Selbstkorrigierende Kodierungen II

Im vorherigen Kapitel haben wir uns damit beschäftigt, Fehler, die bei der Übertragung von Daten entstehen, zu finden und zu korrigieren. Dabei haben wir uns zwei Arten von Kodierungen angeschaut:

- Fehlererkennende Kodierungen sind darauf ausgelegt, dass Fehler in der Kodierung von Information erkannt werden können. Bei aufgetretenem Fehler in dieser Kodierung müssen die Daten neu angefragt werden, um mit dem Fehler umgehen zu können.
- Fehlerkorrigierende Kodierungen gehen einen Schritt weiter, indem Fehler nicht nur erkannt, sondern auch korrigiert werden können. Dies ist von Vorteil, da bei aufgetretenen Fehlern die Daten nicht direkt neu angefragt werden müssen, sondern in einigen Fällen direkt korrigiert werden können.

Dabei haben wir für die fehlerkorrigierenden Kodierungen den Kartentrick gesehen; dies ist eine einfache Konstruktion für 1-fehlerkorrigierende Kodierungen. In der Praxis besteht jedoch keine Garantie, dass bei der Übertragung von Daten höchstens ein Fehler geschieht und somit nur ein Bit geändert wird. In diesem Kapitel wollen wir uns einer neuen Herausforderung stellen und Herangehensweisen betrachten, um mehr als nur einen Fehler korrigieren zu können.

Eines der Ziele dieses Kapitels ist, Techniken zu erarbeiten, die uns erlauben, mehrere Fehler zu korrigieren. Bevor wir damit anfangen, wollen wir uns zuerst damit beschäftigen, wie oft wir mehrere Fehler beobachten können und diese dementsprechend korrigieren wollen.

Aufgabe 1

Wir betrachten das folgende Szenario: Eine Nachricht der Länge 100 soll verschickt werden, wobei wir wissen, dass bei der Übertragung fünf Fehler an zufälligen Positionen geschehen. Wenn wir die Nachricht in fünf Blöcke der Länge 20 einteilen würden, würden wir erwarten, dass pro Block genau ein Fehler geschieht - mit diesem Szenario könnten wir schon umgehen. Wir beabsichtigen nun, dieses Szenario mithilfe von Code zu widerlegen.

(a) Studiere den folgenden Code und erkläre dessen Verhalten. Führe dann das Programm aus und notiere deine Beobachtungen.

(b) Modifiziere das Programm so, dass die Nachricht in 2, 5, 10 und 20 Blöcke eingeteilt werden kann. Lass dann das Programm nochmals laufen und notiere die Resultate.

(c) Bespreche das beobachtete Verhalten mit deiner Banknachbarin oder deinem Banknachbarn und und notiere die wichtigsten Feststellungen.

```python
import random
versuche = 100
max_fehler_zaehler = [0] * 5

for i in range (versuche):
    # Für jeden Versuch wählen wir 5 zufällige Bits zwischen 0-99,
    welche die Fehler repräsentieren
    fehler = random.sample(range(100), 5)

    fehler_pro_block = [0] * 5

    for f in fehler:
        # Bestimme den Block, in dem der Fehler f geschehen ist
        block = f//20

        # Anzahl Fehler in diesem Block erhöhen
        fehler_pro_block[block] += 1

    # Bestimme die grösste Anzahl beobachteter Fehler in einem
    Block
    max_fehler = max(fehler_pro_block)

    # Zähler erhöhen
    max_fehler_zaehler[max_fehler-1] += 1


print (max_fehler_zaehler)
print("Bei {} Versuchen hatten wir ...".format(versuche))
for i in range(5):
    # Die Anzahl ausgeben
    print("... {}-mal maximal {} Fehler in jedem Block.".format(
    max_fehler_zaehler[i], i+1))
```

Lösung

(a) Eine mögliche Ausgabe könnte wie folgt aussehen:

```
Bei 100 Versuchen hatten wir ...
...  8-mal maximal 1 Fehler pro Block.
...  63-mal maximal 2 Fehler pro Block.
...  28-mal maximal 3 Fehler pro Block.
...  1-mal maximal 4 Fehler pro Block.
...  0-mal maximal 5 Fehler pro Block.
```

Wenn man das Programm laufen lässt, merkt man also, dass bis zu zwei Fehler in einem Block am wahrscheinlichsten sind. Insbesondere kommt eine gleichmässige Verteilung der Fehler auf je einen Block seltener vor als bis zu drei Fehler in einem Block.

(b) Eine mögliche Lösung wäre:

