

## Kapitel 1

# Einführung und historischer Überblick

*oder worum es überhaupt geht*

*Und war dein Anfang  
auch gering, dein Ende  
wird gewaltig groß.*

HI0B 8:7

Computer sind erstaunliche Maschinen. Sie scheinen zu allem fähig zu sein. Sie fliegen Flugzeuge und Raumschiffe und kontrollieren Kraftwerke und gefährliche Chemiefabriken. Unternehmen können nicht länger ohne sie arbeiten, und eine wachsende Anzahl von hochentwickelten medizinischen Untersuchungsmethoden könnte ohne sie nicht ausgeführt werden. Sie dienen Anwälten und Richtern, die in einer großen Anzahl von dokumentierten Gerichtsverhandlungen nach juristischen Präzedenzfällen suchen, und helfen Wissenschaftlern, unheimlich komplizierte und aufwendige mathematische Berechnungen auszuführen. Sie steuern und kontrollieren Millionen von Anrufen in Telefonnetzen über Kontinente hinweg. Sie führen Aufgaben mit enormer Präzision aus – vom Kartenlesen und Schriftsetzen bis hin zur grafischen Bildbearbeitung und zum Entwurf von integrierten Schaltungen. Sie können uns viele langweilige Arbeiten abnehmen, wie das akribische Verfolgen von Haushaltsausgaben, und bieten uns gleichzeitig diverse Unterhaltungsmöglichkeiten in Form von Computerspielen oder digitaler Musik. Außerdem arbeiten die heutigen Computer hart daran, die noch leistungsfähigeren Computer von morgen zu entwerfen.

Es ist deshalb um so erstaunlicher, dass man sich digitale Computer – auch die modernsten und komplexesten – einfach als eine große Ansammlung von Schaltern vorstellen kann. Diese Schalter oder so genannten **Bits** werden nicht etwa vom Benutzer „bedient“, sondern sie sind spezielle, innere Schalter, die vom Computer selbst „umgeklappt“ werden. Jedes Bit kann sich in einer von zwei Positionen befinden, oder, anders ausgedrückt, einen der beiden **Werte** 0 oder 1 annehmen. Gewöhnlich ist der Wert eines Bits durch eine elektronische Kenngröße bestimmt, beispielsweise dadurch, dass ein bestimmter Punkt eine positive oder negative Ladung besitzt.

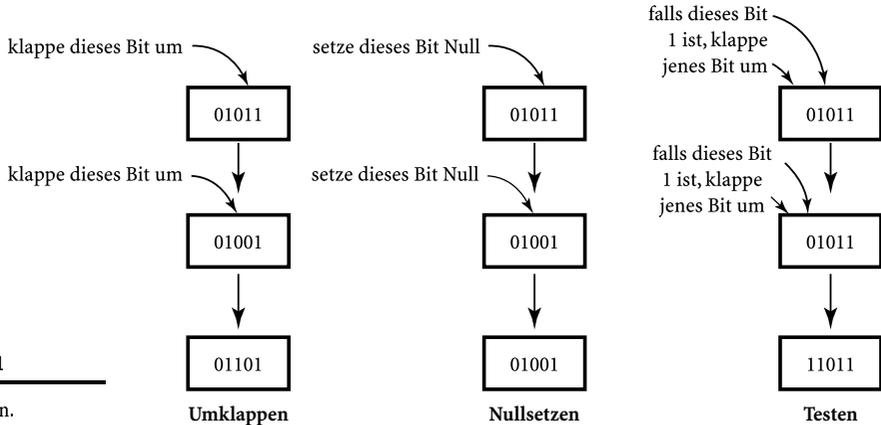


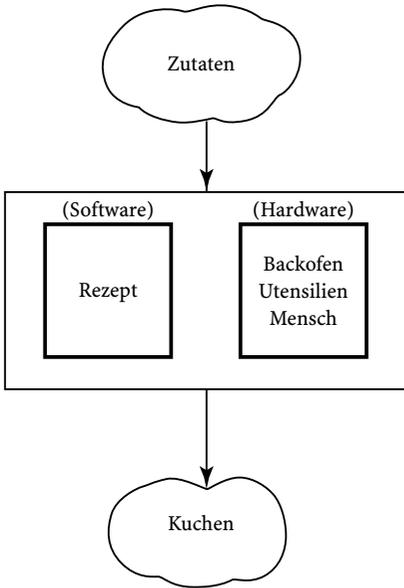
Abbildung 1.1

Bitoperationen.

Ein Computer kann nur eine kleine Anzahl von extrem einfachen Operationen direkt ausführen, wie das Umklappen, Nullsetzen und Testen eines Bits. Das Umklappen ändert den Wert eines Bits, Nullsetzen sichert, dass sich das Bit in der Nullstellung befindet, und das Testen führt zu einer Aktion, wenn sich der Schalter bereits in der Nullstellung befindet, und zu einer anderen, wenn er es nicht tut (siehe Abbildung 1.1).

Computer können sich in vielerlei Hinsicht unterscheiden: in ihrer Größe (also in der Anzahl der verfügbaren Bits), in den elementaren Operationen, die sie ausführen können, in der Geschwindigkeit, mit der diese Operationen ausgeführt werden, im physikalischen Medium, das die Bits verkörpert, deren interner Organisation und maßgeblich in ihrer äußeren Ausstattung. Der letzte Punkt besagt, dass zwei Computer, die sonst in all ihren Funktionen gleichartig sind, einem Betrachter sehr verschieden erscheinen können: Einer könnte einem Fernseher mit Fernbedienung ähneln, während der andere hinter den Wahltafeln und Knöpfen einer automatischen Strickmaschine versteckt ist. Das äußere Erscheinungsbild ist im Vergleich zu den Bits und deren interner Organisation jedoch von untergeordneter Bedeutung. Es sind die Bits, die die externen Reize „wahrnehmen“, die aus der Außenwelt über Touchscreens (berührungsempfindliche Bildschirme), Joysticks, Tastaturtasten, elektronische Nachrichtenverbindungen und sogar Mikrofone und Kameras ankommen. Es sind die Bits, die „entscheiden“, wie auf diese Reize reagiert werden soll, und entsprechend antworten, indem andere Reize über Bildschirme, Drucker, Lautsprecher, Piepser, Robotorarme und anderes nach außen weitergegeben werden.

Wie machen das die Computer? Was wandelt die trivialen Operationen auf Bits in die unglaublichen Leistungen um, die von Computern vollbracht werden? Darauf zu antworten ist zentrales Konzept dieses Buches: der Berechnungsprozess und der Algorithmus, der diesen Prozess beschreibt und steuert.



**Abbildung 1.2**  
Kuchenbacken.

## Kleine Gastronomie

Stellen Sie sich eine Küche vor, in der es einen Vorrat von Zutaten gibt, eine Reihe von Backutensilien, einen Backofen und einen (menschlichen) Bäcker. Das Backen eines leckeren Rosinenkuchens ist ein Prozess, der vom Bäcker mit den Zutaten, mithilfe des Backofens und, am wichtigsten, gemäß dem Rezept ausgeführt wird. Die Zutaten sind die **Eingaben** des Prozesses, der Kuchen ist dessen **Ausgabe** und das Rezept der **Algorithmus**. Der Algorithmus beschreibt also die Aktivitäten, aus denen sich der Prozess zusammensetzt. Die Rezepte oder Algorithmen, die für die hier diskutierte Menge von Prozessen relevant sind, werden allgemein als **Software** bezeichnet, während die Utensilien und der Backofen für die **Hardware** stehen. Der Bäcker kann in diesem Fall als Teil der Hardware betrachtet werden (siehe Abbildung 1.2).

