

Grundlagen zyklischer Codes

Ausarbeitung zum Seminarthema 'Codes'

Vortragender, geschrieben von: Frank Schlözer, mi5646, FH Wedel

Inhaltsverzeichnis

1 Einleitung.....	1
1.1 Grundsätze zu meiner Erzählweise.....	2
1.2 Worum geht's jetzt also?.....	2
2 Grundlagen.....	3
2.1 Informatik, Binärcodes und Bits.....	3
2.2 Mindestabstand.....	4
2.3 Hammingcode.....	6
3. Zyklische Codes.....	8
3.1 Eine erste Annäherung mit Primzahlen.....	8
3.2 Verwendung von Polynomen.....	9
3.3 Fehlerkorrektur bei zyklischen Codes.....	11
3.4 Schieberegister.....	13
3.5 irreduzible Polynome.....	14

1 Einleitung

Damit Informationen im digitalen Zeitalter problemlos übertragen werden können, benötigt man einerseits eine fehlerfreie Übertragung und andererseits eine ökonomische, also sparsame Übertragung. Eine ideale Übertragung ist fehlerfrei und möglichst wenig redundant; d.h. ohne überflüssige Informationen. Es sei gleich vorweg gesagt, dass Fehlerkorrektur niemals 100% funktioniert und immer ein Ausgleich zwischen Informationsrate und Redundanz ist. Nicht fehleranfällige Codes, d.h. Codes, die viel Redundanz aufweisen, haben auf der anderen Seite wenig Informationsgehalt und sind deswegen unökonomisch in Bezug auf die Auslastung des Übertragungskanals. Codes, die viel Informationsgehalt haben, sind zwar ökonomisch in Bezug auf den Übertragungskanal, aber sind dafür fehleranfälliger. Dabei ist die Codierung, also die Umwandlung der Informationen in Codewörter (mir Hilfe von mathematischen Tricks), entscheidend. Eine Codierung kann z.B. alle Informationen mehrfach versenden, um eine fehlerlose Übertragung sicher zu stellen. So kann der Empfänger den Fehler erkennen und ggf. die Originalinformation rekonstruieren. So eine 'Codierung' ist simpel zu verstehen, allerdings relativ primitiv und soll nur als einleitendes Beispiel dienen. Bei doppelter oder dreifacher Wiederholung hätte man eine hohe Redundanz und somit würde die Übertragung recht lange dauern und man würde unnötig viel Kapazität des Übertragungskanals 'verbraten'. Bei einfachen Texten und einer schnellen Internetverbindung mag die dreifache Übertragung einer jeden Information zwar nur Millisekunden länger dauern; bei der Übertragung eines Spielfilmes sieht das Ganze aber schon ganz anders aus. Glücklicherweise gibt es viel intelligentere Codes, die einen guten Ausgleich zwischen Redundanz und Informationsgehalt bieten. Eine schlaue Art der Codierung, die sich zudem speziell für die heutige verwendete Computerhardware eignet, ist die 'zyklische Codierung'; darüber will ich hier berichten. Um die zyklische Codierung zu verstehen, will ich mich dem Thema langsam nähern und zuerst einige Grundlagen klären. Danach arbeiten wir uns über den Hammingabstand und dem Hammingcode schließlich

zur zyklischen Codierung vor. Ich versuchen, möglichst wenig Zeit mit den Grundlagen zu verbringen, auch, da dies das Thema eines anderen Vortrags ist; jedoch sind einige Grundlagen unabdingbar für das Verständnis von zyklischen Codes.

1.1 Grundsätze zu meiner Erzählweise

Ich versuche, die Informationen möglichst einfach darzustellen, damit auch ein Nicht-Mathematiker verstehen kann, worum es hier geht. Einer meiner Grundsätze ist eine möglichst geradlinige, benutzerfreundliche Erklärung, damit Jeder in den Genuss von Informationen kommen kann. Nicht erklärte Fremdwörter und Formeln erschrecken und langweilen das Publikum und fördern Unverständnis. Ich möchte das Publikum nicht beeindrucken, sondern möchte, dass die Leute hinterher etwas schlauer sind als vorher. Zu diesem Zwecke starte ich ganz am Anfang, auch auf die Gefahr hin, auf Experten als Laie interpretiert zu werden. Sie werden sehen, dass zyklische Codes im Detail sehr komplex sind; ich werde aber immer versuchen, Komplexität möglichst simpel zu halten!! Außerdem werde ich hier und da das geschriebene Wort durch ein paar – hoffentlich lustige – Worte auflockern, da das Thema doch sehr trocken ist und man sonst wegen der ganzen Mathematik schnell das Interesse verliert.

