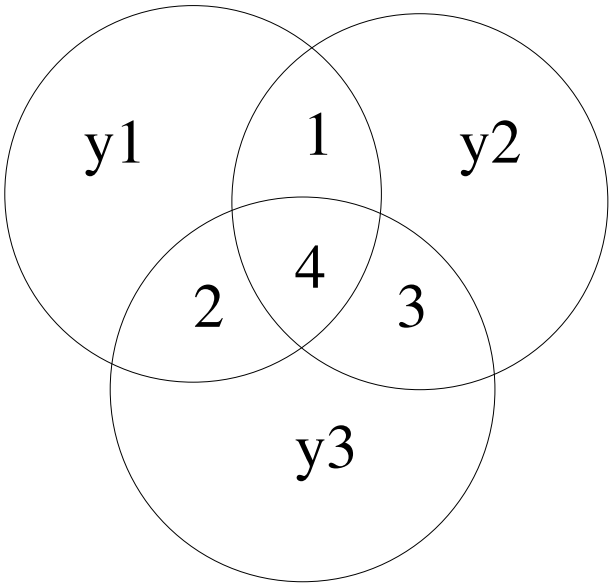

Abbildung 4.1: *Selbstkorrigierende Codes*

4. Die Mathematik hinter der Compact Disc

4.1. Selbstkorrigierende Codes

Wenn wir eine Reihe von 0 und 1 übermitteln, so passieren Fehler. Wir suchen nun eine Möglichkeit, die Fehler zu erkennen und zu korrigieren. Nehmen wir an, wir wollen 4 Bits übermitteln, und haben dazu 8 Bits zur Verfügung. Die Wahrscheinlichkeit einer Fehlübermittlung soll so klein sein, dass es eher unwahrscheinlich ist, dass, bei einem Signal von 8 Bit, 2 oder mehr Bits falsch übermittelt werden.

Nennen wir die Nachricht $x_1x_2x_3x_4$. Eine einfache Idee ist es, die 4 Bits doppelt zu übermitteln, also $x_1x_2x_3x_4x_1x_2x_3x_4$. Stimmen zwei Bits nicht überein, wissen wir, dass ein Fehler vorliegt. Aber, dies nützt uns nichts. Erhalten wir z.B. $1x_2x_3x_40x_2x_3x_4$, so wissen wir nicht, ob nun die Null oder die Eins richtig ist.

Wir können dies auch besser machen. Wir setzen

$$y_1 = x_1+x_2+x_4 \mod 2\,, \quad y_2 = x_1+x_3+x_4 \mod 2\,, \quad y_3 = x_2+x_3+x_4 \mod 2\,.$$

Dann wird die Nachricht $x_1x_2x_3x_4y_1y_2y_3$ übermittelt. Dann wissen wir, die Summen

$$x_1 + x_2 + x_4 + y_1\,, \quad x_1 + x_3 + x_4 + y_2\,, \quad x_2 + x_3 + x_4 + y_3$$

sind alle gerade. Sind alle drei Summen gerade, so können wir davon ausgehen, dass kein Fehler passiert ist. Nehmen wir an, dass genau ein Fehler passiert ist.

Betrachten wir den x_1, x_2, x_3 kommen genau zweimal vor. Ist also eines dieser drei falsch übermittelt worden, dass sind genau zwei der Summen ungerade. Da das falsche Bit in der geraden Summe nicht vorkommt, haben wir es identifiziert. x_4 kommt in allen drei Summen vor. Ist also x_4 falsch, so haben wir drei ungerade Summen. y_1, y_2, y_3 kommen in genau einer Summe vor. Ist der Fehler also dort, so ist nur eine Summe ungerade, und wir finden so das falsche Bit. Wir können also einen Fehler eindeutig zuordnen, und auf diese Weise den Fehler korrigieren. Wir haben sogar noch ein leeres Bit, das wir nicht verwendet haben.

4.2. Reed–Solomon-Codes

Bei einer CD wird das Signal 44 100 Mal pro Sekunde gemessen. Dies genügt, da der Mensch nur bis zu einer Frequenz von 19 000 Hertz hören kann. Jeder Ton wird in einem 16 Bit Code dargestellt. Ein Ton wird also so diskretisiert, dass $2^{16} = 65536$ verschiedene "Lautstärken" hörbar sind. Um Musik in Stereo zu hören, sind also 32 Bits notwendig, also 4 Bytes. Um zu erklären, wie die Codierung funktioniert, machen wir folgendes Beispiel. Nehmen wir an, wir hätten ein Alphabet mit 31 Buchstaben, und wollen ein Wort $a_2a_3a_4a_5$ mit 4 Buchstaben codieren. Wir lösen nun das Gleichungssystem

$$\begin{aligned} a_0 + a_1 + a_2 + a_3 + a_4 + a_5 &= 0 \mod 31 , \\ a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 &= 0 \mod 31 . \end{aligned}$$

