Codierungstheorie

B. Waldmüller

25. April 2009

Inhaltsverzeichnis

1 Einführung

1.1 Vorbemerkungen

Dieses Material habe ich zum Gebrauch beim „Mathematischen Samstag am Söderblom–Gymnasium“ am 25. April 2009 erstellt. Natürlich überdeckt es nicht das Gebiet der Codierungstheorie, so sollte der Titel nicht verstanden werden; ich bin zufrieden, wenn es gelingt, jedem Teilnehmer einen Einblick in dieses Gebiet zu verschaffen. Dabei sind die Vorkenntnisse der Teilnehmer sehr unterschiedlich; es kommen Schüler ab Stufe 9, aber auch Studenten bis hin zum Studenten der Mathematik im Hauptstudium.

Über weite Strecken sollen die Teilnehmer an den Beispielen und Übungen im Text arbeiten. Hier gilt mein besonderer Dank zwei langjährigen treuen Mitgliedern der Mathe–AG: Leon Merten Lohse, der dafür gesorgt hat, dass den Teilnehmern das Programm GAP zur Verfügung steht, und Daniel Ollesch, der sich vorab mit den Übungen befasst hat. Daniel wird an dem Samstag Hilfestellung leisten.

Mit einem * gekennzeichnete Abschnitte setzen höhere Vorkenntnisse voraus, sie können ohne Weiteres weggelassen werden. Manches davon habe ich auch zu meiner eigenen Vergewisserung aufgeschrieben.

1.2 Problemstellung

Wer eine Überweisung ausfüllt oder etwas bestellt, möchte, dass das Geld beim richtigen Empfänger ankommt und dass die richtige Ware geliefert wird. Dazu muss eine Nummer richtig geschrieben werden, die geschriebene Nummer unverfälscht übertragen und anschließend wieder richtig gelesen werden. Treten dabei Fehler auf, sollten diese erkannt und am besten sogar korrigiert werden.

Damit ist der Arbeitsauftrag an die Codierungstheorie formuliert. Die Lösung sieht, grob gesprochen, so aus: Wenn die Bestellnummer zweistellig ist, kann man sie als Zahlenpaar (x, y) ansehen und sich einen Punkt der Ebene darunter vorstellen. Nun fügt man – und das ist der entscheidende Trick – jedem (x, y) eine dritte Ziffer z hinzu, die eigentlich für die Bestellung selbst nicht nötig wäre. Die erhaltenen Tripel oder 3–tupel stellen wir uns natürlich als Raumpunkte vor. Die z–Werte zu den (x, y) wählt man so, dass die Raumpunkte (x, y, z) wieder in einer Ebene liegen, die aber keine zu einer der Koordinatenachsen parallele Gerade enthalten soll: die Ebene soll schräg im Raum liegen. Wenn man nun eine der drei Koordinaten x, y, z verändert, ist man sofort aus der Ebene heraus, und das kann man feststellen. Ändert man freilich zwei der drei Koordinatenwerte, ist man unter Umständen wieder in der Ebene, und dieser Fehler wird dann nicht ohne Weiteres entdeckt.

Bekommt der Empfänger einen Punkt, der nicht in der schrägen Ebene liegt, kann er vielleicht den ursprünglichen Punkt finden, etwa indem er den Ebenenpunkt nimmt, den er mit einer möglichst kleinen Änderung **einer** Koordinate des Punktes erreicht.

Jetzt hast du schon eine Vorstellung, in welcher Richtung die Sache läuft. Bevor wir genauer hinschauen, ist sicher ein Beispiel nützlich.

1.3 Der EAN13–Code

Sehr viele Produkte im Handel tragen eine 13–stellige Nummer aus den Ziffern von 0 bis 9 - als Klartext und als Strichcode. Die ersten zwölf Stellen tragen die Information, an der Stelle 13 steht eine Prüfziffer. Sie wird so gebildet, dass die Summe der Ziffern auf den Stellen ungerader Nummer und das Dreifache der Summe der Ziffern auf den Stellen mit gerader Nummer zusammen auf 0 endet:

$$c_1 + c_3 + \cdots + c_{13} + 3(c_2 + c_4 + \cdots + c_{12}) \equiv 0 \mod 10$$

Ein paar kleine Aufträge:

1. Überprüfe die angegebene Formel an einigen Beispielen.

Abbildung 1: Eine EAN13–Codenummer

2. Begründe: Wird genau eine Ziffer falsch übermittelt, kann das mit Sicherheit festgestellt werden.

3. Verändere zwei Ziffern einer gültigen Codenummer so, dass wieder eine gültige Codenummer entsteht.

4. Warum multipliziert man einen Teil der Ziffern mit 3? Klar, es sollen Zahlendreher entdeckt werden. Kann das System mit Sicherheit entdecken, dass zwei benachbarte Ziffern vertauscht wurden?

5. Hätte man statt des Faktors 3 auch einen anderen Faktor nehmen können, etwa 2, 4, 5 oder 7?

2 Spezialisierung: Lineare Block–Codes

2.1 Begriffsbildung

Nun wollen wir Codes konstruieren und mit einer Struktur versehen. Wir nehmen gleich einige Einschränkungen vor, und wir legen einige Bezeichnungen fest:

- Als Buchstaben, in denen unsere Codeworte geschrieben werden, nehmen wir grundsätzlich nur „Zahlen“, das heißt Objekte, mit denen wir vernünftig rechnen können. So können wir durch eine Rechnung feststellen, ob ein Wort ein Codewort ist, und müssen nicht umständlich in einer Liste nachschauen.

- Das Alphabet bezeichnen wir meist mit F und die Anzahl seiner Buchstaben mit q: $|F| = q$.

- Alle Codeworte sollen die gleiche Länge haben, also die gleiche Anzahl von Stellen. Einen solchen Code nennt man Blockcode.

- Die Länge der Codeworte bezeichnen wir stets mit n.

- Summen und Vielfache von Codeworten sollen wieder Codeworte sein. Dabei addiert man zwei Codeworte komponentenweise und bildet das Vielfache eines Codewortes mit einer Zahl, indem man jede Komponente mit der Zahl multipliziert. Ein solcher Code wird linearer Code genannt.

Der Code C ist also eine Teilmenge der Menge F^n aller Worte der Länge n mit Buchstaben aus F. Insgesamt gilt $|F^n| = q^n$. Denen, die sich mit Linearer Algebra auskennen, ist klar, dass $|C| = q^k$ sein muss für ein $k \leq n$. Der Code C heißt dann ein $[n, k]$–Code. Später wird noch ein dritter Parameter hinzutreten.

2.2 Beispiel: Wiederholungscodes

Als Menge von Nachrichten nehmen wir die Menge aller Worte der Länge 3 über dem Alphabet $F := \{0; 1\}$. Man spricht auch von der Menge aller Tripel mit Einträgen aus F und hat dafür das Symbol F^3:

$$F^3 = \{(a, b, c) \,|\, a, b, c \in F\}$$

Es gibt dann acht Nachrichten.

Wir bilden nun Codes C_1 und C_2 zu den Nachrichten, indem wir einfach die Worte wiederholen:

$$C_1 := \{(a, b, c, a, b, c) \,|\, a, b, c \in F\} \tag{1}$$

$$C_2 := \{(a, b, c, a, b, c, a, b, c) \,|\, a, b, c \in F\} \tag{2}$$

Dann ist C_1 ein $[6, 3]$–Code und C_2 ein $[9, 3]$–Code.

Tritt genau ein Übertragungsfehler auf, wird er sowohl bei C_1 als auch bei C_2 mit Sicherheit entdeckt. Verwendet man C_2, kann man sogar das ursprünglich gesendete Codewort wiederherstellen, wenn nur ein Fehler aufgetreten ist. Man bezahlt dafür dadurch, dass die zu versendenden Nachrichten viel länger werden als sie ursprünglich waren.

