

Jens Gallenbacher

Abenteuer Informatik

IT zum Anfassen –
von Routenplaner bis Online-Banking

3. Auflage

Dr. Jens Gallenbacher
TU Darmstadt
Hochschulstr. 10
64289 Darmstadt
E-Mail: informatik@gallenbacher.de

ISBN 978-3-8274-2965-0

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Springer Spektrum

1.–2. Aufl.: © Spektrum Akademischer Verlag Heidelberg 2006, 2008

3. Aufl.: © Springer-Verlag Berlin Heidelberg 2012

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Planung und Lektorat: Dr. Andreas Rüdinger, Barbara Lühker

Satz: Mitterweger & Partner, Plankstadt

Einbandabbildung: © getty-images

Einbandentwurf: wsp design Werbeagentur GmbH, Heidelberg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Spektrum ist eine Marke von Springer DE. Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media.
www.springer-spektrum.de

Inhalt (im PDF auszugsweise)

Einleitung

VIII

Kapitel 1 – Sag’ mir wohin ...

1

Lösen Sie das Problem des kürzesten Weges: Welchen Weg muss ich nehmen, um mit der kürzesten Strecke bzw. am schnellsten von Ort A nach Ort B zu kommen. Kleine Krabbeltiere werden Ihnen dabei behilflich sein ...

Kapitel 2 – Ordnung muss sein ...

41

Einstein und Freud bevorzugten einhellig das Chaos auf ihrem privaten Schreibtisch. Damit ein Computer aber für uns schnell Informationen auffindet, müssen diese sortiert werden. Lernen Sie, wie der Computer vom Menschen gelernt hat ...

Kapitel 3 – Ich packe meinen Koffer und ...

81

Wenn ein Informatiker einen Rucksack füllt, dann sollte der zur Verfügung stehende Platz bestens ausgenutzt sein! Vollziehen Sie die Kunst des Packens nach.

Kapitel 4 – 10 000 000 Jahre Informatik

107

(nur in der gedruckten Ausgabe)

Eine solche Geschichte kann nur diese Wissenschaft bieten. Lassen Sie sich überraschen ...

Kapitel 5 – Von Kamelen und dem Nadelöhr

121

(nur in der gedruckten Ausgabe)

Wie passen riesige Texte in einen kleinen Speicher, warum können manche Digitalkameras mit dem gleichen Speicherchip mehr Bilder machen als andere und wie funktioniert MP3 eigentlich? Datenkomprimierung ist ein sehr wichtiges Thema für die heutige Gesellschaft, in der immer mehr und immer größere Informationsmengen versendet und gespeichert werden müssen.

Kapitel 6 – Paketpost 161

„Eine Lieferung für Julius Wagner“ – diesen Spruch enthält kein Datenpaket, das im Dschungel des Internet den richtigen Empfänger sucht. Wie es trotzdem ankommt, erfahren Sie in diesem Kapitel.

Kapitel 7 – Alles im Fluss 185

(nur in der gedruckten Ausgabe)

Die Cloaca Maxima der alten Römer bestand aus riesigen Abwasserleitungen und -systemen. Moderne Leitungsnetze kommen mit wesentlich weniger Platz aus, weil ein einfaches Verfahren aus der Informatik genau bestimmen kann, wie viel durch welche Leitungen von wo nach wo fließen kann. Möchten Sie es begreifen?

Kapitel 8 – Ordnung im Chaos 225

Hatte Einstein doch recht und Chaos ist besser als Ordnung? Informatiker nutzen seit langem ein Verfahren, das Informationen scheinbar chaotisch anordnet, sie jedoch trotzdem auf Anhieb auffindbar macht. Vielleicht eine neue Möglichkeit, die Wohnung zu organisieren (Achtung: Ärger mit dem Partner ist vorprogrammiert ...)

Kapitel 9 – Mit Sicherheit 253

Safeknacker sind unmodern geworden. Ihre Kollegen des IT-Zeitalters bedienen sich nur noch eines Computers und des Internets. Nur wer die Prinzipien hinter modernen Sicherheitsmechanismen begreift, kann diese so einsetzen, dass die Diebe kaum noch eine Chance haben!

Kapitel 10 – Rechnen mit Strom 295

(nur in der gedruckten Ausgabe)

Mit Fingern rechnen – das können sich die Meisten noch vorstellen. Ein Computer arbeitet auf Basis elektrischer Ströme und Spannungen. Lernen Sie in diesem Kapitel, wie das funktioniert.

Kapitel 11 – InformaGik 323

Nicht alles ist fauler Zauber: Lernen Sie die Magie der Fehlerkorrekturmechanismen kennen, die zum Beispiel dafür sorgen, dass sich auch verstaubte und verkratzte DVDs noch einwandfrei abspielen lassen.

Kapitel 12 – Allmächtiger Computer!? 339

Computer ohne Grenzen? Wohl kaum! Auch für die schlaun Kisten gibt es weiße Flecken auf der Landkarte: Nicht alles ist von ihnen berechenbar. Erfahren Sie anhand spannender Puzzlespiele, welche Dinge niemals per Computer ermittelbar sein werden.

Glossar 355

Bildnachweis 357

Anhang A: Kopiervorlagen 359

Anhang B: Bastelbögen 371

Einleitung

Informatik – in den Anfängen weckte dieser Begriff die Assoziation mit hageren, bleichen Typen vor flimmernden Bildschirmen. Sie gaben unverständliches Kauderwelsch in einer Sprache von sich, die dem Englischen ähnlich war, aber eben doch nicht englisch. Eintrocknete Kaffeereste und halbleere Pizzakartons vervollständigten das Bild.

Im Laufe des IT-Booms am Ende des 20. Jahrhunderts änderte sich das dann sehr schnell: Jeder, der in seiner Ausbildung auch nur in die Nähe eines Computers gekommen war, wurde schnell und hochbezahlt von der Industrie aufgesogen – der Informatiker war zum modernen Yuppie geworden.

Inzwischen hat sich die Euphorie wieder gelegt: Das Jahr 2000 hat ohne apokalyptische Endzeitstimmung begonnen, viele ITler, die vorsorglich zur Schadensbegrenzung angeheuert wurden, wurden wieder mit anderen Aufgaben betraut. Die Börsen sind mehrfach zusammengebrochen, Computer sind wieder einmal mehr in den normalen Alltag aufgenommen worden.

Über die ganze Zeit war die Informatik für viele immer ein Buch mit sieben Siegeln: In der Schule haben viele gar keinen Unterricht in diesem Fach gehabt und wenn, dann war das oft ein Grundkurs im Programmieren oder im Umgang mit Tabellenkalkulation und Textverarbeitung.

Wie funktioniert aber ein Computer und vor allem, was steckt hinter den vielfältigen Programmen, die für ihn verfügbar sind? Wie kann unser PC in wenigen Sekunden die kürzeste Strecke von Berlin nach München bestimmen und dabei auch noch aktuelle Staumeldungen einbeziehen? Wie werden riesige Datenmengen in einem Augenblick sortiert? Woher nehmen die Banken die Gewissheit, dass ihr Portal für Internet-Banking sicher sei?

Oft sind Verfahren der Informatik beteiligt, die schon ein halbes Jahrhundert oder länger bekannt sind. In diesem Buch möchte ich Ihnen anhand einfacher Experimente die Wissenschaft der Informatik näher bringen. Dafür wird kein Computer benötigt, sondern allenfalls ein Stift und manchmal auch eine Schere.

Ziel sind pro Kapitel mehrere Aha-Erlebnisse: Sie lernen nicht im Detail, wie man einen Computer programmiert, aber Sie werden verstehen, was hinter vielen Standard-Programmen steckt.

Jedes Kapitel hat hierfür einen Experimentierteil, der sehr einfach aufgebaut ist. Dieser erfordert praktisch kein Vorwissen und ist daher auch für Kinder (etwa ab der 3. Klasse) geeignet, wenn er mit den Eltern zusammen durchgearbeitet wird. Der Weg zur Lösung ist hier Ziel: Wie geht ein Informatiker vor, um ein Problem zu knacken?

Auch die Begründung ist meist noch recht einfach zu verstehen: Warum funktioniert das Verfahren?

Wer dann (hoffentlich) ganz versessen darauf ist, noch mehr zu erfahren, kann am Ende jedes Kapitels eine mehr wissenschaftlich gehaltene Zusammenfassung und Vertiefung lesen. Im Mittelpunkt stehen hier die Fragen, warum das Verfahren immer funktioniert (also der Beweis), wo weitere Anwendungen liegen und wie in der Praxis das Verfahren noch verbessert wird. Hier sind zum Verständnis oft Mathematikkenntnisse vonnöten, wie sie in der gymnasialen Oberstufe vermittelt werden.

Dieses Buch kann also auf ganz verschiedene Weise gelesen bzw. – wie ich lieber sagen möchte – erlebt werden. Um das zu unterstützen, sind ein paar kleine Details eingebaut, die ich hier beschreiben möchte:

Beim Lesen von Abenteuer Informatik können Sie eine mehr passive oder mehr aktive Rolle wählen. Selbstverständlich ist es spannend und auch lehrreich, auf wichtige Erkenntnisse der Informatik selbst zu kommen – zum Beispiel nach einem eigenhändig durchgeführten Experiment.

Sehr oft finden Sie daher eine Frage, einen Denkanstoß oder einen Arbeitsauftrag im Text. Direkt dahinter steht das „Denk-Köpfchen“.



Es soll Sie kurz daran hindern, hier schon weiterzulesen. Sie können sich nun entscheiden, ob Sie selbst versuchen, zu einer Lösung zu kommen oder diese einfach im nächsten Abschnitt lesen. Lassen Sie sich davon leiten, wie spannend das Thema für Sie ist.

Eine Quelle zusätzlicher Informationen ist die kleine linke bzw. rechte Spalte des Buches. Hier finden Sie kurze Lebensläufe von wichtigen Persönlichkeiten, Anekdoten und anderes Wissenswerte im Zusammenhang mit dem Haupttext. Das Verständnis des Kapitels ist allerdings auch ohne diese Spalte gewährleistet.

Achten Sie auf das Literatursymbol. Es verweist auf zusätzliche spannende Lektüre (die meistens so populär sind, dass man sie auch in Stadtbibliotheken findet). Die genauen Referenzen dazu stehen am Ende jedes Kapitels. Aufgrund der Schnelllebigkeit des World Wide Web verzichten wir auf zusätzliche Verweise ins Internet. Sie finden jedoch weitere Hinweise auf www.abenteuer-informatik.de.



Den Anfang neuer Kapitel finden Sie im Buch übrigens recht einfach, wenn Sie es von der Seite betrachten: Das große Bild auf der linken Seite ist auch hier zu erkennen und lässt sich als eine Art Index zweckentfremden.

Im Text sind immer wieder grundlegende Begriffe und Methoden der Informatik kurz erklärt. Um sie leichter wiederzufinden, stehen sie in blauen Kästchen. Wenn Sie möchten, legen Sie sich doch ein eigenes Merkblatt mit den Inhalten dieser Kästchen an.

Die Reihenfolge der Kapitel ist übrigens sehr lose. Auch wenn Sie das Buch nicht von vorne nach hinten durchlesen, sondern sich hier von Ihrer „Abenteuerlust“ leiten lassen, kann man die meisten Texte verstehen! Manchmal ist allerdings ein Hinweis auf eines der vorhergehenden Kapitel gegeben, das dann eine wichtige Grundlage beinhaltet.

Mein Tipp ist, auf jeden Fall das erste Kapitel, vielleicht auch noch das zweite Kapitel zu Beginn zu lesen. Danach können Sie unbeschwert stöbern.

Kommen wir zu einem sehr wichtigen Teil, der dieses Buch auch von den allermeisten anderen Informatik-Büchern unterscheidet: der Bastelbogen!

Es ist meine feste Überzeugung, dass ein wissenschaftliches Thema sich am allerbesten erschließt, wenn man es buchstäblich „begreifen“ kann und nicht nur darüber liest. In der Informatik besteht die Vorstellung dieses Begreifens meistens darin, etwas mit dem Computer zu machen – ihn zu programmieren oder vorgefertigte interaktive Lernprogramme zu nutzen.

Das ist bereits sehr viel besser als rein passives Lernen, aber es ersetzt nicht, die Dinge wirklich in die Hand zu nehmen. Nun bestehen die meisten „Dinge“ der Informatik entweder aus mathematischen Modellen oder aus elektrischen Signalen – beides ist schlecht greifbar.

Glücklicherweise sind die mithilfe der Informatik gelösten Probleme sehr wohl real und greifbar und im Anhang finden Sie das Material, um diese Probleme nachzuspielen.

Anhang A enthält Vorlagen, die Sie beim Experimentieren mit dem entsprechenden Kapitel am besten neben dem Buch liegen haben sollten. Manche müssen auch erst auf ein größeres Format zusammengeklebt werden – zum Beispiel mit Klebestreifen.

Insbesondere Vorlagen, die ausgefüllt werden müssen, sollten vorher kopiert werden. Diese sind auch absichtlich in Schwarz-Weiß gehalten (gilt für die Kapitel 1 und 7).

Anhang B enthält dann die eigentlichen Bastelbögen auf steifem Papier. Die meisten Bögen sind eindeutig einem Kapitel zugeordnet, am Ende gibt es dann allerdings einen Restebogen mit einem Sammelsurium. Der Verwendungszweck und die Bastelanleitung sind im jeweils zugehörigen Kapitel zu finden.

Trennen Sie die Bögen am besten nah am Einband mit einem sehr scharfen Messer heraus. Danach können die Einzelteile ausgeschnitten werden. Größere Objekte müssen manchmal erst zusammengeklebt werden. Dabei gibt es zwei Möglichkeiten:

Verwenden Sie Klebefilm, können die Vorlagen nach Verwendung zur besseren Aufbewahrung an den Nahtstellen wieder zusammengeklappt werden. Allerdings ist die Handhabung während der Benutzung besser, wenn man die Teile fest mit Universalklebstoff verbunden hat.

Die Bastelbögen sind sehr wertvoll, weil sie das Buch erst zu einem richtigen Experimentierbuch machen! Passen Sie daher gut darauf auf. Für die kleineren Teile empfehle ich eine Prospekthülle für DIN A5, die man hinten ins Buch klebt. Wenn Sie die Bögen herausgetrennt haben, bleibt im Einband auch eine Lücke, die dann von einer solchen Prospekthülle ausgefüllt wird. Auf diese Weise passen Buch und Kleinteile dann auch weiterhin gut ins Regal.

Für Lehrer, die einen ganzen Satz der Bastelbögen benötigen, gibt es die Vorlagen auf DVD zum Selbstaussdrucken (zusammen mit der Lizenz, das für Lehrzwecke zu tun) unter der ISBN ■■■■■ zu erwerben.

Damit genug der Nutzungshinweise – ich wünsche Ihnen viel Spaß bei Ihrem ganz persönlichen Abenteuer Informatik.

Jens Gallenbacher

Zur dritten Auflage

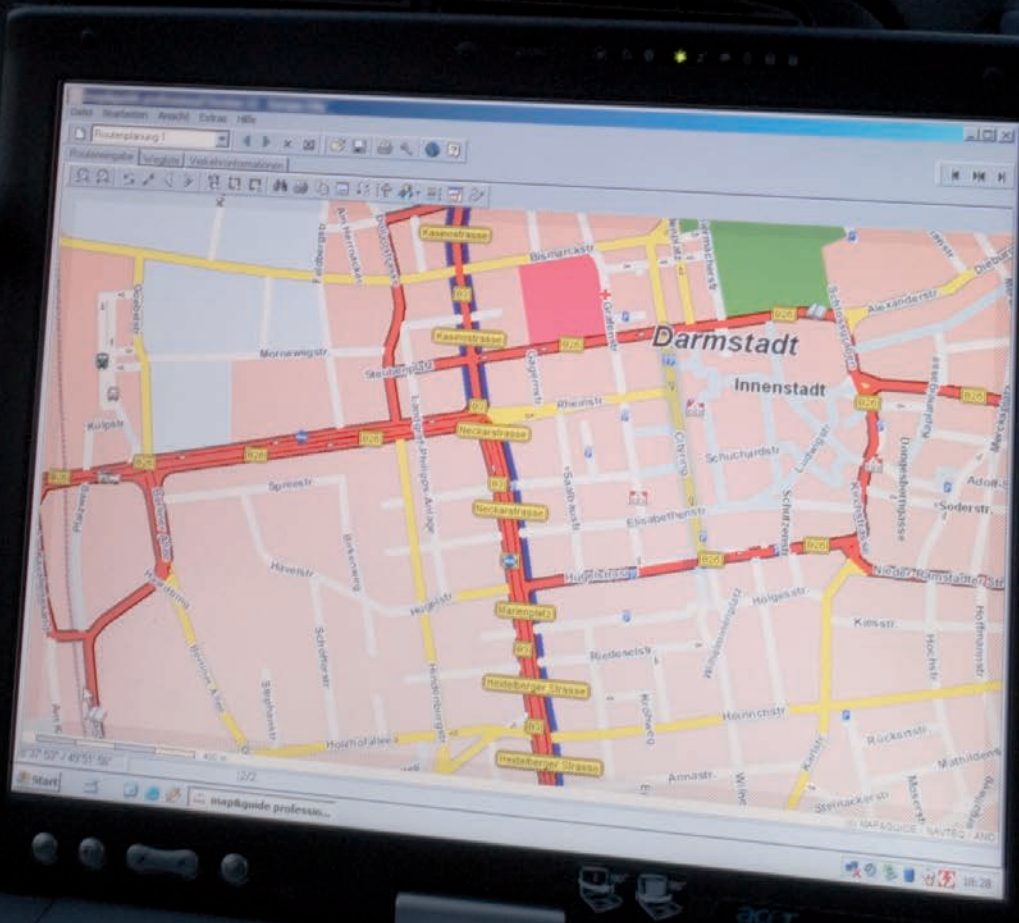
Zunächst geht mein Dank an alle Leser, insbesondere an diejenigen, die durch Zusendung von Verbesserungsvorschlägen dazu beigetragen haben, diese dritte Auflage noch runder zu machen.

Inzwischen gibt es Abenteuer Informatik auch als Ausstellung zum Mitmachen. Auf Wanderschaft in Universitäten, Schulen und auch Häusern wie dem Heinz Nixdorf Forum in Paderborn oder dem Ars Electronica Center in Linz konnten bisher über 200.000 Besucher Informatik buchstäblich begreifen. Seit 2011 gibt es Abenteuer Informatik auch als festen Bestandteil des Science-Center ExperimentINTa in Frankfurt. Weitere Informationen finden Sie auf www.abenteuer-informatik.de.

Zwei besonders beliebte Exponate der Ausstellung habe ich für dieses Buch adaptiert: Freuen Sie sich daher auf acht weitere Bastelbögen und zwei Extra-Kapitel. Erfahren Sie mehr über die Magie der Informatik und auch über die Grenzen der Wissenschaft.

Viel Spaß dabei!

Jens Gallenbacher



1. Sag' mir wohin ...

Einführung

Routenplaner gehören heute schon fast zum Alltag: Viele Autos haben sie bereits eingebaut, wer keinen im Fahrzeug hat, lässt sich den günstigsten Weg zu seinem Ziel oft auf dem heimischen PC ermitteln und druckt ihn aus.

Versuchen wir es doch gleich: Nehmen Sie einen großen Straßenatlas und ermitteln Sie die günstigste Strecke von Stockheim nach Weilheim!

Zu viel Arbeit? Kein Atlas? Also gut – ich hatte in der Einleitung ja versprochen, dass alle notwendigen Materialien hier im Buch seien. Daher arbeiten wir erst einmal mit der folgenden kleinen Welt nach Abbildung 1.1.

Die Karte zeigt Orte, zwei Autobahnen, größere und kleinere Landstraßen. Die roten und blauen Zahlen geben dabei immer die Länge der Straße an, wenn man sie mit dem Auto fährt. Man kann sehen, dass kleine Straßen meistens viel länger sind, als sie scheinen, weil die vielen Kurven und Berge nicht eingezeichnet sind.

Welches ist denn nun der günstigste Weg von Imstadt nach Oppenheim?

Erster Ansatz wäre, zur nächsten Autobahn zu fahren. Aber geht das am besten über Pappstadt oder die Auffahrt Budingen? Außerdem führt ja von Pappstadt aus auch eine direkte Landstraße zum Ziel. Die ist aber wohl gewunden und ziemlich lang. Oder doch lieber die gelbe Straße zum Flughafen und von dort die Autobahn nach Oppenheim nehmen?

Versuchen Sie, die Lösung selbst herauszufinden. Hinweis: Sie müssen 123 Kilometer zurücklegen.

Geschafft? Gut! Dann lehnen Sie sich zurück und genießen Sie den Erfolg.

Wie sind Sie vorgegangen? Sie haben wahrscheinlich alle möglichen Wege durchprobiert und die Entfernung zum Ziel ermittelt. Dann haben Sie sich für den günstigsten Weg entschieden.

Dieses Verfahren gibt es auch bei Computern – es hat sogar einen Namen: die Brute-Force-Methode, also etwa „Brutale Macht“. Warum? Weil auf diese Weise etwas größere Probleme nur mit extrem großer Rechenkraft gelöst werden können. Überlegen Sie einmal, wie viele verschiedene Wege Sie schon bei der kleinen

Damit Sie das Bild neben sich legen können, ist es in Anhang A nochmals zum Heraustrennen abgedruckt.



Abbildung 1.1 Die Landkarte aus dem Atlas

Problemgröße durchspielen mussten. Stellen Sie sich nun vor, wie das bei Karten mit 1000 und mehr Städten ist – hier wären normale Rechner gar nicht mehr zur Lösung fähig.

Außerdem besitzt ein Rechner keine Intelligenz: Während Sie beim Durchprobieren unbewusst alle absurden und unwahrscheinlichen Möglichkeiten verwerfen, muss er diese durchrechnen.

Vorüberlegungen

Wie kommt ein Informatiker nun zu einer besseren Lösung?

Zunächst sollte man mit der *Methode der Abstraktion* arbeiten!

Methode der Abstraktion

In zur Verfügung stehender Information stecken sowohl relevante als auch unwesentliche Anteile. Durch Abstraktion reduzieren Sie die Information auf das für die aktuelle Problemlösung Wesentliche: Dadurch können Sie sich besser auf Ihre Aufgabe konzentrieren.

Man könnte auch sagen: Werfen Sie alles Überflüssige über Bord und konzentrieren Sie sich auf das Wesentliche. Was ist aber bei der gestellten Aufgabe wesentlich?

Alle gegebenen Informationen stecken in der Karte. Welche Typen von Informationen kann man erkennen? Erstellen Sie eine Liste, bevor Sie weiterlesen.



Die Informationen der Karte sind in folgender Tabelle zusammengefasst.

	wichtig	nicht wichtig
Namen der Städte	☐	☐
Position der Städte	☐	☐
Größe der Städte	☐	☐
Verlauf der Straßen	☐	☐
Länge der Straßen	☐	☐
Namen und Nummern der Straßen	☐	☐
Straßentyp	☐	☐
Straße führt von ... nach ...	☐	☐
Landschaftliche Information	☐	☐

Überlegen Sie weiter, welche dieser Informationen wir benötigen, um den kürzesten Weg zwischen zwei Städten zu suchen. Markieren Sie für jede Information, ob diese Ihrer Meinung nach für die Aufgabenstellung wichtig oder nicht wichtig ist. Der kürzeste Weg soll sich hierbei auf die zu fahrende Strecke beziehen, nicht auf die gefahrene Zeit.



Mein Ergebnis ist folgendes:

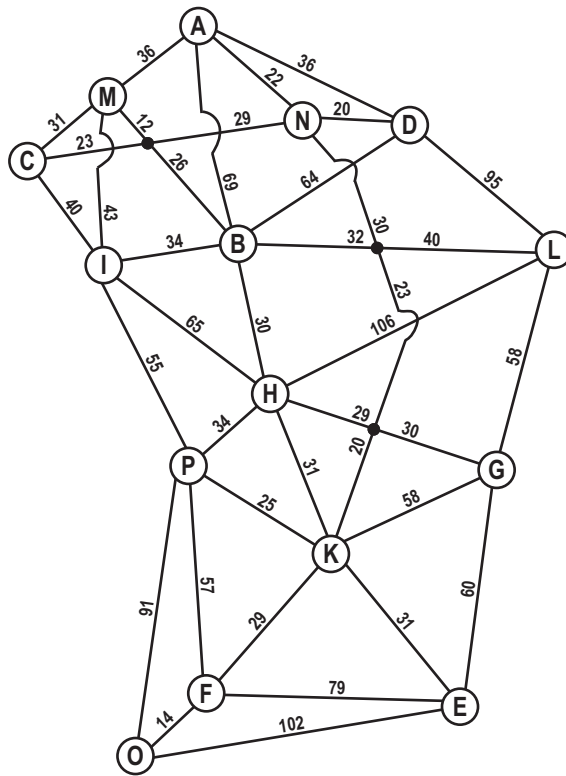
Namen der Städte	WICHTIG! Wenn man nicht weiß, welche Stadt wie heißt, kann auch nicht der kürzeste Weg zwischen Imstadt und Oppenheim bestimmt werden
Position der Städte	NICHT WICHTIG! Es ist uns egal, wo sich die Städte genau befinden. Relevant sind nur die Straßen zwischen den Städten.
Größe der Städte	NICHT WICHTIG! Kommt in unserer Aufgabenstellung nirgendwo vor.
Verlauf der Straßen	NICHT WICHTIG! Es kommt nur auf die Strecke an, nicht auf den Verlauf.
Länge der Straßen	WICHTIG! Um die Reisedistanz zu bestimmen, brauchen wir die einzelnen Strecken zwischen den Orten.
Namen und Nummern der Straßen	NICHT WICHTIG! Zumindest zur Bestimmung der kürzesten Strecke irrelevant.
Straßentyp	NICHT WICHTIG! Da es nur auf die Entfernungen, nicht auf Zeit ankommt, ist egal, ob Autobahn oder Feldweg gefahren wird.
Straße führt von ... nach ...	WICHTIG! Wir benötigen die Information, von welcher Stadt zu welcher anderen eine Straße führt.
Landschaftliche Information	NICHT WICHTIG!

Nun kann die Karte neu gezeichnet werden und zwar so, dass möglichst alle unrelevanten und damit störenden Informationen fehlen. Versuchen Sie es einmal selbst, bevor Sie weiterlesen!



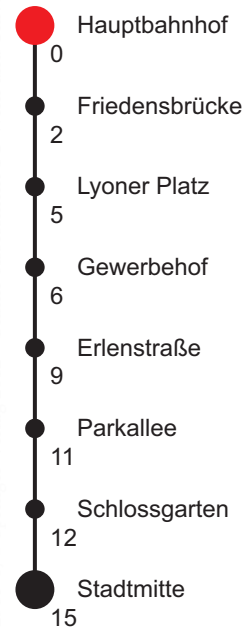
Aus Gründen der Übersichtlichkeit habe ich hier in Abbildung 1.2 die Städte auf ihren ersten Buchstaben reduziert. Das ist jedoch nicht unbedingt notwendig und wenn Ihre Lösung die ausgeschriebenen Namen enthält, können Sie auch damit weiterarbeiten.

Man sieht, dass es noch ein paar Spezialitäten gibt: An vier Stellen kreuzen sich zwei Straßen, ohne dass es Auf- oder Abfahrten von einer zur anderen gäbe (z. B. eine Autobahnbrücke über einer Landstraße). Dies ist mit einem Bogen dargestellt. Außerdem gibt es weitere drei Stellen, an denen sich zwei Straßen schneiden und es Auf- und Abfahrten gibt – gekennzeichnet mit einem Punkt.



Informatiker sind jedoch prinzipiell faul und lieben es übersichtlich: Zu viele spezielle Fälle und Unterscheidungen machen das Denken schwierig. Wie bei Mathematikern, die einen Bruch erst einmal auf den gleichen Nenner bringen, wird hier auch versucht, ein Problem möglichst gleichförmig darzustellen.

Linie 1



Abstraktion begegnet uns ständig! Wegweiser, Straßenschilder, Fahrpläne, Infobroschüren und viele andere Dinge des Alltags sind so aufbereitet, dass die nötigen Informationen möglichst offen und gut erkennbar dargestellt sind.

Abbildung 1.2 Abstraktere Form der Landkarte

Methode der Gleichformung

Versuchen Sie, die verschiedenen Facetten eines Problems auf die gleichen Grundelemente zurückzuführen. Dadurch wird einerseits das Problem übersichtlicher und andererseits benötigt man weniger Lösungsansätze: Für gleichförmige Teilprobleme kann der gleiche Lösungsansatz verwendet werden.

Können Sie die Karte auf diese Weise noch einfacher gestalten?

Überlegen Sie, welches die Eigenschaften der „normalen“ Elemente sind und ob man die speziellen Elemente nicht auch als normale Elemente darstellen kann.



Auf der Karte sind Städte als Kreise eingezeichnet. Ohne dass dies speziell vermerkt ist, kann man offenbar bei Städten problemlos von einer Straße auf eine angrenzende Straße wechseln.

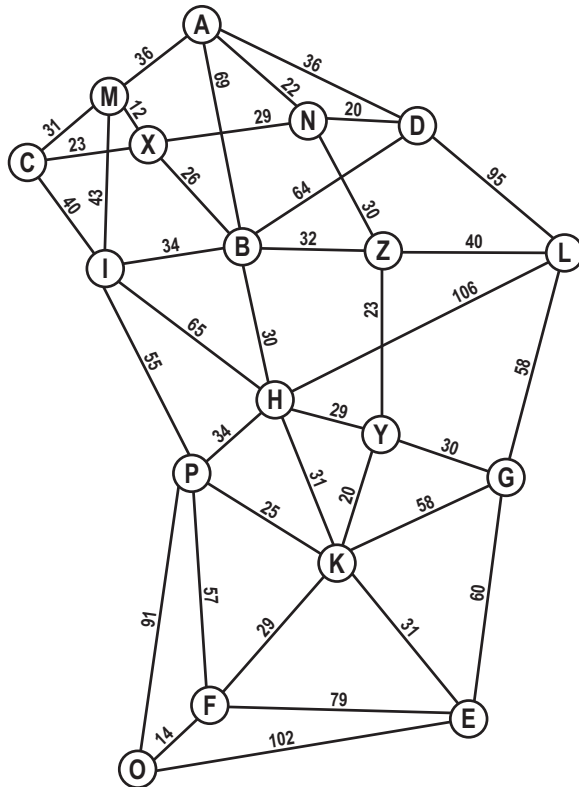


Abbildung 1.3 Landkarte mit durch Orte ersetzten Knotenpunkten

Genau das soll auch an den mit Punkt gekennzeichneten Stellen möglich sein. Also tun wir einfach so, als wenn sich dort Städte befinden. Um sie nicht mit den anderen Städten zu verwechseln, nennen wir sie (X) , (Y) und (Z) . Den fertigen Plan zeigt Abbildung 1.3.

An allen weiteren Kreuzungspunkten zwischen zwei Straßen ist jetzt kein Wechsel mehr möglich. Es ist daher auch keine Unterscheidung, also auch keine Kennzeichnung durch Bogen mehr notwendig.

Die Städte sind immer noch auf ihrer geographischen Position eingezeichnet. Dadurch ergeben sich an manchen Stellen Ballungszentren. Die Straßenführung wird unübersichtlich. Die geographische Position der Städte haben wir jedoch weiter vorne als unrelevant deklariert. Um es Ihnen so einfach wie möglich zu machen, habe ich die Karte in Abbildung 1.4 etwas entzerrt.

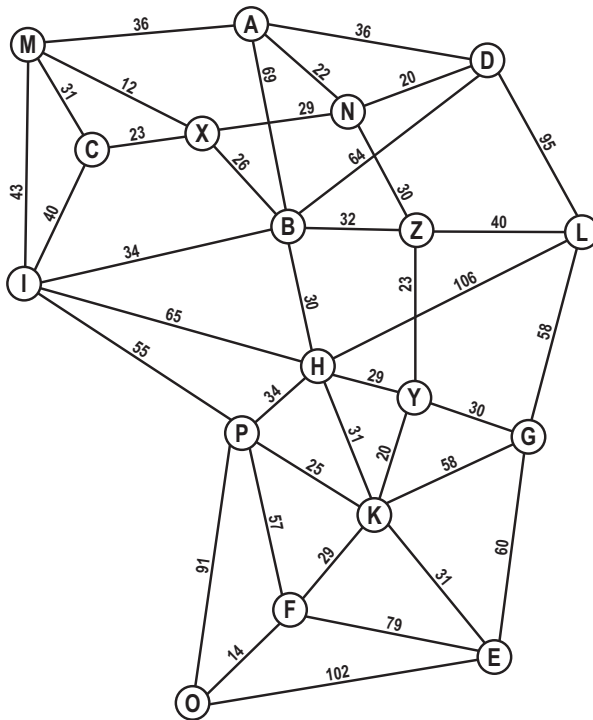


Abbildung 1.4 Entzerzte Landkarte

Bitte überprüfen Sie, dass die Inhalte immer noch übereinstimmen, also die Verbindungen zwischen den Städten gleich sind und die gleiche Längenangabe aufweisen. Lediglich die *Darstellung* hat sich geändert.

Sie können auch überprüfen, dass es sich noch um die gleiche Karte handelt wie am Anfang des Kapitels – lediglich mit weniger Detailinformationen.

Mit dieser Karte werden wir im Folgenden weiter arbeiten, daher ist sie als Kopier-
vorlage am Ende des Buches nochmals besonders groß abgedruckt.

Dijkstra und die Ameisen

Wie kommen wir denn aber nun zum kürzesten Weg von Imstadt nach Oppenheim?

Der direkte Ansatz, immer ganze Wege zu betrachten, ist ja bereits aufgrund des hohen Aufwands gescheitert. Versuchen wir etwas anderes – vielleicht können wir ja hier von der Natur lernen:

Ein Stamm Ameisen hat auf der Suche nach Futter ein ähnliches Problem: Eine Kundschafterin findet ein großes Stück Fleisch. Welchen Weg sollen die Arbeiterinnen nehmen, um die Beute am schnellsten zu sichern?

Setzen wir also den Stamm Ameisen auf unseren Ausgangspunkt Imstadt bzw. (I). Fünf Wege führen von dort weg, also teilen sich unzählige Ameisen auf, um diese zu erkunden. Wir nehmen jetzt einmal an, dass alle Ameisen gleich schnell sind, zum Beispiel 1 km pro Minute (okay, für unser Problem setzen wir offenbar Turbo-Ameisen ein...).

In der Landkarte verfolgen wir den Weg der Ameisen. Abbildung 1.5 zeigt in Rot ihren Fortschritt nach 34 Minuten.

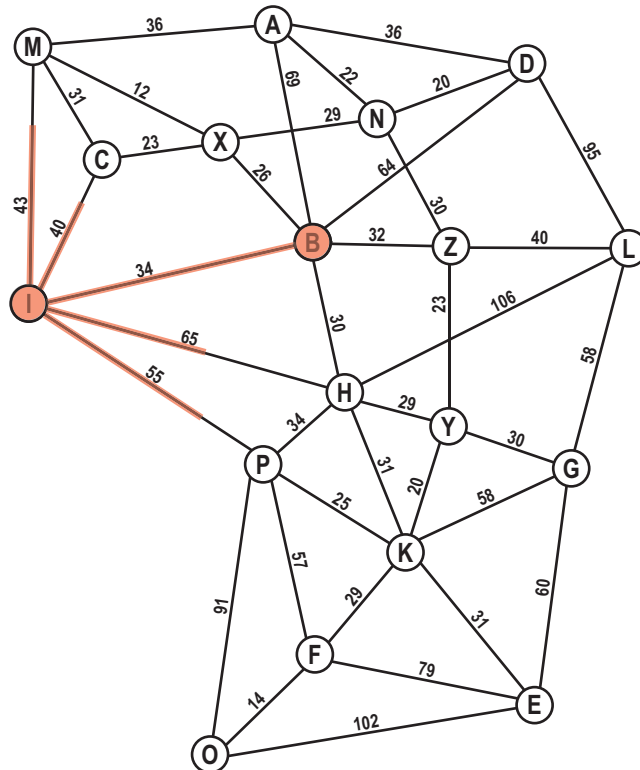


Abbildung 1.5 Die Ameisen auf dem Weg von Imstadt aus

Die Ameisen haben (B) erreicht. Auf den anderen Strecken sind sie noch auf dem Weg. Bitte lassen Sie sich nicht davon täuschen, dass die Strecke mit 34 km am längsten aussieht: Wir verwenden ja explizit keinen maßstabsgetreuen Plan!

Was haben wir dadurch bisher von den Ameisen gelernt?



Genau! Um von (I) nach (B) zu kommen, gibt es garantiert keinen günstigeren Weg als den mit 34 km. Die Ameisen haben ja sämtliche bisher für sie möglichen Wege ausprobiert und sind nach 34 km zuerst bei (B) angekommen.

Wie geht es jetzt weiter? Die Ameisen, die bisher nirgendwo angekommen sind, setzen ihren Weg einfach fort. Die Ameisen bei B teilen sich erneut auf: Wieder sind fünf Wege möglich. Den bisherigen Erfolg dokumentieren sie, indem sie die bisher zurückgelegte Strecke bei (B) vermerken. Abbildung 1.6 zeigt den Plan der Ameisen.

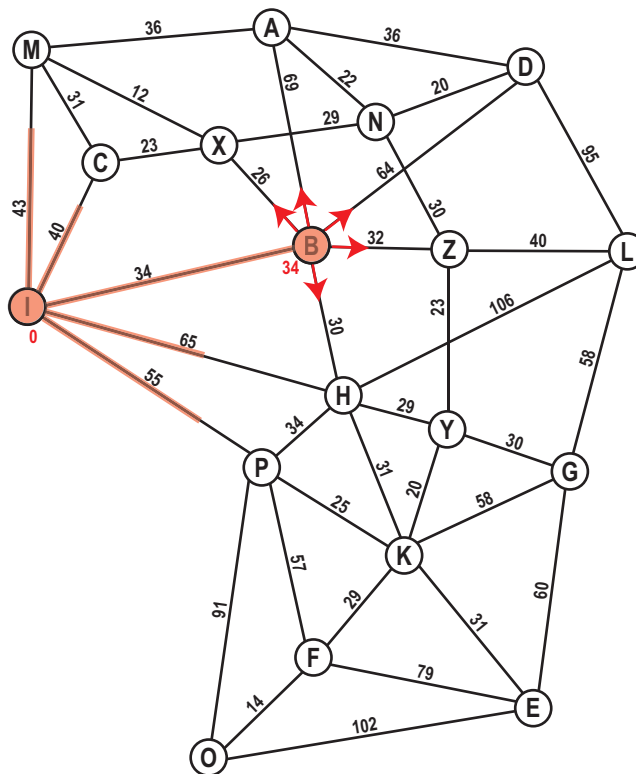


Abbildung 1.6 Die Ameisen bewegen sich von (B) in alle Richtungen

Nach insgesamt 40 Minuten kommt der nächste Ameisentrupp bei (C) an. Die Insekten sehen, dass sie die Ersten sind, markieren die Strecke, verzeichnen die Anzahl der bisher gelaufenen Kilometer und teilen sich auf die zwei bei (C) weitergehenden Wege auf.

In der 43. Minute kommt dann auch der Trupp bei (M) als Erster an. Auch dieser Weg wird vermerkt (siehe Abbildung 1.7). Von hier aus sind drei weitere Strecken zu erkunden.

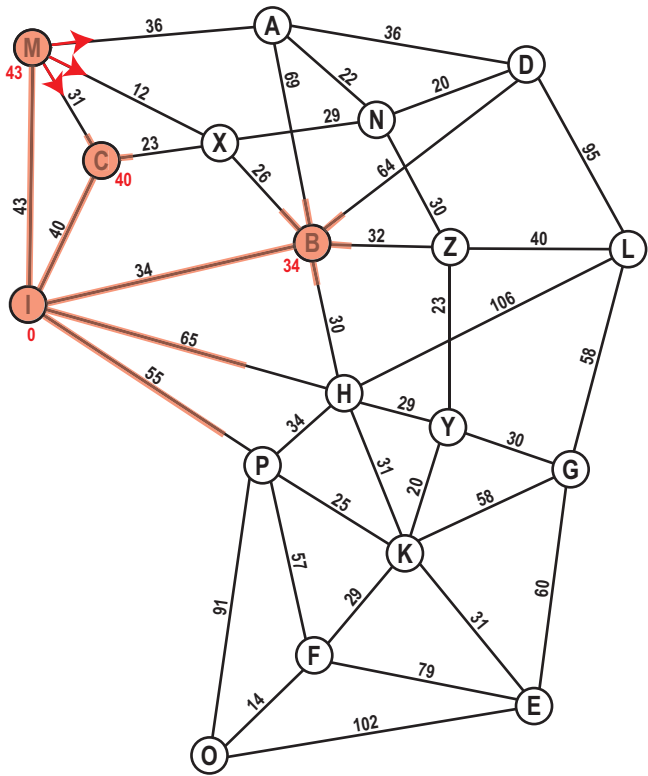


Abbildung 1.7 Drei weitere Wege werden von (M) aus erkundet

Soweit verlief alles nach dem gleichen Schema. Die kürzesten Strecken zu den Städten (B), (C) und (M) stehen nun fest.

Vielleicht haben Sie bemerkt, dass nun Ameisentrupps sowohl von (M) als auch von (C) ausgehend unterwegs sind – zwischen beiden Städten auf Kollisionskurs. Was passiert jetzt, wenn sie sich irgendwo auf der Strecke dazwischen begegnen? Welche Informationen können sie austauschen? Bringt ihnen das etwas für ihr Ziel, das Gelände zu erkunden?



Genau! Der Trupp von (C) weiß, dass dieses Ziel bereits erreicht ist, die kürzeste Strecke dorthin also feststeht. Der Trupp, der von (M) kommt, kann das Gleiche von seinem Ausgangspunkt berichten. Es ist also sinnlos, weiterzumarschieren. Die Strecke wird als „unbrauchbar“ markiert, die Ameisen können wieder zu ihrem Stamm zurück, sie sind sozusagen aus dem Rennen. Abbildung 1.8 zeigt das gleich mit einem dicken roten Kreuz.

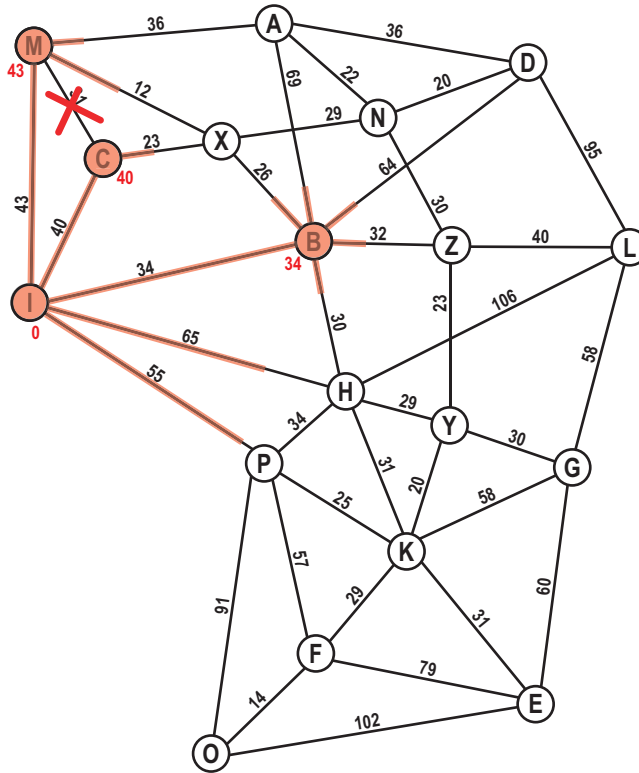


Abbildung 1.8 Der Weg zwischen (C) und (M) wird als unbrauchbar markiert

Als Nächstes kommen zwei Erkundungstrupps gleichzeitig an: In der 55. Minute erreichen die Ameisen (P) und (X). (P) erreichen sie auf direktem Wege von (I) aus. Bei (X) kommt der Trupp an, der von (M) unterwegs ist (43 km bis (M) plus 12 bis (X)).

Wieder teilen sich die Ameisen auf. Von (X) gibt es nur noch einen erfolgversprechenden Weg, bei den anderen treffen sie recht schnell auf Kameraden und geben die Strecke auf. Die von (P) ausgehenden Touren sind alle noch offen. Abbildung 1.9 zeigt den aktuellen Stand.

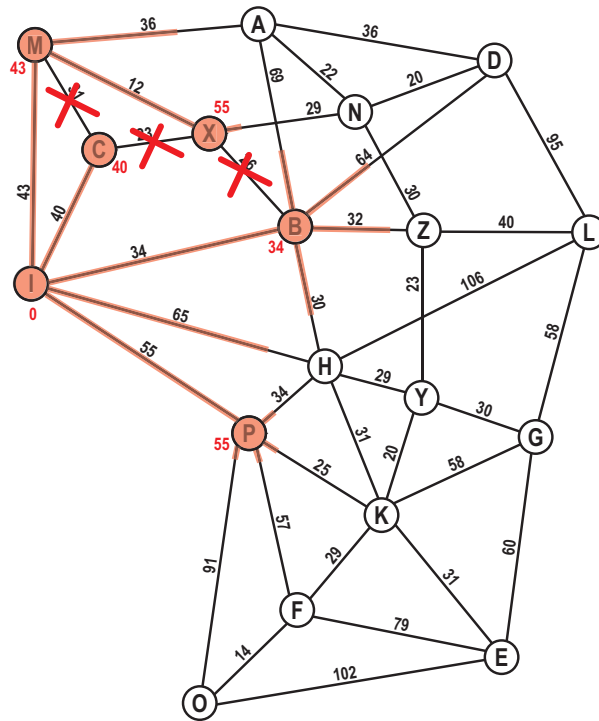


Abbildung 1.9 Weiterer Fortschritt der Ameisen

Haben Sie das Prinzip verstanden?

Statt immer nur einen Weg auszuprobieren und wieder zu verwerfen, wenn sich ein besserer gefunden hat, erkunden die Ameisen gleichzeitig alle sich bietenden Möglichkeiten.

Kommen sie als Erste bei einer Stadt an, wissen sie, dass der genommene Weg der günstigste ist, denn sonst wäre ja ein anderer Erkundungstrupp bereits da (zur Erinnerung: Alle bewegen sich mit der gleichen Geschwindigkeit).

Treffen die Ameisen irgendwo auf Artgenossen, dann wissen sie, dass ihre Reise zu Ende ist, weil sie sich gegenseitig ein „Ich bin schon da“ berichten können. Andere haben also das anvisierte Ziel früher erreicht.

Führen Sie zur Übung das Verfahren noch zu Ende und zeichnen Sie den Weg der Ameisen, die gefundenen Strecken sowie verworfene Wege in die Landkarte ein!

Können Sie sich vielleicht gleichzeitig auch schon vorstellen, wie ein Computer das Ameisenprinzip adaptiert?



Die Lösung wird in Abbildung 1.10 dargestellt. Was für Informationen haben wir dadurch jetzt eigentlich gewonnen?

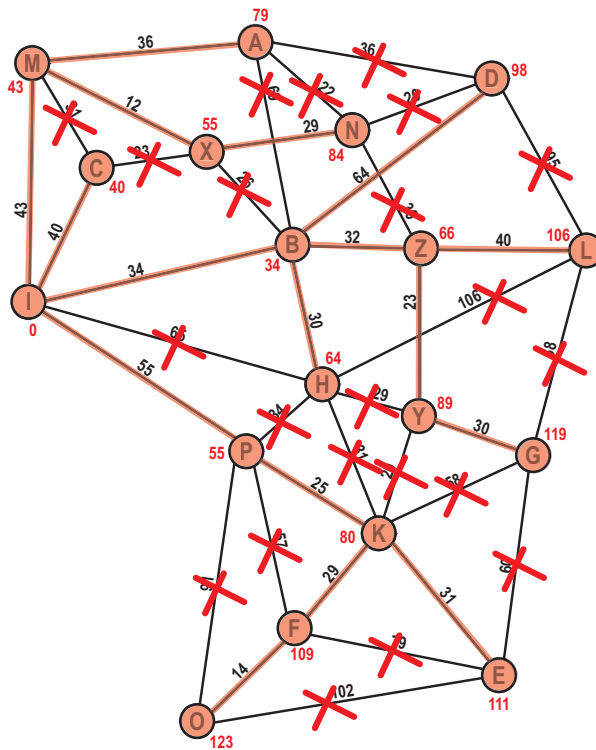


Abbildung 1.10 Die fertig erkundete Landkarte der Ameisen

Um von Imstadt zu einem beliebigen anderen Ort zu kommen, folgen Sie dem Pfad der Ameisen:

Von Imstadt nach Oppenheim kommt man so am günstigsten über Pappstadt, Krupsing und Flughafen. Die Gesamtstrecke beträgt 123 km. Ein günstigerer Weg existiert nicht!

Sie haben aber nicht nur die ursprüngliche Aufgabe gelöst, sondern sozusagen als Abfallprodukt noch die kürzesten Wege von Imstadt zu allen weiteren Städten ermittelt. So fährt man zum Beispiel nach Giwelau am besten über Buding und die beiden Autobahnkreuze, die wir mit Z und Y bezeichnet haben.

Beachten Sie bitte auch die „0“, die nun der Vollständigkeit halber bei Imstadt steht, denn der Weg von Imstadt nach Imstadt beträgt ja ganz offensichtlich 0 km.

Warum ist das Ameisen-Prinzip für einen Informatiker interessant?

- Es führt in absehbarer Zeit zum Ziel. Da die Ameisen ständig in Bewegung sind und keine Strecke doppelt gehen, müssen sie recht bald alles erkundet haben (maximal nach der Zeit, die dem kürzesten Weg zur am weitesten entfernten Stadt entspricht).

- Es werden immer wieder die gleichen, sehr einfachen Anweisungen benutzt, um die Ameisen zu steuern:
 1. Teile den Trupp auf und folge allen Routen.
 2. Wenn ein Ort erreicht wird: günstigste Strecke dorthin gefunden, weiter bei 1.
 3. Wenn man einem anderen Trupp begegnet: Strecke verwerfen, Ende.

Solche Verfahren lassen sich (normalerweise) gut und einfach für den Computer umsetzen. Eine solche Vorschrift nennt sich dann *Algorithmus*.

Algorithmus

Wussten Sie, dass die Herkunft dieses Begriffes nie richtig geklärt wurde? Entweder kommt er vom griechischen „arithmos“, was Zahl bedeutet, oder er ist der veränderte Name des persisch-arabischen Mathematikers „Al-Charismi“, der – englisch ausgesprochen – in etwa wie „algorithm“ klingt. Erste Algorithmen wurden übrigens bereits lange vor der Zeitenwende hauptsächlich von griechischen Mathematikern entwickelt: Etwa im 4. Jahrhundert v. Chr. das euklidische Verfahren zur Bestimmung des größten gemeinsamen Teilers zweier Zahlen oder im 3. Jahrhundert v. Chr. das Sieb des Eratosthenes zur Ermittlung von Primzahlen. Beide Algorithmen werden heute noch in Computern eingesetzt!

Algorithmus

Ein Algorithmus ist eine Handlungsvorschrift zur Lösung eines Problems bzw. einer Kategorie von Problemen. Diese Handlungsvorschriften lassen sich im Allgemeinen in ein Computerprogramm umsetzen. Hierfür müssen sie hinreichend genau formuliert sein.

Wie lösen also unsere Routenplaner im Auto das Problem des kürzesten Weges? Enthalten sie einen Ameisen-Simulator und spielen die genaue Vorgehensweise nach?

Sicherlich wäre das vorteilhafter als die ganz zu Anfang beschriebene „Brute-Force-Methode“. Trotzdem ist der Informatiker hier erneut gefragt, das gefundene Verfahren für den Computer zu optimieren.

Können Sie sich vorstellen, wie?

Das Ameisen-Prinzip liegt uns Menschen sehr, weil wir uns gut vorstellen können, wie die kleinen Krabbeltiere die Pfade erkunden. Dadurch erschließt sich auch recht deutlich, warum das Verfahren wirklich die günstigsten Wege hervorbringt.

Wann immer wir so ein „aus dem Leben gegriffenes“ Verfahren im Computer benötigen und es daher in Bits und Bytes umsetzen, hilft wieder das Prinzip der Abstraktion:

Überlegen Sie, was vom Ameisen-Prinzip für die Problemlösung relevant ist: Computer benötigen keine Anschauung, diese Anteile sind also zu eliminieren.

Ob Edsger W. Dijkstra, zuletzt Professor in Texas, 1959 über eine Ameisenstraße lief, ist unbekannt. Er hat jedoch bereits zu diesem Zeitpunkt ein Verfahren vorgestellt, das unser Ameisen-Prinzip im Computer zur Berechnung eines kürzesten Weges nutzt. Es ist bis heute nach ihm benannt.

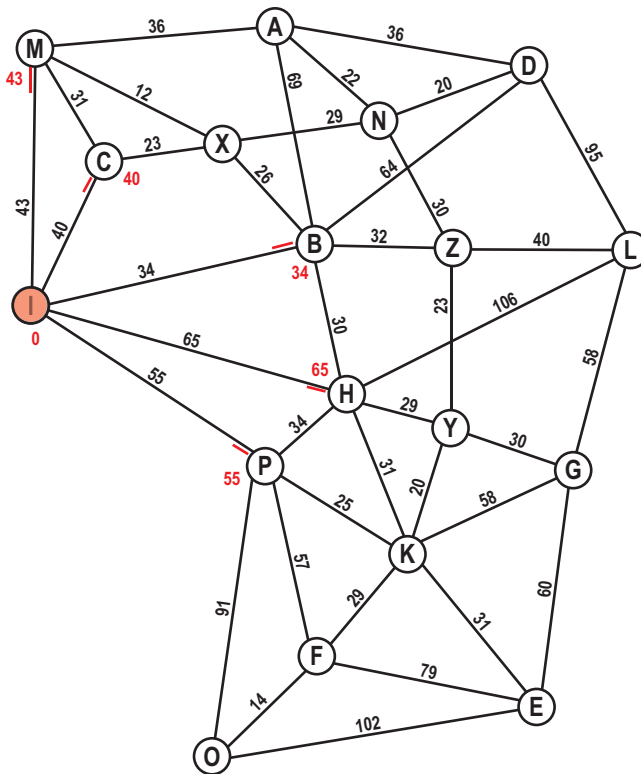
Doch versuchen Sie zunächst einmal selbst, ein solches Verfahren herauszufinden. Vielleicht kommen Sie ja sogar auf eine bessere Methode, die dann nach Ihnen benannt wird.



Der Dijkstra-Algorithmus

Fangen wir noch einmal ganz von vorne an: Sie sitzen in Imstadt und wollen nach Oppenheim.

Die Ameisen unseres letzten Experiments liefen nun einfach in alle anderen direkt erreichbaren Städte, um zu ermitteln, wie lange sie dort hin unterwegs sind. Die hierfür notwendigen Zeiten braucht ein Computer jedoch gar nicht zu ermitteln, er liest einfach die Strecken ab. Wie aus Abbildung 1.11 ersichtlich, sind dies ja einfach die verzeichneten Längen der Verbindungslinien zwischen den Städten.



Edsger Wybe Dijkstra: Er wurde 1930 in Rotterdam geboren, arbeitete nach der Ausbildung zunächst als Professor an der Universität in Eindhoven und wechselte 1984 an die Universität von Texas. Neben seinen Arbeiten zur Berechnung des kürzesten Weges hat er auch einen wesentlichen Beitrag zur Einführung der strukturierten Programmierung geleistet. Dafür erhielt er 1972 auch den Turing-Preis, der das Pendant des Nobelpreises für Informatiker ist. Er starb 2002 in seiner Heimat Nuenen.

Abbildung 1.11 Potenziell erreichbare Städte und die Weglängen dorthin

In der Grafik ist außerdem bei allen Zielpunkten markiert, woher Sie kommen, damit die beschriebene Entfernung zutrifft.

Wie ging es mit den Ameisen weiter? Der Trupp, der zuerst bei einer Stadt ankam, markierte die genommene Strecke als „günstig“ und verteilte sich auf die umliegenden Strecken.

Für den Computer ist nun gar kein Problem, diese Stadt zu bestimmen: Es ist diejenige, die mit der kleinsten Zahl bezeichnet ist, also (B). In Abbildung 1.12 ist dies verzeichnet.

