INCO – Zusammenfassung

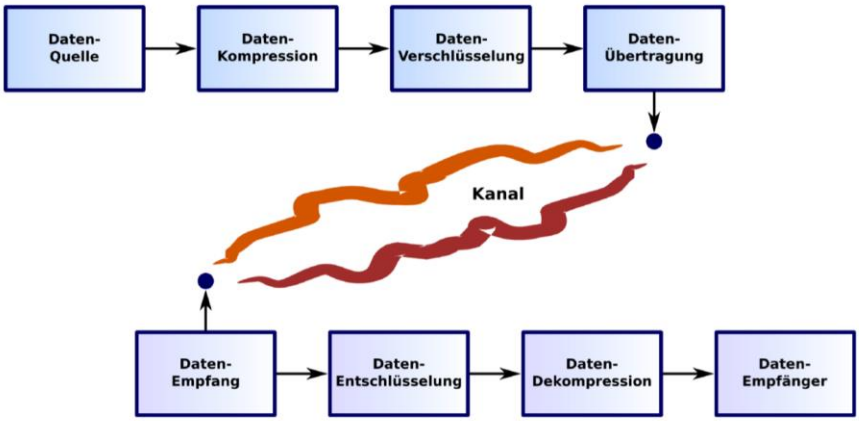


Abbildung 3: Schema eines allgemeinen Kommunikationssystems.

10-er System	2-er System	8-er System	16-er System
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

BCD-Ziffer	Dezimalziffer
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	ungültig
...	ungültig
1111	ungültig

Information

- Information ist irgendwie mitgeteiltes, erlangtes oder gespeichertes Wissen.
- Information erweitert das Wissen, resp. hebt Unwissen auf.
- Information hat einen Nutzen und einen Wert

Information ist insofern ein abstrakter Begriff. Der Informationsgehalt einer Nachricht hängt direkt mit dem Überraschungseffekt der Nachricht zusammen. Bsp: Wenn wir im Lotto gewinnen, hat dies einen sehr grossen Überraschungseffekt, daher hat diese Nachricht einen grossen Informationsgehalt.

Informationsgehalt von Ereignissen

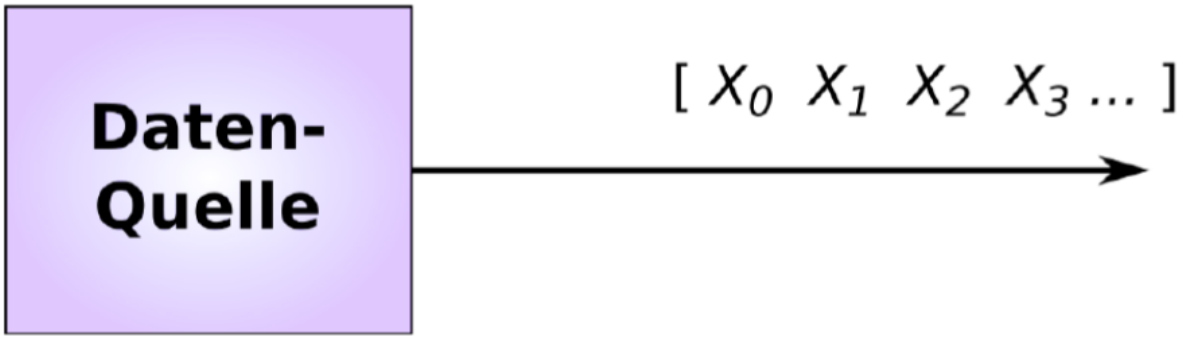


Abbildung 4: Modell einer Datenquelle.

Eine Quelle verschickt eine Nachricht. Diese Nachricht wird als Symbol gekennzeichnet. Bei uns ein grosses X und eine fortlaufende Nummer. Jedes Quellensymbol ist eine Zufallsvariabel und kann einen Wert annehmen. Jeder dieser Werte hat eine gewisse Wahrscheinlichkeit. Diese Werte mit einem kleinen x und einer fortlaufenden Nummer beschrieben.

Beispiel Würfeln:
X0 = erster Wurf -> x2 = 2
X1 = zweiter Wurf -> x3 = 3
Geschrieben als: [X0 = x2, X1 = x3, …]

Die Wahrscheinlichkeit für das Eintreten eines bestimmten Ereignis wird mit P(xn) gekennzeichnet.

P(x) = …

Informationsgehalt

Wenn nun das Ereignis Xk = xn auftritt, so können wir dessen Informationsgehalt I(xn) angeben:

$$I(x_n) = \frac{1}{P(x_n)}$$

Der Informationsgehalt ist der Kehrwert der Auftretenswahrscheinlichkeit (des Überraschungseffekts).

Wenn wir den Informationsgehalt in Bit angeben müssen, dann muss folgende Formel angewendet werden.

$$I(x_n) = \log_2 \frac{1}{P(x_n)} \qquad (\text{Bit})$$

Erwartungswert:

Als Erwartungswert bezeichnen wir jenen Wert, den wir als Mittelwert aus einer sehr grossen Zahl von Ereignissen erhalten.

E(X) = $\Sigma(Sk) * P(Sk)$
Bsp: $11 * \frac{1}{16} + 22 * \frac{1}{16} + 33 * \frac{1}{16} + 44 * \frac{1}{16} + 12 * \frac{1}{8} + 13 * \frac{1}{8} + 14 * \frac{1}{8} + 23 * \frac{1}{8} + 24 * \frac{1}{8} + 34 * \frac{1}{8} = 21.875$

Edge-Cases

1. Alle Ereignisse xn haben die gleiche Auftretenswahrscheinlichkeit -> P(xn) = 1 / n. So haben alle Ereignisse denselben Informationsgehalt

$$I(x_n) = \log_2 N$$

Beachte, dass wir bei einer derartigen QUelle, und wenn N eine Potenz von 2 ist, genauso viele Bits brauchen nämlich Log2(N) um alle mögliche Ereignisse xn binär zu nummerieren.

1. Der **minimale Wert I(min)** der Information tritt auf bei einem Ereignis xn, das praktisch sicher auftritt, das also eine Wahrscheinlichkeit P(xn) →1 hat.
2. Im Gegensatz dazu erhalten wir den **maximalen Wert I(max)** der Information für ein Ereignis xn, das sozusagen nie auftritt, das also eine Auftretenswahrscheinlichkeit P(xn) →0 hat.

statistisch unabhängige Mehrfach-Ereignisse

Eine Quelle, welche statistisch unabhängige Mehrfach-Ereignisse liefert, nennt man auch DMS-Quelle (diskrete Symbole, memory-less, source). Eine solche Quelle hat kein Gedächtnis und die Ereignisse sind voneinander unabhängig wie beim Werfen eines Würfels.

Die Verbundswahrscheinlichkeit für statisch unabhängige Mehrfach-Ereignisse ist wie folgt definiert:

$$P(x_n, x_m) = P(x_n) \cdot P(x_m)$$

Für statistisch unabhängige Mehrfach-Ereignisse ist der Informationsgehalt wie folgt definiert:

$$I(x_n, x_m) = I(x_n) + I(x_m)$$

$$I(x_n, x_m) = \log_2 \frac{1}{P(x_n, x_m)} \quad \text{(Bit)}$$

Bemerkung: Wenn alle Ereignisse xn dieselbe Auftretenswahrscheinlichkeit haben, dann bezeichnet I(xn) gerade die notwendige Anzahl Bits, mit denen sich alle xn durchnummerieren, resp. codieren lassen (bpsw. x0 = (00)).

Entropie

- Eine Datenquelle hat eine tiefe Entropie, wenn das nächste Ereignis (Symbol) mit hoher Wahrscheinlichkeit korrekt vorhersagbar ist.
- Und eine Quelle hat eine hohe Entropie, wenn die Wahrscheinlichkeit klein ist, dass das nächste Ereignis korrekt vorhergesagt werden kann.

Wir sprechen von einer tiefen Entropie bei einer kleinen Unordnung der Nachricht (z.B. 222333444555666) und von einer hohen Entropie bei einer grossen Unordnung der Nachricht (z.B. 174629456109037).

Die Entropie H(X) einer Quelle X ist der Erwartungswert der Information I(xn) dieser Symbole (Nachricht). Der Erwartungswert der Information ist definiert als das Produkt der Auftretenswahrscheinlichkeit P(xn) mit dem Informationsgehalt des Ereignisses I(xn).

$$H(X) = \sum_{n=0}^{N-1} P(x_n) \cdot I(x_n)$$

$$\boxed{H(X) = \sum_{n=0}^{N-1} P(x_n) \cdot \log_2 \frac{1}{P(x_n)}} \qquad (\text{Bit/Symbol})$$

Die Einheit der Entropie ist dieselbe wie jene der Information. Um anzuzeigen, dass es sich bei der Entropie aber um einen Mittelwert der Information pro Symbol handelt, verwenden wir die Einheit *Bit/Symbol*.

Die Entropie sagt aus, wie viel Information uns eine Zufallsvariable dieser Quelle durchschnittlich liefert.

Evtl. binäre Entropiefunktion hinzufügen

Hmin = 0 Bit/Symbol
Hmax = log2 (n) Bit/Symbol -> Diese Fall tritt genau dann auf, wenn jedes der N Symbole xn die gleiche Auftretenswahrscheinlichkeit P(xn) = $\frac{1}{N}$hat.

Mehrfach-Quellen

$$\boxed{H(X,Y) = \sum_{n,m} P(x_n, y_m) \cdot \log_2 \frac{1}{P(x_n, y_m)}} \qquad (\text{Bit/Symbol})$$

$$\boxed{H(X,Y) = H(X) + H(Y)}$$

Evtl. auf bedingte Wahrscheinlichkeit noch eingehen! (siehe Anhang im Skript zur Informatiomstheorie)

Redundanz

Redundanz ist definiert als Differenz der mittleren Codewortlänge L und Entropie H einer Quelle.

R = L - H

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

Quellcodierungs-Theorem

Solange die Redundanz R eines Codes grösser als null ist, kann verlustfrei komprimiert werden. Falls R ≤ 0, so kann nur noch verlustbehaftet komprimiert werden.

Lauflängencodierung

Idee: Eine Sequenz wird durch einen Token ersetzt. Ein Token ist definiert in der Form (M, L, Z). M steht für Marker, L für die Länge der Sequenz bzw. des Runs und Z für das Symbol, das im Run wiederholt wird. L ist immer eine binäre Zahl fester Zählerbreite, damit der DEcoder weiss wie viele Zeichen er lesen muss. Die Breite des Runs soll so gewählt werden, dass der Grossteil der Sequenzen abgedeckt werden kann. Bspw. für einen Run von 40 bis 130 Zeichen, bietet sich eine Zählerbreite von 7 Bits an, da 2^7 -1 = 127 Zeichen für einen Run ermöglicht. 8 Bits wären zuviel, da dann zuviel Speicherplatz verschwendet.

