



# Fit fürs Studium

# Informatik

Boockmeyer · Fischbeck · Neubert



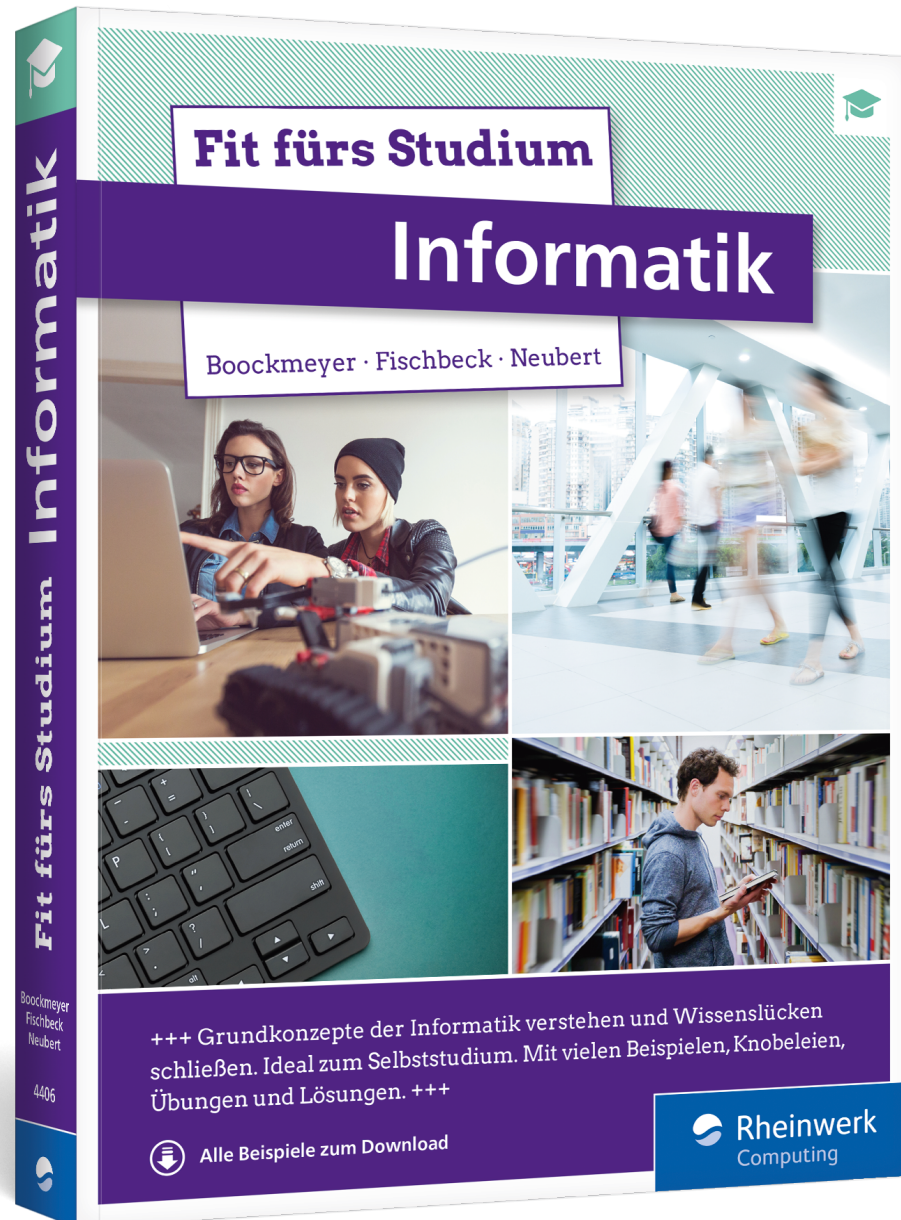
+++ Grundkonzepte der Informatik verstehen und Wissenslücken schließen. Ideal zum Selbststudium. Mit vielen Beispielen, Knobeleyen, Übungen und Lösungen. +++



Alle Beispiele zum Download









Rheinwerk  
Computing



## Leseprobe

Informatik – das Studium ist spannend und der Abschluss ist gefragt, aber gerade der Beginn hält so manche Herausforderung bereit. Dieses Buch hilft Ihnen, den Einstieg zu meistern und Ihr Wissen aufzufrischen – unterhaltsame Knobeleyen inklusive! Schauen Sie in diese Leseprobe herein und erfahren Sie, wie Sie Daten mithilfe von Graphen in Beziehung zueinander setzen.

-  **»Intro«**
-  **»Graphen – Daten in Beziehung setzen«**
-  **Inhaltsverzeichnis**
-  **Index**
-  **Die Autoren**
-  **Leseprobe weiterempfehlen**

Arne Boockmeyer, Philipp Fischbeck, Stefan Neubert

### Fit fürs Studium – Informatik

411 Seiten, broschiert, Mai 2017  
24,90 Euro, ISBN 978-3-8362-4406-0

 [www.rheinwerk-verlag.de/4273](http://www.rheinwerk-verlag.de/4273)



## Geleitwort

Informationen sind ein wertvoller Rohstoff. Wer mehr oder genauere Informationen hat als andere, ist im Vorteil. Das ist schon sehr lange so, wie insbesondere die Geschichte der Verschlüsselung zeigt, in der es seit altersher darum geht, den eigenen Informationsvorteil nicht zu verlieren. Dieser Vorteil kann ausgebaut werden, wenn vorhandene Informationen verarbeitet und weitere Informationen daraus gewonnen werden können. Auch das ist nicht neu: Schlussfolgerungskalküle wurden u. a. bereits von Aristoteles entwickelt. Leibniz beschäftigte sich mit der arithmetischen Modellierung von Logik, entwickelte das Dualsystem und zeigte auf, wie »mechanisch« man mit dualen Zahlen rechnen kann. Aber erst Mitte des 20. Jahrhunderts wurde diese Mechanik technisch umgesetzt – und damit begann die Stunde der Informatik zu schlagen. Nach nur gut 70 weiteren Jahren ist die Automatisierung der Informationsverarbeitung, zu der die Informatik den zentralen Beitrag leistet, so weit fortgeschritten, dass die Menschen beginnen, Informatik-Systemen mehr zu vertrauen als sich selbst, ob bei der Kontrolle der individuellen Fortbewegung oder der Anbahnung zwischenmenschlicher Beziehungen.

Das Potenzial der Informatik und ihrer Anwendungen kann begeistern; alles scheint möglich. Außerdem werden mit Informatik-Anwendungen Milliarden verdient; die Plätze 1, 3, 5, 7, 12 und 13 der vom Magazin »Forbes« Anfang 2017 veröffentlichten Liste der Superreichen werden von Menschen belegt, deren Vermögen auf Informatik-Systemen beruht. Und diese System haben in vielen Fällen – wie Amazon, Facebook oder Google – die Welt verändert. Die richtige Idee für eine neue Informatik-Anwendung kann so schnell reich machen wie einst der Fund einer Ölquelle oder einer Goldader. Ob inhaltliche Begeisterung, die Aussicht auf Weltveränderung oder Wohlstand: Es ist kein Wunder, dass junge Menschen sich von diesem Thema und diesem Fach angezogen fühlen. Ein Informatik-Studium ist attraktiv – aber eventuell zu schwierig? Was ist, wenn ich in der Schule keinen Informatikunterricht hatte? Das ist gar nicht so unwahrscheinlich, denn trotz ihrer umfassenden Relevanz hat die Informatik noch keineswegs überall ihren festen Platz im Schulsystem gefunden. Und es gibt noch viele andere Gründe, sich zu fragen: Bin ich fit fürs Informatikstudium?

Dieses Buch kann Ihnen diese Sorge nehmen – vorausgesetzt, Sie lesen es und gehen auch den Knobeleyen zu Beginn und den Aufgaben am Ende jedes Kapitels nicht aus dem Weg. Die wichtigsten Themen und Bereiche der Informatik werden behandelt, so dass Sie am Ende einen breiten Überblick über das Fach und entsprechende gute Grundkenntnisse besitzen werden. Und das ist auf jeden Fall erstrebenswert, selbst wenn Sie nicht unbedingt Informatik studieren wollen. Wir alle sollten in der Lage sein, Entwick-

lungen, die die Welt und die Menschheit verändern, zu verstehen und auch kritisch zu begleiten. In diesem digitalen Zeitalter ist das ohne Informatikkenntnisse aber nicht möglich. Deshalb ist dieses Buch im Prinzip für jede(n) geeignet.

Die Autoren dieses Buchs haben selbst ein Studium im Bereich der Informatik absolviert und waren schon früh von den Möglichkeiten der Informatik begeistert – zwei der Autoren haben als Schüler höchst erfolgreich am Bundeswettbewerb Informatik teilgenommen. Sie wissen deshalb genau, welche Themen für einen Einstieg geeignet und im Studium wichtig sind. Entscheidend ist aber, dass die Autoren selbst von ihrem Fach begeistert sind und dies auch vermitteln. Lassen Sie sich von dieser Begeisterung anstecken und werden Sie Informatikerin oder Informatiker! Aber, bei aller Begeisterung: Je mehr Sie Theorie und Praxis der Informatik beherrschen, desto mehr Macht wächst Ihnen zu, die Welt und das Leben Ihrer Mitmenschen zu verändern. Der Verantwortung, die sich aus dieser Macht ergibt, sollte sich jeder Informatiker und jede Informatikerin stets bewusst sein. Auch dieser Aspekt wird übrigens im vorliegenden Buch nicht vergessen. Und das ist nun wirklich begeisternd.

Bonn, im April 2017

Dr. Wolfgang Pohl

Geschäftsführer Bundesweite Informatikwettbewerbe (BWINF)

## Intro

Haben Sie sich schon einmal gefragt, warum Sie jetzt ein gedrucktes Buch in den Händen halten können? Wie die Website zum Buch und alle anderen Websites auch programmiert und auf Ihrem Computer angezeigt werden können? Oder vielleicht, wieso eine Office-Anwendung oder ein Computerspiel auf Ihrem Computer ausgeführt werden kann? Die Informatik hat inzwischen einen großen Einfluss auf alle Bereiche unseres Lebens genommen.

Versuchen Sie doch einmal, das Gegenteil zu beweisen: Gehen Sie im Kopf Ihre letzte Woche durch, und schätzen Sie, mit wie vielen Dingen Sie in Kontakt gekommen sind, die *keinen* Bezug zu Informatik haben. Haben Sie etwas gefunden? Dann lassen Sie uns über ein paar Vorschläge genauer nachdenken:

Dachten Sie an Essen oder Lebensmittel? Die Landwirtschaft ist eine der digitalisiertesten Branchen überhaupt, mit dutzenden computergesteuerten Systemen zur Qualitätssicherung, automatischen Bewässerung, Schädlingsbekämpfung, Ernteplanung und so weiter. Handelt es sich um ein maschinell hergestelltes Produkt oder überhaupt um ein Produkt, das in irgendeiner Form ausgeliefert wird, so steuert vermutlich ein Computer die Produktion, und mit Sicherheit hat bei der Auslieferung ein Algorithmus dem Paketboten den schnellsten Weg ans Ziel gezeigt.

Vielleicht hatten Sie an ein Musikstück oder einen Film gedacht? Kaum eine Medienproduktion kommt mittlerweile ohne digitale Aufnahme- und Bearbeitungstechnik aus, und selbst ältere Werke wurden längst in Nullen und Einsen überführt, übers Internet verbreitet und für die Ewigkeit archiviert.

Ein Gebäude? Wahrscheinlich hat eine Software den Architekten und den Statiker dabei unterstützt, Planungsfehler zu vermeiden. Wenn das Gebäude etwas neuer ist, werden vielleicht die Heizung und das Licht von einem Computer gesteuert. Wenn es antik ist, wurde es wahrscheinlich schon mit modernster computergestützter Technologie erfasst, um für die Geschichtsforschung neue Erkenntnisse zu gewinnen.

Ein Lebewesen? Informatiker versuchen, von der Natur zu lernen, und entwickeln Algorithmen, die zum Beispiel vom Verhalten von Ameisen inspiriert sind. Auch medizinische Versorgung ohne die automatische Verarbeitung großer Datenmengen wäre heute undenkbar.

Das nächstgelegene Gewässer? Das Wetter? Höchstwahrscheinlich überwachen zu genau diesem Zeitpunkt dutzende Sensoren den Wasserstand, den Sauerstoffgehalt, die Fließgeschwindigkeit und weitere Werte der Flüsse und Seen in Ihrer Umgebung. Und

einige der leistungsstärksten Computer der Welt berechnen gerade Daten für Ihre Wettervorhersage in den nächsten Nachrichten.

Computer und digitale Systeme, so zeigt uns dieses kleine Gedankenspiel, begleiten uns mittlerweile in allen unseren Lebenslagen und sind aus unserer Welt nicht mehr wegzu-denken.

## Das Buch

Die Informatik ist eine faszinierende Wissenschaft, die unser gesamtes Leben prägt wie kaum ein anderes Fachgebiet. Mit diesem Buch wollen wir Ihnen diese Faszination vermitteln und Sie begeistern und befähigen, ein Studium der Informatik zu beginnen. Sollten Sie daher gerade am Ende Ihrer Schullaufbahn sein und Interesse an einem Informatikstudium haben, ist dieses Buch genau das Richtige für Sie! Das Gleiche gilt, falls die Schulzeit schon etwas länger zurückliegt, und das ganz unabhängig davon, ob Ihnen dort »Informatik« als Fach begegnet ist. Da die einzigen Voraussetzungen zum Lesen einfache Schulmathematik und Spaß am Knobeln und Lösen von logischen Rätseln sind, benötigen Sie keine weiteren Vorkenntnisse, um direkt ins Thema einsteigen zu können. Falls Sie, zum Beispiel aus dem Schulunterricht, einige Informatikkenntnisse mitbringen, hilft Ihnen das Buch dabei, dieses Wissen zu festigen und auszubauen.

Auch ohne Studienpläne sind Sie selbstverständlich herzlich eingeladen, mit diesem Buch einen Einblick in die Informatik zu gewinnen. Unserer Meinung nach gehört ein Grundverständnis digitaler Technologien längst zum wichtigen Allgemeinwissen, und die Denkmuster unserer Wissenschaft sind auch außerhalb der Informatik sehr nützlich. Egal also, ob Sie selbst etwas Informatik lernen möchten oder vielleicht auch als Lehrkraft Ideen für einen ansprechenden Informatikunterricht suchen: Wir sind uns sicher, dass diese Lektüre ein guter Einstieg ist!

## Wie liest man dieses Buch?

Mit Ausnahme von ein paar Grundlagen am Anfang ist es nicht notwendig, die Kapitel in der abgedruckten Reihenfolge zu lesen. Ganz im Gegenteil: Sie können frei nach Interesse und Wissensstand Kapitel überspringen oder spätere Kapitel vorziehen.

