# 5 Codierungstheorie und Kryptographie

## 5.1 Grundbegriffe der Codierungstheorie

Die *Codierungstheorie* beschäftigt sich mit der sicheren Übertragung von Nachrichten. Hierbei gehen wir davon aus, daß ein Sender (Quelle) eine Nachricht über einen Übertragungskanal an einen Empfänger schickt. Die Art der Codierung hängt hierbei i. a. von den realen Verhältnissen zwischen Quelle und Empfänger ab. Dabei kann der Kanal gegebenenfalls auch gestört sein. Beispiele für solche Übertragungen sind

- das Morsen.
- das Abspielen einer Musik-CD. Hierbei wird die „digitalisierte Musik" in ein analoges Signal umgewandelt und an ein Verstärkersystem weiterleitet.
- das Abspeichern einer Datei auf einem Computer. Hierbei wird der Inhalt der Datei vom Speicher auf die Festplatte übertragen.
- die Komprimierung von Dateien. Hierbei wird der Inhalt mit einem Komprimierungsverfahren umgewandelt. Dies geschieht z. B. beim MP3-Verfahren zur Codierung von Musik oder beim Programm WinZip zum Komprimieren von Dateien.
- das Senden von e-mail.
- die Übertragung von der Tastatur eines Computers an den Bildschirm.
- Zu Beginn des elektronischen Zeitalters wurden Programme mittels Lochkarten an einen Computer geschickt.

Hierbei sind folgene Dinge von Bedeutung:

- Die Nachricht soll möglichst effizient übertragen werden.
- Ist der Kanal gestört, so sollen Übertragungsfehler gefunden oder gegebenenfalls sogar korrigiert werden.

**Definition 5.1** Unter der *Codierung* einer Nachricht versteht man ihre Umwandlung von einem Alphabet in ein anderes Alphabet. Unter einem *Alphabet* $\mathcal{A} = \{z_1,z_2,\ldots,z_m\}$ verstehen wir eine endliche Menge von *Zeichen* (oder *Buchstaben*) $z_1,z_2,\ldots,z_m$. Ein *Wort* bzw. eine Zeichenkette der *Länge n* ist dann ein $n$-tupel $W \in \mathcal{A}^n$. Wörter werden meist ohne Klammern geschrieben: $W = w_1w_2\ldots w_n$. Die Menge aller Wörter über dem Alphabet $\mathcal{A}$ ist $\mathcal{W} = \bigcup_{n=0}^{\infty} \mathcal{A}^n$. Hierbei enthält die Menge $\mathcal{A}^0$ das *leere Wort.* △

**Beispiel 5.2** Positive Langzahlen bzgl. der Basis $B$ sind Wörter über dem Alphabet $\mathbb{Z}_B = \{0,1,\ldots,B-1\}$. Wandelt man eine Dezimalzahl in eine binäre Zahl um oder umgekehrt, so ist dies eine Codierung.

Das binäre Alphabet besteht aus den zwei Zeichen {0,1}. Diese Zeichen bezeichnen wir als *Bits.* Die *Bitlänge* eines Worts ist die Anzahl der erforderlichen Bits seiner binären Darstellung. Für ein Wort der Länge $n$ über einem Alphabet $\mathcal{A}$ mit $m$ Zeichen benötigt man $n \cdot \log_2 m$ Bits,[1] seine Bitlänge ist also $n \cdot \log_2 m$, s. Übungsaufgabe 5.1. Beispielsweise kann man jeden Buchstaben des Alphabets $\mathcal{A} = \{A,B,C,D,E,F,G,H\}$, welches aus 8 Zeichen besteht, mit $3 = \log_2 8$ Bits darstellen, z. B. durch

$$\begin{array}{l} A \mapsto 000 \\ B \mapsto 001 \\ C \mapsto 010 \\ D \mapsto 011 \\ E \mapsto 101 \\ F \mapsto 110 \\ G \mapsto 110 \\ H \mapsto 111 \end{array} .$$

und ein 15-buchstabiges Wort im Alphabet $\mathcal{A}$ benötigt dann $15 \cdot 3 = 15 \cdot \log_2 8 = 45$ Bits.

Für Schrifttexte verwenden wir normalerweise das Alphabet $\{A,B,\ldots,Y,Z\}$, welches i. a. durch die kleinen Buchstaben sowie weitere Zeichen wie Leerzeichen, Punkt, etc. ergänzt wird.

Zur Datenspeicherung in Computern wird der *ASCII-Zeichensatz* verwendet. Dieser besteht insgesamt aus 128 Zeichen und enthält alle „normalen Buchstaben". Intern werden diese 128 Zeichen wieder binär dargestellt. Zur Darstellung eines von $128 = 2^7$ verschiedenen Zeichen benötigt man 7 Bits.

Da im Standard-ASCII-Zeichensatz keine internationalen Zeichen wie Umlaute enthalten sind, verwenden IBM-PCs einen *erweiterten ASCII-Zeichensatz,* welcher

[1] gegebenenfalls geeignet gerundet

aus 256 Zeichen besteht. Zur Darstellung von $256 = 2^8$ verschiedenen Zeichen benötigt man 8 Bits. Diese Speichereinheit bezeichnet man als *Byte*.

Ein etwas anderes Beispiel ist das *Morsealphabet*. Es ist das Alphabet der Funkamateure. Jedem Buchstaben entspricht hierbei ein bestimmtes Kurz-Lang-Muster:[2]

| | | | | |
|---|---|---|---|---|
| 1 ↦ • − − − − | 2 ↦ • • − − − | 3 ↦ • • • − − | 4 ↦ • • • • − | 5 ↦ • • • • • |
| 6 ↦ − • • • • | 7 ↦ − − • • • | 8 ↦ − − − • • | 9 ↦ − − − − • | 0 ↦ − − − − − |
| *A* ↦ • − | *B* ↦ − • • • | *C* ↦ − • − • | *D* ↦ − • • | *E* ↦ • |
| *F* ↦ • • − • | *G* ↦ − − • | *H* ↦ • • • • | *I* ↦ • • | *J* ↦ • − − − |
| *K* ↦ − • − | *L* ↦ • − • • | *M* ↦ − − | *N* ↦ − • | *O* ↦ − − − |
| *P* ↦ • − − • | *Q* ↦ − − • − | *R* ↦ • − • | *S* ↦ • • • | *T* ↦ − |
| *U* ↦ • • − | *V* ↦ • • • − | *W* ↦ • − − | *X* ↦ − • • − | *Y* ↦ − • − − |
| *Z* ↦ − − • • | *Ä* ↦ • − • − | *Ö* ↦ − − − • | *Ü* ↦ • • − − | *CH* ↦ − − − − |

Die Morsetelegrafie spielt auch heute noch eine Rolle. Die Kenntnis dieses Alphabets kann Leben retten: Auch nach Ausfall der Funkverbindung kann ein Schiff noch SOS rufen, z. B. durch den Einsatz von Scheinwerfern. △

**Sitzung 5.1** *Mathematica* hat einen noch viel größeren Zeichensatz. Die ersten 128 Zeichen des *Mathematica*-Zeichensatzes entsprechen dem normalen ASCII-Zeichensatz, dessen erste 32 Zeichen *Steuerzeichen* sind. Wir bekommen eine Tabelle der ersten 256 Zeichen durch[3]

```
In[1]:= Partition[Table[FromCharacterCode[k],{k,0,255}],16]
```

Unser normales Alphabet findet man in dem Bereich:

```
In[2]:= Partition[Table[FromCharacterCode[k],{k,65,128}],16]
```

$$\text{Out[2]}= \begin{pmatrix} A & B & C & D & E & F & G & H & I & J & K & L & M & N & O & P \\ Q & R & S & T & U & V & W & X & Y & Z & [ & \backslash & ] & \hat{} & \_ & ` \\ a & b & c & d & e & f & g & h & i & j & k & l & m & n & o & p \\ q & r & s & t & u & v & w & x & y & z & \{ & | & \} & \sim & & \end{pmatrix}$$

Andere Zeichensätze hat *Mathematica* mit größeren Nummern versehen. Beispielsweise erhalten wir die griechischen Buchstaben mit

```
In[3]:= Partition[Table[FromCharacterCode[k],{k,913,976}],16]
```

$$\text{Out[3]}= \begin{pmatrix} A & B & \Gamma & \Delta & E & Z & H & \Theta & I & K & \Lambda & M & N & \Xi & O & \Pi \\ P & & \Sigma & T & \Upsilon & \Phi & X & \Psi & \Omega & & & & & & & \\ \alpha & \beta & \gamma & \delta & \varepsilon & \zeta & \eta & \theta & \iota & \kappa & \lambda & \mu & \nu & \xi & o & \pi \\ \rho & \varsigma & \sigma & \tau & \upsilon & \phi & \chi & \psi & \omega & & & & & & & \end{pmatrix}$$

Die Codierung zwischen den ASCII-Zeichen und den dazugehörigen ASCII-Nummern geschieht mit Hilfe der *Mathematica*-Funktionen `FromCharacterCode` und `ToCharacterCode`. Wir erhalten beispielsweise[4]

[2] Der Punkt bedeutet kurz, der Bindestrich lang.

[3] Wir haben die Ausgabe unterdrückt.

[4] Die Eingabe von $\alpha$ kann entweder durch `<ESC>a<ESC>` oder mit einer der Paletten geschehen.

```
In[4]:= ToCharacterCode["α"]
Out[4]= {945}
```

Der griechische Buchstabe $\alpha$ hat also in *Mathematica* die Nummer 945. □

## 5.2 Präfixcodes

In diesem Abschnitt behandeln wir eine besonders wichtige Eigenschaft von Codes.