- Definiere vorgängig einen Marker, resp. ein gültiges Zeichen, das in den zu verarbeitenden Texten selten vorkommt. -> Wichtig ist, dass der Marker ein Zeichen ist, welcher auch effektiv vorkommt.

- Definiere vorgängig eine Zählerbreite (in Bits), so dass Runs der typischen Länge damit erfasst werden können. -> Da Ziffern in den Zeichen ursprünglich nicht vorhanden sind, müssen wir die Wiederholungszahl in der binären Schreibweise (fix am Anfang) angeben. So können wir z.B. 6 Bits nehmen, damit lassen sich Runs bis zu einer Länge von 63 darstellen.

- Ersetze im Urtext jeden Run durch einen Token. Ist der Run länger, als was der Token abbilden kann, so bilde mehrere Token hintereinander.

- Ersetze im Urtext alle verbleibenden Marker-Zeichen durch einen Token mit der Run-Länge eins.

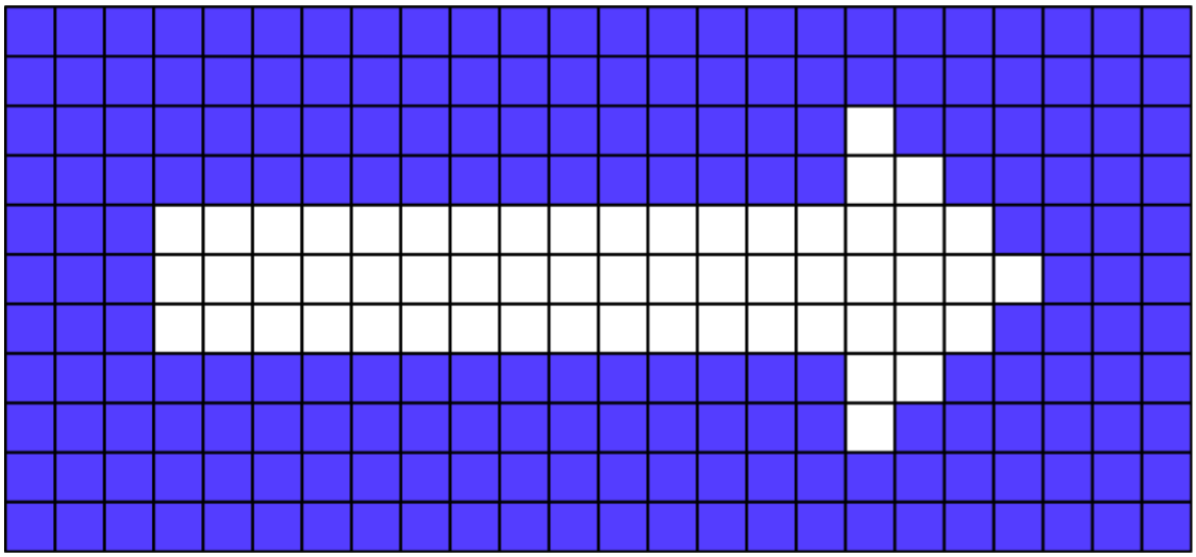
→ Der Decoder sucht lediglich nach Markern und expandiert die betreffenden Token.

```
...TERRRRRRRRRMAUIIIIIIIIIIIIIIIIWQCSSSSSSSSSSL...
->
...TEA09RMA01AUA17IWQCA10SL...
```

Bilder (serialisiert)

- Definiere vorgängig die Bitbreite des Pixelzählers.
- Definiere vorgängig die Pixelfarbe mit der begonnen wird.
- Serialisierung des Bildes -> Man reiht alle Pixel aneinander
- Falls der erste Pixel nicht die vordefinierte Farbe hat, setze die erste Pixelzahl auf null.
- Zähle jeweils alle Pixel gleicher Farbe bis zum nächsten Farbwechsel.
- Sollte eine Pixelzahl grösser werden als das Maximum des Zählers, so teile die Zahl auf: zuerst kommt die maximal erlaubte Zahl, dann eine null und dann noch die restliche Zahl.

Beispiel:



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

- 24 und 11 stehen hier für die Dimension des Bildes (24 x 11 Pixel → total 264 Bits)
- Die Bitbreite beträgt 6 Bits -> 63 Bit → Daher stossen wir bei der ersten Angabe von schwarzen Pixels bereits auf ein Problem. Die Zahl 65 muss umgeschrieben werden.

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

- Wir erhalten nun 19 Werte an je 6 Bits, dementsprechend beträgt unser Bild gesamthaft 114 Bits.
- Nun kann man noch die Kompressionsrate R (wichtig! hat nichts mit der Redundanz zu tun!) **R = neue Bits / alte Bits**

→ R = 114 / 264 ≈ 0.43

Bilder (zeilenweise)

Es besteht auch die Möglichkeit, dass man Bilder zeilenweise komprimiert. Wichtig ist dabei, dass man jeweils den Maximalwert der Bitbreite für den Zeilenumbruch reserviert. Danach ist das Verfahren identisch, jedoch muss die Bilddimension nicht mehr angegeben werden.
Beispiel:

- Wir nehmen nochmals das Bild von oben und nehmen die Bitbreite 5 (damit lassen sich Zahlen von 0 bis 31 darstellen)

→

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

- Wir erhalten nun 36 Einträge à 5 Bits = 180 Bits

→ R = 180 / 264 ≈ 0.68

Algorithmus des Decoders:
(a) Versuche, das nächste Zeichen (x Bit) zu lesen. Falls kein Zeichen gelesen werden kann: → Ende.
(b) Falls das gelesene Zeichen kein Marker M = Y ist, Gib das gelesene Zeichen aus.
Dann: → (a)
(c) Lies den Zähler L (x Bit). Falls Lesen nicht möglich: → Fehler.
(d) Lies das Zeichen Z (x Bit). Falls Lesen nicht möglich: → Fehler.
(e) Gib das Zeichen Z genau L mal aus.
Dann: → (a)

Huffman-Codes

Huffman-Codes haben die Eigenschaft, dass Sie präfixfrei sind.
“Präfixfrei”: Kein Codewort ist eine Vorsilbe eines anderen Codeworts.

Prinzip:
häufige Symbole erhalten kurze Codeworte
seltene Symbole erhalten längere Codeworte

Verfahren:

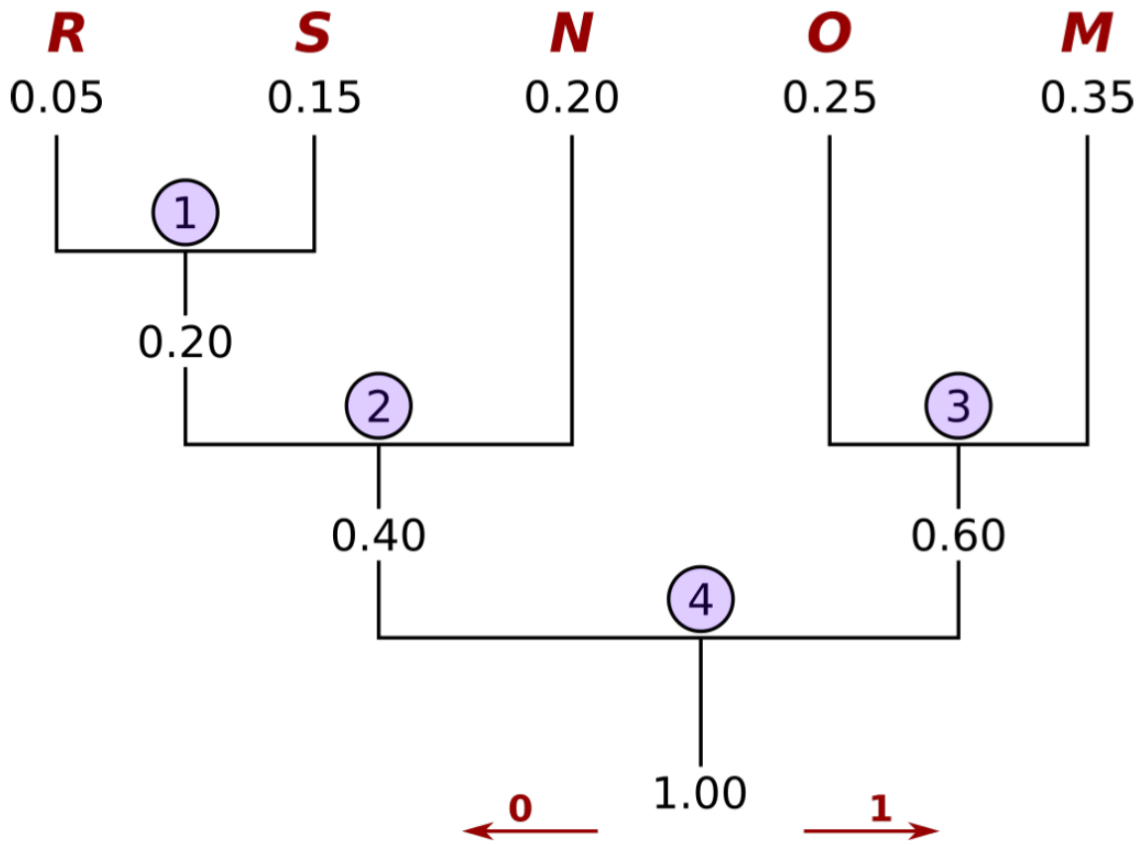


Abbildung 3: Huffman-Baum für die Quelle Ψ.

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

Lempel Ziv Codes

Das LZ-Verfahren basiert auf eine Art Wörterbuch. Dieses Wörterbuch wird während dem Kompressionsverfahren aufgebaut. Daher ist dieses Verfahren nicht sofort effizient, sondern erst bei einer sehr grossen Datenmenge. Man nennt dies auch *asympotisch optimal*.

> Jede Sequenz von Quellensymbolen wird durch einen Index in ein Wörterbuch ersetzt.

Dazu ist zu sagen, dass die Indizes eines LZ-Verfahrens immer gleiche Länge haben, wogegen die Länge der Symbol-Sequenzen im Verlauf des Prozesses ständig zunimmt. Die LZ-Verfahren arbeiten typischerweise Byte-basiert. Aus diesem Grund werden wir sie anhand von ASCII-Texten illustrieren. Beachte aber, dass die Methoden nicht auf Texte oder Bytes beschränkt sind.