Wie bei den Bitoperationen hat der Bäcker mit dem Backofen und den Utensilien nur sehr begrenzte unmittelbare Fähigkeiten. Diese kuchenbackende Hardware kann schütten, mixen, bestreichen, eintropfen lassen, den Backofen anmachen, die Backofentür öffnen, Zeit oder Mengen abmessen, aber nicht unmittelbar Kuchen backen. Es sind die Rezepte – diese magischen Beschreibungen, die die begrenzten Fähigkeiten der kuchenbackenden Hardware in Kuchen verwandeln – und nicht Backöfen oder Bäcker, die Gegenstand dieses Buches sind.

Wie bereits erwähnt, werden die Rezepte hier als Algorithmen bezeichnet, während ihr Studium und das Fachwissen darüber in diesem Buch **Algorithmik** getauft wird. Die Analogie mit dem Rezept ist so exakt wie möglich: Das Rezept, als ab-

straktes Gebilde, entspricht dem Algorithmus; die formal aufgeschriebene Version des Rezeptes, wie man sie in einem Kochbuch findet, entspricht einem **Computerprogramm**. Die Bezeichnung Software bezieht sich dagegen eher auf Programme – präzise Darstellungen von Algorithmen in speziellen, von Computern lesbaren Sprachen – als auf die Algorithmen selbst. Bevor wir aber in Kapitel 3 Programmiersprachen diskutieren, ist diese Unterscheidung völlig unerheblich.

Algorithmen begegnen uns überall. Viele alltägliche Prozesse werden durch Algorithmen gesteuert: das Wechseln eines platten Reifens, die Montage eines Schrankes, das Stricken eines Pullovers, die Division von Zahlen, das Nachschlagen einer Telefonnummer, das Aktualisieren einer Spesenliste oder das Ausfüllen einer Einkommensteuererklärung. Einige davon (beispielsweise die Division) sind in unserer Vorstellung stärker mit Computern verknüpft als andere (beispielsweise die Montage eines Schrankes), aber das ist für uns weniger von Belang. Obwohl Computer für das Thema dieses Buches fundamental sind, werden wir uns keinesfalls auf deren physikalische Erscheinung konzentrieren, außer indirekt in Teilen der Kapitel 3 und 9. Es ist ihr Geist, an dem wir interessiert sind; an den Rezepten, die sie in Gang setzen – also an ihren Algorithmen.

---

## Algorithmik versus Informatik

Die Algorithmik ist mehr als ein Zweig der Informatik. Sie ist der Kern der Informatik und, in aller Bescheidenheit, für die meisten Naturwissenschaften, Wirtschaftswissenschaften und Ingenieurwissenschaften relevant. Die eigentliche Natur der Algorithmik macht sie insbesondere für solche Disziplinen anwendbar, die vom Computereinsatz profitieren, und diese entwickeln sich schnell zu einer überwältigenden Mehrheit.

Immer wieder fragen sich Leute: „Was *ist* Informatik eigentlich? Weshalb gibt es keine U-Boot-, Geschirrspüler- oder Telefonwissenschaften?“<sup>61</sup> Telefone und Geschirrspüler, könnte man argumentieren, sind für das moderne Leben genauso wichtig wie Computer, möglicherweise noch wichtiger. Eine etwas präzisere Frage ist, ob die Informatik solchen klassischen Disziplinen zugeordnet werden kann wie der Mathematik, der Physik, den Neurowissenschaften, der Elektrotechnik, den Sprachwissenschaften, der Logik und der Philosophie.

Dieses Buch versucht nicht, diese Fragen zu beantworten. Es bleibt jedoch zu hoffen, dass es implizit etwas von der Einzigartigkeit und der Universalität der Algorithmik vermitteln kann, und somit auch die Bedeutung der Informatik als autonomes – wenn auch junges – Forschungsgebiet. Weil Computer die Allgemeingültigkeit

---

<sup>61</sup>Die hier geführte Diskussion hat ihren Ursprung in dem englischen Wort *computer science* – Computerwissenschaft. Dieses Fachgebiet wird im Deutschen als Informatik bezeichnet, was die Argumentation im englischen Original bereits vorwegnimmt. *Anm. der Übers.*

der Algorithmik möglicherweise einschränken könnten, empfinden einige Leute die unvermeidbare Verknüpfung zwischen beiden als unglücklich. Jemand sagte einmal: Das Fachgebiet „Computerwissenschaft“ zu nennen, ist tatsächlich so, wie wenn man die Chirurgie als „Messerwissenschaft“ bezeichnet. Wie dem auch sei, es ist klar, dass sich ohne diese Verknüpfung die Algorithmik niemals so entwickelt hätte, wie sie es bisher getan hat. Es besteht jedoch Einigkeit darüber, dass die Bezeichnung „Computerwissenschaft“ irreführend ist, und so ein Begriff wie „Informationswissenschaft“, „Prozesswissenschaft“ oder „Wissenschaft des Diskreten“ vielleicht besser wäre. Wir behaupten nur, dass unser Thema, die Algorithmik, eine Grundlage der Informatik bildet, sie jedoch nicht ersetzt.

Aus einem Teil der später diskutierten Themen, wie der Existenz von Problemen, die Computer *nicht* lösen können, ergeben sich philosophische Folgerungen, nicht nur für die Grenzen der wundervollen Maschinen, die wir konstruieren können, sondern auch für unsere Grenzen als sterbliche Wesen mit endlicher Zahl und endlicher Lebenszeit. Ungeachtet des tiefgründigen Charakters solcher Folgerungen, liegt der Schwerpunkt dieses Buches in dem pragmatischeren Ziel, ein tieferes Verständnis der Grundlagen der von Maschinen ausführbaren Prozesse zu erlangen und die Rezepte oder Algorithmen zu verstehen, die diese steuern.

## Einiges zur Geschichte

---

Blicken wir nun auf einige wichtige Meilensteine bei der Entwicklung der Computer und der Informatik zurück, im Wesentlichen um zu illustrieren, dass dieses Fachgebiet als reguläre wissenschaftliche Disziplin extrem jung ist.

Zwischen 400 und 300 v. u. Z. entwickelte der große griechische Mathematiker Euklid einen Algorithmus zur Bestimmung des größten gemeinsamen Teilers (ggT) zweier positiver ganzer Zahlen. Der größte gemeinsame Teiler von  $x$  und  $y$  ist die größte ganze Zahl, die sowohl  $x$  als auch  $y$  teilt. Beispielsweise ist 16 der größte gemeinsame Teiler von 80 und 32. Die Feinheiten des Algorithmus selbst sind hier nicht weiter von Belang. Wesentlich ist aber, dass der so genannte **Euklidische Algorithmus** als der erste nichttriviale Algorithmus angesehen wird, der jemals entwickelt wurde.

Das Wort *Algorithmus* ist von dem Namen des persischen Mathematikers Mohammed al-Khowârizmî abgeleitet, der im 9. Jahrhundert lebte und dem die Anerkennung für die Entwicklung der schrittweisen Verfahren zur Addition, Subtraktion, Multiplikation und Division von Dezimalzahlen gebührt. Im Lateinischen wurde dieser Name zu „Algorismus“, wovon sich *Algorithmus* herleitet. Offensichtlich waren Euklid und al-Khowârizmî Algorithmentwickler *par excellence*.