Wir empfehlen jedoch, auf jeden Fall mit dem ersten Kapitel über Algorithmen zu beginnen, da Sie dort eingeführte Konzepte und Schreibweisen auch später immer wieder benötigen werden. Wenn Sie dann Gefallen an algorithmischem Denken gefunden haben, können Sie dieses in den Kapiteln 2 bis 9 anwenden, denn dort sprechen wir über verschiedene Standardalgorithmen und Analysetechniken der Informatik – sozusagen die Werkzeugkiste eines jeden Informatikers. Kapitel 10 macht einen Ausflug in die theoretische Informatik.

In den Kapiteln 11 und 12 verlagert sich der Fokus auf die Strukturierung von Daten und Systemen, bevor sich Kapitel 13 bis 15 der Technik hinter Computern, Netzwerken und sicherer Datenübertragung widmen. Ab Kapitel 16 wird es richtig praktisch, denn dort beschäftigen wir uns mit der Frage, wie eigentlich große und komplizierte Software entwickelt wird. In Kapitel 19 schreiben Sie dann selbst Programme am Computer – hierfür ist es ratsam, zumindest die Grundlagen zu Algorithmen, Datenstrukturen, Objekt-orientierung, Computern und Softwareentwicklung gelesen zu haben.

Abschließend beschäftigen wir uns in Kapitel 20 mit ethischen und gesellschaftlichen Fragestellungen: Was darf Informatik alles, und welche Verantwortung tragen wir als Informatiker? Um Ihnen wirklich alles an die Hand zu geben, was Sie für einen erfolgreichen Studienstart benötigen, geben wir Ihnen im Extro noch einen Überblick über Ausbildungswege und mögliche Berufsfelder und verraten, wie ein Informatikstudium eigentlich abläuft.

Die meisten Kapitel fangen mit einer Knobelei an, also einer kleinen Rätselaufgabe, die Sie im Kopf, auf Papier oder mit wenigen Alltagsgegenständen lösen können – insbesondere brauchen Sie außer für Kapitel 19 keinen Computer für die Bearbeitung. Anschließend lösen wir die Knobelei auf und zeigen, welche Konzepte der Informatik sich dahinter verbergen. Stück für Stück führen wir im Hauptteil des Kapitels Begriffe und Techniken ein, die wir am Kapitelende zusammenfassen und in den Gesamtzusammenhang stellen.

Ein paar Aufgaben von einfach bis schwer geben Ihnen dann die Möglichkeit, das Gelesene zu verarbeiten und anzuwenden und zu testen, ob Sie alles verstanden haben. Auch diese Aufgaben können Sie (mit Ausnahme von Kapitel 19) völlig ohne Computer oder andere technische Hilfsmittel bearbeiten. Die Lösungen zu den Aufgaben finden Sie im Anhang.

Folgende Stile und Icons verwenden wir, um Dinge zu kennzeichnen:

### ► Aufgaben

Alles, was Aufgaben betrifft, erkennen Sie auf den ersten Blick am schraffierten Seitenrand.

Zu Beginn des Kapitels gibt es eine »Knobelei zum Einstieg«, mit der Sie ohne Vorkenntnisse in das Thema des Kapitels einsteigen können.

Am Ende der Kapitel fordern Aufgaben (🧐) Sie heraus.

Manche Aufgaben fordern Sie voraussichtlich mehr als andere. Diesen Aufgaben haben wir nicht nur einen, sondern gleich drei Doktorhütchen (🎓🎓🎓) vorangestellt.

### ► Lösungen

Lösungen finden Sie immer hinter den Aufgaben eines Kapitels; jede Lösung trägt die gleiche Nummer wie die Aufgabe, die sie löst.

### ► Begriffe

*Begriffe* erläutern wir immer im Zusammenhang. Sie erkennen einen neu eingeführten Begriff an der *kursiven Schrift*.

#### Kompakt erklärt – bitte genau hinsehen

In diesem Buch lernen Sie viele neue Konzepte, Sachverhalte und Zusammenhänge der Informatik kennen. Manche davon haben wir hervorgehoben, weil Sie bei der Lektüre vielleicht kurz innehalten möchten; denn mit einem Kasten wie diesem unterbrechen wir unseren Gedankengang, um Ihnen wichtige Grundlagen zu vermitteln oder weiterführende Ideen vorzustellen. Diese Textstellen lassen sich zwar leicht wiederfinden, aber wir empfehlen schon beim ersten Lesen ein besonderes Augenmerk.

### ► Pseudocode und Code

Pseudocode und Code erkennen Sie an dieser Schriftart:

```
Wiederhole solange Ende des Buches nicht erreicht  
  Wiederhole solange Seitenende nicht erreicht  
    Falls Abschnittstyp = Aufgabe
```

*Schlüsselwörter* der Programmiersprache bzw. solche Begriffe, die in einer Programmiersprache als Schlüsselwort (vgl. Kapitel 19, »Hands-on: Programmieren mit Java«) umgesetzt werden, drucken wir farbig. Moderne Entwicklungsumgebungen bieten verschiedene reichhaltige Farbdarstellungen Ihres Codes, die außerdem z. B. Zahlenwerte, Kommentare, Zeichenketten in jeweils eigenen Farben darstellen. Schlüsselwörter werden dabei immer berücksichtigt, und so halten wir es auch in diesem Buch.

### Webseite zum Buch

Auf unserer Website [www.fit-fuers-informatikstudium.de](http://www.fit-fuers-informatikstudium.de) stellen wir weiteres Material zum Buch bereit. Dieses umfasst noch ausführlichere Lösungserklärungen zu manchen Aufgaben, eine Linksammlung zu weiterführender Literatur sowie alle Codebeispiele und Werkzeuge für das Programmierkapitel 19.

Wie in jedem Fachbuch wird auch bei uns der Fehlerteufel nicht völlig untätig geblieben sein. Auf unserer Website nehmen wir Fehlermeldungen entgegen und stellen Korrekturen für bereits bekannte Fehler bereit. Außerdem freuen wir uns natürlich sehr über

Feedback: über das Kontaktformular auf unserer Website können Sie uns Fragen stellen und Lob, Anregungen und natürlich auch Kritik loswerden.

### Danksagungen

Zuallererst möchten wir uns bei unserer Lektorin Almut Poll bedanken, ohne die es dieses Buch nicht gegeben hätte. Ihrer Feder entstammen Idee und Konzept des Projekts, für das sie uns als Autoren an Bord geholt hat. Wir danken ihr für das in uns gesetzte Vertrauen und die kompetente Unterstützung beim Schreiben des Buchs! Ebenso danken wir allen weiteren Mitarbeitern des Rheinwerk Verlags, die im Hintergrund dafür gearbeitet haben, dass Sie nun ein fertiges Exemplar des Buches in der Hand halten können.

Unser Dank geht auch an Wolfgang Pohl, der das Geleitwort verfasst und damit dem Buch einen schönen Kontext gegeben hat.

Wir danken Thomas Bläsius, Timo Kötzing und Pascal Lenzner vom Hasso-Plattner-Institut für die fachliche und didaktische Durchsicht der Kapitel. An vielen Stellen sind ihre wertvollen Anmerkungen und Ratschläge in das Buch eingeflossen und haben damit die Verständlichkeit der Erklärungen verbessert. Auch ein paar Fehler konnten wir dank den Dreien ausmerzen.

Ebenso geht ein herzliches Dankeschön an unsere Freunde und Familien, die sich nie darüber beschwert haben, wenn wir sie völlig aus dem Zusammenhang gerissen um Rat gefragt haben. Wir danken insbesondere Jonas Chromik, Lukas Faber, Franka Fischbeck, Ann Katrin Kuessner, Rita Neubert, Valentin Pinkau, Jan Schellenberg, Robert Schmid, Sebastian Serth und Jennifer Stamm dafür, dass sie uns bei der Entstehung des Buches als Testpersonen unterstützt und uns Feedback zu den Texten gegeben haben.

Auch wir selbst befinden uns noch in der Informatikausbildung und sprechen dafür allen Professoren und Dozenten, Mitarbeitern, Kollegen und Kommilitonen am Hasso-Plattner-Institut Potsdam großen Dank dafür aus, dass sie mit Spaß und Leidenschaft bei der Sache sind und wir tagtäglich von ihnen lernen können. Insbesondere sind wir sehr dankbar für all die Gelegenheiten, die uns geboten wurden und werden, Schülerinnen und Schülern, Lehrkräften und anderen Studierenden im Rahmen von Workshops, Camps und Tutorien unser Wissen weiterzugeben und dabei unsere eigenen didaktischen Fähigkeiten zu erproben und zu verbessern.

### Was ist eigentlich Informatik?

Nicht ganz zu Unrecht denken die meisten Menschen beim Begriff »Informatik« zuallererst an Laptops, Smartphones und das Internet. Dennoch haben wir uns dafür ent-

schieden, dieses Buch so zu schreiben, dass Sie es fast komplett ohne Computer lesen und insbesondere die Aufgaben ganz analog bearbeiten können. Die Erklärung dafür liegt darin verborgen, was Informatik außerhalb der Arbeit mit Computern eigentlich noch alles umfasst.

### Die Anfänge der mechanischen Rechenmaschinen

Der Ursprung der Informatik liegt in dem Traum, Berechnungen nicht mühsam von Hand durchführen zu müssen, sondern wiederkehrende und monotone Aufgaben von einer Maschine erledigen zu lassen. Gleich drei Mathematiker bauten im 16. Jahrhundert erste mechanische Rechenmaschinen. Aus heutiger Sicht ist der bedeutendste davon Gottfried Wilhelm Leibniz, dessen Maschine alle vier Grundrechenarten beherrschte. Zwar konnte zu seinen Lebzeiten die »Leibniz'sche Rechenbank« nie voll funktionstüchtig gebaut werden, weil die damaligen Möglichkeiten der Feinmechanik noch nicht ausgereift genug waren, moderne Nachbauten funktionieren jedoch einwandfrei. Die Maschine prägte die bis heute typische Trennung einer automatischen Berechnung in *Eingabe*, *Berechnung* und *Ausgabe*. Darüber hinaus erkannte Leibniz, dass sich das Binärsystem – also das Rechnen mit den Ziffern 0 und 1 anstatt des uns geläufigen Zehnersystems mit den Ziffern 0–9 – besonders gut für maschinelle Berechnungen eignet.

Neben vielen Weiterentwicklungen dieser Rechenmaschinen durch verschiedene Erfinder leitete Joseph-Marie Jacquard im 18. Jahrhundert mit seinem Lochkarten-gesteuerten Webstuhl das nächste Kapitel der Informatikgeschichte ein. Statt ein kompliziertes Webmuster fest in die Mechanik des Webstuhls einbauen zu müssen, ermöglichten austauschbare Karten mit ausgestanzten Löchern, ein und denselben Webstuhl für vielerlei Muster zu verwenden. Im 19. Jahrhundert übernahm der Mathematiker Charles Babbage diese Idee für den Entwurf seiner »Analytical Engine«, eine von einer Dampfmaschine angetriebene Rechenmaschine, die über Lochkarten programmiert werden sollte. Auch wenn die Maschine nie vollständig gebaut wurde, so geht man heutzutage doch davon aus, dass sie voll funktionstüchtig gewesen wäre. Im Gegensatz zu den Konstruktionsplänen der Maschine wurde eine Reihe von Programmen für selbige überliefert. Die Autorin dieser Programme, die Mathematikerin Ada Lovelace, gilt damit als erste Programmiererin überhaupt. Sie hatte außerdem die Vision, dass solche Rechenmaschinen langfristig nicht nur mit Zahlen, sondern auch mit Bildern und Musik arbeiten könnten.

### Programmierbare Computer

Im Jahr 1938 gelang es endlich Konrad Zuse, mit der »Z1« die erste programmierbare mechanische Rechenmaschine zu vollenden. Drei Jahre später stellte er die Nachfolgerin

»Z3« fertig, die als erster Computer der Welt gilt und nicht mehr mechanisch, sondern mit zweitausend Relais arbeitete. Programmiert wurde der Computer über einen Lochstreifen, die Eingabe und Ausgabe erfolgten über eine Tastatur und Lampen.

Ebenfalls in den 1930er Jahren wurden wichtige Grundsteine der theoretischen Informatik geschaffen. Wissenschaftler wie Alan Turing und Alonzo Church formalisierten, welche Probleme überhaupt von Maschinen gelöst werden können. Das von ihnen maßgeblich geformte Forschungsfeld der *Berechenbarkeitstheorie* beschreibt die Grenzen der Möglichkeiten von Computern, und zwar nicht nur die der damals bekannten, sondern die aller Maschinen und Modelle zur automatischen Berechnung, die bis heute entdeckt und entwickelt wurden.

Der Brite Alan Turing war als Kryptoanalytiker auch maßgeblich daran beteiligt, im Zweiten Weltkrieg die abgefangenen verschlüsselten Nachrichten der Deutschen zu entziffern. Die Fähigkeit, die gegnerische Kommunikation abzuhören, war so bedeutend, dass damals gleich mehrere Großrechner im Vereinigten Königreich, in den USA und in Deutschland entwickelt wurden. Neben der Entschlüsselung von Nachrichten dienten sie auch zur Automatisierung von komplexen aerodynamischen Berechnungen für Flugzeuge und Geschosse. Mehr und mehr wurden dabei die fehleranfälligen und langsamen Relais durch Elektronenröhren aus der Radartechnik ersetzt.

Nachdem lange Zeit die Programmierung der Maschinen direkt über Kabel oder Löcher erfolgt war und Befehle über Oen und Ien eingegeben wurden, kam die Informatikerin Grace Hopper um 1950 auf die Idee, Programme stattdessen auf Englisch zu schreiben und diese anschließend mit einem Computer in Maschinenbefehle zu übersetzen. Sie entwickelte darauf aufbauend die ersten Programmiersprachen und die dazugehörigen Übersetzer, auch *Compiler* genannt.

### Miniaturisierung und Siegeszug des Computers

Unter anderem aufgrund ihrer immensen Größe – der US-Amerikanische *ENIAC* nahm 170 m<sup>2</sup> Platz ein – ging man damals davon aus, dass nur wenige Computer weltweit benötigt würden. Mit der Erfindung des Transistors 1947 änderte sich vieles, denn auf einmal war es möglich, viel schnellere, kleinere und wartungsärmere Computer zu konstruieren. Im Gegensatz zu den meisten vorherigen Maschinen speicherten nun Computer Programme und Daten im selben Speicher, entsprechend dem von John von Neumann beschriebenen Aufbau eines Computersystems. Die Entwicklung der ersten Mikroprozessoren durch Intel beziehungsweise Texas Instruments im Jahr 1971 führte schließlich dazu, dass tausende Transistoren auf einem kleinen Chip Platz fanden. Anschließend vergrößerte sich die Anzahl an Transistoren auf einem Chip rasant. Der Wissenschaftler Gordon Moore prognostizierte eine Verdoppelung der Schaltkreisdichte



alle ein bis zwei Jahre. Bis heute hält die Weiterentwicklung der Kernbestandteile von Computersystemen Schritt mit dieser Prognose.

