Mathematik für die Informatik II

Numerik und Diskrete Mathematik

Goethe-Universität Frankfurt am Main

Sommersemester 2018

Dr. Samuel Hetterich

9. Mai 2018

Inhaltsverzeichnis

Vorwort

Dieses Skript ist Grundlage der Vorlesung “Diskrete Mathematik und Numerik - Mathematik für die Informatik II” gehalten von Dr. Samuel Hetterich im Sommersemester 2017 an der Goethe-Universität in Frankfurt am Main.

Das Skript wird im Laufe des Semesters entwickelt - die entsprechenden Abschnitte sollten aber in ihrer endgültigen Form jeweils vor den einzelnen Vorlesungen zur Verfügung stehen. Bei Anmerkungen, Kritik und Korrekturvorschlägen zögern Sie bitte nicht, sich an Dr. Samuel Hetterich (hetterich@math.uni-frankfurt.de) zu wenden.

Teile des vorliegenden Manuskripts sind aus den Skripten zu der gleichen Vorlesung vorangegangener Semester von Herrn Dr. Hartwig Bosse übernommen.

Numerische Differentialgleichungen.

Nummerische Integrale.

Optimierungsprobleme Gradient descent Lagrange Multiplikatoren.

1 Grundlagen

Dieses Kapitel enthält Grundlagen aus der Vorlesung “Mathe für die Informatik I” und richtet sich an all jene, die diese Vorlesung noch nicht gehört haben.

Wir beginnen mit einigen sehr grundlegenden mathematischen Konzepten, die zum Teil schon aus der Schulmathematik bekannt sein sollten und in der Vorlesung häufig als “Handwerkszeug” in den unterschiedlichen Kontexten auftauchen werden.

1.1. Mathematische Logik: Aussagen und Logische Quantoren

Unter einer mathematischen Aussage versteht man eine mathematische Formel, oder eine formal-logische Aussage, der ein Wahrheitswert “wahr” oder “falsch” zugewiesen werden kann.

- Der Ausdruck “$x^2 - 2x + 1$” ist *keine* mathematische Aussage sondern nur ein mathematischer Term.
- Der Ausdruck “$x^2 - 2x + 1 = 0$” ist eine mathematische Aussage (die je nach Wert von x wahr oder falsch ist).
- Der Ausdruck “$1 = 0$” ist eine mathematische Aussage, die falsch ist.
- Der Ausdruck “4 ist eine Quadratzahl” ist eine mathematische Aussage, die richtig ist.
- Die Goldbach-Vermutung “Jede gerade natürliche Zahl größer als 2 kann als Summe zweier Primzahlen geschrieben werden.“ ist eine mathematische Aussage von der bisher nicht klar ist, ob sie wahr oder falsch ist.

Wie Aussagen im “normalen” Leben, muss jede mathematische Aussage ein Verb enthalten. Diese Verben stecken oft in *logischen Quantoren* oder *logischen Operatoren*, die im Grunde Abkürzungen für Textbausteine sind. In den obigen Beispielen steckt das Verb an einigen Stellen in dem Operator “=”, den man als “. . . ist gleich . . .” liest.

In den folgenden Tabelle sind die von uns verwendeten logischen Operatoren und Quantoren aufgelistet.

- **Liste der verwendeten Operatoren:**

Symbol	Name	Zugehörige Formulierung	Beispiel
$\neg$	Negation	Es gilt nicht . . .	$\neg[3 = 4]$
$\vee$	Oder	Es gilt . . . oder . . .	$[n \geq 2] \vee [n \leq 2]$
$\dot{\vee}$	Exklusives Oder	Es gilt **entweder** . . . oder . . .	$[n \geq 2] \dot{\vee} [n \leq 2]$
$\wedge$	Und	Es gilt . . . und . . .	$[n \geq 2] \wedge [n \leq 2]$

- **Liste der verwendeten Quantoren:**

Symbol	Name	Zugehörige Formulierung	Beispiel
$\forall$	All-Quantor	Für alle . . .	$\forall n \in \mathbb{N} : n \geq 0$
$\exists$	Existenz-Quantor	Es existiert (mindestens) ein . . .	$\exists n \in \mathbb{N} : n \geq 5$
$\exists!$		Es existiert **genau** ein . . .	$\exists! n \in \mathbb{N} : n^2 = 25$
$\nexists$		Es existiert **kein** . . .	$\nexists n \in \mathbb{N} : n < 0$

Es gelten in gewissem Sinne “Rechenregeln” für mathematische Aussagen. Dabei spielen die Begriffe der ***Äquivalenz*** und der ***Implikation*** eine entscheidende Rolle, welche mathematische Aussagen in Relation setzen.

Definition 1.1.1.

- Eine mathematische Aussage $\mathcal{A}$ ***impliziert*** eine weitere mathematische Aussage $\mathcal{B}$, wenn aus der Wahrheit der Aussage $\mathcal{A}$ die Wahrheit der Aussage $\mathcal{B}$ folgt. Man schreibt dann

$$\mathcal{A} \Rightarrow \mathcal{B}.$$

- Zwei mathematische Aussagen $\mathcal{A}$ und $\mathcal{B}$ sind ***äquivalent***, wenn $\mathcal{A}$ die Aussage $\mathcal{B}$ impliziert und umgekehrt auch $\mathcal{B}$ die Aussage $\mathcal{A}$ impliziert. In diesem Fall schreibt man

$$\mathcal{A} \Leftrightarrow \mathcal{B}.$$

(Die “Formel”: *... genau ... dann ..., wenn ...* weist auf Äqivalenz in gesprochener Sprache hin.)

Beispiel 1.1.2.

- Es ist $n \geq 5 \Rightarrow n \geq 3$. Umgekehrt ist dies jedoch nicht der Fall.
- Ein Dreieck ist genau dann gleichseitig, wenn alle Seiten die gleiche Länge haben.

Bemerkung 1.1.3.

Interessanterweise sind die Implikationen $\mathcal{A} \Rightarrow \mathcal{B}$ und $\neg\mathcal{B} \Rightarrow \neg\mathcal{A}$ gleichbedeutend. Denn wenn $\mathcal{A}$ die Aussage $\mathcal{B}$ impliziert, dann kann $\mathcal{A}$ nicht wahr sein, wenn $\mathcal{B}$ nicht wahr ist. Ergo impliziert $\neg\mathcal{B}$ die Aussage $\neg\mathcal{A}$.

Bemerkung 1.1.4.

Im Fall der Äquivalenz sind die Aussagen entweder beide wahr oder beide falsch - sie sind gleichwertig. Daher erschließt sich der Name aus dem Lateinischen: *aequus* “gleich” und *valere* “wert sein”.

Eine schöne Veranschaulichung für den Unterschied zwischen Äquivalenz und Implikation ist diese Eselsbrücke, welche den Sachverhalt der Impliaktion veranschaulicht:

> *Wenn es geregnet hat, ist die Straße nass.*
> *Wenn die Straße nass ist, heißt das nicht zwangsläufig, dass es geregnet hat.*

$$\text{“Es hat geregnet.”} \Rightarrow \text{“Die Straße ist nass.”}$$

$$\text{“Die Straße ist nass.”} \not\Rightarrow \text{“Es hat geregnet.”}$$

Aber es ist nach Bemerkung 1.1.3

$$\neg(\text{“Die Straße ist nass.”}) \Rightarrow \neg(\text{“Es hat geregnet.”})$$

was umformuliert heißt

"Die Straße ist nicht nass." $\Rightarrow$ "Es hat nicht geregnet."

Wie angekündigt nun die "Rechenregeln" für mathematische Aussagen.

Lemma 1.1.5. Es seien $\mathcal{A}, \mathcal{B}, \mathcal{C}$ mathematische Aussagen. Dann gelten

$$\begin{aligned} \mathcal{A} \wedge \mathcal{B} &\Leftrightarrow \mathcal{B} \wedge \mathcal{A} \\ \mathcal{A} \vee \mathcal{B} &\Leftrightarrow \mathcal{B} \vee \mathcal{A} \end{aligned} \qquad \text{(Kommutativgesetze)}$$

$$\begin{aligned} [\mathcal{A} \wedge \mathcal{B}] \wedge \mathcal{C} &\Leftrightarrow \mathcal{A} \wedge [\mathcal{B} \wedge \mathcal{C}] \\ [\mathcal{A} \vee \mathcal{B}] \vee \mathcal{C} &\Leftrightarrow \mathcal{A} \vee [\mathcal{B} \vee \mathcal{C}] \end{aligned} \qquad \text{(Assoziativgesetze)}$$

$$\begin{aligned} [\mathcal{A} \wedge \mathcal{B}] \vee \mathcal{C} &\Leftrightarrow [\mathcal{A} \vee \mathcal{C}] \wedge [\mathcal{B} \vee \mathcal{C}] \\ [\mathcal{A} \vee \mathcal{B}] \wedge \mathcal{C} &\Leftrightarrow [\mathcal{A} \wedge \mathcal{C}] \vee [\mathcal{B} \wedge \mathcal{C}] \end{aligned} \qquad \text{(Distributivgesetze)}$$

Bezüglich des Negierens gelten die folgenden Regeln.

Lemma 1.1.6. Es seien $\mathcal{A}, \mathcal{B}$ mathematische Aussagen. Dann gelten

i. $\neg(\mathcal{A} \vee \mathcal{B}) = (\neg\mathcal{A}) \wedge (\neg\mathcal{B})$
ii. $\neg(\mathcal{A} \wedge \mathcal{B}) = (\neg\mathcal{A}) \vee (\neg\mathcal{B})$

Bemerkung 1.1.7.

Negiert man eine Aussage, die mit einem All- oder Existenzquantor beginnen, so taucht in der negierten Aussage stets der andere Quantor auf. Dies ist wichtig für den Beweis von Aussagen durch Widerspruch, hier wird in der einleitenden Widerspruchsannahme die Originalaussage negiert.

1.2. Mengen

Wir beginnen mit dem Begriff der Menge, welche an dieser Stelle aufgrund einiger Komplikationen in den Details nicht im strengen mathematischen Sinne sauber definiert werden kann. Für unsere Zwecke bedienen wir uns der (naiven) Mengendefinition von Georg Cantor (1845-1918), dem Begründer der Mengentheorie:

> *Eine Menge ist eine Zusammenfassung bestimmter wohlunterschiedener Objekte unserer Anschauung oder unseres Denkens, welche Elemente genannt werden, zu einem Ganzen.*

Diese sehr einleuchtende und der alltäglichen Verwendung des Begriffs der Menge sehr nahe Umschreibung führt allerdings bei näherer Untersuchung zu Komplikationen.

Exkurs 1.2.1 (Die Russelsche Antinomie)**.**

Definiert man Mengen als "Zusammenfassung unterscheidbarer Objekte" so ergibt sich das folgende Paradoxon:

Die Menge der Mengen, welche sich nicht selbst enthalten.

Gäbe es diese Menge, und nennen wir sie A, so stellt sich die Frage:

Enthät A sich selbst?

- Beantworten wir Frage (1.2.1) mit JA so ergibt sich ein Widerspruch:
  - Wir nehmen an A enthält die Menge A (weil wir die Frage (1.2.1) mit JA beantworten).
  - Damit ist A (als Menge) keine jener erlesenen Mengen, die wir unter dem Titel "Mengen die sich nicht selbst enthalten" in A versammelt haben.
  - D.h. A ist nicht dabei, ergo: A ist (doch) nicht in A enthalten.
  - Ein Widerspruch!
- Beantworten wir Frage (1.2.1) mit NEIN so ergibt sich ein Widerspruch:
  - Wir nehmen an A enthält die Menge A nicht (weil wir die Frage (1.2.1) mit NEIN beantworten).
  - Damit ist A (als Menge) eine jener erlesenen Mengen, die wir unter dem Titel "Mengen die sich nicht selbst enthalten" in A versammelt haben.
  - D.h. A ist dabei, ergo: A ist doch in A enthalten.
  - Ein Widerspruch!

Wir beginnen mit Konventionen und Definitionen bezüglich der Notation grundlegender Begriffe im Kontext von Mengen. Seien A und B Mengen.

Definition 1.2.2 (Mengendefinitionen und -notationen)**.**

- Mengen werden mit "$\{$" und "$\}$" den *Mengenklammern* geschrieben.
- Die Schreibweise $x \in A$ bedeutet, dass x ein ***Element*** der Menge A ist.
- Ferner bedeutet $A \subset B$ (bzw. $B \supset A$), dass A eine ***Teilmenge*** von B ist, d.h. jedes Element von A ist auch ein Element von B.
- Wir nennen die Mengen A und B ***gleich***, wenn sie die gleichen Elemente enthalten.
- Eine Teilmenge A von B heißt ***echt***, wenn A nicht gleich B ist.
- Mit $A \cup B$ bezeichnen wir die ***Vereinigung*** von A und B; die Menge aller Element, die in A oder in B enthalten sind.
- Außerdem ist $A \cap B$ der ***Durchschnitt*** von von A und B; die Menge aller Elemente, die in A und B enthalten sind.
- Mit $A \setminus B$, gesprochen **"A ohne B"**, bezeichnen wir die Menge aller Elemente von A, die nicht Element von B sind (auch ***Differenz*** genannt).
- Schließlich ist $A \times B$ die ***Produktmenge*** von A und B, d.h. die Menge aller geordneten Paare (x, y) mit $x \in A$ und $y \in B$ (auch *kartesisches Produkt* genannt). Siehe auch Beispiel 1.2.3.
- Eine Menge A heißt ***endlich***, wenn A nur endlich viele Elemente besitzt.
- Die Anzahl der Elemente einer endlichen Menge A wird als die ***Kardinalität*** von A bezeichnet, und mit $|A|$

notiert (auch ***Mächtigkeit*** genannt). Ist A nicht endlich so schreibt man $|A| = \infty$.

- Die ***leere Menge*** notiert mit $\emptyset$ ist diejenige Menge, die keine Elemente enthält.
- Für eine Menge A is die ***Potenzmenge*** $\mathcal{P}(A)$ die Menge aller Teilmengen von A inklusive der leeren Menge $\emptyset$. Siehe auch Beispiel 1.2.4.
- Eine Menge ist definiert, wenn angegeben ist, welche Elemente in ihr enthalten sind. Dies kann *deskriptiv* - durch Angabe einer definierenden Eigenschaft ($A := \{n \in \mathbb{N} : n \text{ ist gerade}\}$) - und *konstruktiv* - durch Aufzählung aller in ihr enthaltenen Elemente ($B := \{2, 4, 6, 8, 10\}$) - geschehen. Wenn bei Mengen mit unendlich vielen Elementen das Bildungsgesetz klar ist, können auch unendliche Aufzählungen verwendet werden ($A := \{2, 4, 6, 8, \ldots\}$).
- Es bezeichnet $\mathbb{N} = \{1, 2, 3, \ldots\}$ die Menge der ***natürlichen Zahlen***. Es bezeichnet $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ die Menge der natürlichen Zahlen mit der Null.
- Es bezeichnet $\mathbb{Z} = \{0, -1, 1, -2, 2, -3, 3, \ldots\}$ die Menge der ***ganzen Zahlen***.
- Es bezeichnet $\mathbb{Q}$ die Menge der ***rationalen*** und $\mathbb{R}$ die Menge der ***reellen Zahlen***.

Beispiel 1.2.3 (Produktmenge)**.**

Für $A = \{1, 2, 3\}$ und $B = \{3, 4\}$ ist

$$A \times B = \{(1, 3), (1, 4), (2, 3), (2, 4), (3, 3), (3, 4)\}.$$

Beispiel 1.2.4 (Potenzmenge)**.**

Für $A = \{1, 2, 3\}$ ist

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

In einem gewissen Sinne lässt sich mit Mengen und den ***Operatoren*** Durchschnitt und Vereinigung (Differenz und dem kartesischen Produkt) rechnen. Es gelten die folgenden "Rechenregeln".

Lemma 1.2.5. Für beliebige Mengen A, B und C gilt:

$$\begin{aligned} A \cap B &= B \cap A \\ A \cup B &= B \cup A \end{aligned} \qquad \text{(Kommutativgesetze)}$$

$$\begin{aligned} (A \cap B) \cap C &= A \cap (B \cap C) \\ (A \cup B) \cup C &= A \cup (B \cup C) \end{aligned} \qquad \text{(Assoziativgesetze)}$$

$$(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$$
$$(A \cup B) \cap C = (A \cap C) \cup (B \cap C) \qquad \text{(Distributivgesetze)}$$

Im Folgenden Beweis stehen die Symbole "$\subset$" und "$\supset$" für folgende Textüberschriften:

" $\subset$ " entspricht: "Wir Zeigen nun: linke Menge ist enthalten in rechter Menge".

" $\supset$ " entspricht: "Wir Zeigen nun: rechte Menge ist enthalten in linker Menge".

Diese Symbole sind also Abkürzungen und nicht als mathematische Symbole zu deuten.

Beweis. [Beweis von Lemma 1.2.5] Wir beweisen exemplarisch die erste der beiden in Lemma 1.2.5 als Distributivgesetz bezeichneten Gleichungen. Zu zeigen ist:

$$(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$$

Nach der Definition von Mengengleichheit müssen wir zeigen, dass die rechte menge in der linken un die linke in der rechten Menge enthalten ist.

"$\subset$": Es sei $x \in (A \cap B) \cup C$ beliebig gewählt. Für x gilt dann:

$$\begin{aligned} & x \in (A \cap B) \cup C \\ \Leftrightarrow \quad & [x \in (A \cap B) \vee x \in C] \\ \Leftrightarrow \quad & [(x \in A \wedge x \in B) \vee x \in C] \end{aligned} \tag{1.1}$$

Es gibt nun zwei Fälle:

Fall 1 $x \in C$**:** Es gilt demnach $x \in A \cup C$ und $x \in B \cup C$.
Fall 2 $x \notin C$**:** Es folgen aus (1.1) demnach sofort $x \in A$ und $x \in B$. Damit gilt auch $x \in A \cup C$ und $x \in B \cup C$.

In beiden Fällen gilt also $x \in A \cup C$ und $x \in B \cup C$ und damit $x \in (A \cup C) \cap (B \cup C)$.

Da x beliebig gewählt war gilt also allgemein für alle $x \in (A \cap B) \cup C$, dass

$$x \in (A \cap B) \cup C \qquad \Longrightarrow \qquad x \in (A \cup C) \cap (B \cup C).$$

Damit gilt nach der Defintiont von "$\subset$" also $(A \cap B) \cup C \subset (A \cup C) \cap (B \cup C)$.

"$\supset$": Es sei $x \in (A \cup C) \cap (B \cup C)$ beliebig gewählt. Für x gilt dann:

$$\begin{aligned} & x \in (A \cup C) && \cap && (B \cup C) \\ \Leftrightarrow \quad & [x \in (A \cup C) && \wedge && x \in (B \cup C)] \\ \Leftrightarrow \quad & [(x \in A \vee x \in C) && \wedge && (x \in B \vee x \in C)] \end{aligned} \tag{1.2}$$

Es gibt nun zwei Fälle:

Fall 1 $x \in C$**:** Es folgt demnach $x \in (A \cap B) \cup C$.
Fall 2 $x \notin C$**:** Es folgen aus (1.2) demnach sofort $x \in A$ und $x \in B$ und damit $x \in (A \cup C) \cap (B \cup C)$.

In beiden Fällen gilt also $x \in (A \cap B) \cup C$

Da x beliebig gewählt war, gilt also allgemein für alle $x \in (A \cup C) \cap (B \cup C)$, dass

$$x \in (A \cap B) \cup C \qquad \Leftarrow \qquad x \in (A \cup C) \cap (B \cup C).$$

Damit gilt nach der Definition von Teilmengen also $(A \cap B) \cup C \supset (A \cup C) \cap (B \cup C)$. □

Lemma 1.2.6. Sei A eine endlichen Menge mit Kardinalität n. Die Potenzmenge $\mathcal{P}(A)$ von A hat Kardinalität 2^n.

In der Mathematik lassen sich Aussagen häufig auf unterschiedliche Weisen zeigen. Für den Satz des Pythagoras sind beispielsweise mehrere hundert verschiedene Beweise bekannt. Der Satz des Pythagoras ist damit übrigens der meistbewiesene mathematische Satz. Für Lemma 1.2.6 ist ebenfalls mehr als ein Beweis bekannt. Zum einen kann man die Menge aller Teilmengen, also die Potenzmenge, mit einer zweiten Menge in Eins-zu-eins-Relation bringt. Der Trick besteht darin, dass dabei jedem Element aus der Potenzmenge genau ein Element aus der zweiten Menge und tatsächlich jedem Element aus der zweiten Menge auch ein Element der Potenzmenge zugeordnet wird. Folglich enthalten beide Mengen also genau gleich viele Elemente. Die zweite Menge stellt sich dann schlussendlich als leicht zu zählen heraus.

Neben diesem Beweis, der die im folgenden Abschnitt erinnerten Begriff im Zusammenhang von Abbildungen verwendet, existiert ein weiterer Beweis über eine Beweistechnik mit dem Namen ***vollständige Induktion***. Wir betrachten beide Beweise im Anschluss an Abschnitt 1.3.

1.3. Abbildungen

Wir starten mit grundlegenden Definitionen von Abbildungen.

Definition 1.3.1 (Abbildungen)**.** Eine ***Abbildung*** (oder auch ***Funktion***) $f : D \to B, x \mapsto f(x)$ bildet Werte aus dem ***Definitionsbereich*** D in den ***Bildbereich*** B ab. Jedem Element $x \in D$ wird durch f genau ein ***Bild*** $f(x) \in B$ zugeordnet. Gilt $f(x) = y$ für ein $y \in B$, so nennt man x das ***Urbild*** von y. Jedes $x \in D$ besitzt ein Bild $f(x)$, aber nicht jedes Element $y \in B$ muss ein Urbild besitzen.

Dies verallgemeinert man für eine Abbildung $f : D \to B$ und eine Teilmenge $Z \subset D$ ist

$$f(Z) = \{f(z) : z \in Z\}$$

die ***Bildmenge*** (manchmal auch einfach das Bild) von Z unter f. Umgekehrt bezeichnen wir für $C \subset B$ mit

$$f^{-1}(C) = \{x \in D : f(x) \in C\}$$

die ***Urbildmenge*** (manchmal auch einfach das Urbild) von C.

Abbildungen können drei ganz grundlegende Eigenschaften besitzen.

Definition 1.3.2 (Injektivität, Surjektivität, Bijektivität)**.** Eine Abbildung heißt

- ***injektiv***, wenn je zwei verschiedene $x, x' \in D$ auch verschiedene Bilder besitzen, d.h. wenn gilt:

$$x \neq x' \implies f(x) \neq f(x')$$

- ***surjektiv***, wenn jeder Bildpunkt $y \in B$ tatsächlich auch ein Urbild $x \in D$ besitzt mit $y = f(x)$, d.h. wenn gilt:

$$\forall\, y \in B\ \exists\, x \in D : f(x) = y$$

- ***bijektiv***, wenn f injektiv und surjektiv ist.

Bemerkung 1.3.3.

Injektiv: Die Definition für "injektiv" kann man sich anschaulich vorstellen als: Die Definitionsmenge D wird in die Bildmenge "injiziert", d.h. man findet für jedes $x \in D$ einen eigenen Funktionswert $y \in B$ vor, "der einmal x war".

Surjektiv: Eine surjektive Abbildung dagegen "deckt die ganze Bildmenge ab", jeder Bildpunkt wird bei einer surjektiven Abbildung auch tatsächlich angenommen. Dies ist nicht selbstverständlich: Das Wort "Bildmenge" klingt zwar wie "die Sammlung aller Bilder f (x)". Tatsächlich kann die Bildmenge aber auch einfach nur eine "grobe Schätzung" sein, wie im Beispiel $g : \mathbb{R} \to \mathbb{R}$ mit $g(x) := x^2$. Dabei sind die Werte von g stets nicht-negativ, d.h. man "erreicht" mit g nicht die ganze Bildmenge $\mathbb{R}$.

Bijektiv: Eine bijektive Abbildung stellt eine Eins-zu-Eins-Relation zwischen Definitionsmenge D und Bildmenge B her. D.h. jedes $x \in D$ hat zum einen "sein eigenes(!)" Bild $f(x) \in B$, aber auch jeder Punkt $y' \in B$ hat genau(!) ein Urbild $x' \in D$. Daraus folgt: Sind D und B endliche Mengen und sind sie über eine bijektive Abbildung $f : D \to B$ verknüpft, so haben D und B gleich viele Elemente. Der Begriff einer bijektiven Abbildung ist in der Mathematik also beim Zählen von Dingen von zentraler Bedeutung.

Es folgt eine weitere sehr umfangreiche Definition.

Definition 1.3.4. Falls f eine bijektive Abbildung ist, so hat für jedes $y \in B$ die Menge $f^{-1}(\{y\})$ genau ein Element $x \in D$ und wir schreiben einfach $x = f^{-1}(y)$. Die Abbildung $f^{-1} : B \to D, y \mapsto f^{-1}(y)$ ist in diesem Fall ebenfalls bijektiv und heißt die ***Umkehrabbildung*** von f.

Für eine Menge B und eine Zahl $k \in \mathbb{N}$ bezeichnen wir mit B^k die Menge aller Abbildungen $f : \{1, \ldots, k\} \to B$. Anstelle der Notation $f : D \to B, x \mapsto f(x)$ schreiben wir mitunter etwas lax $(f(x))_{x \in D}$. Diese Notation wird häufig verwendet, wenn $D = \{1, 2, 3, \ldots, k\}$ für eine Zahl $k \in \mathbb{N}$. Insbesondere schreiben wir die Elemente f der Menge B^k als $(f(1), \ldots, f(k))$; sie werden auch k-Tupel (und im Fall $k = 2$ Paare und im Fall $k = 3$ Tripel) genannt. Allgemeiner bezeichnen wir mit B^D die Menge aller Abbildungen $f : D \to B$. Ist $(A_i)_{i \in I}$ eine

Abbildung, die Elementen einer Menge I Teilmengen A_i einer Menge A zuordnet, so bezeichnet

$$\bigcup_{i\in I} A_i = \{x \in A : \text{ es gibt ein } i \in I \text{ mit } x \in A_i\}$$

die Vereinigung aller Mengen A_i. Analog ist

$$\bigcap_{i\in I} A_i = \{x \in A : \text{ für alle } i \in I \text{ gilt } x \in A_i\}$$

der Durchschnitt aller A_i.

Sei $f : A \to \mathbb{R}$ eine Abbildung von einer endlichen Menge $A \neq \emptyset$ in die reellen Zahlen. Dann existiert eine Bijektion $g : \{1, \ldots, k\} \to A$, wobei $k \in \mathbb{N}$. Wir definieren die ***Summe***

$$\sum_{a\in A} f(a) = f(g(1)) + f(g(2)) + \ldots + f(g(k))$$

und das ***Produkt***

$$\prod_{a\in A} f(a) = f(g(1)) \cdot f(g(2)) \cdots f(g(k)).$$

Falls A die leere Menge ist, interpretieren wir die Summe als 0 und das Produkt als 1.

Wir benötigen die Beweismethode der ***Induktion***. Die Grundlage des Induktionsprinzip ist folgende Tatsache.

Jede nicht-leere Menge natürlicher Zahlen enthält eine kleinste Zahl.

Aus dieser Tatsache folgt

Lemma 1.3.5 ("Induktionsprinzip"). Angenommen eine Menge $A \subset \mathbb{N}$ hat die beiden folgenden Eigenschaften.

i. $1 \in A$
ii. Wenn $1, \ldots, n \in A$, dann gilt auch $n + 1 \in A$.

Dann gilt $A = \mathbb{N}$.

Beweis. Angenommen $A \neq \mathbb{N}$. Dann ist die Menge $B = \mathbb{N} \setminus A$ nicht leer. Folglich gibt es eine kleinste Zahl $x \in B$. Aufgrund von i. ist $x \neq 1$. Ferner gilt $1, \ldots, x - 1 \in A$, weil x ja die kleinste Zahl in B ist. Nach ii. gilt also $x \in A$, im Widerspruch zu unserer Annahme, dass $x \in B$. □

Das Induktionsprinzip ermöglicht es uns, Beweise nach folgendem Schema zu führen.

i. Zeige, dass die Behauptung für $n = 1$ stimmt.
ii. Weise ferner nach, dass die Behauptung für $n + 1$ gilt, wenn sie für $1, \ldots, n$ gilt.

Dann folgt die Behauptung *für alle* $n \in \mathbb{N}$. Als Beispiel zeigen wir die *gaußsche Summenformel* (auch *kleiner Gauß* genannt).

Lemma 1.3.6 ("Kleiner Gauß"). Die Summe der ersten n natürlichen Zahlen ist

$$1 + 2 + \ldots + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Beweis. Wir führen die Induktion über n.

Induktionsverankerung: Im Fall $n = 1$ ist die rechte Seite

$$\frac{1(1+1)}{2} = 1$$

was tatsächlich der Summe der ersten 1 vielen natürlichen Zahlen entspricht.

Induktionsannahme: Wir nehmen als Induktionsvoraussetzung nun an, dass die Formel für $n \in \mathbb{N}$ gilt, also dass

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}. \tag{1.3}$$

Induktionsschluss: Für den Induktionsschluss berechnen wir nun die Summe der ersten $n+1$ vielen natürlichen Zahlen

$$\begin{aligned}\sum_{i=1}^{n+1} i &= (n+1) + \sum_{i=1}^{n} i \\ &= (n+1) + \frac{n(n+1)}{2} \qquad \text{[nach Induktionsannahme (1.3)]} \\ &= \frac{2n + 2 + n^2 + n}{2} = \frac{(n+1)((n+1)+1)}{2}\end{aligned}$$

wie behauptet. □

1.3.1. Beweis von Lemma 1.2.6

Wie schon angekündigt beweisen wir Lemma 1.2.6 auf zwei unterschiedliche Wege.

Beweis. [von Lemma 1.2.6 - Variante 1] Sei A eine Menge mit n Elementen. Sei $\mathcal{W}_n$ die Menge aller "Worte" bestehend aus den Buchstaben *i* und *d* mit genau n Buchstaben.

Es sei f eine bijektive Abbildung der Elemente in A in die Menge der natürlichen Zahlen 1 bis n. Diese Abbildung ordnet die Elemente in A. Für jedes $a \in A$ existiert also eine eindeutiges $1 \leq j \leq n$ sodass $f(a) = j$.

Sei g eine Abbildung die jedem $B \in \mathcal{P}(A)$ das Wort $w \in \mathcal{W}_n$ zuordnet, sodass für alle $a \in A$ gilt

- der $f(a)$-te Buchstabe von w ist i, wenn $a \in B$ und
- der $f(a)$-te Buchstabe von w ist d, wenn $a \notin B$.

Diese Abbildung ist bijektiv.

- **Surjektivität**
  Zu zeigen: Für jedes Wort $w \in \mathcal{W}_n$ lässt sich eine Menge $B \in \mathcal{P}(A)$ finden, sodass $g(B) = w$.
  $\forall\, w \in \mathcal{W}_n\ \exists\, B \in \mathcal{P}(A) : g(B) = w.$
  Sei Q die Menge der Positionen von w, an welchen ein i steht. Sei $B = f^{-1}(Q)$. Es ist nun der $f(a)$-te Buchstabe von w ein i, wenn $a \in B$ und ein d, wenn $a \notin B$. Demnach wird B von g auf w abgebildet.
- **Injektivität**
  Zu zeigen: Es gibt keine zwei verschiedenen Mengen $B, B' \in \mathcal{P}(A)$ mit $g(B) = g(B')$.
  $\nexists B, B' \in \mathcal{P}(A) : [B \neq B' \wedge g(B) = g(B')].$
  Angenommen, es gibt zwei verschiedene Mengen $B, B' \in \mathcal{P}(A)$ sodass $g(B) = g(B')$. Dann gibt es ***ohne Beschränkung der Allgemeinheit*** ein Element $a \in B$, dass nicht in B' enthalten ist (sonst wäre B eine Teilmenge von B' - aber beide Mengen sind verschieden und somit gäbe es dann ein Element $a \in B'$, dass nicht in B enthalten ist - Umbenennung der Mengen liefert die Behauptung). Dann ist aber der $f(a)$-te Buchstabe von $g(B)$ ein i und der $f(a)$-te Buchstabe von $g(B')$ ein d und somit ist $g(B) \neq g(B')$ - ein Widerspruch zu unserer Annahme.

Wie wir schon beobachteten, haben zwei endliche Mengen genau dann die gleiche Kardinalität, wenn es eine bijektive Abbildung zwischen ihnen gibt.

Wir müssen also nur noch zählen, wie viele Wörter bestehend aus zwei Buchstaben und von der Länge n es gibt. Für jede Position gibt es 2 Möglichkeiten: Entweder steht dort ein i oder ein d. Insgesamt gibt es also 2^n unterschiedliche Wörter und somit hat die Potenzmenge einer endlichen Menge mit Kardinalität n selbst Kardinalität 2^n. □

Beweis. [von Lemma 1.2.6 - Variante 2] Wir führen die Induktion über n.

Induktionsverankerung: Im Fall $n = 1$ ist die Aussage einfach zu Prüfen. Die Potenzmenge besteht in diesem Fall aus den beiden Mengen $\emptyset$ und A selbst.

Induktionsannahme: Wir nehmen als Induktionsvoraussetzung an, dass die Potenzmenge einer Menge mit n Elementen Kardinalität 2^n habe.

Induktionsschluss: Für den Induktionsschluss nehmen wir an A habe $n + 1$ Elemente. Nun zeichnen wir ein Element $a \in A$ aus und betrachten die Menge $A' = A \setminus \{a\}$. Es gilt $|A'| = n$. Nach Induktionsvoraussetzung ist $|\mathcal{P}(A')| = 2^n$.

Wir beobachten, dass jede Teilmenge B von A entweder eine Teilmenge von A' ist oder das Element a enthält (in diesem Fall ist aber $B \setminus \{a\}$ eine Teilmenge von A'). Also können wir jeder Teilmenge $B \subset A$ genau eine Teilmenge von A' zuordnen, nämlich $B \setminus \{a\}$. Dabei wird jede Teilmenge von A' genau zwei Teilmengen von A zugeordnet. Es gibt also zweimal soviele Mengen in $\mathcal{P}(A)$ als in $\mathcal{P}(A')$. Demnach ist

$$|\mathcal{P}(A)| = |\mathcal{P}(A')| \cdot 2 = 2^n \cdot 2 = 2^{n+1}.$$

□

Bemerkung 1.3.7.

Mit der Formulierung ***ohne Beschränkung der Allgemeinheit (o. B. d. A.)*** wird zum Ausdruck gebracht, dass eine Einschränkung (z. B. des Wertebereichs einer Variablen) nur zur Vereinfachung der Beweisführung vorausgesetzt wird (insbesondere zur Verringerung der Schreibarbeit), ohne dass die Gültigkeit der im Anschluss getroffenen Aussagen in Bezug auf die Allgemeinheit darunter leidet. Der Beweis wird nur für einen von meh-

reren möglichen Fällen geführt. Dies geschieht unter der Bedingung, dass die anderen Fälle in analoger Weise bewiesen werden können (z. B. bei Symmetrie). Durch o. B. d. A. können auch triviale Sonderfälle ausgeschlossen werden.

Abschließend noch ein Wort zu Beweistechniken. Mathematische Aussagen zu beweisen erfordert neben Fleiß und Sorgfalt in der Darstellung ein hohes Maß an Kreativität. In der Beschäftigung mit mathematischen Beweisen wird man feststellen, dass es allerdings Prinzipien gibt, die immer wieder angewendet werden. Wir haben schon das Induktionsprinzip kennen gelernt. An dieser Stelle noch der Hinweis auf zwei weitere Beweisprinzipien, die zunächst ähnlich aussehen, aber zu unterscheiden sind und schon unbemerkt in obigen Beweisen auftauchten.

► **Beweis durch Kontraposition** Das logische Prinzip hinter diesem Beweisprinzip haben wir schon in Bemerkung 1.1.3 beobachtet. Um zu zeigen, dass $\mathcal{A} \Rightarrow \mathcal{B}$ kann man auch zeigen, dass $\neg\mathcal{B} \Rightarrow \neg\mathcal{A}$.

Beispiel 1.3.8.

Sei n eine gerade Quadratzahl, dann ist $\sqrt{n}$ ebenfalls gerade.

Beweis. Wir möchten zeigen:

$$\text{“}n \text{ ist gerade Quadratzahl”} \Rightarrow \text{“}\sqrt{n} \text{ ist gerade”}.$$

Die Kontraposition ist

$$\neg(\text{“}\sqrt{n} \text{ ist gerade”}) \Rightarrow \neg(\text{“}n \text{ ist gerade Quadratzahl” })$$

also

$$\text{“}\sqrt{n} \text{ ist ungerade”} \Rightarrow \text{“}n \text{ ist ungerade”}$$

Sei also $\sqrt{n}$ eine ungerade, natürliche Zahl, also gibt es ein $k_1 \in \mathbb{N}$ sodass $\sqrt{n} = 2 \cdot k_1 + 1$. Dann ist

$$\begin{aligned} n &= (\sqrt{n})^2 \\ &= (2k_1 + 1)^2 \\ &= 4k_1^2 + 4k + 1 \\ &= 2(2k_1^2 + 2k_1) + 1 \end{aligned}$$

Setzte $k_2 = 2k_1^2 + 2k_1$. Dann ist $n = 2k_2 + 1$ also ungerade, was zu zeigen war. □

► **Beweis durch Widerspruch** Dabei möchte man die Wahrheit einer Aussage $\mathcal{A}$ beweisen und nimmt die Negation von $\mathcal{A}$, nämlich $\neg\mathcal{A}$ als wahr an und führt dies über logische Schlüsse zu einem Widerspruch. Also kann $\neg\mathcal{A}$ nicht wahr sein, also muss $\mathcal{A}$ wahr sein.