**Definition 5.3** Seien $\mathcal{A}$ und $\mathcal{B}$ zwei Alphabete und $\mathcal{W} = \bigcup_{n=0}^{\infty} \mathcal{A}^n$ sowie $\mathcal{W}' = \bigcup_{n=0}^{\infty} \mathcal{B}^n$ die zugehörigen Wortmengen. Dann kann eine injektive Abbildung $c : \mathcal{A} \to \mathcal{W}'$, welche also verschiedenen Buchstaben aus $\mathcal{A}$ verschiedene Wörter aus $\mathcal{W}'$ zuordnet, zu einer Abbildung $C : \mathcal{W} \to \mathcal{W}'$ fortgesetzt werden, indem man sukzessive die Bilder der einzelnen Buchstaben zu einem neuen Wort zusammensetzt:

$$C(x_1x_2\ldots x_n) := c(x_1)c(x_2)\ldots c(x_n) .$$

$C$ ist ein *Code*, und die Bilder unter $c$ heißen *Codewörter*. △

Wird die Nachricht umgewandelt (codiert), so sollte eine einwandfreie Rückumwandlung (Decodierung) möglich sein. Wir betrachten folgendes

**Beispiel 5.4** Seien $\mathcal{A} = \{A,B,C\}$ und $\mathcal{B} = \{0,1\}$ zwei Alphabete sowie $\mathcal{W} = \bigcup_{n=0}^{\infty} \mathcal{A}^n$ und $\mathcal{W}' = \bigcup_{n=0}^{\infty} \mathcal{B}^n$ die zugehörigen Wortmengen. Wir erklären einen Code $C : \mathcal{W} \to \mathcal{W}'$ durch $A \mapsto 1, B \mapsto 0, C \mapsto 01$. Das Wort

$$W = ABCAACCBAC$$

wird somit codiert durch

$$W' = 10011101010101 .$$

Gemäß der Abbildungsvorschrift ist diese Codierung natürlich eindeutig. Wir wollen nun allerdings umgekehrt aus $W'$ wieder $W$ gewinnen. Hierzu beginnen wir mit dem Wortanfang und interpretieren die Buchstaben sukzessive. Zunächst stellen wir fest, daß die erste 1 von $W'$ nur durch ein $A$ entstanden sein kann. Also ist das erste Zeichen von $W$ ein $A$. Das zweite Zeichen von $W'$ ist 0. Dies könnte von $B$ oder von $C$ stammen. Da aber das dritte Zeichen von $W'$ wieder 0 ist, kommt $C$ nicht in Frage, und wir wissen, daß das zweite Zeichen von $W$ ein $B$ ist. Die nächste Zeichenkombination in $W'$ ist 01. Dies ist nun ein Problem, da diese Kombination sowohl $C$ als auch $BA$ bedeuten kann. Damit ist eine eindeutige Decodierung unmöglich.

Ein besserer Code $C' : \mathcal{W} \to \mathcal{W}''$ ist gegeben durch $A \mapsto 0, B \mapsto 10, C \mapsto 11$. Dann wird $W$ codiert durch

$$W' = 0101100111110011 .$$

Man überzeuge sich, daß in diesem Fall $W$ leicht aus $W'$ wiedergewonnen werden kann. Welche Eigenschaft des Codes $C'$ ist hierfür verantwortlich? △

Dies führt uns zu der

**Definition 5.5** Ein Wort $x_1 x_2 \ldots x_k \in \mathcal{A}^k$ heißt *Präfix* eines Worts $W \in \mathcal{A}^m$, wenn es Anfangsstück von $W$ ist, d. h. $W = x_1 x_2 \ldots x_k x_{k+1} \ldots x_m$.

Unter einem *Präfixcode* verstehen wir einen Code $C : \mathcal{W} \to \mathcal{W}'$, bei welchem kein Codewort Präfix eines anderen Codeworts ist. △

Es gilt der

**Satz 5.6** Präfixcodes lassen sich eindeutig decodieren.

*Beweis:* Übungsaufgabe 5.2 □

**Beispiel 5.7** Beispiele für Präfixcodes sind *Blockcodes*, bei welchen alle Bilder $c(x_k)$ dieselbe Länge haben.

Beispiel: Caesarcode:[5] $\mathcal{A} = \{A, B, \ldots, Z\}$. Der Code besteht in einer Verschiebung um drei Buchstaben: $A \mapsto D$, $B \mapsto E$, … , $W \mapsto Z$, $X \mapsto A$, $Y \mapsto B$ und $Z \mapsto C$.

Beispiel eines Blockcodes mit *Blocklänge* 2: $c(A) = 00, c(B) = 01, c(C) = 11$.

Interessanter sind jedoch Präfixcodes, welche keine Blockcodes sind. Ein solcher Präfixcode war in Beispiel 5.4 betrachtet worden.

Die Morsezeichen bilden, wie man leicht nachprüft, *keinen Präfixcode*. Um gemorste Texte dennoch entschlüsseln zu können, müssen die einzelnen Zeichen jeweils durch eine Pause voneinander getrennt übertragen werden. Nimmt man dieses Pausenzeichen mit ins Alphabet auf, so bilden die jeweils um das Pausenzeichen verlängerten Morsecodes dann einen Präfixcode. △

In der Praxis möchte man Codes u. a. so entwerfen, daß die Bildwörter möglichst klein sind, d. h., möglichst wenige Zeichen enthalten. Ein Blockcode der Länge $m$ codiert ein Wort der Länge $n$ durch ein Wort der Länge $m \cdot n$. Eine Komprimierung kann man auf diese Weise nicht erzeugen.

[5] Caesar hat diesen Code als *Geheimschrift* verwendet. Der Code ist hierfür allerdings wenig geeignet. Warum?

Weiß man allerdings etwas über die *Buchstabenhäufigkeit* der in der Praxis auftretenden Wörter aus $\mathcal{W}$, so kann man diese Information dazu benutzen, einen Präfixcode zu erzeugen, welcher Buchstaben mit großer Häufigkeit durch kleine Codewörter und Buchstaben mit kleiner Häufigkeit durch große Codewörter codiert.

Das Alphabet $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ sei nun also zusammen mit einer *Häufigkeitsverteilung*

$$\begin{pmatrix} A_1 & p_1 \\ A_2 & p_2 \\ \vdots & \vdots \\ A_m & p_m \end{pmatrix}$$

gegeben, wobei $p_k \geqq 0$ $(k = 1,\ldots,m)$ und $\sum_{k=1}^{m} p_k = 1$ ist. Die Häufigkeiten $(p_k)_{k=1,\ldots,m}$ bilden somit ein *Wahrscheinlichkeitsmaß* auf dem Alphabet $\mathcal{A}$. Jede natürliche Sprache wie Deutsch, Englisch, Russisch etc. hat die ihr eigene Häufigkeitsverteilung, ja sogar jeder Autor hat seinen eigenen Wortschatz und somit seine eigene charakteristische Häufigkeitsverteilung, durch welche er gegebenenfalls sogar identifiziert werden kann.[6]

Einen solchen Code stellt der *Huffman-Code* dar, welcher als Bildalphabet die Menge {0,1} besitzt. Den Huffman-Code erzeugt man in mehreren Schritten. Hierbei faßt man bei jedem Schritt die zwei Quellensymbole mit den kleinsten Häufigkeiten zusammen[7] und faßt das Buchstabenpaar mit seiner Gesamthäufigkeit als einen neuen Buchstaben auf. Diesen Prozeß iteriert man, bis nur noch zwei Buchstaben übrigbleiben. Bei der Vergabe der Bildbits verwendet man die Iteration rückwärts. Dabei wird jeweils dem häufigeren Buchstaben eine 1, dem selteneren Buchstaben eine 0 zugeordnet.

**Beispiel 5.8** Es sei die Häufigkeitsverteilung

$$\begin{pmatrix} A & 0{,}1 \\ B & 0{,}12 \\ C & 0{,}18 \\ D & 0{,}2 \\ E & 0{,}4 \end{pmatrix}$$

[6] Für den Roman „Stiller Don" hat der russische Schriftsteller Scholochow 1965 den Nobelpreis für Literatur erhalten. Seit 1928 kursierten in Rußland Gerüchte, daß das Werk nicht von ihm stammt. In einer anonymen Studie eines russischen Kritikers wird das Werk dem 1920 an Typhus gestorbenen Schriftsteller Krjukow zugeschrieben. Hieraus ergibt sich die Frage: War die Vergabe des Literaturnobelpreises an Scholochow möglicherweise ungerechtfertigt? Mit Methoden der Statistik kann man nachweisen, daß Krjukow als Autor von „Stiller Don" mit großer Wahrscheinlichkeit nicht in Frage kommt, wohingegen nichts gegen Scholochow als Autor spricht.

[7] Gibt es hierbei mehrere Möglichkeiten, so wählt man eine dieser Möglichkeiten aus.

gegeben. Sukzessive Abarbeitung gemäß der beschriebenen Vorschrift liefert die Iteration

$$\begin{pmatrix} A & 0{,}1 \\ B & 0{,}12 \\ C & 0{,}18 \\ D & 0{,}2 \\ E & 0{,}4 \end{pmatrix} \rightarrow \begin{pmatrix} C & 0{,}18 \\ D & 0{,}2 \\ AB & 0{,}22 \\ E & 0{,}4 \end{pmatrix} \rightarrow \begin{pmatrix} AB & 0{,}22 \\ CD & 0{,}38 \\ E & 0{,}4 \end{pmatrix} \rightarrow \begin{pmatrix} E & 0{,}4 \\ ABCD & 0{,}6 \end{pmatrix}$$

Gemäß der letzten Matrix gilt $E \mapsto 0$, und das Codewort jedes der Buchstaben *A,B,C* und *D* beginnt mit 1. Dem vorletzten Schritt entnimmt man, da die Häufigkeit von *AB* kleiner ist als die Häufigkeit von *CD*, daß der zweite Buchstabe des Codeworts von *A* und *B* jeweils 0 ist, während der zweite Buchstabe des Codeworts von *C* und *D* jeweils 1 ist. Die drittletzte Verteilung liefert $C \mapsto 110$ und $D \mapsto 111$, während die ursprüngliche Verteilung die beiden letzten Codewörter $A \mapsto 100$ sowie $B \mapsto 101$ generiert.

Der gesamte Huffman-Code ist also gegeben durch die Codewörter

$$\begin{aligned} E &\mapsto 0 \\ A &\mapsto 100 \\ B &\mapsto 101 \\ C &\mapsto 110 \\ D &\mapsto 111 \end{aligned}$$

und läßt sich weiterhin durch einen *Codebaum* darstellen, welchen wir in der nächsten *Mathematica*-Sitzung erzeugen. △

**Sitzung 5.2** Mit *Mathematica* kann man den Codebaum auf folgende Weise als *Graph* realisieren. Ein Graph besteht aus einer Menge von *Ecken*[8], welche teilweise durch *Kanten* miteinander verbunden sind.

