Einführung in die Codierungstheorie

Marco Block

Vorlesungen vom 19.01. und 21.01.2009
Algorithmen und Programmierung I - WiSe 2008/2009
Freie Universität Berlin - Fachbereich Mathematik und Informatik

Versionsstand 20.02.2009

Kapitel 1

Notation und Einführung

Der folgende Abschnitt gibt eine kurze Einführung in die Codierungstheorie. Dabei werden notwendige Begriffe und Definitionen vorgestellt.

Daten, die für eine Übertragung codiert werden, sollen anschließend wieder eindeutig decodiert werden können (siehe Abbildung 1.1).

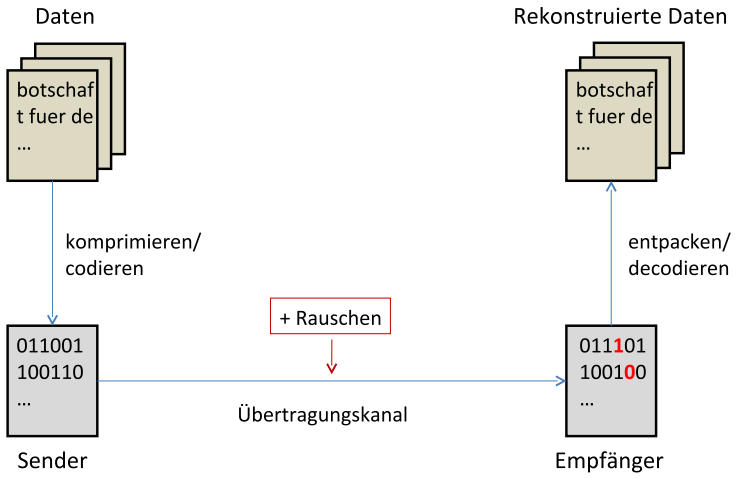

Abbildung 1.1: Übersicht

In einigen Fällen (z.B. Rauschen) kann es bei der Übertragung zu fehlerhaften Informationen kommen. Ziele der Codierungstheorie sind daher eine eindeutige Decodierbarkeit, kompakte (kurze) Codierungen und Fehler bei der Rekonstruktion nach Möglichkeit erkennen und korrigieren zu können.

Im Folgenden werden wir keine verlustbehafteten Codierungsverfahren (Komprimierung/Kompression) besprechen, wie die modernen und erfolgreichen Formate für Bilder, Musik oder Sprache (MPEG, JPEG, GIF, ...). An dieser Stelle sei auf die spezialisierte Literatur in [1, 5] verwiesen.

1.1 Alphabet und Wörter

Die endliche, nicht-leere Menge von Symbolen eines Datenraumes bezeichnen wir als **Alphabet** oder $\sum$. Damit läßt sich der Datenraum beschreiben, den es zu codieren gilt. So beispielsweise Symbole in Texten mit

$$\sum = \{a, b, ..., z, A, B, ..., Z, 0, 1, ..., 9\}.$$

Zum anderen kann ein weiteres Alphabet den Datenraum beschreiben, der uns für eine Übertragung zur Verfügung steht. In den meisten Fällen ist dieser Datenraum binär mit

$$\sum = \{0, 1\}.$$

Ein **Wort** (String) über $\sum$ ist eine endliche, möglicherweise leere Folge von Symbolen aus dem Alphabet $\sum$. Die leere Folge (das leere Wort) bezeichnen wir mit ε.

Die **Länge** eines Wortes ist die Anzahl der vorkommenden Symbole, so gilt beispielsweise $|\varepsilon| = 0$ und $|x_1x_2...x_n| = n$.

Wir bezeichnen mit $\sum^*$ die Menge aller Wörter und mit $\sum^+$ die Menge aller nicht-leeren Wörter über $\sum$, demzufolge ist

$$\sum^+ = \sum^* \setminus \{\varepsilon\}.$$

Mit $\sum^k$ bezeichnen wir die Menge aller Wörter über $\sum$ mit der Länge k.

Damit läßt sich also die Menge alle Wörter $\sum^*$ auch über die Wörter fester längen $\sum^k$ definieren

$$\sum^* = \bigcup_{k \geq 0} \sum^k$$

Um zwei Worte zu verbinden, verwenden wir die Konkatenation ($\circ$) mit

$$\_ \circ \_ : \sum^* \times \sum^* \to \sum^*.$$

Dadurch lassen sich erzeugte Wörter kombinieren, so gilt beispielsweise für das leere Wort $\varepsilon \circ w = w \circ \varepsilon = w$, für alle $w \in \sum^+$.

1.2 Codierung und Information

Der Begriff **Codierung** beschreibt im allgemeinen die Darstellung von Informationen durch Zeichenfolgen. Eine **Information** ist dabei ein Wort über dem Alphabet A.

Eine **codierte Information** ist ein Wort über dem Alphabet B. Dabei gilt meistens $B = \mathcal{B} = \{0, 1\}$ (Menge der Binärzeichen).

Für die Überführung einer Information über dem Alphabet A zum Alphabet B bedarf es einer injektiven Abbildung (siehe Abbildung 1.2) c mit

$$c : A \to B^+,$$

die fortgesetzt wird mittels $c(a_1a_2...a_k) = c(a_1) \circ c(a_2) \circ ... \circ c(a_k)$.

