Einführung

in

Codierungstheorie

Version 2019.5

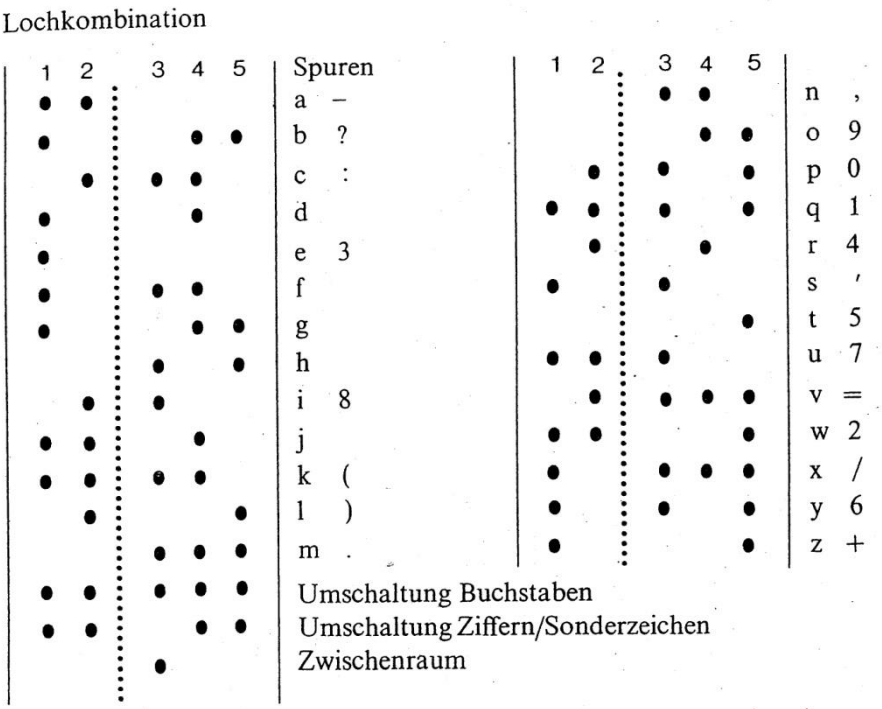
Autor: Dieter Lindenberg

Inhalt

Kodierungen

Blockcode

Bei der Lochstreifencodierung wird oft eine fünfspurige Anordnung mit einer Transportrasterung nach der zweiten Spur benutzt. Da sich mit ihr nur $2^5 = 32$ Zeichen darstellen lassen, benutzt man zwei unterschiedliche Zeichensätze. Allerdings müssen zwei besondere Kombinationen für das Umschalten von Buchstaben auf Ziffern und umgekehrt definiert werden. Damit lässt sich die Anzahl der darstellbaren Zeichen nahezu verdoppeln.



Vorteil: Falls sehr selten umgeschaltet werden muss, ist diese Umschaltmethode sehr effizient.

Nachteil: Wenn sehr häufig umgeschaltet werden muss, hat man praktisch einen 10-bit-Code für nur rund 60 Zeichen.

Ergebnis: Mit einem 5-Spur-Raster bzw. einer 5-bit-Maschine lassen sich mehr als 32 Zeichen darstellen!

Alle Zeichen sind nach der Codierung gleich lang! Man spricht von einem sog. **Blockcode**. Blockcodes benötigen keine Lücken. Mit „Zwischenraum“ ist das sog. Leerzeichen gemeint.

Aufgabe: Die obige Abbildung enthält Fehler. Eine Lochkombination ist doppelt belegt. Welche? Außerdem fehlen einige Lochkombinationen. Welche?

Morsealphabet

Das Morsealphabet wurde von dem Amerikaner Samuel Morse (1791-1872) erfunden und wird heute auf der ganzen Welt verwendet. Buchstaben, die besonders häufig verwendet werden, bekamen von Morse einen kurzen Code zugewiesen (zum Beispiel das "e" oder das "i"); selten verwendete Buchstaben (wie zum Beispiel das "q") wurden mit einem längeren Code dargestellt.

a	. –	n	– .
ä	. – . –	o	– – –
b	– . . .	ö	– – – .
c	– . – .	p	. – – .
ch	– – – –	q	– – . –
d	– . .	r	. – .
e	.	s	. . .
f	. . – .	t	–
g	– – .	u	. . –
h	. . . .	ü	. . – –
i	. .	v	. . . –
j	. – – –	w	. – –
k	– . –	x	– . . –
l	. – . .	y	– . – –
m	– –	z	– – . .
1	. – – – –	6	– . . . .
2	. . – – –	7	– – . . .
3	. . . – –	8	– – – . .
4	. . . . –	9	– – – – .
5	. . . . .	0	– – – – –
Punkt	. – . – . –	Fragezeichen	. . – – . .
Komma	– – . – –	Apostroph	. – – – – .
Irrtum	. . . . .		
Verstanden	. . . – .	Ende (Schlußzeichen)	. – . – .

Der Morsecode ist ohne Lücken (zeitlich: längere Pausen) zwischen den Zeichen nicht dekodierbar.
Beispiel: · – · · – = au, ena, etea, etu, lt

Deshalb müssen alle kodierten Zeichen (nach der Kodierung) durch eine Lücke getrennt werden. Die Ursache dieses Problems liegt darin, dass viele kodierte Zeichen auch Anfangsteil eines anderen kodierten Zeichens sind.

Beispiel: der Code von a ist auch Anfangscode von ä, 1, j, l, p, r und w.

Definition: Ein Code heißt dekodierbar genau dann, wenn jedes kodierte Wort nur auf eine einzige Art zurückübersetzt werden kann.

Hinweis: dekodieren bedeutet ent-kodieren. Man kann also nur etwas dekodieren, was vorher kodiert worden ist.
Es muss also nicht jedes beliebige Wort rückübersetzbar sein, sondern nur solche Worte, die schon vorher übersetzt worden sind.

Jede Sprache wird mithilfe eines Zeichenvorrates gebildet. Dieser Zeichenvorrat wird üblicherweise Alphabet genannt. Beispiele:
Das Alphabet für natürliche Zahlen ist {0, 1, …, 9}.
Das Alphabet für Hexadezimalzahlen ist {0, .., 9, A, ..,F}.
Das Alphabet für Binärzahlen ist {0, 1}.
Das Alphabet für Dezimalzahlen ist {0, 1, …, 9, **,** }.
Das Alphabet für die deutsche Sprache ist {a, .., z, A, .. Z, ä, .., :, ;, !, ?, „ , “ }.

Eine beliebige Aneinanderreihung von Zeichen eines Alphabetes wird als **Wort** bezeichnet. Ob jedes beliebige Wort auch zur Sprache gehört, entscheiden oft bestimmte Regeln, sog. Grammatiken der jeweiligen Sprache.
Beispiel: Jede beliebige Kombination von 0 und 1 ergibt eine Binärzahl. Hier gibt es keine Regeln. Aber nicht jede beliebige Kombination von Zeichen aus {0, 1, …, 9, **,** } ergibt eine Dezimalzahl.

Definition: Gegeben seien zwei Sprachen L_1 und L_2 mit den beiden zugehörigen Alphabeten A_1 und A_2.
Unter dem Begriff **Code** versteht man eine Überführungs**funktion**
$f : A_1 \rightarrow L_2$ (beachte den Unterschied Alphabet – Sprache!)