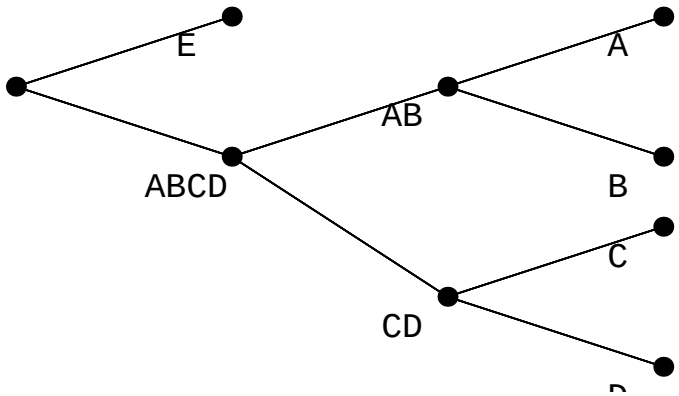
```
In[1]:= Needs["DiscreteMath`Combinatorica`"]

In[2]:= code = Graph[{{0,1,1,0,0,0,0,0,0},{1,0,0,0,0,0,0,0,0},
              {1,0,0,1,1,0,0,0,0},{0,0,1,0,0,1,1,0,0},
              {0,0,1,0,0,0,0,1,1},{0,0,0,1,0,0,0,0,0},
              {0,0,0,1,0,0,0,0,0},{0,0,0,0,1,0,0,0,0},
              {0,0,0,0,1,0,0,0,0}},{{0,0},{1,1},{1,-1},
              {2,0},{2,-3},{3,1},{3,-1},{3,-2},{3,-4}}]
```

[8] Diese werden manchmal auch *Knoten* genannt.

Out[2]= Graph $\left(\begin{pmatrix} 0&1&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0\\ 1&0&0&1&1&0&0&0&0\\ 0&0&1&0&0&1&1&0&0\\ 0&0&1&0&0&0&0&1&1\\ 0&0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0\\ 1&1\\ 1&-1\\ 2&0\\ 2&-3\\ 3&1\\ 3&-1\\ 3&-2\\ 3&-4 \end{pmatrix}\right)$

Als erstes Argument der Funktion `Graph` haben wir hierbei die *Adjazenzmatrix* des Graphen angegeben. Hierfür werden die Ecken zunächst von 1 beginnend durchnumeriert,[9] und der Koeffizient $a_{jk}$ der Adjazenzmatrix gibt die Anzahl der Kanten von Ecke $j$ nach Ecke $k$ an. Das zweite Argument von `Graph` besteht dann aus dem Koordinatenvektor der Ecken.[10] Nun können wir den Graphen zeichnen:

```
In[3]:= ShowLabeledGraph[code,{"","E","ABCD","AB","CD","A","B","C","D"}]
```




Out[3]= -Graphics-

Der Graph ist ein *Baum*, welcher in seiner linken Ecke eine *Wurzel* besitzt. Er stellt den angesprochenen Codebaum dar, von dem man die Huffman-Codierung in naheliegender Weise ablesen kann.

Den Huffman-Code berechnet man mit der Funktion

[9] In unserem Beispiel wurde die Numerierung gemäß dem folgenden Bild von links nach rechts und dann von oben nach unten festgelegt.

[10] Hierbei habe ich die Wurzel in den Koordinatenursprung und die Ecke *E* in den Punkt (1,1) gelegt. Die beiden Koordinaten werden von *Mathematica* nicht im selben Maßstab dargestellt.

```
In[4]:= Clear[Huffman]
        Huffman[Alphabet_] := Module[{k,alphabet,createcode,erst,zweit},
           If[!Abs[Sum[Alphabet[[k,2]],{k,Length[Alphabet]}] - 1.] < 0.01,
             Return["falsche Eingabe"]]; alphabet = Alphabet;
           createcode = Table[{alphabet[[k,1]],{}},{k,Length[alphabet]}];
           Do[alphabet = Sort[alphabet,#1[[2]] ≤ #2[[2]]&];
             erst = First[alphabet]; alphabet = Rest[alphabet];
             zweit = First[alphabet]; alphabet = Rest[alphabet];
             Do[If[!FreeQ[erst[[1]],createcode[[k,1]]],
                PrependTo[createcode[[k,2]],0]],{k,Length[createcode]}];
             Do[If[!FreeQ[zweit[[1]],createcode[[k,1]]],
                PrependTo[createcode[[k,2]],1]],{k,Length[createcode]}];
             alphabet = Prepend[alphabet,{{erst[[1]],zweit[[1]]},
                  erst[[2]] + zweit[[2]]}],
             {j,Length[alphabet] - 1}];
         {alphabet,createcode}]
```

Nach Eingabe unserer Häufigkeitsverteilung

```
In[5]:= Alphabet = {{"A",0.1},{"B",0.12},{"C",0.18},{"D",0.2},{"E",0.4}}
```

$$\text{Out[5]}= \begin{pmatrix} A & 0.1 \\ B & 0.12 \\ C & 0.18 \\ D & 0.2 \\ E & 0.4 \end{pmatrix}$$

läßt sich hieraus der Huffman-Code berechnen:

```
In[6]:= Huffman[Alphabet]
```

$$\text{Out[6]}= \left\{\left(\left\{E, \begin{pmatrix} A & B \\ C & D \end{pmatrix}\right\} \quad 1.\right), \begin{pmatrix} A & \{1,0,0\} \\ B & \{1,0,1\} \\ C & \{1,1,0\} \\ D & \{1,1,1\} \\ E & \{0\} \end{pmatrix}\right\}$$

Hierbei wird ein Zwischenresultat mit ausgegeben. □

Die Huffman-Codierung liefert gemäß Konstruktion immer einen Präfixcode. Man kann zeigen, daß der Huffman-Code in gewisser Weise sogar eine optimale Kompression vornimmt ([Sch1991], Satz 6.6, S. 44).

## 5.3 Prüfzeichenverfahren

Um Übertragungsfehler zu erkennen, werden häufig Daten mit zusätzlicher redundanter Information versehen.

Die einfachste Möglichkeit besteht darin, ein *Prüfzeichen* bzw. eine *Prüfziffer* zu verwenden. In UNIX-Systemen wird jedem ASCII-Zeichen $z$ beispielsweise ein zusätzliches Bit angehängt. Als zusätzliches Zeichen wählt man 0, falls die Anzahl der Einsen der Binärdarstellung von $z$ gerade ist (gerade *Parität*), andernfalls wird 1 angehängt. Das angehängte Bit heißt *Prüfbit*. Es entsteht somit aus jedem 7-Bit-ASCII-Zeichen ein Byte, dessen Parität gemäß Konstruktion gerade ist. Stimmt die Parität nach der Übertragung nicht mehr, so muß ein Übertragungsfehler geschehen sein. Dann hilft gegebenenfalls eine nochmalige Übertragung. Korrigieren kann man den Fehler mit einem solchen Verfahren nicht.

Ein ähnliches Verfahren wird auch beim *Einscannen* an der Supermarktkasse verwendet. Die EAN (Europäische Artikelnummer), welche eingescannt wird, stellt eine 13-stellige Dezimalzahl $a_1 a_2 \ldots a_{13}$ dar, deren letzte Ziffer ein Prüfzeichen ist, welches gemäß der Formel

$$a_{13} = \mathrm{mod}(-(1 \cdot a_1 + 3 \cdot a_2 + 1 \cdot a_3 + 3 \cdot a_4 + \cdots + 1 \cdot a_{11} + 3 \cdot a_{12}),10)$$

berechnet wird. Gibt es beim Einscannen einen Fehler, so piept es an der Kasse. Den Fehler beheben kann das Verfahren nicht, aber der Scanvorgang kann wiederholt werden. Wer Genaueres darüber wissen will, wie die verwendeten *Strichcodes* als Ziffern interpretiert werden, kann dies in [Sch1991] oder in [Jun1995] nachlesen.

Auf ähnliche Weise enthält die ISBN (Internationale Standard-Buchnummer) ein Prüfzeichen. Die ISBN stellt eine 9-stellige Dezimalzahl $a_1 a_2 \ldots a_9$ dar, gefolgt von einem Prüfzeichen, welches gemäß der Formel

$$a_{10} = \mathrm{mod}\Big(\sum_{k=1}^{9} k \cdot a_k, 11\Big)$$

berechnet wird. Da modulo 11 gerechnet wird, kann die Prüfziffer den Wert 10 haben und wird dann als römische Zahl, d. h. als $X$, geschrieben.[11]

Ein wesentlich komplizierteres Prüfzeichenverfahren wird bei der Numerierung deutscher Geldscheine verwendet. Man möchte nämlich sicherstellen, daß sich die Prüfzeichen aufeinanderfolgender Banknoten möglichst „zufällig" verhalten. Dies erschwert Fälschern das Handwerk.

[11] Testen Sie die ISBN des vorliegenden Buchs!

Bei der Numerierung deutscher Banknoten spielt die Diedergruppe $D_5$ eine wichtige Rolle, welche die Symmetrien des regelmäßigen Fünfecks beschreibt. Wir bezeichnen die innere Verknüpfung der Diedergruppe $D_5$ mit $\star$. Wir benötigen hier nichts weiter als die Verknüpfungstafel, welche gegeben ist durch

| $(D_5,\star)$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 1 | 2 | 3 | 4 | 0 | 6 | 7 | 8 | 9 | 5 |
| 2 | 2 | 3 | 4 | 0 | 1 | 7 | 8 | 9 | 5 | 6 |
| 3 | 3 | 4 | 0 | 1 | 2 | 8 | 9 | 5 | 6 | 7 |
| 4 | 4 | 0 | 1 | 2 | 3 | 9 | 5 | 6 | 7 | 8 |
| 5 | 5 | 9 | 8 | 7 | 6 | 0 | 4 | 3 | 2 | 1 |
| 6 | 6 | 5 | 9 | 8 | 7 | 1 | 0 | 4 | 3 | 2 |
| 7 | 7 | 6 | 5 | 9 | 8 | 2 | 1 | 0 | 4 | 3 |
| 8 | 8 | 7 | 6 | 5 | 9 | 3 | 2 | 1 | 0 | 4 |
| 9 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

.

Die Diedergruppe $D_5$ eignet sich hervorragend für den vorliegenden Zweck, da sie genau die Dezimalziffern als Elemente besitzt.

Die Nummer $a_1a_2\ldots a_{11}$ einer deutschen Banknote besteht aus 11 Zeichen. Die ersten beiden und die zehnte sind hierbei Buchstaben, welche nach der Ersetzung

| $A$ | $D$ | $G$ | $K$ | $L$ | $N$ | $S$ | $U$ | $Y$ | $Z$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

wie die restlichen Zeichen ebenfalls durch Ziffern dargestellt werden.