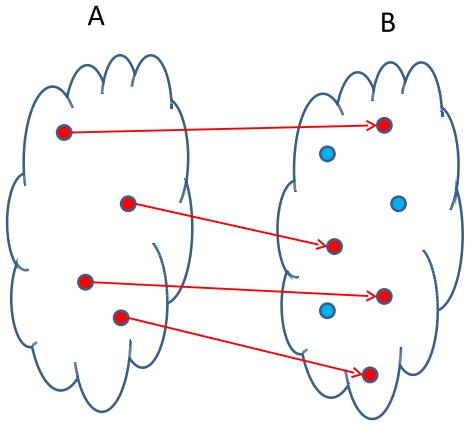

Abbildung 1.2: Injektive Abbildung von der Menge A zur Menge B. Jeder Punkt in A wird höchstens einmal in der Zielmenge B getroffen. Das ist notwendig, damit jedes Element aus A eine Codierung erhält, die später im Rekonstruktionsschritt eindeutig zum Element zurückführt.

Die **Menge der Codewörter** für ein Alphabet A (nennen wir auch kurz Code) ist $C(A) = \{c(a)|a \in A\}$. So könnte beispielsweise für $A = \{a, b, c\}$ die Menge der Codewörter die folgende sein $C(A) = \{00, 01, 10\}$.

Um einen eindeutig decodierbaren Code zu erhalten muss eine Umkehrfunktion d existieren, mit

$$d : \{c(w)|w \in A^+\} \to A^+.$$

Dieser muss eindeutig decodierbar sein, also muss gelten $d(c(w)) = w$.

Kapitel 2

Codierung mit fester Codewortlänge

Die einfachste Möglichkeit einen Code zu erstellen, ergibt sich durch den sogenannten Blockcode, bei dem die Länge aller Codewörter für die Einzelzeichen gleich ist. Im nachfolgenden Kapitel werden wir dann auch Codes mit variablen Codewortlängen kennenlernen.

2.1 Blockcode

Im Blockcode c wird jedes Zeichen durch ein $0-1-$Wort der gleichen Länge n codiert mit $c : A \to \mathcal{B}^n$. Mit einem Blockcode der Länge n lassen sich maximal 2^n Zeichen codieren.

Dazu identifizieren wir die i-te Zweierpotenz, mit der alle Symbole codiert werden können, mit

$$2^{i-1} < |A| \leq 2^i.$$

So benötigen wir beispielsweise für die Symbolmenge $A = \{a, b, c\}$ zwei Bit, denn $|A| = 3 \leq 2^2$. Die Codierung könnte dann wie folgt aussehen:

$c(s)$	s
00	a
01	b
10	c
11	$-$

Da alle Codewörter gleich lang sind, können wir über Abstandsfunktionen hilfreiche Eigenschaften von speziellen Codierungen beschreiben. Diese können uns, so werden wir es später sehen, auch bei der Rekonstruktion von Fehlern helfen. Als wichtigste Abstandsfunktion verwenden wir dabei den Hamming-Abstand.

2.1.1 Hamming-Abstand

Seien $b, b' \in \mathcal{B}^n$ mit $b = (b_1, ..., b_n)$ und $b' = (b'_1, ..., b'_n)$, dann sei der Hamming-Abstand (siehe Abbildung 2.1) wie folgt definiert

$$d_H(b, b') = |\{i | b_i \neq b'_i\}|.$$

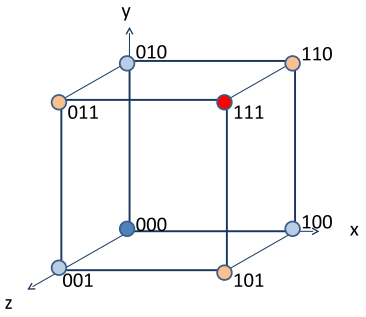

Abbildung 2.1: Hamming-Abstand: In diesem Beispiel gibt es die Codewörter 000 (blau) und 111 (rot). Der Abstand $d_H(000, 111) = 3$ ist so zu interpretieren, dass wir mindestens 3 Kanten entlang laufen müssen, um vom Codewort 000 zu 111 zu gelangen.

Für einen Blockcode c sei der Abstand gerade definiert als das Minimum aller Abstände

$$d_H(c) = min\{d_H(c(a), c(a')) | a \neq a' \wedge a, a' \in A\}.$$

Der Hammingabstand dient im Folgenden als Kenngröße zur Charakterisierung der Fähigkeit einer Blockcodes zu Fehlererkennung und -korrektur.

Angenommen, wir haben die folgenden Codewörter $C = \{001, 110, 111\}$ über einen Blockcode c erhalten, dann sind die paarweisen Abstände

$$d_H(001, 110) = 3$$

$$d_H(001, 111) = 2$$

$$d_H(110, 111) = 1.$$

Das Minimum aller Abstände $min\{3, 2, 1\} = 1$, demzufolge ist $d_H(c) = 1$.

2.1.2 Eigenschaften von Metriken

Es lassen sich aber auch andere Metriken (Abstandsfunktionen) d verwenden, mit $d : X \times X \to \mathbb{R}$ ist Metrik für Menge X, falls für beliebige $x, y, z \in X$ die folgenden Axiome erfüllt sind

(1) $d(x,y) \geq 0, \forall x,y \in X$

(2) $d(x,y) = 0 \iff x = y$ (Definitheit)

(3) $d(x,y) = d(y,x), \forall x,y$ (Symmetrie)

(4) $d(x,z) \leq d(x,y) + d(y,z), \forall x,y,z \in X$ (Dreiecksungleichung)

Zwei weitere wichtige Metriken sind beispielsweise für $r = (x,y), r' = (x',y') \in \mathbb{R}^2$

- Manhattan-Abstand $d_1$mit $d_1(r,r') = |x - x'| + |y - y'|$ und
- Euklidischer Abstand d_2 mit $d_2(r,r') = \sqrt{(x-x')^2 + (y-y')^2}$.

