höhere Auftrittswahrscheinlichkeit) kürzere Codewörter, als Zeichen, die nicht so oft vorkommen.
2. In einem optimalen Code haben die 2 Zeichen, die am seltensten vorkommen, ein Codewort derselben Länge

Forderung: Die zwei Zeichen mit der niedrigsten Wahrscheinlichkeit unterscheiden sich nur in der letzten Position

Wir erreichen einen nichtbinären Huffman Code fast auf die gleiche Art. Das offensichtlichste wäre es, die zweite Beobachtung so abzuändern, daß es heißt:

2. In einem Optimalen Code haben die m Zeichen, die am seltensten vorkommen, ein Codewort derselben Länge

und die Forderung folgendermaßen abzuändern:

> Die m Zeichen mit der niedrigsten Wahrscheinlichkeit unterscheiden sich nur in der letzten Position

Es gibt jedoch ein kleines Problem bei dieser Methode. Betrachten wir einen Tri-ären Huffman Code für eine Quelle mit einem 6 Zeichen Alphabet. Wenn wir die obigen Regeln benutzen, kombinieren wir als erstes die drei Zeichen mit der niedrigsten Wahrscheinlichkeit zu einem kombinierten Zeichen. Dies würde ein reduziertes Alphabet mit 4 Zeichen ergeben. Wenn wir nun die drei Zeichen mit der geringsten Wahrscheinlichkeit in diesem Alphabet bekommen wir ein Reduziertes Alphabet mit nur zwei Zeichen. Wir haben drei Werte zum Zuweisen, aber nur zwei Zeichen.

Anstatt nun am Anfang 3 Zeichen zu kombinieren, hätten wir auch zwei Zeichen kombinieren können. Dies hätte ein reduziertes fünfstelliges Alphabet ergeben. Aus welchem dann durch Kombination von drei Zeichen zum Schluß ein dreistelliges Alphabet hervorgegangen wäre. Oder wir hätten im zweiten Schritt zwei Zeichen kombinieren können.
Welche Alternative sollen wir wählen?

Erinnern wir uns, daß die Symbole mit den niedrigsten Wahrscheinlichkeiten die längsten Codeworte ergeben. Weiterhin, daß die Symbole, die wir kombinieren Codeworte derselben Länge haben. Dies bedeutet, daß alle Zeichen, die wir am Anfang zu einem zusammengesetzten Symbol kombinieren, Codewörter derselben Länge haben, und daß diese Codewörter die längsten aller Codewörter sein werden.
Wenn wir irgendwo weniger als m Zeichen kombinieren, wäre das hier am Angebrachtesten.

Also, wieviele Zeichen sollen wir nun bei einem m-ären Code und einem M-Zeichen Alphabet im ersten Schritt kombinieren? Definieren wir m‘ als die Anzahl der Zeichen die im ersten Schritt kombiniert werden.
m‘ wird dadurch erreicht, daß man von M m-1 solange abzieht, bis eine Zahl kleiner gleich m herauskommt, dies ist dann m‘
m‘ ist immer zwischen m und zwei

Beispiel:

Wir generieren einen tri-ären Huffman Code für einen Quelle mit einem sechs-Zeichen Alphabet und einem Wahrscheinlichkeitsmodell $P(a_1)=P(a_3)=P(a_4)=0{,}2$, $P(a_5)=0{,}25$, $P(a_6)=0{,}1$ und $P(a_2)=0{,}05$ in diesem Fall ist $m = 3$, deswegen ist m‘ entweder 2 oder 3

$$6-2=4,\ \ 4-2=2 \quad \text{da } 2 \leq 3 \text{ folgt } m'=2$$

Wir sortieren die Zeichen nun der Wahrscheinlichkeit nach.