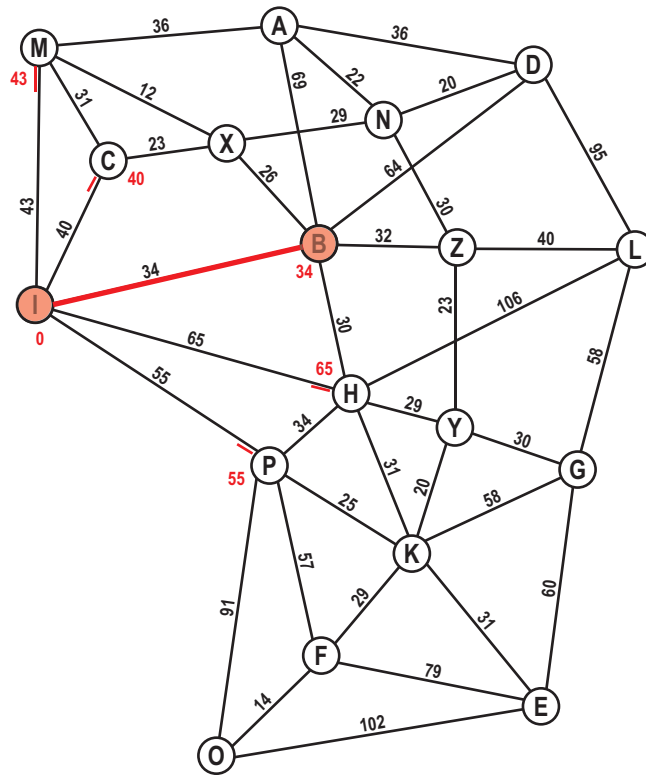


Abbildung 1.12 Der erste kürzeste Weg von Imstadt wurde ermittelt

Von dieser Stadt aus werden wiederum die Entfernungen zu allen Nachbarn bestimmt. Da die Ameisen jedoch bis nach (B) bereits 34 km unterwegs waren, müssen diese addiert werden. Versuchen Sie es! Stoßen Sie auf ein Problem?

Genau! Für (X), (D) und (Z) können wir die Methodik einfach durchführen, aber was ist mit (H)? Hier steht bereits ein Wert!

Wie ist hier zu verfahren? Ziehen wir die Ameisen zurate: Der Weg über (B) nach (H) ist 64 km lang, der direkte Weg von (I) nach (H) 65 km. Der Ameisentrupp von (B) würde also VOR dem Trupp aus (I) ankommen und daher „gewinnen“.

Daher gilt für das Dijkstra-Verfahren die folgende Regel für den Fall, dass an einer Nachbarstadt bereits eine Zahl steht:

- Wenn die Zahl, die neu hingeschrieben werden soll, kleiner ist, dann wird die alte durch die neue Zahl ersetzt und dementsprechend auch die Marke, woher die Ameisen kommen.
- Wenn die neue Zahl größer wäre, passiert gar nichts.

Ergebnis ist daher die Karte aus Abbildung 1.13.

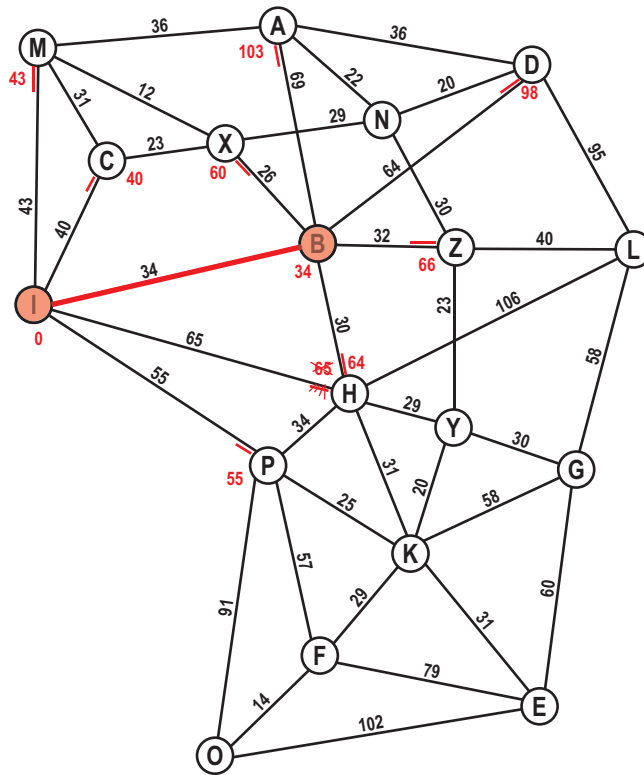


Abbildung 1.13 Potenziell erreichbare Städte und die Weglängen dorthin

Wie geht es nun weiter?

Prinzipiell wie am Anfang: Aus allen mit Zahlen markierten Städten, die noch nicht von Ameisen besucht wurden (noch nicht rot gefärbt), wird die mit der kleinsten Zahl herausgesucht. Dort kommen die Ameisen als Nächstes an.

In diesem Fall ist das (C).

Von (C) aus werden wieder alle benachbarten Städte betrachtet. Nach (M) käme man in 71 km, nach (X) nach 63 km. Beides wird jedoch von der bereits vorhandenen Zahl unterboten, also passiert gar nichts weiter, die nächste nichtmarkierte Stadt mit der kleinsten Zahl wird gesucht. Das Ergebnis sehen Sie in Abbildung 1.14.

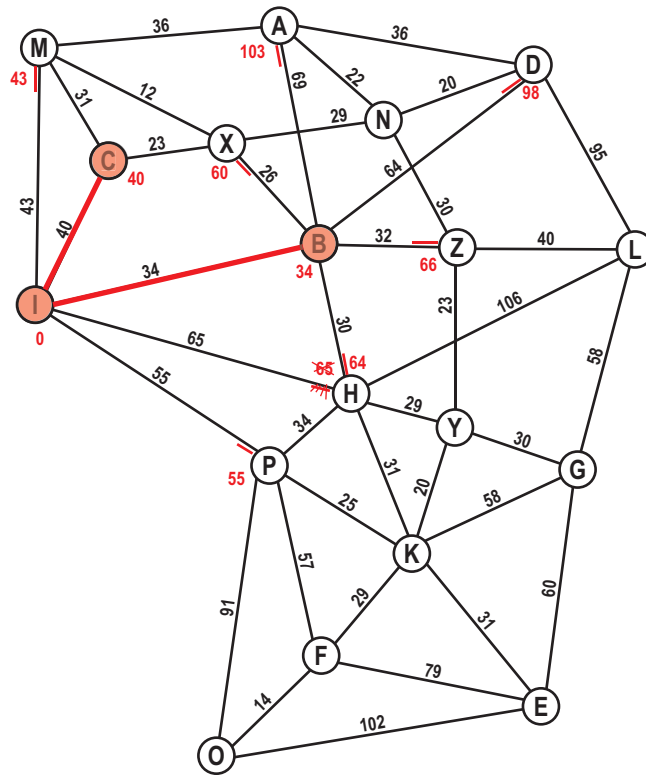


Abbildung 1.14 Erneut wurde ein kürzester Weg gefunden

Bevor Sie nun weitermachen, versuchen Sie doch einmal, den Algorithmus zur Bestimmung kürzester Wege aufzuschreiben.



Algorithmen spielen in der Informatik eine sehr große Rolle: Ein Großteil der vorkommenden Aufgaben aus Wirtschaft und Wissenschaft lässt sich mit Algorithmen lösen, die schon vor längerer Zeit entwickelt wurden (dazu mehr am Ende des Kapitels).

Es gibt auch diverse formale Methoden, die Algorithmen zu Papier zu bringen. Eigentlich geht es jedoch darum, die Beschreibung so genau zu machen, dass jemand anderes den Algorithmus danach durchführen kann.

Behalten Sie das im Hinterkopf, wenn Sie jetzt versuchen, den beschriebenen Dijkstra-Algorithmus insgesamt zu formulieren.



Es gibt natürlich unzählige korrekte Lösungen. Ihre sollte im Kern mit Abbildung 1.15 übereinstimmen.

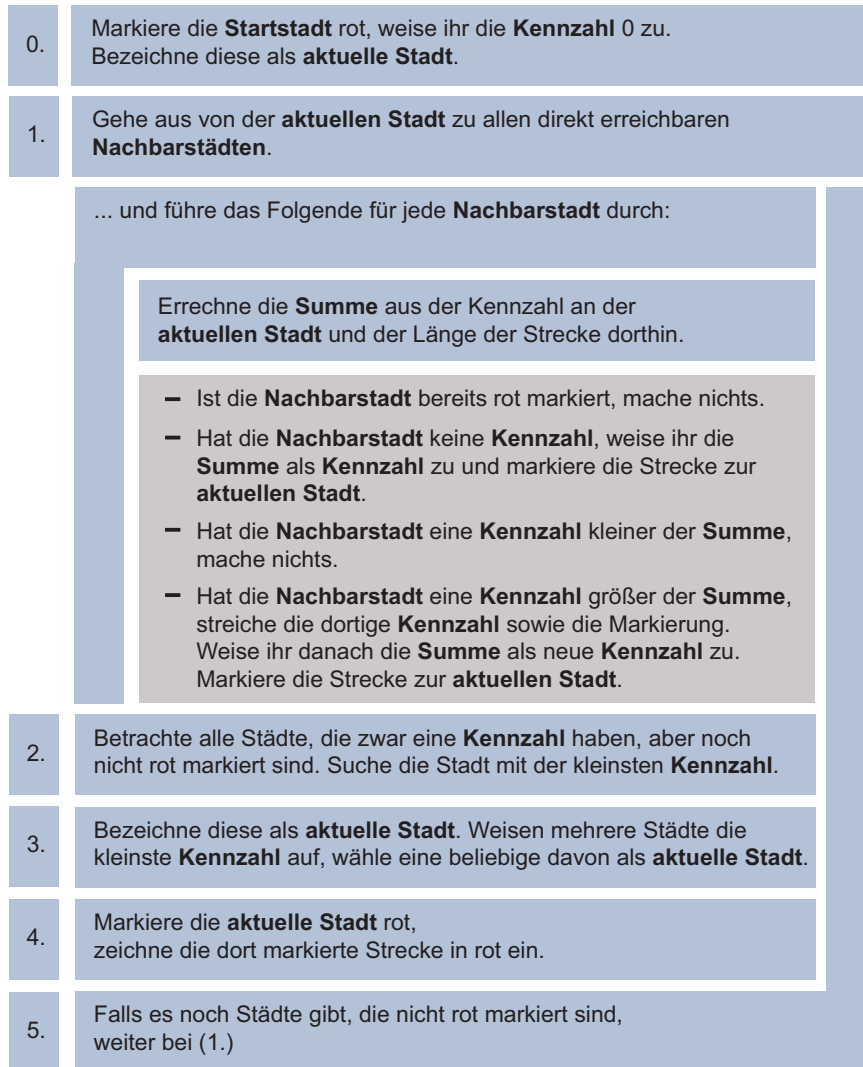


Abbildung 1.15 Der Algorithmus

Führen Sie nun den Algorithmus für das gegebene Beispiel fort.



Das Ergebnis ist in Abbildung 1.16 dargestellt.

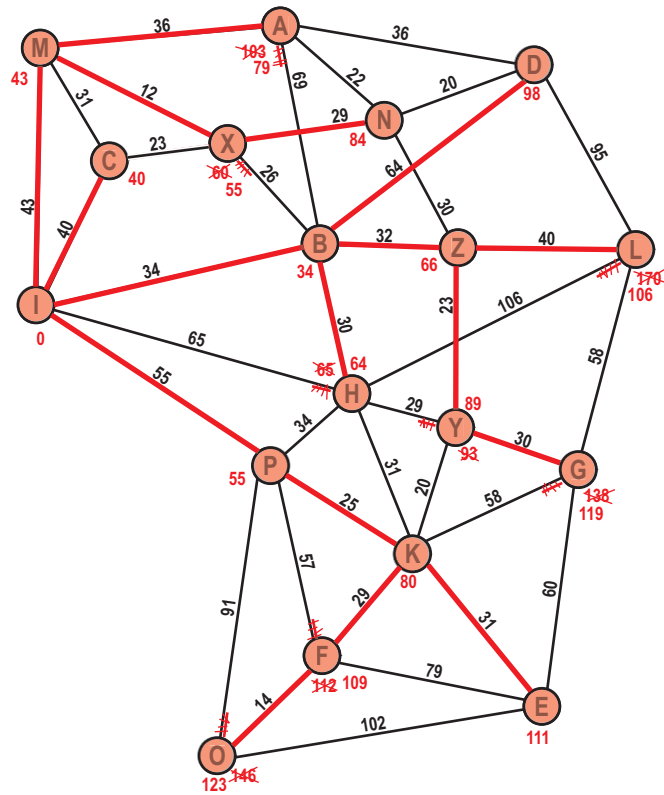


Abbildung 1.16 Alle kürzesten Wege von Imstadt, mit dem Dijkstra-Algorithmus herausgefunden

Erkennen Sie die Gemeinsamkeiten mit dem Ergebnis des Ameisen-Prinzips?

Üben Sie nun den neuen Algorithmus und bestimmen Sie den kürzesten Weg von Lupera nach Eindhofen, nach Giwelu und nach Morbach.

Zur Kontrolle: Die ermittelten Entfernungen sollten 58 km, 110 km und 114 km betragen.



Wenn Sie alles richtig gemacht haben, sollte in etwa Abbildung 1.17 herauskommen.

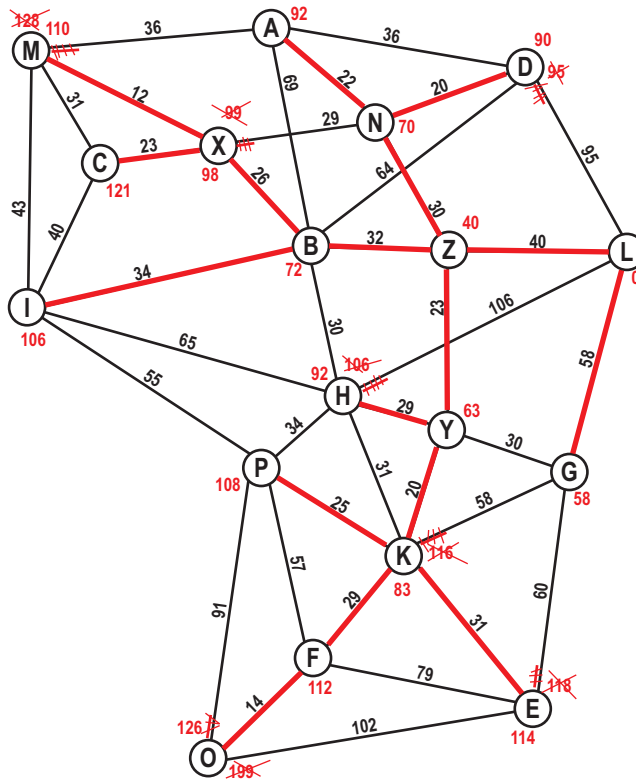


Abbildung 1.17 Die kürzesten Wege von Lupera aus

Ein Routenplaner würde also die Strecke von Lupera nach Eindhofen einzeichnen wie in Abbildung 1.18.

Weil es ab und zu Verwechslungen gibt: Bitte denken Sie daran, dass das dargestellte rote Netz nur die kürzesten Wege von Lupera aus repräsentiert! Wenn Sie es zum Beispiel nutzen würden, um über rote Strecken von Imstadt zum Flughafen zu gelangen, wäre dieser Weg über Budingen, die Kreuze Z und Y sowie Krupsing mit 138 km viel zu weit. Aus Abbildung 1.16 können Sie ablesen, dass dies auch in 109 km zu schaffen ist.

Dijkstra bestimmt immer die kürzesten Wege von einem Startpunkt zu vielen anderen Punkten!

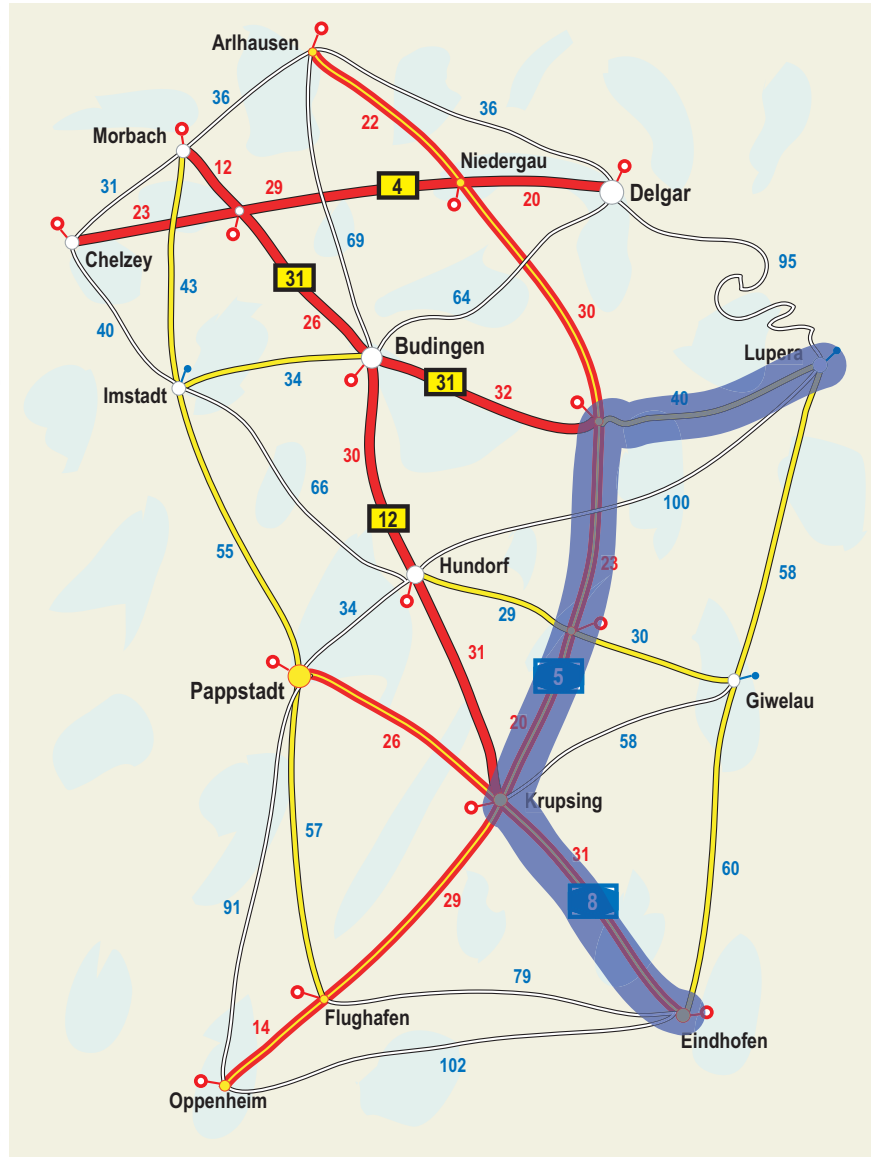


Abbildung 1.18 Landkarte, wie sie vom Routenplaner dargestellt wird

Herzlichen Glückwunsch – Sie wissen nun, wie so ein Routenplaner funktioniert!

Da Sie sicher ganz heiß darauf sind, Ihr neues Wissen gleich praktisch einzusetzen, hier gleich die nächste Aufgabenstellung ...

Die Schilda-Rallye

Sehen Sie hier in Abbildung 1.19 den Touristen-Stadtplan von Schilda, mit seinen acht Hotels und vier Pensionen.

Wie es so kommt, hat der Stadtrat vor kurzem beschlossen, alle Straßen zu Einbahnstraßen zu machen (ganz im Gegensatz zu den traditionellen Gepflogenheiten sind trotzdem noch alle Orte erreichbar...)

Die Fahrzeuge von Schilda-Taxi warten vor den Hotels Adler und Gozo sowie auf dem Parkplatz der Pension Capitol. Aufgrund der neuen Verkehrsführung benötigen die Fahrer einen Plan, wie sie auf dem kürzesten Weg von ihrem Standort zu allen anderen Hotels kommen. Eine Entfernungstabelle ist auch zur Berechnung der neuen Tarife notwendig.

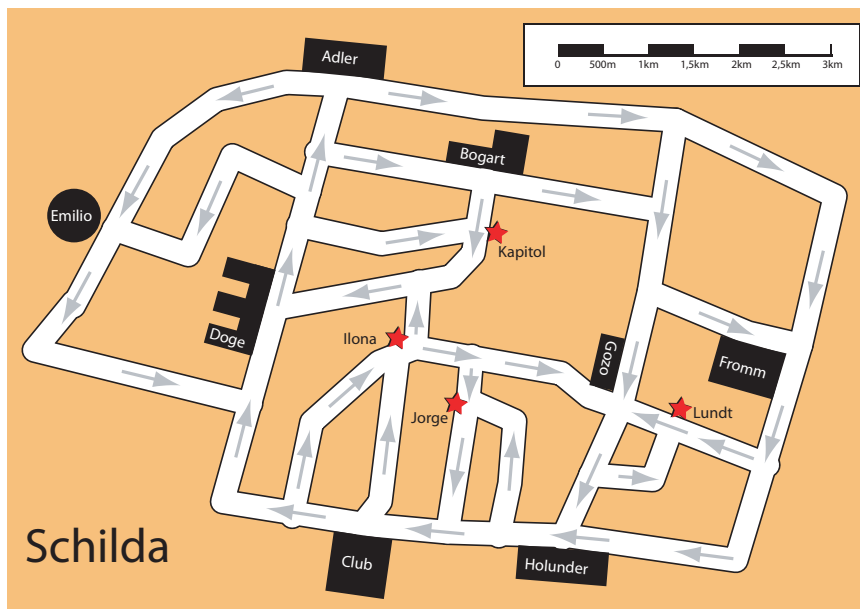


Abbildung 1.19 Stadtplan von Schilda

Ihre Aufgabe: Erstellen Sie drei Tabellen mit den Entfernungen der Hotels Adler und Gozo sowie der Pension Capitol zu allen anderen Gästehäusern.

Tipp: Gehen Sie zunächst genauso vor, wie Sie es gerade gelernt haben. Einen Unterschied gibt es allerdings. Erkennen Sie ihn?

Versuchen Sie, das Problem so weit zu lösen, wie Sie selbst kommen. Wenn es wirklich nicht mehr weitergeht, schauen Sie in die Lösung in den folgenden Absätzen.



Zunächst wird wieder eine Abstraktion benötigt. Die Definition von Orten ist der erste Schritt. Selbstverständlich sind die Positionen aller Hotels relevante Orte. Relevant sind jedoch auch alle Orte, an denen sich die Straße teilt, also ein Abbiegen möglich ist.

In dieser Lösung nach Abbildung 1.20 habe ich die Orte für die Hotels (ihrem Namen entsprechend) A bis L genannt und in der Karte gelb markiert, Straßenkreuzungen ohne Hotel M bis Z mit einer hellgrünen Markierung. Andere Nomenklaturen sind selbstverständlich genauso möglich.

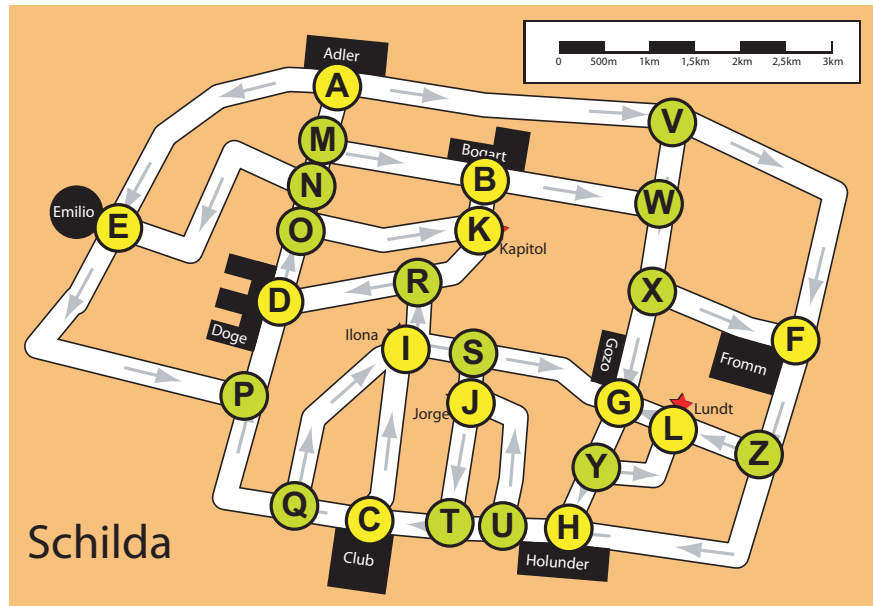


Abbildung 1.20 Schilda mit Markierung aller Straßenkreuzungen

Haben Sie weitere Biegungen als „Orte“ markiert, etwa zwischen (E) und (P) oder zwischen (V) und (F)? Das ist kein Fehler, ist jedoch nicht notwendig, denn ein Auto kann sowieso nur dem Straßenverlauf folgen.

Welches ist der nächste Schritt? Stellen Sie einen Plan auf, ähnlich dem Plan für das letzte Routenplaner-Problem!



Der Unterschied zum letzten Problem sind die Einbahnstraßen – diese wichtige Information muss auch in einer abstrakteren Darstellung der Karte erfasst werden, aber das stellt ja kein Hindernis dar. Abbildung 1.21 zeigt die Lösung.

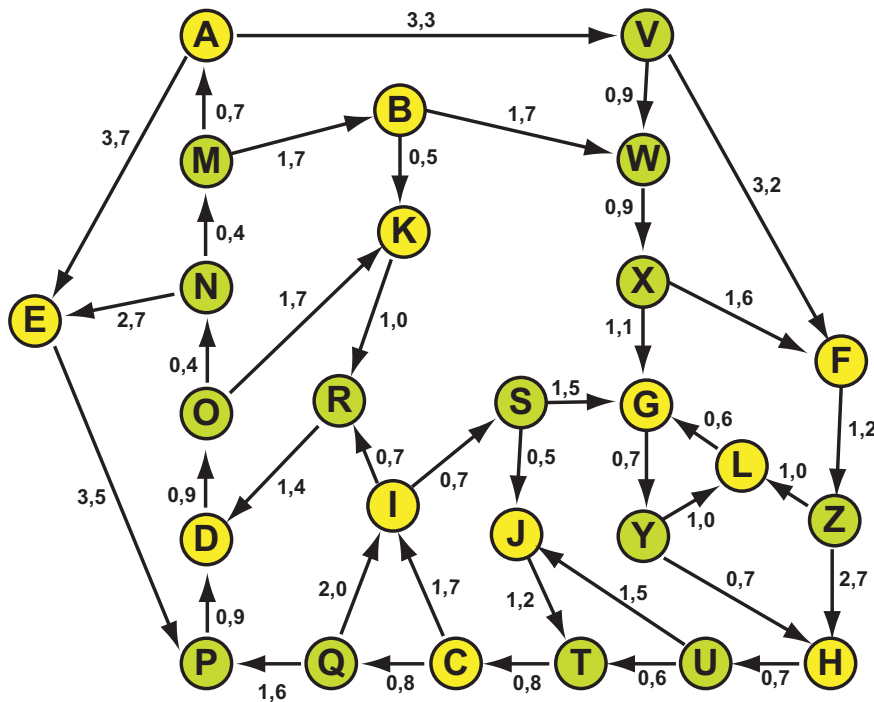


Abbildung 1.21 Schildas Stadtplan abstrakt

Erinnern Sie sich noch an das Prinzip der Ameisen? Es lässt sich immer noch anwenden – nur müssen die Ameisen nun die Regeln der Einbahnstraßen beachten.

Fangen Sie damit an, die Wege-Tabelle vom Hotel Adler aus zu erstellen!



Sie sind ja ganz sicher selbst auf die Lösung nach Abbildung 1.22 gekommen...

Die Entfernungstabelle für das Hotel Adler:

Hotel	Entfernung (km)
Adler	0,0
Bogart	11,5
Club	9,7
Doge	8,1
Emilio	3,7
Fromm	6,5
Gozo	6,2
Holunder	7,6
Ilona	11,4
Jorge	9,8
Kapitol	10,7
Lundt	7,9

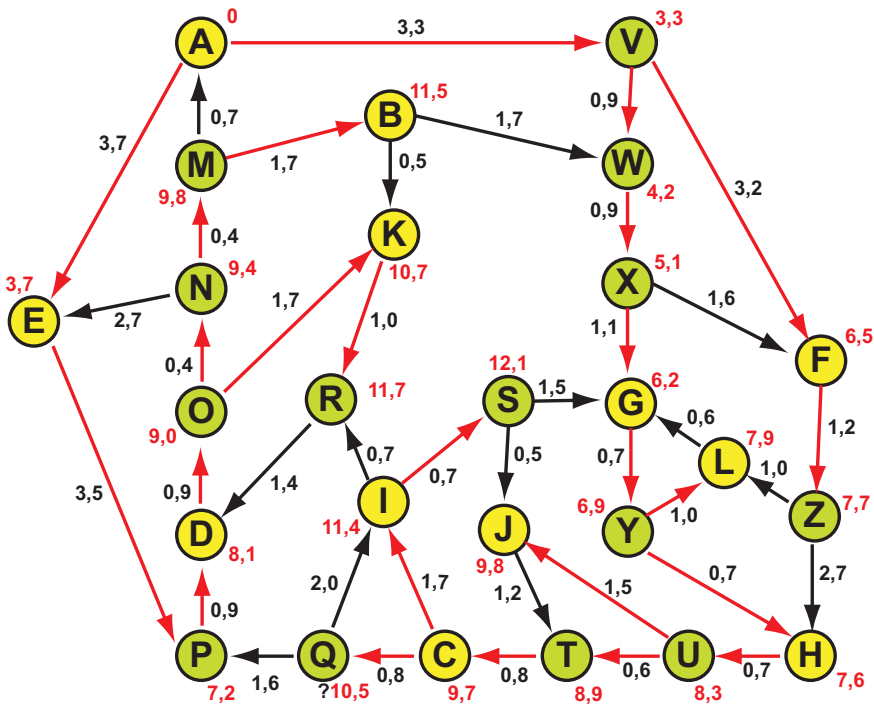


Abbildung 1.22 Kürzeste Wege vom Hotel Adler aus

Üben Sie nun weiter, indem Sie die Entfernungen vom Hotel Gozo zu jedem anderen Hotel und von der Pension Kapitول aus bestimmen.



Die Lösung für das Hotel Gozo entnehmen Sie Abbildung 1.23.

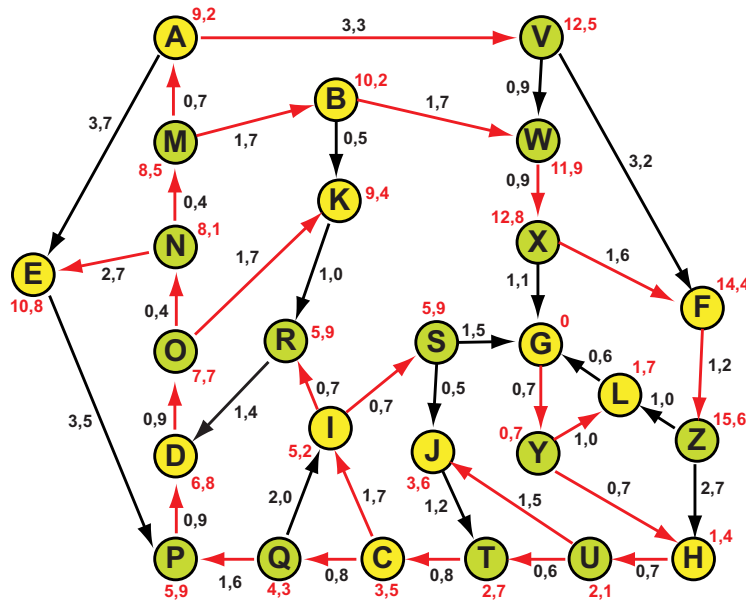


Abbildung 1.23 Kürzeste Wege vom Hotel Gozo aus

Die Entfernungstabelle für das Hotel Gozo:

Hotel	Entfernung (km)
Adler	9,2
Bogart	10,2
Club	3,5
Doge	6,8
Emilio	10,8
Fromm	14,4
Gozo	0,0
Holunder	1,4
Ilona	5,2
Jorge	3,6
Kapitol	9,4
Lundt	1,7

Die Entfernungen von der Pension Kapitel aus entnehmen Sie Abbildung 1.24.

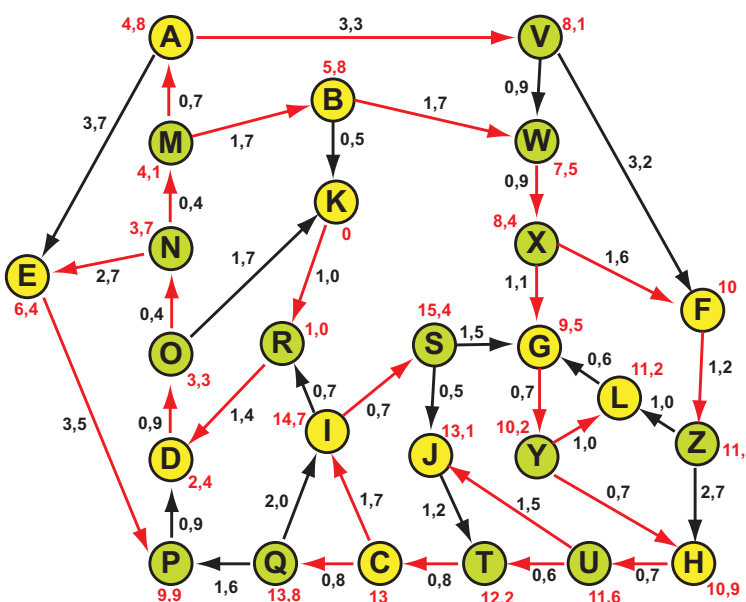
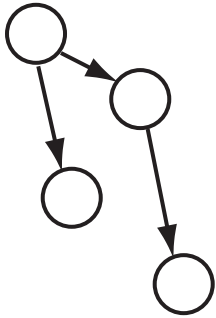


Abbildung 1.24 Kürzeste Wege vom Hotel Kapitel aus

Die Entfernungstabelle für das Hotel Kapitel:

Hotel	Entfernung (km)
Adler	4,8
Bogart	5,8
Club	13
Doge	2,4
Emilio	6,4
Fromm	10
Gozo	9,5
Holunder	10,9
Ilona	14,7
Jorge	13,1
Kapitol	0,0
Lundt	11,2

Was steckt dahinter?



Graph

Die besprochenen Probleme gehören in das Umfeld der so genannten *Graphenalgorithmen*.

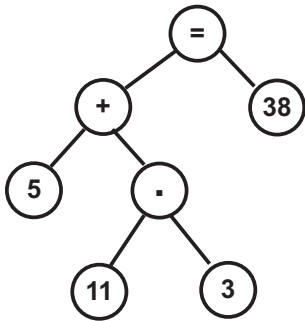
Ein Graph besteht hierbei aus einer Menge von *Knoten* und einer Menge von *Kanten*, die zwischen den Knoten verlaufen.

Graphen werden oft wie in diesem Beispiel dargestellt: die Knoten als Kringel und die Kanten als Linien oder Pfeile dazwischen. Man unterscheidet ungerichtete Graphen, also solche, bei denen die Verbindung zweier Knoten in beide Richtungen geht (ohne Pfeil) und gerichtete Graphen (mit Pfeil).

Wozu ist aber ein Graph gut? Wie andere Modelle der Informatik auch kann der Graph einen Ausschnitt der Wirklichkeit modellieren, um diese einfacher zu verstehen und zu analysieren. Prominentes Beispiel für einen Graphen ist die Landkarte bzw. der Stadtplan, wie hier im Kapitel gezeigt. Knoten sind hierbei die Orte, die Kanten die Verbindungswege. Genauso könnte der Graph jedoch auch ein Computernetzwerk darstellen. Auch andere abstrakte Konzepte wie Produktionspläne oder eine mathematische Gleichung lässt sich per Graph sehr übersichtlich darstellen. In diesem Fall $5 + (11 \cdot 3) = 38$.

Sowohl Knoten als auch Kanten können *Markierungen* tragen. Im Beispiel der Landkarte Ortsnamen für die Knoten und Weglängen für die Kanten. Aufgrund der vielfältigen Anwendungen von Graphen gibt es auch eine Menge von Algorithmen, die dafür existieren. Der besprochene Dijkstra-Algorithmus ist hier ein sehr prominenter Vertreter.

Was zeichnet einen guten Algorithmus aus? Einerseits muss er natürlich die gestellte Aufgabe lösen. Wichtiges Kriterium ist jedoch außerdem, dass er die gestellte Aufgabe schnell löst und so auch bei großen Problemen nicht in die Knie geht.



Mathematische Gleichung

Man rechnet daher aus, wie stark der Zeitbedarf mit der so genannten Problemgröße ansteigt. Auch dies kann man am Beispiel der Landkarte sehr schön demonstrieren. Eine brauchbare Problemgröße ist die Anzahl der Städte auf der Landkarte.

Rekapitulieren wir zunächst den anfänglich kurz erwähnten Brute-Force-Algorithmus zur Bestimmung des kürzesten Weges:

„Bestimme alle möglichen Wege vom Start zum Ziel und suche davon den kürzesten“.

Im schlechtesten Fall müssen also von einem Graphen alle von einem Punkt ausgehenden Wege bestimmt werden. Außerdem sind im schlechtesten Fall alle Knoten mit allen anderen Knoten verbunden (der Fachausdruck dafür ist „vollständiger Graph“). Für drei Knoten ist es noch kein Problem, die Anzahl möglicher Wege vom Startpunkt S aus zu bestimmen. Abbildung 1.25 zeigt die zwei Lösungen.

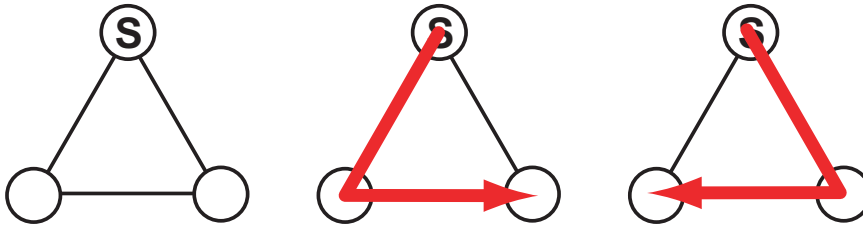


Abbildung 1.25 Es gibt zwei vollständige Wege von S aus

Versuchen Sie das jetzt auch mit den Graphen aus Abbildung 1.26 mit vier bis sieben Knoten.

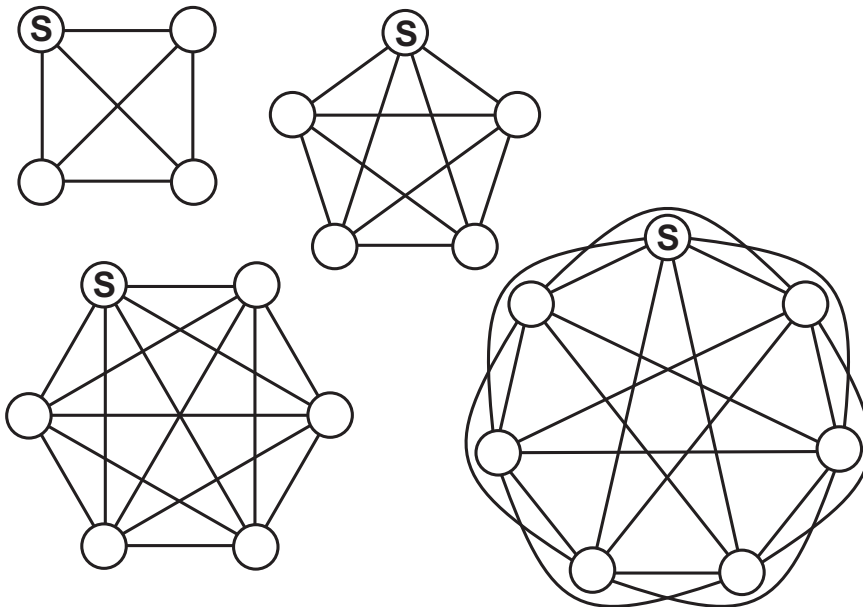


Abbildung 1.26 Graphen mit vier bis sieben Knoten



Wenn Sie versucht haben, die Lösung zeichnerisch zu bestimmen, werden Sie bereits gemerkt haben, wie viel länger die Suche dauert, wenn auch nur ein Knoten hinzukommt. Man kann die Anzahl der Möglichkeiten auch rechnerisch bestimmen:

Für den Graphen mit vier Knoten gilt, dass man ihn aus zwei Komponenten zusammensetzen kann: ein einzelner Knoten (S) plus ein Graph mit drei Knoten. (leer) kann dann mit dem Graphen verbunden werden, wie in Abbildung 1.27 dargestellt.

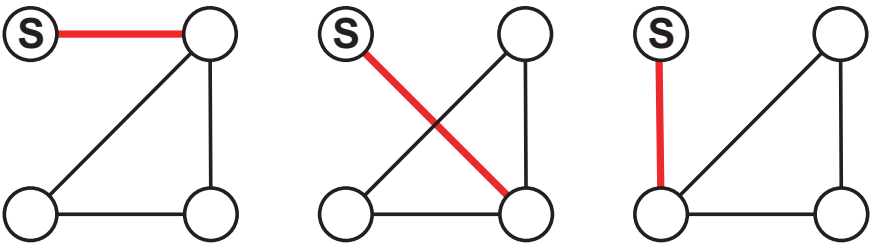


Abbildung 1.27 (S) kann auf drei Arten mit dem vorhandenen Graphen verbunden werden

Da der „bekannte“ Graph drei Knoten besitzt, kann vom „neuen Knoten (S)“ auf drei Arten ein Weg zum bekannten Graphen begonnen werden, nämlich jeweils zu einem der Knoten. Stellen Sie sich vor, dass im 3er-Graphen dann auf die bereits ermittelte Weise ein Weg gesucht wird.

Daher ist die Anzahl möglicher Wege im 4er-Graphen $3 \cdot 2 = 6$.

Für den 5er-Graphen gibt es also vier Möglichkeiten, Wege vom neuen Knoten zum 4er-Graphen zu beginnen. Die Anzahl der Wege insgesamt beträgt daher $4 \cdot 6 = 24$. Auf diese Weise kann man ableiten, dass in einem vollständigen Graphen mit n Knoten $(n-1)!$ verschiedene Wege von einem gesetzten Startpunkt ausgehen.

Da für jeden der Wege $n-1$ Streckenabschnitte eingerechnet werden müssen, bedarf es für den Brute-Force-Algorithmus ungefähr $(n-1)(n-1)!$ Berechnungen, um den kürzesten Weg zu finden. Die folgende Tabelle enthält die Anzahlen für ein paar Problemgrößen.

# Knoten	Schritte
3	4
4	18
5	96
6	600
7	4.320
8	35.280

# Knoten	Schritte
9	322.560
10	3.265.920
15	1.220.496.076.800
20	2.311.256.907.767.808.000
50	298058113376791104827403560027434 73467490088737581301760000000000

Jens Gallenbacher: Abenteuer Informatik, 3. Aufl., ISBN 978-3-8274-2965-0, © Springer-Verlag Berlin Heidelberg 2012
Zur Verwendung für Online-Assessment der TU Wien und Universität Wien

In diesem Fall wächst der Aufwand zur Lösung *exponentiell* an, das bedeutet, wenn n die Problemgröße ist, kann man den Aufwand nur noch mit x^n nach oben hin abschätzen (mit einem beliebigen x).

Optimal wäre ein linearer Aufwand, das bedeutet, dass zur Lösung eines Problems der Größe n der Aufwand $x \cdot n$ beträgt (mit konstantem x). Leider gibt es kaum Algorithmen, die so gut sind. Betrachten wir einmal den von Dijkstra:

Vom Startknoten aus werden alle benachbarten Knoten (also die, zu denen eine Verbindung besteht) betrachtet. Bei einem vollständigen Graphen mit n Knoten sind das $(n-1)$. Ein Beispiel mit 7 Knoten sehen Sie in Abbildung 1.29.

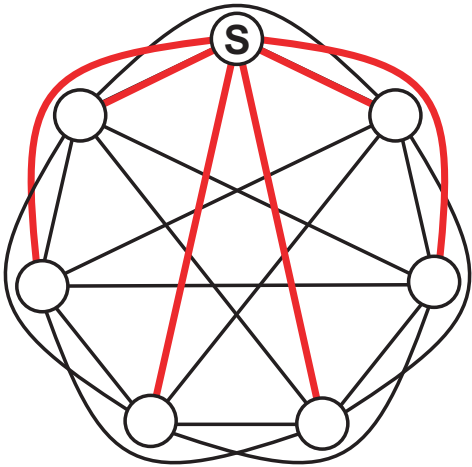


Abbildung 1.29 Vollständiger Graph mit 7 Knoten

Danach ist der Startknoten vollständig „aus dem Rennen“, er wird nicht mehr betrachtet. Nun wird der Knoten mit der kleinsten Markierung gesucht und von dort geht es wiederum zu allen benachbarten Knoten – das sind $(n-2)$. Danach ist auch dieser „aus dem Rennen“.

Dies setzt sich fort. Daher ist der Aufwand für den Dijkstra-Algorithmus

$$(n-1) + (n-2) + (n-3) + \dots + (1)$$

Wir können dies wiederum in einer Tabelle dokumentieren.

# Knoten	Schritte
3	3
4	6
5	10
6	15
7	21
8	28

# Knoten	Schritte
9	36
10	45
15	105
20	190
50	1.225

Bei 12.903 Knoten ergeben sich 83.237.253 Rechenschritte. Das erledigt ein moderner Bürorechner ohne große Schwierigkeiten. An der Tabelle kann man ablesen, dass die Rechenzeit auch schneller als linear mit der Problemgröße anwächst, aber lange nicht so stark wie bei Brute-Force.

Sie erkennen, dass in der Informatik manchmal scheinbare Lösungen nicht praktikabel sind, weil sie bei realen Aufgabenstellungen zu viel Rechenzeit beanspruchen. Ein Informatiker muss daher fast immer die so genannte Komplexität einer Lösung abschätzen. Die Komplexität gibt an, wie stark die Rechenzeit in Relation zur Problemgröße anwächst. Im Falle von Brute-Force ist die Komplexität n^n , während Dijkstra eine Komplexität von n^2 aufweist.

Die Informatik unterscheidet dabei verschiedene Klassen von Problemen:

Alles, was sich in polynomieller Laufzeit lösen lässt, gehört zur Klasse „P“. Das bedeutet, die Laufzeit steigt im Verhältnis zur Problemgröße in einem Verhältnis an, das sich durch ein Polynom ausdrücken lässt. Die Problemgröße steht also im Fall von Hoch-Rechnung immer unten.

Probleme mit polynomieller Laufzeit könnten etwa n^2 , $n^4 + n^2 + n$, $n \log n$ oder auch n^{100} als Komplexität haben.

Darüber hinaus gibt es die Klasse der „Nicht praktisch berechenbaren“ Probleme oder „NP“. Diese haben eine Laufzeit, die sich nicht durch ein Polynom ausdrücken lässt. Das bedeutet, dass in der Komplexitätsfunktion die Problemgröße irgendwo in einem Exponenten auftaucht.

NP-Komplexitäten sind 2^n , $n!$ oder n^n .

Allerdings haben NP-Probleme noch eine gute Seite: Man kann eine Antwort raten und dann in polynomieller Zeit überprüfen, ob diese Antwort korrekt ist.

Zu welcher Klasse von Problemen gehört das gerade besprochene Problem des kürzesten Weges? Selbstverständlich in die Klasse „P“, denn mit dem Dijkstra-Algorithmus existiert eine Lösung, die in polynomieller Laufzeit arbeitet.

Übrigens gilt dies nicht für das alte „Traveling Salesman“-Problem:

„Ein Vertreter aus Darmstadt muss Kunden in den Städten Berlin, Chemnitz, Frankfurt, Hamburg, Hannover, Marburg, München und Stuttgart besuchen. Bestimmen Sie eine Route, die von seinem Wohnort durch alle genannten Städte und zurückführt und die kürzer ist als 1000 km!“.

Für dieses Problem gibt es immer noch keinen Algorithmus mit polynomieller Laufzeit. Vielleicht finden Sie ja einen ... Allerdings ist auch einzusehen, dass es sich um ein NP-Problem handelt: Rät man eine Strecke, kann man in polynomieller Zeit feststellen, ob sie der Bedingung „kürzer als 1000 km“ genügt.

Eine weitere Klasse von Problemen ist nicht in polynomieller Zeit berechenbar und selbst wenn man eine Lösung rät, kann man die Richtigkeit nicht in polynomieller Zeit überprüfen. Diese Erkenntnisse gehen jedoch weit über die praktischen Anwendungen der Informatik hinaus, daher möchte ich hier im Buch nicht weiter darauf eingehen!

Eine der größten Fragen der Mathematik und Informatik ist immer noch kurz gesagt „ $P = NP?$ “, also ob prinzipiell alle Probleme der NP-Klasse auch in polynomieller Zeit lösbar sind (und nur noch kein entsprechender Algorithmus gefunden wurde), oder ob es Probleme gibt, die grundsätzlich nicht in polynomieller Zeit lösbar sind. Es gibt immer wieder Veröffentlichungen in dieser Richtung, aber sie haben sich bisher als informationstechnische Pendants der Kernfusion im Reagenzglas herausgestellt (also als Flop).

Wenn Ihnen jetzt P und NP noch völlig unklar sind, dann machen Sie sich keine Sorgen – fast allen Informatikern geht es ähnlich!

$P=NP?$

Die Lösung dieser Frage ist momentan der heilige Gral der Informatik – oft gesucht, (bisher) nicht gefunden. Viele Probleme sind sehr ähnlich zu solchen, die bereits in polynomieller Zeit gelöst wurden, es ist aber für sie selbst keine entsprechende Lösung bekannt.



Der schnellste Weg und weitere Anwendungen

Kann man nun den Dijkstra-Algorithmus immer nur zur Berechnung kürzester Wege verwenden?

Ein Informatiker versucht natürlich, für einen so schönen Algorithmus weitere Anwendungen zu finden, denn es ist ja effektiver, für neue Aufgaben bereits bekannte (und bewährte) Verfahren zu adaptieren, als das Rad immer wieder neu zu erfinden.

Betätigen Sie sich doch einmal in dieser Disziplin und versuchen Sie, die folgenden Aufgabenstellungen mit dem Dijkstra-Algorithmus zu lösen.

Wie sieht es etwa aus, wenn wir für unsere erste Karte nicht den kürzesten, sondern den schnellsten Weg suchen?

Dabei gehen wir mal davon aus, dass wir mit dem Auto auf der Autobahn im Schnitt 120 km/h schaffen, auf der Landstraße (rot und gelb) jeweils 80 km/h und auf den kleinen Gemeindestraßen (weiß) nur 50 km/h. Zusätzlich veranschlagen wir für jede Ortsdurchfahrt pauschal 10 Minuten (der Einfachheit halber auch bei den Orten, durch die wir auf der Autobahn fahren, auch hier gibt es ja oft Tempolimits aufgrund Lärmschutz etc.)!

Finden Sie den schnellsten Weg von Imstadt zu jeder anderen Stadt!



Mit dieser Karte kann dann der ganz normale Dijkstra-Algorithmus durchgeführt werden. Das Ergebnis sehen Sie in Abbildung 1.31.

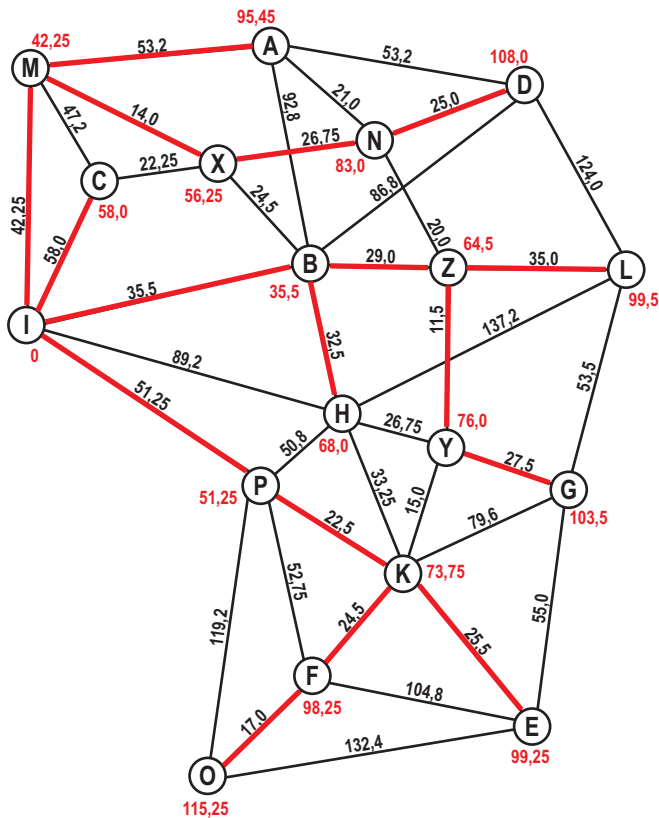


Abbildung 1.31 Lösung des Schnellste-Wege-Problems

Wenn Sie das Ergebnis mit der Lösung für die kürzeste Wegstrecke vergleichen, stellen Sie fest, dass es recht ähnlich, aber doch nicht identisch ist.

Resümee

Wie das vorige Beispiel gezeigt hat, kann man mit Dijkstra also nicht nur den kürzesten Weg, sondern allgemein den „besten“ Weg berechnen, wobei man etwas Beliebiges als „gut“ bezeichnen kann. Etwas erweitert (dann als „Bellmann-Ford-Algorithmus“ bekannt) können unter bestimmten Bedingungen auch maximale Wege berechnet werden.

Neben der offensichtlichen Nutzungsmöglichkeit in Routenplanern sind Anwendungen zum Beispiel beim Verschieben von Daten durchs Internet gegeben: Auch hier müssen die Vermittlungsstellen, die so genannten Router, einen günstigen Weg durch das Wirrwarr aus Leitungen finden – doch dazu mehr in einem der folgenden Kapitel.

Literatur



- 1 Buch: T. Ottmann, P. Widmayer „**Algorithmen und Datenstrukturen**“; Spektrum Akademischer Verlag; 3. Auflage 1996; ISBN 3-8274-0110-0

Nicht nur Näheres zum **Dijkstra-Algorithmus** lässt sich hier nachlesen, sondern viele andere Grundlagen der Informatik werden hier anschaulich, aber mit der für ein Lehrbuch notwendigen mathematischen Genauigkeit dargelegt.



- 2 Buch: Robert Sedgewick „**Algorithms in Java, Part 5, Graph Algorithms**“; Addison-Wesley International; Juli 2003; ISBN 0-2013-6121-3

Sedgewick beschreibt recht anschaulich, wie viele grundlegende Algorithmen der Informatik funktionieren und wie man sie in der bekannten und verbreiteten Programmiersprache Java umsetzt. Für diejenigen, die die Informatik nicht nur mit Papier und Bleistift nachvollziehen möchten! Leider ist dieser Teil – im Gegensatz zu den Teilen 1–4 – momentan nicht auf Deutsch übersetzt erhältlich.



- 3 Buch: Peter Gritzmann, René Brandenberg „**Das Geheimnis des kürzesten Weges**“; Springer; 2. Auflage 2003; ISBN 3-540-00045-3

Hier sind Wegeprobleme und die damit verbundenen Fragestellungen Thema eines ganzen, unterhaltsamen Buches. Sie erfahren in Romanform noch Weiteres über die mathematischen Zusammenhänge.



- 4 Veröffentlichung: Edsger W. Dijkstra „**A note on two problems in connexion with graphs – Numerische Mathematik**“; Vol 1.; S. 269–271

Für Interessierte: Dies ist der Originalartikel zum Dijkstra-Algorithmus.



- 5 Buch: Michael R. Garey, David S. Johnson „**Computers and Intractability**“; W. H. Freeman & Company; 1979; ISBN 0-7167-1045-5

Eine gelungene Einführung in die Grundlagen und Hintergründe des bekannten $P = NP$ Problems. Leider nur auf Englisch erhältlich!



AIIESEC-IAESTE

REDAKTION

AKAKRAFT-1923

IBZ

ELEKTROTECHNISCHER-VEREIN

STUDENTENVERB.

HRZ-TELEFONZENTRALE

HAUSHALTSTELLE

PERSONALRAT

KASSE

Bibliothek-Institut für Pädagogik

AKKURAT

ZENTR. TECHN. DIENST

AKKURAT

Lehrstuhl

Lehrstuhl für Pädagogik

2. Ordnung muss sein ...

Einführung

Ordnung ist das halbe Leben. Schön ist jedoch, wenn man nicht das halbe Leben damit zubringen muss, sie zu halten! Wie gut, dass es heute Computer gibt, die einen darin unterstützen.

Wie aus dem deutschen Kunstwort „Informatik“ schon ersichtlich wird, hat das Geschäft eines Informatikers sehr viel mit Information zu tun: Diese sollen nicht einfach nur irgendwo abgelegt werden, sondern man möchte sie auch möglichst schnell wieder auffinden.

