Kantonsschule Im Lee

Informatik

Datenintegrität

Skript

Cyril Wendl

Inhaltsverzeichnis

1 Datenintegrität **2**
1.1 Einführung . . . 2
1.2 Prüfbits . . . 2
1.3 Fehler erkennen und korrigieren . . . 5
1.4 Selbstkorrigierende Codes . . . 9
1.5 Effiziente Codes . . . 13
1.6 Der XOR-Operator . . . 16
1.7 RAID . . . 17
1.7.1 RAID 0: Striping . . . 17
1.7.2 RAID 1: Mirroring . . . 18
1.7.3 RAID 4: Paritätsbits . . . 19
1.7.4 RAID 5: Verteilte Paritätsbits . . . 19
1.7.5 RAID 6: Zwei Paritätsbits . . . 21

A Lernziele **23**

Kapitel 1

Datenintegrität

1.1 Einführung

Stellen Sie sich vor, Sie feiern Ihren 18. Geburtstag und wollen ihren besten Freunden einige Fotos aus ihrer Kindheit zeigen. Die Fotos sind aber leider alle verpixelt und können nicht richtig angezeigt werden! Oder stellen Sie sich vor, sie wollen in einigen Jahren Bilder von ihrem Date anschauen, die Bilder auf ihrer Festplatte sind jedoch unleserlich. *Bad luck*? Fast genau dies ist dem Profi-Fotografen Tony Northrup geschehen, als er den 18. Geburtstag seiner Tochter mit Fotos aus deren Kindheit feiern wollte: Video.

Bei der Übertragung von Bits über das Internet können einzelne Bits spontan verändert oder gelöscht werden – dies kann fatale Folgen haben! Bei spontaner Änderung von einem 1 zu einem 0 (oder umgekehrt) spricht man von sogenanntem *bit rot*.

Dies ist ein klassisches Beispiel für einen Fehler, der bei der Übertragung von Daten auftreten kann. Mit dem Begriff „Datenintegrität“ bezeichnen wir die Korrektheit und Vollständigkeit von Daten. Datenintegrität ist ein wichtiges Thema in der Informatik, da es darum geht, sicherzustellen, dass Daten während ihrer Speicherung und Übertragung nicht verändert oder beschädigt werden.

1.2 Prüfbits

Buchstaben können auf unterschiedliche Weise kodiert sein – bei American Standard Code for Information Interchange (ASCII) werden beispielsweise sieben Bits pro Zeichen verwendet:

Buchstabe	Kodierung
A	`010000001`
B	`010000010`
C	`010000011`
...	...

Bei der Datenübertragung können aber einzelne Bits spontan von einem `1` zu einem `0` werden (oder umgekehrt). Was könnte getan werden, um dieser Gefahr vorzubeugen?

Ein erster, einfacher Ansatz bestünde darin, jedem Code noch ein weiteres Bit anzuhängen, so dass die Summe der Bits immer gerade ist:

Buchstabe	Kodierung
A	0100000010
B	0100000100
C	0100000111
...	...

Wenn sich *ein* Bit nun spontan verändert, wird der Code als fehlerhaft erkannt, da die Summe der Bits, welche den Wert `1` haben, nicht mehr gerade ist. Ein Computer könnte nun also anfordern, dass der Code nochmals geschickt wird.

Die Idee von Prüfziffern ist, dass jede Prüfziffer nach dem gleichen Prinzip berechnet ist. Stimmt bei einem erhaltenen Code die Formel mit der Prüfziffer nicht überein, wird der Code als fehlerhaft erkannt.

Aufgabe 1.1

Folgende Prüfziffern sind alle nach der gleichen Regel berechnet. Können Sie die Regel erkennen?

- 31**6**
- 1298**0**
- 3333574**2**
- 2**8**
- 10**9**

Was wäre die Kontrollziffer für die Zahl 4163?

Mit einem European Article Number (EAN) wird ein Standard für die Kodierung von Waren entwickelt. Fast alle käuflich erhältlichen Waren auf dem Markt enthalten einen solchen Code, welcher aus 13 Ziffern besteht. Häufig wird er in Form eines Strichcodes dargestellt, wobei zwei Striche eine Ziffer darstellen, sowie 2 mal 2 Striche, die den Rand angeben (siehe Abbildung 1.1).

Abbildung 1.1: Bestandteile eines EAN

Ein EAN besteht aus 12 Ziffern und einer Prüfziffer:

$$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13} \tag{1.1}$$

Das Scannen eines falschen Codes könnte zu falschen Preisen an der Kasse führen, weshalb EAN ebenfalls eine Prüfziffer enthalten, die sich wie folgt berechnet:

1. $s = x_1 + x_3 + x_5 + x_7 + x_9 + x_{11} + 3 \times (x_2 + x_4 + x_6 + x_8 + x_{10} + x_{12})$
2. Anschliessend wird x_{13} so gewählt, dass $s + x_{13}$ durch 10 teilbar ist.

Aufgabe 1.2

Welche folgender Fehler werden durch einen EAN erkannt?

- Vertippen (eine Ziffer ist falsch)
- Vertauschen zweier benachbarter Ziffern
- Auslassen (eine Ziffer wird vergessen)
- 2 mal Vertippen → 2 Ziffern werden falsch getippt

Aufgabe 1.3

Berechnen Sie für 978314221213 die EAN-13-Prüfziffer.

Aufgabe 1.4

Bestimmen Sie für den folgenden EAN-Code alle Paare von benachbarten Ziffern, deren Austausch in der Reihenfolge nicht erkannt werden kann:

```
4030538657465
  3  6  9  12
```

Aufgabe (Challenge) 1.5

Begründen Sie allgemein, für welche Arten von Ziffern ein Vertauschen zweier benachbarten Ziffern in einer EAN-Prüfziffer nicht entdeckt wird.

Aufgabe 1.6

Nehmen wir an, dass an der vierten Position im Code aus Aufgabe 1.4 eine 8 statt der 0 getippt wurde. Finden Sie mindestens 5 Möglichkeiten, an anderen Stellen den Fehler so zu machen, dass die fehlerhafte Darstellung einem EAN-Code entspricht und somit dieser Doppelfehler unbemerkt bleibt.