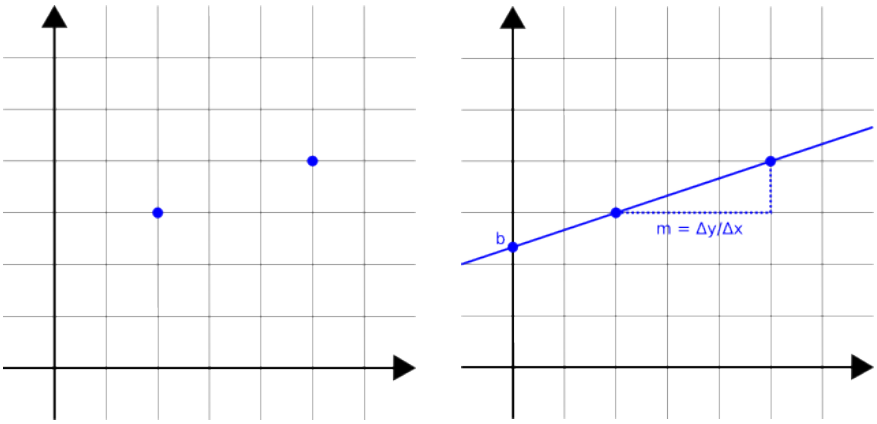
```python
import random
versuche = 100
anzahl_bloecke = [2, 5, 10, 20]
```

```
# Wir teilen nun die Nachricht in b Blöcke auf
for b in anzahl_bloecke:

    max_fehler_zaehler =  [0] * 5

    for i in range (versuche):
        # Für jeden Versuch wählen wir 5 zufällige Bits zwischen 0-99, welche die Fehler repräsentieren
        fehler = random.sample(range(100), 5)

        fehler_pro_block = [0] * b

        for f in fehler:
            # Bestimme den Block, in dem der Fehler f geschehen ist
            block = f//(100//b)

            # Anzahl Fehler in diesem Block erhöhen
            fehler_pro_block[block] += 1

        # Bestimme die grösste Anzahl beobachteter Fehler in einem Block
        max_fehler = max(fehler_pro_block)

        # Zähler erhöhen
        max_fehler_zaehler[max_fehler-1] += 1


    print (max_fehler_zaehler)
    print("Bei {} Versuchen und {} Blöcken hatten wir ..."
    .format(versuche, b))
    for i in range(len(max_fehler_zaehler)):
        # Die Anzahl ausgeben
        print("... {}-Mal maximal {} Fehler in jedem Block
    .".format(max_fehler_zaehler[i], i+1))
```

Wir können feststellen, dass bei vielen Blöcken (i.e. 20) die Fehler öfter gleichmässig aufgeteilt sind und dass es in der Regel nicht mehr als einen Fehler pro Block gibt. Allerdings ist die Wahrscheinlichkeit immer noch relativ hoch (um die 30% bei 20 Blöcken), dass mindestens zwei Fehler in einem Block geschehen, weswegen wir das Problem der Fehlerkorrektur bei zwei oder mehr Fehlern betrachten und lösen werden.

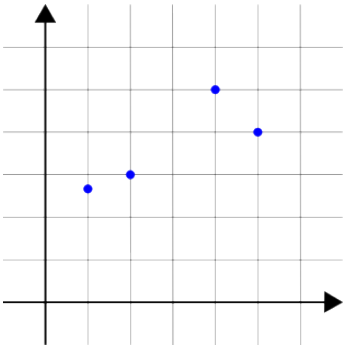
Reed-Solomon Codes

Eine Möglichkeit mehrere Fehler zu korrigieren baut auf dem Konzept der Polynome auf. Zur Veranschaulichung beschränken wir uns hier auf deren simpelste Form, und zwar auf lineare Funktionen.
Im Folgenden werden wir also sehen, wie wir eine lineare Funktion anwenden können, um zuerst einen und dann mehrere Fehler zu korrigieren.
Aus dem Mathematikunterricht wissen wir, dass eine Gerade durch zwei Punkte komplett beschrieben werden kann:



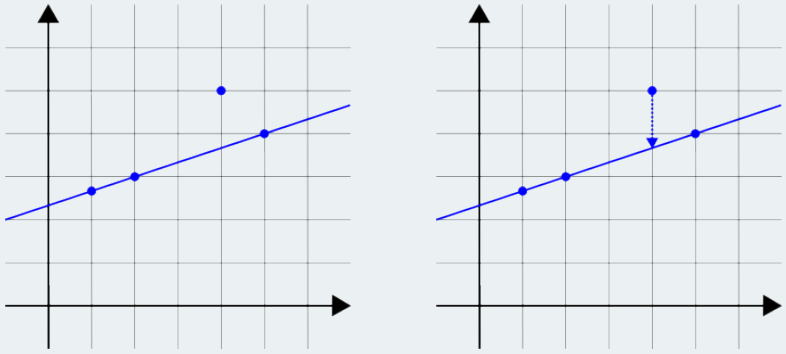
Beispiel 1

Aber wie können wir lineare Funktionen nutzen, um Fehler zu korrigieren? Angenommen wir haben eine Anzahl Punkte auf einer Geraden. Wenn einer dieser Punkte nach oben oder nach unten verschoben worden ist, würden wir das sofort merken. Schauen wir uns dies anhand des folgenden Beispiels an. Wir haben vier Punkte gegeben, wobei leider einer dieser Punkte vertikal verschoben wurde. Welcher dieser Punkte ist es?



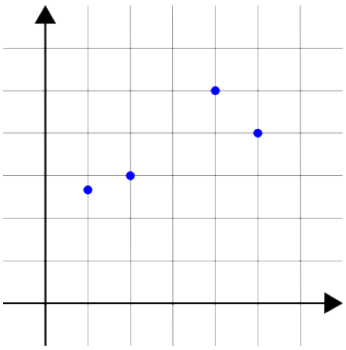
Lösung

In diesem Beispiel ist es einfach ersichtlich, welcher der Punkte verändert wurde: Da von den vier Punkten noch drei an ihrem ursprünglichen Ort sind, muss die Gerade genau drei Punkte durchlaufen. Die einzige Möglichkeit für diese Aufgabe ist eingezeichnet.
Ausserdem können die ursprünglichen Koordinaten des verschobenen Punktes ermittelt werden, da wir wissen, dass dieser nur vertikal verschoben worden ist.