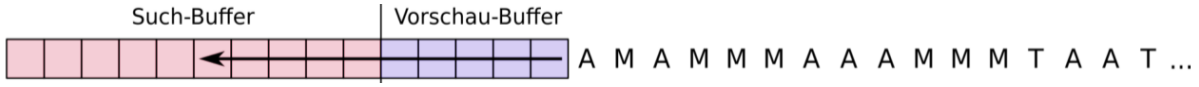
LZ77

Das LZ77-Verfahren verwendet ein sogenanntes Sliding Window. Das heisst, der zu komprimierende String wandert durch einen Buffer, der in diesem Fall in zwei Bereiche aufgeteilt ist, nämlich in den Vorschau-Buffer und den Such-Buffer.

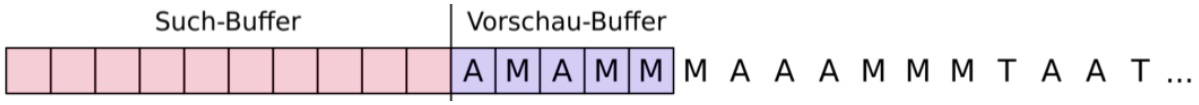
Token: (Offset, Länge, Zeichen)

- Dabei bezeichnet der *Offset* die Distanz zwischen dem **ersten Zeichen links im Vorschau-Buffer und dem Ort der besten Übereinstimmung im Such-Buffer**.
- Die *Länge* gibt an, wieviele Zeichen übereinstimmen. Und das Zeichen ist ein zusätzliches Zeichen, das im Token mit codiert wird.
- Dieses zusätzliche *Zeichen* ist nötig, damit das Verfahren nicht stecken bleibt, wenn es keine Übereinstimmung gibt

Beispiel-Verfahren-Codierung:

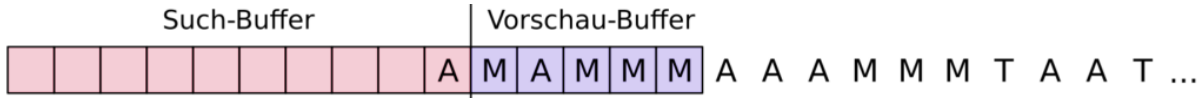


1. Fülle den Vorschau-Buffer mit Zeichen auf.



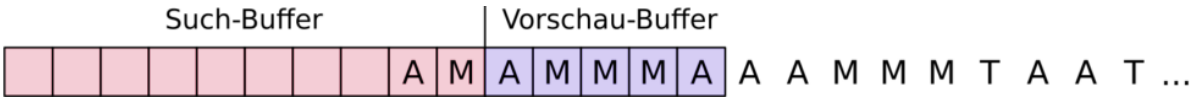
2. Da der Such-Buffer noch leer ist, gibt es keine Übereinstimmung, setze das zu übermittelnde Zeichen in den Such-Buffer

→ Token: (0, 0, A)



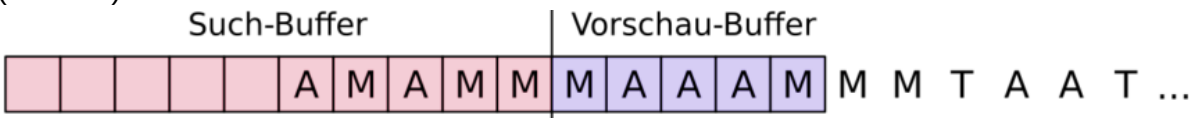
3. Überprüfe ob es im Vorschau- und Such-Buffer eine Übereinstimmung gibt, setze das zu übermittelnde Zeichen in den Such-Buffer

→ Token: (0, 0, M)



4. Wiederhole Punkt 3, solange bis das letzte Zeichen sich im Vorschau-Buffer befindet

→ Token: (2, 2, M) -> Vergleiche Bild in Punkt 3: an der zweiten Stelle (offset) gibt es eine Übereinstimmung von 2 (Länge) und zusätzlich kann noch ein “M” übermittelt werden (Zeichen)



Wichtig! Es muss auch bei der letzten Übermittlung jeweils ein Zeichen übermittelt werden können.

5. Somit erhalten wir folgende Zeichen

→ (0, 0, A) (0, 0, M) (2, 2, M) (4, 2, A) (6, 4, T)

6. Kompressionsrate ausrechnen

a) Ursprüngliche Grösse berechnen:

Nehmen wir an, dass jedes Zeichen ein Byte benötigt (1 Byte = 8 Bit). Ursprünglich hatten wir 13 Zeichen → 13 * 8 = 104 Bit

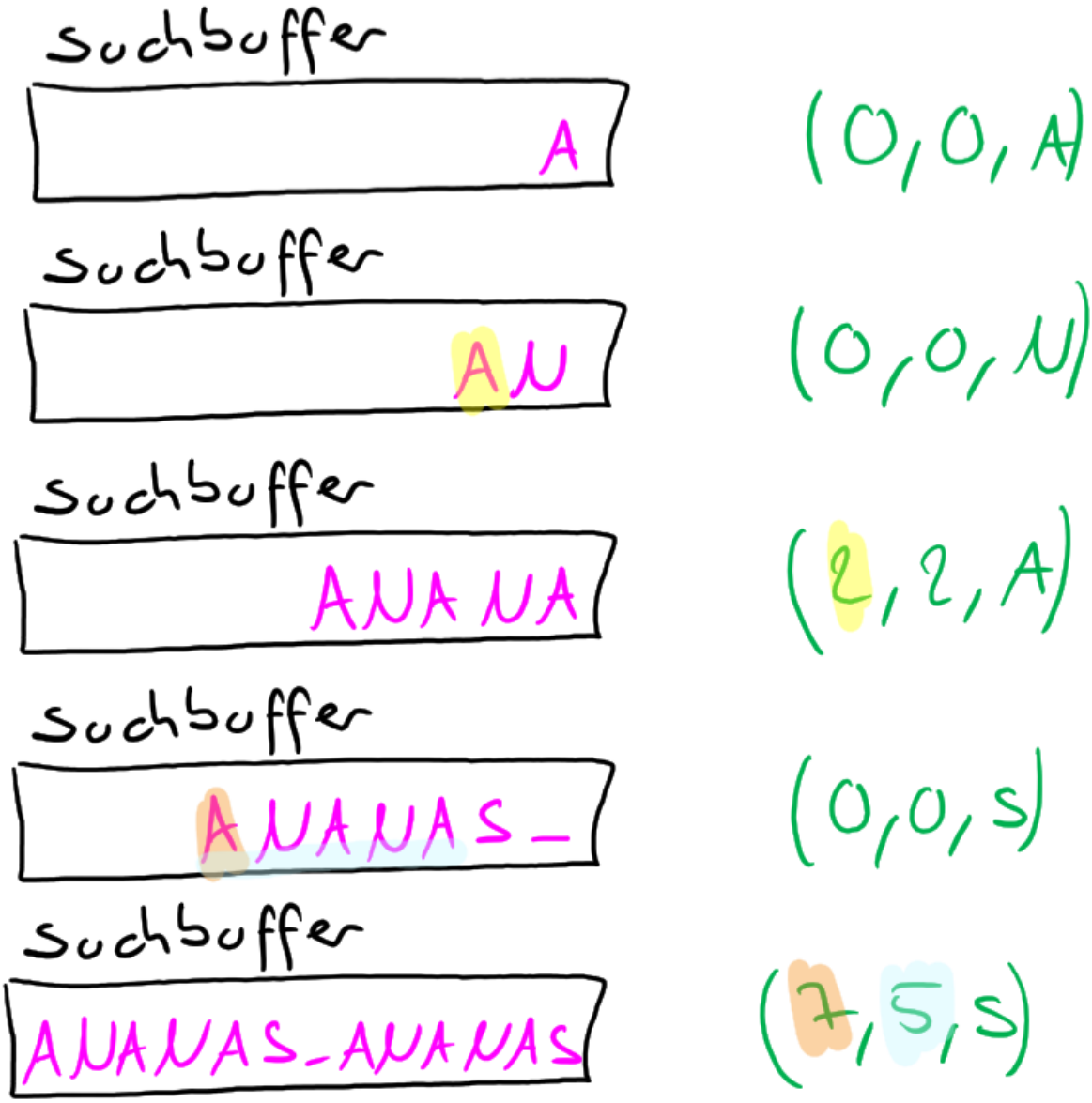
b) Grösse der Token berechnen:

Der Such-Buffer beinhaltet Platz für 10 Zeichen, dieser ist relevant für den Offset
→ 4 Bit notwendig
Der Vorschau-Buffer beinhaltet Platz für 5 Zeichen. Diese Zeichen sind relevant für die Längen angabe, welche übermittelt werden kann. Wobei das letzte Zeichen jeweils nicht mit der Länge übermittelt werden kann und somit müssen bis 4 abbilden können.
→ 3 Bit notwendig
Und dann kommt natürlich noch das eigentliche Zeichen dazu
→ 8 Bit (1 Byte) notwendig
Somit umfasst ein Token eine Grösse von (4 + 3 + 8 =) 15 Bit auf.
Da wir gesamthaft 5 Token haben umfassen diese gesamthaft (5 * 15 =) 75 Bit.