Kommen wir von der Software zur Hardware. Eine der ersten Maschinen, die einen Prozess ausführte, der durch so etwas wie einen Algorithmus gesteuert wurde, war ein 1801 von dem Franzosen Joseph-Marie Jacquard entwickelter Webstuhl. Das

Weben des Musters wurde durch Lochkarten gesteuert. Die Löcher in den Karten, die mithilfe eines speziellen Mechanismus abgetastet wurden, kontrollierten die Auswahl der Fäden und andere Aktionen der Maschine. Interessanterweise hatte dies aber nichts mit dem eng gefassten numerischen Begriff der „Berechnung“ zu tun.

Eine der wichtigsten und schillerndsten Persönlichkeiten in der Geschichte der Informatik war Charles Babbage. Nachdem dieser englische Mathematiker 1833 eine bereits konzipierte „Rechenmaschine“ zur Lösung bestimmter mathematischer Gleichungen fertiggestellt hatte, erfand und konzipierte er eine bemerkenswerte Maschine, die er „Analytische Maschine“ nannte. Im Gegensatz zur Rechenmaschine, die zum Ausführen einer bestimmten Aufgabe konstruiert war, sollte die analytische Maschine fähig sein, Algorithmen oder Programme auszuführen, die vom Benutzer durch Lochkarten codiert waren. Wäre die analytische Maschine gebaut worden, dann wäre sie das mathematische Gegenstück zu Jacquards Webstuhl gewesen, durch den sie tatsächlich inspiriert war. Es ist überflüssig zu sagen, dass Babbages Maschine mechanischer Natur war, also auf Hebeln, Zahnrädern und Zahnradgetrieben beruhte, und nicht auf Elektronik und Silizium. Trotzdem bilden die Ideen, die mit Babbages Entwurf der analytischen Maschine verknüpft sind, die Grundlage der inneren Struktur und Wirkungsweise heutiger Computer. Generell wird Babbage als jemand betrachtet, der seiner Zeit weit voraus war und dessen Ideen erst viel später anerkannt wurden.

Ada Byron, Gräfin von Lovelace, war Babbages Programmiererin. Sie ist eine der interessantesten Persönlichkeiten der Geschichte des wissenschaftlichen Rechnens. Ihr wird zugeschrieben, die Grundlagen der Programmierung gelegt zu haben, mehr als hundert Jahre bevor der erste arbeitstüchtige Computer verfügbar war.

Ein amerikanischer Ingenieur namens Herman Hollerith entwickelte eine ebenfalls mit Lochkarten arbeitende Maschine, die vom amerikanischen Bundesbüro zur Durchführung von Volkszählungen bei der Volkszählung im Jahre 1890 eingesetzt wurde. Der erste Mehrzweckcomputer wurde jedoch erst in den 1940er Jahren gebaut, teils als Antwort auf die rechentechnischen Bedürfnisse von Physikern und Astronomen, teils als natürliche Nebenerscheinung der Verfügbarkeit der entsprechenden elektromechanischen und elektronischen Bausteine. Ironischerweise förderte auch der Zweite Weltkrieg mit seinen Aktivitäten auf den Gebieten der Waffenkonstruktion und der Nachrichtenentschlüsselung die Entwicklung. Einige der Schlüsselfiguren dieser kritischen und aufregenden Periode waren der Engländer Alan Turing, die Amerikaner Howard Aiken, John Mauchly, J. Presper Eckert und Herman Goldstine sowie der berühmte deutsch-amerikanische Mathematiker John von Neumann.

Kommen wir zu Software und Algorithmen zurück. Die mittleren 1930er Jahre waren Zeuge einiger der fundamentalsten Arbeiten zur Algorithmentheorie. Sie enthielten Resultate über die Leistungsfähigkeit und die Grenzen der von Maschinen ausführbaren Algorithmen. Es ist erstaunlich, dass diese Arbeiten, mit denen wir

uns teilweise später im Buch befassen werden, der tatsächlichen materiellen Umsetzung der Idee des Computers vorausgehen. Trotzdem sind sie von universeller und nachhaltiger Bedeutung. Einige der Schlüsselfiguren, übrigens alle Mathematiker, sind an dieser Stelle wieder Alan Turing, der Deutsche Kurt Gödel, der Russe Andreï A. Markov und die Amerikaner Alonzo Church, Emil Post und Stephen Kleene.

Die 1950er und 1960er Jahre erlebten weitreichende und schnelle technologische Fortschritte im Computerdesign und bei der Konstruktion. Dies kann einerseits auf den Eintritt ins Zeitalter der Kern- und Weltraumforschung und andererseits auf den Boom bei Großunternehmen und Banken sowie auf verschiedene Regierungsaktivitäten zurückgeführt werden. Die präzise Vorhersage vieler kernphysikalischer Phänomene erforderte sehr erhebliche Rechenleistungen, genau wie die Planung und Simulation von Weltraummissionen. Die Erforschung des Weltraums verlangte außerdem Fortschritte bei der computergestützten Kommunikation, die Möglichkeit zuverlässiger Analyse und Filterung und sogar die Verbesserung der von oder zu Satelliten und Raumschiffen übertragenen Daten. Die Wirtschaft, das Bankwesen und die Regierungsaktivitäten brauchten Computer, die bei der Sicherung, der Bearbeitung und der Abfrage von Informationen über eine große Anzahl von Menschen, Bestandseinträgen, steuerrechtlichen Details usw. helfen.

Ein interessanter Beleg für die Bedeutung der technologischen, maschinenorientierten Entwicklungen in dieser Periode findet sich in den Namen der weltgrößten Computerfirma IBM und einer der weltgrößten computerorientierten Berufsorganisationen, der ACM. Der erste Name wurde um 1920, der letzte in den späten 1940er Jahren geprägt. In beiden Fällen stammt das „M“ von dem Wort „Maschine“: International Business Machines und Association for Computing Machinery. (IBM entwickelte sich aus einer Firma, die 1896 von Herman Hollerith zur Produktion seiner Maschinen zur Datenerfassung gegründet wurde.)

Zur Wiederentdeckung der Informatik als unabhängige akademische Disziplin kam es Mitte der 1960er Jahre, als verschiedene Universitäten Institute für Informatik gründeten. Im Jahre 1968 gab die ACM eine weitgehend freudig begrüßte Empfehlung für einen Studienplan für Lehrveranstaltungen in Informatik heraus, der auch heute die Grundlage für die meisten Lehrveranstaltungspläne für das Vordiplomstudium bildet. Dieser Studienplan wird regelmäßig überarbeitet. Heute besitzt fast jede akademische Einrichtung ein Institut für Informatik oder eine Forschungsgruppe Informatik innerhalb ihres Instituts für Mathematik oder Elektrotechnik. In den 1960er Jahren zeigte sich ein wieder auflebendes Interesse an den Arbeiten zu Algorithmen aus den 1930er Jahren. Seitdem ist das Fachgebiet Gegenstand aufwendiger und weitreichender Forschungen.