3 Geometrisierung

3.1 Der Raum F^n

Wie gesagt: Ein linearer Code C ist eine Menge von n–tupeln mit Einträgen aus einem „Körper“ F mit der Eigenschaft, dass Summen solcher n–tupel und Vielfache von n–tupeln mit Elementen von F wieder in C liegen. Damit trägt C in natürlicher Weise eine geometrische Struktur! So, wie du ein Paar $(x; y)$ reeller Zahlen als Koordinatenpaar eines Punktes P der Ebene ansehen kannst, kannst du auch ein $c \in C$ als Satz von Koordinaten ansehen. Bildest du für eine reelle Zahl r das „Vielfache“

$$r(x, y) := (rx, ry) \quad ,$$

ergibt dies die Koordinaten eines Punktes auf der Geraden durch P und durch den Ursprung des Koordinatensystems – einmal $(x, y) \neq \vec{0}$ vorausgesetzt[1]. Durchläuft r alle reellen Zahlen, durchläuft der Punkt die ganze Gerade. Wir übertragen das auf unser C und sehen für $c \in C$, $c \neq \vec{0}$, die Menge

$$\{ac \,|\, a \in F\}$$

als Gerade durch den Nullpunkt an. Über den Körper F müssen wir dabei im Augenblick nur wissen, dass er eine 0 und eine 1 mit den üblichen Eigenschaften enthält und dass es in ihm ein Element t so gibt, dass

$$F = \{0, 1, t, t^2, \ldots, t^{q-2}\}$$

ist, also alle Elemente $\neq 0$ in F Potenzen dieses t sind. Dabei ist noch $t^{q-1} = 1$.

Es ist sicher ratsam, dass wir ein konkretes Beispiel betrachten. Als Körper nehmen wir eine Menge E mit $q = 4$ Elementen, die wir also

$$E = \{0, 1, t, t^2\}$$

schreiben können, und wir nehmen $C = E^2$. Zeichne dir ein ebenes Koordinatensystem und trage die Elemente von E an den Achsen an[2], dann siehst du schon die 16 Elemente von C. Markiere nun die Geraden durch den Nullpunkt, du wirst genau fünf davon finden!

Aufgabe Betrachte den „Raum“ E^3 aller 3–tupel mit Einträgen aus E. Ein 3–tupel können wir als Koordinatensatz eines Raumpunktes ansehen. Wir bilden also ein räumliches Koordinatensystem und veranschaulichen uns damit die Elemente von E^3.[3]

[1] Mit $\vec{0}$ bezeichne ich jeweils das Tupel, dessen sämtliche Einträge 0 sind.
[2] siehe Abbildung 2
[3] siehe Abbildung 3

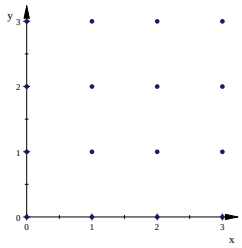

Abbildung 2: E^2 geometrisch

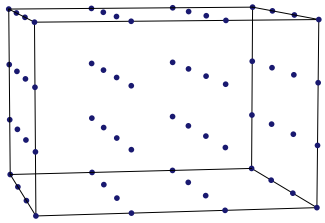

Abbildung 3: E^3 geometrisch

1. Wieviele Elemente hat E^3?
2. Markiere die Gerade durch den Nullpunkt und $A(1,0,0)$.[4]
3. Markiere die Gerade durch den Nullpunkt und $B(1,t,t^2)$.
4. * Finde die Punkte auf der geraden durch A und B.
5. Wieviele Geraden durch den Nullpunkt gibt es in E^3?

3.2 Der Hamming–Abstand

Nun wollen wir in unserem „Raum“ F^n Abstände von Punkten definieren. Sinnvoll in unserem Kontext ist ein Abstandsbegriff, der von Richard Hamming vorgeschlagen wurde. Der Hamming–Abstand $\mathrm{d_H}(u,v)$ zweier n–tupel $u,v \in F^n$ ist die Anzahl der Stellen, an denen sich u und v unterscheiden:

$$\mathrm{d_H}(u,v) := |\{i \,|\, u_i \neq v_i\} \tag{3}$$

So ist zum Beispiel $\mathrm{d_H}((1,1,1),(0,1,0)) = 2$.

Der Hamming–Abstand hat die vertrauten Eigenschaften: es ist

$$\mathrm{d_H}(u,v) = 0 \text{ nur für } u = v,\ \mathrm{d_H}(u,v) = \mathrm{d_H}(v,u) \text{ für alle } u,v \in F^n,$$

und es gilt sogar die Dreiecksungleichung:

$$\mathrm{d_H}(u,v) \leq \mathrm{d_H}(u,w) + \mathrm{d_H}(w,v) \text{ für alle } u,v,w \in F^n$$

Ferner gilt

$$\mathrm{d_H}(u,v) = \mathrm{d_H}(u+w, v+w) \text{ für alle } u,v,w \in F^n\ ,$$

die Abbildung $u \mapsto u + w$ für ein festes $w \in F^n$ ist also so etwas wie eine Verschiebung im Raum F^n.

[4] Ich habe noch Extrazettel mit dieser Abbildung vorbereitet, die kannst du zum Zeichnen verwenden.

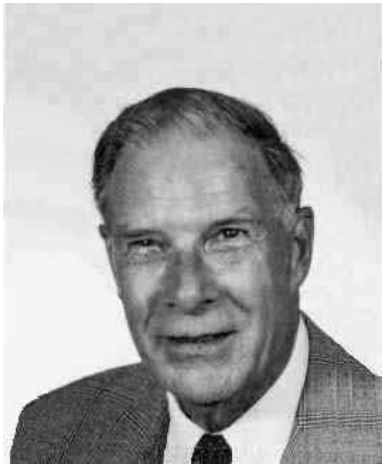

Abbildung 4: Richard Hamming, 1915–1998

3.3 Der minimale Abstand zweier Codeworte

Damit unser Code $C \subseteq F^n$ möglichst unempfindlich gegen Übermittlungsfehler ist, muss der Hamming–Abstand verschiedener Codeworte möglichst groß sein: Ein Code, der zwei Worte enthält, die sich nur an einer Stelle unterscheiden, ist offensichtlich unbrauchbar! Deshalb interessiert man sich für den minimalen Hamming–Abstand zweier Codeworte aus C

$$\mathrm{d}(C) := \min\{\mathrm{d_H}(u,v) \,|\, u,v \in C, u \neq v\} \quad . \tag{4}$$

Ist $\mathrm{d}(C) = d$, nennt man C auch einen $[n,k,d]$–Code.

Wie berechnet man nun $\mathrm{d}(C)$? Es gibt

$$\binom{|C|}{2} = \frac{|C| \cdot (|C|-1)}{2} = \frac{q^k(q^k-1)}{2}$$

Paare verschiedener Elemente von C, so viele Hamming–Abstände muss man also berechnen und den kleinsten Wert heraussuchen. Da wir uns nur mit linearen Codes beschäftigen, kommen wir mit etwas weniger Arbeit aus. Für $u, v \in C$ ist ja

$$\mathrm{d_H}(u,v) = \mathrm{d_H}(u-v, v-v) = \mathrm{d_H}(u-v, \vec{0}) \quad ,$$

unser $\mathrm{d}(C)$ ist die minimale Anzahl der Einträge $\neq 0$, die bei den von $\vec{0}$ verschiedenen Codeworten auftritt.

Aufgabe Bestimme $\mathrm{d}(C_1)$ und $\mathrm{d}(C_2)$ für die Wiederholungscodes auf Seite 4.

3.4 Die Singleton–Schranke

Es sei C ein $[n,k,d]$–Code. Je näher k an n liegt, desto „dichter“ liegen die Elemente von C in F^n, und man wird erwarten, dass d dann nicht mehr groß sein kann. Wir präzisieren diese Erwartung nun.