Die elfte und letzte Ziffer $a_{11}$ ist eine Prüfziffer. Um sie zu berechnen, benötigen wir als letzte Zutat der Geldscheinnumerierung die Permutation

$$T = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 5 & 7 & 6 & 2 & 8 & 3 & 0 & 9 & 4 \end{pmatrix}.$$

Die Beziehung

$$T(a_1) \star T^2(a_2) \star \cdots \star T^{10}(a_{10}) \star a_{11} = 0$$

definiert die Prüfziffer $a_{11}$ eindeutig, weil $D_5$ eine Gruppe ist.

**Sitzung 5.3** Wir programmieren dieses Verfahren. Zunächst erklären wir die Verknüpfung $\star$ der Diedergruppe:

```
In[1]:= Star[j_,k_] := Mod[j + k,5]/; j < 5&&j ≥ 0&&k < 5&&k ≥ 0
        Star[j_,k_] := Mod[5 + j – k,5]/; j < 10&&j ≥ 5&&k < 10&&k ≥ 5
        Star[j_,k_] := Mod[5 + j – k,5] + 5/; j < 10&&j ≥ 5&&k < 5&&k ≥ 0
        Star[j_,k_] := Mod[j + k,5] + 5/; j < 5&&j ≥ 0&&k < 10&&k ≥ 5
```

Nun können wir wieder die Verknüpfungstafel erzeugen:

```
In[2]:= Table[Star[j,k],{j,0,9},{k,0,9}]
```

$$\text{Out[2]}= \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 0 & 6 & 7 & 8 & 9 & 5 \\ 2 & 3 & 4 & 0 & 1 & 7 & 8 & 9 & 5 & 6 \\ 3 & 4 & 0 & 1 & 2 & 8 & 9 & 5 & 6 & 7 \\ 4 & 0 & 1 & 2 & 3 & 9 & 5 & 6 & 7 & 8 \\ 5 & 9 & 8 & 7 & 6 & 0 & 4 & 3 & 2 & 1 \\ 6 & 5 & 9 & 8 & 7 & 1 & 0 & 4 & 3 & 2 \\ 7 & 6 & 5 & 9 & 8 & 2 & 1 & 0 & 4 & 3 \\ 8 & 7 & 6 & 5 & 9 & 3 & 2 & 1 & 0 & 4 \\ 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{pmatrix}$$

Die Permutation $T$ läßt sich auf einfache Weise erklären:

```
In[3]:= T[0] = 1; T[1] = 5; T[2] = 7; T[3] = 6; T[4] = 2;
        T[5] = 8; T[6] = 3; T[7] = 0; T[8] = 9; T[9] = 4;
```

Damit haben wir die Ingredienzien zur Banknotenüberprüfung:

```
In[4]:= CheckBanknote[string_] := Module[{liste,test,k},
            liste = ToCharacterCode[string];
            Do[Which[liste[[k]] == 65,liste[[k]] = 0,liste[[k]] == 68,
                liste[[k]] = 1,liste[[k]] == 71,liste[[k]] = 2,liste[[k]] == 75,
                liste[[k]] = 3,liste[[k]] == 76,liste[[k]] = 4,liste[[k]] == 78,
                liste[[k]] = 5,liste[[k]] == 83,liste[[k]] = 6,liste[[k]] == 85,
                liste[[k]] = 7,liste[[k]] == 89,liste[[k]] = 8,liste[[k]] == 90,
                liste[[k]] = 9,True,liste[[k]] = liste[[k]] - 48],
              {k,Length[liste]}]; test = liste[[11]];
            Do[test = Star[Nest[T,liste[[k]],k],test],{k,10,1,-1}];
            If[test === 0,"Banknotennummer o.k.","Banknotennummer nicht o.k."]]
```

Wir überprüfen nun einige Banknoten:

```
In[5]:= CheckBanknote["GL7008058Z5"]
```
Out[5]= Banknotennummer o.k.

```
In[6]:= CheckBanknote["GL7008058Z0"]
```
Out[6]= Banknotennummer nicht o.k.

```
In[7]:= CheckBanknote["AA6186305Z2"]
```
Out[7]= Banknotennummer o.k.

```
In[8]:= CheckBanknote["AA6196305Z2"]
```
Out[8]= Banknotennummer nicht o.k.

Da für die Permutation $T$ die Beziehung

$$x \star T(y) \neq y \star T(x) \qquad \text{für alle } x \neq y$$

gilt:

```
In[9]:= Table[If[Not[Star[x,T[y]] == Star[y,T[x]]],".",False],
          {x,0,9},{y,0,9}]
```

Out[9]=

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| False | . | . | . | . | . | . | . | . | . |
| . | False | . | . | . | . | . | . | . | . |
| . | . | False | . | . | . | . | . | . | . |
| . | . | . | False | . | . | . | . | . | . |
| . | . | . | . | False | . | . | . | . | . |
| . | . | . | . | . | False | . | . | . | . |
| . | . | . | . | . | . | False | . | . | . |
| . | . | . | . | . | . | . | False | . | . |
| . | . | . | . | . | . | . | . | False | . |
| . | . | . | . | . | . | . | . | . | False |

werden von der Codierung alle Einzelfehler, aber auch das Vertauschen zweier Ziffern, erkannt (s. [Sch1991], S. 58 ff.). □

## 5.4 Fehlerkorrigierende Codes

In diesem Abschnitt stellen wir ein einfaches Beispiel eines fehlerkorrigierenden Codes vor, eine Variante des sogenannten *Reed-Solomon-Codes.*

**Sitzung 5.4** Wir wollen ein Wort so mit redundanter Information versehen, daß wir *einen* Übertragungsfehler korrigieren können. Als Beispiel wählen wir das Wort `"WORT"`.

Zunächst werden die Buchstaben durch ihre Nummern im Alphabet ersetzt, also $W \mapsto 23$, $O \mapsto 15$, $R \mapsto 18$ und $T \mapsto 20$. Diese Ersetzung lassen wir *Mathematica* durchführen:

```
In[1]:= Digitalisiere[wort_] := ToCharacterCode[wort] - 64
```

```
In[2]:= liste = Digitalisiere["WORT"]
Out[2]= {23,15,18,20}
```

Die Rücktransformation wird von

```
In[3]:= Verbalisiere[liste_] := FromCharacterCode[liste + 64]
```

erledigt:

```
In[4]:= Verbalisiere[liste]
Out[4]= WORT
```

Die betrachtete Ersetzung liefert ein $n$-tupel von Zahlen $a_k$, deren Index wir bei $k = 2$ starten lassen:

```
In[5]:= a[2] = 23; a[3] = 15; a[4] = 18; a[5] = 20;
```

Nun ergänzen wir die Zahlen $a_k$ ($k = 2,\ldots,n+1$) durch die beiden Werte $a_0$ und $a_1$ derart, daß diese die Gleichungen

$$\sum_{k=0}^{n+1} a_k \equiv 0 \pmod{31} \tag{5.1}$$

und

$$\sum_{k=0}^{n+1} k \cdot a_k \equiv 0 \pmod{31} \tag{5.2}$$

erfüllen. Die Gleichung (5.2) liefert zunächst eine eindeutige Lösung $a_1 \in \mathbb{Z}_{31}$ und (5.1) liefert schließlich eine eindeutige Lösung $a_0 \in \mathbb{Z}_{31}$. Hierbei ist die Wahl von $p = 31$ willkürlich und nur von der Mindestanzahl von 26 Buchstaben geleitet.[12]

Die folgende Rechnung löst das angegebene Gleichungssystem:

```
In[6]:= Solve[{Sum[a[k],{k,0,5}] == 0,
            Sum[k a[k],{k,0,5}] == 0,Modulus == 31},{a[0],a[1]}]
Out[6]= {{Modulus → 31,a(0) → 1,a(1) → 16}}
```

Die gesamte Codierung wird (für eine beliebige Primzahl $p \in \mathbb{P}$) von der Funktion `Reed-Solomon` durchgeführt:

```
In[7]:= Clear[ReedSolomon]
        ReedSolomon[wort_,p_] := Module[{liste,len,a0,a1,k,sol},
            liste = Digitalisiere[wort];
            len = Length[liste];
            sol = Solve[{a0 + a1 + Apply[Plus,liste] == 0,
                  a1 + Sum[(k + 1) * liste[[k]],{k,len}] == 0,Modulus == p},{a0,a1}];
            liste = Prepend[Prepend[liste,a1],a0]/.sol[[1]];
            Verbalisiere[liste]
          ]
```

Für unser Beispiel erhalten wir:

```
In[8]:= ReedSolomon["WORT",31]
Out[8]= APWORT
```

Nun nehmen wir an, bei der Übertragung habe es einen Fehler gegeben, d. h., ein Zeichen sei falsch übertragen worden. Konkret sei in unserem Beipiel `APWIRT` übertragen worden:

```
In[9]:= wort = "APWIRT"
Out[9]= APWIRT
```

Wir wollen nun untersuchen, ob sich hinter `APWIRT` wirklich der „Wirt" verbirgt. Hierzu konvertieren wir die Zeichenfolge wieder in die entsprechenden Buchstabennummern:

[12] Zur Dekodierung benötigen wir – wie wir sehen werden – einen Körper. Jede andere Primzahl könnte also ebenfalls Verwendung finden. In der Praxis werden Zweierpotenzen verwendet. Dies erfordert allerdings das Rechnen in einem Galoisfeld, s. Abschnitt 7.4.

```
In[10]:= liste = Digitalisiere[wort]
Out[10]= {1,16,23,9,18,20}
```

Nun überprüfen wir die Summen

$$e := \sum_{k=0}^{5} a_k \pmod{31} \qquad \text{und} \qquad s := \sum_{k=0}^{5} k\,a_k \pmod{31} \ : \tag{5.3}$$

```
In[11]:= e = Mod[Apply[Plus, liste], 31]
Out[11]= 25
```

```
In[12]:= s = Mod[Sum[(k - 1) * liste[[k]], {k, 1, 6}], 31]
Out[12]= 13
```

In unserem Fall sind also $e = 25$ und $s = 13$. Wegen $e \neq 0$ ist mindestens ein Fehler bei der Übertragung aufgetreten. Wir nehmen nun in der Folge an, es sei *genau ein* Fehler gewesen. Nur in diesem Fall können wir den Fehler korrigieren.