Zeichen	Wahrsch.	Codewort
a_5	0,25	$c(a_5)$
a_1	0,20	$c(a_1)$
a_3	0,20	$c(a_3)$
a_4	0,20	$c(a_4)$
a_6	0,10	$c(a_6)$
a_2	0,05	$c(a_2)$

da m‘=2 ist, ordnen wir die Codeworte mit den zwei niedrigsten Wahrscheinlichkeiten zu:

$c(a_6) = \alpha_1 * 0$
$c(a_2) = \alpha_1 * 1$

Wobei α_1 ein tri-ärer String ist, und * für Konkatenation steht.
Hier nun das reduzierte Alphabet:

Zeichen	Wahrsch.	Codewort
a_5	0,25	$c(a_5)$
a_1	0,20	$c(a_1)$
a_3	0,20	$c(a_3)$
a_4	0,20	$c(a_4)$
a'_6	0,15	α_1

Nun kombinieren wir die drei Zeichen mit der niedrigsten Wahrscheinlichkeit in ein zusammengesetztes Zeichen a'_2 und weisen die Codeworte zu:

$c(a_3) = \alpha_2 * 0$
$c(a_4) = \alpha_2 * 1$
$c(a'_6) = \alpha_2 * 2$

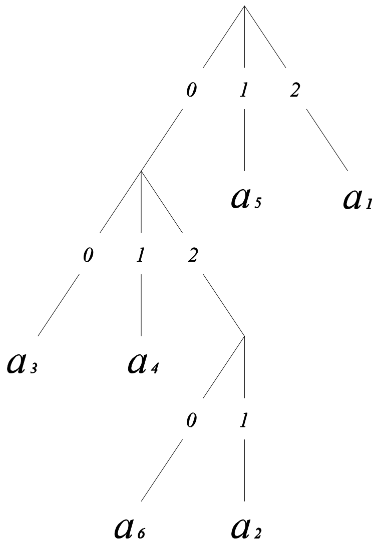
Aber $c(a'_6) = \alpha_1$, deshalb:
$\alpha_1 = \alpha_2 * 2$
Wir setzen ein:
$c(a_6) = \alpha_2 * 20$
$c(a_2) = \alpha_2 * 21$

Wiederum das reduzierte Alphabet:

Zeichen	Wahrsch.	Codewort
a'_3	0,45	α_2
a_5	0,25	c(a_5)
a_1	0,20	c(a_1)

Somit $\alpha_2 = 0$, c(a_5) = 1 und c(a_1)= 2 Wenn wir nun einsetzen erhalten wir die Codewortzuweisungen:

Zeichen	Wahrsch.	Codewort
a_5	0,25	2
a_1	0,20	021
a_3	0,20	00
a_4	0,20	01
a_6	0,10	1
a_2	0,05	020



2.6 Anpassender (adaptiver) Huffman Code

Die Huffman Codierung erfordert das Wissen über die Wahrscheinlichkeiten der Quellensequenz. Wenn dieses Wissen nicht verfügbar ist, wird die Huffman Codierung eine Prozedur mit zwei Durchgängen. Im ersten Durchgang werden die Statistiken gesammelt, und im zweiten Durchgang wird codiert.
Um diesen Algorithmus in eine Prozedur mit nur einem Durchgang zu konvertieren, haben Faller und Gallagher unabhängig voneinander Algorithmen entwickelt, die den Huffman Code anpassend entwickeln, basierend auf den Statistiken von bereits begegneten Zeichen. Diese wurden dann später von Knuth und Vitter verbessert.