1 Lemma
Es sei C ein $[n,k,d]$–Code über einem Körper F mit $|F| = q$. Dann ist

$$k \leq n - d + 1 \quad .$$

Man kann die Aussage des Lemma auch so aussprechen, dass unter der Voraussetzung des Lemmas

$$d \leq n - k + 1 \tag{5}$$

sein muss. Der Term auf der rechten Seite heißt **Singleton–Schranke** für d. Wir werden Codes kennenlernen, die in dem Sinn optimal sind, dass $d = n - k + 1$ ist.

3.5 Kugeln in F^n

Eine Kugel[5] im Raum mit Mittelpunkt M und Radius r besteht aus allen Punkten P des Raumes, deren Abstand von M höchstens r ist:

$$B_r(M) = \{P \mid \mathrm{d_e}(P, M) \leq r\}$$

mit dem gewöhnlichen euklidischen Abstand $\mathrm{d_e}$. In unsrem F^n bilden wir nun Kugeln mit dem Hamming–Abstand!

2 Definition
Es sei $v \in F^n$ und $r \geq 0$. Unter der Kugel $B_r(v)$ um v mit Radius r verstehen wir die Menge

$$B_r(v) := \{w \in F^n \mid \mathrm{d_H}(v, w) \leq r\} \quad .$$

Am besten machst du dich anhand der Aufgaben selbst mit dem neuen Begriff vertraut, bevor du dir die Ergebnisse anschaust.

Aufgabe

1. Schaue dir noch einmal die Darstellungen der Räume E^2 und E^3 auf Seite 5 an. Wie sieht für ein $v \in E^2$ die Kugel $B_1(v)$ aus? Wie sehen für ein $v \in E^3$ die Kugeln $B_1(v)$ und $B_2(v)$ aus?
2. Wieviele Elemente hat $B_1(v)$ für $v \in E^2$, für $v \in E^3$, für $v \in F$ mit $|F| = q$?
3. Wieviele Elemente hat $B_2(v)$ für $v \in E^3$, für $v \in F^n$ mit $|F| = q$?
4. Es sei $v \in F^n$ und es sei j eine positive ganze Zahl. Wieviele $w \in F^n$ gibt es mit $\mathrm{d_H}(v, w) = j$? Und welche geometrische Bedeutung hat die Menge dieser w?
5. Wieviele Elemente hat $B_r(v)$ für $v \in F^n$ für eine ganze Zahl $r \geq 0$?

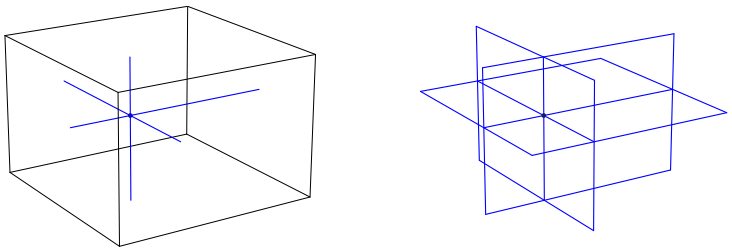

Abbildung 5: $B_1(v)$ (links) und $B_2(v)$ (rechts) für ein $v \in E_3$

3 Lemma
Es sei $v \in F^n$, und es sei j eine positive ganze Zahl. Dann ist

$$|\{w \in F^n \mid \mathrm{d_H}(v, w) = j\}| = \binom{n}{j} (q-1)^j \quad .$$

[5] Wir brauchen hier Vollkugeln, also Kugeln einschließlich Rand.

4 Lemma
Für $v \in F^n$ und $r \in \mathbb{Z}$, $r > 0$, ist

$$|B_r(v)| = 1 + \sum_{j=1}^{r} \binom{n}{j} \cdot (q-1)^j \quad .$$

3.6 Fehlererkennende und fehlerkorrigerende Codes

Es sei $C \subseteq F^n$ ein $[n, k, d]$–Code, und es sei $c \in C$ ein Codewort. Bei der Übermittlung dieses Codewortes können Fehler auftreten, empfangen wird dann ein Wort $c' \in F^n$. Die Anzahl der Fehler ist nichts Anderes als $\mathrm{d_H}(c, c')$!

Der minimale Hamming–Abstand in C ist d, das heißt, dass c das einzige Codewort in $B_{d-1}(c)$ ist. Treten bei der Übermittlung höchstens $d-1$ Fehler auf, ist das empfangene Wort c' das ursprünglich versandte Wort c, und dann ist alles in Ordnung, oder c' ist kein Codewort, und das wird entdeckt. Man sagt, unser Code C sei $(d-1)$–fehlererkennend.

5 Definition
Ein Code, bei dem bei der Übermittlung eines jeden Codewortes c bis zu t Fehler sicher erkannt werden, heißt t–fehlererkennend.

6 Lemma
Ein $[n, k, d]$–Code ist $(d-1)$–fehlererkennend.

Wenn nun erkannt wurde, dass das empfangene Wort c' fehlerhaft ist, möchte man gern das ursprünglich versandte Codewort c zurückgewinnen. Gehen wir davon aus, dass maximal t Fehler aufgetreten sind, muss c in $B_t(c')$ liegen. Nehmen wir an, es gebe zwei Codeworte c_1 und c_2 in $B_t(c')$. Dann ist nach der Dreiecksungleichung

$$\mathrm{d_H}(c_1, c_2) \leq \mathrm{d_H}(c_1, c') + \mathrm{d_H}(c', c_2) \leq t + t = 2t \quad .$$

Falls $t < \frac{1}{2}d$ ist, folgte aus der obigen Ungleichung $\mathrm{d_H}(c_1, c_2) < d$, und das kann nicht sein, denn d ist der minimale Hamming–Abstand, der zwischen zwei verschiedenen Codeworten bestehen kann. Treten weniger als $\frac{1}{2}d$ Fehler auf, können diese also bei unserem C korrigiert werden!

7 Definition
Es sei $C \subseteq F^n$ ein Code, und es sei t eine positive ganze Zahl. Für jedes $v \in F^n$ enthalte $B_t(v)$ höchstens ein $c \in C$. Dann heißt C ein t–fehlerkorrigierender Code.

8 Lemma
Es sei C ein $[n, k, d]$–Code, und es sei $d = 2m+1 \geq 3$ ungerade. Dann ist C ein m–fehlerkorrigierender Code.

Beispiel Unser Wiederholungscode C_1 auf Seite 5 ist 1–fehlererkennend, C_2 ist 2–fehlererkennend und 1–fehlerkorrigierend.

3.7 Decodierung

Es sei C ein $[n, k, d]$–Code. Wir gehen davon aus, dass ein Codewort $c \in C$ gesendet und ein Wort $c' \in K^n$ empfangen wurde. Man wünscht sich natürlich, dass $c' = c$ ist, denn dann ist alles in Ordnung, aber Codierungstheorie soll ja helfen, wenn Fehler aufgetreten sind.

Die Situation ist in Abbildung 6 dargestellt. Die Punkte c_1 und c_2 stehen für Codeworte und der Kreis k_1 um c_1 für die Kugel $B_d(c_1)$ mit dem Radius d um c_1. Im Inneren dieser Kugel ist c_1 das einzige Codewort.

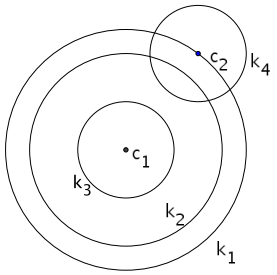

Abbildung 6: Codeworte und Kugeln

1. Gab es bei der Übertragung d oder sogar mehr Fehler, kann man wenig machen. Auf der Schale der d–Kugel $U_d(c)$ können Codeworte liegen, und wenn c' wieder ein Codewort ist, hat man nicht einmal eine Chance, den Fehler zu entdecken, geschweige denn, ihn zu korrigieren.

2. Vielleicht waren einige Stellen von c' nicht lesbar. Man spricht dann von **Auslöschung** einzelner Stellen. Wenn maximal $d-1$ Stellen nicht lesbar und keine weiteren Fehler aufgetreten sind, muss c' in der Kugel $B_{d-1}(c)$ liegen, in der Abbildung als k_2 dargestellt, und das einzige Codewort in dieser Kugel ist c. In diesem Fall kann das Originalcodewort c also rekonstruiert werden.