Nehmen wir nun an, bei der Übermittlung von a_3 sei ein Fehler aufgetreten, und $a_3' = a_3 + 2$ sei übermittelt worden. Dann erhalten wir die erste Gleichung

$$a_0 + a_1 + a_2 + a_3' + a_4 + a_5 = 2 \mod 31 .$$

Somit wissen wir, sofern nur ein Fehler aufgetreten ist, dass ein Buchstabe um 2 verschoben ist. Betrachten wir die zweite Gleichung, so erhalten wir

$$a_1 + 2a_2 + 3a_3' + 4a_4 + 5a_5 = 6 \mod 31 .$$

Da $6/2 = 3$, wissen wir, dass der Fehler an der dritten Stelle aufgetreten ist, so dass wir die dritte Stelle korrigieren können.

Betrachten wir nun eine Gruppe von 8 Buchstaben $a_4a_5a_6 \cdots a_{11}$. Wir lösen dann das System von 4 Gleichungen

$$a_0 0^k + a_1 1^k + a_2 2^k + a_3 3^k + \cdots + a_{11} 11^k = 0 \mod 31 , \qquad (k = 0, 1, 2, 3) .$$

Tritt ein Fehler auf, korrigieren wir den Fehler wie im ersten Beispiel. Treten zwei Fehler auf, so müssen wir eine quadratische Gleichung lösen, um herauszufinden, wo sich die beiden Fehler verstecken, und wie gross sie sind. Die Unbekannten sind die Grössen der beiden Fehler und die beiden Stellen, wo die Fehler aufgetreten sind. Wir haben also so viele Gleichungen wie Unbekannte.

4.3. Lineare rekursive Gleichungen

Wir wollen nun Gleichungen der Form

$$f_{n+1} = a_0 f_n + a_1 f_{n-1} + \cdots + a_k f_{n-k}$$

betrachten. Solche Gleichungen treten bei vielen Problemen auf. Insbesondere interessieren uns die Fälle $k = 1, 2$. Starten wir aber mit $k = 0$. Dann lautet die Gleichung $f_{n+1} = af_n$. Setzen wir die Gleichung für f_n ein, erhalten wir $f_{n+1} = a(af_{n-1}) = a^2 f_{n-1}$. Es zeigt sich, dass $f_n = a^n f_0$, und wir haben eine explizite Lösung erhalten.

Schwieriger scheint es, wenn $k = 1$, also $f_{n+1} = af_n + bf_{n-1}$. Motiviert durch die Lösung im Falle $k = 0$ versuchen wir eine Lösung der Form $f_n = \lambda^n$. Setzen wir dies in die Gleichung ein, erhalten wir

$$\lambda^{n+1} = a\lambda^n + b\lambda^{n-1} \, .$$

Ist $\lambda \neq 0$, so ist dies $\lambda^2 - a\lambda - b = 0$. Wir finden die beiden Lösungen

$$\lambda_{1/2} = \frac{a \pm \sqrt{a^2 + 4b}}{2} \, .$$

Nehmen wir an, dass $a_2 + 4b > 0$. Man rechnet sofort nach, dass

$$f_n = \alpha\lambda_1^n + \beta\lambda_2^n$$

eine Lösung ist. Die beiden Gleichungen

$$\begin{aligned} f_0 &= \alpha + \beta \, , \\ f_1 &= \alpha\lambda_1 + \beta\lambda_2 \, , \end{aligned}$$

können wir nach α, β auflösen: $\alpha = \frac{f_1 - f_0\lambda_2}{\lambda_1 - \lambda_2}$, $\beta = \frac{f_0\lambda_1 - f_1}{\lambda_1 - \lambda_2}$.