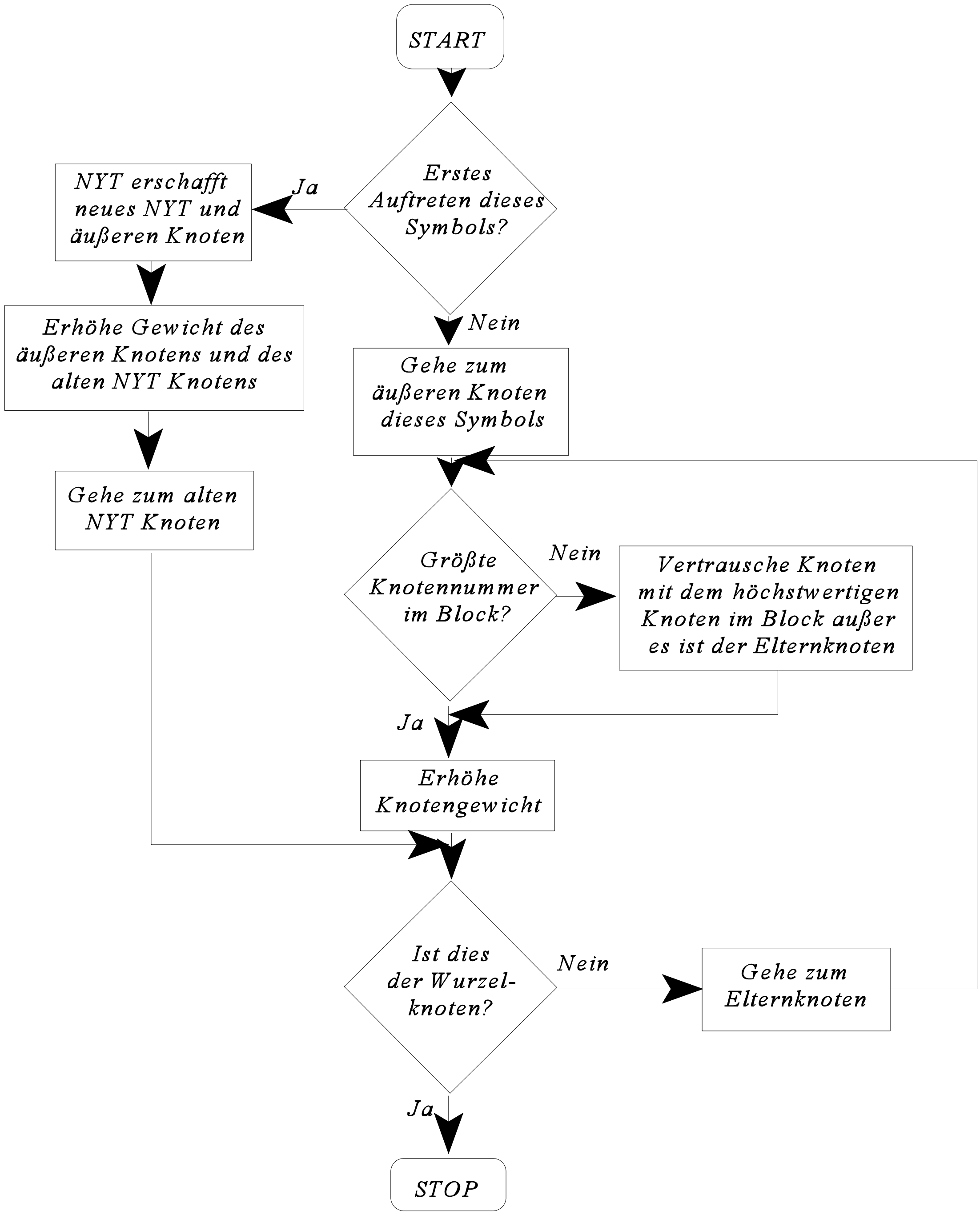
Wenn wir theoretisch das k+1te Zeichen codieren, und dabei die Statistiken der ersten k Zeichen verwenden, könnten wir den Code mit Hilfe des Huffman Verfahrens jedes Mal neuberechnen, wenn ein neues Zeichen übertragen wird. Durch die starke Rechenlast, die dieses Verfahren erzeugt ist es allerdings nicht sehr geeignet.
Deswegen gibt es das anpassende Huffman Verfahren.

2.6.1 Beschreibung

Das anpassende oder auch adaptive Huffman Verfahren basiert darauf, daß der Huffman Baum auf Basis der bereits übertragenen Zeichen und deren Auftrittszahl konstruiert wird. Jedes Mal, wenn ein neues Zeichen übertragen wird, wird es in den Huffman Baum einsortiert, und dieser wird wenn nötig umsortiert. Der Baum wird sozusagen on-the-fly konstruiert. Wie dies geschieht, wird im folgenden erklärt.