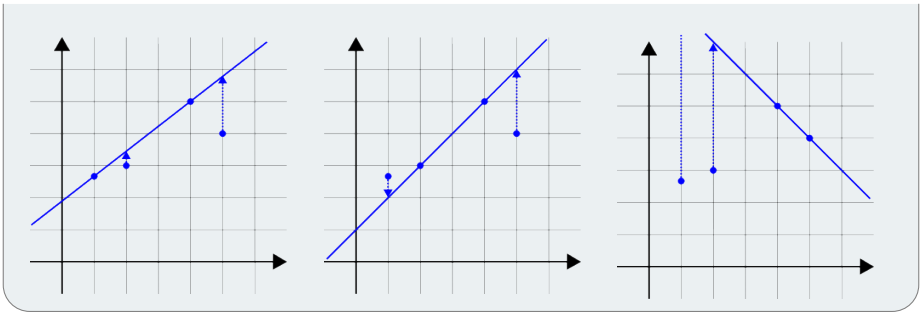
Aufgabe 2

Wir wollen uns nun die gleiche Situation anschauen und herausfinden, ob wir auch zwei Fehler korrigieren können. können. Dabei gehen wir davon aus, dass genau zwei Fehler geschehen sind, also genau zwei Punkte vertikal verschoben wurden. Kannst du diese zwei Punkte eindeutig bestimmen?
- Falls ja, markiere die zwei verschobenen Punkte und die ursprüngliche Gerade.
- Falls nein, zeichne die verschiedenen möglichen Lösungen ein. Können wir in diesem Fall den Fehler korrigieren?



Lösung

In diesem Fall gibt es sogar drei Geraden, die unsere Bedingungen erfüllen. Bei allen drei müssen wir genau zwei Punkte vertikal verschieben, um zum gegebenen Aufbau zu kommen. Da wir keine Möglichkeit haben das Original zu bestimmen, können wir die Fehler nicht korrigieren.



Bei vier Punkten haben wir also zu wenig Information, um zwei Fehler genau zu erkennen und auch zu korrigieren. Wir wollen nun herausfinden, ob sich dies verbessert, wenn wir mehr Punkte auf der Geraden haben.

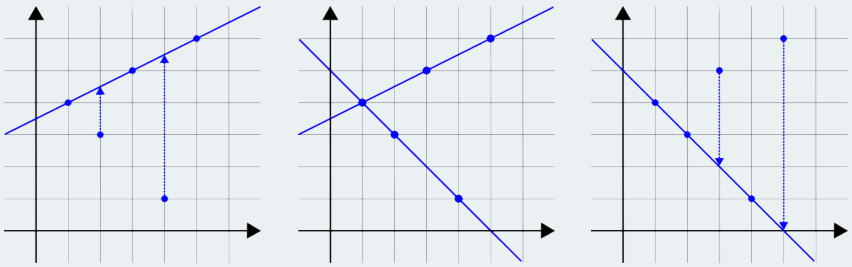
Aufgabe 3

Diskutiere die folgende Frage mit mit deiner Banknachbarin, mit deinem Banknachbarn: Wie sieht die Situation mit fünf Punkten auf einer Geraden aus? Können wir dann immer die originale Gerade bestimmen und somit zwei Fehler korrigieren? Wie ändert sich eure Antwort, wenn wir sechs Punkte betrachten?

- Wenn ihr der Meinung seid, dass dies immer möglich ist, versucht ein Argument zu formulieren und somit einen Beweis für die Aussage zu skizzieren.
- Wenn ihr der Meinung seid, dass dies nicht immer möglich ist, versucht eine Konfiguration zusammenzustellen, bei der zwei Fehler nicht eindeutig erkannt werden können.

Lösung

Wie in der Darstellung ersichtlich, ist das leider nicht immer möglich. Das liegt daran, dass bei fünf Punkten und zwei Fehlern genau drei Punkte auf der ursprünglichen Geraden liegen müssen. Den Widerspruch kann man also konstruieren, indem man zwei Geraden schneidet und am Schnittpunkt den dritten Punkt für beide Geraden hinzufügt:



Um maximal zwei Fehler in jedem Fall korrigieren zu können, sind also fünf Punkte noch nicht genug. Bei sechs Punkten haben wir das Problem von vorher nicht mehr, denn bei zwei Fehlern müssen immer noch vier Punkte auf einer Geraden liegen. Da zwei Punkte nicht gleichzeitig auf zwei verschiedenen Geraden liegen können, können wir immer die ursprüngliche Gerade bestimmen und so in jedem Fall zwei Fehler korrigieren.

Aufgabe 4

(a) Wenn wir zuverlässig immer drei Fehler korrigieren wollen, wie viele Punkte müssen auf der Geraden gegeben sein?

(b) Wie sieht das bei vier, fünf, sechs Fehlern aus? Kannst du eine allgemeine Formel für k Fehler herleiten?

Lösung

(a) Wir wissen, dass wir mind. sechs Punkte brauchen, um zwei Fehler in jedem Fall korrigieren zu können. Als nächstes überlegen wir uns, ob ein Punkt mehr, also sieben Punkte, genug sind, um drei Fehler zu korrigieren: Bei sieben Punkten und drei Fehlern wissen wir, dass genau vier Punkte auf der ursprünglichen Geraden liegen müssen. Dies reicht allerdings nicht, denn auch hier können wir zwei Geraden mit je drei Punkten schneiden, wobei der Schnittpunkt auf beiden Geraden liegt. In dem Fall haben beide Geraden 4 Punkte und wir können die ursprüngliche nicht bestimmen (siehe auch die Lösung von Aufgabe drei für eine visuelle Darstellung zu fünf Punkten).
Wenn also sieben Punkte nicht genug sind, können wir uns überlegen, ob acht Punkte genug sind. Bei acht Punkten und drei Fehlern müssen in jedem Fall fünf Punkte auf einer Geraden liegen. Da sich zwei verschiedene Geraden immer maximal einen Punkt teilen, kann es keine andere Gerade mit fünf Punkten geben, weswegen die ursprüngliche Gerade eindeutig ist.
Die Antwort ist also acht.