Beispiel 1.3.9.

Ein Beispiel findet man in der Variante 1 des Beweises von Lemma 1.2.6: Nachweis der Injektivität von g.

1.4. Relationen

Objekte, die zu unterscheiden sind, können dennoch in Bezug zueinander stehen. Der mathematische Begriff der Relation misst dieses „in (einem bestimmten) Bezug zueinander stehen“, er gibt für zwei Objekte entweder die Antwort „Ja, die beiden Objekte stehen in (in diesem bestimmten) Bezug zueinander“ oder „Nein, die Objekte stehen nicht (in diesem bestimmten) Bezug zueinander“. Wir erinnern zunächst an den Begriff des Kartesisches Produkts. Für zwei Mengen A, B heißt $A \times B := \{(a, b) \ : \ a \in A \wedge b \in B\}$ das **kartesische Produkt** von A und B.

Definition 1.4.1. Eine **(binäre) Relation** zwischen zwei Mengen A und B ist eine Teilmenge $R \subseteq A \times B$. Im Falle $A = B$ spricht man von **einer Relation auf** A.

Eine Relation zwischen A und B ist also eine Teilmenge aller *geordneten* Paare der Form (a, b) mit $a \in A$ und $b \in B$.

Beispiel 1.4.2.

- Der Begriff „größer als“ induziert eine Relation auf der Menge aller Zahlen $\mathbb{N}$:

$$R := \{(a, b) \in \mathbb{N} \times \mathbb{N} \ : \ a > b\} = \{(2, 1), \ (3, 1), (3, 2), \ (4, 1), (4, 2), (4, 3), \ (5, 1), \ldots\}$$

- Funktionen sind Relationen:
  Die Paare aus Wert $x \in \mathbb{R}$ und Funktionswert $f(x)$ einer Funktion $f : \mathbb{R} \to \mathbb{R}$ bilden eine Relation auf $\mathbb{R}$

$$R := \{(x, y) \in \mathbb{R} \times \mathbb{R} \ : \ f(x) = y\}$$

- Der Begriff „verwandt sein mit“ (engl.: „related to“) beschreibt eine Relation auf der Menge aller Menschen.

Definition 1.4.3. Eine Relation auf A heißt

- **reflexiv**, wenn für alle $a \in A$ gilt

$$(a, a) \in R.$$

- **symmetrisch**, wenn für alle $a, b \in A$ gilt

$$(a, b) \in R \Rightarrow (b, a) \in R.$$

- **transitiv**, wenn für alle $a, b, c \in A$ gilt

$$(a, b) \in R \text{ und } (b, c) \in R \Rightarrow (a, c) \in R.$$

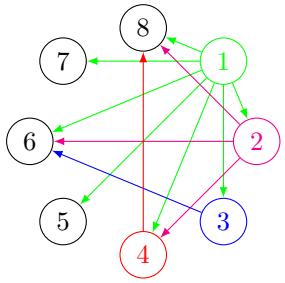
Beispiel 1.4.4.

- Wir betrachten die Teilbarkeitsrelation $R := \{(a, b) \in \mathbb{N} \times \mathbb{N} \ : \ a \text{ teilt } b\}$ auf der Menge der natürlichen Zahlen.
  Diese Relation ist reflexiv und transitiv, aber nicht symmetrisch.
- Auf $\mathbb{N}$ definiert $(a, b) \in R :\Leftrightarrow a > b$ die Relation $R := \{(a, b) \in \mathbb{N} \times \mathbb{N} \ : \ a > b\}$.
  Die Relation ist transitiv aber nicht reflexiv und nicht symmetrisch.

Bemerkung 1.4.5.

Im Fall einer endlichen Menge A kann eine Relation auf A durch einen gerichteten Graphen (mit möglichen Schlingen) visualisiert werden: Von einem Element $a \in A$ führt genau dann ein Bogen (gerichtete Kante) zu einem Element $b \in A$, wenn $(a, b) \in R$ gilt. Der gerichtete Graph in der unteren Abbildung zeigt den Graphen zur Teilbarkeitsrelation auf $\{1, \ldots 8\}$, d.h. den Graphen für

$$R := \{(a, b) \in \{1, \ldots, 8\} \times \{1, \ldots, 8\} : a \neq b \text{ und } a \text{ teilt } b\}.$$



1.4.1. Äquivalenzrelationen

Das Ziel bei der Verwendung von Äquivalenzrelationen ist, den Begriff „gleich“ (im Sinne von identisch) zu verallgemeinern auf „ähnlich“ bzw. „gleich bezüglich einer Eigenschaft“. So können z.B. zwei Gegenstände gleich sein im Bezug auf ihre Farbe (also die gleiche Farbe haben) ohne jedoch identisch zu sein.

Um den Begriff „gleich“ auf „ähnlich“ zu verallgemeinern, müssen wir sicherstellen, dass für die Verallgemeinerung weiter gilt, dass

- ein Gegenstand stets zu sich selbst ähnlich ist. (Reflexivität)
- wenn a zu b ähnlich ist, dann auch b zu a. (Symmetrie)
- wenn a ähnlich ist zu b und dies wiederum ähnlich ist zu c, so ist auch a ähnlich zu c. (Transitivität)

Definition 1.4.6 (Äquivalenzrelation)**.** Eine Relation heißt ***Äquivalenzrelation***, wenn sie reflexiv, symmetrisch und transitiv ist.

Wir schauen uns einige Beispiel von Relationen an.

Beispiel 1.4.7.

Es sei A die Menge aller Schüler einer Schule und

$$R := \{(a, b) \in A \times A \ : \ a \text{ ist in derselben Schulklasse wie } b\} \ .$$

Die Relation R ist eine Äquivalenzrelation

- Jeder Schüler $a \in A$ ist Schüler seiner (eigenen) Schulklasse, es gilt also $(a, a) \in R$. (Reflexivität)
- Ist $(a, b) \in R$, so ist Mitschüler a in derselben Schulklasse wie b.
  Also ist auch $(b, a) \in R$, denn b ist umgekehrt auch in derselben Schulklasse wie a. (Symmetrie)
- Wenn $(a, b) \in R$ und $(b, c) \in R$ gelten, dann gilt auch $(a, c) \in R$, denn es ist a in derselben Schulklasse wie b und b in derselben Schulklasse wie c, und das heißt a ist in der Schulklasse von c. (Transitivität)

Beispiel 1.4.8.

Es sei $R := \{(n, m) \in \mathbb{N} \times \mathbb{N} \ : \ m \text{ hat denselben Rest beim Teilen durch 2 wie } n\}$.

Die Relation R ist eine Äquivalenzrelation

- Jede Zahl $n \in \mathbb{N}$ hat einen festen Rest beim Teilen durch 2, es gilt also $(n, n) \in R$. (Reflexivität)
- Ist $(n, m) \in R$, so hat m denselben Rest beim Teilen durch 2 wie n.
  Also ist auch $(m, n) \in R$, denn n hat denselben Rest beim Teilen durch 2 wie m. (Symmetrie)
- Wenn $(n, k) \in R$ und $(k, m) \in R$ gelten, dann gilt beim Teilen durch 2:
  k hat denselben Rest wie n und m denselben Rest wie k, und das heißt m hat denselben Rest wie n. (Transitivität)

Die beiden „Schulklassen“ in den die Zahlen hier wahlweise gehen heißen „Gerade Zahlen“ oder „Ungerade Zahlen“.

Formal korrekt, aber trotzdem seltsam, ist das folgende Beispiel:

Beispiel 1.4.9 (Jede Relation über die Leere Menge ist eine Äquivalenzrelation)**.**

Es sei $A := \emptyset$ und $R \subseteq A \times A$. Dann ist R eine Äquivalenzrealtion.

Zunächst gilt besondererweise $R = \emptyset$ denn $A \times A = \emptyset \times \emptyset = \emptyset$.

- Reflexivität: Wegen $A = \emptyset$ gibt es *kein* $a \in A$.
  Also gibt es insbesondere kein $a \in A$ mit der Zusatzeigenschaft $(a, a) \notin R$.

Kein $a \in A$ verletzt also die Reflexivität von R. R ist also reflexiv.

- Symmetrie: Wegen $R = \emptyset$ gibt es kein Paar $(a, b) \in R$.
  Also gibt es insbesondere kein Paar $(a, b) \in R$ mit der Zusatzeigenschaft $(b, a) \notin R$.
  Kein Paar $(a, b) \in R$ verletzt also die Symmetrie von R. R ist also symmetrisch.
- Transitivität: Wegen $R = \emptyset$ gibt es kein Paar $(a, b) \in R$.
  Also gibt es insbesondere kein $(a, b) \in R$ mit der Zusatzeigenschaft, dass es $(b, c) \in R$ gibt.
  Also gibt es kein Paar (a, b) für dass zwar $(b, c) \in \mathbb{R}$ gilt, aber nicht $(a, c) \in \mathbb{R}$.
  Kein Paar $(a, b) \in R$ verletzt also die Transitivität von R. R ist also transitiv.

Beispiel 1.4.10 (keine Äquivalenzrelation)**.**

Die Relation $R := \{(n, m) \in \mathbb{N} \times \mathbb{N} \ : \ n \leq m\}$ ist **nicht symmetrisch** und deswegen *keine* Äquivalenzrelation:

Es gilt zwar $(1, 2) \in R$ wegen $1 \leq 2$ aber es gilt nicht $(2, 1) \notin R$.

Die Relation R ist zwar reflexiv und transitiv, aber eben nicht symmetrisch.

Bemerkung 1.4.11.

Am Gegenbeispiel 1.4.10 sieht man:

> *Um zu zeigen, dass eine Relation* keine *Äquivalenzrelation ist, genügt es zu zeigen, dass eine der benötigten Eigenschaften nicht gilt. Um wiederum zu zeigen, dass eine Relation z.B. nicht symmetrisch ist, genügt ein einziges Gegenbeispiel!*

Definition 1.4.12 (Äquivalenz)**.** Es sei R eine Äquivalenzrelation auf A.

Für $(a, b) \in R$ schreibt man kurz $a \sim_R b$, und sagt: a und b sind ***äquivalent bezüglich*** R.

Wenn klar ist, welche Relation Gemeint ist, wird „$\sim_R$“ zu „$\sim$“ vereinfacht.

Das Symbol $\sim$ verallgemeinert das Symbol „$=$“, es gelten auf Ebene der logischen Operatoren die selben Regeln:

$a = b$	ist äquivalent zu	$b = a$.	(Symmetrie)	Aus $a = b$ und $b = c$	folgt $a = c$.	(Transitivität)
$a \sim b$	ist äquivalent zu	$b \sim a$.		Aus $a \sim b$ und $b \sim c$	folgt $a \sim c$.	

Definition 1.4.13 (Äquivalenzklasse)**.** Es sei R eine Äquivalenzrelation auf A.

Für jedes Element $a \in A$ ist die ***Äquivalenzklasse***

$$[a]_R := \{b \in A \ : \ (a, b) \in R\}$$

die Menge der zu a äquivalenten (bzw. ähnlichen) Elemente aus A.

Beispiel 1.4.14 (Schul**klassen** sind Äquivalenz**klassen**).

Es sei A die Menge aller Schüler einer Schule und

$$R := \{(a,b) \in A \times A \ : \ a \text{ ist in derselben Schulklasse wie } b\} \ .$$

Für jeden Schüler a bilden die Mitschüler seiner Schulklasse seine Äquivalenzklasse:

$$\begin{aligned} [a]_R &= \{b \in A \ : \ (a,b) \in R\} \\ &= \{b \in A \ : \ b \text{ ist in derselben Schulklasse wie } a\} = \big\{ \text{ Alle Schüler in der Schulkasse von } a \big\} \end{aligned}$$

Gilt $(a,b) \in R$, d.h. a und b sind in derselben (Schul-)Klasse (z.B. “6c”), so gilt $[a]_R = [b]_R$:

$$\begin{aligned} [a]_R &= \big\{ \text{ Alle Schüler in der Schulklasse von } a \big\} = \big\{ \text{ Alle Schüler der 6c } \big\} \\ [b]_R &= \big\{ \text{ Alle Schüler in der Schulklasse von } b \big\} = \big\{ \text{ Alle Schüler der 6c } \big\} \end{aligned}$$

Beispiel 1.4.15.

Es sei $R := \{(n,m) \in \mathbb{N} \times \mathbb{N} \ : \ m \text{ hat denselben Rest beim Teilen durch 2 wie } n\}$.

- Die Zahl 7 hat einen Rest von 1 beim Teilen durch 2, es gilt also:

$$\begin{aligned} [7]_R &= \{m \in \mathbb{N} \ : \ (7,m) \in R\} \\ &= \{m \in \mathbb{N} \ : \ m \text{ hat selben Rest beim Teilen durch 2 wie } 7\} \\ &= \{m \in \mathbb{N} \ : \ m \text{ hat Rest 1 beim Teilen durch 2}\} \\ &= \big\{ \text{ Alle ungeraden Zahlen} \big\} \end{aligned}$$

  Analog gilt $[3]_R = \big\{ \text{ Alle ungeraden Zahlen} \big\}$ und $[5]_R = \big\{ \text{ Alle ungeraden Zahlen} \big\}$ etc.
- Die Zahl 8 hat einen Rest von 0 beim Teilen durch 2, es gilt also:

$$\begin{aligned} [8]_R &= \{m \in \mathbb{N} \ : \ (8,m) \in R\} \\ &= \{m \in \mathbb{N} \ : \ m \text{ hat selben Rest beim Teilen durch 2 wie } 8\} \\ &= \{m \in \mathbb{N} \ : \ m \text{ hat Rest 0 beim Teilen durch 2}\} \\ &= \big\{ \text{ Alle geraden Zahlen} \big\} \end{aligned}$$

  Analog gilt $[2]_R = \big\{ \text{ Alle geraden Zahlen} \big\}$ und $[6]_R = \big\{ \text{ Alle geraden Zahlen} \big\}$ etc.

Die beiden „Schulklassen",’ in die die Zahlen hier wahlweise gehen, heißen „Grade Zahlen“ oder „Ungrade Zahlen“.

Anhand der Beispiele erkennt man bereits, dass zwei Äquivalenzklassen entweder „grundverschieden“ sind (leerer Schnitt) oder aber identisch sind. Dies ist einleuchtend, denn die Äquivalenzklasse $[a]_R$ besteht aus allen Elementen, die zu a äquivalent sind, d.h. gleich sind bezüglich der Eigenschaft, die R definiert.

Fordert man z.B. „1) Nenne alles, was dieselbe Länge hat wie a.“ und erhält als Antwort unter anderem b, so bekommt man auf die Aufforderung „2) Nenne alles, was dieselbe Länge hat wie b.“ dieselbe Antwort wie bei 1). Dies kann man dann mit beliebigen Eigenschaften wiederholen (Gewicht, Eckenanzahl etc.).

Diese Einsicht verallgemeinern wir zu „Zwei Elemente aus A haben bezüglich R genau dann dieselbe Äquivalenzklasse, wenn sie bezüglich R äquivalent sind“. Genauer gilt:

Lemma 1.4.16. Es sei R eine Äquivalenzrelation über A.

i) Für $a, b \in A$ gilt dann entweder $[a]_R = [b]_R$ oder $[a]_R \cap [b]_R = \emptyset$.
ii) Es gilt $[a]_R = [b]_R$ genau dann wenn $(a, b) \in R$ gilt.

Beweis. Es sei R eine Äquivalenzrelation über A und $a, b \in R$ mit $a \neq b$.

Wir zeigen zunächst ii).

'$\Rightarrow$' Annahme: Es gelte $[a]_R = [b]_R$. Wegen $(b, b) \in R$ gilt $b \in [b]_R$ und damit $b \in [a]_R$ bzw. $(a, b) \in R$.

'$\Leftarrow$' Annahme: Es gelte $(a, b) \in R$. Wir zeigen $[a]_R \subseteq [b]_R$ und $[b]_R \subseteq [a]_R$.
Sei nun $x \in [a]_R$, so gilt $(a, x) \in R$ und da R eine Äquivalenzrelation ist gilt damit

$$(a, x), (a, b) \in R \quad \Rightarrow \quad (x, a), (a, b) \in R \quad \Rightarrow \quad (x, b) \in R \quad \Rightarrow \quad x \in [b]_R$$

Da x beliebig war folgt also $[a]_R \subseteq [b]_R$. Ganz analog beweist man $[b]_R \subseteq [a]_R$. Es gilt also: $[a]_R = [b]_R$.

Wir zeigen nun i): Es seien $a, b \in A$ mit $[a]_R \cap [b]_R \neq \emptyset$. Zu zeigen ist: $[a]_R = [b]_R$.

Sei $c \in [a]_R \cap [b]_R$ beliebig. Nach Def. der Äquivalenzklassen gilt: $(a, c), (b, c) \in R$ und da R eine Äquivalenzrelation ist gilt damit

$$(a, c), (b, c) \in R \quad \Rightarrow \quad (a, c), (c, b) \in R \quad \Rightarrow \quad (a, b) \in R$$

Aus ii) schließen wir nun: $[a]_R = [b]_R$. □

Aus Lemma 1.4.16 schließen wir, dass es genügt, ein *einziges* Element einer Äquivalenzklasse zu kennen, um die Äquivalenzklasse zu rekonstruieren:

Definition 1.4.17 (Vertreter einer Äquivalenzklasse)**.** Es sei $M \subseteq A$ eine Äquivalenzklasse einer Äquivalenzrelation R auf der Menge A.

Ein Element $x \in M$ heißt dann ***Vertreter*** der Äquivalenzklasse M, denn es gilt $[x]_R = M$.

Beispiel 1.4.18 (Jeder Schüler ist Vertreter seiner Schul**klasse**)**.**

Es sei A die Menge aller Schüler einer Schule und

$$R := \{(a, b) \in A \times A \ : \ a \text{ ist in derselben Schulklasse wie } b\} .$$

Es sei nun Alice eine Schülerin der Schulklasse “6c”. Alle Mitschüler aus der Schulklasse von Alice bilden die Äquivalenzklasse von Alice:

$$\begin{aligned}[\text{Alice}]_R &= \{b \in A \ : \ (\text{Alice}, b) \in R\} \\ &= \{b \in A \ : \ b \text{ ist in derselben Schulklasse wie Alice } \} = \{ \text{ Alle Schüler der 6c } \}\end{aligned}$$

Jeder Schüler und jede Schülerin aus der 6c ist nun ein *Vertreter* seiner (Schul-)Klasse, denn anhand des einzelnen Schülers kann man natürlich die ganze Klasse ermitteln:

Ist Bob auch in der 6c, d.h. Alice und Bob sind in derselben (Schul-)Klasse “6c”, so gilt $[\text{Alice}]_R = [\text{Bob}]_R$:

$$[\text{Bob}]_R = \{ \text{ Alle Schüler in der Schulklasse von Bob } \} = \{ \text{ Alle Schüler der 6c } \}$$

Beispiel 1.4.19.

Es sei $R := \{(n, m) \in \mathbb{N} \times \mathbb{N} \ : \ m \text{ hat denselben Rest beim Teilen durch 2 wie } n\}$.

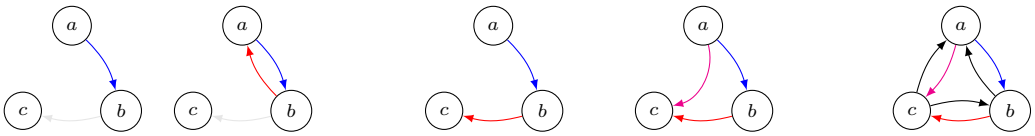
Die beiden Äquivalenzklassen von R sind:

$$\begin{aligned} M_g &:= \{2n \ : \ n \in \mathbb{N}\} \quad \text{(alle graden Zahlen)} \quad \text{und} \\ M_u &:= \{2n+1 \ : \ n \in \mathbb{N}\} \quad \text{(alle ungraden Zahlen)} \end{aligned}$$

Ein Vertreter von M_g ist $x = 6 \in M_g$, und es gilt tatsächlich $[6]_R = M_g$ (s. Beispiel 1.4.15).

1.4.2. Äquivalenzklassen: Veranschaulichung als Graph

Die Einsichten aus Lemma 1.4.16 lassen sich leichter anhand von Graphen veranschaulichen bzw. verstehen. Ist R eine Äquivalenzrealtion so „erzeugen“ die Regeln der Symmetrie und Transitivität Bögen im zugehörigen gerichteten Graphen der Relation:



$(a, b) \in R \quad \Rightarrow (b, a) \in R$ Symmetrie

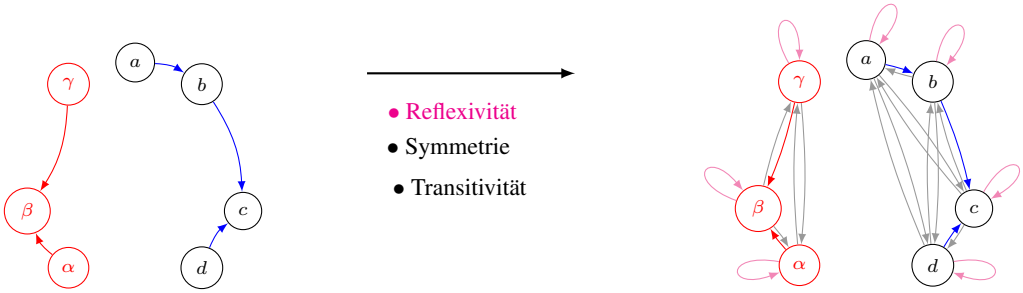
$(a, b), (b, c) \in R \quad \Rightarrow (a, c) \in R$ Transitivität

Transitivität & Symmetrie

Aus der Skizze entnimmt man: Ist G der gerichtete Graph einer Äquivalenzrelation und sind zwei Knoten v und w *irgendwie* über einen Weg aus Bögen (und Gegenbögen) verbunden, so sind v und w in$R$ auch direkt verbunden. Dies liefert:

Der gerichtetete Graph G einer Äquivalenzrelation zerfällt in disjunkte *Cliquen* G'_i:

- Jeder dieser Untergraphen G'_i ist eine Clique (bzw. vollständig), d.h. jeder mögliche Bogen der Form (a, b) mit Knoten von G'_i ist Teil des Graphen.
- Je zwei verschiedene Untergraphen G'_i und G'_j sind disjunkt: D.h. es gibt keine Bögen von G_i nach G_j (und umgekehrt).



Bekannte Paare einer Relation

Kleinste mögliche Relation mit allen vorgegebenen Paaren.

Abbildung 1.1.: Kleinstmögliche Äquivalenzrelation mit einigen bekannte Tupeln.

Jede dieser Cliquen ist eine Äquivalenzklasse der Äquivalenzrelation R. Im Falle der Relation $R = \{(a, b) : b$ spielt im selben Verein wie $a\}$ sind dies grade die möglichen Vereine.

Teil I.

Diskrete Mathematik

2 Rechnen mit ganzen Zahlen - Anwendungen

2.1. Grundlagen

In den folgenden Abschnitte werden wir Anwendungen in der Diskreten Mathematik kennenlernen, welche auf Grundlagen der Algebra, der linearen Algebra und elementaren Zahlentheorie aufbauen. Wir erinnern einige Objekte aus der Mathematik für die Informatik I.

Erinnerung

Modulo-Rechnung

Zunächst Erinnern wir an die grundlegende Definition.

Definition 2.1.1 (Rest und Äquivalenz Modulo m)**.** Es seinen $a, b \in \mathbb{Z} \setminus \{0\}$. Mit $\mathsf{Rest}(a, b)$ bezeichenen wir den ***Rest***, der beim Teilen von a durch b entsteht:

$$\mathsf{Rest}(a, b) := \min\{r \in \mathbb{N} \ : \ \exists m \in \mathbb{Z} \text{ mit } \ a = m \cdot b + r\}.$$

Es sei $m \in \mathbb{N} \setminus \{1\}$, dann gilt für $a, b \in \mathbb{Z}$

$$a \equiv b \pmod{m} \quad \Leftrightarrow \mathsf{Rest}(a, m) = \mathsf{Rest}(b, m)$$

Man sagt in diesem Fall „a und b sind äquivalent *mod* m“.

Die Zahl m bezeichnet man dabei als ***(den) Modul*** der *Modul-Gleichung* $a \equiv b \pmod{m}$.

Die Relation Äquivalenz Modulo m ist eine Äquivalenzrelation.

Lemma 2.1.2. Für jedes $m \in \mathbb{N} \setminus \{1\}$ ist $\mathcal{R}_m := \{(a, b) \in \mathbb{Z} \times \mathbb{Z} \ : \ a \equiv b \pmod{m}\}$ eine Äquivalenzrelation.

Euklidischer Algorithmus und der Satz von Bézout

Der Euklidischen Algorithmus berechnet neben dem $\mathrm{ggT}(a, b)$ zwei ganze Zahlen $s, t \in \mathbb{Z}$, für die $s \cdot a + t \cdot b = \mathrm{ggT}(a, b)$ gilt. Diese Zahlen s, t heißen ***Bézout-Multiplikatoren***. Wir werden uns bald mit dem RSA-Verfahren (ein Kryptoverfahren) und den Chinesischen Restsatz beschäftigen. Die ***Bézout-Multiplikatoren*** spielen dabei eine wichtige Rolle.

Algorithmus 2.1.3 (Erweiterter Euklidischer Algorithmus)**.**

Input: $a, b \in \mathbb{N}$ mit $a \geq b$.
Output: $r_{j-1} = \mathrm{ggT}(a, b)$ und die Gleichung $r_{j-1} = s_{j-1} \cdot a + t_{j-1} \cdot b$.
Setze $r_0 := a$ und $r_1 := b$.
Setze $s_0 := 1$ und $s_1 := 0$.
Setze $t_0 := 0$ und $t_1 := 1$.
Setze $j := 1$.
while $r_j > 0$ **do**
Setze $m_j := \lfloor \frac{r_{j-1}}{r_j} \rfloor$.
Setze $r_{j+1} := r_{j-1} - m_j r_j$.
Setze $s_{j+1} := s_{j-1} - m_j s_j$.
Setze $t_{j+1} := t_{j-1} - m_j t_j$.
Setze $j := j + 1$.
end while
return $(r_{j-1}, s_{j-1}, t_{j-1})$.

Lemma 2.1.4. Es sei (r_n, s_n, t_n) der Output des Algorithmus 2.1.3 mit Input $a, b \in \mathbb{N}$ mit $a \geq b$. Dann gilt für die Zwischenergebnisse

$$r_i = s_i \cdot a + t_i \cdot b \qquad \text{für alle } i \in \{0, 1, \ldots, n\}.$$

Insbesondere gilt also $r_n = \mathrm{ggT}(a, b) = s_n \cdot a + t_n \cdot b$.

Satz 2.1.5 (Satz von Bézout)**.** Für zwei Zahlen $a, b \in \mathbb{N}$ gibt es Zahlen $s, t \in \mathbb{Z}$ mit $s \cdot a + t \cdot b = \mathrm{ggT}(a, b)$.

Der Eukidischen Algorithmus in Form von Algorithmus 2.1.3 lässt sich wie im Anhang gezeigt ANHANG angenehm übersichtlich in Tabellenform durchführen.

2.1.1. Konsequenzen des Satzes von Bézout: Lemma von Euklid

Bevor wir uns mit dem Chinesischen Restsatz beschäftigen, ernten wir noch eine zahlentheoretische Aussage über die Faktorisierung natürlicher Zahlen.

Lemma 2.1.6 (Euklid)**.** Teilt eine Primzahl p das Produkt $a \cdot b$ zweier Zahlen $a, b \in \mathbb{N}$, so teilt p auch mindestens einen der Faktoren a oder b.

Beweis. Wir nehmen an, dass p kein Teiler von a ist und zeigen, dass dann $p|b$ gelten muss:

Es gelte $p \nmid a$. Da p Prim ist gilt dann $\mathrm{ggT}(a, p) = 1$. Nach Satz von Bézout gibt es damit $s, t \in \mathbb{Z}$ mit $s \cdot p + t \cdot a = 1$. Multiplikation mit b auf beiden Seiten der Gleichung ergibt:

$$s \cdot p \cdot b + t \cdot (\underbrace{a \cdot b}_{\text{nach Voraussetzung Vielfaches von } p}) = b$$

Da p beide beide Summanden auf der linken Seite teilt, teilt p auch die rechte Seite, also b. □

Induktiv ergibt sich daraus unmittelbar das folgende Korollar.

Korollar 2.1.7. Ist ein Produkt $\prod_{i=1}^{m} a_i$ von $m > 2$ ganzen Zahlen durch die Primzahl p teilbar, so ist mindestens ein Faktor a_i durch p teilbar.

2.2. Der Chinesische Restsatz

Mit Hilfe des nachfolgend diskutierten Chinesischen Restsatzes ist es möglich, Berechnungsprobleme mit großen Zahlen in mehrere Probleme mit kleineren Zahlen aufzuteilen:

> *Statt mit der ganzen Zahl $a \in \mathbb{Z}$ zu rechnen, rechnet man mit den Resten $(a_1, \ldots, a_k)$ bezüglich paarweise teilerfremder Moduln $m_1, \ldots, m_k$.*

Diese Strategie setzt aber voraus, dass man nach Durchführung der "vereinfachten Rechnung mit Resten", die ursprünglich zu berechnende Zahl aus den Resten "zurückgewinnen kann". Eine Antwort auf diese Frage der Rekonstruierbarkeit wird durch den ***Chinesischen Restsatz*** gegeben, der in einem speziellen Fall bereits Sun Tsu etwa 300 n. Chr. bekannt war.

Im Folgenden werden wir Systeme von Modulgleichungen der Form $x \equiv a_i \pmod{m_i}$ lösen. Solche Systeme nennt man „simultane Kongruenzen", da x gleichzeitig (also simultan) mehrere Kongruenzen erfüllen muss.

Soll ein $x \in \mathbb{Z}$ mehrere solcher Modulgleichungen mit verschiedenen m_i erfüllen, so ist dies genau dann ***nicht immer möglich***, wenn die m_i einen gemeinsamen Teiler haben, wie das folgende Beispiel zeigt.

Beispiel 2.2.1.

Das folgende System von Modulgleichungen ist nicht lösbar:

$$\begin{aligned} x &\equiv 0 \pmod 6 \qquad x \text{ ist gerade wegen } x = k \cdot 6 + 0 \\ x &\equiv 1 \pmod 8 \qquad x \text{ ist ungerade wegen } x = \ell \cdot 8 + 1 \end{aligned}$$

Die erste Gleichung impliziert, dass x gerade ist, aber x muss ungerade sein, um die zweite Gleichung zu erfüllen.

Das folgende System von Modulgleichungen ist dagegen mit $x = 10$ lösbar:

$$\begin{aligned} x &\equiv 4 \pmod 6 \\ x &\equiv 2 \pmod 8 \end{aligned}$$

Sind die entsprechenden Module jedoch teilerfremd, so garantiert der Chinesische Restsatz immer eine Lösung:

Satz 2.2.2 (Chinesischer Restsatz)**.** Seien $m_1, \ldots, m_k \in \mathbb{N}$ paarweise teilerfremd und $a_1, \ldots, a_k \in \mathbb{Z}$.

- Dann existiert eine Lösung $x = a$ für $a \in \mathbb{Z}$ des Systems von Kongruenzen

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

- Die Lösung ist modulo $m := m_1 \cdot m_2 \cdot \ldots \cdot m_k$ eindeutig, d.h. die Lösungsmenge ist $\{a + \ell \cdot m \ : \ \ell \in \mathbb{Z}\}$.

Ein typischer Fehler in der Analyse des Chinesischen Restsatzes ist, fälschlich die Umkehrung des Satzes anzunehmen.

Bemerkung 2.2.3 (Umkehrung und Berechenbarkeitsaspekt des Chinesischen Restsatzes)**.**

Es gilt

NICHT paarweise teilerfremde Moduln. $\implies$ *System* **kann** *lösbar sein,* **muss aber nicht**.

Hat ein System von Kongruenzen ***nicht*** paarweise teilerfremde Moduln $m_1, \ldots, m_k$, so kann das System trotzdem lösbar sein (s. Beispiel 2.2.1), wenn die Reste $a_1, \ldots, a_k$ geeignet gewählt wurden.

Betrachtet man den Chineschen Restsatz unter dem Aspekt "praktischer Berechenbarkeit" so übersieht man leicht die wichtige Implikation der zweiten Aussage.

Es gilt

DOCH paarweise teilerfremde Moduln. $\implies$ *Lösungen liegen "sehr weit" auseinander!*

Hat ein System von Kongruenzen paarweise teilerfremde Moduln $m_1, \ldots, m_k$, so liegen je zwei Lösungen des Systems "*sehr, sehr weit*" von einander entfernt. Die Lösungen unterscheiden sich nämlich um ein Vielfaches von $m := m_1 \cdot m_2 \cdot \ldots \cdot m_k$, die Lösungen liegen also mindestens "m-weit" auseinander.

Wir beweisen nun den Chinesischen Restsatz.

Beweis. Gegeben sie das System simultaner Kongruenzen aus Satz 2.2.2 mit $m_1, \ldots, m_k \in \mathbb{N}$ paarweise teilerfremd und $a_1, \ldots, a_k \in \mathbb{Z}$.

- **Eindeutigkeit:** Sind x und $\widetilde{x}$ Lösungen von $(\star)$, d.h. $x \equiv \widetilde{x} \equiv a_i \pmod{m_i}$ für alle i, so gilt $m_i | (x - \widetilde{x})$ für alle i. Aus der Eindeutigkeit der Primfaktorzerlegung und der paarweisen Teilerfremdheit der m_i folgt $m | (x - \widetilde{x})$, d.h. $x - \widetilde{x} \equiv 0 \pmod{m}$.
- **Existenz:** Der nachfolgende Existenzbeweis liefert gleichzeitig ein Verfahren zur Bestimmung von x. Wir bestimmen im folgenden Basislösungen e_ℓ, $1 \leq \ell \leq k$, dies sind Lösungen des speziellen Glei-

chungssystems $e_\ell = 1 \pmod{m_\ell}$ und $e_j = 0 \pmod{m_j}$ für $j \neq \ell$. Es gelten dann:

e_1	e_2	$\dots$	e_k
$e_1 \equiv 1 \pmod{m_1}$	$e_2 \equiv 0 \pmod{m_1}$		$e_k \equiv 0 \pmod{m_1}$
$e_1 \equiv 0 \pmod{m_2}$	$e_2 \equiv 1 \pmod{m_2}$		$e_k \equiv 0 \pmod{m_2}$
$\vdots$	$\vdots$	$\dots$	$\vdots$
$e_1 \equiv 0 \pmod{m_k}$	$e_2 \equiv 0 \pmod{m_k}$		$e_k \equiv 1 \pmod{m_k}$

Aus dieser setzen wir die Lösung wie folgt zusammen:

$$x := a_1 \cdot e_1 + a_2 \cdot e_2 + \ldots + a_k \cdot e_k$$

Es gilt dann:

$$\begin{array}{lllll}
x \equiv a_1 \cdot 1 & +a_2 \cdot 0 & +\ldots+ & a_k \cdot 0 & \pmod{m_1} \quad x \equiv a_1 \pmod{m_1} \\
x \equiv a_1 \cdot 0 & +a_2 \cdot 1 & +\ldots+ & a_k \cdot 0 & \pmod{m_2} \quad x \equiv a_2 \pmod{m_1} \\
\vdots & & \ddots & & \qquad\qquad \vdots \\
x \equiv a_1 \cdot 0 & +a_2 \cdot 0 & +\ldots+ & a_k \cdot 1 & \pmod{m_2} \quad x \equiv a_k \pmod{m_k}
\end{array}$$

Berechnung der Basislösungen: Es sei $\ell \in \{1, \dots k\}$. Setze

$$M_\ell := \prod_{\substack{i=1 \\ i \neq \ell}}^{k} m_i = \frac{m_1 \cdot m_2 \cdot \ldots \cdot m_k}{m_\ell} \qquad \text{(Produkt aller } m_i \textbf{ außer } m_\ell\text{)}$$

Da die Moduln $m_1, m_2, \dots, m_k$ paarweise teilerfremd sind, gilt $\mathrm{ggT}(m_\ell, M_\ell) = 1$. Nach dem Satz von Bézout existieren daher ganze Zahlen s, t, so dass gilt:

$$1 = s \cdot m_\ell + \underbrace{t \cdot M_\ell}_{e_\ell}$$

Diese Zahlen s, t können mit dem erweiterten Euklidischen Algorithmus berechnet werden. Wir setzen nun $e_\ell := t \cdot M_\ell = 1 - s \cdot m_\ell$. Damit gilt:

$$\begin{array}{llll}
e_\ell = 1 - s \cdot m_\ell & \equiv 1 & \pmod{m_\ell} & \\
e_\ell = t \cdot M_\ell & \equiv 0 & \pmod{m_j} & \text{für } j \neq \ell
\end{array}$$

□

Beispiel 2.2.4.