Wir brauchen nicht weiter auf die aktuelle technologische Situation eingehen: Computer sind einfach überall. Wir benutzen sie zum Surfen im Internet, also zum Empfangen und Übergeben von Informationen, zum Lesen, zum Hören und zum Sehen und natürlich zum Durchsuchen und zum Kaufen. Es gibt Desktops, Laptops und

Palms, sodass wir niemals ohne einen Computer auskommen müssen, und die sich schnell schließende Lücke zwischen Handys und Computern kündigt das Zeitalter der tragbaren Computer an. Fast jedes moderne Gerät wird durch einen Computer kontrolliert, schon ein einziges Auto enthält zum Beispiel Duzende davon. Kinder wünschen sich und bekommen Computer zum Geburtstag, von Informatikstudenten wird an den meisten Universitäten erwartet, dass sie ihren eigenen Computer für Studienarbeiten besitzen; und es gibt keine industrielle, wissenschaftliche oder kommerzielle Tätigkeit, die nicht entscheidend durch Computer unterstützt wird.

---

## Ein seltsamer Zwiespalt

Trotz alledem (oder vielleicht als Folge dessen) ist die breite Öffentlichkeit sehr gespalten, wenn es um Computerliteratur geht. Es gibt noch immer diejenigen, die überhaupt nichts über Computer wissen, und es gibt Mitglieder der stets wachsenden Klasse von Computergebildeten. Angefangen von den 10jährigen Computerbesitzern besteht diese sich stets vergrößernde Gruppe von alltäglichen Computerbenutzern aus Managern, Ingenieuren, Bankiers, Technikern und natürlich professionellen Programmierern, Systemberatern und Angehörigen der Computerindustrie selbst.

Weshalb ist der Zwiespalt seltsam? Wir haben es mit einer Wissenschaft zu tun, über die einige Menschen überhaupt nichts wissen, während eine rasch anwachsende Zahl von Menschen offenbar alles darüber weiß! Das kommt vor. Das wirklich ungewöhnliche Phänomen ist jedoch, dass große und wichtige Teile der *Wissenschaft* des computergestützten Rechnens nicht hinreichend bekannt sind. Das trifft nicht nur auf Mitglieder der ersten sondern auch auf Mitglieder der zweiten Gruppe zu.

Eine Zielsetzung dieses Buches ist zu versuchen, einen wesentlichen Aspekt der Computerrevolution zu beleuchten, indem einige der fundamentalen Konzepte, Ergebnisse und Trends vorgestellt werden, die der *Wissenschaft* des Rechnens zugrunde liegen. Es ist sowohl an Neulinge als auch an Experten gerichtet. Ein Leser ohne Vorkenntnisse wird hier (hoffentlich) etwas über den „Geist“ der Computer lernen und über die Denkweise, auf deren Grundlage Computer funktionieren, während anderswo nach Material gesucht wird, das sich eher mit ihrem „Fleisch“ befasst. Der computererfahrene Leser, dem die ersten Kapitel etwas zu ausführlich erscheinen mögen, wird aus den später folgenden Kapiteln (hoffentlich) viel lernen können.

---

## Einige Grenzen von Computern

Bevor wir uns auf die Reise begeben, wollen wir dem Inhalt des Eröffnungsparagrafen dieses Kapitels einige intellektuelle Leistungen gegenüberstellen, zu denen heutige Computer bislang unfähig sind. Auf diese Gegenbeispiele werden wir im letzten Kapitel des Buches zurückkommen, das sich mit der Beziehung zwischen Computern und menschlicher Intelligenz beschäftigt.

Heutige Computer sind zur Sofortanalyse enormer Datenmengen in der Lage, die sich aus vielen Röntgenbildern des Gehirns eines Patienten ergeben, die bei der Untersuchung aus schrittweise wachsenden Sichtwinkeln aufgenommen wurden. Die analysierten Daten werden anschließend vom Computer zum Erzeugen von Schnittbildern des Gehirns verwendet, die Informationen über die Gewebestruktur des Gehirns liefern, sodass Unregelmäßigkeiten wie Tumore oder Überschussflüssigkeiten präzise lokalisiert werden können. Bemerkenswerter Weise kann im Gegensatz dazu kein heute verfügbarer Computer ein einzelnes, gewöhnliches Bild vom Gesicht genau desselben Patienten analysieren und das Alter des Patienten mit einer Fehlerspanne von, sagen wir, fünf Jahren bestimmen. Die meisten 12jährigen Kinder können das aber! Noch bemerkenswerter ist die Fähigkeit eines einjährigen Babys, das Gesicht seiner Mutter auf einem Foto zu erkennen, das es zuvor niemals gesehen hat. Das ist eine Leistung, die Computer nicht ansatzweise nachahmen können (und das liegt nicht bloß daran, dass sie keine Mütter haben ...).

Computer sind fähig, so genau und effizient wie nur irgend vorstellbar zu kontrollieren und zu steuern. Hochentwickelte Industrieroboter werden eingesetzt, um komplexe Maschinenanlagen zu bauen, die aus Hunderten von Komponenten bestehen. Im Gegensatz dazu sind die gegenwärtig hochentwickeltesten Computer nicht dazu in der Lage, einen Roboter so zu steuern, dass er ein Vogelnest aus einem Haufen dünner Zweige bauen kann. Das ist eine Leistung, die jeder 12 Monate alte Vogel ausführen kann.

Heutige Computer können Schach auf dem Level von internationalen Großmeistern spielen, und somit die große Mehrheit der menschlichen Gegner schlagen. Werden jedoch die Spielregeln nur leicht geändert (indem beispielsweise ein Springer zwei Züge auf einmal machen darf oder der Zug mit der Königin auf fünf Felder beschränkt ist), ist der beste Computer nicht zu einer Anpassung fähig, ohne dass er durch Menschen neu programmiert oder konstruiert wird. Im Gegensatz dazu wird ein 12jähriger Amateurspieler in sehr kurzer Zeit mit den neuen Regeln ein halbwegs gutes Spiel absolvieren können und mit seiner Erfahrung immer besser werden.

Wie bereits erwähnt, hängt die Unähnlichkeit mit den Unterschieden zwischen menschlicher und computerbasierter Intelligenz zusammen. Wir werden diesen Sachverhalt später in Kapitel 15 besser diskutieren können, nachdem wir mehr über Algorithmen und ihre Eigenschaften gelernt haben.

## Ein Rezept

---

Es folgt ein Rezept für Mousse au Chocolat aus Sinclair und Malinowski, *French Cooking* (Weathervane Books, 1978, S. 73). Die Zutaten – also die Eingaben des Rezeptes – bestehen aus 250 g Halbbitterschokolade, 2 Teelöffeln Wasser, einer viertel Tasse Puderzucker, 6 getrennten Eiern usw. Die Ausgabe wird als sechs bis

acht Portionen leckerer *Mousse au Chocolat* beschrieben. Hier ist das Rezept oder der Algorithmus für die Zubereitung.

*Die Schokolade mit 2 Teelöffeln Wasser im heißen Wasserbad schmelzen. Nach dem Schmelzen den Puderzucker unterrühren und die Butter stückweise hinzufügen. Die Mischung beiseite stellen. Die Eigelbe solange schlagen, bis sie dicklich und zitronenfarben sind (etwa 5 Minuten) und vorsichtig unter die Schokoladenmasse ziehen. Falls nötig, zum Schmelzen der Schokolade nochmals leicht erhitzen. Rum und Vanille einrühren. Eiweiße schaumig schlagen, dann mit zwei Teelöffeln Zucker steif schlagen und vorsichtig unter die Schokoladen-Eigelb-Masse ziehen. In einzelne Schälchen füllen und mindestens 4 Stunden kühl stellen. Je nach Wunsch mit Schlagsahne servieren. Ergibt 6 bis 8 Portionen.*

Dies ist die zur Zubereitung von *Mousse au Chocolat* nötige „Software“; dies ist der Algorithmus, der den Prozess der Zubereitung der Mousse aus den Zutaten steuert. Der Prozess selbst wird von der „Hardware“ ausgeführt, in diesem Fall von der Person, die *Mousse au Chocolat* zubereitet, mit den verschiedenen Utensilien: Topf, Herdplatte, Rührgerät, Löffel, Kurzzeituhr usw.