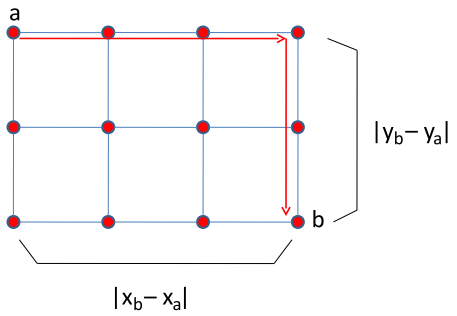

Abbildung 2.2: Beim Manhattan-Abstand werden die Differenzen komponentenweise berechnet und aufsummiert. Die Bezeichnung bezieht sich auf die Straßenverläufe Manhattans.

Die Decodierung eines Blockcodes mittels Maximum-Likelihood-Methode: Eine Nachricht $b \in \mathcal{B}^n$ wird decodiert als Zeichen $a \in A$ mit $d_H(c(a), b)$ ist minimal für alle a.

2.1.3 Fehlererkennender Code

Sei $c : A \to \mathcal{B}^n$ ein Blockcode, dann heißt c k-fehlererkennend, wenn $\forall a \in A$, $b \in \mathcal{B}^n$ gilt, dass

$$1 \leq d_H(b, c(a)) \leq k \Rightarrow b \notin C(A),$$

d.h. wenn sich ein empfangenes in bis zu k Stellen von einem Codewort unterscheidet, dann ist es selbst kein Codewort.

Eine graphische Interpretation, die die Eigenschaft k-fehlererkennend illustriert, ist Abbildung 2.3 zu entnehmen.

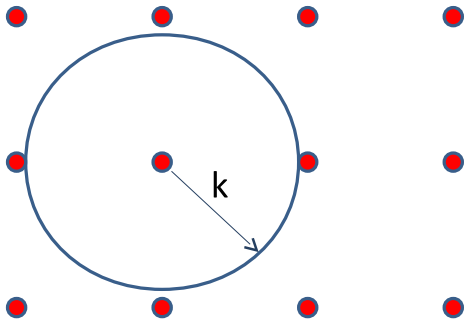

Abbildung 2.3: In dieser zweidimensionalen Abbildung läßt sich die Eigenschaft k-fehlererkennend anschaulich interpretieren. So läßt ein Abstand k von jedem Datenpunkt aus $\mathcal{B}^n$ mindestens k Fehler erkennen. Eine erfolgreiche Rekonstruktion der Daten spielt dabei keine Rolle.

Eine Codierung C ist allgemein k-fehlererkennend, wenn $d_H(C) \geq k + 1$.

Wenn wir beispielsweise die Codierungen aus Abbildung 2.1 verwenden, dann hat dieser folglicherweise die Eigenschaft 2-fehlererkennend zu sein. Bei einem Fehler von 000 aus landen wir in den hellblauen und bei zwei Fehlern in den orangen Bereichen. Da also der minimale Abstand von allen Codewörtern zueinander drei ist, können zwei Fehler erkannt werden.

2.1.4 Fehlerkorrigierender Code

Sei $c : A \to \mathcal{B}^n$ ein Blockcode, dann heißt c k-fehlerkorrigierend, wenn $\forall a \in A$, $b \in \mathcal{B}^n$ gilt, dass

$$d_H(b, c(a)) \leq k \Rightarrow \forall a' \neq a : d_H(b, c(a')) > k \ .$$

Die Abbildung 2.4 liefert eine graphische Interpretation.

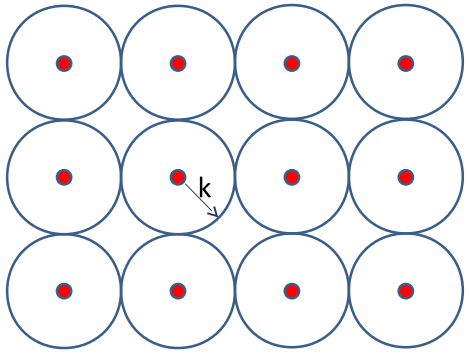

Abbildung 2.4: Bei der Eigenschaft k-fehlerkorrigierend gibt es für jeden Datenpunkt aus $\mathcal{B}^n$ ein k für das eine erfolgreiche Rekonstruktion vorgenommen werden kann.

Wenn c k-fehlerkorrigierend und b eine fehlerhafte Nachricht ist, kann $c(a)$ als eindeutiges Codewort (bei Maximum-Likelihood-Decodieren siehe Abschnitt 2.1.2) mit $\leq k$ Fehlern zugeordnet werden.

Als Beispiel soll wieder die Abbildung 2.1 dienen. Es ist leicht zu sehen, dass der Code 1-fehlerkorrigierend ist. Eine Codierung C ist allgemein k-fehlerkorrigierend, wenn $d_H(C) \geq 2k + 1$.

2.1.5 Informationsrate

Um den Informationsgehalt für eine Codierung zu ermitteln, lässt sich die **Informationsrate** IR mit

$$IR = \frac{\#Informationsbit}{Codewortlaenge}$$

ermitteln. Zu den im folgenden Abschnitt vorgestellten Blockcode-Varianten werden wir die dazugehörigen Informationsraten ermitteln.

2.2 Blockcode-Varianten

Jetzt gibt es viele verschiedene Möglichkeiten Blockcodes zu erstellen, die sich in Informationsrate, Fehlererkennung und -korrektur unterscheiden. Im Folgenden werden ein paar elementare Varianten vorgestellt.