Das Grundproblem unterscheidet sich dabei nicht von der Frage, die auch für das Arbeitszimmer, die Küche und andere Bereiche des Lebens gilt: „Wer Ordnung hält, ist nur zu faul zum Suchen“ oder (etwas vornehmer ausgedrückt) „Der Aufwand, Ordnung zu schaffen, muss dadurch gerechtfertigt werden, dass man daraus erhöhten Komfort zieht“, zum Beispiel mehr Platz, schnelleres Auffinden von Dingen usw.

Normalerweise versucht man daher, das Chaos erst gar nicht aufkommen zu lassen – jeden neuen Gegenstand gleich an den richtigen Platz zu setzen. In verschiedenen Situationen funktioniert das jedoch nicht: Post kommt meistens als ungeordneter Stapel an, Spielkarten werden gemischt ausgegeben usw. Hier wird eine vollständige Sortierung notwendig. In diesem Kapitel beschäftigen wir uns damit, wie Computer die Aufgabe lösen, eine unsortierte Menge von Daten so zu organisieren, dass der Zugriff auf einen einzelnen Datensatz sehr schnell geht.

Um das wiederum experimentell nachzuvollziehen, eignen sich Spielkarten sehr gut: Jeder hat Erfahrung damit, hat schon einmal die zugeteilten Karten auf der Hand sortiert, um dann im Doppelkopf, Rommee oder Canasta präziser und schneller spielen zu können. Im Anhang finden Sie ein paar ganz spezielle Spielkarten zum Ausschneiden, die statt Farben und Bildern Zahlen enthalten, denn dies sind die Größen, mit denen ein Computer meistens rechnen muss: Geburtsdaten und Sozialversicherungsnummern für Rentenauszahlung, Matrikelnummern von Studierenden für die Prüfungsverwaltung, Kontonummern und viele mehr. Bereiten Sie also jetzt das Kartenspiel vor, indem Sie die je 16 Karten auf den acht Seiten ausschneiden. Mischen ist erst einmal nicht notwendig!

Das Sortierproblem

Bei allen Aufgaben, die man mit dem Computer lösen möchte, ist ein guter Ansatz, sie erst einmal manuell zu erforschen, um ein Gefühl dafür zu bekommen und die Problemstellung zu erfassen. Wie wir beim Auffinden des kürzesten Weges im letzten Kapitel gesehen haben, ist dabei oft die Problemgröße relevant. Das ist in unserem Fall die Anzahl zu sortierender Objekte.

Machen Sie daher einen ersten Versuch:

Nehmen Sie etwa die Hälfte der Karten und sortieren Sie sie nach der blauen Zahl. Ergebnis sollte eine zusammenhängende Reihe von Karten (ohne Lücken) auf Ihrem Schreibtisch sein. Falls hier nicht genug Platz ist, dürfen selbstverständlich auch mehrere Reihen untereinander die Sortierung ergeben.

Eigentlich war das aber nur die Vorarbeit für den eigentlichen Versuch: Mischen Sie die sortierten Karten wieder (so, dass immer noch alle richtig herum liegen). Bilden Sie dann sechs Stapel zu jeweils 20 Karten, die Sie bereitlegen. Sie benötigen außerdem eine Uhr mit Sekundenangabe oder eine Stoppuhr: Messen Sie die Zeit, die Sie zum Sortieren des ersten Stapels nach den blauen Zahlen benötigen und schreiben Sie das Ergebnis auf.

Danach sammeln Sie die Karten wieder ein, nehmen einen weiteren Stapel hinzu, so dass es jetzt 40 Karten sind. Sortieren Sie nun nach den roten Zahlen. Auf diese Weise sparen Sie das Mischen zwischendurch, weil die roten Zahlen unsortiert sind, wenn die blauen sortiert sind und umgekehrt. Notieren Sie auch hier die benötigte Zeit. Wiederholen Sie das mit 60, 80, 100 und 120 Karten.

Stellen Sie nun die Ergebnisse grafisch dar: Nehmen Sie ein Stück kariertes Papier und übertragen Sie die nebenstehende Abbildung oder kopieren Sie diese einfach aus dem Buch! Machen Sie dann für jeden Ihrer Messwerte ein Kreuz über der entsprechenden Anzahl von Karten (20, 40, 60, ...) in der Höhe, die der benötigten Zeit zum Sortieren entspricht.

Verbinden Sie die Kreuze mit Linien!

Vielleicht haben Sie ja auch noch willige „Opfer“ in Ihrer Umgebung, mit denen Sie den gleichen Versuch durchführen können: Lassen Sie sie zunächst eine Anzahl von Karten (ungefähr 60) sortieren, um den Vorgang zu üben und messen Sie dann die Zeit zum Sortieren von 20, 40, 60 usw. Karten. Übertragen Sie die Ergebnisse möglichst jeweils mit einer eigenen Farbe in Ihr Diagramm. Ein Beispiel dafür sehen Sie in der Abbildung 2.1.

Nehmen Sie sich nach jedem Versuch auch kurz die Zeit, um die Strategie des Sortierers mit ein paar Stichworten festzuhalten.

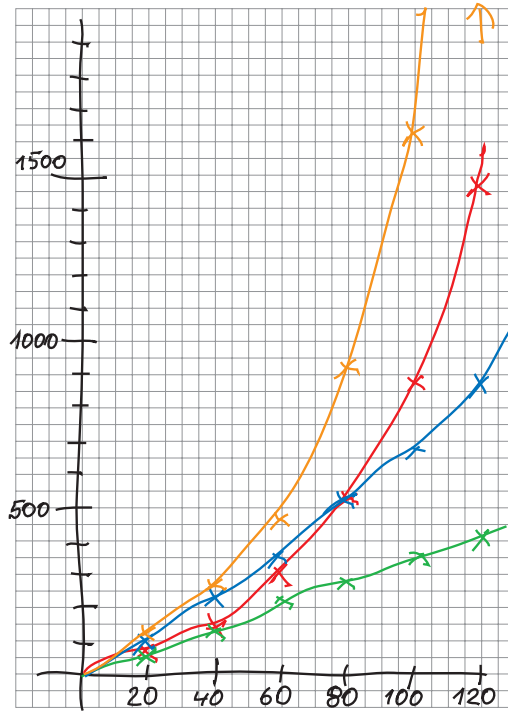


Abbildung 2.1 Graph Anzahl der Karten vs. Zeit in Sekunden

Was können wir aus dem Diagramm lesen?

Die erste Information ist, dass die Zeit zum Sortieren mit der Anzahl der Karten im Allgemeinen ansteigt. Das ist jetzt noch keine bahnbrechende Entdeckung, denn das kennen wir von allen möglichen alltäglichen Vorgängen: Das Spülen von 20 Tellern dauert auch länger als das von zehn Stück. Aber vielleicht gibt es ja noch andere Faktoren für den Zeitbedarf? Allgemein ist für die Informatik die Identifizierung einer Problemgröße sehr wichtig.

Die Problemgröße

Ein wichtiger Vorgang bei der Lösung von Aufgaben aus der Informationstechnik ist das Bestimmen der so genannten Problemgröße. Sie stellt ein Maß dar, wie schwierig bzw. umfangreich die Aufgabe bzw. das Problem ist. Daher hängt von der Problemgröße entscheidend der zu leistende Aufwand ab.

In vielen Fällen ist die Problemgröße recht offensichtlich – wie die Anzahl von Städten einer Landkarte für die Bestimmung eines kürzesten Weges. In anderen Fällen müssen wir erst einmal darüber nachdenken, und die Problemgröße hängt nicht nur von der Aufgabe selbst ab, sondern auch davon, was sich im Allgemeinen bei gleich bleibendem Aufgabentyp an der tatsächlichen Aufgabenstellung ändert:

Beim Sortieren von Objekten ist ein wesentlicher Faktor die Anzahl der Objekte! Das konnten wir in unserem kleinen Experiment feststellen. Aber auch die Größe der zu sortierenden Elemente könnte als Problemgröße dienen: Es ist sicherlich aufwendiger, 100-stellige Zahlen zu sortieren als 6-stellige...

Hier müssen wir uns überlegen, was in einer Aufgabenstellung aus der Realität normalerweise feststeht und was variabel ist. Die Größe der zu sortierenden Elemente ist dabei meistens recht wenig flexibel: Geburtsdaten, Kontonummern, Namen usw. haben eine recht definierte Länge. Wenn wir als IT-Unternehmen ein neues Verfahren für die Sortierung von Daten anbieten, könnten wir also mit dem Slogan „Wir sortieren Kontonummern mit der doppelten Länge in der gleichen Zeit wie unsere Konkurrenten“ sicherlich weniger punkten als mit der Aussage „Wir sortieren die doppelte Anzahl von Kontonummern in der gleichen Zeit wie unsere Konkurrenten“.

Für das Sortieren nehmen wir daher im Folgenden immer die Anzahl der Objekte als Problemgröße an.

Problemgrößen und Aufwand

Von der Problemgröße hängt der Aufwand zum Erfüllen der Aufgabe ab. Wie verhält sich das denn nun in unserem Sortier-Experiment? Normalerweise geht man ja intuitiv vom so genannten linearen Verlauf aus: Pro Objekt wird eine bestimmte Zeit benötigt – zum Spülen von 100 Tellern benötigt man also 100-mal die Zeit, die für einen einzelnen Teller notwendig ist. Unter Umständen sogar etwas weniger, weil man ja langsam Übung bekommt und so für die letzten Teller kürzer braucht.

Schauen Sie sich daraufhin einmal Ihr Diagramm bzw. Ihre Zahlen an. Benötigten Sie zum Sortieren von 40 Karten in etwa doppelt so lange wie für 20? Haben Sie 100 Karten in der fünffachen Zeit von 20 sortiert? Im Beispiel wird das von der grünen Linie ausgedrückt. Wenn Sie das wirklich geschafft haben, sind Sie schon ein Spitzen-Sortierer und heben sich von 99 % aller anderen positiv heraus.

Normal ist, wenn Sie für die Sortierung von 40 Karten knapp dreimal so lange benötigen wie für 20, nicht doppelt. Für 60 Karten ist die fünf- bis sechsfache Zeit fällig usw. Das sieht man im Beispiel in etwa an der roten und der blauen Linie. Manche Sortierer verdoppeln jeweils sogar die benötigte Zeit, wenn sie einen weiteren Stapel Karten mitsortieren. Das trifft ungefähr für die gelbe Linie zu.

Versuchen Sie selbst, dieses Phänomen zu erklären! Was ist der entscheidende Unterschied zwischen dem Sortieren von Karten und dem Spülen von Tellern?



Der Unterschied lässt sich auf eine sehr grundlegende Methode der Informatik zurückführen: „divide et impera“ oder zu deutsch „teile und herrsche“. Diese Methode ist allerdings gar nicht so neu, sondern wird von fast jedem seit Jahrhunderten für die Erledigung alltäglicher Arbeiten genutzt. Lesen Sie selbst:

Das Prinzip „divide et impera“

Sehr häufig hat man im Leben Probleme zu lösen, die zu unüberschaubar und groß sind, um sie in einem Ansatz zu lösen. Vielmehr teilen wir das Gesamtproblem auf in mehrere, handhabbare Stücke, die wir lösen. Die Teillösungen werden danach nur noch zusammengefasst.

Dieses Prinzip wird auch in der Informatik sehr stark verwendet: Ein Programm zerlegt die gestellte Aufgabe zunächst in mehrere kleinere Einheiten, genannt Teilprobleme (divide = teile) und weist danach andere Programme an, diese zu lösen (impera = herrsche, befehle). Dabei ist sehr wichtig, dass die Teilprobleme unabhängig voneinander gelöst werden können, denn sonst müssten die Programme miteinander kommunizieren, unter Umständen auf Lösungen voneinander warten, was den Aufwand wiederum sehr erhöht.

Divide et impera

ist weder eine Erfindung der Informatiker noch etwas, das (wie man bei lateinischen Sprüchen denkt) bereits die Römer in die Welt setzten. Der Satz wird König Ludwig XI. von Frankreich (1423 – 1483) zugeschrieben, der ihn als Maxime hatte. Es handelte sich ursprünglich um eine Militärstrategie, den Gegner zu entzweien, um ihn dann leichter zu beherrschen.

Die Informatik hat ihn dann für ihre friedlichen Zwecke umfunktioniert: ein Programm beherrschbarer zu machen, indem man es einteilt.

Unterschiedliche Probleme können unterschiedlich gut aufgeteilt werden. Das Tellerwasch-Problem stellt hier keine Herausforderung dar:

„100 Teller waschen“ hat direkt als Teilproblem „1 Teller waschen“. Dieses wird 100-mal ausgeführt. Jeder Teller kann unabhängig voneinander gewaschen werden, der Gesamtaufwand besteht also in der Summe der einzelnen Aufwände, also 100-mal der Zeit, die zum Waschen eines Tellers benötigt wird. Man kann die Zeit sogar durch Parallelisierung optimieren: Stellen wir 100 Tellerwäscher ein, schaffen wir das Problem in der Zeit, die man zum Waschen eines einzelnen Tellers benötigt...

Wie sieht das aber beim Sortieren von Karten aus? Probieren wir, auch dieses Problem auf die gleiche Weise aufzuteilen. Wir gehen davon aus, pro Arbeitsschritt eine (unsortierte) Karte auf die Hand zu nehmen, die wir dann in die Folge bereits sortierter Karten einordnen.

„100 Karten sortieren“ hat direkt als Unterproblem „1 Karte sortieren“. Wie lange benötigen wir aber, um eine Karte zu sortieren? Das hängt sehr stark von der Anzahl Karten ab, die bereits auf dem Tisch liegen!

Die erste Karte geht sehr schnell, wir können sie einfach hinlegen. Bei der zweiten Karte müssen wir bereits die Entscheidung treffen, ob sie links oder rechts angelegt wird – der Aufwand hat sich vergrößert.

Befinden sich bereits 99 Karten auf dem Tisch, müssen wir die neue Karte mit sehr vielen der bereits sortierten Karten vergleichen, um die richtige Position herauszufinden.

Hier liegt der große Unterschied: Der Aufwand zum Lösen eines Teilproblems ist nicht mehr unabhängig von der Problemgröße, wie das beim Tellerwaschen der Fall war! Je mehr Karten wir sortieren, desto länger dauert im Mittel das Sortieren einer Karte. Daher steigt der Aufwand mit jeder Karte, die zu unserer Sortierung hinzukommt, einerseits um den Faktor „die Anzahl der Karten ist höher, daher muss die Anzahl der Sortiervorgänge erhöht werden“, aber zusätzlich auch um den Faktor „jeder Sortiervorgang dauert im Mittel etwas länger“.

Eine der wesentlichen Aufgaben eines Informatikers ist, diese Erhöhung des Zeitaufwands zu erforschen und dieses Wissen zu gebrauchen, um die verwendeten Programme zu verbessern, so dass auch hohe Problemgrößen noch bewältigt werden können.

Sortieren auf Computerisch ...

Egal, welche verschiedenen Strategien Sie und Ihre Versuchspersonen bisher angewandt haben – diese sind sicherlich in der einen oder anderen Weise auch in der Informatik bekannt, es existieren also *Sortier-algorithmen* dafür.

Allerdings fällt es schwer, diese zu erkennen, wenn man jemandem beim Sortieren von Karten zuschaut. Ein Mensch hat einfach zu viele Freiheitsgrade, muss beim Agieren keine strikten Regeln einhalten. Hier hilft oft, einfach Computer zu spielen:

Überlegen Sie, welchen Beschränkungen ein Computer unterworfen ist, wenn er in seinem Speicher Zahlen sortieren soll. Wie könnte man diese Beschränkungen in eine Art „Spiel“ umsetzen, so dass man mit ähnlichen Voraussetzungen die Karten sortiert? Wenn Sie ungefähr wissen, wie der Speicher eines Computers funktioniert, versuchen Sie zunächst, die Frage selbst zu beantworten, ansonsten lesen Sie bitte einfach weiter.



Der größte Unterschied zwischen Mensch und Maschine besteht beim Sortieren in der Speicherung: Der menschliche Kartensortierer besitzt quasi überall „Speicher“ – auf der Hand, als Stapel auf dem Tisch, ausgelegt auf dem Tisch usw. Beim Computer ist der Speicher streng in Positionen organisiert und jede Position, genannt Speicherstelle oder Speicheradresse, kann genau einen Wert bzw. in unserem Spiel eine Karte beinhalten.

Das können wir in unserem Experiment allerdings simulieren. Basteln Sie sich einen eigenen Computerspeicher für den Kartentisch: Dazu schneiden Sie einfach die Vorlage aus dem Anhang aus und kleben sie so zusammen, dass sie einen langen Streifen bildet – genau wie ein Speicher mit eigener Adressierung. Für die hier beschriebenen Experimente sollten Sie mindestens 20 bis 30 Speicherstel-

len basteln. Wenn Sie wollen, können Sie aber den Bogen ohne Beschriftung mehrfach kopieren, um einen noch viel längeren Bereich zu erhalten! Ein Stück des Ergebnisses sehen Sie in Abbildung 2.2. Die grünen Buchstaben auf den ersten Speicherstellen sollen genauso wie die roten Zahlenbereiche erst einmal unbeachtet bleiben!

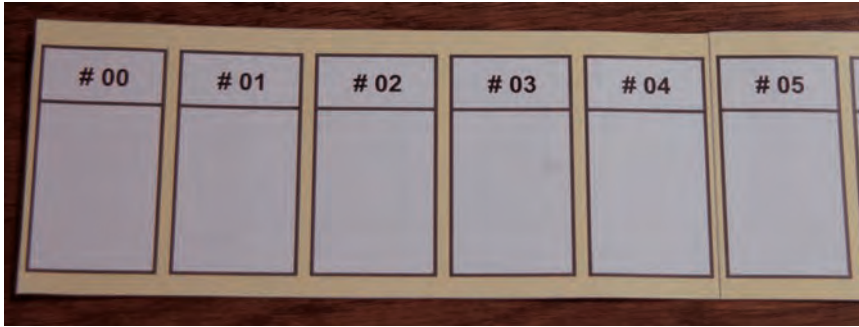


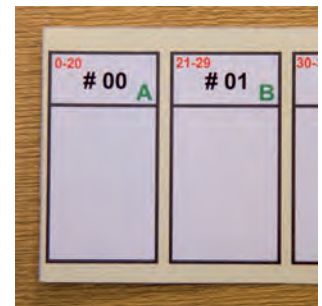
Abbildung 2.2 Der Papierspeicher

Bringen Sie nun ein paar Karten in den neuen Speicher. Für Sie gilt selbstverständlich die Spielregel: **Karten dürfen nur genau auf Speicherpositionen liegen, nicht dazwischen.** Legen Sie zwei beliebige Karten auf die Adressen #00 und #01.

Den Inhalt der Speicheradressen können Sie selbstverständlich auslesen – einfach anschauen, welche Karte darauf ist. Ein Problem gibt es allerdings beim Schreiben! Was passiert, wenn Sie einen Wert (= eine Karte) auf Position #01 schreiben? Bei unserem Papierspeicher passiert nicht viel: Eine Speicheradresse kann prinzipiell als Stapel beliebig viele Karten fassen. Nimmt man die oberen Karten wieder weg, kommen die unteren wieder zum Vorschein.

Ein Computer funktioniert anders. Jede Adresse kann genau einen Wert speichern. Gibt man einen neuen Wert hinein, überschreibt man damit den alten Inhalt. Wir müssen also noch als Spielregel festlegen, dass auf jeder Speicheradresse nur eine Karte liegen darf: **Sobald eine neue Karte auf einer Position abgelegt wird, kommt die eventuell bereits dort liegende Karte auf den Ablagestapel und ist aus dem Spiel.**

Probieren Sie das versuchsweise anhand eines sehr einfachen Experiments: Die Speicheradressen #00 und #01 sind momentan jeweils mit einer Karte belegt. Tauschen Sie den Inhalt dieser beiden Positionen aus und halten Sie sich dabei genau an die Spielregeln.



Speicheranfang



Haben Sie das Problem erkannt? Normalerweise würden wir für den Austausch einfach eine der Karten in die Hand nehmen, die andere verschieben und dann die Karte aus der Hand auf den verbliebenen Platz legen. Dadurch hätten wir aber die Hand als „Speicher“ genutzt – das ist gegen die Spielregel, nach der Karten nur auf Speicherpositionen des Papierspeichers liegen dürfen.

Nehmen wir dagegen eine der Karten und legen sie direkt auf die andere Position, wird diese überschrieben, die Karte mit dem anderen Wert kommt aus dem Spiel – sie ist verloren und der Austausch ist nicht mehr möglich.

Folgerung daraus ist, dass wir den zusätzlichen Speicherplatz („die Hand“) wirklich benötigen: Legen Sie die Karte von Speicherstelle #01 zunächst auf Position #02 ab. Danach können Sie die Karte von #00 auf #01 verschieben und jetzt erst die Karte von #02 auf #00 legen. Das entspricht unseren Spielregeln und genau so funktioniert die Sache auch im Computer:

Für viele Vorgänge wird ein Zwischenspeicher benötigt. Dafür besitzen die meisten Computer auch spezielle Vorkehrungen – Speicheradressen, die besonders schnell und komfortabel abgefragt werden können. Wie setzen wir das im Papierspeicher um? Wo haben wir im echten Leben einen komfortablen Zwischenspeicher für Karten? Wie wäre es mit folgender Erweiterung der Spielregel: **Sie dürfen nun maximal eine Karte in der Hand behalten, während Sie die anderen Karten im Speicher verschieben.**

Zeit für unsere nächsten Schritte als Computer-Simulator. Wählen Sie zufällig sechs der grün hinterlegten Spielkarten (mit einfachen Buchstaben) aus und legen Sie diese unsortiert auf die Adressen #00 bis #05. Abbildung 2.3 zeigt ein Beispiel.

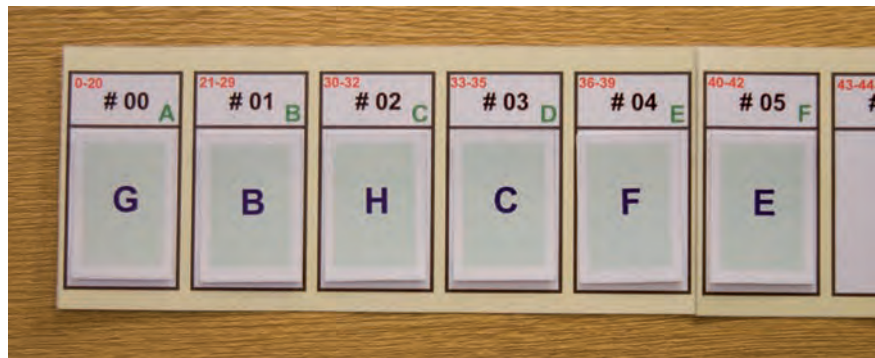


Abbildung 2.3 Unsortierte Karten im Speicher

Nun wollen wir die Folge alphabetisch sortieren. Fangen wir vorne an: „B“ muss ganz nach vorne. Nehmen Sie also „G“ in die Hand und schieben Sie „B“ auf die Position #00. Nun ist #01 leer, wir können das zweite Element der Sortierung, „C“ dorthin verschieben. Welches Element muss nun auf die freie Adresse #03? Das ist sicherlich „F“ – verschieben Sie es! Auf die nun freie Position kommt das in der Hand gespeicherte „G“. Abbildung 2.4 zeigt, wie die Folge jetzt sein sollte.

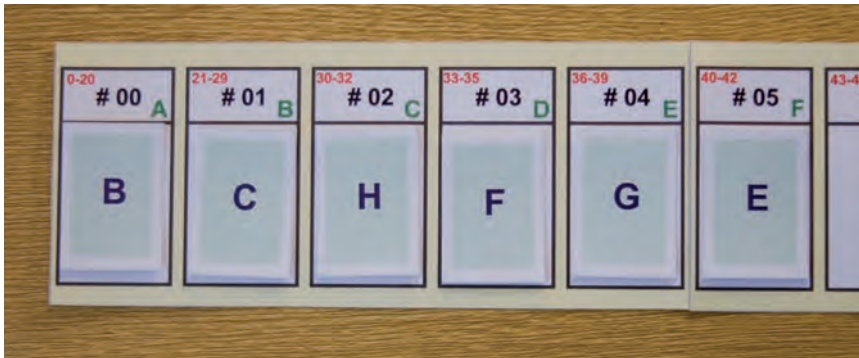


Abbildung 2.4 Nach den ersten Sortierschritten

Nun müssen nur noch „H“ und „E“ getauscht werden und die Folge ist perfekt sortiert, wie Abbildung 2.5 zeigt.

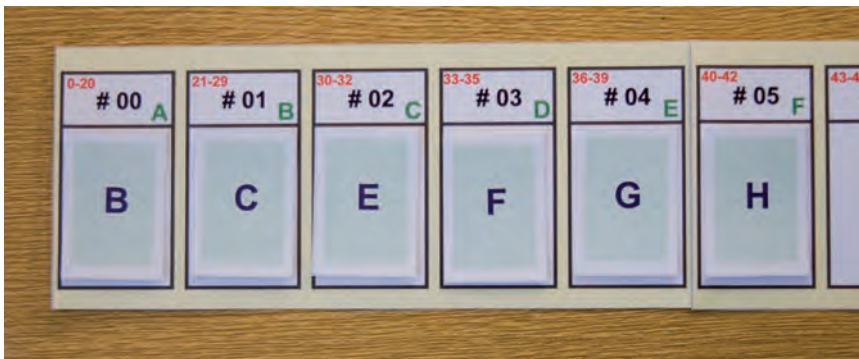


Abbildung 2.5 Die sortierte Folge

War das eine korrekte Vorgehensweise, wenn man einen Computer simulieren möchte? Betrachten Sie für die Antwort erst einmal nur den allerersten Schritt: „B“ muss ganz nach vorne. Nehmen Sie also „G“ in die Hand und schieben Sie „B“ auf die Position #00.



Zwei Erkenntnisse haben wir hier genutzt, die wir als Menschen sofort haben, die der Computer nicht ohne Weiteres besitzen kann:

1. „B“ ist im Speicher die Karte mit dem vordersten Buchstaben
2. Die Karte mit dem vordersten Buchstaben liegt auf Position #01

Woher wissen wir das? Weil wir natürlich alle Karten gleichzeitig sehen und bereits unser Unterbewusstsein spontan die Situation für uns analysiert.

Ein Computer kann zwar auf seinen gesamten Speicher zugreifen, aber im Normalfall immer nur gleichzeitig eine Adresse anschauen. Es ist für ihn daher im Normalfall ein aufwendiger Prozess, herauszufinden, welche der vorhandenen Karten denn nun ganz nach vorne muss.

Wir fügen das auch noch unseren Spielregeln hinzu. Schneiden Sie auch noch die Pfeile aus dem Anhang aus. **Wenn wir eine Speicherstelle auslesen wollen, muss immer ein Pfeil auf die entsprechende Adresse zeigen. Alle anderen Adressen gelten als unsichtbar.**

Für erste Experimente können Sie mit dem Pfeil „Speicherzeiger“ beginnen. Wir werden tatsächlich einige Pfeile gebrauchen, um den Speicher an mehreren, aber definierten Stellen sichtbar zu machen.

Ich verspreche es: Jetzt haben wir wirklich alle Regeln festgelegt, um Computer zu spielen und dürfen mit der Umsetzung des ersten Sortierverfahrens beginnen.

Alles, was Sie im folgenden Abschnitt erforschen, ist übrigens tatsächlich in heutigen Computersystemen wiederzufinden. Besonders in Bereichen, in denen an korrekten Programmen viel Geld hängt, folgt man gerne dem Informatik-Motto „never change a running system“, also „verändere nie ein System, wenn es (doch) läuft“.

Selection-Sort

Aufgabe ist, die Folge von sechs Buchstabenkarten im Speicher von Anfang bis zum Ende zu sortieren. Abbildung 2.6 zeigt die Aufstellung. Sie haben die Zeiger „Speicherzeiger“ sowie „Anfang“, „Ende“ und „Bestes“ zur Verfügung.

Wie bereits oben dargestellt, ist ein immer wiederkehrendes Prinzip der Informatik die Unterteilung eines großen Problems in kleinere, handhabbare Portionen. Eine Unterteilung des Sortierproblems könnte also lauten

Sortieren von n Karten:

- Sortiere die Karte mit dem kleinsten Buchstaben an den Anfang
- Sortiere die restlichen $n-1$ Karten

Diese Vorgehensweise nennt man auch „Sortieren durch Auswahl“ oder mit englischem Fachbegriff „Selection-Sort“, weil immer eine Karte ausgewählt und korrekt einsortiert wird.

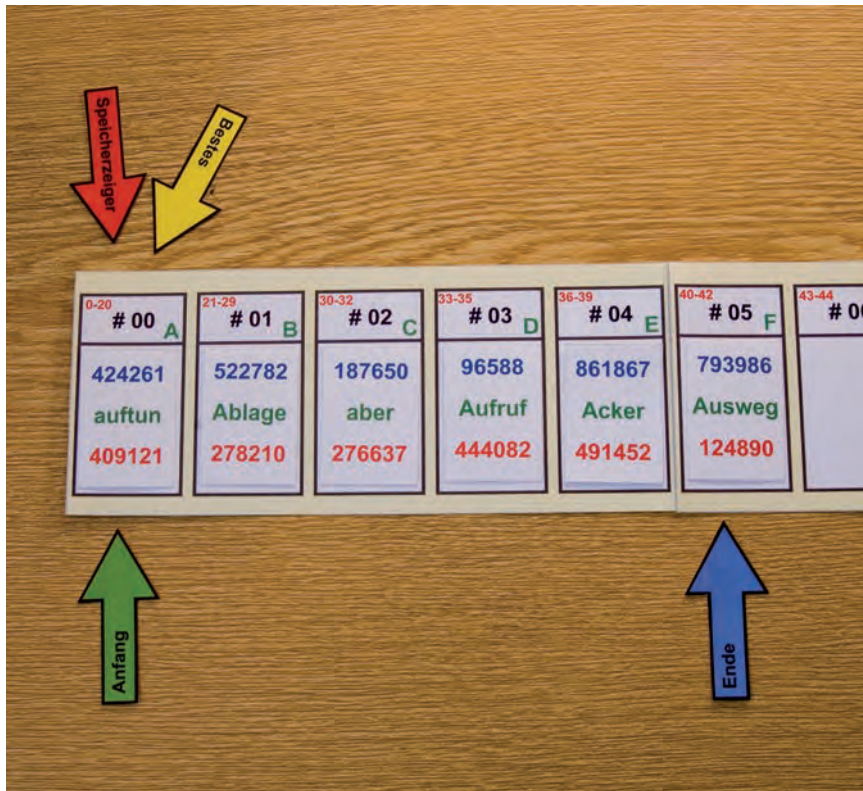


Abbildung 2.6 Sortierfolge mit Zeigern

Versuchen Sie einmal, diese Vorschrift zu konkretisieren. Spielen Sie mit den Materialien und formulieren Sie einen allgemeinen Algorithmus für Selection-Sort. Die Zeiger helfen Ihnen dabei, Positionen zu markieren. Sie dürfen sie im Algorithmus benutzen.

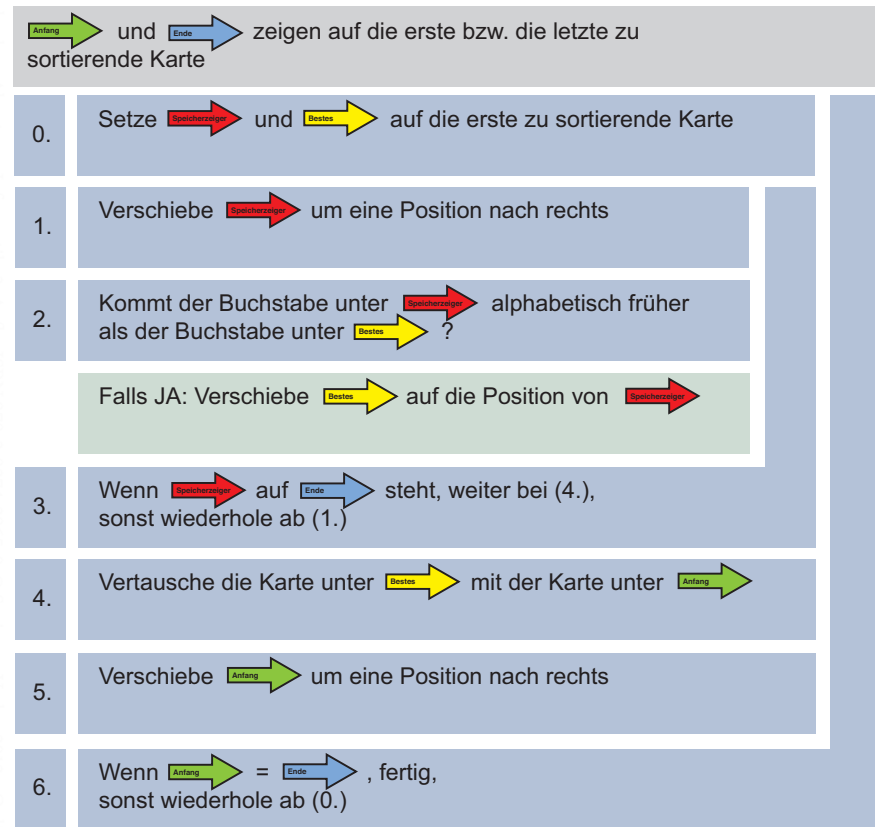
Sie können die aus dem ersten Kapitel bekannte Algorithmusschreibweise mit den Kästchen verwenden oder einfach die Vorschrift so aufschreiben, wie es für Sie selbst am besten nachvollziehbar ist.



Nassi-Shneiderman-Diagramm

Die hier verwendete Notation für Algorithmen ist eine vereinfachte Form der von Isaac Nassi und Ben Shneiderman entwickelten modularen Darstellungsart. Im deutschen Sprachraum sind sie auch unter dem Begriff „Struktogramm“ bekannt. Suchen Sie doch einfach mal in Wikipedia unter dem Begriff „Struktogramm“, wenn Sie mehr darüber erfahren möchten!

Ein Vorschlag ist folgende ausformulierte Vorschrift:



Warum funktioniert das Ganze?

In den Anweisungen (1.) bis (3.) wird der kleinste Buchstabe gesucht und gefunden. Dieser muss ganz an den Anfang und wird daher getauscht mit der Karte, die den Platz dort belegt (4.).

Nun weiß man, dass die Karte an der ersten Position korrekt liegt, dass diese also nie mehr verschoben werden muss: Sie gehört nicht mehr zum zu sortierenden Bereich. Daher rücken wir bei (5.) den Anfangszeiger um eine Position nach rechts und wiederholen das Sortieren so lange, bis die unsortierte Folge nur noch aus einer einzelnen Karte besteht. Eine Karte ist aber sowieso immer sortiert, daher sind wir fertig!

Ich will das Vorgehen hier gar nicht weiter erklären. „Begreifen“ Sie lieber den Algorithmus, indem Sie mehrere zufällige Folgen von Karten sortieren. Nehmen Sie hierfür auch gerne die Karten mit den echten Zahlen. Sie können sowohl die rote als auch die blaue Reihe nutzen – wenn Sie das abwechselnd tun, sparen Sie sich das Mischen zwischendurch.

Besser durch Blubberblasen?

Damit wir noch etwas zum Spielen haben, erkläre ich gleich ein weiteres, sehr bekanntes Sortierverfahren, das wegen seiner Einfachheit besonders in älteren Programmen recht häufig eingesetzt wird.

Die generelle Idee ist: Gehe die Reihe mit Karten von Anfang bis Ende durch. Vergleiche dabei immer zwei direkt nebeneinander liegende Karten. Sind diese richtig sortiert, mache gar nichts, ansonsten vertausche beide.

Können Sie wieder das grundlegende Prinzip der Informatik erkennen? Jede Vertauschung stellt etwas mehr Ordnung her, bis alle kleinen Fortschritte zusammen die gesamte Arbeit vollbracht haben.

Versuchen Sie das anhand einer zufälligen Reihe, wie in Abbildung 2.7.

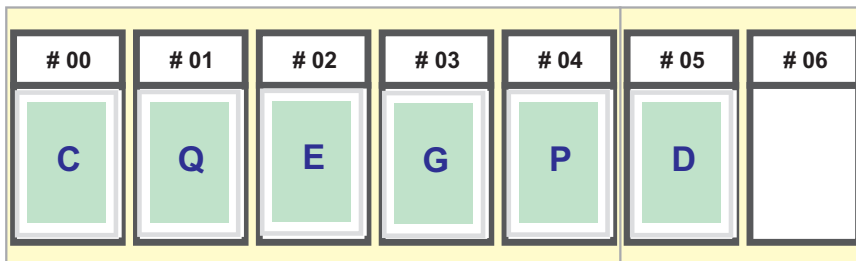


Abbildung 2.7 Unsortierte Folge

Nach dem Durchgehen der Reihe ist die Folge bereits etwas mehr sortiert (Abbildung 2.8).

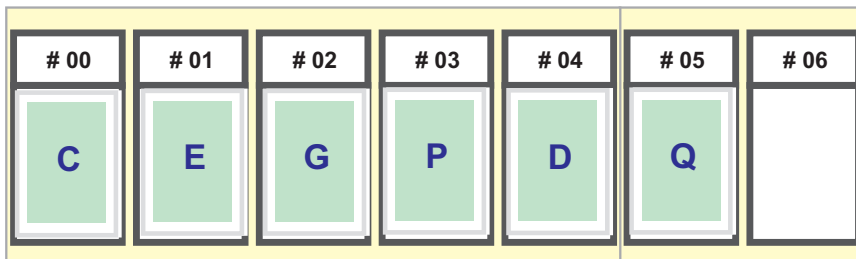


Abbildung 2.8 Folge nach einem Durchlauf

Überlegen Sie, was genau Sie über die Folge nach dem ersten Durchlauf sagen können und ob man diese Aussage so formulieren kann, dass sie für jede beliebige Kombination von Karten gilt, nachdem Sie den Algorithmus einmal durchgeführt haben.



Überlegungen dieser Art müssen Informatiker oft führen: das Schließen auf eine allgemeine Regel aufgrund eines Beispiels.

In diesem Fall sind nach dem ersten Durchlauf zwei Karten korrekt sortiert: C und Q. Ist das aber immer so? Nein! Nimmt man als Ausgangsfolge zum Beispiel P-E-Q-C-G-D, kommt nach dem ersten Durchlauf E-P-C-G-D-Q heraus wie in Abb. 2.9 dargestellt. C steht also nicht am Anfang, aber wiederum Q am Ende.

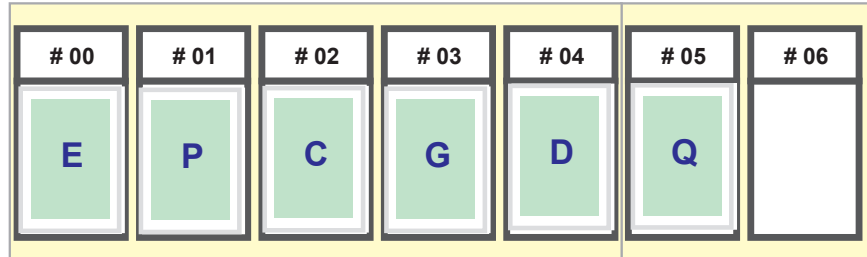


Abbildung 2.9 Ergebnis nach dem ersten Durchlauf mit anderer Ausgangsreihenfolge

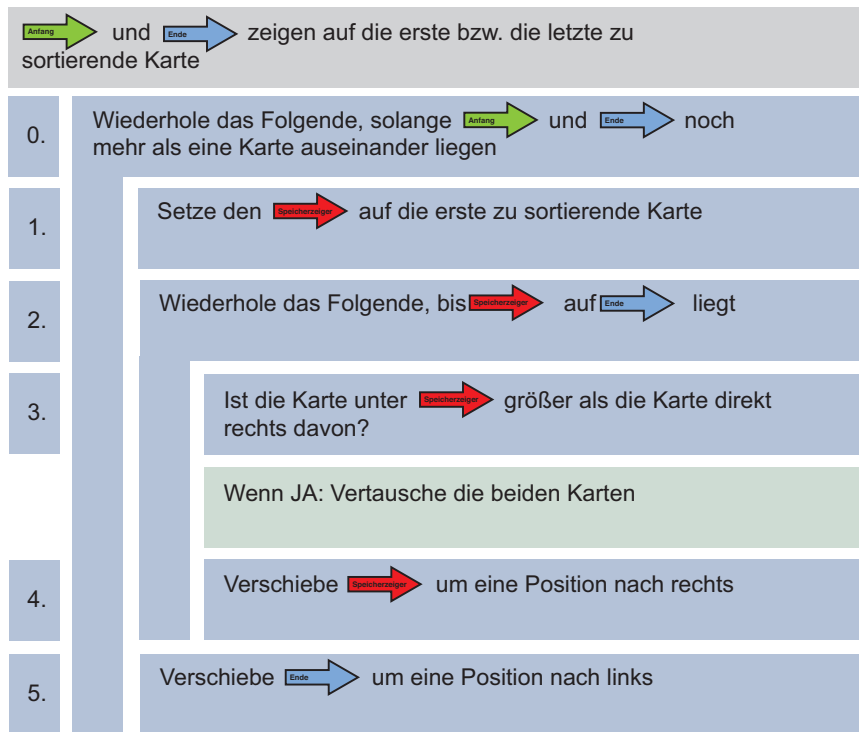
Wenn Sie darüber nachdenken, muss das auch so sein: Der Reihe nach wird durch das Vertauschen zweier Karten die größere der beiden immer nach rechts gebracht. Die größte Karte wechselt daher auf die Position ganz rechts, wo sie hingehört.

Auch dieses Sortierverfahren bringt also pro Durchlauf eine Karte an die richtige Stelle. Wenn wir es so oft durchführen, wie Karten vorhanden sind, müssen also danach alle Karten sortiert sein.

Versuchen Sie, nach diesem Prinzip einen Algorithmus zu formulieren, in ähnlicher Weise wie oben Selection-Sort. Sie können wieder die Zeiger Start, Ende und Speicherzeiger verwenden, um den Bereich zu markieren, der noch unsortiert ist.



Die Formulierung könnte sein



Hier wird eine Formulierung verwendet, wie sie in der Praxis tatsächlich häufig vorkommt: Wenn in einer Anweisung „Wiederhole Folgendes, ...“ vorkommt, bezieht sich „Folgendes“ auf den eingerückten Text darunter.



Der Algorithmus nennt sich übrigens „Bubble Sort“, weil man sich vorstellt, dass die Karten durch den immerwährenden Tausch an die korrekte Position kommen, wie Luftblasen im Wasser an die Oberfläche steigen.

Gut, besser, bester?

Das Finden verschiedener Algorithmen zur Lösung eines Problems ist für Informatiker sehr wichtig! Mindestens genauso wichtig ist jedoch, unter allen möglichen Algorithmen den besten zu finden.

Für einen guten Algorithmus kann es selbstverständlich sehr viele verschiedene Kriterien geben, zum Beispiel wie viel Programmieraufwand man benötigt oder wie elegant ein entsprechendes Computerprogramm aussieht. Für manche Anwendungen gilt sogar, dass ein unübersichtliches Programm sehr viel besser ist

Benchmark-Tests

Wir beschäftigen uns hier damit, die Laufzeit von Algorithmen durch theoretische Betrachtungen abzuschätzen. Ein anderer Ansatz ist, es einfach auszuprobieren. Das nennt man „Benchmarking“ (von engl. Benchmark = Maßstab). Hier werden die zu testenden Programme (oder auch ganze Systeme) mit Testdaten laufen gelassen. Der benötigte Zeitaufwand zum Lösen der Aufgaben wird gemessen und verglichen.

als ein übersichtliches, weil es von Konkurrenten schwerer abgeschrieben werden kann.

Normalerweise untersuchen wir jedoch immer die Zeit, die ein Algorithmus zur Lösung einer bestimmten Problemgröße benötigt – je schneller, desto besser.

Genau wie bei den Menschen gibt es selbstverständlich auch Computer, die mehr oder weniger zügig arbeiten und so kann das gleiche Verfahren auf unterschiedlichen Rechnern unterschiedlich viel Zeit in Anspruch nehmen. Daher müssen wir eine vergleichbare, nur auf den Algorithmus, nicht auf die Maschine zutreffende Basis schaffen:

Hierfür legen wir einfach fest, welche Schritte des Algorithmus im Computer Zeit kosten. Man spricht hier auch von *Operationen*. Diese werden dann bei der Durchführung einfach gezählt.

Da normalerweise Zugriffe auf den Speicher immer besonders zeitintensiv sind, können für unseren Papierspeicher folgende Schritte ausgewählt werden:

- Verschieben einer Karte auf eine neue Speicherposition
- Vergleich zweier Karten

Selbstverständlich gibt es noch viele weitere Operationen, wie zum Beispiel Verschieben eines Zeigers, Wenn-dann-Anweisung usw. Wie bereits oft in diesem Buch verwenden wir jedoch das Prinzip der Abstraktion, um uns das Leben so einfach wie möglich zu machen. In diesem Fall, indem wir nur die relevanten Operationen auswählen.

Elementaroperationen

Operationen bzw. Arbeitsschritte, die wir bei der Bestimmung der Laufzeit eines Algorithmus für wichtig bzw. zeitkritisch erachten. Welche Operationen hier ausgewählt werden, ist nicht global definiert, sondern ist vom jeweiligen Anwendungsfall abhängig. Oft handelt es sich um Speichervorgänge (Lesen und Schreiben im Speicher).

Preisfrage: Wie viele unserer Elementaroperationen benötigt das weiter oben beschriebene Verfahren zum Tausch zweier Karten im Papierspeicher?

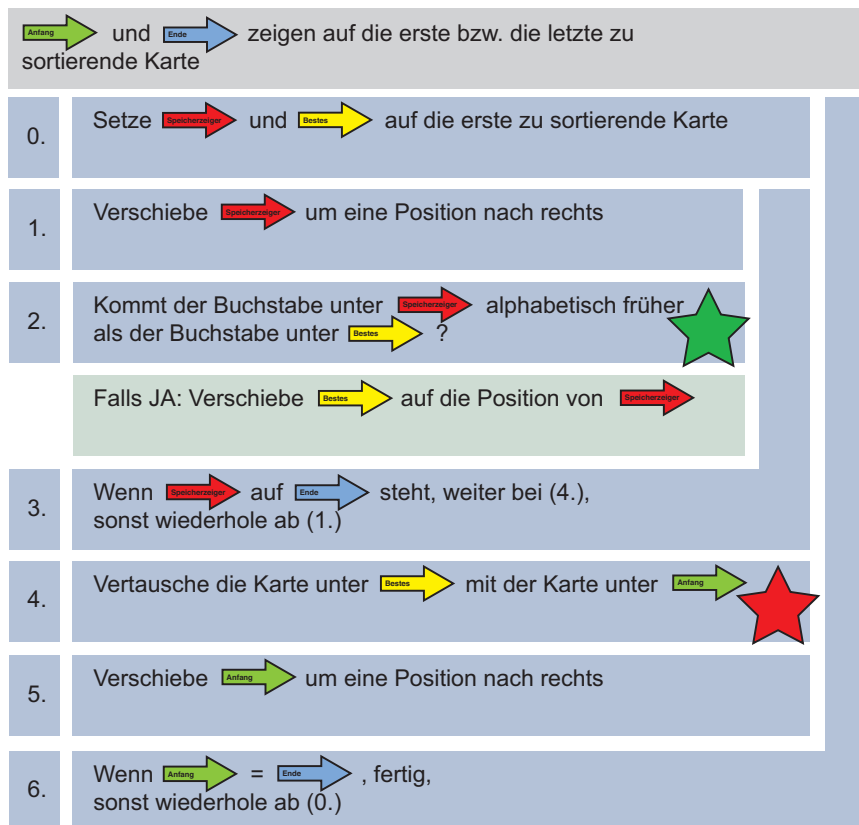


Um die Karten auf den Speicherpositionen A und B zu tauschen, sind *drei* Verschiebeoperationen nötig:

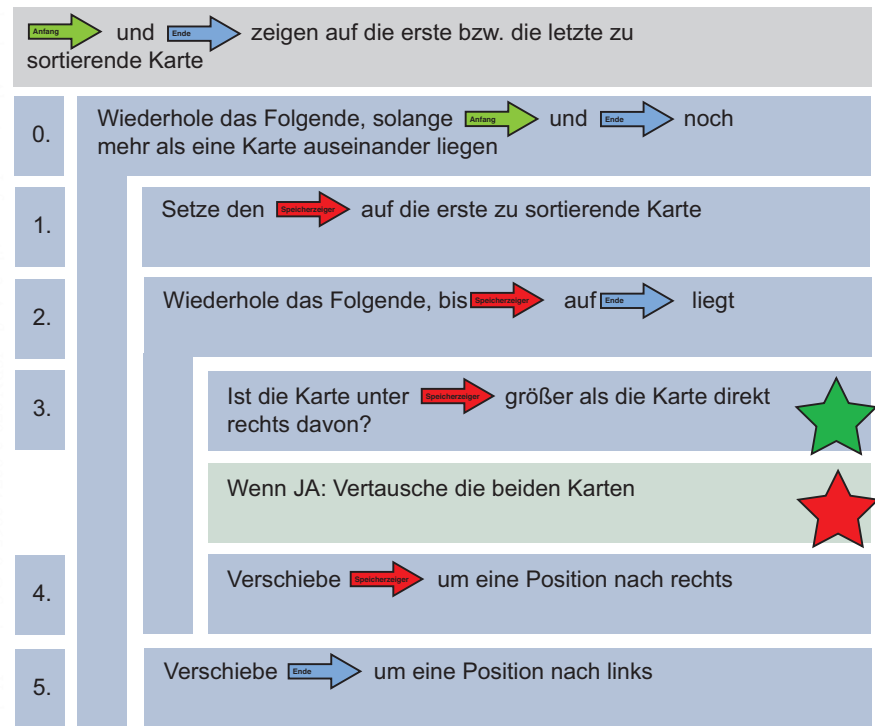
- Verschiebe Karte von A auf Zwischenspeicher
- Verschiebe Karte von B nach A
- Verschiebe Karte von Zwischenspeicher nach B

Wenn wir auf diese Weise die bisher aufgeschriebenen Algorithmen für Selection-Sort und Bubblesort analysieren, erhalten wir an den mit farbigen Sternen markierten Stellen Elementaroperationen. Grün steht für einen Vergleich (1 Elementaroperation), Rot für den Austausch zweier Speicherpositionen (3 Elementaroperationen).

Selection-Sort



Bubblesort



Jetzt sind Sie an der Reihe: Überprüfen Sie empirisch, welches Sortierv Verfahren das bessere ist, indem Sie verschiedene Kartenanzahlen mit beiden Verfahren sortieren. Notieren Sie sich dann (zum Beispiel mit einer Strichliste) jedes Mal den Vergleich zweier Karten und (dreifach) den Austausch zweier Karten. Wenn Sie die bunten Sortierkarten nutzen, können Sie abwechselnd die roten und blauen Zahlen nutzen, um dazwischen nicht immer wieder mischen zu müssen.



Selbstverständlich ist das Ergebnis davon abhängig, wie gut die Karten jeweils gemischt waren. Wenn man zufällig eine bereits sortierte Folge als Ausgangssituation hat, dann benötigt man zum Beispiel gar keine Vertauschungen. Bis auf solche Spezialfälle kann man an der entstehenden Tabelle jedoch recht gut ablesen, wie die Rechenzeit in Abhängigkeit der Problemgröße ansteigt.

Bei meinen Versuchen mit bis zu 35 Karten bin ich auf Ergebnisse nach Tabelle 2.1 gekommen. Dabei habe ich in gesonderten Spalten auch den „Aufwand pro Karte“ vermerkt, indem ich die Anzahl der Elementaroperationen nochmals durch die Anzahl der Karten geteilt habe.

Wie kann man diese interpretieren?

Genau wie beim komplett manuellen Sortieren, wie wir es ganz am Anfang durchgeführt hatten, steigt der Aufwand pro Karte mit der Anzahl der Karten. Man sagt, die Verfahren sind nicht linear, was soviel bedeutet, dass sich der Aufwand nicht direkt aus der einfachen Problemgröße ableiten lässt.

Weiterhin kann man ablesen, dass Bubblesort fast durchgehend schlechter ist als Selection-Sort: Der Aufwand pro Karte ist bei Bubblesort ungefähr doppelt so groß wie bei Selection-Sort.

Tabelle 2.1 Aufwand für das Sortieren verschiedener Kartenzahlen

Karten (= Problem- größe)	Selection-Sort	Bubblesort	Selection-Sort /Karte	Bubblesort /Karte
2	4	4	2,0	2,0
3	9	12	3,0	4,0
4	15	6	3,8	1,5
5	22	22	4,4	4,4
6	30	33	5,0	5,5
7	39	57	5,6	8,1
8	49	79	6,1	9,9
9	60	111	6,7	12,3
10	72	96	7,2	9,6
11	85	127	7,7	11,5
12	99	201	8,3	16,8
13	114	183	8,8	14,1
14	130	241	9,3	17,2
15	147	273	9,8	18,2
16	165	309	10,3	19,3
17	184	403	10,8	23,7
18	204	366	11,3	20,3
19	225	441	11,8	23,2
20	247	535	12,4	26,8
21	270	594	12,9	28,3
22	294	534	13,4	24,3
23	319	616	13,9	26,8
24	345	756	14,4	31,5
25	372	828	14,9	33,1
26	400	787	15,4	30,3
27	429	861	15,9	31,9
28	459	1014	16,4	36,2
29	490	1006	16,9	34,7
30	522	1074	17,4	35,8
31	552	1194	17,8	38,5
32	586	1251	18,3	39,1
33	620	1188	18,8	36,0
34	653	1335	19,2	39,3
35	690	1426	19,7	40,7

Der folgende Abschnitt (bis zur nächsten Überschrift) erfordert etwas mathematisches Vorwissen. Daher ist es nicht schlimm, wenn Sie ihn gegebenenfalls nicht komplett verstehen oder gleich ganz überspringen möchten.

Insgesamt ist der Aufwand pro Karte bei beiden Verfahren linear zur Problemgröße. Bei Selection-Sort gilt etwa:

$$\text{Aufwand pro Karte (Selection – Sort)} \approx 0,6 \cdot \text{Problemgröße}$$

Bei Bubblesort

$$\text{Aufwand pro Karte (Bubblesort)} \approx 1,3 \cdot \text{Problemgröße}$$

Da sich der Gesamtaufwand berechnen lässt durch

$$\text{Gesamtaufwand} = \text{Aufwand pro Karte} \cdot \text{Anzahl Karten},$$

lässt sich also leicht einsehen, dass der Gesamtaufwand beider Verfahren vom Quadrat der Problemgröße abhängt:

$$\text{Aufwand (Selection – Sort)} \approx$$

$$0,6 \cdot \text{Problemgröße} \cdot \text{Problemgröße} = 0,6 \cdot \text{Problemgröße}^2$$

$$\text{Aufwand (Bubblesort)} \approx 1,3 \cdot \text{Problemgröße} \cdot \text{Problemgröße} = 1,3 \cdot \text{Problemgröße}^2$$

Das ist eine ganz entscheidende Erkenntnis:

Beim Vergleich von Algorithmen lässt man normalerweise konstante Faktoren weg. Dadurch wären beide Sortierverfahren gleichwertig, nämlich quadratisch.

Bleibt nur noch die Frage, warum?

Nicht nur in der Informatik, auch in anderen Wissenschaften sucht man immer möglichst einfache Beschreibungsmöglichkeiten. Eine kurze, prägnante Formel ist immer vorzuziehen gegenüber einem komplizierten und langen Ausdruck. Wichtig ist, dass alle wesentlichen Aspekte berücksichtigt wurden.

Was ist nun am Aufwand eines Algorithmus wesentlich?

Konstante Faktoren können durch die Wahl eines schnelleren Computers ausgeglichen werden. Sind zum Beispiel 1.000.000 Karten zu sortieren, bräuchte Selection-Sort ungefähr 600 Milliarden Elementaroperationen, Bubblesort 1.300 Milliarden. Selbstverständlich würde man in der Praxis den schnelleren Algorithmus einsetzen. Andererseits kann das Manko jederzeit durch Einsatz eines schnelleren Rechners ausgeglichen werden. Für bestimmte Gegebenheiten mag sogar Bubblesort besser geeignet sein: Es werden immer nur direkt benachbarte Speicherstellen miteinander verglichen. Denkt man daran, dass die Daten auf einer Festplatte liegen, geht also der Vergleich ohne große Bewegung des Schreib-/Lesekopfes vonstatten. Bei Selection-Sort hat man diesen Vorteil nicht.

Wesentlich ist allerdings, wie stark der Zeitaufwand anwächst, wenn die Problemgröße anwächst! Dieses Verhältnis lässt sich nicht durch Wahl eines stärkeren Rechners verändern. Genauso wenig wie durch Feinjustage des Algorithmus!

Wenn wir einen besseren Algorithmus finden möchten, ist es wesentlich, dieses Verhältnis zwischen Problemgröße und Aufwand zu verbessern. Konstante Faktoren spielen eine untergeordnete Rolle!