Wir wollen herausfinden, an welcher Position $x$ der Fehler eingetreten ist. Wir wissen, daß das Zeichen $a_x$ ersetzt wurde durch $a_x + e \pmod{31}$. Dieser Fehler produziert in der Summe $\sum_{k=0}^{5} k\,a_k \pmod{31}$ den Fehler $x \cdot e \pmod{31}$. Also ist

$$s \equiv x \cdot e \pmod{31}$$

bzw.

$$x \equiv s \cdot e^{-1} \pmod{31} \ .$$

In unserem Fall ist also wegen

```
In[13]:= x = Mod[s * PowerMod[e, -1, 31], 31]
Out[13]= 3
```

der Fehler an der dritten Position eingetreten. Nun können wir diesen rekonstruieren:

```
In[14]:= liste[[x + 1]] = Mod[liste[[x + 1]] - e, 31]
Out[14]= 15
```

und erhalten das korrigierte Wort

```
In[15]:= Verbalisiere[liste]
Out[15]= APWORT
```

Die folgende Funktion faßt diese Schritte zusammen:

```
In[16]:= Clear[InverseReedSolomon]
         InverseReedSolomon[wort_, p_] := Module[{liste, len, e, s, sol, x, k},
            liste = Digitalisiere[wort];
            len = Length[liste];
            e = Mod[Apply[Plus, liste], p];
            s = Mod[Sum[(k - 1) * liste[[k]], {k, 1, len}], p];
            x = Mod[s * PowerMod[e, -1, p], p];
            liste[[x + 1]] = Mod[liste[[x + 1]] - e, p];
            liste = Rest[Rest[liste]];
            Verbalisiere[liste]
           ]
```

Nun gelingt die Fehlerkorrektur in einem Schritt:

```
In[17]:= InverseReedSolomon["APWIRT", 31]
Out[17]= WORT
```

Schließlich betrachten wir ein weiteres Beispiel:

```
In[18]:= code = ReedSolomon["MATHEMATIK", 31]
Out[18]= JMMATHEMATIK
```

Wir bauen einen Fehler ein:

```
In[19]:= code = StringReplace[code, "H" → "S"]
Out[19]= JMMATSEMATIK
```

und reparieren diesen wieder:

```
In[20]:= InverseReedSolomon[code, 31]
Out[20]= MATHEMATIK
```

Bauen wir allerdings einen zweiten Fehler ein:

```
In[21]:= code = StringReplace[code, "K" → "J"]
Out[21]= JMMATSEMATIJ

In[22]:= InverseReedSolomon[code, 31]

Part :: "partw":Part 24 of {10, 13, 13, 1, 20, 19, 5, 13, 1, 20, 9, 10} does not
exist

Set :: "partw":Part 24 of {10, 13, 13, 1, 20, 19, 5, 13, 1, 20, 9, 10} does not
exist
Out[22]= MATSEMATIJ
```

so gelingt die Rekonstruktion erwartungsgemäß nicht. □

Wendet man den betrachten Reed-Solomon-Code auf vierbuchstabige Blöcke an, so werden diese also mit einer Redundanz von 2 zusätzlichen Buchstaben versehen. Man sagt, dieser Code habe die *Informationsrate* 4/6. Wird ein 6-tupel fehlerfrei oder mit nur einem Fehler übertragen, so können wir diesen beheben und den Text als fehlerfrei betrachten.

Wir nehmen nun an, ein Text mit 1 Million Buchstaben sei gegeben. Sind in unserem Text 2 von tausend Buchstaben fehlerhaft, so enthält dieser insgesamt also ca. 2000 Fehler. Wir codieren den Text nun mit unserem Reed-Solomon-Code. Danach hat dieser 1,5 Millionen Buchstaben. Unsere Buchstaben sind mit einer Wahrscheinlichkeit von $\frac{2}{1000} = 0{,}2\%$ fehlerhaft, also hat ein sechsbuchstabiges Wort *mehr als einen Fehler*[13] mit einer Wahrscheinlichkeit von 0,00006:

```
In[23]:= w = Sum[Binomial[6,k] * 0.002^k * 0.998^(6 - k),{k,2,6}]
Out[23]= 0.0000596807
```

Wir müssen nun also nur noch ca.

```
In[24]:= w * 1.5 * 10^6
Out[24]= 89.5211
```

Fehler in unserem Text erwarten.

Diese Quote kann aber noch weiter verbessert werden. Hierzu entwickeln wir beispielsweise einen Reed-Solomon-Code, bei welchem Blöcke $a_4a_5\ldots a_{11}$ der Länge 8 mit 4 redundanten Zeichen $a_0a_1a_2a_3$ versehen werden, welche den Bedingungen

$$\begin{aligned} a_0 + a_1 + a_2 + \cdots + a_{11} &\equiv 0 \pmod{31} \\ a_1 + 2a_2 + \cdots + 11a_{11} &\equiv 0 \pmod{31} \\ a_1 + 4a_2 + \cdots + 11^2a_{11} &\equiv 0 \pmod{31} \\ a_1 + 8a_2 + \cdots + 11^3a_{11} &\equiv 0 \pmod{31} \end{aligned}$$

genügen. Dieser Code hat wieder eine Informationsrate von 4/6. Es zeigt sich aber, daß wir mit diesem Code bis zu 2 Fehler korrigieren können, s. Übungsaufgabe 5.12! Man nennt dies einen 2-fehlerkorrigierenden Code. Die Wahrscheinlichkeit, daß ein Wort der Länge 12 mehr als 2 Fehler besitzt, ist aber nur gleich

```
In[25]:= w = Sum[Binomial[12,k] * 0.002^k * 0.998^(12 - k),{k,3,12}]
Out[25]= 1.73639 × 10^-6
```

und der Erwartungwert beträgt bei diesem Verfahren nur noch

```
In[26]:= w * 1.5 * 10^6
```

[13] also 2,3,4,5 bzw. 6 Fehler. Wir bestimmen die gesuchte Wahrscheinlichkeit mit Hilfe der Binomialverteilung.

```
Out[26]= 2.60459
```

Fehler in unserem Dokument. Dies ist eine ziemlich beeindruckende Quote, wenn wir daran denken, daß unser Dokument ohne Fehlerkorrektur 2000 Fehler besaß.

Eine Musik-CD ist selten fehlerfrei. Die Anzahl der Fehler kann hier durchaus in den Hunderttausenden liegen. Damit die Musik dennoch in vernünftiger Qualität abgespielt werden kann, wird hier ebenfalls ein fehlerkorrigierender Code (mit einer Informationsrate von $3/4$) verwendet. Da die Fehler bei einer CD nicht zufällig, sondern typischerweise in Clustern auftreten (z. B., wenn die CD zerkratzt ist), findet ein spezieller fehlerkorrigierender Code Verwendung.

## 5.5 Asymmetrische Verschlüsselungsverfahren

Während wir im Rahmen der Codierungstheorie versucht haben, Übertragungsfehler zu erkennen bzw. auszubessern, nehmen wir nun an, daß die fehlerfreie Übertragung sichergestellt ist. Das Ziel dieses Abschnitts ist es, neuere Verfahren zur *Verschlüsselung* von Nachrichten vorzustellen.

Wir gehen wieder davon aus, daß ein Sender über einen Übertragungskanal eine Nachricht an einen Empfänger schickt. Da ein Unbefugter die Übertragung gegebenenfalls abhören kann, wird die Nachricht *verschlüsselt* (*chiffriert*) und muß dann vom Empfänger wieder *entschlüsselt* (*dechiffriert*) werden.

$$\text{Nachricht/Klartext} \xrightarrow{\text{Verschlüsselung}} \text{Geheimtext/Kryptogramm}\,.$$

Die *Kryptographie* ist die Lehre vom Verschlüsseln, während die *Kryptoanalyse* die Lehre vom erfolgreichen Brechen einer Verschlüsselungsmethode ist. Die *Kryptologie* verbindet diese beiden Disziplinen.

Wichtige Anwendungen moderner kryptographischer Verfahren sind

- die Übertragung von Telefongesprächen über Satellit;
- Pay-TV;
- elektronische Bankgeschäfte;
- E-Commerce;
- Paßwortverfahren, z. B. bei Computerbetriebssystemen für Multiuserbetrieb;

- abhörsicheres Anmelden auf einem entfernten Computer über eine Datenleitung (secure shell);
- sichere e-mail [PGP].

**Beispiel 5.9** Man kann einen Text mit dem Caesarverfahren verschlüsseln, s. Beispiel 5.7, bei welchem jeder Buchstabe um 3 Plätze verschoben wird. Das *Verfahren* wäre hier also Caesar, der *Schlüssel* des Verfahrens die Zahl 3. Die Verschlüsselung ist einfach und bei Kenntnis des Schlüssels ist auch die Entschlüsselung einfach: Man wendet das Caesarverfahren mit dem Schlüssel $-3$ an.

Der Kryptoanalyst kann die verschlüsselte Nachricht aber auch ohne Kenntnis des Schlüssels entschlüsseln. Hat nämlich das zugrundeliegende Alphabet $m$ Buchstaben, so gibt es bei dem Verfahren Caesar nur $m$ verschiedene Schlüssel. Der Kryptoanalyst kann also *alle* Schlüssel durchprobieren und somit alle möglichen Klartexte erzeugen. Im allgemeinen macht dann nur einer davon Sinn. △

**Sitzung 5.5** Die *Mathematica*-Funktion

```
In[1]:= CaesarV["",n_] := ""

        CaesarV[x_String,n_] :=
          FromCharacterCode[Mod[ToCharacterCode[x] + n - 65,26] + 65]/;
            Length[ToCharacterCode[x]] === 1

        CaesarV[x_String,n_] :=
          Module[{length,first,rest},length = StringLength[x];
              first = StringTake[x,1];
              rest = StringDrop[x,1];
              StringJoin[CaesarV[first,n],CaesarV[rest,n]]]/;
            Length[ToCharacterCode[x]] > 1

        CaesarE[x_String,n_] := CaesarV[x,-n]
```

programmiert das Caesarverfahren über dem Alphabet $\{A,B,\ldots,Z\}$ rekursiv, wobei gemäß der ersten Definition von `CaesarV` Leerzeichen erhalten bleiben. Wir versuchen nun, das Verfahren zu brechen. Angenommen, wir erhalten das Kryptogramm

```
In[2]:= test = "QXIIT TCIOXUUTGC"
Out[2]= QXIIT TCIOXUUTGC
```

dann bilden wir einfach die Liste *aller* möglicher Klartexte[14]