2.2.1 Variante 1: Paritätsbit

Ein Blockcode c_p mit einem zusätzlichen Paritätsbit, mit $c_p : A \to \mathcal{B}^{n+1}$ codiert zunächst das Wort $c(a)$ und hängt anschließend das Paritätsbit $b_p(a)$, mit

$$b_p(a) = \begin{cases} 0 & , \textit{gerade Anzahl von Einsen} \\ 1 & , \textit{sonst} \end{cases}$$

an, so dass per Definition

$$c_p(a) =_{def} \; c(a) \circ b_p(a).$$

Daraus folgt, dass $d_H(c_p) \geq 2$, weil alle Codewörter $c_p(a)$ gerade Anzahl von Einsen haben. Die Informationsrate beträgt $IR(c_p) = \frac{n}{n+1}$.

Soll beispielsweise das Codewort $c(s) = 110$ mit Paritätsbit angegeben werden, so ist $c_p(s) = 1100$.

2.2.2 Variante 2: Doppelcodierung

Bei der Doppelcodierung c^2 schreiben wir den Code einfach zweimal hin $c^2 : A \to \mathcal{B}^{2n}$, mit

$$c^2(a) =_{def} \; c(a) \circ c(a).$$

Daraus läßt sich ableiten, dass $d_H(c^2) \geq 2$. Fehlerkorrigierend ist diese Codierung aber nicht zwangsläufig. Die Informationsrate beträgt $IR(c^2) = \frac{n}{2n} = \frac{1}{2}$.

Beispielsweise ist für $c(s) = 0110$ der Code mit Doppelcodierung $c^2(s) = 01100110$.

2.2.3 Variante 3: Doppelcodierung mit Paritätsbit

Verwenden wir zusätzlich zur Doppelcodierung noch ein Paritätsbit $c_p^2 : A \rightarrow \mathcal{B}^{2n+1}$, mit

$$c_p^2(a) =_{def} \; c(a) \circ c(a) \circ b_p(a)$$

so läßt sich beobachten, dass $d_H(c_p^2) \geq 3$. Damit ist c_p^2 sogar 1-fehlerkorrigierend.

Beweis: $d_H(c(a), c(a')) \geq 2 \Rightarrow d_H(c_p^2(a), c_p^2(a')) \geq 4$. Nun ist aber der Fall interessant, bei dem $d_H(c(a), c(a')) = 1$, dann ist aber $d_H(c_p^2(a), c_p^2(a')) = 3$, da der Abstand verdoppelt wird und das Paritätsbit kippt.

Für die Doppelcodierung mit Paritätsbit erhalten wir folgende Informationsrate

$$IR(c_p^2) = \frac{n}{2n+1} < \frac{1}{2}.$$

Aufpassen muss man lediglich beim Paritätsbit, da für $c(s) = 1011$ das letzte Bit für $c_p^2(s) = 101110111$ nur aus dem Codewort $c(s)$ ergibt.

2.2.4 Übersicht der bisherigen Codierungen

Wir erkaufen uns die Möglichkeit zur Fehlererkennung bzw. -korrektur durch zusätzliche Bits, die für den Informationsgehalt redundant sind.

z	$c(z)$	$c_p(z)$	$c^2(z)$	$c_p^2(z)$
a	00	000	0000	00000
b	01	011	0101	01011
c	10	101	1010	10101
d	11	110	1111	11110

Es ist leicht zu sehen, dass $c^2(z)$ aus Spalte 4 trotz doppelter Datenmenge nicht 1-fehlerkorrigierend ist, da beispielsweise a mit einem Fehlbit an der ersten Position den gleichen Hamming-Abstand hätte, wie c an der dritten Position:

$$d_H(c^2(a), 1000) = d_H(c^2(c), 1000) = 1.$$

Die Spalte 5 hingegen ist 1-fehlerkorrigierend.

2.2.5 Variante 4: Kreuzsicherungscode

Beim Kreuzsicherungscode c_x werden die Codes in einer Tabelle angeordnet und für jede Zeile und jede Spalte jeweils ein Paritätsbit ermittelt (siehe Abbildung 2.5).

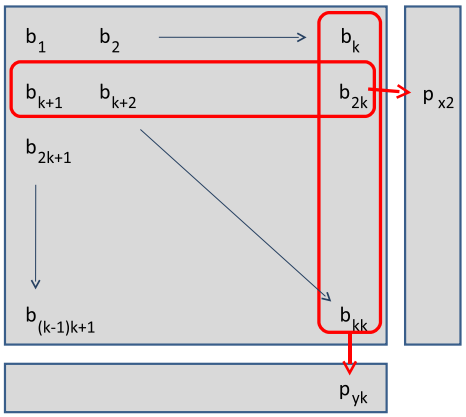

Abbildung 2.5: Der Kreuzsicherungscode bringt eine Nachricht in die Matrixform $n \times n$ und speichert zusätzlich für alle Zeilen und Spalten $2n$ Paritätsbits.

Da es sich bei dem vorliegenden Skript um eine Einführung in die Codierungstheorie handelt, können die Codierungen mit festen Codewortlängen nur kurz angerissen werden. Als weiterführende Literatur empfehle ich an dieser Stelle die Bücher von Wilfried Dankmeier [1] und Ralph-Hardo Schulz [2].

Kapitel 3

Codierung mit variabler Codewortlänge

Im vorhergehenden Kapitel haben wir beim Blockcode eine feste Codewortlänge gefordert. Dadurch war es uns möglich, Fehler zu erkennen und gegebenenfalls zu korrigieren. Es läßt sich leicht nachvollziehen, dass eine feste Codewortlänge gegebene Daten unabhängig vom Inhalt codiert und damit keine Komprimierung möglich ist.