1.2 Worum geht's jetzt also?

Das Thema dieses Vortrags ist 'zyklische Codes'. Ein Code ist eine Art, um Informationen darzustellen, zu speichern und zu übertragen. Ein frühes Beispiel eines Codes ist der Morsecode. Informationen wurden in zusammenhängenden 'Chunks' übertragen, die einzelne Buchstaben repräsentierten. Dabei hatten häufig vorkommende Buchstaben kürzere Codewörter als selten auftretende Buchstaben. Übertrug der Funker also damals ein 'E', musste er die Morsetaste nur einmal kurz drücken. Wollte er hingegen ein 'X' übertragen, musste er die Morsetaste einmal kurz, dann zweimal lang und dann wieder einmal kurz drücken. Der Morsecode war also ein Code variabler Länge (und damit kein Blockcode) und der Auftrittswahrscheinlichkeit der Buchstaben grob angepasst. Der Morsecode war also insofern schon recht 'pfiffig'.

Der Morsecode hatte aber keine Möglichkeit, einen Fehler zu korrigieren. Wenn man z.B. von der einen Seite des Atlantiks zur anderen Seite des Atlantiks einen Text morsen wollte und unterwegs traten Störungen auf der Leitung auf (z.B. ein Hai biss ins Kabel), dann konnte man auf der anderen Seite des Atlantiks nicht erkennen, ob ein Fehler aufgetreten war, oder ob vielleicht doch ein 'T' (einmal lang) anstatt eines 'E' (einmal kurz) übertragen werden sollte. In Zeiten des Morsecodes konnte der Funker noch den Fehler mit Hilfe des Sinnzusammenhangs korrigieren; in der heutigen Zeit verfügt die Maschine aber (noch) nicht über die Fähigkeit, abstrakt zu denken.

Ein guter Code braucht also auch Fehlererkennung und muss das Korrigieren von Fehlern ermöglichen! Fehlerkorrektur ist generell ein wichtiger Aspekt von Codes und 'zyklische Codes' sind ein Teil eben dieser Fehlerkorrektur, worauf ich mich in diesem Vortrag konzentrieren will. Es sei nur kurz angemerkt, dass es neben der Fehlerkorrektur noch zwei andere wichtige Themen bezüglich Codes gibt: nämlich die Kompression, also die Reduzierung von Datenmengen, und die Verschlüsselung, also das Umwandeln eines lesbaren Klartextes in einen nicht-lesbaren, verschlüsselten Text, der aber mit dem richtigen Passwort wieder rekonstruiert werden kann.

Unser Thema sind aber zyklische Codes, um Fehler zu erkennen und zu korrigieren. Heutzutage werden überall Informationen übertragen. In der Waschmaschine, im Toaster, von der DVD zum Prozessor im DVD-Player, zum DVB-T Fernseher, von der Vermittlungsstelle der Telekom zum DSL-Modem im eigenen Haus; überall werden Informationen übertragen. Die Informationen werden codiert und dann mit einem elektrischen Signal in Form von Spannungen auf das Übertragungsmedium (z.B. Kupferkabel) gebracht. Den Vorgang der Übertragung des digitalen Signals in ein analoges, elektrisches Signal nennt man auch Modulation. Beim Empfänger wird es dann wieder in ein digitales Signal zurück gewandelt. Aber eben bei diesem Vorgang können Fehler auftreten. Auf dem Kupferkabel des Telefons kann z.B. eine Überspannung vorliegen. Außerdem gibt es immer ein mehr oder minder starkes so genanntes 'Hintergrundrauschen', das das physikalische, elektronische Signal verfälscht. Wenn z.B. gerade irgendwo im Haus eine Lampe angeschaltet wird, kann sich die Spannung kurzzeitig auch auf das Kupferkabel des Telefons übertragen und im Modem kommt dann statt einer '0' plötzlich eine '1' an.

Eine sehr gebräuchliche und anschauliche Art, wie Codes praktisch eingesetzt werden, ist die Übertragung von Informationen aus dem Internet. Uns müssen dabei die technischen Einzelheiten, wie genau das Signal moduliert wird und was das DSL-Modem technisch tut, nicht interessieren. Uns interessiert nur die Information aus der Sicht des Informatikers und Mathematikers. Eine Information steht also irgendwo auf einer Webseite im entfernten China. Diese Information ist HTML-Quelltext, also eine Aneinanderreihung von Buchstaben und Zeichen, vom Prinzip her genauso wie die Seite eines Buches. Dabei läuft die Information durch tausende von Kilometern von Glasfaserkabeln und Kupferkabeln. Zwangsläufig geht bei dieser Aktion irgendwo was schief und die Information kommt beim Empfänger in abgewandelter Form an. Die Aufgabe eines vernünftigen Codes ist also die fehlerfreie Übertragung eben dieser Buchseite bis zum Computer, der irgendwo in Deutschland steht. Der Code muss mit genug intelligent gewählter Zusatzinformation versehen sein, so dass die Information beim Empfänger relativ fehlerfrei rekonstruiert werden kann (weniger als 1 Fehlerbit auf eine Million Bits ist ein akzeptabler Wert).