Wie unterscheidet sich denn nun ein Sortierverfahren, das zu einer anderen Kategorie gehört? Nehmen wir an, es gäbe eines, das kubisch, also proportional zu *Problemgröße*³ funktioniert. Das würde für die 1.000.000 Karten bereits etwa 1 Trillion Elementaroperationen benötigen, wäre also nicht nur etwas schlechter, sondern würde gleich um Größenordnungen länger benötigen, die Aufgabe zu lösen. Glücklicherweise haben wir ja bereits bessere Verfahren kennen gelernt.

Schön wäre allerdings, wenn wir ein Verfahren hätten, das besser ist. Hier kann ich vorwegnehmen: Die Elite der „normalen“ Sortierverfahren läuft proportional zu *Problemgröße* · $\log_2$ *Problemgröße*. Damit lassen sich 1.000.000 Karten ungefähr mit einem Vielfachen von 10.000 Elementaroperationen sortieren, was wiederum um Größenordnungen besser ist als Bubble- oder Selection-Sort.

Die Aufwandsabschätzung

Mit der Aufwandsabschätzung wird in der Informatik oft die Qualität von Algorithmen bestimmt. Diese Abschätzung gibt an, wie stark die notwendige Rechenzeit in Bezug zur Problemgröße anwächst. Konstante Faktoren werden hier nicht berücksichtigt. Auf diese Weise kann für kleine Problemgrößen die tatsächliche Rechenzeit eines „schlechteren“ Algorithmus unter der des besseren liegen.

Mit der hohen Kunst der Aufwandsabschätzung wollen wir nun einen noch besseren Sortieralgorithmus finden.

Nur einige Sortieralgorithmen:

Bogosort
 Bubblesort
 Cocktailsort
 Combsort
 Gnomesort
 Heapsort
 Insertionsort
 Introsort
 Mergesort
 Quicksort
 Selectionsort
 Shellsort
 Smoothsort
 Stoogesort
 Tournamentsort

 $n \log n$ oder die Kür des Sortierens

Bevor ich weiter auf die hohe Sortierkunst eingehe, möchte ich das „ $n \log n$ “ aus der Überschrift erklären. Informatiker sagen zum oben viel verwendeten Begriff „Problemgröße“ häufig einfach „ n “. Das Sortieren von 1.000 Karten wird einfach als „ $n = 1000$ “ ausgedrückt.

$n \log n$ ist daher ein Ausdruck für die bereits oben erwähnte Formel $\text{Problemgröße} \cdot \log_2 \text{Problemgröße}$.

Im letzten Abschnitt haben wir experimentell nachvollzogen, warum die bisher durchgesprochenen Sortiervverfahren eine quadratische Aufwandsabschätzung (also n^2) besitzen. Man kann das jedoch auch anders begründen. Drücken wir der Einfachheit halber im Folgenden die Problemgröße bzw. die Anzahl der Karten immer mit n aus.

Ein Sortierschritt der Verfahren sorgt jeweils dafür, dass ein einzelnes Element korrekt positioniert wird (am Ende der Folge). Wir benötigen also n Sortierschritte, bis alle Karten korrekt liegen.

Betrachten wir einen einzelnen Sortierschritt: Hier wird die gesamte Liste durchgegangen – bei Selection-Sort, um das größte Element zu finden, bei Bubblesort, um jeweils Paare in die korrekte Reihenfolge zu bringen. Im ersten Schritt müssen wir also n Karten anfassen, im zweiten Schritt eine weniger usw. bis im letzten Schritt nur eine Karte betrachtet wird. Im Durchschnitt sind also pro Schritt etwa $\frac{n}{2}$ Karten anzufassen (was einem Vielfachen von n Operationen entspricht).

Insgesamt gibt es also

- n Durchläufe
- mit je $\frac{n}{2}$ Operationen

n Sortierschritte mal $\frac{n}{2}$ Karten ergibt einen Aufwand von $\frac{n^2}{2}$ Operationen. Da wir konstante Faktoren nicht betrachten wollten, fassen wir das zu einer Aufwandsabschätzung von n^2 zusammen.

Widmen wir uns der nächsten großen Aufgabe: Ein besseres Sortiervverfahren soll gefunden werden!

Das können wir offenbar erreichen, indem wir

- entweder die Anzahl der Sortierschritte reduzieren oder
- den Aufwand pro Sortierschritt reduzieren

Wie schon im letzten Kapitel können wir hier auch wieder alltägliche Situationen als Vorbild nutzen. In diesem Fall ein im Sport typischer Wettkampf im K.-o.-Verfahren. Allerdings lassen wir in unserem Fall nicht Sportler, sondern Karten gegeneinander antreten. Verwenden Sie den Spielplan aus dem Anhang für den Wettkampf. Legen Sie die unsortierte Folge auf die schwarzen Felder wie in Abbildung 2.10.

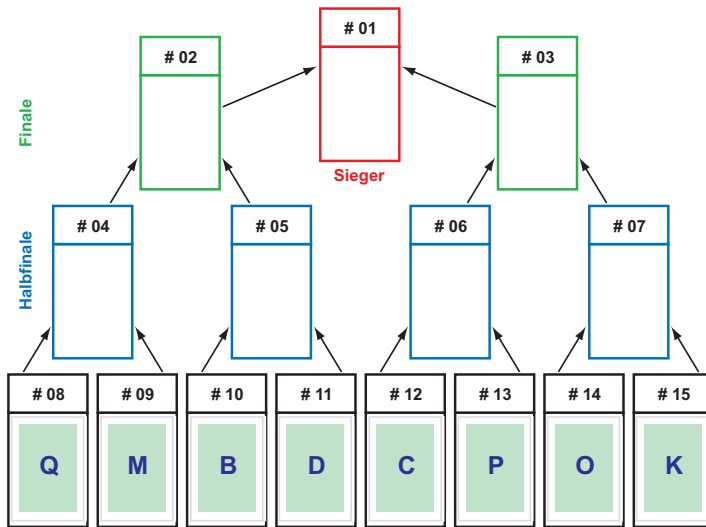


Abbildung 2.10 Die Karten treten gegeneinander an

Daraufhin spielen wir das erste K.-o.-System aus. Die jeweils größere Karte eines Paares „gewinnt“ und rückt ins Halbfinale. Abbildung 2.11 zeigt ein Beispiel.

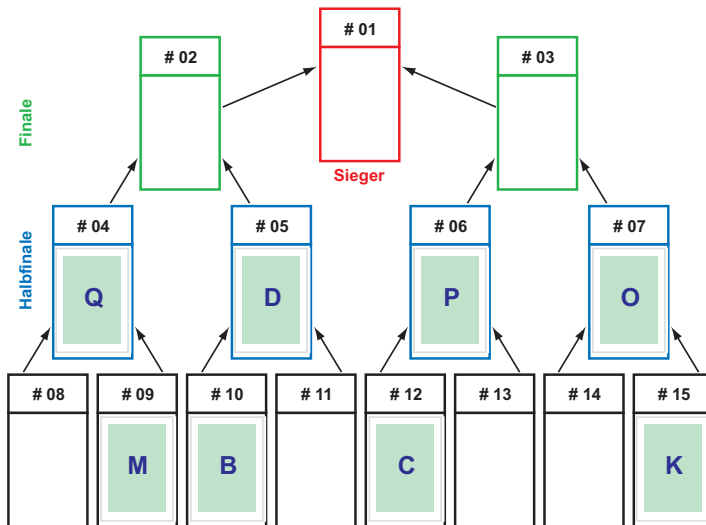


Abbildung 2.11 Halbfinale

Als Nächstes rücken die jeweils größeren Karten des Halbfinals auf die grünen Felder vor. Abbildung 2.12 zeigt das.

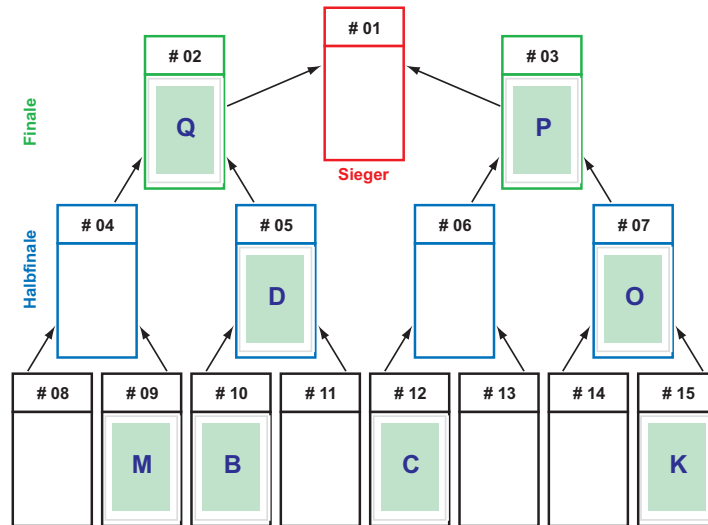


Abbildung 2.12 Halbfinale

Selbstverständlich wäre es schade, wenn jetzt sozusagen die Plätze auf dem Siebertreppchen verwaizen würden, wie in der Abbildung die Felder #04 und #06. Hier können die jeweils nächstplatzierten Karten von unten nachrücken. Das ist für den Algorithmus sehr wichtig. Sie sehen das Ergebnis in Abbildung 2.13.

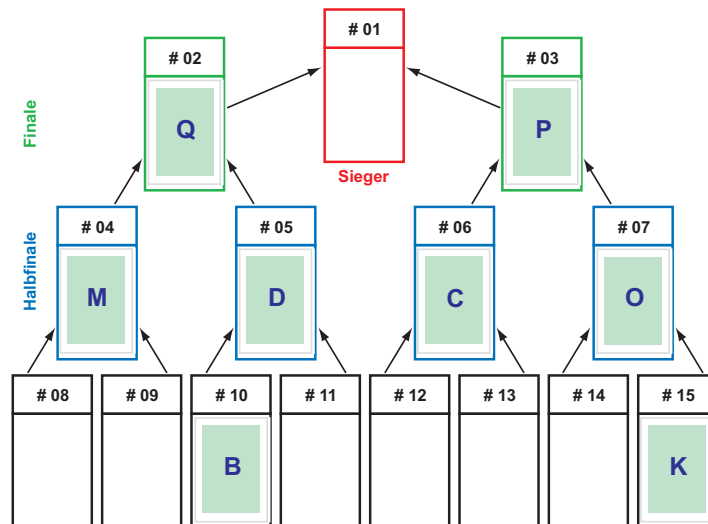


Abbildung 2.13 Halbfinale

Nun wird noch das Finale ausgespielt – die Karte Q gewinnt und kommt auf das rote Siegerfeld. Die entstehende Lücke wird selbstverständlich sofort von den je-

weils nächstplatzierten Karten der darunterliegenden Ebenen ausgefüllt. In diesem Fall rückt die Karte M nach, darunter gibt es keine Nachfolger. Abbildung 2.14 zeigt das fertige K.-o.-System.

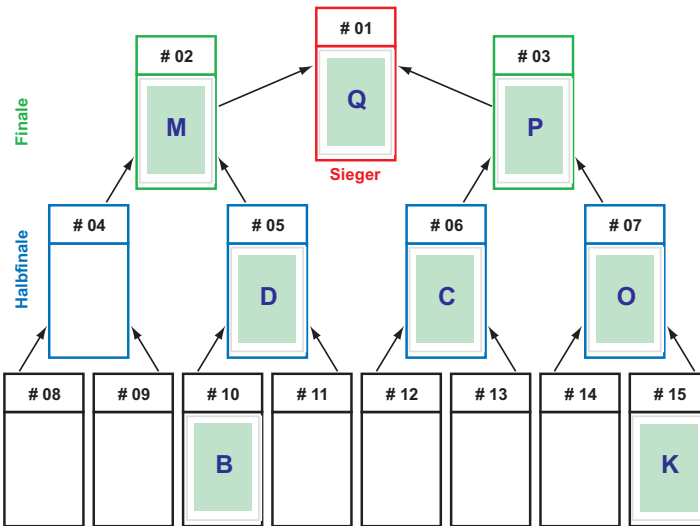


Abbildung 2.14 Das K.-o.-System ist fertig ausgespielt

Was bringt uns das Ganze jetzt?

Durch das K.-o.-System haben wir garantiert die größte Karte gefunden, denn die liegt auf dem Siegerfeld. Sie gehört an das Ende der sortierten Liste. Dort schieben wir sie daher auch hin (Abbildung 2.15).

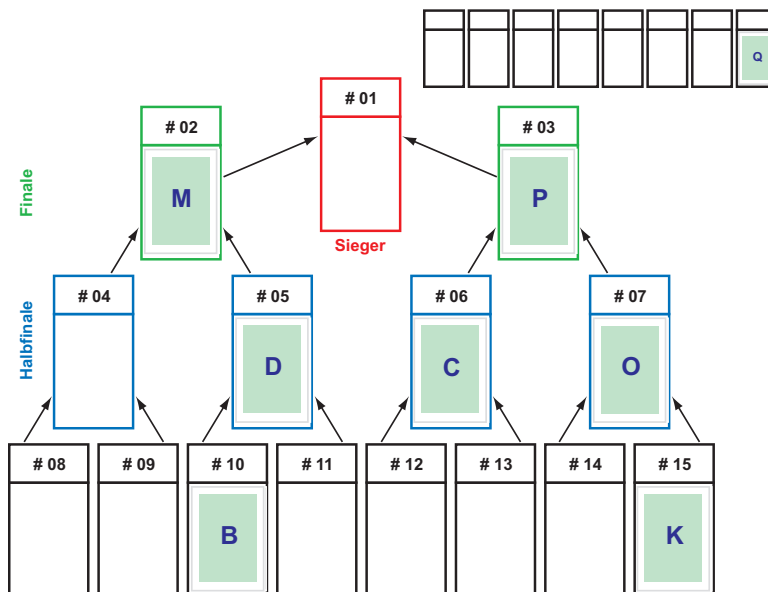


Abbildung 2.15 Die höchste Karte ist Q

Der Siegerplatz ist nun vakant und selbstverständlich rücken die nächsten Karten nach: Die höchste Karte auf Finalposition ist P. Auf deren Platz rückt dann O und hierauf K.

Wir wissen nun, dass P die höchste Karte ist und legen sie auf den ersten freien Platz am Ende der sortierten Liste, wie in Abbildung 2.16.

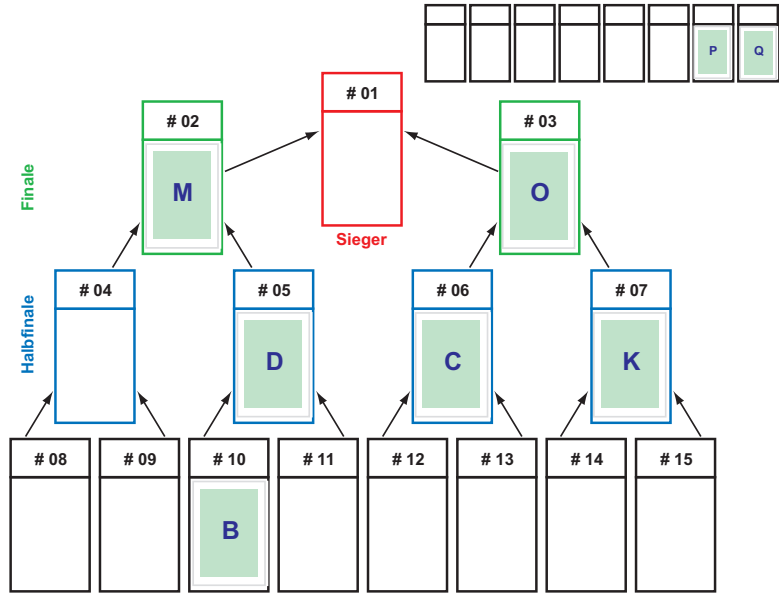


Abbildung 2.16 Nun sind P und Q richtig sortiert

Genau das gleiche Prinzip verwenden Sie nun, um immer wieder das Siegerfeld zu füllen und die Karten darunter nachrücken zu lassen, bis die sortierte Liste vollständig ist. Probieren Sie es aus!

Sie werden jetzt vielleicht überlegen, wozu der ganze Aufwand eigentlich gut ist, denn wir haben ja bereits zwei gut funktionierende Sortierverfahren kennen gelernt. Was ist anders am K.-o.-Verfahren, das übrigens nach dem englischen Wort für Wettbewerb „Tournament-Sort“ genannt wird? Für diese Analyse benötigen wir erst einmal die genaue formale Fassung des Algorithmus.

Im Aufstellen formaler Algorithmen sind Sie ja inzwischen bereits Meister. Wollen Sie das ein weiteres Mal unter Beweis stellen, indem Sie für Tournament-Sort eine Ablaufvorschrift erstellen?



Mein Vorschlag ist folgender:

0. Gehe die leeren Ebenen von unten nach oben durch und fülle jedes Feld mit der größeren Karte der beiden zugehörigen Wettbewerber.
1. Fülle dabei weiter unten entstehende Lücken sofort wieder mit einer der Karten darunter auf:
2.

bei zwei Wettbewerbern mit der größeren Karte	bei nur einem Wettbewerber mit diesem	bei keinem Wettbewerber bleibt das Feld leer
---	---------------------------------------	--
3. Mache Folgendes so lange, bis keine Karte mehr im Siegerfeld liegt
4. verschiebe die Siegerkarte auf das erste freie Feld am Ende der sortierten Liste
5. Fülle dabei weiter unten entstehende Lücken sofort wieder auf wie oben beschrieben

Versuchen Sie nun, diesen Algorithmus genau so zu analysieren, wie wir dies bereits oben getan haben. Wie viele Durchläufe benötigt er für die Sortierung? Wie hoch ist der Aufwand pro Durchlauf?

Kleiner Hinweis: Im Prinzip besteht der Algorithmus aus einer Art Vorbereitungsschritt und der eigentlichen Sortierung. Diese können wir auch getrennt voneinander betrachten.



Im ersten Teil wird die K.-o.-Tabelle vorbereitet, die erste Karte auf dem Siegerfeld ermittelt. Hierfür muss jedes (anfangs leere) Feld der K.-o.-Tabelle mit einer Karte belegt werden. Wie viele leere Felder gibt es aber für eine K.-o.-Tabelle mit n Karten?

Auf der untersten Ebene sind n Felder. Auf der untersten Siegerebene sind es halb so viele (je eine von zwei Karten gewinnt). Es folgen wiederum halb so viele, also ein Viertel usw., bis nur noch ein Feld, das Siegerfeld, in der obersten Ebene steht.

Insgesamt gibt es also $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \dots + 1$ Felder auf den Siegerebenen.

Einer mathematischen Rechenregel nach ergibt diese so genannte geometrische Reihe ungefähr wieder n , also $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \dots + 1 \approx n$.

Sie können sich das einfach auch durch Ausprobieren klar machen:

Legen Sie eine Anzahl Karten in einer Reihe aus und dann für jedes Feld der K.-o.-Tabelle eine Münze darüber – also in der Reihe oberhalb eine Münze für je zwei Karten, in der Reihe darüber wieder eine Münze für je zwei Münzen darunter usw.

Zählen Sie die Anzahl der Münzen. Je nachdem, ob die Anzahl der Karten gerade ist bzw. wie oft sie durch zwei teilbar ist, ist sie etwas kleiner oder etwas größer als die Zahl der Münzen, aber immer ungefähr gleich.

Die gesamte Anzahl der Siegerplätze eines K.-o.-Systems ist immer etwa gleich groß wie die Anzahl der Wettbewerber.

Wir haben nun also herausgefunden, dass jedes Feld der leeren K.-o.-Tabelle gefüllt werden muss und dass es insgesamt ungefähr n Felder gibt! Nennen wir das einmal n Durchläufe.

Beim Füllen eines Feldes müssen gegebenenfalls weiter unten frei werdende Felder mit Nachrückern bestückt werden. Maximal passiert das so oft, wie es Ebenen in der Tabelle gibt. Man nennt die Anzahl der Ebenen auch die *Höhe* der Tabelle oder einfach „ h “.

Der Aufwand unseres Tournament-Sorts wird also bestimmt von

- n Durchläufen mit jeweils
- h Verschiebeoperationen

Oder *Aufwand* = $n \cdot h$.

Es wäre nun schön, wenn wir die Höhe irgendwie ausrechnen könnten. Ermitteln Sie zunächst durch Probieren, welche Höhe eine Tabelle mit 8, 9, 10, 15, 16, 32 Karten besitzt.



8 Karten ergeben 4 Ebenen. Bei 9, 10, 15 oder 16 Karten sind es immer 5 Ebenen. Die K.-o.-Tabelle mit 32 Karten hat 6 Ebenen.

Ebene	Anzahl Karten
1	1
2	2
3	4
4	8
5	16
6	32
7	64
...	...
h	2^{h-1}
$h+1$	2^h

Klar wird dies, wenn man bedenkt, dass jede Ebene immer halb so viele Karten enthält (aufgerundet) wie die darunterliegende Ebene. Die oberste Ebene enthält eine Karte. Umgekehrt ausgedrückt kann also jede untere Ebene (maximal) doppelt so viele Karten enthalten wie die darüber liegende Ebene.

Wenn wir die oberste Ebene mit der Nummer 1 versehen, ergibt sich für das Fassungsvermögen also die Tabelle links.

Eine Tabelle mit 2^h Karten besitzt nach dieser Tabelle also $h+1$ Ebenen!

Da wir die Anzahl der Karten mit n bezeichnet hatten, gilt also nun der Zusammenhang

$$n \approx 2^h$$

Das Ungefähr-Zeichen verwende ich hier, weil Gleichheit nur gelten würde, wenn die Anzahl der Karten einer „glatten“ Zweierpotenz, also 2, 4, 8, 16, ... entspräche.

Mathematiker sehen nun sofort, dass man die Formel mit dem Logarithmus auflösen kann:

$$\log_2 n \approx h$$

Für alle anderen: Sie konnten sich experimentell eine Vorstellung machen, wie die Anzahl der Ebenen mit der Anzahl der Karten wächst: Je mehr Ebenen bereits vorhanden sind, desto mehr Karten kann man hinzufügen, ohne die Anzahl der Ebenen zu erhöhen. Mit anderen Worten: Je mehr Karten vorhanden sind, desto langsamer wächst die Anzahl der Ebenen. Dieses Verhältnis wird mathematisch vom Logarithmus ausgedrückt!

Kommen wir jetzt zu unserer Betrachtung zurück, wie viel Rechenzeit der Algorithmus in Bezug zur Anzahl der Karten benötigt.

Oben hatten wir bereits zum Aufbau der K.-o.-Tabelle einen Aufwand von $n \cdot h$ bestimmt. Die Höhe h der Tabelle können wir nun mit $\log_2 n$ angeben. Daher haben wir für diesen ersten Teil des Algorithmus einen Aufwand von $n \cdot \log_2 n$ oder einfach $n \log n$. Im zweiten Teil des Algorithmus wird jeweils die Siegerkarte in die sortierte Liste verschoben und dann das leere Feld wieder gefüllt. Das passiert so oft, wie Karten vorhanden sind, also n -mal. Jedes Mal muss das leere Feld gefüllt werden und die Anzahl der Verschiebungen entspricht wiederum maximal der Höhe, also $\log_2 n$. Wieder ergibt sich daher insgesamt ein Aufwand von $n \cdot \log_2 n$. Man könnte jetzt sagen, der Aufwand beider Teile zusammen beträgt $2 \cdot n \cdot \log_2 n$, aber konstante Faktoren lassen wir ja einfach weg, daher kommen wir auf den gewünschten Algorithmus mit $n \log n$ Aufwand. Sie können diese viel bessere Laufzeit auch experimentell nachvollziehen:

Sortieren Sie unterschiedliche Kartenzahlen mit Tournament-Sort. Hierfür nehmen Sie am besten den bereits gebastelten Papierspeicher und legen ihn an die untere Kante eines Tisches. Die Tabellenpositionen können nun darüber sehr schön mit gelben Haftnotizen oder auch mit Münzen markiert werden.

Zählen Sie anhand einer Strichliste jeweils, wie viele Verschiebungen und wie viele Vergleiche Sie durchführen mussten.



Tabelle 2.2 Aufwandsvergleich mit Tournament-Sort

<i>n</i>	Selection	Tournament	Select/Karte	Tournament/Karte
2	4	5	2,0	2,5
3	9	12	3,0	4,0
4	15	16	3,8	4,0
5	22	28	4,4	5,6
6	30	33	5,0	5,5
7	39	41	5,6	5,9
8	49	48	6,1	6,0
9	60	68	6,7	7,6
10	72	73	7,2	7,3
11	85	80	7,7	7,3
12	99	90	8,3	7,5
13	114	97	8,8	7,5
14	130	110	9,3	7,9
15	147	115	9,8	7,7
16	165	126	10,3	7,9
17	184	161	10,8	9,5
18	204	168	11,3	9,3
19	225	179	11,8	9,4
20	247	190	12,4	9,5
25	372	244	14,9	9,8
30	522	289	17,4	9,6
50	1372	577	27,4	11,5
100	5247	1368	52,5	13,7
150	11622	2300	77,5	15,3
200	20497	3096	102,5	15,5
250	31872	3931	127,5	15,7
300	45747	5197	152,5	17,3
350	62122	6091	177,5	17,4
400	80997	7073	202,5	17,7
450	102372	7943	227,5	17,7
500	126247	8869	252,5	17,7

In Tabelle 2.2 sind zum Vergleich nochmals die Werte von Selection-Sort aufgeführt, also des besseren der beiden zuerst betrachteten Sortieralgorithmen. Außerdem sind ein paar Werte für recht hohe Kartenanzahlen eingefüllt, die ich zugegebenerweise nicht per Hand, sondern per Computer ermittelt habe. Diese zeigen jedoch, dass sich ein Verfahren mit einem geringeren Aufwand besonders bei hohen Problemgrößen auszahlt – bei sehr kleinen Kartenzahlen ist Tournament-Sort sogar schlechter!

Bei Selection-Sort steigt der Aufwand pro Karte linear mit der Gesamtanzahl Karten, bei Tournament-Sort lediglich logarithmisch. Sie sehen, dass daher auch große Mengen mit Tournament-Sort noch verarbeitet werden können. Bei bereits 500 Karten benötigt dieser Algorithmus nur noch etwa 7 % der Elementaroperationen von Selection-Sort für die gleiche Aufgabe.

Was steckt dahinter

Dieses Kapitel stand voll im Zeichen dessen, was in der Informatik *Aufwandsabschätzung* oder auch *Komplexität* genannt wird. Bereits im letzten Kapitel war notwendig, die Qualität zweier Algorithmen gegeneinander abzuwiegen. Dort ging es jedoch hauptsächlich darum, die gegebene Aufgabe des Routenplaners überhaupt zu bewältigen. Einer der Algorithmen – Brute-Force – versagte dabei bereits für kleine Beispiele.

In diesem Kapitel haben Sie nun mehrere Algorithmen kennen gelernt, die auf den ersten Blick gleichwertig sind, da sie für überschaubare Beispiele alle recht gut funktionieren.

Computer werden jedoch oft genau in Bereichen eingesetzt, die für uns Menschen nicht mehr überschaubar sind. Der Zentralrechner einer Bank muss nicht 30, 500 oder 10.000 Datensätze sortieren, sondern teilweise Milliarden von Transaktionen in die korrekte Reihenfolge bringen. Hier ist zwingend erforderlich, die dafür verwendeten Algorithmen nicht intuitiv zu beurteilen, sondern anhand genauerer Maßstäbe. Im Allgemeinen wird die Abschätzung der Komplexität als Maßstab verwendet!

Wie bereits oben dargestellt, ist dabei oft gar nicht so wichtig, ob ein Algorithmus doppelt so lange benötigt wie ein anderer. Wesentlich ist, wie viel mehr Rechenzeit ein Algorithmus benötigt, wenn die Problemgröße anwächst. Genau dieses Verhalten macht sich nämlich besonders stark bemerkbar, wenn wir mit sehr großen Datenmengen arbeiten.

Wenn wir die Verfahren auf diese Weise analysieren, bekommen wir heraus, welches für unsere Problemstellung am besten geeignet ist. Die Sache hat aber auch noch einen weiteren Vorteil: Wenn wir die Analyse durchführen, machen wir uns besonders deutlich, an welchen Stellen des jeweiligen Verfahrens Rechenzeit verbraucht wird. Das gibt gute Hinweise darauf, wo wir noch etwas verbessern können, um zu einem günstigeren Ergebnis zu gelangen.

Im Beispiel oben hatten wir festgestellt, dass sowohl Selection-Sort als auch Bubblesort grob gesagt n Durchläufe mit jeweils n Operationen benötigen, also insgesamt eine Komplexität von n^2 besitzen. Die Folge war, dass wir auf ein Verfahren hinarbeiten konnten, in dem die Anzahl der Durchläufe gleich blieb, aber die Anzahl der Operationen pro Durchlauf deutlich optimiert wurde: Tournament-Sort.

Einen anderen Ansatz zur Optimierung verfolgt übrigens der von Tony Hoare 1960 erfundene Quicksort-Algorithmus, der auch heute noch in vielen Formen eingesetzt wird: Hier benötigt jeder Durchlauf prinzipiell noch n Operationen, aber die Anzahl der Durchläufe wird reduziert. Auch Quicksort hat daher im Durchschnitt die Komplexität von $n \log n$.

Großes Oh

Im bisherigen Verlauf des Kapitels habe ich die Freiheit der populärwissenschaftlichen Darstellung genutzt und den Algorithmen einfach einem bestimmten Aufwand zugeordnet, also etwa n^2 oder $n \log n$.

Der wirkliche Aufwand eines Algorithmus hängt aber von vielen Faktoren ab: wie die Daten am Anfang sortiert sind, wie das Verfahren genau auf einem Computer umgesetzt wurde usw. Da dieser tatsächliche Aufwand daher gar nicht oder nur sehr schwer zu bestimmen ist, spricht man gewöhnlich von der Ordnung eines Aufwands, wenn man ihn so abschätzt, wie wir das bisher getan haben.

Bubblesort und Selection-Sort sind *von der Ordnung* n^2 , während Tournament-Sort und Quicksort *von der Ordnung* $n \log n$ sind.

Die Informatiker bedienen sich hierfür des Buchstabens O, abgeleitet vom griechischen Omikron, das der Mathematiker Paul Bachmann bereits 1892 verwendete, um die Ordnung von Funktionen zu definieren.

Man sagt daher genauer, Bubblesort und Selection-Sort haben eine Komplexität von $O(n^2)$, gesprochen „groß-Oh von n quadrat“. Quicksort und Tournament-Sort haben eine Komplexität von $O(n \log n)$. Diese Notation wird Ihnen im Großteil der Informatik-Fachliteratur begegnen.

O (n) oder das Streben nach dem perfekten Algorithmus

Lange Zeit galten Tournament-Sort & Co. als die besten möglichen Sortiervverfahren. Es gibt sogar mathematische Beweise dafür, dass es eigentlich gar keine besseren Verfahren als solche von der Komplexität $O(n \log n)$ geben kann. Informatiker sind da jedoch recht flexibel und oft lassen sich auch die strengsten mathematischen Beweise dadurch umgehen, dass man die Voraussetzungen ändert. Dies gilt zum Beispiel für das Sortiervverfahren Proxmap-Sort.

Proxmap-Sort ist eigentlich bei genauerer Betrachtung gar nichts Revolutionäres, sondern etwas, das Generationen von Menschen bereits beim Sortieren von Karten oder ähnlichen Objekten intuitiv durchführen:

Nehmen Sie ein Päckchen gewöhnliche Spielkarten und einen großen Tisch. Aufgabe ist, die Karten nach ihren Werten zu sortieren. Die Herangehensweise nach den bisherigen Algorithmen wäre, die Karten erst einmal genauso wie sie sind auf dem Tisch auszulegen und dann einen der besprochenen Algorithmen durchzuführen.

Effektiv machen das aber in Realität die wenigsten Menschen. Vielmehr nutzen wir bereits beim Auslegen der Karten unser Wissen über deren Verteilung. Die Verteilung besagt, wie viele Karten welches Typs im Stapel sind. Normalerweise wissen wir also, dass jede Karte von As bis König in einer Farbe genau einmal vorkommt. Nehmen wir zu Beginn eine 7 auf, dann legen wir sie nicht auf die linke Seite, sondern bereits ungefähr an die Position, an die sie gehört,

also etwa in die Mitte des Tisches. Ein As kommt an den Anfang, ein König ans Ende. Auf diese Weise brauchen wir im Idealfall jede Karte nur ein einziges Mal anzufassen und legen sie direkt an die korrekte Position.

Genau das Gleiche macht Proxmap: Für jede Karte rät er, wo diese in einer Liste hinkommt und legt sie direkt dort ab. Ist die Stelle ausnahmsweise bereits besetzt, muss man eventuell eine oder zwei Karten verschieben, um Platz zu schaffen. Im Wesentlichen kommt der Algorithmus jedoch damit aus, jede Karte nur ein einziges Mal zu betrachten.

Probieren Sie das gleich einmal mit den blauen Zahlen auf Ihren Sortierkarten aus: Mischen Sie 10 der Karten und versuchen Sie, diese direkt auf den ersten 10 Feldern Ihres Papierspeichers zu verteilen.



Das Vorgehen ist klar: Die Karten zeigen Werte zwischen 0 und 999999. Daher liegt die Vermutung nahe, dass die erste Stelle bereits ein guter Anhaltspunkt für die Position der Karte ist.

Sie werden bemerken, dass hier trotzdem oft noch so genannte Kollisionen auftreten: Mehrere Karten kommen eigentlich auf den gleichen Platz. Proxmap-Sort geht daher danach nochmals die gesamte Liste durch und sortiert mit einem bekannten Verfahren, zum Beispiel Selection-Sort nach.

Dieses ungünstige Verhalten hat hauptsächlich zwei Gründe:

- Mit 10 Karten auf 10 Speicherpositionen füllen wir den gesamten verfügbaren Speicher lückenlos. Es ist kein Raum für statistisch doch nicht ganz so gleich verteilte Karten.
- Mit der geringen Anzahl von 10 Karten ist die Chance sehr hoch, dass wir zufällig recht ähnliche Karten ziehen. Ein ähnliches Problem haben Meinungsforscher, wenn sie zu wenige Testpersonen haben. Ein repräsentatives Ergebnis bekommt man nur mit einer entsprechend hohen Stichprobe.

Verteilen Sie daher in einem weiteren Experiment 15 gemischte Karten auf 20 Speicherplätze und sortieren Sie dabei wieder nach der blauen Zahl.

Wenn Sie Muße haben, können Sie auch den Papierspeicher noch auf 100 Plätze erweitern (ich verspreche: Mehr wird es nicht mehr). Sie können das mit den unbeschrifteten Teilen aus dem Anhang machen oder einfach mit aufgeklebten Haftnotizen. Beschriften Sie die Speicherstellen mit #00 bis #99. Jetzt verteilen Sie 75 Karten entsprechend der blauen Zahlen im Speicher.

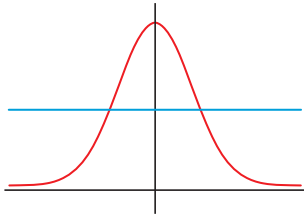


Nun ist auch noch die zweite Stelle relevant: Bei 20 Speicherplätze kommen Zahlen zwischen 0 und 49.999 auf das 0. Feld, solche zwischen 50.000 und 99.999 auf das 1. usw.

Kollisionen sind jetzt sehr selten, obwohl wir immerhin 75 % des Speichers füllen! Sie müssen am Ende die Lücken nur noch schließen, indem Sie die Karten zusammenrutschen. Das kostet ebenfalls maximal eine Verschiebeoperation pro Karte.

Macht man den Speicherplatz noch größer bzw. die Anzahl zu sortierender Karten kleiner, werden Kollisionen äußerst unwahrscheinlich und es gilt tatsächlich, dass jede Karte im Durchschnitt nur zwei Mal angefasst werden muss – einmal für das Einsortieren und nochmals für das Zusammenrutschen.

Damit ist der Aufwand „ $2 \cdot \text{Anzahl der Karten}$ “ oder $2n$. Da wir konstante Faktoren prinzipiell weglassen, kommen wir für Proxmap-Sort auf $O(n)$.



Die **Gauß-Verteilung** (rot) gegenüber der **Normalverteilung** (blau). Auf der x-Achse ist dabei der Wertebereich, auf der y-Achse die Wahrscheinlichkeit. Während die blaue Linie die gleiche Wahrscheinlichkeit aller Werte anzeigt, hat die rote Linie ein deutliches Maximum und fällt dann in beide Richtungen stark ab. Wegen ihrer Form wird die Gauß-Verteilung auch Glockenkurve genannt.

Alles ganz normal?

Versuchen Sie nun, genau die gleichen Karten nochmals auf die Speicherplätze zu verteilen, jetzt aber aufgrund der roten Zahlen. Plötzlich funktioniert unser Verfahren nicht mehr so gut. In den allermeisten Fällen gibt es jetzt deutlich mehr Kollisionen. Woran liegt das?

Augenscheinlich sind die roten Zahlen genauso zufällig verteilt wie die blauen. Hier trägt jedoch der Augenschein: Die blauen Zahlen sind gleich verteilt. Das bedeutet, jede Zahl zwischen 0 und 999.999 kommt mit derselben Wahrscheinlichkeit vor. Auf diese Weise ergibt sich auch eine gleichmäßige Verteilung der Karten auf die Speicherplätze – ein Garant für wenige Kollisionen.

Die roten Zahlen sind jedoch normal verteilt (auch Gauß-verteilt genannt). Hier kommen die Zahlen um 500.000 deutlich häufiger vor als solche am Rand des Zahlenbereichs, wie 100.000 oder 900.000. Daher kommt es um die entsprechenden Speicherstellen auch vermehrt zu Kollisionen – Proxmap-Sort kann nicht mehr optimal funktionieren.

Ein Beispiel, das dies noch klarer macht, ist die alphabetische Sortierung tatsächlicher Worte. Die grünen Wörter auf den Sortierkarten sind repräsentativ für die deutsche Sprache. Sie sind gleichmäßig aus dem Duden entnommen.

Die grünen Buchstaben auf den Speicherstellen 0 bis 25 können zur Proxmap-Sortierung der Worte verwendet werden. Versuchen Sie, wie viele Kollisionen es mit 20 Karten gibt...



Offenbar ist unsere Sprache ebenfalls nicht gleich verteilt: Selbstverständlich kommen Wörter mit den Anfangsbuchstaben A oder S viel häufiger vor als solche mit X oder Y.

Egal ob normal verteilte Zahlen oder Worte: Proxmap-Sort scheint damit nicht recht zu funktionieren ... oder können wir ihn so modifizieren, dass er besser läuft?

Auch wenn man in der Gesellschaft Unrecht nicht mit Unrecht bekämpfen sollte – in der Informatik funktioniert es, unregelmäßig verteilte Karten mit einer unregelmäßigen Verteilung der Felder des Sortierverfahrens auszugleichen.

Sortieren Sie nochmals 15 Karten nach den roten Zahlen auf den ersten zwanzig Speicherplätzen – diesmal aber gemäß der roten Beschriftung der Speicherstellen.



Plötzlich funktioniert die Sache wieder mit wenigen Kollisionen. Die rote Beschriftung des Papierspeichers berücksichtigt die Verteilung der Karten und reserviert für wahrscheinlichere Zahlen auch mehr Speicherplätze als für unwahrscheinliche.

Sie merken also, dass Proxmap-Sort dann arbeiten kann, wenn die Verteilung der zu sortierenden Größen bekannt ist und man dadurch für jede Größe die Position im Speicher erraten kann. In der Praxis analysiert das Verfahren erst die zu sortierenden Daten, um die Verteilung kennen zu lernen. Damit kann der Computer recht gut raten, wo ein neuer Datensatz in den Speicher gehört.

Proxmap – ein Sortierverfahren?

Weiter oben hatte ich geschildert, dass eigentlich bei $O(n \log n)$ Schluss ist mit der Optimierung von Sortierverfahren. Mit Proxmap-Sort erreicht man aber unter günstigen Bedingungen $O(n)$, nur eben – wie bereits erwähnt – unter veränderten Voraussetzungen. Nach den letzten Betrachtungen sollten Sie auch den Unterschied zwischen den klassischen Sortierverfahren und Proxmap erkennen. Welcher ist das?



Für die klassischen Sortierverfahren muss man lediglich eine so genannte Ordnungsrelation zwischen zwei zu sortierenden Größen kennen: Für je zwei Karten muss klar sein, welche die größere ist bzw. ob beide als gleichwertig anzusehen sind. Dabei ist egal, wie viel größer oder kleiner die eine Karte gegenüber der anderen ist. Auch die Verteilung der Werte spielt keine Rolle – die Sortierverfahren laufen ohne das Wissen darüber zuverlässig und mit gleicher Leistungsfähigkeit ab. Sie basieren ja auch im Wesentlichen darauf, dass immer wieder zwei Kartenwerte miteinander verglichen werden.

Proxmap-Sort benötigt zusätzliche Informationen: Wie wir Menschen beim Sortieren unser Wissen über die Verteilung der Karten implizit nutzen, versucht auch Proxmap die korrekte Position einer Karte anhand ihres Wertes zu erraten. Das setzt voraus, dass man überhaupt etwas über die Verteilung der Größen kennt. Falls dies nicht der Fall ist, kann Proxmap-Sort auch nicht sinnvoll verwendet werden.

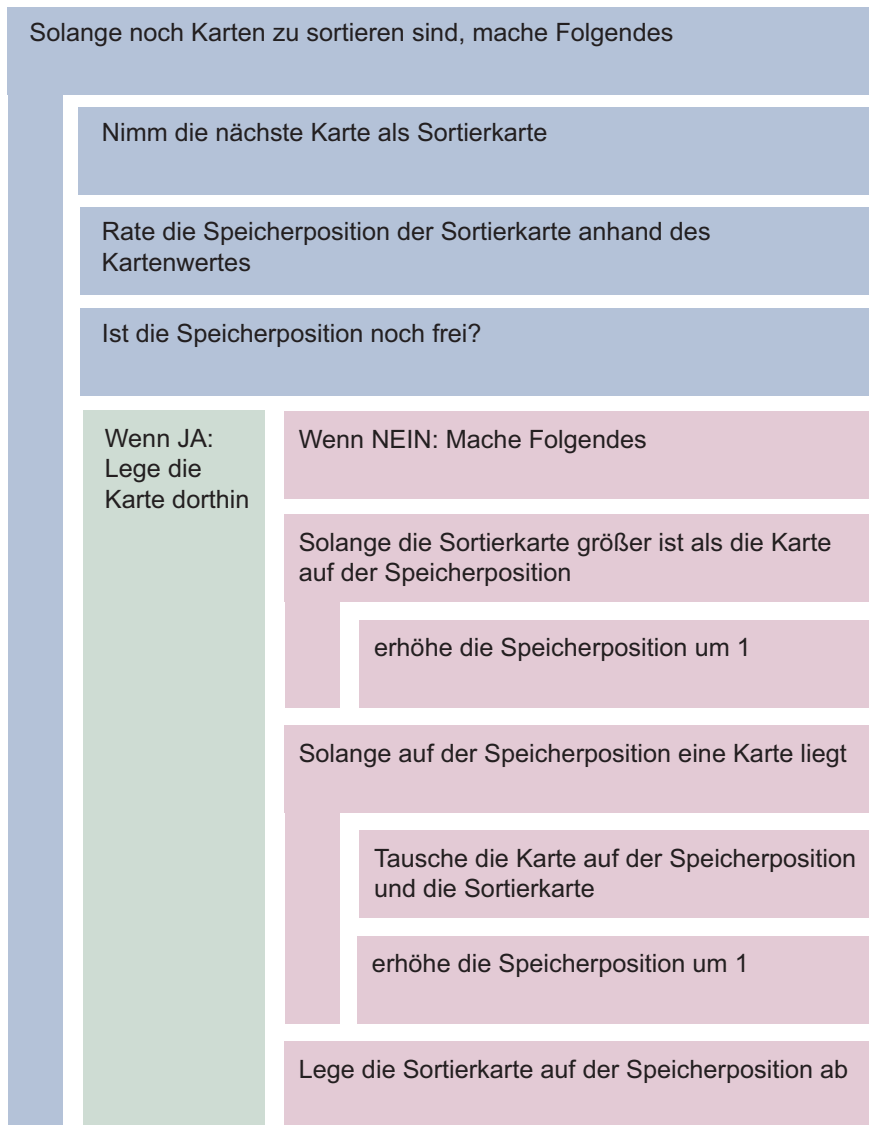
Ein einprägsames Beispiel ergibt sich, wenn man als Sortierkriterium keine Zahlen oder Buchstaben nimmt: „Größer als“ sei bei der Herstellung eines Autos definiert als „kommt später im Produktionsprozess“. Damit ist

Motor zusammenbauen < Motor einsetzen < Armaturenbrett einbauen < Auto waschen

Proxmap-Sort wird sich aber sehr schwer tun, etwa dem Vorgang „Motor einsetzen“ den korrekten Platz zuzuweisen unter all den anderen Vorgängen wie „Achse einbauen“, „Reifen aufziehen“ usw. Das liegt daran, dass man zwar von je zwei Vorgängen sagen kann, in welcher Reihenfolge sie durchgeführt werden müssen, man kann aber keinem einzelnen Vorgang eine genaue Durchführungszeit zuweisen.

Proxmap – der Algorithmus

Vielleicht haben Sie bemerkt, dass wir bisher bei Proxmap noch nicht ganz korrekt gearbeitet haben: Kam es zu einer Kollision, konnte plötzlich unser Papierspeicherplatz trotzdem mehrere Werte fassen. Um das Kapitel sauber abzuschließen, erhalten Sie hier daher noch einen Vorschlag für die „saubere“ Implementierung des Algorithmus:



Versuchen Sie es – der Algorithmus funktioniert. Testen Sie das auch wieder mit einer Strichliste für jede Speicher- und Verschiebeoperation und ermitteln Sie dadurch die Qualität experimentell. Achten Sie darauf, dass Sie den Speicher immer nur zu etwa 75 % füllen, weil sonst Proxmap-Sort nicht mehr günstig laufen kann: Es gibt zu viele Kollisionen.

Resümee

Einerseits wissen Sie nun, wie ein Computer auf verschiedene Arten große Datenmengen sortieren kann, um dann später wieder schnell darauf zuzugreifen. Fast noch wesentlicher ist jedoch die Erkenntnis, dass man das gleiche Ziel fast immer auf sehr unterschiedliche Arten und Weisen erreichen kann.

Um zu beurteilen, welche dieser Möglichkeiten nun die beste ist, benötigt man einen Qualitätsmaßstab. Im Fall von Algorithmen ist die Ordnung der Komplexität ein solcher Maßstab. Sie gibt an, wie stark die Rechenzeit in Abhängigkeit der Problemgröße anwächst.

Oft kann man zusätzliche Informationen über die gegebenen Daten dazu verwenden, günstigere Algorithmen zu verwenden, wie im Falle von Proxmap-Sort, wo das Wissen um die Verteilung der Werte zu einem Verfahren mit der Komplexität $O(n)$ führt, während bei den klassischen, nur auf Vergleichen beruhenden Verfahren der günstigste Fall bei $O(n \log n)$ liegt.

Literatur



Buch: Robert Sedgewick „**Algorithmen in Java. Grundlagen, Datenstrukturen, Sortieren, Suchen. Teil 1–4**“; Pearson Studium; 3. Aufl. 2003; ISBN 3-8273-7072-8

Sedgewick beschreibt sehr anschaulich, wie verschiedene Sortieralgorithmen (und andere) funktionieren und dann auch, wie sie in der aktuellen Programmiersprache Java implementiert werden. Wer gut Englisch kann, sollte sich eher den Doppelband mit Teil 5 kaufen, der leider nicht auf Deutsch erhältlich ist.



3. Ich packe meinen Koffer und ...

Es war einmal ein tapferer Mann aus dem Volk, der rettete die Prinzessin Tausendschön vor dem Räuber Abi Lala und seinen zweiundvierzig Schergen. Leider musste sich der König nun etwas als Belohnung ausdenken: Wäre der Mann ein Prinz gewesen, hätte er ihm einfach die Hand der Prinzessin anbieten können. Aber für einen gewöhnlichen Bürger würde der König wohl oder übel seine Schatzkammer öffnen müssen.

Um den Schaden jedoch zu begrenzen, gab der König dem Mann eine Holzkiste und sprach „Du darfst dir alles aus der Schatzkammer nehmen, was in die Kiste passt, so dass sich der Deckel noch schließen lässt.“ Er dachte, dass jeder sicher so von den Schätzen geblendet sei, dass er eher billigen Tand als wirklich wertvolle Stücke auswählte.

Hier hat er jedoch nicht mit unserer Beteiligung gerechnet! Die Informatik hat dem Mann aus dem Volk sogar eine eigene Kategorie von Aufgaben gewidmet, genannt „Rucksackprobleme“.

Das Rucksackproblem

Bereits in den vorherigen Kapiteln haben wir sehr viel mit Abstraktion gearbeitet und das Problem zunächst vereinfacht, um darüber besser nachdenken zu können. Hier nehme ich diesen Schritt schon jetzt vorweg und biete Ihnen die gesammelten „abstrakten“ Schätze aus dem Anhang – rechteckig und aus einer glatten Zahl von Quadraten zusammengesetzt.

Schneiden Sie sie aus und versuchen Sie, die Kiste möglichst optimal zu füllen. Wir gehen hier mal davon aus, dass jedem Schatz ein Zertifikat mit einem genauen Wert (selbstverständlich einheitlich in Golddublonen bemessen) beiliegt – dieser steht als ganze Zahl neben der Abbildung.

Die so genannten „Trivialfälle“ sind hier übrigens bereits eliminiert: Nehmen wir an, es existierten zwei Gegenstände mit exakt der gleichen Größe, aber unterschiedlichem Wert. Dann legt man selbstverständlich den billigeren Gegenstand im Vorfeld bereits als „uninteressant“ ab. Daher steht hier nur ein Objekt pro Größe zur Wahl (eben das jeweilige teuerste).



Sie sehen bereits an dem einfachen Beispiel, dass es gar nicht so einfach ist, das Optimum aus allen Möglichkeiten herauszufinden: Es passen sechs Goldbarren in die Kiste, aber auch zwei Wunderlampen. Vielleicht sind aber drei Juwelenboxen wertvoller? Oder gar der Mini-Thron – dann passt aber fast sonst nichts in die Kiste...

Wie auch bereits zuvor, können wir hier die Lösung durch stures Ausprobieren finden: Alle Möglichkeiten werden systematisch durchgegangen, die wertvollste Kombination wird beibehalten. Das ist für das vorliegende Beispiel machbar, aber bereits sehr zeitaufwendig. Überlegen Sie, wie viele Möglichkeiten Sie bereits hätten, wenn die Kiste auch nur doppelt so groß wäre...

Abbildung 3.1 Gefüllte Schatzkiste mit einem Wert von 46 Dublonen. Geht das noch besser?



Über die Problematik des sturen Probierens hatten wir bereits im Kapitel über Routenplanung philosophiert, das möchte ich hier nicht wiederholen. Allerdings versichere ich Ihnen, dass der Aufwand zum Probieren hier mit der Größe der Kiste ebenfalls sehr stark anwächst. Das Problem wird damit schnell unübersichtlich und lässt sich praktisch gar nicht mehr lösen.

Mehr ist weniger

Divide et impera

Teile und herrsche – Erinnern Sie sich noch daran? Wenn nicht: In Kapitel 2 wird dieses Prinzip ausführlich erörtert.

Statt herumzuprobieren wollen wir daher das Problem konsequent angehen. Eine entscheidende Strategie der Informatik haben Sie ja bereits kennen gelernt: „divide et impera“.

Wenn das Gesamtproblem zu groß zum Lösen ist, versuchen wir es in kleinere Pakete zu zerteilen. Das machen wir so lange, bis die Teilprobleme handhabbar klein sind.

Um das Problem in kleinere Happen zu unterteilen, müssen wir allerdings zunächst feststellen, wodurch überhaupt ein Problem dieser Art als „groß“ oder „klein“ gilt – wir bestimmen die Problemgröße. Beim Routenplaner ist das zum Beispiel die Anzahl der Orte auf einer Landkarte, beim Sortieren die Anzahl der zu sortierenden Objekte.

Ihre Aufgabe: Überlegen Sie, was beim Packen des Rucksacks die Problemgröße ist.



Zugegeben: Die Frage war vielleicht ein wenig unfair formuliert, aber ansonsten hätte ich schon zu viel verraten. Sie sind bestimmt trotzdem auf die korrekte Antwort gekommen.

Ist „Größe der Kiste“ vielleicht eine Problemgröße?

Ist „Anzahl unterschiedlicher Gegenstände“ vielleicht Problemgröße?

Beides ist richtig: Wie bereits oben ausgeführt, wird eine Lösung immer schwieriger, je größer die Kiste ist und je mehr Gegenstände man daher gleichzeitig einpacken muss (das gilt übrigens nur, wenn man wirklich an einer exakten Lösung interessiert ist, aber dazu mehr weiter unten). Zusätzlich steigt die Problemgröße selbstverständlich auch an, wenn man mehr unterschiedliche Gegenstände ausprobieren muss.

Sie können Ihre Vermutung über eine Problemgröße normalerweise leicht untermauern: Überlegen Sie, welches das „kleinste Problem“ ist, wenn Sie Ihre Überlegungen zugrunde legen.

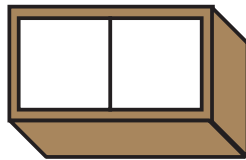


Abbildung 3.2 Mini-Schatzkiste

„Größe der Kiste“: Nehmen Sie eine sehr kleine Kiste, zum Beispiel die aus Abbildung 3.2. Ist hier die Lösung des Problems besonders schwierig? Nein, denn es gibt überhaupt nur zwei Möglichkeiten – die Kiste leer lassen (und das wäre wirklich dumm) oder einen Goldbarren hineinladen, was immerhin sechs Dublonen bringt.

„Anzahl unterschiedlicher Gegenstände“: Nehmen Sie die große Kiste, aber wir gehen davon aus, in der Schatzkammer befände sich nur eine einzige Sorte Schatz, zum Beispiel Goldbarren. Hier gibt es auch nicht viel zu entscheiden: Man lädt so viele Goldbarren in die Kiste, wie hineinpassen. Abbildung 3.3 zeigt eine solche Kiste.



Abbildung 3.3 Schatzkiste nur mit Goldbarren, immerhin 36 Dublonen wert

Beim Sortieren von Karten war eine mögliche Vorgehensweise, mit einer einzelnen Karte anzufangen (die per Definition sortiert ist). Dann wurde die Problemgröße jeweils um eine Karte erweitert, die in die bereits sortierte Folge leicht eingefügt werden konnte.

Vielleicht finden Sie ja auf dieser Basis auch eine Lösung für das Rucksackproblem? Versuchen Sie es: Fangen Sie mit kleinen Kisten oder wenigen unterschiedlichen Objekten an und erhöhen Sie dann die Problemgröße langsam.



Die Lösung gestaltet sich schwieriger als erwartet:

Fängt man mit einer kleinen Kiste an und steigert deren Größe, kommt man recht schnell auf optimale Ergebnisse für Kisten mit 2, 3, 4 und 5 Quadraten (wir nennen diese fortan 2er-Kiste, 3er-Kiste usw. oder Kisten der Größen 2, 3, 4 usw.). Danach fängt das Rätseln an: Welche der Teillösungen müssen kombiniert werden, um eine noch größere Kiste zu packen? Auch hier können wir nur alle Möglichkeiten ausprobieren, was uns zum Ausgangspunkt unserer Betrachtungen zurückwirft.

Versuchen wir also, gleich die große 12er-Kiste zu nehmen, fangen aber zunächst an, sie ausschließlich mit Goldbarren zu füllen, wie schon in Abbildung 3 gezeigt. Danach erweitern wir die Problemgröße: Auch Geldbündel sind nun erlaubt. Recht schnell kann man feststellen, dass vier Geldbündel (mit 44 Dublonen Wert) besser sind als die sechs Goldbarren (die nur 36 Dublonen wert waren).

Offenbar sind also Goldbarren schlechter als Geldbündel, wir können sie also für die folgenden Betrachtungen außer Acht lassen. Weiter geht es und auch die Schmuckkassette mit 15 Dublonen Wert soll nun im Angebot sein. Wiederum können wir den Wert unserer Kiste steigern, indem wir die Geldbündel entsorgen und stattdessen drei Schmuckkassetten einladen. 45 Dublonen sind nun das Ergebnis.

Als Nächstes nehmen wir den wertvollen Sextanten hinzu, der mit so viel Zubehör kommt, dass er fünf Quadrate ausfüllt. In unsere Kiste passen daher nur zwei Sextanten mit einem Gesamtwert von 42 Dublonen. Das ist weniger als 45. Daher bleiben wir bei den Schmuckkassetten, oder?