Definition: Ein Code genügt der **Fano-Bedingung** genau dann, wenn kein codiertes Zeichen mit dem Anfang eines anderen codierten Zeichens übereinstimmt.
Für alle x, y aus der Menge A_1 mit $x \neq y$ gilt: f(x) ist nicht Präfix von f(y)

Beispiel: Die Telefonnummern genügen der *Fano-Bedingung*.

Die *Fano-Bedingung* ist hinreichend, aber nicht notwendig für dekodierbaren Code.
Beispiel: Alphabet $A_1 = \{a, b\}$, Alphabet $A_2 = \{0, 1\}$
Code: f(a) = 0, f(b) = 01
Dieser Code ist dekodierbar: 001000101010 = a b a a b b b a

Das Wort 0011 ist nicht rückübersetzbar !

Optimale Codes müssen 1. eindeutig dekodierbar
2. wirtschaftlich sein.

Die erste Bedingung ist natürlich selbstverständlich: Codes, die nicht dekodierbar sind, sind einfach unsinnig!

Die zweite Bedingung ist leider nicht eindeutig formuliert. Was bedeutet *„wirtschaftlich“*? Zum Beispiel sind Blockcodes einfacher zu dekodieren als andere Codes. Andererseits benötigen Blockcodes deutlich mehr Speicherplatz und wesentlich längere Übertragungszeiten. Je nach Problemstellung kann man unter *wirtschaftlich* also etwas anderes verstehen.

Definition: Unter allen möglichen Codes zu einem Alphabet wird einer mit der kleinsten **mittleren Wortlänge** als **optimaler Code** bezeichnet.

Wenn die relativen Häufigkeiten der einzelnen Zeichen unterschiedlich groß sind, so sind Codes mit konstanter Wortlänge (sog. Blockcodes) unwirtschaftlich.

Definition: Die relative Häufigkeit $p_i = p(x_i)$ des Zeichens x_i eines Alphabets wird definiert als $p_i = p(x_i) = \frac{absolute\ Häufigkeit\ von\ x_i}{Anzahl\ der\ untersuchten\ Zeichen}$

Bemerkung: Für alle x_i gilt $0 \leq p(x_i) \leq 1$ und $\sum_{i=1}^{N} p(x_i) = 1$

In den folgenden 9 Aufgaben soll das Ausgangsalphabet {A, B, C} codiert werden durch das Alphabet {0, 1}. Es gibt drei unterschiedliche Kodierungen: 1, 2 und 3. Außerdem gibt es drei unterschiedliche relative Häufigkeiten der auftretenden Zeichen: a, b und c. Beispiel: die Aufgaben 1a und 1b haben dieselbe Kodierung aber unterschiedliche Häufigkeiten. Die Aufgaben 1b und 2b haben unterschiedliche Kodierungen, aber dieselben Häufigkeiten. Berechne jeweils die mittlere Codewortlänge s !

Im ersten Beispiel a ist die relative Häufigkeit gleich groß.

	f(A)	f(B)	f(C)	p(A)	p(B)	p(C)	s
1a	0	10	11	1/3	1/3	1/3	
2a	01	1	00	1/3	1/3	1/3	
3a	01	10	00	1/3	1/3	1/3	

Im zweiten Beispiel b sind die relativen Häufigkeiten unterschiedlich groß.

	f(A)	f(B)	f(C)	p(A)	p(B)	p(C)	s
1b	0	10	11	5/8	2/8	1/8	
2b	01	1	00	5/8	2/8	1/8	
3b	01	10	00	5/8	2/8	1/8	

Im dritten Beispiel c sind die relativen Häufigkeiten auch unterschiedlich groß.

	f(A)	f(B)	f(C)	p(A)	p(B)	p(C)	s
1c	0	10	11	5/8	1/8	2/8	
2c	01	1	00	5/8	1/8	2/8	
3c	01	10	00	5/8	1/8	2/8	

Aufgaben

1. Untersuche, ob alle drei obigen Codes der *Fano-Bedingung* genügen!

2. Kann ein Code für unterschiedliche relative Häufigkeiten optimal sein?

3. Kann ein Code für unterschiedliche relative Häufigkeiten unterschiedlich gut sein?

4. Können unterschiedliche Codes gleichwertig sein?

5. Was ändert sich, wenn man in den obigen Kodierungen jeweils die Nullen und Einsen vertauscht?

Eine Auswertung obiger Beispiele zeigt:

Alle obigen Codes genügen der Fano-Bedingung ⇒ sie sind dekodierbar

Derselbe Code **kann** für unterschiedliche relative Häufigkeiten optimal sein.

Derselbe Code **kann** für unterschiedliche relative Häufigkeiten unterschiedlich gut sein. Aber normalerweise ändert sich die Güte eines Codes, wenn sich die relativen Häufigkeiten ändern.

Unterschiedliche Codes können gleichwertig sein.

Ein Blockcode ist offensichtlich nicht optimal.

Vertauscht man in der Codierung die Nullen und Einsen, so erhält man zwar andere, aber gleichwertige Codes.

Ein bisher guter Code wird normalerweise schlechter, wenn sich die relativen Häufigkeiten ändern. Deshalb ändert man üblicherweise die Codierung, falls sich die Häufigkeiten ändern.

Beispiele: Ein Bild soll codiert werden. Die obere Hälfte zeigt einen überwiegend blauen Himmel, die untere eine grüne Wiese. Anstatt einen einzigen Code für das gesamte Bild zu erstellen, wäre es günstiger, für beide Bildhälften jeweils einen eigenen Code zu erstellen.

Ein Komet, dessen Helligkeit sich mit der Zeit ändert, wird beobachtet. Bei der digitalen Filmübertragung wird der Code immer dann geändert, wenn sich große Änderungen ergeben.

Die folgende Tabelle vergleicht die relativen Häufigkeiten der Buchstaben in der deutschen und englischen Sprache.

Buchstabe	Häufigkeit in %		Buchstabe	Häufigkeit in %	
	Deutsch	Englisch		Deutsch	Englisch
a	6,47	8,04	n	9,84	7,09
b	1,93	1,54	o	2,98	7,60
c	2,68	3,06	p	0,96	2,00
d	4,83	3,99	q	0,02	0,11
e	17,48	12,51	r	7,54	6,12
f	1,65	2,30	s	6,83	6,54
g	3.06	1,96	t	6,13	9,25
h	4,23	5,49	u	4,17	2,71
i	7,73	7,26	v	0,94	0,99
j	0,27	0,16	w	1,48	1,92
k	1,46	0,67	x	0,04	0,19
l	3,49	4.14	y	0,08	1,73
m	2,58	2,53	z	1,14	0,09