2 Grundlagen

2.1 Informatik, Binärcodes und Bits

In der Informatik werden Bits verwendet. Eine '1' steht für 'ein' oder 'Strom fließt'; eine '0' hingegen für 'aus' oder 'Strom fließt nicht'. Auf einem Kupferkabel würde man dann in sehr schneller Folge ein Volt für '1' und -1 Volt für '0' oder 'aus' durch jagen. Ein Bit kann also entweder eine '0' oder eine '1' sein. Ein Byte ist dabei eine Aneinanderreihung von 8 Bits. Bei einem Byte gibt es $2^8 = 256$ Möglichkeiten, die Bits aneinander zu reihen. Buchstaben wurden früher (und noch heute) im so genannten ASCII-Code übertragen. Ein Buchstabe nimmt dabei den Platz von 8 Bit ein, also einem Byte. Jeder Buchstabe bekommt dabei eine Nummer und diese wird in die Binärdarstellung umgewandelt (z.B. **10000100** für den Buchstaben 'b').

2.2 Mindestabstand

Eine Grundlage, um zyklische Codes zu verstehen, ist der Mindestabstand. Der Mindestabstand ist die geringste Differenz (in Anzahl Bits) zwischen allen gültigen Codewörtern eines Codes. Es ist ja so, dass man Fehler nur erkennen kann, wenn nicht der ganze Vorrat an Codewörtern eines Codes genutzt wird. Tritt nun eine Kombination von Bits auf, die nicht einem gültigen Codewort entspricht, weiß man zumindest, dass ein Fehler aufgetreten ist.

Als Beispiel benutzen wir mal zwei Bits. Wir einigen uns darauf, dass '00' 'aus' bedeutet, und '11' 'ein' entspricht. Senden wir nun '00' durch die Leitung, aber am anderen Ende kommt '01' heraus, wissen wir, dass ein Fehler aufgetreten ist, denn für '01' gibt es in unserer (gedachten) Tabelle keine Entsprechung. Wir können die ursprüngliche Information zwar nicht rekonstruieren, aber wir wissen zumindest, dass ein Fehler aufgetreten ist.

Der Mindestabstand dieses Codes ist 2. Warum ist das so? Nun, weil der bitweise Abstand zwischen allen gültigen Codewörtern (00,11) des Codes mindestens 2 beträgt. An der ersten Stelle unterscheiden sich die Bits und an der zweiten Stelle unterscheiden sich die Bits ebenfalls.

Ab einem Mindestabstand von 3 lassen sich auch Fehler korrigieren. Nehmen wir als Beispiel mal eine Aneinanderreihung von 3 Bits. Wenn wir nun nur 2 gültige Codewörter, nämlich '111' und '000' haben, dann lassen sich Fehler wie folgt korrigieren: Kommt nun beim Empfänger eine '101' an, dann ist dieses Codewort kein gültiges Codewort, soviel ist klar. Gleichzeitig ist aber '101' näher an '111' als an '000', denn zu '101' ist der Hammingabstand 1 und zu '000' ist der Hammingabstand 2. Wenn nun nicht gerade ein großes Unglück passiert ist (bei der Übertragung könnte ein Doppelfehler aufgetreten sein), dann lässt sich '101' zu '111' zuordnen. Es lässt sich also 1 Fehler korrigieren. Man sieht also, dass die Codierung mit Hilfe des Mindestabstandes keine absolute Sicherheit vor Fehlern garantiert. Dies ist aber grundsätzlich bei einem fehlerbehafteten Übertragungskanal nicht möglich. Ein Codewort kann durch Zufall immer so verändert werden, dass ein anderes, gültiges Codewort entsteht, obwohl das bei entsprechend groß gewähltem Mindestabstand immer unwahrscheinlicher wird.

Für den Mindestabstand gibt es einige Formeln, die man sich merken sollte:

- Mindestabstand, um mindestens t_{korr} Fehler zu korrigieren:

$$d_{\text{min}} \geq 2 * t_{\text{korr}} + 1$$

Um einen Fehler korrigieren zu können, braucht man also mindestens einen Mindestabstand von 3 (d_{min} ist größer oder gleich $2 * 1 + 1$)

- Mindestabstand, um mindestens t_{erk} Fehler zu erkennen:

$$d_{\text{min}} \geq t_{\text{erk}} + 1$$

Um einen Fehler erkennen zu können, braucht man also mindestens einen Mindestabstand von 2 (d_{min} ist größer oder gleich $1 + 1$)

Letztendlich geht es bei der Fehlerkorrektur und Fehlererkennung (in Zukunft jetzt nur noch Fehlerkorrektur) also um das Finden von geeigneten Codes, dessen gültige

Codewörter einen gewissen Mindestabstand besitzen. Bei 2 Bits und einen Mindestabstand von 2 ist das vielleicht noch einfach. Aber bereits bei 3 Bits und einem gesuchten Mindestabstand von 2 wird das Ganze etwas schwieriger. Ich erspare uns jetzt Details, aber letztendlich muss man mit Hilfe einer systematischen Vorgehensweise und einer Tabelle geeignete Codewörter suchen, dann andere Codewörter herausstreichen, dann wieder neu suchen u.s.w., bis man die maximale Anzahl von geeigneten Codewörtern, die untereinander einen Mindestabstand von 2 haben, gefunden hat. Der Aufwand wächst exponentiell; bei entsprechender Codewortlänge kann man also sehr schnell verzweifeln. Man kann sich hier höchstens mit Computerprogrammen behelfen.