Aufgabe 1.7

Die ursprünglichen International Standard Book Number (ISBN)-10-Codes zur eindeutigen Bezeichnung von Büchern bestanden aus 9 Ziffern und einem Prüfsymbol aus den folgenden Ziffern:

`{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, X}`

wobei `X` die Zahl 10 repräsentierte. Für die neun Ziffern x_1, x_2, $\cdots$, x_9, hat man das Prüfsymbol x_{10} wie folgt bestimmt:

$$10 \times x_1 + 9 \times x_2 + 8 \times x_3 \cdots + 2 \times x_9 + x_{10}$$

ist ein Vielfaches von 11. Falls $x_{10} = 10$ ist das Prüfsymbol `X`.

1. Bestimmen Sie für die folgenden Symbolfolgen, welche ISBN-10-Codes sind und welche es nicht sind:
   - `0201441241`
   - `270004302X`
   - `8090040489`
   - `010144124X`
2. Die neunstellige Zahl eines Buches ist `354042278`. Bestimmen Sie das Prüfsymbol des ISBN-10-Codes des Buches.

1.3 Fehler erkennen und korrigieren

Sie kennen Fehlererkennung bereits aus Ihrem Alltag: So beispielsweise bei der Rechtschreibprüfung in Word, die Ihnen das korrekte Wort vorschlägt (siehe Abbildung 1.2)

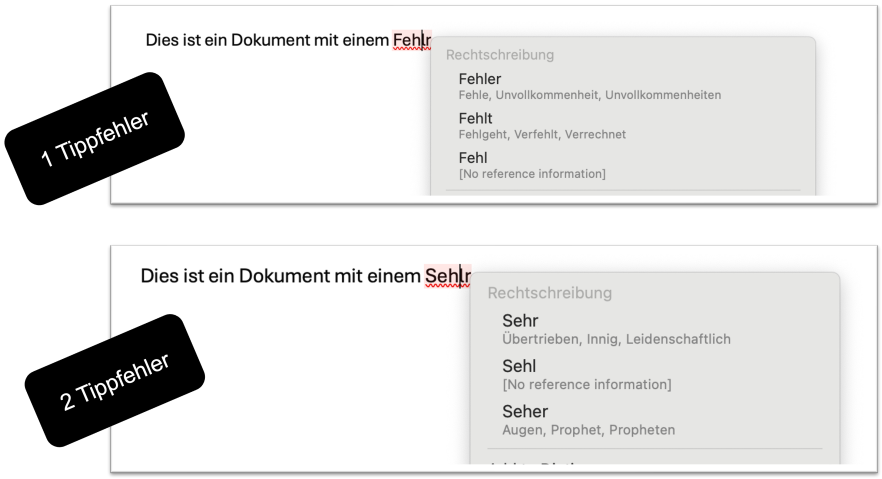

Abbildung 1.2: Rechtschreibprüfung im Programm Microsoft Word

Beispiel 1.1:
Wie viele Tippfehler werden im Beispiel aus Abbildung 1.2 in jedem Fall erkannt?

Im gezeigten Beispiel wird ein Tippfehler noch erkannt. Es wird jedoch nicht jeder Tippfehler garantiert erkannt: So könnte beispielsweise ein Tippfehler vom ursprünglich gemeinten Wort „Fehler“ hin zu „Zehler“ nicht erkannt werden, da „Zehler“ sowohl einen Buchstaben Abstand zum (korrekten, beabsichtigten) Wort „Fehler“ wie auch zum (nicht beabsichtigten, aber korrekten) Wort „Zähler“ hat.

Was können wir aus Beispiel 1.1 lernen? Damit ein Fehler automatisch korrigiert werden kann, muss der Unterschied zwischen dem vertippten Wort und allen anderen möglichen Wörtern grösser sein als der Abstand zum ursprünglichen Wort.

Im Folgenden unterscheiden wir zwischen **k-fehlererkennenden** sowie **k-fehlerkorrigierenden** Kodierungen. Eine Kodierung heisst **k-fehlererkennend**, wenn das Umflippen von 1, 2, $\cdots$ bis und mit **k** Bits in der Nachricht als Fehler erkannt wird. Falls bis zu **k** Fehler nicht nur erkannt sondern auch behoben werden können, spricht man von **k-fehlerkorrigierenden Kodierungen**.

Beispiel 1.2:
Tabelle 1.1 zeigt, wie eine Nachricht so kodiert werden kann, dass Fehler erkannt werden:

- In Kodierung 1 handelt es sich schlicht und einfach um die ursprüngliche Nachricht.
- In Kodierung 2 wird die ursprüngliche Nachricht verdreifacht.
- In Kodierung 3 wird die ursprüngliche Nachricht verdoppelt, dazu kommt ein **Paritätsbit** (unterstrichen) am Ende der Nachricht.

	Kodierung$_1$	**Kodierung$_2$**	**Kodierung$_3$**
Weiss	00	000000	00000
Gelb	01	010101	01011
Blau	10	101010	10101
Schwarz	11	111111	11110

Tabelle 1.1: Kodierungstabelle

Wie viele Fehler werden durch die unterschiedlichen Kodierungen bestimmt erkannt?

- In der ursprünglichen Kodierungen können Fehler nicht erkannt werden. Wird beispielsweise Weiss (`00`) fälschlicherweise als `01` versandt, geht der Empfänger davon aus, dass Gelb gesandt wurde und erkennt keine Fehler.
- In der zweiten Kodierung werden zwei Fehler erkannt, da es zwei „Mutationen“ benötigt, um zu einem anderen Code zu werden.
- Bei der dritten Kodierung verhält es sich ebenso wie bei der zweiten.

Kodierung$_2$ und Kodierung$_3$ sind also betreffend Fehlererkennung gleich gut, allerdings ist Kodierung$_3$ kürzer und daher zu bevorzugen.

Um zu bestimmen, wie viele Fehler eine bestimmte Kodierung erkennen kann, müssen wir uns die Abstände zwischen allen möglichen Code-Wörtern anschauen.

Beispiel 1.3:
Angenommen, wir hätten drei Code-Wörter, `ANNA`, `ENZO` und `HANS`, dann könnten die Abstände zwischen allen Code-Wörtern wie folgt bestimmt werden:

A N N A	A N N A	E N Z O
E N Z O	H A N S	H A N S
x o x x	x x o x	x x x x
Abstand 3	Abstand 3	Abstand 4

Tabelle 1.2: Abstände zwischen den Codewörtern `ANNA`, `ENZO` und `HANS`

Die „`x`“ bezeichnen Stellen, an denen sich die Buchstaben unterscheiden und die „`o`“ Stellen, an denen die Buchstaben gleich sind. Der Abstand zwischen einem Wort-Paar ergibt sich aus der Summe der „`x`“.