Zu Lösen ist

$$\begin{array}{lll}
x \equiv 3 & \pmod{4} & \qquad \text{d.h.} \quad a_1 = 3 \quad m_1 = 4 \\
x \equiv 4 & \pmod{5} & \qquad\qquad\;\; a_2 = 4 \quad m_2 = 5 \\
x \equiv 1 & \pmod{9} & \qquad\qquad\;\; a_3 = 1 \quad m_3 = 9
\end{array}$$

Berechnen der Basis-Lösungen aus den **Moduln** m_i:

Modul	Prod. d. Restlichen	Bézout aus Euklid. Algo.	Basislösung
$m_1 = 4$	$M_1 = 5 \cdot 9 = \mathbf{45}$	$\underbrace{-11 \cdot 4}_{-44} + 1 \cdot 45 = 1 \Rightarrow$	$e_1 := \mathbf{45}$
$m_2 = 5$	$M_2 = 4 \cdot 9 = \mathbf{36}$	$\underbrace{-7 \cdot 5}_{-35} + 1 \cdot 36 = 1 \Rightarrow$	$e_2 := \mathbf{36}$
$m_3 = 9$	$M_3 = 4 \cdot 5 = \mathbf{20}$	$\underbrace{9 \cdot 9}_{81} + (-4) \cdot 20 = 1 \Rightarrow$	$e_3 := \mathbf{-80}$

Berechnen einer Lösung aus den Resten a_i und Basis-Lösungen e_i:
Eine der (unendlich vielen!) Lösungen ist:

$$\begin{aligned} \mathbf{x} &= 3 \cdot e_1 + 4 \cdot e_2 + 1 \cdot e_3 \\ &= 3 \cdot 45 + 4 \cdot 36 + 1 \cdot (-80) = \mathbf{199} \end{aligned}$$

Diese Lösung ist eindeutig modulo $m := 4 \cdot 5 \cdot 9 = 180$, da $180 < 199$ gilt, ist 199 nicht die kleinste positive Lösung, sondern

$$\mathsf{Rest}(199, 180) = 199 - 180 = 19 \quad \text{ist kleinste positive Lösung}$$

Die Menge aller Lösungen lautet: $\{199 + k \cdot 180 \ : \ k \in \mathbb{Z}\}$ bzw. $\{19 + k \cdot 180 \ : \ k \in \mathbb{Z}\}$.

2.2.1. Anwendung des Chinesischen Restsatzes: Probabilistischer Gleichheitstest

Wir möchten nun einen Anwendung des Chinesischen Restsatzes diskutieren. Dabei sind zwei große Zahlen zu vergleichen, indem nur wenige Bits tatsächlich verglichen werden.

► **Die Aufgabe:** Zwei Personen an den Enden eines Nachrichtenkanals wollen zwei 10 000-Bit lange, binäre Nachrichten auf Gleichheit hin überprüfen. Dabei sollen möglichst wenige Bits übertragen werden.

► **Der Algorithmus:** Das folgende Verfahren erlaubt einen Vergleich der beiden als Zahlen $a, b < 2^{10\,000}$ interpretierbaren Nachrichten, wobei anstatt der bis zu 10 000 Bits für die gesamte Nachricht nur $k \cdot 101$ Bits gesendet werden (bzw. $k \cdot 202$ Bits bei erforderlicher Übertragung der Moduln). Das Verfahren garantiert schon für $k = 1$ sehr hohe Sicherheit.

Algorithmus 2.2.5 (Probabilistischer Gleichheitstest)**.**

Input: $a, b \in \mathbb{N}$ mit $a, b < 2^{10\,000}$.
Output: „$a \neq b$“ oder „*mit großer Wahrscheinlichkeit* $a = b$“.

1. Wähle zufällig Primzahlen $p_1, \ldots, p_k \in [2^{100}, 2^{101}]$.
2. Übertrage a modulo p_i für alle $i = 1, \ldots, k$.

if $a \not\equiv b \pmod{p_i}$ für ein i **then**
  return „$a \neq b$“
else
  return „*mit großer Wahrscheinlichkeit* $a = b$“
end if

► **Die Analyse:** Ist die Ausgabe des Verfahrens „$a \neq b$“ so ist dies stets richtig, da aus $a \not\equiv b \pmod{p_i}$ (für eine der Primzahlen p_i) sofort $a \neq b$ folgt. Wir schätzen nun die Wahrscheinlichkeit ab, dass das Verfahren zu einer Fehlentscheidung der Form „$a = b$“ führt, obwohl $a \neq b$ gilt. Zentral ist dabei, dass es für $a \neq b$ nicht viele p_i geben kann mit $a \equiv b \pmod{p_i}$. Folgende Beobachtung hilft enorm bei der Analyse.

Lemma 2.2.6. Für zwei natürliche Zahlen $a, b \in \mathbb{N}$ mit $a \neq b$ gibt es höchstens 99 Primzahlen $p_i \in [2^{100}, 2^{101}]$ mit $a \equiv b \pmod{p_i}$.

Beweis. Wir beweisen die Aussage durch Widerspruch.

Annahme: Es gibt 100 verschiedene Primzahlen $q_1, \ldots q_{100} \in [2^{100}, 2^{101}]$ mit $a \equiv b \pmod{q_i}$ für alle $i = 1, \ldots 100$.

Nach dem Chinesischen Restsatz ist dann aber $a \equiv b \pmod{m}$ mit $m := q_1 \cdots q_{100}$. Es gilt nach der Wahl der Primzahlen aus dem Intervall $[2^{100}, 2^{101}]$, dass

$$m > (2^{100})^{100} = 2^{10\,000}.$$

Da jedoch $a, b < 2^{10\,000} < m$ sind sie identisch $a = b$. (Denn $x \equiv b \pmod{m}$ wird durch $x = b$ erfüllt und sonst keiner Zahle zwischen 0 und m.) □

Eine Fehlentscheidung ist also überhaupt nur dann möglich, falls das Verfahren zufälligerweise *ausschließlich* solche Primzahlen $q > 2^{100}$ auswählt für welche $a \equiv b \pmod{q}$ gilt. Von diesen gibt es höchstens 99. Wie groß die Wahrscheinlichkeit für dieses Ereignis ist, hängt von der Anzahl der Primzahlen in dem Intervall $[2^{100}, 2^{101}]$ ab. Mit Hilfe des berühmten Primzahlsatzes lässt sich dies Anzahl abschätzen.

Lemma 2.2.7. In dem Intervall $[2^{100}, 2^{101}]$ gibt es ungefähr 2^{93} Primzahlen.

Beweis. Nach dem berühmten Primzahlsatz gilt für die Anzahl $\pi(n)$ der Primzahlen in $[1, n]$ die asymptotische Formel $\pi(x) \simeq n/\ln(n)$. In $[2^{100}, 2^{101}]$ gibt es daher approximativ 2^{93} Primzahlen:

$$\begin{aligned}\pi\left(2^{101}\right) - \pi\left(2^{100}\right) &\simeq \frac{2^{101}}{\ln\left(2^{101}\right)} - \frac{2^{100}}{\ln\left(2^{100}\right)} \geq \frac{1}{\ln\left(2^{100}\right)}\left(2^{101} - 2^{100}\right) \\ &= \frac{2^{100}}{100 \cdot \ln(2)} \geq 2^{93}\end{aligned}$$

□

Lemma 2.2.8. Die Fehlerwahrscheinlichkeit, bei unterschiedlichen Inputzahlen $a, b < 2^{10\,000}$ ist nach oben beschränkt durch $\left(\frac{99}{2^{93}}\right)^k$.

Beweis. Nach Lemma 2.2.6 gibt es höchstens 99 „ungeeigneten“ Primzahlen in $[2^{100}, 2^{101}]$. Die Wahrscheinlichkeit zufällig eine von diesen zu erwischen ist nach Lemma 2.2.7 ungefähr $99/2^{93} \simeq 0,9996 \cdot 10^{-26}$. Bei k-facher unabhängiger Wahl einer Primzahl aus $[2^{100}, 2^{101}]$ ist die Fehlerwahrscheinlichkeit also höchstens $\left(\frac{99}{2^{93}}\right)^k$. □

Bemerkung 2.2.9.

Schon für $k = 1$ ist die Fehlerwahrscheinlichkeit des probabilistischen Gleichheitstest also verschwindend klein.

Man beachte: Diese Schranke für die Fehlerwahrscheinlichkeit ist unabhängig von den Nachrichten a und b. Wenn wir dagegen an zufälligen Bitpositionen prüfen, erkennen wir die Ungleichheit oft nicht, wenn a und b an fast allen Bitpositionen übereinstimmen.

2.3. Die Eulersche Phi-Funktion

Nachdem wir die Eulersche φ-Funktion erinnert haben, werden wir uns mit der Aufgabe beschäftigen, Urbilder der φ-Funktion zu finden. Die φ-Funktion spielt bei der späteren Behandlung von Anwendungen wie dem RSA-Schema eine zentrale Rolle.

Die eulersche φ-Funktion

Definition 2.3.1 (Eulersche φ-Funktion)**.** Für $n \in \mathbb{N}$ ist die Eulersche φ-Funktion definiert als

$$\varphi(n) := |\{m \in \{1, \ldots, n\} \ : \ \mathrm{ggT}(m, n) = 1\}|$$

Satz 2.3.2. Für eine Primzahl $p \in \mathbb{N}$ gilt:

- $\varphi(p) = (p-1)$
- $\varphi(p^k) = p^{k-1}(p-1)$ mit Exponenten $k \in \mathbb{N}$, $k \neq 0$.
- Ist $n = p_1^{k_1} \cdot p_2^{k_2} \cdots p_\ell^{k_\ell}$ die Primzahlzerlegung von n mit $p_1 < p_2 < \ldots < p_\ell$ und $k_1, k_2 \ldots, k_\ell \neq 0$, so gilt:

$$\varphi(n) = p_1^{k_1-1} \cdot (p_1 - 1) \ \cdot p_2^{k_2-1} \cdot (p_2 - 1) \ \cdots \ p_\ell^{k_\ell-1} \cdot (p_\ell - 1)$$

Lemma 2.3.3. Es seien $n, m \in \mathbb{N}$. Es gilt genau dann $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$, wenn $\mathrm{ggT}(m, n) = 1$.

Satz 2.3.2 liefert nun die Grundlage, um $\varphi(n)$ zu berechnen: Zunächst muss n in seine Primfaktorzerlegung zerlegt werden, und dann kann mittels Regel iii) die Berechnung von $\varphi(n)$ erfolgen.

Wir möchten nun die umgekehrte Frage stellen. Lässt sich zu einem gegebenen m entscheiden, welche natürlichen Zahlen durch die Eulersche φ-Funktion auf m abgebildet wird.

2.3.1. Rückwärtsberechnung von φ

In diesem Abschnitt widmen wir uns der folgenden Aufgabe:

Gegeben ist eine Zahl $m \in \mathbb{N}$ und gesucht sind alle Zahlen n mit $\varphi(n) = m$.

Dabei gibt es (vermutlich) entweder *keine Lösung* oder aber *mehr als eine Lösung*.

Exkurs 2.3.4 (Anzahl Lösungen von $\varphi(n) = m$)**.**

Im Jahr 1922 stellte R.D. Carmichael bereits die Vermutung auf, das es für jedes $n \in \mathbb{N}$ mindestens eine weitere Zahl $k \in \mathbb{N}$, $k \neq n$ gibt mit $\varphi(n) = \varphi(k)$. Tatsächlich hatte Carmichael im Jahr 1907 zunächst einen – falschen – Beweis für diese Aussage veröffentlicht.

Unter massivem Computereinsatz zeigten A. Schlafly und S. Wagon (1994), dass die Aussage zumindestens bis zu der (extrem großen) Zahl von $10^{(10^7)}$ richtig ist. Durch Erweiterung dieses Resultats verbesserte K. Ford diese untere Schranke auf $10^{\left(10^{10}\right)}$ (Ann. Math. 150:283–311, 1999).

Wie prüft man systematisch für eine Zahl $m \in \mathbb{N}$, ob sie im Wertebereich der eulerschen φ-Funktion liegt? Zunächst beobachten wir, dass nur gerade Zahlen und die 1 in Frage kommen.

Lemma 2.3.5. Sei $m \in \mathbb{N}$ und $m \geq 2$. Dann ist m gerade, wenn es ein $n \in \mathbb{N}$ gibt mit $\varphi(n) = m$. Außerdem ist $\varphi(n) = 1$ nur für $n = 1, 2$ erfüllt.

Beweis. Sei $m \in \mathbb{N}$. Wir nehmen an, es gibt ein $n \in \mathbb{N}$ so dass $\varphi(n) = m$ ist.

Wir unterscheiden zwei Fälle.

- **Fall 1:** $n \geq 3$**.**
  - **Fall 1.1:** $n = 2^k$**.** Nach Satz 2.3.2 gilt

$$m = \varphi(n) = \varphi(2^k) = 2^{k-1}(2-1). \tag{2.1}$$

    Nach Voraussetzung ist $n \geq 3$ und somit $k \geq 2$. Also ist mit (2.1) $m = \varphi(n)$ gerade und $m \neq 1$.
  - **Fall 1.2:** $n = 2^k \cdot n'$ **mit** $2 \nmid n'$ **und** $n' > 1$**.** In der Primfaktorzerlegung von n' taucht die Potenz von mindestens einer ungerade Primzahl p auf so dass $n = p^k \cdot n''$ mit $p \nmid n''$. Nach Satz 2.3.2 gilt

$$m = \varphi(n) = \varphi(p^k) \cdot \varphi(n'') = p^{k-1}(p-1) \cdot \varphi(n'') \tag{2.2}$$

    Da p ungerade ist, ist $p - 1$ gerade und mit 2.2 auch $\varphi(n)$ und somit m.
    Außerdem ist $p \geq 3$, also $(p-1) \geq 2$ und mit 2.2 $m \geq 2$ also insbesondere $m \neq 1$.
- **Fall 2:** $n < 3$**.** Es bleiben die beiden Möglichkeiten $n = 1$ oder $n = 2$. In beiden Fällen prüft man ganz leicht $\varphi(n) = 1$.

□

Bemerkung 2.3.6.

Das Lemma 2.3.5 impliziert nicht, dass es für ***jede*** gerade Zahl m tatsächlich auch eine natürliche Zahl $n \in \mathbb{N}$ gibt, so dass $\varphi(n) = m$. Beispielsweie gibt es keine natürliche Zahl die durch die eulersche φ-Funktion auf die 14 abgebildet wird, wie wir unten sehen werden.

Der Fall $m = 1$ ist durch Lemma 2.3.5 gelöst.

Für gerade $m \geq 2$ suchen wir alle Zahlen n mit $\varphi(n) = m$. Wir erreichen das Ziel nach der folgenden Definition in vier Schritten.

Definition 2.3.7 (Reine Lösung)**.** Sei $m \in \mathbb{N}$. Wir sagen $n \in \mathbb{N}$ ist eine ***reine Lösung*** für m wenn $\varphi(n) = m$ und $n = p^k$ für eine Primzahl p und $k \in \mathbb{N}$ ist.

Beispiel 2.3.8.

Für $m = 100$ sind die reinen Lösungen 101^1 und $5^3 = 125$ und es gilt $\varphi(101) = 100$ und $\varphi(5^3) = 5^2 \cdot (5-1) = 100$.

Schema F zur Rückwärtsberechnung von φ

Um die Urbilder bezüglich der eulerschen φ-Funktion einer natürlichen Zahl $m \in \mathbb{N}$ zu ermitteln, bedient man sich ihrer Multiplikativität. Das heißt um nicht alle Zahlen von 1 bis n untersuchen zu müssen hilft die Tatsache, dass wenn eine Zahl n auf m abgebildet wird, sich der Funktionswert (also $\varphi(n) = m$) in ein Produkt der Bilder der Primfaktoren aus der Primfaktorzerlegung von n zerlegen lässt. Ist nämlich

$$n = \prod_{i=1}^{t} p_i^{k_i} \qquad \Longrightarrow \qquad \varphi(n) = \prod_{i=1}^{t} \varphi(p_i^{k_i}) = \prod_{i=1}^{t} a_i$$

wobei die a_i nach Lemma 2.3.5 gerade oder 1 sind. Der letzte Fall tritt nur ein, wenn n durch 2 aber nicht durch 4 teilbar ist. Außerdem sind die Primfaktoren $p_i^{k_i}$ reine Lösungen der a_i. Man geht also zunächst auf die Sache nach Faktorisierungen von m in gerade Faktoren und sucht dann teilerfremde reine Lösungen, welche auf die Faktoren abbilden.

Wir geben jetzt drei Schritte an, welche dieses Programm zum Auffinden aller Lösungen für die Rückwärtsberechnung der φ-Funktion umsetzen. Im Anschluss werden wir beweisen, dass das Anwenden dieser Schritte zu korrekten Lösungen führt und auch jede korrekte Lösung gefunden wird.

► **Schritt 1.** Finde alle Faktorisierungen (Zerlegungen) von m mit ausschließlich geraden Faktoren.

► **Schritt 2.** Es werden für jede Faktorisierung $m = a_1 \cdot a_2 \cdots a_\ell$ aus Schritt 1 die folgenden Schritte durchgeführt.

- ► **Schritt 2.1.** finde für alle Faktoren a_i die Menge aller ***reinen Lösungen***. Es kommen jeweils nur zwei Zahlen in Frage:
  - Die Zahl $a_i + 1$ ist eine reine Lösung von a_i, wenn sie eine Primzahl ist. Dann sei $b_i^{[1]} = a_i + 1$
  - Für die größte Primzahl p_1 aus der Primfaktorzerlegung von $a_i = \prod_{j=1}^{\ell} p_j^{k_j}$ prüft man ob

$$\prod_{j=2}^{s} p_j^{k_j} = (p_1 - 1). \tag{2.3}$$

Ist dies der Fall, ist $b_i^{[2]} = p_1^{k_1+1}$ eine reine Lösung von a_i.

Gibt es für einen Faktor keine reine Lösung, dann wird die Faktorisierung $m = a_1 \cdot a_2 \cdots a_\ell$ verworfen und Schritt 2.2 nicht ausgeführt.

► **Schritt 2.2.** Nun werden je paarweise teilerfremde reine Lösungen der Faktoren miteinander multipliziert. Sei

$$b^{[s_1,s_2,\ldots,s_\ell]} = b_1^{[s_1]} \cdot b_2^{[s_2]} \cdots b_\ell^{[s_\ell]}$$

wobei die $b_i^{[s_i]}$ je eine reine Lösung des Faktors a_i sind.

► **Schritt 3.** Für jede gefundene ungerade Lösung b ist auch $2b$ eine Lösung.

Die drei Schritte in algorithmischer Schreibweise.

Algorithmus 2.3.9 (Auffinden aller Lösungen von $\varphi(n) = m$)**.**

Input: $m \in \mathbb{N}$.
Output: Alle Zahlen n mit $\varphi(n) = m$.
1. Setze $\mathbb{L} := \{\}$.
2. Bestimme alle möglichen Zerlegungen $m_u = a_{u1} \cdots a_{u\ell(u)}$ von m in *gerade* Faktoren a_{ui}.
3. Sei d die Anzahl aller Zerlegungen aus 2.
for $u = 1$ **to** d **do**
  for $i = 1$ **to** $\ell(u)$ **do**
    4. Berechne für a_{ui} die Primfaktorzerlegung von $a_{ui} = \prod_{j=1}^{v} p_j^{k_j}$
    if $a_{ui} + 1$ ist eine Primzahl **then**
      Dann setze $b_i^{[1]} = a_{ui} + 1$
    else if $\prod_{j=1}^{v} p_j^{k_j} = p_1 - 1$ **then**
      Dann setze $b_i^{[2]} = p_1^{k_1+1}$
    else
      Setze u=u+1 und Exit FOR
    end if
  end for
  5. Für alle Tupel $(b_1^{[s_1]}, \ldots, b_\ell^{[s_\ell]})$ mit *paarweise verschiedenen* $b_i^{[s_i]}$ füge Lösung in $\mathbb{L}$ ein:
    $\mathbb{L} := \mathbb{L} \bigcup \{b_1^{[s_1]} \cdots b_\ell^{[s_\ell]}\}$
end for
6. Für jede *ungerade* Lösung $n \in \mathbb{L}$ füge Lösung $2n$ in $\mathbb{L}$ ein:
  $\mathbb{L} := \mathbb{L} \bigcup \{2 \cdot n \ : \ n \in \mathbb{L}\ ,\ n \text{ ungerade}\}$
return $\mathbb{L}$.

Wir beweisen nun, dass der Algorithmus 2.3.9 alle und nur korrekte Lösungen findet. Dazu betrachten wir zunächst die insbesondere in Schritt 2 konstruierten Lösungen.

Lemma 2.3.10. Schritt 2.1 findet für jeden Faktor aus einer fixierten Faktorisierung $a_1 \cdot a_2 \cdots a_\ell$ aus Schritt 1 die Menge aller reinen Lösungen.

Beweis. Erfüllt die größte Primzahl p_1 aus der Primfaktorzerlegung von a_i die Gleichung (2.3), dann gilt

$$\varphi(p_1^{k_1+1}) = p_1^{k_1} \cdot (p_1 - 1) = p_1^{k_1} \cdot \prod_{j=2}^{\ell} p_j^{k_j} = \prod_{j=1}^{\ell} p_j^{k_j} = a_i.$$

Man prüft das alleine für die größte Primzahl aus der Faktorisierung, denn schon für die zweitgrößte Primzahl p_2 in der Primfaktorzerlegung von a_i gilt, dass

$$\prod_{\substack{j=1 \\ j\neq 2}}^{\ell} p_j^{k_j} = p_1 \cdot \prod_{j=3}^{\ell} p_j^{k_j} > p_2 - 1$$

und somit

$$\varphi(p_2^{k_2+1}) = p_2^{k_2} \cdot (p_2 - 1) \neq p_2^{k_2} \cdot \prod_{\substack{j=1 \\ j\neq 2}}^{\ell} p_j^{k_j} = \prod_{j=1}^{\ell} p_j^{k_j} = a_i.$$

Für eine Primzahl $m+1$ ist $\varphi(m+1) = (m+1-1) = m$. □

Lemma 2.3.11. Sei $a_1 \cdot a_2 \cdots a_\ell$ eine fixierte Faktorisierung aus Schritt 1, welche nicht in Schritt 2.1 verworfen wurde. Die in Schritt 2.2 Produkte von reinen Lösungen werden auf m abgebildet. Eine in Schritt 3 gefundene Zahl wird auch ebenfalls auf m abgebildet.

Beweis. Sei $b^{[s_1,s_2,\ldots,s_\ell]} = b_1^{[s_1]} \cdot b_2^{[s_2]} \cdots b_\ell^{[s_\ell]}$ eine in Schritt 2.2 gebildete Lösung. Da die $b_i^{[s_i]}$ paarweise teilerfremd sind, gilt für eine solche Lösung

$$\varphi(b^{[s_1,s_2,\ldots,s_\ell]}) = \varphi(b_1^{[s_1]} \cdot b_2^{[s_2]} \cdots b_\ell^{[s_\ell]}) = \varphi(b_1^{[s_1]}) \cdot \varphi(b_2^{[s_2]}) \cdots \varphi(b_\ell^{[s_\ell]}) = a_1 \cdot a_2 \cdots a_\ell = m.$$

Sei b eine ungerade gefundene Lösung. Es gelte also $\varphi(b) = m$ und 2 und b sind teilerfremd. Dann gilt aber $\varphi(2 \cdot b) = \varphi(2) \cdot \varphi(b) = 1 \cdot \varphi(b) = m$. □

Lemma 2.3.12. Es werden alle Lösungen gefunden.

Beweis. Sei $n \in \mathbb{N}$ eine Zahl mit $\varphi(n) = m$. Wir prüfen, ob diese Zahl mit Hilfe der vier Schritte gefunden wird. Betrachtet man die Primfaktorzerlegung von $n = 2^k \cdot \prod_{i=1}^{t} p_i^{k_i}$ in das Produkt paarweise verschiedener ungerader Primzahlen p_i für $i = 1, \ldots, t$ und einer Potenz von 2. Dann gibt es drei Fälle.

- ***Es ist*** $k = 0$**.** Dann ist n ungerade. Es ist $\varphi(n) = \prod_{i=1}^{t} \varphi(p_i^{k_i}) = \prod_{i=1}^{t} a_i$ wobei $p_i^{k_i}$ jeweils eine reine Lösung zu a_i ist. Die Faktoren a_i sind nach Lemma 2.3.5 alle gerade. Die Zahl n wird also bezüglich der Faktorisierung $\prod_{i=1}^{t} a_i = m$ gefunden.
- ***Es ist*** $k = 1$**.** Dann ist $\frac{n}{2}$ ungerade. Es ist außerdem $\varphi(n) = \varphi(2) \cdot \prod_{i=1}^{t} \varphi(p_i^{k_i}) = \prod_{i=1}^{t} \varphi(p_i^{k_i}) = \varphi(\frac{n}{2})$ und $\prod_{i=1}^{t} \varphi(p_i^{k_i}) = \prod_{i=1}^{t} a_i$ wobei $p_i^{k_i}$ jeweils eine reine Lösung zu a_i ist. Die Faktoren a_i sind nach Lemma 2.3.5 alle gerade. Die Zahl $\frac{n}{2}$ wird also bezüglich der Faktorisierung $\prod_{i=1}^{t} a_i = m$ gefunden. Da $\frac{n}{2}$ ungerade ist, findet man n im Schritt 3, in welchem alle ungeraden Lösungen mit 2 multipliziert werden.
- ***Es ist*** $k \geq 2$**.** Es ist $\varphi(n) = 2^k \cdot \prod_{i=1}^{t} \varphi(p_i^{k_i}) = 2^{k-1} \cdot \prod_{i=0}^{t} a_i$ wobei $p_i^{k_i}$ jeweils eine reine Lösung zu a_i und 2^k eine reine Lösung zu 2^{k-1} ist. Die Faktoren a_i und 2^{k-1} sind nach Lemma 2.3.5 und weil $k \geq 2$ ist alle gerade. Die Zahl n wird also bezüglich der Faktorisierung $2^{k-1} \cdot \prod_{i=1}^{t} a_i = m$ gefunden.

□

Beispiel 2.3.13.

Wir betrachten zunächst nur die Schritte 2.1 und 2.2 für $m = 400$ und die Zerlegung $m = 4 \cdot 100$.

Finde reine Lösungen für $a_1 = 4$.

$4 \xrightarrow{+1} 5$ ✓ - Primzahl. $\rightarrow 5$ ist reine Lösung.

$\downarrow$ faktorisieren

$2^2 \longrightarrow 2 \rightarrow 1 = (2-1)$ ✓ $\rightarrow 2^3$ ist reine Lösung.

Reine Lösungen sind: $\mathbf{b_1^{[1]} = 5}$ und $\mathbf{b_1^{[2]} = 8}$

Finde reine Lösungen für $a_2 = 100$.

$100 \xrightarrow{+1} 101$ ✓ - Primzahl. $\rightarrow 101$ ist reine Lösung.

$\downarrow$ faktorisieren

$5^2 \cdot 2^2 \longrightarrow 5 \rightarrow 2^2 = (5-1)$ ✓ $\rightarrow 5^3$ ist reine Lösung.

Reine Lösungen sind: $\mathbf{b_1^{[1]} = 101}$ und $\mathbf{b_1^{[2]} = 125}$

Wir kombinieren die reinen Lösungen

$$b^{[1,1]} = b_1^{[1]} \cdot b_2^{[1]} = 5 \cdot 101 = 505$$

$$b^{[2,1]} = b_1^{[2]} \cdot b_2^{[1]} = 8 \cdot 101 = 808$$

$$b^{[1,2]} = b_1^{[1]} \cdot b_2^{[2]} = 5 \cdot 125$$ ↯ - nicht teilerfremd

$$b^{[2,2]} = b_1^{[2]} \cdot b_2^{[2]} = 8 \cdot 125 = 1000$$

und erhalten drei Lösungen für die Zerlegung $400 = 4 \cdot 100$.

Beispiel 2.3.14.

Gegeben ist $m = 20$, gesucht sind alle Lösungen n von $\varphi(n) = 20$.

► **Schritt 1.** Es gilt: $m = 20 = 2 \cdot 10$, d.h. m hat zwei Zerlegungen in gerade Faktoren “20” und “$2 \cdot 10$”.

Für jede Zerlegung führe Schritte 2.1 und 2.2 durch.

Zerlegung $m = 20$.

► **Schritt 2.1**

Finde reine Lösungen für $a_1 = 20$.

$20 \xrightarrow{+1} 21 = 3 \cdot 7$ ↯ - keine Primzahl. → 21 ist keine Lösung.

$\downarrow$ faktorisieren

$5 \cdot 2^2 \longrightarrow 5 \to 2^2 = (5-1)$ ✓ → 5^2 ist reine Lösung.

Reine Lösungen sind: $\mathbf{b_1^{[1]} = 25}$

► **Schritt 2.2** Entfällt an dieser Stelle, die Faktorisierung besteht nur aus einem Faktor.

Zerlegung $m = 2 \cdot 10$.

► **Schritt 2.1**

Finde reine Lösungen für $a_1 = 2$.

$2 \xrightarrow{+1} 3$ ✓ - Primzahl. → 3 ist reine Lösung.

$\downarrow$ faktorisieren

$2 \longrightarrow 2 \to 1 = (2-1)$ ✓ → 2^2 ist reine Lösung.

Reine Lösungen sind: $\mathbf{b_1^{[1]} = 3}$ und $\mathbf{b_1^{[2]} = 2^2}$

Finde reine Lösungen für $a_2 = 10$.

$10 \xrightarrow{+1} 11$ ✓ - Primzahl. → 11 ist reine Lösung.

$\downarrow$ faktorisieren

$5 \cdot 2 \longrightarrow 5 \to 2 \neq (5-1)$ ↯ → 5^2 ist keine Lösung.

Reine Lösungen sind: $\mathbf{b_2^{[1]} = 11}$

► **Schritt 2.2** Lösungen für Zerlegung $m = 2 \cdot 10$ sind ...

$\mathbf{b^{[1,1]} = b_1^{[1]} \cdot b_2^{[1]} = 3 \cdot 11 = 33}$ ✓ - weil 3 und 11 teilerfremd sind

$\mathbf{b^{[2,1]} = b_1^{[2]} \cdot b_2^{[1]} = 4 \cdot 11 = 44}$ ✓ - weil 4 und 11 teilerfremd sind

Zusammentragen der bisherigen Lösungen:

$\mathbb{L} = \{25, 33, 44\}$

► **Schritt 3.** Für jede gefundene ungerade Lösung n füge $2n$ hinzu:

$\mathbb{L} = \{25, 2 \cdot 25,\ 33, 2 \cdot 33,\ 44\} = \{25, 50, 33, 66, 44\}$

Beispiel 2.3.15.

Gegeben ist $m = 14$, gesucht sind alle Lösungen n von $\varphi(n) = 14$.

► **Schritt 1.** Es gilt: $m = 14$, d.h. m hat genau eine Zerlegungen in gerade Faktoren "14" selbst.

Für diese Zerlegung führen wir nun Schritte 2.1 und 2.2 durch.

Zerlegung $m = 14$.

► **Schritt 2.1.**

Finde reine Lösungen für $a_1 = 14$.

$14 \xrightarrow{+1} 15 = 3 \cdot 5$ ↯ - keine Primzahl. $\rightarrow 15$ ist keine Lösung.

$\downarrow$ faktorisieren

$7 \cdot 2 \longrightarrow 7 \rightarrow 2 \neq (7-1)$ ↯ $\rightarrow 7^2$ ist keine Lösung.

Reine Lösungen sind: $\emptyset$

► **Schritt 2.2.** Entfällt an dieser Stelle, die Faktorisierung besteht nur aus einem Faktor, außerdem haben wir keine reinen Lösungen gefunden.

► **Schritt 3.** Dieser Schritt entfällt, es sind keine Lösungen gefunden worden.

Beispiel 2.3.16.

Gegeben ist $m = 40$, gesucht sind alle Lösungen n von $\varphi(n) = 40$.

► **Schritt 1.** Es gilt: $m = 40 = 4 \cdot 10 = 2 \cdot 20 = 2 \cdot 2 \cdot 10$, d.h. m hat 4 Zerlegungen in gerade Faktoren.

Für jede Zerlegung und jeden Faktor suche die reinen Lösungen:

Zerlegung $m = 40$

► **Schritt 2.1.**

Finde reine Lösungen für $a_1 = 40$.

$40 \xrightarrow{+1} 41$ ✓ - Primzahl. $\rightarrow 41$ ist reine Lösung.

$\downarrow$ faktorisieren

$5 \cdot 2^3 \longrightarrow 5 \rightarrow 1 = (2-1)$ ↯ $\rightarrow 5^2$ ist keine Lösung.

Reine Lösungen sind: $\mathbf{b_1^{[1]} = 41}$

► **Schritt 2.2.** Entfällt an dieser Stelle, die Faktorisierung besteht nur aus einem Faktor.

Zerlegung $m = 4 \cdot 10$

► **Schritt 2.1.**

Finde reine Lösungen für $a_1 = 4$.

$4 \xrightarrow{+1} 5$ ✓ - Primzahl. $\rightarrow 5$ ist reine Lösung.

$\downarrow$ faktorisieren

$2^2 \longrightarrow 2 \rightarrow 1 = (2-1)$ ✓ $\rightarrow 2^3$ ist reine Lösung.

Reine Lösungen sind: $\mathbf{b_1^{[1]} = 5}$ und $\mathbf{b_1^{[2]} = 8}$

Finde reine Lösungen für $a_2 = 10$.

$10 \xrightarrow{+1} 11$ ✓ - Primzahl. $\rightarrow 11$ ist reine Lösung.

$\downarrow$ faktorisieren

$5 \cdot 2 \longrightarrow 5 \rightarrow 2 \neq (5-1)$ ↯ $\rightarrow 5^2$ ist keine Lösung.

Reine Lösungen sind: $\mathbf{b_2^{[1]} = 11}$

► **Schritt 2.2.** Lösungen für Zerlegung $m = 4 \cdot 10$ sind

$\mathbf{b^{[1,1]}} = \mathbf{b_1^{[1]} \cdot b_2^{[1]} = 5 \cdot 11 = 55}$ ✓ - weil 5 und 11 teilerfremd sind

$\mathbf{b^{[2,1]}} = \mathbf{b_1^{[2]} \cdot b_2^{[1]} = 8 \cdot 11 = 88}$ ✓ - weil 8 und 11 teilerfremd sind

Zerlegung $m = 2 \cdot 20$

► **Schritt 2.1.**

Finde reine Lösungen für $a_1 = 2$.

$2 \xrightarrow{+1} 3$ ✓ - Primzahl. $\rightarrow 3$ ist reine Lösung.

$\downarrow$ faktorisieren

$2 \longrightarrow \quad 2 \rightarrow 1 = (2-1)$ ✓ $\rightarrow 2^2$ ist reine Lösung.

Reine Lösungen sind: $\mathbf{b_1^{[1]} = 3}$ und $\mathbf{b_1^{[2]} = 2^2}$

Finde reine Lösungen für $a_1 = 20$.

$20 \xrightarrow{+1} 21 = 3 \cdot 7$ ↯ - keine Primzahl. $\rightarrow 21$ ist keine Lösung.

$\downarrow$ faktorisieren

$5 \cdot 2^2 \longrightarrow \quad 5 \rightarrow 2^2 = (5-1)$ ✓ $\rightarrow 5^2$ ist reine Lösung.

Reine Lösungen sind: $\mathbf{b_1^{[2]} = 25}$

► **Schritt 2.2.** Lösungen für Zerlegung $m = 2 \cdot 20$ sind

$\mathbf{b^{[1,2]} = b_1^{[1]} \cdot b_2^{[2]} = 3 \cdot 25 = 75}$ ✓ - weil 3 und 25 teilerfremd sind

$\mathbf{b^{[2,2]} = b_1^{[2]} \cdot b_2^{[2]} = 4 \cdot 25 = 100}$ ✓ - weil 4 und 25 teilerfremd sind

Zerlegung $m = 2 \cdot 2 \cdot 10$

► **Schritt 2.1.**

Finde reine Lösungen für $a_1 = a_2 = 2$.

$2 \xrightarrow{+1} 3$ ✓ - Primzahl. $\rightarrow 3$ ist reine Lösung.

$\downarrow$ faktorisieren

$2 \longrightarrow 2 \rightarrow 1 = (2-1)$ ✓ $\rightarrow 2^2$ ist reine Lösung.

Reine Lösungen sind: $\mathbf{b_1^{[1]} = b_2^{[1]} = 3}$ und $\mathbf{b_1^{[2]} = b_2^{[2]} = 2^2}$

Finde reine Lösungen für $a_2 = 10$.

$10 \xrightarrow{+1} 11$ ✓ - Primzahl. $\rightarrow 11$ ist reine Lösung.