Letztendlich braucht man aber für die Generierung von Codes mit einem Mindestabstand, der Fehlerkorrektur ermöglicht, eine Berechnungsvorschrift. Genau hier kommt der Hammingcode ins Spiel, ein 'Vorläufer' der zyklischen Codes und wichtig, um die zyklischen Codes genauer zu verstehen.

2.3 Hammingcode

Ja, wer war eigentlich dieser Richard Hamming??



Dank Wikipedia sind wir heute ja allwissend und haben sogar ein Foto von dem Herrn Hamming vorrätig. Laut Legende lebte der Herr Hamming in den 40er Jahren in einem Wald voller elektromechanischer Arrays der Firma Bell Labs, die er eifrig mit Lochkarten fütterte. Leider waren eben diese Arrays sehr fehleranfällig und lasen nach einer gewissen Zeit, bedingt durch mechanische Abnutzung, die Lochkarten nur noch fehlerhaft. Dies führte dazu, dass fehlerhafte Lochkarten übersprungen wurden und im Anschluss zu aufwändigen Zusatzarbeiten, bei der ein Mitarbeiter allein mit der Neusortierung der Lochkarten beschäftigt werden musste. Folglich lies sich Herr Hamming den Hammingcode einfallen, damit die Maschine Fehler selbst erkennen und korrigieren konnte. In den 50er Jahren schließlich wurde der Hammingcode publiziert.

Der Hammingcode ist ein Blockcode (d.h. Alle Codewörter haben die gleiche Länge) und hat stets einen Mindestabstand von 3. Folglich kann der Hammingcode einen Fehler erkennen und korrigieren; allerdings können Doppelfehler (und weitere Fehler gerader Anzahl) nicht erkannt oder korrigiert werden. Es gibt auch eine Erweiterung des Hammingcodes, der einen Mindestabstand von 4 hat. Wir betrachten aber nur den ursprünglichen Hammingcode.

0	1	1	0	1	1	0
k Informationsbits				m Prüfbits		
Codewortlänge n						
u ₁	u ₂	u ₃	u ₄	y ₁	y ₂	y ₃

$$n = 2^m - 1$$

$$k = n - m$$

Der Hammingcode ist ein recht effizienter Code, bei dem die Informationsbits, also die Träger der eigentlichen Informationsbits, durch so genannte Prüfbits ergänzt werden. Die Anzahl der Prüfbits bezeichnen wir mit **m**. Durch die Anzahl der Prüfbits lässt sich die Codewortlänge errechnen; diese beträgt nämlich $n = 2^m - 1$. Ein Hammingcode mit 3 Prüfstellen hat also eine Codewortlänge $n = 2^3 - 1 = 7$ Bits. Damit bleiben also bei 3 Prüfbits 4 Bits für die eigentliche Information. Der Hammingcode wird also mit wachsender Codewortlänge effizienter in Hinsicht auf die Informationsrate. Bei einer Anzahl von 5 Prüfbits gibt es schon 26 Informationsbits; bei 6 Prüfbits gibt es schon 57 Informationsbits. Dafür, dass dieser Code einen Fehler korrigieren kann, ist er schon wesentlich effizienter als z.B. das dreifache Senden eines jeden Bits. Allerdings ist für das Senden (und Empfangen) relativ viel Rechenaufwand notwendig, was berücksichtigt werden muss. Der Rechenaufwand wächst stark mit der Anzahl der Prüfbits. Viel wichtiger ist in diesem

Zusammenhang allerdings das verstärkte Auftreten von Fehlern (bei größerer Codewortlänge)!! Da mit dem Hammingcode nur ein Fehler pro Codewort erkannt und korrigiert werden kann, kann der Code bei größerer Codewortlänge nutzlos werden, da eventuell 2 Fehler gleichzeitig auftreten. Man muss sich also vorher über die Beschaffenheit des Übertragungsmediums bzw. der Hardware informieren.

Die Codewörter des Hammingcodes werden mit einer bestimmten Systematik gefunden, auf die wir hier nicht weiter eingehen wollen. Wichtig ist nur, zu wissen, dass die Prüfbits jeweils mit Hilfe von m voneinander unabhängigen Gleichungen errechnet werden:

$$y1 = (u1 + u2 + u4) \text{ mod } 2$$

$$y2 = (u1 + u3 + u4) \text{ mod } 2$$

$$y3 = (u2 + u3 + u4) \text{ mod } 2$$

Diese Gleichungen ermöglichen nicht nur das Erkennen eines Fehlers, sondern auch die Korrektur, da sich aus den so genannten Syndromgleichungen auch die exakte Fehlerposition errechnen lässt.

