Zahlencodierung II

Auf dem letzten Blatt haben wir Zahlensysteme im Stellenwertsystem und ganzen Zahlen behandelt. Nun erweitern wir die Sichtweise auf weitere Zahlensysteme. Des Weiteren kommen neue Aspekte hinzu, welche bei der Datenübertragung wichtig sind: Die Fehlererkennung und die Fehlerbehebung.

BCD Code

BCD bedeutet "Binary Coded Decimal" und ist ein System zur Codierung von Ziffern. Es wurde vor allem für kaufmännische Systeme entwickelt.

Ziffer	BCD Code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
(leer)	1010
(leer)	1011
(leer)	1100
(leer)	1101
(leer)	1110
(leer)	1111

Wie wir sehen können, gibt es mehr BCD Codes als Ziffern. Nicht verwendeten Codes werden bei Zahlensystemen **Redundanz** genannt. Die 6 redundanten Zeichen sollten also nie vorkommen, sonst liegt ein Fehler vor. Bei der BCD Codierung werden die 6 redundanten Zeichen **Pseudo-Tetrade** genannt.

Die Umrechnung zwischen dem Dezimalsystem und dem BCD System ist denkbar einfach, es werden alle Ziffern einzeln umgerechnet:

371_{10} ≙ 0011 0111 0001 (BCD)
3 7 1

Bei der Addition von BCD Zahlen gibt es eine Spezialität, da die Berechnung ungültige (redundante) Zeichen ergeben kann.

$3 + 8_{10}$ ≙ 0011 + 1000 = 1011 diese BCD Zahl entspricht keiner Dezimalzahl, sie liegt innerhalb der Redundanz. In diesem Fall muss die Pseudotetrade mittels Addition von 6_{10} bzw. 0110 übersprungen werden und in der nächsthöheren Stelle (links) 1 bzw. 0001 dazugezählt werden. ➔ ***1***

Ziffer	1 aus 10 Code
0	0000000001
1	0000000010
2	0000000100
3	0000001000
4	0000010000
5	0000100000
6	0001000000
7	0010000000
8	0100000000
9	1000000000

1 aus n Code

 S. 412

Eine komplett andere Idee steckt hinter den 1 aus n Codes. Dabei wird aus einer fixen Länge n jeweils 1 Bit mit dem Wert 1 codiert, die anderen n-1 Bits bleiben auf 0. Beispiel 1 aus 10 Code:

Diese Darstellung eignet sich optimal, um einen neuen Begriff einzuführen, nämlich den

Hamming Abstand.  S. 417

Dieser definiert, wie viele Bits sich von einer Zahl zur nächsten ändern. Die tiefste Anzahl Änderungen pro Sprung definiert den Hamming Abstand des Zahlensystems.

Beim 1 aus 10 Code beträgt der Hamming Abstand 2, weil von jeder Zahl zur nächsten mindestens 2 Bits den Wert wechseln.
Daraus können wir ableiten, dass es ungültige (redundante) Kombinationen gibt, nämlich wenn nur 1 Bit gewechselt würde. Tatsächlich sind die 1 aus n Codes mit einer grossen Redundanz ausgestattet. Daher sind sie optimal, um Fehler in der Übertragung zu erkennen.

Je höher die Redundanz, desto einfacher die Erkennung von Übertragungsfehlern!

Wird zum Beispiel der Code 0001000100 übertragen, kann der Empfänger sicher sein, dass die Daten nicht korrekt sind. Dieser Code befindet sich unter den redundanten Kombinationen. ➔***2***

Fehlererkennung und Fehlerbehebung

Fehlererkennung (Paritätsbit)

Da die oben behandelten Codes dazu neigen wertvolle Bits zu verschwenden, wir aber die Möglichkeit der Fehlererkennung nicht missen möchten, gibt es andere Verfahren um Fehler erkennen zu können. Ein häufig verwendetes Verfahren ist das **Paritätsbit**.

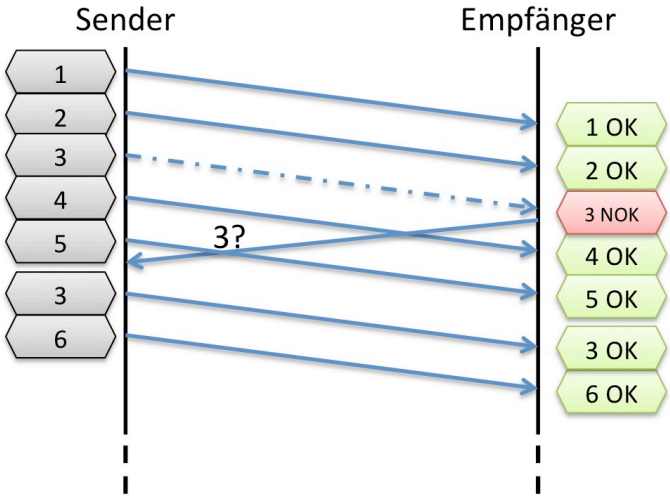
Dabei wird bei einem Code bestimmter Länge (z.B. immer nach 8 Bits) ein Paritätsbit angehängt, dies zählt, wie viele "1" Bits vorkommen. Bei gerader Parität "even parity" wird bei einer gerade Anzahl "1" Bits eine "0" angehängt. Bei einer ungeraden Parität "odd parity" wird bei einer ungeraden Anzahl "1" Bits eine "0" angehängt, sonst eine "1".