Erst in den 1980er Jahren fanden Computer nach und nach Einzug in Privathaushalte, nachdem zuvor vor allem Großgeräte für Unternehmen entwickelt worden waren. Ab 1990 revolutionierte die Öffnung des Internets für die allgemeine kommerzielle Nutzung die weltweite Kommunikation. Zuvor war dieses Netzwerk auf Forschungseinrichtungen und Universitäten beschränkt gewesen, wo insbesondere die E-Mail für schnellen Nachrichtenversand genutzt wurde. Das 1989 von Tim Berners-Lee entwickelte World Wide Web ist heutzutage die bekannteste Ausprägung des Internets.

Seitdem halten Computer Einzug in immer mehr Bereiche unseres Lebens. Angefangen bei Taschenrechnern, digitalen Armbanduhr und Mobiltelefonen explodierte die Verbreitung von Computern spätestens 2007. Zwar waren Smartphones keine neue Erfindung, erst jedoch die Einführung des iPhones in jenem Jahr führte dazu, dass heute fast jeder eine universale, automatische Rechenmaschine in der Hosentasche trägt, die millionenfach leistungsstärker ist, als es sich die Erfinder der ersten mechanischen Rechner hätten erträumen können. Diese Verbreitung und Universalität von Computern macht den Beruf des Informatikers enorm spannend.

### **Was macht ein Informatiker?**

Kommunikation, Mobilität, Multimedia, Finanzwesen, Ingenieurwesen, Fabrikproduktion, Forschung und Entwicklung, Medizin, Unternehmensorganisation, Politik – die Anwendungsfelder von informatischen Systemen umfassen inzwischen praktisch alles. Bei aller Weiterentwicklung ist jedoch das Grundprinzip geblieben: Die Informatik beschäftigt sich noch immer damit, wie Berechnungen automatisiert durchgeführt werden. Da die zugrundeliegenden Problemstellungen aus beliebigen Domänen stammen können, ist das Aufgabenfeld eines Informatikers so universell wie kaum ein anderes.

Hauptsächlich beschäftigt sich ein Informatiker damit, wie Probleme gut, effizient und automatisiert gelöst werden können. Dass diese Automatisierung in erster Linie durch Computer geschieht, ist zunächst einmal zweitrangig, denn eine strukturierte und präzise Beschreibung von Problemlösungen ist für Mensch wie Maschine gleichermaßen wertvoll. Die enorme Rechenkraft von heutigen Computern sorgt jedoch dafür, dass die Lösungsverfahren auch schnell ausgeführt werden können, und macht damit die Ergebnisse des Informatikers anwendbar.

Wenn man so will, ist der Computer das wichtigste Werkzeug des Informatikers für die Ausführung der Lösungen. In der Arbeit mit Computern fungiert man jedoch in erster Linie als Übersetzer zwischen der Sprache des Problems und der Sprache des Compu-

ters. Bevor diese Übersetzung stattfinden kann, muss aber zunächst das Problem gelöst und diese Lösung in eine strukturierte Form gebracht werden.

Neben mathematisch-logischem Denken ist dabei vor allem die Kompetenz eines Ingenieurs gefragt: Ohne den Blick fürs Ganze, Erfindergeist, Kreativität und Teamfähigkeit kommt man in der Informatik nicht sehr weit, denn typischerweise sind die gestellten Probleme so umfangreich und knifflig, dass deren Lösung viel Knobelei, Ausprobieren und Teamarbeit erfordert.

In die grundlegenden Denkweisen, Lösungsansätze und Ideen der Informatik wollen wir Ihnen in den folgenden Kapiteln einen Einblick geben. Die wichtigsten Werkzeuge dafür sind Papier und Stifte. Haben Sie sich dies bereitgelegt? Dann kann es ja losgehen!



## Kapitel 9

## Graphen

*Graphen eignen sich sehr gut dafür, zusammengehörige oder abhängige Daten zu speichern. Wir werden in diesem Kapitel erklären, was Graphen sind und wie man mit ihnen Daten in Beziehung setzt. Außerdem werden wir einfache Algorithmen auf Graphen ausprobieren.*

## Morgendliches Anziehen

Jeden Morgen dasselbe! Nach dem Duschen nimmt man die Kleidungsstücke, die man am Tag tragen möchte, aus dem Schrank und zieht sie dann an. Aber wie oft passiert es, dass man schlaftrunken das T-Shirt anzieht und danach das Unterhemd noch in der Hand hält? Natürlich ist es ärgerlich, kostet aber bei wenigen Kleidungsstücken noch nicht so viel Zeit. Aber wie ist es, wenn man im Winter feststellt, dass man das T-Shirt vergessen hat, wenn man schon den Schneeanzug trägt?

Damit wir mit solchen Situationen in der Zukunft keine Zeit mehr verschwenden, wollen wir uns überlegen, wie wir diesen Vorgang besser organisieren können. Stellen Sie sich vor, Sie möchten heute die folgenden Kleidungsstücke tragen:

- |                   |               |               |
|-------------------|---------------|---------------|
| ▶ Handschuhe      | ▶ Pullover    | ▶ Socken      |
| ▶ lange Unterhose | ▶ Schneeanzug | ▶ T-Shirt     |
| ▶ Mütze           | ▶ Schuhe      | ▶ Unterwäsche |

Überlegen Sie sich doch einmal, welche Abhängigkeiten beim Anziehen Ihrer Kleidungsstücke existieren, und schreiben Sie sie auf. Dass die Unterwäsche vor dem T-Shirt angezogen werden muss, könnte man beispielsweise so notieren:

Unterwäsche → T-Shirt

Da es egal ist, in welcher Reihenfolge Sie zum Beispiel die Unterwäsche und die Socken anziehen, kommt diese Kombination in der Liste der Abhängigkeiten nicht vor.

Fassen Sie im nächsten Schritt die Abhängigkeiten zusammen, sodass jedes Kleidungsstück nur noch einmal vorkommt. Welches Bild kommt dabei zustande? Wie könnten Sie nun mit diesem Bild eine Reihenfolge finden, in der Sie die Kleidungsstücke ohne Probleme anziehen können?

In Abbildung 9.1 haben wir die Abhängigkeiten der Kleidungsstücke graphisch dargestellt. Möglicherweise haben Sie in Ihrem Bild einige Pfeile mehr. Diese haben wir entfernt, weil sie indirekt in der Darstellung enthalten sind. Beispielsweise muss natürlich das T-Shirt vor dem Schneeanzug angezogen werden, jedoch ist dies dadurch gegeben, dass das T-Shirt vor dem Pullover und dieser vor dem Schneeanzug angezogen werden muss.

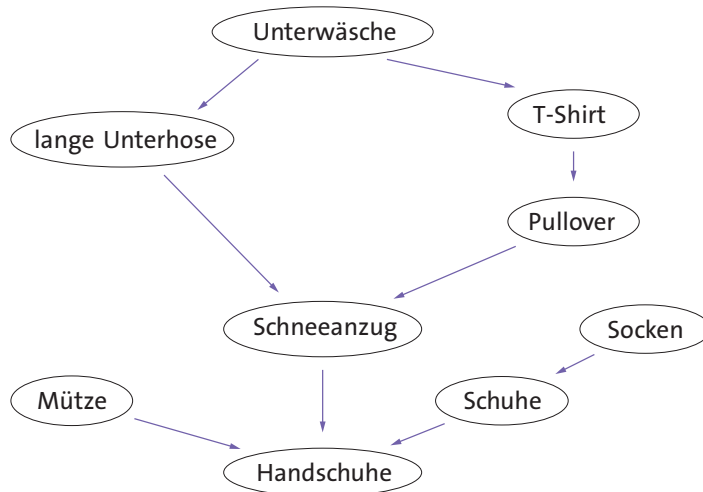


Abbildung 9.1 Abhängigkeiten der Kleidungsstücke

Aus dieser Abbildung können Sie nun schon einiges ablesen. Beispielsweise müssen die Schuhe, der Schneeanzug und die Mütze angezogen sein, bevor Sie die Handschuhe anziehen können. Außerdem sind die Handschuhe das letzte Kleidungsstück, das angezogen wird. Beginnen können Sie mit der Unterwäsche, den Socken oder der Mütze, da diese keine vorherigen Abhängigkeiten, also keine eingehenden Pfeile, haben. Genau so können Sie auch eine Reihenfolge für die Kleidungsstücke finden, die insgesamt funktioniert. Sie können ein Kleidungsstück anziehen, wenn es keine vorherigen Abhängigkeiten hat. Ziehen Sie ein Kleidungsstück an, kann es aus dem Diagramm entfernt werden. Anschließend haben Sie neue Kleidungsstücke, die Sie anziehen können. Diesen Algorithmus nennt man auch *topologisches Sortieren*. Eine mögliche Reihenfolge zum Anziehen der Kleidungsstücke:

Unterwäsche → lange Unterhose → T-Shirt → Pullover → Schneeanzug → Socken → Schuhe → Mütze → Handschuhe

## Verknüpfte Daten

Verknüpfte Daten sind überall in der Welt vorhanden. Etwas später in diesem Kapitel werden wir uns zum Beispiel mit Straßenverbindungen zwischen Städten und einem Freundschaftsnetzwerk beschäftigen. Solche Verknüpfungen in den Datenstrukturen darzustellen, die Sie bisher kennengelernt haben, ist nur schwer möglich.

Graphen bieten sich genau dafür an. Sie bestehen grundsätzlich aus einer Menge von *Knoten* und *Kanten*. Ein Knoten im Graphen steht stellvertretend für ein Objekt. Das kann beispielsweise wie in der Knebelerei ein Kleidungsstück oder eine Stadt auf einer Landkarte sein. Eine Kante verbindet zwei Knoten, wie zum Beispiel eine Straße zwei Städte verbindet. Wenn zwei Knoten mit einer Kante verbunden sind, nennt man sie *benachbart*. In vielen Algorithmen auf Graphen spricht man von der *Nachbarschaft* eines Knotens. Damit sind dann alle Knoten gemeint, die mit diesem Knoten benachbart sind.

Knoten notiert man im Regelfall mit einem Kreis. Eine Kante ist eine Linie zwischen zwei Knoten. In Abbildung 9.2 sehen Sie einen einfachen Graphen, in dem zwei Knoten A und B durch eine Kante verbunden sind. Beim Zeichnen von Graphen ist es nicht wichtig, wo die Knoten platziert werden. A und B könnten also beliebig verschoben werden; die Aussage, dass die beiden Knoten verknüpft sind, bleibt bestehen. Da es bei Graphen nur um das Darstellen dieser Verknüpfung geht, ist die genaue Positionierung unwichtig.



Abbildung 9.2 Ein einfacher Graph mit zwei Knoten und einer Kante zwischen diesen Knoten

## Varianten von Graphen

Graphen modellieren immer einen Sachverhalt. Manchmal reicht die Information, dass zwei Knoten verbunden sind, jedoch nicht aus, sondern es müssen abhängig vom Kontext weitere Informationen gespeichert werden. Zusätzlich zur Beziehung zwischen zwei Knoten kann man in einem Graphen auch gut speichern, wie genau diese Beziehung ausgestaltet ist. Gilt sie in beide Richtungen? Ist sie mit einer Art Kosten verbunden? Für diese zwei typischen Anforderungen gibt es Varianten von Graphen, die diese zusätzlichen Informationen modellieren können.

### Gerichtete Kanten

*Gerichtete Kanten* geben die Möglichkeit, einseitige Verbindungen in einem Graphen darzustellen. Die Richtung der Kante wird durch einen Pfeil im Graphen markiert, wie Abbildung 9.3 zeigt. In diesem Beispiel ist Knoten A mit Knoten B verbunden, und Knoten A zeigt auf Knoten B.



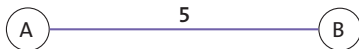
**Abbildung 9.3** Ein einfacher gerichteter Graph. Die Kante zeigt von Knoten »A« zu Knoten »B«.

Solche Graphen haben Sie schon in der Knobelei kennengelernt. Die Abhängigkeiten zwischen den Kleidungsstücken waren einseitig. Ein Straßennetz ist ein anderes Beispiel, bei dem gerichtete Kanten oft zum Einsatz kommen. So können wir Einbahnstraßen als gerichtete Kanten modellieren, um beispielsweise beim Berechnen von Routen darzustellen, dass die Straße nur in eine Richtung befahren werden kann.

Wenn keine Kante des Graphen eine Richtung hat, wird der Graph *ungerichtet* genannt.

### Gewichtete Kanten

Möchten wir zum Beispiel die Distanzen zwischen zwei verbundenen Städten ausdrücken, können wir dies einfach mit *Kantengewichten* tun. Kantengewichte werden als Zahl an der Kante notiert, wie in Abbildung 9.4 dargestellt, und manchmal auch als *Kantekosten* bezeichnet.



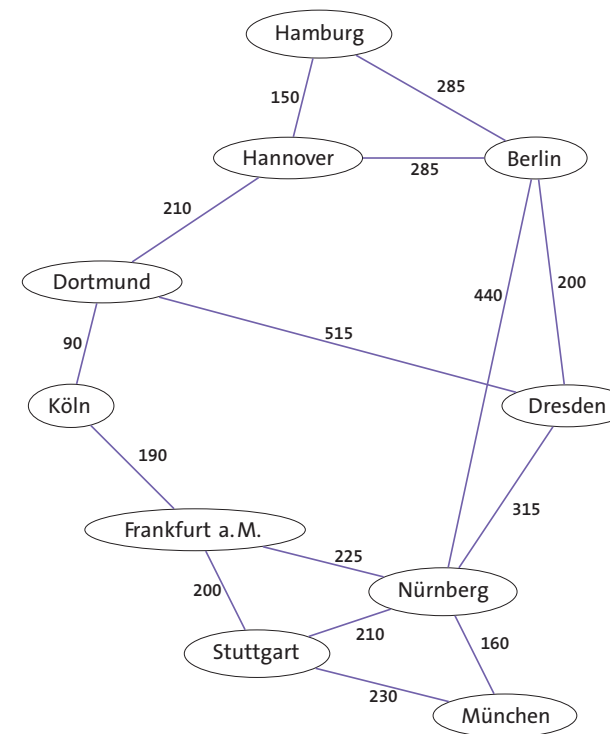
**Abbildung 9.4** Ein einfacher gewichteter Graph. Die Kante zwischen den Knoten »A« und »B« hat das Gewicht 5.

