

Kapitel 2

Algorithmen und Komplexität

In diesem Kapitel beschäftigen wir uns mit **Algorithmen**: automatischen Verfahren zur Lösung von Problemen. Wir beginnen damit, den Begriff des Algorithmus selbst zu erklären – zumindest so genau wie für unsere Zwecke nötig – und machen uns anhand zahlreicher Beispiele mit seinen Eigenschaften vertraut. Daraufhin erläutern wir, was es bedeutet, dass ein mathematisches Problem **algorithmisch lösbar** ist. Ein besonders wichtiger Gesichtspunkt ist dabei *Effizienz*, ein Maß dafür, wie „praktikabel“ ein Algorithmus ist. Wir unterscheiden zwischen effizient lösbaren und effizient verifizierbaren Problemen und besprechen, welche Methoden es zur Verkürzung der Laufzeit von Algorithmen gibt.

2.1 Algorithmen

Was ist ein Algorithmus?

Seit ihrem Anbeginn sucht die Mathematik nach Methoden, mit deren Hilfe die Lösung eines Problems möglichst schnell und sozusagen „automatisch“ gefunden werden kann. Solche Verfahren werden als **Algorithmen** bezeichnet. Die schriftliche Addition, Multiplikation und Division, die wir schon in der Grundschule lernen, sind Beispiele von Algorithmen, ebenso wie das Sieb des Eratosthenes und der Euklidische Algorithmus, denen wir im letzten Kapitel begegnet sind. Mit der Entwicklung des Computers haben die Suche nach Algorithmen und deren systematische Betrachtung seit der zweiten Hälfte des zwanzigsten Jahrhunderts eine noch größere Bedeutung gewonnen.

Was aber ist ein Algorithmus? Wir stellen uns darunter eine Anleitung vor, die wir – wie ein Kochrezept – nur Schritt für Schritt befolgen müssen, um das vorgegebene Problem zu lösen. Zur Erläuterung betrachten wir zwei sehr unterschiedliche „Algorithmen“. Der erste ist in der Tat ein Kochrezept (welches wir im Eigenversuch getestet haben).

ALGORITHMUS PFANNKUCHEN

Eingabe: Ein Ei, eine Tasse Mehl, eine Tasse Milch, eine Prise Salz, ein Teelöffel Sonnenblumenöl.

1. Verrühre das Ei, das Mehl und die Milch in einer Schüssel zu Pfannkuchenteig. Füge das Salz hinzu.
2. Erhitze das Öl in einer Bratpfanne auf mittelhoher Flamme, bis es brutzelt.
3. Fülle den Pfannkuchenteig in die Pfanne und schwenke diese, bis der Teig verteilt ist.
4. Wende den Pfannkuchen nach 2-3 Minuten.
5. Nach weiteren 2 Minuten ist der Pfannkuchen zum Verzehr bereit.

Der zweite „Algorithmus“ ist eine Anleitung zum Verfassen eines Bestseller-Krimis.

ALGORITHMUS BESTSELLER-KRIMI

1. Erfinde eine kantige, aber sympathische Hauptfigur, vorzugsweise eine Privatdetektivin oder Polizeikommissarin.
2. Erfinde außerdem eine (fast) perfekte Straftat.
3. Entwirf eine Handlung, in welcher diese Straftat von der Hauptfigur durch Ermittlungen und evtl. eine Reihe glücklicher Zufälle aufgeklärt wird.
4. Beschreibe diese Handlung auf unterhaltsame und spannende Weise in einem Roman.

Es fällt sofort auf, dass diese beiden Anleitungen von sehr verschiedener Natur sind. Während die erste die einzelnen Schritte und ihre Abfolge klar beschreibt, lässt die zweite viele Details offen. Auch wenn diese Beispiele überspitzt erscheinen mögen, veranschaulichen sie genau die wesentliche Eigenschaft von Algorithmen: Ein solcher darf von der ausführenden Person nicht erwarten, eigene Denkarbeit zu leisten und kreativ zu sein. Die korrekte Befolgung der Anweisungen

sollte stets – unabhängig von persönlicher Fähigkeit oder Begabung – dasselbe (korrekte) Ergebnis liefern.