2.6.2 Regeln

Den Huffman Code kann man mit Hilfe eines Binärbaums beschreiben (s.o.)
Die Rechtecke sind äußere Knoten, oder auch Blätter genannt, und beziehen sich auf die Zeichen im Alphabet der Quelle. Das Codewort für ein Zeichen erhält man, wenn man den Baum von der Wurzel bis zum Blatt durchgeht, und 0 für einen linken Zweig, 1 für einen rechten Zweig einsetzt.
Um zu beschreiben, wie das anpassende Huffman Verfahren funktioniert, fügen wir zwei zusätzliche Eigenschaften zum Binärbaum hinzu:
Das Gewicht eines Knotens und eine Knotennummer.
Das Gewicht jedes äußeren Knotens ist einfach die Anzahl wie oft das Zeichen des entsprechenden Blattes auftrat. Das Gewicht jedes inneren Knotens ist die Summe der Gewichte seiner Nachkommen.
Die Knotennummer ist eine einmalige Nummer, die jedem inneren und äußeren Knoten zugeordnet wird. Wenn wir ein Alphabet der Größe n haben, dann ist die Anzahl der inneren und äußeren Knoten 2n-1, die numeriert werden als $y_1,...,y_{2n-1}$, so daß wenn x_j das Gewicht des Knotens y_j ist gilt: $x_1 \leq x_2 \leq ... \leq x_{2n-1}$. Weiterhin sind die Knoten y_{2j-1} und y_{2j} Nachkommen desselben Elternknoten, oder auch Siblings genannt, und die Knotennummer der Elternknoten ist größer als y_{2j-1} und y_{2j}. Die letzten zwei Charakteristika nennt man auch die Sibling Property, und jeder Baum, der diese erfüllt, ist ein Huffman Baum.
Im anpassenden Huffman Verfahren weiß weder der Übermittler, noch der Empfänger irgendetwas über die Statistik der Quellensequenz am Anfang der Übertragung. Der Baum sowohl auf der Seite des Übermittlers, als auch auf der Seite des Empfängers besteht aus einem einzigen Knoten, der allen noch nicht übertragenen Symbolen entspricht (NYT - not yet transmitted) und hat das Gewicht 0.
Wenn die Übertragung fortschreitet, werden den Symbolen entsprechende Knoten dem Baum hinzugefügt, und der Baum wird mit Hilfe einer Update Prozedur umkonfiguriert.
Vor der Übertragung wird ein fester Code für jedes Symbol zwischen Übermittler und Sender vereinbart. Ein einfacher (kurzer) Code ist wiefolgt:
Wenn die Quelle ein Alphabet $(a_1,a_2,....,a_m)$ der Größe m hat, suche e und r so, daß $m = 2^e+r$ und $0\leq r<2^e$. Das Zeichen a_k wird codiert als die (e+1)-bit binäre Darstellung von k-1, wenn $1 \leq k \leq 2r$ ansonsten wird a_k codiert als die e-bit binäre Darstellung von k-r-1. Zum Beispiel, nehmen wir an, m = 26, dann ist $e = 4$ und $r = 10$. Das Symbol a_1 wird codiert als 00000, das Symbol a_2 als 00001, und das Symbol a_{22} als 1011
Wenn ein Symbol zum ersten Mal auftritt, wird der Code für NYT übertragen, gefolgt von dem festen Code für das Symbol. Ein Knoten für das Symbol wird dann erschaffen, und das Symbol wird aus der NYT Liste gestrichen.
Sowohl Übermittler, als auch Empfänger beginnen mit demselben Baum. Beide verwenden dieselbe Updating Prozedur. Deswegen bleibt der Codierungs- und Decodierungsprozeß synchron.