Die Bedeutung eines Kantengewichts kann sehr vielfältig sein. Es kann die Distanz zwischen zwei Städten beschreiben, die Übertragungsgeschwindigkeit einer Datenleitung oder auch den Grad einer Freundschaft zwischen zwei Personen. Die Bedeutung hängt ganz von dem Kontext ab, für den der Graph modelliert wird, und spielt dann in den Algorithmen eine Rolle, die auf dem Graphen ausgeführt werden. Angenommen, das Kantengewicht eines Graphen steht für die Distanz zwischen zwei Städten. Wenn dann die kürzesten Wege in der als Graph modellierten Landkarte gesucht werden sollen, entscheidet dieses Gewicht darüber, ob ein Weg über eine Straße (beziehungsweise Kante) führt. Je nachdem, wie hoch das Gewicht ist, kann eventuell auch ein kürzerer Weg gefunden werden, der vielleicht sogar mehr, aber dennoch in Summe günstigere Kanten verwendet.

Hat ein Graph keine Kantengewichte, wird er *ungewichtet* genannt.

### Beispiele für Graphen

Eines der natürlichsten Beispiele für Graphen sind Landkarten. In Abbildung 9.5 haben wir einige deutsche Großstädte und deren Distanzen dargestellt. Die Knoten sind dabei die Städte selbst; die Kanten sind einige Autobahnen, die diese Städte verbinden. Das Gewicht der Kanten sind die Distanzen in Kilometern zwischen den benachbarten Städten. Eine typische Fragestellung zu so einem Graphen ist, ob zwei bestimmte Städte (möglicherweise über andere Städte) miteinander verbunden sind und wie groß die daraus resultierende Distanz zwischen den beiden Städten ist. So können wir mit dem Graphen aus Abbildung 9.5 ermitteln, dass die Entfernung von Hamburg nach München 885 Kilometer beträgt. Der Weg führt von Hamburg über Berlin und Nürnberg nach München.



**Abbildung 9.5** Einige deutsche Großstädte und ihre Verbindungen über Autobahnen. Die Kantengewichte stehen für die Distanz zwischen den Städten.

Das Straßennetzwerk von Deutschland ist schon beim Betrachten einer Straßenkarte als Graph erkennbar. Ein etwas versteckteres Beispiel für Graphen sind soziale Netzwerke. In Abbildung 9.6 haben wir das soziale Netzwerk einer Schulklasse dargestellt. Die Knoten sind die Schüler; eine Kante zeigt an, dass zwei Schüler befreundet sind.

Für dieses soziale Netzwerk könnten wir nun bestimmen, welche Person die beliebteste ist, also am meisten Freunde hat. Die Anzahl der Freunde einer Person entspricht der Anzahl an Kanten am entsprechenden Knoten. Dieser Wert wird auch der *Grad* eines Knotens genannt. Entdecken Sie den beliebtesten Schüler dieser Klasse?

Außerdem könnten wir Cliques finden. Eine Clique ist eine Menge von Personen, die alle miteinander befreundet sind. Auch in der Graphentheorie nennt man dies eine *Clique* und definiert, dass jedes Paar von Knoten in der Clique durch eine Kante verbunden sein muss. Finden Sie die größte Clique in dieser Klasse?

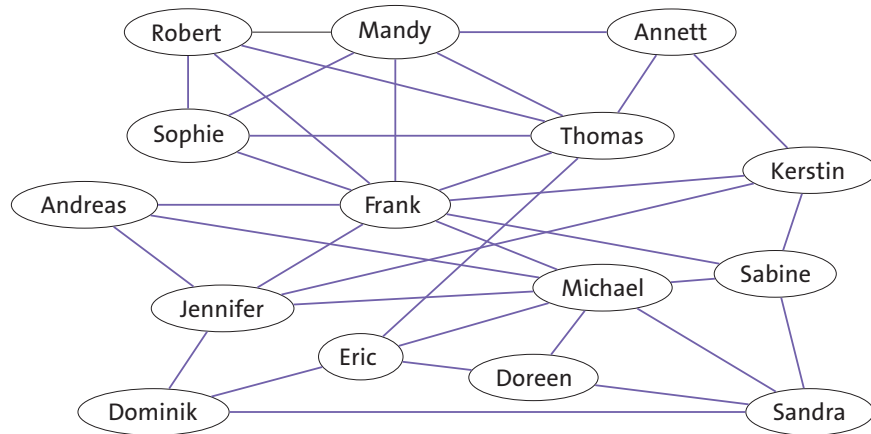


Abbildung 9.6 Das soziale Netzwerk einer Schulklasse

In der Klasse in Abbildung 9.6 ist Frank der beliebteste Schüler, da er 9 Freunde hat. Die größte Clique besteht aus Robert, Mandy, Thomas, Frank und Sophie.

## Suchen und Bewegen in Graphen

Eine der Grundoperationen auf Graphen ist das Suchen nach einem Knoten oder einer Verbindung. Beispielsweise ist es oft relevant, zu wissen, ob ein Knoten von einem anderen Knoten erreichbar ist, sei es, um eine Nachricht in einem Kommunikationsnetzwerk übermitteln zu können oder eine Baustelle im Straßennetz zu umfahren. Dafür werden Suchverfahren verwendet, die systematisch jeden Knoten im Graphen besuchen und dabei nach einem Verbindungsweg zwischen den beiden Knoten Ausschau halten. Jede Suche beginnt bei einem bestimmten Startknoten und endet, sobald entweder das Suchziel (zum Beispiel ein anderer Knoten) erreicht wurde oder alle Knoten betrachtet wurden.

Die zwei Standardverfahren dafür nennen sich *Tiefensuche* und *Breitensuche*. Beide basieren auf der Idee, nach und nach alle Nachbarn eines Knoten zu besuchen, ebenso die Nachbarn der Nachbarn und so weiter. Da der ursprüngliche Knoten selbst auch ein Nachbar seines Nachbarn ist, müssen die Algorithmen dafür speichern, welche Knoten sie schon besucht haben, um nicht im Kreis Nachbarschaftsbeziehungen zu verfolgen. Wann immer ein Knoten *besucht* wird, werden einmal alle seine Nachbarn betrachtet und als *gesehen* aber noch nicht *abgearbeitet* abgespeichert, falls sie nicht schon zuvor besucht oder gesehen wurden. Die beiden Algorithmen verfahren dann fast gleich und unterscheiden sich lediglich darin, in welcher Reihenfolge *gesehene* Knoten besucht werden. Die Breitensuche betrachtet zunächst die nähere Umgebung des Startknotens, bevor sie sich weiter entfernt. Wie eine Welle besucht der Algorithmus also zunächst alle Knoten, die direkt mit dem Startknoten benachbart sind, dann die Knoten mit zwei Kanten Entfernung vom Start, die mit drei und so weiter. Die Tiefensuche dagegen verfolgt zunächst einen Weg durch den Graphen, bis sie erfolgreich war oder in einer Sackgasse gelandet ist, und betrachtet erst dann die Umgebung des Weges. Auf den Graphen in Abbildung 9.7 können Sie diesen Unterschied nachvollziehen.

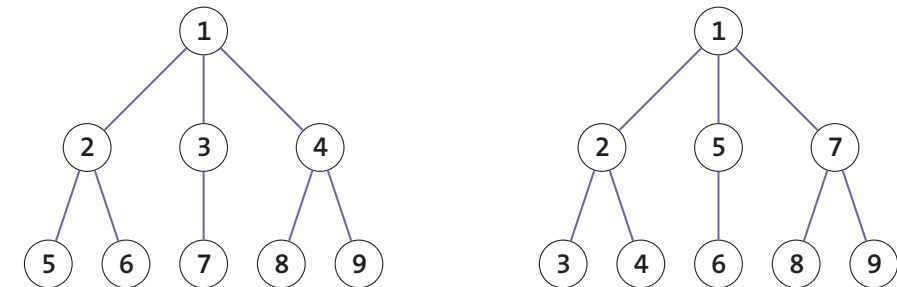


Abbildung 9.7 Breitensuche (links) und Tiefensuche (rechts) besuchen Knoten in unterschiedlicher Reihenfolge.

## Implementierung

Die beiden Suchverfahren können wie in Listing 9.1 implementiert werden. In den ersten beiden Zeilen werden die angesprochenen Listen der besuchten und der gesehenen Knoten angelegt. Zum Anfang kennt der Algorithmus bereits den Startknoten und kann ihn daher der Liste gesehener Knoten hinzufügen. Anschließend werden die folgenden Befehle so lange wiederholt, bis die Liste der gesehenen Knoten leer ist (Zeile 3). In jeder Iteration nimmt sich der Algorithmus den nächsten gesehenen Knoten (Zeilen 4 und 5). Dieser Knoten wird im aktuellen Schleifendurchlauf *besucht*, falls er dies nicht schon zuvor wurde (Zeile 6). Falls das der Fall ist, wird der folgende Code nicht ausgeführt, sondern es wird mit der nächsten Iteration fortgesetzt. Sollte er noch nicht besucht worden



sein, fügt der Algorithmus ihn der Liste der besuchten Knoten hinzu (Zeile 7). So wird verhindert, dass der Algorithmus endlos läuft, weil er immer wieder dieselben Knoten besucht.

Mit *aktuellerKnoten.Nachbarn* wird auf die Liste der Nachbarn des aktuellen Knotens zugegriffen. Diese werden in diesem Schritt gesehen und deshalb der entsprechenden Liste hinzugefügt. Jeder Knoten kann sich mehrfach in der Liste der gesehenen Knoten befinden. Da er aber aufgrund der Überprüfung in Zeile 6 nur einmal vom Algorithmus bearbeitet wird, ist das in Ordnung.

Eingabe: Startknoten *s*

```

1: besuchteKnoten := VerketteteListe()
2: geseheneKnoten := VerketteteListe()
3: geseheneKnoten.first := s
4: Wiederhole solange geseheneKnoten nicht leer
5:   aktuellerKnoten := geseheneKnoten[0]
6:   Entferne erstes Element von geseheneKnoten
7:   Falls aktuellerKnoten nicht in besuchteKnoten dann
8:     Füge aktuellerKnoten zu besuchteKnoten hinzu
9:     Wiederhole für alle n in aktuellerKnoten.Nachbarn
10:      Falls n nicht in besuchteKnoten dann
11:        Füge n zu geseheneKnoten hinzu

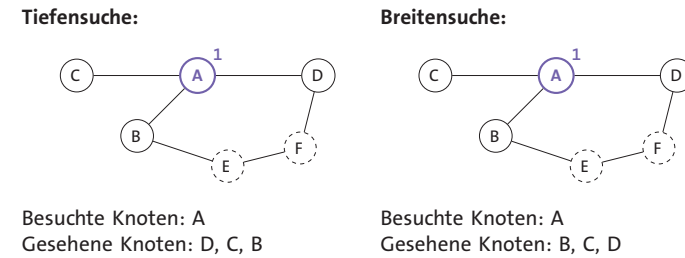
```

**Listing 9.1** Eine Suche über alle Knoten eines Graphen. Es ist nicht spezifiziert, wonach gesucht wird oder was die Ausgabe ist.

Die Implementierung ist für die Tiefen- und die Breitensuche nahezu gleich. Lediglich die Wahl der verwendeten Datenstruktur für die Liste gesehener Knoten führt zu den unterschiedlichen Suchabläufen. Darum unterscheiden sich die Algorithmen nur im Verhalten in Zeile 11. Die Tiefensuche fügt die Knoten zur Liste der gesehenen Knoten vorn hinzu, während die Breitensuche sie hinten anfügt.

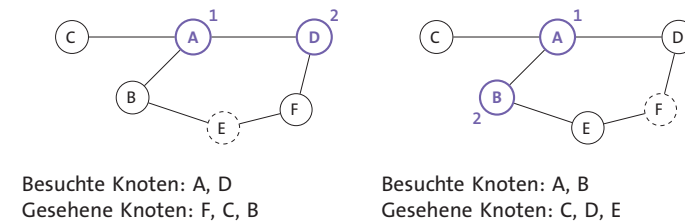
### Beispiel

Im folgenden Beispiel führen wir auf demselben Graphen die Tiefen- und die Breitensuche parallel aus, um die Algorithmen zu veranschaulichen. Die Tiefensuche ist jeweils links dargestellt, die Breitensuche rechts. Besuchte Knoten sind fett markiert, gesehene Knoten durch eine dünne und noch nicht gesehene Knoten durch eine gepunktete Linie. Wir beginnen bei Knoten A, wie Abbildung 9.8 zeigt. Beide Algorithmen sehen die Knoten B, C und D. Wir gehen davon aus, dass die Knoten alphabetisch gespeichert sind und deshalb auch in dieser Reihenfolge der Liste hinzugefügt werden.



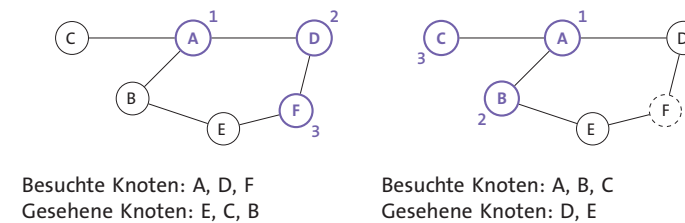
**Abbildung 9.8** Der vorderste Knoten, Knoten »A«, wird besucht. Dadurch werden die Knoten »B«, »C« und »D« gesehen.

Nun wird jeweils der vorderste Knoten aus der Liste mit den gesehenen Knoten besucht. Während die Tiefensuche D als Nächstes besucht, besucht die Breitensuche zuerst B (Abbildung 9.9). Die Tiefensuche fügt den neu gesehenen Knoten F vorn der Liste hinzu, die Breitensuche fügt Knoten E hinten an.



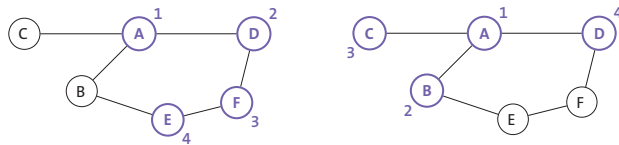
**Abbildung 9.9** Im zweiten Schritt besucht die Tiefensuche Knoten »D«, die Breitensuche Knoten »B«.

Bei der Tiefensuche steht nun Knoten F vorn und wird als Nächstes besucht, wie der linke Teil von Abbildung 9.10 zeigt. Dabei wird Knoten E entdeckt. Die Breitensuche besucht als Nächstes Knoten C und sieht daher keine neuen Knoten.