Insofern stimmt die Leserin hoffentlich mit uns überein, dass das Pfannkuchen-Rezept sich (sofern das für nicht-mathematische Beispiele überhaupt möglich ist) als Algorithmus qualifiziert, während die Anleitung zum Besteller-Schreiben hinter diesen Ansprüchen weit zurückbleibt.

Wir formulieren nun etwas genauer, welche Forderungen ein Algorithmus erfüllen soll:

- (a) Er hat eine endliche Beschreibung;
- (b) er besteht nur aus „elementaren“ Schritten, seine Durchführung erfordert insbesondere keine Erfindungskraft;
- (c) er darf zwar beliebig große, zu jedem Zeitpunkt aber nur **endliche** Ressourcen (Papier, Tinte, Speicherplatz, ...) benötigen;
- (d) zu jedem Zeitpunkt ist der nächste auszuführende Schritt eindeutig bestimmt (**Determinismus**).

Wir können stattdessen ebenfalls sagen: *Ein Algorithmus ist ein Verfahren, das sich in einer gängigen Programmiersprache auf einem Computer implementieren lässt.* Selbstverständlich ist das keine *Definition* im formalen Sinne der Mathematik. Mit etwas Aufwand ist es zwar möglich, den Algorithmusbegriff mathematisch zu erfassen (siehe unten), aber für dieses Buch begnügen wir uns mit obiger informeller Beschreibung und füllen sie im Folgenden etwas mit Leben.

Beispiele und Erläuterungen zum Algorithmusbegriff

Wir beginnen mit einem Verfahren, das schon in der Grundschule gelehrt wird – der schriftlichen Addition. Der Einfachheit halber formulieren wir es für Zahlen im **Binärsystem**. (Siehe Anmerkung 2.1.4.)

	a	0	1
b			
0		0	1
1		1	10

(a) Übertrag 0

	a	0	1
b			
0		1	10
1		10	11

(b) Übertrag 1

Abbildung 2.1. Binäre Additionstafeln, zur Verwendung im Algorithmus ADDITION.

ALGORITHMUS ADDITION

Eingabe: Zwei natürliche Zahlen m und n , dargestellt in Binärschreibweise.

1. Schreibe beide Zahlen so untereinander, dass die letzten beiden Ziffern von m und n übereinanderstehen, ebenso die vorletzten beiden Ziffern und so weiter. Ziehe einen Strich unter die beiden Zahlen.
2. Haben die Zahlen ungleiche Stellenanzahl, so ergänze die kürzere durch führende Nullen, bis beide Zahlen die gleiche Anzahl s von Stellen haben.
3. Notiere in einem separaten „Übertragskästchen“ eine Null und setze $j := 1$.
4. Es sei a die j -te Stelle von m und b die j -te Stelle von n , jeweils von *rechts* gezählt. Lies aus der dem Wert im Übertragskästchen entsprechenden Tabelle aus Abbildung 2.1 den Wert in der a -ten Spalte und b -ten Zeile ab. Wir nennen diese Zahl k .
5. Trage die letzte Ziffer der Zahl k an der j -ten Stelle unterhalb der Zahlen m und n ein.
6. Ist k einstellig, so ersetzen wir die Zahl im Übertragskästchen durch eine Null, andernfalls durch eine Eins.
7. Ist $j \neq s$ (d.h. wir sind noch nicht ganz links angekommen), dann ersetzen wir j durch $j + 1$ und kehren zu 4. zurück.
8. Andernfalls notieren wir die Ziffer aus dem Übertragskästchen als erste Stelle vor dem Rest unseres Ergebnisses und sind fertig.

Die Leserin sei dazu aufgefordert, das Verfahren an einem Beispiel selbst auszuführen. Für die Addition der Zahlen 3 und 6 sind die einzelnen Zwischenschritte in Abbildung 2.2 dargestellt. Im Binärsystem entspricht der Zahl 3 die Ziffern-

	0	0	1	1	1
11	011	011	011	011	011
110	110	110	110	110	110
		1	01	001	1001

Abbildung 2.2. Berechnung von 3 + 6 mit Hilfe von ADDITION.

folge 11 und der Zahl 6 die Ziffernfolge 110; das Ergebnis 1001 ist in der Tat die Binärdarstellung der Zahl 9.