Verwenden wir aber variable Codewortlängen, dann könnten wir beispielsweise Symbole die sehr häufig auftreten mit kürzeren Codewörtern belegen. Dadurch erhalten wir bei geschickter Codewahl meistens bessere Komprimierungseffekte. Unser Ziel könnte im Folgenden, als die Suche nach der minimal codierten Nachricht (in Bezug auf die Gesamtlänge), formuliert werden. Es gibt verschiedene Methoden, die dieses Problem lösen können. Wir lernen mit Präfixcodes ein auf statistischer Codierung beruhendes Verfahren kennen.

3.1 Absolute und relative Häufigkeiten

Um die absolute Häufigkeit eines Symbols zu ermitteln, werden in dem zu codierenden Text die auftretenden Symbole notiert und die Anzahl ihres Auftretens identifiziert. Unsere einfache Idee, häufig auftretende Symbole durch kürzere Codewörter zu repräsentieren, behalten wir dabei im Hinterkopf.

3.2 Präfixcodes allgemein

Ein Code, bei dem kein Codewort Präfix eines anderen Codewortes ist, heißt **Präfixcode**. Präfixcodes lassen sich in einer Baumstruktur gut darstellen. An den Blättern des Baumes sind die codierten Zeichen notiert und die Kantenmarkierungen der Wege von Wurzel zu Blättern liefern das Codewort.

Dabei sind alle Kanten zum ersten (linken) Kind mit '0' und zum zweiten mit '1' markiert.

3.2.1 Codierung

Gegeben sei nun ein Präficdode in Baumdarstellung. Um die Codewörter zu extrahieren, müssen wir die Kanten entlang laufen. Schauen wir uns das Beispiel in Abbildung 3.1 an.

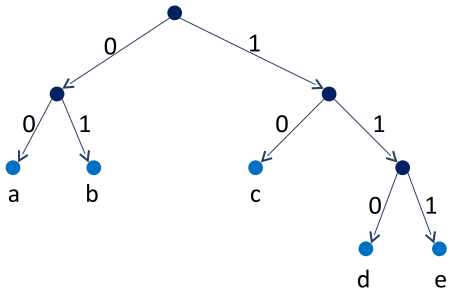

Abbildung 3.1: Beispiel für eine Präfixcodierung.

Beginnend von der Wurzel ergibt sich das Codewort für das jeweilige Zeichen aus der Reihenfolge der besuchten Kanten.

So durchlaufen wir beispielsweise die Kanten 0 und 0 um bis zum Zeichen a zu gelangen. Es gibt ergibt sich daraus die folgende Codetabelle:

Symbol	Codewort
a	00
b	01
c	10
d	110
e	111

Anhand der Abbildung 3.1 ist auch leicht zu sehen, dass kein Codewort Präfix eines anderen sein kann. Würde das der Fall sein, müsste an einem inneren Knoten ein Zeichen stehen, was aber laut Abschnitt 3.2 nicht erlaubt ist.

Eine Zeichenfolge, beispielsweise „ade“ führt demnach über die Codetabelle zu „00110111“.

3.2.2 Decodierung

Um nun eine codierte Nachricht wieder in eine Folge von Symbolen zu wechseln, beginnen wir mit dem ersten Symbol auf der linken Seite und durchlaufen beginnend mit der Wurzel die entsprechenden Kanten entlang, bis wir bei einem Zeichen angekommen sind. Sei beispielsweise der Code „1100101“ gegeben, dann führen die ersten 3 Kanten zu d und der Rest entsprechend zu „bb“.

3.2.2.1 Eindeutige Decodierbarkeit

Es läßt sich zeigen, dass eine mit einem Präfixcode codierte Nachricht eindeutig decodierbar ist. Führen wir einen indirekten Beweis d.h. $p \Rightarrow q$ ist semantisch äquivalent zu $\neg q \Rightarrow \neg p$:

Lemma 1: Eine mit einem Präfixcode codierte Nachricht ist eindeutig decodierbar.

Beweis: Angenommen $\exists a_1 a_2 ... a_m \neq a'_1 a'_2 ... a'_k$ mit $c(a_1 a_2 ... a_m) = c(a'_1 a'_2 ... a'_k)$ und $m \neq k$. Sei nun i der kleinste Index mit $a_{i+1} \neq a'_{i+1}$ und $a_i = a'_i$, dann folgt daraus, dass $c(a_1 a_2 ... a_i) = c(a'_1 a'_2 ... a'_i)$.

Jetzt gibt es drei zu betrachtende Fälle:

Fall 1: $|c(a_{i+1})| = |c(a'_{i+1})| \Rightarrow c(a_{i+1}) \neq c(a'_{i+1})$, das ist aber ein Widerspruch zu der Annahme, dass die Gesamtcodierungen gleich sind.

Fall 2: $|c(a_{i+1})| < |c(a'_{i+1})| \Rightarrow$aus der Annahme folgt, $c(a_{i+1})$ ist Präfix von $c(a'_{i+1})$. Das ist widerum ein Widerspruch zum Präfixcode.

Fall 3: $|c(a_{i+1})| > |c(a'_{i+1})| \Rightarrow$aus der Annahme folgt, $c(a'_{i+1})$ ist Präfix von $c(a_{i+1})$. Das ist ebenfalls ein Widerspruch zum Präfixcode.

Damit haben wir **Lemma 1** bewiesen.

Hinweis: Es gibt aber auch eindeutig decodierbare Codes, die keine Präfiscodes sind.

3.2.2.2 Ungleichung von Kraft

Die Ungleichung von Kraft besagt im allgemeinen, dass für einen Binärbaum die vorkommenden Blatttiefen die folgende Ungleichung erfüllen. Damit lassen sich in Bezug auf die Präfixcodes, gültige von ungültigen schnell unterscheiden.