**Abbildung 9.10** Da die Tiefensuche Knoten »F« besucht, entdeckt sie Knoten »E«. Die Breitensuche entdeckt keinen neuen Knoten, da »C« keine Nachbarn mehr hat.

Die Tiefensuche verfolgt weiter ihren Weg in die Tiefe und besucht *E* als Nächstes. Dadurch wird *B* erneut gesehen und vorn der Liste hinzugefügt. Die Breitensuche arbeitet nach wie vor die direkten Nachbarn von *A* ab und besucht daher *D*. Nun entdeckt auch sie Knoten *F*. Abbildung 9.11 zeigt den aktuellen Stand.

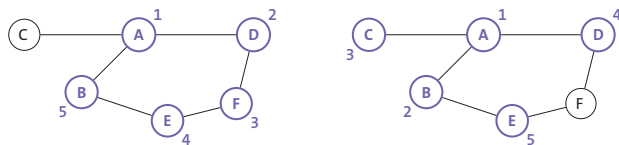


Besuchte Knoten: A, D, F, E  
Gesehene Knoten: B, C, B

Besuchte Knoten: A, B, C, D  
Gesehene Knoten: E, F

**Abbildung 9.11** Die Tiefensuche besucht »E« als Nächstes und findet »B« erneut. Er wird daher der Liste doppelt hinzugefügt. Die Breitensuche findet Knoten »F«, da sie Knoten »D« besucht.

Die Tiefensuche besucht nun *B*. Damit beendet sie ihren Rundlauf, da Knoten *A* schon besucht wurde und deshalb nicht noch einmal der Liste hinzugefügt wird, wie auch Abbildung 9.12 zeigt. Die Breitensuche besucht als Nächstes Knoten *E* und sieht *F* ein zweites Mal. Knoten *F* wird daher erneut der Liste hinzugefügt.

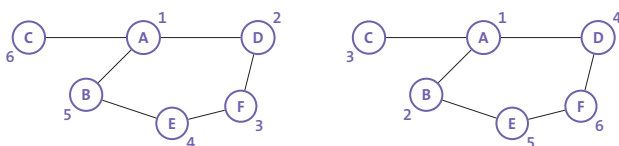


Besuchte Knoten: A, D, F, E, B  
Gesehene Knoten: C, B

Besuchte Knoten: A, B, C, D, E  
Gesehene Knoten: F, F

**Abbildung 9.12** Die Tiefensuche beendet ihren Rundlauf, bei der Breitensuche fehlt nur noch Knoten »F«.

Die Tiefensuche besucht als Nächstes Knoten *C*. Bei der Breitensuche wird Knoten *F* besucht. Es wurden nun alle Knoten besucht, wie Sie in Abbildung 9.13 erkennen können.



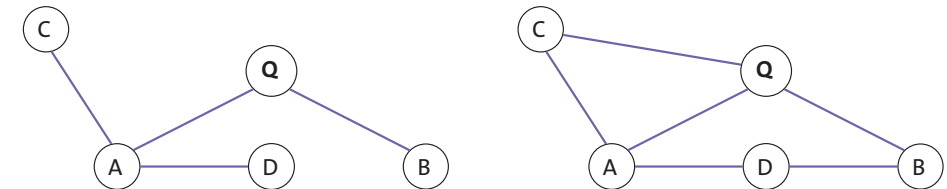
Besuchte Knoten: A, D, F, E, B, C  
Gesehene Knoten: B

Besuchte Knoten: A, B, C, D, E, F  
Gesehene Knoten: F

**Abbildung 9.13** Beide Suchverfahren haben nun alle Knoten besucht. Die verbleibenden gesehene Knoten sind alle schon besucht und werden daher verworfen.

Beide Suchverfahren arbeiten noch die restlichen Elemente in ihren Listen ab. Da diese Knoten aber alle schon besucht wurden, terminiert die Suche anschließend.

## Eigenschaften von Graphen



**Abbildung 9.14** Zwei Graphen, die die Wasserversorgung von vier Städten zeigen. Die Quelle (»Q«) ist genauso wie die Städte durch einen Knoten repräsentiert. Die Kanten zwischen den Knoten sind Wasserleitungen zur Versorgung der Städte.

Wasser in unseren Städten sehen wir als etwas Selbstverständliches an. Damit aber tatsächlich Wasser in jedem Haushalt aus den Leitungen kommt, muss das Wasser von einer Quelle zu den Städten transportiert werden. Abbildung 9.14 zeigt zwei Varianten, wie vier Städte mit der Quelle verbunden sind und somit mit Wasser versorgt werden. Vergleichen Sie die beiden Versorgungsnetzwerke. In welchem der beiden Netzwerke gibt es Probleme, die das andere Netzwerk behoben hat?

## Bäume und Zyklentreiheit

Die linke Variante ist sehr anfällig für Versorgungsausfälle. Muss eine der Wasserleitungen gewartet werden oder fällt aus anderen Gründen aus, so sind einige der Städte von der Wasserversorgung abgeschnitten. Rechts gibt es selbst beim Ausfall einer Leitung immer noch mindestens einen anderen Weg, auf dem das Wasser von der Quelle zu den Städten transportiert werden kann.

Der linke Graph ist ein Baum. Bäume haben Sie bereits in Kapitel 7, »Suchen«, als Suchbäume kennengelernt. Markant für Bäume ist, dass sie keine Kreise, auch *Zyklen* genannt, enthalten, also *zyklentrei* sind.

### Pfade und Kreise

In der Knebeli hatten wir bereits angesprochen, dass Knoten nicht nur direkt durch eine Kante verbunden sein können, sondern auch indirekt über mehrere Kanten. Eine solche Folge von Kanten, die zwei Knoten miteinander verbindet, nennt man *Pfad*. Bei-

spielsweise gibt es zwischen der Quelle und der Stadt  $D$  im rechten Teil von Abbildung 9.14 mehrere Pfade. Einer der Pfade verläuft über Knoten  $B$  und ist zwei Kanten lang. Ein anderer Pfad verläuft über die Knoten  $C$  und  $A$  und ist somit drei Kanten lang. Endet ein Pfad im selben Knoten, in dem er begonnen hat, so nennt man diesen Pfad auch *Kreis*. Ist der Graph *gerichtet*, können die Kanten auch nur in die eine Richtung verwendet werden.

Für ein Versorgungsnetzwerk ist es ungünstig, wenn das Netzwerk wie ein Baum aufgebaut ist. Viele Algorithmen funktionieren dagegen auf Bäumen besonders gut. Beispielsweise ist es sehr einfach, den längsten Pfad in einem Baum zu finden, während diese Aufgabe auf allgemeinen Graphen sehr schwer zu lösen ist.

## Zusammenhang

Wenn zwei Leitungen gleichzeitig ausfallen, garantiert jedoch keine der beiden Varianten aus Abbildung 9.14, dass alle Städte mit Wasser versorgt werden können, wie Abbildung 9.15 zeigt. In diesem Fall spricht man davon, dass der Graph nicht mehr *zusammenhängend* ist, da nicht mehr jeder Knoten von jedem anderen Knoten aus erreichbar ist. Stattdessen besteht er nun aus zwei Teilgraphen, die auch *Zusammenhangskomponenten* genannt werden. Diese Eigenschaft werden Sie später zur Lösung von Aufgabe 3 im Rahmen eines weiteren Beispiels aus der Praxis benötigen.

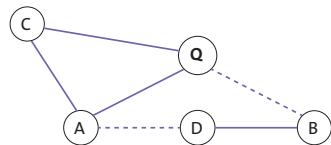


Abbildung 9.15 Zwei der Wasserleitungen sind ausgefallen. Die Städte »B« und »D« sind daher von der Versorgung abgeschnitten. Der Graph ist nun nicht mehr zusammenhängend.

## Eulersche Graphen

Damit die Wasserleitungen nicht ausfallen, müssen sie regelmäßig von einem Roboter gewartet werden. Ein solcher Roboter wird in die Wasserleitungen hineingelassen und fährt sie daraufhin alle ab. Damit die Wartung so kurz wie möglich dauert, soll der Roboter keine Leitung doppelt befahren müssen. Gibt es in dem Netzwerk einen Pfad, der jede Kante genau einmal enthält?

Glücklicherweise lässt sich diese Eigenschaft sehr leicht überprüfen. Ein solcher Pfad wird auch *Eulerweg* genannt und existiert genau dann, wenn der Graph zusammenhän-

gend ist und alle bis auf zwei Knoten einen geraden Knotengrad haben, also mit einer geraden Anzahl an Nachbarn verbunden sind. Die beiden anderen Knoten können entweder beide eine ungerade Anzahl an verbundenen Kanten haben oder beide eine gerade Anzahl.

Das Wasserversorgungsnetzwerk rechts in Abbildung 9.15 erfüllt diese Bedingung. Alle Knoten haben eine gerade Anzahl an verbundenen Kanten, bis auf die Knoten  $A$  und  $Q$ . Die Knoten  $A$  und  $Q$  sind daher die Start- und Endknoten des Pfades. Ein gültiger Eulerweg, bei dem jede Kante genau einmal befahren wird, wäre zum Beispiel:

$$A \rightarrow C \rightarrow Q \rightarrow A \rightarrow D \rightarrow B \rightarrow Q$$

Der Roboter kann also sehr schnell das ganze Netzwerk warten, weil er dabei keine Leitung doppelt befahren muss.

Übrigens: Wenn ein Eulerweg auf demselben Knoten endet, auf dem er gestartet ist, spricht man von einem *Eulerkreis*. Einen Eulerkreis kann es aber nur geben, wenn wirklich alle Knoten einen geraden Knotengrad haben. Zum Finden eines Eulerkreises benutzt man zum Beispiel eine erweiterte Tiefensuche. Ein Graph, in dem ein Eulerkreis existiert, wird auch als *eulerscher Graph* bezeichnet.

## Planarität

Das Netzwerk soll nun erweitert werden, um noch ausfallsicherer zu werden. Daher planen wir, die Stadt  $D$  auch direkt mit der Quelle  $Q$  zu verbinden und außerdem mit Direktleitungen  $A$  und  $B$ ,  $B$  und  $C$  sowie  $C$  und  $D$  zu verbinden. Das daraus resultierende Netzwerk ist in Abbildung 9.16 dargestellt.

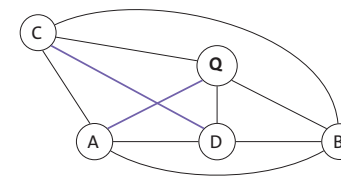


Abbildung 9.16 Im erweiterten Netzwerk überschneiden sich zwei Leitungen.

Nun gibt es in diesem neuen Plan ein Problem: Zwei Wasserleitungen überschneiden sich, und solche Überschneidungen bedeuten Zusatzaufwand. Versuchen Sie einmal, den Planern zu helfen und eine Anordnung derselben Kanten zu finden, die ohne Überschneidungen auskommt! Sie werden feststellen, dass dies nicht möglich ist. Der Graph in Abbildung 9.16 ist daher nicht *planar*, da nicht alle Kanten überschneidungsfrei gezeichnet werden können.

Für dieses Beispiel der Wasserversorgung bedeutet ein nicht planarer Graph, dass es nicht möglich ist, alle Leitungen zu verlegen, ohne dass sich zwei überschneiden. Aber auch in anderen Bereichen wie der Kartographie ist diese Eigenschaft wichtig. Ist ein Graph planar, lassen sich dessen Knoten mit vier verschiedenen Farben einfärben, ohne dass zwei benachbarte Knoten die gleiche Farbe haben. Das ist zum Beispiel nützlich bei der Gestaltung von Landkarten. Wählt man dort die Staaten als Knoten und verbindet benachbarte Staaten mittels Kanten, erhält man einen planaren Graphen und kann deshalb jede Landkarte mit nur vier Farben komplett einfärben.

## Zusammenfassung und Einordnung

In diesem Kapitel haben Sie Graphen kennengelernt, eine Datenstruktur, die sich sehr gut eignet, um Beziehungen zwischen Objekten darzustellen. Graphen können auf verschiedene Arten erweitert werden; wir haben im Speziellen gewichtete und gerichtete Graphen betrachtet. Außerdem haben Sie anhand eines Wassernetzwerks einige der wichtigsten Eigenschaften von Graphen näher untersucht. Mit der Tiefen- und der Breitensuche kennen Sie nun zwei Standardalgorithmen, mit denen Graphen durchsucht werden können. Diese beiden Algorithmen besuchen die Knoten eines Graphen in unterschiedlicher Reihenfolge, da sie verschiedene Datenstrukturen für die Verwaltung der noch abzuarbeitenden Knoten verwenden.

Die Anwendungsfälle für Graphen sind breit gefächert. Beispielsweise speichern Navigationsgeräte alle Städte und Straßen als Graphen ab. Dank fortgeschrittener Algorithmen, wie zum Beispiel dem Algorithmus von Dijkstra zum Finden kürzester Wege, ist es dann möglich, Ihnen beim Autofahren die kürzeste Route vorzuschlagen. Der Algorithmus von Dijkstra basiert genauso wie viele andere Algorithmen im Bereich der Graphen auf den vorgestellten Suchverfahren. Auch in der Softwareentwicklung müssen regelmäßig Graphenprobleme gelöst werden. So lassen sich die Abhängigkeiten eines Software-Projektes genauso wie die Abhängigkeiten der Kleidungsstücke in der Knobelei darstellen – das trifft im Übrigen auf jeden Prozess zu. Um herauszufinden, in welcher Reihenfolge Aufgaben erledigt werden müssen, können Sie genau wie in der Knobelei die topologische Sortierung verwenden.



### Aufgabe 1: Eigenschaften von Graphen

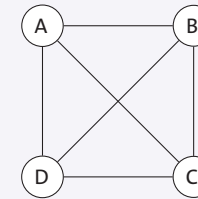
Schauen Sie sich die vier Graphen in Abbildung 9.17 an. Welche der folgenden Eigenschaften haben die Graphen?

- a) Ist der Graph planar?
- b) Ist der Graph zyklensfrei?

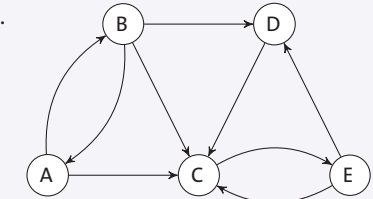
Sollte der Graph ungerichtet sein:

- c) Lässt sich ein Eulerweg oder ein Eulerkreis finden?
- d) Ist er zusammenhängend?

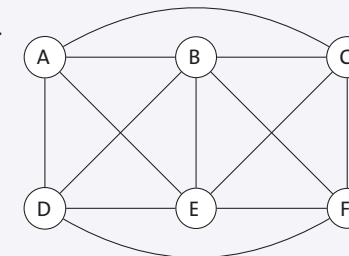
1.