Der minimale Abstand in diesem Beispiel ist 3, daher könnten in dieser Kodierung bis zu 2 Fehler, bzw. „Mutationen“ erkannt werden. `ANNA` → `ANZA` würde als Fehler erkannt, `ANNA` → `ENZO` jedoch nicht, da `ENZO` ein korrektes Code-Wort ist.

Binär funktioniert die Abstandsbestimmung genau gleich wie in Beispiel 1.3.

Beispiel 1.4:
Basierend auf Kodierung$_2$ aus Tabelle 1.1 könnten die Abstände zwischen allen Code-Wörtern wie folgt bestimmt werden:

```
0 0 0 0 0 0
0 1 0 1 0 1
o x o x o x
```
Abstand 3

```
0 0 0 0 0 0
1 0 1 0 1 0
x o x o x o
```
Abstand 3

```
0 0 0 0 0 0
1 1 1 1 1 1
x x x x x x
```
Abstand 6

```
0 1 0 1 0 1
1 0 1 0 1 0
x x x x x x
```
Abstand 6

```
0 1 0 1 0 1
1 1 1 1 1 1
x o x o x o
```
Abstand 3

```
1 0 1 0 1 0
1 1 1 1 1 1
o x o x o x
```
Abstand 3

Tabelle 1.3: Abstände zwischen allen Codewörtern der Kodierung$_2$ aus Tabelle 1.1

Damit können auch in diesem Beispiel bis zu 2 Fehler erkannt werden.

Aufgabe 1.8

Xavier schlägt vor, die Code-Wörter für die Farben aus Beispiel 1.2 anders zu konstruieren. Man soll zu den Darstellungen (zwei Bits, z.B. 11) der Farben das Prüfbit anhängen (z.B. 11$\underline{0}$), sodass die Anzahl der Einsen gerade wird. Danach sollte man diese drei Bits in zwei Kopien (z.B. 11$\underline{0}$11$\underline{0}$ für „Schwarz“) zu einem Code-Wort machen. Konstruieren Sie alle Code-Wörter der Kodierung und bestimmen Sie, wie viele Fehler man damit erkennen kann.

Aufgabe 1.9

Bestimmen Sie, wie viele Fehler jede der folgenden Kodierungen erkennen kann. Bestimmen sie dazu als erstes den Mindest-Abstand innerhalb jeder Kodierung:

	4-Kopien	**Xavier**	**1-Parity**	**0-Parity**
Weiss	00000000	000000	000	001
Gelb	01010101	011011	011	010
Blau	10101010	101101	101	100
Schwarz	11111111	110110	110	111

Tabelle 1.5: Kodierungstabelle

Aufgabe (Challenge) 1.10

Finden Sie eine binäre Kodierung mit 3 Code-Wörtern der Länge 5 und Mindest-Abstand 3.

Aufgabe (Challenge) 1.11

Gibt es eine binäre Kodierung mit 3 Code-Wörtern der Länge 5 und Mindest-Abstand 3, welche die Code-Wörter 00000 und 11111 beinhaltet?

1.4 Selbstkorrigierende Codes

Bisher haben wir gelernt, Kodierungen für gewisse Nachrichten zu finden, so dass ein gewisser Mindestabstand gewährleistet wird. Durch einen Mindestabstand von 2 (Bits) zwischen jedem Nachrichtenpaar wird gewährleistet, dass ein Fehler bestimmt erkannt wird, da es zwei Fehler (ungewollte Bit-Änderungen) bräuchte, damit aus einem gültigen Codewort ein anderes, gültiges Codewort wird.

Falls wir einen Mindestabstand von 3 haben, können wir einen Fehler sogar *korrigieren*: Bei einer einzigen Änderung eines Bits wäre das resultierende Codewort dann immer noch näher an am ursprünglichen Codewort als an allen anderen Codewörtern.

Aufgabe 1.12

Finden Sie für die drei Nachrichten Rot, Blau und Grün eine Kodierung aus Bitfolgen der Länge 5, die 1-fehlerkorrigierend ist.

Das bedeutet, wenn ein Fehler vorkommt, weiss man nicht nur, dass die Nachricht fehlerhaft ist, sondern man kann das ursprüngliche Codewort aus der fehlerbehafteten Bitfolge eindeutig rekonstruieren. Was können Sie über den Abstand einer solchen Kodierung sagen?

Farbe	**Kodierung**
Rot	00000
Grün	
Blau	

Aufgabe (Challenge) 1.13

Könnte man Aufgabe 1.12 auch mit nur 4 Bits lösen?

In Challenge 1.13 haben wir versucht, eine binäre Kodierung mit 4 Bits für 3 Nachrichten zu finden. Um die Aufgabe einfacher zu lösen, können wir uns ein Hilfsmittel zunutze machen. Dabei zeichnen wir alle möglichen Codes der Länge 4 auf und verbinden diejenigen Codes, welche Abstand 1 haben, miteinander (siehe Abbildung 1.5). Wenn wir nun einen Fehler korrigieren wollen, brauchen wir mindestens einen Abstand von 3 zwischen jedem benachbarten Bitpaar.

Definition 1.1:
Um die Anzahl möglicher 1-fehlerkorrigierender Codewörter für eine gewisse Code-Länge herauszufinden, gehen wir wie folgt vor:

1. Wähle irgendein Codewort als erstes Codewort aus, z.B., 0000.
2. Streiche alle Code-Wörter, welche von allen bisher ausgewählten Codewörtern weniger als Abstand 3 entfernt sind, also alle Codewörter mit Abstand 1 oder 2.
3. Wiederhole Schritte 1 und 2, bis alle Codewörter entweder ausgewählt oder durchge-

strichen sind.

Auf diese Weise erhalten wir eine maximale Anzahl an Codewörtern, die 1-fehlerkorrigierend sind. Diese Methode funktioniert auch für andere Code-Längen, wie z.B. 3 (Abbildung 1.4) oder 5 (Abbildung 1.6).

Soll der Code nur 1-fehlererkennend und nicht 1-fehlerkorrigierend sein, reicht es aus, in Schritt 2 alle Codewörter mit Abstand 1 zu streichen.