[14] Man beachte, daß das Druckkommando `Print` zwar eine Ausgabe auf dem Bildschirm erzeugt, aber genau wie `Do` das Resultat `Null` hat.

```
In[3]:= Do[Print[CaesarE[test,n]],{n,1,26}]
PWHHS SBHNWTTSFB
OVGGR RAGMVSSREA
NUFFQ QZFLURRQDZ
MTEEP PYEKTQQPCY
LSDDO OXDJSPPOBX
KRCCN NWCIROONAW
JQBBM MVBHQNNMZV
IPAAL LUAGPMMLYU
HOZZK KTZFOLLKXT
GNYYJ JSYENKKJWS
FMXXI IRXDMJJIVR
ELWWH HQWCLIIHUQ
DKVVG GPVBKHHGTP
CJUUF FOUAJGGFSO
BITTE ENTZIFFERN
AHSSD DMSYHEEDQM
ZGRRC CLRXGDDCPL
YFQQB BKQWFCCBOK
XEPPA AJPVEBBANJ
WDOOZ ZIOUDAAZMI
VCNNY YHNTCZZYLH
UBMMX XGMSBYYXKG
TALLW WFLRAXXWJF
SZKKV VEKQZWWVIE
RYJJU UDJPYVVUHD
QXIIT TCIOXUUTGC
```

Ein kurzer Blick auf die Ergebnisliste zeigt uns, welches der richtige Schlüssel ist:

```
In[4]:= CaesarE[test,15]
Out[4]= BITTE ENTZIFFERN
```

Ganz offenbar gibt es bei diesem Verfahren zu wenige Schlüssel. □

Wir betrachten nun den allgemeinen Fall kryptographischer Verfahren. $N$ sei die zu verschlüsselnde Nachricht. Ferner sei $V$ das Verschlüsselungsverfahren und $s$ der zugehörige Schlüssel. Wir verschlüsseln also gemäß $C = V_s(N)$ und erhalten das Kryptogramm $C$. Der Empfänger benötigt das Entschlüsselungsverfahren $E$ und einen Schlüssel $t$. Mit diesem kann er dann das Kryptogramm wieder entschlüsseln: $N = E_t(C)$. Insgesamt gilt also die Gleichung

$$E_t(V_s(N)) = N \ .$$

Dies muß für alle Nachrichten gelten, mathematisch ist die Funktion $E_t$ also auf dem Bild von $V_s$ die Umkehrfunktion von $V_s$, $E_t = V_s^{-1}$, welche im Prinzip eindeutig bestimmt ist durch $V_s$. Bei vielen kryptographischen Verfahren ist dies nicht nur prinzipiell so, sondern man kann $E_t$ auch ganz leicht aus $V_s$ bestimmen. Beispielsweise ist beim Caesarverfahren $V_n$ (Verschiebung um $n$ Buchstaben) die Entschlüsselung gegeben durch $E_n = V_{-n}$. Zum Entschlüsseln ist also dasselbe Verfahren geeignet, und auch der Entschlüsselungsschlüssel $t$ läßt sich einfach aus dem Verschlüsselungsschlüssel $s$ bestimmen. Solche Verschlüsselungsverfahren werden *symmetrisch* genannt.

Während das Caesarverfahren offensichtlich ein unsicheres symmetrisches Verschlüsselungsverfahren ist, gibt es durchaus brauchbare und bewährte symmetrische Verschlüsselungsverfahren, welche allerdings erheblich komplizierter sind. Ein bewährtes, vollständig publiziertes symmetrisches Verschlüsselungsverfahren mit $E_t = V_s$, welches also (einschließlich des Schlüsselpaars) vollständig symmetrisch ist, ist das DES-Verfahren, s. z. B. [Buc1999], Kapitel 5. Das DES-Verfahren ist bislang noch nicht gebrochen worden, aber die Schlüssellänge muß vergrößert werden, um die zukünftige Sicherheit zu gewährleisten. Hierauf wollen wir aber hier nicht näher eingehen.

Wir wollen uns in der Folge mit *asymmetrischen Verschlüsselungsverfahren* beschäftigen. Natürlich erwarten wir von jedem guten Verschlüsselungsverfahren, daß Verschlüsselung $V_s$ und Entschlüsselung $E_t$ schnell vonstatten gehen. Das Verfahren soll also *effizient* sein. Ferner soll es möglichst nicht zu brechen sein (*Sicherheit*). Von einem asymmetrischen Verfahren fordern wir noch folgende zusätzliche Eigenschaften:

1. Verschlüsselungsverfahren $V$ und Entschlüsselungsverfahren $E$ sind allen Teilnehmern bekannt.

2. Ferner wird der Verschlüsselungsschlüssel $s$ vom Empfänger *öffentlich* gemacht (z. B. im Internet oder in einem öffentlichen Schlüsselbuch) und kann von jedem potentiellen Sender zur Verschlüsselung verwendet werden. Den zugehörigen Entschlüsselungsschlüssel $t$ hält der Empfänger aber geheim. $s$ heißt der *öffentliche Schlüssel* (public key) und $t$ heißt der *private Schlüssel* (private key) des Empfängers. Dies manifestiert die Asymmetrie des Verfahrens.

3. Die Kenntnis der Verschlüsselungsfunktion $V_s$ läßt die Berechnung der Entschlüsselungsfunktion $E_t$ nicht mit vertretbarem Aufwand zu. Insbesondere läßt sich also $t$ nicht aus $s$ bestimmen.

Wegen (2) heißen asymmetrische Verfahren auch *Public-Key-Verfahren*. Die Bedingung (3) ist wegen (2) unbedingt nötig, damit das Verfahren sicher ist.

Eines der Hauptprobleme konventioneller Geheimschriften ist die *Schlüsselübergabe*. Bei der Übergabe des Schlüssels könnte ein Agent in den Besitz des Geheimschlüssels kommen. Jede Verschlüsselung wäre danach völlig nutzlos!

Anders beim Public-Key-Verfahren: Dadurch, daß die Verschlüsselungsschlüssel öffentlich gemacht werden, ist eine Schlüsselübergabe nicht erforderlich. Will nun Anna eine Nachricht an Barbara schicken, so verschlüsselt sie die Nachricht mit dem öffentlichen Schlüssel $s_B$ von Barbara und schickt die Nachricht an Barbara. Barbara entschlüsselt die Nachricht mit ihrem privaten Schlüssel $t_B$. Sobald die Nachricht verschlüsselt wurde, kann sie nur noch mit dem privaten Schlüssel von Barbara entschlüsselt werden. Auch Anna kann sie zu diesem Zeitpunkt nicht mehr entschlüsseln!

Falls bei einem Public-Key-Verfahren $E = V$ gilt, kann es auch zur *Benutzerauthentifikation* verwendet werden: Anna will Barbara eine Nachricht schicken, bei der Barbara sicher sein kann, daß sie von Anna stammt. Also erzeugt sie mit ihrem privaten Schlüssel $t_A$ eine Unterschrift unter der Nachricht ($U = V_{t_A}(n)$) und schickt sie an Barbara. Dann kann Barbara mit dem öffentlichen Schlüssel $s_A$ von Anna verifizieren (ist $E_{s_A}(U) = n$?), daß die Nachricht von Anna kam. Wenn diese *digitale Signatur* geschickt durchgeführt wird, indem die Unterschrift in geeigneter Weise an die Nachricht gekoppelt wird, kann man sogar erreichen, daß der Empfänger sicher sein kann, daß die Nachricht *unverfälscht* angekommen ist (*Nachrichtenintegrität*). Diese Kopplung wird dadurch realisiert, $n$ in geeigneter Weise abhängig von $N$ zu wählen. Zur Realisierung einer digitalen Signatur verwendet man zusätzlich eine sogenannte *Hashfunktion*, s. [Buc1999], Kapitel 10.

Bevor wir mit dem RSA-Verfahren [RSA1978] ein asymmetrisches Verfahren vorstellen wollen, besprechen wir die *Diffie-Hellman-Schlüsselvereinbarung* [DH1976], welche es zwei Teilnehmern ermöglicht, einen gemeinsamen Schlüssel zu vereinbaren, ohne diesen über einen gegebenenfalls unsicheren Kanal senden zu müssen. Auf diese Weise kann man also die Schlüsselübergabe auf sichere Weise bewerkstelligen, ohne Gefahr zu laufen, daß der gemeinsame Schlüssel bekannt wird. Der gemeinsame Schlüssel bei der Diffie-Hellman-Schlüsselvereinbarung ist eine natürliche Zahl.

Anna und Barbara vereinbaren eine natürliche Zahl $g \in \mathbb{N}$ und eine Primzahl $p \in \mathbb{P}$. Diese sind nicht geheim und können der Allgemeinheit zugängig gemacht werden. Dann wählt Anna eine zufällige natürliche Zahl $a < p$ und berechnet $\alpha := \mathrm{mod}(g^a, p)$. Auch Barbara wählt eine zufällige natürliche Zahl $b < p$ und berechnet $\beta := \mathrm{mod}(g^b, p)$. Als nächstes werden die beiden Zahlen $\alpha$ und $\beta$

ausgetauscht. Anna berechnet schließlich $s = \mathrm{mod}(\beta^a, p)$, und Barbara berechnet $t = \mathrm{mod}(\alpha^b, p)$. Wegen

$$s \equiv \beta^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv \alpha^b \equiv t \pmod{p}$$

ist $s = t$, d. h. die beiden berechneten Schlüssel stimmen überein. Das Verfahren ist also *durchführbar*. Wir müssen uns nun überlegen, warum ein Angreifer keine Chance hat, $s$ zu bestimmen. Der einzige Angriff könnte darin bestehen, beim Austausch von $\alpha$ und $\beta$ diese beiden Zahlen abzuhören. Wir können ferner davon ausgehen, daß ein Angreifer $g$ und $p$ kennt. Auf der anderen Seite haben Anna und Barbara $a$ und $b$ geheim gehalten, so daß diese dem Angreifer unbekannt sind. Um nun beispielsweise $t \equiv \alpha^b \pmod{p}$ aus $\alpha$ zu bestimmen, müßte der Angreifer $b$ bestimmen und somit die Gleichung

$$\beta \equiv g^b \pmod{p}$$

nach $b$ auflösen. Dies ist aber, wie wir gesehen haben, praktisch unmöglich: Die Bestimmung des modularen Logarithmus ist in vernünftiger Zeit[15] unmöglich, falls man nur $g$ und $p$ groß genug gewählt hat. Dafür werden Anna und Barbara aber sorgen.