Im Fall $a^2 + 4b = 0$ erhalten wir nur die Lösung $\lambda = a/2$. Wir können $a \neq 0$ annehmen, da sonst die Gleichung $f_{n+1} = 0$ lautet. Setzen wir $g_n = f_n(\frac{2}{a})^n$, also $f_n = g_n(\frac{a}{2})^n$. Setzen wir dies in die Gleichung ein, erhalten wir

$$g_{n+1}\Big(\frac{a}{2}\Big)^{n+1} = ag_n\Big(\frac{a}{2}\Big)^n - \frac{a^2}{4}g_{n-1}\Big(\frac{a}{2}\Big)^{n-1} = (2g_n - g_{n-1})\Big(\frac{a}{2}\Big)^{n+1} \, .$$

Die Gleichung für g ist dann $g_{n+1} = 2g_n - g_{n-1}$. Dies kann anders als $g_{n+1} - g_n = g_n - g_{n-1}$ schreiben. Somit sind die Differenzen konstant. Man erkennt daher, dass $g_n = n$ eine Lösung ist. Zusammenfassend erhalten wir die Lösung

$$f_n = (\alpha n + \beta)\Big(\frac{a}{2}\Big)^n \, .$$

Aus den Anfangsbedingungen erhalten wir $\beta = f_0$ und $\alpha = \frac{2}{a} f_1 - f_0$.

Der Fall $a_2 + 4b < 0$ lässt sich wie der Fall $a_2 + 4b > 0$ lösen, wobei λ_1, λ_2 dann komplexe Zahlen sind.

Beispiel 4.1. Die Fibonacci-Folge ist durch die Rekursion $f_{n+1} = f_n + f_{n-1}$ mit $f_0 = f_1 = 1$ gegeben. Wir erhalten

$$\lambda_{1/2} = \frac{1 \pm \sqrt{5}}{2} \, .$$

Für die Parameter α, β erhalten wir

$$\alpha = \frac{1 - \frac{1-\sqrt{5}}{2}}{\sqrt{5}} = \lambda_1/\sqrt{5} \, , \quad \beta = 1 - \alpha = -\lambda_2/\sqrt{5} \, .$$

Wir erhalten somit

$$f_n = \frac{\lambda_1^{n+1} - \lambda_2^{n+1}}{\sqrt{5}} \, .$$

Da $\lambda_2 \approx -0.618$, können wir die Fibonacci-Zahl f_n berechnen, indem wir $\lambda_1^{n+1}/\sqrt{5}$ berechnen und dann auf die nächste ganze Zahl runden. ■

Ist nun $k = 2$, so lautet die Gleichung $f_{n+1} = af_n + bf_{n-1} + cf_{n-2}$. Der Ansatz $f_n = \lambda^n$ führt auf die Gleichung

$$\lambda^3 - a\lambda^2 - b\lambda - c = 0 \, .$$

Diese Gleichung hat Nullstellen λ_1, λ_2, λ_3, wobei möglicherweise zwei oder mehrere Nullstellen zusammenfallen. Sind alle Nullstellen verschieden, so haben wir die Lösung $f_n = \alpha\lambda_1^n + \beta\lambda_2^n + \gamma\lambda_3^n$. Ist $\lambda_1 \neq \lambda_2 = \lambda_3$, so lautet die Lösung $f_n = \alpha\lambda_1^n + (\beta n + \gamma)\lambda_2^n$. Sind sogar alle drei Nullstellen identisch, so lautet die Lösung $f_n = (\alpha n^2 + \beta n + \gamma)\lambda_1^n$.

4.4. Die Codierung der CD

Auf einer CD wird der Code zuerst in Gruppen von 24 Byte geteilt. In einem 1. Schritt, werden ähnlich zu den Beispielen im letzten Abschnitt 8 zusätzliche Bytes

dazugefügt. 24 von diesen Codes werden in einer 24×32 Matrix zusammengefügt. Nun wird daraus eine 28×32 Matrix gemacht, indem jeder Spalte mit der gleichen Methode wie im ersten Schritt 4 zusätzliche Bytes dazugefügt werden. Durch das Zusammenspiel der beiden Codierungen und der Matrix, können sehr viele Fehler korrigiert werden. Die Daten werden in Paketen von jeweils 32 Bit geschrieben, und nach jeden 32 Bits wird ein Kontrollbit eingefügt.

Staub auf der CD oder ein Kratzer kann einen Teil der Fläche unlesbar machen. Würde man also, analog der Schallplatte, die Musik in der Reihenfolge auf die CD brennen, in der die Töne auftreten, so würde bei einem Kratzer grössere Stücke fehlen. Daher werden benachbarte Pakete nicht nebeneinander auf die CD gebrannt.