3.



2.



4.

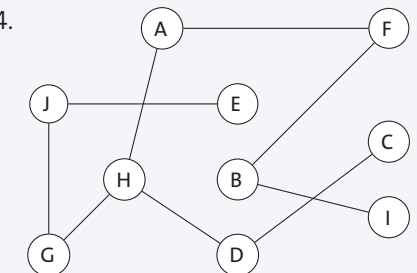


Abbildung 9.17 Welche Eigenschaften haben diese vier Graphen?



### Aufgabe 2: Suchen in Graphen

Betrachten Sie erneut den Graphen der Freundschaften der Schulklasse aus Abbildung 9.6. Können Sie die folgenden Fragen zu diesem Graphen beantworten?

- a) Wie viele Schritte sind mit der Tiefensuche mindestens notwendig, um von Mandy zu Dominik zu finden? Wie viele Schritte dauert es mit der Breitensuche mindestens?



b) Lässt sich mit diesen Aussagen eine generelle Aussage darüber treffen, welche der beiden Suchen schneller den gewünschten Knoten findet?



### Aufgabe 3: Graphmodellierung

Brynay ist ein kleines Land weit entfernt von Deutschland. In Brynay gibt es zehn Städte, die mit einem Straßennetz verbunden sind. Die Straßen in Brynay sind in der Regel in beide Richtungen befahrbar, außer einige Straßen in den Bergen und sobald Baustellen auftreten.

Ganz im Westen liegt die Stadt Cerer. Sie ist über eine 90 km lange Straße mit der etwas nördlicher liegenden Stadt Omyr verbunden. Von Omyr führt eine 140 km lange Straße in die Berge nach Hatan. Da diese Straße so schmal ist, kann man sie nur von Omyr nach Hatan befahren. Von Hatan kommt man, ebenfalls nur über eine einseitig befahrbare, 60 km lange Straße, nach Taisul. Taisul ist außerdem über eine 75 km lange Straße mit Omyr verbunden.

Die Stadt Taisul ist über eine 300 km lange Straße mit der Stadt Nallar verbunden. Nallar hat eine 172 km lange Anbindung an die sehr zentral liegende Stadt Dium. Von Dium führt eine 180 km lange Straße in den Norden nach Omyr und eine 200 km lange Straße in den Westen nach Ustria. Auf der Strecke nach Omyr befindet sich eine Baustelle, weshalb man nur von Omyr nach Dium fahren kann. Normalerweise kommt man auch über eine 112 km lange Straße von Ustria nach Cerer, jedoch gibt es dort aktuell ebenfalls eine Baustelle, weshalb man zurzeit nur von Cerer nach Ustria fahren kann.

Von Ustria geht es in die südliche Küstenregion, und zwar zur 81 km entfernten Stadt Baw. 108 Kilometer weiter im Osten liegt Ritson, das über eine Straße von Baw zu erreichen ist. Aus Ritson heraus führt wiederum eine Straße nach Nallar, die 175 Kilometer lang ist, die jedoch gerade aufgrund einer Baustelle nur von Nallar nach Ritson befahrbar ist. Ritson ist außerdem die einzige Möglichkeit, zum etwas nördlicher liegenden Isot zu kommen; die Strecke ist 90 Kilometer lang.

Bitte entwerfen Sie einen Stadtplan von Brynay. Zeichnen Sie in diesen außerdem die Distanzen zwischen den benachbarten Städten. Versuchen Sie, mithilfe dieses Planes die folgenden Fragen zu beantworten:

- Ist der Stadtplan zusammenhängend und planar?
- Wie weit ist die kürzeste Strecke von Hatan nach Isot? Ist es von Isot nach Hatan genau so weit?
- Angenommen, zwei Autos fahren gleichzeitig mit der gleichen Geschwindigkeit in Ritson los. Das eine Auto möchte nach Omyr, das andere nach Taisul. Welches der Autos kommt eher an seinem Ziel an?

d) Welche Straßen dürfen nicht durch Bauarbeiten vollständig blockiert werden, weil der Graph dann nicht mehr zusammenhängend wäre und deshalb Städte nicht mehr erreicht werden könnten?

## Lösungen

### Aufgabe 1: Eigenschaften von Graphen

Tabelle 9.1 zeigt, welche Eigenschaften welcher Graph erfüllt. Je nachdem, ob der Graph gerichtet oder ungerichtet ist, wurden bestimmte Zusammenhangseigenschaften ausgelassen.

Eigenschaft	Graph 1	Graph 2	Graph 3	Graph 4
Planarität	✓	×	✓	✓
Zyklenfreiheit	×	×	×	✓
Zusammenhang	✓	✓		✓
Eulerweg	×	✓		×
Eulerkreis	×	×		×

Tabelle 9.1 Die Eigenschaften der Graphen. Nicht relevante Eigenschaften sind ausgelassen.

Wir möchten gerne einige Erklärungen zum Ergebnis hinzufügen:

- Der erste Graph ist planar, weil man eine der Kanten auch außen herum zeichnen kann. Somit überschneiden sich keine Kanten mehr. Da alle Knoten eine ungerade Anzahl an Kanten haben, lässt sich kein Eulerweg und somit auch kein Eulerkreis finden.
- Aufgrund der beiden langen Kanten ist der zweite Graph nicht mehr planar. Da zwei Knoten eine ungerade Anzahl an Kanten haben, lässt sich zwar ein Eulerweg, aber kein Eulerkreis finden.
- Der dritte Graph ist gerichtet, und da wir in diesem Buch nicht gezeigt haben, wie die drei Eigenschaften Eulerweg, Eulerkreis und Zusammenhang auf gerichteten Graphen definiert sind, haben wir diese Zellen leer gelassen. Diese Eigenschaften lassen sich aber auch für gerichtete Graphen definieren, wie Sie auf der Website zum Buch nachlesen können.

- d) Der vierte Graph ist ein Baum, da er zusammenhängend und zyklensfrei ist. Er ist zwar in diesem Fall nicht planar gezeichnet, da es sich jedoch um einen Baum handelt, ist dies einfach möglich.

### Aufgabe 2: Suchen in Graphen

a) Die Tiefensuche benötigt mindestens 3 Schritte. Dieser Fall tritt ein, wenn die Speicherung der Knoten so aussieht, dass die Tiefensuche im ersten Schritt Frank besucht, anschließend Jennifer und am Ende Dominik. Die Breitensuche hingegen braucht mindestens 12 Schritte. Auch dieser Fall ist abhängig von der Speicherung der Knoten. Im ersten Schritt muss die Breitensuche Frank besuchen. Damit werden dessen Nachbarn der gesehenen Liste hinzugefügt, wobei Jennifer die Erste sein muss, die dieser Liste hinzugefügt wird. Jennifer wird jedoch erst besucht, wenn alle weiteren direkten Nachbarn von Mandy besucht wurden. Wenn Jennifer besucht wird, muss ebenfalls Dominik als Erster der Liste der gesehenen Knoten hinzugefügt werden. Auch in diesem Fall müssen aber erst alle anderen Nachbarn von Frank abgearbeitet werden.

b) Auch wenn in diesem Beispiel die Tiefensuche wesentlich weniger Schritte benötigt hat als die Breitensuche, lässt sich daraus nicht generell schließen, dass die Tiefensuche einen Knoten schneller findet als die Breitensuche. Ein einfaches Gegenbeispiel zeigt dies: Angenommen, die Speicherung ist genauso wie vorhin, und die Tiefensuche besucht nach Frank die Knoten Jennifer und Dominik, gesucht wird aber ein anderer direkter Nachbar von Mandy, zum Beispiel Thomas. Dann ist die Tiefensuche bereits ganz tief im Graphen, während der gesuchte Knoten recht nah an Mandy liegt und von der Breitensuche schon im zweiten Schritt gefunden werden kann.

Generell trifft man im Vergleich der Tiefen- zur Breitensuche die Aussage, dass die Tiefensuche einen Knoten vermutlich schneller besucht, wenn dieser weit vom Startknoten entfernt liegt, während die Breitensuche eher naheliegende Knoten schneller findet. Letztendlich hängt aber die genaue Anzahl der Schritte von der Speicherung des Graphen und somit von der Reihenfolge, in der die Knoten gesehen und besucht werden, ab.

### Aufgabe 3: Graphmodellierung

In Abbildung 9.18 sehen Sie den Stadtplan von Brynay. Ebenfalls darin eingezeichnet sind die Distanzen zwischen den Städten. Die Baustellen und nur in eine Richtung befahrbaren Straßen werden mit gerichteten Kanten dargestellt. Dagegen stehen ungerichtete Kanten stellvertretend für beidseitig befahrbare Straßen und somit für zwei gerichtete Kanten. Wir können diese beiden Kanten typen mischen, um den Graphen übersichtlicher zu gestalten.

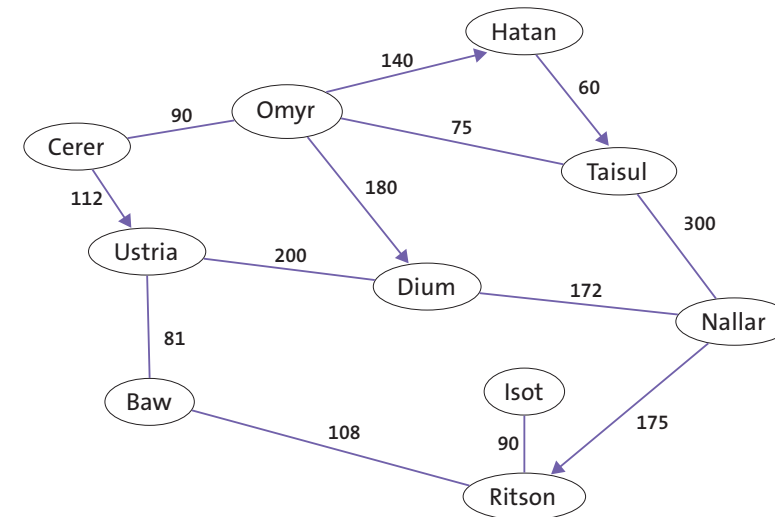


Abbildung 9.18 Der Stadtplan von Brynay

- a) Der Graph der Städte ist zusammenhängend, da von jeder Stadt aus jede andere Stadt erreicht werden kann. Selbst die entlegenen Städte wie Hatan oder Isot kann man sowohl erreichen als auch wieder verlassen. Da es viele beidseitig befahrbare Straßen gibt, ist es einfach, den Zusammenhang festzustellen. Außerdem ist dieser Graph bereits planar gezeichnet.
- b) Die kürzeste Strecke von Hatan nach Isot ist 616 Kilometer lang. Sie führt über Taisul, Omyr, Cerer, Ustria, Baw und Ritson. Aufgrund einiger Baustellen ist die kürzeste Strecke in die andere Richtung leider 1166 Kilometer lang. Sie führt von Isot über Ritson, Baw, Ustria, Dium, Nallar, Taisul und Omyr nach Hatan.
- c) Das Auto nach Taisul ist eher am Ziel als das Auto nach Omyr. Das Auto nach Omyr muss sogar über Taisul fahren. Wäre eine der Baustellen zwischen Cerer und Ustria beziehungsweise zwischen Omyr und Dium fertig, wäre das Auto nach Omyr schneller am Ziel. Für die Baustelle zwischen Nallar und Ritson gilt das Gegenteil. Das Auto nach Omyr würde nicht schneller am Ziel ankommen.
- d) Die Straßen, die Hatan und Isot mit den anderen Städten verbinden, dürfen nicht bebaut werden, da diese Städte sonst nicht mehr erreichbar wären oder man von Hatan nicht mehr wegkönnte. Bei der aktuellen Baustellensituation gibt es aber noch mehr kritische Straßen: Käme beispielsweise eine Baustelle auf der Strecke von Taisul nach Omyr (in dieser Richtung) hinzu, könnte man aus den meisten Städten nicht mehr nach Omyr und Cerer fahren. Umgekehrt darf auch die Strecke von Baw nach Ustria nicht blockiert werden, weil man dann aus Baw, Ritson und Isot nicht mehr den Rest des Landes erreichen könnte.

## Auf einen Blick

1	Algorithmen .....	26
2	Zahlen und Kodierungen .....	48
3	Datenstrukturen .....	64
4	Einfache Sortieralgorithmen .....	80
5	Komplexität .....	100
6	Effizientere Sortieralgorithmen .....	118
7	Suchen .....	140
8	Backtracking und Dynamische Programmierung .....	160
9	Graphen .....	172
10	Formale Sprachen .....	192
11	Objektorientierung .....	214
12	Datenbanken .....	232
13	Computer .....	260
14	Netzwerke .....	280
15	Verschlüsselung .....	298
16	Softwareentwicklung .....	310
17	Teamarbeit .....	320
18	Fehler .....	334
19	Hands-on: Programmieren mit Java .....	350
20	Ethik in der Informatik .....	378
21	Extro .....	396

# Inhalt

Geleitwort .....	15
Intro .....	17

## 1 Algorithmen ..... 26

<b>Was ist ein Algorithmus?</b> .....	28
<b>Wie wird ein Algorithmus notiert?</b> .....	30
Graphische Notation .....	31
Pseudocode .....	32
<b>Schleifen</b> .....	33
<b>Verzweigungen</b> .....	34
<b>Logische Aussagen</b> .....	37
Logisches NICHT .....	38
Logisches UND .....	38
Logisches ODER .....	39
Klammerung und Vorrangsregeln .....	39
Besondere Aussagen .....	40
<b>Funktionen</b> .....	41
<b>Zusammenfassung und Einordnung</b> .....	42

## 2 Zahlen und Kodierungen ..... 48

<b>Zahlensysteme und Einheiten</b> .....	50
Rechnen im Binärsystem .....	51
Einheiten .....	52
<b>Kodierungen</b> .....	54
Natürliche Zahlen .....	54
Ganze Zahlen .....	55
Kommazahlen .....	56



Text .....	57
Bilder .....	58
<b>Zusammenfassung und Einordnung .....</b>	<b>60</b>

### 3 Datenstrukturen ..... 64

<b>Geordnete Daten .....</b>	<b>65</b>
Repräsentation im Speicher .....	69
Andere Operationen auf den Datenstrukturen .....	71
<b>Ungeordnete Daten .....</b>	<b>72</b>
<b>Datenzuordnungen .....</b>	<b>74</b>
<b>Zusammenfassung und Einordnung .....</b>	<b>75</b>

### 4 Einfache Sortieralgorithmen ..... 80

<b>Selection Sort .....</b>	<b>82</b>
<b>Insertion Sort .....</b>	<b>87</b>
<b>Bubble Sort .....</b>	<b>89</b>
<b>Ordnungen .....</b>	<b>92</b>
<b>Zusammenfassung und Einordnung .....</b>	<b>93</b>