Daß die Diffie-Hellman-Schlüsselvereinbarung gelingt, liegt also an der Tastache, daß die modulare Exponentialfunktion eine *Einwegfunktion* ist: Die Berechnung von $y = g^x \pmod{p}$ ist effizient möglich, während die Umkehrfunktion $x = \log_g y \pmod{p}$ praktisch unberechenbar ist. Einwegfunktionen spielen auch eine wichtige Rolle bei asymmetrischen kryptographischen Verfahren.

Das bekannteste asymmetrische kryptographische Verfahren ist das RSA-Verfahren, bei welchem ebenfalls die modulare Exponentialfunktion geschickt ausgenutzt wird. Es wird hierbei o. B. d. A. angenommen, daß die Nachricht $N$ als natürliche Zahl vorliegt. Es folgt das *kryptographische Protokoll* des RSA-Verfahrens:

1. Barbara bestimmt

   (a) zwei mindestens (dezimal) 100-stellige zufällig ausgewählte Primzahlen $p$ und $q$.

   (b) $n = p \cdot q$.[16]

   (c) $\varphi = (p-1) \cdot (q-1)$.

   (d) eine zufällige natürliche Zahl $e < \varphi$ mit $\gcd(e,\varphi) = 1$. Das Paar $s_B = (e,n)$ ist Barbaras öffentlicher Schlüssel und wird publiziert.

[15] Falls man hierfür 100 Jahre benötigt, wird man mit dem Verfahren zufrieden sein.

[16] Eine 200-stellige Dezimalzahl hat eine Bitlänge von 665. Man führt heute das RSA-Verfahren meist mit $2^9 = 512$, mit $3 \cdot 2^8 = 768$ oder auch mit $2^{10} = 1024$ Bits durch.

(e) $d = e^{-1} \pmod{\varphi}$. Die Zahl $t_B = d$ konstituiert Barbaras privaten Schlüssel.[17]

2. Barbara löscht (aus Sicherheitsgründen) $p$,$q$ und $\varphi$.

3. Anna verschlüsselt ihre Nachricht $N$ mit der Rechnung

$$C = V_{s_B}(N) = \mathrm{mod}(N^e,n) .$$

Falls $N > n$ ist, wird die Nachricht in Blöcke zerlegt und es werden die einzelnen Blöcke verschlüsselt.

4. Barbara entschlüsselt (gegebenenfalls die einzelnen Blöcke) gemäß

$$E_{t_B}(C) = \mathrm{mod}(C^d,n) .$$

Zunächst wollen wir zeigen, daß das RSA-Verfahren durchführbar ist. Hierfür haben wir zu zeigen, daß (in $\mathbb{Z}_n$)

$$E_{t_B}(V_{s_B}(N)) = N$$

ist. Dies folgt aber mit dem kleinen Satz von Fermat: Wegen $d \cdot e \equiv 1 \pmod{\varphi}$ ist $d \cdot e = 1 + k\varphi$ für ein $k \in \mathbb{N}_0$. Also folgt

$$E_{t_B}(V_{s_B}(N)) \equiv (N^e)^d \equiv N^{de} \equiv N^{1+k\varphi} \equiv N^{1+k\cdot(p-1)\cdot(q-1)} \pmod{n} .$$

Wir rechnen zunächst modulo $p$ und zeigen mit Induktion:

$$N^{1+K(p-1)} \equiv N \pmod{p} \qquad (K \in \mathbb{N}_0) .$$

Für $K = 0$ ist dies klar, und der Induktionsschluß liefert mit dem Satz von Fermat[18]

$$N^{1+(K+1)(p-1)} \equiv N^p \cdot N^{K(p-1)} \equiv N \cdot N^{K(p-1)} \equiv N^{1+K(p-1)} \equiv N \pmod{p} .$$

Ebenso ergibt sich modulo $q$:

$$N^{1+K(q-1)} \equiv N \pmod{q} \qquad (K \in \mathbb{N}_0) .$$

Folglich gilt wie behauptet:

$$N^{de} \equiv N^{1+k(p-1)(q-1)} \equiv N \pmod{pq}$$

und damit (in $\mathbb{Z}_n$)

[17] Die Buchstaben $e$ und $d$ stammen aus dem Englischen und stehen für encryption und decryption.

[18] Mittels $N^{p-1} = 1$ kann man auch einen direkten Beweis führen. Dann muß aber der Fall $p \mid N$ gesondert behandelt werden.

$$E_{t_B}(V_{s_B}(N)) = N \ .$$

Das RSA-Verfahren entschlüsselt eine verschlüsselte Nachricht also wieder korrekt. Aber warum ist das RSA-Verfahren sicher? In Schritt (2) des Verfahrens hat Barbara die Zahlen $p$,$q$ und $\varphi$ vernichtet. Dies ist sinnvoll und notwendig, da die Kenntnis einer dieser Zahlen die Berechnung des geheimen Schlüssels $d$ ermöglicht, s. Übungsaufgabe 5.13.

Da $n$ veröffentlicht wird, kann man das RSA-Verfahren brechen, wenn es einem gelingt, $n$ in seine Primfaktoren $n = p \cdot q$ zu zerlegen. Während es relativ schnell möglich ist zu überprüfen, ob eine natürliche Zahl (mit hoher Wahrscheinlichkeit) prim ist oder nicht, s. Abschnitt 4.6, ist es mit heutigen Methoden völlig unmöglich, eine 200-stellige Dezimalzahl zu faktorisieren, s. Abschnitt 3.5. Dies klärt die Sicherheit des RSA-Verfahrens. Das Produkt $n = p \cdot q$ ist also die Einwegfunktion des RSA-Verfahrens.

Ist das RSA-Verfahren auch effizient? Dies ist offenbar gegeben, wenn man die modulare Exponentialfunktion mit dem in Abschnitt 4.4 auf S. 90 vorgestellten Divide-and-Conquer-Algorithmus oder einem ähnlichen Verfahren berechnet.

**Sitzung 5.6** Unter Verwendung von `NextPrime` erklären wir folgende Hilfsfunktionen:

```
In[1]:= ListToNumber[liste_] := First[liste] /; Length[liste] == 1
        ListToNumber[liste_] :=
          1000 * ListToNumber[Reverse[Rest[Reverse[liste]]]] + Last[liste]

        TextToNumber[string_] := ListToNumber[ToCharacterCode[string]]

        NumberToList[zahl_] := {zahl} /; zahl < 1000
        NumberToList[zahl_] :=
          Append[NumberToList[(zahl - Mod[zahl, 1000])/1000], Mod[zahl, 1000]]

        NumberToText[zahl_] := FromCharacterCode[NumberToList[zahl]]

        Verschlüssele[nachricht_] := PowerMod[TextToNumber[nachricht], e, n]

        Entschlüssele[zahl_] := NumberToText[PowerMod[zahl, d, n]]
```

und führen die Initialisierung des RSA-Algorithmus wie folgt durch:

```
In[2]:= InitialisiereRSA := Module[{},
          p = NextPrime[Random[Integer,{10^100,10^101}]];
          q = NextPrime[Random[Integer,{10^100,10^101}]];
          n = p * q;
          φ = (p - 1) * (q - 1);
          e = φ;
          While[Not[GCD[φ,e] === 1],
            e = NextPrime[Random[Integer,{10^40,10^50}]]];
          d = PowerMod[e, -1,φ];
        ]
```

Wir rufen `InitialisiereRSA` auf

```
In[3]:= InitialisiereRSA
```

und erhalten folgenden öffentlichen Schlüssel:

```
In[4]:= {e,n}
Out[4]= {40998536436223191577023612301326813733991005886029,
          17703595610997329310827973313390724504652355647464577868359302378840109843916609080910569408207600462317202347157855440179204731066636509323991051983269464484494147997662912857370089136440855363136589 43}
```

Leserinnen und Leser sind nun eingeladen, hieraus den privaten Schlüssel $d$ zu bestimmen! Nach dem heutigen Stand der Forschung sollte dies nicht möglich sein.

Wir erklären nun eine Nachricht:

```
In[5]:= nachricht = "Dies ist meine Nachricht"
Out[5]= Dies ist meine Nachricht
```

und verschlüsseln diese mit dem oben erklärten öffentlichen Schlüssel:

```
In[6]:= resultat = Verschlüssele[nachricht]
Out[6]= 24160272612134484489820088587582508709681010955869795833220655256957329813305233
          54379632413521457746614738914847705292200155149443891477261578028469551530686919008306393183097792351183945237530036541 37
```

Die Nachricht läßt sich leicht mit unserem privaten Schlüssel rekonstruieren:

```
In[7]:= Entschlüssele[resultat]
Out[7]= Dies ist meine Nachricht
```

Dies funktioniert allerdings nur, falls kein Übertragungsfehler aufgetreten ist. Schon der kleinste Fehler macht die Rekonstruktion unmöglich:[19]

```
In[8]:= Entschlüssele[resultat + 1]
```

Ein letztes Beispiel:

[19] Das Ergebnis ist i. a. keine darstellbare Nachricht.

```
In[9]:= resultat = Verschlüssele["Wie ist es mit dieser Meldung?"]
Out[9]= 9425449873527351303001047817793880907458301440951061115999004653238949046408437323900198199466310977742638786293206508160284998123063735563905870755153813806632249399477660471673138597185256223909089 68
In[10]:= Entschlüssele[resultat]
Out[10]= Wie ist es mit dieser Meldung?
```

Unser RSA-Code funktioniert nur, falls $N < 10^{200}$ ist. Ist die Nachricht länger, so müssen wir sie in Blöcke dieser Länge unterteilen. □

Abschließend sollte ich darauf hinweisen, daß es bislang unbewiesen ist, daß es Einwegfunktionen überhaupt gibt. Dies ist genau dann der Fall, falls $P \neq NP$ gilt [BDG1988], eine berühmte Vermutung aus der Komplexitätstheorie. Aber nach heutigem Wissensstand sind sowohl die modulare Exponentialfunktion als auch das Ausmultiplizieren natürlicher Zahlen Einwegfunktionen. In der Praxis haben sich das RSA-Verfahren wie auch andere asymmetrische kryptographische Verfahren bestens bewährt. Dieses Thema ist ein florierendes Forschungsgebiet der heutigen Mathematik.

## 5.6 Ergänzende Bemerkungen

Weiterführende Bücher zur Codierungstheorie sind [Jun1995] sowie [BFKWZ1998]. Die Darstellung der Reed-Solomon-Codes in Abschnitt 5.4 wurde aus [Lin1998] übernommen.