(1) Sei c ein Präfixcode für $A = \{a_1, a_2, ..., a_m\}$, es gilt:

$$\sum_{i=1}^{m} \frac{1}{2^{|c(a_i)|}} \leq 1$$

(2) Seien $n_1, n_2, ..., n_m$ natürliche Zahlen und $\sum_{i=1}^{m} \frac{1}{2^{n_i}} \leq 1$. Dann gibt es einen Präfixcode für m Zeichen mit den Codewortlängen $n_1, n_2, ..., n_m$.

Die beiden Behauptungen (1) und (2) wollen wir im folgenden noch beweisen.

Beweis: (1) Induktion nach der Anzahl der Blätter im Codewortbaum. O.B.d.A. haben alle inneren Knoten den Verzweigungsgrad 2.

Induktionsanker: $m = 1$, $m = 2$ sind erfüllt.

$$\frac{1}{2^0} = 1 \text{ und } \frac{1}{2^1} + \frac{1}{2^1} = 1$$

Induktionsvorraussetzung: Behauptung ist für Blätter $\leq n$ erfüllt.

Induktionsschritt: Dazu betrachten wir die Zwillingsblätter der Tiefe n und ihren Vaterknoten. Jetzt entfernen wir beide Blätter $v_1$und v_2 und wenden die Induktionsvorraussetzung auf den entstehenden Baum mit m Blättern an

$$\begin{aligned} \sum_{i=1}^{m+1} 2^{-n_i} &= 2^{-n} + 2^{-n} + \sum_{i=3}^{m+1} 2^{-n_i} \\ &= 2^{-(n-1)} + \sum_{i=3}^{m+1} 2^{-n_i} \\ &\quad Baum\, mit\, Blättern\, v, v_3, v_4, ..., v_{m+1}\, (nach\, I.V.) \\ &\leq 1 \end{aligned}$$

Beweis: (2) Wir ordnen die Längen $n_1 \leq n_2 \leq ... \leq n_m$ der Codewörter und setzen $c(a_1) = 00...0$ (sind gerade n_1 viele 0en).

Unter der Annahme, dass $c(a_1), ..., c(a_i)$ mit $i < m$ bereits bestimmt ist, wählen wir nun $c(a_{i+1})$ als lexikographisch kleinstes Wort, das die Codewörter $c(a_1), ..., c(a_i)$ nicht als Präfix enthält.

Die Frage lautet, gibt es ein solches?

Wenn ja, dann müsste genug Platz im Baum vorhanden sein, wenn nicht ist folglicherweise eines der Codewörter $c(a_1), ..., c(a_i)$ ein Präfix von $c(a_{i+1})$.

Insgesamt gibt es $2^{n_{i+1}}$ Wörter der Länge n_{i+1}. Wir summieren einfach die durch die anderen Codewörter belegten Längen auf und erhalten:

$c(a_1)$ verbietet $2^{n_{i+1}-n_1}$

$c(a_2)$ verbietet $2^{n_{i+1}-n_2}$

...

$c(a_i)$ verbietet $2^{n_{i+1}-n_i}$

Demnach sind verboten: $\sum_{j=1}^{i} 2^{n_{i+1}-n_j} = 2^{n_{i+1}} \cdot \sum_{j=1}^{i} 2^{-n_j} < 2^{n_{i+1}}$, q.e.d.

Eine Veranschaulichung der Ermittlung der noch freien Positionen gibt Abbildung 3.2.

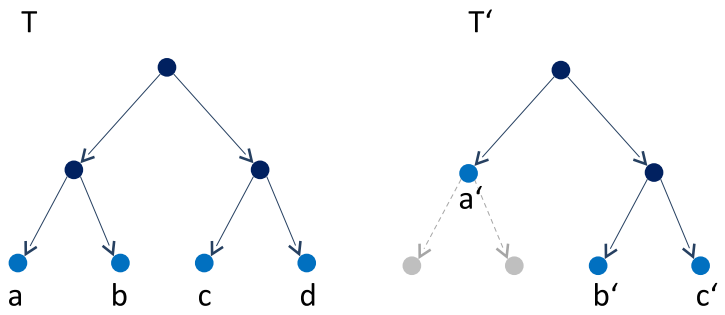

Abbildung 3.2: Der Baum T auf der rechten Seite kann maximal 4 Codewörter der Länge 2 aufnehmen, also $2^2 = 4$. Auf der linken Seite in Baum T' gibt es beispielsweise ein Codewort der Länge 1. Damit sind aber $2^{2-1} = 2^1 = 2$ Codewörter aus der maximalen Anzahl bereits verbraucht und es können nur noch $2^4 - 2^{2-1} = 2$ Wörter der Länge 2 untergebracht werden.

Schauen wir uns ein konkretes Beispiel an: Angenommen, wir haben die aufsteigend sortierten Codewortlängen $1, 3, 3, 3, 4, 4$. Ein Test zeigt uns, ob die Ungleichung von Kraft erfüllt ist und damit ein gültiger Codebaum aufgestellt werden kann:

$$1 \cdot \frac{1}{2} + 3 \cdot \frac{1}{8} + 2 \cdot \frac{1}{16} = 1$$

Die Codewörter könnten beispielsweise $\{0, 100, 101, 110, 1110, 1111\}$ sein (siehe Abbildung 3.3).

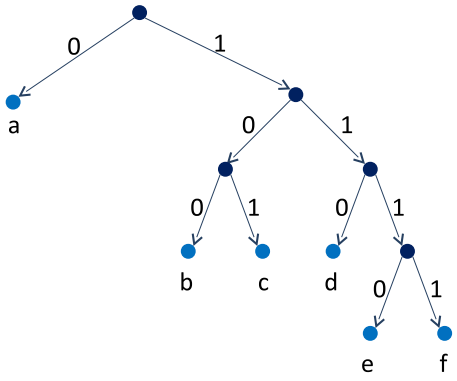

Abbildung 3.3: Beispiel für eine Präfixcodierung.