---

## Detailierungsgrade

Sehen wir uns die elementarsten Anweisungen in diesem Rezept etwas genauer an. Betrachten wir die Anweisung „Puderzucker unterrühren“. Weshalb heißt es im Rezept nicht „ein bisschen Puderzucker nehmen, ihn in die geschmolzene Schokolade schütten, unterrühren; etwas mehr nehmen, schütten, unterrühren ...“? Oder warum heißt es nicht etwas genauer „2365 Körnchen Puderzucker nehmen, diese in die geschmolzene Schokolade schütten, einen Löffel nehmen und den Zucker unter kreisenden Bewegungen einrühren ...“? Oder um noch genauer zu sein, warum nicht „den Arm in einem Winkel von  $14^\circ$  auf die Zutaten zubewegen, mit einer Geschwindigkeit von ungefähr 45 Zentimeter pro Sekunde ...“? Die Antwort liegt auf der Hand. Die Hardware weiß, wie man Puderzucker in geschmolzene Schokolade einrührt, und braucht keine weiteren Details. Wie wäre es, wenn wir die Sache umdrehen und fragen, ob die Hardware möglicherweise weiß, wie man gezuckerte und gebutterte Schokoladenmasse herstellt? In diesem Fall könnte man den gesamten ersten Teil des Rezeptes durch die simple Anweisung „die Schokoladenmasse zubereiten“ ersetzen. Im Extremfall weiß die Hardware vielleicht sogar, wie man *Mousse au Chocolat* zubereitet. Damit wäre es möglich, das gesamte Rezept durch die Anweisung zu ersetzen „*Mousse au Chocolat* zubereiten“. Wenn uns Hardware mit einem solchen Erfahrungsschatz zur Verfügung steht, dann ist eine einzige Anweisungszeile ein perfektes Rezept zur Herstellung von *Mousse au Chocolat*. Dieses

kurze Rezept ist klar, es enthält keine Fehler und garantiert, dass die geforderte Ausgabe wie gewünscht hergestellt wird.

Solche Gedankenexperimente verdeutlichen, dass der Detaillierungsgrad sehr wichtig ist, wenn es um die elementaren Anweisungen eines Algorithmus geht. Er muss auf die speziellen Fähigkeiten der Hardware zugeschnitten sein und ebenso dem Verständnisgrad eines potentiellen Lesers oder des Algorithmusbenutzers entsprechen.

Betrachten wir ein anderes Beispiel, das uns bereits in unserer Kindheit begegnet ist und das etwas näher am Rechnen mit dem Computer liegt – die schriftliche Multiplikation von Zahlen. Angenommen, wir sollen 528 mit 46 multiplizieren. Wir wissen genau, was zu tun ist. Wir multiplizieren 8 mit 6, mit dem Ergebnis 48, und schreiben die Einerstelle auf, also 8, und merken uns die Zehnerstelle 4. Dann multiplizieren wir 2 mit 6 und addieren die gemerkte 4. Das Ergebnis ist 16. Wir schreiben die Einerstelle 6 links von der bereits notierten 8 und merken uns die Zehnerstelle 1 usw.

Hier können wir die gleichen Fragen stellen wie vorhin. Weshalb „8 mit 6 multiplizieren“ und nicht „den Eintrag in der achten Reihe und der sechsten Spalte einer Multiplikationstabelle nachsehen“ oder „6 achtmal zu sich selbst addieren“? Warum können wir nicht das gesamte Problem in einem Schlag durch den einfachen und zufriedenstellenden Algorithmus „multipliziere die beiden Zahlen“ lösen? Diese letzte Frage ist ziemlich subtil: Weshalb dürfen wir 8 mit 6 direkt multiplizieren, nicht aber 528 mit 46? Wiederum ist offensichtlich, dass der Detaillierungsgrad ein ausschlaggebendes Merkmal für unsere Akzeptanz des Multiplikationsalgorithmus ist. Wir nehmen an, dass die entsprechende Hardware (in diesem Fall wir selbst) fähig ist, 8 mal 6 direkt auszurechnen, nicht aber 528 mal 46. Wir nehmen an, dass wir die erste Multiplikation in unserem Kopf ausführen können oder wir zumindest eine andere Möglichkeit kennen, diese Multiplikation auszuführen, sodass uns niemand sagen muss, wie man das Ergebnis in einer Tabelle nachschlägt.

Diese Beispiele verdeutlichen die Notwendigkeit, sich gleich zu Beginn auf die elementaren Aktionen zu einigen, auf die ein Algorithmus zurückgreifen kann. Ohne dies gibt es keinen Ansatzpunkt, wenn man versucht, auch nur für irgendetwas einen Algorithmus zu finden. Darüber hinaus sind unterschiedliche Probleme auf natürliche Weise mit unterschiedlichen elementaren Aktionen verknüpft. In Backrezepten muss gerührt, vermischt, geschüttet oder erhitzt werden. Bei der Multiplikation von Zahlen muss man addieren, Ziffern miteinander multiplizieren und sich, was sehr wesentlich ist, Ziffern merken können. Das Nachschlagen einer Telefonnummer erfordert es, Seiten umzuschlagen, den Finger entlang einer Liste zu bewegen und den gesuchten Namen mit dem Namen zu vergleichen, auf den der Finger gerade zeigt.

Bei der hier diskutierten Sorte von Algorithmen müssen diese elementaren Anweisungen klar und präzise festgelegt werden. Solche Ausdrücke wie „die Eiweiße schaumig schlagen“ sind dabei nicht akzeptabel, denn die Vorstellung von schaumig

kann von Person zu Person ganz verschieden sein. Anweisungen müssen angemessen von Nicht-Anweisungen, wie „ergibt 6 bis 8 Portionen“, unterscheidbar sein. Unscharfe Sätze, wie „etwa 5 Minuten“, haben in Computeralgorithmen nichts zu suchen. Das gleiche trifft auf Mehrdeutigkeiten zu, wie „Je nach Wunsch, mit Schlag-Sahne servieren.“ Im Gegensatz zu den Algorithmen, für die wir uns hier interessieren, setzen Rezepte für Schokoladencreme zu viele Dinge als selbstverständlich voraus. Der wesentlichste Punkt ist die Tatsache, dass dabei ein Mensch Teil der Hardware ist. Weil wir uns nicht auf einen solchen Luxus verlassen können, müssen wir weitaus genauer sein. Die Gesamtqualität eines Algorithmus hängt entscheidend von der Auswahl der zugelassenen elementaren Aktionen ab und von deren Eignung für das vorliegende Problem.