Auch wenn es sich etwas umständlich liest, haben wir nichts anderes als das übliche Verfahren der schriftlichen Addition beschrieben. (Wir haben übrigens das Binär- nur deshalb anstelle des Dezimalsystems gewählt, weil dann die Additionstabeln überschaubarer bleiben.) Diese Anleitung genügt unseren Kriterien für einen Algorithmus. Die Beschreibung – bestehend aus dem Algorithmus zusammen mit den Additionstabeln aus Abbildung 2.1 – ist sicherlich endlich. Jeder Schritt enthält eine Anweisung, die wir guten Gewissens als „elementar“ bezeichnen können, und verwendet außerdem nur endlich viele Ressourcen. Zu guter Letzt ist die Abfolge der Schritte eindeutig bestimmt; unser Verfahren ist also deterministisch.

Vielleicht möchten wir aber nicht nur zwei Zahlen addieren, sondern gleich eine ganze Liste. Wir könnten das übliche Verfahren für die schriftliche Addition mehrerer Zahlen ausformulieren, aber es geht noch einfacher:

ALGORITHMUS ADDITION-VIELE

Eingabe: Eine Liste von mindestens einer und höchstens endlich vielen natürlichen Zahlen.

1. Enthält die gegebene Liste nur eine Zahl, so sind wir fertig.
2. Andernfalls wende den Algorithmus ADDITION auf die letzten beiden Zahlen der Liste an.
3. Ersetze die letzten beiden Zahlen der Liste durch die in Schritt 2. errechnete Zahl.
4. Kehre zu Schritt 1. zurück.

Hier sehen wir eine wichtige Eigenschaft von Algorithmen – sie lassen sich kombinieren. Das heißt, wir können in einer Algorithmenbeschreibung andere Algorithmen verwenden, die wir bereits formuliert haben. Das macht das Leben

einfacher, denn wie das Beispiel der schriftlichen Addition zeigt, kann es ziemlich aufwendig sein, selbst einfache Verfahren detailliert in ihre Einzelschritte zu zerlegen.

Für den Rest des Buches lassen wir daher die Grundrechenarten in den natürlichen Zahlen als elementare Anweisungen zu, da für diese wohlbekannte Algorithmen existieren (siehe Aufgabe 2.1.1). Ebenso werden wir bereits formulierte Algorithmen in späteren Kapiteln wiederverwenden.

Ein weiteres Beispiel möchten wir noch erwähnen; sei dazu $n \in \mathbb{N}$. Es ist eine bekannte Tatsache (siehe etwa [Lo]), dass $n \cdot \pi$ und $n \cdot e$ selbst keine natürlichen Zahlen sind – unabhängig von der Wahl von n . Aber wie steht es mit der Zahl $n \cdot (\pi + e)$? Um zu versuchen, eine natürliche Zahl n zu finden, für die auch $n \cdot (\pi + e) \in \mathbb{N}$ ist, könnten wir folgenden „Algorithmus“ verwenden:

ALGORITHMUS $N^*(PI+E)$

1. Setze $n := 1$.
2. Berechne die Zahl $a_n := n \cdot (\pi + e)$.
3. Falls die errechnete Zahl a_n ganzzahlig ist, sind wir fertig.
4. Andernfalls kehre zu Schritt **2.** zurück, wobei n durch die Zahl $n + 1$ ersetzt wird.