An dieser Stelle ist es eine gute Übung zu zeigen, dass $c(a_5) = 1110$ mit $n_5 = 4$ nach obigem Beweis korrekt ist, da $14 < 16$. Sollten wir stattdessen $c(a_5) = 111$ mit $n_5 = 3$ verwenden scheint es zunächst ebenfalls zu funktionieren, da $7 < 8$ ist, aber beim Folgecodewort $c(a_6) = 1111$ schlägt die Berechnung fehl, da $16 \not< 16$ ist.

3.2.3 Codierung mit Wahrscheinlichkeitsverteilungen der Zeichen

Die Umstellung von absoluten zu relativen Häufigkeiten führt uns zu den Wahrscheinlichkeitsverteilungen. Gegeben seien $A = \{a_1, ..., a_n\}$ und $p : A \to (0, 1]$, mit $p_i = p(a_i)$ ist die Wahrscheinlichkeit des Auftretens von Zeichen a_i. Es soll weiterhin gelten, dass $\sum_{i=1}^{n} p_i = 1$.

Sei nun $c : A \to \mathcal{B}^+$ ein Präfixcode, dann ist der Erwartungswert $n(c)$ für die Länge eines Codewortes mit

$$\begin{aligned} n(c) &= \sum_{i=1}^{n} |c(a_i)| \cdot p_i \\ &= \sum_{i=1}^{n} d_T(a_i) \cdot p_i \end{aligned}$$

gleich dem Erwartungswert für die Tiefe eines Blattes, mit $d_T(a_i)$ entspricht der Tiefe des Blattes a_i im Codebaum T. Der Einfachheit halber wollen wir auch $n(T)$ als Notation erlauben und meinen damit den Erwartungswert für den Codebaum T gegebenen Präfixcode c_T, also $n(c_T)$.

3.3 Huffman-Algorithmus

Mit dem Huffman-Algorithmus läßt sich ein optimaler Präfixcode konstruieren. Beweisen werden wir das aber etwas später. Zunächst schauen wir uns die Arbeitsweise des Algorithmus an.

Der Algorithmus arbeitet bottom-up, d.h. er bearbeitet kleinere Teilprobleme zuerst und hangelt sich zu größeren weiter. Dabei verwendet er die Greedy-Strategie.

```
Initialisierung:
   Zeichen bezüglich Häufigkeit aufsteigend sortieren
   und als Liste von n Bäumen (je einem Knoten) mit
   Markierung a_i|p_i ablegen.
Rekursionsschritt:
   Streiche die zwei Bäume T_1 und T_2 mit minimalem Gewicht
   aus der Liste und füge einen neuen Baum ein, dessen Wurzel
   die Bäume T_1 und T_2 als Teilbäume hat und mit der Summe
   der Gewichte markiert ist. Wiederhole solange den Rekursions-
   schritt, bis nur noch ein Baum vorhanden ist.
```

Nach der Erzeugung des Codebaumes, entsprechen die Wegmarkierungen von der Wurzel bis zu den Blättern den jeweiligen Codewörtern (wie es bereits in Abschnitt 3.2 gezeigt wurde).

Ein Beispiel zur Erzeugung eines Codebaumes mit den entsprechenden absoluten Häufigkeiten $\{(a, 5), (b, 9), (c, 12), (d, 13), (e, 16), (f, 45)\}$ ist in Abbildung 3.4 zu sehen.

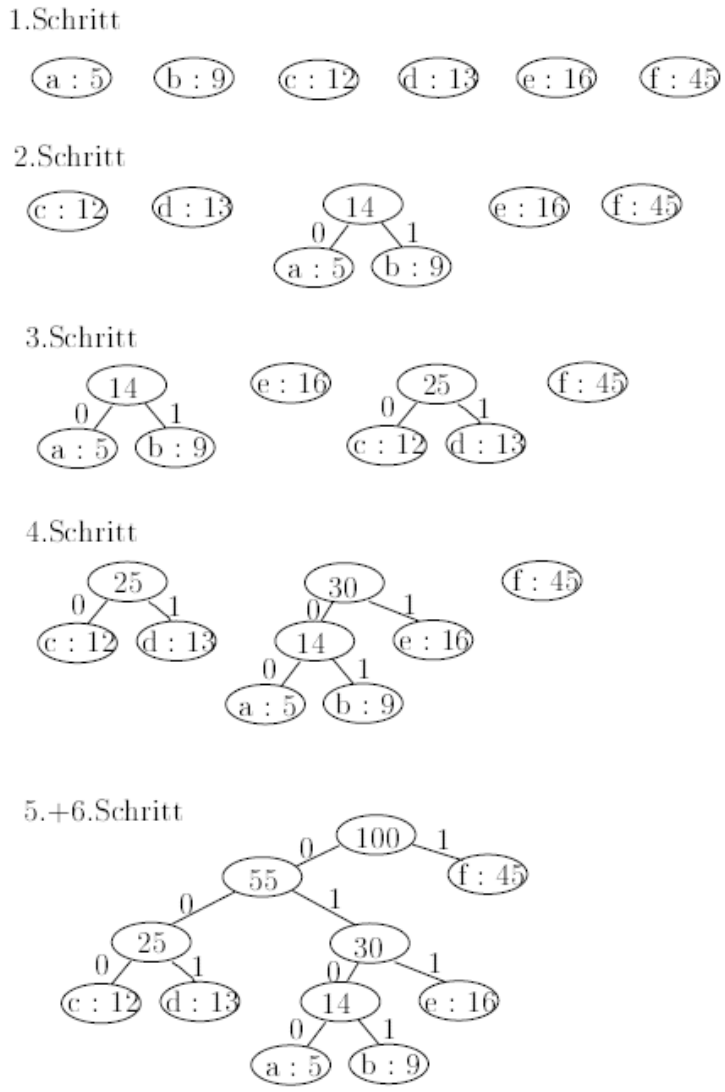

Abbildung 3.4: Beispiel für eine Präfixcodierung analog zu [4].