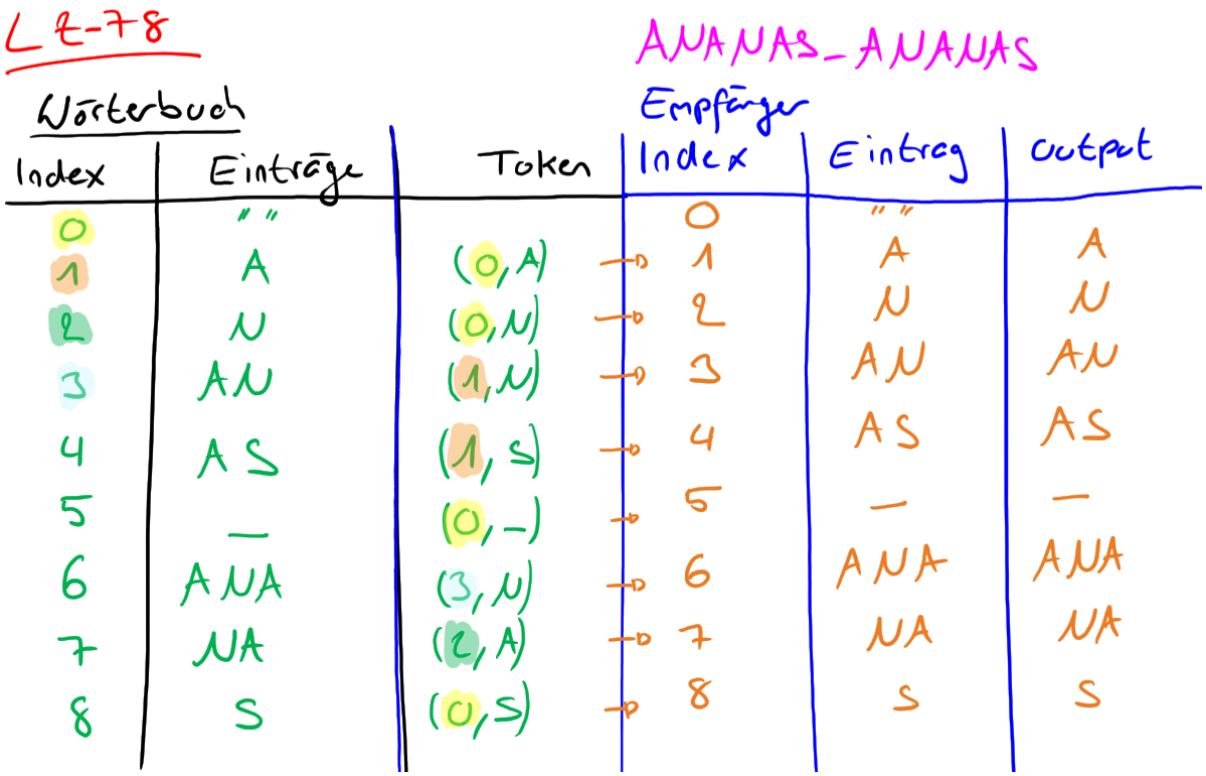
c) Kompression ausrechnen

75 / 104 = 0.72
→ wir haben eine Kompression erzielt.

Beispiel-Verfahren-Decodierung:
Der Decoder braucht nur den Suchbuffer. Danach geht er die einzelnen Tokens Stück für Stück durch und liest diese aus.


Suchbuffer
A
(0,0,A)
Suchbuffer
AN
(0,0,N)
Suchbuffer
ANANA
(2,2,A)
Suchbuffer
ANANAS_
(0,0,S)
Suchbuffer
ANANAS_ANANAS
(7,5,S)

LZ-78



Es wird nun aber nicht ein Sliding Window verwendet, sondern das Verfahren baut im Verlauf des Prozesses ein eigenes Wörterbuch auf. Dabei werden Token gebildet, die beschreiben, wie der jeweils letzte Eintrag im Wörterbuch gebildet wurde. Dies ist wichtig, denn der Empfänger wird aus den Token sein eigenes Wörterbuch aufbauen, das natürlich identisch mit jenem des Senders sein muss. Beide Wörterbücher, jenes beim Sender und jenes beim Empfänger, sind am Anfang leer (siehe Abbildung oben: index 0)

Verfahren:
Das Verfahren funktioniert so, dass die Nachricht immer von links nach rechts gelesen wird, bis der gelesene Teilstring im Wörterbuch nicht mehr vorhanden ist. In dem Moment, wo der Teilstring nicht mehr im Wörterbuch vorkommt, besteht er aus einem Teil, der noch im Wörterbuch vorkommt plus einem zusätzlichen Zeichen. Der Token, der nun gebildet wird, beschreibt genau dies: Er beschreibt, welcher bestehende Eintrag im Wörterbuch mit welchem zusätzlichen Zeichen erweitert wird, um den neuen Eintrag zu bilden.

Der Token besteht also aus zwei Elementen:
Token: (, Zeichen)
Der *Index* referenziert den bestehenden Eintrag im Wörterbuch, das *Zeichen* gibt an, in welcher Weise der bestehende Eintrag erweitert wird.

Beispiel-Verfahren:
Man beginnt die Nachricht von links zu lesen, dabei stoppt man bereits beim ersten Zeichen A, da dieser Eintrag noch nicht im Wörterbuch vorhanden ist. Daher erstellt man den ersten Token (0, A) -> der neue Eintrag beruht auf dem leeren String (Index 0). Danach liest man weiter, da das N ebenfalls noch nicht vorhanden ist, erstellt man den zweiten Eintrag, wiederrum auf Basis von Index 0, (0, N). Der nächste Buchstabe der Nachricht ist

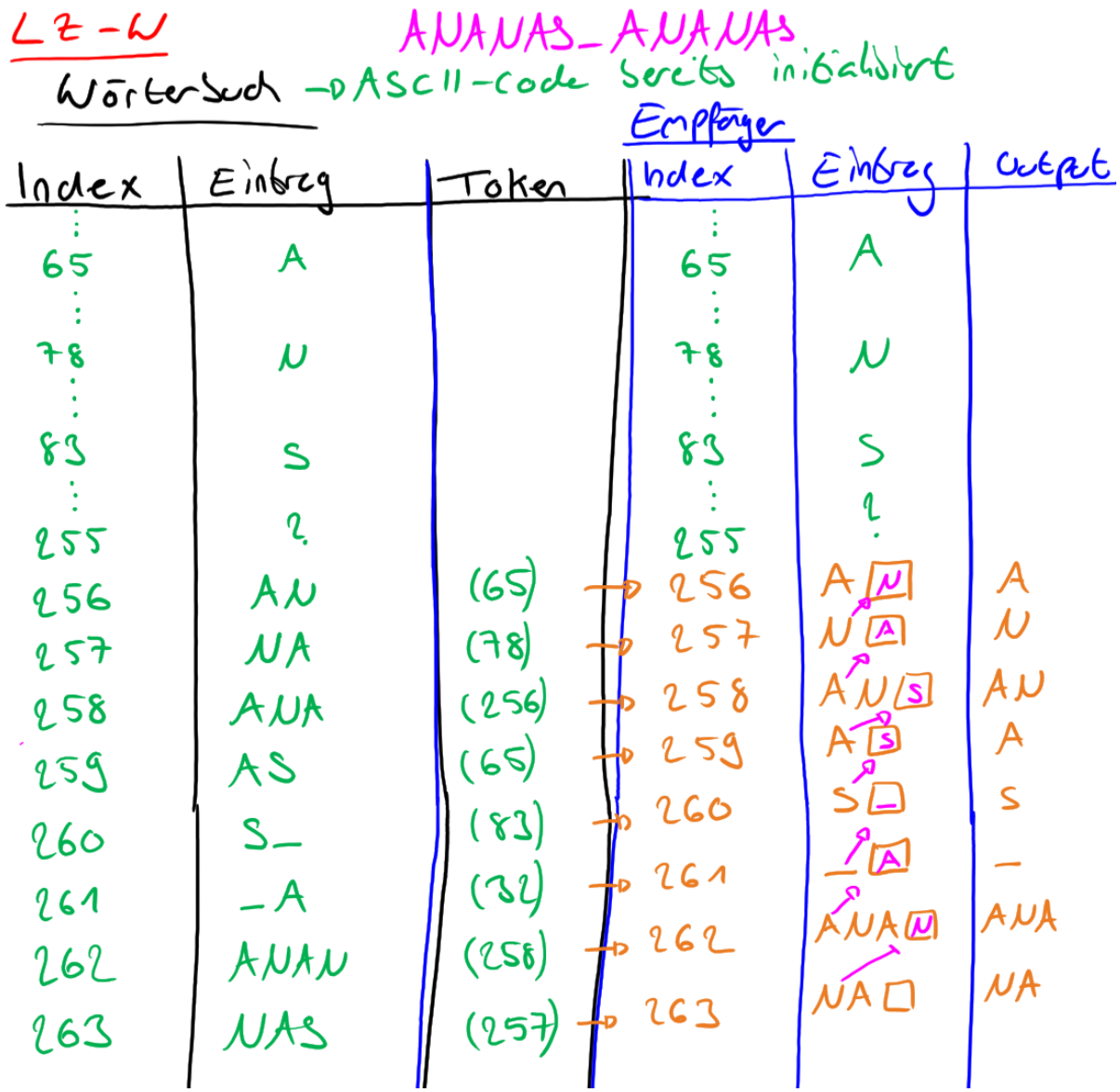
ein “A”, dieses “A“ (Index 1) ist bereits vorhanden, so schaut man sich noch zusätzlich der nächste Eintrag an was ein N ist. “AN” ist jedoch noch nicht im Wörterbuch vorhanden, deshalb erstellt man den nächsten Token auf Basis des “A” (1, N). Dieser Vorgang wiederholt man nun für sämtliche Buchstaben der Nachricht.

Man erkennt unschwer, dass die String in den Einträgen je länger, desto breiter werden. Das heisst, je länger das Verfahren läuft, umso mehr Zeichen werden in einem Token codiert. Bei sehr langen Nachrichten ergibt sich nach einer Weile eine optimale Kompression.

Berechnung Kompressionsrate ist zu ergänzen

Decoder:
Der Decoder (in Bild als Empfänger dargelegt) sieht nur die Token und bildet sich dabei sein eigenes Wörterbuch.

LZW



Das LZW-Verfahren ist eine Weiterentwicklung von LZ78 mit dem Ziel, den Token zu verkleinern. Der LZW-Token besteht in der Folge nur noch aus einem Wörterbuch-Index. Das Verfahren ist etwas umständlicher als LZ78, aber in der Regel leicht effizienter.

Das Wörterbuch von LZW ist vorinitialisiert mit allen Zeichencodes von 0 bis 255 (einschliesslich den ASCII-Zeichen im Bereich 0 bis 127). Das sieht dann so aus:

Im Gegensatz zum LZ78 enthält der Token nun aber nur den Index des schon bestehenden Eintrags im Wörterbuch, nicht aber das neue Zeichen.
Token: (Index)

Beispiel-Vorgehen:
Da der Buchstabe “A” intial bereits im Wörterbuch vorhanden ist, wird ein neuer Index für “AN” erstellt. Bei diesem Index wird jedoch nur den davon ausgehende Index angegeben. In diesem Fall der Index 65. Das noch zu übermittelnde ”N” ist aktuell noch nicht übermittelt

worden, dies geschieht erst mit dem nächsten Eintrag. Denn der neue Antrag beginnt beim letzten Zeichen des vorhergehenden Eintrags. Dementsprechend wird nun ein neuer Eintrag ins Wörterbuch für “NA” mit dem Index 257 erstellt. Dabei wird wiederum nur der ausgehende Index übermittelt (Index 77 für das “N”). Zu diesem Zeitpunkt weiss, das Wörterbuch, was der zweite Buchstabe beim Index 256 ist und schreibt diesen ins Wörterbuch. Dieses Verfahren wird nun n-Mal durchgeführt, bis sämtiche Buchstabe eingelesen sind.