Eine CD hat ein weiteres technisches Problem. Die Daten werden auf einer etwa 5 km langen Bahn mit Tälern und Bergen auf die Scheibe geschrieben. Ein Laser tastet die Täler und Berge ab. Sind die Täler oder Berge aber zu kurz, so entstehen durch Interferenz Fehler beim Lesen. Daher muss ein Tal oder ein Berg mindestens 3 Bit lang sein. Man interpretiert dann den Wechsel von Berg zu Tal oder von Tal zu Berg als 1, und keinen Wechsel als 0. Das bedeutet, dass eine 1 von mindestens zwei Nullen gefolgt sein muss. Bei einer 1 synchronisiert sich der Laser, da er dann einen Anhaltspunkt erhält, wo er ist. Sind Täler oder Berge zu lang, gerät die Uhr des Lasers aus dem Takt. Aus diesem Grunde dürfen nicht mehr als 10 Nullen nacheinander auftreten.

Wollen wir $2^8 = 256$ verschiedene Zahlen so darstellen, dass eine Eins von mindestens zwei aber höchstens 10 Nullen gefolgt wird, so müssen wir wissen, wieviele der 2^n Zahlen in einer n-Bit-Folge die Regeln erfüllen. Schauen wir uns ein einfacheres Problem an. Nehmen wir an, wir suchen, in wievielen der 2^n Zahlen mit n-Bits keine zwei Einsen nacheinander auftreten. Nennen wir die gesuchte Zahl F_n. Dann ist klar, dass $F_1 = 2$ und $F_2 = 3$ (nur 11 ist nicht erlaubt). In einer Zahl mit n Bits schauen wir uns das letzte Bit an. Ist dieses Bit 0, so sind in den ersten $n-1$ Bits alle Zahlen, die die Regeln erfüllen erlaubt. Somit haben wir F_{n-1} Zahlen. Ist das letzte Bit eine 1, so ist an der zweitletzten Stelle nur eine 0 erlaubt. Davor können wir wieder alle erlaubten Zahlen einsetzen, also F_{n-2} Zahlen. Dies ergibt die Gleichung $F_n = F_{n-1} + F_{n-2}$. Dies ist die Fibonacci-Folge.

Im Falle der CD erhalten wir, wenn wir mal die maximale Länge der 0-Folgen ignorieren, die Rekursion $F_n = F_{n-1} + F_{n-3}$, wobei $F_1 = 2$, $F_2 = 3$, $F_3 = 3 + 1 = 4$. Die Nullstellen sind $\lambda_1 \approx 1.46557$, $\lambda_2 \approx -0.232786 + 0.792552\,\mathrm{i}$, $\lambda_2 \approx -0.232786 - 0.792552\,\mathrm{i}$. Die zweiten beide Nullstellen haben einen Betrag kleiner als 1. Also ist die Formel ungefähr $\alpha\lambda_1^n$. Lösen wir das Gleichungssystem für die Anfangsbedingungen,

erhalten wir $\alpha = 1.3134$. Wir suchen $\alpha\lambda_1^n \geq 256$, also $n \geq [\ln(256/\alpha)]/[\ln \lambda_1] \approx 13.7936$. Wir haben also $n = 14$, was $F_{14} = 277$ ($\alpha\lambda_1^{14} = 277.012$) entspricht. Für 256 verschieden Zahlen brauchen wir also mindestens 14 Bits. Von den 277 Zahlen besteht eine aus lauter Nullen, zwei aus einer Folge von 13 Nullen und einer 1, drei aus einer Folge von genau 12 Nullen und vier aus einer Folge von genau 11 Nullen. Somit erfüllen 267 Zahlen unsere Regeln, und wir benötigen also 14 Bits, um alle 256 Zahlen darstellen zu können.

Setzen wir nun zwei Sequenzen aus $14+14 = 28$ Bits zusammen, so kann z.B. die erste Sequenz mit einer 1 enden, und die zweite mit einer 1 beginnen, oder die erste mit 9 Nullen enden, und die zweite mit 4 Nullen beginnen. Um dies zu verhindern, muss man gefährliche Sequenzen entfernen. Aus diesem Grunde werden nicht 14 sondern 17 Bits verwendet, um die 256 Zahlen darzustellen. Zusätzlich werden nach 33×17 Bits noch 27 Synchronisationsbits dazugefügt.

Bei einer Sekunde Musik in Stereo ist die Musik in $44\,100 \times 32 = 1\,411\,200$ Bits kodiert. Auf der CD werden dies

$$1\,411\,200 \times \frac{28 \times 32}{24 \times 24} \times \frac{33 \times 17 + 27}{32 \times 8} = 5\,042\,100$$

Bits. Liest der Player ein Bit mit der kleinen Wahrscheinlichkeit von 10^{-4} falsch, gibt dies im Durchschnitt immer noch über 500 Fehler pro Sekunde, die korrigiert werden müssen. Daher ist ein effizientes Verfahren zum Korrigieren der Fehler wichtig.