Eine schöne Einführung in die Kryptographie liefert [Kob2001], und ein elementares, sehr empfehlenswertes Textbuch ist [Buc1999]. Weitere asymmetrische Verfahren werden beispielsweise in [Sal1990] betrachtet. Kryptosysteme auf elliptischen Kurven scheinen besonders erfolgversprechend zu sein.

### ÜBUNGSAUFGABEN

**5.1 (Bitlänge)** *Zeigen Sie: Für ein Wort der Länge $n$ über einem Alphabet $\mathcal{A}$ mit $m$ Zeichen benötigt man $n \cdot \log_2 m$ Bits. Wie muß gerundet werden?*

**5.2 (Präfixcodes)** *Beweisen Sie konstruktiv: Präfixcodes lassen sich eindeutig decodieren.*

**5.3 (Präfixcodes)** *Programmieren Sie die Codierung und Decodierung des Präfixcodes* $C : \bigcup_{n\in\mathbb{N}} \{A,B,C\}^n \to \bigcup_{n\in\mathbb{N}} \{0,1\}$*, welcher gegeben ist durch*

$$A \mapsto 0\,, \quad B \mapsto 10\,, \quad C \mapsto 11\,.$$

*Wörter seien hierbei als Buchstabenliste gegeben.*

*Codieren und decodieren Sie* $\{A,B,C,A,A,C,C,B,A,B\}$ *sowie ein zufällig erzeugtes Wort der Länge 1000. Überprüfen Sie die Ergebnisse.*

**5.4** *Zeigen Sie, daß bei Einführung des Pausenzeichens* | *in das Morsealphabet* $\{\bullet,-\}^{20}$ *der Morsecode ein Präfixcode ist. Decodieren Sie den folgenden Text:*

• • | – • | – • • | • | • – • | • • – • | • – | – – – – | • – • • | • • | – – – – | • | – • |

• – – • | • – • | • • – – | • • – • | • • – | – • | – – • | • • – • | • • – – | • – • |

• • – • | • • – | – • | – • – | • – | – – | • – | – | • | • • – | • – • | • | • • • • | • – | – |

– • • | • | • – • | – • • • | • | • – – | • | • – • | – • • • | • | • – • | • – – • | • – • | • – | – • – |

– | • • | • • • | – – – – | • | • • – • | • | • – • | – | • • | – – • | – • – | • | • • | – | • | – • |

• • | – – | • • • • | – – – – • | • – • | • | – • | • • – | – • | – • • | – – • | • | – • • • | • | – • |

• • • – | – – – | – • | – – | – – – | • – • | • • • | • | – – • • | • | • • | – – – – | • | – • |

– • | • – | – – – – | – – • • | • • – | • – – | • | • • | • • • | • | – •

## 5.5 (Huffman-Code)

*(a) Gegeben sei die Häufigkeitsverteilung*

$$\begin{pmatrix} A & 0{,}4 \\ B & 0{,}12 \\ C & 0{,}2 \\ D & 0{,}18 \\ E & 0{,}1 \end{pmatrix}.$$

*Bilden Sie den Huffman-Code dieser Verteilung.*

*(b) Bilden Sie nun die Menge der Paare der Verteilung aus (a) und wenden Sie den Huffman-Algorithmus hierauf an. Erläutern Sie das Ergebnis.*

[20] Das Morsealphabet ist also die Menge {•, – , |}.

*(c) Berechnen Sie die Huffman-Codierung für die Häufigkeitsverteilung*

$$\begin{pmatrix} A & 0{,}08 \\ B & 0{,}16 \\ C & 0{,}08 \\ D & 0{,}06 \\ E & 0{,}1 \\ F & 0{,}07 \\ G & 0{,}07 \\ H & 0{,}05 \\ I & 0{,}13 \\ K & 0{,}03 \\ L & 0{,}01 \\ M & 0{,}02 \\ N & 0{,}07 \\ O & 0{,}03 \\ P & 0{,}03 \\ R & 0{,}01 \end{pmatrix}.$$

*(e) Verwenden Sie den Code zur Codierung und Decodierung des Texts* `HUFFMAN HAT IMMER RECHT.`[21]

**5.6 (Entropie)** *Die Entropie als Maß für den Informationsgehalt einer Häufigkeitsverteilung $V = (\mathcal{A},(p_k))$ des Alphabets $\mathcal{A}$ mit n Buchstaben ist gegeben durch*

$$H(V) = \sum_{k=1}^{n} p_k \log_2 \frac{1}{p_k} = -\sum_{k=1}^{n} p_k \log_2 p_k \,.$$

*Sie hängt also nicht von den einzelnen Buchstaben $A_k$, sondern nur von ihren Wahrscheinlichkeiten $p_k$ ab.*

*(a) Stellen Sie die Entropie eines binären Alphabets $\mathcal{A} = \{0,1\}$ mit Wahrscheinlichkeiten p und $q = 1-p$ in Abhängigkeit von p graphisch dar und bestimmen Sie $p_0$, für welches die Entropie maximal ist.*

*(b) Berechnen Sie die Entropie eines n-buchstabigen Alphabets $\mathcal{A}$ bei Gleichverteilung $H(\frac{1}{n},\frac{1}{n},\ldots,\frac{1}{n})$. Zeigen Sie, daß dies zugleich die maximale Entropie eines n-buchstabigen Alphabets $\mathcal{A}$ ist.*

*(c) Berechnen Sie die Entropie der trivialen Verteilung $H(1,0,\ldots,0)$. Erklären Sie!* Hinweis*: Beachten Sie* $\lim_{x\to 0} x \ln x$.

[21] Ignorieren Sie hierbei Leerzeichen.

*(d) Gegeben sei eine Häufigkeitsverteilung* `Alphabet` *wie in Sitzung 5.2. Erklären Sie eine Funktion* `NormierteEntropie[`*Alphabet*`]`*,*

*welche das Verhältnis von Entropie zu maximaler Entropie berechnet. Welchen Wertebereich hat die Funktion* `NormierteEntropie`*?*

*(e) Berechnen Sie die normierte Entropie für die Häufigkeitsverteilungen*

$$\textit{(i)}\begin{pmatrix} A & 0{,}9 \\ B & 0{,}05 \\ C & 0{,}025 \\ D & 0{,}013 \\ E & 0{,}012 \end{pmatrix};$$

$$\textit{(ii)}\begin{pmatrix} A & 0{,}3 \\ B & 0{,}25 \\ C & 0{,}2 \\ D & 0{,}15 \\ E & 0{,}1 \end{pmatrix}.$$

*Interpretieren Sie diese Ergebnisse in bezug auf die Kompressionsfähigkeit.*

**5.7** *Bei dem Codebaum auf S. 113 kann man den Buchstaben* `D` *nicht mehr vollständig sehen. Dies beruht auf einem Programmierfehler im* `Combinatorica`*-Package. Laden Sie die entsprechenden Funktionen mit* `??` *und beheben Sie diesen Bug durch Neudefinition der entsprechenden Funktion.*

**5.8 (ISBN-Prüfziffer)**

*(a) Schreiben Sie eine Prozedur* `ISBNPrüfziffer[`*n*`]`*, welche die Prüfziffer, also das letzte Zeichen der ISBN-Nummer, aus den ersten 9 Ziffern n berechnet.*

*(b) Berechnen Sie die Prüfziffern meiner Bücher*

*(i)* Mathematik mit DERIVE*: ISBN-Nummer beginnt mit 3-528-06549;*

*(ii)* Höhere Analysis mit DERIVE*: ISBN-Nummer beginnt mit 3-528-06594;*

*(iii)* DERIVE für den Mathematikunterricht*: ISBN-Nummer beginnt mit 3-528-06752;*

*(iv)* Hypergeometric Summation*: ISBN-Nummer beginnt mit 3-528-06950;*

*(v)* Die reellen Zahlen als Fundament und Baustein der Analysis*: ISBN-Nummer beginnt mit 3-486-24455;*

*sowie dreier beliebiger anderer Bücher.*

*(c) Schreiben Sie eine Prozedur* `CheckISBNPrüfziffer[n]`*, welche testet, ob eine vollständige ISBN-Nummer n korrekt ist. Eine vollständige ISBN-Nummer ist hierbei entweder eine 10-stellige Zahl oder eine 9-stellige Zahl zusammen mit dem Abschlußzeichen X.*

*(d) Testen Sie die Prozedur an obigen Beispielen sowie an 10 zufälligen 10-stelligen Dezimalzahlen.*

**5.9 (EAN-Prüfziffer)** *Schreiben Sie eine Prozedur* `EANPrüfziffer[n]`*, welche die Prüfziffer aus den ersten 12 Ziffern n berechnet. Berechnen Sie die Prüfziffer von 402570000102.*

**5.10** *Beweisen Sie die Gruppeneigenschaften der Diedergruppe $D_5$ mit Mathematica.*

**5.11 (Reed-Solomon-Code)** *Testen Sie die Funktionen* `ReedSolomon` *und* `InverseReedSolomon` *an 10 Beispielen, bei denen Sie jeweils an einer zufällig ausgewählten Stelle einen zufälligen Fehler eingebaut haben!*

**5.12 (2-fehlerkorrigierender Code)** *Programmieren Sie den am Ende des Abschnitts 5.4 erläuterten 2-fehlerkorrigierenden Code und testen Sie ihn auf geeignete Weise.*

**5.13 (RSA-Verfahren)** *Zeigen Sie, daß die Kenntnis einer der Zahlen p, q oder $\varphi$ die unkomplizierte Berechnung des privaten Schlüssels d ermöglicht.*

**5.14 (RSA-Verfahren** [Wie1999]–[Wie2000]**)**

*(a) Das RSA-Verfahren hat leider* Fixpunkte*, d. h., es gibt Texte, deren Kryptogramm mit dem Original übereinstimmt. Die Wahrscheinlichkeit hierfür ist allerdings bei genügend großem n sehr gering.*

*Zeigen Sie, daß bei der „ungeschickten" Wahl $e = 1 + \operatorname{lcm}(p-1, q-1)$* jede Verschlüsselung *einen Fixpunkt liefert.*

*(b) Zeigen Sie, daß das RSA-Verfahren korrekt bleibt, wenn wir den Modul $\varphi$ als $\varphi = \operatorname{lcm}(p-1, q-1)$ erklären. Begründen Sie, warum dies eine bessere Wahl ist als die im Text angegebene.*

*(c) Zeigen Sie, daß der Defekt aus (a) auch auftritt, falls man den Modul $\varphi = \operatorname{lcm}(p-1, q-1)$ gemäß (b) erklärt. Warum ist dies in diesem Fall dennoch nicht schlimm?*