Wir können das mal für das vorangegangene 7-Bit Codewort ausprobieren. Statt dem korrekten Codewort:

0	1	1	0	1	1	0
---	---	---	---	---	---	---

bauen wir an der dritten Stelle einen Fehler ein und erhalten:

0	1	0	0	1	1	0
u_1	u_2	u_3	u_4	y_1	y_2	y_3

Wir errechnen die Syndromgleichungen und erhalten:

$$y1 = (0 + 1 + 0) \text{ mod } 2 = 1$$

$$y2 = (0 + 0 + 0) \text{ mod } 2 = 0$$

$$y3 = (1 + 0 + 0) \text{ mod } 2 = 1$$

Jetzt können wir folgende interessante Feststellung machen: Nur eine der Syndromgleichungen, nämlich y_1 , ist korrekt. y_2 und y_3 sind dagegen falsch. Daraus können wir wiederum folgern, dass der Fehler bei u_3 liegen muss, da u_3 sowohl in y_2 und y_3 den Fehler produziert, nicht aber in y_1 als Variable auftritt, weswegen das Ergebnis von y_1 ja auch korrekt ist. Da der Fehler also bei u_1 lokalisiert wurde und die richtige Zahl im Binärsystem ohnehin nur das Gegenstück zur Zahl '0', also '1' sein kann, ist die korrigierte Zahl also '1' und damit der Fehler korrigiert.

Das Hammingcode-Verfahren lässt sich mit Matrizen noch weiter vereinfachen. Dabei stellt man alle Elemente (Informationsbits, Prüfbits, ect.) als Elemente von Vektoren dar. Dies hat einige mathematische und technische Vorteile, auf die wir aber hier nicht weiter eingehen wollen.

3. Zyklische Codes

Der zyklische Code ist eine Weiterentwicklung des Hammingcodes. Der zyklische Code hat den Vorteil, dass durch eine zyklische Verschiebung der Bits des Codewortes ein neues, gültiges Codewort entsteht. Eine Verschiebung eines ungültigen Codewortes hingegen erzeugt nur ungültige Codewörter. Zyklische Verschiebung kann sehr gut in heutiger Hardware implementiert werden, nämlich in Schieberegistern. Außerdem kann man mit Hilfe von zyklischen Codes beliebig viele Fehler korrigieren.

3.1 Eine erste Annäherung mit Primzahlen

Zyklische Codes kann man gut mit Hilfe folgender Überlegung erklären: Es wird eine 'Generatorzahl' gewählt, die eine Primzahl sein muss. Eine Primzahl hat die Eigenschaft, dass sie nur durch 1 und durch sich selbst teilbar ist und nicht in andere Faktoren zerlegbar ist. Das Ziel ist es ja, mit Hilfe der Prüfbits den Fehler genau zu lokalisieren, genau so wie beim Hammingcode. Die Idee ist nun, die Information (ohne sie zu verändern) in ein Format zu bringen, so dass die neue Information ganzzahlig ohne Rest durch die Primzahl teilbar ist. Wenn nun beim Empfänger die Information *nicht* ganzzahlig ohne Rest durch die Primzahl teilbar ist, wissen wir, dass ein Fehler aufgetreten ist.

Zu wünschen wäre jetzt noch, dass bei einem Fehler ein einzigartiger Rest herauskommt, so dass wir eindeutig auf die Fehlerposition schließen können. Das ist aber (mit einer Primzahl als Generator) zu viel des Guten, wie wir gleich beim Beispiel sehen werden.

Dabei hat der zyklische Code einen ähnlichen Aufbau wie der Hammingcode. Das Codewort besteht wieder aus k Informationsstellen und m Prüfbits. Insgesamt hat das Codewort eine Länge von n Bits.

1	0	0	0	0	1	1	1	1	0	1	1
k Informationsbits								m Prüfbits			
Codewortlänge n											

Als Beispiel wählen wir

Generatorzahl	13_{10}	->					1	1	0	1
Information	135_{10}	->	1	0	0	0	0	1	1	1

Zuerst hängen wir 4 Stellen (Anzahl Bits Generatorzahl) an das Informationswort heran. Somit wird die Information selbst nicht verändert, denn diese ist in den ersten 8 Bit des neuen Informationsworts unverändert vorhanden.

1	0	0	0	0	1	1	1					->	135_{10}
1	0	0	0	0	1	1	1	0	0	0	0	->	2160_{10}

Das Codewort ist aber noch nicht fertig. Wir möchten ja, dass das Codewort ganzzahlig ohne Rest durch 13 teilbar ist. Da aber $2160 \bmod 13 = 2$ ist, müssen wir

$13 - 2 = 11$ auf die Zahl addieren. Schließlich erhalten wir

	2160 ₁₀	1	0	0	0	0	1	1	1	0	0	0	0
+	11 ₁₀									1	0	1	1
	2171 ₁₀	1	0	0	0	0	1	1	1	1	0	1	1

Wenn diese Zahl nun übertragen wird, kann der Empfänger mit der einfachen Rechnung (modulo 13) prüfen, ob bei der Übertragung ein Fehler aufgetreten ist.

Falls Selbiger aufgetreten ist, ist die übermittelte Zahl nie ganzzahlig ohne Rest durch 13 teilbar.

Leider macht die Codierung mit ganzen Zahlen aber keinen Sinn, da man nicht eindeutig auf die Fehlerposition schließen kann, wie man anhand der folgenden Tabelle sehen kann. Fehler an unterschiedlichen Bitpositionen erzeugen den gleichen 'Fehlercode'.