3. Wurde c falsch übermittelt, traten aber weniger als d Fehler auf, wird dies immerhin erkannt, denn dann ist c' kein Codewort.

4. Traten weniger als $\frac{1}{2}d$ Fehler auf, ist c das einzige Codewort in $U_r(c')$ für $r < \frac{1}{2}d$, denn die Kugeln mit dem Radius r um Codeworte überschneiden sich nicht. In der Abbildung sind diese Kugeln durch k_3 und k_4 dargestellt. In dem Fall kann c rekonstruiert werden.

3.8 Ein $[7, 4, 3]$–Code

Der schon erwähnte Richard Hamming hat sich eine Familie sehr raffinierter Codes mit schönen Eigenschaften ausgedacht. Die allgemeine Konstruktion ist hochinteressant, aber ohne eine gewisse Vertrautheit mit Linearer Algebra nicht recht zugänglich. Ich will euch hier wenigstens ein Beispiel eines Hamming–Codes zeigen[6]. Es ist ein $[7, 4, 3]$–Code, und anhand der Abbildung 7 ist er leicht zu verstehen.

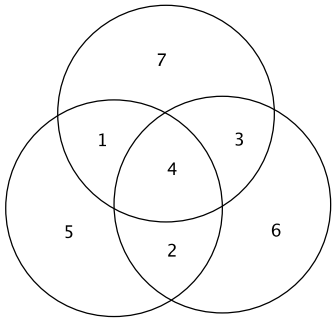

Abbildung 7: Zur Konstruktion eines $[7, 4, 3]$–Codes

[6]Siehe den Aufsatz von J. van Lint im Sonderheft Computeralgebra, S. 54.

Eine Nachricht ist ein Wort (c_1, c_2, c_3, c_4) der Länge 4 über dem Alphabet $F = \{0, 1\}$; die Einträge stehen in den Feldern $1, 2, 3$ und 4. In den Feldern $5, 6$ und 7 stehen Prüfziffern aus F, und zwar werden sie so gewählt, dass die Summe der Ziffern in jedem Kreis 0 ergibt. Es gilt also:

$$c_5 = c_1 + c_2 + c_4$$
$$c_6 = c_2 + c_3 + c_4$$
$$c_7 = c_1 + c_3 + c_4$$

Es ist klar, dass die Codeworte in F^7 liegen und dass es 2^4 davon gibt. Du kannst dir auch überlegen, dass der Code linear ist. Um den minimalen Hamming–Abstand zweier Codeworte zu finden, müsstest du alle 16 Codeworte bilden und schauen, wie groß die kleinste Anzahl von Einträgen $\neq 0$ ist, die in einem Codewort $\neq \vec{0}$ auftritt. Einen anderen Weg, der für euch gangbar ist, sehe ich nicht. Immerhin kannst du dazu GAP benutzen, wenn du magst: Starte GAP, definiere F und lasse dir die Elemente von F anzeigen. Definiere dann die Menge T aller 4–tupel mit Einträgen aus F, das ist die Menge der Nachrichten. Nun brauchst du nur noch die Prüfziffern zu ergänzen, dann hast du die Menge C. Die nötigen GAP–Befehle sind nicht allzu schwierig:

```
F:=GF(2);
List(F);
T:=Tuples(F,4);
C:=List(T,X->[X[1],X[2],X[3],X[4],X[1]+X[2]+X[4],X[2]+X[3]+X[4],X[1]+X[3]+X[4]]);
```

Unser neuer Code ist 2–fehlererkennend und 1-fehlerkorrigierend. Wollen wir diese Eigenschaften bei einem Wiederholungscode haben, brauchen wir bei Nachrichten der Länge 4 Codeworte der Länge 12, wie du dir selbst überlegen kannst; das bedeutet einen wesentlich höheren Aufwand. Es kommt aber noch eine weitere bemerkenswerte Eigenschaft unseres Codes hinzu:

Aufgabe

1. Berechne, wieviele Elemente von F^7 in einer 1–Kugel um ein $c \in C$ liegen.
2. Wie viele 1–Kugeln um Elemente aus C gibt es?
3. Begründe, dass niemals Elemente von F^7 gleichzeitig in $B_1(u)$ und in $B_1(v)$ liegen für verschiedene $u, v \in C$: Die Schnittmenge der 1–Kugeln um verschiedene Elemente von C ist stets leer!
4. Folgere aus den bisher erhaltenen Ergebnissen, dass die Vereinigung aller 1–Kugeln um Elemente aus C den ganzen Raum F^7 ergibt! Jedes $v \in F^7$ liegt also in genau einer 1–Kugel um ein Element aus C.

4 Das Programm GAP

Anfang der siebziger Jahre, als Computer noch ziemlich exotische Geräte waren, arbeitete an der Technischen Hochschule in Aachen Professor Neubüser mit einer Arbeitsgruppe an der „Untersuchung algebraischer Strukturen auf Computern“. Im Laufe der Zeit entstand in Zusammenarbeit mit Mathematikern in Australien und in England das Programm GAP. Ausgeschrieben bedeutet GAP Groups–Algorithms–Programming, und es ist ein Werkzeug für Spezialisten, um Dinge konkret auszurechnen. Es wird auch heute daran gearbeitet, federführend ist seit einigen Jahren eine Arbeitsgruppe im schottischen St. Andrews.

Der wichtigste Befehl für die Arbeit mit GAP heißt

```
quit;
```

Wenn dich das Programm mit kryptischen Fehlermeldungen überschüttet und bei jeder eine neue Schleife aufmacht, kommst du damit wieder zurück in deine alte Schleife und auch wieder aus dem Programm heraus. Jeder Befehl muss mit einem Semikolon abgeschlossen werden, Befehle tragen eine Art von Klartextnamen, die für den Fachmann sehr hilfreich sind, und durch die Tabulatortaste kann man einen angefangenen Befehl vervollständigen lassen oder, wenn dies nicht eindeutig geht, sich durch ein weiteres Drücken alle verfügbaren Vervollständigungen anzeigen lassen.

GAP ist sehr komplex, und das muss auch so sein, denn GAP kann in einer Unzahl verschiedenster algebraischer Strukturen konkret rechnen. Für numerische Mathematik ist es gänzlich unbrauchbar, es kennt nicht einmal Dezimalzahlen; die benutzt ein Algebraiker auch bloß beim Einkaufen.

Wenn du nun eine Gefühlsmischung aus Spannung, Neugierde und Furcht empfindest, solltest du dich an die Bearbeitung der Aufgaben über endliche Körper begeben.

5 Endliche Körper

5.1 Zur Geschichte

Ursprünglich sprach man von „Algebraischen Zahlenkörpern“ (Richard Dedekind, gegen 1870), aber das wurde bald verkürzt zu „Körpern “. Es war nicht an etwas Räumliches gedacht, sondern an etwas Organisches, Vollständiges. In der Tat sind die Zahlenmengen $\mathbb{N}$ und $\mathbb{Z}$ der natürlichen und der ganzen Zahlen unvollständig: in $\mathbb{N}$ kann man nicht uneingeschränkt subtrahieren wie in $\mathbb{Z}$, in $\mathbb{Z}$ kann man nicht uneingeschränkt dividieren. Diese Zahlenmengen sind keine Körper. Will man die Grundrechenarten ohne Einschränkungen ausführen, muss man schon zur Menge $\mathbb{Q}$ der rationalen Zahlen übergehen, die bildet dann endlich einen Körper.

Eine abstrakte Definition des Körpers und endliche Körper als Beispiele dafür gab Heinrich Weber in seinem Lehrbuch der Algebra, und deshalb habe ich ihn in der Einladung als Erfinder der Körper hervorgehoben. Im englischen Sprachraum heißen die endlichen Körper Galois–fields, nach dem jungen Franzosen Evariste Galois, der sich zur Zeit der französischen Revolution mit der Lösbarkeit von Gleichungen beschäftigte. Auf einen Streit, wer denn nun als Erfinder endlicher Körper anzusehen ist, will ich mich aber nicht einlassen.