Ganz so einfach ist es nicht, denn legen wir zu den zwei Sextanten noch einen Goldbarren (was gut passt), steigt der Wert unserer Kiste sogar auf 48. Unsere Entscheidung, die Goldbarren als „zu billig“ außer Acht zu lassen, war also ziemlich voreilig! Auch hier müssen wir im späteren Verlauf der Problemlösung immer wieder ganz an den Anfang zurückspringen, um das Optimum herauszuholen, was effektiv auch einfach das Durchspielen aller Möglichkeiten bedeutet.

Ist hier also „divide et impera“ gar nicht zu gebrauchen? Müssen wir uns bei diesem Problem geschlagen geben?

Selbstverständlich nicht: Die Informatik hat noch ein paar Tricks auf Lager. In diesem Fall heißt der Trick: „Mehr ist weniger“ oder „Wenn man mehr Probleme löst, hat man letztlich weniger Arbeit“.

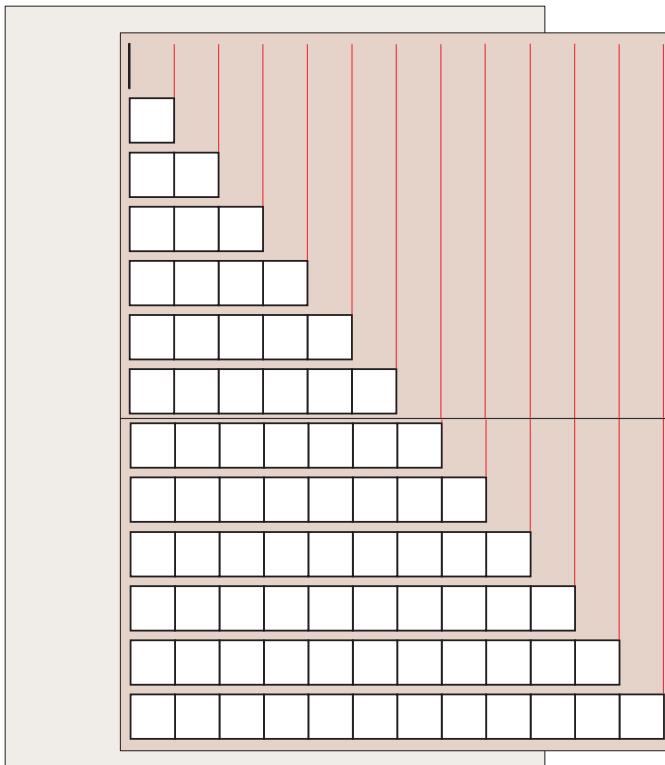
Wie ist das genau zu verstehen? Das Teilen von Problemen ist manchmal schwierig und man weiß nicht genau, welche Teilprobleme man genau lösen muss, um

sie dann zu einer Gesamtlösung zu kombinieren. In solchen Fällen ist es manchmal hilfreich, einfach *alle* möglichen Teilprobleme zu lösen und danach zu entscheiden, welche man kombiniert. So etwas nennt man *dynamische Programmierung*.

Dynamische Programmierung

Strategie zur Lösung komplexer Probleme, besonders zur Lösung von Optimierungsaufgaben. Man löst zunächst viele Teilprobleme und verwendet diese Bausteine, um die nächstgrößeren Probleme zu lösen usw. bis das Gesamtproblem gelöst ist.

Um dieses Prinzip auf unser Rucksackproblem anzuwenden, benötigen wir viele Kisten: von einer ganz kleinen Kiste (mit überhaupt keinem Platz darin) bis zur größten Kiste mit 12 Quadraten Platz. Kleben Sie hierfür die beiden braunen Teile aus dem Anhang zusammen. Ich nenne dieses Gebilde „Maxikiste“, da hier sozusagen die Kisten aller ganzen Größen bis 12 enthalten sind. Außerdem benötigen Sie noch ein leeres Blatt Papier. Nehmen Sie die Maxikiste hochkant und legen Sie das Blatt so darunter, dass es links hervorschaut, wie in Abbildung 3.4 gezeigt.



Richard Ernest Bellman

(1920 – 1984) studierte Mathematik und beschäftigte sich danach mit theoretischer Physik. Ab 1952 befasste er sich dann bei der Rand-Corporation mit Optimierungsproblemen und adaptierte ein vorher nur von Physikern verwendetes Prinzip für die Informatik, das er „Dynamische Programmierung“ nannte. Heute ist ein Algorithmus zur Erstellung optimaler Binärbäume nach ihm benannt.

Abbildung 3.4 Viele leere Kisten in der Maxikiste

Jetzt fangen wir ganz klein an: Gehen Sie davon aus, es gäbe nur die kleinste Sorte Schatz zur Auswahl – die Goldbarren. Wir gehen also bei dieser Sorte Problemgröße von der kleinsten aus. Gemäß dem Prinzip der dynamischen Programmierung ermitteln wir nun für den anderen Typ Problemgröße – der Kistengröße – nicht nur eine Lösung, sondern gleich alle bis zur Größe 12. Hierfür sind in der Maxikiste alle Kisten bis zur Größe 12 enthalten!

Mit nur einer Sorte Schatz ist es leicht, die Kisten entsprechend optimal zu füllen. Tun Sie das und schreiben Sie links auf den Zettel daneben den Wert jeder Kiste!



Abbildung 3.5 zeigt die Maxikiste. Selbstverständlich kann man in die kleinste Kiste mit der Größe 0 nichts hineingeben. Auch in die Kiste mit nur einem Quadrat Platz passt kein Goldbarren – sie bleibt daher frei. Die restlichen Kisten können mit ein bis sechs Barren gefüllt werden, man stopft einfach so viele hinein, wie passen. Die größte Kiste hat einen Wert von sechs Barren zu sechs Dublonen, also 36 Dublonen.

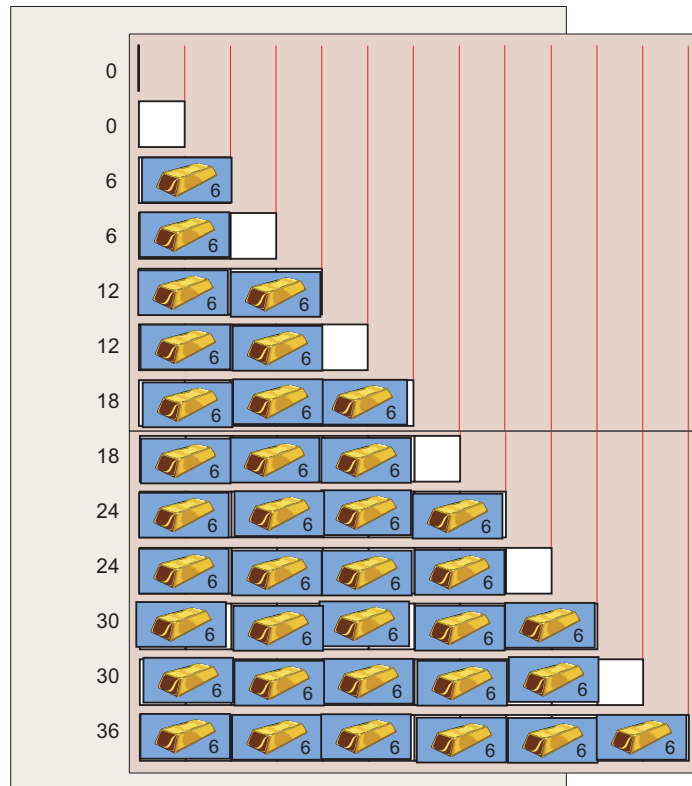


Abbildung 3.5 Lösung des Rucksackproblems bis zur Kistengröße 12, wenn nur Goldbarren zur Wahl stehen

Um es noch einmal festzuhalten: Die Maxikiste zeigt nun die optimale Lösung mit der kleinsten Problemgröße – nur die kleinsten Schätze dürfen ausgewählt werden.

Wie können wir diese Ergebnisse verwenden, um noch den nächstgrößeren Schatz einzubeziehen? Diesmal zeige ich Ihnen, was dafür zu tun ist, und wir überlegen hinterher zusammen, warum diese Strategie sinnvoll ist.

Nehmen Sie einen Schatz „Geldbündel“ (gelb) und legen Sie ihn rechts an die kleinste Kiste an – das ist die Kiste mit Größe 0. Sie haben jetzt sozusagen den neuen Schatz zu der Kiste hinzugelegt. In welche (größere) Kiste würden die Inhalte nun passen? Selbstverständlich in die Kiste der Größe 3, wie Sie an den roten Linien ablesen können. Abbildung 3.6 zeigt das.

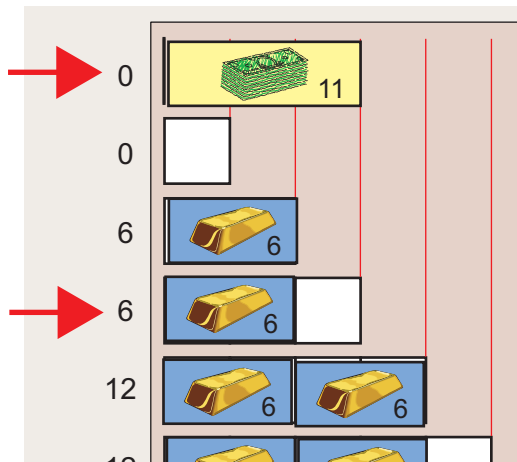


Abbildung 3.6 Den „Inhalt“ der Kiste mit Größe 0 und einen Schatz der Größe 3 könnten wir zusammen in die Kiste der Größe 3 packen

Bestimmen Sie nun den Wert beider markierter Zeilen: Der Wert der unteren Zeile steht bereits links davon, er beträgt sechs Dublonen. Der Wert der oberen Zeile ist die Summe aus dem Wert der Kiste (0) und dem Wert des neuen Schatzes (11).

Es gibt nun zwei Möglichkeiten:

1. Der Wert der oberen Zeile ist höher. Die neue Kombination aus dem vorigen Inhalt der kleineren Kiste plus dem neuen Schatz ist also wertvoller als der vorher bestimmte „optimale“ Inhalt. Dieser ist also unter Berücksichtigung des neuen Schatzes nicht mehr optimal, wir tauschen ihn aus gegen die obere Kombination (das heißt, wir entfernen die Schätze aus der Kiste und setzen neue vom Schatzvorrat ein, bis die untere markierte Kiste genau die gleichen Schätze enthält wie die obere plus dem neuen Schatz).
2. Der Wert der oberen Zeile ist nicht höher. Offenbar ist das alte Optimum auch noch unter Berücksichtigung des neuen Schatzes optimal. Es wird nichts verändert.

Im Beispiel beträgt der Wert der oberen Zeile 11 Dublonen, der Wert der unteren Zeile liegt bei lediglich 6 Dublonen. Es lohnt sich also, die Kiste neu zu packen. Tun Sie das! Selbstverständlich müssen Sie auch den Wert links von der Kiste auf den neuesten Stand bringen. Das Ergebnis sehen Sie in Abbildung 3.7.

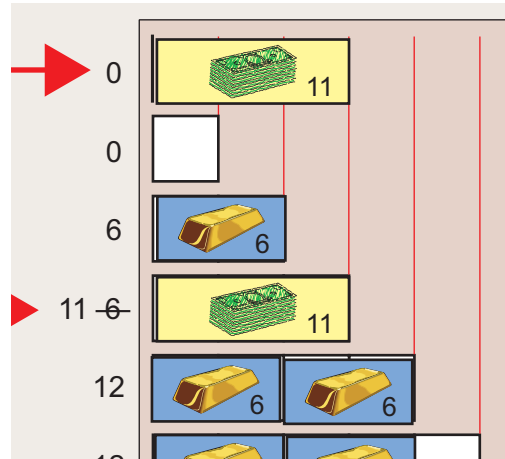


Abbildung 3.7 Der „Inhalt“ der Kiste mit Größe 0 und ein Schatz der Größe 3 wurden nun in die Kiste der Größe 3 gepackt

Nun verschieben wir den Schatz um eine Position nach unten und legen ihn rechts an die nächstgrößere Kiste an. Jetzt vergleichen wir den neuen, kombinierten Inhalt mit dem vorhandenen Inhalt der 4er-Kiste. Abbildung 3.8 zeigt das Vorgehen.

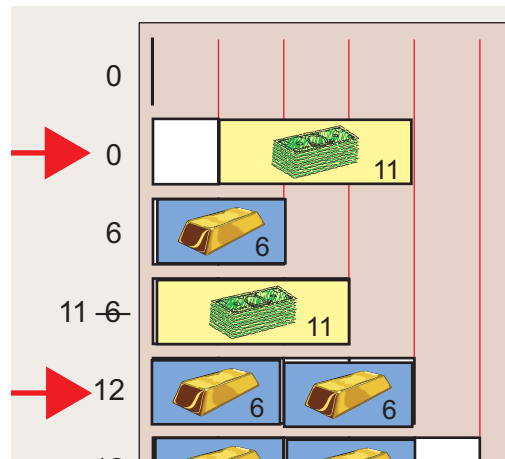


Abbildung 3.8 Ist die vorhandene 4er-Kiste mit zwei Goldbarren günstiger als die neue Kombination aus Geldbündel und dem (leeren) Inhalt der 1er-Kiste?

Wieder vergleichen Sie den Wert der bisher optimalen 4er-Kiste (12) und den Wert der neuen Kombination (11). Diesmal ist die alte Kombination günstiger und Sie verändern den Inhalt der Kisten nicht.

Wieder wird der neue Schatz eine Position nach unten verschoben. Versuchen Sie diesmal, erst selbst die Entscheidung zu treffen, ob Kisteninhalte ausgetauscht werden (und wenn ja welche). Abbildung 3.9 zeigt die Ausgangsposition. Lesen Sie dann weiter.

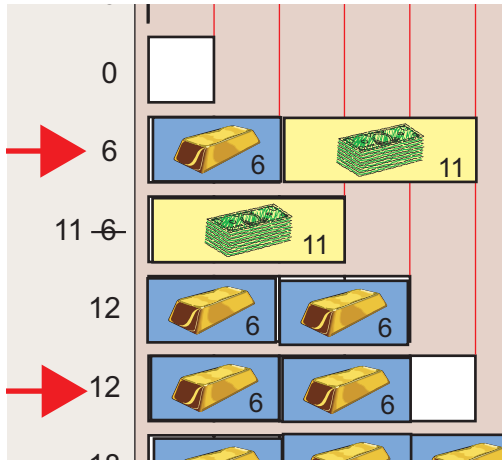


Abbildung 3.9 Ist die vorhandene 5er-Kiste günstiger als die neue Kombination aus Geldbündel und dem Inhalt der 2er-Kiste?



Selbstverständlich ist hier der neue Inhalt vorzuziehen, da er einen Wert von 17 gegenüber dem vorhandenen Wert von 12 hat. Abbildung 3.10 zeigt das Ergebnis.

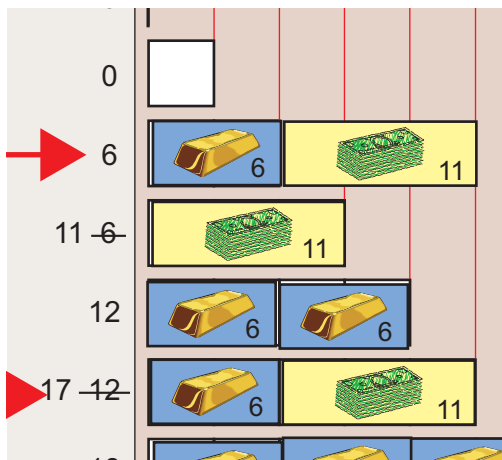


Abbildung 3.10 Auch die 5er-Kiste muss neu gepackt werden

Wieder wird das Geldbündel eine Zeile tiefer geschoben. Nun liegt es an einer Kiste, die wir bereits umgepackt haben. Das ist aber gar kein Problem – folgen Sie einfach dem bisherigen Schema und vergleichen Sie die Kombination mit der 6er-Kiste. Auch hier tauschen wir den Inhalt aus, weil wir eine Verbesserung erzielen, wie in Abbildung 3.11 zu sehen.

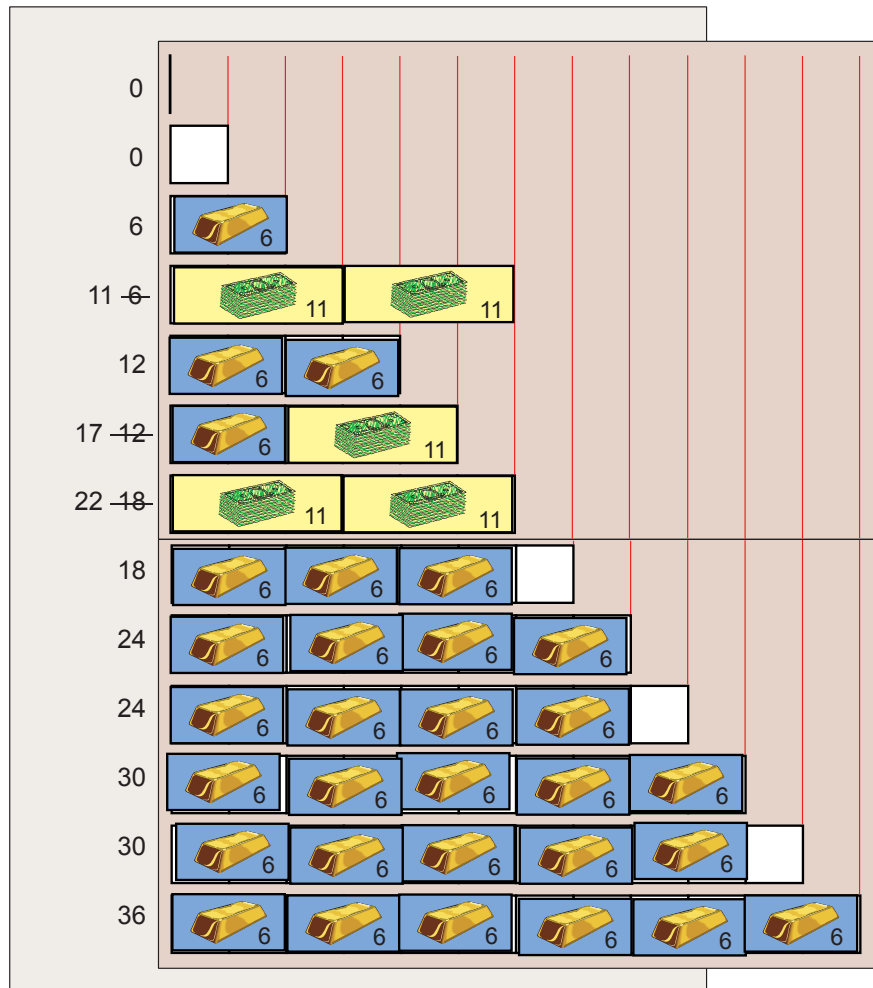


Abbildung 3.11 Die 6er-Kiste hat nach dem Tausch des Inhalts einen Wert von 22

Führen Sie nun das Verfahren für die gesamte Maxikiste durch! Selbstverständlich können Sie aufhören, wenn Sie mit einer 13er-Kiste vergleichen müssten, die in unserem Schema nicht vorkommt.



Das Ergebnis sehen Sie in Abbildung 3.12. Die meisten Kisten enthalten nun ein oder mehrere wertvolle Geldbündel.

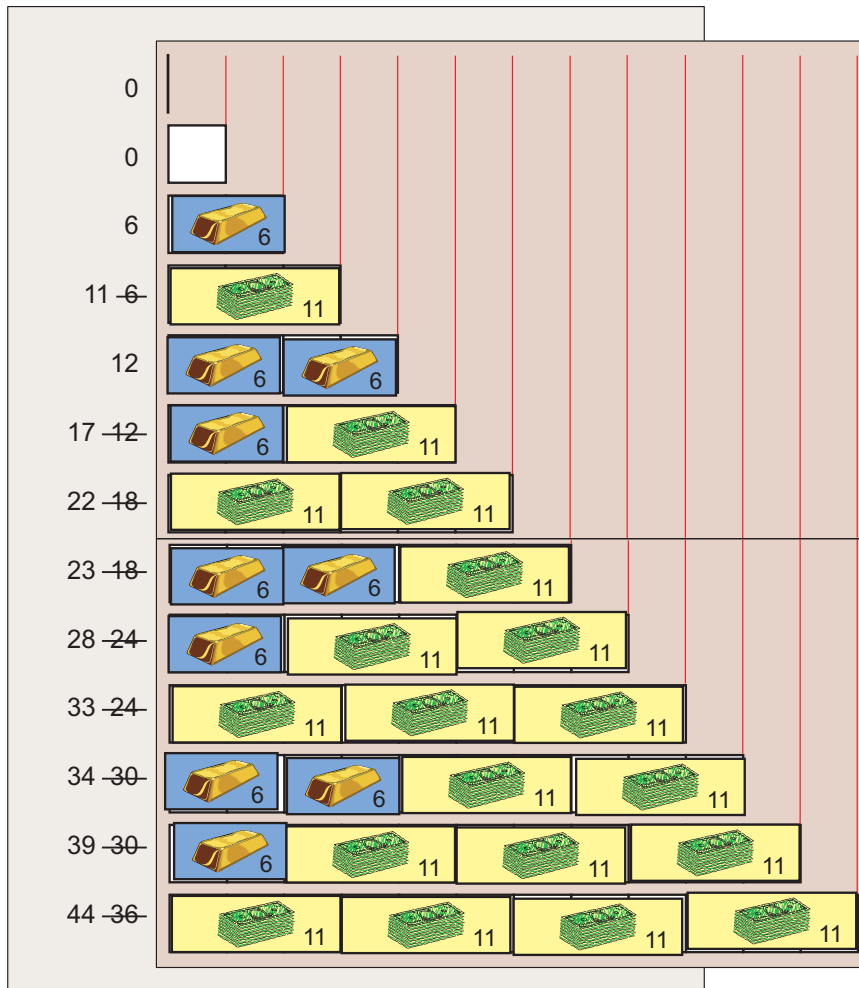


Abbildung 3.12 Nun ist auch das Geldbündel als Schatz berücksichtigt

Weiter geht es und der nächstgrößere Schatz – die Schmuckschatulle (orange) – steht nun ebenfalls zur Auswahl.



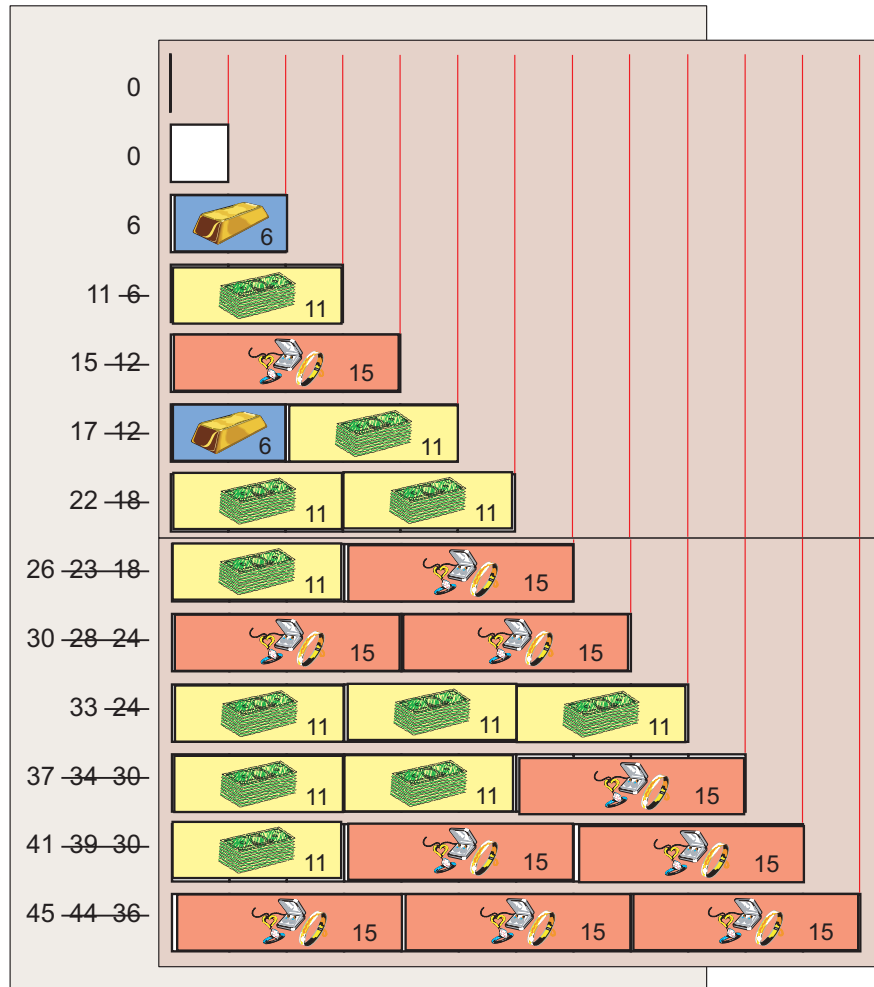


Abbildung 3.13 Die Schmuckschatulle verändert die optimalen Ergebnisse erneut

Sie sehen das Ergebnis in Abbildung 3.13. Erkennen Sie, dass sich die Bestückung der Kisten mit nur einem Quadrat mehr Platz komplett ändern kann: Die optimale 8er-Kiste ist nur mit Schmuckschatullen bestückt, während die optimale 9er-Kiste lediglich Geldbündel enthält.

Als Nächstes fügen wir den Sextanten (rosa) in der optimalen Kiste ein.



In Abbildung 3.14 sehen Sie, dass eine optimal gefüllte Kiste keinesfalls randvoll sein muss. Der Sextant ist offenbar so viel wert, dass es sich sogar lohnt, bei 6er- und 11er-Kisten ein Feld freizulassen.

Das zeigt auch, warum es in unserem Verfahren sehr wichtig ist, die 1er-Kiste jedes Mal zu betrachten, auch wenn diese immer leer bleibt: Nur durch Anlegen des Sextanten an die 1er-Kiste konnte die Verbesserung des Wertes der 6er-Kiste erreicht werden.

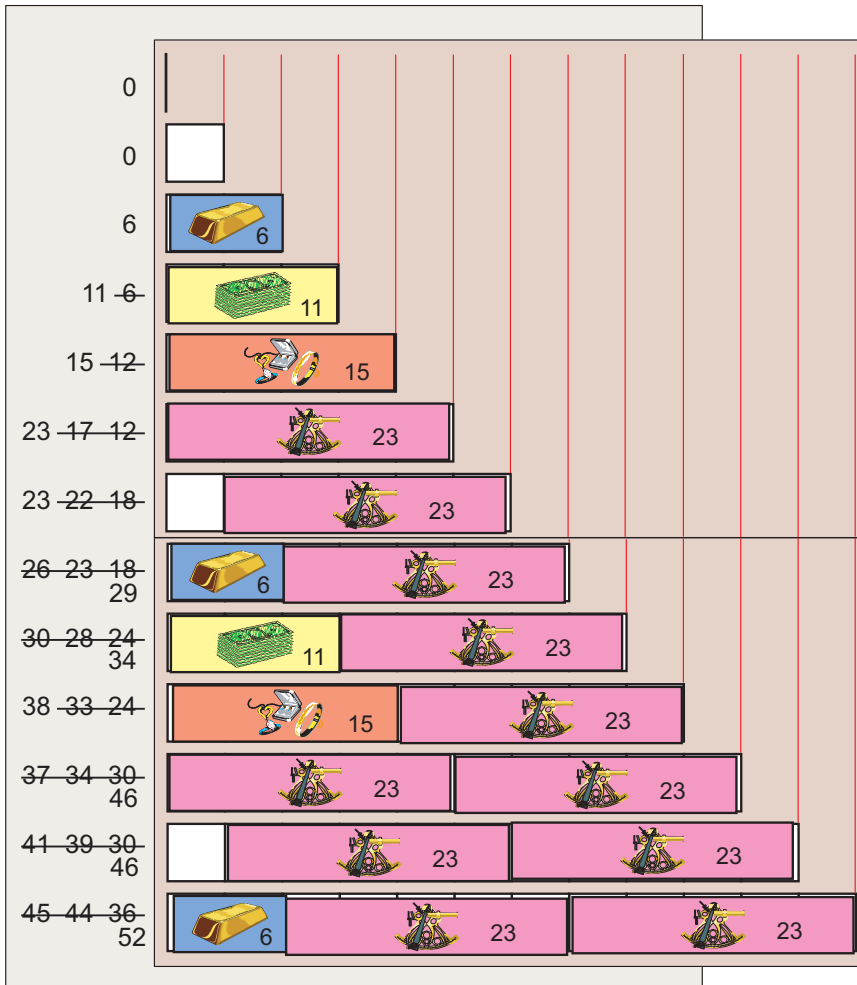


Abbildung 3.14 Der Sextant verändert die optimalen Ergebnisse erneut

Es stehen aber immer noch größere Schätze zur Verfügung, als Nächstes sind die hellblauen Zauberlampen an der Reihe.



Sie kommen nur in wenige Kisten, wie Abbildung 3.15 zeigt.

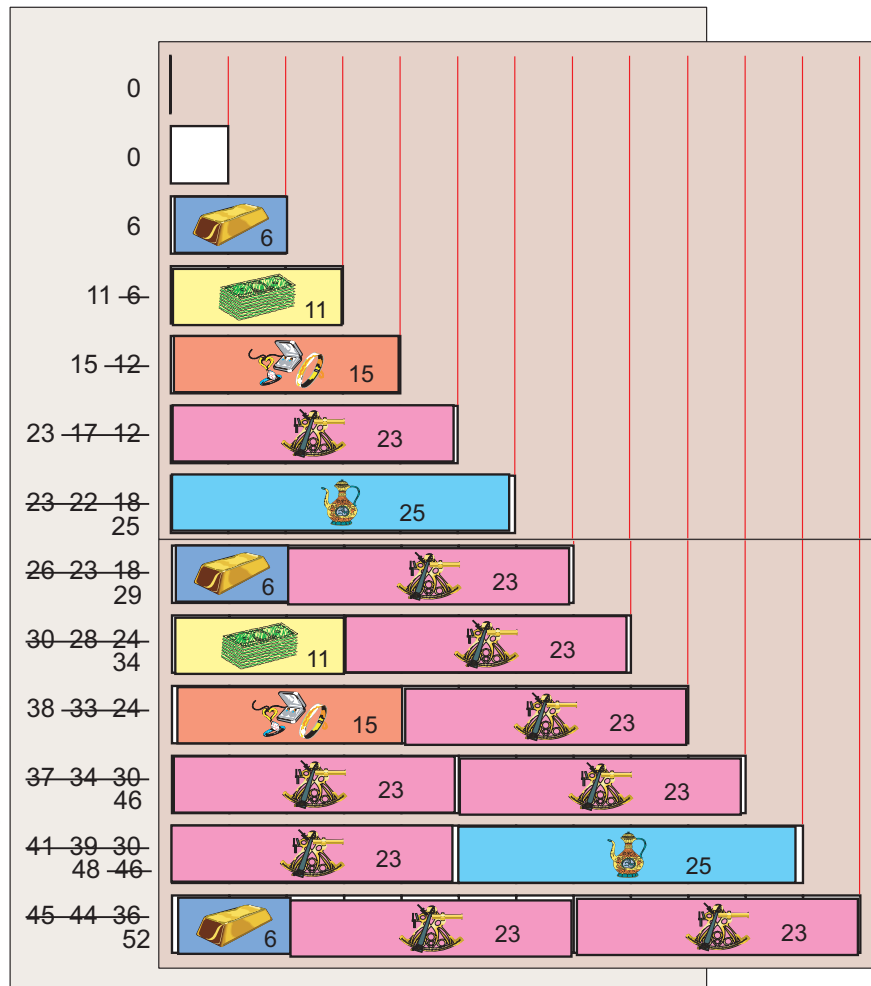


Abbildung 3.15 Nur zwei Wunderlampen werden für die optimalen Kisten benötigt

Jetzt sollten Sie den Dreh heraushaben: Führen Sie das Verfahren fort, bis alle Schätze berücksichtigt wurden.



Abbildung 3.16 zeigt das Endergebnis. Zwei der Schätze, Krone und Zepter (grün) und Mini-Thron konnten sich gar nicht durchsetzen – sie waren für alle Kisten zu wenig wertvoll.

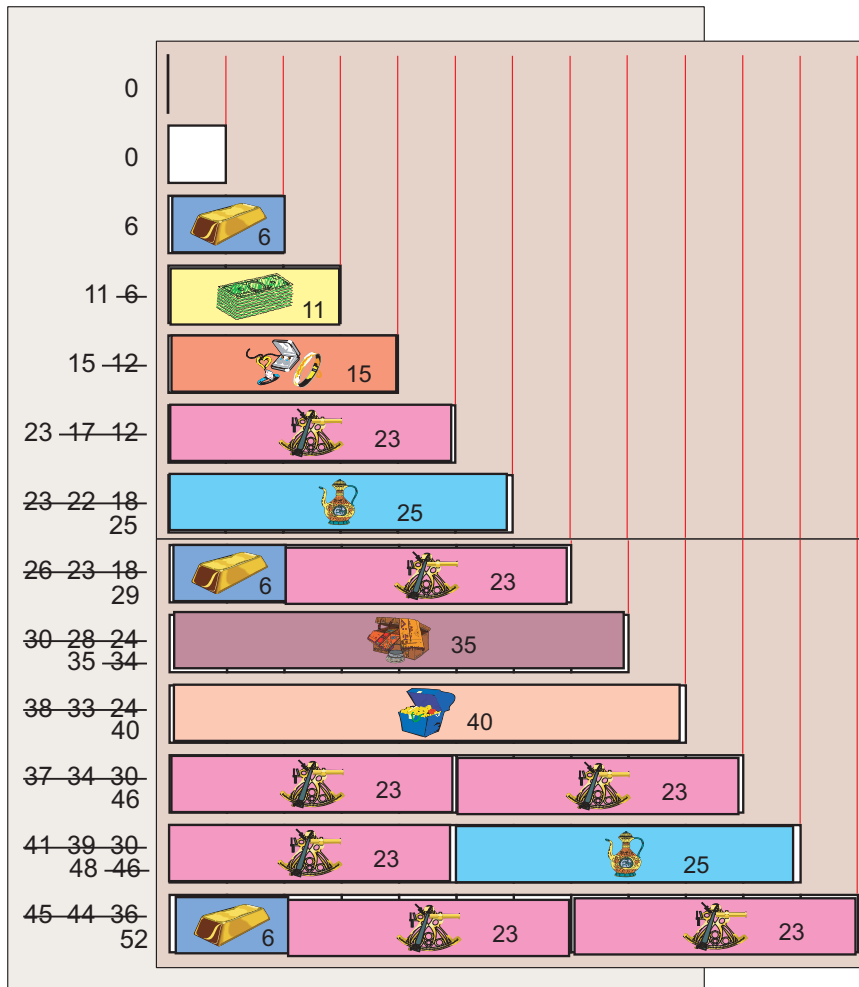


Abbildung 3.16 Endergebnis in der Maxikiste

Die Lösung für unseren Helden vom Beginn des Kapitels ist: Er nimmt sich zwei Sextanten und einen Goldbarren, um eine Belohnung im Wert von 52 Dublonen zu erhalten. Das ist fast die Hälfte mehr als die 36 Dublonen, die man durch einfaches Gold-Stapeln erzielen konnte.

Wer hätte gedacht, dass am Ende der Goldbarren doch noch sinnvoll einsetzbar ist, nachdem am Anfang des Verfahrens fast alle 2er-Schätze aus den optimalen Kombinationen verschwunden waren.

Funktioniert das denn auch immer?

Bei der Durchführung des Verfahrens oben bin ich Ihnen noch eine Begründung schuldig geblieben, ob, warum und wie es funktioniert.

Gehen wir nochmals zum Start zurück, als es sozusagen nur einen Schatz gab: den Goldbarren. Die einzelnen Kisten waren bis zum Rand mit Goldbarren gefüllt, mehr passten nicht hinein.

Wir können daher festhalten, dass alle Kisten optimal gefüllt waren.

Jetzt nehmen wir den nächstgrößeren Schatz hinzu. Wir fangen mit der kleinsten (optimal gepackten) Kiste an und schauen, was passiert, wenn man deren Inhalt mit dem neuen Schatz kombiniert. Das machen wir nacheinander mit allen Kisten, bis der Inhalt zu groß wird.

Allgemein gesprochen kombinieren wir also den Inhalt einer Kiste A mit dem neuen Schatz S. Um das unterzubringen, benötigen wir eine Kiste B mit der Größe von Kiste A plus der Größe des Schatzes S.

Wie erreicht man, dass die Kiste B optimal gepackt ist? Hier gibt es zwei Möglichkeiten:

1. In einer optimal gepackten Kiste B kommt Schatz S nicht vor
In diesem Fall besteht die optimale Kiste aus dem, was schon vor der Betrachtung von Schatz S als optimal festgestellt wurde: Der Inhalt der Kiste aus dem letzten Durchlauf.
2. Kiste B benötigt Schatz S, um optimal gepackt zu sein
In diesem Fall packen wir schon einmal Schatz S in die Kiste. Wie viel Platz haben wir jetzt noch? Genau! Es steht noch so viel Platz zur Verfügung, wie

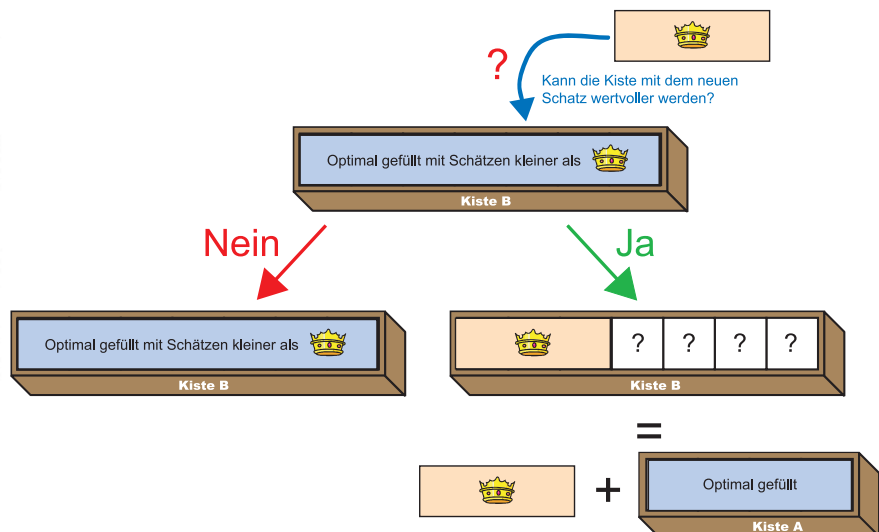


Abbildung 3.17 Entweder der neue Schatz ist sinnvoll in der Kiste unterzubringen oder nicht – eine weitere Möglichkeit gibt es nicht

auch Kiste A bietet. Kiste A war aber nach der Voraussetzung bereits optimal gepackt, also liegt es nahe, deren Inhalt zusätzlich zu Schatz S in die Kiste B zu packen. Genau das machen wir auch in obigem Verfahren!

Abbildung 3.17 zeigt das grafisch. Indem wir also immer bei den kleineren Kisten anfangen, stellen wir sicher, dass diese bereits optimal sind, wenn wir zu den größeren Kisten kommen. Auf diese Weise können wir die kleineren Kisten schon zur „Konstruktion“ der größeren benutzen.

Wozu dient diese Erkenntnis?

Könige sind rar geworden! Die Schatzkammern sind Bankkonten gewichen und die stehen meistens unter der starken Überwachung der Parlamente. Prinzessinnen sind emanzipiert und lassen sich nicht mehr so leicht von bösen Räufern entführen.

Wozu also die Vorbereitung auf einen Gang durch die Schatzkammer? Selbstverständlich gibt es für die Lösung des Rucksackproblems sehr viele ernsthafte Anwendungen.

Auf der Hand liegt der Einsatz bei einer Spedition: Ein Fuhrunternehmen hat einen LKW zur Verfügung und kann verschiedene Waren mit unterschiedlichen Gewinnspannen transportieren. Wie packt er den LKW, um den größtmöglichen Nutzen zu haben?

Bei einer erweiterten Version geht es gleich um eine ganze LKW-Flotte: Stellen Sie sich vor, der Spediteur hat einen Auftrag angenommen, eine ganze Reihe unterschiedlicher Objekte zu transportieren – egal wie viele Fuhren er hierfür benötigt. Wenn er es dank dichter Packung schafft, die Ladung optimal auf die LKWs aufzuteilen (also mit wenig ungenutztem Raum), kann er unter Umständen ganze Fuhren einsparen und damit seinen Gewinn steigern.

Denken Sie auch an Arbeitsabläufe: Ein Softwareprojekt muss in einem Monat fertig werden. Wie verteilt man die verschiedenen Aufgabenpakete auf die zehn vorhandenen Mitarbeiter, so dass möglichst alle gleichmäßig bis zum Ende ausgelastet sind und somit das Projekt in der kürzesten Zeit fertig wird?

Während man bei menschlichen Mitarbeitern nicht immer genau schätzen kann, wie lange sie tatsächlich für die Durchführung eines Arbeitspakets brauchen, geht das bei Computern deutlich besser: Genau das gleiche Problem muss gelöst werden, wenn eine Aufgabe von einem Computer mit vielen Prozessoren möglichst schnell gelöst werden muss. Auch hier gilt es, die vorhandenen Teilprozesse so auf die Prozessoren zu verteilen, dass alle ungefähr die gleiche Last haben und so alle ungefähr gleichzeitig fertig werden. Das funktioniert natürlich sowohl im menschlichen als auch im Computer-Fall nur, wenn die einzelnen Teilaufgaben voneinander unabhängig bearbeitet werden können.

Der Algorithmus

Jetzt sind Sie wieder gefragt: Können Sie aus dem Beispiel von oben einen allgemeinen Algorithmus ableiten, der dann künftig für alle Rucksackprobleme dieser Art verwendbar ist?

Um die Reihenfolge der einzelnen Operationen besser kenntlich zu machen, dürfen Sie gerne auch wieder Zeiger verwenden. Hierfür nehmen Sie die Zeiger „Kiste A“ und „Kiste B“ im Bastel-Anhang und schneiden sie aus!

Tipp: In der Lösung werden Sie einige so genannte „Schleifen“ benutzen. Dies sind die Konstrukte der Art „Wiederhole das Folgende, bis eine bestimmte Bedingung zutrifft“. In der bisher verwendeten Notation für Algorithmen werden diese so genannten „kopfgesteuerten Schleifen“ dargestellt wie in Abbildung 3.18. Wenn die Bedingung von Anfang an nicht stimmt, werden die Anweisungen aus dem grauen Kästchen kein einziges Mal ausgeführt.

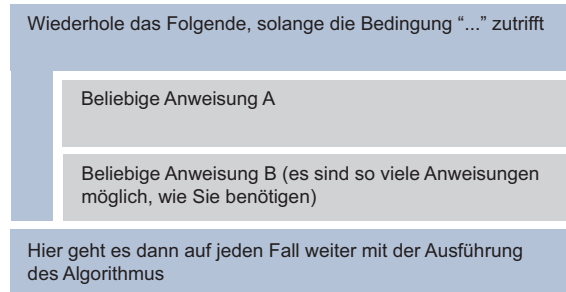


Abb. 3.18: Notation für die kopfgesteuerte Schleife

Im Gegensatz dazu stehen die „fußgesteuerten Schleifen“ der Art „Wiederhole das Obenstehende, solange eine bestimmte Bedingung zutrifft“. Die Notation sehen Sie in Abbildung 3.19. Hier werden die Anweisungen aus dem grauen Kästchen auf jeden Fall einmal ausgeführt, bevor die Bedingung überprüft wird.

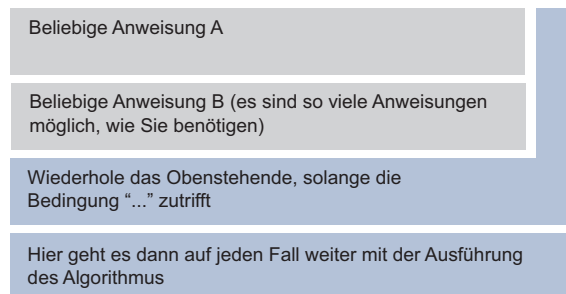


Abb. 3.19: Notation für die fußgesteuerte Schleife



Sie dürfen die Zeiger verwenden, um die Kisten zu markieren, die miteinander verglichen werden sollen. Der Algorithmus kann dann zum Beispiel folgendermaßen aussehen:

Setze  neben die Kiste mit Größe 1

Wiederhole das Folgende, solange  auf eine Kiste zeigt

Fülle die Kiste neben  bis zum Anschlag mit dem kleinsten Schatz

Schiebe  um eine Position nach unten

Nehme den zweitkleinsten Schatz zur Hand

Solange man noch einen Schatz in der Hand hält

Setze  neben die Kiste mit Größe 0

Wiederhole das Folgende, solange  auf eine Kiste zeigt

Lege den Schatz rechts an die Kiste, auf die  zeigt

Lege  neben die Kiste, die so groß ist wie die neben  und der Schatz zusammen

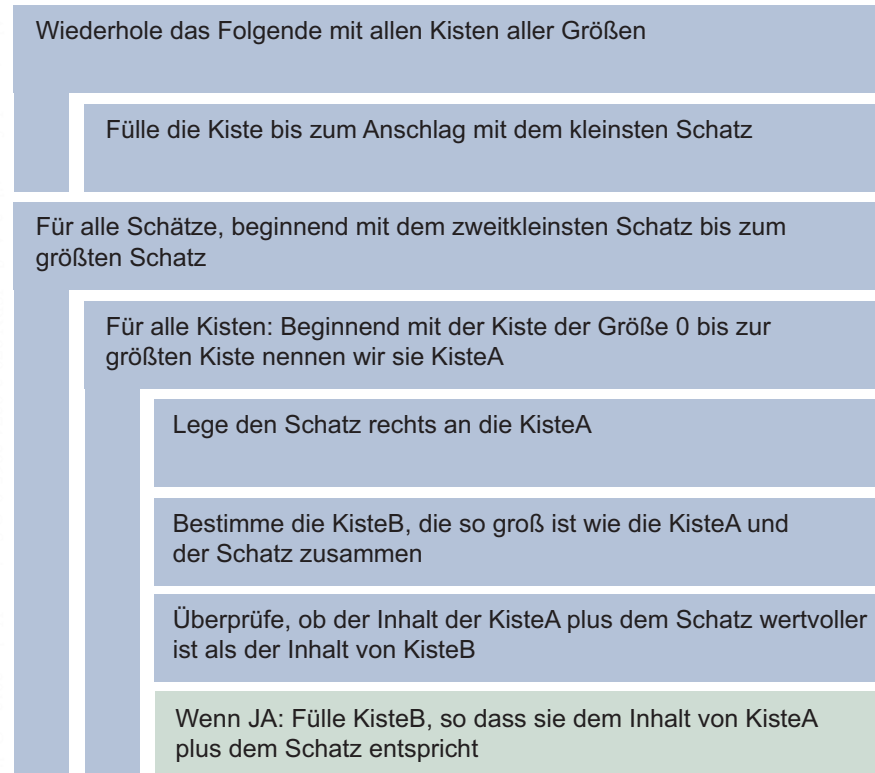
Überprüfe, ob der Inhalt von  plus dem Schatz wertvoller ist als der Inhalt von 

Wenn JA: Fülle  so, dass sie dem Inhalt von  plus dem Schatz entspricht

Schiebe  um eine Position nach unten

Lege den Schatz aus der Hand und nimm den nächstgrößeren Schatz in die Hand, falls es noch einen größeren gibt

Wenn Sie bereits firm mit der Durchführung der Algorithmen sind und sich merken, welche Kisten Sie gerade betrachten, lässt sich das Verfahren auch etwas knapper darstellen:



Sie sehen: Es ist günstig, dieses Buch auch im nächsten Urlaub griffbereit zu haben, wenn es darum geht, die wertvollsten Schnäppchen und Mitbringsel im knappen Fluggepäck unterzubringen ...

Was steckt dahinter?

Vielleicht erkennen Sie, dass Sie das Prinzip der dynamischen Programmierung bereits kennen gelernt haben: Beim Routenplaner war es ebenfalls einfacher, sämtliche Routen zu Orten zu bestimmen, die weniger weit weg als unser Ziel waren.

Genau das machten auch die Ameisen: Sie erkundeten gleichzeitig alle möglichen Orte, bis sie am Ziel ankamen. Genau das haben wir aber auch beim Packen der Kisten gemacht: Wir haben alle kleineren Kisten gepackt und darauf aufbauend den optimalen Inhalt der größeren bestimmt.

Das erinnert sehr stark an ein Prinzip, das in der Mathematik bereits sehr lange bekannt ist: die vollständige Induktion.

Vollständige Induktion

Die vollständige Induktion ist ein zweistufiges mathematisches Beweisverfahren. Es funktioniert insbesondere für Aussagen in Bezug auf ganze Zahlen. Man beweist, dass die Aussage für eine bestimmte kleine Zahl a gilt. Ferner beweise man: Falls die Aussage für die Zahl $n - 1$ gilt, dann gilt sie auch für die Zahl n .

Damit hat man dann bewiesen, dass die Aussage für alle ganzen Zahlen größer oder gleich a gilt.

Vollständige Induktion kennt man prinzipiell, seit 1654 der französische Mathematiker und Philosoph Blaise Pascal mathematische Beweise nach diesem Schema durchgeführt hat. Erst mit der Einführung eines formalen Systems für natürliche Zahlen um 1880 wurde daraus dann ein allgemein gültiges Beweisverfahren, das heute auch in der Informatik häufig eingesetzt wird.

Beispielsweise möchten wir beweisen, dass folgender Satz gilt:

$$1 + 2 + 3 + \dots + n = \frac{n \cdot (n + 1)}{2}$$

Zunächst kommt die so genannte Induktionsverankerung: der Beweis, dass der Satz für eine bestimmte kleine Zahl gilt. In diesem Fall wählen wir $n = 1$.

Die Rechnung ist

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

Wir sehen leicht, dass diese Rechnung stimmt.

Als Nächstes kommt der so genannte Schluss von n auf $n + 1$. Dabei nehmen wir an, dass wir den Satz für die Zahl $n - 1$ bereits bewiesen hätten und versuchen, ihn ebenfalls für die Zahl n zu beweisen.

Als Formel ausgedrückt nehmen wir als gegeben an

$$1 + 2 + 3 + \dots + (n - 1) = \frac{(n - 1) \cdot n}{2}$$

Zu beweisen ist

$$1 + 2 + 3 + \dots + n = \frac{n \cdot (n + 1)}{2}$$

Formen wir das etwas um, steht dort

$$1 + 2 + 3 + \dots + (n - 1) + n = \frac{n \cdot (n + 1)}{2}$$

Den ersten Teil der linken Seite kennen wir bereits aus unserer Annahme. Wir setzen hierfür die Formel ein, die wir bereits als bewiesen angenommen hatten:

$$\frac{(n-1) \cdot n}{2} + n = \frac{n \cdot (n+1)}{2}$$

Eine kleine Umformung der rechten Seite führt zu

$$\frac{(n-1) \cdot n + 2 \cdot n}{2} = \frac{n \cdot (n+1)}{2}$$

Nun kann man leicht sehen, dass die Gleichung gilt, wir haben bewiesen, dass der Satz für n gilt unter der Voraussetzung, dass er auch für $n-1$ gilt.

Da wir ihn für $n = 1$ bewiesen hatten, folgt daraus also, dass er ebenfalls für $n = 1 + 1 = 2$ gilt. Wenn er für $n = 2$ gilt, ist er ebenfalls für $n = 3$ bewiesen usw. Insgesamt können wir also sagen, dass die Behauptung für alle $n \geq 1$ gilt. Kommen wir nach diesem mathematischen Intermezzo zur dynamischen Programmierung zurück. Auch hier gibt es quasi eine Verankerung, bei der das Problem für eine kleine Problemgröße gelöst wird. Im Laufe des Algorithmus leiten wir dann jeweils aus einer kleineren Lösung die nächstgrößere ab.

Im Beispiel des Rucksackproblems bestand die Verankerung darin, das Problem für den Fall zu lösen, dass es nur einen einzigen Schatz zur Wahl gibt: den kleinsten.

Danach konstruierten wir aus der Problemlösung der Größe n (eine optimal gefüllte Kiste der Größe n) und dem nächsten Schatz der Größe s die Problemlösung der Größe $n + s$: Die Kiste der Größe $n + s$ wird optimal, wenn man sie entweder so lässt wie sie ist oder mit dem Inhalt der Kiste n + Schatz s füllt – je nachdem, welche der Möglichkeiten wertvoller ist.

Die Begründung habe ich bereits weiter oben vorgeführt. Sie setzt voraus, dass man tatsächlich eine optimale Kiste der Größe n hat. Genau genommen handelt es sich hier um eine Art Induktionsschluss. Auf diese Weise wird gezeigt, dass das Verfahren für immer höhere Problemgrößen ebenfalls korrekt funktioniert.

Versuchen Sie jetzt, für den Dijkstra-Algorithmus aus dem ersten Kapitel die Verankerung und den Induktionsschluss zu finden!



Verankerung:

Der kürzeste Weg vom Startort S zu sich selbst ist 0.

Schluss:

Wir nehmen an, wir haben den (tatsächlich) kürzesten Weg vom Startort S zu den Orten O_1 bis O_{n-1} bestimmt. Dann ist der nächste Ort O_n derjenige, der einem Ort O_x aus der Menge der Orte O_1 bis O_{n-1} benachbart ist und bei dem die Summe aus dem (bereits bestimmten optimalen) Weg von S nach O_x und dem Weg von O_x nach O_n minimal ist.

Sie erkennen hier, dass der Übergang zwischen der Informatik und der Mathematik fließend ist. Viele Teilgebiete haben beide Wissenschaften gemeinsam, so die diskrete Mathematik (oder Mathematik ganzer Zahlen), die auch hier verwendet wurde.

Das verflixte NP

Eines der großen Probleme bei der vorgestellten Lösung des Rucksackproblems lässt sich aber an der Beziehung zwischen dynamischer Programmierung und vollständiger Induktion auch ablesen:

Die Lösung funktioniert nur für „ganzzahlige“ Problemgrößen.

Was bedeutet das? Alle Kisten waren sozusagen „linear“, sie bestanden aus einer ganzzahligen Anzahl von Quadraten, ebenso die Schätze. Hier konnte man leicht von einer kleineren Kiste auf eine entsprechend größere schließen.

Reale Probleme arbeiten selbstverständlich mit dreidimensionalen Kisten. Jeder „Schatz“ hat unterschiedliche Ausmaße in Länge, Breite und Höhe (oder ist sogar völlig unregelmäßig, wie eine lose Krone). Außerdem lassen sich reale Schätze nicht in ganzzahlige Kästchen (zum Beispiel glatte Zentimeter-Werte) einteilen. Darüber hinaus geben die echten Schatzkammern oft nicht beliebig viele Schätze einer Sorte her. Was passiert, wenn unser Algorithmus anzeigt, dass man zwei Zepter einpacken soll und es ist aber nur eines da?

Hier versagt das Verfahren! Sie sehen, dass oft eine kleine Veränderung der Aufgabenstellung ausreicht, um aus einem „einfachen“ Problem eines zu machen, für dessen Lösung kein Algorithmus existiert, der in polynomieller Zeit arbeitet. Bereits im Zusammenhang mit dem Dijkstra-Algorithmus diskutierten wir ausführlich darüber.

Man hat eine ganze Menge von Problemen, die man momentan nicht in polynomieller Zeit lösen kann, in eine Kategorie gepackt, die so genannten „NP-vollständigen Probleme“. Dazu gehören zum Beispiel das nicht ganzzahlige Rucksackproblem und auch das Travelling-Salesman-Problem (allerdings sind lange nicht alle Probleme, die nicht in polynomineller Zeit lösbar sind, NP-vollständig!)

Polynomiell, nicht polynomiell

Auch in Kapitel 1 haben wir im Abschnitt „Was steckt dahinter“ bereits einführend über dieses Thema philosophiert. Lesen Sie doch dort nochmals nach, falls hier etwas unklar bleiben sollte! Dort steht übrigens auch, was das Travelling-Salesman-Problem eigentlich ist ...

Man konnte bisher für keines der Probleme eine Lösung in polynomieller Zeit finden, aber auch keinen Beweis dafür, dass eine solche Lösung nicht möglich sei. Gleichzeitig **hat** man aber nachgewiesen, dass – sollte es eine entsprechende Lösung für **eines** der Probleme geben – **alle** Probleme dieser Art in polynomieller Zeit lösbar sind.

1 Million Dollar

Im Klartext: Wenn Sie sich hinsetzen und für eines der beschriebenen Probleme einen Algorithmus basteln, der mit einer (beliebig schlechten, aber) polynomiellen Rechenzeit im Verhältnis zur Problemgröße auskommt, sind Sie der Held der Informatik für die nächsten Jahre!