Es fällt auf, dass das LZW-Verfahren etwas mehr Token erzeugt, als die LZ78-Methode, aber die Token sind - wie beabsichtigt - kürzer als bei der LZ78-Methode.

Berechnung Kompression:

Nehmen wir wiederum an, dass das Wörterbuch klein sei, also zum Beispiel maximal 511 Einträge aufnehmen kann (inkl. den schon initialisierten Einzelsymbolen), so ist ein Token 9 Bit gross. Die 8 Token benötigen demnach 8·9 = 72 Bit.
Der entsprechende Originaltext ist 13 Zeichen lang, das letzte Zeichen wurde aber noch nicht übertragen. Damit erhalten wir den Kompressionsfaktor R:
$R = \frac{8 * 9}{12 * 8} = 0.75$

Decoder:

Es sei im Folgenden gezeigt, wie der Decoder arbeitet. Verwendet werden die Token von oben. Der Decoder startet mit einem initialisierten Wörterbuch, das die Einträge mit den Indizes von null bis 255 enthält. Jetzt erhält der Decoder den ersten Token (65). Damit bildet er einen Wörterbucheintrag. Er besteht aus einem Doppel-Symbol, wobei aber der zweite Buchstabe noch nicht bekannt ist. In der Tabelle oben ist das fehlende Zeichen mit dem einem orangen Rahmen und in pinker Schrift dargestellt. Mehr kann der Empfänger vorläufig nicht tun. Der Output des Decoders ist das Zeichen A, entsprechend dem Token. Man sieht also, dass jeder Wörtbucheintrag immer erst beim Eintreffen des nächsten Tokens fertiggestellt werden kann. Das so entstehende Wörterbuch ist natürlich wieder identisch mit dem Wörterbuch des Senders. Tritt der nächste Token (265) ein, so folgt

JPEG-Verfahren

Beim JPEG-Verfahren handelt es sich um ein verlustbehaftetes Komprimierungsverfahren. Irrelevante Informationen, in casu für das Auge irrelevante Informationen werden eliminiert und sind nach der Kompression nicht mehr vorhanden. Demzufolge spricht man bei solchen Verfahren auch von Irrelevanzreduktion, um höhere Kompressionsraten zu erzielen.

JPEG unterscheidet sich von Quellcodierungsverfahren:

- JPEG ist verlustlos im Gegensatz zu Quellcodierungsverfahren wie Huffman, LZW, etc.
- JPEG trägt den physiologischen Eigenschaften des Betrachters Rechnung und eliminiert irrelevante Information (aus dem Auge des Betrachters). -> Irrelevanz statt nur Redundanzreduktion.
    - Der Mensch kann besser Hell/Dunkel-Wechsel als Farbwechsel unterscheiden
    - Der Mensch kann besser grobe als feine Muster unterscheiden

Kompression bei JPEG:

a) Durch den Farbraumwechsel von RGB1 zu YCrCb2 erhält man die zwei Bilder Cr und Cb, die nur Farbwechsel enthalten, aber keine Helligkeitswechsel. Da das menschliche Auge diese Farbwechsel schlechter erkennen kann, als Helligkeitswechsel, lassen sich die Cr- und Cb-Bilder unterabtasten. Das heisst, sie werden verkleinert und verlieren an Auflösung.

(b) Durch die Anwendung der DCT3 auf 8 x 8 Pixel Bilder, erhält man deren Frequenzgehalt, resp. die Unterscheidung von groben und feinen Mustern. Da das menschliche Auge für feine Muster (hohe Frequenzen) unempfindlicher ist, können diese Muster reduziert oder ganz entfernt werden.

! Die Entropie nimmt bei der Kompression ab, da Information verloren geht!

Wozu dient die Quantisierungsmatrix im JPEG-Verfahren?

-> Nach der DCT werden die 8 x 8 Pixel grossen Frequenzbilder pixelweise durch die Elemente der Quantisierungsmatrix dividiert. Dabei wird eine Ganzzahl-Division ohne Rest ausgeführt. Steht an einer Stelle in der Quantisierungsmatrix eine Eins, so bleibt der entsprechende Wert im Frequenzbild erhalten. Für jeden Wert grösser als eins wird der betreffende Wert im Frequenzbild reduziert. Ist der Wert in der Quantisierungsmatrix grösser als der Pixelwert im Frequenzbild, so resultiert null und die betreffende Frequenzinformation verschwindet ganz im Bild. Im Frequenzbild möchte man kleine Frequenzen erhalten und grosse reduzieren. Daher nehmen die Werte in der Quantisierungsmatrix mit der Frequenz zu.

- bewusst verlustbehaftet
- Komprimierung kann nicht rückgängig gemacht werden
- Komprimierung basiert auf Wahrnehmung des menschlichen Auge
    - Farbinformationen sind nicht so relevant
        - Grün: am empfindlichsten
        - Rot: empfindlich
        - Blau: unempfindlich
    - Helligkeit ist relevant
- Ortsbild wird in Frequenzbild umgewandelt (hohe und tiefe Frequenzen sind separiert)

Luminanz = Helligkeit
Chrominanz = Farben
DCT = Diskrete Cosinus Tranformation

Verfahren:
1.Schritt

a) RGB → Y C_r C_b
b) C_r C_b wird reduziert (Down-Sampling)

Zwischenschritt

Bild wird auf 8 x 8 eingeteilt

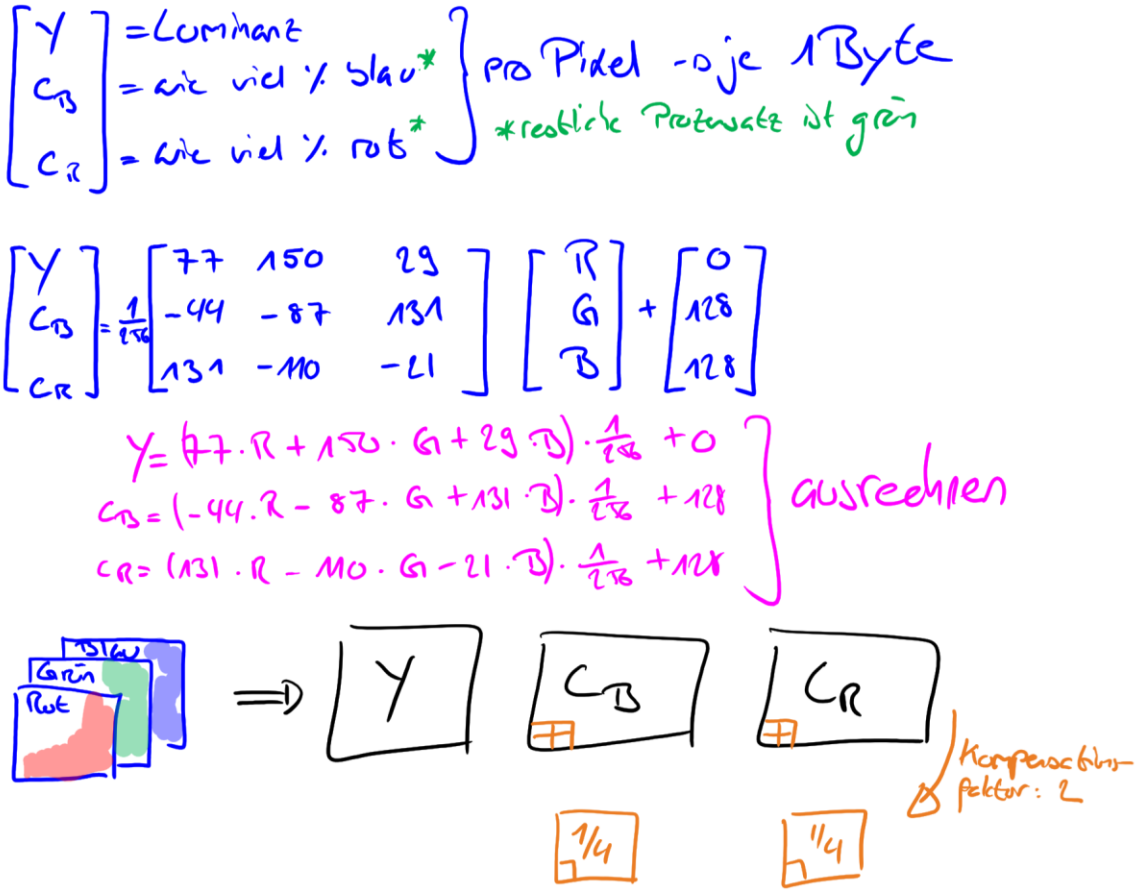
2.Schritt

a) DCT (Ortsbild wird zum Frequenzbild)
b) Quantisierung

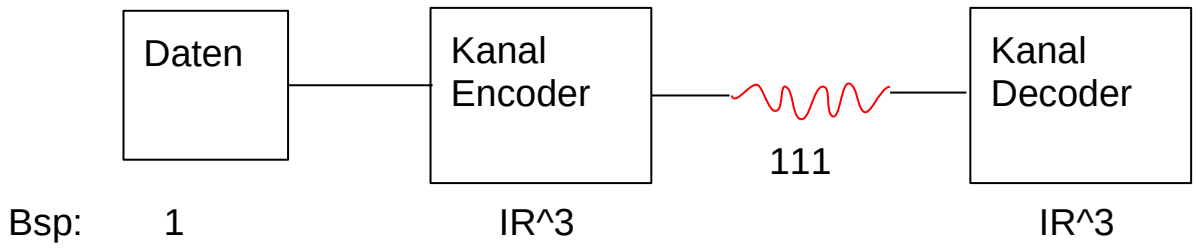
Komprimierung

Serialisierung
RLE
Hoffman

JPEG-Header



Kanalcodierung



- Durch das Verdreifachen kann der Decoder feststellen, dass das “Paket” korrekt angekommen ist
- Die Wahrscheinlichkeit, dass der Kanal korrekt übermittelt wurde ist sehr nahe bei 1

Beider Kanalcodierung ist das Hauptziel, die Daten zuverlässig von der Quelle bis zur Senke zu übertragen. Dies kann oftmals nur durch zusätzliche Fehlerschutzbits, die in den Bitstrom eingefügt werden, erreicht werden. Der Decoder hat keine Möglichkeit ein