Auf den ersten Blick sieht das wie ein Algorithmus aus. Bei genauerer Betrachtung fällt aber auf, dass wir zum Beispiel nichts dazu gesagt haben, *wie* die Rechnung in Schritt **2.** durchgeführt werden soll. Natürlich können wir mit Hilfe eines Computers oder Taschenrechners (und mit viel Aufwand auch per Hand) die Zahl a_n mit beliebiger Genauigkeit, d.h. gerundet bis auf eine vorgegebene Zahl von Dezimalstellen, ausrechnen. Aber wenn alle so berechneten Nachkommastellen gleich Null sind, so heißt das noch lange nicht, dass $a_n \in \mathbb{N}$ gilt; es könnte sein, dass die gewählte Genauigkeit nicht ausreicht! (Man berechne als Beispiel die Zahl $a_{56602103}$ mit einem Taschenrechner.) Also ist eine solche Rechnung nicht ausreichend, um Schritt **3.** korrekt auszuführen. In Anbetracht dieser Überlegungen ist hier die Forderung nach „elementaren“ Schritten verletzt: Das Verfahren stellt keinen Algorithmus dar.

Dieses Beispiel führt vor Augen, dass wir bei der Formulierung von Algorithmen etwas vorsichtig sein müssen. Nichtsdestotrotz hoffen wir, die Leserin davon überzeugt zu haben, dass wir Algorithmen *stets als solche erkennen können*. Ist bei jedem der angegebenen Schritte eindeutig klar, dass und wie dieser automatisch ausgeführt werden kann, so haben wir einen Algorithmus vor uns – sonst nicht. Wir ermutigen explizit dazu, sich bei jeder neuen Algorithmenbeschreibung

zu vergewissern, dass die angegebenen Schritte in der Tat entweder elementar sind oder nur die Ausführung eines bereits bekannten Verfahrens erfordern.

Noch eine Bemerkung zum Determinismus, also Teil (d) der zu Anfang des Abschnittes formulierten Bedingungen: Sicherlich erscheint diese Forderung auf den ersten Blick einleuchtend. Allerdings kann es sinnvoll sein, sie etwas abzuschwächen, indem wir **Zufallsentscheidungen** erlauben. Die Vorteile solcher **randomisierter Algorithmen** lernen wir in Abschnitt 2.5 kennen.

Formale Definitionen des Algorithmusbegriffs

Die Suche nach Algorithmen erhielt Ende des neunzehnten und Beginn des zwanzigsten Jahrhunderts eine neue Bedeutung. Die ersten „echten“ Computer wurden zwar erst Mitte des zwanzigsten Jahrhunderts, aber die im Rahmen der industriellen Revolution erfolgte Mechanisierung verschiedenster Prozesse hatte bereits begonnen, den Blick von Mathematikerinnen für die algorithmische Lösung von Problemen zu schärfen.

David Hilbert, einer der führenden mathematischen Denker seiner Zeit, stellte daher in den frühen Jahren des zwanzigsten Jahrhunderts ein ambitioniertes Programm auf. Er wollte die Mathematik ein für alle Mal auf eine formale Grundlage stellen, die keine Widersprüche enthielte und in der jede wahre mathematische Aussage auch beweisbar sei. Insbesondere suchte Hilbert nach einer Methode (also einem Algorithmus), mit welcher die Wahrheit einer beliebigen mathematischen Aussage entschieden werden kann. (Dies ist bekannt als **Hilberts Entscheidungsproblem**.)

Hilberts Fragen motivierten in den dreißiger Jahren gleich mehrere Mathematiker (darunter Alan Turing und Alonso Church), den Begriff des Algorithmus mathematisch zu formalisieren. Obwohl die dabei entstandenen Konzepte auf den ersten Blick wenig gemein haben, stellte sich schnell heraus, dass sie äquivalent sind. Dasselbe gilt für alle Algorithmusdefinitionen, die seitdem vorgestellt wurden. Insbesondere sind die Verfahren, die sich in einer der heute üblichen Programmiersprachen implementieren lassen, genau dieselben, welche in Turings Maschinenmodell beschrieben werden können. Aus diesem Grund geht man heute davon aus, dass diese vielfältigen Definitionen tatsächlich genau das widerspiegeln, was wir intuitiv unter einem „Algorithmus“ verstehen. Das wird auch als die „Church-Turing-These“ bezeichnet und rechtfertigt, dass wir uns in diesem Buch mit einem informellen Algorithmus-Begriff begnügen.