(b) Mit einem ganz ähnlichen Argument können wir uns auch anschauen, wie viele Punkte nötig sind, um k Fehler zu korrigieren:
Wir sehen, dass $2k+2$ Punkte genug sind, denn bei so vielen Punkten und k Fehlern müssen mindestens $k+2$ Punkte auf einer Geraden sein. Weil sich zwei verschiedene Geraden nicht mehr als einen Punkt teilen, kann es auch nur eine Gerade mit so vielen Punkten geben. Wir können also k Fehler bei $2k+2$ Punkten immer korrigieren.
Wir müssen uns jetzt allerdings noch überlegen, ob weniger Punkte nicht schon genug wären: Bei $2k+1$ Punkten würden $k+1$ Punkte auf der originalen Geraden bleiben. Allerdings könnte es sein, dass die k Fehler dazu führen, dass es noch eine andere Gerade mit $k+1$ Punkten gibt - diese Gerade würde die ursprüngliche Gerade in einem Punkt schneiden. In diesem Fall könnten wir nicht mehr bestimmen, welche Gerade der richtigen entspricht, also können wir die Fehler nicht korrigieren.
Für k Fehler brauchen wir also mindestens $2k+2$ Punkte; für vier Fehler brauchen wir 10 Punkte, für fünf brauchen wir 12 Punkte und für sechs Fehler brauchen wir 14 Punkte.

Wir wollen nun die Idee der linearen Funktionen dafür verwenden, um unsere Nachrichten zu kodieren und fehlerresistent zu gestalten. Wir wissen schon, dass sich eine Gerade durch zwei Parameter definieren lässt: m definiert die Steigung einer Geraden und b den Achsenabschnitt. Wollen wir also eine Nachricht kodieren, so können wir die Nachricht selber verwenden, um diese zwei Parameter zu definieren. Sobald wir eine definierte Gerade haben, müssen wir ausreichend Punkte auf der Geraden wählen, um die Anzahl Fehler, die wir erwarten, zu korrigieren. Die Punkte könnten wir frei wählen, müssen aber jeweils die x- und y-Werte kommunizieren, um den Punkt eindeutig zu definieren. Dieses Problem können wir umgehen, indem wir uns auf vorbestimmte Werte einigen; wir wählen also die Auswertung der linearen Funktion an den Punkten 0, 1, 2, ..., bis wir genug Punkte haben. Dann müssen wir nur die y-Werte der Punkte kommunizieren.
Schauen wir uns diese Idee an einem Beispiel an.

Beispiel 2

Sei die Nachricht 0110 gegeben. Eine mögliche Aufteilung dieser Nachricht wäre $m = 01_2 = 1$ und $b = 10_2 = 2$. In diesem Fall haben wir also die Nachricht in der Hälfte aufgeteilt: Die Steigung ist gegeben durch die binäre Zahl 01_2 und der Achsenabschnitt durch die binäre Zahl 10_2 - eine Gerade mit Steigung 1 und Achsenabschnitt 2.
Wir wollen die Nachricht resistent auf drei Fehler kodieren und brauchen dafür 8 Punkte (Auswertung der Funktion an den Punkten 0-7):

$$(0,2), (1,3), (2,4), (3,5), (4,6), (5,7), (6,8), (7,9)$$

Wir müssen allerdings nur die y-Koordinaten für die Kodierung nutzen, also wäre die Kodierung von 0110 die binäre Darstellung von

$$2 - 3 - 4 - 5 - 6 - 7 - 8 - 9$$

also:

$$10 - 11 - 100 - 101 - 110 - 111 - 1000 - 1001$$

Wir betrachten hier nicht, wie wir dieses Trennzeichen kodieren könnten, und belassen es so in der Nachricht drin. In der Praxis müsste auch dieses Zeichen kodiert werden – es liegt hier also eine mögliche Fehlerquelle vor.

Aufgabe 5

Verwende dieses Verfahren, um die Beispielnachricht 0110 1-, 2- bzw. 3-fehlerkorrigierend zu kodieren. Was fällt dir dabei auf?

Lösung

Wir haben schon im Beispiel 2 gesehen, dass die Nachricht 0110 einer Geraden mit Steigung 1 und Achsenabschnitt 2 entspricht.
Für eine 1-fehlerkorrigierende Kodierung brauchen wir vier Punkte : $(0, 2)$, $(1, 3), (2, 4), (3, 5)$. Wie im Beispiel ist die x-Koordinate nicht relevant, also haben wir die Kodierung $2 - 3 - 4 - 5$. Die binäre Darstellungsweise ist wie folgt: $10 - 11 - 100 - 101$

Für 2-fehlerkorrigierende Kodierungen sind sechs Punkte notwendig. Die zugehörige Kodierung ist $2 - 3 - 4 - 5 - 6 - 7$. Diese Kodierung sieht in binärer Darstellung wie folgt aus: $10 - 11 - 100 - 101 - 110 - 111$

Die 3-fehlerkorrigierende Kodierung von 0110 haben wir im Beispiel schon gesehen:

$$10 - 11 - 100 - 101 - 110 - 111 - 1000 - 1001$$

Wir merken schnell, dass die Kodierungen aufeinander aufbauen. Die 1-fehlerkorrigierende Kodierung widerspiegelt den ersten Teil der zweiten Kodierung, die zweite Kodierung wiederum das erste Stück der dritten Kodierung. Dies kann sehr nützlich sein, wenn wir eine Nachricht für das Auftreten mehrerer Fehler fehlerrobust machen wollen. Wir müssen dann nicht alle Punkte neu berechnen, sondern nur den neuen Teilbereich, den wir bei der kleineren Fehlerkorrektur nicht gebraucht haben.

Aufgabe 6

Was passiert, wenn wir m und b anders wählen, zum Beispiel indem wir $m = 0$ setzen und mit b die ganze Nachricht nutzen? Untersuche das Verhalten an der Nachricht 1011, indem du diese 2-fehlerkorrigierend kodierst.