Bsp: even parity: 00101101 > 001011010
00101110 > 001011100
00101111 > 001011111

Dadurch wird künstlich der Hamming Abstand erhöht. ➔***3***

Die Paritätsbit sind eine einfache Möglichkeit Fehler in einem Datenstrom zu erkennen. Wenn nun ein Fehler beim Empfänger festgestellt wird, muss der Empfänger beim Sender die Information erneut anfordern. Dies wird u.a. beim TCP Protokoll so gemacht. Dieses Verfahren nennt sich **Rückwärtsfehlerkorrektur**.

Finden Sie 5 Einsatzbereiche der Rückwärtsfehlerkorrektur!



Schema Rückwärtsfehlerkorrektur

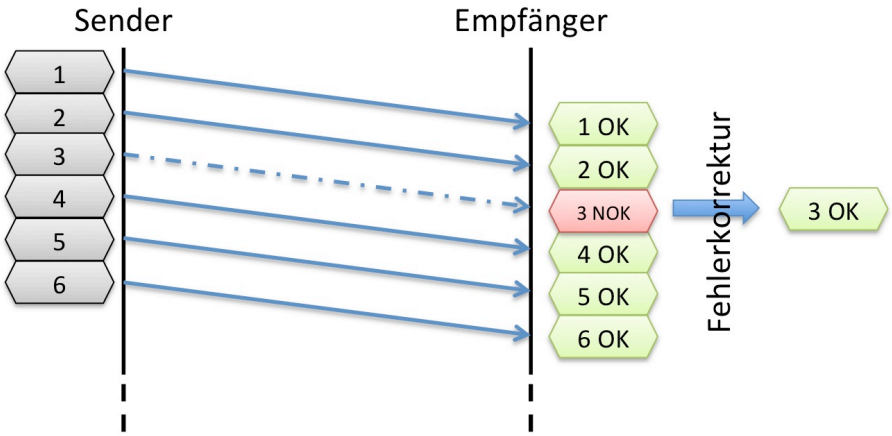
Damit die Rückwärts Fehlerkorrektur funktioniert, muss diese auf einem Protokoll basieren, welches dies zulässt. Folgende Herausforderungen gilt es zu berücksichtigen:

- Sender und Empfänger müssen das Protokoll aushandeln (even / odd / Anzahl Bits bis Paritätsbit) das nennt sich **Handshake.**
- Die Daten müssen vor dem Senden codiert werden, beim Empfangen decodiert. Allfällige Fehler werden so entdeckt.
- Die Daten müssen "nummeriert" werden, damit sie identifizierbar sind. (Bsp: Sende mir Paket Nr. 3 nochmals)
- Der Sender muss die Pakete eine gewisse Zeit behalten, da er nicht weiss wie lange es dauert bis der Empfänger sich bei einem fehlerhaften Paket meldet. (Sendepuffer)
- Es gibt Protokolle, bei denen alle korrekten Daten bestätigt werden und es gibt Protokolle, bei denen nur die fehlerhaften zurückgemeldet werden. (Acknowledge, ACK)
- Fehlerhafte Daten verlangsamen den kompletten Datenstrom, falls viele Fehler vorkommen, muss der Sender ab und zu eine Pause einlegen. Bei mehreren Empfängern bekommen diese eventuell kurzzeitig keine Daten. Diese Problematik wird mittels einer **Flusskontrolle** behandelt.
- Die vom Sender erneut gesendeten Daten (re-send) müssen vom Empfänger neu eingeordnet werden bevor sie an den Nutzer übermittelt werden. Dies bedingt einen Empfängerpuffer.
- Es muss definiert sein nach wie vielen erfolglosen re-send's die Übertragung als fehlgeschlagen zählt.

Fehlerbehebung (Wiederholungscode, Hammingcode)

Eine elegantere Lösung für die Fehlervermeidung wäre doch, wenn der Empfänger den Fehler selbständig beheben könnte, ohne die Daten erneut anzufordern. Das erneute Senden kann in bestimmten Fällen gar nicht möglich sein:

- Unidirektionale Signale: TV-Signal, Marsroboter,...
- Zeitliche Verzögerung nicht tolerierbar (Audio-CD, DSL,..)
- Das darunterliegende Protokoll erlaubt kein erneutes Senden.