$\downarrow$ faktorisieren

$5 \cdot 2 \longrightarrow 5 \rightarrow 2 \neq (5-1)$ ↯ $\rightarrow 5^2$ ist keine Lösung.

Reine Lösungen sind: $\mathbf{b_2^{[2]} = 11}$

► **Schritt 2.2.** Lösungen für Zerlegung $m = 2 \cdot 2 \cdot 10$ sind

~~$\mathbf{b^{[1,1,2]}}$~~ $= \mathbf{b_1^{[1]} \cdot b_2^{[1]} \cdot b_3^{[2]} = 3 \cdot 3 \cdot 11 = \cancel{99}}$ ↯ - weil 3 und 3 nicht teilerfremd sind

$\mathbf{b^{[2,1,2]} = b_1^{[2]} \cdot b_2^{[1]} \cdot b_3^{[2]} = 4 \cdot 3 \cdot 11 = 132}$ ✓ - weil $3, 4$ und 11 teilerfremd sind

~~$\mathbf{b^{[1,2,2]}}$~~ $= \mathbf{b_1^{[1]} \cdot b_2^{[1]} \cdot b_3^{[2]} = 3 \cdot 4 \cdot 11 = \cancel{132}}$ ✓ - weil $3, 4$ und 11 teilerfremd sind - aber gleich $\mathbf{b^{[2,1,1]}}$

~~$\mathbf{b^{[2,2,2]}}$~~ $= \mathbf{b_1^{[1]} \cdot b_2^{[1]} \cdot b_3^{[2]} = 4 \cdot 4 \cdot 11 = \cancel{176}}$ ↯ - weil 4 und 4 nicht teilerfremd sind

Zusammentragen der bisherigen Lösungen:

$\{41,\ 55, 88,\ 75, 100,\ 132\}$

► **Schritt 3.** Für jede gefundene ungerade Lösung n füge $2n$ hinzu:

$\mathbb{L} = \{41,\ 42,\ 55,\ 110,\ 88,\ 75,\ 150,\ 100, 132\}$

3 Kryptographie

3.1. Der Satz von Euler

In diesem Abschnitt erinnern wir zunächst die multiplikativen Gruppe $(\mathbb{Z}_n^*, \odot_n)$ (kurz $\mathbb{Z}_n^*$ genannt). Diese Gruppe spielt eine wichtige Rolle im Herleiten der Sätze, die wir für das "RSA-Schema" benötigen (ein Verschlüsselungsverfahren aus der Kryptographie).

Mit Hilfe des Satzes von Bézout lassen sich über den erweiterten euklidischen Algorithmus Inverse in $\mathbb{Z}_n^*$ berechnen. Wir möchten uns nun noch damit beschäftigen, wie Potenzen der Form

$$a^k := a \odot_n a \odot_n \cdots \odot_n a \qquad \text{für } a \in \mathbb{Z}_n^*$$

berechnet werden können. Die Bebachtungen werden wir in dem Satz von Fermat und dem Satz von Euler festhalten.

Wir erinnern uns für $n \in \mathbb{N}$ an die Definition der Menge

$$\mathbb{Z}_n^* := \{k \in \mathbb{N} \ : \ k \leq n \ \text{ mit } \ \mathrm{ggT}(k, n) = 1\}$$

der teilerfremden Zahlen zu n. Schon bekannt ist, dass $(\mathbb{Z}_n^*, \odot_n)$ eine abelsche Gruppe bildet.

Satz 3.1.1. Es sei $n \in \mathbb{N}$ und $n \geq 2$, dann ist $(\mathbb{Z}_n^*, \odot_n)$ eine abelsche Gruppe.

Will man für ein $a \in \mathbb{Z}_n^*$ das passende Inverse $\overline{a}$ berechnen, so führt man folgende Berechnungsvorschrift aus:

1. Berechne via erweitertem Euklidischen Algorithmus Bézout-Multiplikatoren $s, t \in \mathbb{Z}$ mit $s \cdot a + t \cdot n = \mathrm{ggT}(a, n) = 1$.
2. Berechne $\overline{a} = \mathsf{Rest}(s, n)$.

Satz 3.1.2. Es sei $(G, \circ)$ eine endliche abelsche Gruppe mit neutralem Element e.

Für alle $a \in G$ gilt dann: $a^{|G|} = e$ ($|G|$ = Anzahl der Elemente von G).

Im Folgenden werden wir den Satz von Euler beweisen. Hierzu benötigen wir die Eigenschaften der Gruppe $\mathbb{Z}_n^*$, genauer gesagt rechnen wir mit $|\mathbb{Z}_n^*|$, der Anzahl der Elemente von $\mathbb{Z}_n^*$. Die Anzahl kennen wir schon, sie entspricht dem Wert der Eulerschen φ-Funktion von n.

3.1.1. Der Satz von Euler und der kleine Fermat

Wir wenden nun den Satz 3.1.2 auf Gruppen der Form $(\mathbb{Z}_n^*, \odot_n)$ an, und erhalten den Satz von Euler und den kleinen Satz von Fermat. Das folgende Korollar folgt direkt aus Satz 3.1.2 und der Tatsache, dass $(\mathbb{Z}_n^*, \odot_n)$ eine

endliche abelsche Gruppe ist:

Korollar 3.1.3. Es sei $n \in \mathbb{N}$ und $n \geq 2$. In der abelschen Gruppe $(\mathbb{Z}_n^*, \odot_n)$ gilt $a^{\varphi(n)} = 1$ für jedes $a \in \mathbb{Z}_n^*$.

Aus diesem Korollar folgern wir den Satz von Euler.

Satz 3.1.4 (Satz von Euler)**.** Sind $k, n \in \mathbb{N}$ teilerfremd (d.h. $\mathrm{ggT}(k,n) = 1$) mit $n \geq 2$ so folgt: $k^{\varphi(n)} \equiv 1 \pmod{n}$.

Satz 3.1.5 (kleiner Satz von Fermat)**.** Sind $k, p \in \mathbb{N}$ und p ist Primzahl so folgt: $k^p \equiv k \pmod{p}$.

Beweis. Es seien $k, p \in \mathbb{N}$ und es sei p eine Primzahl. Es gibt zwei Fälle zu untersuchen:

1. Gilt $\mathrm{ggT}(k,p) = 1$ so folgt wegen $\varphi(p) = p - 1$ aus dem Satz von Euler
$$k^p = k^{p-1} \cdot k \equiv 1 \cdot k \pmod{p}.$$
2. Gilt $\mathrm{ggT}(k,p) \neq 1$ so folgt sofort $\mathrm{ggT}(k,p) = p$ (weil p nur die Teiler 1 und p hat, kommt als gemeinsamer Teiler von k und p nur noch p in Frage).
Es gilt also $k = \ell \cdot p$ mit $\ell \in \mathbb{N}$ und damit ist auch $k^p = \ell^p \cdot p^p$ ein Vielfaches von p, dh. es gilt $\mathsf{Rest}(k,p) = 0 = \mathsf{Rest}(k^p, p)$ bzw. $k^p \equiv k \pmod{p}$.

□

Bemerkung 3.1.6.

Der kleine Satz von Fermat stellt keine Bedingung an $k \in \mathbb{N}$, im Gegensatz zum Satz von Euler bei dem $\mathrm{ggT}(k,n) = 1$ gelten muss ($k^{\varphi(n)} \equiv 1 \pmod{n}$ falls $\mathrm{ggT}(k,n) = 1$). Dafür muss aber der Modul (die Zahl p) eine Primzahl sein.

Im Beweis wurde ersichtlich, dass k und p entweder teilerfremd sind - dann folgt der kleine Satz von Fermat direkt aus dem Satz von Euler - oder k ein Vielfaches von p ist - dann ist k^p mod p nichts anderes als $k^p \equiv 0^p \equiv 0 \equiv k \pmod{p}$.

Beispiel 3.1.7 (Satz von Euler)**.**

1. Für die Zahlen 3 und 5 gilt $\mathrm{ggT}(3,5) = 1$.
Es gilt mit $\varphi(5) = 4$ also gilt nach Satz von Euler (Satz 3.1.4):
$3^4 = 81 \equiv 1 \pmod{5}$. Dies hätte man jedoch auch leicht selbst schließen können.
2. Mit etwas größeren Zahlen wird die Sache etwas schwieriger:
Für 7 und $22 = 2 \cdot 11$ gilt $\mathrm{ggT}(7,22) = 1$. Es gilt mit $\varphi(22) = 1 \cdot 10 = 10$ also
$7^{10} = 282475249 \equiv 1 \pmod{22}$ bzw. $\mathsf{Rest}(7^{10}, 22) = 1$

Beispiel 3.1.8.

In Satz 3.1.2 ist die Bedingung $\mathrm{ggT}(k,n) = 1$ immens wichtig. Wählt man beispielsweise nicht-teilerfremde Zahlen $k = 2$ und $n = 8$, so gilt:

$$\varphi(8) = \varphi(2^3) = 2^2 \cdot 1 \qquad \text{und} \qquad 2^4 = 16 \equiv 0 \pmod 8.$$

3.2. Schnelles Potenzieren

Wir werden nun mit Hilfe des Satzes von Euler eine Methode kennen lernen, wie man in Modulgleichungen schnell potenzieren kann und dann eine Methode, wie man Potenzieren mit großen Exponenten effizienter gestalten - die Anzahl der benötigten Multiplikationen reduzieren kann.

3.2.1. Schnelles Potenzieren in Modulgleichungen

Bevor wir die allgemeine Aussage in Lemma 3.2.2 festhalten, zunächst ein Beispiel.

Beispiel 3.2.1 (Satz von Euler)**.**

Eine der klassischen Anwendungen des Satzes von Euler ist die Vereinfachung von Potenzen in Modulgleichungen. Es soll berechnet werden:

$$\mathsf{Rest}(2^{26}, 21)$$

Hierzu verwenden wir $\varphi(21) = \varphi(3 \cdot 7) = 2 \cdot 6 = 12$ zusammen mit dem Satz von Euler. Es gilt dann nämlich

$$2^{12} \equiv 1 \pmod{21} \qquad \text{da } \mathrm{ggT}(2, 21) = 1.$$

Es folgt, dass für jeden Exponenten $m \in \mathbb{N}$ gilt

$$2^m \equiv 2^{m-\varphi(21)} \equiv 2^{m-12} \pmod{21}, \text{ denn}$$

$$2^m \equiv 2^{m-\varphi(21)+\varphi(21)} \equiv 2^{m-\varphi(21)} \cdot 2^{\varphi(21)} \equiv 2^{m-\varphi(21)} \cdot 1 \pmod{21}$$

Dies wendet man mehrfach an und erhält

$$2^{26} \equiv 2^{14} \equiv 2^2 \equiv 4 \pmod{21}$$

Insgesamt gilt also

$$2^{26} \equiv 2^{\mathsf{Rest}(26,12)} \pmod{21}$$

und ganz allgemein für zwei Zahlen $k, n \in \mathbb{N}$ mit $\mathrm{ggT}(k,n) = 1$ und $n \geq 2$ gilt für alle $m \in \mathbb{N}$, dass

$$k^m \equiv k^{\mathsf{Rest}(m,\varphi n)} \pmod{n}.$$

Wir halten den allgemeinen Fall im folgenden Lemma fest.

Lemma 3.2.2. Für $a, n \in \mathbb{N}$ mit $n \geq 2$ und $\mathbf{ggT(a, n) = 1}$ gilt $a^m \equiv a^{\mathsf{Rest}(m,\varphi(n))} \pmod{n}$ für alle $m \in \mathbb{N}$

Beweis. Es seien $k, m, n \in \mathbb{N}$ und es gelte $\mathrm{ggT}(k, n) = 1$. Dann gilt nach Satz von Euler $k^{\varphi(n)} \equiv 1 \pmod{n}$.

Sei nun $r := \mathsf{Rest}(m, \varphi(n))$ dann gibt es ein $\ell \in \mathbb{N}$ mit $m = \ell \cdot \varphi(n) + r$. Daraus folgern wir

$$\begin{aligned} k^m &= k^{\ell\cdot\varphi(n)+r} = k^{\ell\cdot\varphi(n)} \cdot k^r \\ &= \underbrace{(a^{\varphi(n)})}_{\equiv 1 \pmod{n}}{}^{\ell} \cdot k^r \equiv 1^{\ell} \cdot k^r \pmod{n} \end{aligned}$$

□

Wir betrachten ein weiteres Beispiel.

Beispiel 3.2.3.

Es soll berechnet werden der $\mathsf{Rest}(5^{12014}, 21)$. Hierzu verwenden wir $\varphi(21) = \varphi(3\cdot 7) = 2\cdot 6 = 12$ zusammen mit dem Satz von Euler: Es gilt $5^{12} \equiv 1 \pmod{21}$, da $\mathrm{ggT}(5, 21) = 1$ gilt.

Man kann also (s. Lemma 3.2.2) im Exponenten 12014 alle Vielfachen von 12 entfernen. Es gilt $12014 = 12 \cdot 10^3 + 12 + 2$ bzw. $\mathsf{Rest}(12013, 12) = 2$.

Es folgt, dass $5^{12013} \equiv 5^2 \pmod{21}$ gilt, und damit gilt

$$\mathsf{Rest}(5^{12013}, 21) = \mathsf{Rest}(5^2, 21) = \mathsf{Rest}(25, 21) = 4.$$

3.2.2. Allgemeines schnelles Potenzieren

Will man nun eine Potenz a^m berechnen und ist nicht an dem Rest modulo einer Zahl n interessiert, führt man mit dem naiven Ansatz $m - 1$ Multiplikatonen durch.

$$a^m = \underbrace{a \cdot a \cdot \ldots \cdot a}_{(m-1)-\text{oft}}$$

Das lässt sich beschleunigen, wenn man sukzessive Potenzen mit Zweierpotenzen im Exponenten berechnet. Dazu berechnet man die Binärdarstellung von m. Setze dazu $s = \lfloor \log_2(m) \rfloor$ und beobachte, dass die Binärdarstellung den Koeffizienten der folgenden Darstellung entspricht:

$$m = \sum_{i=0}^{s} b_i \cdot 2^i \qquad \text{mit } b_i \in \{0, 1\}$$

Beispiel 3.2.4.

Betrachten wir die natürliche Zahl 43. Es ist $s = \lfloor \log_2(43) \rfloor = 5$. Demnach lässt sich 43 schreiben als

$$\begin{aligned}\sum_{i=0}^{5} b_i \cdot 2^i &= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 43\end{aligned}$$

Dabei sind die Koeffizienten $b_5 = 1, b_4 = 0, b_3 = 1, b_2 = 0, b_1 = 1$ und $b_0 = 1$.

Mit Hilfe der Binärdarstellung lässt sich die Potenz a^m geschickt umschreiben. Es ist nämlich

$$\begin{aligned}a^m &= a^{\sum_{i=0}^{s} b_i \cdot 2^i} \\ &= a^{b_0 \cdot 2^0 + b_1 \cdot 2^1 + b_2 \cdot 2^2 + \ldots + b_s \cdot 2^s} \\ &= a^{b_0 \cdot 2^0} \cdot a^{b_1 \cdot 2^1} \cdot a^{b_2 \cdot 2^2} \cdot \ldots \cdot a^{b_s \cdot 2^s} \\ &= \left(a^{2^0}\right)^{b_0} \cdot \left(a^{2^1}\right)^{b_1} \cdot \left(a^{2^2}\right)^{b_2} \cdot \ldots \cdot \left(a^{2^s}\right)^{b_s} \\ &= \prod_{i \in \{0,1,\ldots,s\} : b_i = 1} a^{2^i}\end{aligned}$$

Hat man den folgenden Pool von Zahlen zur Verfügung, dann lässt sich a^m als Produkt von höchstens $s+1$ vielen Faktoren berechnen.

$$\mathcal{P}_{a,s} = \left\{a, a^2, a^4, a^8, \ldots, a^{2^i}, \ldots, a^{2^s}\right\}$$

Dieser Pool $\mathcal{P}_{a,s}$ beinhaltet $s+1$ Zahlen. Bleibt die Frage, ob es einfach ist, die Zahlen in $\mathcal{P}_{a,s}$ zu berechnen.

Lemma 3.2.5. Die Menge $\mathcal{P}_{a,s}$ lässt sich mit s Multiplikationen berechnen.

Beweis. Wir beweisen die Aussage mit vollständiger Induktion und führen die Induktion über s.

Induktionsverankerung: Im Fall $s = 1$ ist die Aussage einfach zu Prüfen. Die Menge $\mathcal{P}_{a,1} = \{2, 2^1\}$ besteht aus zwei Elementen und lässt sich mit einer Multiplikation $(2 \cdot 2)$ berechnen.

Induktionsannahme: Wir nehmen als Induktionsvoraussetzung an, dass die Menge $\mathcal{P}_{a,s}$ mit s Multiplikationen berechnen lässt.

Induktionsschluss: Für den Induktionsschluss beobachten wir, dass

$$\mathcal{P}_{a,s+1} = \mathcal{P}_{a,s} \cup \left\{a^{2^{s+1}}\right\} \qquad \text{und} \qquad a^{2^s} \in \mathcal{P}_{a,s}.$$

Also berechnet man in s Multiplilkationen $\mathcal{P}_{a,s}$ und mit einer Multiplikation $a^{2^{s+1}} = a^{2^s} \cdot a^{2^s}$ und erhält $\mathcal{P}_{a,s+1}$ in $s+1$ Multiplikationen. □

Bemerkung 3.2.6.

Praktisch berechnet man die Zahlen in $\mathcal{P}_{a,s}$ also Schritt für Schritt. Denn es ist allgemein $a^{2^i} = a^{2^{i-1}} \cdot a^{2^{i-1}}$ für alle $i \in \mathbb{N}$.

In algorithmischer Schreibweise liest sich das schnelle Potenzieren nun folgendermaßen.

Algorithmus 3.2.7 (SPOT)**.**

Input: $a, m \in \mathbb{N}$.
Output: a^m.

1. Berechne die Binärdarstellung von $m = \sum_{i=0}^{s} b_i \cdot 2^i$ mit $b_i \in \{0, 1\}$.
2. Setze $s = \lfloor \log_2(m) \rfloor$.
3. Setze $\mathcal{P}_{a,s} = \{a\}$.

for $i = 1$ **to** s **do**

5. Lese $a^{2^{i-1}}$ aus $\mathcal{P}_{a,s}$ aus und berechne $a^{2^i} = a^{2^{i-1}} \cdot a^{2^{i-1}}$.
6. Speichere a^{2^i} in $\mathcal{P}_{a,s}$.

end for

7. Lese alle a^{2^i} aus $\mathcal{P}_{a,s}$ aus, für welche $b_i = 1$.
8. Berechne

$$a^m = \prod_{i \in \{0,1,\ldots,s\} : b_i = 1} a^{2^i}.$$

Wir fassen unsere Beobachtung über die Komplexitätsreduktion in der folgenden Proposition zusammen.

Proposition 3.2.8. Seien $a, m \in \mathbb{N}$. Der Algorithmus SPOT benötigt höchstens $2 \cdot \lfloor \log_2(m) \rfloor$ Multiplikationen um a^m zu berechnen.

Beweis. Nach Lemma **??** brauchen wir $s = \lfloor \log_2(m) \rfloor$ viele Multiplikationen um $\mathcal{P}_{a,s}$ zu erzeugen. In Schritt 8 des Algorithmus wird mit diesen Werten in $\mathcal{P}$ der Wert a^m berechnet. Schritt 7 benötigt höchstens $s = \lfloor \log_2(m) \rfloor$ weitere Multiplikationen. □

Beispiel 3.2.9.

Wähle $a = 7$ und $m = 10$. Es sind naiv 9 Multiplikationen nötig um $7^{10} = 282475249$ zu berechnen.

Es ist allerdings $s = \lfloor \log_2(10) \rfloor = 3$. Der Algorithmus benötigt also höchstens 6 Multiplikationen. Tatsächlich sind es sogar nur 4.

Die Binärdarstellung von 10 ist $10 = 2+8 = 0 \cdot 1 + 1 \cdot 2 + 0 \cdot 4 + 1 \cdot 8$. Also sind $b_0 = 0, b_1 = 1, b_2 = 0, b_3 = 1$. Wir bestimmen die Menge $\mathcal{P}_{7,3} = \{a = 7, a^2 = 7^2 = 49, a^{2^2} = 49^2 = 2401, a^{2^3} = 2401^2 = 5764801\}$.

Damit ist dann $7^{10} = 49 \cdot 5764801 = 282475249$.

3.3. Kryptographische Anwendung: Das RSA-Verfahren

In der Kryptographie geht es darum, Nachrichten sicher zu transportieren, d.h. eine Nachricht $\mathcal{A}$ so zu verändern, dass ein Dritter aus der veränderten Nachricht $\mathcal{C}$ (dem Chiffrat) nicht auf die Originalnachricht $\mathcal{A}$ schließen kann.

Beispiel 3.3.1 (One-Time-Pad)**.**

Ein einfaches und sicheres Verfahren in der Kryptographie ist das Verwenden sogenannter „One-Time-Pads“ oder „One-Time-Keys“.

Ist ein Bit-String $\mathcal{A} := (b_1, \ldots, b_n) \in \{0,1\}^n$ der Länge n zu verschlüsseln, so generiert man einen zufälligen Bitstring $\mathcal{K} := (k_1, \ldots, k_n) \in \{0,1\}^n$ (der one-time-*key*) und addiert die passenden Elemente modulo 2, d.h. $\mathcal{C} := (b_1 \oplus_2 k_1, \ldots, b_n \oplus_2 k_n)$

Fasst man die Bitstrings als Vektoren über $\mathbb{Z}_2$ auf, und $\oplus$ als die elementweise Addion modulo 2, so gilt

$$\mathcal{C} = \mathcal{A} \oplus \mathcal{K} \qquad \text{und} \qquad C \oplus \mathcal{K} = \mathcal{A} \oplus (\mathcal{K} \oplus \mathcal{K}) = \mathcal{A} \oplus (0, \ldots, 0) = \mathcal{A}.$$

Das Entschlüsseln der Verschlüsselten Nachricht erfolgt also (genau wie das Verschlüsseln) durch Addtion von $\mathcal{K}$.

Sicherheit: Gelingt es, die Original-Nachricht $\mathcal{A}$ aus $\mathcal{C}$ zu ermitteln, so kennt man $\mathcal{A}$ und $\mathcal{C}$, und damit auch $\mathcal{K}$, denn es gilt:

$$\mathcal{A} \oplus \mathcal{C} = \overbrace{\mathcal{A} \oplus \mathcal{A}}^{(0,\ldots,0)} \oplus \mathcal{K} = \mathcal{K}$$

Dies bedeutet, dass das Verfahren im folgenden Sinne sicher ist: Ein Unbefugter hat genau dann eine Chance, eine Nachricht zu dechiffrieren, wenn er $\mathcal{K}$ bestimmen kann. Ist $\mathcal{K}$ „sicher“ und unbestimmbar, so kann auch $\mathcal{A}$ nicht aus $\mathcal{C}$ ermittelt werden.

Der Nachteil von One-Time-Pads ist, dass das Pad $\mathcal{K}$ beiden Kommunikationsparteien bekannt sein muss. Die Kommunikationsparteien müssen „sich einmal vorab treffen“ bzw. genauer gesagt, die Kommunikationsparteien müssen $\mathcal{K}$ vorab einmal über einen abhörsicheren Kanal austauschen.

Das Problem: Der Austausch von $\mathcal{K}$ kann nicht (wirklich) mittels eines (anderen) One-Time-Pads $\widetilde{\mathcal{K}}$ erfolgen, denn dazu müsste zunächst $\widetilde{\mathcal{K}}$ ausgetauscht werden, etc. etc.

Es muss also auf andere Weise ein sicherer Kommunikationskanal erzeugt werden. Genau dies leisten sogenannte *Public-Key*-Verfahren.

3.3.1. Das RSA-Schema.

Für die klassischen Chiffrierverfahren besteht ein Sicherheitsproblem darin, dass man den Schlüssel $\mathcal{K}_e$ der Codiervorschrift (ebenso wie die Schlüssel $\mathcal{K}_d$ der Decodiervorschrift) geheimhalten und zuvor vereinbaren muss (beim One-time-pad also $\mathcal{K}_e = \mathcal{K}$). Dies erzeugt ein „Henne-Ei“-Problem, denn der Schlüssel $\mathcal{K}_e$ kann nicht ohne Verschlüsselung sicher zwischen den Parteien kommuniziert werden.

Diffie und Hellman haben 1976 den (damals völlig neuartigen) Vorschlag der sogenannten Public-key-Kryptographie gemacht:

Jeder Teilnehmer des Systems besitzt

- einen ***öffentlichen Schlüssel*** e (zum ***E***ncodieren) und
- einen ***geheimen Schlüssel*** d (zum ***D***ecodieren).

Die Ver- und Entschlüsselalgorithmen E (encode) und D (decode) müssen nun die spezielle Eigenschaft haben, dass

- Für $\mathcal{C} := E(\mathcal{A})$ stets $D(\mathcal{C}) = \mathcal{A}$ gilt (D ist Umkehrfunktion zu E).
- $E(\mathcal{A})$ für eine Nachricht $\mathcal{A}$ mittels e leicht berechnet werden kann,
- $D(\mathcal{C})$ mit Kenntnis von d leicht zu berechnen ist,
- $D(\mathcal{C})$ ohne Kenntnis von d jedoch „sehr schwer“ zu berechnen ist. („*Trapdoor-Funktion*“)

Das bekannteste öffentliche Chiffriersystem ist das 1978 von Rivest, Shamir und Adleman vorgeschlagene RSA-Schema. Es beruht darauf, dass es schwer ist, eine Zahl m in ihre Primfaktoren zu zerlegen. In Folgenden Schema ist das Verfahren erklärt.

RSA-Verfahren

- **Schlüsselerzeugung:**
  - Wähle (sehr große) Primzahlen p, q und berechne $\mathsf{N} := p \cdot q$
    - Berechne $\varphi(\mathsf{N}) := (p-1) \cdot (q-1)$
    - Wähle $\mathsf{e} \in \mathbb{Z}^*_{\varphi(\mathsf{N})}$ d.h. e und $\varphi(\mathsf{N})$ sind teilerfremd.
    - Berechne $\mathsf{d} := \mathsf{e}^{-1}$ in $\mathbb{Z}^*_{\varphi(\mathsf{N})}$

      Bestimme s, t mittels erweitertem euklidischem Algorithmus, so dass gilt $s \cdot \mathsf{e} + t \cdot \varphi(\mathsf{N}) = 1$. Setze $\mathsf{d} = \mathsf{Rest}(s, \varphi(\mathsf{N}))$
    - Lösche p, q und $\varphi(\mathsf{N})$
  - Veröffentliche: e, N.
  - Halte geheim: d

- **Encodierung:**
  - Nachricht: $a \in \{2, \ldots, \mathsf{N}-1\}$
  - Berechne: $c := \mathsf{Rest}(a^{\mathsf{e}}, \mathsf{N})$

- **Decodierung:**
  - Berechne: $a = \mathsf{Rest}(c^{\mathsf{d}}, \mathsf{N})$

Zur Veranschaulichung folgt direkt ein Beispiel.

Beispiel 3.3.2.

▶ **Schlüsselerzeugung:** Sei $p = 5$, $q = 11$. Dann ist

$$\begin{aligned} \mathsf{N} &= 5 \cdot 11 = 55 \\ \varphi(\mathsf{N}) &= 4 \cdot 10 = 40 \end{aligned}$$

Der Encode-Exponent e muss teilerfremd zu $40 = 2^3 \cdot 5$ sein, darf also nicht durch 2 oder 5 teilbar sein.

Wird als Verschlüsselungs-Exponent $\mathsf{e} := 3$ gewählt, dann ist $\mathsf{d} = 27$ (da $e \cdot d = 81 \equiv 1 \pmod{40}$).

▶ **Encodierung:** Für die Nachricht $a := \mathbf{4}$ berechnen wir das Chiffrat $\mathsf{Rest}(a^{\mathsf{e}}, \mathsf{N})$

$$c = \mathsf{Rest}(\mathbf{4}^3, 55) = \mathsf{Rest}(64, 55) = \mathbf{9}$$

▶ **Decodierung:** Das Chiffrat $c = \mathbf{9}$ liefert wieder $a = \mathbf{4}$, denn es gilt $\mathsf{Rest}(c^{\mathsf{d}}, \mathsf{N}) = \mathsf{Rest}(9^{27}, 55) = \mathbf{4}$

Wir möchten im Folgenden nachweisen, dass das RSA-Verfahren auch "funktioniert".

Lemma 3.3.3 (Beweis der Korrektheit)**.** Beim RSA-Schema gilt: Ist $c := \mathsf{Rest}(a^{\mathsf{e}}, \mathsf{N})$ so gilt $a = \mathsf{Rest}(c^{\mathsf{d}}, \mathsf{N})$.

Beweis. Zu zeigen ist hier, dass $c^{\mathsf{d}} \equiv a \pmod{\mathsf{N}}$ gilt, denn dann folgt aus $2 \leq a \leq \mathsf{N} - 1$ die Behauptung.

Nach Konstruktion von d gilt

$$\mathsf{e} \odot_{\varphi(\mathsf{N})} \mathsf{d} = \mathsf{Rest}\left(\mathsf{e} \cdot \mathsf{d}, \varphi(\mathsf{N})\right) = \mathbf{1}$$

es gilt also

$$\mathsf{e} \cdot \mathsf{d} = \mathbf{1} + k \cdot \varphi(\mathsf{N}) \qquad \text{für ein } k \in \mathbb{Z}.$$

Es gilt nun:

$$\begin{aligned} c^{\mathsf{d}} = \mathsf{Rest}(a^{\mathsf{e}}, \mathsf{N})^{\mathsf{d}} \equiv \overbrace{(a^{\mathsf{e}})^{\mathsf{d}}}^{=a^{\mathsf{e}\cdot\mathsf{d}}} &\equiv a^{\overbrace{k\cdot\varphi(\mathsf{N})+\mathbf{1}}^{\mathsf{e}\cdot\mathsf{d}}} \pmod{\mathsf{N}} \\ &\equiv a^{k\cdot\varphi(\mathsf{N})} \cdot a^{\mathbf{1}} \pmod{\mathsf{N}} \\ &\overset{(\star)}{\equiv} a \pmod{\mathsf{N}} \end{aligned}$$

Für $\mathrm{ggT}(a, \mathsf{N}) = 1$ folgt die Gleichung $(\star)$ aus dem Satz von Euler, da dann $a^{\varphi(\mathsf{N})} \equiv 1 \pmod{\mathsf{N}}$ gilt.

Um die Gleichung $(\star)$ im allgemeinen Fall zu zeigen verwendet man den chinesischen Restsatz. Wir beweisen nun $(\star)$ komplett, dh. der obige Fall $\mathrm{ggT}(a, \mathsf{N}) = 1$ ist im Folgenden „inklusive".

Wir zeigen zunächst, dass gilt:

$$\left.\begin{aligned} a^{\mathsf{e}\cdot\mathsf{d}} &\equiv a \pmod{p} \\ a^{\mathsf{e}\cdot\mathsf{d}} &\equiv a \pmod{q} \end{aligned}\right\} (\star\star)$$

1. Annahme: es gelte $p | a$. Dann gelten $a \equiv 0 \pmod{p}$ und $a^{e\cdot d} \equiv 0 \pmod{p}$.
   Aus der Transitivität der Modulrechnung folgert man $a^{\mathsf{e}\cdot\mathsf{d}} \equiv a \pmod{p}$.
2. Annahme: es gelte $p \nmid a$. Es gilt gilt $a^{\varphi(\mathsf{N})} \equiv 1 \pmod{p}$, denn:

Die Zahlen a und p sind teilerfremd: Teiler von p sind 1 und p. Weiter gilt $p \not| a$, d.h. $\mathrm{ggT}(a,p) = 1$. Nach Satz von Euler gilt mit $\varphi(p) = p-1$ die Gleichung $a^{p-1} \equiv 1 \pmod{p}$ und damit

$$a^{\varphi(\mathsf{N})} = a^{(p-1)(q-1)} = \left(a^{p-1}\right)^{q-1} \equiv (1)^{q-1} \pmod{p}$$

Es gilt also $a^{\mathsf{e}\cdot\mathsf{d}} \equiv a \pmod{p}$ wegen

$$a^{\mathsf{e}\cdot\mathsf{d}} = a^{k\cdot\varphi(\mathsf{N})+1} = \left(a^{\varphi(\mathsf{N})}\right)^k \cdot a^1 \equiv (1)^k \cdot a \pmod{p}$$

3. Die Aussage $a^{\mathsf{e}\cdot\mathsf{d}} \equiv a \pmod{q}$ für q zeigt man analog zur Fallunterscheidung für p.

Betrachtet man $(\star\star)$, so sieht man, dass sowohl $\widetilde{x} := a^{\mathsf{e}\cdot\mathsf{d}}$ als auch $\widehat{x} := a$ jeweils Lösungen sind von

$$\left.\begin{array}{rcll} x & \equiv & a & \pmod{p} \\ x & \equiv & a & \pmod{q} \end{array}\right\} (\star\star\star)$$

Die Moduln p, q sind teilerfremd. Nach Chinesische Restsatz sind also die Lösungen von $(\star\star\star)$ äquivalent modulo $\mathsf{N} = p\cdot q$, es gilt $\widetilde{x} = \widehat{x} + k\cdot\mathsf{N}$ mit $k \in \mathbb{Z}$. Dies liefert $a^{\mathsf{e}\cdot\mathsf{d}} \equiv a \pmod{\mathsf{N}}$. □

Bemerkung 3.3.4 (Krypt-Analyse)**.**

Das Lemma 3.3.3 beweist, dass das RSA-Verfahren in so fern funktioniert, als dass aus dem erzeugten Chiffrat der ursprüngliche Message eindeutig ermittelt werden kann (in der Kenntnis des geheimen Schlüssels d). Ob das Verfahren aber auch "sicher" ist, muss gesondert diskutiert werden. Hierbei wird "sicher" ganz vielfältig interpretiert. Sicher in Bezug auf das entschlüsseln eines abgehörten Chiffrats oder das Auffinden des geheimen Schlüssels durch verschiedene Angriffe.

In Bezug auf das Berechnen des geheimen Schlüssels d aus dem öffentlichen Schlüssel e ist das RSA-Verfahren sicher, als dass man dazu $\varphi(\mathsf{N})$ benötigt.

Die Kenntnis von $\varphi(\mathsf{N}) = (p-1)(q-1)$ ist jedoch äquivalent zur Kenntnis von p und q, d.h. der Faktorisierung von N. Man kann $p+q$ und $p-q$ mit leicht durführbaren Operationen aus N und $\varphi(\mathsf{N})$ berechnen (und erhält so p und q):

$$\left.\begin{array}{rcl} \varphi(\mathsf{N}) = (p-1)(q-1) & = & pq - (p+q) + 1 \\ & = & \mathsf{N} - (p+q) + 1 \end{array}\right\} \quad \Rightarrow \quad (p+q) = \mathsf{N} + 1 - \varphi(\mathsf{N})$$

$$\left.\begin{array}{rcll} (p-q)^2 & = & p^2 - 2pq + q^2 & \\ & = & p^2 + 2pq + q^2 & -4\,pq \\ & = & (p+q)^2 & -4\,\mathsf{N} \end{array}\right\} \quad \Rightarrow \quad (p-q) = \sqrt{(p+q)^2 - 4\mathsf{N}}$$

Wer also aus N die Zahl $\varphi(\mathsf{N})$ effizient (bzw. schnell) berechnen kann, der kann N auch schnell faktorisieren. Für das Faktorisieren von Zahlen (z.B. von N) ist aber bisher kein effizienter („schneller") Algorithmus bekannt.

Dieses Argument ist beispielhaft für die Reduktion in der Komplexitätstheorie. Man reduziert das Berechnen des geheimen Schlüssels aus dem öffentlichen Schlüssel (und N) im RSA-Verfahren auf das Faktorisieren großer Zahlen. Ist das Berechnen des geheimen Schlüssels einfach, so auch das Faktorisieren. Das Berechnen des geheimen Schlüssels im RSA-Verfahrens ist einfacher als das Faktorisieren großer Zahlen. Ist man in der

Lage den Schlüssel (effizient) zu berechnen, so auch die Faktorisierung großer Zahlen. Das RSA-Verfahren ist also sicher in Bezug auf das Extrahieren des geheimen Schlüssels aus dem öffentlichen Schlüssel, so lange das Faktorisierung als schwer angenommen wird. Wird das Faktorisieren einfach (also ein effizientes Verfahren gefunden) heißt das im Umkehrschluss aber noch nicht, dass das RSA-Verfahren unsicher ist.

Exkurs 3.3.5 (Faktorisierungs-Challenge)**.**

Um ein Gefühl für die Schwierigkeit des Faktorisierens großer Zahlen zu vermitteln, dienen die folgenden Anhaltspunkte. Die RSA Laboratories haben regelmäßig Zahlen der Form $N = p \cdot q$ als Faktorisierungs-Challenge veröffentlicht:

- Im Jahr 1977 wurde die 129-stellige Dezimalzahl „RSA-129“ als Herausforderung für das RSA-System veröffentlicht (Preisgeld von \$100). Diese Zahl wurde erst 1994 faktorisiert, unter Beteiligung von 600 Freiwilligen und einem Rechenaufwand von ca. 5000 MIPS-Jahren.
- Im Jahr 2003 wurde mit 5-monatiger Rechenleistung auf 120 Maschinen die Zahl RSA-576 der Bitlänge 576 faktorisiert (Preisgeld \$10.000).
- Im Jahr 2005 wurde die Zahl RSA-640 der Bitlänge 640 faktorisiert (Preisgeld \$20.000).
- Das letzte Preisgeld betrug \$100.000 auf eine Zahl RSA-1024 (1024-Bit, 309 Dezimalstellen) und \$200.000 auf eine Zahl RSA-2048 (2048 Bit, 617 Dezimalstellen).