	WORT												WERT10	MOD 13
Fehlerstelle	2048	1024	512	256	128	64	32	16	8	4	2	1		
12	0	0	0	0	0	1	1	1	1	0	1	1	123	6
11	1	1	0	0	0	1	1	1	1	0	1	1	3195	10
10	1	0	1	0	0	1	1	1	1	0	1	1	2683	5
9	1	0	0	1	0	1	1	1	1	0	1	1	2427	9
8	1	0	0	0	1	1	1	1	1	0	1	1	2299	11
7	1	0	0	0	0	0	1	1	1	0	1	1	2107	1
6	1	0	0	0	0	1	0	1	1	0	1	1	2139	7
5	1	0	0	0	0	1	1	0	1	0	1	1	2155	10
4	1	0	0	0	0	1	1	1	0	0	1	1	2163	5
3	1	0	0	0	0	1	1	1	1	1	1	1	2175	4
2	1	0	0	0	0	1	1	1	1	0	0	1	2169	11
1	1	0	0	0	0	1	1	1	1	0	1	0	2170	12

3.2 Verwendung von Polynomen

Deswegen werden für die zyklische Codierung Polynome verwendet! Zur Erinnerung hier ein zufällig gewähltes Polynom 4. Grades: $1 * x^4 + 0 * x^3 + 1 * x^2 + 0 * x + 1$

Man benutzt dabei Polynome mit Koeffizienten aus endlichen Körpern, genauer gesagt aus dem Körper Z_2 . Das macht Sinn, weil wir die Information ja ohnehin in die Binärdarstellung umwandeln und somit den Bits einfach Koeffizienten zuordnen. Das Bit an der Position 1 wird dann zum Koeffizienten von $x^0 = 1$, das Bit an der zweiten Stelle der Koeffizient von x^1 u.s.w.

Bei der Vorgehensweise mit Polynomen brauchen wir nun, anstatt einer Zahl, ein Infopolynom. Wir nennen u die zu übertragende Information, und $u(x)$ das neue Infopolynom. Dabei sind die Bits von u , wie bereits erwähnt, die Koeffizienten des Infopolynoms. Als Beispiel benutzen wir

u	1	0	0	1
daraus wird				
$u(x) = 1 * x^3 + 0 * x^2 + 0 * x + 1$				

g	1	1	0	1
daraus wird				
$g(x) = 1 * x^3 + 1 * x^2 + 0 * x + 1$				

Wie genau die Generatorpolynome gefunden werden und welche Generatorpolynome überhaupt in Frage kommen, wird später noch kurz angerissen. Im Moment gehen wir aber davon aus, dass die Generatorpolynome schon bekannt sind. Folgende Polynome können verwendet werden, sortiert nach m . m ist dabei der Grad von g .

$m=1$	$x^1 + 1$
$m=2$	$x^2 + x^1 + 1$
$m=3$	$x^3 + x^1 + 1$ und $x^3 + x^2 + 1$
$m=4$	$x^4 + x^1 + 1$ und $x^4 + x^3 + x^2 + x^1 + 1$ und $x^4 + x^3 + 1$

Wenn die Infozahl k Bits hat, ist der Grad des Infopolynoms $k-1$. Die Generatorzahl hat m Bits und das Generatorpolynom den Grad $m-1$.

Jetzt müssen wir also, ähnlich der vorangegangenen Vorgehensweise mit den ganzen Zahlen, erstmal den *Rest modulo Generatorpolynom* finden und diesen dann auf das Infopolynom dazu addieren, um ein Codepolynom zu erhalten, das ohne Rest durch das Generatorpolynom geteilt werden kann. Es geht also zunächst darum, den Rest $h(x)$ herauszufinden, damit wir ihn im nächsten Schritt auf das Infopolynom addieren können.

Dafür müssen wir beim Infopolynom erstmal 'Platz schaffen' und müssen genug Stellen hinzufügen, genauer gesagt m Stellen. Das geht am Einfachsten, wenn wir das Infopolynom mit x^m multiplizieren.

Nun zum Beispiel zurück. Wir müssen also erstmal das Infopolynom

$$u(x) = 1 * x^3 + 0 * x^2 + 0 * x + 1 \text{ mit } x^3 \text{ multiplizieren:}$$

$$u'(x) = u(x) * x^3 = 1 * x^6 + 0 * x^5 + 0 * x^4 + 1 * x^3 + 0 * x^2 + 0 * x + 0$$

Jetzt müssen wir $u'(x)$ durch $g(x)$ dividieren, um $h(x)$ zu erhalten!