Dieses Vorgehen funktioniert auch für andere Code-Längen, beispielsweise 2 (Abbildung 1.3), 3 (Abbildung 1.4) oder 5 (Abbildung 1.6).

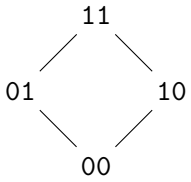

Abbildung 1.3: 2D-Hyperwürfel

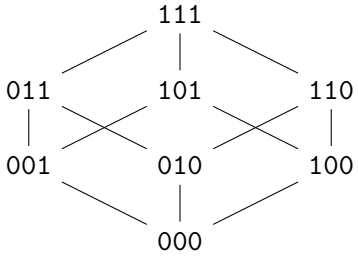

Abbildung 1.4: 3D-Hyperwürfel

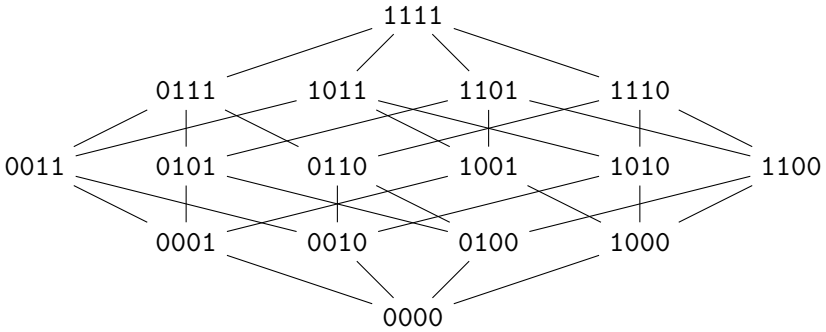

Abbildung 1.5: 4D-Hyperwürfel

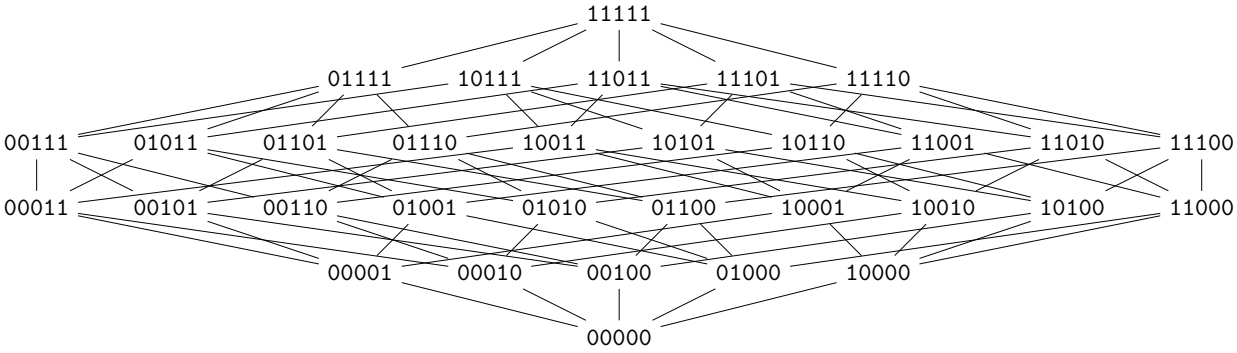

Abbildung 1.6: 5D-Hyperwürfel

Aufgabe 1.14

Man verwendet eine Kodierung mit folgenden 4 gültigen Code-Wörtern der Länge 5:

(A) `00000`
(B) `00110`
(C) `11001`
(D) `11111`

Folgende vier fehlerhafte Bitfolgen wurden empfangen. Bei welchen dieser vier Bitfolgen kann man eindeutig den Fehler korrigieren, falls man sicher ist, dass in jeder Bitfolge genau ein Fehler passiert ist?

1. `10000`
2. `11101`
3. `01111`
4. `00100`

Aufgabe 1.15

Zeichnen Sie alle **1-Fehler-erkennenden** Kodierungen im unten stehenden Hyperwürfel ein, ausgehend vom Code `011`. Wie viele Code-Wörter können Sie finden?

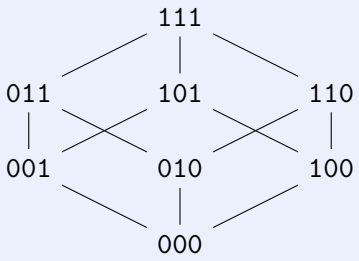

Abbildung 1.7: 3D-Hyperwürfel

Aufgabe 1.16

Zeichnen Sie alle **1-Fehler-korrigierenden** Kodierungen im unten stehenden Hyperwürfel ein, ausgehend vom Code `011`.

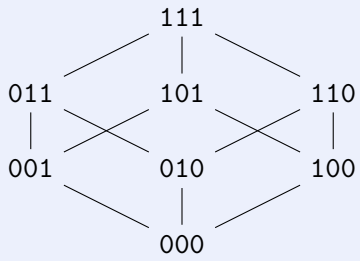

Abbildung 1.9: 3D-Hyperwürfel

Aufgabe 1.17

Zeichnen Sie alle **1-Fehler-erkennenden** Kodierungen im unten stehenden Hyperwürfel ein, ausgehend vom Code `0000`.

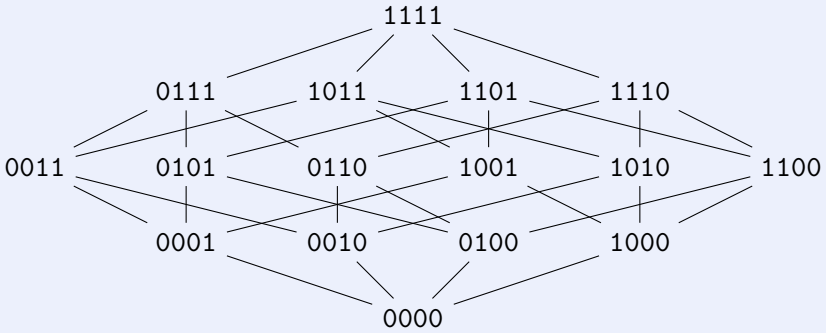

Abbildung 1.11: 4D-Hyperwürfel

Aufgabe 1.18

Zeichnen Sie alle **1-Fehler-korrigierenden Kodierungen** im unten stehenden Hyperwürfel ein, ausgehend vom Code `0000`. Wie viele 1-Fehler-korrigierenden Codewörter gibt es?