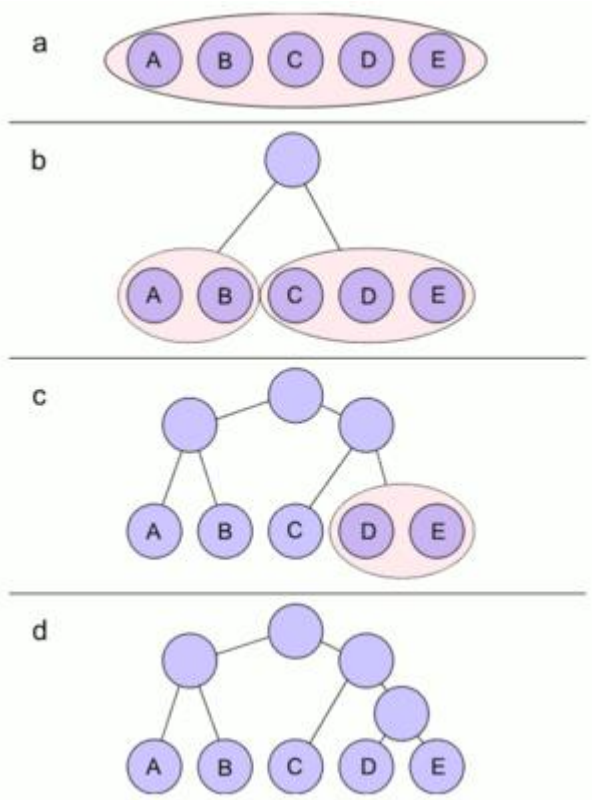
Fano-Code

Sehr häufig werden Alphabete durch Binärzahlen kodiert. Derartige Codes lassen sich dann durch Binärbäume darstellen. Jedes kodierte Zeichen entspricht einem Blatt dieses Binärbaumes.
Das Problem ist: Wie erstellt man einen möglichst guten Code bzw. Binärbaum? Dazu gibt es zwei unterschiedliche Algorithmen zur Konstruktion dieser Bäume. Der nach ***Robert Fano*** (* 1917 in Italien; † 2016 in Amerika) benannte Algorithmus arbeitet mit folgender Vorschrift:

- Sortiere die Zeichen nach fallenden Häufigkeiten (diese Sortierung ist oft nicht eindeutig)
- Teile die Zeichen **entlang dieser Reihenfolge** so in 2 Gruppen, dass die Summe der Häufigkeiten in den beiden Gruppen möglichst gleich ist (diese Unterteilung ist oft nicht eindeutig). Die beiden Gruppen entsprechen dem linken und rechten Teilbaum des zu erstellenden Baumes. Wichtig: jedes Zeichen aus der einen Gruppe hat größere oder gleiche Wahrscheinlichkeit als jedes Zeichen aus der anderen Gruppe (deswegen auch der Punkt 1).
- Befindet sich mehr als ein Zeichen in einer der entstandenen Gruppen, wende den Algorithmus rekursiv auf diese Gruppe an

Das Beispiel zeigt die Konstruktion des *Fano-Codes* für ein kleines Alphabet. Die gleichen Werte werden weiter unten auch für den Huffman-Code verwendet, damit die Ergebnisse vergleichbar sind. Die fünf zu kodierenden Zeichen haben folgende absoluten Häufigkeiten:

A	B	C	D	E
15	7	6	6	5



Sortiert sind die Werte schon, also direkt zu Schritt 2, dem Partitionieren. Am Anfang sind alle Zeichen in einer Partition (im Bild a).
Die exakte Mitte läge bei 19.5 Zeichen. 15 ist 4.5 unter dem Mittelwert, 15+7 hingegen nur 2.5 drüber. Das Alphabet wird also so in 2 Teile unterteilt, dass der eine Teil die Zeichen A und B und der andere Teil den Rest (C, D und E) enthält (Bild b). Beide Partitionen enthalten noch mehr als 1 Zeichen, müssen also weiter zerteilt werden. Die Partition mit A und B kann nur in 2 Teile mit je einem Zeichen zerlegt werden. Die andere Gruppe hat 2 Möglichkeiten. 6+6 ist weiter von der Mitte entfernt, als 6 alleine. Also wird in die 2 Partitionen mit C und D+E unterteilt (Bild c). Schließlich wird noch D und E zerteilt. Der

entstandene Entscheidungsbaum ist im Bild Abschnitt d dargestellt.

Die Bitlängen für die einzelnen Zeichen betragen also:

2 Bits für A, B und C und je 3 Bits für D und E. Das ergibt eine durchschnittliche Bitzahl von

$$\frac{2Bit * (15 + 7 + 6) + 3Bit * (6 + 5)}{39} \approx 2.28$$ Bits pro Zeichen.

Da die Kodierung, wie oben erwähnt, nicht immer eindeutig ist, hier nur ein Beispiel für eine mögliche Kodierung aufgrund dieses Baumes:

A	B	C	D	E
00	01	10	110	111

<u>Aufgaben</u>:

Erstelle jeweils den Codebaum, gib anschließend für jedes Zeichen die Kodierung an und berechne die mittlere Codewortlänge (Lösung teilweise auf Seite 11):

1. {A, B, C, D} mit p(A) = p(B) = 1/4, p(C) = 1/3 , p(D) = 1/6
2. {A, B, C, D, E} mit p(A)=2/9, p(B) = 1/9, p(C) = 1/3 , p(D) = 2/9, p(E) = 1/9
3. {A, B, C, D, E, F} mit p(A)=1/3, p(B)= p(C) = 1/6 , p(D) = p(E) = p(F) = 1/9
4. {A, B, C, D} mit p(A) = 16/25, p(B) = p(C) = 4/25 , p(D) = 1/25
5. {A, B, C, D} mit p(A) = 2/5, p(B) = p(C) = p(D) = 1/5 Zeige, dass es hier nach dem Fano-Verfahren drei unterschiedliche mögliche Kodierungen gibt!

Das Verfahren nach *Fano* funktioniert auch für ein Bildalphabet mit m Zeichen (m >= 2). In der Konstruktionsbeschreibung ist dann jeweils 2 durch m zu ersetzen.

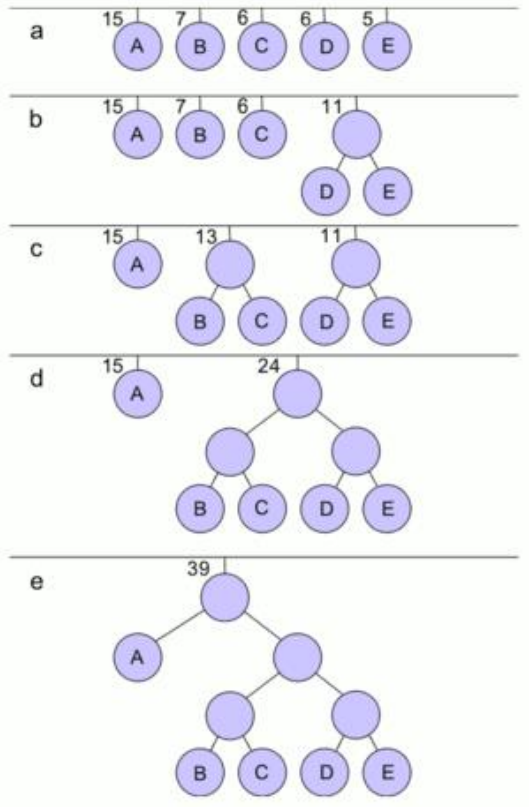
Leider ist der vom *Fano-Algorithmus* erzeugte Baum nicht immer optimal!