---

## Abstraktion

Vorhin wurde behauptet, dass reale Computer nur extrem einfache Operationen auf extrem einfachen Objekten ausführen können. Dies mag im Widerspruch zu der eben geführten Diskussion stehen, dass verschiedene Algorithmen mithilfe von elementaren Aktionen auf unterschiedlichen Detaillierungsgraden entworfen werden können. Doch die Rezept-Analogie ist weiterhin angebracht. Es kann sein, dass man einem Kochlehrling das Rezept für Mousse au Chocolat geben muss, aber nach einigen Jahren Erfahrung wird die Anweisung „Mousse au Chocolat zubereiten“ ausreichen. Man kann sagen, dass Begriffe wie „Mousse au Chocolat“, „Zitronenbaiser“ und „Bavariacreme“ auf einer höheren Abstraktionsebene liegen als die Aktionen „mixen“, „untermischen“ und „schütten“, die in den Rezepten bei der Herstellung zum Einsatz kommen. In gleicher Weise kann ein Computer bei entsprechender Programmierung dazu befähigt werden, Abstraktionen auf höherer Ebene, wie Zahlen, Texte und Bilder, zu erkennen.

Wie beim Kochen gibt es auch beim Computer viele Abstraktionsebenen. Jede davon ist zur Beschreibung einer anderen Art von Algorithmen geeignet. Derselbe Computer wird von einem 12jährigen Computerspieler auf einer anderen Ebene betrachtet als von seiner Schwester, die im Internet surft, anders als von seinem Vater, der ein Tabellenkalkulationsprogramm benutzt, um die Durchschnittsnoten seiner Studenten zu berechnen, und anders als von seiner Mutter, die ein Programm zur Durchführung einer Wirksamkeitsstudie für einen Impfstoff schreibt. Keiner von ihnen weiß etwas über die Bits, die den gerade laufenden Rechenprozess ausmachen, oder kümmert sich überhaupt darum.

Dieser Abstraktionsprozess, bei dem Details außer acht gelassen werden, um allgemeine Muster zu erkennen, ist der Kern fast jedes menschlichen Vorhabens. Das Lesen dieses Buches wirkt sich beispielsweise auf ihr Gehirn aus, das aus unterschiedlichen Regionen besteht, die alle aus Neuronen und anderen Zellen zusammengesetzt sind. Diese Zellen bestehen aus komplexen Molekülen, die aus Atomen aufgebaut

sind, die sich wiederum aus noch elementareren Teilchen zusammensetzen. All diese unterschiedlichen Abstraktionsebenen sind relevant dafür, was in ihrem Gehirn passiert, aber sie können nicht alle zugleich betrachtet werden. In Wirklichkeit sind sie Gegenstand unterschiedlicher Forschungsgebiete: Teilchenphysik, Chemie, Molekularbiologie, Neurobiologie und Psychologie. Es würde einen Psychologen, der Experimente zum Kurzzeitgedächtnis durchführt, nur ablenken, wenn er über die Beziehung zwischen Atomen und Molekülen im Gehirn nachdenken würde.

Das Gleiche gilt für die Informatik. Wenn wir gezwungen wären, die ganze Zeit auf Bitebene zu denken, wäre der Computer kaum nützlich. Stattdessen können wir beispielsweise eine Bitgruppe (typischerweise acht Bit oder ein „Byte“) als Bezeichnung für einen Buchstaben ansehen. Nun können wir uns Bytefolgen als Wörter vorstellen, Wortfolgen mit Interpunktion als Sätze. Dies kann fortgeführt werden bis hin zu Paragraphen, Kapiteln und Büchern. Es gibt auf jeder dieser Ebenen entsprechende Algorithmen. Beispielsweise wendet man die Rechtschreibprüfung auf Wörter und nicht auf Buchstaben an, die linksbündige Ausrichtung gilt für Paragraphen und nicht für Buchstaben und ein Inhaltsverzeichnis wird für Bücher erstellt. In jedem Fall können wir den Algorithmus dafür beschreiben, während wir die Bits vollständig ignorieren, aus denen die Wörter, die Paragraphen oder das gesamte Buch bestehen. Im Fortgang des Buches, und insbesondere in den Kapiteln 3 und 9, werden wir die technischen Hilfsmittel diskutieren, die solche Abstraktionen ermöglichen. Inzwischen werden wir jeden Algorithmus auf der ihm angemessenen Abstraktionsebene beschreiben.

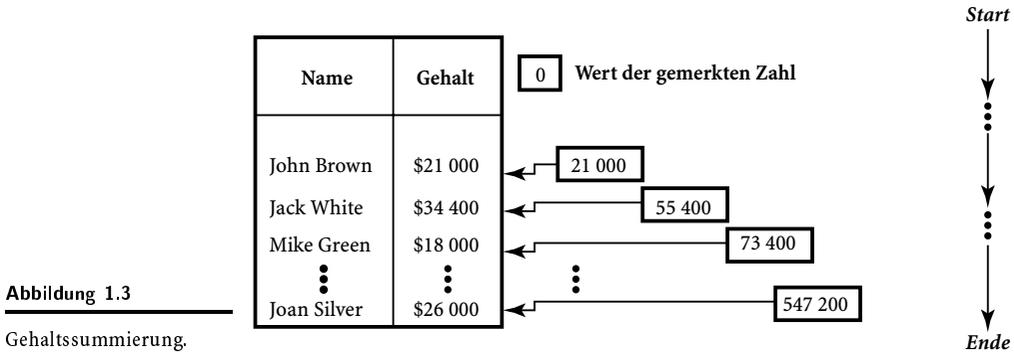
## Kurze Algorithmen für lange Prozesse

---

Angenommen, uns ist eine Liste mit Personaldaten gegeben, in der ein Eintrag für jeden Angestellten eines bestimmten Unternehmens jeweils den Namen des Angestellten, personengebundene Details und das Gehalt enthält. Wir sind an der Gesamtsumme der Gehälter aller Angestellten interessiert. Ein möglicher Algorithmus sieht so aus:

- (1) „Merke dir“ die Zahl 0;
- (2) gehe die Liste durch, addiere jeweils das Gehalt des Angestellten zur gemerkten Zahl;
- (3) gib die gemerkte Zahl aus, wenn das Ende der Liste erreicht ist.

Bevor wir weiter gehen, sollten wir uns zunächst davon überzeugen, dass dieser einfache Algorithmus die Aufgabe erfüllt. Die „gemerkte“ Zahl, die wir uns auf einem Blatt Papier gespeichert oder aufgeschrieben vorstellen können, besitzt den Startwert 0. Nachdem die Addition für den ersten Angestellten in Klausel (2) ausgeführt wurde, nimmt diese Zahl tatsächlich den Wert des Gehaltes dieses Angestellten an. Nach dem zweiten Angestellten, ist der Wert der gemerkten Zahl die Summe aus den



Gehältern der ersten beiden Angestellten. Am Ende hat sie offensichtlich den Wert der Summe der Gehälter aller Angestellten (siehe Abbildung 1.3).