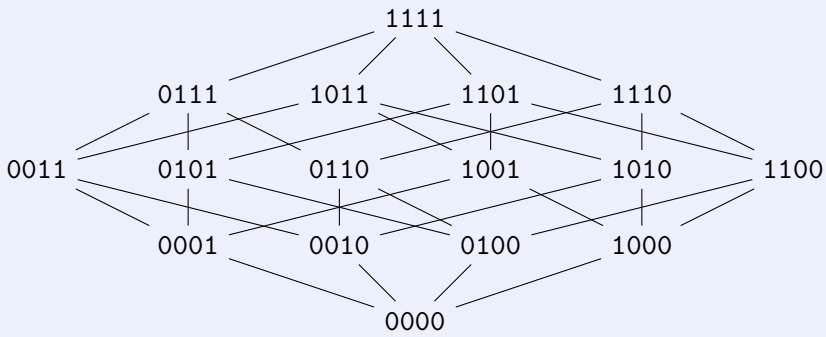

Abbildung 1.12: 4D-Hyperwürfel

1.5 Effiziente Codes

Bisher haben wir primär eine Art kennengelernt, wie Fehler automatisch behoben werden können. Allerdings haben wir uns bisher in allen Beispielen auf sehr wenige gültige Codewörter beschränkt. Stellen Sie sich nur schon vor, Sie möchten alle Zeichen des Alphabets schicken. Die Suche nach geeigneten Codewörtern würde schnell sehr lang und unübersichtlich, der entsprechende Hyperwürfel könnte wohl kaum auf einem A4-Blatt dargestellt werden. Daher schauen wir uns in diesem Kapitel einen anderen Ansatz an, welcher für beliebig lange Codes funktioniert. Dazu schauen wir uns zuerst einen Zaubertrick an.

Dazu braucht es zwei Zauberer. Der erste Zauberer verlässt zu beginn des Spiels den Raum.

Der zweite Zauberer bittet eine beliebige Person, ein $n \cdot m$-Feld von Spiel-Karten auf dem Tisch auszulegen, so dass einige Karten mit dem Kopf nach oben zeigen und andere mit dem Kopf nach unten (siehe Abbildung 1.13).

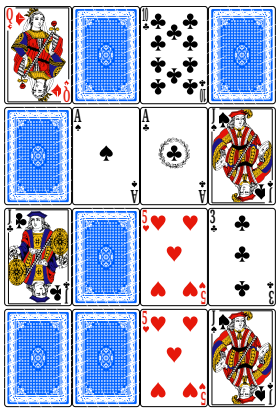

Abbildung 1.13: Erster Schritt: Auslegen der Karten durch eine beliebige Person ($n = m = 4$)

Der zweite Zauberer (der immer noch im Raum ist) legt nun noch eine weitere Zeile (oben) und eine weitere Spalte (rechts) dazu. Der Zauberer kann das Hinzufügen der neuen Zeilen und Spalten beispielsweise damit begründen, dass dies „das Spiel noch etwas schwieriger machen soll“ (siehe Abbildung 1.14).

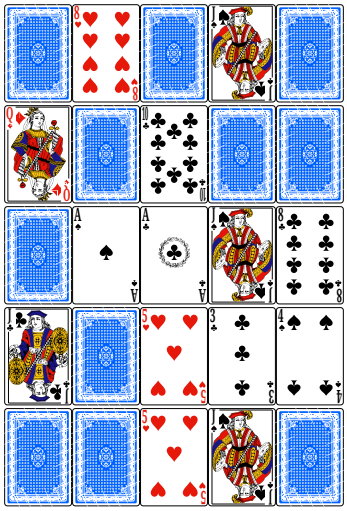

Abbildung 1.14: Zweiter Schritt: Hinzufügen einer Reihe und einer Spalte durch den Hilfszauberer

Eine zuschauende Person darf nun genau eine Karte umdrehen (siehe Abbildung 1.15).

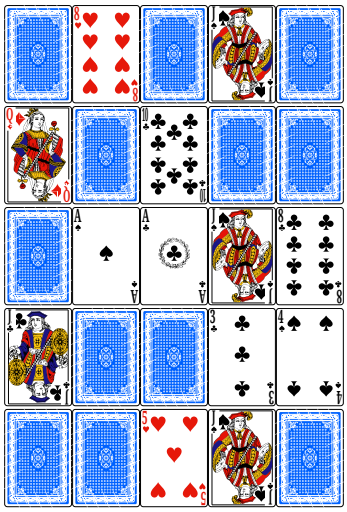

Abbildung 1.15: Dritter Schritt: Karten, nachdem jemand aus dem Publikum eine Karte umgedreht hat

Der erste Zauberer, welcher bisher während dem ganzen Spiel ausserhalb des Raums war, betritt nun wieder den Raum und errät, welche Karte umgedreht wurde.

Aufgabe (Challenge) 1.19

Können Sie erraten, welche Karte in Abbildung 1.15 umgedreht wurde, ohne die vorherigen Bilder anzuschauen?

Um zu erraten, welche Karte umgedreht wurden, zählt der Zauberer nun, wie viele Karten mit dem Kopf nach oben in jeder Zeile und Spalte liegen.

- Zeilen: 2, 2, 4, **3**, 2
- Spalten: 2, 2, **3**, 4, 2

Die umgedrehte Karte muss sich also genau auf der vierten Zeile und in der dritten Spalte befinden, da nur dort eine ungerade Anzahl Karten mit dem Kopf nach oben ist!

Erklärung: Die helfende Person hat in Schritt 3 die Karten nicht ganz „zufällig“ hinzugefügt, sondern sie so arrangiert, dass die Summe der Karten mit dem Kopf nach oben in jeder Zeile und in jeder Spalte gerade war.

Definition 1.2 (Kartentrick)**:**
In der Kartentrick-Kodierung werden Codewörter in eine Tabelle eingefügt, danach werden für jede Spalte und jede Zeile Kontrollbits rechts, bzw. oben angefügt.