### 5 Komplexität ..... 100

<b>Verschiedene Wege führen zum Ziel .....</b>	<b>102</b>
<b>Eingabegröße .....</b>	<b>103</b>
<b>Messen der Laufzeit .....</b>	<b>104</b>
<b>Berechnen der Laufzeit .....</b>	<b>104</b>
<b>Die Landau-Notation .....</b>	<b>107</b>
<b>Typische Laufzeiten .....</b>	<b>110</b>
<b>Zusammenfassung und Einordnung .....</b>	<b>112</b>

### 6 Effizientere Sortieralgorithmen ..... 118

<b>Merge Sort .....</b>	<b>119</b>
<b>Quick Sort .....</b>	<b>124</b>
<b>Rekursion und Divide and Conquer .....</b>	<b>127</b>
<b>Noch schneller sortieren .....</b>	<b>130</b>
<b>Zusammenfassung und Einordnung .....</b>	<b>132</b>

### 7 Suchen ..... 140

<b>Lineare Suche .....</b>	<b>141</b>
<b>Binäre Suche .....</b>	<b>144</b>
<b>Suchbäume .....</b>	<b>147</b>
Suchen in Suchbäumen .....	148
Hinzufügen eines Elements .....	150
Erstellen von Suchbäumen .....	151
Balancierte Bäume .....	152
<b>Zusammenfassung und Einordnung .....</b>	<b>154</b>

### 8 Backtracking und Dynamische Programmierung ..... 160

<b>Die perfekte Kiste .....</b>	<b>161</b>
<b>Branch and Bound .....</b>	<b>163</b>
<b>Dynamische Programmierung .....</b>	<b>164</b>
<b>Zusammenfassung und Einordnung .....</b>	<b>166</b>

### 9 Graphen ..... 172

<b>Verknüpfte Daten .....</b>	<b>174</b>
<b>Varianten von Graphen .....</b>	<b>175</b>
Gerichtete Kanten .....	175

Gewichtete Kanten .....	176
Beispiele für Graphen .....	177
<b>Suchen und Bewegen in Graphen .....</b>	<b>178</b>
Implementierung .....	179
Beispiel .....	180
<b>Eigenschaften von Graphen .....</b>	<b>183</b>
Bäume und Zyklenfreiheit .....	183
Zusammenhang .....	184
Eulersche Graphen .....	184
Planarität .....	185
<b>Zusammenfassung und Einordnung .....</b>	<b>186</b>

## 10 Formale Sprachen ..... 192

<b>Grammatiken .....</b>	<b>194</b>
Reguläre Grammatiken .....	196
Kontextfreie Grammatiken .....	197
Höhere Grammatiken .....	197
<b>Automaten .....</b>	<b>198</b>
Endliche Automaten .....	198
Höhere Automaten .....	201
<b>Sprachen und Mengenoperationen .....</b>	<b>202</b>
<b>Reguläre Ausdrücke .....</b>	<b>205</b>
<b>Zusammenfassung und Einordnung .....</b>	<b>207</b>

## 11 Objektorientierung ..... 214

<b>Objekte und Klassen .....</b>	<b>216</b>
<b>Objektorientierte Programmierung .....</b>	<b>217</b>
<b>Vererbung .....</b>	<b>218</b>
<b>Abstrakte Klassen .....</b>	<b>219</b>
<b>Sichtbarkeiten .....</b>	<b>221</b>
<b>Zusammenfassung und Einordnung .....</b>	<b>223</b>

## 12 Datenbanken ..... 232

<b>Strukturierte Datenspeicherung .....</b>	<b>235</b>
Grundbegriffe .....	236
Darstellung .....	237
Kardinalitäten .....	238
Schlüssel .....	239
<b>Operationen auf Datenbanken .....</b>	<b>240</b>
Daten abfragen und sortieren .....	240
Gruppierung von Daten .....	244
Einträge einfügen .....	246
Einträge modifizieren .....	246
Einträge löschen .....	246
<b>Empfohlene Strukturierung von Daten .....</b>	<b>247</b>
Ein Wert pro Zelle .....	248
Redundanzen vermeiden .....	250
<b>Zusammenfassung und Einordnung .....</b>	<b>251</b>

## 13 Computer ..... 260

<b>Logische Schaltungen .....</b>	<b>262</b>
Die Knobelei als Schaltplan .....	263
Exklusives Oder .....	264
Algorithmen als logische Schaltungen .....	264
<b>Hardware-Komponenten und ihr Zusammenspiel .....</b>	<b>265</b>
<b>Betriebssysteme .....</b>	<b>268</b>
Kernfunktionen von Betriebssystemen .....	268
Verbreitete Betriebssysteme .....	271
Betriebssystemnahe Programmierung .....	271
<b>Betriebssystemunabhängigkeit .....</b>	<b>272</b>
Interpreter .....	273
Bytecode-Sprachen .....	273
<b>Virtuelle Computer .....</b>	<b>274</b>
<b>Zusammenfassung und Einordnung .....</b>	<b>275</b>

## 14 Netzwerke ..... 280

Eine mögliche Lösung für die Poststelle .....	282
<b>Netzwerke</b> .....	283
Clients und Server .....	283
Weitere Netzwerkgeräte .....	285
<b>Internetstruktur</b> .....	286
Services im Internet .....	288
Daten im Internet versenden .....	288
Adressauflösung zum Finden der IP-Adresse .....	289
<b>Einheitliche Kommunikation</b> .....	290
Eine HTTP-Anfrage .....	290
Die Antwort des Webservers .....	291
Die Anfrage zusätzlicher Ressourcen .....	292
<b>Zusammenfassung und Einordnung</b> .....	293

## 15 Verschlüsselung ..... 298

Warum verschlüsseln? .....	300
<b>Symmetrische Verschlüsselung</b> .....	300
<b>Asymmetrische Verschlüsselung</b> .....	302
<b>Hybridverfahren</b> .....	304
<b>Verschlüsselungen knacken</b> .....	304
<b>Zusammenfassung und Einordnung</b> .....	307

## 16 Softwareentwicklung ..... 310

<b>Algorithmus vs. Software</b> .....	311
<b>Die Werkzeuge eines Softwareentwicklers</b> .....	313
<b>Große Probleme lösen</b> .....	315
Top-down-Methode .....	315
Bottom-up-Methode .....	316
<b>Zusammenfassung und Einordnung</b> .....	318

## 17 Teamarbeit ..... 320

<b>Warum Teams?</b> .....	322
<b>Softwareentwicklung im Team</b> .....	322
<b>Kommunikation in Teams</b> .....	323
<b>Aufgabenverwaltung und Kommunikationswerkzeuge</b> .....	325
<b>Versionsverwaltung</b> .....	325
Änderungen kleinschrittig speichern .....	326
Daten mit einem Server synchronisieren .....	326
Mit anderen Entwicklern zusammenarbeiten .....	327
Verschiedene Entwicklungszweige verfolgen .....	329
<b>Zusammenfassung und Einordnung</b> .....	330

## 18 Fehler ..... 334

<b>Warum ist Software fehlerhaft?</b> .....	336
<b>Bugs</b> .....	337
<b>Verschiedene Fehlerarten</b> .....	337
Kompilierungsfehler .....	337
Laufzeitfehler .....	338
Logische Fehler .....	339
Designfehler .....	340
Umgebungsfehler .....	342
Kommunikationsfehler .....	342
<b>Techniken zur Fehlervermeidung</b> .....	343
Testen .....	343
A/B Testing .....	344
Programmierstil .....	345
Pair Programming .....	345
Code Review .....	346
<b>Zusammenfassung und Einordnung</b> .....	346

## 19 Hands-on: Programmieren mit Java ..... 350

<b>Die Programmiersprache Java</b> .....	351
<b>Hallo Leser</b> .....	352
Ausführung .....	352
Erklärung des Programmcodes .....	353
<b>Variablen</b> .....	354
<b>Klassen, Objekte und Methoden</b> .....	355
Eigenschaften von Objekten .....	356
Verhalten von Objekten .....	356
<b>Datentypen</b> .....	359
Primitive Datentypen vs. Objektdatentypen .....	359
Zahlen .....	360
Wahrheitswerte .....	361
Zeichen und Zeichenketten .....	361
(Dynamische) Arrays .....	362
Listen, Queues und Stacks .....	364
Sets und Maps .....	365
<b>Kontrollstrukturen</b> .....	366
Verzweigungen .....	366
Schleifen .....	367
<b>Fehlersuche</b> .....	369
<b>Eine kleine Werkzeugkiste</b> .....	371

## 20 Ethik in der Informatik ..... 378

<b>Recht und Ordnung</b> .....	379
Software für den Überwachungsstaat .....	380
Die Hutfarben der Hacker .....	381
<b>Informatik in der Wirtschaft</b> .....	381
Automatisierung statt Arbeitsplatz .....	382
Netzneutralität .....	383
<b>Der Wert persönlicher Daten</b> .....	384

<b>Gemeingüter und Open Source</b> .....	385
Wissen für jedermann .....	386
Kostenlose und quelloffene Software .....	386
Probleme der Anarchie .....	387
<b>Verantwortung für Technologie</b> .....	388
Parteiische Informationen .....	389
Vermeidbare Fehlfunktionen .....	390
Unvermeidbare Folgen .....	391
<b>IT-Gerechtigkeit</b> .....	392
<b>Der technisierte Mensch</b> .....	393
Abhängigkeit von Technik .....	393
Arbeitszeit: 24/7 .....	394
<b>Zusammenfassung und Einordnung</b> .....	394

## 21 Extro ..... 396

<b>Wie wird man Informatiker?</b> .....	397
<b>Ressourcen</b> .....	402
<b>Wie geht es weiter?</b> .....	403
Index .....	405



## Alle Knobeleyen auf einen Blick

Wo ist der Ausgang des Labyrinths? .....	27
Gib mir 31! .....	49
Speicherung gleicher Daten .....	65
Bücher sortieren .....	81
Schokolade aufteilen .....	101
Sortieren im Team .....	119
Finden und Sortieren .....	141
Das Kistenproblem .....	161
Morgendliches Anziehen .....	173
Sätze erzeugen .....	193
Das Vereinsfest .....	215
Max' Lieblingsfilme .....	233
Addieren auf Hardware-Ebene .....	261
Die Post des Kanzleramts .....	281
Fdhvdu .....	299
Konflikte .....	321
Auf Fehlersuche .....	335

# Index

## A

A/B-Test .....	344
Abfragesprache .....	236
Abstrakte Klasse .....	219
Abstrakte Methode .....	219
Abstraktion .....	217
Addition .....	30
Adressauflösung .....	289
AES .....	301
Akzeptierender Zustand → Endzustand	
Algorithmus .....	29
<i>ausführen</i> .....	28
Allgemeine Grammatik .....	198
Android .....	271
Anfrage (Netzwerk) .....	284
Angewandte Informatik .....	398
Antisymmetrie .....	93
Apache License .....	387
Arbeitsspeicher .....	266
Architektur .....	312
Arithmetischer Ausdruck .....	197
Array .....	65
<i>dynamisches</i> .....	70
ASCII .....	57
Assemblercode .....	267
Asymmetrische Verschlüsselung .....	302
Asymptotische Schranke .....	109
Attribut .....	216, 236, 356
Ausdruck, arithmetischer .....	197
Ausdruck, regulärer → Regulärer Ausdruck	
Ausführungsumgebung .....	274
Ausgabe .....	22
Ausgabedaten .....	28
Aussage, logische .....	37
Authentizität .....	303
Automat	
<i>endlicher</i> .....	198
<i>linear beschränkter</i> .....	201
Automatenmodell .....	198
Average Case .....	104
AVL-Baum .....	153

## B

Bachelorstudium .....	397
Backtracking .....	163
Baum .....	147
Baum-Topologie .....	296
Befehl .....	28
Befehlssatz .....	29
Benachbart → Nachbarschaft	
Berechenbarkeitstheorie .....	23
Berechnung .....	22
Best Case .....	89, 104
Best-Effort-Routing .....	383
Betriebssystem .....	268
Bibliothek .....	314
Binärbaum .....	147
Binäre Suche .....	144, 154
Binärsystem .....	50
Bindestrich-Informatik .....	398
Bit .....	50
Black Box .....	128
Black Hat Hacker .....	381
Blatt .....	147
Block .....	33
Bottom-up-Methode .....	316
Branch .....	329
Breitensuche .....	179
Brute-Force-Methode .....	305
Bubble Sort .....	89
Bug .....	337
Bus .....	266
Bus Factor .....	323
Bus-Topologie .....	296
Byte .....	52
Bytecode .....	274

## C

C .....	271
C# .....	274
Caesar-Verschlüsselung .....	299

Chomsky-Hierarchie .....	207
Church-Turing-These .....	202
Client .....	284
Client-Server-Kommunikation .....	284
Clique .....	178
Code Review .....	346
Code-Block .....	354
Coding Convention .....	345
Commit .....	326
Compiler .....	23, 272
Computer, virtueller .....	274
Content-Type .....	291
Counting Sort .....	131
CPU .....	265
Creative Commons .....	386

## D

Datei .....	270
Dateikopf .....	58
Dateiverwaltung .....	270
Daten, geordnete .....	65
Datenbank .....	235
<i>relationale</i> .....	236
Datentyp, primitiver .....	359
Debugger .....	313
Debugging .....	337
Definition .....	219
Deklaration .....	219
DES .....	301
Designfehler .....	340
Dezimalsystem .....	50
Digitale Revolution .....	379
Digitales Zeitalter .....	379
Distribution .....	271
Divide and Conquer → Teile und Herrsche	
Division .....	30
DNS .....	289
DNS-Server .....	289
Dokumentation .....	315
Domain Name Service → DNS	
Doppelt verkettete Liste .....	70
Dynamische Programmierung .....	166
Dynamisches Array .....	70

## E

Eclipse .....	371
Eingabe .....	22
Eingabealphabet .....	199
Eingabedaten .....	28
Eingabegröße .....	86, 103
Einloggen .....	270
Einwegfunktion .....	303
Elternklasse .....	218
Emoji .....	58
Encoding .....	54
Endlicher Automat .....	198
Endzustand .....	199
Enigma .....	306
Entität .....	236
Entitätstyp .....	236
Entity-Relationship-Diagramm .....	237
Entwicklungsumgebung .....	313
Entziffern .....	304
ER-Diagramm → Entity-Relationship- Diagramm .....	
Eulerkreis .....	185
Eulerscher Graph .....	185
Eulerweg .....	184
Exception .....	370
Exponent .....	56
Exponentialdarstellung .....	56