Obwohl mittlerweile für das Faktorisieren der RSA-Challenge-Zahlen keine Prämien mehr gezahlt werden, wurde im Dezember 2009 die Zahl RSA-768 faktorisiert.

Nicht vorhersehbare Entwicklungen, wie die Entwicklung deutlich schnellerer Algorithmen oder eines Quantencomputers, der die Faktorisierung von Zahlen durch Verwendung des Shor-Algorithmus effizient durchführen könnte, bergen zumindest für die mittel- und langfristige Sicherheit der RSA-verschlüsselten Daten gewisse Risiken.

3.3.2. Angriffe gegen das unmodifizierte RSA-Verfahren

In der Praxis wird das RSA-Verfahren in der oben beschrieben Form nicht eingesetzt - es hat in dieser Form mehrere Schwächen. Eine Schwäche entsteht durch zu kleine Klartexte a, wie sie z.B. im Chat auftreten.

Wenn der Klartext a und der Verschlüsselungsexponent e beide klein sind, so dass $a^e < N$ gilt, so folgt $c = \mathsf{Rest}(a^e, N) = a^e$. In diesem Fall kann ein Angreifer die e-te Wurzel aus c berechnen und das Chiffrat c auf diese Weise entschlüsseln, denn:

> *Gewöhnliches Wurzelziehen (z.B. $\sqrt[e]{c}$) aus ganzen Zahlen ist eine leicht zu bewerkstelligende Rechenoperation, nur das Wurzelziehen* modulo einer großen Zahl N *ist schwierig.*

Um solche Angriffe zu verhindern, wird ein *Padding-Verfahren* eingesetzt:

Man hängt der Bitfolge $(a)_2$ des Klartextes a eine zufällige Bitfolge R an. R hat eine vorgegebene Struktur, die unter mehreren möglichen zufällig gewählt wird (s. z.B. ISO 9796). Verschlüsselt wird nun die als Zahl aufgefasste Bitfolge $\widetilde{a} := (a, R)_2$, wobei nun gilt $(\widetilde{a})^e > N$ und damit $c = \mathsf{Rest}((\widetilde{a})^e, N) \neq (\widetilde{a})^e$.

Beispiel 3.3.6.

Sei $p = 7$, $q = 11$. Dann ist

$$\begin{aligned} N &= 7 \cdot 11 = 77 \\ \varphi(N) &= 6 \cdot 10 = 60 \end{aligned}$$

Als Verschlüsselungs-Exponent kommen nur Zahlen in Frage, die teilerfremd zu $60 = 2^2 \cdot 3 \cdot 5$ sind, die also nicht durch 2 oder 3 oder 5 teilbar sind.

Wird als Verschlüsselungs-Exponent $e := 13$ gewählt, dann ist $d = 37$ (da $d \cdot e = 381 \equiv 1 \pmod{\varphi(N)}$).

Wir verschlüsseln nun die Nachricht $a := 2$, d.h. wir berechnen $\mathsf{Rest}(2^e, N)$.

Das Berechnen von a^e kann man durch sukzessives Quadrieren beschleunigen: Aus der Binärdarstellung von $e = 13 = 8 + 4 + 1$ entnimmt man $a^e = a^8 \cdot a^4 \cdot a^1$ und berechnet durch Quadrieren (modulo N) sukzessive die Reste von a^2, $a^4 = \left(a^2\right)^2$, $a^8 = \left(a^4\right)^2$. Es gilt

$$\begin{aligned} 2^2 &= 4 && \equiv 4 \pmod{77} \\ 2^4 &= (4)^2 = 16 && \equiv 16 \pmod{77} \\ 2^8 &= (16)^2 = 256 && \equiv 25 \pmod{77} \end{aligned}$$

Und damit ergibt sich: $\mathsf{Rest}(a^e, N) = \mathsf{Rest}\left(25 \cdot 16 \cdot 2,\ 77\right) = \mathsf{Rest}(800, 77) = 30$.

Entschlüsselt man die Zahl 30 wieder auf dem selben Weg, berechnet man zunächst

$$\begin{aligned} 30^2 &= 900 && \equiv 53 \pmod{77} \\ 30^4 &\equiv 53^2 \equiv 2809 && \equiv 37 \pmod{77} \\ 30^8 &\equiv 37^2 \equiv 1369 && \equiv 60 \pmod{77} \\ 30^{16} &\equiv 60^2 \equiv 3600 && \equiv 58 \pmod{77} \\ 30^{32} &\equiv 58^2 \equiv 3364 && \equiv 53 \pmod{77} \end{aligned}$$

Beim Entschlüsseln von $c = 30$ entsteht also wieder $a = 2$, denn es gilt $\mathsf{Rest}(c^d, N) = 2$ wegen

$$\begin{aligned} 30^{37} &= 30^{32} \cdot 30^4 \cdot 30^1 \\ &\equiv 53 \quad \cdot 37 \quad \cdot 30 \equiv 764 \cdot 77 + 2 \equiv 2 \pmod{77} \end{aligned}$$

3.3.3. RSA-Signaturschema

Mit Hilfe eines Signaturverfahrens lässt sich die Authentizität der Verfasserschaft einer Nachricht überprüfen oder belegen. Der Sender versieht eine Nachricht mit einer Signatur, die ihn eindeutig als ihr Verfasser ausweist, er unterschreibt die Nachricht sprichwörtlich. Ein “Fälscher” ist nicht in der Lage diese Signatur nachzuahmen/zu fälschen.

Das Signaturverfahren mittels RSA fußt darauf, dass das Ver- und Entschlüsseln im RSA-Verfahren inverse Abbildungen sind. Ist (e, N) das öffentliche Schlüsselpaar und d der private Schlüssel, so gilt

- das Verschlüsseln einer Nachricht x liefert $E(x) := \mathsf{Rest}(x^{\mathsf{e}}, \mathsf{N})$,
- das Entschlüsseln eines Chiffrretextes z liefert $D(z) := \mathsf{Rest}(z^{\mathsf{d}}, \mathsf{N})$.

Die Funktion $D(\cdot)$ ist die Inverse zu $E(\,\cdot\,)$. Das heißt:

- Es gilt $D(\,E(x)\,) = x$. (Entschlüsseln des Chiffrats $E(x)$ liefert x).
- Es gilt aber auch umgekehrt $E(D(z)) = z$, denn:

$$\begin{aligned}
&\text{a)} \quad E(x) &&= \mathsf{Rest}\,(\,x^{\mathsf{e}}\,,\mathsf{N}) &&\equiv x^{\mathsf{e}} &&\pmod{\mathsf{N}}\\
&\text{b)} \quad E(\,D(z)\,) &&= \mathsf{Rest}\,(\,D(z)^{\mathsf{e}}\,,\mathsf{N}) &&\equiv D(z)^{blue e} &&\pmod{\mathsf{N}}\\
& &&&&\overset{\text{a)}}{\equiv} z^{\mathsf{d}\cdot\mathsf{e}} &&\pmod{\mathsf{N}}
\end{aligned}$$

Wir haben in der Krypt-Analyse des RSA-Verfahrens gezeigt, dass es *in der Praxis unmöglich ist* d aus den Zahlen e, N zu berechnen. „Praktisch unmöglich“ heißt hier, der Rechenaufwand ist größer als die momentan verfügbaren Rechner ermöglichen.

Dies bedeutet aber auch, dass jemand, der d nicht kennt, für x nicht $D(x) := \mathsf{Rest}(x^{\mathsf{d}}, \mathsf{N})$ berechnen kann. Nur derjenige, der die Schlüssel e, N erzeugt hat, kennt d und kann für x ein Paar $(x, D(x))$ berechnen.

Für ein Paar (x, z) lässt sich jedoch mit e, N leicht püfen, ob $z = D(x)$ gilt:

Ist $z = \mathsf{Rest}(x^{\mathsf{d}}, \mathsf{N})$ so muss $\mathsf{Rest}(z^{\mathsf{e}}, \mathsf{N}) = x$ gelten, wegen

$$\mathsf{Rest}(z^{\mathsf{e}}, \mathsf{N}) = \mathsf{Rest}((x^{\mathsf{d}})^{\mathsf{e}}, \mathsf{N}) = \mathsf{Rest}(x^{\mathsf{d}\cdot\mathsf{e}}, \mathsf{N}) = x.$$

Bemerkung 3.3.7 (Signaturschema mit RSA).

Person A (Alice) möchte eine Nachricht a an Person B (Bob) schicken und signieren.

A besitzt den privaten Schlüssel d und hat das Paar (e, N) veröffentlicht.

Wir gehen davon aus, dass B sicher sein kann, dass das Paar (e, N) tatsächlich von A stammt.

1. Person A verschlüsselt a mit dem *eigenen privaten Schlüssel* d, berechnet also $c := \mathsf{Rest}(a^{\mathsf{d}}, \mathsf{N})$
2. Person A sendet (a, c) an Person B.
3. Person B erhält ein Paar (x, y). Er prüft nun, ob (x, y) von A stammt:
   - Er entschlüsselt y mit A's öffentlichem Schlüssel e, d.h. er berechnet $z := \mathsf{Rest}(y^{\mathsf{e}}, \mathsf{N})$.
   - Gilt $x = z$ so stammt (x, y) tatsächlich von A, denn nur A ist in der Lage aus x das passende $y = \mathsf{Rest}(x^{\mathsf{d}}, \mathsf{N})$ zu berechnen.

Der Schritt 2 sollte verschlüsselt erfolgen:

Eigentlich verschlüsselt A die (besondere) Nachricht $\overline{a} := (a, c)$ mit B's öffentlichen Schlüsseln $(\mathsf{e}_B, \mathsf{N}_B)$ und erhält $\overline{c} = \mathsf{Rest}((\overline{a})^{\mathsf{d}_B}, \mathsf{N}_B)$.

Der Empfänger B entschlüsselt dann $\overline{c}$ mit seinem privaten Schlüssel d_B, erhält ein paar (x, y) und untersucht dies wie im dritten Schritt beschrieben.

4 Normen und Metriken

Im Folgenden wollen wir schon ein wenig Vorarbeit für die Numerik leisten und spezielle Normen und Metriken definieren. Wir beginnen zunächst mit Normen.

4.1. Norm - ein Längenbegriff für Vektoren

Unter einer Norm (von lat. *norma* »Richtschnur«) in einem Vektorraum versteht man anschaulich, das *jedem Vektor eine Länge zugeordnet wird*. Das kann man tatsächlich auf unterschiedliche Weisen realisieren. Dabei sollten aber bestimmte Eigenschaften des alltäglichen Längenbegriffs erhalten bleiben. Wir kennen schon die Euklidische Norm ($||x|| = \sqrt{x_1^2 + x_2^2 + x_3^2}$), welche im dreidimensionalen Vektorraum dem gewohnten Längenbegriff entspricht. Eine Länge sollte stets positiv sein, der und auch nur der Nullvektor sollte Länge Null haben, die Länge eines skalierten Vektors sollte sich um den Skalierungsfaktor verändern und die Länge der Summe zweier Vektoren sollte nicht länger sein, als die Längen der beiden Vektoren selbst.

Diese Eigenschaften nun sollte eine Abbildung besitzen, damit wir sie eine Norm nennen.

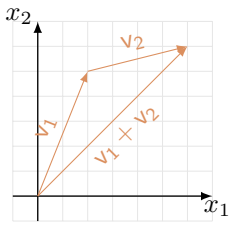
Definition 4.1.1 (Norm)**.** Es sei V ein Vektorraum über einem Körper $\mathbb{K}$.

Eine Abbildung $\mathsf{norm} : V \to \mathbb{R}$ nennt man ein ***Norm*** auf V, falls gelten:

N1.	$\mathsf{norm}(x) \geq 0$	$\forall x \in V$	(Positivität)
	$\mathsf{norm}(x) = 0 \quad \Leftrightarrow \quad x = 0$		
N2.	$\mathsf{norm}(\lambda \cdot x) = \lvert\lambda\rvert \cdot \mathsf{norm}(x)$	$\forall \lambda \in \mathbb{K},\ x \in V$	(Absolute Homogenität)
N3.	$\mathsf{norm}(x + y) \leq \mathsf{norm}(x) + \mathsf{norm}(y)$	$\forall x, y \in V$	(Dreiecksungleichung)

Bemerkung 4.1.2 (Dreiecksungleichung)**.**

Die Dreiecksungleichung verdankt ihren Namen der Anschauung, dass die beteiligten Vektoren v_1, v_2 und $v_1 + v_2$ die Seiten eines Dreiecks bilden.



Eine Norm hat also per Definition die drei Eigenschaften **N1**, **N2** und **N3**. Ein gewohntes Beispiel ist die aus dem Alltag bekannte Euklidische Norm.

Beispiel 4.1.3.

Die Abbildung $||\cdot||_2 : \mathbb{R}^2 \to \mathbb{R}$ mit $||x||_2 = \sqrt{x_1^2 + x_2^2}$ (Euklidische Norm) ist eine Norm, denn es gelten die definierenden Eigenschaften.

N1. $\sqrt{x_1^2 + x_2^2} \geq 0 \quad \forall x \in \mathbb{R}^2$
$\sqrt{x_1^2 + x_2^2} = 0 \quad \Leftrightarrow \quad x_1 = x_2 = 0 \Leftrightarrow x = \mathbf{0}.$

N2. $||\lambda \cdot x||_2 = \sqrt{(\lambda x_1)^2 + (\lambda x_2)^2} = \sqrt{\lambda^2(x_1^2 + x_2^2)} = |\lambda| \cdot \sqrt{x_1^2 + x_2^2} \quad \forall \lambda \in \mathbb{R},\ x \in \mathbb{R}^2.$

N3. $||x + y||_2 \leq ||x||_2 + ||y||_2$

4.1.1. p-Normen für Vektoren über den Vektorräumen $\mathbb{K}^n$

Es gibt viele verschiedene Normen auf Vektorräumen. Wir möchten nun eine wichtige Klasse von Normen, die p-Normen, kennenlernen. Die euklidische Norm, die wir schon kennen, ist eine p-Norm. Wir starten direkt mit der Definition.

Definition 4.1.4 (p-Normen für Vektoren)**.** Für $p \in \mathbb{N}$ nenne wir die Abbildung $\|\cdot\|_p : \mathbb{K}^n \to \mathbb{R}$ mit

$$\|x\|_p = (|x_1|^p + \ldots + |x_2|^p)^{\frac{1}{p}} = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}$$

die ***p*-Norm** auf $\mathbb{K}^n$.

Die Abbildung $\|\cdot\| : \mathbb{K}^n \to \mathbb{R}$ mit

$$\|x\|_\infty = \max\left\{|x_1|^p, \ldots, |x_n|^p\right\}.$$

bezeichnen wir als die ***∞-Norm*** oder ***Maximumsnorm*** auf $\mathbb{K}^n$.

Beispiel 4.1.5.

Für den Vektor $\begin{pmatrix} 1 \\ -2 \\ -3 \end{pmatrix} \in \mathbb{R}^3$ berechnen wir nun verschiedene p-Normen.

- In der Berechnung der 1-Norm $\|x\|_1 = |x_1| + \ldots + |x_n|$ kann man die äußere Wurzel weglassen, da für positive z stets $\sqrt[1]{z} = z$ gilt.

$$\left\| \begin{pmatrix} 1 \\ -2 \\ -3 \end{pmatrix} \right\|_1 = |1| + |-2| + |-3| = 6$$

- Die bekannteste p-Norm ist die 2-Norm, die wir schon als euklidische Norm kennen. Bei dieser Norm

können die Betragsstriche auch weggelassen werden, da stets $x_i^2 \geq 0$ gilt.

$$\left\| \begin{pmatrix} 1 \\ -2 \\ -3 \end{pmatrix} \right\|_2 = \sqrt{(1)^2 + (-2)^2 + (-3)^2} = \sqrt{1+4+9} = \sqrt{14}$$

- In der Berechnung der ∞-Norm $\|x\|_\infty = \max\{|x_1|, \ldots, |x_n|\}$ wird das *betragsmäßig* größte Element in x gesucht.

$$\left\| \begin{pmatrix} 1 \\ -2 \\ -3 \end{pmatrix} \right\|_\infty = \max\{|1|, |-2|, |-3|\} = 3$$

Bemerkung 4.1.6.

Verschiedene p-Normen haben alltägliche Anschauungen.

- Die 1-Norm wird auch *Manhatten-Norm* genannt. In dem rechtwinkligen Straßensystem Manhattens von einem Ort X zu einem Ort Y laufen, dann legt man einen Weg der Länge zurück, welche die 1-Norm angibt.
  Die 1-Norm tritt in bewegten Systemen auf, welche für verschiedene Raumdimensionen unterschiedliche Antriebe haben. Laufen diese Antriebe *nacheinander*, so lässt sich die Gesamtlaufzeit mit Hilfe der 1-Norm berechnen. Ein 2-D Plotter, dessen Motoren für links-rechts und für oben-unten nur *nacheinander* laufen, benötigt um vom Nullpunkt zum Punkt mit den Koordination a und b zu kommen die *Summe* beider Laufzeiten. Fährt er mit Geschwindigkeit 1 benötigt er eine Gesamtlaufzeit von
  $$|a| + |b| = \left\| \begin{pmatrix} a \\ b \end{pmatrix} \right\|_1 .$$
- Die 2-Norm, aka euklidische Norm, misst für einen Vektor $x \in \mathbb{R}^n$, wie wir schon wissen, die *tatsächliche Länge*.
- Die ∞-Norm tritt ebenfalls in bewegten Systemen auf, welche für verschiedene Raumdimensionen unterschiedliche Antriebe haben. Laufen diese Antriebe *gleichzeitig*, so lässt sich die Gesamtlaufzeit mit Hilfe der ∞-Norm berechnen. Ein 2-D Plotter, dessen Motoren für links-rechts und für oben-unten nur *gleichzeitig* laufen, benötigt um vom Nullpunkt zum Punkt mit den Koordination a und b zu kommen das *Maximum* beider Laufzeiten. Fährt er mit Geschwindigkeit 1 benötigt er eine Gesamtlaufzeit von
  $$\max\{|a| + |b|\} = \left\| \begin{pmatrix} a \\ b \end{pmatrix} \right\|_\infty .$$

Bemerkung 4.1.7.

Die ∞-Norm ist das Ergebnis einer Grenzwertbildung. Für einen festen Vektor $x \in \mathbb{K}^n$ sei $|x_1| = \max\{|x_1|^p, \ldots, |x_n|^p\}$. Dann ist

$$\|x\|_p = (|x_1|^p + \ldots + |x_2|^p)^{\frac{1}{p}} = \left(|x_1|^p \cdot \left(\frac{|x_1|^p}{|x_1|^p} + \ldots + \frac{|x_n|^p}{|x_1|^p} \right) \right)^{\frac{1}{p}} \tag{4.1}$$

und wir beobachten, dass $(|x_1|^p)^{\frac{1}{p}} = |x_1|^{p\cdot\frac{1}{p}} = |x_1|$ ist und für jeden der n Brüche in der Klammer gilt

$$0 \leq \frac{|x_i|^p}{|x_1|^p} \leq 1 \tag{4.2}$$

(die Zähler und Nenner sind stets positiv und es ist $|x_i| \leq |x_1|$ für alle $i \in 1, \dots n$). Darum gilt mit (4.1) und (4.2)

$$|x_1| \leq \|x\|_p \leq |x_1| \cdot n^{\frac{1}{p}}. \tag{4.3}$$

Vorgreifend auf die Analysis bemerken wir, dass $\lim_{p\to\infty} n^{\frac{1}{p}} = 1$ ist und mit (4.3) dann gilt

$$\lim_{p\to\infty} \|x\|_p = |x_1| = \|x\|_\infty.$$

Allgemein halten wir fest, dass für wachsendes p die betragsmäßig großen Komponenten des Vektors die p-Norm dominieren. In der ∞-Norm bestimmt alleine der betragsmäßig größte Eintrag den Wert der Norm.

Die p-Normen sind tatsächlich Normen, denn sie erfüllen die Eigenschaften **N1** bis **N3**.

Lemma 4.1.8. Für jedes $p \in \mathbb{N}$ und $p = \infty$ ist die p-Norm $\|\cdot\|_p$ eine Norm.

Beweis. Den Beweis überlassen wir dem Leser. □

4.2. Metrik - ein Abstandsbegriff auf Mengen

Grundsätzlich kann man sich für eine Menge von Objekten eine Geometrie konstruieren. Die einfachste Idee ist sicherlich, je zwei Elementen einer Menge einen Abstand zuzuordnen. Ein Abstand sollte stets positiv sein, symmetrisch (x ist von y genauso weit entfernt wie y von x) und der *Umweg* über ein drittes Element sollte nicht kürzer sein, als der direkte Weg. Genau diese drei Eigenschaften definieren eine Metrik.

Definition 4.2.1 (Metrik)**.** Es sei A eine beliebige Menge. Eine Abbildung $\mathrm{d} : A \times A \to \mathbb{R}$ ist eine Metrik, wenn gelten:

M1.	$\mathrm{d}(x,y) \geq 0$	$\forall x, y \in A$	(Positivität)
M2.	$\mathrm{d}(x,y) = \mathrm{d}(y,x)$	$\forall x, y \in A$	(Symmetrie)
M3.	$\mathrm{d}(x,y) \leq \mathrm{d}(x,z) + \mathrm{d}(z,y)$	$\forall x, y, z \in A$	(Dreiecksungleichung)

Beispiel 4.2.2.

Wir definieren für $A = \mathbb{R}^n$ die Abbildung $\mathrm{d}_2 : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ so, dass $\mathrm{d}_2(x,y) = \|x - y\|_2$. Diese Abbildung ordnet zwei Vektoren x und y die Länge (in der euklidischen Norm) des Differenzenvektors als Länge zu. Ist diese Abbildung auch wirklich eine Metrik? Wir überprüfen die Eigenschaften **M1** bis **M3**.

M1. Folgt direkt aus der Positivität der euklidischen Norm.
Denn nach **N1** ist $\mathrm{d}_2(x,y) = \|x - y\|_2 \geq 0$ für alle $x, y \in \mathbb{R}^n$.

M2. Folgt aus der Symmetrie der euklidischen Norm.
Denn nach **N2** ist $\mathrm{d}_2(x,y) = \|x-y\|_2 = \|-1\cdot(y-x)\|_2 = |-1|\cdot\|y-x\|_2 = \|y-x\|_2 = \mathrm{d}_2(y,x)$ für alle $x, y \in \mathbb{R}^n$.

M3. Folgt aus der Richtigkeit der Dreiecksungleichung der euklidischen Norm.
Denn nach **N3** ist $\mathrm{d}_2(x,y) = \|x-y\|_2 = \|x-z+z-y\|_2 \leq \|x-z\|_2 + \|z-y\|_2 = \mathrm{d}_2(x,z) + \mathrm{d}_2(z,y)$ für alle $x, y, z \in \mathbb{R}^n$.

4.2.1. Induzierte Metriken

Das Beispiel 4.2.2 legt nahe, dass man über jede Norm eine Metrik durch abbilden auf die Länge der Differenzenvektoren gegeben ist. Das ist in der Tat so. Im Prüfen der Metrikeigenschaften in dem Beispiel ist an keiner Stelle die konkrete Formel für die Euklidische Norm eingegangen. Wir halten allgemein fest.

Lemma 4.2.3. Für jede Norm $\|\cdot\|$ auf $\mathbb{K}^n$ ist die Abbildung $\mathrm{d} : \mathbb{K}\times\mathbb{K} \to \mathbb{R}$ mit $\mathrm{d}(x,y) = \|y-x\|$ eine Metrik.

Beweis. Ersetze in der Überprüfung der Eigenschaften **M1** bis **M3** in Beispiel 4.2.2 die euklidische Norm $\|\cdot\|_2$ durch eine beliebige Norm $\|\cdot\|$. □

Das Lemma 4.2.3 motiviert solche aus Normen entstehende Metriken auszuzeichnen.

Definition 4.2.4. Sei d eine Metrik auf einem Vektorraum $\mathbb{K}^n$ und sei $\|\cdot\|$ eine Norm auf diesem. Gilt $\mathrm{d}(x,y) = \|x-y\|$ für alle $x, y \in \mathbb{K}$, dann nennt man die Metrik d durch die Norm $\|\cdot\|$ ***induziert***.

Wir kennen nun schon viele Metriken auf Vektorräumen, denn jeder der unendlich vielen p-Normen induziert eine Metrik. Wir lernen nun noch eine weiter Metrik kennen, welche in der Codierungstheorie eine wichtige Rolle spielen wird.

4.2.2. Hammingabstand

In der Informationstheorie spielt der Hammingabstand eine wichtige Rolle. Er ist ein Maß für den Abstand zweier Zeichenfolgen oder Wörter (Strings) gleicher Länge. Dabei ist er nur sensibel für die Anzahl der Stellen, an denen sich diese Zeichenfolgen unterscheiden. Man zählt die benötigte Anzahl von Überschreibungen um die eine in die andere Zeichenfolge zu überführen. Anders ausgedrückt entspricht der Hammingabstand der Anzahl von Fehlern, welche die eine in die andere Zeichenfolge verändert haben könnte. Formal definieren wir den Hammingabstand folgendermaßen.

Definition 4.2.5 (Hammingabstand)**.** Es sei A eine beliebige Menge. Der ***Hammingabstand*** ist die Abbildung $\mathrm{dist} : A^n \times A^n \to \mathbb{R}$ mit

$$\mathrm{dist}(x,y) = |\,\{i \in \{1,\ldots,n\} \;:\; x_i \neq y_i\}\,|$$

welche zwei Tupeln $x, y \in A^n$ *die Anzahl* der Positionen, an denen sie sich unterscheiden, zuordnet.

Beispiel 4.2.6.

Für die beiden Tupel $x = (1, 0, 3)$ und $y = (0, 0, 0)$ aus $\mathbb{N}_0^3$ gilt $\text{dist}(x, y) = 2$.

$$\begin{array}{lccc} x = & (1, & 0, & 3) \\ y = & (0, & 0, & 0) \\ & \text{ungleich} & \text{gleich} & \text{ungleich} \end{array}$$

Es wird ersichtlich, dass sich x und y im ersten und dritten Eintrag unterscheiden - $x_1 \neq y_1$, $x_2 = y_2$ und $x_3 \neq_2$.

Bemerkung 4.2.7.

Beim Hammingabstand geht nicht ein *wie sehr* sich zwei einzelne Einträge unterscheiden, sondern nur *ob* sie sich unterscheiden. Es ist zum Beispiel

$$\text{dist}((3,0,0),(2,0,0)) = \text{dist}((27,0,0),(0,0,0)) = \text{dist}((47,0,0),(11,0,0)) = 1.$$

Wir rechnen nach, ob der Hammingabstand eine Metrik ist.

Lemma 4.2.8. Sei A eine beliebige Menge. Dann ist der Hammingabstand eine Metrik auf A^n.

Beweis. Wir prüfen, ob der Hammingabstand die Eigenschaften **M1** bis **M3** erfüllt.

M1. Folgt direkt aus der Definition des Hammingabstands. Eine Anzahl von Positionen ist stets positiv.

M2. Folgt direkt aus der Definition des Hammingabstands. Die Anzahl von Positionen an denen sich x von y unterscheidet ist gleich der Anzahl von Positionen an denen sich y von x unterscheidet für zwei beliebige Tupel $x, y \in A^n$.

M3. Es seien $x, y, z \in A^n$ Tupel der Länge n über A. Man kann immer das Tupel z erzeugen, wenn man in x die Stellen an denen sich x und z unterscheiden auf den Wert von z setzt. Dafür muss man genau $\text{dist}(x, z)$ viele Stellen ändern. Setzt man danach in z die Positionen an welchen sich z von y unterscheidet auf den Wer von y, erhält man y. Dazu muss man $\text{dist}(z, y)$ viele Positionen verändern. Dieser Vorgang startet bei x und endet bei y. Diese beiden Tupel können sich demnach höchstens um $\text{dist}(x, z) + \text{dist}(z, y)$ viele Stellen unterscheiden. Es gilt die Dreiecksungleichung $\text{dist}(x, y) \leq \text{dist}(x, z) + \text{dist}(z, y)$.

□

Bemerkung 4.2.9.

Dass in der Dreiecksungleichung tatsächlich ein $\leq$ steht (und nicht ein $=$ oder ein $<$), kann man sich klarmachen, wenn man …

- … $y = x = z$ setzt. Es gilt dann

$$0 = \text{dist}(x, y) \quad \text{und} \quad \text{dist}(x, z) + \text{dist}(z, y) = 0 + 0 = 0 = \text{dist}(x, y).$$

► $\dots x = y$ und $z \neq x$ setzt. Es gilt dann

$$0 = \text{dist}(x, y) \quad \text{und} \quad \text{dist}(x, z) + \text{dist}(z, y) = 2 \cdot \text{dist}(x, z) > 0 = \text{dist}(x, y).$$

Bemerkung 4.2.10.

Ist $A = \mathbb{K}$ ein Körper und interpretiert man Vektoren als Tupel, dann ist der Hammingabstand eine induzierte Metrik auf den Vektorräumen $\mathbb{K}^n$. Das sieht man ein, wenn man die 0-Norm $\| \cdot \|_0 : \mathbb{K}^n \to \mathbb{R}$ als $\|x\|_0 = \text{dist}(x, 0)$, die Anzahl der von 0 verschiedenen Einträge von x, definiert. (Dass es sich dabei tatsächlich um eine Norm handelt, ist eine Übungsaufgabe - die 0-Norm ist keine p-Norm!)

Es gilt dann für zwei beliebige Vektoren $x, y \in \mathbb{K}^n$, dass $\text{dist}(x, y) = \text{dist}(x - y, 0) = \|x - y\|_0$ ist - der Hammingabstand also von der 0-Norm induziert ist. Ein Eintrag des Vektors $x - y$ ist nämlich genau dann 0, wenn die beiden Vektoren in diesem übereinstimmen.

4.3. Geometrie unterschiedlicher (induzierter) Metriken

Bei vielen geometrische Objekte spielt Abstand und damit die zugrundeliegende Metrik eine definierende Rolle. Eine Kreisscheibe oder Kugel sind nichts anderes als Mengen von Punkten, deren Abstand zu einem fest definierten Punkt (dem Mittelpunkt) höchstens einem fest definiertem Wert (dem Radius) entspricht. Ändern wir die Metrik, also die Abstandsdefinition in einem Vektorraum, dann ändert sich die *Form* solcher geometrischer Objekte, deren Definition die Metrik involviert. Kugeln sind solche Objekte und wir definieren zunächst, was eine Kugel bezüglich einer allgemeinen Metrik ist.

Definition 4.3.1 (Kugeln)**.** Sei A eine Menge, auf welcher eine Metrik d definiert ist und $x \in A$. Dann definieren wir

$$\mathcal{B}_r(x, A) = \{y \in A : \mathrm{d}(x, y) \leq r\}$$

die ***Kugel*** (oder den Ball) um x mit Radius r in A und

$$\mathcal{S}_r(x, A) = \{y \in A : \mathrm{d}(x, y) = r\}$$

die ***Sphäre*** (oder die Oberfläche) um x mit Radius r in A
Ist aus dem Kontext klar, um welche Menge A es sich handelt, wird diese im Symbol auch weggelassen. Aus $\mathcal{B}_r(x, A)$ bzw $\mathcal{S}_r(x, A)$ wird dann $\mathcal{B}_r(x)$ bzw. $\mathcal{S}_r(x)$.
Handelt es sich bei der Metrik d um den Hammingabstand auf A^n, so nennen wir $\mathcal{B}_r(x, A^n)$ auch die ***Hamming-Kugel*** um x mit Radius r.

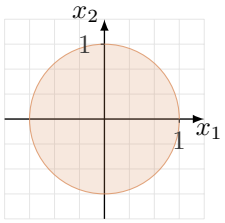
Beispiel 4.3.2.

Wir werden in Folgenden die Kugel $\mathcal{B}_1(0, V)$ zunächst im reelen Vektorraum, dann auch in Vektorräumen über endlichen Körpern bezüglich unterschiedlicher Metriken betrachten.

- Sei $V = \mathbb{R}^n$. Die Euklidische Norm induziert eine Metrik auf dem $\mathbb{R}^n$. Es gilt nach Lemma 4.2.3 $\mathrm{d}_2(v,w) = \|v-w\|_2$. Ist einer der Vektoren 0, dann gilt $\mathrm{d}_2(v,0) = \|v-0\|_2 = \|v\|_2$. Es ist also

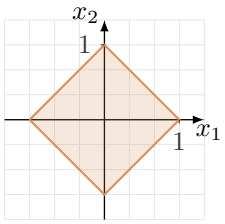
$$\mathcal{B}_1(0,\mathbb{R}^n) = \left\{ v \in \mathbb{R}^n : \mathrm{d}_2(v,0) = \|v\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2} \leq 1 \right\}.$$

  Das sind genau die Punkt, die im Kreis/in der Kugel mit Radius 1 um die 0 liegen wie Abbildung **??** für $n = 2$ veranschaulicht.



- Die 1-Norm induziert die Metrik $\mathrm{d}_1(v,w) = \|v-w\|_1$. Die durch diese Metrik definierte Kugel mit Radius 1 und die 0 lautet

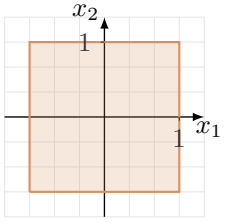
$$\mathcal{B}_1(0,\mathbb{R}^n) = \left\{ v \in \mathbb{R}^n : \mathrm{d}_1(v,0) = \|v\| = \sum_{i=1}^{n} |v_i| \leq 1 \right\}.$$



- Die ∞-Norm, welche jeden Vektor $v \in \mathbb{R}^n$ auf $\|v\|_1 = \max_i |v_i|$ abbildet, induziert die Metrik $\mathrm{d}_\infty(v,w) = \|v-w\|_\infty$. Die durch diese Metrik definierte Kugel mit Radius 1 und die 0 lautet

$$\mathcal{B}_1(0,\mathbb{R}^n) = \left\{ v \in \mathbb{R}^n : \mathrm{d}_{\max}(v,0) = \|v\| = \max_i |v_i| \leq 1 \right\}.$$

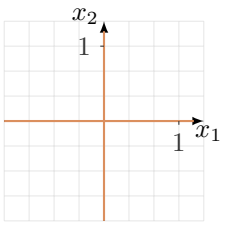
  Das sind genau die Punkte, die in einem auf der Seite(nfläche) liegenden Quadrat/Würfel um die 0 liegen.



- Die Kugel im $\mathbb{R}^2$ bezüglich des Hammingabstands ist

$$\mathcal{B}_1(0,\mathbb{R}^n) = \{ v \in \mathbb{R}^n : \mathrm{dist}(v,0) \leq 1 \}.$$

  Das sind genau die Punkte, die auf den Achsen liegen. Für einen anderen Mittelpunkt sind es die verschobenen Achsen durch diesen Punkt.



- Die Kugel im $(\mathbb{F}_3)^2$ bezüglich des Hammingabstands ist

$$\mathcal{B}_1(0, \mathbb{R}^n) = \{v \in \mathbb{R}^n : \text{dist}(v, 0) \leq 1\}.$$

Das sind genau die Punkte, die "auf den Achsen liegen". Für einen anderen Mittelpunkt sind es die verschobenen Achsen durch diesen Punkt. So sind die Hammingkugeln um die Punkte $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ und $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ mit Radius 1 im $\mathbb{R}^2$ gegeben durch

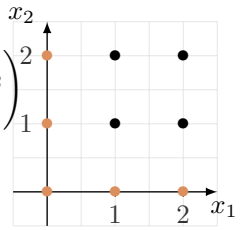
$$\mathcal{B}_1\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, (\mathbb{F}_3)^2\right) = \left\{\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \end{pmatrix}\right\}$$

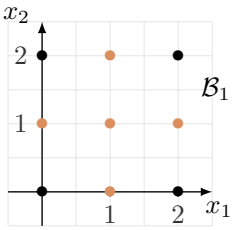
und

$$\mathcal{B}_1\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}, (\mathbb{F}_3)^2\right) = \left\{\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}\right\},$$

alle Punkte die Hammingabstand höchstens 1 zu $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ bzw. $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ haben. Die zugehörigen Diagramme sind:

$\mathcal{B}_1\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, (\mathbb{F}_3)^2\right)$





$\mathcal{B}_1\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}, (\mathbb{F}_3)^2\right)$

Hammingabstand ist Norm -> $\mathbb{F}_2^3$ - Kugel um $(0, 0, 0)$ - Diagramm

Die Hammingkugeln sind also ein nicht sehr anschauliches Konstrukt auf den ersten Blick. Ist aber, wie im Kontext der Codierungstheorie, der Hammingabstand der *richtige* Abstandsbegriff, kann man diese Vorstellung eines Abstandes für die Anschauungen wiederum intuitiv verwenden (wir haben Metriken mit dem Ziel definiert, intuitiv von Abständen reden zu können). Wir werden im Kontext der Codierungstheorie häufig von Hammingkugeln um Codewörter oder Wörter sprechen.