Lösung

Eine Gerade mit Steigung $m = 0$ entspricht einer horizontalen Geraden auf der Höhe des Achsenabschnitts b. In diesem Fall entspricht b der Nachricht, also ist die y-Koordinate aller Punkte jeweils die Nachricht selber.

Wenn wir daher die Nachricht nicht aufteilen, sondern nur dem Achsenabschnitt zuschreiben, dann entspricht die kodierte Nachricht einfach einer Wiederholung der ursprünglichen Nachricht.

Für die Nachricht 1011 bedeutet das, dass $b = 1011_2 = 11$. Für eine 2-fehlerkorrigierende Nachricht brauchen wir sechs Punkte: $(0, 11), (1, 11), (2, 11), (3, 11), (4, 11), (5, 11)$. Wie bisher können wir die x-Koordinaten ignorieren und erhalten die Nachricht $11-11-11-11-11-11$; die binäre Darstellung ist wie folgt:

$$1011 - 1011 - 1011 - 1011 - 1011 - 1011$$

Vergleich: Reed-Solomon Codes und andere Kodierungen

Aufgabe 7

Kodiere die Nachricht 0110 1011 mit 1– und 2–fehlerkorrigierenden Kodierungen. Wende dabei die folgenden Strategien an:

1. Einfache Redundanz: Jedes Bit wird sooft wie nötig wiederholt.
2. Kartentrick: Die Konstruktion, die wir gesehen haben, erlaubt nur, einen Fehler zu korrigieren. Ihr könnt euch kurz überlegen, wie man dies für 2 Fehler erweitern könnte, was jedoch für diese Aufgabe optional ist.
3. Fehlermeldungstabelle: Auch hier haben wir die Konstruktion für eine 2-fehlerkorrigierende Kodierung nicht besprochen, also ist diese auch optional.
4. Reed-Solomon Kodierung: Verfahre so, wie wir es in diesem Kapitel besprochen haben.

Lösung

	1-FK	2-FK
Redundanz:	000 111 111 000 111 000 111 111	00000 11111 11111 00000 11111 00000 11111 11111
Kartentrick:	11011 01100 10111	-
FMT	0110 1011 1110	-
Reed-Solomon	$11 - 16 - 21 - 26$	$11 - 16 - 21 - 26 - 31 - 36$

Diskussion

Kommt zu Dreier- bzw. Vierergruppen zusammen und überlegt, was die Stärken und Schwächen der jeweiligen Kodierungen sind. Dabei könnt ihr Faktoren wie Einfachheit, Länge und Erweiterbarkeit mit in Betracht ziehen. Welche Kodierung würdet ihr bei Nachrichten verwenden, die sehr lang sind? Welche Kodierung verwendet ihr bei einem Nachrichtenaustausch mit vielen Fehlern?

Kodieren durch Wiederholen:

\+ −

Kartentrick:

\+ −

Fehlermeldungstabelle:

\+ −

Reed-Solomon Kodierung:

\+ −

Zusammenfassung

Wir haben gesehen, wie wir mittels linearer Funktionen Fehler finden und diese auch korrigieren können. Diese Vorgehensweise wird bei der Reed-Solomon Kodierung verwendet. Dabei werden aber normalerweise Polynome höheren Grades gebraucht. Die Kodierung erlaubt nicht nur die Korrektur von fehlerhaften Informationen, sondern auch die Wiederherstellung von fehlenden Informationen. Reed-Solomon Codes wurden das erste Mal 1977 von der NASA für das Voyager-Programm eingesetzt. Dank ihnen konnten Messungen und Bilder der Planeten Jupiter und Saturn die Erde erreichen und dekodiert werden.

Erweiterung des Kartentricks

Wir wollen nun genauer auf den Kartentrick eingehen und sehen, ob und wie wir diesen erweitern können, um mehrere Fehler korrigieren zu können. In einem ersten Schritt möchten wir für das Erreichen dieses Zieles bei den Lernenden ein Gespür dafür aufbauen, wieso wir mittels Kartentrick immer einen Fehler korrigieren können. Um dies zu verstehen, repetieren wir das Konzept des Hamming-Abstands und die Bedingungen zur Fehlerkorrektur.

Aufgabe 8

Welchen Hamming-Abstand haben folgende Wörter?

(a) 01111 und 11111

(b) 101010 und 010101

(c) 1001 1011 und 0111 1001

(d) 0010, 1110 und 1001

Lösung

(a) 01111 und 11111 haben einen Abstand von 1, da sie sich nur an erster Stelle unterscheiden.

(b) 101010 und 010101 haben einen Abstand von 6, da sie sich an allen Stellen unterscheiden.

(c) 1001 1011 und 0111 1001 haben einen Abstand von 4 mit Unterschieden an den Stellen 1,2,3 und 7.

(d) 0010 und 1110 haben einen Abstand von 2; 0010 und 1001 haben auch einen Abstand von 2; 1110 und 1001 haben einen Abstand von 3. Insgesamt ist also der Abstand der Kodierung der drei Wörter 2.

Wenn wir uns die Codewörter aus der letzten Kodierung anschauen, dann haben wir gesehen, dass diese einen Abstand von 2 haben. Intuitiv bedeutet das, dass wir mindestens an zwei Stellen ein Bit ändern müssen, um ein anderes Wort zu erhalten. Dies gibt uns sehr viel Information, denn wir können daraus ableiten, dass wir kein anderes Wort mit nur einer Bit-Änderung erreichen. Das ist auch der Grund, warum wir bei Abstand 2 einen Fehler immer erkennen können.