Das würde aus Ihnen auch sofort einen Dollar-Millionär machen! So viel Preisgeld hat nämlich das Clay Mathematics Institute of Cambridge, Massachusetts (CMI), ausgeschrieben für denjenigen, der eines von sieben „Millenium Problems“ löst.

Die Frage „ $P=NP?$ “ gehört auch zu diesen Problemen, und um das Geld zu bekommen, müssen Sie nur entweder zeigen, dass NP-vollständige Probleme in polynomieller Zeit lösbar sind oder beweisen, dass dies nicht möglich ist...

Resümee

Sie haben in diesem Kapitel mit der dynamischen Programmierung erfahren, dass es manchmal durchaus sparsamer sein kann, alle Lösungen eines Problems bis zu einer Größe n zu bestimmen, als sich nur auf die Lösung der gewünschten Problemgröße zu konzentrieren.

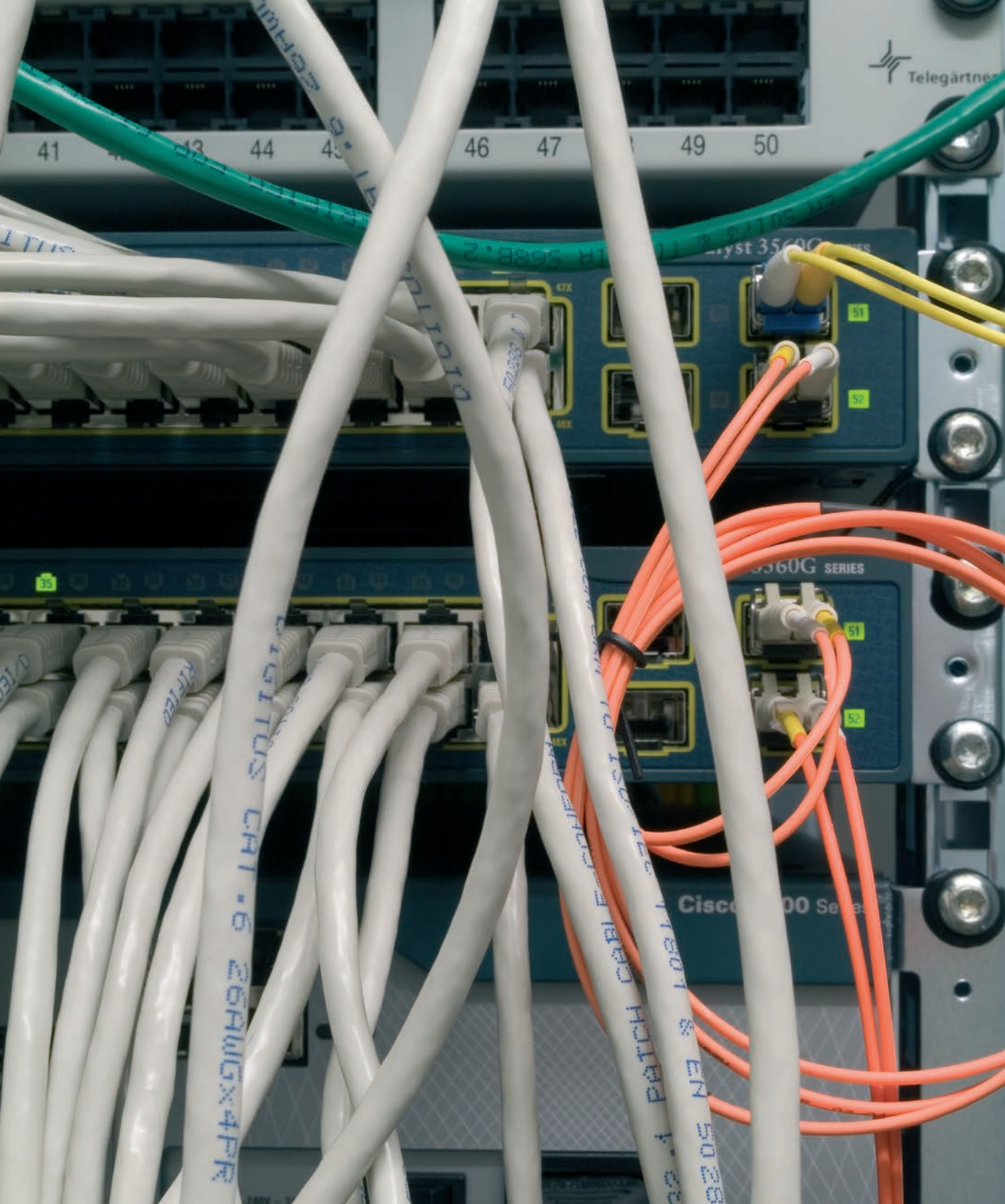
Gleichzeitig ist das hier behandelte Rucksackproblem ein schönes Beispiel dafür, wie wenige Veränderungen der Aufgabenstellung manchmal ausreichen, aus einem einfachen Problem eines zu machen, das quasi nur noch durch mehr oder weniger geschicktes Ausprobieren lösbar ist.

Literatur

Robert Sedgewick „**Bundle of Algorithms in Java, Parts 1–5**“; 3rd Edition 2003; Addison-Wesley; ISBN 0-2017-7578-6



Da Robert Sedgewick für viele Probleme dieses Buches einen Algorithmus beschreibt und in Java implementiert, taucht er auch oft in den Literaturlisten am Ende der Kapitel auf. Das Rucksackproblem wird in Teil 1–4 des Buches behandelt, die Referenz finden Sie bereits am Ende von Kapitel 2. Hier habe ich die ISBN der englischen Version angegeben, die gleich beide Bände enthält – wenn Sie der englischen Sprache mächtig sind, sollten Sie diese Ausgabe in Erwägung ziehen.



6. Paketpost

Das Internet ist heute allgegenwärtig. Innerhalb von 10 Jahren wurde aus einem weitgehend wissenschaftlich genutzten Kommunikationsmittel ein Medium, das gleichberechtigt neben Fernsehen und Rundfunk steht.

Aber wissen Sie eigentlich, was dieses Gefüge aus Rechnern zusammenhält? Wie wird sichergestellt, dass eine Nachricht korrekt am Bestimmungsort ankommt – trotz einiger Millionen Kilometer Leitung und hunderttausender Vermittlungscomputer, die teilweise verwirrend miteinander vernetzt sind? Kommen Sie mit auf eine Erkundungstour!

Zu diesem Thema selbst Experimente durchzuführen, würde recht aufwendig. Ich möchte daher versuchen, es anhand einer Analogie anschaulich zu machen. Dafür möchte ich Sie in eine Zeit entführen, in der Telekommunikation noch aus dem Senden handschriftlicher Botschaften bestand:

Willkommen am Hof des Königs Informatix. Er mag seinen Hofstaat gerne gut organisiert und das erfordert wiederum eine hervorragende Kommunikation der Untertanen miteinander.

In kleinen Einheiten funktioniert das sowieso sehr gut – sozusagen auf Zuruf. Hören wir daher einmal kurz in ein typisches Gespräch des Hofküchenpersonals hinein. Küchenchef Tom führt dort mit seinen vier Gehilfen Adam, Bert, Chris und Dieter das Regiment.

Adam: „Chris, sollen die Mohrrüben für deinen Auflauf in Scheiben oder in Würfel geschnitten werden?“

Chris: „Weiß nicht. Hallo Meister Tom – ist es relevant, wie die Mohrrüben für den Karottenauflauf geschnitten werden?“

Tom: „Nein Chris, aber es sollte einheitlich sein!“

Tom: „Alle mal herhören – hat schon jemand Mohrrüben geschnitten?“

Dieter: „Ja, Tom, ich habe sie in Scheiben geschnitten!“

Tom: „Chris, wir sollten einheitlich Scheiben nehmen.“

Chris: „Adam, also Scheiben bitte.“

Adam: „Okay, Chris.“

Können Sie sich nach diesem Beispiel bereits denken, wie die Gespräche in der Küche allgemein ablaufen?

- Wer spricht zu wem?
- Wer kann zuhören?
- Wer hört tatsächlich zu?



Gesprochen wird abwechselnd. Jeder Sprecher wendet sich dabei eindeutig an:

- eine bestimmte Person („Chris, wir sollten ...“)
- oder alle Personen („Alle mal herhören – ...“)

Zuhören könnte prinzipiell jeder im Raum, allerdings sind nicht alle Gespräche für jeden interessant und so wird normalerweise nur der Angesprochene tatsächlich aufmerksam.

Genauso wie die Personen in der Hofküche können Sie sich das interne Computernetzwerk einer kleinen Firma oder zwischen den Rechnern eines Haushaltes vorstellen: Jeder Computer kann abwechselnd etwas sagen, also eine Nachricht senden. Alle anderen Computer können diese empfangen und darauf reagieren. In Abbildung 6.1 sehen Sie entsprechend das Netzwerk der „Küchenmeister GmbH“. Im Gegensatz zu Menschen bevorzugen Computer allerdings Zahlen statt Namen.

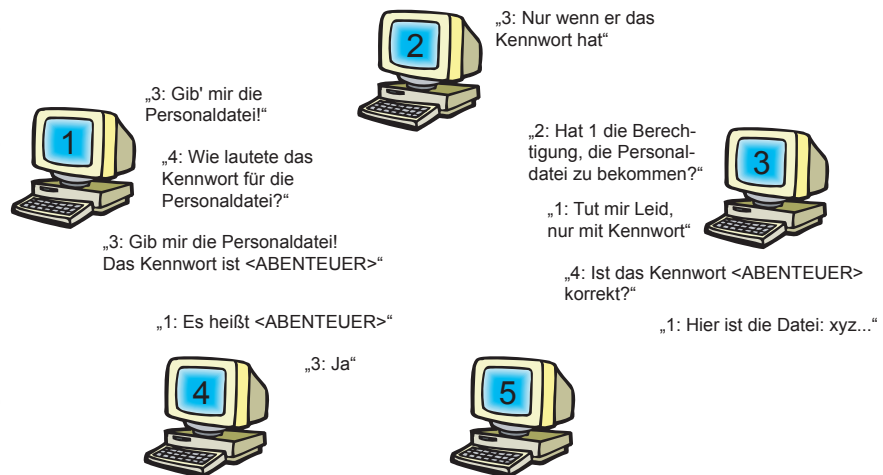


Abbildung 6.1 Gesprächige Computer

Versuchen Sie, den Sinn des Gesprächs der Computer aus den Gesprächsstücken herauszubekommen!



Offenbar möchte Nummer 1 von Nummer 3 die Personaldatei. Nummer 2 verwaltet die Zugriffsberechtigungen, Nummer 4 verwaltet die Kennworte.

Da Nummer 1 erst fragt, ohne das Kennwort zu nennen, wird ihm die Auskunft verweigert. Erst nachdem er Nummer 4 nach dem Kennwort fragt und dieses bei seiner Anfrage mitliefert, bekommt er von Nummer 3 die gewünschten Daten.

Zurück an den Hof: Wenn ein neuer Lehrling in die Küche kommt, stellt das kein Problem dar. Er stellt sich vor, zum Beispiel als „Emil“ und wird daraufhin in alle Gespräche eingebunden. Die Kommunikation funktioniert wie zuvor tadellos.

Was passiert aber nun, wenn König Informatix eine neue Rationalisierungsmaßnahme ausprobieren möchte: In der Hofküche ist viel ungenutzter Platz, daher soll die Hofbäckerei ebenfalls dort einziehen. Die alte Bäckerstube kann Informatix dann zur Unterbringung seiner Sammlung alter Rechenapparate nutzen. Bäckermeister Ulf wirkt mit seinen sieben Gesellen nun ebenfalls in der Hofküche, so dass nun 14 Menschen miteinander kommunizieren. Lauschen wir kurz hinein:

Gerd: „Inge, für den Kuchen benötigen wir einen dunklen Zuckerguss.“

Tom: „Dieter, die Schokolade gleichmäßig umrühren.“

Hans: „Gerd, ist der Ofen angeheizt?“

Tom: „Gerd, ich brauche ein paar Plätzchen als Verzierung für die Schoko-Speise!“

Gerd: „Ja Tom!“

Inge: „Okay, Gerd, kann ich anrühren!“

Gerd: „Nein, Hans.“

Sie sehen bereits an diesem sehr kleinen Beispiel, dass es schwierig wird, den Gesprächen zu folgen. Überlegen Sie, was passiert, wenn sich eine ganze Halle voller Menschen durch ständige Zurufe über ihre Arbeit austauschen muss. Wenn jeder nur selten etwas von jemand anderen möchte, funktioniert die Sache noch. Muss aber viel kommuniziert werden, wird die gegenseitige Störung so groß, dass man eventuell gar nicht mehr oder nur sehr verzögert zu Wort kommt. Die Leistungsfähigkeit des gesamten Betriebes leidet dann darunter. So passiert das auch am Hof von Informatix – seit der Zusammenlegung der Küche mit der Bäckerei muss der König den Gürtel deutlich enger schnallen, weil viel weniger produziert wird. Davon ist er natürlich alles andere als begeistert. Essen ist seiner Meinung nach die wichtigste Tätigkeit des Tages.

Helfen Sie dem hungrigen König! Wie könnte er – trotz Rationalisierungszwang – die Produktivität wieder steigern?



Die Analyse der Situation ist eindeutig: Zu viele Leute sind am Gespräch beteiligt. Die Gesprächskreise stören sich gegenseitig, weil sie das gleiche „Medium“, in diesem Fall denselben Raum, benutzen.

Lösung ist eine schalldämmende Trennwand! Die Küche wird aufgeteilt. Nun können beide Arbeitsgruppen wieder ungestört miteinander reden.

Allerdings ist das noch nicht die ganze Lösung! Wenn Sie den Dialog oben genau angeschaut haben, ist Ihnen sicher aufgefallen, dass zum Beispiel Tom und Gerd miteinander geredet haben – sie sind in unterschiedlichen Gruppen. Einen Vorteil hatte die Zusammenlegung nämlich schon: Köche und Bäcker konnten ihre Aufträge koordinieren und sich gegenseitig zuarbeiten. Die hierfür notwendige Kommunikation ist nun aber durch die Zwischenwand blockiert.

Eine Lösung ist, in die Wand ein kleines Fenster einzubauen, durch das sich gerade zwei Personen unterhalten können. Je ein Geselle aus den Gruppen wird dann zum „Kommunikator“ ernannt. Die Kommunikatoren sitzen vor dem Fenster und können sich durch die Wand unterhalten, ohne dass die Gespräche in den jeweiligen Räumen beeinträchtigt werden. Abbildung 6.2 zeigt das schematisch. In den vorangegangenen Kapiteln haben wir sehr ausführlich über Abstraktion gesprochen. Daher werde ich im Folgenden auch abstrakte Darstellungen verwenden, um die Sachverhalte möglichst klar darzustellen.

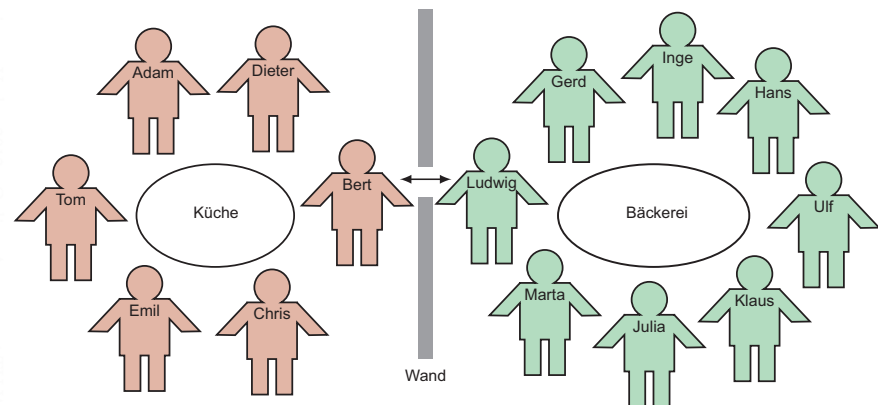


Abbildung 6.2 Wieder in zwei Werkstätten ...

Die Hofangestellten können nun wieder uneingeschränkt in ihren Gruppen miteinander reden. Um mit jemandem aus der anderen Gruppe zu sprechen, müssen sie über Bert und Ludwig gehen, die den Auftrag haben, die Verbindung zu halten.

Überlegen Sie nun, wie die letzte Kommunikation unter den neuen räumlichen Gegebenheiten abgelaufen wäre. Am besten notieren Sie es kurz.



Im Dialog sind die veränderten Passagen rot markiert. Es hat sich selbstverständlich nur dort etwas verändert, wo die Kommunikation nun zwischen zwei Räumen stattfindet.

Gerd: „Inge, für den Kuchen benötigen wir einen dunklen Zuckerguss.“

Tom: „Dieter, die Schokolade gleichmäßig umrühren.“

Hans: „Gerd, ist der Ofen angeheizt?“

Tom: „Bert, sag’ Gerd, ich brauche ein paar Plätzchen als Verzierung für die Schoko-Speise!“

Bert: „Ludwig, sag’ Gerd, Tom benötigt ein paar Plätzchen als Verzierung für eine Schoko-Speise!“

Ludwig: „Gerd, Tom benötigt ein paar Plätzchen als Verzierung für eine Schoko-Speise!“

Gerd: „Ludwig, sag’ Tom: Ja!“

Ludwig: „Bert, sag’ Tom von Gerd: Ja!“

Bert: „Tom, Gerd sagt: Ja!“

Inge: „Okay, Gerd, kann ich anrühren!“

Gerd: „Nein, Hans.“

Die Entlastung der einzelnen Räume hat also ihren Preis: Möchte man mit jemandem sprechen, der nicht mit im Raum sitzt, geht das immer über mehrere Ecken. In unserem Fall wird eine einfache Nachricht verdreifacht – sie muss jeweils von Bert und von Ludwig wiederholt werden, um den Adressaten zu erreichen.

Genau die gleichen Effekte kann man auch in Computernetzwerken beobachten: Statt einfacher Gespräche werden kurze Datenblöcke verschickt, die so genannten **Datenpakete**. Das Versenden eines Datenpakets blockiert das Netzwerk, genau wie die Kommunikation durch Worte andere Gespräche im Raum unterbricht.

Bei den ersten Computernetzwerken konnten prinzipiell alle angeschlossenen Rechner gleichzeitig miteinander reden, was sehr schnell zur Überlastung führte. Weiterer Malus war die Tatsache, dass prinzipiell jeder Teilnehmer alle Gespräche innerhalb eines Netzes belauschen konnte.

Oft waren die Programme auch noch sehr „gesprächig“. So fragten einige in kurzen Abständen das gesamte Netzwerk „Hallo, mein Name ist Hugo, wer ist denn noch so alles angeschlossen?“, woraufhin alle anderen Computer im Netz eine Antwort schickten. Auf diese Weise kam es häufig vor, dass ein Netzwerk mit zum Beispiel 20 Computern tadellos lief, schloss man jedoch einen einzigen weiteren Rechner an, ging gar nichts mehr: Die Leitungen waren komplett mit „Hallo“-Meldungen wie der obigen belegt, so dass sinnvolle Anfragen keine Chance hatten.

Die Lösung ist prinzipiell so einfach wie in Informatix' Hofküche: Kleine Computernetzwerke, in denen direkt jeder mit jedem anderen sprechen kann, dürfen eine gewisse Anzahl von Teilnehmern nicht überschreiten. Man spricht hier übrigens auch von **LAN oder Local Area Network**: Das sind die Computer, die in einer kleinen Firma oder eine Abteilung eng miteinander verbunden sind.

Ein **WAN oder Wide Area Network** verbindet mehrere LANs miteinander. In der Hofküche ist der Job von Bert und Ludwig, das WAN aufrechtzuerhalten.

Effektiv gibt es heute im Internet sehr viele Hierarchiestufen zwischen dem LAN, das Ihren Rechner vielleicht mit Ihrem Drucker verbindet und dem WAN, das die ganz großen Internet-Anbieter miteinander verbindet.

Zurück zum Hofstaat: König Informatix ist so begeistert von den Möglichkeiten, die die Trennwand mit Loch bietet und überträgt das sofort auf seinen gesamten Hofstaat: Nicht nur Bäcker und Köche sollen miteinander reden können, sondern alle anderen auch. Es würde sicher nicht funktionieren, alle in einen einzigen Raum mit Trennwänden zu setzen, daher wird pro Arbeitsgruppe ein Bote bestimmt, der als Kommunikator den Kontakt zu den anderen Kommunikatoren hält, indem er Nachrichten überbringt. Abbildung 6.3 zeigt, wie die Gruppen über ihre Kontaktpersonen Gunter, Olga, Bert, Ludwig, Lina und Rolf Nachrichten austauschen können – diese bilden quasi ein eigenes Netzwerk auf höherer Ebene.

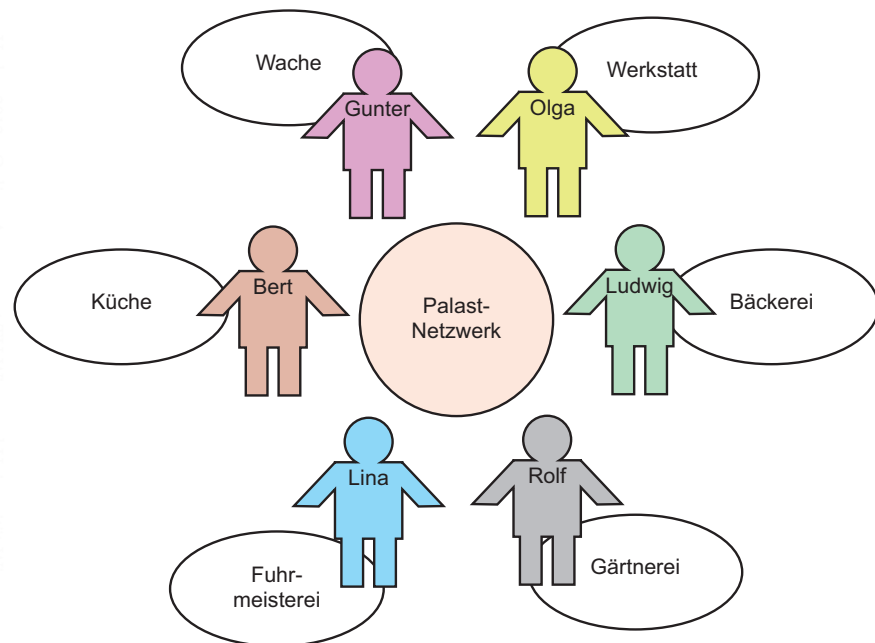


Abbildung 6.3 Alle Palastangestellten tauschen sich über ihre Kontaktpersonen aus

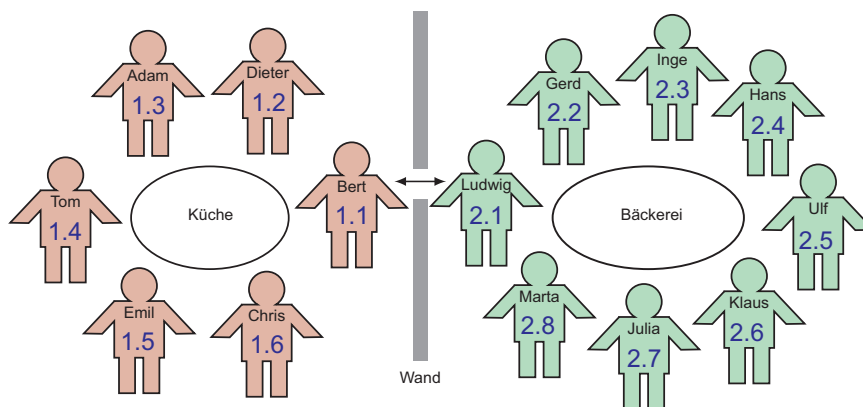
Stellen Sie sich vor, es soll nun auch Kontakt zu den Angestellten in den Palästen der benachbarten Könige Technokratix und Computix gehalten werden. Hierfür würden wiederum Kommunikatoren bestimmt, die die Verbindung herstellen. In diesem Netzwerk könnte dann jemand für die Kommunikation mit dem Kaiserpalast abgestellt werden usw. Auf diese Weise kann ein riesiges Netzwerk aufgebaut werden. Allerdings stoßen unsere Boten im Beispiel schnell auf ein Problem, das auch die Post in den Anfängen hatte: Auch wenn die hier verwendeten Vornamen eindeutig sind, kann man das für reale Verhältnisse kaum annehmen.

Selbst wenn man dies gewährleisten könnte, würde es für alle Beteiligten sehr verwirrend. So müsste sich jeder Beteiligte immer merken, mit welchen Gesprächspartnern er direkt reden kann und für welche er die Kontaktperson ansprache.

Die Post hat dieses Dilemma durch die Einführung von Zahlen als eindeutiges Identifikationsmerkmal gelöst – zumindest in Teilbereichen der Adresse: Postleitzahl und Hausnummer.

Es wäre nun möglich, einfach jedem möglichen Gesprächspartner zu seinem Namen eine beliebige Nummer zu verpassen. Dann hätte man allerdings die Vorteile der Zahlen nicht ausgenutzt: Postleitzahlen oder zum Beispiel auch Telefonnummern folgen zusätzlich auch einem Schema. Die ersten Ziffern grenzen die Region ein und erst die letzten Ziffern bestimmen den Ort bzw. die Person genauer.

Ein solches Verfahren kann sich auch König Informatix zunutze machen. Das kommt seinem Ordnungsdrang sehr entgegen. Schauen wir uns hierfür nochmals den kleinen Gesprächskreis in der Palastküche an. Jeder Angestellte bekommt eine Nummer zugewiesen. Dabei ist es sinnvoll, den Kommunikatoren eine einheitliche Nummer – zum Beispiel die 1 – zu geben. Eine vorangestellte weitere Nummer bestimmt dann, in welchem Küchenteil sich der Teilnehmer befindet – 1 für die eigentliche Küche und 2 für die Bäckerei. Sie sehen das in Abbildung 6.4. Auf diese Weise könnten auch die anderen Bereiche des Palastes ihre eigene Nummer erhalten: 3 für die Werkstatt, 4 für die Wache usw.



Übrigens: Die Postleitzahl gibt es für die Öffentlichkeit in Deutschland seit 1944, als die Bevölkerung aufgefordert wurde, zwecks schnellerer Beförderung die Nummer der Oberpostdirektion der Adresse anzugeben, zum Beispiel 5b für Oberpreußen.

Die Hausnummer wurde bereits viel früher „erfunden“: Kaiserin Maria Theresia verfügte sie 1770 zu Zwecken der leichteren Volkszählung in Wien.

Abbildung 6.4 Jeder Gesprächspartner hat nun eine Nummer

Wie helfen aber die Nummern bei der Kommunikation? Nehmen wir einmal an, Inge möchte mit Maria reden. Inge hat die Nummer 2.3, Maria die 2.8. Inge erkennt, dass Maria sozusagen die gleiche „Vorwahl“ hat wie sie, daher kann sie ihr Anliegen einfach in den Raum rufen, Maria hört sie.

Wenn Inge mit Dieter sprechen möchte, ist das etwas anderes: Dieter hat die Nummer 1.2, befindet sich also in einem anderen Gesprächsnetz. Daher sagt sie ihr Anliegen ihrem Kommunikator, der 2.1. Dieser weiß, dass der zuständige Kommunikator für Nachrichten an die 1.1 Bert ist und gibt die Nachricht an ihn weiter. Dieser leitet sie schließlich an Dieter mit der 1.2.

Auf die gleiche Weise funktioniert das Internet. Jeder Computer ist eindeutig anhand einer Zahlenkombination identifizierbar: Es handelt sich um vier Nummern zwischen 0 und 255, die meistens mit Punkt getrennt dargestellt sind, zum Beispiel 130.83.242.159. Die ganze Kombination nennt man IP-Nummer oder IP-Adresse.

Diese Art der Adressierung aus verschiedenen, hierarchisch geordneten Zahlen kennen Sie übrigens auch vom täglichen Gebrauch: Eine vollständige Telefonnummer besteht auch aus einer Landesvorwahl (49 für Deutschland), einer Ortsvorwahl (z. B. 6151 für Darmstadt) und einer Anschlussnummer (z. B. 123456). Die gesamte Nummer ist dann +49 6151 123456.

IP

Internet-Protokoll. Dieser Standard ist bereits im September 1981 bekanntgegeben worden. Ursprünglich für die Forschungsbehörde im amerikanischen Pentagon entwickelt, ist IP bis heute die Grundlage der Kommunikation zwischen Computern im Internet. Die 1998 veröffentlichte Weiterentwicklung IPv6 konnte sich trotz deutlich größerer Möglichkeiten bis heute nicht durchsetzen.

Genau wie in unserem Beispiel haben alle Rechner in einem kleinen Netzwerk, z. B. einer Firma, dieselben vorderen Zahlen als IP-Nummer. Sie unterscheiden sich lediglich in der hinteren Zahl. Die vorderen Zahlen bestimmen dann, um welches Netzwerk es sich handelt. Die Rolle der Kommunikatoren im obigen Beispiel übernehmen im Computernetz die so genannten Router. Es sind spezialisierte Rechner, die Datenpakete im Internet in die richtige Richtung weiterleiten.

Abbildung 6.5 zeigt schematisch einen Ausschnitt des Internet. Die Geschäftsführer der Informatix AG haben sich die Ordnung des alten Königs zunutze gemacht und sie auf ihre heutige Infrastruktur übertragen.

Beachten Sie, wie die Router (in der Skizze als flache Kästchen dargestellt) zwei oder mehr IP-Adressen besitzen, weil sie ja auch in mehreren Computernetzen „sprechen“ müssen.

Im Informatix-Beispiel wären das etwa Bert oder Ludwig, die einerseits innerhalb ihrer Gruppe kommunizieren und dafür eine Nummer aus dem entsprechenden Bereich besitzen. Andererseits sprechen sie jedoch auch zusätzlich noch mit den anderen Kommunikatoren und haben hierfür eine weitere Adresse.

Übertragen Sie Ihr Wissen der Kommunikationsstruktur in Informatix altem Königreich auf das Internet: Setzen Sie sich an den Rechner mit der IP-Adresse 10.43.3.32 in der Abteilung für Küchengeräte. Wie laufen die Datenpakete zu den Computern 10.43.3.19, 192.168.0.9 und 10.43.4.2? Schildern Sie genau, welche Nachrichten die Computer bzw. Router untereinander austauschen!



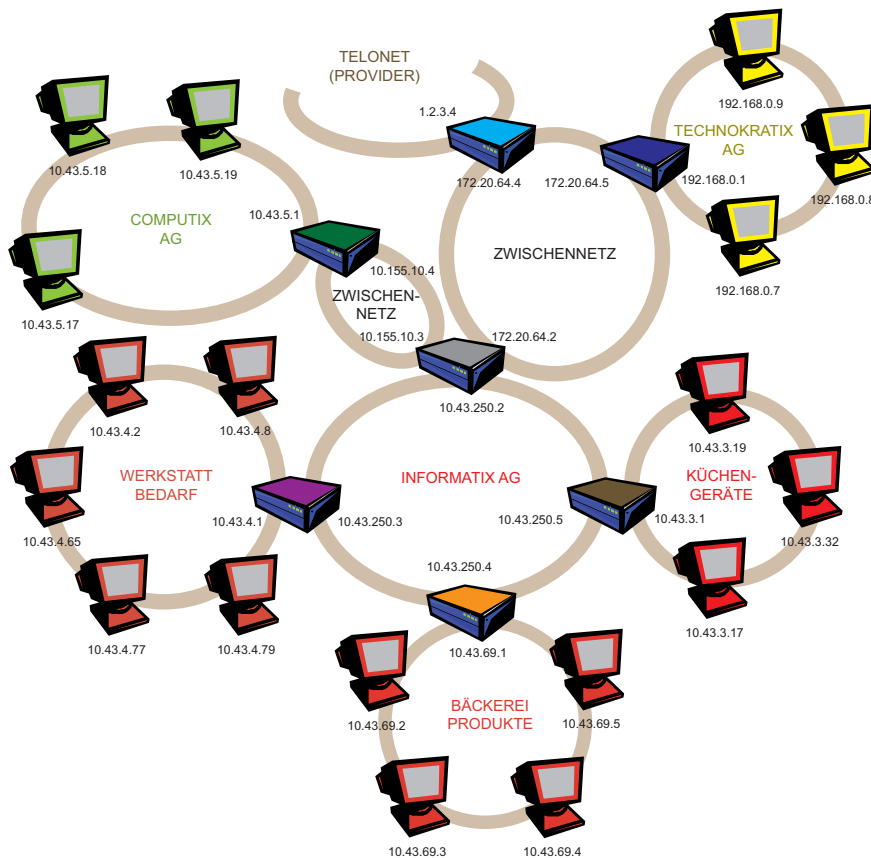


Abbildung 6.5 Internetausschnitt mit der Informatix AG

Fall 1: Ausgangs-Adresse: 10.43.3.32, Ziel-Adresse: 10.43.3.19

Der Computer erkennt, dass die ersten drei Zahlen von Ausgangs- und Zieladresse identisch sind und schickt die Nachricht direkt an die Ziel-Adresse, fertig:

Von: 10.43.3.32 – An: 10.43.3.19 – Nachricht

Fall 2: Ausgangs-Adresse: 10.43.3.32, Ziel-Adresse: 192.168.0.9

Hier sind die ersten Zahlen komplett unterschiedlich, der Computer schickt die Nachricht also an seinen zugeordneten Router, der sie dann immer weiter zum Ziel schickt:

Von: 10.43.3.32 – An: 10.43.3.1 – Nachricht von 10.43.3.32 für 192.168.0.9

Von: 10.43.250.5 – An: 10.43.250.2 – Nachricht von 10.43.3.32 für 192.168.0.9

Von: 172.20.64.2 – An: 172.20.64.5 – Nachricht von 10.43.3.32 für 192.168.0.9

Von: 192.168.0.1 – An: 192.168.0.9 – Nachricht von 10.43.3.32 für 192.168.0.9

An diesem Beispiel sehen Sie, dass die eigentliche Nachricht nochmals Sender und Empfänger beinhalten muss. Ansonsten wüssten die Router nicht, an wen sie diese weiterleiten sollen. Der Empfänger kann auch nur so feststellen, wer der Absender eigentlich war, denn direkt empfängt er die Nachricht ja von seinem Router 192.168.0.1.

Fall 3: Ausgangs-Adresse: 10.43.3.32, Ziel-Adresse: 10.43.4.2

Dieser Fall funktioniert genau wie Fall 2.

Von: 10.43.3.32 – An: 10.43.3.1 – Nachricht von 10.43.3.32 für 10.43.4.2

Von: 10.43.250.5 – An: 10.43.250.3 – Nachricht von 10.43.3.32 für 10.43.4.2

Von: 10.43.4.1 – An: 10.43.4.2 – Nachricht von 10.43.3.32 für 10.43.4.2

Soweit ist die Sache sehr einfach: Die Nachricht wird so lange von Netzwerk zu Netzwerk geschickt, bis sie am Ziel ankommt. Anhand des Planes können Sie auch genau bestimmen, welche Stationen hierbei passiert werden müssen.

Allerdings besitzen die beteiligten Computer keinen solchen Plan und das ist auch gar nicht notwendig. Überlegen Sie daher, welche Informationen die einzelnen Computer (nicht die Router) benötigen, um am Internet teilzunehmen. Welche anderen Netzwerkadressen müssen sie kennen?



Tatsächlich muss natürlich jeder seine eigene IP-Adresse kennen. Hinzu kommt die Information, dass sich alle Computer mit gleichen ersten drei Zahlen in der IP-Adresse im selben Netzwerk befinden und man sie direkt ansprechen kann.

Zusätzlich ist nur noch die Adresse des zugeordneten Routers notwendig. An diesen werden alle anderen Nachrichten geschickt! Auch die normalen Bediensteten des König Informatix brauchten nur zu wissen, wem sie eine Nachricht geben mussten, wenn diese für jemanden außerhalb des Zimmers bestimmt war. Den Rest erledigten dann die Kommunikatoren. Im Internet sind das die Router.

Für die einzelnen Computer ist die Welt also einfach – die eigentliche Vermittlungsarbeit wird von den Routern erledigt. Diese besitzen Tabellen, in denen steht, wo sie eine Nachricht hinschicken müssen, damit diese näher zum Ziel kommt. So eine Tabelle kann sehr einfach sein.

Betrachten wir zum Beispiel die Tabelle für den grünen Router der Computix AG.

Nachricht von	Nachricht an	Schicke an
10.43.5.*	Irgendjemand	10.155.10.3
Irgendjemand	10.43.5.*	Zieladresse

Der zugeordnete Router ...
... heißt bei heutigen Computersystemen übrigens auch „Standardgateway“ oder einfach „Gateway“.

Die Tabelle ist so einfach, weil der Router nur zwei Netze miteinander verbindet. Alles, was von drinnen kommt, muss an den nächsten Router weitergeleitet werden. Das besagt die erste Regel. Das Sternchen ist hierbei ein Platzhalter, der so viel wie „irgendeine Zahl“ bedeutet.

Die zweite Regel legt dann fest, dass Pakete an ein bekanntes Ziel (nämlich eines aus dem inneren Netzwerk der Computix AG) direkt dorthin zugestellt werden. Mehr Regeln braucht der grüne Router nicht!

Vielleicht wundern Sie sich, warum nicht alle möglichen Fälle abgedeckt sind. Was macht der Router mit einer Nachricht von 192.168.0.7 an 10.43.69.2?



Diese Nachricht würde in keiner sinnvollen Konstellation beim grünen Router vorbeikommen! Entweder es handelt sich um einen Irrläufer oder ein Computer aus dem Netz der Computix AG hat einen falschen Absender angegeben (zum Beispiel um Zugang zu einer Webseite zu bekommen, die nur für Computer der Technokratix AG freigegeben ist). In beiden Fällen gibt es nur eine richtige Reaktion: Er ignoriert die Nachricht einfach.

Überlegen Sie nun, welche Tabelle der braune Router besitzen muss.



Diese Tabelle ist bereits etwas komplizierter. Der Router kennt drei weitere Router, an die er Nachrichten schicken kann. Für jeden dieser Router gibt es eine Regel.

Nachricht von	Nachricht an	Schicke an
10.43.3.*	10.43.69.*	10.43.250.4
10.43.3.*	10.43.4.*	10.43.250.3
10.43.3.*	Irgendjemand anderen	10.43.250.2
Irgendjemand	10.43.3.*	Zieladresse

Beachten Sie die dritte Zeile: Hier steht beim Empfänger „Irgendjemand anderen“, weil diese Regel nur greifen soll, wenn die Nachricht an jemanden geht, der nicht in den genauer gefassten Regeln der ersten beiden Zeilen enthalten ist.

So etwas kommt bei Computernetzen häufig vor: Man macht spezielle Regeln und allgemeine Regeln. Es wird dann die für den gegebenen Fall speziellste Regel angewendet.

Kleine Welt

Das Kleine-Welt-Phänomen ist 1967 vom Soziologen Stanley Milgram formuliert worden. Eigentlich sagt es etwas über die große Welt aus: Wenn man die Bekanntschaftsbeziehungen jedes Menschen aufzeichnet, dann sind überraschenderweise fast alle Menschen der Welt über sehr wenige „Umwege“ miteinander bekannt. Wir leben also sozusagen doch in einer kleinen Welt! Das Gleiche gilt auch für das Internet: Obwohl jeder Computer und Netzverbindungsknoten nur wenige andere kennt, sind alle über sehr wenige Strecken oder „Hops“ erreichbar. Das bringt nicht immer nur Vorteile mit sich: Zum Beispiel haben Computerviren sehr kurze „Übertragungswege“.

Steigern wir unseren Regulierungsdrang weiter: Stellen Sie nun die Tabelle für den grauen Router in der Mitte auf! Sie können davon ausgehen, dass alle Nachrichten für Computer außerhalb der Technokratix AG, der Computix AG und der Informatix AG über den Provider Telonet verschickt werden.



Hier ist die Sache erneut etwas komplizierter. Der Router hat sogar drei Anschlüsse in verschiedenen Netzen und in seinem Bereich steht auch der blaue Router, der die Verbindung mit allen nicht eingezeichneten Computern im Internet herstellen kann.

Nachricht von	Nachricht an	Schicke an
Irgendjemand	10.43.5.*	10.155.10.4
Irgendjemand	10.43.3.*	10.43.250.5
Irgendjemand	10.43.4.*	10.43.250.3
Irgendjemand	10.43.69.*	10.43.250.4
Irgendjemand	192.168.0.*	172.20.64.5
10.43.5.*	Irgendjemand anderen	172.20.64.4
10.43.4.*	Irgendjemand anderen	172.20.64.4
10.43.3.*	Irgendjemand anderen	172.20.64.4
10.43.69.*	Irgendjemand anderen	172.20.64.4

An diesen Beispielen sehen Sie, dass das Internet prinzipiell durch das Zusammenspiel sehr vieler Router funktioniert. Diese wiederum vermitteln Nachrichten aufgrund fester Routing-Tabellen. Solche Tabellen können kleiner oder größer ausfallen, da die Router jedoch immer nur einen kleinen Ausschnitt des Internet abdecken, bleiben diese übersichtlich.

Erkennen Sie, dass auch das Internet „typisch informatisch“ aufgebaut ist? Das Prinzip „Divide et impera“, das dieses Buch schon über mehrere Kapitel durchzieht, können Sie auch hier ganz deutlich erkennen: Das Internet als Ganzes ist unübersichtlich und kaum zu durchschauen. Es besteht jedoch aus einzelnen Komponenten, die jeweils nur einen kleinen Teil des Internet selbst kennen. Die Funktionsweise jeder dieser Komponenten ist recht leicht zu verstehen. Nur auf diese Weise kann ein solch riesiges Netz funktionieren!

Ach wie gut, dass jeder weiß ...

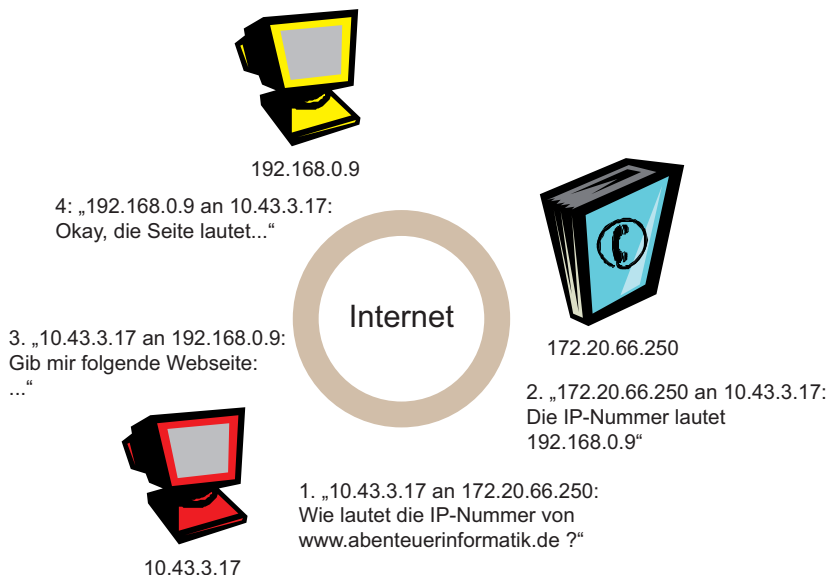
Rumpelstilzchen möchte im Märchen seinen Namen um jeden Preis verbergen. Genau das Gegenteil trifft auf die Computer im Internet zu – fast jeder Internet-Teilnehmer hat heute zusätzlich zur IP-Nummer noch einen IP-Namen.

Wofür braucht man diesen? Die Computer sind anhand ihrer Nummer eindeutig identifizierbar und außerdem arbeiten Computer doch sehr gerne mit Zahlen!

Stimmt! Tatsächlich kommunizieren die Rechner miteinander auch ausschließlich auf Basis der IP-Nummern. Ein Problem haben allerdings die Benutzer: Könnten Sie sich vorstellen, im Internet zu surfen und dabei ausschließlich mit IP-Nummern zu arbeiten? Wenn Sie auf die Seiten dieses Verlags schauen möchten, rufen Sie nicht <http://www.elsevier.de/> auf, sondern <http://194.175.173.45/>. Stellen Sie sich die riesigen Telefonbücher vor, in denen die IP-Nummern aller Computer im Internet verzeichnet sind.

Glücklicherweise können uns Computer aber Arbeit abnehmen und so hat man 1983 den so genannten „Domain Name Service (DNS)“ erfunden, übersetzt etwa „Namensbereichsdienst“. Spezielle Computer im Internet – die DNS-Server (oder einfach Nameserver) – übernehmen die Rolle des Telefonbuchs. Wenn Sie einen Namen in der Adresszeile Ihres Internet-Browsers eintippen, kann damit der Computer erst einmal gar nichts anfangen. Er benötigt zur Kommunikation mit diesem Rechner unbedingt die entsprechende IP-Nummer.

Er muss diese daher in Erfahrung bringen und fragt einen DNS-Server danach. Von diesem muss er allerdings die IP-Nummer kennen, sonst könnte er ihn ja auch nicht erreichen. Abbildung 6.6 zeigt die Kommunikation schematisch.



Wie groß ist das Internet?

Diese Frage beschäftigte schon mehrere Studien. Allerdings kann man sie ganz verschieden interpretieren: Wie viele Computer sind angeschlossen? Wie viele Computer haben eine eigene Adresse? Wie viele Computer sind „Server“, bieten also irgendwelche Dienste rund um die Uhr an? Wie viele Webseiten gibt es?

Die Grafik unten zeigt, wie viele Computer in den unterschiedlichen Jahren einen Namen hatten, der fest einer Adresse zugewiesen ist. Da nahezu jeder Computer, der zeitweise am Internet teilnehmen möchte, einen Namen bekommt, ist das ungefähr die Menge von Computern, die prinzipiell (aber nicht dauerhaft) am Internet angeschlossen sind. Die Skala ist in Millionen Computern eingeteilt.

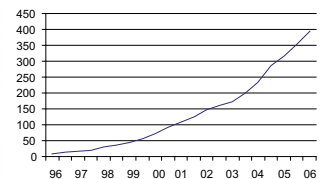


Abbildung 6.6 Kommunikation mit dem DNS-Server

Meilensteine der Internet-Verwaltung

1972 Gründung der IANA und einheitliche Vergabe von IP-Nummern.

1985 Erste IP-Namen werden vergeben.

1992 InterNIC wird gegründet, um die IP-Namen zu verwalten.

1998 Die ICANN wird gegründet, um InterNIC als oberste „Namensbehörde“ abzulösen.

2000 ICANN@large wird etabliert, eine Art Mitbestimmungsgremium: Zum ersten Mal kann jeder Internet-Benutzer Vorstandsmitglieder der ICANN wählen.

Im Beispiel möchte der Benutzer des roten Computers die Internetseite von „www.abenteuerinformatik.de“ aufrufen. Der Computer kennt die IP-Nummer des Zielrechners nicht und fragt beim Nameserver (in der Grafik blau) nach. Dieser antwortet mit der korrekten Nummer, woraufhin die eigentliche Frage nach der Webseite gestellt werden kann.

Wenn Sie also beim Recherchieren im World-Wide Web eine Bezeichnung wie „www.irgendetwas.de“ benutzen, ist dies immer der Name eines Computers, nicht etwa die Bezeichnung für ein spezielles Dokument!

Jetzt wissen Sie auch, warum Ihr Computer die IP-Nummer eines Nameservers kennen muss, damit Sie sinnvoll im Internet arbeiten können!

Obwohl dieses Buch eher Verfahren der Informatik beschreibt, möchte ich der Vollständigkeit halber kurz auf die Verteilung von IP-Nummern und IP-Namen eingehen. Die gesamte Kommunikation im Internet beruht darauf, dass eine IP-Nummer bzw. ein IP-Name ganz eindeutig einem einzelnen Computer zugewiesen ist. Nur auf diese Weise können Nachrichten gezielt an den Adressaten zugestellt werden.

Das gleiche Problem gibt es auch bei der normalen Post: Wenn drei Herren mit dem Namen „Peter Müller“ in einem Hochhaus wohnen, weiß der Briefträger nicht, wem er entsprechende Post zustellen soll. Hier muss eine zusätzliche Angabe (z. B. „1. Stock“) für Klarheit sorgen. Während der Briefträger gegebenenfalls nachfragen könnte, würden im Internet die Nachrichten entweder falsch oder gar nicht zugestellt werden.

Eine Organisation namens IANA (Internet Assigned Numbers Authority) vergibt daher IP-Nummern auf Antrag an lokale Behörden, Internet-Provider oder Organisationen. Dabei werden nie einzelne Nummern, sondern immer ganze Blöcke zugeteilt. Diese können dann vom „Besitzer“ weiter unterteilt und vergeben werden.

Normalerweise spricht man von einer Class-A-Adresse, wenn nur die erste Zahl von der IANA festgelegt wird, zum Beispiel ist der Bereich 13.*.* komplett der Firma Xerox zugeteilt. Die Sternchen stehen für beliebige Zahlen. Xerox muss diese selbst verwalten und innerhalb des eigenen Bereichs dafür sorgen, dass es keine Computer mit doppelter IP-Nummer gibt. Im Bereich einer Class-A-Adresse können ungefähr 256^3 , das heißt ca. 16 Millionen Rechner eine individuelle IP-Nummer haben.

Bei der Class-B-Adresse sind die beiden ersten Zahlen festgelegt. So ist 130.83.*.* der Technischen Universität Darmstadt zugewiesen. Innerhalb dieses Bereichs können etwa 256^2 , also ca. 65.000 Computer mit eigener Nummer angeschlossen werden. Es gibt auch noch Class-C-Netze mit drei festgelegten Zahlen. In den Anfängen des Internets war man auf diese drei Bereichseinteilungen festgelegt. Heute können die Nummernbereiche auch individueller eingeteilt und vergeben werden, zum Beispiel ist auch ein Bereich mit nur vier Adressen möglich. Der Einfachheit halber werden wir hier jedoch immer von den klassischen Bereichen ausgehen.

Auch Namen werden zentral zugeteilt. Während die Hierarchie bei den IP-Nummern von vorne nach hinten geht, ist das bei den IP-Namen genau umgekehrt: Von der ICANN (Internet Corporation for Assigned Names and Numbers) wer-

den Domains, also Namensbereiche, an Personen und Organisationen vergeben. Diese können dann innerhalb ihres Bereichs beliebig weitere Namen vergeben. Einzige Einschränkung: Der gesamte Name darf mit Punkten nicht mehr als 255 Zeichen haben.

Beispielsweise ist der gesamte Namensbereich mit Endung „.de“ der deutschen Organisation DENIC (Deutsches Network Information Center) übertragen worden. Möchte man den Bereich „schildbuerger.de“ für sich beanspruchen, kann man dies bei der DENIC anmelden. Diese reservieren dann gegen Gebühr die Domain und Sie können weitere Unterbereiche vergeben, zum Beispiel „verwaltung.schildbuerger.de“ oder „theater.schildbuerger.de“. An erster Position steht immer der Name des Computers, der auch oft die Bezeichnung seiner Funktion enthält, also zum Beispiel „www.verwaltung.schildbuerger.de“ für den Web-Server der Schildaer Verwaltung. Ein Computer kann auch mehrere Namen tragen, zum Beispiel wenn er gleichzeitig Web-Server und Mail-Server ist.

Alle Organisationen, die einen Namensbereich weiter administrieren und für andere Unterbereiche zur Verfügung stellen, betreiben einen DNS-Server oder teilen sich einen DNS-Server mit einem Partner. Dieser enthält dann das Telefonbuch mit der Name-Nummer-Übersetzung für die meisten Computer im Internet. Auf jeden Fall **muss** er die Name-Nummer-Übersetzungen für die Computer im eigenen Namensbereich enthalten. Auf diese Weise können die DNS-Server sich gegenseitig nach Nummer fragen, die sie selbst nicht gespeichert haben. Spätestens der DNS-Server der für den Namensbereich verantwortlichen Organisation kann dann eine korrekte Auskunft erteilen.

Das wird an einem Beispiel am klarsten:

Nehmen wir an, die Kantine der Informatix AG möchte selbst ins Internet und meldet daher den Namen „kantine.informatix.de“ an. Diese neue Domain ist anfangs lediglich dem DNS-Server der Informatix AG selbst bekannt. Was passiert nun, wenn ein Teilnehmer vom Rechner „king.technokratix.com“ aus auf die Seite „www.kantine.informatix.de“ abruft?

1. king.technokratix.com an dns.technokratix.com: Wie lautet die IP-Nummer von www.kantine.informatix.de?
2. dns.technokratix.com an dns.com: Wie lautet die IP-Nummer von www.kantine.informatix.de?
3. dns.com an dns.de: Wie lautet die IP-Nummer von www.kantine.informatix.de?
4. dns.de an dns.informatix.de: Wie lautet die IP-Nummer von www.kantine.informatix.de?
5. dns.informatix.de an dns.de: Die Nummer lautet 10.49.88.17
6. dns.de an dns.com: Die Nummer lautet 10.49.88.17
7. dns.com an dns.technokratix.com: Die Nummer lautet 10.49.88.17
8. dns.technokratix.com an king.technokratix.com: Die Nummer lautet 10.49.88.17

Sie sehen also, dass unter Umständen eine einfache Anfrage nach einer Nummer DNS-Server quer durch die Welt beschäftigen kann. Vielleicht haben Sie auch schon einmal bemerkt, dass der Abruf der allerersten Web-Seite eines bestimmten Servers sehr lange dauert, während dann die folgenden Seiten sehr schnell geladen sind. In diesem Fall wurde die Verzögerung wahrscheinlich durch die langwierige Anfrage beim DNS-Server verursacht.

Was steckt dahinter?

Sie haben bisher in diesem Kapitel kennen gelernt, wie Nachrichten im Internet geroutet werden. Das ist ein für das Verständnis sehr wichtiger Teil, aber nicht das Einzige, was es zur Kommunikation zwischen Computern zu sagen gibt.

Wie bei allen komplexen Systemen muss auch das Internet so zerlegt werden, dass die einzelnen Teile handhabbar bleiben. So konnten wir erkennen, dass sowohl die einzelnen teilnehmenden Computer als auch die Verbindungsglieder – die Router – immer nur einen kleinen Ausschnitt der Vermittlungsarbeit übernehmen und daher auch immer nur einen Ausschnitt des gesamten Wissens um die Verbindungen im Internet benötigen.

Bisher haben wir immer von „Nachrichten“ oder „Datenpaketen“ gesprochen, die Computer austauschen. Damit die Kommunikation jedoch funktioniert, müssen alle Internet-Teilnehmer die gleiche Sprache sprechen. Und hier gibt es wiederum eine Vielzahl technischer Details auf verschiedenen Ebenen: vom Aufbau der WWW-Seiten bis hin zur elektrischen Spannung in den Kabeln, die zwischen einzelnen Komponenten verlaufen.

Auch das kann man in seiner Gesamtheit nicht überblicken und es muss daher aufgeteilt werden. Hier gilt das hierarchische OSI-Modell. Ausgeschrieben bedeutet das „Open Systems Interconnection Reference Model“, also sinngemäß übersetzt etwa „Allgemeines Modell zur Verbindung von Computersystemen“.

Es teilt die Kommunikation hierarchisch in sieben Schichten ein. Um diese zu verstehen, begeben wir uns ein weiteres Mal in das Königreich von Informatix. Seine Tochter Juliana hat sich in den Prinzen Romero des benachbarten Königreichs Technokratien verliebt und tauscht nun heiße Nachrichten mit diesem aus. Das funktioniert weitgehend über die normalen Botendienste des Palastes, da die Familien der Turteltauben kein Problem miteinander haben, im Gegensatz zu den Verwandten eines ähnlich klingenden Paares ...

Normalerweise nutzen die beiden hierfür Tinte und Papier. Die Botschaften werden dann von den jeweiligen Kommunikatoren der Paläste überbracht. Die Paläste befinden sich allerdings in Sichtweite und so kann Juliana nachts von ihrem Fenster aus auch Lichtzeichen mit einer hellen Laterne geben (Abbildung 6.7).

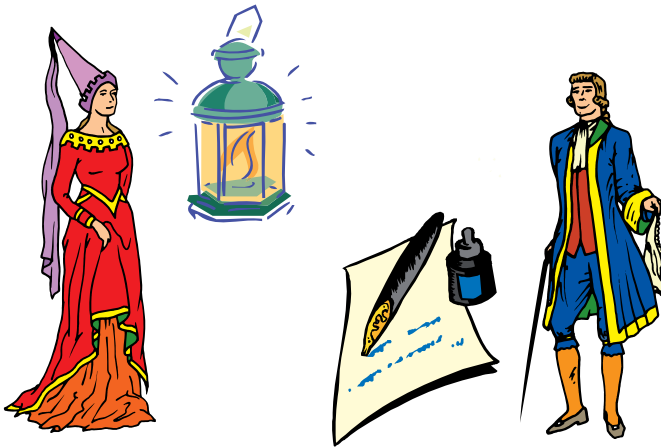


Abbildung 6.7 OSI-Layer 1: Die Physikalische Schicht beschreibt die Signale des Übertragungsmediums

Bei Papier und Tinte oder bei den Lichtzeichen handelt es sich um ein physikalisch vorhandenes Übertragungsmedium. Alle diesbezüglichen Aspekte sind im **OSI-Layer 1 (Physical Layer = Physikalische Schicht)** festgelegt. Bei Computern ist das zum Beispiel die genaue Beschreibung der verwendeten elektrischen Signale im Netzkabel oder die Farbe des verwendeten Lichtes in Glasfaserkabel.