Nehmen wir an, wir wollen folgendes Wort verschicken:

`0 1 0 1 0 1`

Dieses Wort der Länge 6 liesse sich gut in eine 2×3-Tabelle eintragen:

0	1	0
1	0	1

Tabelle 1.11: Ursprüngliche Nachricht als Tabelle

Nun fügen wir der Tabelle die Kontrollbits oben und rechts hinzu:

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

Tabelle 1.12: Ursprüngliche Nachricht + Kontrollbits

Die Kontrollbits sind so gewählt, dass jede Spalte und jede Zeile eine gerade Anzahl Einsen enthält. Wenn nun ein Bit geflippt wird, gibt es genau eine Spalte und eine Zeile, in welcher die Anzahl Einsen *nicht* gerade ist, somit können wir den Ort des abgefälschten Bits lokalisieren und den Fehler automatisch beheben.

Um das neue Codewort (mit Kontrollbits) zu bilden, wird wie folgt vorgegangen: Fügen Sie dem ursprünglichen Code-Wort zuerst die erste Zeile, und danach die letzte Spalte *ohne erstes Bit* an.

Aufgabe 1.20

Wir haben $2^{12} = 4096$ Nachrichten der Länge 12 Bits. Nutzen Sie den Kartentrick, um Code-Wörter für die folgenden Nachrichten zu generieren:

1. `010111101010`
2. `000000000000`
3. `111100101100`

Aufgabe 1.21

Man verwendet für die Nachrichten $a_1a_2a_3a_4a_5a_6$, der Länge 6 eine 2×3-Tabelle mit 6 Kontrollbits wie in Definition 1.2. Die folgenden Nachrichten sind leider beschädigt worden. Bestimmen Sie die ursprüngliche Nachricht (das gesendete Code-Wort), wenn Sie annehmen, dass genau 1 Bit umgeflippt wurde.

1. 0 1 0 1 0 1 1 1 0 0 0 0
2. 0 1 1 1 0 1 1 1 0 1 0 0
3. 1 1 1 0 1 0 1 1 1 1 0 1
4. 1 1 1 0 1 0 1 1 1 1 1 0

1.6 Der XOR-Operator

Um das Paritätsbit zweier binären Codes zu berechnen wird der **XOR-Operator** („Exklusives Oder“, $\oplus$) verwendet. Die Addition modulo 2 zweier Bits (XOR) ist definiert in Tabelle 1.14.

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Tabelle 1.14: XOR (Addition modulo 2) für alle 4 möglichen Bit-Kombinationen

Das Ergebnis ist also genau dann 1, wenn genau eines der beiden Eingangsbits gleich 1 ist. XOR wird auch „Addition modulo 2“ genannt, da das Resultat einer XOR-Operation immer genau der Summe beider Bits modulo 2 entspricht.

XOR kann nicht nur auf einzelne Bits, sondern auch auf mehrere Bitfolgen angewendet werden. Dabei wird jedes Bit der Folge 1 mit dem Bit der Folge 2 an derselben Stelle verglichen. Das Resultat einer XOR-Operation wird Parität (P) genannt.

Beispiel 1.5:
Gegeben seien drei Datenblöcke:

$$D_1 = 1011_2, \quad D_2 = 1100_2, \quad D_3 = 0110_2$$

Die Parität P wird durch XOR-Verknüpfung berechnet:

$$P = D_1 \oplus D_2 \oplus D_3$$

Wir berechnen schrittweise:

$$D_1 \oplus D_2 = 1011_2 \oplus 1100_2 = 0111_2$$

$$P = 0111_2 \oplus 0110_2 = 0001_2$$

Das Paritätsbit wird zusammen mit den Daten gespeichert. Fällt eine Festplatte aus, kann der fehlende Wert durch Umstellen der XOR-Operation wiederhergestellt werden.

Aufgabe 1.22

Gegeben sind die folgenden binären Codes:

$$A = 1101_2, \quad B = 1010_2$$

Berechnen Sie $A \oplus B$.

Aufgabe 1.23

Gegeben sind vier binäre Codes:

$$A = 1011_2, \quad B = 1100_2, \quad C = 0110_2, \quad D = 1001_2$$

Berechnen Sie $A \oplus B \oplus C \oplus D$.

1.7 RAID

Redundant Array of Independent Disks (RAID) ist eine spannende Technologie, um die Funktionsweise von Paritätsbits in der Praxis zu Illustrieren. RAID ist ein Standard zur Organisation mehrerer Festplatten zu einem sogenannten logischen Laufwerk. Dabei werden meherere physische Festplatten zu einer „virtuellen“ Festplatte zusammengefasst. Ziel ist es, die Datensicherheit, die Ausfallsicherheit und/oder die Lese- und Schreib-Leistung zu verbessern. RAID wird häufig in Servern und Speichersystemen eingesetzt, um Redundanz zu gewährleisten oder die Geschwindigkeit von Speicherzugriffen zu erhöhen.

Im Kontext der fehlerkorrigierenden und fehlererkennenden Kodierung spielt RAID eine wichtige Rolle. Während RAID 0 sich auf Performance ohne Redundanz konzentriert, nutzen andere RAID-Level, insbesondere RAID 5 und RAID 6, Paritätsdaten zur Fehlerkorrektur. Im folgenden schauen wir uns nur diejenigen RAID-Typen an, welche heute am weitesten verbreitet sind.

1.7.1 RAID 0: Striping

RAID 0 verteilt Daten in Blöcken auf mehrere Festplatten (*striping*), um die Lese- und Schreibgeschwindigkeit zu erhöhen. Dabei gibt es keine Redundanz, sodass der Ausfall einer einzigen Festplatte zu einem vollständigen Datenverlust führt.

- Vorteil: Sehr hohe Geschwindigkeit bei Lese- und Schreiboperationen.
- Nachteil: Keine Fehlerkorrektur oder Redundanz; hoher Datenverlust bei Festplattendefekt.

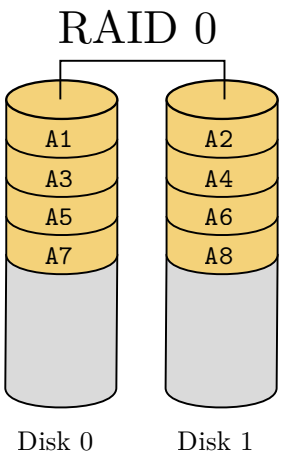

Abbildung 1.16: RAID-0-Architektur mit zwei physischen Festplatten

Aufgabe 1.24

Ein System verwendet RAID 0 mit zwei Festplatten, wobei die Daten in 4-Byte-Blöcken verteilt werden. Gegeben ist die folgende Datei:

```
ABCDEFGH
```

Wie werden die Daten auf den beiden Festplatten gespeichert?