Feedback an den Encoder zusenden, dies bedeutet, dass der Decoder selbstständig entscheiden können muss, wie die richtige, also ursprüngliche Bitfolge lautet. Dabei sind zwei grundsätzliche Vorgehensweisen denkbar:

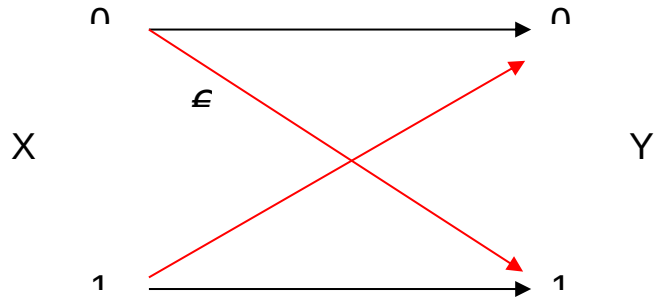
- Erste Variante: Es wird über einen bestimmten Datensatz, zum Beispiel ein Datenrahmen, eine **Prüfsumme** gebildet. Stimmen die Prüfsummen beim Sender und Empfänger überein, kann von einer fast 100 %-ig fehlerlosen Übertragung ausgegangen werden. Der Sender erhält ein „Acknowledgement“ (ACK). Stimmen die Prüfsummen nicht überein, so bleibt der ACK aus. Nach einer bestimmten Zeit des Ausbleibens des ACK wird der Datensatz nochmals gesendet (Retransmission), bis die Prüfsummen schliesslich übereinstimmen (Beispiel TCP, Transmission Control Protocol).
- Zweite Variante: Es ist eine **„Forward Error Correction“** angewandt. In den Sendebitstrom werden zusätzliche Bits eingeschleust, um damit beim Empfänger einfache oder mehrfache Bitfehler korrigieren zu können. Ein Wiedersenden (Retransmission) ist nicht vorgesehen. Der Sendebitstrom mit den eingeschleusten zusätzlichen Bits bietet die Grundlage, damit der Empfänger bis zu einem gewissen Grad Fehler selbst zu korrigieren vermag

→ Im Folgenden gehen wir nur auf die zweite Variante ein. Es ist eine sehr schnelle Variante im Vergleich zur ersten Variante.

Diskreter Kanal:
Wenn in einem zeitdiskreten Kanal die Werte, welche Eingangs- und Ausgangsvariabeln einnehmen können, endlich oder zählbar unendlich sind, bezeichnet man den Kanal als diskreten Kanal

Binary Symmetrie Channel (BSC)

Ist nur der Kanal in der oberen Abbildung rot dargestellt.



€ = Bit Error Rate (BER)

Wahrscheinlichkeit in N Bit F falsch sind:

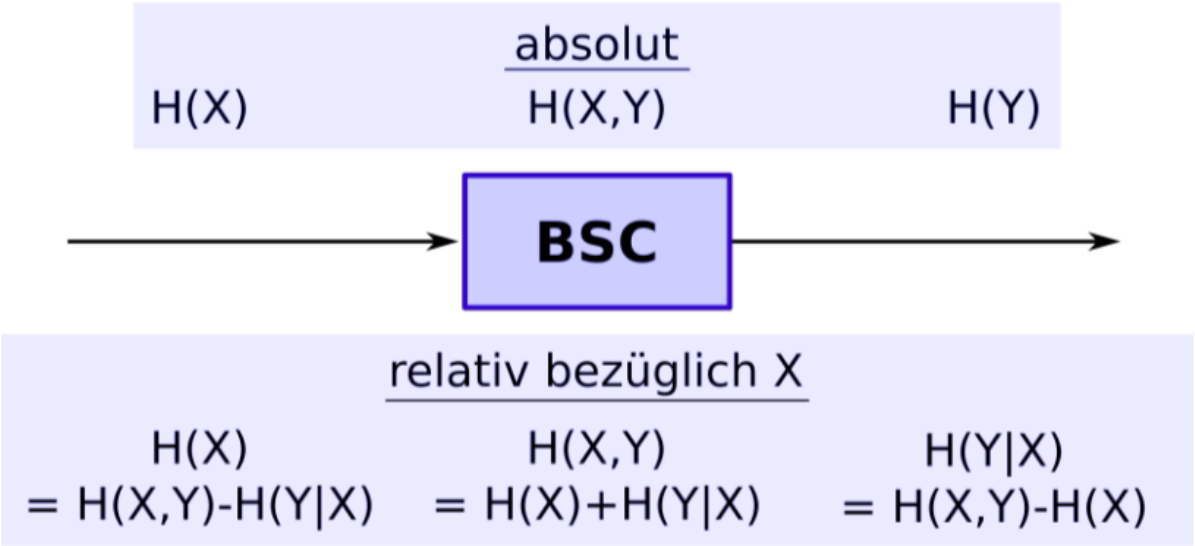
$P_{F,N} = \binom{N}{F} * \epsilon^F * (1-\epsilon)^{N-F}$

Der Binomialkoeffizient lässt sich im Taschenrechner mit der Taste “nCr” berechnen!

Beispiele:

$\binom{N}{F} = \binom{6}{1} = \frac{6}{1} = 6$
$\binom{N}{F} = \binom{6}{2} = \frac{6*5}{2*1} = 15$
$\binom{N}{F} = \binom{6}{3} = \frac{6*5*4}{3*2*1} = 20$
$\binom{N}{F} = \binom{6}{4} = \frac{6*5*4*3}{4*3*2*1} = 15$

Formel für Bedingte Wahrscheinlichkeit, Formel für totale Wahrscheinlichkeit, etc.



Entropie von Y wenn ich X kenne
H(Y*|X) = H(X,Y) - H(X)
**aufgrund von Fehlern*

Gemeinsame Info von X & Y
I(X;Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y)

Kanalkapazität
C_{BSC} = max (P_x) I(X;Y) Bit / Symbol = Bit^1 / Bit^2

Bit^1 = Informationsbit // Bit^2 = Übertragungsbit
→ Das Ziel sollte sein eine möglichst grosse Anzahl von Informationsbit zu erhalten => Grosse Blöcke

$C_{BSC} = 1 - h(\epsilon)$

$h(\epsilon) = \epsilon * log2(\frac{1}{\epsilon}) + (1-\epsilon) * log2(\frac{1}{1-\epsilon})$ → **binäre Entropiefunktion**

Kanalcodierungs-Theorem

“Unter allen Codes mit R<C gibt es welche, mit denen die Fehlerwahrscheinlichkeit beliebig klein gemacht werden kann. Falls R>C gibt es keinen Code mit beliebig kleinen Fehlerwahrscheinlichkeit => Null kann bei der Fehlerwahrscheinlichkeit nicht erreicht werden, Redundanz dient dem Fehlerschutz → Je grösser R, umso besser”

Die Kanalkapazität eines diskreten gedächtnisfreien Kanals ist gegeben durch die Beziehung
$C = \max_{P(x)} [H(Y) - H(Y|X)]$

C: Bit / Kanalkapazität
R: $\frac{Anzahl\ Informationsbit}{Anzahl\ Codebit}$

Cyclic redundancy check (CRC)

CRC ist eine Art der Kanalcodierung wird beim Übermitteln von Daten zum Schutz von Übermittlungsfehlern eingesetzt. Aus jeweils K Nutzbits, werden P Prüfbits berechnet, welche an die Nutzbits angehängt werden. Das übertragene Datenwort besteht somit aus N = K + P Bits.

Grundidee:
Die Idee besteht darin, dass das N-stellige CRC-Codewort 2N verschiedene Bitmuster annehmen kann, davon aber nur 2K Möglichkeiten benutzt werden. Tritt bei einer Übertragung ein Fehler auf und erhält der Empfänger ein Bitmuster, das keinem gültigen Codewort entspricht, so erkennt er daran den Übertragungsfehler.

Der Trick von CRC besteht nun darin, dass die Prüfbits so gewählt werden, dass möglichst viele typische Arten von Übertragungsfehlern erkannt werden. In dieser Hinsicht ist CRC besonders erfolgreich und wird entsprechend oft eingesetzt, zum Beispiel bei Ethernet, USB und einer ganzen Anzahl weiterer Standards der Datenübertragung.

Eigenschaften:
CRC-Codes sind linear und zyklisch

Die Berechnung der CRC-Prüfbits können mathematisch beschrieben werden. Dabei wird das binäre Datenwort als Polynom dargestellt.
→ u = (100101) ≈ U(z) = $1 * z^5 + 0 * z^4 + 0 * z^3 + 1 * z^2 + 0 * z^1 + 1 * z^0 = z^5 + z^2 + z^0$

Zum Berechnen der Polynome befinden wir uns in der 1-Bit-Arithmetik, das heisst es gibt keine Übertrag (Carry):

Bei der Addition und Subtraktion ist es eine XOR-Verknüpfung

a	b	$a \pm b$
0	0	0
0	1	1
1	0	1
1	1	0

Bei der Multiplikation entspricht die Verknüpfung eines AND-Operators.

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

Generatorpolynom:
Bei CRC wird ein Generatorpolynom G(z) verwendet, um aus dem Datenwort *u* das Codewort *c* zu erzeugen. Die Idee ist, dass jedes Codewort ein Vielfaches (eine Linearkombination) des Generatorpolynoms G(z) ist. Dadurch sind derartige Codes automatisch zyklisch und linear.

0	0	1	0	0	1	0	1	$G(z)$ ist ein gültiges Codewort.
1	0	0	1	0	1	0	0	Permutation ergibt ein anderes Codewort (zyklischer Code).
1	0	1	1	0	0	0	1	Die Summe beider Codeworte ist ein Codewort (linearer Code).