Huffman-Code

Da der vom *Fano-Algorithmus* erzeugte Baum nicht immer optimal ist, wurde ein Algorithmus gesucht, der immer optimale Codes liefert. *David A. Huffman* (* 09.08.1925 in Ohio, † 07.10.1999 in Kalifornien) hat ihn 1952 gefunden. **Dieser sog. Huffman-Algorithmus liefert immer einen optimalen Baum für die gegebenen Wahrscheinlichkeiten.**

Während der *Fano*-Baum von der Wurzel zu den Blättern erstellt wird, arbeitet der Algorithmus zur Konstruktion des *Huffman*-Baums in entgegengesetzter Richtung.

- Erstelle einen "Wald" mit Bäumen für jedes Zeichen. Diese Bäume enthalten zunächst nur einen Knoten: das Zeichen. Ordne diesen Wald nach fallenden Häufigkeiten.
- Suche die beiden Bäume im Wald, die für die Zeichen mit der kleinsten Wahrscheinlichkeit stehen. Entferne diese Bäume aus dem Wald. Erstelle einen neuen Baum, der die beiden entfernten Bäume als Subbaum hat. Füge diesen neuen, größeren Baum in den Wald ein. Benutze dabei die Summe der Wahrscheinlichkeiten der Unterbäume.
- Wiederhole, bis nur noch ein Baum übrig ist.



Das gleiche Beispiel wie beim *Fano*-Code führt zu folgenden Schritten.

Der Wald wird erstellt. Im Bild ist in Abschnitt a der Zustand zu sehen. Am oberen Ende sind immer die Häufigkeiten für die Bäume angezeigt, da diese vom Algorithmus benötigt werden.

Die beiden Bäume mit den geringsten Häufigkeiten sind E und C oder D. Die Bäume D und E werden entfernt, ein neuer Baum mit einer Häufigkeit von 5+6=11 wird erstellt und in den Wald eingefügt (Bild b). Allerdings sollte Bild b noch der Größe nach sortiert werden.

Als nächstes werden B und C zusammengefasst. Der neue Baum hat die Häufigkeit 13 (Bild c).

Nun werden die 2 bereits zusammengefassten Bäume ein weiteres Mal verbunden (Bild d).

Schließlich wird A mit dem Rest zu einem Baum verbunden (Bild e). Der Algorithmus ist hiermit beendet, da nur noch ein Baum im Wald vorhanden ist.

Die Codelängen für die einzelnen Zeichen sind diesmal 1 Bit für A und 3 Bit für alle anderen Zeichen. Dies führt zu einer durchschnittlichen Bitzahl von

$$\frac{1Bit * 15 + 3Bit * (7 + 6 + 6 + 5)}{39} \approx 2.23$$ Bits pro Zeichen.

Im Vergleich zu 2.28 Bits pro Zeichen, wie bei Fano, ist das eine Verbesserung.

Aufgaben

Die folgenden vier Aufgaben wurden schon einmal mit dem *Fano*-Algorithmus gelöst. Löse sie diesmal mit dem *Huffman*-Algorithmus und vergleiche anschließend (fülle die untenstehende Tabelle aus)!
Erstelle jeweils den Huffman-Codebaum, gib anschließend für jedes Zeichen die Kodierung an und berechne die mittlere Codewortlänge:

1. {A, B, C, D} mit p(A) = p(B) = 1/4, p(C) = 1/3 , p(D) = 1/6

2. {A, B, C, D, E} mit p(A)=2/9, p(B) = 1/9, p(C) = 1/3 , p(D) = 2/9, p(E) = 1/9

3. {A, B, C, D, E, F} mit p(A)=1/3, p(B)= p(C) = 1/6 , p(D) = p(E) = p(F) = 1/9

4. {A, B, C, D} mit p(A) = 16/25, p(B) = p(C) = 4/25 , p(D) = 1/25

5. {A, B, C, D} mit p(A) = 2/5, p(B) = p(C) = p(D) = 1/5

Aufgabe	s_{Fano}	$s_{Huffman}$
1	2	
2	20/9	
3	2,5	
4	1,56	
5	2	

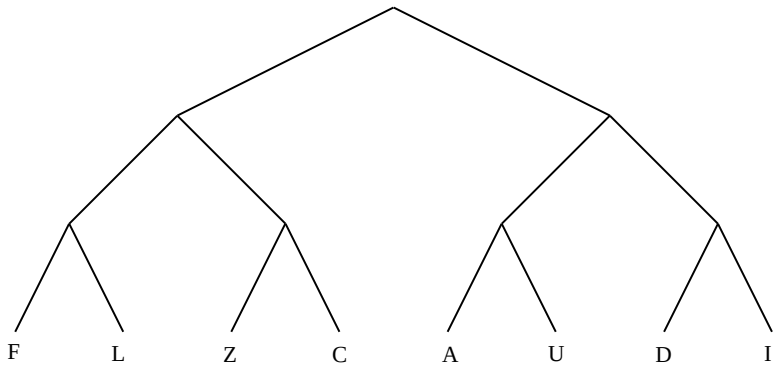
(mittlere Codewortlänge ist bei Fano und Huffman hier identisch)

Shannonsche Informationstheorie

Nach *Huffman* oder *Fano* kann die Güte des Codes erst **nach** der Codierung berechnet werden. Im Folgenden wollen wir schon **vorher** berechnen, wie gut irgendeine Kodierung überhaupt sein kann für ein Ausgangsalphabet mit gegebener Häufigkeitsverteilung.

1. Spezialfall: Betrachte ein Ausgangsalphabet mit $N = 2^n$ gleich wahrscheinlichen Elementen. Also $p_i = p(x_i) = \frac{1}{N} = \frac{1}{2^n}$

Beispiel: 8 gleich wahrscheinliche Elemente lassen sich in einer Ebene eines binären Codebaumes unterbringen. Alle Codeworte haben hier die Länge 3. Auch die mittlere Codewortlänge s hat die Länge 3.



Bemerkung: Die Codewortlänge eines Zeichens x wird (vorläufig, in diesem Spezialfall 1) auch als **Informationsgehalt H(x)** bezeichnet.
Die Einheit ist bit (basic indissoluble information unit).
Der Informationsgehalt H(x) ist in diesem Spezialfall damit auch ganzzahlig.

Wenn ein Zeichen selten vorkommt, wird es durch viele Bits dargestellt, und sein Informationsgehalt ist damit entsprechend groß.

Für unsere Bedingung (Spezialfall 1) gilt für jedes Zeichen x:

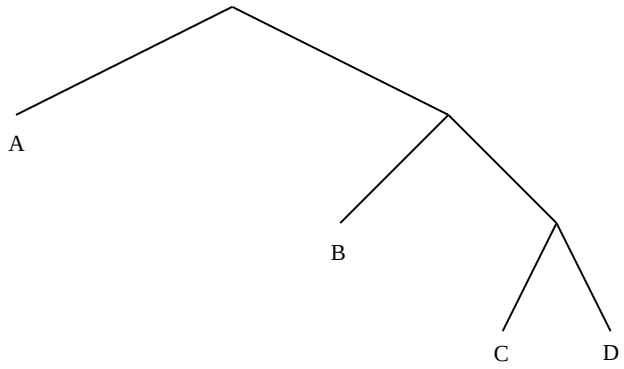
$H(x) = \log_2 N \text{ bit} = \log_2(2^n) \text{ bit} = n \text{ bit}$

2. Spezialfall: Voraussetzung: Das Alphabet lässt sich solange in paarweise gleich wahrscheinliche Teilmengen zerlegen, bis alle Zeichen isoliert sind. (schwächere Voraussetzung als im Spezialfall 1)

Auch in diesem Spezialfall 2 versteht man unter dem Informationsgehalt H(x) die Codewortlänge des Zeichens x .