Die erwartete Codewortlänge $n(c)$ dieses Textes ist

$$n(c) = 4 \cdot \tfrac{5}{100} + 4 \cdot \tfrac{9}{100} + 3 \cdot \tfrac{12}{100} + 3 \cdot \tfrac{13}{100} + 3 \cdot \tfrac{16}{100} + 1 \cdot \tfrac{45}{100} = 2.24.$$

Nun wollen wir zeigen, dass der durch den Huffman-Algorithmus konstruierte Codebaum, in Bezug auf die mittlere Codewortlänge, optimal ist, d.h. es gibt keinen anderen Präfixcode mit einer kürzeren mittleren Codewortlänge. Dazu beweisen wir zunächst das folgende Lemma.

Lemma 2: Seien $a_1, ..., a_n$ so durchnummeriert, dass $p(a_1) \leq p(a_2) \leq ... \leq p(a_n)$. Dann gibt es einen optimalen Codebaum B, so dass die beiden Blätter a_1 und a_2 einen gemeinsamen Vaterknoten haben.

Das ist übrigens gerade die Idee, die dem Huffman-Algorithmus innewohnt (greedy-Strategie)!

Beweis: Sei B ein beliebiger optimaler Codebaum. Seien a_i und a_j, mit $i < j$, zwei Blätter mit gemeinsamen Vaterknoten, die sich auf der tiefsten Ebene des Baumes befinden, sagen wir t.

Wir modifizieren B zu B', indem wir a_1 und a_i als auch a_2 und a_j vertauschen. Zu zeigen ist nun, dass $n(B') \leq n(B)$ ist und damit B' ein optimaler Codebaum ist.

Tauschen wir nun a_1 mit a_i, mit a_1 befindet sich in B in der Tiefe $t' \leq t$, denn a_i und a_j liegen ja bereits maximal tief. Die neue mittlere Codewortlänge unterscheidet sich zu der von B wie folgt:

$$\begin{aligned} n(B') &= \textstyle\sum_{j=1}^{m} p_j d_{B'}(a_j) \\ &= n(B) - p(a_i) \cdot t - p(a_1) \cdot t' + p(a_1) \cdot t + p(a_i) \cdot t' \\ n(B') - n(B) &= (t' - t) \cdot (p(a_i) - p(a_1)) \\ & \quad Term\,1 \leq 0;\ Term\,2 \geq 0 \end{aligned}$$

Daraus schlussfolgern wir, dass die Differenz ≤ 0 ist und daher ist auch B' optimal.

Analog dazu läßt sich auch a_2 mit a_j vertauschen und zeigen, dass sich die mittlere Codewortlänge nicht verschlechtert.

Jetzt zeigen wir durch Induktion über n, dass der Huffman-Codebaum optimal ist unter allen Präfixbäumen für die gegebene Wahrscheinlichkeitsverteilung (angelehnt an[3]).

Induktionsanker: $n = 1$, ist klar.

Induktionsvorraussetzung: Sei nun $n > 1$ und es gelte $p(a_1) \leq p(a_2) \leq ... \leq p(a_n)$. Für $(b, a_3, ..., a_n)$ mit $p(b) = p(a_1) + p(a_2)$ erzeugt der Huffman-Algorithmus einen optimalen Codebaum der Größe $n - 1$. Ergebnis sei Baum B.

Induktionsschritt: Wir expandieren b mit a_1 und a_2 und erhalten B', den der Huffman-Algorithmus auf $(a_1, a_2, ..., a_n)$ liefern würde.

Es gilt nun $n(B') = n(B) + (p(a_1) + p(a_2))$. Sei B'' ein optimaler Codebaum für $(a_1, a_2, ..., a_n)$. Wegen bereits gezeigtem **Lemma 2** können wir annehmen, dass in B'' die Blätter a_1, a_2 einen gemeinsamen Vater a haben.

Indem wir die Blätter a_1 und a_2 aus B'' entfernen, erhalten wir einen Codebaum B''' für $(a, a_3, ..., a_n)$ mit $p(a) = p(a_1) + p(a_2)$. Da B für die Verteilung optimal ist (laut Induktionsvorraussetzung), gilt $n(B) \leq n(B''')$. Ferner gilt $n(B'') = n(B''') + (p(a_1) + p(a_2))$. Daher folgern wir $n(B') \leq n(B'')$ und daher ist der Baum B' optimal, was zu zeigen war.

Literaturverzeichnis

[1] Dankmeier D.: „*Grundkurs Codierung: Verschlüsselung, Kompression, Fehlerbeseitigung.*“, 3.Auflage, Vieweg-Verlag, 2006

[2] Schulz R.-H.: „*Codierungstheorie: Eine Einführung*“, 2.Auflage, Vieweg+Teubner, 2003

[3] Schöning U.: „*Algorithmik*“, ISBN-13: 978-3827410924, Spektrum Akademischer Verlag, 2001

[4] Cormen T.H., Leiserson C.E., Rivest R.L.: „*Introduction to Algorithms*“, MIT-Press, 2000

[5] Wikibook zur Datenkompression: *http://de.wikibooks.org/wiki/Datenkompression*