Name	Polynom G(z)	Bemerkungen
CRC-1	$z+1$	Parity-Bit (gerade)
CRC-4	z^4+z+1	Identisch zu (15,11)-Hamming-Code
CRC-5-USB	z^5+z^2+1	Identisch zu (31,26)-Hamming-Code
CRC-5-ITU	$z^5+z^4+z^2+1$	
CRC-7	z^7+z^3+1	Identisch zu (127,120)-Hamming-Code
CRC-8-CCITT	z^8+z^2+z+1	
CRC-12	$z^{12}+z^{11}+z^3+z^2+z+1$	
CRC-16-CCITT	$z^{16}+z^{12}+z^5+1$	Verwendet für X.25, SD, Bluetooth.
CRC-16-IBM	$z^{16}+z^{15}+z^2+1$	Verwendet z.B. für Modbus, USB
CRC-32	$z^{32}+z^{26}+z^{23}+z^{22}+z^{16}+z^{12}+z^{11}$ $+z^{10}+z^8+z^7+z^5+z^4+z^2+z+1$	Verwendet für Ethernet, ZIP, PNG, SATA, MPEG-2, ZMODEM
CRC-64-ISO	$z^{64}+z^4+z^3+z+1$	

CRC-16:
Das CRC-16-IBM-Polynom hat im wesentlichen dieselben Eigenschaften wie das Polynom CRC-16-CCITT. Unter der Voraussetzung, dass der Datenblock nicht länger ist als 4095 Byte, gilt:

- Alle 1-Bit Fehler werden erkannt.
- Alle Fehler mit einer ungeraden Anzahl von Bitfehlern werden erkannt.
- Alle 2-Bit Fehler werden erkannt.
- Alle Fehlerbursts, die nicht länger als 16 Bits sind, werden erkannt.
- Für Fehlerburts der Länge 17 Bits oder mehr liegt die Wahrscheinlichkeit für das Erkennen bei über 99.997%

CRC-32
Das Generatorpolynom hat eine ungerade Anzahl von Termen und kann somit nicht alle ungeraden Bitfehler erkennen (die Begründung folgt weiter unten). Die Wahrscheinlichkeit, einen Fehler nicht zu entdecken, hängt auch von den verwendeten Daten ab, sowie von der Fehlercharakteristik des verwendeten Kanals. Empirische Untersuchungen für gute Fehlercodes benötigen normalerweise sehr große Stichproben. In einem Beispiel wurden bei 157’000 Blöcken der Länge 2048 Bits keine unentdeckten Fehler beobachtet. Eine Berechnung der Wahrscheinlichkeit für unentdeckte Fehler ergab, dass diese kleiner als 10^{-14} ist.

CRC-8
ist ein Prüfpolynom, welches von der CCITT in der Empfehlung I.432 für die Prüfung des vier Byte langen Headers einer ATM-Zelle vorgeschlagen wird. Mit der angehängten Prüfsumme wird der Header fünf Byte lang. Man kann zeigen, dass sämtliche 1, 2 und 3-Bit Fehler in
dem fünf Byte langen Header mit Sicherheit erkannt werden

Verfahren

Senderseite

Sei *U(z)* die Nutzbits eines Polynoms und *G(z)* das Generatorpolynom mit dem Grad *p*.

1. U ist um *p* Stellen nach links zu schieben und wird mit Nullen aufgefüllt.
2. Polynomdivision mit dem Generatorpolynom *G* durchführen
   → Rest der Division entspricht den *p* Prüfbits P

$$\frac{z^p \cdot U}{G} = Q + \frac{P}{G}$$

Die Multiplikation von U mit z^pentspricht dem Schieben um die *p* Stellen nach links

Das Polynom für das Codewort *C* wird gebildet, indem das Prüfbit-Polynom *P* zu den Nutzdaten *U* addiert wird. Dadurch werden die zuvor angehängten Nullen überschrieben.

$$C = z^p \cdot U + P$$

Beispiel einer Polynomdivision auf Senderseite:

```
10010110100000 : 110101 = 111101010
110101
------
 100001
 110101
 ------
  101000
  110101
  ------
   111011
   110101
   ------
     111000
     110101
     ------
       110100
       110101
       ------
            10    <== Rest P
```

Wichtig!
Der Grad des Prüfpolynoms muss p entsprechen. Ist der Rest (wie in diesem Beispiel) zu kurz, so muss also links mit Nullen aufgefüllt werden. Der berechnete Rest (die gesuchte Prüfsumme) ergibt sich zu p = (00010) und das generierte Codewort wird demnach c = (10010110100010)

Übertragung

Bei der Übertragung können einzelne Bits verfälscht werden. Der Empfänger wird daher nicht mehr das korrekte Codewort c empfangen, sondern eine allenfalls modifizierte Form davon. Wir bezeichnen das empfangene, möglicherweise fehlerhaft Codewort mit $\underline{\widetilde{c}}$. Den Fehlervektor nennen wir e. Er hat dieselbe Länge wie c und $\underline{\widetilde{c}}$ und hat eine 1 an jenen Stellen, wo ein Fehler vorliegt. Falls e = (00. . .0), so ist kein Fehler aufgetreten. Unter Verwendung des Fehlervektors e können wir schreiben:

$$\underline{\widetilde{c}} = \underline{c} + \underline{e}$$

Polynomschreibweise:

$$\widetilde{C}(z) = C(z) + E(z)$$

Empfängerseite

Der Empfänger dividiert das empfangene Wort $\widetilde{C}$ durch das Generatorpolynom G. Da erwartet wird, dass $\widetilde{C}$ ein Vielfaches von G ist, sollte es keinen Rest geben. Dass C ein Vielfaches von G ist, sieht man auch daran, dass der Rest der Division U/G im Sender zu $z^p * U$addiert (resp. subtrahiert) wurde. Entsteht trotzdem ein Rest, so ist ein Übertragungsfehler aufgetreten. Im anderen Fall nehmen wir an, die Übertragung sei korrekt verlaufen. Allerdings kann der äusserst seltene Fall auftreten, wo die Division trotz fehlerhaften Daten $\widetilde{C}$ ohne Rest duch G teilbar ist. In diesem Fall entsprechen die fehlerhaften Nutzdaten zufällig gerade wieder einem Vielfachen von G

Beispiel Polynomdivision auf Empfängerseite:

```
10010110100010 : 110101 = 111101010
110101
------
 100001
 110101
 ------
  101000
  110101
  ------
   111011
   110101
   ------
     111000
     110101
     ------
       110101
       110101
       ------
            00    <== Rest P
```

Fehlererkennung

.

- Ein Fehler in den Nutzdaten `1101111010`

→ Es gibt einen Restwert bei der Division

- Ein Fehler in der Prüfsumme 1100111000

→ Es gibt einen Restwer bei der Division

- Ungerade Anzahl von Fehlern

1. Wenn das empfangene Bitmuster C(z) ohne Rest durch das Generatorpolynom G(z) dividierbar ist, wird eine fehlerfreie Übertragung angenommen.
2. Ein geeignetes Generatorpolynom G(z) erkennt sämtliche Burstfehler, die nicht länger sind als der Grad p des Generatorpolynoms. Das beinhaltet natürlich auch 1-Bit Fehler als Burstfehler der Länge eins. Wenn das Generatorpolynom also mehr als einen Term enthält, werden alle 1-Bit Fehler erkannt. Ausserdem erkennt das Polynom alle Burstfehler des Grads p+1, ausser jenem, wo das Fehlermuster dem Polynom entspricht.
3. Ein Generatorpolynom mit gerader Anzahl von Termen erkennt jede ungerade Anzahl von Bitfehlern. Ebenso ist dies der Fall, wenn der Faktor (z+1) im Generatorpolynom enthalten ist.
4. Man kann ferner zeigen, dass 2-Bit Fehler nur erkannt werden, wenn eine gewisse Nachrichtenlänge nicht überschritten wird. Bei optimal gewählten Generatorpolynomen vom Grad p mit gerader Anzahl von Termen ist diese Länge LMax = 2p−1 −1. Ist beispielsweise p = 16 beträgt diese Länge immerhin 32767, also mehr als 4000 Bytes.
5. Entspricht ein Fehlerpolynom E(z) einem beliebigen Vielfachem des Generatorpolynoms G(z), kann der Empfänger im Bitmuster C(z) keinen Fehler erkennen

Fehlerkorrektur

Mittels CRC lassen sich sämtliche 1-Bit-Fehler auch gleich korrigieren. Dies ist möglich, wenn das Codewort nicht länger als eine gewisse kritische Länge L_{krit}

Beispiel:
Das CRC-5-ITU Generatorpolynom hat die bereits bekannte Polynomdarstellung $G(z) = z^5 + z^4 + z^2 + 1$, resp. g = (110101). Bei diesem Generatorpolynom ist $L_{krit} = 15$. Somit können 1-Bit Fehler erkannt und eindeutig einem bestimmten Bit zugeordnet werden, wenn das empfangene Bitmuster $\widetilde{C}(z)$ nicht länger ist als 15 Bit.

Fehler bei Bit	Rest
–	00000
0	00001
1	00010
2	00100
3	01000
4	10000
5	10101
6	11111
7	01011

Fehler bei Bit	Rest
8	10110
9	11001
10	00111
11	01110
12	11100
13	01101
14	11010
15	00001
16	00010

Implementierung

Man stellt links das Schieberegister SR dar und rechts davon das Bitmuster. Anschliessend fügt man Stück für Stück das Bitmuster in das Schieberegister. Wenn im Schieberegister ganz links eine 1 steht, so dividiert (XOR) man das Schieberegister durch das Generatorpolynom, wenn eine eine 0 steht, so fügt man das nächste Bit des Bitmuster ins Schieberegister. Das Schrieberegister hat eine fixe Grösse. Am Ende steht im Schieberegister die CRC-Prüfsumme, d.h. wenn dort 0 steht ist es ein gültiges Codewort, wenn ≠ 0dann ist es kein gültiges Codewort.

Als Beispiel betrachten wir einen CRC-5-ITU Decoder mit g = (110101), der das empfangene Bitmuster c = (1011001101010) auf Übertragungsfehler überprüfen soll.

```
  SR       Bitmuster       Kommentar
-------------------------------------------------------
000000 <-- 1011001101010   Ausgangslage, SR ist leer.
000001 <-- 011001101010    Erstes Bit ins SR geschoben,   MSB = 0.
000010 <-- 11001101010     Nächstes Bit ins SR geschoben, MSB = 0.
000101 <-- 1001101010      Nächstes Bit ins SR geschoben, MSB = 0.
001011 <-- 001101010       Nächstes Bit ins SR geschoben, MSB = 0.
010110 <-- 01101010        Nächstes Bit ins SR geschoben, MSB = 0.
101100 <-- 1101010         Nächstes Bit ins SR geschoben, MSB = 1.
011001 <-- 1101010         XOR mit G = (110101) auf SR angewendet.
110011 <-- 101010          Nächstes Bit ins SR geschoben, MSB = 1.
000110 <-- 101010          XOR mit G = (110101) auf SR angewendet.
001101 <-- 01010           Nächstes Bit ins SR geschoben, MSB = 0.
011010 <-- 1010            Nächstes Bit ins SR geschoben, MSB = 0.
110101 <-- 010             Nächstes Bit ins SR geschoben, MSB = 1.
000000 <-- 010             XOR mit G = (110101) auf SR angewendet.
000000 <-- 10              Nächstes Bit ins SR geschoben, MSB = 0.
000001 <-- 0               Nächstes Bit ins SR geschoben, MSB = 0.
000010 <--                 Letztes Bit ins SR geschoben,  MSB = 0.
000010                     Ende, CRC-Prüfsumme steht im SR.
```

Hardware-Realisierung (Kap. 7.1 im Script)

Parity-Bits

Die Parity-Bit Methode ist ein einfaches Verfahren zum Schutz von fehlerhaften Übermittlung von Daten. Dabei wird im zu übertragenden Codewort, sämtliche einsen gezählt. Wenn es eine gerade (even) Anzahl von einsen hat, dann wird an hintersten Stelle eine 0 angehängt, bei einer ungeraden (odd) Anzahl wird zu hinterst eine 1 angehängt.
Mit dieser Varianten werden nur die ungeraden Anzahl von Fehlern erkannt, gerade Anzahl von Fehlern bleiben unerkannt.