Julianas und Romeros Nachrichten werden von unterschiedliche Boten transportiert, die die Nachrichten schneller oder langsamer weiterleiten. Manchmal passiert es auch, dass ein unzuverlässiger Zusteller im Wirtshaus hängen bleibt und am nächsten Morgen den Brief vergisst. Daher nummerieren die beiden Liebenden ihre Schreiben durch. Auf diese Weise können sie feststellen, wenn einer fehlt. Der andere kann ihn dann nochmals schicken (Abbildung 6.8).

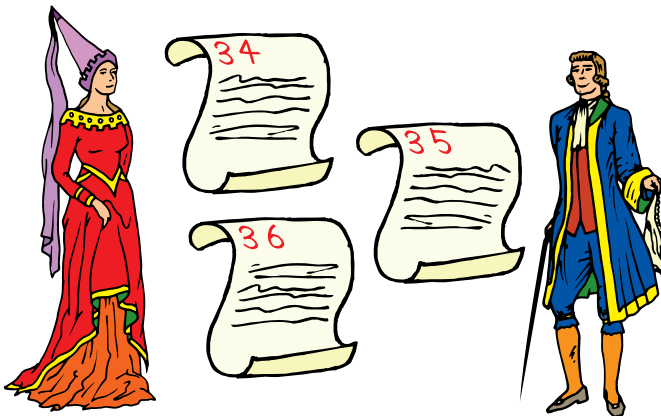


Abbildung 6.8 OSI-Layer 2: Die Datenverbindingsschicht sorgt für die zuverlässige Übertragung einer Nachricht

Die Nummerierung ist eine Art rudimentäre Fehlerkorrektur. Bei der Verbindung zwischen zwei Computern kann ebenfalls vieles schief gehen. Ein Datenpaket kann ganz verloren gehen oder aber auch durch Störungen auf der Leitung verfälscht ankommen. Daher werden zusätzliche Daten wie Nummern, Prüfsummen

usw. eingefügt, mit denen man kontrollieren kann, ob eine Nachricht unverändert angekommen ist. Wenn nicht, wird sie zum Beispiel nochmals angefordert. Wie das genau funktioniert, ist in **OSI-Layer 2 (Data-link Layer = Datenverbindungsschicht)** festgelegt. Auch wird beschrieben, wie die einzelnen Datenpakete prinzipiell aufgebaut sind und wie ein Rechner einen anderen im gleichen LAN anspricht.

Mittelgroße Netzwerke können heute auch allein auf Basis dieser zweiten Schicht effektiv arbeiten und benötigen keinen Router. Ein so genannter Switch verbindet die einzelnen Computer und leitet Datenpakete nur an die korrekten Adressaten weiter. Er stellt dabei meistens automatisch fest, welcher Computer an welchem seiner Anschlüsse hängt, so dass kaum administrativer Aufwand nötig ist.

In unserem Beispiel müssen die beiden Turteltauben natürlich ihre Briefe korrekt adressieren, damit diese ankommen und vom Botendienst richtig zugestellt werden. Der Adressat und der Absender müssen daher auf dem Umschlag stehen (Abbildung 6.9).

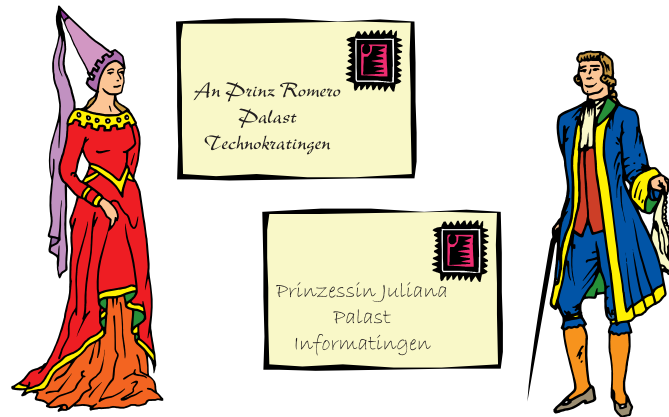


Abbildung 6.9 OSI-Layer 3: Die Netzwerkschicht sorgt für die Vermittlung der Nachrichten im Internet

Die prinzipielle Kommunikation der Computer in einem großen Netz wie dem Internet ist in **OSI-Layer 3 (Network Layer = Netzwerkschicht oder Paketschicht)** geregelt. Die Erklärungen vom Anfang des Kapitels beziehen sich fast alle auf die Netzwerkschicht, da die Funktionsweise von Routern hier beschrieben ist.

Juliana und Romero sind nicht die Einzigen, die in den Palästen Briefe schreiben. Auch die gesamte Verwaltung der Königreiche, vom Steuereintreiber bis zum Küchenchef, beruht auf schriftlicher Kommunikation. Daher hat der Oberpostmeister viel zu tun. Er muss dafür sorgen, dass auch bei hoher Belastung jeder Brief transportiert wird. Wenn alle verfügbaren Boten unterwegs sind, gibt er erst die Nachrichten mit einer hohen Priorität weiter, also etwa persönliche Briefe des Königs oder solche, die mit „Dringend“ beschriftet sind (Abbildung 6.10).

Bei den „echten“ Netzwerken wird dies von **OSI-Layer 4 (Transport Layer = Transportschicht)** erledigt. Hier wird die komplette Vermittlung der Datenpa-

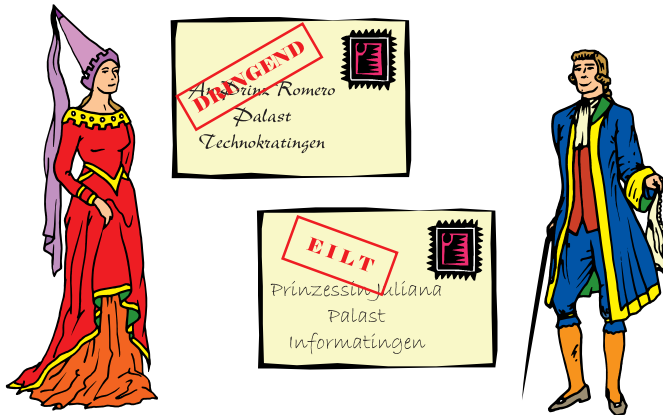


Abbildung 6.10 OSI-Layer 4:
Die Transportschicht regelt den
Verkehr der Datenpakete

kete vom Ursprung zum Ziel geregelt. Auch hier kann man bestimmte „Vorfahrtsregeln“ einplanen. Wenn Sie zum Beispiel mit jemandem über das Netzwerk telefonieren, enthalten die Datenpakete jeweils einen kleinen Teil Ihrer digitalisierten Sprache. Kleine Verzögerungen von Sekundenbruchteilen sorgen dann bereits zu einer Verstümmelung der Sprache. Andere Daten haben wiederum eine niedrigere Priorität: Ob eine WWW-Seite innerhalb einer zehntel oder einer halben Sekunde angezeigt wird, registrieren wir normalerweise gar nicht.

Romero hat sich ein Spiel ausgedacht: Er schreibt den ersten Absatz einer Geschichte und schickt ihn an Juliana, sie verfasst dann den nächsten Absatz und so weiter, bis einer von beiden die Geschichte beendet. Oft dauert es etwas, bis einer von beiden einen neuen Absatz geschrieben hat und daher schreiben beide sich währenddessen auch „normale“ Briefe. Manchmal kommt es sogar vor, dass die beiden zwei oder mehr Geschichten gleichzeitig anfangen. Damit man sofort erkennt, welche Fortsetzungsgeschichte in einem Brief weitergeführt wird, malen Romero und Juliana ein individuelles Symbol für jede ihrer Geschichten auf den Umschlag (Abbildung 6.11).

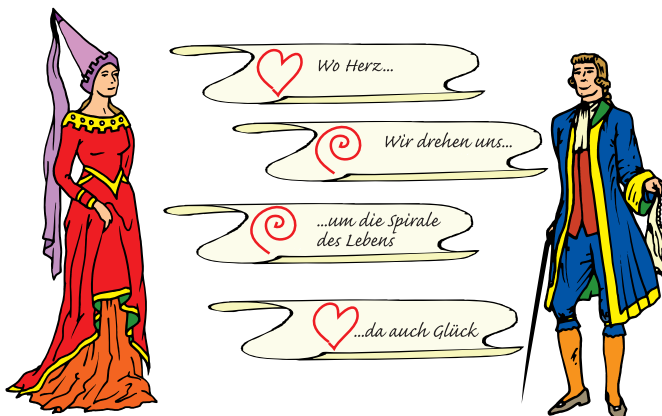


Abbildung 6.11 OSI-Layer 5:
Die Sitzungsschicht hält virtuelle
Verbindungen zwischen
zwei Computern im Internet

Das Internet ist paketorientiert. Das bedeutet, dass die Kommunikation über kleinere Datenpakete stattfindet und nicht durch eine feste Verbindung zwischen zwei Computern. Wenn Sie Ihrem Nachbarn eine E-Mail schicken, könnte theoretisch passieren, dass das erste und dritte Datenpaket auf kürzestem Wege zugestellt werden, während das zweite den Umweg über Australien nimmt. Router sollen oft die Last auf den Datenleitungen gleichmäßig verteilen und prinzipiell könnte dadurch so etwas zustande kommen. Das zweite Paket würde natürlich auch gegenüber dem ersten und dritten verzögert ankommen, wahrscheinlich kommt das dritte Paket sogar vor dem zweiten am Ziel an.

Trotzdem soll die E-Mail am anderen Ende wieder zusammengesetzt werden, als wenn sie über eine einzige, feste Verbindung geschickt worden wäre. Die gesamte Übertragung findet hierfür in einer „Sitzung“ statt, die von **OSI-Layer 5 (Session Layer = Sitzungsschicht)** beschrieben wird. Ähnlich dem Symbol, das Romero und Juliana auf die Briefe malen, beschreibt im Internet eine eindeutige Nummer die Zugehörigkeit eines Datenpakets zu einer Sitzung.

Nachts sind die Botendienste beider Königreiche geschlossen. Daher haben die beiden Liebenden ausgemacht, sich dann ihre Nachrichten anhand von Lichtzeichen mit einer starken Laterne zu schicken. Da nur „Licht an“ und „Licht aus“ erkennbar ist, verwenden sie das Morsealphabet (s. Kapitel „Von Kamelen und dem Nadelöhr“), um die einzelnen Zeichen zu Codieren (Abbildung 6.12).

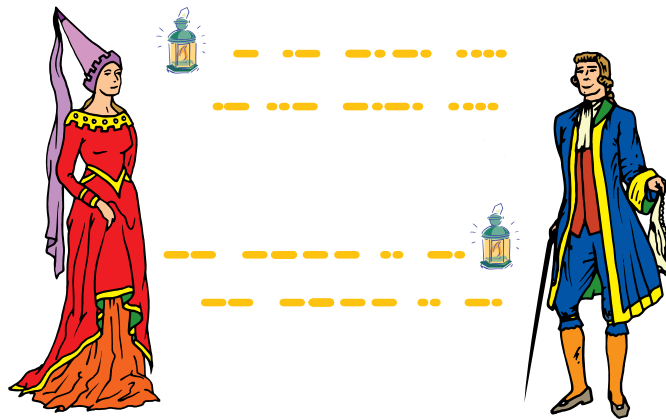


Abbildung 6.12: OSI-Layer 6:
Die Darstellungsschicht beschreibt die Codierung der Nachrichten

Computer übermitteln untereinander in den bisherigen fünf OSI-Schichten prinzipiell nur Zahlen. Erst **OSI-Layer 6 (Presentation Layer = Darstellungsschicht)** legt fest, wie diese Zahlen in Buchstaben oder zum Beispiel auch Zeichnungen gewandelt werden, um für uns Menschen verständlicher zu sein. Eine wesentliche Vereinbarung ist hier zum Beispiel, welche Zeichentabelle genutzt wird (zwei sehr bekannte sind ASCII und Unicode). Auch die Komprimierung von Daten, um das Netz besser auszunutzen, wird in dieser Schicht beschrieben.

Glücklicherweise liegen die beiden Königreiche von Informatix und Technokratix so dicht zusammen, dass in ihnen die gleiche Sprache gesprochen wird. Allerdings

studiert Juliana Philosophie und Romero Physik. So kommt es manchmal zu kleinen Missverständnissen, wie sie zwischen Geistes- und Naturwissenschaftlern üblich sind. Am Ende können beide sich selbstverständlich immer in der Sprache der Liebe verständigen und verstehen sich wieder (Abbildung 6.13).

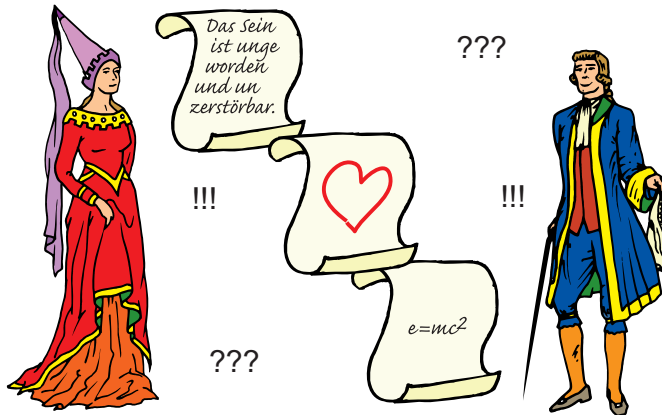


Abbildung 6.13 OSI-Layer 7: Die Anwendungsschicht interpretiert die Inhalte der Datenpakete

Menschen spezialisieren sich oft auf spezielle Fachgebiete und übernehmen die dort verbreiteten ganz eigenen Sprachen. So sind sie für Außenstehende manchmal kaum zu verstehen. In viel stärkerem Maße trifft das auf Computer zu. Für jede Art von Dienstleistung, die sie erbringen, kommunizieren sie mit einem ganz eigenen Vokabular (der Fachbegriff ist „Protokoll“). So gibt es eigene Sprachen für das Web (HTTP = Hypertext Transfer Protocol), E-Mail (SMTP = Small Mail Transfer Protocol), Austausch von Dateien (FTP = File Transfer Protocol) oder sogar für das Computer-Pendant der Zeitansage (NTP = Network Time Protocol). Diese sind in **OSI-Layer 7 (Application Layer = Anwendungsschicht)** spezifiziert. Damit zwei Computer miteinander reden können, müssen sie sich also auch auf dieser Ebene verstehen. Ein E-Mail-Programm hat zum Beispiel Schwierigkeiten, mit einem WWW-Server zu reden, weil es SMTP spricht und der Server aber nur HTTP versteht.

Für verschiedene Zwecke werden einer Nachricht also immer weitere Daten hinzugefügt, um sie im Internet zu verschicken – genau wie bei einem „echten“ Brief: Er wird geschrieben, unterschrieben, in einem Umschlag gepackt, mit den Adressen versehen. Eine Marke wird aufgeklebt. Die Post stempelt den Umschlag usw.

Sicherlich haben Sie auch in weiteren Bereichen einschlägige Erfahrung bezüglich der Kommunikation mit anderen Menschen gemacht. Nehmen Sie sich Romero und Juliana zum Vorbild und versuchen Sie, Ihnen bekannte Prozesse in das OSI-Modell einzuordnen.



Vielleicht fragen Sie nun, warum das OSI-Modell so wichtig ist, dass ich es hier in diesem populärwissenschaftlichen Buch aufgreife. Die erste Antwort ist, dass das Internet heute eines der wichtigsten Kommunikationsmittel geworden ist und das Wissen um den grundlegenden Aufbau viele Probleme mit der Bedienung oder dem Aufbau privater Netzwerkkomponenten vereinfacht.

Es gibt aber meiner Meinung nach noch einen anderen, gewichtigeren Aspekt: Ob Computer vernetzt werden oder es sich um Kommunikationsprozesse zwischen natürlichen Gesprächspartnern handelt – die Prinzipien der Informatik funktionieren auch hier und sorgen dafür, komplexe Sachverhalte und unübersichtliche Systeme einfach und handhabbar zu machen. Das gelingt nur, weil die Informatik immer wieder Prinzipien des menschlichen Denkens aufgreift und auf technische Systeme überträgt.

In vielen Wissenschaften gibt es das Prinzip der Modellbildung: Die Wirklichkeit wird durch ein Modell beschrieben, so dass die eigenen Beobachtungen damit möglichst vollständig erklärbar sind. Die Modelle verfeinern sich selbstverständlich. Für unsere archaischen Vorfahren war ein Weltbild völlig plausibel, in dem die flache Erde im Mittelpunkt steht, mit an den Himmel gehefteten Sternen und feurigen Rennwagen, die dort umherkreisen. Im Laufe der Zeit hat sich das Modell dann den neuen Erkenntnissen und Anforderungen angepasst: Zunächst rückte die Sonne in den Mittelpunkt, die Erde wurde auf die Umlaufbahn verwiesen. Im heutigen Modell ist auch unsere Sonne nur ein winziges Licht einer großen Galaxis.

Auch in der Informatik verwenden wir Modelle. Meistens nicht, um die Wirklichkeit möglichst exakt zu beschreiben, sondern um einen für die intendierte Problemlösung möglichst passenden Ausschnitt zu beschreiben (erinnern Sie sich noch an das Prinzip der Abstraktion)? So stellen wir unsere Welt zum Beispiel im Computer genau so dar, dass alle notwendigen Informationen zum Berechnen eines kürzesten Weges vorhanden sind.

Am Internet sehen Sie, dass dies auch umgekehrt möglich ist: Hier werden bekannte Strukturen der Realität verwendet, um ein technisches System nach dem gleichen Modell aufzubauen. Man folgt quasi dem Beispiel der Wirklichkeit und dafür gibt es einen speziellen Begriff: Paradigmenbildung.

Paradigma

Das Paradigma ist ein (Denk-)Muster oder Vorbild, anhand dessen ein technisches System oder ein abstraktes Konzept konstruiert wird. Dadurch wird es für die Anwender leichter, sich im System zurechtzufinden.

Bestes Beispiel für Paradigmenwechsel in der Informatik sind die Programmiersprachen: Zunächst sehr stark auf die direkte Umsetzung der Bedürfnisse der technischen Systeme zugeschnitten, wurde nach und nach immer mehr „menschliche“ Konzepte übernommen. Zum Beispiel mit der objektorientierten Programmierung.

Bei dieser besteht die Software in einer Ansammlung von Objekten, die miteinander kommunizieren – genau wie an einer „echten“ Arbeitsstelle: Dort sind die meisten Angestellten damit beschäftigt, Tätigkeiten an andere zu delegieren. Wenn sie das gut machen, läuft auch die Produktion gut (zugegebenerweise nur, wenn die Arbeit auch irgendwo unten in der Hierarchie wirklich verrichtet wird, idealerweise von Maschinen ...). Diesem Beispiel folgend, besteht die Programmierung weitgehend darin, den Objekten richtiges Delegieren „beizubringen“.

Resümee

Der Aufbau des Internet zeigt, wie menschlich letztlich die Informationstechnologie ist: Im Prinzip bedient sie sich jahrhundertealter Strukturen und Vorgehensweisen. Man darf dies auf keinen Fall mit einer Schwäche verwechseln:

Neue technische Möglichkeiten verlangen es zunächst immer von uns Menschen, dass wir uns anpassen, um die Vorteile zu genießen. Beispiel hierfür ist die Entwicklung der Computer, die zunächst nur von hoch spezialisierten Experten bedient werden konnten. Inzwischen sind die Computer der menschlichen Denkart weitgehend angepasst (wenn auch viele Menschen das stark anzweifeln, wenn der DVD-Player mal wieder nicht so funktioniert wie erwartet).

Genauso ist dies auch mit Datennetzen: Zunächst technisch sehr kompliziert und aufwendig gestaltet, waren sie nicht zuverlässig nutzbar. Man hat sich daher Strukturen ausgedacht, die „menschlicher“ ausgeprägt und dadurch beherrschbar sind. Das heutige, moderne Internet ist aus diesem Grund prinzipiell immer noch mit der Hierarchie einer mittelalterlichen Pferdepost vergleichbar! Nur viel größer und natürlich viel schneller ...



8. Ordnung im Chaos

Viele Chefs glauben, dass ein Chaos auf dem Schreibtisch der Mitarbeiter für Chaos in deren Kopf steht. Verschiedene Abhandlungen über Karriereplanung empfehlen daher dringend, Ordnung zu halten. Albert Einstein und Sigmund Freud waren beide eher Favoriten des Chaos: Laut Berichten von Zeitgenossen hatten sie den typischen „Krater“ aus Dokumenten und angefangenen Schriftstücken auf ihrem Arbeitsplatz. Für Einstein war dieses Chaos der beste Beweis für die Sätze der Thermodynamik, nach denen Chaos der natürliche Zustand ist. Sigmund Freud hatte sogar etwas weitergehende Theorien, was den psychologischen Hintergrund eines Ordnungsliebhhabers angeht, aber das möchte ich hier gar nicht wiederholen.



Wichtig ist, dass es offenbar zwei Geisteshaltungen bezüglich des Arbeitsplatzes gibt – die der offensichtlichen Ordnung und die des offensichtlichen Chaos. Die Informatik ist bereits in vielen anderen Gebieten Vermittler zwischen unterschiedlichen Disziplinen und vielleicht kann sie auch in diesem Fall eine Brücke bilden.

Eine der wichtigen Methoden, um Daten im Computerspeicher abzulegen und schnell wieder darauf zuzugreifen, führt nämlich zu einer chaotischen Anordnung! Trotzdem können einzelne Datensätze in Windeseile wieder hervorgeholt werden. Trügt also hier der Schein? Im folgenden Kapitel werden wir uns dieser Thematik näher widmen.

Es wäre übrigens günstig, wenn Sie das Kapitel „Sortieren“ vorher durcharbeiten, da dieses Kapitel größtenteils daran anknüpft.

Warum Ordnung?

Im zweiten Kapitel haben wir uns ausschließlich mit dem Sortieren beschäftigt, ohne wirklich darüber nachzudenken, warum das eigentlich sinnvoll ist. Das Ordnunghalten begleitet uns Menschen seit frühester Kindheit und so stellen wir es gar nicht mehr in Frage: Natürlich muss eine Folge von Zahlen sortiert werden, ansonsten kann man doch gar nichts damit anfangen, oder?

Um zu ergründen, ob das auch für Maschinen gilt, versetzen wir uns ein weiteres Mal in die Lage des Computers. Packen Sie den Papierspeicher aus und 16 Kärt-

chen mit Zahlen darauf. Mischen Sie die Karten und verteilen Sie diese auf den ersten 16 Speicherpositionen. Die Zeiger „Anfang“, „Ende“ und „Speicherzeiger“ erleichtern das Nachspielen der Vorgehensweise des Computers (s. Abbildung 8.1).

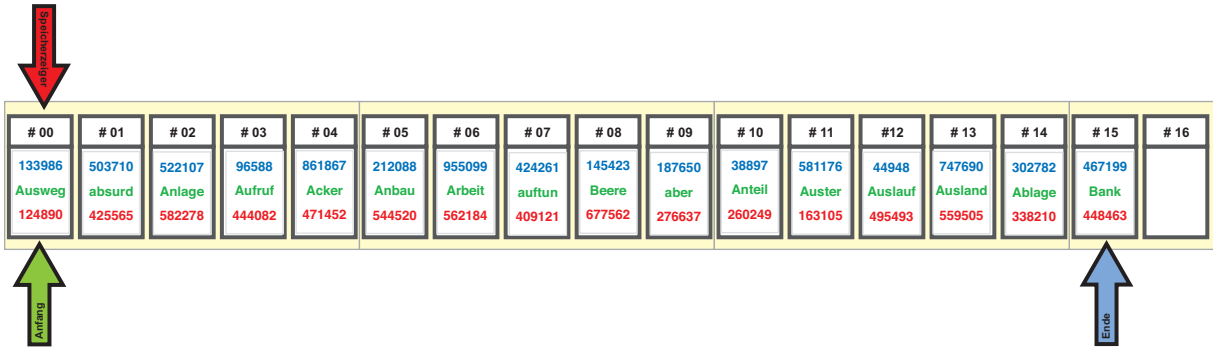


Abbildung 8.1 Papierspeicher mit chaotischem Inhalt

Denken Sie daran, der Computer kann immer nur die Karte unter einem der Zeiger „sehen“ und muss den Zeiger neu setzen, um eine andere Karte anzuschauen.

Finden Sie als Computer eine bestimmte vorhandene Karte (zum Beispiel die mit der blauen 38897). Versuchen Sie dabei, möglichst geschickt vorzugehen. Schreiben Sie auf, wie viele „Rechenschritte“ Sie benötigen haben und wiederholen Sie den Versuch ein paar Mal. Auf diese Weise können Sie abschätzen, wie lange die Suche im Mittel dauert.



Egal, wie Sie es drehen und wenden, letztendlich bleibt bei einem chaotischen Speicherinhalt immer nur die Möglichkeit, eine Karte nach der anderen zu überprüfen, bis man die gesuchte gefunden hat – zum Beispiel nach dem Algorithmus in Abbildung 8.2. Man nennt das auch „sequentielle Suche“, weil man der Reihe nach sucht. Der Algorithmus geht übrigens davon aus, dass die gesuchte Karte definitiv im Speicher vorhanden ist.

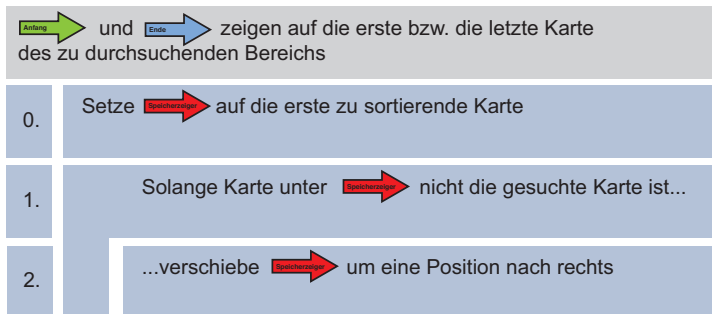


Abbildung 8.2 Sequentielle Suche

Im Mittel muss man etwa die Hälfte der Karten anschauen, bis man fündig wird, denn die gesuchte Karte kann sich ja zufällig ganz am Anfang oder auch ganz am Ende befinden. Bei 16 Karten wird man also durchschnittlich etwa nach 8 Karten fertig sein.

Als Nächstes sortieren Sie bitte die Karten, zum Beispiel nach der blauen Zahl. Verwenden Sie zum Üben am besten eines der Verfahren aus dem zweiten Kapitel. Selbstverständlich dürfen Sie die Karten auch „einfach so ordnen“. Abbildung 8.3 zeigt eine sortierte Folge.



# 00	# 01	# 02	# 03	# 04	# 05	# 06	# 07	# 08	# 09	# 10	# 11	# 12	# 13	# 14	# 15	# 16
38897	44948	96588	133986	145423	187650	212088	302782	424261	467199	503710	522107	581176	747690	861867	955099	
Anteil	Auslauf	Aufruf	Ausweg	Beere	aber	Anbau	Ablage	auf tun	Bank	absurd	Anlage	Auster	Ausland	Acker	Arbeit	
260249	495493	444082	124890	677562	276637	544520	338210	409121	448463	425565	582278	163105	559505	471452	562184	




Abbildung 8.3 Papierspeicher mit sortierter Kartenfolge

Versuchen Sie nun wiederum, in möglichst wenigen Schritten eine bestimmte Karte zu finden. Kleiner Tipp: Fangen Sie mit dem Speicherzeiger auf der Position #07 an, also etwa in der Mitte.



Vielleicht erinnern Sie sich noch an das Zahlenratespiel: Ein Spieler denkt sich eine Zahl zwischen 0 und 100 aus und ein anderer soll die Zahl in möglichst wenigen Versuchen raten. Nach jedem Versuch sagt der erste Spieler, ob die gesuchte Zahl größer oder kleiner ist.

Hier beginnt man am besten in der Mitte und halbiert dann immer die Anzahl noch möglicher Zahlen. Genau das gleiche Prinzip können wir auch mit dem Papierspeicher zum Suchen einer Karte benutzen, falls die Karten im Speicher sortiert sind:

Wir beginnen in der Mitte und kontrollieren, ob die Karte die gesuchte ist. Wenn nein, schauen wir, ob die Zahl größer oder kleiner als die gesuchte ist. Ist die gesuchte Zahl größer, muss sie sich rechts vom Speicherzeiger befinden. Ist sie kleiner, muss sie links sein. Das ergibt sich direkt aus der Sortierung!

Auf diese Weise haben wir bereits die Hälfte der vorhandenen Karten ausgeschlossen und brauchen nur noch in der anderen Hälfte zu suchen. Wir setzen den Speicherzeiger wiederum in die Mitte und wenden das gleiche Verfahren an, um erneut die Hälfte der Karten auszuschließen.

Einen entsprechenden Algorithmus sehen Sie in Abbildung 8.4. Das Verfahren heißt auch „binäre Suche“ in einer sortierten Liste, weil man den Suchbereich immer in zwei Hälften teilt. Auch dieser Algorithmus funktioniert übrigens nur korrekt, wenn sich die gesuchte Karte auf jeden Fall im Speicher befindet.

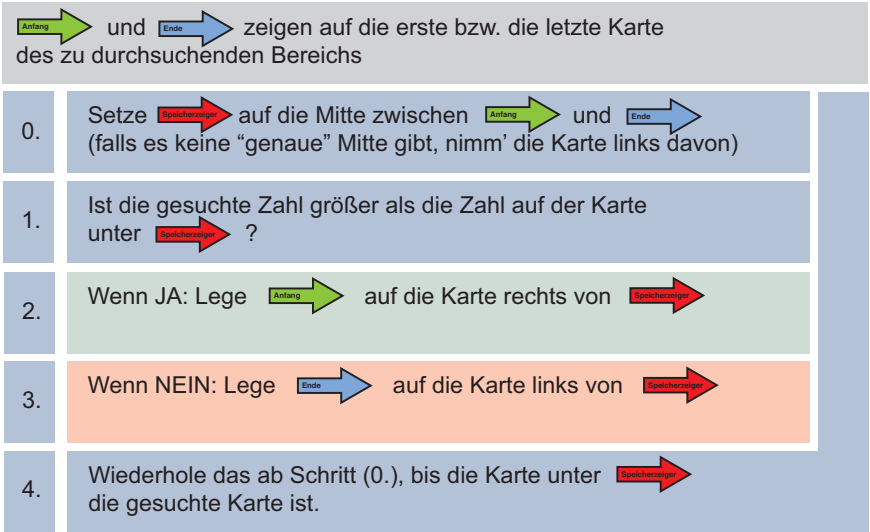


Abbildung 8.4 Algorithmus zur binären Suche

Wenn Sie die binäre Suche ein paar Mal durchführen, merken Sie, dass Sie hier durchschnittlich 4, maximal 5 Vergleiche benötigen, um die Karte zu finden. Das Verfahren bringt uns also schneller ans Ziel.

Das macht sich bei größeren Datenmengen noch viel stärker bemerkbar: Die binäre Suche schließt pro Schritt die Hälfte aller Daten aus. Umgekehrt ausgedrückt benötigen wir also für die Suche in einer doppelt so großen Datenbank nur einen einzigen Schritt mehr. Bei der sequentiellen Suche bräuchte der Algorithmus auch tatsächlich doppelt so viele Schritte.

Die Experten wissen natürlich bereits aus Kapitel 2, dass sich der Suchaufwand bei n Karten formal durch $\log_2 n$ ausdrücken lässt.

Ordnung oder Chaos?

Wir wissen jetzt, dass Ordnung nicht nur einen ästhetischen Wert für uns Menschen darstellt, sondern selbst für Maschinen und deren Algorithmen echte Vorteile bringt. Als Nächstes möchte ich Ihnen die Frage stellen, was Ordnung eigentlich ist. Hierfür machen wir ein kleines Spiel:

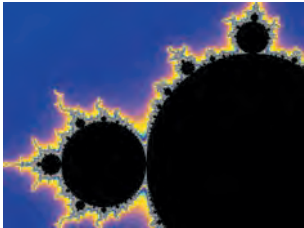
Die folgenden Tabellen enthalten immer die gleichen 10 Zahlen in unterschiedlicher Reihenfolge. Welche Tabellen sind sortiert, welche nicht? Können Sie den Grad der Unordnung in den Ihrer Meinung nach unsortierten Tabellen beurteilen?

Pos.	Reihe A	Reihe B	Reihe C	Reihe D
0	727	932018	—	973412
1	213121	213121	213121	727
2	545955	832213	903512	961347
3	832213	961347	961347	545955
4	903512	973412	979834	945630
5	932018	903512	932018	903512
6	945630	945630	545955	—
7	961347	727	727	213121
8	973412	979834	973412	832213
9	979834	545955	945630	979834
10	—	—	832213	932018

Das Chaos im Wandel der Zeit

Das Wort kommt aus dem Griechischen und meint „den unendlichen leeren Raum“ oder „die gestaltlose Urmasse des Weltraums“. Biblisch wurde die Welt aus dem Chaos geschaffen. Früher verstand man also etwas Ursprüngliches, noch nicht Geformtes unter dem Chaos.

Heute bezeichnet man jede Form von Unordnung – besonders auch die von uns allen geschaffene – als Chaos. Die Chaostheorie beschäftigt sich hingegen eher mit dem Gegenteil: Ordnung soll in scheinbar chaotischen Systemen gefunden werden. Bekanntestes Beispiel aus der Informatik sind die selbstähnlichen Fraktale – zum Beispiel die aus einer sehr einfachen Formel gewonnene Mandelbrot-Menge („Apfelmännchen“), die auch beliebig fein dargestellt immer wieder die gleichen, sich wiederholenden Grundfiguren enthält.



Solche Bilder kann man leicht mit ein paar Zeilen Programmcode oder mit dem freien Programm Fractal Explorer selbst herstellen.



Selbstverständlich erkennen Sie Reihe A als sortiert, die Reihen B bis D erscheinen dagegen auf den ersten Blick etwas chaotisch zu sein. Vielleicht haben Sie aber doch bereits eine Ordnung sehen können:

Schlüssel hierfür ist das Sortierkriterium. Bei Reihe A ist das ganz offensichtlich – die Zahlen sind der Größe nach von der kleinsten zur größten aufgeführt. Die Reihe B ist da schon etwas subtiler: Betrachten Sie nur die drei hinteren Stellen der Zahlen, also die letzten drei Ziffern. Stimmt nun die Sortierung wieder?

Ich behaupte jetzt, dass auf die gleiche Weise auch für die Reihen C und D ein Sortierkriterium existiert. Es ist allerdings eine ziemlich harte Nuss, darauf zu kommen. Wenn Sie Lust haben, versuchen Sie, die Regel herauszufinden, sonst lesen Sie einfach weiter.



Hinter Reihe C steckt die doppelte Quersumme: Nehmen Sie die Ziffern der Zahlen einzeln und addieren Sie sie. Das Gleiche machen Sie nochmals mit dem Ergebnis. Die Resultate liegen bei den gegebenen Zahlen zwischen 1 und 10 und genau danach sind sie sortiert.

Beispiel:

Die Quersumme von 945630 ist $9+4+5+6+3+0=27$

Die Quersumme von 27 ist $2+7=9$

Daher ist 945630 in Zeile 9 eingeordnet.

Um das Geheimnis der Reihe D zu knacken, denken Sie an die schriftliche Division mit Rest aus der Grundschule zurück. Dort rechnet man nicht mit Brüchen, sondern nur mit ganzen Zahlen. Dabei kann immer ein Divisionsrest übrig bleiben.

27 geteilt durch 12 ergibt 2, Rest 3

In der Informatik braucht man häufig nur den Rest dieser Division und gar nicht das Ergebnis, daher hat man die Berechnung des Restes zur eigenen Rechenart gemacht, genannt „modulo“ oder abgekürzt „mod“. Man schreibt zum Beispiel

$27 \bmod 12 = 3$

Damit gleich Verwechslungen vermieden werden: Dies ist ein verwandter, aber trotzdem etwas anderer Gebrauch von „mod“ als bei den Mathematikern!

Übrigens:

Die einzige Zahl mit Quersumme 0 ist die 0 selbst ...

Wiederholung: Schriftliche Division mit Rest

In Zeiten des Taschenrechners verlernen wir sehr schnell die schriftliche Division mit Rest, wie man sie in der Grundschule betreibt. Das ist völlig normal – selbst bei den Studierenden der Informatik oder Mathematik gibt es sehr viele, die sich diese Technik erst wieder aneignen müssen. Daher hier eine kleine Wiederholung für alle, die sich nicht mehr genau daran erinnern ...

Prinzipiell nutzen wir unser Dezimalsystem aus, um größere Divisionen der Form

Dividend : Divisor = Ergebnis

in kleinere, leichter zu beherrschende Operationen zu unterteilen. Insofern ein sehr Informatik-nahes Vorgehen! Am Beispiel lässt das sich immer am besten lernen, daher versuchen wir es mit

$$498737894 : 13 = ?$$

Zunächst nehmen wir so viele der ersten Ziffern des Dividenden, dass die entstehende Zahl gerade größer wird als der Divisor. In diesem Fall also „49“. Wir teilen diese Zahl durch den Divisor und schreiben das (einstellige) Ergebnis hinter das Gleichheitszeichen. Dabei beachten wir nur ganze Anteile des Ergebnisses.

$$498737894 : 13 = 3$$

Die Zahl ist aber meistens nicht glatt teilbar. In diesem Fall bedeutet die 3 also, dass der Divisor 13 genau 3-mal glatt in 49 unterzubringen ist. Da 3 mal 13 jedoch 39 ergibt, bleibt ein Rest. Diesen ermitteln wir formal, indem wir die gerade errechnete Ziffer mit dem Divisor multiplizieren (= 39) und direkt unter die ersten Ziffern des Dividenden schreiben. Per Subtraktion ermitteln wir dann den Divisionsrest dieser ersten Operation und schreiben ihn hin. Mit diesem Rest muss dann die Division noch weiter betrieben werden.

$$498737894 : 13 = 3$$

39

10

Allerdings haben wir ja noch einen weiteren „Rest“, nämlich die Ziffern, die wir bisher gar nicht betrachtet hatten. Sie sind in der folgenden Formel nochmals grau wiederholt, weil man sie normalerweise nicht hinschreibt, aber trotzdem mit ihnen rechnet:

$$498737894 : 13 = 3$$

39

$$108737894 : 13 = 3...$$

Jetzt geht es wirklich à la Informatik weiter: Wir verfahren mit dieser neuen Berechnung (fast) ebenso wie am Anfang mit der kompletten Division, nur dass die bereits ermittelten Anteile (in diesem Fall die 3) einfach stehen bleiben. Statt wieder zu ermitteln, wie viele Ziffern notwendig sind, damit sie größer als der Divisor werden, nehmen wir jetzt immer nur die nächste Ziffer zum Rest hinzu. In diesem Fall also die 8.

Danach wird wieder geteilt sowie der Rest und damit eine neue Rest-Berechnung ermittelt.

$$498737894 : 13 = 3$$

39

$$108737894 : 13 = 38$$

104

$$4737894 : 13 = 38...$$

Das machen wir, bis der entstandene Divisionsrest so klein ist, dass er kleiner als der Divisor ist – die Zahl lässt sich nicht mehr weiter teilen und wir schreiben den Rest einfach daneben. In Abbildung 8.5 ist die Division zur Kontrolle nochmals komplett dargestellt. Die grauen Teile schreibt man normalerweise nicht mehr auf, sie dienen hier nur der Verdeutlichung, dass im Prinzip immer wieder der gleiche „Algorithmus“ durchgeführt wird. Für die „modulo“-Rechnung, wie sie in diesem Kapitel häufig verwendet wird, benötigen wir allerdings nur den Rest, nicht das eigentliche Ergebnis.

498737894 : 13 = 38364453 Rest 5
39
 108737894 : 13 = 3...
104
 4737894 : 13 = 38...
39
 837894 : 13 = 383...
78
 57894 : 13 = 3836...
52
 5894 : 13 = 38364...
52
 694 : 13 = 383644...
65
 44 : 13 = 3836445...
39
 5 : 13 = 38364453 Rest 5

Abbildung 8.5 Schriftliche Division

Um herauszufinden, wie die Zahlen der Reihe D geordnet sind, berechnen Sie einfach immer den Divisionsrest beim Teilen durch 11.

Beispiel:

$$945630 \bmod 11 = 4$$

So ist 945630 in Zeile 4 einsortiert.

Es sind also wirklich alle der oben angegebenen Reihenfolgen geordnet, allerdings nur in Reihe A nach einem uns Menschen offensichtlichen Kriterium. An die Ordnung von Reihe B könnten wir uns vielleicht auch noch gewöhnen, aber Berechnungen anstellen, um Zahlen zu sortieren, wie in den Reihen C und D – das geht zu weit. So werden Computer auch immer die „menschlichen“ Sortierkriterien verwenden, wenn es darum geht, den Benutzern etwas zu präsentieren. Beispiele hierfür sind Telefonverzeichnisse, Kundendateien, eine Hand voll Spielkarten bei einem Skatprogramm usw.

Wie sieht es allerdings aus, wenn die Daten ausschließlich im Computer gespeichert werden und dort auch verbleiben? Wenn jeder, der einen bestimmten Datensatz sucht, einfach den Computer danach fragt? Computer rechnen sehr schnell, haben also kein Problem mit „seltsamen“ Sortierkriterien. Das heißt, prinzipiell spricht nichts dagegen, Daten im Computerspeicher nach etwas anderem als der Größe zu sortieren. Die nächste Frage ist: Spricht etwas dafür?

Um sie zu beantworten, schauen wir uns nochmals die Tabelle an. Um einen Eintrag aus Reihe A zu finden, kann man die binäre Suche anwenden. Genauso kann man auch suchen, um mit dem entsprechenden Kriterium einen Eintrag aus den Reihen B bis D zu finden. Aber es geht auch schneller:

In den hinteren Reihen bestimmt das Sortierkriterium nicht nur die relative Position der Einträge untereinander (so wie „832213 kommt vor 973412“), sondern es legt die absolute Position in der Tabelle fest.

Um einen Eintrag in Reihe B zu finden, nehmen Sie einfach die dritte Ziffer von rechts – sie ist identisch mit der Zeilennummer. Bei Reihe C ist das die doppelte Quersumme und bei Reihe D der Divisionsrest beim Teilen durch 11. Das erklärt auch das Rätsel, warum in Reihe D, Zeile 6 kein Eintrag steht: Unter den zu speichernden Zahlen gibt es einfach keine mit Rest 6 bei der Division durch 11.

Erkennen Sie den Vorteil? Statt nach einem Eintrag suchen zu müssen, berechnet der Computer einfach dessen Position. Zum Speichern genügt ein einzelner Schreibvorgang. Um die Daten wieder abzurufen, genügt ein einziger Lesevorgang. Dieses Verfahren setzt man daher tatsächlich immer dann ein, wenn es sehr wichtig ist, Daten sehr schnell abzuspeichern und sehr schnell wieder abzurufen.

Wie oben in den Reihen B bis D werden die Daten dabei oft in einer für den Menschen chaotisch aussehenden Weise angeordnet. Daher nennt man das Verfahren „Hashing“, von englisch „Hash“ – „Durcheinander, Kuddelmuddel“.

Wenn Sie genau darüber nachdenken, kennen Sie Hashing auch bereits sicherlich aus Ihrem Alltag, hier schließe ich mal von mir auf andere: Wie oft gibt es die Situation, in der man irgendein Objekt wegsortiert, ohne darüber nachzudenken. Auf diese Weise landet die Einkaufsquittung im Bücherregal oder der Lieblingskugelschreiber in der Werkzeugkiste. Überraschenderweise finde ich diese Gegenstände (meistens) sehr leicht wieder. Indem ich so tue, als wenn ich den Kugelschreiber erneut weglegen wollte, komme ich erneut auf die gleiche Idee „Werkzeugkiste“ und werde dort fündig!

Ungleich schwieriger wird die Sache manchmal, wenn ich beim Aufräumen eine ganz bewusste Entscheidung getroffen habe: Den Pass für die Reise nächste Woche habe ich an einen Ort gelegt, wo ich ihn garantiert sicher wiederfinde – nur fällt mir zwei Tage später absolut nicht mehr ein, wo dieser Ort ist. Natürlich weiß ich noch ganz genau, wo ich den Paß gefunden hatte, bevor ich ihn am vermeintlich sicheren Ort deponierte ... hätte ich ihn nur dort gelassen!

Kommt Ihnen das irgendwie bekannt vor?

Genau dieses „intuitive“ Prinzip wird beim Hashing genutzt. Aufgrund einer Formel wird der Speicherort für einen Datensatz festgelegt und wenn wir ihn danach suchen, finden wir ihn aufgrund der gleichen Formel wieder. Bevor ich nun genauer darauf eingehe, wie das Verfahren günstig umgesetzt werden kann, möchte ich eine andere Frage aufwerfen: Wo ist der Haken? Warum benutzt man überhaupt noch eine konventionelle Sortierung?

Das können wir experimentell beantworten: Nehmen Sie die 16 Sortierkarten mit Buchstaben, mischen Sie sie und legen Sie dann eine nach der anderen auf einem großen Schreibtisch ab. Schauen Sie sich dabei jeweils den Buchstaben an und legen die Karte dann verdeckt so ab, dass Sie sie wahrscheinlich wiederfinden.

Loci

Die so genannte Loci-Methode zur Steigerung der Merkfähigkeit macht sich übrigens zunutze, dass wir uns recht gut daran erinnern können, wo wir bestimmte Dinge verstauen. Möchten Sie sich etwa die Liste für den nächsten Einkauf merken, legen Sie die gewünschten Objekte in Gedanken in Ihrem Büro oder Wohnzimmer ab. Im Laden müssen Sie sich dann nur noch das Zimmer vorstellen, um sich wieder an die Einkaufsliste zu erinnern. Soweit zumindest die Theorie ... Bitte probieren Sie das besser nicht gerade am Abend vor einem Feiertag aus ...

Suchen Sie nun die Karte mit dem „F“! Wahrscheinlich finden Sie diese recht schnell.

Jetzt machen Sie das Gleiche noch einmal, allerdings diesmal mit einer sehr kleinen Fläche: Die Karten müssen alle auf einem Bogen Papier DIN-A4 Platz haben.

Versuchen Sie nun die Karte mit dem „L“ zu finden! Ist das ebenfalls so einfach? Wahrscheinlich sind Sie hier eher geneigt, die Karten beim Hinlegen zu sortieren.

Führen Sie das Experiment mit verschiedenen Testpersonen durch. Wahrscheinlich haben die meisten kein Problem, eine Karte auf der großen Fläche wiederzufinden, brauchen allerdings mehrere Versuche, um dies auf der kleinen Fläche zu schaffen. Genauso fällt es uns deutlich leichter, Gegenstände in einem großen Raum einfach „intuitiv“ zu platzieren und dann wiederzufinden als in einem kleinen.

Hashing benötigt Platz! Das ist im Computer nicht anders als beim realen Versuch mit den Karten. Daher benutzt man das Verfahren immer dann, wenn der schnelle Zugriff auf die Daten wichtiger ist als kleiner Speicherverbrauch. Das wird im Folgenden noch klarer, wenn wir das eigentliche Verfahren besprochen haben!

Die nächste Frage ist: Warum kann das Kriterium für Hashing nicht so gewählt werden, dass letztlich auch eine sortierte Folge herauskommt? Die Antwort haben wir im Wesentlichen im Kapitel über das Sortieren bereits unter dem Stichwort „Proxmap-Sort“ beantwortet: Leider haben Zahlen und andere Schlüsselbegriffe, die von Menschen erzeugt wurden, die Tendenz, sich in kleinen Bereichen zu häufen.

Wenn man zum Beispiel die Neuerscheinungen des Jahres 2006 nach ihren Standard-Buchnummern (ISBN) sortiert, werden wahrscheinlich viele kleine Zahlenbereiche dicht gefüllt sein und große Bereiche fehlen (die Nummern, die die Verlage bereits 2005 und früher vergeben haben und die, die noch zu vergeben sind). Das liegt daran, dass die meisten Verlage ihre Buchnummern aufsteigend lückenlos vergeben.

Ein anderes Beispiel ist die Datei einer Krankenkasse, in der die Mitglieder nach Geburtsdatum angeordnet sind – auch hier werden bestimmte Jahreszahlen besonders häufig vorkommen, weil es einfach mehr Menschen mit einem Alter von 45 gibt als Menschen mit 100.

Betrachten Sie nochmals die Tabellen von oben: Versuchen Sie, die gegebenen Zahlen mit dem Kriterium „erste Stelle = Hunderttausender“ in die korrekte Zeile einzuordnen. Das funktioniert auch mit den kleineren fünf Zahlen. Allerdings müssten die nächsten fünf Zahlen alle in die bereits besetzte Zeile 9! Wie das aber mit Speicherstellen so ist: Sie bieten nur Platz für einen einzigen Wert.

Pos.	Reihe
0	727
1	
2	213121
3	
4	
5	545955
6	
7	
8	832213
9	903512

Beim Proxmap-Sort haben wir uns durch dynamische Anpassung der Skala für die Tabelle beholfen: In Fall des Beispiels würden nur wenige Zeilen für die Zahlen zwischen 0 und 900.000 reserviert und mehr Zeilen für Zahlen zwischen 900.000 und 999.999, so wie in der nächsten Tabelle. Um diese Einteilung zu finden, muss man jedoch erst eine statistische Analyse der einzusortierenden Zahlen durchführen. Das ist manchmal einfach nur sehr aufwendig, oft jedoch sogar unmöglich, weil man zu Beginn noch gar nicht weiß, mit welchen Zahlen man rechnen muss.

Pos.	Reihe
00-20	727
21-41	213121
42-62	545955
63-83	832213
84-90	903512
91-93	932018
94-95	945630
96-96	961347
970-975	973412
976-999	979834

Effektiv wird mit dieser veränderten Einteilung die **Gleichverteilung** über dem Speicherplatz bewirkt: Wenn man eine Anzahl von Zahlen einsortiert, dann werden diese ungefähr immer gleichmäßig aufgeteilt. Auf diese Weise ist es am wenigsten wahrscheinlich, dass zwei Zahlen die gleiche Speicherstelle belegen würden.

Wie ganz am Anfang aus der Tabelle hervorgegangen ist, können wir diese Gleichverteilung jedoch auch anders herstellen: Indem wir das Sortierkriterium verändern. Wie, das wollen wir jetzt im Experiment herausfinden.

Wussten Sie übrigens ...

... wenn 30 Menschen zufällig zusammenkommen (zum Beispiel in einer Schulklasse), dann ist es sehr wahrscheinlich, dass zwei oder mehr davon am gleichen Tag Geburtstag feiern. Genau genommen liegt die Wahrscheinlichkeit ungefähr bei 70 %. Bei 40 Personen sind das bereits ca. 90 %. Sie sehen an diesem Beispiel, dass es fast unmöglich ist, Kollisionen ganz zu vermeiden.

Sie benötigen die Sortierkarten mit den roten Zahlen und den Papierspeicher. 10 Speicherstellen sollen ausreichen. Mischen Sie die Karten und nehmen Sie dann immer 10 Karten, um sie auf 10 Speicherpositionen zu verteilen. Dabei nutzen Sie bitte zunächst die Hunderttausender-Stelle der roten Zahlen als Kriterium. Die Zahl 544.520 kommt zum Beispiel in die Speicherstelle 05 und die Zahl 66.580 auf die Speicherstelle 0 (da die Zahl kleiner als 100.000 ist).

Führen Sie diesen Versuch 10- bis 20-mal durch und schreiben Sie jeweils auf, wie viele Karten nicht mehr in eine Speicherstelle gepasst haben, weil dort bereits eine Karte lag. So etwas nennt man übrigens „Kollision“, weil die gewünschte Einordnung einer Karte mit der dort bereits liegenden Karte „kollidiert“.

Bestimmen Sie die mittlere Anzahl der Kollisionen, indem Sie die Summe durch die Anzahl der Versuche teilen. Zum Beispiel bin ich bei 20 Versuchen auf 97 Kollisionen gekommen, was durchschnittlich 4,85 Kollisionen entspricht.

Jetzt machen Sie den gleichen Versuch nochmals – allerdings nehmen Sie nun die letzte Stelle, also die Einer-Stelle, um die Karten auf die Speicherstellen zu sortieren. Die Zahlen 544.520 und 66.580 kommen nun also beide auf die Speicherstelle 0 – was bereits zu einer Kollision führt, wenn wir diese Karten in einem gemischten Paket finden.

Bestimmen Sie wieder die durchschnittliche Anzahl der Kollisionen. Ist dieses Ergebnis besser oder schlechter als das Ergebnis mit den Hunderter-Stellen?



Natürlich ist das mit statistischen Versuchen immer so eine Sache, aber wahrscheinlich haben Sie ein ähnliches Ergebnis wie ich heraus: Bei 20 Versuchen waren das 75 Kollisionen, also durchschnittlich 3,75. Man kann also unter gleichen Voraussetzungen mit diesem Kriterium ungefähr eine Karte mehr speichern.

Offenbar geht es beim Hashing also tatsächlich hauptsächlich darum, ein gutes Sortierkriterium zu finden. In diesem Fall scheint zum Beispiel das Kriterium „Einer-Stelle“ besser geeignet zu sein als „Hunderttausender-Stelle“, weil es eine gleichmäßigere Verteilung der Karten auf die Speicherstellen ergibt.

Trotzdem sollte man selbstverständlich eine Karte auch einordnen können, wenn ihr eigentlicher Platz bereits belegt ist. Denken wir wieder an das Beispiel Zimmer zurück: Wenn Sie einen Gegenstand in eine Schublade legen möchten, diese ist aber bereits bis zum Rand gefüllt, fällt Ihnen sicherlich noch ein „zweitbester“ Ort ein, etwa die nächste Schublade darunter. Wenn Sie beim Suchen nach dem Gegenstand dann am ersten Platz keinen Erfolg haben, ziehen Sie sicherlich auch ganz intuitiv die nächste Schublade auf, um den Gegenstand dann zu finden.

Beim Hashing nennt man dies „Kollisionsbehandlung“. Diese wird durchgeführt, sobald ein Datensatz nicht an seiner eigentlichen Stelle in den Speicher geschrieben werden kann, weil diese bereits belegt ist. Die einfachste Strategie zur Kollisionsbehandlung ist das „Zweitbeste“-Prinzip.

sionsbehandlung erinnert an unser Vorgehen beim Aufräumen im Zimmer: „Nimm’ einfach die nächste Schublade“ oder im Fall des Computers „Nimm’ einfach die nächste freie Speicherstelle“.

Versuchen Sie nun einfach einmal, einen neuen Satz aus zehn gemischten Karten auf die zehn Speicherstellen zu hashen. Ein Beispiel dafür sehen Sie in Abbildung 8.6. Dort werden die Karten von links nach rechts gemäß dem Kriterium „Einer-Stelle der roten Zahl“ auf die Speicherstellen verteilt. Bitte beachten Sie, dass man wieder von vorne anfängt, wenn man beim Suchen der nächsten freien Speicherstelle am Ende angekommen ist!

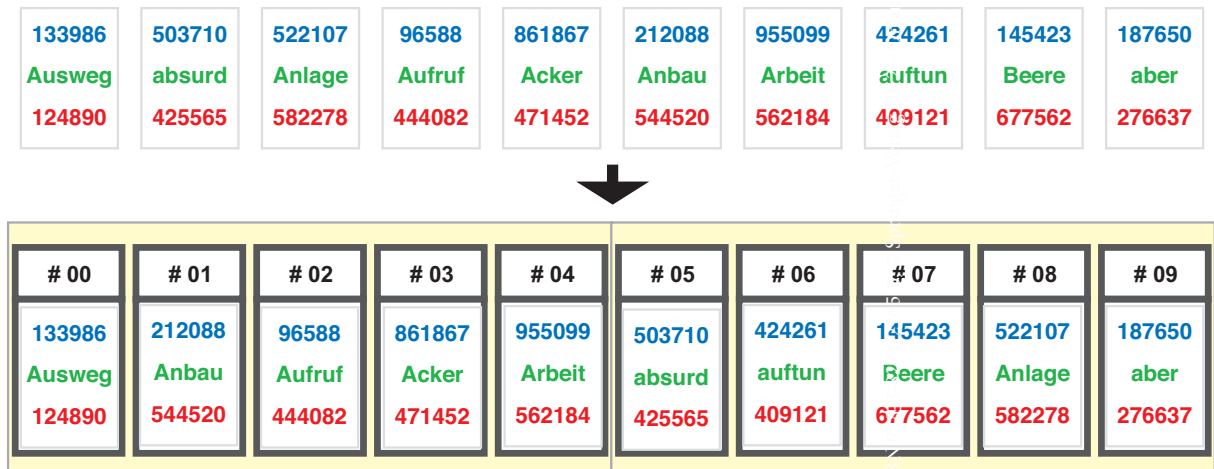


Abbildung 8.6 Hash mit Kollisionsbehandlung



Beim Suchen nach einer Karte darf man sich nun natürlich ebenfalls nicht darauf beschränken, an der ihr zugeteilten Speicherstelle zu schauen. Falls sich dort eine andere Karte befindet, muss man auch die nächsten Speicherpositionen durchsuchen.