Update Prozedur

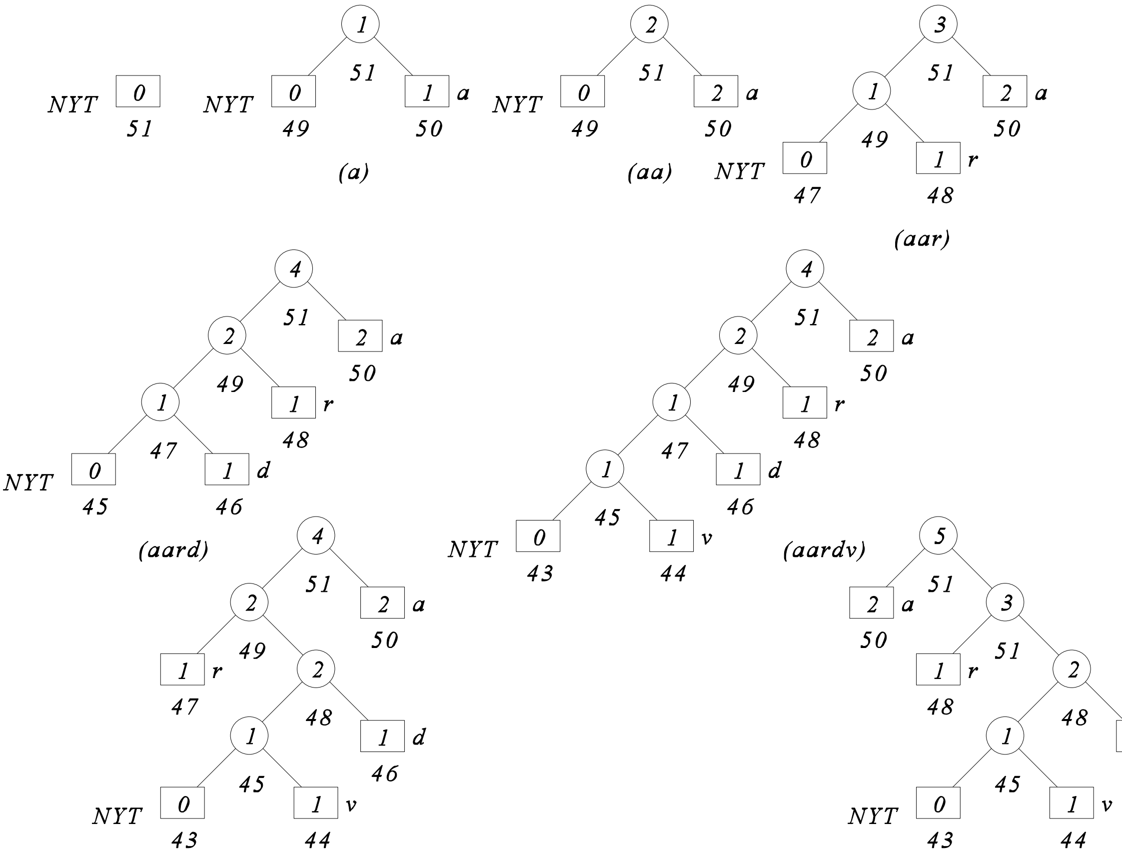
2.6.3 Update Prozedur

Die Update Prozedur erfordert, daß die Knoten in einer festen Ordnung numeriert werden. Die größte Knotennummer bekommt die Wurzel des Baums, wenn wir tiefer in den Baum von der Wurzel fortschreiten werden die Nummern kleiner, die kleinste Nummer wird dem NYT Code zugeteilt.

Die Knoten mit demselben Gewicht werden als *Block* bezeichnet.

Die Aufgabe der Update Prozedur ist es, die Sibling Property aufrechtzuerhalten während sie den Baum verändert, um die neuesten Schätzungen der Auftrittswahrscheinlichkeiten der Zeichen einzubringen. Damit die Update Prozedur beim Übermittler und beim Empfänger mit den gleichen Informationen arbeitet, wird der Baum beim Übermittler aktualisiert, sobald das Symbol codiert wird, und der Baum beim Empfänger, sobald das Symbol dekodiert wird.
Die Prozedur funktioniert folgendermaßen:

Beispiel

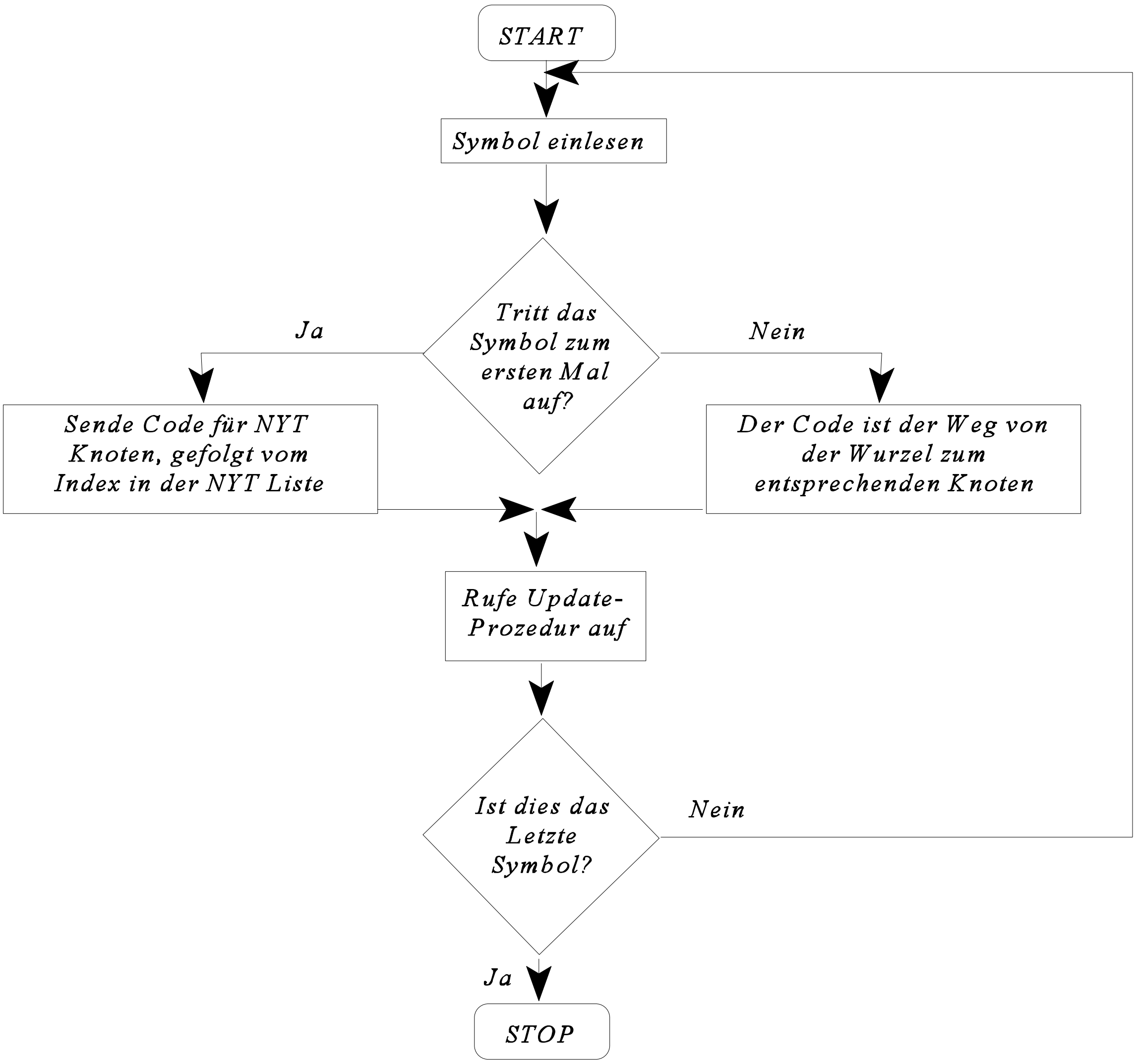


2.6.4 Codierung:

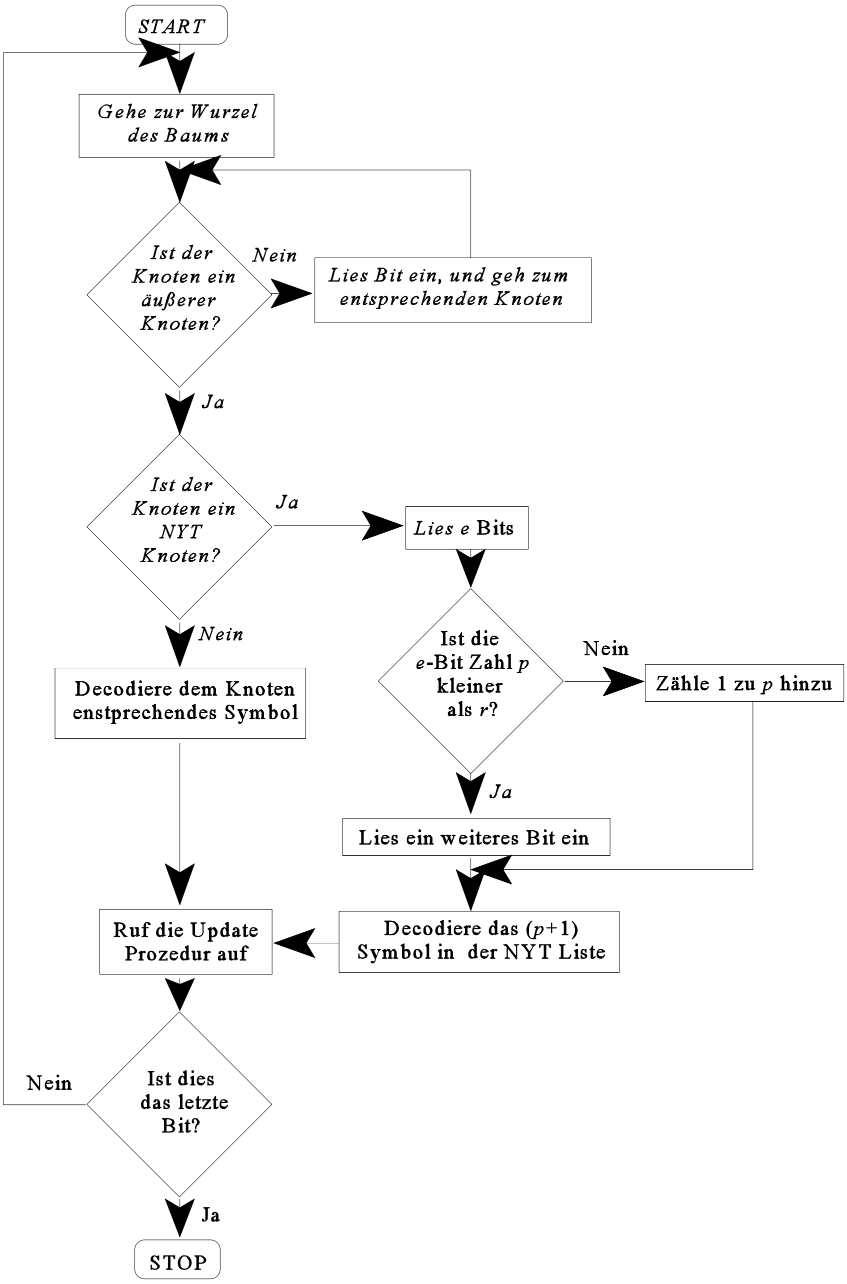
Anfangs besteht der Baum sowohl beim Übermittler als auch beim Empfänger aus einem einzigen Knoten, dem NYT Knoten, deswegen ist das Codewort für das allererste Symbol, das auftritt aus dem schon vorher vereinbarten Code. Nach dem allerersten Symbol senden wir