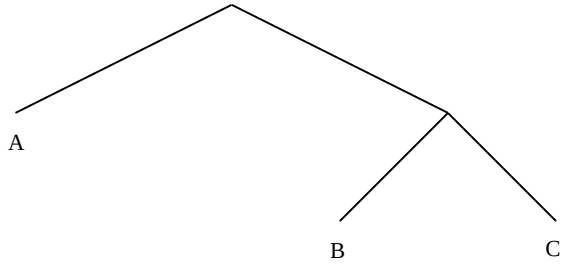
Kodiere folgende zwei Beispiele nach *Huffman* oder *Fano* !

1. {A, B, C, D} mit p(A)=1/2, p(B)=1/4, p(C)=1/8 , p(D)=1/8



Für die mittlere Codewortlänge s gilt hier: s = 1,75

2. {A, B, C} mit p(A)=1/2, p(B)=1/4, p(C)=1/4



Für die mittlere Codewortlänge s gilt hier: s = 1,5

Das zweite Beispiel zeigt übrigens, dass die Anzahl N der Zeichen keine Zweierpotenz sein muss.

Man sieht an diesen Beispielen, dass der Informationsgehalt eines Zeichens mit seiner „Seltenheit“ steigt.

Auch in diesem Spezialfall 2 ist der Informationsgehalt eines jeden Zeichens ganzzahlig.

Zwischen der Auftrittswahrscheinlichkeit p(x) eines Zeichens x und seinem Informationsgehalt H(x), der Anzahl der binären Entscheidungen bis zu seiner Auffindung, existiert ein einfacher Zusammenhang:

Nach der 1. Teilung hat jede der beiden Teilmengen die Gesamtwahrscheinlichkeit ½ , nach der 2. Teilung haben die neu entstehenden Teilmengen jeweils die Wahrscheinlichkeit ¼ , nach der k-ten Teilung noch $(½)^k$.

Wird nun ein Zeichen x_i nach k Schritten isoliert, so hat es die Wahrscheinlichkeit $p_i = p(x_i) = (½)^k$.

Umgekehrt kann man aus dieser Gleichung natürlich (interessanterweise) auch die Anzahl der Schritte k (= Informationsgehalt) bestimmen, wenn p_i bekannt ist:

$$p_i = \left(\frac{1}{2}\right)^k \;\Leftrightarrow\; \frac{1}{p_i} = 2^k \;\Leftrightarrow\; k = \log_2 \frac{1}{p_i} = -\log_2 p_i = H(x_i)$$

Beispiel: Ein Zeichen x habe die Auftrittswahrscheinlichkeit 1/8 . Dann folgt nach obiger Gleichung **(Voraussetzung: Das Alphabet lässt sich solange in paarweise gleich wahrscheinliche Teilmengen zerlegen, bis alle Zeichen isoliert sind)**, dass zu seiner Codierung drei bit notwendig sind. Das wäre dann auch sein Informationsgehalt H(x) .

Bemerkung: In diesem Spezialfall 2 (und damit natürlich auch im Spezialfall 1) ist der Informationsgehalt H(x), also die Anzahl der Binärentscheidungen, die für das Auffinden des Zeichens x mindestens nötig sind, eine natürliche Zahl.

Da alle Zeichen unterschiedlichen Informationsgehalt besitzen, berechnen wir nun den **mittleren Informationsgehalt H** eines Zeichens:

$$H = \sum_{i=1}^{N} p_i \cdot H(x_i) = -\sum_{i=1}^{N} p_i \cdot \log_2 p_i$$

Dieser Zahlenwert entspricht in diesem Spezialfall **(das Alphabet lässt sich solange in paarweise gleich wahrscheinliche Teilmengen zerlegen, bis alle Zeichen isoliert sind)** der mittleren Wortlänge s eines optimalen Codes.,

Verallgemeinerung der beiden Spezialfälle

Definition: Für ein Zeichen x_i mit der Auftrittswahrscheinlichkeit $p_i = p(x_i)$ eines beliebigen Alphabets ist $H(x_i) := -\log_2 p_i$ bit der **Informationsgehalt von x_i .**

Bemerkung: $H(x_i)$ ist im allgemeinen Fall keine natürliche Zahl mehr. Trotzdem entspricht $H(x_i)$ ungefähr der Codelänge des Zeichens x_i in einer optimalen Codierung.

Definition: Unter der **Entropie H der Nachrichtenquelle** versteht man den **mittleren** Informationsgehalt eines Zeichens.

$$H = -\sum_{i=1}^{N} p_i \cdot \log_2 p_i$$

Beispiel: Für das Alphabet in deutschen Texten (vgl. Tabelle) gilt

p(e) = 0,1470 $\Rightarrow$ H(e) $\approx$ 2,766 bit
p(w) = 0,0142 $\Rightarrow$ H(w) $\approx$ 6,138 bit
p(x) = 0,0001 $\Rightarrow$ H(x) $\approx$ 13,288 bit

Für das gesamte Alphabet einschließlich Umlaute gilt
H = 4,112 bit. Beachte, dass man dafür sonst 5 bit benötigt.

Aufgabe: Die folgenden Aufgaben wurden bereits mit dem *Fano*- und dem *Huffman*-Algorithmus kodiert. Berechne nun zusätzlich die *Entropie* H !

1. {A, B, C, D} mit p(A) = p(B) = 1/4, p(C) = 1/3 , p(D) = 1/6
2. {A, B, C, D, E} mit p(A)=2/9, p(B) = 1/9, p(C) = 1/3 , p(D) = 2/9, p(E) = 1/9
3. {A, B, C, D, E, F} mit p(A)=1/3, p(B)= p(C) = 1/6 , p(D) = p(E) = p(F) = 1/9
4. {A, B, C, D} mit p(A) = 16/25, p(B) = p(C) = 4/25 , p(D) = 1/25
5. {A, B, C, D} mit p(A) = 2/5, p(B) = p(C) = p(D) = 1/5

Aufgabe	s_{Fano}	$s_{Huffman}$	H
1	2	2	
2	20/9	20/9	
3	2,5	2,5	
4	1,56	1,56	
5	2	2	

Lösung:

Aufgabe	s_{Fano}	$s_{Huffman}$	H
1	2	2	**≈1,9591**
2	20/9	20/9	**≈2,1971**
3	2,5	2,5	**≈2,4466**
4	1,56	1,56	**≈1,4439**
5	2	2	**≈1,9219**

Zwischen einer optimalen Codierung und der Entropie des Ausgangsalphabets existiert folgender mathematischer Zusammenhang, den der Amerikaner Claude Elwood Shannon (* 30.04.1916 in Michigan, † 24.02.2001) herausfand:

Shannonsches Codierungstheorem (1948):

1. Die *Entropie* **H** ist die höchste untere Grenze für die mittleren Codewortlängen **s** aller Codes, d.h. es gilt stets $H \leq s$
2. Unter der Berücksichtigung der relativen Häufigkeiten der Einzelzeichen und ihrer Kombinationen (Zeichenketten) lässt sich die mittlere Codewortlänge s immer beliebig dicht an die *Entropie* H annähern.