4.4. Norm - ein allgemeiner Längenbegriff

Grundsätzlich haben wir Normen auf beliebigen Vektorräumen definiert. Die Elemente der Vektorräume können die unterschiedlichsten mathematischen Objekte sein. Es kann sich um reelle Funktionen, Matrizen, Polynome und vieles mehr handeln.Die Vektorräume müssen auch nicht zwangsläufig unendlich-dimensional und damit isomorph

zu einem K^n sein. Die p-Vektornormen sind für diese Vektorräume nicht definiert. Wir möchten uns nun mit Matrixnormen beschäftigen.

4.4.1. Matrixnormen

Die Menge der reellen $m \times n$-Matrizen ist ein $\mathbb{R}$-Vektorraum. Die Matrizen $\mathrm{E}^{[i,j]}$ deren einziger von 0 verschiedener Eintrag der Eintrag in der i-ten Zeile und j-ten Spalte gleich 1 lautet, bilden eine Basis des Vektorraums $\mathbb{R}^{m\times n}$. Demnach hat dieser Vektorraum Dimension $m \cdot n$ und ist isomorph zum Vektorraum $\mathbb{R}^{m\cdot n}$.

Die Isomorphie ist über die Existenz einer isomorphen (Ein Isomorphismus ist eine bijektive lineare Abbildung) Abbildung definiert (die Abbildung, welche Vektoren aus dem Vektorraum $\mathbb{R}^{m\times n}$ mit Vektoren aus dem Vektorraum $\mathbb{R}^{m\cdot n}$ identifiziert.). Mit Hilfe dieser Abbildung kann aus jeder Norm auf dem $\mathbb{R}^{m\cdot n}$ eine Norm auf dem $\mathbb{R}^{m\times n}$ gewonnen werden.

Definition 4.4.1 (Matrixnormen über Normen)**.** Sei $f : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\cdot n}$ ein Isomorphismus und $\|\cdot\|^v$ eine Vektornorm auf $\mathbb{R}^{m\cdot n}$. Dann nennen wir die Abbildung $\|\cdot\|^* : \mathbb{R}^{m\times n} \to \mathbb{R}$ mit $\|A\|^* = \|f(A)\|^v$ die über die Vektornorm $\|\cdot\|^v$ gebildete Matrixnorm $\mathbb{R}^{m\times n}$.

Diese Matrixnormen sind tatsächlich Normen auf dem Vektorraum $\mathbb{R}^{m\times n}$.

Satz 4.4.2. Sei eine Norm $\|\cdot\|^v$ auf $\mathbb{R}^{m\cdot n}$ gegeben. Die über $\|\cdot\|^*$ gebildete Matrixnorm auf $\mathbb{R}^{m\times n}$ ist eine Norm.

Beweis. Wir prüfen, ob $\|\cdot\|^*$ die Eigenschaften **N1** bis **N3** erfüllt.

N1. Da $\|\cdot\|^v$ **N1** erfüllt, ist $\|A\|^* = \|f(A)\|^v \geq 0$ für alle $A \in \mathbb{R}^{m\times n}$.
Da $\|\cdot\|^v$ **N1** erfüllt, gilt genau dann $\|A\|^* = \|f(A)\|^v = 0$, wenn $f(A) = 0$ ist. Da f aber eine injektive lineare Abbildung ist, gilt genau dann $f(v) = 0$, wenn $v = 0$ ist. Also gilt genau dann $\|A\| = 0$, wenn $A = 0$ ist.

N2. Da f eine lineare Abbildung ist, und $\|\cdot\|^v$ **N2** erfüllt, ist

$$\|\lambda \cdot A\|^* = \|f(\lambda \cdot A)\|^v = \|\lambda \cdot f(A)\|^v = \lambda \cdot \|f(A)\|^v = \lambda \cdot \|A\|^*$$

für alle $\lambda \in \mathbb{R}$ und $A \in \mathbb{R}^{m\times n}$.

N3. Da f eine lineare Abbildung ist, und $\|\cdot\|^v$ **N3** erfüllt, ist

$$\begin{aligned}\|A + B\|^* = \|f(A + B)\|^v &= \|f(A) + f(B)\|^v \\ &\leq \|f(A)\|^v + \|f(B)\|^v = \|A\|^* + |B\|^*\end{aligned}$$

für alle $A, B \in \mathbb{R}^{m\times n}$.

□

Bevor wir uns Beispiele von über Normen gebildete Matrixnormen auf dem $\mathbb{R}^{m\times n}$ anschauen können, einigen wir uns auf einen zugrundeliegenden Isomorphismus f. Dazu wählen wir die Abbildung $f : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\cdot n}$ mit $f(E^{[i,j]}) = e^{[i+j]}$, welche als der ***kanonische*** Isomorphismus von $\mathbb{R}^{m\times n}$ nach $\mathbb{R}^{m\cdot n}$ bezeichnet wird. Diese

Abbildung bildet eine Matrix A auf den Vektor ab, welcher aus den aneinander gehängten Spalten der Matrix A entspricht.

Beispiel 4.4.3.

Betrachte den Vektorraum der 2×2 Matrizen. Die Basis besteht aus den vier Matrizen

$$E^{[1,1]} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad E^{[2,1]} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad E^{[1,2]} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \quad E^{[2,2]} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

Den kanonischen Isomorphismus angewendet erhalten wir

$$f\left(\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\right) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad f\left(\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}\right) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix},$$

$$f\left(\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}\right) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad f\left(\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\right) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Eine beliebige Matrix $A = \begin{pmatrix} 4 & 2 \\ 0 & -2 \end{pmatrix}$ wird also abgebildet auf

$$\begin{aligned} f\left(\begin{pmatrix} 4 & 2 \\ 0 & -2 \end{pmatrix}\right) &= f(4 \cdot E^{[1,1]} + 0 \cdot E^{[2,1]} + 2 \cdot E^{[1,2]} + (-2) \cdot E^{[2,2]}) \\ &= 4 \cdot f(E^{[1,1]}) + 0 \cdot f(E^{[2,1]}) + 2 \cdot f(E^{[1,2]}) + (-2) \cdot f(E^{[2,2]}) \\ &= 4 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 0 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + 2 \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + (-2) \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \\ 2 \\ -2 \end{pmatrix}. \end{aligned}$$

Beispiel 4.4.4.

Wir betrachten unterschiedliche über p-Normen gebildete Matrixnormen.

- Betrachten wir zunächst die über der euklidischen Norm auf dem $\mathbb{R}^{m \cdot n}$ gebildete Matrixnorm. Dann gilt

$$\|A\|_2^* = \|f(A)\|_2 = \sqrt{\sum_{j=1}^{n} \sum_{i=1}^{m} |A_{ij}|^2}.$$

  Diese Norm wird auch als ***Frobeniusnorm*** bezeichnet.

► Betrachten wir dann die über der ∞-Norm auf dem $\mathbb{R}^{m\cdot n}$ gebildete Matrixnorm. Dann gilt

$$\|A\|_\infty^* = \|f(A)\|_\infty = \max_{\substack{j=1,\dots,n\\ i=1,\dots,m}} \{|A_{ij}|\}\,.$$

Diese Norm entspricht dem Betrag des betragsmäßig größten Eintrags.

Klassischer Weise ist man an Matrixnormen mit der folgenden Eigenschaft interessiert.

Definition 4.4.5. Eine Abbildung $f : \mathbb{R}^{m\times n} \times \mathbb{R}^{n\times \ell} \to \mathbb{R}_+$ heißt ***submultiplikativ***, wenn für alle $A \in \mathbb{R}^{m\times n}$ und $B \in \mathbb{R}^{n\times \ell}$ gilt

$$f(A \cdot B) \leq f(A) \cdot f(B).$$

Leider haben nicht alle über Vektornormen definierte Matrixnormen dieses Eigenschaft.

Beispiel 4.4.6.

Die über der ∞-Norm gebildete Matrixnorm ist nicht submultiplikativ. Wählt man $A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ und $B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ mit $A \cdot B = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$ ist

$$\|A \cdot B\|_\infty^* = 2 \geq 1 \cdot 1 = \|A\|_\infty^* \cdot \|B\|_\infty^*,$$

Es gibt einen alternativen Weg submultiplikative Matrixnormen aus Vektornormen zu gewinnen.

Natürliche Matrixnormen

Für jede Vektornorm lässt sich, wie in Definition 4.4.1 gesehen, eine zugehörige Matrixnorm definieren. Die dort gezeigte Konstruktion ist jedoch nicht die einzige Möglichkeit. Alternativ lässt sich eine anschauliche Matrixnorm als so genannte Operatornorm definieren. Diese Normen *messen* für eine Matrix A, um welchen Faktor sich die Länge eines Vektors, welcher mit der Matrix A multipliziert wird, höchstens verändert. Dabei wird die Länge des Vektors $x \in \mathbb{R}^n$ und die Länge des Vektors $A \cdot x \in \mathbb{R}^m$ mit der zugrundeliegenden Vektornorm *vermessen* und der Quotient gebildet.

Definition 4.4.7 (natürliche Matrixnormen)**.** Sei $\|\cdot\|^v$ eine Norm auf $\mathbb{R}^n$ und $\mathbb{R}^m$. Dann nennen wir eine Abbildung $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}_+$ mit

$$\|A\| = \max_{\substack{x\in\mathbb{R}^n\\ x\neq 0}} \left\{ \frac{\|A \cdot x\|^v}{\|x\|^v} \right\}$$

die durch die Norm $\|\cdot\|^v$ induzierte ***(natürliche) Matrixnorm*** (oder Operatornorm).

Bemerkung 4.4.8.

Die Definition 4.4.7 der induzierten Matrixnorm involviert ein Optimierungsproblem über eine überabzählbar große Menge von Vektoren. Die Berechnung erscheint auf den ersten Blick schwierig zu sein. In Lemma 4.4.14 werden wir alternative Formeln für drei wichtige induzierte Matrixnormen kennen lernen.

Wie rechnen nach, ob induzierte Matrixnormen Normen sind.

Lemma 4.4.9. Gegeben sei eine Norm $\|\cdot\|^v$ auf $\mathbb{R}^n$ und $\mathbb{R}^m$. Die durch $\|\cdot\|^v$ induzierte Matrixnorm $\|\cdot\|$ auf $\mathbb{R}^{n\times m}$ ist eine Norm.

Beweis. Wir prüfen, ob die induzierte Matrixnorm $\|\cdot\|$ die Eigenschaften **N1** bis **N3** erfüllt.

N1. Da $\|\cdot\|^v$ **N1** erfüllt, ist für alle $x \in \mathbb{R}^n \setminus \{0\}$ der Bruch $\frac{\|A\cdot x\|^v}{\|x\|^v} \geq 0$ und somit $\|A\| \geq 0$ für alle $A \in \mathbb{R}^{m\times n}$.
Da $\|\cdot\|^v$ **N1** erfüllt, ist $\|A\| = 0$ falls für alle $x \in \mathbb{R}^n \setminus \{0\}$ gilt $Ax = 0$. Diese ist aber alleine für die $0 \in \mathbb{R}^{m\times n}$ (die Nullmatrix) der Fall.

N2. Da $\|\cdot\|^v$ **N2** erfüllt, ist

$$\|\lambda \cdot A\| = \max_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} \left\{ \frac{\|\lambda \cdot A \cdot x\|^v}{\|x\|^v} \right\} = \max_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} \left\{ \lambda \cdot \frac{\|A \cdot x\|^v}{\|x\|^v} \right\}$$

$$= \lambda \cdot \max_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} \left\{ \frac{\|A \cdot x\|^v}{\|x\|^v} \right\} = \lambda \cdot \|A\|$$

für alle $\lambda \in \mathbb{R}$ und $A \in \mathbb{R}^{m\times n}$.

N3. Da $\|\cdot\|^v$ **N3** erfüllt, ist

$$\|A + B\|^* = \max_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} \left\{ \frac{\|(A+B)\cdot x\|^v}{\|x\|^v} \right\} = \max_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} \left\{ \frac{\|A\cdot x + B\cdot x\|^v}{\|x\|^v} \right\}$$

$$\overset{\text{N3}}{\leq} \max_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} \left\{ \frac{\|A\cdot x\|^v + \|B\cdot x\|^v}{\|x\|^v} \right\}$$

$$\leq \max_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} \left\{ \frac{\|A\cdot x\|^v}{\|x\|^v} \right\} + \max_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} \left\{ \frac{\|B\cdot x\|^v}{\|x\|^v} \right\}$$

für alle $A, B \in \mathbb{R}^{m\times n}$. Die letzte Ungleichung gilt, weil das Maximum der einzelnen Summanden nicht für das gleiche x auftreten muss.

□

Eine erste *Vereinfachung* im Berechnen von induzierten natürlichen Matrixnormen erhält man, wenn man einsieht, dass das Maximum nicht über alle von 0 verschiedenen Vektoren zu ermitteln ist, sondern es die Menge der normierten Vektoren schon tut.

Lemma 4.4.10. Sei $\|\cdot\|^v$ eine Norm auf $\mathbb{R}^n$ und $\mathbb{R}^m$. Dann lässt sich für jede Matrix $A \in \mathbb{R}^{n\times m}$ die induzierte natürliche Matrixnorm berechnen mittels

$$\|A\| = \max_{\substack{x\in\mathbb{R}^n \\ \|x\|=1}} \{\|A\cdot x\|^v\}.$$

Beweis. Jeder Vektor $x \in \mathbb{R}^n$ lässt sich als das Produkt von $\|x\|$ mit seinem normierten Vektor $\tilde{x} = \frac{1}{\|x\|}\cdot x$ schreiben. Offensichtlich gilt

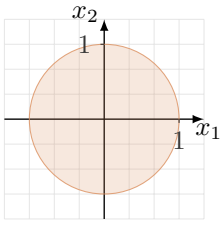
$$\frac{\|A\cdot x\|^v}{\|x\|^v} = \frac{\|A\cdot\tilde{x}\cdot\|x\|^v\|^v}{\|x\|^v} = \frac{\|A\cdot\tilde{x}\|^v\cdot\|x\|^v}{\|\tilde{x}\|^v} = \frac{\|A\cdot\tilde{x}\|^v}{\|\tilde{x}\|^v} = \|A\cdot\tilde{x}\|^v.$$

Das Maximum ändert sich also nicht, wenn man die zugrundeliegende *Prüfmenge* auf normierte Vektoren reduziert. □

Wir erhalten mit Lemma 4.4.10 die schon zu Beginn angedeutete Anschauung induzierter natürlicher Matrixnormen. Die der Maximumsbildung zu Grunde liegende Menge von normierten Vektoren ist die Sphäre $\mathcal{S}_1(0,\mathbb{R}^n)$ mit Radius 1 um den Ursprung. Um die induzierte Matrixnorm einer Matrix A zu berechnen, betrachtet man schlicht das Bild dieser Sphäre und sucht den Vektor mit der größten Länge bezüglich der zugrunde liegenden Vektornorm $\|\cdot\|^v$.

Beispiel 4.4.11.

Wir betrachten die durch die euklidischen Norm $\|\cdot\|_2$ auf $\mathbb{R}^2$ induzierte Matrixnorm auf $\mathbb{R}^{2\times2}$ zur euklidischen Norm $\|\cdot\|_2$. Wähle dazu $A = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$.



Induzierte natürliche Matrixnormen sind submultiplikativ.

Lemma 4.4.12. Sei $\|\cdot\|$ eine Norm auf $\mathbb{R}^n$ und $\mathbb{R}^m$. Die induzierte natürliche Matrixnorm $\|\cdot\|$ auf $\mathbb{R}^{n\times m}$ ist submultiplikativ.

Beweis. Übungsaufgabe. □

Natürliche Matrixnormen berechnen

Für manche durch p-Normen induzierte natürliche Matrixnormen gibt es einfache Formeln.

Definition 4.4.13. Wir bezeichnen die durch die p-Norm auf $\mathbb{R}^n$ und $\mathbb{R}^m$ induzierte natürliche Matrixnorm auf $\mathbb{R}^{m\times n}$ als p-Matrixnorm und schreiben für diese ebenfalls $\|\cdot\|_p$.

Lemma 4.4.14. Sei $A \in \mathbb{R}^{m\times n}$. Es gelten folgende Formel.

- $\|A\|_1 = \max\limits_{j=1,\cdots,n} \sum\limits_{i=1}^{m} |A_{ij}|$.
- $\|A\|_2 = \sqrt{k}$, wobei k der größte Eigenwert von $A^T \cdot A$ ist.
- $\|A\|_\infty = \max\limits_{i=1,\cdots,m} \sum\limits_{j=1}^{n} |A_{ij}|$.

Bemerkung 4.4.15.

Aus offensichtlichen Gründen wir die 1-Matrixnorm als *Spaltensummennorm*, die ∞-Matrixnorm als *Zeilensummennorm* und die 2-Matrixnorm als *Spektralnorm* (das Spektrum einer Matrix ist die Menge der Eigenwerte) bezeichnet.

Beispiel 4.4.16.

Für die Matrix $A = \begin{pmatrix} 4 & 2 \\ 0 & -2 \end{pmatrix}$ berechnen wir die folgenden Normen.

$$\|A\|_1 = \max\{|4| + |0|, |2| + |-2|\} = \max\{4, 4\} = 4$$

$$\|A\|_\infty = \max\{|4| + |2|, |0| + |-2|\} = \max\{6, 2\} = 6$$

Außerdem ist $A^T \cdot A = \begin{pmatrix} 16 & 8 \\ 8 & 16 \end{pmatrix}$ dessen charakteristische Polynom

$$\det\left(\begin{pmatrix} 16 & 8 \\ 8 & 16 \end{pmatrix}\right) = (16-\lambda)^2 - 16 = (x-12)\cdot(x-20)$$

die Eigenwerte $k_1 = 20$ und $k_2 = 12$ liefert. Es ist demnach

$$\|A\|_2 = \sqrt{\max\{12, 20\}} = 20.$$

5 Codes

5.1. Einführung

"*Die Kodierungstheorie ist die mathematische Theorie der fehlererkennenden und -korrigierenden Codes. Solche Codes kommen überall dort zur Anwendung, wo digitale Daten gegen bei Übertragung oder Speicherung auftretende Fehler geschützt werden sollen. Beispiele sind die Kommunikation mit Objekten im Weltraum und das Speichern von Daten auf einer CD.*" – Wikipedia

Bei der Übertragung von Informationen können aus vielen Gründen Fehler auftreten. Je nachdem wo diese Fehler auftreten, können die Folgen gravierend sein. Ziel der Codierungstheorie ist es, redundante Information in die Datenübertragung einzubauen, so dass

- zum einen Fehler bei der Datenübertragung *bemerkt* werden können und
- zum anderen Fehler auch *korrigiert* werden können.

Exkurs 5.1.1 (Anwendungen von Codes in der Praxis)**.**

Solche Kommunikation findet zum Beispiel zwischen Computern (Internet), aber auch innerhalb von Computern statt, z.B. zwischen der CPU (Hauptrecheneinheit) und dem Arbeitsspeicher eines Rechners.

Etwas spektakulärer ist noch die Anwendung in der Kommunikation mit Sateliten, hier ist es zum Beispiel nicht ohne weiteres möglich bei einer unverständlichen Datenübertragung "einfach mal hinzufahren und nachzufragen".

Codes treten aber auch im ganz gewöhnlichen Alltag auf: Beim Telefonieren werden (in der Deutschen Sprache) zum Buchstabieren oft Vornamen verwendet. Die "DIN 5009" regelt wie in Deutschland am Telefon buchstabiert werden soll (s. nachfolgende Tabelle). Die Verlängerung der Buchstaben zu einem Namen fügt weitere Silben an, die resultierenden längeren Wörter lassen sich leichter voneinander unterscheiden:

Die Buchstaben "B", "C" und "D", unterscheiden sich in der Gesprochenen Form "Beh", "Zeh" und "Deh" nur um einen Buchstaben, die Wörter klingen quasi gleich. Die zugehörigen Namen "Berta", "Cäsar" und "Dora" sind dagegen klar unterscheidbar. Diese Wörter werden erst verwechselbar, wenn die Datenleitung völlig verrauscht ist, d.h. wenn bei der Übertragung fast jede Information verloren geht.

Ein weiteres Beispiel für Codes im Alltag ist die Verwendung des Wortes "Jun**o**" statt des Monatsnamens "Juni" weil dieser dem Monatsnamen "Jul**i**" zu ähnlich ist. In der Sprache der Codierungstheorie haben "Juni" und "Juli" einen Hammingabstand von 1 (s. Def. 5.2.4).

Buchstabe	Code DIN 5009	Code Nato	Buchstabe	Code DIN 5009	Code Nato	Buchstabe	Code DIN 5009	Code Nato
A	Anton	Alpha	J	Julius	Juliett	S	Samuel	Sierra
B	Berta	Bravo	K	Kaufmann	Kilo	T	Theodor	Tango
C	Cäsar	Charlie	L	Ludwig	Lima	U	Ulrich	Uniform
D	Dora	Delta	M	Martha	Mike	V	Viktor	Victor
E	Emil	Echo	N	Nordpol	November	W	Wilhelm	Whiskey
F	Friedrich	Foxtrot	O	Otto	Oscar	X	Xanthippe	X-Ray
G	Gustav	Golf	P	Paula	Papa	Y	Ypsilon	Yankee
H	Heinrich	Hotel	Q	Quelle	Quebec	Z	Zacharias	Zulu
I	Ida	India	R	Richard	Romeo			

5.1.1. Wiederholung: Endliche Körper

Erinnerung

Die Menge $\mathbb{Z}_n$ ist mit der Resteaddition und Restemultiplikation ein Körper, wenn n eine Primzahl ist. Des Weiteren gibt es für jede Primzahlpotenz p^k einen endlichen Körper dieser Ordnung (meint Anzahl der Elemente - Ordnung der abelschen Gruppe der Addition). Die Konstruktion dieser Körper mit Ordnung p^k ist allerdings nicht leicht zu skizieren, es handelt sich dabei um Gallois-Erweiterungen von $\mathbb{Z}_n$. Wir halten also fest.

Satz 5.1.2. Sei $p \in \mathbb{N}$ eine Primzahl. Dann ist der Restklassenring $\mathbb{Z}_p$ eine Körper. Für jede Primzahl p und jede positive natürliche Zahl n existiert (bis auf Isomorphie) genau ein Körper mit p^n Elementen. Sei $n \in \mathbb{N}$. Alle endlichen Körper der Ordnung n sind isomorph.

Definition 5.1.3 (Endliche Körper)**.** Sei $p \in \mathbb{N}$ eine Primzahl und $n \in \mathbb{N}$. Dann bezeichnen wir den (bis auf Isomorphie) eindeutigen Körper mit p^n Element als $\mathbb{F}_p^n$. Dabei heißt p die ***Charakteristik*** von $\mathbb{F}_{p^n}$. Unendliche Körper haben Charakteristik 0.

5.2. Allgemeine Codes über allgemeinen Alphabeten

In diesem Abschnitt führen wir die wesentlichen Begriffe der Codierungstheorie ein.

Definition 5.2.1.

- Ein ***Alphabet*** $A = \{a_1, \ldots, a_k\}$ ist eine endliche Menge von Symbolen a_i mit $k \geq 2$.
- Ein ***Wort*** w über dem Alphabet A ist ein Tupel $(a_{i_1}, \ldots, a_{i_\ell}) \in A^\ell$ von Elementen aus A.
- Die Menge aller Wörter bezeichnet man mit $A^* = \cup_{\ell=1}^{\infty} A^\ell$.

Wörter werden "liegend" notiert, z.B. ist $(1,1,0)$ ein Wort aus $\{0,1\}^3$, also ein Codewort der Länge 3 über dem Alphabet $\{0,1\}$. Um (horizontalen) Platz zu sparen verwendet man bei der Notation das Transponieren:

$$\begin{pmatrix}1\\1\\0\end{pmatrix}^{\mathrm{T}} = (1,1,0)\ .$$

Wir wiederholen noch mal die Definition des Hamming-Abstands speziell für Wörter eines Alphabets.

Definition 5.2.2.

- Ein ***Code*** C über dem Alphabet A ist eine nicht-leere Teilmenge von A^*.
- Ein ***Blockcode*** der Länge $n \in \mathbb{N}$ über A ist eine nicht-leere Teilmenge von A^n.
- Codes über dem Alphabet $\mathbb{F}_2 = \{0,1\}$ nennt man ***binäre*** Codes.

Wir betrachten im Folgenden Blockcodes über endlichen Körpern der Form $\mathbb{F}_p$.

Beispiel 5.2.3.

Die Menge der Wörter in $(\mathbb{F}_3)^2$ (Wörter der Länge 2 über $\mathbb{F}_3 = \{0,1,2\}$) ist:

$$(\mathbb{F}_3)^2 = \left\{ \begin{pmatrix}0\\0\end{pmatrix}^T, \begin{pmatrix}0\\1\end{pmatrix}^T, \begin{pmatrix}0\\2\end{pmatrix}^T, \begin{pmatrix}1\\0\end{pmatrix}^T, \begin{pmatrix}1\\1\end{pmatrix}^T, \begin{pmatrix}1\\2\end{pmatrix}^T, \begin{pmatrix}2\\0\end{pmatrix}^T, \begin{pmatrix}2\\1\end{pmatrix}^T, \begin{pmatrix}2\\2\end{pmatrix}^T \right\}$$

Ein Blockcode C der Länge 2 über $\mathbb{F}_3$ ist eine Teilmenge von $(\mathbb{F}_3)^2$, zum Beispiel

$$C := \left\{ \begin{pmatrix}0\\0\end{pmatrix}^T, \begin{pmatrix}0\\2\end{pmatrix}^T, \begin{pmatrix}1\\1\end{pmatrix}^T, \begin{pmatrix}2\\0\end{pmatrix}^T, \begin{pmatrix}2\\2\end{pmatrix}^T \right\}$$

Definition 5.2.4 (Hamming-Abstand)**.** Es seien $x, y \in A^n$ zwei Wörter gleicher Länge über dem Alphabet A.

Der ***Hammingabstand*** von x zu y ist *die Anzahl* der Positionen, an denen sich x und y unterscheiden:

$$\mathrm{dist}(x,y) := |\,\{i \in \{1,\dots,n\} \ :\ x_i \neq y_i\}\,|$$

Beispiel 5.2.5.

Für $x := (1,0,3)$ und $y := (0,0,0)$ gilt $\mathrm{dist}(x,y) = 2$ wegen

$$\left.\begin{array}{l} x = (\,1,0,3\,) \\ y = (\,0,0,0\,) \end{array}\right\} \quad x_1 \neq y_1 \qquad x_2 = y_2 \qquad x_3 \neq y_3$$

Wichtig ist hier zu bemerken, dass in den Hammingabstand nicht eingeht "wie sehr" sich zwei einzelne Einträge

unterscheiden, sondern nur *ob* sie sich unterscheiden, d.h. es gilt:

$$\text{dist}\left(\begin{pmatrix}3\\0\\0\end{pmatrix}^T, \begin{pmatrix}2\\0\\0\end{pmatrix}^T\right) = \text{dist}\left(\begin{pmatrix}27\\0\\0\end{pmatrix}^T, \begin{pmatrix}0\\0\\0\end{pmatrix}^T\right) = \text{dist}\left(\begin{pmatrix}47\\0\\0\end{pmatrix}^T, \begin{pmatrix}11\\0\\0\end{pmatrix}^T\right) = \mathbf{1}\,.$$

Wie wir im Abschnitt über allgemeine Metriken schon gesehen haben, sind Hammingkugeln ein nicht sehr anschauliches Konstrukt auf den ersten Blick. Ist aber, wie im Kontext der Codes, der Hammingabstand der "richtige" Abstandsbegriff, kann man diesen für Anschauungen intuitiv verwenden (wir haben Metriken mit dem Ziel definiert, intuitiv von Abständen reden zu können). Wir werden im folgenden deshalb auch häufig von Hammingkugeln um Codewörter oder Wörter sprechen.

5.2.1. Dekodieren von Wörtern: Fehler erkennen & korrigieren

Zum Dekodieren von übertragenen Codewörtern verwendet man fast immer das Verfahren "Maximum-Likelyhood-Decoding" (MLD). Dieses Verfahren weist einem (empfangenen oder gelesenen) Wort $w \in A^n$ dasjenige Codewort c in $C \subseteq A^n$ zu, dass den kleinsten Hammingabstand zu w besitzt, sofern nur *ein* solcher nächster Nachbar existiert (und nicht mehrere):

Definition 5.2.6 (Dekodieren via MLD)**.** Das MLD-Verfahren liefert zu einem Wort w das nächst-liegende Wort c aus dem Code C. Gibt es mehrere solche Wörter, liefert es einen Fehler:

Es sei $w \in A^n$ ein Wort der Länge n und $C \subseteq A^n$ ein Blockcode der Länge n.

Es sei $d := \min\{\text{dist}(c, w) \ : \ c \in C\}$ der Minimalabstand von w zum Code C.

- Gibt es *nur ein* $\widetilde{c} \in C$ mit $\text{dist}(\widetilde{c}, w) = d$, so liefert MLD-Verfahren das Codewort $\widetilde{c}$.
  (in der Hamming-Kugel $\mathcal{B}_d(w)$ gibt es nur ein Wort $\tilde{c}$)
- Gibt es *mehrere* $c \in C$ mit $\text{dist}(\widetilde{c}, w) = d$, so liefert MLD-Verfahren einen Fehler.
  (in der Hamming-Kugel $\mathcal{B}_d(w)$ gibt es mehr als nur ein Wort)

Beispiel 5.2.7.

Es sei $C = \{(jan), (feb), (mrz), (apr), (mai), (jun), (jul)\} \subset \{a, \ldots, z\}^3$ ein Code.

- Soll $(jax) \notin C$ per MLD dekodiert werden, so erhält man (jan)
  wegen $\text{dist}(\,(jax), (jan)\,) = 1$ und $\text{dist}(\,(jax), c\,) > 1$ für alle $c \in C$ mit $c \neq (jan)$.
- Soll $(jon) \notin C$ per MLD dekodiert werden, so erhält man einen Fehler
  wegen $\text{dist}(\,(jon), (jan)\,) = \text{dist}(\,(jon), (jun)\,) = 1$.

Hier wird deutlich: Je weiter die Codewörter in einem Code auseinander sind, um so besser läuft das Dekodieren

von fehlerhaften Codewörtern. Entsprechend ist der (wie folgt definierte) Minimalabstand eines Codes ein Maß für die Güte des Codes:

Definition 5.2.8 (Minimalabstand)**.** Der Minimalabstand eines Codes C ist definiert als

$$\mathsf{d}(C) := \min\{\mathrm{dist}(x, y) \ : \ x, y \in C, x \neq y\}.$$

Bemerkung 5.2.9.

Mit Hammingkugeln formuliert, ergibt sich folgendes Bild. Für einen Code $C \subset A^*$ über dem Alphabet A betrachtet man für jedes Codewort $x \in C$ die Hammingkugeln $\mathcal{B}_{\mathsf{d}(C)}(x, A^*)$ mit Radius "Minimalabstand", so enthalten diese stets nur das Codewort x. Keine dieser Hammingkugeln mit Radius $\mathsf{d}(C)$ berührt ein weiteres Codewort in C neben dem Codewort im Mittelpunkt selbstverständlich. Es ist also

$$\bigcap_{x \in C} \mathcal{B}_{\mathsf{d}(C)}(x, A^*) \setminus \{x\} = \emptyset.$$

Dies ändert sich, wenn man Hammingkugeln mit Radius $\mathsf{d}(C)+1$ wählt. Dann gibt es mindestens ein Codewort, dessen Hammingkugel ein weiteres Codewort beinhaltet.

Ein Code C mit großem Minimalabstand hat eine große Toleranz gegenüber Störungen von Codewörtern. Bei einem Code C mit $\mathsf{d}(C) = 5$ erzeugt das Einfügen von bis zu 4 Fehlern in einem Codewort $c \in C$ stets ein Wort w, mit $w \notin C$. Ein solches fehlerhaftes Codewort (mit maximal 4 Fehlern) wird beim decodieren stets als fehlerhaft erkannt, man sagt ein solcher Code mit $\mathsf{d}(C) = 5$ ist "4-Fehlererkennend" (s. Def. 5.2.10).

Definition 5.2.10 (Fehlererkennung, Fehlerkorrektur)**.**

Ein Code C heißt

- **k-fehlererkennend** mit $k \in \mathbb{N} \cup \{0\}$, wenn für jedes Codewort $c \in C$ gilt:
  Einfügen von bis zu k Fehlern in c erzeugt stets ein Wort w, mit $w \notin C$.
  (Für alle $c \in C$ ist $\mathcal{B}_k(c) \setminus \{c\} = \emptyset$)
- **k-fehlererkorrigierend** mit $k \in \mathbb{N} \cup \{0\}$, wenn für jedes Codewort $c \in C$ gilt:
  Einfügen von bis zu k Fehlern in c erzeugt stets ein Wort w, das beim Decodieren per MLD c liefert.
  (Für alle $c \in C$ und alle $c' \in \mathcal{B}_k(c)$ ist $\mathcal{B}_k(c') \setminus \{c'\} = \{c\}$)

Bemerkung 5.2.11.

- Es sei nun C ein Code, der k-fehlererkennend ist. Wird dann in der Datenübertragung eines beliebigen Codewortes $c \in C$ eine oder bis zu k Postionen fehlerhaft übertragen, so ist für die Gegenseite stets ersichtlich, dass ein Übertragungsfehler aufgetreten ist: Das resultierende fehlerhafte Wort ist nicht Teil des Codes.
- Ist ein Code C dagegen *nicht* k-fehlererkennend, so gilt für mindestens ein Codewort $c \in C$: das Einfügen von k-vielen (evtl. speziellen!) Fehlern führt zu einem Wort w, dass ebenfalls ein Codewort ist. Dies be-

deutet nicht, dass das Einfügen von *beliebigen* k Fehlern in ein *beliebiges* Codewort immer "schiefgeht" wie das folgende Beispiel zeigt.

Beispiel 5.2.12.

Es sei

$$C := \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}^T, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}^T, \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}^T \right\}$$

Dieser Code ist 1-fehlererkennend, weil das Einfügen von nur einem Fehler in eines der CodeWörter nie ein anderes Codewort liefert. Der Code C ist nicht 2-fehlererkennend:

Fügt man an den Postionen c_2 und c_3 des Codewortes $c = (1, 2, 3)$ die speziellen Fehler "$2 \mapsto 1$" und "$3 \mapsto 1$" ein, so erhält man das Codewort $(1, 1, 1) \in C$. Dieser (spezielle) Doppelfehler bliebe also beim Übertragen unbemerkt.

Beispiel 5.2.13.

Es sei

$$C := \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}^T, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}^T, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}^T \right\}$$

Dieser Code ist 0-fehlererkennend, d.h. der Code bietet (im Worst case) nicht einmal die Möglichkeit, einen Übertragungsfehler zu entdecken, selbst wenn nur eine Stelle geändert wurde:

Bereits das einfügen eines einzigen (speziellen) Fehlers in $(0, 0, 0) \in C$ liefert $(1, 0, 0) \in C$. Ein solcher Fehler bliebe also beim Übertragen unbemerkt. Hier spielt es *keine* Rolle, dass das Einfügen eines einzelnen Fehlers in $(0, 1, 1) \in C$ stets bemerkt werden kann.

Wie in den Beispielen 5.2.12 und 5.2.13 sichtbar wird, ist für das Erkennen von Übertragungsfehlern der kleinste Abstand zweier Codewörter wichtig. Genauer gilt:

Lemma 5.2.14. Ein Code $C \subseteq A^n$ mit $d := \mathsf{d}(C)$ ist $(d-1)$-fehlererkennend und $\lfloor \frac{d-1}{2} \rfloor$-fehlerkorrigierend und nicht d-fehlererkennend und nicht $\lceil \frac{d}{2} \rceil$-fehlerkorrigierend .

Beweis. Es sei $C \subseteq A^n$ ein Blockcode mit Minimalabstand $d := \mathsf{d}(C)$.

- Es sei $c \in C$ ein beliebiges Codewort.
  Ändert man in c höchstens $d-1$ Stellen, so entsteht ein Wort $w \in A^n$ mit $\mathrm{dist}(w, c) \leq d-1$. Läge w im Code, so müsste $\mathsf{d}(C) \leq d-1$ gelten (Denn: $\mathsf{d}(C)$ ist das Minimum über alle $\mathrm{dist}(c, \widetilde{c})$ mit $c, \widetilde{c} \in C$, also gilt $\mathsf{d}(C) \leq \mathrm{dist}(c, w)$, falls $w \in C$). Dies liefert den Widerspruch $d \leq d-1$. Entsprechend gilt $w \notin C$, da c beliebig war gilt: C ist $d-1$ fehlererkennend.
- Da der Code Minimalabstand d hat gibt es zwei Codewörter c und c' mit $\mathrm{dist}(c, c') = d$. Ändert man c an genau den d Stellen, an denen sich c und c' unterscheiden und setzt diese auf den entsprechenden Wert von c', so gelangt man mit c' zu einem Wort, dass im Code liegt: C ist nicht d-fehlererkennend.