jedes Mal, wenn wir ein neues Symbol codieren, das zum ersten Mal auftritt, den Code für den NYT Knoten, den wir bekommen, indem wir den Huffman Baum von der Wurzel bis zum NYT Knoten durchqueren.
Dies warnt den Empfänger, daß das Symbol, dessen Code jetzt folgt noch keinen Knoten im Huffman Baum hat. Dem Code für den NYT Knoten folgt der vorher vereinbarte Code für das Symbol. Wenn ein zu codierendes Symbol einen Knoten im Baum hat, dann wird der Code übertragen, den man erhält, wenn man den Huffman Baum von der Wurzel bis zum dem Symbol entsprechenden Blatt durchquert

2.6.5 Decodierungsprozedur



Wenn wir den empfangenen Binärstring einlesen, durchqueren wir den Baum, wie wir es schon bei der Codierungsprozedur gemacht haben.
Wenn wir auf ein Blatt stoßen, wir das diesem Blatt entsprechende Symbol decodiert, wenn das Blatt der NYT Knoten ist, dann untersuchen wir die nächsten *e* Bits, um zu sehen, ob die resultierende Zahl kleiner als *r* ist, wenn sie kleiner als *r* ist, dann lesen wir ein weiteres Bit ein, um den Code für das Symbol zu vervollständigen.



Den Index für das Symbol erhalten wir, indem wir Eins zu der Dezimalzahl hinzuzählen, die dem e oder e+1 Bit String entspricht. Wenn das Symbol decodiert wird, wird der Baum upgedated und das nächste empfangene Bit wird benutzt, um den Baum zu durchqueren.

2.7 Anwendungen des Huffman Codes

2.7.1 Bildkompression

Bei der Bildkompression kann man dem Huffman Code den Wert eines Pixels zuordnen, z.B. für Graustufen sind das dann Werte zwischen 0 und 255
Oder man kann die Differenz zwischen Zwei Pixeln codieren, was meist zu besseren Resultaten führt.

2.7.2 Textkompression

Für Textkompression scheint der Huffman Code offensichtlich geeignet zu sein, wir haben ein klares Alphabet, und meist sogar ein klares Wahrscheinlichkeitsmodell, die Wahrscheinlichkeit für die Buchstaben in einem Roman wird sich nicht so stark von der in einem anderen Roman unterscheiden.
Man kann allerdings bei Textkompression noch mehr herausholen, wenn man Abhängigkeiten im Text beachtet.

Das ganze gilt auch für Audiokompression, wobei man auch hier wieder die Abhängigkeiten betrachten kann

Aber dies ist eine andere Geschichte