Man beachte, dass beim Körper Z_2 Addition der Subtraktion entspricht!

$$\begin{array}{r}
 1*x^6+0*x^5+0*x^4+1*x^3+0*x^2+0*x+0 : 1*x^3+1*x^2+0*x+1 = 1*x^3+1*x^2+1*x+1 \\
 1*x^6+1*x^5+0*x^4+1*x^3+0*x^2+0*x+0 \\
 \hline
 1*x^5+0*x^4+0*x^3+0*x^2+0*x+0 \\
 1*x^5+1*x^4+0*x^3+1*x^2+0*x+0 \\
 \hline
 1*x^4+0*x^3+1*x^2+0*x+0 \\
 1*x^4+1*x^3+0*x^2+1*x+0 \\
 \hline
 1*x^3+1*x^2+1*x+0 \\
 1*x^3+1*x^2+0*x+1 \\
 \hline
 \mathbf{h(x)=1*x+1}
 \end{array}$$

$h(x)$ wird nun auf $u'(x)$ aufaddiert, um $v(x)$ (das Codewortpolynom) zu erhalten. Nun erhalten wir also

$$v(x) = 1*x^6 + 0*x^5 + 0*x^4 + 1*x^3 + 0*x^2 + 1*x + 1$$

Dieses Codewortpolynom ist nun ganzzahlig und ohne Rest durch $g(x)$ teilbar.

In binärer Darstellung sieht das Codewort so aus:

1	0	0	1	0	1	1
u ist noch unverändert im Codewort enthalten				m Prüfbits		

3.3 Fehlerkorrektur bei zyklischen Codes

Bei einem zyklischen Code kann man nun Fehler korrigieren. Ein zyklischer Code hat also in der einfachsten Form den Hammingabstand 3. Anders als bei den ersten zyklischen Versuchen mit einer Primzahl als Generator, gibt es nun auch keine Doppelbelegung von Fehlerquellen mehr.

Tritt bei der Datenübertragung ein Fehler auf und ein Bit wird 'umgedreht', lässt sich der Fehler nun rekonstruieren, und zwar auf folgende Weise:

Das Wort, das auf der anderen Seite der Leitung ankommt, wird ja *modulo Generatorpolynom* gerechnet. Nur wenn die Zahl ohne Rest durch das Generatorpolynom teilbar ist, ist Alles richtig (sofern kein Doppelfehler aufgetreten ist). Wenn ein anderer Rest herauskommt, kann man in einer Tabelle nach schauen, dessen Werte man zuvor errechnet hat. Jeder Rest ist eindeutig einer Fehlerposition zuordnungsbar.

Schauen wir uns nun ein Beispiel an. Als Codewort nehmen wir mal

0	1	1	1	0	1	0
Originalinformation				3 Prüfbits		

Als Generatorpolynom verwenden wir $x^3 + x^1 + 1$.

Die Tabelle mit der eindeutigen 'Fehlerposition – Rest' Zuordnung sieht wie folgt aus:

Fehlerposition	Rest
kein Fehler	0
1	1
2	x^1
3	x^2
4	$x^1 + 1$
5	$x^2 + x^1$
6	$x^2 + x^1 + 1$
7	$x^2 + 1$

Nacheinander habe ich an der 7., 6. und 5. Stelle ein Bit umgedreht und dann geschaut, was bei einer Division des Fehlerwortes *modulo Generatorpolynom* herauskommt. Man liest die Stelle übrigens immer von rechts nach links.

Beispiel: 7. Stelle fehlerhaft

7	6	5	4	3	2	1
1	1	1	1	0	1	0
Originalinformation				3 Prüfbits		

$$1*x^6 + 1*x^5 + 1*x^4 + 1*x^3 + 0*x^2 + 1*x + 0 : 1*x^3 + 0*x^2 + 1*x + 1 = 1*x^3 + 1*x^2 + 1$$

$$1*x^6 + 0*x^5 + 1*x^4 + 1*x^3 + 0*x^2 + 0*x + 0$$

.....

$$1*x^5 + 0*x^4 + 0*x^3 + 0*x^2 + 1*x + 0$$

$$1*x^5 + 0*x^4 + 1*x^3 + 1*x^2 + 0*x + 0$$

.....

$$1*x^3 + 1*x^2 + 1*x + 0$$

$$1*x^3 + 0*x^2 + 1*x + 1$$

.....

$h(x) = 1*x^2 + 1$ → Fehler trat an Stelle 7 auf.

Beispiel: 6. Stelle fehlerhaft

7	6	5	4	3	2	1
0	0	1	1	0	1	0
Originalinformation				3 Prüfbits		

$$0 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 0 : 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1 = 1 \cdot x + 1$$

$$0 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x + 0$$

.....

$$1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 0$$

$$1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1$$

.....

$$h(x) = 1 \cdot x^2 + 1 \cdot x^1 + 1 \rightarrow \text{Fehler trat an Stelle 6 auf.}$$

Beispiel: 5. Stelle fehlerhaft

7	6	5	4	3	2	1
1	1	1	1	0	1	0
Originalinformation				3 Prüfbits		

$$0 \cdot x^6 + 1 \cdot x^5 + 0 \cdot x^4 + 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 0 : 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1 = 1 \cdot x^2$$