- Es sei $c \in C$ ein beliebiges Codewort.
  Ändert man in c höchstens $k := \lfloor \frac{d-1}{2} \rfloor$ Stellen, so entsteht ein Wort $w \in A^n$ mit $\text{dist}(c, w) \leq k$. Liefert das MLD-Verfahren einen Fehler oder ein anderes Codewort in C, so gibt es ein $\widetilde{c} \in C$ mit $\ell := \text{dist}(\widetilde{c}, w) \leq \text{dist}(c, w)$.
  Dreiecksungleichung: Ändert man in $\widetilde{c}$ also ℓ-viele Einträge, so erhält man w, ändert man dann in w weiterhin k Einträge, so erhält man c. Insgesamt muss man also höchstens $\ell + k$ Einträge in $\widetilde{c}$ ändern, um c zu erhalten, es gilt also

$$\begin{aligned} \text{dist}(\widetilde{c}, c) \quad \leq \quad k + \ell \quad &\leq \quad 2k \qquad\qquad |\text{wegen } \ell \leq k \\ &\leq \quad 2\lfloor \tfrac{d-1}{2} \rfloor \\ &\leq \quad d - 1 \qquad < d \end{aligned}$$

  Dies liefert einen Widerspruch: Der Abstand zweier Wörter in C ist mindestens der Minimalabstand $d = \mathsf{d}(C)$, aber die Wörter $\widetilde{c}$ und c müssten Abstand $\text{dist}(\widetilde{c}, c) < d$ haben, damit w fehlerhaft dekodiert wird.
- Da der Code Minimalabstand d hat gibt es zwei Codewörter c und c' mit $\text{dist}(c, c') = d$. Ändert man c an genau $\lfloor \frac{d-1}{2} \rfloor$ Stellen, an denen sich c und c' unterscheiden und setzt diese auf den entsprechenden Wert von c', so gelangt man zu einem Wort $\tilde{c}$ mit $\ell = \text{dist}(c, \tilde{c}) = \lfloor \frac{d}{2} \rfloor$ und $\ell' = \text{dist}(c', \tilde{c}) = d - \lfloor \frac{d}{2} \rfloor$.
  Wenn d gerade ist, dann ist $\ell = \ell'$ und das MLD-Verfahren liefert "FEHLER". Ist d ungerade, ist $\ell' > \ell$ und das MLD-Verfahren liefert c': C ist nicht d-fehlererkennend.

□

Satz 5.2.15 (Hamming-Schranke)**.** Seien $|A| = a$ und $n \in \mathbb{N}$. Sei C ein k-fehlerkorrigierender Blockcode der Länge n über A. Dann gilt

$$|C| \leq a^n \cdot \left(\sum_{i=0}^{k} \binom{n}{i} \cdot (a-1)^i \right)^{-1}$$

Beweis. Nach Voraussetzung ist für je zwei verschiedene Codewörter $c, c' \in C$ der Schnitt ihrere Hammingkugeln mit Radius k leer, also

$$\mathcal{B}_k(c) \cap \mathcal{B}_k(c') = \emptyset.$$

Für $c \in C$ und $0 \leq i \leq k$ gibt es genau $\binom{n}{i} \cdot (b-1)^k$ Wörter in A^n mit Hamming-Abstand i zu c, also ist

$$a^n = |A|^n \geq \left| \bigcup_{c \in C} \mathcal{B}_k(c) \right| = \sum_{c \in C} |\mathcal{B}_k(c)| = |C| \cdot \sum_{i=0}^{k} \binom{n}{i} \cdot (a-1)^i.$$

Die Behauptung folgt damit. □

5.3. Lineare Codes

Lineare Codes sind die Verallgemeinerung von Codes, die durch Hinzufügen von Prüfbits an die eigentlichen Datenwörter entstehen:

Üblicherweise konstruiert man Codes wie folgt.

- Zu den gegebenen Originaldaten $(x_1, \ldots, x_\ell)$ werden Prüfpositionen $p_{\ell+1}, \ldots, p_n$ hinzugefügt, die sich

über eine *lineare Funktion* aus den x_i berechnen:

$$p_i(x) = \lambda_{i1} \cdot x_1 + \ldots + \lambda_{i\ell} \cdot x_\ell.$$

- Das entstandene Codewort $c = (x_1, \ldots, x_\ell, p_{\ell+1}(x), \ldots, p_n(x))$ ist dann Teil eines linearen Codes (in einem linearen Code ist die Summe von zwei Codewörtern wieder ein Codewort (s. Def 5.3.5)).

In solchen linearen Codes lassen sich *leicht* alle Codewörter generieren und ebenso einfach lassen sich "normale Wörter" auf Mitglidschaft im Code testen: Für das Codieren und Dekodieren können einfache Matrix-Multiplikationen verwendet werden.

Wir betrachten im Folgenden ausschließlich lineare Codes $C \subseteq (F_p)^n$, also Codes über dem Alphabet $\mathbb{F}_p$. Die Menge $(\mathbb{F}_p)^n$ lässt sich als ein Vektorraum auffassen, dies werden wir ausnutzen, um das Codieren und Decodieren per Matrixmultiplikation zu ermöglichen.

5.3.1. Der Vektorraum $(\mathbb{F}_p)^n$

Wie wir im vorherigen Kapitel gesehen haben, ist für eine Primzahl $p \in \mathbb{N}$ das Tupel $\mathbb{F}_p := (\{1, \ldots, p-1\}, \oplus_p, \odot_p)$ ein endlicher Körper.

Genauso wie es zu dem Körper $\mathbb{R}$ Vektorräume der Form $\mathbb{R}^n$ gibt, kann man für jeden Körper $\mathbb{F}_p$ und jedes n einen Vektorraum $(\mathbb{F}_p)^n$ definieren. Ganz analog zum $\mathbb{R}^n$ gelten im Vektorraum $(\mathbb{F}_p)^n$ Rechengesetze, so kann man

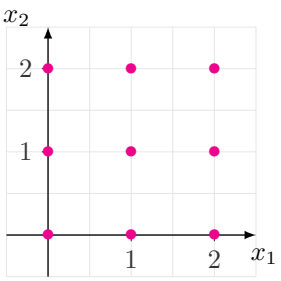
- zwei Vektoren $v, w \in (\mathbb{F}_p)^n$ addieren $v \oplus_p w$ (oder subtrahieren)
  (Elementweise Addition mittels $\oplus_p$).
- einen Vektor $v \in (\mathbb{F}_p)^n$ mit einer Zahl $\lambda \in \mathbb{F}_p$ multiplizieren $\lambda \odot_p v$
  (Jedes Element von v wird mittels $\oplus_p$ mit λ multipliziert).

Beispiel 5.3.1.

Der Vektorraum $(\mathbb{F}_3)^2$ hat die folgenden 9 Elemente:

$$(\mathbb{F}_3)^2 = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \end{pmatrix} \right\}$$

Zeichnet man die Punkte von $(\mathbb{F}_3)^2$ in einen zwei-dimensionalen $\mathbb{R}^2$ ein, so erhält man die Punkte eines Gitters.



Addiert man zwei Vektoren so geschieht dies modulo $p = 3$, gleich der Multiplikation mit einem Skalar aus

$\mathbb{F}_3$. Es gilt z.B:

$$\begin{pmatrix}1\\1\end{pmatrix} \oplus_3 \begin{pmatrix}1\\2\end{pmatrix} = \begin{pmatrix}1 \oplus_3 1\\1 \oplus_3 2\end{pmatrix} = \begin{pmatrix}2\\0\end{pmatrix} \qquad 2 \odot_3 \begin{pmatrix}1\\2\end{pmatrix} = \begin{pmatrix}2 \odot_3 1\\2 \odot_3 2\end{pmatrix} = \begin{pmatrix}2\\1\end{pmatrix}$$

Bemerkung 5.3.2.

Das Bilden von negativen Zahlen bzw. negativen Vektoren in $(\mathbb{F}_p)^n$ erfolgt modulo p:

Will man für den Vektor $v := \begin{pmatrix}1\\2\end{pmatrix} \in (\mathbb{F}_5)^2$ den negativen Vektor $w := -v$ berechnen, so enthält w in jedem Eintrag w_i das *additiv Inverse* zu v_i modulo p:

Es gilt: $-v = \begin{pmatrix}4\\3\end{pmatrix}$ denn $\begin{pmatrix}4\\3\end{pmatrix} \oplus_5 \begin{pmatrix}1\\2\end{pmatrix} = \begin{pmatrix}4 \oplus_5 1\\3 \oplus_5 2\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix}$

Unterräume

Aus der linearen Algebra kennen wir schon das Konzept des Untervektorraums (Unterraums) eines Vektorraums.

Definition 5.3.3 (Untervektorraum)**.** Es sei V ein Vektorraum über einem Körper $\mathbb{K}$.

Ein Untervektorraum $W \subseteq V$ des Vektorraums V ist eine Teilmenge von V, die ...

... den Nullvektor $\mathbf{0}$ aus V enthält:	i) $\mathbf{0} \in W$.
... unter Addion abgeschlossen ist:	ii) $w + \widetilde{w} \in W$ für alle $w, \widetilde{w} \in W$.
... unter skalarer Multiplikation abgeschlossen ist:	iii) $\lambda \cdot w \in W$ für alle $\lambda \in \mathbb{K}$, $w \in W$.

Beispiel 5.3.4.

Die Menge

$$W := \left\{ \begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}2\\2\end{pmatrix} \right\} \subsetneq (\mathbb{F}_3)^2$$

ist ein Untervektorraum von $(\mathbb{F}_3)^2$.

Beschreibung von Untervektorräumen

Die Unterräume des reelen Vektorraums der Dimension 3, der $\mathbb{R}^3$, sind schlicht Ebenen und Geraden, die den Nullvektor enthalten.

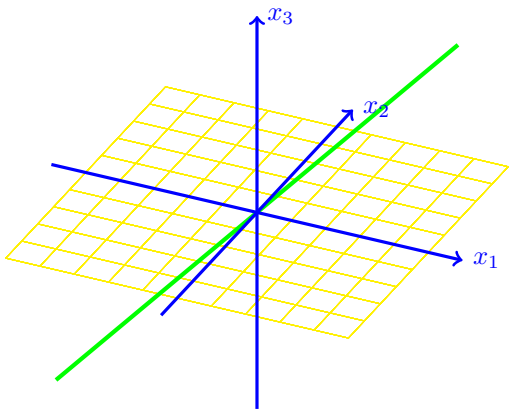
Zum Beispiel sind die x_1-x_2-Ebene $E_{x_1x_2}$, in der alle Vektoren als letzte Koordinate eine 0 haben

$$E_{xy} = \left\{x \in \mathbb{R}^3 \ : \ x_3 = 0\right\} = \left\{ \lambda \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \mu \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \ : \ \lambda, \mu \in \mathbb{R} \right\},$$

und die Gerade durch den Punkt $(1,1,1)^T$ und den Ursprung

$$G_{111} = \left\{x \in \mathbb{R}^3 \ : \ x_1 + x_2 - 2 \cdot x_3 = 0 \text{ und } x_1 - 2 \cdot x_2 + x_3 = 0\right\} = \left\{ \lambda \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \ : \ \lambda \in \mathbb{R} \right\},$$

Unterräume des $\mathbb{R}^3$.



Die Darstellung/Beschreibung eines solchen Unterraumes kann auf zwei unterschieldiche Art und Weisen erfolgen

- Man gibt ein Set von *Gleichungen* an, die alle Punkte in W erfüllen müssen.
- Man gibt ein *Herstellungsrezept* an, wie man die Vektoren in W aus einer Basis mittels Linearkombination herstellt.

In den beiden obigen Darstellungen von E_{xy} ist

- Die Menge der einschränkenden Gleichung(en):

$$0 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 = 0$$

- Die Vektoren $\left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right\}$ bilden eine Basis, aus der sich alle Elemente in E_{xy} linearkombinieren lassen.

In den beiden obigen Darstellungen von G_{111} ist

- Die Menge der einschränkenden Gleichung(en):

$$x_1 + x_2 - 2 \cdot x_3 = 0$$

$$x_1 - 2 \cdot x_2 + x_3 = 0$$

- Der Vektor $\left\{ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\}$ bildet eine Basis, aus der sich alle Elemente in G_{111} linearkombinieren lassen.

Mittels Basen lassen sich leicht Elemente von Unterräumen erstellen, mittels Gleichungssysteme lassen sich beliebige Vektoren aus $\mathbb{R}^3$ darauf überprüfen, ob sie in Unterräumen liegen. Beides werden wir später bei linearen Codes genau so wiederfinden, in Form der Generatormatrix und der Kontrollmatrix.

5.3.2. Lineare Codes sind lineare Räume

Wir haben schon angedeutet, dass Codes üblicherweise mit Hilfe von linearen Funktionen gebildet werden. Tatsächlich führt dieses Verfahren dazu, dass diese Codes immer Unterräume von Vektorräumen über endlichen Körpern sind. Wir definieren nun also eine spezielle Klasse von Codes, mit der man wegen der zugrundeliegenden Vektorraumstruktur leicht rechnen kann.

Definition 5.3.5 (linearer Code)**.** Ein Code $C \subseteq (\mathbb{F}_p)^n$ heißt ***linearer Code***, wenn C ein Untervektorraum von $(\mathbb{F}_p)^n$ ist.

Hat C als Vektorraum die Dimension k, dann spricht man von einem $[n, k]$***-Code***.

Hat C zusätzlich Minimalabstand $d := \mathsf{d}(C)$, dann spricht man von einem $[n, k, d]$***-Code***.

Beispiel 5.3.6.

Der folgende Code $C \subset (\mathbb{F}_5)^3$ ist ein *linearer* $[3, 1, 2]$-Blockcode der Länge 3 mit Einträgen aus $\mathbb{F}_5 = \{0, 1, 2, 3, 4\}$:

$$C := \{(0,0,0),\ (1,1,0),\ (2,2,0),\ (3,3,0),\ (4,4,0)\ \}$$

- C ist ein linearer Raum, weil für $c, \widetilde{c} \in C$ stets $c \oplus_3 \widetilde{c} \in C$ und $\lambda \odot_3 c \in C$ gilt.
  Zum Beispiel gelten: $\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}^T \oplus_5 \begin{pmatrix} 4 \\ 4 \\ 0 \end{pmatrix}^T = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}^T \in C$ und $4 \odot_5 \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix}^T = \begin{pmatrix} 3 \\ 3 \\ 0 \end{pmatrix}^T \in C$.
- Die Dimension des Codes ist 1, da $\{(1,1,0)\}$ eine Basis des linearen Raumes C ist: jedes Codewort in C lässt sich als Vielfaches $\lambda \odot_5 (1,1,0)$ von $(1,1,0)$ schreiben mit $\lambda \in \mathbb{F}_5 = \{0,1,2,3,4\}$.
- Es gilt $\mathsf{d}(C) = 2$ da sich je zwei Wörter in zwei Positionen unterscheiden.

Bemerkung 5.3.7.

Während man n direkt aus der Länge der Codewörter ablesen kann, muss man die Dimension k und den Minimalabstand d (umständlich) berechnen!

Es sei $C \subseteq (\mathbb{F}_p)^n$. Als Untervektorraum von $(\mathbb{F}_p)^n$ erfüllt C die folgenden Eigenschaften

i) Für $c, \widetilde{c} \in C$ gilt: $c \oplus_p \widetilde{c} \in C$.
ii) Für $c \in C$ und $\lambda \in \mathbb{F}_p$ gilt: $\lambda \odot_p c \in C$.
iii) Der Nullvektor $\mathbf{0}$ aus $\mathbb{F}_p$ ist Teil des Codes: $\mathbf{0} := (\underbrace{0, \ldots, 0}_{n- \text{ viele}}) \in C$.

Der Minimalabstand $\mathsf{d}(C)$ lässt sich für ***lineare*** Codes leicht berechenen:

Lemma 5.3.8 (Minimalabstand linearer Codes)**.** Der Minimalabstand eines linearen Codes C erfüllt $\mathsf{d}(C) = \min\{\text{dist}(c, \mathbf{0}) : c \in C, c \neq \mathbf{0}\}$.

Beweis.

"$\leq$" ist klar.
"$\geq$" Seien $x, y \in C \subseteq (\mathbb{F}_p)^n$ mit $x \neq y$ und $d := \text{dist}(x, y)$.
Für $z := x - y$ gilt nun $z \neq \mathbf{0}$ (wegen $x \neq y$) und $\text{dist}(x, y) = \text{dist}(z, \mathbf{0})$ wegen:

$$\begin{aligned} \text{dist}(x, y) &= |\{\, i \in \{1, \ldots, n\} \;:\; x_i \quad\; \neq y_i \,\}| \\ &= |\{\, i \in \{1, \ldots, n\} \;:\; x_i - y_i \neq 0 \,\}| \;=\; \text{dist}(z, \mathbf{0}) \end{aligned}$$

Daraus folgt

$$\mathsf{d}(C) = \text{dist}(x, y) = \text{dist}(z, \mathbf{0}) \geq \min\{\text{dist}(c, \mathbf{0}) : c \in C, c \neq \mathbf{0}\}$$

□

Lemma 5.3.8 ist äußerst hilfreich beim Berechnen von $\mathsf{d}(C)$. Nach der eigentlichen Definition von $\mathsf{d}(C)$ müsste man in einem Code den Abstand für *alle Paare* von Codewörtern berechnen, um an $\mathsf{d}(C)$ zu gelangen. Für lineare Codes genügt es bereits, alle Codewörter $c \in C$ *einmal* anzuschauen.

Beispiel 5.3.9.

Für den Code aus Beispiel 5.3.6 gilt: $\mathsf{d}(C) = 2$, wegen

$$\text{dist}\left(\begin{pmatrix}1\\1\\0\end{pmatrix}^T, \begin{pmatrix}0\\0\\0\end{pmatrix}^T\right) = \text{dist}\left(\begin{pmatrix}2\\2\\0\end{pmatrix}^T, \begin{pmatrix}0\\0\\0\end{pmatrix}^T\right) = \ldots = \text{dist}\left(\begin{pmatrix}4\\4\\0\end{pmatrix}^T, \begin{pmatrix}0\\0\\0\end{pmatrix}^T\right) = 2.$$

Dieser Code ist also 1-fehler-erkennend und $\lfloor \frac{2-1}{2} \rfloor = 0$-fehler-korrigierend.

5.3.3. Die Generatormatrix generiert Codewörter

Ist C ein linearer Code, so ist C ein Vektorraum. Damit hat C eine Basis, aus der sich alle Elemente in C erstellen lassen. Kennt man eine solche Basis, kann man die Basisvektoren in einer sogennanten "Generatormatrix" zusammenstellen:

Mittels der Generatormatrix lassen sich zum einen (wie aus einer Basis) alle Codewörter generieren, zum anderen kann man mit einer Generatormatrix einem gegebenen Original-Datenwort ein eindeutiges Codewort zuordnen.

Definition 5.3.10 (Generatormatrix). Sei C ein linearer Blockcode der Dimension k, und $g_1, \ldots, g_k \in C$ seien linear unabhängig. Dann ist die Matrix G mit Zeilen g_i eine Generatormatrix von C:

$$G := \begin{pmatrix} g_1 \\ \vdots \\ g_k \end{pmatrix}$$

Beispiel 5.3.11.

Es sei

$$C = \left\{ \begin{pmatrix}0\\0\\0\end{pmatrix}^T, \begin{pmatrix}1\\0\\1\end{pmatrix}^T, \begin{pmatrix}0\\1\\1\end{pmatrix}^T, \begin{pmatrix}1\\1\\0\end{pmatrix}^T \right\} \subsetneq (\mathbb{F}_2)^3$$

Die Codewörter Nr.2, Nr.3 und Nr.2 sind jeweils paarweise linear unabhängig.

Alle drei zusammen sind jedoch linear abhängig, denn es gilt:

$$\begin{pmatrix}1\\0\\1\end{pmatrix}^T \oplus_2 \begin{pmatrix}0\\1\\1\end{pmatrix}^T \oplus_2 \begin{pmatrix}1\\1\\0\end{pmatrix}^T = \begin{pmatrix}0\\0\\0\end{pmatrix}^T \quad \text{bzw.} \quad \begin{pmatrix}1\\0\\1\end{pmatrix}^T \oplus_2 \begin{pmatrix}0\\1\\1\end{pmatrix}^T = \begin{pmatrix}1\\1\\0\end{pmatrix}^T$$

Also hat dieser Code sechs verschiedene Generatormatrizen:

kanonisch $G_{2} := \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ $\quad G_{24} := \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$ $\quad G_{34} := \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$

$G_{32} := \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$ $\quad G_{42} := \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$ $\quad G_{43} := \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$

Bemerkung 5.3.12.

- Da die k Zeilen $g_1, \ldots, g_k$ von G linear unabhängig sind, und der zugehörige Code genau Dimension k hat, ist $\{g_1, \ldots, g_k\}$ eine **Basis** des Vektorraums C.
- Ein Code hat im Allgemeinen mehr als eine Generatormatrix.
- Für einen linearen $[n, k]$-Blockcode $C \subseteq (\mathbb{F}_p)^n$ ist eine Generatormatrix stets eine $k \times n$ Matrix mit Einträgen aus $\mathbb{F}_p$, d.h. $G \in (\mathbb{F}_p)^{k \times n}$.

Mit einer Generatormatrix eines linearen Codes kann man Orignialdaten-Wörter in Codewörter umrechnen und somit auch alle Codewörter eines Codes generieren

Alle Codewörter generieren

Für einen Code $C \subset (F_3)^3$ sei $G = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ eine Generatormatrix.

Nach Definition einer Generatormatrix gilt C hat Dimension 2, da G zwei Zeilen hat.

Die beiden Zeilen von G bilden also eine Basis von C, d.h. jedes Codewort $c \in C$ lässt sich aus den Codewörtern $(1,1,0)$ und $(1,1,1)$ linearkombinieren, d.h.

$$c = \lambda \cdot (1,1,0) \oplus_3 \mu \cdot (1,1,1) \qquad \text{mit } \lambda, \mu \in \mathbb{F}_3$$

Dies können wir (mittels Matrixmultiplikation) kurz schreiben als:

$$c = (\lambda, \mu) \cdot \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \qquad \text{mit } \lambda, \mu \in \mathbb{F}_3$$

Es gilt also: $C = \{x \cdot G \ : \ x \in (\mathbb{F}_3)^2\}$. Erstellt man nun eine Tabelle für alle Möglichkeiten $\mu, \lambda \in \{0,1,2\}$ so erhält man alle Codewörter aus C:

	$\lambda = 0$	$\lambda = 1$	$\lambda = 2$
$\mu = 0$	$(0,0,0)$	$(1,1,0)$	$(2,2,0)$
$\mu = 1$	$(1,1,1)$	$(2,2,1)$	$(0,0,1)$
$\mu = 2$	$(2,2,2)$	$(0,0,2)$	$(1,1,2)$

Es gilt z.B. für $\lambda = 1$ und $\mu = 2$:

$$(1,2) \cdot \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} = \begin{array}{rr} & 1 \cdot (1, \ 1, \ 0) \\ \oplus_3 & 2 \cdot (1, \ 1, \ 1) \\ \hline & (0, \ 0, \ 2) \end{array}$$

Den Datenwörtern Codewörter zuordnen

Für einen Code $C \subset (F_3)^3$ sei $G = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ eine Generatormatrix.

Jedem Datenwort $x \in (\mathbb{F}_3)^2$ kann ein Codewort $c_x = x \cdot G$ zugeordnet werden

$$c_x = (x_1, x_2) \odot \begin{pmatrix} 1\ 1\ 0 \\ 1\ 1\ 1 \end{pmatrix} = \begin{array}{r} x_1 \odot (\ 1\ 1\ 0\ ) \\ \oplus\, x_2 \odot (\ 1\ 1\ 1\ ) \\ \hline (x_1 \oplus x_2, \ x_1 \oplus x_2, \ x_1) \end{array}$$

	$x_1 = 0$	$x_1 = 1$	$x_1 = 2$
$x_2 = 0$	$(0,0,0)$	$(1,1,0)$	$(2,2,0)$
$x_2 = 1$	$(1,1,1)$	$(2,2,1)$	$(0,0,1)$
$x_2 = 2$	$(2,2,2)$	$(0,0,2)$	$(1,1,2)$

Man sieht: Verschiedene Datenwörter $x, \widetilde{x} \in (\mathbb{F}_3)^2$ mit $x \neq \widetilde{x}$ erzeugen auch verschiedene Codewörter: $c_x = x \cdot G \neq \widetilde{x} \cdot G = c_{\widetilde{x}}$.

Satz 5.3.13. Ist $G \in (\mathbb{F}_p)^{k \times n}$ eine Generatormatrix von $C \subseteq (\mathbb{F}_p)^n$, und $x, \widetilde{x} \in (\mathbb{F}_p)^k$ sind Datenwörter, so gilt für die zugeordneten Codewörter: $x \cdot G \neq \widetilde{x} \cdot G$.

Berechnet man die zugeordneten Codewörter $x \cdot G$ für alle $x \in (\mathbb{F}_p)^k$, so erhält man C:

$$C = \{x \cdot G \ : \ x \in (\mathbb{F}_p)^k\}$$

Beweis. Da die Zeilen von G linear *unabhängig sind**, müssen für $x, \widetilde{x} \in (\mathbb{F}_3)^k$ mit $x \neq \widetilde{x}$ auch verschiedene Codewörter entstehen: $x \cdot G \neq \widetilde{x} \cdot G$.

Wäre dies nicht der Fall, würde aus $x \cdot G = \widetilde{x} \cdot G$ und $x \neq \widetilde{x}$ folgen:

$$0 = x \cdot G - \widetilde{x} \cdot G = (x - \widetilde{x}) \cdot G \qquad \text{mit } (x - \widetilde{x}) \neq 0.$$

Dies ist aber gleichbedeutend damit, dass die Zeilen $g_1, \ldots, g_k$ von G *linear abhängig* sind (Widerspruch zu $\star$). Ist nämlich $y := x - \widetilde{x}$ so gilt $y \cdot G = y_1 \cdot g_1 \oplus_p \ldots \oplus_p y_k \cdot g_k = 0$ mit $y \neq 0$. □

Korollar 5.3.14. Ein linearer Code $C \subseteq (\mathbb{F}_p)^n$ der Dimension k enthält p^k Codewörter (unabhängig von n).

5.3.4. Systematische Positionen: Wo sind Daten, wo sind Kontrollstellen?

Üblicherweise konstruiert man Codes wie folgt: Zu den gegebenen Originaldaten $(x_1, \ldots, x_k)$ werden Prüfpositionen $p_{k+1}, \ldots, p_n$ hinzugefügt, die sich aus den x_i berechnen. Das entstandene Codewort $c = (x_1, \ldots, x_k, p_{k+1}, \ldots, p_n)$ ist dann Teil eines linearen Codes.

Startet man umgekehrt mit einem linearen Code, so stellt sich die Frage: “Welche Datenworte werde ich welchen Codewörtern zuordnen?” Viel wichtiger ist allerdings die damit verbundene Frage: “Und wenn ich damit fertig bin, wie lese ich aus einem Codewort die Originaldaten wieder aus??”

Hat man eine Zuordnung die einem Datenwort (x_1, x_2) ein Codewort der Form $c(x) = (x_1, x_2, p_3)$ zuordnet, so kann man schnell sehen, dass $c(x)_1$ und $c(x)_2$ die Originaldaten sind und p_3 vermutlich eine Kontrollgröße ist. Hat man allerdings einfach nur einen Code, ohne eine solche Zuordnung, stellt sich sofort die Frage: “Wo werde ich in den Codewörtern meine Originaldaten unterbringen?”.

Um diese Frage zu beantworten, müssen wir einen etwas sperrigen Begriff einführen, dem der systematischen Positionen in einem Code. Grob gesagt ist ein bestimmtes Set von Codewort-Positionen “systematisch” wenn sich in dem Code auf diesen Positionen die Originaldaten vollständig unterbringen lassen:

Definition 5.3.15 (Systematische Stellen)**.** Ein Code $C \subseteq (\mathbb{F}_p)^n$ heißt *systematisch* in den Stellen $J := \{j_1, \ldots, j_k\}$, wenn zu jedem Vektor $x = (x_1, \ldots, x_k) \in (\mathbb{F}_p)^k$ genau ein Codewort $c \in C$ exisitiert mit $x = (c_{j_1}, \ldots, c_{j_k})$.

Beispiel 5.3.16.

Der Code

$$C := \left\{ \begin{pmatrix} 0\\0\\0\\0\\0 \end{pmatrix}^T, \begin{pmatrix} 0\\0\\0\\1\\1 \end{pmatrix}^T, \begin{pmatrix} 0\\1\\1\\1\\0 \end{pmatrix}^T, \begin{pmatrix} 0\\1\\1\\0\\1 \end{pmatrix}^T, \begin{pmatrix} 1\\1\\0\\0\\0 \end{pmatrix}^T, \begin{pmatrix} 1\\1\\0\\1\\1 \end{pmatrix}^T, \begin{pmatrix} 1\\0\\1\\1\\0 \end{pmatrix}^T, \begin{pmatrix} 1\\0\\1\\0\\1 \end{pmatrix}^T \right\} \subseteq (\mathbb{F}_2)^5$$

ist systematisch auf den Positionen $1, 3, 5$, d.h. $j_1 = 1$, $j_2 = 3$ und $j_3 = 5$. Liest man aus jedem Codewort c nur $(c_{j_1}, c_{j_2}, c_{j_3}) = (c_1, c_3, c_5)$ aus, so erhält man jeden Vektor aus $(\mathbb{F}_2)^3$ genau einmal:

$$\left\{ \begin{pmatrix}0\\0\\0\end{pmatrix}^T, \begin{pmatrix}0\\0\\1\end{pmatrix}^T, \begin{pmatrix}0\\1\\0\end{pmatrix}^T, \begin{pmatrix}0\\1\\1\end{pmatrix}^T, \begin{pmatrix}1\\0\\0\end{pmatrix}^T, \begin{pmatrix}1\\0\\1\end{pmatrix}^T, \begin{pmatrix}1\\1\\0\end{pmatrix}^T, \begin{pmatrix}1\\1\\1\end{pmatrix}^T \right\} = (\mathbb{F}_2)^3$$

Der Code C ist *auch* systematisch z.B. auf den Positionen $2, 3, 4$.

Der Code C ist *nicht* systematisch auf…

- … z.B. den Positionen $1, 2, 3$, denn die ersten beiden Codewörter liefern beide den Nullvektor (Doppelung).
- … nur zwei Positionen i, j (egal was i, j sind), weil stets Doppelungen auftreten.
- … $m > 3$ Positionen, weil dann der komplette Raum $(\mathbb{F}_2)^m$ in C vorkommen müsste, und $(\mathbb{F}_2)^m$ hat $2^m > 2^3$ Elemente, also mehr als die 8 Elemente von C.

Satz 5.3.17. Ein linearer $[n, k]$-Code $C \subseteq (\mathbb{F}_p)^n$, der systematisch ist auf den Positionen $1, 2, \ldots, k$ besitzt eine Generatormatrix der Form $G = (\mathrm{id}_k \; A)$ wobei id_k eine $k \times k$-Einheitsmatrix ist.

Eine solche Generatormatrix nennen wir eine *kanonische Generatormatrix*.

Bemerkung 5.3.18.

Nicht jeder Code C besitzt eine kanonische Generatormatrix. Allerdings kann man nach Umordnen der Wörter häufig aus C einen neuen Code $\widetilde{C}$ erzeugen, der eine kanonische Generatormatrix besitzt:

Der Code in Beispiel 5.3.16 besitzt keine kanonische Generatormatrix, weil der Code *nicht systematisch* auf den Positionen $1, 2, 3 = \dim(C)$ ist. Tauscht man aber in jedem Codewort c die Buchstaben an den Positionen 2 und 5 so erhält man den Code aus Beispiel 5.3.19, der eine kanonische Generatormatrix besitzt.

Beispiel 5.3.19.

Der Code

$$C := \left\{ \begin{pmatrix}0\\0\\0\\0\\0\end{pmatrix}^T, \overset{g_2}{\begin{pmatrix}0\\1\\0\\1\\0\end{pmatrix}}^T, \overset{g_3}{\begin{pmatrix}0\\0\\1\\1\\1\end{pmatrix}}^T, \begin{pmatrix}0\\1\\1\\0\\1\end{pmatrix}^T, \overset{g_1}{\begin{pmatrix}1\\0\\0\\0\\1\end{pmatrix}}^T, \begin{pmatrix}1\\1\\0\\1\\1\end{pmatrix}^T, \begin{pmatrix}1\\0\\1\\1\\0\end{pmatrix}^T, \begin{pmatrix}1\\1\\1\\0\\0\end{pmatrix}^T \right\} \subseteq (\mathbb{F}_2)^5$$

ist systematisch auf den Positionen $1, 2, 3$ mit $3 = \dim(C)$ und besitzt damit eine kanonische Generatormatrix:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \begin{matrix} \leftarrow g_1 \\ \leftarrow g_2 \\ \leftarrow g_3 \end{matrix}$$

5.3.5. Eine Kontrollmatrix prüft Wörter auf Mitgliedschaft im Code.

Definition 5.3.20. Es sei $C \subseteq (\mathbb{F}_p)^n$ ein linearer Code. Eine Matrix $H \in (\mathbb{F}_p)^{n\times \ell}$ ist eine Kontrollmatrix von C, falls für alle $x \in (\mathbb{F}_p)^n$ gilt:

$$x \in C \Leftrightarrow x \cdot H = (0, 0, \ldots, 0)$$

Beispiel 5.3.21.

Für den Code $C = \left\{ \begin{pmatrix}0\\0\\0\end{pmatrix}^T, \begin{pmatrix}1\\0\\1\end{pmatrix}^T, \begin{pmatrix}0\\1\\1\end{pmatrix}^T, \begin{pmatrix}1\\1\\0\end{pmatrix}^T \right\} \subset (\mathbb{F}_2)^3$ ist $H = \begin{pmatrix}1\\1\\1\end{pmatrix}$ eine Kontrollmatrix.

Für ein beliebiges Wort $x \in (\mathbb{F}_2)^3$:

$$(x_1, x_2, x_3) \cdot \begin{pmatrix}1\\1\\1\end{pmatrix} = x_1 \oplus_2 x_2 \oplus_2 x_3 = \begin{cases} 0 & \text{falls } x \text{ enthält} \quad \text{gerade viele 1-Einträge} \\ 1 & \text{falls } x \text{ enthält ungerade viele 1-Einträge} \end{cases}$$

Satz 5.3.22. Es sei $C \subseteq (\mathbb{F}_p)^n$ ein linearer Blockcode mit dimension $k := \dim(C)$ und kanonischer Generatormatrix $G = (\mathrm{id}_k\ A)$.

Dann ist

$$H := \begin{pmatrix} -A \\ \mathrm{id}_{n-k} \end{pmatrix}$$

eine Kontrollmatrix von C.

Bemerkung 5.3.23.

Achtung: Das Bilden von negativen Zahlen in $(\mathbb{F}_p)^n$ erfolgt modulo p:

Will man für eine Matrix $A \in (\mathbb{F}_p)^{n\times m}$ die negative Matrix $A := -A$ berechnen, so enthält A in jedem Eintrag A_{ij} das "*additiv Inverse* zu A_{ij} modulo p":

Für $A := \begin{pmatrix} 0 & 2 \\ 1 & 3 \end{pmatrix} \in (\mathbb{F}_5)^{2\times 2}$ gilt $-A = \begin{pmatrix} 0 & 3 \\ 4 & 2 \end{pmatrix}$.

Beispiel 5.3.24.

Der Code $C \subset (\mathbb{F}_2)^5$ aus Beispiel 5.3.19 hat die folgende kanonische Generatormatrix:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \qquad \text{d.h. } G = (I_3\ A) \text{ mit } A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Es gilt also:
$$H = \begin{pmatrix} -0 & -1 \\ -1 & -0 \\ -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$ ist eine Kontrollmatrix für C

Beispiel 5.3.25.

Der Code $C \subset (\mathbb{F}_3)^3$ mit $C = \left\{ \begin{pmatrix}0\\0\\0\end{pmatrix}^T, \begin{pmatrix}0\\1\\1\end{pmatrix}^T, \begin{pmatrix}0\\2\\2\end{pmatrix}^T, \begin{pmatrix}1\\0\\1\end{pmatrix}^T, \begin{pmatrix}1\\1\\2\end{pmatrix}^T, \begin{pmatrix}1\\2\\0\end{pmatrix}^T, \begin{pmatrix}2\\0\\2\end{pmatrix}^T, \begin{pmatrix}2\\1\\0\end{pmatrix}^T, \begin{pmatrix}2\\2\\1\end{pmatrix}^T \right\}$

ist systematisch an den Positionen 1 und 2.