Aufgaben

2.1.1. Aufgabe. Formuliere Algorithmen für:

- (a) Die Multiplikation zweier natürlicher Zahlen.

- (b) Die Subtraktion einer natürlichen Zahl von einer anderen.
- (c) Die Division mit Rest einer natürlichen Zahl durch eine andere.

Formuliere außerdem einen Algorithmus, der bestimmt, welche von zwei gegebenen natürlichen Zahlen größer ist als die andere.

2.1.2. Aufgabe. Überprüfe, dass der in Abschnitt 1.4 besprochene „Euklidische Algorithmus“ tatsächlich die Kriterien für einen Algorithmus erfüllt.

Weiterführende Übungen und Anmerkungen

2.1.3. Der Begriff „Algorithmus“ leitet sich vom Namen des persischen Mathematikers **al-Chwarizmi** ab, der im neunten Jahrhundert ein wichtiges Rechenwerk verfasste [ANF, S. 158].

2.1.4. Das **Binärsystem**, das aus der Schule bekannt sein dürfte, verwendet nur die Ziffern Null und Eins. (Im Gegensatz dazu verwendet das übliche **Dezimalsystem** die Ziffern Null bis Neun.) Die Zahlen Null bis Zehn zum Beispiel werden im Binärsystem wie folgt dargestellt: 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010. In Computern werden Zahlen intern stets im Binärsystem dargestellt. Das liegt daran, dass Speicherzellen üblicherweise genau zwei Zustände (Null und Eins) habe und daher genau eine Ziffer einer Zahl in Binärdarstellung enthalten können.

2.1.5. Einen relativ einfachen Beweis der Irrationalität von π findet man in [NZM]. Dort wird auch im Rahmen der Aufgaben hergeleitet, dass die Eulersche Konstante e irrational ist. Die Frage, auf die sich der „Algorithmus“ $N^*(\pi+e)$ bezog, nämlich ob $\pi+e$ irrational ist, ist dagegen offen!

Ein noch schwierigeres Unterfangen ist es, unter den irrationalen Zahlen noch zwischen sogenannten **algebraischen** Zahlen wie $\sqrt{2}$ und $\sqrt[3]{3}$ und **transzendenten** Zahlen wie e und π zu unterscheiden. Für eine Diskussion dieser Begriffe verweisen wir auf Kapitel 2 in [Lo].

2.1.6. Aufgabe (P). Wir betrachten den folgenden einfachen Algorithmus:

ALGORITHMUS COLLATZ

Eingabe: Eine natürliche Zahl n .

1. Schreibe die Zahl n auf.
2. Falls $n = 1$ ist, sind wir fertig.
3. Falls n gerade ist, ersetze n durch $\frac{n}{2}$. Andernfalls ersetze n durch $3n + 1$.
4. Kehre zu Schritt 1. zurück.

- (a) Implementiere den Algorithmus COLLATZ in einer beliebigen Programmiersprache.
- (b) Führe den Algorithmus für die Zahlen 1 bis 100 aus. Was fällt auf?

Die Vermutung liegt nahe, dass der Algorithmus COLLATZ für jeden Startwert irgendwann die Zahl 1 erreicht, also anhält. Das wird als das $3n + 1$ -**Problem** bezeichnet oder auch als **Collatz-Vermutung** (nach dem Mathematiker Lothar Collatz, der dieses Problem 1936 formulierte). Obwohl sie durch Computorexperimente für mehr als die ersten 27 000 000 000 000 000 natürlichen Zahlen verifiziert wurde, ist bis heute kein Beweis der Collatz-Vermutung bekannt!

2.2 Algorithmisch lösbare und unlösbare Probleme

Unser Ziel ist es, Algorithmen zur automatischen Lösung von **Problemen** zu verwenden. Dafür sollten wir zunächst klären, was wir überhaupt unter einem Problem verstehen. Auch hier soll uns ein informelles Konzept genügen: Ein Problem besteht aus einer Sammlung von **Instanzen** oder **Eingaben**; für jede von diesen suchen wir nach einer geeigneten **Ausgabe**. Häufig (aber nicht immer) gibt es zu jeder Instanz eine einzige geeignete Ausgabe. Das Problem kann dann oft prägnant als Frage formuliert werden, etwa wie folgt:

PROBLEM SUMME

Eingabe: Zwei natürliche Zahlen n und m .