Eine Definition eines Körpers ist ziemlich lang, nach meiner Meinung hättest du davon keinen großen Nutzen. Für uns reicht es völlig aus, uns einen Körper als eine Menge vorzustellen, die eine 0 und eine 1 enthält und in der man die vier Grundrechenarten vernünftig ausführen kann. Durch 0 teilen darfst du aber auch in allgemeinen Körpern nicht.

Wir schauen uns nun einige Beispiele an.

5.2 Der Körper F mit zwei Elementen

Das kleinste Körper hat nur zwei Elemente, also außer seiner 0 und seiner 1 keine weiteren. Wir wollen die Elemente auch einfach mit 0 und 1 bezeichnen, und der Körper soll F heißen:

$$F = \{0, 1\}$$

Wenn du die Aufgaben

$$0 \cdot 0,\ 0 \cdot 1,\ 1 \cdot 0,\ 1 \cdot 1,\ 0 + 0,\ 0 + 1,\ 1 + 0,\ 1 + 1$$

gelöst hast, hast du praktisch schon alle Rechnungen ausgeführt, die auftreten können. Viel kann man von einer so armen Struktur nicht erwarten, sollte man denken. Aber Vorsicht: ein Computer versteht im Prinzip auch nur 0 und 1, und dennoch hast du bunte Bilder auf dem Bildschirm. Lasse dich also probeweise auf den Körper F mit zwei Elementen ein und bearbeite die folgenden Aufgaben.

1. Was kommt bei den acht Aufgaben oben heraus?

2. In F wie in jedem Körper gelten die üblichen Rechengesetze (Assoziativgesetz und Kommutativgesetz für $+$ und $\cdot$, Distributivgesetz). Wie sieht es dann mit den binomischen Formeln aus? Überlege dir, dass es in F nur eine gibt, und berechne $(a+b)^2$, $(a+b)^3$ und $(a+b)^4$.

3. Löse die folgenden Gleichungen!

   (a) $x+1=0$

   (b) $x^2+1=0$

   (c) $x^2+x+1=0$

   Anders als im Alltag kannst du zur Not alle Körperelemente einfach durchprobieren.

4. Als Lösung der Gleichung $x+1=0$ hast du $x=1$ gefunden. Eigentlich erwartet man ja $x=-1$, also die Gegenzahl von 1. Hier muss man der Tatsache ins Auge sehen, dass $1=-1$ ist. In einem Körper, in dem $1+1=0$ ist, ist $-$ und $+$ dasselbe. Mache dir das einen Augenblick lang klar.

5. Berechne $(x^2+x+1)(x+1)$.

6. Das Polynom x^3+x^2+x+1 hat die Nullstelle $x=1$. Dann sollte man mit Hilfe der Polynomdivision den Linearfaktor $x+1$ abspalten können! Führe das durch!

5.3 Der Körper F mit GAP

Wir wollen uns vorsichtig GAP nähern und ausprobieren, was das Programm bei den Aufgaben des letzten Abschnitts für uns tun kann.

Wenn du GAP gestartet hast, siehst du einen Eingabewinkel. Das Programm kennt den Körper mit zwei Elementen unter dem Namen $GF(2)$. Wir wollen GAP dafür den vertrauten Namen F vorgeben und uns anschauen, wie GAP die Elemente schreibt.

```
F:=GF(2);
List( F );
```

Anstelle von 0 und 1, wie du vielleicht erwartet hast, gibt GAP $0*Z(2)$ und $Z(2)$ aus. Muss das sein? Ja, natürlich. Für die $1\in\mathbb{Q}$ ist $1+1=2$, für die $1\in F$ ist $1+1=0$. GAP kann in einer Unzahl von Strukturen rechnen, da muss stets klar sein, in welcher Struktur man sich gerade befindet. Immerhin können wir den Elementen bequemere Namen geben:

```
e:=Z(2); o:=0*Z(2);
```

Nun kannst du die Terme $0+0$ usw. auswerten.

Schauen wir uns nun die zweite Aufgabe an. Wenn du einen der Terme eingibst, wirst du sofort eine Fehlermeldung erhalten; GAP will wissen, was a und b sein sollen. Du musst GAP mitteilen, dass a und b Variable sein sollen, mit denen du über dem Körper F rechnen willst:

```
a:=Indeterminate(F,"a");
b:=Indeterminate(F,"b");
```

Nun kannst du dir die Terme in Aufgabe 2 von GAP auswerten lassen. Gib sie einfach ein.

Bei der nächsten Aufgabe geht es um Nullstellen von Polynomen in der Variablen x. Auch die Variable x muss gegenüber GAP erst wieder erklärt werden! Du weißt ja wie das geht. Wenn du das erledigt hast, kannst du GAP nach den Nullstellen der Polynome fragen:

```
x+e; RootsOfUPol(last);
```

Hier habe ich erst das Polynom selbst eingegeben, um überprüfen zu können, ob die Eingabe stimmt. Das „last“ bedeutet für GAP immer die letzte Ausgabe, wie das % bei MuPAD.

Beim Polynom x^2+x+1 findet GAP keine Lösung, es gibt ja auch keine in F. Aber du kannst GAP in einem größeren Bereich suchen lassen:

```
x^2+x+e; RootsOfUPol("split", last);
```

Die Aufgabe 5 kannst du ohne Hilfe. Für Aufgabe 6 brauchst du neue Befehle. Die nötigen Eingaben sind:

```
EuclideanRemainder(x^3+x^2+x+e,x+e);
EuclideanQuotient(x^3+x^2+x+e,x+e);
```

Auf die Ausgaben wirst du dir schon einen Reim machen können.

Wenn du Eingaben sparen willst, kannst du mit

```
Read("poly.gap");
```

die Datei poly.gap einlesen lassen, dann hast du fertige Befehle und eine Gebrauchsanweisung dafür zur Hand.

5.4 Der Körper K mit acht Elementen

Wir schauen uns jetzt einen Körper mit acht Elementen an, in GAP ist er mit $GF(8)$ bezeichnet. Das „GF“ steht für Galois–field, und 8 gibt die Anzahl der Elemente an.

Starte GAP und definiere K als $GF(8)$:

```
K:=GF(8);
```

GAP meldet nur, dass es den Befehl verstanden hat. Wir wollen die Elemente des Körpers gern sehen, und wir lassen uns eine Liste ausgeben:

```
List(K);
```

Die Elemente der Liste werden, durch Kommata getrennt, von GAP aufgezählt, und die Aufzählung wird durch eckige Klammern eingeschlossen. Hier sind es acht Elemente, und die Schreibweise ist für dich sicher etwas gewöhnungsbedürftig. Die Elemente auf den Plätzen drei bis acht sind Potenzen eines Elementes $Z(2^3)$. Hier begegnest du wieder dieser eminent wichtigen Eigenschaft endlicher Körper. Die Fachleute sagen, die multiplikative Gruppe des Körpers sei zyklisch, und sie drücken damit aus, dass alle von Null verschiedenen Elemente des Körpers Potenzen eines geeigneten Körperelementes sind.

Wir wollen dieses Element $Z(2^3)$ der Einfachheit halber mit a bezeichnen:

```
a:=Z(2^3);
```

Lasse dir von GAP eine Anzahl von Potenzen von a berechnen und schaue dir die Ergebnisse genau an. Du wirst sicher herausfinden, dass $Z(2^3)^7 = Z(2^3)^0$ das Einselement des Körpers ist. GAP bezeichnet es mit $Z(2)^0$. Entsprechend ist $0 * Z(2)^0$ die Null. Prüfe das nach, zum Beispiel, indem du

```
a^3;
a^3*a^7;
0*Z(2)^0+a^3;
```

eingibst.