Die kanonische Generatormatrix lautet

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad \text{d.h. } G = (I_2\ A) \text{ mit } A = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Es gilt also:
$$H = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$$ ist eine Kontrollmatrix für C

Um Satz 5.3.22 zu beweisen, benötigen wir ein Kriterium, dass entscheidet wann eine Matrix H, die $c \cdot H = (0, \dots, 0)$ für alle C erfüllt, auch eine Kontrollmatrix ist:

Lemma 5.3.26 (Kriterium für Kontrollmatrix)**.** Es sei $C \subseteq (\mathbb{F}_p)^n$ ein linearer Code mit Generatormatrix G.

Eine Matrix $H \in (\mathbb{F}_p)^{n \times \ell}$ ist eine Kontrollmatrix für C genau dann wenn gelten

i) $G \odot_p H = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \in (\mathbb{F}_p)^{k \times \ell}$

ii) $\operatorname{rang}(H) = n - k$ mit $k = \dim(C)$.

Beweis. Der Beweis ist eine Übungsaufgabe. □

Beweis. [Satz 5.3.22] Es seien G und H wie in Satz 5.3.22 beschrieben.

Um zu zeigen, dass H eine Kontrollmatrix für C ist, verwenden wir die Kriterien aus Lemma 5.3.26: Es gilt $\operatorname{rang}(H) = n - k$ denn die $n - k$ Spalten von H sind offensichtlich linear unabhängig, wegen des "Nachspanns"aus id_{n-k}:

Die j-te Spalte H_j der Matrix H hat die Form $\begin{pmatrix} -A_j \\ e_j \end{pmatrix}$ wobei A_j die j-te Spalte aus A ist. Die Nachspannvektoren $e_1, \dots e_{n-k}$ sind linear unabhängig, also sind auch die Spalten $H_1, \dots, H_{n-k}$ linear unabhängig.

Weiter gilt:

$$G \odot_p H = (I_k, A) \odot_p \begin{pmatrix} -A \\ I_{n-k} \end{pmatrix} = \mathrm{id}_k \odot_p (-A) \quad \oplus_p \quad A \odot_p \mathrm{id}_{n-k} = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$$

□

5.3.6. Paritäts-Bits und Kontrollstellen

Üblicherweise konstruiert man Codes wie folgt: Zu den gegebenen Originaldaten $(x_1, \ldots, x_k)$ werden Prüfpositionen $p_{k+1}, \ldots, p_n$ hinzugefügt, die sich aus den x_i berechnen. Das entstandene Codewort $c = (x_1, \ldots, x_k, p_{k+1}, \ldots, p_n)$ ist dann Teil eines linearen Codes, wenn die p_i sich als Linearkombination der Stellen x_i berechnen.

Lemma 5.3.27 (Codes mit angefügten Kontrollstellen)**.** Es seien $p_i := (\mathbb{F}_p)^k \to \mathbb{F}_p$ lineare Funktionen, d.h. es gelten:

$$\left.\begin{array}{rcl} p_1(x) & = & \lambda_{11} \cdot x_1 \oplus_p \ldots \oplus_p \lambda_{1k} \cdot x_k \\ p_2(x) & = & \lambda_{21} \cdot x_1 \oplus_p \ldots \oplus_p \lambda_{2k} \cdot x_k \\ \vdots & & \vdots \\ p_m(x) & = & \lambda_{m1} \cdot x_1 \oplus_p \ldots \oplus_p \lambda_{mk} \cdot x_k \end{array}\right\} \text{ mit } \lambda_{ij} \in \mathbb{F}_p$$

Dann ist

$$C := \{ \ (x\,, \ p_1(x), p_2(x), \ldots, p_m(x)\ ) \ : \ x \in (F_p)^k \}$$

ein linearer Code der Dimension k mit genersicher Generatormatrix $G := (I_k \ A)$ mit

$$A = \begin{pmatrix} p_1(e_1) & p_2(e_1) & \ldots & p_{n-k}(e_1) \\ p_1(e_2) & p_2(e_2) & \ldots & p_{n-k}(e_2) \\ \vdots & \vdots & & \vdots \\ p_1(e_k) & p_2(e_k) & \ldots & p_{n-k}(e_k) \end{pmatrix} = \begin{pmatrix} \lambda_{11} & \lambda_{21} & \ldots & \lambda_{m1} \\ \lambda_{12} & \lambda_{22} & \ldots & \lambda_{m2} \\ \vdots & \vdots & & \vdots \\ \lambda_{1k} & \lambda_{2k} & \ldots & \lambda_{mk} \end{pmatrix}$$

Beispiel 5.3.28.

Es seien $D := (\mathbb{F}_3)^2$ die Menge der Orignialdatenwörter.

Die Prüfstellen p_1, p_2 in jedem Codewort $(x_1, x_2, p_1, p_2) \in C \subseteq (\mathbb{F}_3)^4$ berechnen sich als:

$$p_1(x) := \underbrace{\mathbf{1}}_{\lambda_{11}=\mathbf{1}} \cdot x_1 \oplus_3 \underbrace{\mathbf{2}}_{\lambda_{12}=\mathbf{2}} \cdot x_2 \quad \text{und} \quad p_2(x) := \underbrace{\mathbf{2}}_{\lambda_{21}=\mathbf{2}} \cdot x_1 \oplus_3 \underbrace{\mathbf{0}}_{\lambda_{22}=\mathbf{0}} \cdot x_2$$

Die zugehörige kanonische Generatormatrix erhält man **ohne weiteres Rechnen** sofort:

$$G = \begin{pmatrix} 1 & 0 & \overset{\text{Formel für } p_1}{\mathbf{1}} & \overset{\text{Formel für } p_2}{\mathbf{2}} \\ 0 & 1 & \mathbf{2} & \mathbf{0} \end{pmatrix}$$

... man kann sie auch aus den Datenwörtern $x = (10)$ und $x = (01)$ errechnen:

$$G = \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} & \mathbf{2} \\ 0 & \mathbf{1} & \mathbf{2} & \mathbf{0} \end{pmatrix} \qquad \begin{array}{lllll} x = (\mathbf{1}, 0) & p_1 = \mathbf{1} \cdot \mathbf{1} \oplus_3 \mathbf{2} \cdot 0 & = \mathbf{1} & p_2 = \mathbf{2} \cdot \mathbf{1} \oplus_3 \mathbf{0} \cdot 0 & = \mathbf{2} \\ x = (0, \mathbf{1}) & p_1 = \mathbf{1} \cdot 0 \oplus_3 \mathbf{2} \cdot \mathbf{1} & = \mathbf{2} & p_2 = \mathbf{2} \cdot 0 \oplus_3 \mathbf{0} \cdot \mathbf{1} & = \mathbf{0} \end{array}$$

... oder aus dem vollständigen Code auslesen:

$$C = \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}^T, \begin{pmatrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{0} \end{pmatrix}^T, \begin{pmatrix} 0 \\ 2 \\ 1 \\ 0 \end{pmatrix}^T, \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \end{pmatrix}^T, \begin{pmatrix} 1 \\ 1 \\ 0 \\ 2 \end{pmatrix}^T, \begin{pmatrix} 1 \\ 2 \\ 2 \\ 2 \end{pmatrix}^T, \begin{pmatrix} 2 \\ 0 \\ 2 \\ 1 \end{pmatrix}^T, \begin{pmatrix} 2 \\ 1 \\ 2 \\ 1 \end{pmatrix}^T, \begin{pmatrix} 2 \\ 2 \\ 2 \\ 1 \end{pmatrix}^T \right\}$$

Erstellt man also einen Code durch anhängen von Prüfstellen an die Originaldaten, so erhält man einen linearen Code, dessen kanonische Generatormatrix leicht anzugeben ist:

Die Koeffizienten der linearen Funktionen p_i bilden die Spalten der Matrix A in $G = (I, A)$

Beweis. Es Sei C ein Code wie in Lemma 5.3.27 beschrieben. Für $j = 1, \ldots, k$ setzen wir

$$c_j := (\, e_j, \;\; p_1(e_j), p_2(e_j), , \ldots, p_m(e_j))$$

wobei e_j der j-te kanonische Einheitsvektor ist. Diese Vektoren sind offensichtlich linear **un**abhäng, und bilden eine Basis von C. Um letzteres zu beweisen, zeigen wir, dass sich jedes $c \in C$ durch linearkombinieren der Vektoren c_j herstellen lässt:

Ist $x \in (\mathbb{F}_p)^k$, dann gilt $x = x_1 \cdot e_1 \;\; \oplus_p \ldots \oplus_p \;\; x_k \cdot e_k$ mit $x_i \in \mathbb{F}_p$. Da die Funktionen p_i linear sind gilt

$$p_i(x) = x_1 \cdot p_i(e_1) \oplus_p \ldots \oplus_p x_k \cdot p_i(e_k) \; .$$

Für das zu x zugehörige Codewort gilt also:

$$\begin{pmatrix} x \\ p_1(x) \\ \vdots \\ p_m(x) \end{pmatrix}^T = x_1 \cdot \begin{pmatrix} e_1 \\ p_1(e_1) \\ \vdots \\ p_m(e_1) \end{pmatrix}^T \oplus_p \cdots \oplus_p x_k \cdot \begin{pmatrix} e_k \\ p_1(e_k) \\ \vdots \\ p_m(e_k) \end{pmatrix}^T$$

Da $\{c_1, \ldots, c_k\}$ eine Basis von C ist, hat C dimension k, und die kanonische Generatormatrix hat die Zeilen c_1 bis c_k. □

5.4. Binäre Hammingcodes

Hammingodes sind leicht zu konstruierende lineare Codes, die eine "wunderschöne" Möglichkeit bieten, Übertragungsfehler zu korrigieren:

Ein Hammingcode ist ein Code C mit vielen Codewörtern und einem deswegen geringen Minimalabstand von

"nur" $\mathsf{d}(C) = 3$. Das heißt ein Hammingcode ist zwar nur 1-fehlerkorrigierend - aber das auf eine effiziente Weise. Tritt in einem Codewort ein (einzelner) Fehler auf, so ergibt sich aus den Prüfbits, die nicht zum Rest des Codes passen *sofort* die Fehlerstelle.

- Im Hammingcode liegen die Prüfstellen bzw. Prüfbits "verstreut" über den ganzen Code: In jedem Codewort c sind die Positionen c_{2^i} mit einem Index der Form $2^i \in \{1, 2, 4, 8, \ldots\}$ Prüfbits. Die restlichen Bits sind Datenbits d_j.
- Die Prüfbits verwenden ihrerseits zum Berechnen des Wertes c_{2^i} nur spezielle Datenbits. Das Prüfbit c_{2^i} berechnet sich aus allen Datenbits d_ℓ, so dass in der Binärdarstellung von ℓ das i-te Bit den Wert 1 hat.
- Wird also in einem Codewort c beim Übertragen beispielsweise das Bit d_{13} verändert, mit $13 = 8 + 4 + 1 = (10101)_2$, so passen im neu entstandenen Wort die Paritätsbits p_8, p_4 und p_1 nicht mehr zu den neuen Daten. Nur in diesen Paritätsbits wird d_{13} zur Berechnung mitverwendet. Die Positionen der nicht mehr passenden Paritätsbits summieren also zur Fehlerposition: $13 = 8 + 4 + 1$

Klingt kompliziert? Ist es aber eigentlich nicht, wenn man sich einmal Tabelle 5.1 anschaut.

Definition 5.4.1 (unsortierter Hammingcode)**.** Ein Hammingcode mit $\ell \in \mathbb{N}$ Prüfbits ist ein linearer $[n, k]$-Code $C \subset (\mathbb{F}_2)^n$ der Länge $n = 2^\ell - 1$ und Dimension $k = n - \ell$. Der Code ist sytematisch in der Menge von Stellen c_i, wenn i *keine* Potenz von 2 ist ($i = 3, 5, 6, 7, 9, \ldots$).

Für jedes Codewort $c \in C$ gilt:

- die Positionen c_{2^j} mit $2^j \in \{1, 2, \ldots, 2^{\ell-1}\}$ sind Prüfbits, die restlichen Bits sind Datenbits.
- Jedes Prüfbit c_{2^j} ist die Summe aller derjenigen Datenbits d_m, bei denen in der Binärdarstellung von m das j-te Bit den Wert 1 hat.

Korollar 5.4.2. Für einen Hammingcode gilt:

1. Jedes Datenbit d_ℓ wird genau bei der Berechnung derjenigen $\mathbf{p_{2^i}}$ verwendet, deren Indizes "$\mathbf{2^i}$" zu ℓ summieren.
2. Jedes Datenbit d_ℓ wird mindestens bei der Berechnung von zwei Paritätsbits verwendet (der Index ℓ eines Datenbits d_ℓ hat in Binärdarstellung mindestens zwei 1-en).

Beispiel 5.4.3.

Es gelten zum Beispiel (s. Tabelle 5.1):

Datenbit d_7 wird nur für $\mathbf{p_4}, \mathbf{p_2}, \mathbf{p_1}$ verwendet, denn es gilt: $7 = 4 + 2 + 1$
Datenbit d_{11} wird nur für $\mathbf{p_8}, \mathbf{p_2}, \mathbf{p_1}$ verwendet, denn es gilt: $11 = 8 + 2 + 1$
Datenbit d_{12} wird nur für $\mathbf{p_8}, \mathbf{p_4}$ verwendet, denn es gilt: $12 = 8 + 4$

Ändert man also in einem Codewort c eines Hammingcodes ein Datenbit d_k, und berechnet für die (an einer Position geänderten) Datenbits die Paritätsbits *neu*, so ändern sich *genau die* Paritätsbits in deren Berechnung d_k vorkommt. Welche Paritätsbits ändern sich? Es sind die Paritätsbits p_{2^j}, deren Indizes zu k summieren! Ändert man z.B. Bit d_5 so sind p_1 und p_4 für die neuen Daten anders als für die Originaldaten in c. Dies kann man Verwenden um das Orignialdatenwort aus w zu rekonstruieren:

Prüfbits p_i vs. Datenbits d_i im [15-4]-Hammingcode

Pos.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Typ	p_1	p_2	d_3	p_4	d_5	d_6	d_7	p_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}

Binärdarstellung der Positionsnummern:

Name	$\mathbf{p_1}$	$\mathbf{p_2}$	d_3	$\mathbf{p_4}$	d_5	d_6	d_7	$\mathbf{p_8}$	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}
Position dez.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Position binär	1	0	1	0	1	1	1	0	1	1	1	0	1	0	1
	0	1	0	1	0	1	1	0	0	1	1	1	1	1	1
	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$\mathbf{p_1} = d_3 \oplus d_5 \oplus d_7 \oplus d_9 \oplus d_{11} \oplus d_{13} \oplus d_{15}$

$\mathbf{p_2} = d_3 \oplus d_6 \oplus d_7 \oplus d_{10} \oplus d_{11} \oplus d_{14} \oplus d_{15}$

$\mathbf{p_4} = d_5 \oplus d_6 \oplus d_7 \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{15}$

$\mathbf{p_8} = d_9 \oplus d_{10} \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{15}$

Tabelle 5.1.: Berechnung von Prüfbits im $[15-4]$-Hammingcode

Korollar 5.4.4 (Hammingcodes per MLD dekodieren). Es sei $c \in H_\ell \subset (\mathbb{F}_2)^N$ ein Codewort im Hammingcode mit ℓ Prüfbits.

Beim Übertragen von c werde *genau ein* Bit geändert und es entsteht das Wort w.

Zum Rekonstruieren von c aus w geht man wie folgt vor:

1. Berechne anhand der Datenbits im Wort w alle Paritätsbits $p_{2^i}(w)$.
2. Bestimme alle Paritätsbit-Positionen 2^i an denen sich empfangene Paritsbits w_{2^j} und neu berechnete Paritätsbits $p_{2^j}(w)$ unterscheiden: diff $:= \{2^j \ : \ w_{2^j} \neq p_{2^j}(w)\}$.
   Die Summe m aus allen diesen Positionsnummern ist die Position, an der c gändert wurde:
   Für $m := \sum_{k \in \text{diff}} k$ gilt:

$$c = (w_1, \ldots, w_{m-1}, \overline{w}_m, w_{m+1}, \ldots w_N) \quad \text{mit } \overline{w}_m = \begin{cases} 1 & \text{falls } w_m = 0 \\ 0 & \text{falls } w_m = 1 \end{cases}$$

Beispiel 5.4.5 (Worte decodieren mit Hammingcodes).

Position	1	2	3	4	5	6	7
Name	p_1	p_2	d_3	p_4	d_5	d_6	d_7
Gesendet	0	0	0	0	0	0	0
Empfangen	0	0	0	0	1	0	0

Formel:

$p_4 =$				$\mathbf{d_5}$	$+d_6$	$+d_7$
$p_2 =$		d_3			$+d_6$	$+d_7$
$p_1 =$		d_3		$+\mathbf{d_5}$		$+d_7$

Position:	1	2	3	4	5	6	7
Daten:	·	·	0	·	**1**	0	0

Berechnung:

$p_4 =$				**1**	+0	+0	= **1**
$p_2 =$		0			+0	+0	= 0
$p_1 =$		0		$+\mathbf{1}$		+0	= **1**

Kontrollbits:	**1**	**0**	·	**1**	·	·	·

Fehlerkorrektur

Daten auslesen:	·	·	0	·	1	0	0
Kontrollbits neu berechnen:	**1**	**0**	·	**1**	·	·	
Empfangene Kontrollbits:	**0**	**0**	·	**0**	·	·	
Clash:	X	✓		X			
Clash Positionen:	**1**			**4**			

Geändert wurde Position: $\mathbf{5} = \mathbf{1} + \mathbf{4}$ d.h. $d_{\mathbf{5}} = 0$

Korrigiert	0	0	0	0	**0**	0	0

Es ändern sich genau die Paritätsbits, in deren Berechnung d_5 verwendet wird, d.h. es ändern sich genau p_1, p_4 wegen $5 = 4 + 1$

Satz 5.4.6. Ein Hammingcode mit ℓ Prüfbits ist ein $[n, k, 3]$-Code mit

- Länge $n = 2^\ell - 1$,
- Dimension $k = 2^\ell - 1 - \ell$ und
- Minimalabstand $\mathsf{d}(C) = 3$.

Beweis. Es sei C ein Hammingcode mit ℓ Prüfbits. Die Länge $n = 2^\ell - 1$ folgt direkt aus der Definition eines Hammingcodes. Die Idee hinter der Definition ist jedoch die folgende:

Die ℓ-vielen Prüfbits stehen für die Zweier-Potenzen $\{1, 2, , \ldots, 2^{\ell-1}\}$, aus diesen lassen sich die Zahlen $1, \ldots 2^\ell - 1$ erstellen, dieses sind die Indizes der Bits c_i in einem Codewort c.

Die Dimension $k = n - \ell = 2^\ell - \ell - 1$ zeigt man ganz analog zum Beweis von Lemma 5.3.27. Im Wesentlichen folgt die Dimension $n - \ell$ daraus, dass man von den Vorhandenen n Positionen im Code nur $n - \ell$ frei wählen kann, die ℓ-Prüfbits berechnet man aus den $n - \ell$ vielen Datenbits.

Da C mit den eingestreuten Prüfpositionen nach Lemma 5.3.27 ein linearer Code ist, lässt sich der Minimalabstand über $\mathsf{d}(C) = \min_{c \in C \setminus \{\mathbf{0}\}} \mathrm{dist}(c, \mathbf{0})$ berechnen (Lemma 5.3.8).

Es sei $c \in C \setminus \{\mathbf{0}\}$ ein beliebiges Codewort. Wir zeigen: $\mathrm{dist}(c, \mathbf{0}) \geq 3$ bzw. in c sind mindestens 3 Bits nicht 0:

- Sind alle der Datenbits $c_i \in \{c_3, c_5, c_6, c_7, c_9, \ldots, c_n\}$ gleich Null, so gilt auch für alle Prüfpositionen $c_{2^j} = 0$ mit $j = 0, \ldots, \ell - 1$, d.h. $c = \mathbf{0}$. Widerspruch.
- Ist nur eines der Datenbits $c_i \in \{c_3, c_5, c_6, c_7, c_9, \ldots, c_n\}$ ungleich Null (und alle anderen sind gleich Null), so wird c_i mindestens für das Berechnen von zwei Prüfbits c_{2^j}, c_{2^m} verwendet (nach Corollar 5.4.2). Es gilt also $c_{2^j} = c_i = 1$ und $c_{2^m} = c_i = 1$.

- Sind genau zwei der Datenbits $c_i, c_j \in \{c_3, c_5, c_6, c_7, c_9, \ldots, c_n\}$ ungleich Null (und alle anderen Datenbits sind gleich Null), so unterscheiden sich die Indizes i und j in ihrer Binärdarstellung um mindestens ein Bit, die Position dieses Bits sei $h \in \{0, 1, \ldots \ell - 1\}$.
  Es gilt dann: Wird c_i zum Berechnen des Prüfbits c_{2^h} verwendet, so wird c_j nicht verwendet und umgekeht. Es folgt also $c_{2^h} = 1$.
- Sind drei oder mehr Datenbits in c ungleich Null, so ist die Aussage, die zu zeigen ist bereits erfüllt.

□

Sortiert man die Positionen im Hammincode geschickt, so entsteht ein „neuer“ Code, der eine leicht erstellbare, kanonische Generatormatirix besitzt. Der sortierte oder kanonische Hammingcode entsteht durch sortieren eines gewöhnlichen Hammingcodes:

- Die Datenbits $d_3, d_5, d_6, d_7, d_9, \ldots, d_n$ werden nach vorn sortiert.
- Die Kontrollbits $p_{2^{\ell-1}}, \ldots, p_4, p_2, p_1, p_0$ werden in *absteigender Reihenfolge* nach hinten sortiert.

Definition 5.4.7 (kanonischer Hammingcode)**.**

Ein kanonischer Hammingcode C mit $\ell \in \mathbb{N}$ (angehängten) Prüfbits ist ein Code $C \subset \mathbb{F}_2^n$ der Länge $n := 2^\ell - 1$ mit den folgenden Eigenschaften:

Für jedes Codewort $c \in C$ gilt:

- $c \in C$ ist von der Form $c = (d, p)$ mit Datenbits $d \in (\mathbb{F}_2)^{n-\ell}$ und Prüfbits $p \in (\mathbb{F}_2)^\ell$,
  mit besonderer Indizierung: $(d, p) = (d_3, d_5, d_6, d_7, d_9, \ldots, d_n \ , \ p_{2^{\ell-1}}, \ \ldots \ , p_2, p_1)$,
  - die Prüf-Bits in $p = (p_{2^{\ell-1}}, \ldots, p_4, p_2, p_1)$ werden *absteigend* mit Zweierpotenzen indiziert.

  - die Daten-Bits $d = (d_3, d_5, d_6, d_7, d_9, \ldots, d_n)$ werden mit „Nicht-Zweierpotenzen“ indiziert[1] d.h. mit Zahlen $\{1, \ldots, n\} \setminus \{2^{\ell-1}, \ldots, 2^1, 2^0, \} = \{3, 5, 6, 7, 9, \ldots, n\}$
- Jedes Prüfbit $p_{2^{\mathtt{i}}}$ ist die Summe aller derjenigen Datenbits $d_{\mathtt{j}}$, deren Index $\mathtt{j}$ in der Binärdarstellung „$2^{\mathtt{i}}$ verwendet“, d.h. in der Binärdarstellung des zugehörigen Index $\mathtt{j}$ hat das $\mathtt{i}$-te Bit den Wert 1.

Lemma 5.4.8. Ein kanonischer, binärer Hammingcode C_ℓ mit ℓ Kontrollbits besitzt eine kanonische Generatormatrix $(\mathrm{id}_{2^{\ell-1}-\ell}, A)$.

Dabei gilt: Die j-te Zeile von A ist die Binärdarstellung der j-ten Zahl in $\{1, \ldots, n\} \setminus \{1, 2, 4, \ldots, 2^{\ell-1}\}$.

Beweis. Ein kanonischer Hammingcode C_ℓ mit ℓ Kontrollbits besitzt $n := 2^\ell - 1$ Stellen und hat eine Dimension von $k := n - \ell$.

Jedes Codewort $c \in C_\ell$ hat die Form $c = (d, p)$ mit Datenbits $d \in (\mathbb{F}_2)^{n-\ell}$ und Kontrollbits $p \in (\mathbb{F}_2)^\ell$.

Die Indizierung ist wie in Definition 5.4.7 beschrieben:

- die $n - \ell$ Datenbits werden mit $d = (d_3, d_5, d_6, d_7, d_9, \ldots d_n) \in \mathbb{F}_2^{n-\ell}$ bezeichnet,
- die ℓ Kontrollbits werden mit $p = (p_{2^{\ell-1}}, \ldots, p_{2^1}, p_{2^0}) \in (\mathbb{F}_2)^\ell$ bezeichnet.

Nach Lemma 5.3.27 besteht jede Zeile $(\mathtt{d}, \mathtt{p})$ der kanonischen Generatormatrix G aus einem Einheitsvektor $\mathtt{d} \in \{e_1, \ldots, e_k\} \subset (\mathbb{F}_2)^{n-\ell}$ mit angehängten passenden Prüfbits $\mathtt{p} = (p_{2^{\ell-1}}(\mathtt{d}), \ldots, p_1(\mathtt{d}))$.

Es sei nun m die j-te Zahl in der Daten-Indexmenge $\{3, 5, 7, 9, \ldots, n\}$.

Um $\mathtt{d} = e_j$ zu erhalten, setzen wir im Datenwort $\mathtt{d} = (\mathtt{d}_3, \mathtt{d}_5, \mathtt{d}_6, \ldots \mathtt{d}_n)$ alle Bits Null außer $\mathtt{d}_m = 1$.

Für die zugehörigen Prüfbits gilt $p_{2^i}(\mathtt{d}) = 1$ genau dann, wenn 2^i in der Binärdarstellung von m vorkommt.

Also ist das Tupel $(p_{2^{\ell-1}}(\mathtt{d}), \ldots, p_2(\mathtt{d}), p_1(\mathtt{d}))$ die Binärdarstellung von m, und die betrachtete Zeile von G lautet:

$$(\ \mathtt{d},\ p_{2^{\ell-1}}(\mathtt{d}), \ldots, p_2(\mathtt{d}), p_1(\mathtt{d})\ )$$

d.h. in der j-ten Zeile $(\mathtt{d}, \mathtt{p})$ von G, in der das Bit $\mathtt{d}_j$ den Wert 1 hat, steht in den Paritätsbits $\mathtt{p}$ die Binärdarstellung der j-ten Index-Zahl m aus $\{3, 5, 7, 9, \ldots, n\}$. □

Beispiel 5.4.9.

Für $\ell = 3$ ist hat der zugehörige sortierte binäre Hammingcode $n := 2^\ell - 1 = 7$ Bits und davon sind

- die ersten $4 = n - \ell$ Bits Datenbits d_3, d_5, d_6, d_7.
- die letzten $\ell = 3$ Bits Paritsbits p_4, p_2, p_1.

Jedes Codewort hat also die Form $(d, p) = (d_3, d_5, d_6, d_7,\ p_4, p_2, p_1)$.

Der Code hat entsprechend Dimension $k := \dim(C) = 4$ und somit $2^k = 16$ Worte. Es gelten die Formeln:

$$\begin{array}{lcccccccc} & & & & & & d_6 \text{ kommt vor in: } p_2 \text{ \& } p_4 & & \text{wegen } 4+2=6 \\ p_4 & = & & & d_5 & \oplus_2 & d_6 & \oplus_2 & d_7 \\ p_2 & = & d_3 & & & \oplus_2 & d_6 & \oplus_2 & d_7 \\ p_1 & = & d_3 & \oplus_2 & d_5 & & & \oplus_2 & d_7 \\ & & \text{wegen } 2+1=3 & & \text{kommt vor in: } p_1 \text{ \& } p_2 & & & & \end{array}$$

Für das Codewort mit Datenwerten $(d_3, d_5, \mathbf{d_6}, d_7) = (0, 0, \mathbf{1}, 0)$ bzw. $\mathbf{d_6 = 1}$ und $d_3 = d_5 = d_7 = 0$ gilt:

$$\left.\begin{array}{lcccccccccc} p_4 & = & & & 0 & \oplus_2 & \mathbf{1} & \oplus_2 & 0 & = & \mathbf{1} \\ p_2 & = & 0 & & & \oplus_2 & \mathbf{1} & \oplus_2 & 0 & = & \mathbf{1} \\ p_1 & = & 0 & \oplus_2 & 0 & & & \oplus_2 & 0 & = & \mathbf{0} \end{array}\right\} \quad 1,\ 1,\ 0 \text{ Binärdarstellung von } \mathbf{6}$$

Eines der Codewörter des Codes ist also

$$(d_3, d_5, \mathbf{d_6}, d_7, \quad \underbrace{p_4,\ p_2,\ p_1}_{\text{Biärdarstellung von 6}}) = (0, 0, \mathbf{1}, 0, \quad \underbrace{1,\ 1,\ 0}_{\text{Biärdarstellung von 6}})$$

Die kanonische Generatormatrix enthält dieses Codewort in der dritten Zeile, alle weiteren Zeilen berechnen sich ganz analog. Die kanonische Generatormatrix hat also die folgende Form:

$$G = \begin{array}{c} \begin{array}{cccc|ccc} d_3 & d_5 & d_6 & d_7 & p_4 & p_2 & p_1 \end{array} \\ \left(\begin{array}{cccc|ccc} \mathbf{1} & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & \mathbf{1} & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & \mathbf{1} & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 1 & 1 & 1 \end{array}\right) \end{array} \begin{array}{l} \leftarrow 3 \text{ binär } (= p_i \text{ in denen } d_3 \text{ vorkommt}) \\ \leftarrow 5 \text{ binär } (= p_i \text{ in denen } d_5 \text{ vorkommt}) \\ \leftarrow 6 \text{ binär } (= p_i \text{ in denen } d_6 \text{ vorkommt}) \\ \leftarrow 7 \text{ binär } (= p_i \text{ in denen } d_7 \text{ vorkommt}) \end{array}$$

5.4.1. Hamming Codes über beliebigen Körpern

Man kann Hammingcodes auch für endliche Körper mit q Elementen definieren. Dann definiert man schlicht:

Definition 5.4.10 (Hamingcode über beliebigen endlichen Körpern)**.** Sei $k \in \mathbb{N}$ und $n = \frac{q^k-1}{q-1}$ für eine Primzahlpotenz q. Ein ***Hammingcode*** ist ein $[n, n-k]$-Code über $\mathbb{F}_q$, der 1-fehlerkorrigierend ist.

Aus der Definition 5.4.10 wird zunächst nicht deutlich, ob es solche Hammingcodes über beliebigen endlichen Körpern auch tatsächlich gibt. Folgendes Lemma macht klar: Es gibt über jedem endlichen Körper und $n = \frac{q^k-1}{q-1}$ 1-fehlerkorrigierend $[n, n-k]$-Codes.

Lemma 5.4.11. Sei $k \in \mathbb{N}$. Über jedem endlichen Körper $\mathbb{F}_q$ gibt es 1-fehlerkorrigierende lineare Codes mit Blocklänge $n = \frac{q^k-1}{q-1}$ und Dimension $n-k$.

Beweis. Der Beweis ist für die Vorlesung zu umfangreich und es wird auf die Literatur verwiesen. □

5.4.2. Hamming Codes sind perfekt

Wir haben in Abschnitt ?? gesehen, wie "dicht" man die Codewörter höchstens zusammenpacken kann, so dass der Code noch k-fehlerkorrigierend ist, bzw. eine obere Schranke an die Anzahl der Codewörter in einem k-fehlerkorrigierenden Code berechnet. Konkreter gibt diese Hamming-Schranke eine obere Schranke an die Anzahl von Codewörtern, die man auf eine feste Anzahl von Bits und mit einer festen Anzahl an Symbolen im zugrundeliegenden Alphabet A verteilen kann, so dass der Code noch k-fehlerkorrigerend ist für ein festes $k \in \mathbb{N} \cup \{0\}$. Ein Code, der entsprechend dieser oberen Schranke viele Codewörter besitzt, ist diesbezüglich also optimal. Ob es tatsächlich Codes gibt, die diese Schranke auch annehmen, ist aber noch nicht bewiesen, vielleicht gibt es eine bessere obere Schranke. Codes, welche die Schranke tatsächlich annehmen, nennen wir k-perfekt.

Definition 5.4.12. Es sei A eine nichtleere und endliche Menge und es seinen $n, r \in \mathbb{N}$. Ein Code $C \in A^n$ heißt ***k-perfekt***, wenn gilt

$$A^n = \bigcup_{c \in C} \mathcal{B}_k(c).$$

Das bedeutet, dass zu jedem $x \in A^n$ gibt es genau ein Codewort $c \in C$ mit $x \in \mathcal{B}_k(c)$.

Beispiel 5.4.13.

Es sei $A = \{0, 1\}$, $n = 2 \cdot k + 1 \geq 3$ und $C = \{(0, \ldots, 0), (1, \ldots, 1)\} \subset A^n$.

Satz 5.4.14. Es sei A endlich und nicht leer und es seinen $n, k \in \mathbb{N}$. Für einen Code $C \in A^n$ sind dann äquivalent

- C ist k-perfekt.
- C ist k-fehlerkorrigierend und es gilt

$$|C| = |A|^n \cdot \left(\sum_{i=0}^{k} \binom{n}{i} \cdot (a-1)^i \right)^{-1}.$$

Beweis. Vergleiche den Beweis zu Satz 5.2.15 (Hamming-Schranke). □

Wir beweisen nun, dass es tatsächlich 1-perfekte Codes gibt, nämlich ist jeder Hammingcode 1-perfekt.

Satz 5.4.15. Jeder Hammingcode ist 1-perfekt.

Beweis. Es seien also q, n und m gegeben wie in Definition 5.4.10. Sei nun $C \subset (\mathbb{F}_q)^n$ ein Hammingcode mit Dimension $m - n$. Da C ein Hammingcode ist, ist er 1-fehlerkorrigierend und somit befinden sich in der Hammingkugel mit Radius 1 um jedes Codewort $c \in C$ keine weiteren Codewörter. In jeder dieser Hamminkugeln liegen $1 + n(q-1)$ Wörter - das Codewort c im Mittelpunkt und jedes Wort, dass an einer Stelle einen anderen

Wert aus $\mathbb{F}_q$ als c annimmt. Es gibt n Stellen und $q-1$ andere Zahlen in $\mathbb{F}_q$ als den Wert den c dort annimmt. Es ist also

$$|C| \cdot (1 + n(q-1)) = |C| \cdot \sum_{k=0}^{1} \binom{n}{k} (q-1)^k. \tag{5.1}$$

Bleibt zu zählen, wie viele Codewörter es in C gibt. Ein Hammingcode hat Dimension $n-m$ und ist ein Vektorraum über $\mathbb{F}_q$. Es gibt q^{n-m} Vektoren im $\mathbb{F}_q^{n-k}$. Mit (5.2) und dem Zusammenhang aus der Definition, dass $n = \frac{q^m-1}{q-1}$ ist, gilt also

$$|C| \cdot (1 + n(q-1)) = q^{n-m} \cdot (1 + n(q-1)) = q^{n-m} \cdot q^m = q^n = |\mathbb{F}_q|^n. \tag{5.2}$$

Mit Satz 5.4.14 folgt die Behauptung. □

Teil II.

Numerik

A Anhang

A.1. Der euklidische Algorithmus in Tabellenform

Der Eukidischen Algorithmus in Form von Algorithmus 2.1.3 lässt sich angenehm übersichtlich in Tabellenform durchführen.

Init

Laufindex j	Faktor Vor-Vor-r m_j	aktueller Rest r_j	Faktor vor a s_j	Faktor vor b t_j
0	/	a	1	0
1		b	0	1

Berechnen von m_1 in Zeile 1

Laufindex j	Faktor Vor-Vor-r m_j	aktueller Rest r_j	Faktor vor a s_j	Faktor vor b t_j
0	/	a	1	0
1	$\lfloor \frac{a}{b} \rfloor$	b	0	1

Berechnen von r_2, s_2, t_2 in Zeile 2

Laufindex j	Faktor Vor-Vor-r m_j	aktueller Rest r_j	Faktor vor a s_j	Faktor vor b t_j
0	/	a	1	0
1	$\lfloor \frac{a}{b} \rfloor$	b	0	1
2		r_2	1	$-\lfloor \frac{a}{b} \rfloor$

a	1	0
$-\lfloor \frac{a}{b} \rfloor \cdot b$	$-\lfloor \frac{a}{b} \rfloor \cdot 0$	$-\lfloor \frac{a}{b} \rfloor \cdot 1$
$= r_2$	$= 1$	$-\lfloor \frac{a}{b} \rfloor$

$r_2 = a - \lfloor \frac{a}{b} \rfloor \cdot b$

$s_2 = 1 - \lfloor \frac{a}{b} \rfloor \cdot 0 = 1$

$t_2 = 0 - \lfloor \frac{a}{b} \rfloor \cdot 1 = -\lfloor \frac{a}{b} \rfloor$

Berechnen von m_2 in Zeile 2

Laufindex j	Faktor Vor-Vor-r m_j	aktueller Rest r_j	Faktor vor a s_j	Faktor vor b t_j
0	/	a	1	0
1	$\lfloor \frac{a}{b} \rfloor$	b	0	1
2	$\lfloor \frac{b}{r_2} \rfloor$	r_2	1	$-\lfloor \frac{a}{b} \rfloor$

...