Ganz ähnlich können wir uns die allgemeine Situation überlegen: Wenn wir einen Abstand von k haben, dann bedeutet es, wir müssen die Bits an mindestens k Stellen ändern, um ein anderes Wort zu erhalten. Das bedeutet auch, dass wir mit nur $k-1$ Änderungen niemals ein anderes Wort bekommen können. Deswegen werden in diesem Fall bis zu $k-1$ Fehler immer erkannt und wir würden sagen, dass diese Kodierung $k-1$-fehlererkennend ist.

Wir wollen jetzt dasselbe Gespür für die Fehlerkorrektur aufbauen. Wenn wir uns die zwei Wörter 01 und 10 anschauen, dann haben diese einen Abstand von zwei. Ein Fehler wird dabei immer erkannt: Wir wissen zum Beispiel, dass 11 in diesem Fall nicht Teil der Sprache war. Aber können wir auch das ursprüngliche Wort bestimmen? Nein, denn wir wissen nicht, wo der Fehler geschehen ist, wir wissen nur, dass ein Fehler geschehen ist.

Aufgabe 9

Begründe zuerst mithilfe einer spezifischen Sprache und danach allgemein, wieso wir immer einen Fehler korrigieren können, wenn die Sprache einen Abstand von 3 hat.

Lösung

Schauen wir uns die Sprache 000 und 111 an (diese Sprache hat einen Abstand von 3). Wenn wir das Wort 000 versenden und dabei ein Fehler passiert, dann können die entstandenen Wörter 001, 010 oder 100 sein. Auf der anderen Seite entstehen 110, 101 oder 011, wenn wir mit dem Wort 111 anfangen und ein Fehler passiert. Wir sehen, dass sich diese Wörter nicht überschneiden, und können somit immer einen Fehler erkennen und auch korrigieren.

Allgemein können wir das gleiche Argument verwenden. Wenn eine Sprache einen Abstand von 3 hat, dann sind die Mengen der möglichen fehlerhaften Wörter disjunkt, weswegen wir jedes "korrumpierte" Wort eindeutig einem Wort aus der Sprache zuordnen können.

Aufgabe 10

Wie sieht es bei mehreren Fehlern aus? Wie gross muss der Abstand mindestens sein, sodass wir immer k Fehler korrigieren können?

Lösung

Wir wissen, dass wir nur dann k Fehler korrigieren können, wenn die Mengen der möglichen fehlerhaften Wörter disjunkt sind. Damit dies der Fall ist, muss zwischen beliebigen zwei fehlerhaften Wörtern der Hamming-Abstand mindestens 1 sein. Somit muss der Hamming-Abstand zweier beliebiger Wörter der Sprache mindestens $2k + 1$ sein.

Ein Abstand von drei ist also immer genug, um einen Fehler korrigieren zu können; und ein Abstand von $2k + 1$ ist ausreichend um immer k Fehler korrigieren zu können. Im letzten Kapitel haben wir gesehen, dass wir mit dem Kartentrick immer eine Kodierung erhalten, die 1-fehlerkorrigierend ist. Wir wollen nun verstehen, wieso dies so ist.

Aufgabe 11

(a) Formuliere zuerst die Bedingung, die die Kodierung erfüllen muss, um einen Fehler korrigieren zu können.

(b) Versuche als Nächstes zu argumentieren, wieso diese Bedingung bei dem Kartentrick erreicht wird.

Lösung

(a) Wir haben vorher gesehen, dass wir immer einen Fehler korrigieren können, wenn der Abstand der Sprache mindestens 3 ist.

(b) Wenn wir eine beliebige Stelle eines Wortes ändern, dann sehen wir, dass sich nicht nur das Bit ändert, sondern auch die Kontrollbits der zusätzlichen Spalte und Zeile. Wir sehen auch, dass ein beliebiges weiteres Bit ändern können und die Invariante vom Abstand 3 beibehalten ist.

1	1	0	0
1	0	0	1
0	1	0	1

Aufgabe 12

Eine Schülerin behauptet, dass die Kartentrick-Kodierung sogar zu einem Abstand von vier führt. Ist diese Aussage wahr oder falsch?

- Falls sie wahr ist, versuche sie zu begründen.
- Falls sie falsch ist, finde ein Gegenbeispiel.

Lösung

Diese Aussage ist korrekt, und zwar dank des Bits in der oberen rechten Ecke. Wir können uns nochmals etwas genauer überlegen, was passiert, wenn wir ein Bit im Wort ändern: Das Bit selbst ändert sich, sowie die Bits der zugehörigen Zeile und Spalte, und das Bit in der oberen rechten Ecke. Wenn wir ein weiteres Bit in derselben Zeile ändern, dann setzen wir das Bit in der Zeile und das Bit in der oberen rechten Ecke zurück, allerdings bleiben die zwei Kontrollbits der Spalte gewechselt, also behalten wir einen Abstand von vier.

Die Kartentrick-Kodierung liefert also sogar einen Abstand von vier. Das ist genug, um bis zu drei Fehler zu erkennen, aber noch nicht genug, um zwei Fehler zu korrigieren. Tatsächlich gibt es Aufstellungen, in denen zwei Fehler erkannt, aber nicht korrigiert werden können.

Aufgabe 13

(a) Ändere in diesem Beispiel zwei Bits, ohne dass es eindeutig ist, welche zwei Bits geändert wurden.

(b) Gibt es Konfigurationen, in denen wir zwei Bits ändern könnten und immer noch genau sagen können, wo der Fehler passiert ist?

1	1	0	0
1	0	0	1
0	1	0	1

Lösung

(a) Wir können beliebige zwei Bits im ursprünglichen Wort ändern (schwarzer Bereich) und werden nie in der Lage sein, die zwei gewählten Bits eindeutig zu bestimmen.

(b) Wenn wir ein beliebiges Bit und das Bit in der oberen rechten Ecke ändern, dann können wir den Kartentrick immer noch so verwenden wie bisher. Dies zeigt uns auch, dass wir das obere rechte Bit eigentlich gar nicht brauchen, sondern nur der Vollständigkeit halber eingeführt haben.