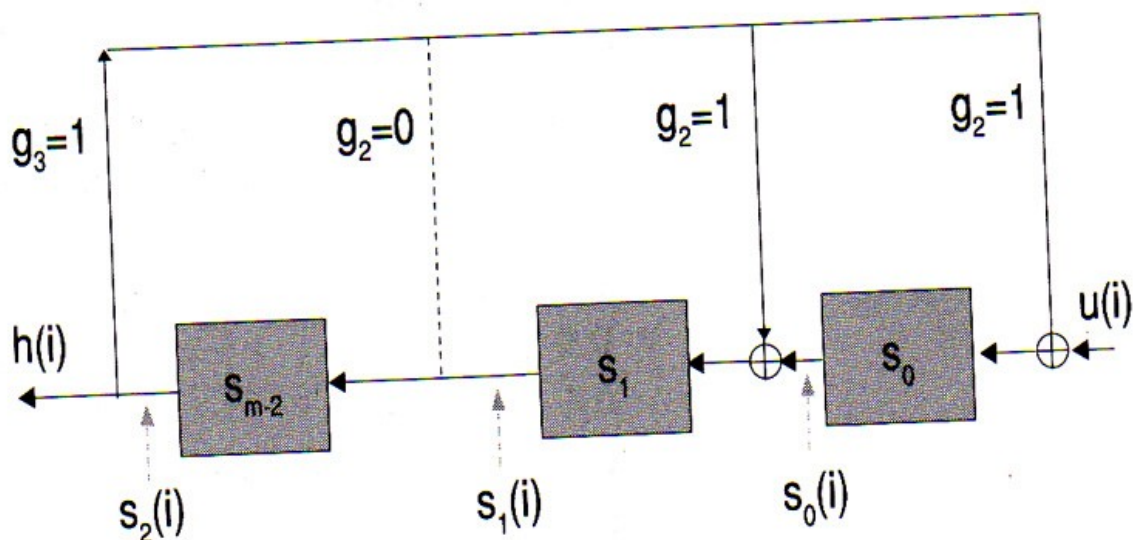
$$0 \cdot x^6 + 1 \cdot x^5 + 0 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 0$$

.....

$$h(x) = 1 \cdot x^2 + 1 \cdot x^1 + 0 \rightarrow \text{Fehler trat an Stelle 5 auf.}$$

3.4 Schieberegister

Der eigentliche Vorteil ist aber, dass man zyklische Codes technisch sehr einfach mit so genannten *rückgekoppelten Schieberegistern* erzeugen kann.



Dabei ist das Generatorpolynom fest in die Hardware 'verbaut', indem es bei einer '1' eine Rückkopplungsverbindung gibt, bei einer '0' hingegen nicht (gestrichelte Linie). Die Rückkopplung ist mit einem XOR-Bauteil verbunden (Fadenkreuz), welches technisch das Gleiche bewirkt, wie ein MOD-2 Addierer. Der Ausgang geht auf 1, wenn beide Eingangssignale verschieden sind; 0 ansonsten.

So lässt sich nach n (Anzahl Bits Codewort) Takten $h(x)$ bestimmen. Das geht also mit solchen Schieberegistern sehr, sehr schnell und simpel.

Solche Schieberegister erzeugen die Codes systematisch, d.h. man kann von der Bitfolge immer sehr schnell auf das originale Infowort schließen, da die Infobits unverändert am Anfang des Codeworts stehen und die Prüfbits am Ende. Man kann den Code auch unsystematisch generieren, aber das hat so gut wie keine praktische Bedeutung. Bei unsystematischen Codewörtern kann man das originale Infowort nicht (ohne Zwischenschritte) extrahieren.

3.5 irreduzible Polynome

Ich hatte schon vorher angedeutet, dass das Generatorpolynom gewisse Bedingungen erfüllen muss. Nicht alle Polynome sind als Generatorpolynom geeignet. Deswegen haben wir bisher unser Generatorpolynom aus einer Tabelle geholt. Welche Eigenschaften muss also das Generatorpolynom im Detail erfüllen? Nun, eine wichtige Bedingung, damit zyklische Codes 'funktionieren', ist die Verwendung von *irreduziblen Polynomen*. Nur mit irreduziblen Polynomen (i.P.) kann man Fehler hinterher eindeutig zuordnen und somit korrigieren. I.P. haben die Eigenschaft, nicht in weitere Faktor-Polynome zerlegbar zu sein; im Prinzip funktionieren sie also wie Primzahlen.

Wie genau man irreduzible Polynome errechnet, ist hier nicht von Belang. Man kann sich folgende irreduzible Polynome für die Arbeit mit zyklischen Codes merken. Sortiert sind sie nach dem Grad des Generatorpolynoms:

$m=1$	$x^1 + 1$
$m=2$	$x^2 + x^1 + 1$
$m=3$	$x^3 + x^1 + 1$ und $x^3 + x^2 + 1$
$m=4$	$x^4 + x^1 + 1$ und $x^4 + x^3 + x^2 + x^1 + 1$ und $x^4 + x^3 + 1$

Diese Polynome lassen sich also nicht mehr in 'kleinere' Polynome aufteilen.

Uns interessiert vor Allem, wie wir mehrere Fehler korrigieren können. Hier kommen wieder die irreduziblen Polynome ins Spiel, denn **zur Mehrfach-Fehlerkorrektur muss das Generatorpolynom das Produkt aus mehreren irreduziblen Polynomen sein**. Das Thema Mehrfach-Fehlerkorrektur soll aber nicht mehr Thema dieses Seminars sein.