Eine spannende Frage ist nun, was herauskommt, wenn man Potenzen von a addiert. Probiere es aus!

Die Summe von zwei Potenzen von a ist im Allgemeinen wieder eine Potenz von a, und du wirst kein rechtes System dabei erkennen, außer dass $a^k + a^k$ stets Null ergibt. Natürlich steckt ein System dahinter: Es gibt ein Polynom $p(x)$ vom Grad drei, dessen Koeffizienten nur Null oder Eins sind und das a als Nullstelle hat. Du kannst selbst dieses Polynom suchen oder du kannst es dir von GAP ausgeben lassen. Dazu kannst du den kryptischen Befehl

```
DefiningPolynomial(K);
```

verwenden. In gewöhnlicher Schreibweise handelt es sich um das Polynom $x^3 + x + 1$. Du kannst die Möglichkeiten von GAP nutzen und in das Polynom a für x einsetzen:

```
p:=DefiningPolynomial(K);
Value(p,a);
```

Dieses Polynom spielt für den Körper K eine außerordentlich wichtige Rolle, und ich werde versuchen, dir wenigstens eine kleine Ahnung dieser Wichtigkeit zu vermitteln: Lege eine Liste

$$[1, x, x^2, \ldots, x^7]$$

von Polynomen über dem Körper $F = \{0, 1\}$ mit zwei Elementen an und bilde zu jedem dieser Polynome den Rest, den du bei Division durch das Polynom p erhältst. Du kannst das zu Fuß erledigen, das ist eine gute Übung, oder du kannst GAP benutzen. Dann musst du allerdings einige kryptische Eingaben machen:

```
R:=PolynomialRing( GF(2), ["x"] );
x:=IndeterminatesOfPolynomialRing(R)[1];
p:=x^3+x+1;
L:=List( [0..6], k->x^k );
reste:=List( L, X->EuclideanRemainder(X,p) );
```

Die Befehle sind so, dass du möglichst wenig Tipparbeit hast; ich lasse GAP gleich auf den Listen operieren. Probiere ruhig, einzelne Ergebnisse selbst zu berechnen.

Wenn du dir die Liste *reste* der Ergebnisse anschaust, wirst du feststellen, dass jedes der 7 Polynome $\neq 0$ vom Grad ≤ 2 genau einmal in der Liste auftritt. Die Sache läuft nun so: Wenn du wissen willst, was $a^4 + a^3$ ist, bildest du die Reste von x^4 und von x^3 bei Division durch p und addierst diese. Dann schaust du, welches x^k zur erhaltenen Summe gehört. Das entsprechende a^k ist die Summe von a^3 und a^4. Hier ergibt sich $a^3 + a^4 = a^6$.

Solltest du einmal Algebra lernen, wirst du sehen, dass man $GF(8)$ mit Hilfe von $GF(2)$ und $p = x^3 + x + 1$ überhaupt erst konstruiert; diese Methode geht auf den Berliner Mathematiker Leopold Kronecker zurück. Wir können uns damit jetzt nicht näher befassen.

6 Reed–Solomon–Codes

Bei der Aufbereitung der Daten, die auf eine CD geschrieben werden sollen, benutzt man Reed–Solomon–Codes. Ich zeige dir an einem Beispiel, wie man die konstruiert.

1. Man braucht einen endlichen Körper, er darf nicht zu klein sein. Wir nehmen hier unseren Körper K mit acht Elementen, den du schon kennst.

2. Wähle aus K eine Anzahl n verschiedener Elemente. Du darfst keines zweimal verwenden! Der Code wird aus Worten der Länge n bestehen. Wir nehmen hier $n = 7$, und die Menge M der gewählten Zahlen sei etwa

$$M := \{Z(8), Z(8)^2, \ldots, Z(8)^7\} \quad .$$

3. Wähle eine natürliche Zahl $k \leq n$. Der Code wird q^k Elemente haben, in unserem Beispiel werden das $|C| = 8^k$ sein. Wir wählen hier $k = 3$.

4. Es sei P_k die Menge aller Polynome vom Grad $< k$ mit Koeffizienten aus K. Jedes Polynom $f \in P_k$ liefert nun ein Codewort c_f. Es entsteht einfach durch Einsetzen der oben gewählten Zahlen in f. In unserem Beispiel wäre

$$c_f = \big(f(Z(8)), f(Z(8)^2), \ldots, f(Z(8)^7)\big) \quad .$$

Der so konstruierte Code ist linear, denn Summen und K–Vielfache von Polynomen aus P_k liegen wieder in P_k. Verschiedene Polynome liefern auch verschiedene Codeworte, denn ein Polynom vom Grade $< k$ hat höchstens $k - 1$ Nullstellen. Das heißt aber gleichzeitig, dass höchstens $k - 1$ der Einträge eines jeden Codewortes $\neq \vec{0}$ die Null aus K sein können. Daraus folgt für den Parameter d des Codes die interessante Tatsache

$$d \geq n - (k - 1) \quad ,$$

hier konkret $d \geq 7 - 2 = 5$. Weil das Polynom

$$f = (x - Z(8))(x - Z(8)^2)$$

ein Codewort liefert, das an den beiden ersten Stellen $0 \in K$ hat, ist $d = 5$. Unser Code ist folglich 4–fehlererkennend und 2–fehlerkorrigierend!

9 Lemma
Es sei C ein $[n, k, d]$–Reed–Solomon–Code. Dann ist $d = n - (k - 1)$.

Für die Handhabung von Reed–Solomon–Codes gibt es leistungsfähige Algorithmen, und man kann leicht solche Codes mit großen d–Werten konstruieren. Das macht sie für die Praxis sehr geeignet.

7 Über CDs

Wir wollen uns nun anschauen, wie Daten auf Audio–CDs geschrieben werden.

7.1 Die Rohdaten

Gespielte Musik kann man mit Mikrofonen aufnehmen, man erhält dann ein kontinuierliches Signal, das heißt eine Größe, die zu jedem Zeitpunkt einen Wert annimmt. Man kann das Signal als Kurve auf dem Bildschirm eines Oszilloskops anschauen. Moderne Datenverarbeitung handelt aber mit diskreten Größen, also Sammlungen von Einzelwerten: Man muss das kontinuierliche Signal diskretisieren, also etwa den Wert der Größe in gleichen Zeitabständen messen. Man nennt dies „Abtasten“ des Signals.

Menschen hören noch Frequenzen von 20.000 Hz. Wenn man das kontinuierliche Signal angemessen erfassen will, muss man je Sekunde mindestens 40.000 Messwerte erheben!

Die Messwerte sind Zahlen zwischen 0 und $2^{16} - 1$, nehmen wir als Beispiel

$$m = 53261 = 2^0 + 2^2 + 2^3 + 2^{12} + 2^{14} + 2^{15} \quad . \tag{6}$$

Man stellt m nun als 16-Tupel über $F = \{0, 1\}$ dar:

$$m \longleftrightarrow (1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1) \tag{7}$$

zerlegt dieses 16–tupel in zwei 8–tupel und fasst jedes dieser 8–tupel als Element eines Körpers K mit 2^8 Elementen auf:

$$m \longleftrightarrow ((1, 0, 1, 1, 0, 0, 0, 0), (0, 0, 0, 0, 1, 0, 1, 1)) \longleftrightarrow (\alpha_1, \alpha_2) \tag{8}$$

Mit $z := Z(2^8)$ in der Sprache von GAP wäre hier

$$\alpha_1 = z^0 + z^2 + z^3 \in K \quad .$$

Jeder Messwert liefert somit ein Paar von Elementen aus $K \cong GF(2^8)$. Üblicherweise werden Stereoaufnahmen gemacht, also zu jedem Zeitpunkt zwei Werte gemessen. Dann liefert jede Abtastung ein 4–tupel

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in K^4 \quad .$$

Man fasst nun jeweils sechs dieser 4–tupel zusammen und hat dann die Information, die auf die CD geschrieben werden soll, als 24–tupel

$$b = (\alpha_1, \alpha_2, \ldots, \alpha_{24}) \in K^{24} \tag{9}$$

vorliegen. Ein solches Wort wird Audiowort genannt, und es enthält $24 \cdot 8 = 192$ Bit.