Block-Codes

Die Aufgabe der Kanalcodierung ist die zuverlässige Übertragung von Information von Quelle (Sender) über Kanal bis zur Senke (Empfänger).

Kanal ist kaum beeinflussbar

$$u = Informationsbit, x = gesendetes\ Codewort, y = empfangenes\ Bitmuster, ue = decodierte\ Informationsbit$$

$$Anzahl\ Informationsbit\ in\ u\ werden\ mit\ K\ angegeben!$$

Forward Error Correction

Da kein Feedback vom Decoder zum Encoder möglich ist, muss der Decoder selbständig entscheiden können, wie die richtige, also ursprüngliche gesendete Bitfolge lautet. Dafür werden zusätzliche Bits eingeschleust, damit der Empfänger einfache oder mehrfache Bitfehler korrigieren kann.

Diskreter Kanal

Wenn in einem zeitdiskreten Kanal die Werte, welche Eingangs- und Ausgangsvariabeln einnehmen können, endlich oder zählbar unendlich sind, bezeichnet man den Kanal als diskreten Kanal.

Kanalcodierungstheorem nach Shannon

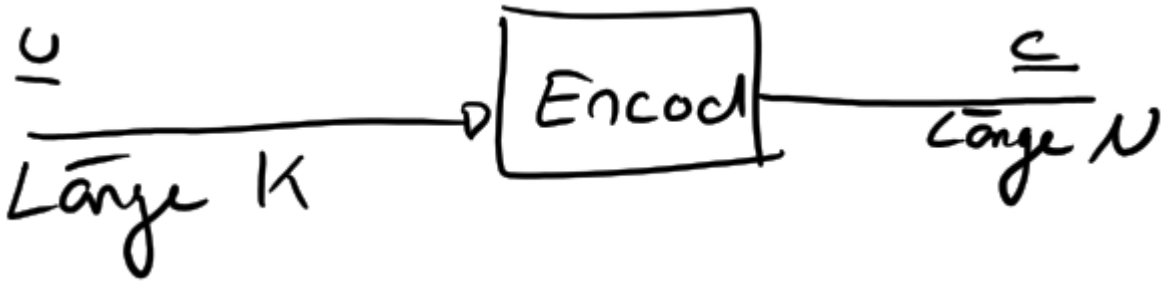
$$C = max_{P(x)}\,[H(Y) - H(Y|X)]\ (in\ Bit/Kanalbenützung)$$

In Prosa: Zur Berechnung der „Kanalkapazität“ in Bit/Kanalbenützung ist die Entropie am Kanalausgang und die bedingte Entropie am Kanalausgang, wenn der Eingang gegeben ist, verwendet. Die bedingte Entropie H(Y|X) hat mit den Fehlern zu tun, die bei der Übertragung auf dem Kanal entstehen. Falls keine Übertragungsfehler auftreten, so ist die bedingte Entropie H(Y|X) = 0. Als „Kanalkapazität“ ist das erzielbare Maximum der Differenz definiert, das auftritt, wenn die Wahrscheinlichkeitsverteilung der Eingangssymbole variiert. Pro memoria: Falls Nullen genau gleich häufig wie Einsen auftreten, ist H(Y) maximal, weil dann maximale Ungewissheit besteht.

Wenn die Übertragungsrate R<C, existiert ein Code mit ausreichend grosser Blocklänge N, mit dem die Fehlerwahrscheinlichkeit beliebig klein gemacht werden kann. Umgekehrt, wenn R>C, ist die Fehlerwahrscheinlichkeit von jedem Code immer >0, sei N noch so lang.

(N,K) Block-Codes

Ein binärer Block-Code ist ein Code, bei dem der Encoder die Informationssequenz in Blöcke aufteilt.



$Es\ gilt: K < N, da\ das\ Codewort\ aus\ N\ Codebits\ besteht, welche\ K\ Informationsbit\ enthalten$

$$Coderate\ R = \frac{K}{N}$$

(N,K) Block-Codes können systematisch, linear und zyklisch sein.

▼ Alle CRC-Codes sind linear, systematisch und zyklisch!

$$\textbf{systematisch}: = Die\ K\ Informationsbit\ sind\ 'en\ bloc'\ im\ Codewort\ ersichtlich.$$
$Jeder\ Code, welcher\ mit\ einer\ Generatormatrix, in\ welcher\ entweder\ links\ oder\ rechts\ Einheits-matrix\ enthält\ ist\ systematisch.\ (Da\ Einheitsmatrix\ einfach\ Nutzbits\ abbildet)$
$Beispiel: u = [01]\ c = [011]$

$$linear: = Die\ (mod-2)\ Summe\ zweier\ Codewörter\ ergibt\ wieder\ ein\ Codewort.$$
$$Beispiel: c_1 = [011], c_2 = [110], c_3 = [101] \rightarrow c_1 + c_3 = c_2$$

▼ Alle Block-Codes mit einer Generatormatrix sind linear

$$zyklisch: = Die\ Permutation\ eines\ Codeworts\ ergibt\ wieder\ ein\ Codewort.$$
$$Beispiel:\ c_1 = [011], c_2 = [110], c_3 = [101]$$

Wie viele Informationsworte u derselben Länge K sind möglich? Wie viele Codeworte?

$$Anzahl\ Informationsworte\ u = 2^K\ //\ Analog: Anzahl\ Codeworte = 2^K$$

Hamming-Gewicht

$$w_H(x) = Anzahl\ 1\ im\ Codewort]$$

Hamming-Distanz

$$d_H(x_i, x_j) = Anzahl\ unterschiedliche\ Positionen\ in\ den\ Codewörtern\ x_i, x_j$$
$$Beispiel: x_i = [011], x_j = [010] \rightarrow d_H = 1, da\ ein\ Bit\ verschieden.$$

minimale Hamming-Distanz d_{min}

$$d_{min} = min_k\ w_H(x_k)\ \ (die\ minimale\ Hamming - Distanz\ zwischen\ zwei\ (oder\ mehr)\ Codewörtern$$
$$Beispiel: x_i = [011], x_j = [000], x_h = [011] \rightarrow d_{min} = 1.$$

Wie viele Fehler kann ich mit (N,K) Block-Codes sicher erkennen?

$$Anzahl\ sicher\ erkennbare\ Fehler: d_{min} - 1$$

Wie viele Fehler kann ich mit (N,K) Block-Codes sicher korrigieren?

$$Anzahl\ sicher\ korrigierbare\ Fehler: t \leq \frac{(d_{min} - 1)}{2}$$

Wie gross sind N und K des folgenden Blockcodes?

$$\underline{G} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$Vorgehensweise:$$
$$1.\ G\ besteht\ aus\ Einheitsmatrix\ I\ und\ Parity - Matrix$$
$$2.\ Da\ I\ = K * K -> K = Anzahl\ Zeilen\ der\ Generatormatrix$$
$$3.\ N => Anzahl\ Spalten\ der\ Generatormatrix$$

Lösung: K=2, N=5

(N, K, t) Block-Codes

t = kann t Fehler sicher korrigieren

Wahrscheinlichkeit, dass ein solcher (N,K,t) Blockcode über ein BSC korrekt überträgt:

$$P_0 = (1 - \epsilon)^N$$
$$P_1 = \binom{N}{1} * \epsilon^K * (1 - \epsilon)^{N-K}$$
$$P_{\leq t} = \sum_{K=0}^{t} \binom{N}{K} * \epsilon^K * (1 - \epsilon)^{N-K}$$

Parity-Check-Matrix

I = K x K

P = K x N-K

$G = [I\ P]$ => Prüfmatrix $H = [P^T I]$
$G = [P\ I]$ => Prüfmatrix $H = [I\ P^T]$
Beispiel:

Repitäöön:

(4, 2) Blockcode

- N = 4
- K = 2
- R = 1/2
- linear, falls es eine Generatormatrix gibt
- systematisch, falls eine Einheitsmatrix in G vorkommt

Repetition-Code R^N: Jedes Symbol wird n-mal wiederholt. Bspw. R^3 ergibt die möglichen Code-Wörter R^3 = {(000),(111)}. R^3 ist ebenfalls linear & systematisch.

Syndrom

$$s = y \cdot H^T = (x + e) \cdot H^T = x \cdot H^T + e \cdot H^T = e \cdot H^T$$

$$mit\ s = Syndrom, y = empfangenes\ Bitmuster, e = Fehlervektor, H^T = trsp.\ Parity - Check - Matrix, x \cdot H^T = 0$$

Wie kann ich eine Syndromtabelle erstellen?

$$S = E \cdot H^T (dargestellt\ als\ Matrizen)$$

$$mit\ S = alle\ Syndrome, E = alle\ Fehlervektore, H^T = transponierte\ Parity - Check - Matrix$$

$$Ueberlegung: Fuer\ gueltige\ Codewoerter\ ist\ das\ Syndrom\ [0].$$

$Deswegen\ kann\ mit\ \ e$
$\cdot H^{T}\ (mit\ allen\ Fehlervektoren\ e)\ alle\ moegliche\ Syndrome\ bestimmt\ werden$

$$Vorgehen:$$

$1.)\ Alle\ Fehlervektoren\ e\ identifizieren, 2.)\ Alle\ \ e$
$\cdot H^{T}\ berechnen\ und\ Syndrome\ neben\ Fehlevektor\ e\ schreiben$

Wie kann ich die Anzahl Syndrome bestimmen?

$$Anzahl\ Syndrome\ =\ 2^{N-K}$$

$mit\ N = Länge\ des\ Codes, d.h. Anzahl\ Codebit, K = Anzahl\ Nutzbit/Informationsbit$

-> Siehe Übung 9/10 für Blockcodes.
noch zu ergänzen: HT* G = Null-Matrix, etc.

Faltungscodes

...ist noch zu erstellen...