Schema Vorwärtsfehlerkorrektur

Damit die auf dem obigen Bild dargestellte Fehlerkorrektur funktioniert, muss den zu übermittelnden Daten Redundanz hinzugefügt werden.

Wiederholungscode

Eine einfach zu realisierende Variante zur Fehlerbehebung stellt folgendes Konstrukt dar. Wir senden jedes Bit 5 Mal:

Nachricht: 010 Übermittlung: 00000 11111 00000

Bei fehlerhafter Übermittlung kommt vielleicht sowas an beim Empfänger: 01110
Ist eine Fehlererkennung möglich? Ist eine Fehlerkorrektur möglich? Begründen Sie!
Wie würden Sie entscheiden, wie war die Originalnachricht und warum?
Wie gross ist der Mehraufwand bzw. der Zuwachs an Daten n Prozent, welche mit diesem Verfahren anfallen?

Hammingcode S. 418

Codierung mittels Hammingcode beim Absender

Der Ansatz des Wiederholungscode ist an sich nicht übel, der enorme Nachteil ist uns schnell klar, es müssen viel zu viele Daten übermittelt werden. Je weniger Zusatzinformationen (künstliche Redundanz) wir zur Originalnachricht hinzufügen müssen und trotzdem eine Fehlererkennung ermöglichen, desto besser ist das Verfahren.

Originalnachricht: 1001000
1. Schritt: Auffüllen der Originalnachricht in ein Raster:

Nr.	11	10	9	8	7	6	5	4	3	2	1
Data	1	0	0		1	0	0		0		

In diesem Raster sind alle 2er Potenzen leer zu lassen, diese werden im 3. Schritt aufgefüllt und dienen der Redundanz.

2. Schritt: Binäres Addieren aller "1" Positionen:
Im Beispiel haben die beiden Positionen 11 und 7 den Wert "1". Die Nummern der Positionen werden binär codiert und anschliessend addiert. (Allfälliger Übertrag weglassen)

Reihe 11 = 1011
Reihe 7 = 0111
Total = 1100

3. Schritt: Auffüllen des Rasters mit den berechneten Werten:

Nr.	11	10	9	8	7	6	5	4	3	2	1
Data	1	0	0	**1**	1	0	0	**1**	0	**0**	**0**

4. Schritt: Senden der Nachricht:
Die Zeile "Data" wird nun gesendet. die künstliche Redundanz ist nun eingefügt. ***➔4***

De-Codierung mittels Hammingcode beim Empfänger
Der Empfänger soll nun anhand der Nachricht erkennen ob diese korrekt ist und bei Fehlern selber Korrekturen anbringen können.

1. Schritt: Nachricht in Raster abfüllen:
Der Empfänger verwendet das gleiche Raster wie der Sender, in welchem die 2er Potenzen als Redundanz dienen. (Kontrollbits)

Nr.	11	10	9	8	7	6	5	4	3	2	1
Data	1	0	0	**1**	1	0	0	**1**	0	**0**	**0**

2. Schritt: Binäres Addieren aller "1" Positionen:
Auffallend: Das Verfahren ist fast gleich wie beim Absender, der Unterschied liegt darin dass der Empfänger **alle** Bits zusammenzählt.

Reihe 11 = 1011
Reihe 8 = 1000
Reihe 7 = 0111
Reihe 4 = 0100
Total = 0000

Hinweis: Auch bei dieser "Addition" werden keine Überträge gerechnet, es handelt sich also eher um ein "Zählen" der "1" Werte als um eine echte Addition.

Falls das Total genau 0 ergibt, wurden die Daten korrekt übertragen.

3. Schritt: Nutzen der Originaldaten
Die originalen Nutzdaten können aus dem Raster entnommen und weiterverarbeitet werden. In diesem Beispiel ist es die Bitfolge 1001000.

Simulation einer Fehlübertragung

Wir führen die Schritte des Empfängers nochmals durch, diesmal mit falsch übertragenen Daten. Es handelt sich dabei um die falsche Übertragung EINES Bits. Mehrbitfehler können durch den Hammingcode nicht sicher korrigiert werden!

1. Schritt: Nachricht in Raster abfüllen:
Der Empfänger verwendet das gleiche Raster wie der Sender, in welchem die 2er Potenzen als Redundanz dienen. (Kontrollbits)

Nr.	11	10	9	8	7	6	5	4	3	2	1
Data	1	0	1	**1**	1	0	0	**1**	0	**0**	**0**

2. Schritt: Binäres Addieren aller "1" Positionen:
Auffallend: Das Verfahren ist fast gleich wie beim Absender, der Unterschied liegt darin dass der Empfänger **alle** Bits zusammenzählt.