7.2 Erste Codierung

Das Wort b aus Gleichung 9 muss fehlerfrei wiederhergestellt werden, deshalb wird es codiert. Man schreibt es erst einmal als Wort c eines $[28, 24, 5]$–Reed–Solomon–Codes C über K:

$$b \longleftrightarrow c \in C \subseteq K^{28} \tag{10}$$

7.3 Interleaving und zweite Codierung

Man könnte nun versuchen, dieses Codewort c aus Gleichung 10 auf die CD zu schreiben. Der Code C ist 4–fehlererkennend und 2–fehlerkorrigierend, so schlecht ist das nicht. Das Problem bei CDs ist, dass es Kratzer gibt, und dann fallen sehr viele Einträge in Folge aus, viel mehr als die zwei, die korrigiert werden können. Deshalb sollen benachbarte Einträge in c auf der CD an Plätzen stehen, die weit auseinander liegen. Das erreicht man mit dem folgenden Trick, ich stelle eine einfache Version dar: Man nimmt 28 im Datenstrom aufeinander folgende Worte

$$c_i = (c_{i1}, c_{i2}, c_{i3}, \ldots, c_{i28}), \quad i = 1, 2, 3, \ldots, 28$$

und ordnet sie in einem rautenförmigen Muster an:

$$\begin{array}{ccccc} c_{11} & c_{21} & c_{31} & c_{41} & \ldots \\ & c_{12} & c_{22} & c_{32} & \ldots \\ & & c_{13} & c_{23} & \ldots \\ & & & c_{14} & \ldots \\ & & & & \ddots \end{array} \tag{11}$$

Die Einträge eines jeden C–Wortes stehen quasi auf einer diagonalen Linie.

Die einzige vollständige **Spalte** in diesem Schema ist

$$\begin{pmatrix} c_{28\ 1} \\ c_{27\ 2} \\ \vdots \\ c_{i\ 29-i} \\ \vdots \\ c_{1\ 28} \end{pmatrix} \quad .$$

Diese Spalte codiert man als Wort $\bar{c}$ eines $[32, 28, 5]$–Reed–Solomon–Codes $\bar{C}$:

$$(c_{28\ 1},\ c_{27\ 2},\ c_{26\ 3},\ \ldots,\ c_{1\ 28}) \longleftrightarrow \bar{c} \in \bar{C} \subseteq K^{32} \tag{12}$$

Das nächste C–Wort c_{29} fügt man einfach rechts an das Schema in Gleichung 11 an, erhält die nächste volle Spalte und daraus das nächste $\bar{C}$–Wort.

Die sechs Abtastungen liegen nun in einem Wort von $32 \cdot 8 = 256$ Bit vor.

7.4 Die CD

Auf einer Schallplatte gibt es eine Spur, die von einer Nadel abgetastet wird. Auch eine CD hat eine Spur.[7] Sie ist etwa 5 Kilometer lang, 0,6 Mikrometer breit, und sie wird von einem Laser gelesen. In der Spur gibt es Vertiefungen, so genannte Pits, und Nichtvertiefungen, so genannte Lands. Die Übergänge Pit/Land und Land/Pit stehen für eine 1, etwa 0,3 Mikrometer Pit oder Land für eine 0.

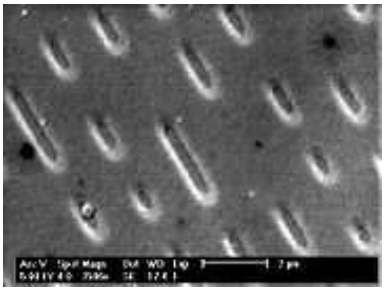

Abbildung 8: Pits und Lands auf einer CD unter dem Elektronenmikroskop

Leider kann man das Wort $\bar{c}$ aus Gleichung 12 nicht direkt auf die CD schreiben. Die Übergänge dürfen nicht zu dicht auf einander folgen, und es darf nicht zu lange Strecken ohne Übergänge geben. Um dieses zu gewährleisten, werden in $\bar{c}$ weitere Bits ergänzt. Insgesamt liefern unsere sechs Abtastungen 588 Bits, die auf die CD geschrieben werden können.

Eine Sekunde Musik erfordert etwa 44.000 Abtastungen, und die ergeben $44000/6 \cdot 588 =$ 4.312.000 Bits, und dies entspricht etwa 1,3 Millimeter Spurlänge auf der CD.

7.5 Decodierung

Wird die CD abgespielt, muss das Gerät aus 588 gelesenen Bits das Wort $\bar{c}$ aus Gleichung 12 rekonstruieren. Stellt man nun fest, dass das gelesene Wort $\bar{c}'$ falsch ist, wird es zu einem Codewort $\bar{c}$ korrigiert, falls es in der 1–Kugel $B_1(\bar{c})$ eines Wortes $\bar{c} \in \bar{C}$ liegt. Andernfalls wird das gelesene Wort einfach gestrichen. Der Code C der ersten Codierung kann bis zu vier Auslöschungen in einem Wort korrigieren. Durch den Interleavingtrick verteilen sich die 28 ausgelöschten Stellen auf 28 Codeworte c_i. Die Information in bis zu vier aufeinander folgenden fehlerhaften Worten $\bar{c}_i$ kann also rekonstruiert werden, das entspricht $4 \cdot 588 \cdot 0,0003 \approx 0,7$ Millimeter Spurlänge auf der CD.

In Wirklichkeit wird ein komplizierteres Interleaving durchgeführt, und zwar erstellt man ein Schema aus $4 \cdot 28 = 112$ Worten $c_1, c_2, \ldots, c_{112} \in C$:

$$\begin{array}{cccccccccc} c_{11} & c_{21} & c_{31} & c_{41} & c_{51} & c_{61} & c_{71} & c_{81} & c_{91} & \ldots \\ & & & & c_{12} & c_{22} & c_{32} & c_{42} & c_{52} & \ldots \\ & & & & & & & & c_{13} & \ldots \\ & & & & & & & & & \ddots \end{array}$$

[7] Ich folge hier der Darstellung auf der Seite 28 in dem schönen Buch von Wolfgang Willems.

Dieses Schema enthält dann vier vollständige Spalten, die erste von ihnen ergibt

$$(c_{109\ 1}, c_{105\ 2}, c_{101\ 3}, \ldots, c_{1\ 28}) \ .$$

Hier kann die Information aus bis zu 16 auf einander folgenden fehlerhaften Worten $\bar{c}$ rekonstruiert werden, das entspricht $2,8$ Millimeter Spurlänge auf der CD.

8 Noch ein konkretes Beispiel

Als Nachrichtenvorrat nehmen wir die Menge aller Worte der Länge vier, die man mit dem Alphabet $\{0,1\}$ schreiben kann, also die Menge der 4–tupel mit Einträgen aus dem Körper F mit zwei Elementen. Dem 4–tupel $a = (a_0, a_1, a_2, a_3)$ ist in natürlicher Weise ein Polynom vom Grad ≤ 3 mit Koeffizienten in F zugeordnet:

$$a = (a_0, a_1, a_2, a_3) \mapsto a_0 + a_1 x + a_2 x^2 + a_3 x^3 \ ,$$

und die Verwandtschaft zwischen dem 4–tupel a und dem Polynom ist so eng, dass wir auch das Polynom mit a bezeichnen wollen.