1.7.2 RAID 1: Mirroring

RAID 1 speichert identische Kopien der Daten auf zwei Festplatten (*mirroring*). Fällt eine Festplatte aus, können die Daten von der zweiten Festplatte wiederhergestellt werden.

- Vorteil: Sehr hohe Datensicherheit, da jede Information doppelt gespeichert wird.
- Nachteil: Speicherplatz wird ineffizient genutzt, da nur die Hälfte der Gesamtkapazität verfügbar ist.

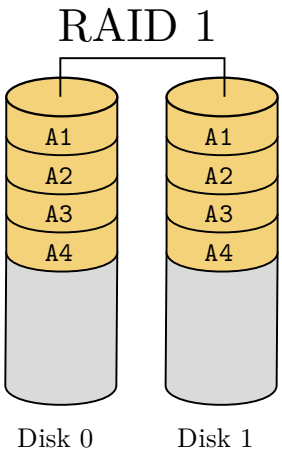

Abbildung 1.17: RAID-1-Architektur mit zwei physischen Festplatten

Absicherung gegen Datenverlust kann mit lediglich 2 Festplatten nicht Speicherplatz-effizienter als

mit RAID 1 erfolgen, d.h., ein Verlust von 50% des physischen Speicherplatzes muss in Kauf genommen werden. Effizientere Lösungen bieten sich an, wenn weitere Speicherplatten vorhanden sind. Selbstverständlich kann RAID 1 auch mit 4, 6, 8 oder mehr Festplatten umgesetzt werden.

Aufgabe 1.25

Ein Server verwendet RAID 1 mit zwei Festplatten. Was passiert, wenn eine der beiden Festplatten ausfällt? Kann das System weiterhin genutzt werden?

1.7.3 RAID 4: Paritätsbits

RAID 4, 5 und 6 nutzen für die Datensicherheit statt *mirroring* (Spiegelung der Daten) einen effizienteren Ansatz, um Daten über mehrere Laufwerke zu verteilen, indem sie Paritätsbits verwenden. Dabei verwendet RAID 5 ein einzelnes Paritätsbit, welches es über verschiedene Festplatten verteilt, so dass jeder „Striping-Block“ $A_1A_2A_3\cdots$ über genau ein Paritätsbit A_p verfügt (Abbildung 1.19).

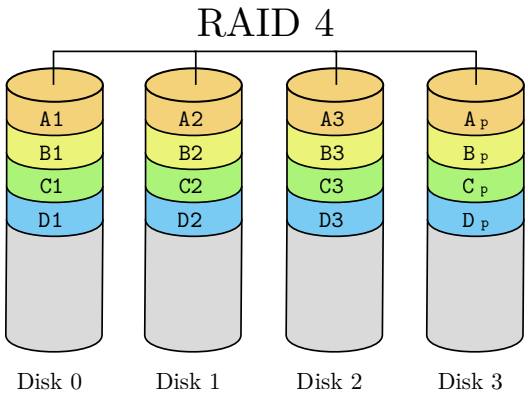

Abbildung 1.18: RAID-4-Architektur mit vier physischen Festplatten und einem Paritätsbit

1.7.4 RAID 5: Verteilte Paritätsbits

Da das Paritätsbit bei jedem Schreibvorgang neu geschrieben werden muss (auch bei kleinen Dateien), wird bei RAID-4 die Parität-Festplatte übermässig stark beansprucht. Daher werden die Paritätsbits bei RAID-5 im Gegenteil zu RAID-4 auf alle Festplatten verteilt. Ansonsten funktioniert RAID-5 identisch wie RAID-4 (siehe Abbildung 1.19).

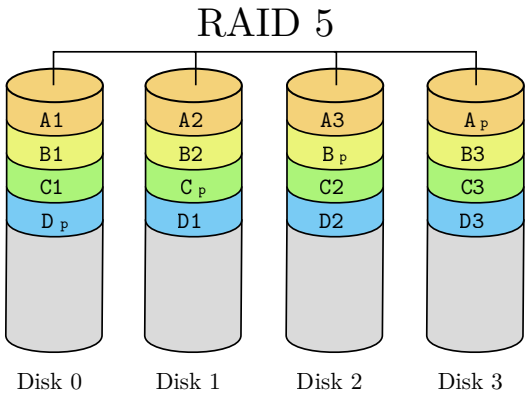

Abbildung 1.19: RAID-5-Architektur mit vier physischen Festplatten und einem *verteilten* Paritätsbit

Falls nun eine Festplatte ausfällt, können die verlorengegangenen Daten einfach rekonstruiert werden, indem das *parity bit* über die verbliebenen Datenblöcke erneut berechnet wird. Somit ist ein Speicher-System bei RAID 5 gegen den Ausfall einer Festplatte gesichert.

Bei RAID 5 ist das Paritätsbit für jeden Block auf einer anderen Festplatte gespeichert, wodurch die Performance beim Schreiben von Daten erhöht wird, da die Schreib-Last so auf alle Festplatten gleichmässig verteilt werden kann.

Im Folgenden wird aufgezeigt, wie diese berechnet werden.

RAID 5 verteilt die Paritätsinformationen über alle Festplatten. Die Berechnung erfolgt mit einer XOR-Operation. Beispiel mit vier Datenblöcken D_1, D_2, D_3, D_4:

$$P = D_1 \oplus D_2 \oplus D_3 \oplus D_4 \tag{1.2}$$

Beispiel 1.6:
Nehmen wir an, wir berechnen die Parität für folgende Datenblöcke:

$$D_1 = 1101_2$$
$$D_2 = 1011_2$$
$$D_3 = 0110_2$$
$$D_4 = 1001_2$$

Die Paritätsberechnung ergibt:

$$\begin{aligned} P &= 1101_2 \oplus 1011_2 \oplus 0110_2 \oplus 1001_2 \\ &= 1001_2 \end{aligned}$$

Falls ein Datenblock ausfällt, kann er durch Umstellen der XOR-Operation rekonstruiert werden.

Aufgabe 1.26

Ein RAID-5-System besteht aus drei Festplatten. Die gespeicherten Datenblöcke lauten:

$$D_1 = 1011_2$$
$$D_2 = 1100_2$$

Berechne das Paritätsbit P für RAID 5.