Bemerkung: Oft lässt sich diese Annäherung erst bei einem Ausgangsalphabet von Zeichenketten durchführen.

er 409	ge 147
en 400	es 140
ch 242	ne 122
de 227	un 119
ei 193	st 116
nd 187	re 112
te 185	he 102
in 168	an 102
ie 163	be 101

Die häufigsten Digramme der deutschen Sprache (Häufigkeiten in %%)

ein	122	sch	66	ind	46	sse	39	nic	31
ich	111	cht	61	enw	45	aus	36	sen	31
nde	89	den	57	ens	44	ers	36	ene	30
die	87	ine	53	ies	44	ebe	35	nda	30
und	87	nge	52	ste	44	erd	33	ter	30
der	86	nun	48	ten	44	enu	33	ass	29
che	75	ung	48	ere	43	nen	32	ena	29
end	75	das	47	lic	42	rau	32	ver	29
gen	71	hen	47	ach	41	ist	31	wir	29

Die häufigsten Trigramme der deutschen Sprache (Häufigkeiten in %%)

Satz: Gegeben sei ein Alphabet mit (beliebig vielen) N Zeichen. Falls alle Zeichen gleich wahrscheinlich sind (d.h. $p_i = p(x_i) = \frac{1}{N}$), dann ist die *Entropie* H (= mittlerer Informationsgehalt eines Zeichens = mittlere Codewortlänge) maximal und es gilt $H_{max} = \log_2 N$
Für jede ungleiche Verteilung gilt: $0 \leq H \leq H_{max}$

Teilbeweis:

$$
\begin{aligned}
H_{max} &= -\sum_{i=1}^{N} p_i \cdot \log_2 p_i \\
&= -\sum_{i=1}^{N} \frac{1}{N} \cdot \log_2 \frac{1}{N} \\
&= \sum_{i=1}^{N} \frac{1}{N} \cdot \log_2 N \\
&= \frac{1}{N} \cdot \log_2 N \cdot \sum_{i=1}^{N} 1 \\
&= \frac{1}{N} \cdot \log_2 N \cdot N \\
&= \log_2 N
\end{aligned}
$$

Redundanz

Im allgemeinen Sprachgebrauch versteht man unter *Redundanz* mehrfach vorkommende, und damit teilweise überflüssige Information. In Konferenzen nerven insbesondere redundante Gesprächsbeiträge.

Redundanz kann allerdings manchmal durchaus erwünscht sein. Zum Beispiel sind Physikstudenten sehr erfreut, wenn in einem Lehrbuch ein bestimmter Sachverhalt mehrmals auf unterschiedliche Art und Weise erläutert wird. Physik-Experten finden weniger Gefallen an redundanten Lehrbüchern.

Mehrfach vorhandene Informationen (zum Beispiel Kurslisten beim Kurslehrer, beim Jahrgangsstufenleiter und im Sekretariat) haben Vor- und Nachteile:

- nicht alle Daten / Dateien sind aktuell,
- manchmal wird auf die falschen Daten zugegriffen
- neue Informationen müssen an verschiedenen Stellen eingetragen werden
- hat der Kurslehrer seine Liste verloren, kann er sie mit den Informationen vom Jahrgangsstufenleiter oder vom Sekretariat wieder herstellen.

In Bezug auf Informatik und Daten bedeutet Redundanz, dass mehr Daten vorhanden sind, als eigentlich zur Übermittlung bzw. Speicherung der Informationen unbedingt notwendig wären.

Beispiel: der folgende Satz aus der deutschen Sprache ist eindeutig zu verstehen, obwohl einige Buchstaben fehlen:
„In de Somerferin werd ic nac Spanin fliegn.“
Verglichen mit dem richtigen deutschen Satz *„In den Sommerferien werde ich nach Spanien fliegen“* fehlen also 8 Buchstaben. Der richtige deutsche Satz enthält insgesamt 43 Buchstaben. 8 Buchstaben scheinen also zum Verständnis überflüssig zu sein. In diesem willkürlich gewählten Beispiel entspricht das immerhin $8/43 \approx 19\%$. Wahrscheinlich sind auch noch mehr Buchstaben überflüssig, denn den weiter verkürzten Satz wird man auch noch verstehen: *„In Somerferin ic nac Spanin flign.“*

Allgemein gilt allerdings auch: je größer die Redundanz einer Nachricht, desto sicherer ist die eigentliche Nachricht. Wenn z.B. bei großer Redundanz ein paar Buchstaben fehlen, so ist das für das Gesamtverständnis nicht weiter tragisch.

Andererseits gilt: Je kleiner die Redundanz, desto weniger Speicherplatz benötigt die Nachricht und desto schneller kann sie übermittelt werden.

Bei **normalen** Binärzahlen ist offensichtlich überhaupt keine Redundanz vorhanden. Beispiel: %1101 = dez 13 und %1001 = dez 9
An diesem Beispiel erkennt man: wenn auch nur ein einziges Bit fehlen oder falsch übertragen sein sollte, so ist die eigentliche Nachricht nicht mehr rekonstruierbar.

Der Begriff *Redundanz* ist qualitativ hinreichend erklärt. Wir wollen nun versuchen, Redundanz auch quantitativ zu messen. Das wird bei der deutschen Sprache sehr schwierig sein. Deshalb beschränken wir uns auf die Untersuchung von Kodierungen.

Ist eine bestimmte Kodierung sehr redundant, d.h. hätte man diese Kodierung auch wesentlich kürzer vornehmen können?

Natürlich ist klar, dass die Entropie der Nachrichtenquelle (d.h. des Ausgangs-alphabets) und (nach der Kodierung) die mittlere Codewortlänge s miteinander verglichen werden müssen.

Definition: Die **absolute Redundanz R** eines Codes ist gleich der Differenz aus mittlerer Codewortlänge und Entropie des Ausgangsalphabets:
$R := s - H$
Die **relative Redundanz** ist der Quotient aus der absoluten Redundanz und der mittleren Codewortlänge:
$\bar{R} := \frac{s-H}{s} = 1 - \frac{H}{s}$

Betrachte das Beispiel: Alphabet = {A, B, C, D, E, F, G},
p(A) = 0,3; p(B) = p(C) = 0,2; p(D) = p(E) = 0,1; p(F) = p(G) = 0,05

Die Entropie beträgt hier: $H = -\sum_{i=1}^{N} p_i \cdot \log_2 p_i = \cdots \approx 2{,}546$

a) Bei einer Huffman-Codierung würde sich s = 2,6 ergeben.
Damit folgt für die Huffman-Codierung: $R \approx 0{,}054$ und $\bar{R} \approx 0{,}02 = 2\%$

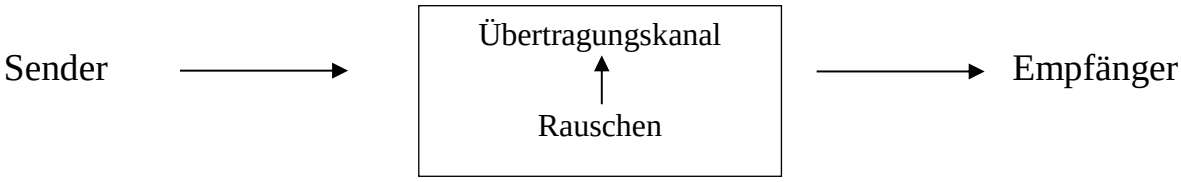
b) Als Blockcode würde man für die 7 Buchstaben A bis G einen 3-Bit-Code benötigen.
Damit folgt für einen Blockcode:
$R \approx 3 - 2{,}546 \approx 0{,}454$ und $\bar{R} \approx \frac{0{,}454}{3} \approx 0{,}15 = 15\%$

Welche Bedeutung hat nun der Wert der (relativen) Redundanz?