Das Codewort c zu a ist dann das Polynom, das man erhält, wenn man das Polynom zu a mit dem festen **Erzeuger–Polynom** $e := x^3 + x + 1$ multipliziert[8]:

$$a = (a_0, a_1, a_2, a_3) \mapsto a_0 + a_1 x + a_2 x^2 + a_3 x^3 \mapsto (a_0 + a_1 x + a_2 x^2 + a_3 x^3)(1 + x + x^3)$$

Auf den ersten Blick sieht die Konstruktion nicht einfach aus. Bevor wir fragen, was wir für den Aufwand bekommen, sollst du dich ein wenig mit dem Gegenstand vertraut machen. Bearbeite die folgenden Aufgaben. Rechne so weit wie möglich mit der Hand und benutze dann GAP. Um lästige Tipparbeit zu sparen, kannst du nach dem Start von GAP

```
Read("poly.gap");
```

eingeben.

1. Wieviele Elemente hat der Code C, den wir gerade konstruiert haben, und wieviele Polynome vom Grad ≤ 6 mit Koeffizienten in F gibt es insgesamt?
2. Durch welche Polynome werden die 4–tupel $(1,1,0,1)$, $(1,0,0,0)$ und $(0,1,1,0)$ codiert?
3. Zu welchen Nachrichten gehören die Polynome $1 + x^4 + x^5$ und $1 + x^2 + x^6$?
4. Die Polynome $1+x+x^4+x^5$, $x+x^6$ und $1+x$ gehören nicht zu C. Kannst du das begründen?
5. Nimm an, die Worte aus der vorigen Aufgabe seien fehlerhaft übertragene Codeworte. Welche Nachrichten könnte man ihnen zuordnen?
6. Du hast schon das Element $Z(8)$ kennengelernt, einen Erzeuger der multiplikativen Gruppe des Körpers K vom Typ $GF(8)$. Der Körper F ist in K enthalten, wir können also unsere Polynome auch als Polynome über K ansehen. Prüfe nach, dass $Z(8)$ Nullstelle von e ist.
7. Prüfe nach, dass e Teiler von $x^7 - 1$ ist.
8. Genau dann gehört ein Polynom vom Grad ≤ 6 mit Koeffizienten aus F zu C, wenn $Z(8)$ Nullstelle des Polynoms ist. Prüfe dies an Beispielen nach.
9. *Beweise: Wenn $a = (a_0, a_1, \ldots, a_6)$ Codewort ist, dann ist auch $a\tau := (a_6, a_0, a_1, \ldots, a_5)$ Codewort, und umgekehrt. Das Wort $a\tau$ erhält man aus a durch zyklische Vertauschung, und der Code C heißt wegen dieser Eigenschaft zyklisch.

[8] Dieses Beispiel ist in dem Aufsatz von Felix Ulmer in dem Sonderheft ‚Computeralgebra' beschrieben.

10. *Beweise: Bei jedem Polynom in C sind mindestens drei Koeffizienten $\neq 0$. [Lösungshinweis: Gäbe es in C ein Polynom mit weniger als drei Koeffizienten $\neq 0$, erhielte man durch zyklische Vertauschung in C ein von e verschiedenes Polynom vom Grad ≤ 3, und das kann es nicht geben.]

11. *Beweise: Der Hamming–Abstand zweier verschiedener Polynome in C ist mindestens 3. [Lösungshinweis: Verwende die vorige Aufgabe.]

9 Hamming–Codes*

Dieses Kapitel werden wir am Samstag nicht besprechen können, aber der Gegenstand ist so interessant, dass ich ihn hier knapp skizzieren möchte.

Es sei F ein Körper, und es sei $|F| = q$. Im Raum F^m gibt es $q^m - 1$ Punkte $\neq \vec{0}$. Jeweils $q-1$ Punkte $\neq \vec{0}$ liegen auf einer Geraden durch den Nullpunkt, es gibt also

$$n := \frac{q^m - 1}{q - 1}$$

solcher Geraden. Wir wählen einen Punkt $\neq \vec{0}$ auf jeder dieser Geraden, das gibt dann n Punkte bzw. m–tupel

$$h_1, h_2 \ldots, h_n \in F^m \quad ,$$

und setzen

$$C := \left\{ c \in F^n \,\middle|\, \sum_{i=1}^{n} c_i h_i = \vec{0} \right\} \quad .$$

Dann ist $C \subseteq F^n$ ein linearer Code.

Wir können für $h_1, h_2, \ldots, h_m$ die Einheitsvektoren in F^m wählen, also hat die $n \times m$– Koeffizientenmatrix des Linearen Gleichungssystems in der Definition von C den Rang m und die Lösungsmenge C dieses Linearen Gleichungssystems die Dimension $n - m$. Folglich ist C ein $[n, n-m]$–Code.

Den Parameter d des Codes bestimmen wir mit der folgenden kleinen Überlegung: Es sei $c \in F^n$, und es seien

$$c_{i_1}, c_{i_2}, \ldots, c_{i_j}$$

die Einträge $\neq 0$ in c. Dann liegt c genau dann in C, wenn die Linearkombination

$$\sum_{l=1}^{j} c_{i_l} h_{i_l}$$

den Nullvektor ergibt, also wenn

$$\{h_{i_1}, h_{i_2}, \ldots, h_{i_j}\}$$

linear abhängig ist. Da nie zwei verschiedene h_i auf derselben Geraden durch $\vec{0}$ liegen, muss die Anzahl j der h_{i_l} mindestens 3 sein: Es folgt

$$d \geq 3 \quad .$$

Nun erzeugen h_1 und h_2 eine Ebene, und diese Ebene enthält mindestens noch die Gerade durch $\vec{0}$ und $h_1 + h_2$. Ein Punkt dieser Geraden muss unter den h_i sein. Ohne Beschränkung der Allgemeinheit können wir

$$h_3 = h_1 + h_2$$

annehmen, das heißt aber, dass

$$(1, 1, -1, 0, 0, \ldots, 0) \in C$$

und somit $d(C) \leq 3$ ist. Es folgt

$$d = 3 \quad ,$$

unser Code ist ein $[n, n-m, 3]$–Code.

Der so konstruierte $[n, n-m, 3]$–Code mit $n = \frac{q^m-1}{q-1}$ heißt **Hamming–Code**. Er hat noch weitere schöne Eigenschaften, auf die ich hier nicht mehr eingehe. Du findest mehr darüber in dem Buch von Willems auf Seite 19.

10 Literaturhinweise

Zuerst nenne ich das Buch „Codierungstheorie und Kryptographie“ von Wolfgang Willems, Professor an der Universität Magdeburg (Birkhäuser: Basel 2008). Das Buch bietet eine ausgezeichnete Einführung in die Codierungstheorie.

Interessantes über endliche Körper bietet das Buch „Galoisfelder, Kreisteilungskörper und Schieberegisterfolgen“ von Heinz Lüneburg (Bibliographisches Institut: Mannheim 1979). Das Buch ist leider vergriffen, aber ein Exemplar steht bei Herrn Ribbert in der Zentralbücherei.

Alles über GAP findet man unter
http://www.gap-system.org ,
einer Seite der Universität St. Andrews in Schottland.[9]

Eine gut lesbare Darstellung unseres Gegenstandes hat Felix Leinen, Professor an der Universität Mainz, für eine Lehrerfortbildung erstellt. Sie ist unter der Adresse
www.mathematik.uni-mainz.de/schule/lehrkraeftefortbildungen/2006.2/cd-codierung-2.pdf
im Netz zu finden. Die Darstellung des Rautenmusters auf der Seite 24 dieses Papiers ist allerdings fehlerhaft.

Hoffentlich hast du ein Exemplar des Sonderheftes der Fachgruppe Computeralgebra der Deutschen Mathematiker–Vereinigung zum Jahr der Mathematik bekommen, sonst kannst du dir das Heft gegen eine Gebühr schicken lassen oder aus dem Internet herunterladen:
http://www.fachgruppe-computeralgebra.de/JdM
Für unser Thema sind die Artikel von F. Ulmer und von J. van Lint von Interesse. Die Autoren des Heftes haben sich besondere Mühe gegeben, für Nichtfachleute verständlich zu schreiben; grundsätzlich ist das Lesen von mathematischer Fachliteratur ein eher mühevolles Geschäft.

[9] The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.4.12; 2008. (http://www.gap-system.org)