Reihe 11 = 1011
Reihe 9 = 1001
Reihe 8 = 1000
Reihe 7 = 0111
Reihe 4 = 0100
Total = 1001 = 9_{10}

Da das Total nicht 0 ist, wurden die Daten nicht korrekt übertagen. Das fehlerhafte Bit ist also auf Position 9 zu finden.

3. Schritt: Korrektur und Nutzen der Originaldaten

Nr.	11	10	**9**	8	7	6	5	4	3	2	1
Data	1	0	**0**	**1**	1	0	0	**1**	0	**0**	**0**

Die originalen Nutzdaten können aus dem Raster entnommen und weiterverarbeitet werden. In diesem Beispiel ist es die Bitfolge 1001000.

Zeit für uns das Gelernte zu üben ➔***5***

Aufgabenteil

1) BCD Code

Rechnen Sie folgende Zahlen um:

981_{10} ≅ _____(BCD)

347_{10} ≅ _____(BCD)

0110 1000 1001 (BCD) ≅ _____$_{10}$

1110 1011 1111 (BCD) ≅ _____$_{10}$

0111 0010 0011 (BCD) ≅ _____$_{10}$

Nun folgen die Additionen im BCD System und kontrollieren Sie anschliessend mit den Dezimalzahlen:

14 + 61 84 + 27

Vergleichen Sie den Speicherbedarf bei der Speicherung der Zahl 78‘945 in Binär- und BCD-Code. Welches System benötigt wie viel Platz? Begründen Sie!

2) Diverse Codes

1 aus n Codes

Wie viele ungültige Zeichen gibt es beim 1 aus 10 Code? _____

Ist beim 1 aus 10 Code die Redundanz grösser als beim BCD Code? _____

2 aus n Codes

Erstellen Sie eine Tabelle des 2 auf 5 Codes:

Dezimal	2 aus 5 Code
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Wie viele Kombinationen sind beim 2 aus 5 Code redundant? _____

Welcher Hamming Abstand hat der 2 aus 5 Code? _____

Welcher Code hat die höhere Redundanz, "2 aus 5" oder "1 aus 10"? _____

3) Paritätsbit

Wir übertragen Daten mit "even parity", sie erhalten folgende Daten, welche sind richtig übertragen worden?

110011001 ☐	111001111 ☐	000000111 ☐
000000000 ☐	110000101 ☐	001010110 ☐

Können Sie mit dem Paritätsbit auch Fehler korrigieren? _____

Wir erweitern die Idee des Paritätsbits ein wenig, indem wir nach 8 Bytes (Zeilen) jeweils ein Paritätsbit pro Spalte berechnen und dies dem Empfänger auch zusenden: (even parity)

	1	2	3	4	5	6	7	8	Paritätsbit pro Zeile
Zeile 1	1	1	0	0	0	1	1	1	**1**
Zeile 2	0	0	0	1	1	0	1	0	**1**
Zeile 3	0	0	0	0	1	1	0	0	**0**
Zeile 4	1	1	1	1	1	1	1	0	**1**
Zeile 5	1	0	0	1	0	1	1	0	
Zeile 6	0	1	1	0	1	1	0	0	
Zeile 7	1	0	1	1	0	0	1	0	
Zeile 8	1	1	0	0	0	0	1	1	
Paritätsbit pro Spalte	**1**	**0**							(leer)

Füllen Sie die fehlenden Paritätsbits auf!
Stellen Sie sich nun vor das 6. Bit der 6. Zeile würde versehentlich falsch als 0 übertragen, was passiert beim Empfänger?

Wie gross (in Prozent) ist der Mehraufwand der Datenübertragung in so einem System?

4) Hammingcode beim Sender

Zeichnen Sie ein Raster für das Versenden von 16Bits:

Wie viele zusätzliche Bits haben Sie benötigt? _____

Berechnen Sie den Zuwachs an Bits in Prozent und vergleichen Sie mit dem Wiederholungsverfahren!

Erstellen Sie für die folgenden Daten den Hammingcode des Absenders **von Hand**!

Originaldaten	Hammingcode
11110101	

Weiterführende Themen für Fortgeschrittene

Suchen Sie Informationen zum **Gray Code** und überlegen Sie wozu dieser nützlich sein könnte. Überlegen Sie sich wie es mit der Redundanz, dem Hamming Abstand und der Fehlererkennung aussieht.

Suchen Sie Informationen zum Hammingcode (nicht zu verwechseln mit dem Hamming Abstand).
Was ist der grosse Vorteil des Hamming Code im Vergleich zum Paritätsbit?

Erfinden Sie eine Prüfung mit 2 Paritätsbits. Am besten eine mit der Möglichkeit Fehler zu korrigieren.