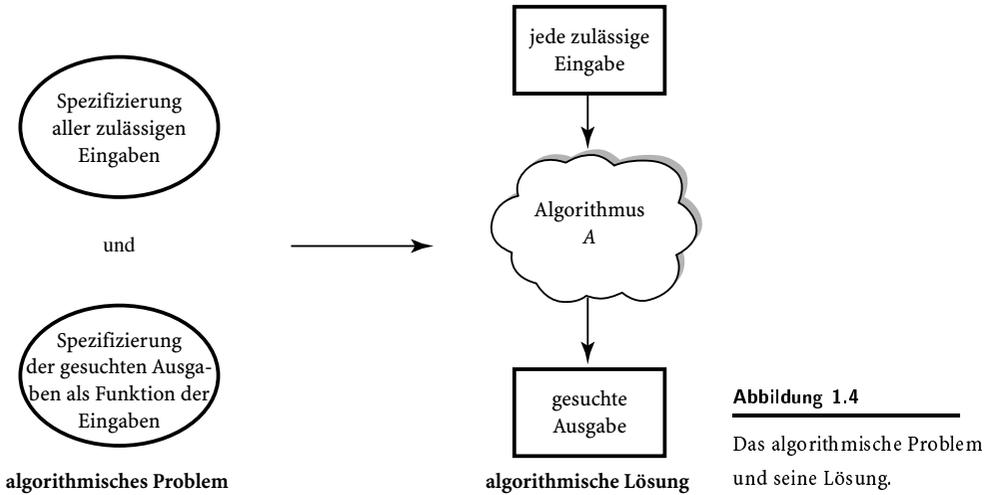
Interessanterweise ist der *Wortlaut* dieses Algorithmus kurz und von fester Länge, während der durch ihn beschriebene und kontrollierte *Prozess* von der Länge der Angestelltenliste abhängt und deshalb sehr, sehr lange dauern kann. Zwei Unternehmen, das erste mit einem und das zweite mit einer Million Angestellten, können beide ihre Angestelltenlisten in denselben Algorithmus eingeben, und das Gehaltssummierungsproblem wird für beide gleich gut gelöst. Natürlich ist der Prozess für das erste Unternehmen kurz, während er sich für das zweite Unternehmen ziemlich in die Länge zieht. Der Algorithmus ist jedoch fest.

Aber nicht nur der Wortlaut des Algorithmus ist kurz und von fester Länge, sondern auch die Anzahl der „Utensilien“ ist hier klein und fest, denn sowohl das kleine als auch das große Unternehmen benutzen während des Prozesses nur eine gemerkte Zahl. Natürlich wird der potentielle *Wert* der gemerkten Zahl für größere Unternehmen höher sein, aber immer muss nur eine Zahl gemerkt werden.

## Das algorithmische Problem

Wir haben nun einen festen Algorithmus, der viele Prozesse variabler Länge beschreibt, wobei die genaue Dauer und die Natur des Prozesses von den Eingaben des Algorithmus abhängen. Selbst das einfache Beispiel des Gehaltssummierungsproblems besitzt eine Vielzahl möglicher Eingaben: Einmannunternehmen, Unternehmen mit einer Million Angestellten, Unternehmen, bei denen einige der Gehälter 0 Euro betragen, oder solche, bei denen alle Gehälter gleich sind. Manchmal muss ein Algorithmus auch mit bizarren Eingaben umgehen können, wie bei Unternehmen ohne jegliche Angestellte oder mit Angestellten, die ein negatives Gehalt beziehen (was bedeuten würde, dass die Angestellten für das Vergnügen der Arbeit bezahlen).

Tatsächlich können wir davon ausgehen, dass der Gehaltsalgorithmus für eine *unendliche* Anzahl von Eingaben vernünftig arbeitet. Es gibt eine unendliche Zahl



zulässiger Angestelltenlisten. Der Algorithmus sollte für jede von ihnen die Gehaltssumme berechnen können.

Die Frage der unendlich vielen potentiellen Eingaben scheint nicht recht in die Rezept-Analogie zu passen. Auch wenn ein Rezept unabhängig von der Häufigkeit seiner Anwendung perfekt funktionieren sollte, so sind die Mengen der Zutaten gewöhnlich fest vorgeschrieben. Somit besitzt das Rezept im Wesentlichen nur eine potentielle Eingabe (zumindest auf der Ebene der Mengen; natürlich werden die Moleküle und Atome jedes Mal verschieden sein). Sicher könnte man das Rezept für Mousse au Chocolat generisch folgendermaßen formulieren: „X Schokoladenstücke zu etwa 30g, X/4 Teelöffel Wasser, X/32 Tassen Puderzucker usw.“ und die letzte Zeile könnte lauten: „Ergibt 3X/4 bis X Portionen.“ Das würde besser zu dem eigentlichen Begriff eines Algorithmus passen. In seiner vorliegenden Form ist das Rezept ein Algorithmus etwas trivialer Natur, da es auf eine spezielle Zutatenliste zugeschnitten ist. Es kann mehrmals verwendet werden (oder, in algorithmischer Terminologie, mehrmals **laufen gelassen** oder **ausgeführt werden**), aber im Wesentlichen mit der gleichen Eingabe, da eine Tasse Mehl genauso gut wie eine andere ist.

Die Eingabe selbst muss für den Zweck des Algorithmus **zulässig** sein. Dies bedeutet, dass beispielsweise die Bestsellerliste der *New York Times* als Eingabe des Algorithmus zur Gehaltssummierung nicht annehmbar ist, genau wie Erdnußbutter und Gelatine im Rezept für Mousse au Chocolat unannehmbar sind. Wir brauchen eine Art *Spezifikation* der erlaubten Eingaben. Jemand muss genau entscheiden, welche Angestelltenlisten zulässig sind und welche nicht; an welcher Stelle der Liste das Gehalt erscheint, ob die Zahl ausgeschrieben ist (beispielsweise 32000 Euro) oder vielleicht irgendwie abgekürzt (beispielsweise 32K Euro); wo der Eintrag für einen Angestellten endet und ein anderer beginnt usw.

Allgemein kann man sagen, dass Rezepte oder Algorithmen Lösungen zu einer bestimmten Art von Problemen sind, die als **rechnerische** oder **algorithmische Probleme** bezeichnet werden. Beim Beispiel der Gehaltssummierung kann das Problem folgendermaßen spezifiziert werden: Gesucht ist eine Zahl, die die Summe der Gehälter auf der Angestelltenliste einer Organisation angibt. Diese Liste kann in ihrer Länge variieren, muss aber in einer besonderen Weise organisiert sein. Ein solches Problem kann als Suche nach dem Inhalt einer „Blackbox“ angesehen werden, die durch eine präzise Definition der zulässigen Eingaben und eine präzise Definition der gesuchten Ausgaben als Funktion dieser Eingaben spezifiziert ist; damit ist die Art und Weise gemeint, in der jede Ausgabe von der Eingabe abhängt (siehe Abbildung 1.4). Ein algorithmisches Problem wurde gelöst, wenn ein dazugehöriger Algorithmus gefunden wurde. Die Blackbox wurde tatsächlich mit Inhalt gefüllt; sie „arbeitet“ nach diesem Algorithmus. Mit anderen Worten, die Blackbox kann aus jeder zulässigen Eingabe eine entsprechende Ausgabe erzeugen, indem sie den durch diesen Algorithmus beschriebenen und gesteuerten Prozess ausführt. Das Wort „jede“ ist im vorangegangenen Satz sehr wichtig. Wir sind nicht an Lösungen interessiert, die nicht für alle spezifizierten Eingaben funktionieren. Eine Lösung, die nur für einige zulässige Eingaben richtig arbeitet, erhält man leicht. Ein extremes Beispiel ist der triviale Algorithmus:

- (1) Gib 0 aus.

Dieser Algorithmus funktioniert hervorragend für mehrere interessante Angestelltenlisten: Listen ohne jeden Angestellten, Listen von Angestellten, die alle 0 Euro (oder ein Vielfaches davon) verdienen, sowie Listen, die eine perfekte Balance zwischen positiven und negativen Gehältern aufweisen.