Je kleiner die (relative) Redundanz einer Kodierung ist, desto besser (bezüglich der Codewortlängen) ist diese Kodierung.
Allerdings lassen sich mögliche Übertragungsfehler auch umso schlechter korrigieren!

Fehlerkorrigierende Codes

Jede Weitergabe von Nachrichten kann beschrieben werden durch



Dabei versteht man unter *Rauschen* eine Fehlererzeugung durch physikalische Nebeneffekte.

Man kann z.B. auch stark gestörte deutsche Texte verstehen, weil sehr viele Buchstabenkombinationen unmöglich sind. Beispiel:

Gmäeß eneir Sutide eneir elgnihcesn Uvinisterät, ist es nchit witihcg in wlecehr Rneflogheie die Bstachuebn in eneim Wort snid, das ezniige was wcthiig ist, ist dass der estre und der leztte Bstabchue an der ritihcgn Postoiin snid. Der Rset knan ein ttoaelr Bsinöldn sien, tedztorm knan man ihn onhe Pemoblre lseen. Das ist so wiel wir nciht jeedn Bstachuebn enzelin leesn, snderon das Wrot als gseatems.

Nach obigem Beispiel scheint die deutsche Sprache also gegenüber Störungen relativ unempfindlich zu sein. Betrachte aber auch das folgende Beispiel:

Sender: „ich komme heute nacht“
Empfänger1: „ich kom**n**e heut**t** nacht“
Empfänger2: „ich komme heute n**i**cht“

Der Empfänger Nr. 1 wird die Nachricht trotz zweier Übertragungsfehler richtig verstehen. Der Empfänger Nr. 2 wird die Nachricht falsch verstehen, obwohl bei ihm nur ein einziger Übertragungsfehler auftrat.

Wieder scheint die deutsche Sprache zu kompliziert zu sein für die Betrachtung von nachrichtentechnischen Übermittlungsfehlern. Wir befassen uns deshalb mit Sicherheitsaspekten von Übertragungen von „simplen“ Binärcodes.

Hamming-Code

Definition: Gegeben sei ein binärer Blockcode. Der **Hamming-Abstand** d zweier unterschiedlicher binärer Codewörter w_1, w_2 ist die Anzahl der Stellen, um die sie sich unterscheiden.

Beispiele:

$w_1 = 11000$	$w_1 = 11000$
$w_2 = 11001$	$w_3 = 00110$
---	---
$d(w_1,w_2) = 1$	$d(w_1,w_3) = 4$

Definition: Das Minimum aller Hamming-Abstände eines Codes wird als der **Hamming-Abstand** h des Codes bezeichnet.

Satz: Alle Verfälschungen von Codewörtern, die <u>weniger</u> als $\frac{h}{2}$ bits betreffen, können <u>eindeutig</u> korrigiert werden.
(ohne Beweis)

Betrachte folgendes Beispiel im Zusammenhang mit dem letztgenannten Satz:
Alphabet = {A, B}, $f(A) = w_1 = 10$, $f(B) = w_2 = 01$
gesendet: $w_1 = 10$, empfangen: $w = 11$, korrigiert zu?

Beispiel: Alphabet = {A, B, C, D}
$f(A) = w_1 = 11000$
$f(B) = w_2 = 00110$
$f(C) = w_3 = 10011$
$f(D) = w_4 = 01101$

Man zeigt leicht, dass $3 \leq d(w_i,w_k) \leq 4$ für alle i, k und $i \neq k \Rightarrow h = 3$

Alle Fehler, die nur 1 Bit betreffen, können in diesem Beispiel eindeutig korrigiert werden. Denn sie haben zum richtig gemeinten Wort den Abstand 1, zu allen anderen Worten mindestens den Abstand 2.

Nachteil: Für diese vier Buchstaben benötigt man eine 5-Bit-Darstellung!

Aufgaben

1. Untersuche die Codierung f mit dem Ausgangsalphabet {A, B} und f(A) = 000; f(B) = 111.

2. Konstruiere einen minimalen Blockcode für das Ausgangsalphabet {A, B, C, D}, so dass der Code den *Hamming-Abstand* 2 hat.

3. {a, b, c, d, e, f, g, h};
p(a) = p(b) = ¼ ; p(c) = p(d) = 1/8; p(e) = p(f) = p(g) = p(h) = 1/16.

a) Bestimme den Informationsgehalt für alle Zeichen. Berechne anschließend auch die Entropie dieses Alphabets!

b) Führe eine optimale Codierung durch, zeichne den Codebaum, gib für jedes Zeichen die Codierung an und berechne die relative Redundanz des Codes!

c) Erstelle einen Blockcode, bei dem die natürliche Reihenfolge erhalten bleibt. Zeichne den Codebaum, gib für jedes Zeichen die Codierung an und berechne die relative Redundanz des Codes!

4. Führe alle Teilaufgaben von Aufgabe 3 auch für das folgende Beispiel durch:
{a, b, c, d, e, f}; p(a) = 1/3; p(b) = p(c) = 1/6; p(d) = p(e) = p(f) = 1/9.

5. Bestimme für den Text „einfuehrung in die codierungstheorie“ das dazugehörige Alphabet mit den Auftrittswahrscheinlichkeiten (ohne Leerzeichen) und wende Aufgabe 3 an!

Lösungen

1. Der Hamming-Abstand beträgt offensichtlich 3, d.h. 1-Bit-Fehler können korrigiert werden.

2. Wähle z.B.
A = 000,
B = 011,
C = 110,
D = 101
Ein Hamming-Abstand von 2 ist allerdings nicht sinnvoll, weil dann noch nicht einmal 1–Bit-Fehler korrigiert werden können.