Diskussion

Leider können wir den Kartentrick in der Art, die wir gelernt haben, nicht nutzen, um zwei Fehler korrigieren zu können. Wie könnte es funktionieren? Versuche dir mit deiner Banknachbarin oder deinem Banknachbarn Erweiterungen des Kartentricks zu überlegen, die es ermöglichen würden, zwei Fehler korrigieren zu können.

Aufgabe 14

Eine vorherige Klasse hat dieselbe Aufgabe bekommen und ist dabei sehr kreativ geworden. Leider haben nicht alle Gruppen eine korrekte Erweiterung gefunden, und deine Aufgabe ist es jetzt, die richtige von der falschen Erweiterung zu unterscheiden.

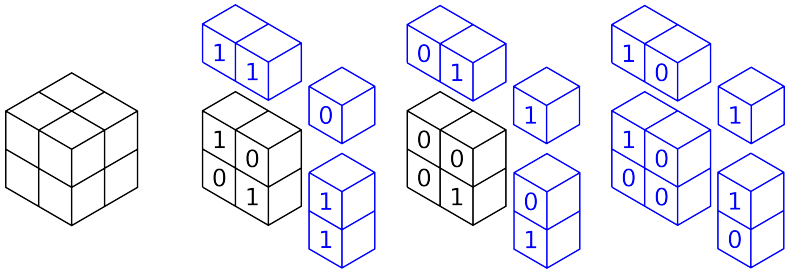
(1) James behauptet, dass wir weitere Paritätsbits unten und links brauchen können, um mehr Abstand zu bekommen.

1	1	0	0
1	0	0	1
0	1	0	1

$\Rightarrow$

	1	1	0	0
1	1	0	0	1
1	0	1	0	1
0	1	1	0	

(2) Andrea denkt sich, dass die Idee des Hyperwürfels sich hier gut ausnutzen lässt, sie erweitert das normale Feld zu einem Würfel.



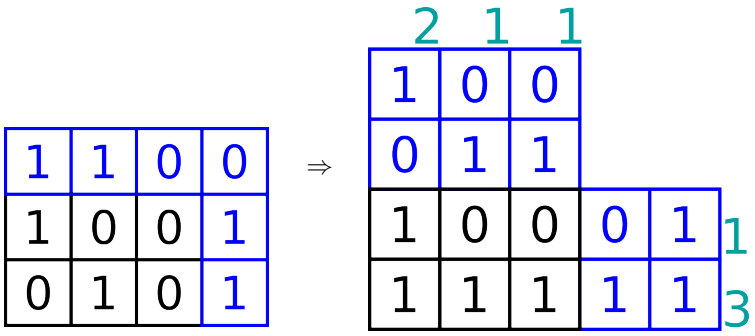
(3) Sanja hat das Gefühl, dass es ausreicht, den Kartentrick einfach zweimal nacheinander anzuwenden.

1	1	0	0
1	0	0	1
0	1	0	1

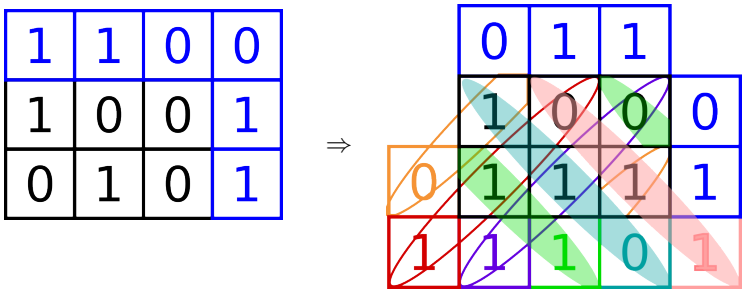
$\Rightarrow$

0	0	0	0	0
1	1	0	0	0
1	0	0	1	0
0	1	0	1	0

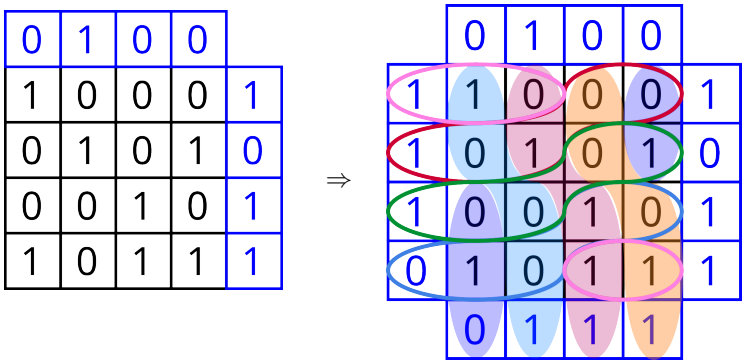
(4) Marvin mag die Idee von Sanja, befürchtet aber, dass mod 2 hier nicht genug ist. Er verwendet stattdessen mod 4.



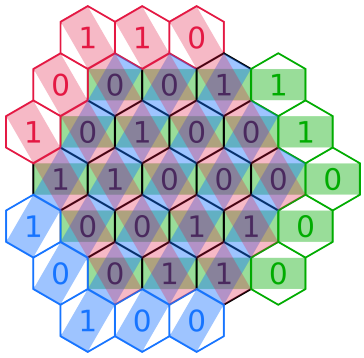
(5) Sandra behauptet auch, dass man weitere Paritätsbits benötigt. Sie berechnet weitere Bits in der Diagonalen.



(6) Oliver gefällt die Idee von Sandra, möchte aber statt Diagonalen Treppenstufen verwenden.



(7) Leonardo hat das Gefühl, man könnte zwei Fehler korrigieren, indem man die Form der Karten ändert.