Später werden wir diese Dinge als Effizienz und Handhabbarkeit von Algorithmen bezeichnen. Hier stellen wir die minimale Forderung, dass ein Algorithmus tatsächlich ein Problem löst, auch wenn dies ineffizient geschieht. Natürlich kann ein Problem selbstspezifizieren, wie sich ein potentieller Algorithmus bei unerwünschten Eingaben verhalten soll. Dann sind aber diese Eingaben, obwohl unerwünscht, noch immer zulässig. Es wäre beispielsweise vorstellbar, dass das Gehaltssummierungsproblem die Forderung enthält, dass der Algorithmus den Namen eines Angestellten einer speziellen Liste hinzufügt, wenn der Eintrag für den Angestellten keine Zahl sondern ein Fragezeichen oder andere unsinnige Angaben im Gehaltsfeld enthält. Diese spezielle Liste wird dem Lohnbüro zur weiteren Bearbeitung übermittelt. Eine solche unorthodoxe Angestelltenliste ist trotzdem zulässig; man kann mit ihr nicht in der gewöhnlichen Weise umgehen, aber es gibt eine spezielle Behandlungsweise, die ihrer unregelmäßigen Beschaffenheit entspricht. Die Aufgabe des algorithmischen Problems ist also, *unzulässige* Eingaben vom Algorithmus fernzuhalten, während der Algorithmus für die Behandlung von speziellen Klassen ungewöhnlicher oder *unerwünschter* Eingaben selbst zuständig ist.

## Einschränkungen für elementare Aktionen

---

Es gibt eine andere wichtige Frage, mit der wir uns an dieser Stelle beschäftigen müssen. Sie betrifft das Ausführen der elementaren Aktionen oder Operationen, die vom Algorithmus vorgegeben sind. Offensichtlich muss jede dieser Aktionen in einer endlichen Zeitspanne ausgeführt werden, da der Algorithmus anderenfalls zu keinem Ende kommt. Deshalb sind unendlich lange dauernde Aktionen schlecht. Aktionen, die in unendlich kleinen Zeitspannen ablaufen, sind ebenfalls ausgeschlossen, eine Tatsache, die kaum einer Erklärung bedarf. Es ist undenkbar, dass eine Maschine jemals dazu in der Lage sein wird, Aktionen in verschwindend kleinen Zeitspannen auszuführen. Die Lichtgeschwindigkeit kann zum Beispiel immer als eine Grenze für die Schnelligkeit von Maschinen herangezogen werden. Ähnliche Einschränkungen müssen auch für die Ressourcen (also die Utensilien) eingeführt werden, die zum Ausführen der elementaren Aktionen benutzt werden. Die Gründe dafür werden wir hier jedoch nicht diskutieren.

Offensichtlich treffen diese Annahmen hinsichtlich der elementaren Aktionen auf reale Computer tatsächlich zu. Die elementaren Aktionen zur Bitmanipulation sind beispielsweise präzise und eindeutig, und sie haben einen beschränkten Zeit- und Ressourcenbedarf. Wie versprochen, ist die hier beschriebene Theorie der Algorithmik also direkt auf Probleme anwendbar, die für die computergestützte Lösung bestimmt sind.

## Das Problem und dessen Lösung: Zusammenfassung

---

Ein algorithmisches Problem besteht aus:

1. einer Spezifikation von möglicherweise unendlich vielen, zulässigen Eingaben,
2. einer Spezifikation der gesuchten Ausgaben als Funktion der Eingaben.

Wir gehen davon aus, dass vorab entweder eine Beschreibung der zulässigen elementaren Aktionen oder eine Hardwarekonfiguration mit ihren eingebauten elementaren Aktionen vorliegt. Eine Lösung eines algorithmischen Problems besteht aus einem Algorithmus mit elementaren Anweisungen, die Aktionen aus der vorher vereinbarten Menge beschreiben. Dieser Algorithmus löst das Problem, wenn er auf einer zulässigen Eingabemenge ausgeführt wird und die Ausgabe wie gewünscht erzeugt. Zu Beginn des 10. Kapitels werden wir diese Gedanken verallgemeinern, bis dahin reicht die vorliegende Definition aus.

Es ist wichtig, die beträchtlichen Schwierigkeiten zu erkennen, die auftreten können, wenn algorithmische Probleme zufriedenstellend gelöst werden sollen. Es war vielleicht ein Fehler, mit einem Rezept für Mousse au Chocolat zu beginnen, und danach einen einfachen Summationsalgorithmus anzugeben, denn es könnte nun den Anschein haben, die Dinge wären leicht. Nichts ist weiter von der Wahrheit

entfernt als das. Algorithmische Probleme können in der Praxis unglaublich komplex sein, und es kann Jahre dauern, sie erfolgreich zu lösen. Es kommt aber noch schlimmer. Wie wir in den späteren Kapiteln sehen werden, lassen viele Probleme einfach keine zufriedenstellenden Lösungen zu, während es für andere überhaupt keine Lösungen gibt. Obwohl eine Menge talentierter Leute beträchtliche Arbeit leisten, ist der Status vieler Probleme im Hinblick auf gute algorithmische Lösungen noch unbekannt.

Natürlich können wir die in diesem Buch behandelten Sachverhalte nicht mit übermäßig langen und komplexen Beispielen illustrieren. Wir können aber ein Gefühl für die Schwierigkeiten beim Algorithmenentwurf bekommen, indem wir über die folgenden (formlos beschriebenen) algorithmischen Probleme nachdenken. Bei dem ersten Problem besteht die Eingabe aus einer zulässigen Schachposition (also aus der Beschreibung der Situation zu einem Zeitpunkt während des Schachspiels). Die Ausgabe ist der beste Zug für Weiß (also die Beschreibung eines Zuges, der die Gewinnchancen für Weiß im Spiel maximiert). Das zweite Problem beschäftigt sich mit der Verteilung von Zeitungen. Angenommen, 20000 Zeitungen müssen auf 1000 Verkaufsstellen in 100 Städten mithilfe von 50 Lastwagen verteilt werden. Die Eingabe besteht aus den Entfernungen zwischen den Städten und zwischen den Verkaufsstellen jeder Stadt, den an jeder Verkaufsstelle benötigten Zeitungen, der gegenwärtigen Position jedes Lastwagens, dem Fassungsvermögen jedes Lastwagens sowie der Größe des Tanks, dem Benzinverbrauch und Details über die verfügbaren Fahrer einschließlich ihres gegenwärtigen Aufenthaltsortes. Die Ausgabe sollte aus einer Liste bestehen, die Fahrer und Bestimmungsorte in Form einer detaillierten Reiseroute so auf die Lastwagen verteilt, dass die Summe der insgesamt gefahrenen Kilometer minimal ist. Tatsächlich verlangt das Problem nach einem Algorithmus, der für jede Anzahl von Zeitungen, Verkaufsstellen, Städten und Lastwagen funktioniert, sodass diese Zahlen zu Variablen und somit selbst zum Teil der Eingabe werden.

Bevor wir Begriffe wie die Korrektheit und die Effizienz von Algorithmen diskutieren oder tiefergehende Fragen behandeln können, die sich mit der Natur oder der tatsächlichen Existenz von Lösungen bestimmter algorithmischer Probleme beschäftigen, müssen wir mehr über die Struktur von Algorithmen und die Struktur der von ihnen manipulierten Objekte lernen.

*Das Frühere, längst hab' ich's kundgetan.*

JESAJA 48:3





<http://www.springer.com/978-3-540-24342-7>

Algorithmik

Die Kunst des Rechnens

Harel, D.; Feldman, Y.

2006, XVIII, 608 S., Hardcover

ISBN: 978-3-540-24342-7