## F

Falsch .....	37
Farbpalette .....	59
Feed .....	389
Fehler, logischer .....	339
Fehlertolerantes System .....	342
Fehlerwirkung .....	337
Fehlerzustand .....	337
Fehlhandlung .....	337
Festkommadarstellung .....	56
Festplatte .....	270
FIFO .....	68
Filterblase .....	389
Firewall .....	285

Freeware .....	386
Fremdschlüssel .....	239
Funktion .....	41

## G

Ganze Zahl .....	54
Gatter .....	262
Geheimtext .....	299
Geordnete Daten .....	65
Gerichtete Kante .....	175
Getter .....	222
Git .....	331
GNU General Public License .....	387
Grad .....	178
Grammatik .....	
<i>allgemeine</i> .....	198
<i>kontextsensitive</i> .....	197
<i>reguläre</i> .....	196
Groß-O-Notation → Landau-Notation	
Gruppieren .....	244
GUI .....	312

## H

Hack .....	381
Hackerethik .....	381
Halbaddierer .....	263
Halteproblem .....	346
Hashfunktion .....	303
Häufigkeitsanalyse .....	305
Header .....	291
Hexadezimalsystem .....	51
Höhe .....	147
Höhere Programmiersprache .....	272
HTML .....	291
HTTP .....	290
HTTPS .....	290
Hybridverfahren .....	304
Hypertext Markup Language → HTML	
Hypertext Transfer Protocol → HTTP	

## I

IDE .....	313
Identität .....	216
Implementieren .....	30
In place .....	92
Index .....	65
Industrie 4.0 .....	382
Informatik .....	
<i>angewandte</i> .....	398
<i>praktische</i> .....	398
<i>technische</i> .....	398
<i>theoretische</i> .....	397
Initialisierung .....	69
Insertion Sort .....	87
Instanz .....	216
Integrationstest .....	344
Integrierter Schaltkreis .....	262
Integrität .....	303
Internet Protocol → IP	
Interpreter .....	273
IP .....	291
IP-Adresse .....	288
IPv4 .....	297
IPv6 .....	297
ISP → Provider	
Issue .....	325
Issue-Tracking .....	325
Ist-ein-Beziehung .....	218
Iteration .....	33

## J

Java .....	274, 351
Java Development Kit → JDK	
JavaScript .....	273
JDK .....	351
Just-in-Time Compiler .....	273

## K

Kante .....	147, 175
<i>gerichtete</i> .....	175

Kantengewicht	176
Kantenkosten → Kantengewicht	
Kapselung	221
Kardinalität	237
Kellerautomat	201
Kindklasse	218
Kindknoten	147
Klartext	299
Klasse	216
<i>abstrakte</i>	219
Kleene'scher Abschluss	205
Klonen	327
Knoten	147, 175
Kodierung	54
Kollision	303
Kommandozeile → Terminal	
Kommunikationsfehler	342
Kompilat	272
Kompilieren	272
Kompilierungsfehler	337
Komplexität	86
Konstruktor	358
Kontext	197
Kontextfreie Sprache	197
Kontextsensitive Grammatik	197
Kontradiktion	41
Korrektheit	102
Kreis	184
Kryptographie	300
Kurzschlussauswertung	122
Lineare Suche	142, 154
Linux	271
Liste	
<i>doppelt verkettete</i>	70
<i>verkettete</i>	66
Logische Aussage	37
Logische Schaltung	262
Logischer Fehler	339
Lokale Variable	356

## M

Mantisse	56
<i>Normalisierung</i>	56
Map	74, 366
Maschinencode	267
Masterstudium	398
Median	127
Menge	72
<i>Relation</i>	74
Merge	122, 328
Merge Conflict	328
Merge Sort	328
Messaging-System	325
Metadaten	385
Methode	216, 356
<i>abstrakte</i>	219
MIT License	387
Modell	217
Modellierung	217
Modulo	30
Multiplikation	30

## N

Nachbarschaft	175
Nachbedingung	30
Natürliche Sprache	30
Natürliche Zahl	54
Nebenläufigkeit	269
Netzneutralität	383
Netzwerk	283
Netzwerkschnittstelle	283
Nichtterminal	194

## L

Landau-Notation	108
Laufvariable	34
Laufzeit	102, 273
<i>exponentielle</i>	111
<i>konstante</i>	111
<i>lineare</i>	111
<i>logarithmische</i>	111
<i>quadratische</i>	111
Laufzeitfehler	338
Leere Summe	85
Leeres Wort	195
LIFO	67
Linear beschränkter Automat	201

Normalisierung	252
Normalisierung der Mantisse	56
Notation, wissenschaftliche	56
Null-Wert	69, 162
Nutzerverwaltung	271

## O

Objekt	216
Objekttyp	359
Objektorientierte Programmierung	217
One-Time-Pad	306
Open Source	271, 386
Operation	236
Ordnerhierarchie	270
Ordnung	81, 92
Ordnungsrelation	92–93
Out of place	92
Overflow	55

## P

Pair Programming	323, 345
Paket	283
Paketverwaltung	314
Pfad	147, 183
PHP	273
Pivotelement	124
Pixel	58
Planar	185
Plattformunabhängigkeit	273
Pledge-Algorithmus	45
Polymorphie	219
Port	284
Prädikat	109
Pragmatismus	217
Praktische Informatik	398
Primärschlüssel	239
Primitiver Datentyp	359
private	221
Produktionsregel	194
Programmablaufplan	30–31
Programmiersprache	30
<i>höhere</i>	272

Programmierung	
<i>dynamische</i>	166
<i>objektorientierte</i>	217
<i>prozedurale</i>	217
Programmzähler	266
Protokoll	290
Provider	286
Prozedurale Programmierung	217
Prozess	269
Prozessor	262
Prozessorcache	267
Prüfsumme	303
Pseudocode	30
public	221
pull	327
Pull Request	330
Punkt-Operator	356
push	327
Python	273

## Q

Quantencomputer	307
Quantor	109
Queue	67, 364

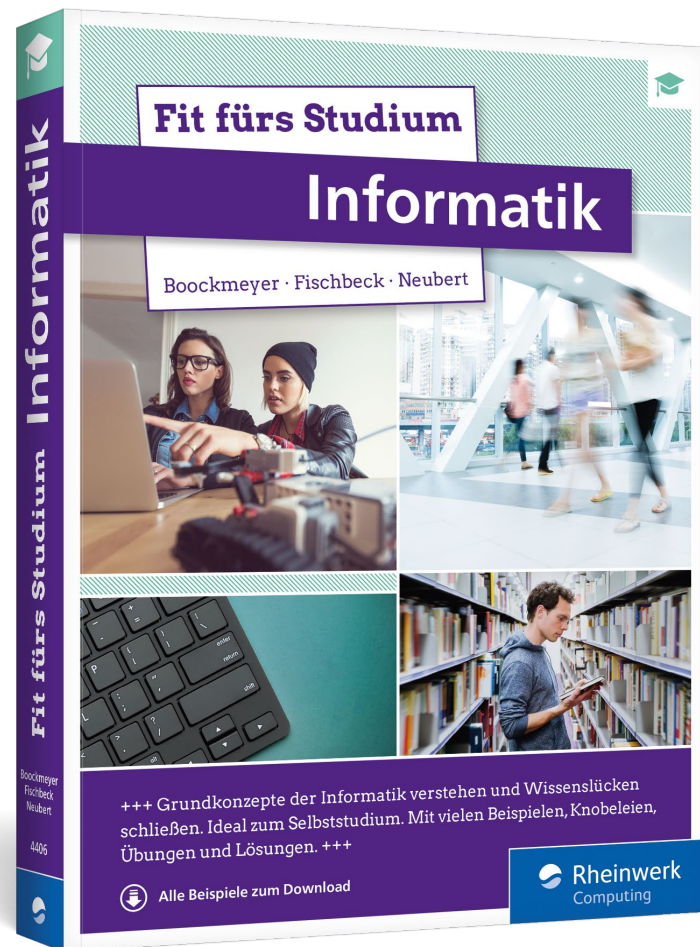
## R

RAM → Arbeitsspeicher	
Rastergrafik	58
Rationale Zahl	54
Rechenwerk	265
Rechteverwaltung	271
Reelle Zahl	54
Reflexivität	93
Register	267
Reguläre Grammatik	196
Reguläre Sprache	196
Regulärer Ausdruck	205
Rekursion	118, 120, 127
Rekursionstiefe	122
Relation	236
Relationale Datenbank	236
Repository	314, 326



Ressource .....	291
Ring-Topologie .....	295
Router .....	285
RSA-Verfahren .....	302
<b>S</b>	
Scheduling .....	269
Schleife .....	33
Schleifenkörper → Block .....	74, 239
Schlüssel .....	74, 239
Schlüsselattribut .....	236
Schlüsselwort .....	30
Schranke, asymptotische .....	109
Seekabel .....	287
Selection Sort .....	85, 103
Server .....	283
Service .....	284
Set .....	72
Setter .....	222, 358
Short-circuit evaluation → Kurzschluss- auswertung .....	221, 356
Sichtbarkeit .....	221, 356
Sigma .....	85
Signatur .....	219, 303
Skriptsprache .....	273
Skytale .....	300
Softskills .....	324, 398
Software .....	311
Sortieren durch Auswählen → Selection Sort .....	174
Sortieren durch Einfügen → Insertion Sort .....	266
Sortierung, topologische .....	195
Speicher .....	197
Sprache .....	30
<i>kontextfreie</i> .....	196
<i>natürliche</i> .....	196
<i>reguläre</i> .....	196
SQL .....	240
Stack .....	67
Stack Trace .....	370
Stapel → Stack .....	194, 195
Startsymbol .....	198
Startzustand .....	291
Statuscode .....	295
Stern-Topologie .....	265
Steuerwerk .....	354
String .....	305
Structured Query Language → SQL .....	30
Substitution .....	147, 154
Subtraktion .....	144, 154
Suchbaum .....	142, 154
Suche .....	145
<i>binäre</i> .....	285
<i>lineare</i> .....	194
Suchraum .....	300
Switch .....	342
Symbol .....	236
Symmetrische Verschlüsselung .....	41
System, fehlertolerantes .....	398
<b>T</b>	
Tabelle .....	118, 120, 129
Tautologie .....	73
Technische Informatik .....	73
Teile und Herrsche .....	194, 352
Teilmenge .....	143
<i>echte</i> .....	143
Terminal .....	347
terminieren .....	397
Test first .....	269
Theoretische Informatik .....	179
Thread .....	130
Tiefensuche .....	315
Timsort .....	296
Top-down-Methode .....	174
Topologie, vollständig verbundene .....	262
Topologische Sortierung .....	93
Transistor .....	301
Transitivität .....	195
Triple-DES .....	201
Tupel .....	200
Turingmaschine .....	29
<b>U</b>	
Übergangstabelle .....	342
Umgebung .....	176
Umgebungsfehler .....	
Ungerichtet .....	

Ungewichtet .....	176
Unicode Standard .....	58
Unittest .....	344
<b>V</b>	
Variable .....	32
<i>lokale</i> .....	356
Vektorgrafik .....	59
Vererbung .....	218
Vererbungshierarchie .....	218
Verkettete Liste .....	66
Verkettung .....	204
Verschachteln .....	33
Verschlüsselung .....	
<i>asymmetrische</i> .....	302
<i>symmetrische</i> .....	300
Versionsverwaltungswerkzeug .....	314
Verzweigung .....	34
Virens Scanner .....	285
Virtuelle Maschine .....	274
Virtueller Computer .....	274
Volladdierer .....	276
Vollständig verbundene Topologie .....	296
Von-Neumann-Architektur .....	267
Vorbedingung .....	30
<b>W</b>	
Wahr .....	37
Wahrheitswert .....	37
Wahrheitswertetabelle .....	38
Webserver .....	284
Wert .....	74
White Hat Hacker .....	381
Wissenschaftliche Notation .....	56
WLAN .....	283
Wohldefiniert .....	29
Worst Case .....	103
Worst-Case-Komplexität .....	86
Wort .....	195
<i>leeres</i> .....	195
Wurzel .....	147
<b>Z</b>	
Zahl .....	
<i>ganze</i> .....	54
<i>natürliche</i> .....	54
<i>rationale</i> .....	54
<i>reelle</i> .....	54
Zeichenkodierung .....	57
Zensur .....	380
Zurückgeben .....	28
Zusammenhangskomponente .....	184
Zustand .....	198
Zustand, akzeptierender → Endzustand .....	199
Zustandsübergangsfunktion .....	32
Zuweisung .....	32
Zuweisungsoperator .....	55
Zweierkomplement .....	183
Zyklenfrei .....	183
Zyklus .....	183



Arne Boockmeyer, Philipp Fischbeck, Stefan Neubert

## Fit fürs Studium – Informatik

411 Seiten, broschiert, Mai 2017

24,90 Euro, ISBN 978-3-8362-4406-0

 [www.rheinwerk-verlag.de/4273](http://www.rheinwerk-verlag.de/4273)



**Arne Boockmeyer** ist Softwareentwickler und Master-Student am Hasso-Plattner-Institut in Potsdam. Er befasst sich mit der Didaktik des Faches Informatik, ist als Dozent zum Thema Programmierung mit Java tätig und als Tutor für Competitive Programming am Lehrstuhl für Algorithm Engineering aktiv. Er begeistert regelmäßig Schüler für die Informatik und freut sich immer, wenn er seine Faszination davon weitergeben kann.



**Philipp Fischbeck** kennt typische Hürden beim Lernen aus seiner Arbeit mit Jugendlichen in der Schülerakademie des Hasso-Plattner-Instituts und mit Studierenden als Tutor. Seine Interessenschwerpunkte im Master-Studium sind Algorithmik und Computergrafik. Wer sich vom Spaß am Problemlösen und Rätseln anstecken lassen möchte, ist bei ihm genau richtig.



**Stefan Neubert** hat zahlreiche Informatik-Workshops und -Camps für Schülerinnen und Schüler entwickelt und durchgeführt. Er unterstützt seit Jahren die Bachelor-Lehrveranstaltung »Theoretische Informatik« als Tutor und arbeitet mit Leidenschaft dafür, komplexe Themen verständlich zu machen. Der Masterstudent am Hasso-Plattner-Institut begeistert Schüler mit seiner verständlichen Sprache und anschaulichen Beispielen – vor allem dann, wenn es auch mal anspruchsvoller wird.

*Wir hoffen sehr, dass Ihnen diese Leseprobe gefallen hat. Gerne dürfen Sie diese Leseprobe empfehlen und weitergeben, allerdings nur vollständig mit allen Seiten. Die vorliegende Leseprobe ist in all ihren Teilen urheberrechtlich geschützt. Alle Nutzungs- und Verwertungsrechte liegen beim Autor und beim Verlag.*

Teilen Sie Ihre Leseerfahrung mit uns!