Frage: Was ist $n + m$?

Sowohl Eingaben als auch Ausgaben werden üblicherweise als **Zeichenketten** dargestellt, d.h. als eine endliche Aneinanderreihung von Buchstaben eines gewissen Alphabets. Um den informellen Charakter unserer Betrachtungen zu erhalten, gehen wir üblicherweise nicht genau darauf ein, wie die Instanzen als Zeichenketten codiert werden. Handelt es sich dabei um natürliche Zahlen (wie bei den Problemen, welche uns am meisten interessieren), so stellen wir sie uns stets als in ihrer Binär- oder Dezimaldarstellung gegeben vor.

Ein Algorithmus **löst** nun ein Problem, falls er für jede Eingabe nach endlich vielen Schritten seine Ausführung mit einer korrekten Ausgabe beendet. Zum Beispiel löst der im vorigen Abschnitt besprochene Algorithmus ADDITION das Problem SUMME. Ein Problem, für welches ein Lösungsalgorithmus existiert, wird als **algorithmisch lösbar** (oder auch **entscheidbar**) bezeichnet; andernfalls ist es **algorithmisch unlösbar** oder **unentscheidbar**.

Ferner werden wir **Entscheidungsprobleme**, deren gesuchte Ausgabe stets entweder „ja“ oder „nein“ ist, von allgemeineren **Berechnungsproblemen** unterscheiden. Zum Beispiel ist das Problem, welches im Mittelpunkt dieses Buches steht, ein Entscheidungsproblem:

PROBLEM PRIMALITÄT

Eingabe: Eine natürliche Zahl $n \geq 2$.

Frage: Ist n eine Primzahl?

Im nächsten Abschnitt machen wir die – vielleicht überraschende – Entdeckung, dass jedes Berechnungsproblem auf ein äquivalentes Entscheidungsproblem reduziert werden kann. Daher beschränken wir uns im Folgenden meist auf die in vielerlei Hinsicht einfachere Betrachtung von Entscheidungsproblemen.

Diese haben zwei verschiedene Typen von Eingaben: Für **positive** Instanzen ist die gesuchte Antwort „ja“, während sie für die übrigen, **negativen**, Instanzen „nein“ lautet. Für das Problem PRIMALITÄT sind die positiven Instanzen genau die Primzahlen und die negativen genau die zusammengesetzten Zahlen.

Zu jedem Entscheidungsproblem gibt es ein **duales** Problem, nämlich das, in welchem die Rollen der positiven und negativen Instanzen vertauscht sind. Das zu PRIMALITÄT duale Problem ist dann das der Zusammengesetztheit:

PROBLEM ZUSAMMENGESETZTHEIT

Eingabe: Eine natürliche Zahl $n \geq 2$.

Frage: Ist n eine zusammengesetzte Zahl?

Es mag unsinnig erscheinen, zwischen einem Entscheidungsproblem und seinem dualen Problem zu unterscheiden: Ist eines der beiden algorithmisch lösbar, so auch das andere; wir müssen nur die Antworten „ja“ und „nein“ vertauschen. In den Abschnitten 2.4 und 2.5 werden wir aber Konzepte der Komplexitätstheorie kennenlernen, in denen positive und negative Instanzen verschiedene Rollen spielen und für die diese Unterscheidung daher wichtig ist!

Zu guter Letzt weisen wir noch darauf hin, dass Probleme, für die es nur *endlich viele* verschiedene Eingaben gibt, nach unserer Definition automatisch algorithmisch lösbar sind. Es gibt dann nämlich eine endliche Liste, die zu jeder möglichen Eingabe eine korrekte Ausgabe enthält; der Algorithmus muss

Primzahltests für Einsteiger
Zahlentheorie – Algorithmik – Kryptographie
Waldecker, R.; Rempe-Gillen, L.
2016, XX, 211 S., Softcover
ISBN: 978-3-658-11216-5