Je mehr Festplatten verfügbar sind, desto mehr Speicherplatz ist prozentual für die Speicherung von Daten verfügbar. Da immer eine Festplatte für die Datensicherheit verwendet wird, stehen bei 3 Festplatten 2 für die Speicherung von Daten zur Verfügung, bei 4 Festplatten 3, bei 5 Festplatten 4 usw.

1.7.5 RAID 6: Zwei Paritätsbits

Bei RAID-6 werden sogar zwei Paritätsbits verwendet, womit diese Konfiguration Schutz von zwei Festplatten-Ausfällen bietet (Abbildung 1.20). Dadurch fallen jedoch auch zwei Festplatten als mögliche Speichermedien weg, da diese für die Datensicherheit benötigt werden.

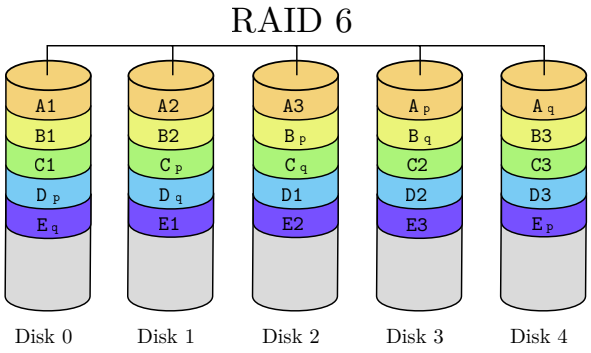

Abbildung 1.20: RAID 5-Architektur mit fünf physischen Festplatten und zwei Paritätsbits

RAID 6 verwendet zwei Paritätswerte: eine XOR-basierte Parität (P) wie in RAID 5 und eine zweite Parität Q, die durch eine Galois-Feld-Multiplikation berechnet wird. Für Beispiele einer Berechnung des zweiten Paritätsbits sei auf die Webseite von Oracle hingewiesen.

Aufgabe 1.27

Marina gründet eine IT-Firma und kauft sich 8 Festplatten für die Speicherung ihrer Daten. Da sie viele Daten hat, möchte sie möglichst viel nutzbaren Speicherplatz haben, gleichzeitig möchte sie gegen den Ausfall von *einer* Festplatte geschützt sein. Welche RAID-Konfiguration empfehlen Sie Marina (0, 1, 5 oder 6)? Begründen Sie Ihre Antwort.

Aufgabe 1.28

Lara möchte ihre schönsten Fotos sichern und hat sich dazu zwei Festplatten gekauft. Da ihre Fotos für sie wichtig sind, möchte sie sich gegen den Ausfall einer Festplatte schützen und entscheidet sich daher für eine RAID-1-Konfiguration.

- Weshalb kann Lara selbst dann, wenn sie sich zwei gleich grosse Festplatten kauft, nur maximal die Hälfte des Speicherplatzes für sich nutzen?
- Welche alternative RAID-Konfiguration könnte Lara in Betracht ziehen, um mehr Speicherplatz zu erhalten? Was wären Nachteile davon?
- Welche RAID-Alternative gäbe es für Lara, um sowohl mehr Speicher als auch Redundanz zu gewährleisten?

Aufgabe 1.29

Bestimmen Sie für den Fall, dass sie 6 Festplatten gleicher Grösse besitzen, den verfügbaren Speicherplatz für die Konfigurationen RAID 0, RAID 1, RAID 4, RAID 5 und RAID 6. Geben Sie den verfügbaren Speicherplatz als Bruch an (z.B. 3/6). Geben Sie zudem die Ausfallsicherheit an (wie viele Festplatten können *im schlimmsten Fall* ausfallen ohne Datenverlust?).

Konfiguration	**Verfügbare Festplatten**	**Ausfallsicherheit**
RAID 0		
RAID 1		
RAID 4		
RAID 5		
RAID 6		

Anhang A

Lernziele

- ☐ Ich kann die Prüfziffern eines EAN-Codes berechnen bei gegebener Formel.
- ☐ Ich kann bestimmen, ob ein gegebener EAN-Code korrekt ist.
- ☐ Ich kann bestimmen, welche Fehler (Vertippen einer Ziffer, Vertauschen von zwei Ziffern, Ziffer vergessen) durch eine Prüfziffer erkannt werden.
- ☐ Ich kann den Abstand zweier Bitfolgen gleicher Länge bestimmen.
- ☐ Ich kann den Mindest-Abstand einer Kodierung bestimmen.
- ☐ Ich kann bestimmen, wie viele Fehler eine Kodierung erkennt.
- ☐ Ich kann bestimmen, wie viele Fehler eine Kodierung korrigieren kann.
- ☐ Ich kann erklären, welchen Abstand k-fehlererkennende Kodierungen haben müssen.
- ☐ Ich kann erklären, welchen Abstand k-fehlererkorrigierende Kodierungen haben müssen.
- ☐ Ich kann den n-dimensionalen Hyperwürfel für kleine n ($n \leq 4$) zeichnen.
- ☐ Ich kann Kodierungen angeben, die einen bestimmten Abstand haben, bzw. eine bestimmte Anzahl Fehler erkennen oder korrigieren können.
- ☐ Ich kann die „Kartentrick-Kodierung“ einer Bitfolge angeben.
- ☐ Ich kann erklären, wie die optimale Kartentrickkodierung (= optimale Länge und Breite für das Rechteck) für eine Bitfolge mit einer gegebenen Länge aussieht.
- ☐ Ich kann überprüfen, ob eine gegebene Kartentrick-Kodierung korrekt ist.
- ☐ Ich kann bei einer Kartentrick-Kodierung mit einem Fehler den Fehler finden und korrigieren.
- ☐ Ich kann den `XOR`-Operator auf mehrere Bitfolgen gleicher Länge anwenden.
- ☐ Ich kann RAID-0-, RAID-1-, RAID-5- und RAID-6-Konfigurationen schematisch illustrieren und deren Nutzen erklären, indem ich die Begriffe *striping*, *mirroring* und *Paritätsbits* verwende.
- ☐ Ich kann für eine vorgegebene Anzahl Festplatten sowie Anforderungen an Redundanz und Schreibgeschwindigkeit bestimmen, welche der vier RAID-Konfigurationen am besten geeignet ist.
- ☐ Ich kann ein Paritätsbit mit dem `XOR`-Operator berechnen.

Glossar

ASCII American Standard Code for Information Interchange. 2

EAN European Article Number. 3–5, 23

ISBN International Standard Book Number. 5

RAID Redundant Array of Independent Disks. 17, 23