Lösung

(1) Das würde nicht funktionieren, denn die zusätzlichen Bits führen nicht zu grösserem Abstand.

(2) Dies stellt eine korrekte Erweiterung dar.

(3) Diese Erweiterung funktioniert nicht, denn die zusätzliche Spalte wird immer nur 0 haben.

(4) Auch diese Erweiterung funktioniert nicht, denn zwei diagonale Bits könnten nicht eindeutig korrigiert werden.

(5) Dies stellt eine korrekte Erweiterung dar.

(6) Dies stellt auch eine korrekte Erweiterung dar.

(7) Dies würde nicht funktionieren, denn wir erreichen nicht einen Abstand von fünf.

Diskussion

Wir haben jetzt gesehen, dass wir den Kartentrick auf verschiedene Weisen korrekt erweitern können! Dabei sind einige Ideen besser umsetzbar als andere. Diskutiere mit deiner Banknachbarin oder deinem Banknachbarn:

- Welche dieser Vorgehensweisen lassen sich einfach visualisieren?
- Welche dieser Vorgehensweisen sind einfach zu implementieren?
- Welche dieser Vorgehensweisen lassen sich einfach auf die Korrektur von drei oder mehr Fehler erweitern?

Zusammenfassung

Wenn wir uns den Kartentrick mit den Diagonalen anschauen, dann sehen wir, dass wir immer einen Abstand von 5 erhalten. Wir haben auch gesehen, dass wir nicht einfach die Paritätsbits wiederholen können, denn in dem Fall kann es sein, dass wir zwar bemerken, dass irgendwo ein/zwei Fehler vorgefallen sind, aber wir können nicht eindeutig bestimmen, wo diese Fehler passiert sind.

Zusammenfassung Kapitel

In diesem Kapitel haben wir Vorgehensweisen diskutiert, wie wir selbstkorrigierende Kodierungen gegen mehrere Fehler resistent machen können.

Als Erstes haben wir unsere Aufmerksamkeit den Reed-Solomon Codes gewidmet. In einer stark vereinfachten Version davon haben wir gesehen, wie eine lineare Funktion einen Fehler detektieren und korrigieren kann. Daraufhin haben wir dasselbe Prinzip gebraucht, um die Korrektur mehrerer Fehler zu ermöglichen.

Wir haben dann die einfache Redundanz, die Kartentrick-Kodierung und die Fehlermeldungstabelle den Reed-Solomon Codes gegenübergestellt. In einer gestellten Aufgabe ging es dann darum, die Mechanismen zur n-Fehlerkorrektur zu kontrastieren, indem man die Kodierungen auf konkrete Beispiele anwendet. Aber dabei war nicht offensichtlich, wie man den Kartentrick in diesem Fall genau brauchen konnte.

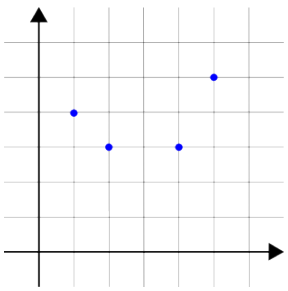
Aus diesem Grund sind wir letztendlich auf die Erweiterung der Kartentrick-Kodierung eingegangen. Hier haben wir verschiedene Ansätze studiert, die mehrere Fehler zum Teil erfolgreich korrigieren konnten, zum Teil auch nicht. Zusätzlich half die Darstellung der Lösungsansätze, um sie anschaulicher zu machen.

Doch sind unsere Kodierungen optimal? Um diese Frage zu beantworten, haben wir uns der Theorie des Begründers der Informationstheorie Claude Shannon bedient: Dieser formulierte 1948 ein Kodiertheorem, das eine Obergrenze bei der Nachrichtenübertragung über einen Kanal in geräuschvoller Umgebung definiert. Seit der Erfindung von Reed-Solomon Codes (1960) wurden zahlreiche neue Kodierungen entworfen, die sich dieser Obergrenze stetig annähern.

Kontrollfragen

Aufgabe 1

Kannst Du ermitteln, wie viele Fehler mindestens bei der Übertragung der folgenden Punkte geschehen sind?



- Überlege dir die möglichen Szenarien, die zu diesen Fehlern geführt haben könnten, und zeichne sie ein.
- Verwende dabei verschiedene Farben für die jeweilige unterschiedliche Anzahl Fehler, die bei der Übermittlung aufgekommen ist.

Aufgabe 2

Verwende das gelernte Verfahren der Reed-Solomon Kodierung, um die Beispielnachricht 0111 1000

- 1-fehlerkorrigierend
- 2-fehlerkorrigierend
- 3-fehlerkorrigierend

zu kodieren.

Aufgabe 3

Welchen Hamming-Abstand haben folgende Sprachen?

- 10 und 01
- 10000 und 01000
- 1010 1011 und 0111 0011
- 1100, 0101 und 10011
- 1101 1001, 0110 0111, 0000 1011 und 1001 0011

Aufgabe 4

Wähle eine der Varianten des Kartentricks.

(a) Beschreibe zuerst die Variante genau. Wie kannst du diese anwenden, um 2 und 3 Fehler korrigieren zu können? Wieso funktioniert die gewählte Variante?

(b) Kodiere die Nachricht 1001 0101 1101 1101 (du kannst die Nachricht verlängern oder leicht kürzen, wenn eine andere Nachrichtenlänge besser passt):

- ✦ 2-fehlerkorrigierend
- ✦ 3-fehlerkorrigierend

Aufgabe 5

Wähle eine der falschen Varianten des Kartentricks und beschreibe eine Änderung, die dazu führen würde, dass die Variante korrekt benutzt werden könnte. Ändere die Variante nicht umfassend, sondern überlege dir, ob kleine Änderungen schon dazu führen können, ein Verfahren zu korrigieren.