3.a)

x_i	$p(x_i)$	$H(x_i)$	$p(x_i) \cdot H(x_i)$
a	0,2500	2	0,500
b	0,2500	2	0,500
c	0,1250	3	0,375
d	0,1250	3	0,375
e	0,0625	4	0,250
f	0,0625	4	0,250
g	0,0625	4	0,250
h	0,0625	4	0,250
		Entropie:	2,75

b) Eine Kodierung nach Huffman liefert eine mittlere Wortlänge s = 2,75
Damit folgt eine absolute (und relative) Redundanz R = 0

c) Ein Blockcode für die acht Buchstaben hat die Länge 3. Damit folgt die absolute Redundanz R = 3 – 2,75 = 0,25 und die relative Redundanz
$\bar{R} = \frac{3-2,75}{3} = \frac{1}{12} = 8,\bar{3}\%$

Gray-Code

Ausgangspunkt für diesen (zwar nicht fehlerkorrigierenden aber fehlervermeidenden) Code ist das folgende Problem: Auf mehreren Adern einer elektrischen Datenleitung sollen Daten parallel übertragen werden. Theoretisch ändern sich die Bits bei einem neuen Messwert auf jeder betroffenen Leitung exakt gleichzeitig, und zwar sowohl am Eingang der Leitung als auch am Ausgang. Tatsächlich aber ändern sich die Bits auf der Leitung nicht gleichzeitig. Das kann verschiedene Ursachen haben: Unterschiedliche Qualität der verschiedenen Bauteile (z.B. der Leitungsdicke, elektrischer Widerstand, Kapazität), unterschiedliche Laufzeiten der Daten durch die Bauteile, Asymmetrien usw. Dadurch kommt es zu ungewollten Zwischenzuständen:

Beispiel: Auf einer 4-adrigen Leitung werden Zahlen von dez 0 bis dez 15, also binär von %0000 bis %1111 übertragen. Angenommen, der anzugebene Wert ändert sich um eine einzige Einheit von %0111 auf %1000. Falls das höchstwertige Bit sich schneller ändern sollte als die anderen Bits, so wird zwischenzeitlich der Wert %1111 übertragen. Das kann je nach Anwendungsproblem (z.B. Überwachung der Temperatur in Kernkraftwerken) zu äußerst ernsten Konsequenzen führen.

Dieses Problem (ungleiche Übertragungsgeschwindigkeit von einzelnen Bits) tritt grundsätzlich immer und überall auf. In Computern wird es dadurch gelöst, dass man vor jeder Weiterreichung von Zahlen immer eine gewisse Zeit (etwa 1 µs oder 1 ns, siehe Taktfrequenz!) wartet, bis sich die endgültige Zahl auch mit Sicherheit eingestellt hat. Allerdings geht dadurch sehr viel Zeit verloren, die man sich in manchen Situationen nicht leisten kann. Wenn sich in einem Kernkraftwerk die Temperatur schlagartig erhöht, würde ein Zeitverlust von einer µs schon zur Katastrophe führen.
Auch bei automatisch eingeleiteten Bremsvorgängen bei Autos kommt es sehr auf eine minimale Zeitreaktion an.

Es gibt viele physikalische Anwendungen, in denen entweder gezählt werden muss (z.B. Umdrehungen pro Sekunde) oder sich stetig ändernde Messwerte (z.B. Temperaturen) weiter geschickt werden müssen. Das heißt, hier ändern sich die zu übertragenden Werte grundsätzlich nur um den Summanden 1.

Aus diesem Grund sucht man eine Binärdarstellung von natürlichen Zahlen, bei der sich zwei aufeinanderfolgende Zahlen nur an einer einzigen Stelle (also nur in einem Bit) unterscheiden.
Derartige Darstellungen von Binärzahlen sind sehr einfach zu konstruieren. Es ist jedoch oft auch sehr wichtig, dass man diese Darstellung sehr einfach und vor allem sehr schnell wieder dekodieren kann.

Eine Lösung dieses Problems ist der sog. ***Gray-Code*.** Hierbei unterscheiden sich benachbarte Codewörter nur in einer einzigen binären Ziffer. Die Hamming-Distanz aller benachbarter Codewörter ist somit 1. Der Code ist nach dem Physiker *Frank Gray* benannt, welcher 1953 das Patent auf dieses Verfahren erhielt.

Die folgenden Punkte zeigen, wie man Schritt für Schritt aus einer normal binär codierten Dezimalzahl (Binärcode) eine Gray-codierte Binärzahl erhält:

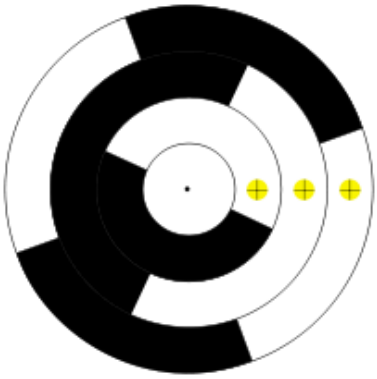
- x_1: Dualzahl im Binärcode
- x_2: Rechts-Shift der Dualzahl um 1 Bit
- x_3: XOR-Verknüpfung von x_1 und x_2; dies ist die gewünschte Zahl im Gray-Code.

Beispiel: $x_1 = 0101$ *//= dez 5*
$x_2 = 0010$ *//shift right*
$x_3 = 0111$ *//Gray-Code*

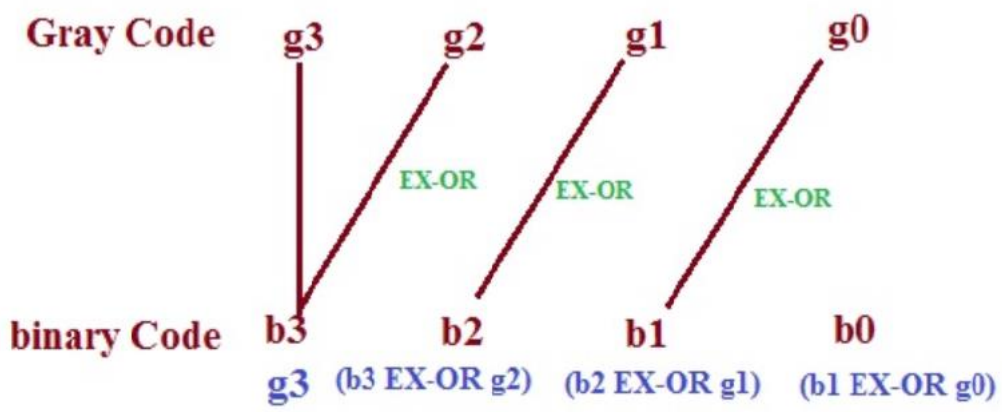
4-Bit-Gray-Code:

```
 0 = 0000
 1 = 0001
 2 = 0011
 3 = 0010
 4 = 0110
 5 = 0111
 6 = 0101
 7 = 0100
 8 = 1100
 9 = 1101
10 = 1111
11 = 1110
12 = 1010
13 = 1011
14 = 1001
15 = 1000
```

Die drei Punkte über der abgebildeten Scheibe entsprechen Laserstrahlen, deren Reflektion (oder auch Durchlässigkeit) an der sich drehenden Scheibe gemessen wird. Es entstehen so nacheinander die Zahlen von 0 bis 7 im Gray-Code.



Dekodierung des Gray-Codes:



Beispiel: Die Zahl 5 wird im Gray-Code so dargestellt: 0111
also: $g_3 = 0$, $g_2 = 1$, $g_1 = 1$ und $g_0 = 1$

Dann folgt: $b_3 = g_3 = 0$,
$b_2 = b_3$ XOR $g_2 = 0$ XOR $1 = 1$,
$b_1 = b_2$ XOR $g_1 = 1$ XOR $1 = 0$,
$b_0 = b_1$ XOR $g_0 = 0$ XOR $1 = 1$

also $b_3b_2b_1b_0 = 0101$ = dez 5