Suchen wir in Abbildung 5 beispielsweise die Karte mit der roten Zahl 409121. Eigentlich würden wir sie auf Position #01 vermuten, da die letzte Stelle 1 ist. Dort befindet sich allerdings eine andere Karte. Also versuchen wir es eine Position weiter. Auch dort werden wir nicht fündig. Insgesamt müssen wir 6 Karten anschauen, bis wir die gesuchte Zahl erreichen! Mit der binären Suche in einer sortierten Folge hätten wir maximal 4 Vergleiche benötigt, das Hashing ist also sogar ungünstiger!

Weiter oben hatte ich bereits beschrieben, dass das intuitive Aufräumen in einem sehr kleinen Zimmer nicht funktioniert. Hier können Sie nun das Pendant beim Hashing erkennen: Wenn der Speicherplatz zu dicht belegt ist, nimmt man besser die klassische Sortierung der Daten, zusammen mit der binären Suche. Das innovative Hashing benötigt mehr Platz!

Aber wieviel Speicherplatz ist eigentlich notwendig? Als Nächstes wollen wir durch Probieren ermitteln, wie gut das Hashing mit verschiedenen Belegungsfaktoren abschneidet.

Bitte mischen Sie den Kartenstapel wieder. Nehmen Sie nun 10 Karten und hashen diese in die 10 Speicherstellen – genau wie oben. Diesmal führen Sie jedoch eine Strichliste: Für jede Speicherposition, die Sie ansprechen müssen (also auch belegte) machen Sie einen Strich. Falls also eine Karte erst in die vierte Speicherstelle abgelegt werden kann, weil die drei ersten Positionen belegt sind, ergibt das vier Striche.

Zur Kontrolle: Im Beispiel nach Abbildung 5 müssten beim Hashing 24 Striche gemacht werden. Pro Karte waren also durchschnittlich 2,4 Striche gemacht, also 2,4 Speicherzugriffe notwendig.

Führen Sie diesen Versuch 10- bis 20-mal durch, um zufällige Ergebnisse auszuschließen. Teilen Sie dann die Anzahl der Striche durch die Anzahl der Versuche und dann nochmals durch 10. Auf diese Weise erhalten Sie die mittlere Anzahl von Speicherzugriffen, die pro Karte benötigt wird.



Das Beispiel oben scheint statistisch bereits ziemlich durchschnittlich zu sein, denn wenn man sehr viele Versuche durchführt, kommt man tatsächlich auf ungefähr 2,4 Speicherzugriffe pro Element.

Führen Sie nun den obigen Versuch mit der Strichliste noch ein paarmal durch. Sortieren Sie nun aber nur 9, 8, 7, 6, 5, 4 und 3 Karten auf einmal in den Speicher. Natürlich müssen Sie dann die Anzahl von Strichen am Ende auch durch die Anzahl der Versuche und die Anzahl der Karten (also zum Beispiel 9) teilen, um auf die durchschnittliche Zugriffszahl pro Karte zu kommen.



Wahrscheinlich kommen Sie auf ähnliche Ergebnisse wie ich, die in folgender Tabelle zusammengefasst sind:

Anzahl Karten	3	4	5	6	7	8	9	10
Zugriffe pro Karte	1,1	1,2	1,3	1,4	1,6	1,8	2,0	2,4

Sie konnten also nachvollziehen, dass das Hashing immer besser wird, je weniger Karten man in den Speicher einsortiert. Allerdings sollte es einen guten Kompromiss zwischen Speicherausnutzung und Anzahl der Zugriffe geben, zum Beispiel eine Kartenzahl zwischen 7 und 8.

Das entspricht einem Belegungsgrad von 70 % bis 80 %. Der Belegungsgrad gibt an, wie viel Speicher prozentual ausgenutzt ist.

Nun könnte es ja sein, dass unser kleiner Versuch mit 10 Karten nicht authentisch für große Hashtabellen ist. Daher habe ich für Sie das Experiment schon mit deutlich mehr Karten durchgeführt, die in einen Speicher mit 10.000 Stellen einsortiert werden. Das Ergebnis können Sie in folgender Tabelle ablesen:

Anzahl Karten	Zugriffe pro Karte
10.000	63,1
9.500	10,1
9.000	5,5
8.500	3,8
8.000	3,0
7.500	2,5
7.000	2,2
6.500	1,9
6.000	1,8
5.500	1,6
5.000	1,5
4.500	1,4
4.000	1,3
3.500	1,3
3.000	1,2
2.500	1,2
2.000	1,1
1.500	1,1
1.000	1,1
500	1,0

Belegen wir also die 10.000 Speicherstellen vollständig, benötigen wir pro Suche durchschnittlich etwas über 63 Speicherzugriffe. Zum Vergleich: Die binäre Suche würde in einer sortierten Liste mit durchschnittlich 13 Speicherzugriffen auskommen. Allerdings wird das Hashing bereits attraktiv, wenn wir den Speicher zu 95 % füllen: Bei 9.500 Datensätzen benötigt man nur noch gut 10 Speicherzugriffe, was besser als die binäre Suche ist.

Ein Belegungsgrad von 80 % senkt diese Zahl sogar noch auf durchschnittlich 3. In der Praxis versucht man tatsächlich, den Belegungsgrad beim Hash-Verfahren unter 85 % zu halten. In Programmen, bei denen es auf noch schnelleren Zugriff ankommt, kann man den Speicherplatz auch noch weniger nutzen. So liegt die durchschnittliche Anzahl von Zugriffen nur noch knapp über 1, wenn man nur 20 % des verfügbaren Speichers in Anspruch nimmt.

Was steckt dahinter?

Die wichtigsten Grundsätze des Hashings konnten Sie bereits selbst herausfinden: Es kommt hauptsächlich darauf an, ein Kriterium für die Verteilung der Datensätze auf die Speicherpositionen zu finden, das den Speicherplatz gleichmäßig ausnutzt.

Um hier noch weiter in die Tiefe zu gehen, benötigen wir noch ein paar Fachausdrücke: Den Wert oder die Zeichenfolge, nach der sortiert wird, nennt man auch „Schlüssel“. Im Beispiel oben war der Schlüssel einer Karte also die rote Zahl.

Aus dem Schlüssel wird dann direkt über eine Berechnung die Speicherposition bestimmt. Diese Berechnung war in unseren bisherigen Versuchen immer recht einfach – meistens wurde einfach die erste oder letzte Ziffer genommen. Hier könnte jedoch auch eine komplizierte Formel verwendet werden. Die Berechnungsvorschrift nennt man auch „Hashfunktion“.

Wenn wir den Schlüssel mit „s“ abkürzen und die Hashfunktion mit „h“, ergibt sich also für die Speicherposition eines Datensatzes bzw. einer Karte.

$$\text{Position} = h(s)$$

Im Beispiel nach Abbildung 5 ist die Hashfunktion die letzte Ziffer. Diese ist identisch mit dem Divisionsrest beim Teilen durch 10. Es gilt also

$$h(s) = s \bmod 10$$

Die Karte mit der 425565 kommt also auf die Position

$$h(425565) = 425565 \bmod 10 = 5$$

Ebenfalls im Experiment ausprobiert haben wir, dass die Hashfunktion sowohl zum Einordnen der Datensätze bzw. Karten dient als auch, um einen bestimmten Datensatz wieder aufzufinden.

Wenn wir also irgendwann zum Beispiel wissen möchten, welches Wort auf der Karte mit der roten Zahl 425565 steht, müssen wir die Karte finden. Dazu nutzen wir wieder die Hashfunktion, kommen auf die Speicherposition 5 und können dort das Wort „absurd“ lesen.

Wie ermittelt man nun aber eine sinnvolle Hashfunktion? Ein Kriterium dafür haben wir bisher bereits implizit angewendet. Vom Wertebereich der Hashfunktion ist die Größe des Speichers abhängig, der zur Verfügung gestellt werden muss:

Die Hashfunktion „ $h(s)$ = letzte Ziffer von s “ kann Zahlen zwischen 0 und 9 hervorbringen. Daher muss auch ein Speicherbereich mit 10 Positionen (#00 bis #09) reserviert werden. Für die Hashfunktion „ $h(s)$ = letzte drei Ziffern von s “ müsste entsprechend ein Speicherbereich von 1000 Positionen zur Verfügung stehen, denn drei Ziffern können Werte zwischen 000 und 999 annehmen.

Natürlich müssen Sie dafür sorgen, dass der Werte- und damit der Speicherbereich groß genug ist, um alle Schlüssel aufzunehmen und zusätzlich großzügig Platz zu lassen: Nur bei einem Belegungsfaktor unter 0,8 ist das Hashing effektiv.

Wie erreicht man darüber hinaus, dass die Zahlen über den Speicherstellen möglichst gleich verteilt sind? Natürlich ist das von den jeweiligen Daten abhängig, die man erwartet.

Die folgende Tabelle enthält drei unterschiedliche Sätze von Beispielschlüsseln. Versuchen Sie, für jeden der Sätze eine günstige Hashfunktion für eine Speichergröße von 100 Plätzen zu bestimmen. Kleiner Tipp: Wenn Sie darüber nachdenken, wofür die Zahlen stehen könnten, wird die Auswahl leichter.

Satz 1	Satz 2	Satz 3
19031980	3398776	8,99
21121979	3398777	1,98
09081981	3398778	5,89
06061981	3398779	7,98
14101980	3398780	4,00
29081981	6981154	9,79
11031981	6981155	2,19
17061980	6981156	4,98



Fangen wir einmal mit Satz 2 an: Hierbei könnte es sich zum Beispiel um die Buchnummern von Neuveröffentlichungen eines Verlages handeln. Die einführenden Ziffern stehen dabei etwa für das Fachgebiet und danach werden die Zahlen fortlaufend vergeben. Es scheint vernünftig, die letzten beiden Ziffern als Hashfunktion zu nutzen. Falls dann zufällig zwei Buchreihen die gleichen End-

ziffern haben, kommt es zu recht vielen Kollisionen. Dagegen lässt sich aber auch nicht sonderlich viel machen, auch mit komplizierteren Hashfunktionen nicht.

Satz 1 enthält Zahlen, die zum Beispiel die Geburtsdaten eines Abiturjahrgangs repräsentieren. Man sieht bereits, dass es ziemlich schwierig ist, einfach zwei Ziffern herauszuschneiden, die dann die Speicherstelle angeben:

Die Jahreszahlen beschränken sich auf drei (wie das in einer Schulklasse meistens der Fall ist). Bei den Monaten hat man nur Zahlen zwischen 01 und 12, die Tage liegen zwischen 01 und 31, was auch nicht den gesamten Speicher ausnutzt.

Am ehesten käme man noch auf eine vernünftige Verteilung, wenn man die zweite und die vierte Ziffer zu einer neuen Zahl zusammensetzt, also die Einer-Stelle des Tags und des Monats. Allerdings ist dies auch keine besonders gute Lösung: Die Ziffern 1 und 2 kommen bei der Einer-Stelle des Monats doppelt so häufig vor wie alle anderen Ziffern – sie sind im Januar und Februar, aber auch im November und Dezember enthalten.

Die gleiche Problematik zeigt sich auch bei Satz 3. Die Zahlen könnten zum Beispiel die verschiedenen möglichen Preise eines Einzelhandels darstellen. Da die Cent-Beträge meistens auf 9 oder 8 enden, kann man hier ebenfalls schlecht eine Verteilung bilden.

Allgemein gibt es offenbar ziemlich viele Fälle, in denen die Sortierschlüssel sehr stark von unserem Denken im Dezimalsystem geprägt sind: Bestimmte Ziffern stehen für bestimmte Funktionen (Jahreszahl, Mitgliedsnummer, ...) Daher kommt es sehr oft zu Häufungen bestimmter Werte.

Günstig wäre, quasi gegen das Dezimalsystem arbeiten zu können. Das Gleiche gilt für das binäre System, da viele Größen im Computer hiervon sehr abhängig sind. Um das zu verwirklichen, bietet sich wieder die Divisionsrestmethode an: Verwendet man den Rest bei der Division durch 10, schneidet man quasi nur die letzte Stelle des Dezimalsystems ab, bei der Division durch 100 die letzten beiden Stellen usw.

Genau das Gleiche gilt etwa für das binäre Zahlensystem, wenn Sie durch 2 teilen. Um unabhängig von jeglichen Zahlensystemen zu werden, sollten Sie durch eine Primzahl teilen. Probieren Sie das doch gleich einmal mit den Daten von Satz 1 und Satz 3 aus (das feste Komma der Zahlen in Satz 3 können Sie dabei einfach ignorieren).

Ordnen Sie die Zahlen nach den folgenden Hashfunktionen in den Speicher:

$$h(s) = s \bmod 11$$

$$h(s) = s \bmod 103$$



s	s mod 11	s mod 103
19031980	0	52
21121979	10	78
09081981	7	59
06061981	2	19
14101980	2	44
29081981	5	34
11031981	4	63
17061980	1	30
899	8	75
198	0	95
589	6	74
798	6	77
400	4	91
979	0	52
219	10	13
498	3	86

Die Primzahlen sorgen hier für eine Entkopplung der Speicherpositionen von ihren dezimalen Abhängigkeiten. Selbst sehr ähnliche Schlüssel können komplett unterschiedliche Hashwerte aufweisen. Selbstverständlich wird dann auch die Größe des verwendeten Speichers durch die Primzahl des Divisors bestimmt.

Man tut durch das beschriebene Verfahren das Bestmögliche, um Häufungen zu vermeiden, ausschließen kann man sie jedoch keinesfalls. So wird ein guter Hash nicht nur durch seine Hashfunktion, sondern auch durch die Kollisionsbehandlung bestimmt.

Im Beispiel mit dem Papierspeicher bestand diese einfach in der Regel „gehe so lange eine Position weiter, bis ein Speicherplatz frei wird“. Um das formal auszudrücken, erweitern wir die Hashfunktion. Diese heißt nun nicht mehr $h(s)$, sondern $h_0(s)$. Die 0 steht dabei für den 0. Versuch, den Schlüssel im Speicher unterzubringen (Sie erinnern sich: Informatiker fangen gerne bei 0 an zu zählen ...)

Die Funktion zur Kollisionsbehandlung wird dann mit $h_1(s)$, $h_2(s)$, $h_3(s)$ usw. bezeichnet, wobei 1, 2, 3 für den 1., 2. und 3. Versuch stehen. Meistens möchte man eine einzige Funktion zur Kollisionsbehandlung haben und verwendet eine weitere Variable für den Versuch: $h_i(s)$ steht für den i -ten Versuch. Auf diese Weise kann nacheinander h_0 , h_1 , h_2 usw. durchgeführt werden, bis man eine freie Stelle in der Hashtabelle findet.

Nehmen wir etwa die oben bereits benutzte Hashfunktion (schon in neuer Schreibweise)

$$h_0(s) = s \bmod 103$$

Eine Kollisionsbehandlung, die einfach immer einen Platz weiter sucht, wäre dann

$$h_i(s) = (h_0(s) + i) \bmod 103$$

Zu lesen einfach „für den i -ten Versuch, s unterzubringen, gehe vom Speicherplatz, den die Hashfunktion ausgesucht hat, i Plätze nach rechts, fange nach dem Erreichen des Endes wieder von vorne an“.

Die Suche nach einer vernünftigen Kollisionsbehandlung ist noch schwerer als die Suche nach einer guten Hashfunktion. Die höchste Anforderung ist dabei, dass alle Plätze des Speichers irgendwann „gefunden“, also adressiert werden. Ansonsten könnte es vorkommen, dass erst die Hälfte des Speichers gefüllt ist und ein Schlüssel trotzdem nicht eingeordnet wird, weil die schlechte Kollisionsbehandlung genau die freien Stellen gar nicht adressiert.

Mit der einfachen Kollisionsbehandlung „gehe immer eine feste Anzahl von Speicherpositionen nach links oder rechts“ erfüllt man diese Bedingung im Regelfall. Leider kann das aber zu anderen Problemen führen: Kommt es doch einmal zu Häufungen bei der Hashfunktion, werden die Kollisionen auch immer wieder auf den gleichen Speicherstellen stattfinden. Daher wäre es besser, auch die Kollisionsbehandlung vom Schlüssel abhängig zu machen. Hier ist es jedoch äußerst schwierig, eine Funktion zu finden, die zusätzlich alle Speicherplätze erreicht.

Daher werden in der Praxis tatsächlich fast ausschließlich sehr einfache Hashfunktionen nach der Divisionsrestmethode mit ebenfalls einfachen Kollisionsbehandlungen verwendet. Im Experiment haben wir ja auch gesehen, dass diese im Allgemeinen sehr gut funktionieren.

Ordnung im Chaos!

Um auf die Frage vom Anfang dieses Kapitels zurückzukommen: Was ist denn nun besser – Ordnung oder Chaos? Ich denke, dieses Kapitel hat gezeigt, dass Ordnung ein sehr gutes Mittel ist, effektiv und schnell auf Daten zuzugreifen und sie damit auch effektiv und schnell verarbeiten zu können.

Allerdings hat es ebenfalls gezeigt, dass so manche Anordnung, die auf den ersten Blick wie ein Chaos aussieht, letztlich sehr geordnet sein kann – dem Betrachter fehlt lediglich der Zugang in Form einer mehr oder weniger komplexen Hashfunktion. Das gilt meiner Meinung nach für Daten im Computerspeicher ebenso wie für den Büroschreibtisch: Oft ist der Besitzer eines scheinbar chaotischen Papiergrabs in der Lage, ein gewünschtes Dokument in Sekundenbruchteilen zwischen den Schichten hervorzuzaubern – das Hashing funktioniert.

Schwierig wird es mit diesem Verfahren, wenn viele Menschen zusammen arbeiten und auf den gleichen Datenbestand zugreifen. Meistens gelingt es in der wirklichen Welt nicht, die Hashfunktion für einen Schreibtisch auch anderen zu vermitteln – hier hilft nur traditionelles Sortieren und Ablegen. Im Computer ist das dagegen kein Problem – es können auch mehrere Prozesse auf einen gehashten Speicher zugreifen.

Die Frage ist also weniger, ob Ordnung oder Chaos besser ist, sondern für jede Anwendung – im Computer wie im Alltag – muss betrachtet werden, welche Art der Ordnung sinnvoll und nützlich ist. Manchmal wirkt es dabei ordentlich, manchmal eher unordentlich – aber eigentlich kommt man immer auf eine Ordnung!

Bleibt noch die Frage, welche Anwendungen es für Hashing im Computer gibt. Im Prinzip kann man es überall gut einsetzen, wo man Daten sehr schnell ablegen und wieder hervorholen muss.

Das ist etwa bei Datenbanken der Fall – denken Sie an elektronische Wörterbücher, bei denen ein bestimmtes Wort sehr schnell gefunden werden muss. Hashing wird auch in Betriebssystemen gebraucht, um den verfügbaren Hauptspeicher zu verwalten und einem Programm, das Speicher anfordert, schnell freien Platz zuzuweisen.

Auch in Echtzeitsystemen können bestimmte Formen des Hashings verwendet werden. Echtzeitsysteme sind Anwendungen, bei denen es darauf ankommt, in einer bestimmten, definierten Zeit eine Entscheidung zu treffen. Beispiel dafür ist ein Fließband, auf dem Produkte zur Qualitätskontrolle an Sensoren vorbeilaufen. Innerhalb von Sekundenbruchteilen muss ein Computer anhand der Messdaten entscheiden, ob er das Produkt mit einem Luftstrahl vom Band befördert oder nicht. Hier können Hashverfahren helfen, die darauf getrimmt sind, nicht mehr als eine bestimmte Anzahl von Kollisionen pro Suchvorgang aufzuweisen (z. B. 2).

Zufällig

Am Ende dieses Kapitels möchte ich noch kurz auf ein Thema eingehen, das eng mit dem Hashing verwandt ist, auch wenn das auf den ersten Blick nicht so scheint: Die Erzeugung von Zufallszahlen.

Zufällige Ereignisse im Computer assoziieren wir normalerweise eher mit Computerspielen, in denen der Benutzer immer wieder mit für ihn nicht vorhersehbaren Ereignissen konfrontiert wird. Eine andere Zufallskomponente ist eher unerwünschter Natur: die Zeit, die zwischen Systemabstürzen oder Programmfehlern vergeht (trotzdem werden in England Wetten darauf abgeschlossen ...)

Dass man Zufallszahlen auch nutzen kann, um ganz konkrete Berechnungen anzustellen, soll das folgende Experiment zeigen. Die berühmte Zahl π kann man mit komplizierten mathematischen Reihenentwicklungen bestimmen oder man kann sie anhand ihrer geometrischen Definition ermitteln.

Abbildung 8.7 zeigt einen Kreis mit dem Radius 1. Dieser hat laut Definition einen Flächeninhalt von π , in jedem Viertel also $\pi/4$. Ein solches Viertel ist nochmals sehr groß in Abbildung 8.8 dargestellt.

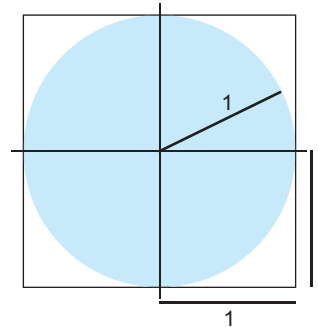


Abbildung 8.7 Der Kreis mit Radius 1 hat den Flächeninhalt π

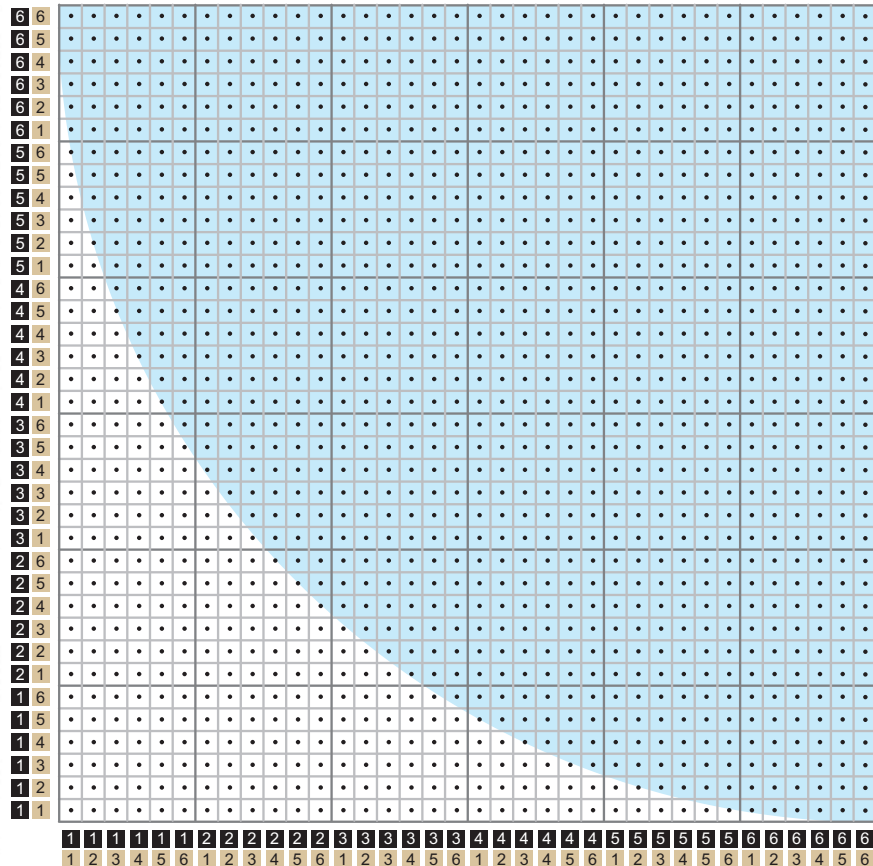


Abbildung 8.8 Ein Viertel des Kreises mit Würfelwerten in Zeilen und Spalten

Dieses ist in kleine Quadrate eingeteilt, mit deren Hilfe wir die genaue Fläche auszählen können. Der Flächeninhalt ist dann näherungsweise

$$\frac{\text{Anzahl der Kästchen mit Mittelpunkt im blauen Breich}}{\text{Gesamte Anzahl Kästchen}}$$

Wenn wir alle Kästchen durchgehen, das sind genau $36^2 = 1296$ Stück, kommen wir auf 1018 mit Mittelpunkt im blauen Bereich (bei einigen muss man sehr genau hinschauen, wo der Mittelpunkt nun ist). Der Flächeninhalt ist also ungefähr

$$\frac{1018}{1296} = 0,7855$$

Multiplizieren wir dies mit 4, kommt 3,1420 heraus, was bereits ein ziemlich guter Näherungswert für die Zahl π ist – das „echte“ π ist auf fünf Stellen genau 3,1416. Um diesen guten Wert zu erhalten, mussten jedoch 1296 Kästchen ausgezählt werden.

Mit deutlich weniger Zählen kommen Sie aus, wenn Sie nicht jedes Kästchen betrachten, sondern nur solche, die Sie per Würfel zufällig bestimmt haben. Hierzu benötigen Sie zwei unterschiedliche Würfel, zum Beispiel einen schwarzen und einen beigen. Um ein Kästchen zu bestimmen, würfeln Sie zunächst für die Zeile und dann für die Spalte. Falls der Mittelpunkt im blauen Bereich ist, vermerken Sie eine 1, sonst eine 0. Wenn Sie 50 Kästchen ausgezählt haben, ziehen Sie eine Zwischenbilanz.

Teilen Sie die Anzahl der Einsen durch die Gesamtanzahl, also durch 50. Haben Sie zum Beispiel 40 Einsen erwürfelt, ergibt das für die Fläche einen Näherungswert von $\frac{40}{50} = 0,8$. Mit 4 multipliziert kommen für π immerhin schon ziemlich gute 3,2 heraus.

Jetzt wenden Sie sicherlich ein, dass es deutlich mehr Aufwand bedeutet, 50 Kästchen auszuwürfeln, als schnell alle Kästchen durchzuzählen. Nicht so für einen Computer – der bestimmt die nötigen Zufallszahlen im Handumdrehen. Tatsächlich wirkt sich das besonders bei Anwendungen mit sehr vielen Dimensionen aus. Bei der Kugel zeigt sich der Effekt also noch besser als beim Kreis. Auch bei einer „Hypersphäre“ mit 100 Dimensionen kann der Rauminhalt am genauesten bestimmt werden, indem man 10.000-mal würfelt und sozusagen Stichproben nimmt, wo es die Würfelergebnisse anzeigen.

Sie fragen sich wahrscheinlich, wozu man so etwas überhaupt berechnen möchte. Zu Recht! Selbstverständlich ist das – genau wie die Berechnung von π oben – ein rein akademisches Beispiel. Denken Sie aber etwa an die Entwicklung einer neuen Fahrzeugkarosserie, bei der 100 verschiedene Werte computergestützt fein justiert und optimiert werden müssen. Keine Maschine der Welt ist in der Lage, alle möglichen Kombinationen durchzuprobieren. Durch die Verwendung von Zufallszahlen kann man sich aber schnell dem optimalen Ergebnis annähern.

Voraussetzung für das Gelingen des Experiments oben sind Würfel, die tatsächlich zufällige Ergebnisse produzieren und nicht etwa durch Herstellungsfehler eine Zahl häufiger zeigen als eine andere.

Auch sehr gut müssen die „Computer-Würfel“ sein, genannt Zufallszahlengeneratoren, besser noch Pseudozufallszahlengeneratoren, denn eines können Computer im Normalfall nicht liefern: Zufallszahlen!

Woran liegt das?

Computer sind Maschinen, deren Hauptzweck darin besteht, Abläufe immer und immer wieder so zu wiederholen, wie sie einmal von den Programmierern vorgegeben wurden. Hier wäre es fatal, wenn der gleiche Algorithmus mit den gleichen Eingabewerten zu unterschiedlichen Zeiten unterschiedliche Ergebnisse hervorbrächte. Computer sind in dieser Hinsicht jedoch sehr zuverlässig!

Zufall ist jedoch ein anderes Konzept! Zufall ist sozusagen kultivierte Unzuverlässigkeit und das widerspricht dem braven Computer (daher bringt er ja beim Hashverfahren auch kein echtes Chaos zustande, sondern nur eine ganz besondere Ordnung).

Aus diesem Grund simulieren Computer den Zufall nur – sie müssen alle Zufallszahlen berechnen und weil diese dann lediglich zufällig aussehen, nennt man sie Pseudozufallszahlen. Da dies jedoch ein sehr langes Wort ist, werde ich im Folgenden einfach meistens weiter von Zufallszahlen reden.

Im Normalfall werden solche Pseudozufallszahlen rekursiv bestimmt: Man fängt mit einer festen Zahl als „Zufallszahl 0“ an und berechnet jeweils die folgende Zufallszahl anhand einer Funktion aus der letzten Zufallszahl.

Sehr einfach wäre zum Beispiel eine Funktion wie

$$z(i) = (z(i-1) + 5) \bmod 11$$

Die resultierende Zufallszahlenfolge mit „1“ als Zufallszahl 0 wäre dann

1, 6, 0, 5, 10, 4, 9, 3, 8, ...

Auf den ersten Blick sieht das bereits zufällig aus, aber bei genauerem Hinsehen fällt dann der Summand 5 sehr deutlich auf, wenn man aufeinander folgende Zahlen betrachtet. Die folgende Zahlenreihe wurde nach einem sehr einfachen und in den 1980er-Jahren sehr häufig eingesetzten Verfahren berechnet. John von Neumann hat es bereits 1949 vorgestellt. Als Startwert habe ich 42 genommen. Kommen Sie auf die Berechnungsvorschrift?

42 - 76 - 77 - 92 - 46 - 11 - 12 - 14 - 19 - 36 - 29 - 84 - 5 - 2



Es handelt sich um den so genannten „Mid-Square-Generator“: Man nehme die Zahl, quadriere sie (multipliziere sie mit sich selbst) und benutze die mittleren Ziffern als neue Zahl.

Beispiel: $42 \times 42 = 1764$. Die mittleren Ziffern sind 76, das ist die nächste Zufallszahl.

Dieser Generator ist jedoch in der Praxis nicht besonders gut geeignet. Berechnen Sie zum Beispiel einmal die Zufallszahlenfolge mit 24 als Startwert.

Es ergibt sich eine sehr kurze Folge 24 - 57 - 24 - 57 - usw. Prinzipiell kommen also nur zwei Zahlen abwechselnd heraus. Auch mit vielen anderen Startwerten kommt man nur auf sehr kurze Folgen.

Für ein computergestütztes Roulette-Spiel wäre dieser Generator daher sicherlich nicht geeignet, denn es würden immer nur einige Zahlen erscheinen und andere gar nicht. Eine wichtige Anforderung an einen Zufallszahlengenerator ist also, dass er irgendwann einmal alle Zahlen hervorbringt! Auch im Beispiel der π -Berechnung von oben wäre das Ergebnis nicht akkurat, wenn bestimmte Zeilen oder Spalten ganz ausgespart würden, weil die Würfel die entsprechenden Zahlen nie zeigten.

Das erinnert doch sehr stark an die Forderungen beim Hashing: Auch dort sollte möglichst der gesamte Speicher ausgenutzt werden. Die entsprechende Funktion war sehr einfach:

$$h(s) = s \bmod m$$

Diese Funktion als Zufallszahlengenerator würde direkt noch keine gute Folge ergeben:

$$z(i) = z(i-1) \bmod 40$$

Hier kommt immer wieder die Zahl heraus, die man als Startwert festlegt – probieren Sie es selbst einmal aus! Wir benötigen noch etwas, das diesen Startwert verändert. Daher multiplizieren wir ihn mit einem festen Faktor und addieren dann noch etwas dazu:

$$z(i) = (z(i-1) \cdot 21 + 17) \bmod 40$$

Mit 1 als Startwert erhalten wir

1 - 38 - 15 - 12 - 29 - 26 - 3 - 0 - 17 - 14 - 31 - 28 - 5 - 2 - 19 - 16 - 33 - 30 - 7 - 4 - 21 - 18 - 35 - 32 - 9 - 6 - 23 - 20 - 37 - 34 - 11 - 8 - 25 - 22 - 39 - 36 - 13 - 10 - 27 - 24 - 1

Der Generator liefert also alle 40 Möglichkeiten, bevor er von Neuem mit der gleichen Zahl anfängt. Und die resultierende Zahlenfolge sieht trotz der einfachen Berechnungsvorschrift sehr zufällig aus. Es gibt auch leichte mathematische Regeln, wie man bei einem solchen so genannten linearen Kongruenzgenerator die Werte so setzt, dass tatsächlich alle möglichen Zahlen durchlaufen werden.

Echt zufällig

In einigen Anwendungen reichen Pseudozufallszahlen nicht aus. Denken Sie zum Beispiel an die Liste von Transaktionsnummern, die Sie von Ihrer Bank bekommen. Kommt diese aus einem Pseudozufallszahlengenerator und bekommt jemand die Funktionsweise heraus, kann er aus einer Hand voll Eingaben alle weiteren geheimen Zahlen selbst ausrechnen. Für solche Anwendungen sollte man daher auf „echte“, also physikalische Zufallszahlengeneratoren zurückgreifen. Die schnellsten nutzen quantenphysikalische Vorgänge aus.

Sie benötigen gar keine Zufallszahl zwischen 0 und 39, sondern nur eine zwischen 0 und 9? Gar kein Problem: Nehmen Sie einfach nur die hinterste Ziffer. Auf diese Weise entstehen aus den großen Zufallszahlen, die von den Standard-Generatoren der Computer erzeugt werden, die kleineren Zufallszahlen, die man für eine Anwendung benötigt – etwa ein Spiel. Die meisten der „eingebauten“ Generatoren arbeiten mit Zahlen zwischen 0 und 18.446.744.073.709.551.615 (über 18 Trilliarden). Hier dauert es also eine Weile, bis der Zyklus von Neuem beginnt.

Vielleicht fragen Sie nun, wie es kommt, dass im bevorzugten Skat-Programm doch nach jedem Start eine unterschiedliche Hand ausgeteilt wird, das Computer-Schachspiel meistens einen anderen Eröffnungszug vollzieht. Das passt doch nicht zusammen mit dem beschriebenen vorbestimmten Zufall. Und Sie haben Recht! Der Computer verwendet zur ersten Initialisierung seines Zufallszahlengenerators eine „echte“ Zufallszahl. Meistens nimmt er hierfür die Zeit, die seit dem Einschalten des Computers vergangen ist, in Millisekunden. Da diese Zeit maßgeblich nicht von der Technik bestimmt wird, sondern vom Bediener, ist sie mit einer starken Zufallskomponente behaftet – je nachdem, ob Sie sich vor dem Start des Spiels noch am Kopf gekratzt haben und wie lange dies gedauert hat, wird der Zufallszahlengenerator anders initialisiert.

Warum generiert der Computer danach in der Regel nur noch Pseudozufallszahlen, statt echter Zufallszahlen auf Zeit-Basis? Sobald das Spiel gestartet wurde und der Benutzer nicht eingreift, benötigt der Computer normalerweise immer die gleiche Zeitspanne für die gleichen Programmteile. Auf diese Weise ist etwa beim Skatprogramm die Zeit zwischen dem Austeilen der ersten Karte und der zweiten Karte sehr genau definiert. Hier liefern daher die Pseudozufallszahlen-Generatoren wesentlich bessere Ergebnisse.

Sie sehen: Selbst bei einem Thema, das eigentlich per Definition chaotisch ist, nämlich Zufallszahlen, bleibt der Computer ordentlich und vorausschaubar. Das Chaos ist auch hier – genau wie beim Hashing – lediglich ein Pseudo-Chaos...



Viel weiter möchte ich hier in die Thematik auch gar nicht einsteigen. Interessierte seien auf die angegebenen Bücher zu dem Thema verwiesen.

Resümee

Die Ausgangsfrage für dieses Kapitel „Ordnung oder Chaos“ konnte – zumindest im Sinne der Informatik – ganz eindeutig für die Ordnung beantwortet werden! Allerdings hat der Leser jetzt auch jede Möglichkeit, das scheinbare Chaos auf seinem Schreibtisch einfach als Ordnung zu deklarieren – genau wie in modernen Computern scheinbar chaotisch angeordnete Daten eine der schnellsten Zugriffsmethoden darstellen: das Hashverfahren. Und wie scheinbar chaotische Zufallszahlen eigentlich doch ganz ordentlich entstehen ...

Sie konnten jedoch auch nachvollziehen, dass Informatik nicht immer nur eine Wissenschaft ist, die ihre Methoden durch konstruktive Herangehensweisen bildet. In einigen Fällen muss hier auch empirisch gearbeitet werden. Im Klartext heißt das Ausprobieren! So bleibt neben allen formalen Kriterien für die Formulierung einer guten Hashfunktion als letzte Sicherheit nur das Ausprobieren.

Literatur

Artikel: Der Spiegel, Heft 42/1992 „**Konisch vermüllt**“



Artikel: Handelsblatt vom 11. Mai 2004 „**Das Chaos managen wie ein Feldwebel auf dem Schlachtfeld**“.

Beide Artikel beschäftigen sich mit dem Pro und Contra von Ordnung und Chaos auf eine sehr amüsante Art und Weise.

Buch: Donald E. Knuth „**The Art of Computer Programming 1–3**“; Addison-Wesley Professional; 1998; ISBN 0-20148541-9



Eines der Standardwerke für Computerprogrammierung. Unter anderem beschäftigt sich der Autor im zweiten Band auch mit Pseudozufallszahlen. Das Englisch ist auch mit Sprachkenntnissen aus der Schule gut lesbar und einprägsam geschrieben.

Konto

Geschäftsbank Giro minus, 1002499

Zeitraum



10 Tage



von

01

03

TT MM JJJJ

Aktueller Kontostand:

Dispolimit:

Summe vorgemerakter Umsätze:

Umsatzauskunft aktualisieren

Bereits gebuchte Umsätze

Datum	Wertstellung	Art	Buchungshinweis	Betrag (€)	Sal
09.03.	09.03.	Auszahlung	PGA 39824471 KRT0001/10.07 09.03 16.30 002245 53291SCHILDA HAUPTPOST EC-CARD MIT PIN	-250,00	1.1
08.03.	08.03.	Lastschrift	ZAHLUNGSBELEG 483728882701 RE.KTO.NR. 983664729810 RE.NR.000049931122/27.02. Telefon AG & CO. KG	-24,72	1.3
08.03.	08.03.	Überweisung	4711 / 08150815 Tino Tümmeler	-137,13	14
07.03.	07.03.	Gutschrift	EBay - Artikel Buch Abenteuer Informatik	38,75	15
06.03.	06.03.	Gutschrift	Gehalt März GaZ Kassel	812,20	15
06.03.	06.03.	Gutschrift	EBAY PLEXTOR WERNER UND KARIN DEINUADO	18,33	34.4

9. Mit Sicherheit

Die Zeiten, in denen man einer lächelnden Dame hinter zentimeterdickem Panzerglas sein Ersparnis anvertraute oder etwas Geld für den nächsten Einkauf abholte, sind wohl endgültig vorbei. Der moderne Bankschalter von heute ist der heimische PC, eine Internet-Verbindung erlaubt den schnellen Zugriff auf die eigenen Finanzen.

Was sich jedoch nicht verändert hat, ist das Bestreben einiger nicht so gesetzestreuer Elemente, die Eigentumsverhältnisse zu ihren Gunsten zu manipulieren. Es hat sich inzwischen herumgesprochen, dass der klassische Bankraub mit vorgehaltener Waffe unrentabel ist. Daher versuchen es immer mehr Gauner auf elektronische Art und Weise.

Vielleicht haben Sie sich immer schon einmal gefragt, welche Methoden hier genutzt werden und vor allem, welche Möglichkeiten zum Schutz der Daten (und des Geldes) existieren. Hier können Sie das sogar experimentell erforschen!

Um dieses Kapitel komplett verstehen zu können, wäre es übrigens günstig, vorher das Kapitel „Paketpost“ durchzuarbeiten.

Von Griechen, Julius Cäsar und anderen

Was ist E-Banking anderes als eine Art der Nachrichtenübermittlung von Ihrem Geldinstitut an Sie und umgekehrt? Diese Nachrichten bestehen aus Kontoauszügen, Überweisungsaufträgen, Wertpapierordern usw. Sicheres E-Banking bedeutet dabei weitgehend, dass die Nachrichten unverseht beim vorgesehenen Empfänger ankommen und dass sie auch nirgendwo außer beim vorgesehenen Empfänger ankommen.

Dieses Problem wiederum ist nicht neu – bereits einige Jahrhunderte vor unserer Zeitrechnung widmete man sich der Thematik: Besonders auf Feldzügen und bei anderen kriegerischen Auseinandersetzungen war die Gefahr immer groß, dass eine Nachricht vom Gegner abgefangen, verfälscht oder ausspioniert wurde.

Die Spartaner nutzen daher die so genannten Skytalen, um wichtige Befehle zu verschlüsseln. Diese kommen immer paarweise, ein Exemplar ist für jeden Kommunikationspartner bestimmt. Es handelt sich eigentlich nur um einfache, runde

Holzstäbe mit exakt identischen Durchmessern. Ein dünner Streifen Papyrus oder Pergament wird nun spiralförmig um eine Skytale gewickelt und so beschriftet, dass auf jeder Windung nur ein einzelner Buchstabe steht. Abbildung 9.1 zeigt dies – der besseren Verständlichkeit halber mit lateinischen statt griechischen Buchstaben. Am Rand sehen Sie den resultierenden Streifen mit dem Geheimtext. Er ist nur wieder lesbar, wenn man ihn um eine gleichartige Skytale wickelt.

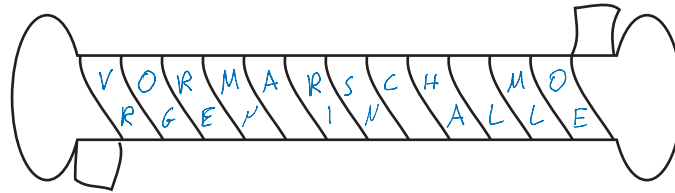


Abbildung 9.1 Schema-Zeichnung einer Skytale

Die Skytale war damit sozusagen die erste Chiffriermaschine der Welt. Sie erzeugte einen Vertauschungs-Code oder auch eine Transpositions-Chiffre. So werden alle Geheimbotschaften bezeichnet, die genau dieselben Zeichen enthalten wie der Klartext, allerdings in einer veränderten Reihenfolge.

Versuchen Sie den Streifen mit der Geheimbotschaft doch einmal ohne entsprechende Skytale zu lesen, indem Sie einfach immer drei Zeichen überspringen. Achtung! Leerzeichen müssen hier berücksichtigt werden!

Transpositions-Chiffren sind recht einfach zu knacken und werden heute allenfalls noch in Rätselbüchern benutzt, wo dann der Leser zum Beispiel aufgefordert wird, in „ATLAS NACHSEHN“ einen geographischen Begriff zu finden (Lösung: „SACHSEN ANHALT“).

Bereits Julius Cäsar beschreibt in seinem berühmten Werk „Commentarii de bello gallico“ („Anmerkungen zum gallischen Krieg“) ein anderes Verschlüsselungsverfahren: Jeder Buchstabe des Klartextes wird durch einen anderen Buchstaben ersetzt, der im Alphabet eine entsprechende Zahl von Positionen später kommt. Ist man am Ende des Alphabetes angelangt, fängt man wieder von vorne an.

Es handelt sich um einen Ersetzungscode oder auch Substitutions-Chiffre, da die Zeichen gegen andere ausgetauscht werden, die Position im Text jedoch gleich bleibt. Den berühmten Cäsar-Code können Sie sehr einfach verwenden, wenn Sie die Cäsar-Scheiben aus dem Anhang basteln. Schneiden Sie hierfür die kleinere und die größere Scheibe mit blauen und roten Buchstaben aus. Sie können die Scheiben verbinden, indem Sie in der Mitte entweder mit einer sehr feinen Schere je ein Loch schneiden und einen Verschluss für Versandtaschen durch beide Löcher stecken. Eine sehr gute Methode ist auch, einen Reißnagel genau in die Mitte zu pieksen und diesen hinten mit einer Scheibe zu sichern, die Sie von einem gewöhnlichen Weinkorken abschneiden.

Abbildung 9.2
Codestreifen aus der
Skytale

V
O
R
M
A
R
S
C
H
A
L
L
E

R
E
X
S
M
U
R
O
A
G
U
R
I
E
N
S
H
E
C
F
U
X
A
L
A
F
O
C
A

Nun können Sie mit dem grünen Pfeil einen von 26 Schlüsseln auswählen. Für jeden Buchstaben einer Nachricht schreiben Sie den entsprechenden Codebuchstaben hin, der nun in rot direkt über dem blauen Klartextbuchstaben steht.

Versuchen Sie es: Codieren Sie die Nachricht „ABENTEUER INFORMATIK“ mit der Cäsar-Scheibe und dem Schlüssel „R“.



Die Scheibe wird so eingestellt, dass der grüne Pfeil auf dem roten R steht, wie in Abbildung 9.3 dargestellt. Danach wird Buchstabe für Buchstabe im blauen Bereich gesucht und durch den roten Buchstaben darüber ersetzt. Die Geheimbotschaft lautet also

„RSVEKVLVI ZEWFIDRKZB“



Abbildung 9.3 Die Cäsar-Scheibe mit gewähltem Schlüssel R

Sie werden mir zustimmen, dass es schwierig ist, diese Botschaft zu verstehen, wenn man nicht im Besitz des korrekten Schlüssels ist. Wenn Sie sich nun mit einem Freund vertraulich austauschen wollen, machen Sie einen Schlüssel aus und verwenden Sie die Cäsar-Scheibe, um die Nachrichten vor unerwünschten Augen und Ohren zu schützen.

Nehmen wir an, Sie haben den Schlüssel „N“ ausgemacht. Was schließen Sie aus der folgenden Nachricht?

„NORAQRFFRA HZ NPUG ORV ZVE“



Hacker

Laut Brockhaus sind dies Personen, die in der Lage sind, beliebige Schutzmaßnahmen gegen das Ausspähen und Kopieren von Daten in Netzwerken oder auf Datenträgern zu umgehen. Der Begriff geht zurück auf das intensive Tippen („Hacken“) auf der Tastatur, durch das die ersten Hacker im Zufallsverfahren versuchten, Passworte herauszufinden.

Selbstverständlich konnten Sie Ihre Scheibe auf den Schlüssel „N“ einstellen. Nun mussten die Buchstaben im roten Ring gesucht und durch den entsprechenden blauen ersetzt werden. Es handelt sich um eine Einladung zum Abendessen.

Viele werden jetzt fragen, ob diese Kommunikation auch wirklich sicher ist. Tatsächlich beschäftigt sich ständig eine große Anzahl von Experten nicht mit der Erschaffung neuer Verschlüsselungsverfahren, sondern damit, die vorhandenen irgendwie zu brechen und auf diese Weise Nachrichten zu lesen, die für jemand anderen bestimmt waren.

Als „Täter“ fallen einem da sofort Geheimdienste, Detekteien, aber auch Betrüger, Industriespione usw. ein. Diese gehören jedoch zur absoluten Minderheit. Die meisten Hacker sitzen in den Universitäten und Entwicklungsabteilungen spezialisierter Firmen. Sie haben ganz sicher Spaß an ihrer Tätigkeit, aber trotzdem keinerlei kriminelle Absichten! Vielmehr dienen ihre Attacken dazu, die Systeme sicherer zu machen und besser vor unberechtigtem Zugriff zu schützen. Die einzige Chance, „echten“ Hackern immer einen Schritt voraus zu sein, besteht darin, die Sicherheitslücken vor ihnen zu finden!

Codebrecher

Jetzt dürfen Sie auch einmal Hacker spielen. Ihre Spitzel fangen folgende verschlüsselte Nachricht ab

„TXC KPITG WPIIT OLTX HDTWCT SPKDC LPG STG PTAITHIT ZAJV JCS VTHRWTXI JCS LJHHITHXRW XC PAATH LDWA OJ HRWXRZTC STG YJTCVHIT PQTG LPG SJBB ZDCCIT CXRWIH QTVGTXUTC JCS ATGCTC JCS LTCC XWC SXT ATJIT HPWTC HEGPRWTC HXT BXI STB LXGS STG KPITG CDRW HTXCT APhi WPQTC LTCC CJC TILPH OJ IJC LPG HD BJHHIT TH STG PTAITHIT PAAOTXIPJHGXRWITC WXTTH XWC PQTG STG KPITG CDRW HEPTI DSTG VPG XC STG CPRWI TILPH WDATC JCS STG LTV VXCXV SPQTX JTQTG STC ZXGRWWDU DSTG HDCHI TXCTC HRWPJGXVTC DGI HD PCILDGITIT TG LDWA PRW CTXC KPITG XRW VTWT CXRWI SPWXC TH VGJHTAI BXG STCC TG UJTGRWITIT HXRW“

Eine Möglichkeit wäre jetzt natürlich, nacheinander alle 26 Schlüssel des Cäsar-Codes auszuprobieren, bis einer die Nachricht decodiert, aber es gibt noch eine gezieltere Methode: Wie Sie im Kapitel über Datenkomprimierung sehen konnten, kommen einzelne Buchstaben in unserer Sprache sehr unterschiedlich häufig vor. Diesen Umstand nutzen Kryptographen auf der ganzen Welt, um den benutzten Schlüssel in einer Geheimbotschaft zu knacken.

Ich habe für Sie daher einen „Apparat“ vorbereitet, mit dem die Entschlüsselung jedes Cäsar-Codes zum Kinderspiel wird.

Schneiden Sie hierzu die großen Scheiben im Anhang aus und setzen Sie sie zusammen, wie bereits die kleinen Cäsar-Scheiben. Auf die größere Scheibe müssen Sie etwas malen. Wenn Sie möchten, können Sie diese daher mit einer Folie überziehen oder bekleben und sie dann mit wasserlöslichen Folienschreibern mehrfach benutzen.

Das Prinzip des Code-Knackens besteht darin, die Häufigkeiten einzelner Buchstaben in der Geheimbotschaft zu ermitteln und dann mit den Häufigkeiten zu vergleichen, die die Buchstaben in unserer Sprache normalerweise haben. Zunächst müssen also die Häufigkeiten in der Geheimbotschaft bestimmt werden. Dazu sind die Rechtecke ganz außen auf der Scheibe da.

Gehen Sie die Geheimbotschaft oben Buchstabe für Buchstabe durch und füllen Sie für jeden Buchstaben von innen nach außen ein kleines Kästchen aus und zwar oberhalb des entsprechenden roten Buchstabens auf dem äußeren Ring der Scheibe. Machen Sie dies so lange, bis eines der Rechtecke komplett ausgefüllt ist (50 Kästchen).



Die so entstandenen Balken geben nun die Häufigkeit jedes Buchstaben im Geheimtext an. Sie müssen lediglich noch mit den Häufigkeiten der Buchstaben übereingebracht werden, wie sie tatsächlich in unserer Sprache vorkommen. Diese Häufigkeiten sind bereits fest vorgegeben durch die blauen Balken auf der inneren Scheibe.

Drehen Sie die Scheiben so gegeneinander, dass die Balken innen und außen ungefähr übereinstimmen! Hierfür bitte nur Augenmaß verwenden, kleine Differenzen wird es immer geben!



Kryptographie

Kryptographie ist die Wissenschaft der Datenverschlüsselung. Kryptographen finden einerseits neue Methoden der Codierung, andererseits werden sie auch eingesetzt, um vorhandene Verfahren zu brechen.

Die Balken stimmen dann ziemlich gut überein, wenn der grüne Pfeil auf dem „P“ steht, wie Sie auch in Abbildung 9.4 sehen können. Dies war offensichtlich der verwendete Schlüssel. Wenn Sie nun den Text decodieren, kommt der Anfang eines Grimm’schen Märchens zum Vorschein!

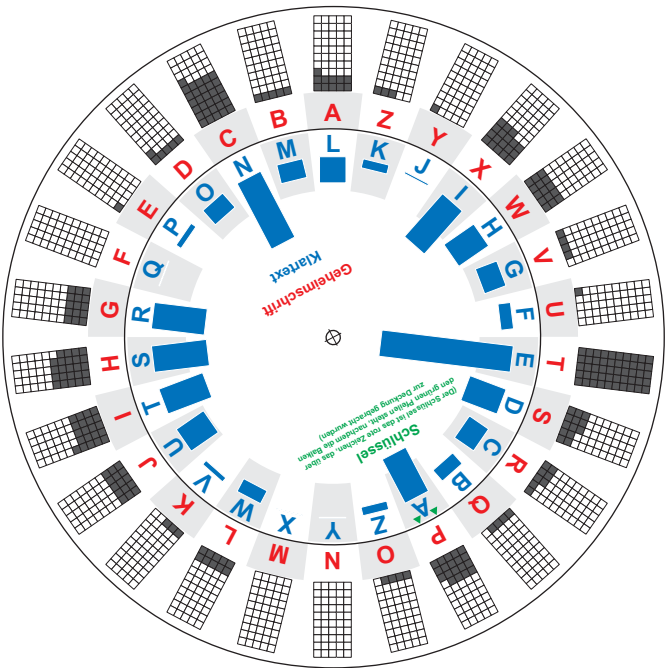


Abbildung 9.4 Nur auf einer Position der großen Cäsar-Scheibe passen die Balken ungefähr zueinander

Tatsächlich beruhen sehr viele Verfahren zur Entschlüsselung fremder Geheimbotschaften auf einer statistischen Analyse. Selbstverständlich ist der Cäsar-Code sehr einfach und daher auch sehr einfach zu knacken! Sehr viel schwieriger wird es bereits, wenn die Buchstaben beliebig gegeneinander vertauscht werden dürfen, also ein Schlüssel zum Beispiel so aussieht:

Klartext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Code	J	P	Q	A	H	Y	T	S	B	N	W	E	Z	U	C	Y	G	X	F	I	L	R	K	O	D	M

Auch hier ist allerdings meistens das „E“ anhand seiner herausragenden Häufigkeit sofort zu identifizieren. Hierbei kann man sich behelfen, indem man für häufige Buchstaben in Klartext mehrere Codes verwendet. Eine weitere, sehr erfolgreiche Methode ist, die Daten vor der Verschlüsselung mit dem Huffman’schen Verfahren aus dem Kapitel „Datenpresse“ zu komprimieren. Dadurch sind die Häufigkeiten in etwa gleich verteilt und sogar ein recht einfacher Cäsar-Code reicht aus, eine recht gute Verschlüsselung zu garantieren.

Computer sind natürlich in Dingen wie Code-Knacken, das oft stures Ausprobieren von verschiedenen Schlüsseln beinhaltet, deutlich schneller als ihre menschlichen Pendanten. Andererseits kann man mit Hilfe von Computern auch deutlich komplexere Verschlüsselungen umsetzen. Stellen Sie sich zum Beispiel eine Cäsar-Codescheibe vor, bei der nicht ein einzelner Buchstabe, sondern immer die Kombination mehrerer Buchstaben einem individuellen Code zugeordnet werden (z. B. „ABC“ → „RDW“, „ABD“ → „FLQ“ usw.). Wenn die Anzahl von Buchstaben pro Code-Paket groß genug ist und der Klartext vorher mit der Methode von Huffman komprimiert wurde, ist selbst ein einfacher Code wie der von Cäsar entwickelte sehr effektiv und sicher.

Es wäre jetzt allerdings müßig, für unsere Experimente auf solche „echt sicheren“ Verschlüsselungen zurückzugreifen! Stellen Sie sich vor, sie müssten eine Cäsar-Scheibe mit 950163143821 Metern Durchmesser basteln – und das wäre die Größe, wenn man Blocks zu nur je 10 Buchstaben verschlüsselte ... Übrigens: 950163143821 Meter entsprechen ungefähr dem 10.000sten Teil eines Lichtjahres, also etwas mehr als dem mittleren Abstand zwischen der Sonne und dem Jupiter.

Für unsere Experimente erlauben wir uns daher, nicht auf Details wie „muss wirklich wasserdicht sein“ zu achten. Vielmehr kommt es darauf an, die tatsächlichen Herausforderungen für eine sichere Verschlüsselung im Internet kennen zu lernen. Diese geht deutlich über das Finden von statistisch nicht knackbaren Verfahren hinaus! Um das jedoch mit einfachen Mitteln nachzuvollziehen, gelte für die folgenden Experimente die nachstehende Aussage als gegeben:

„Die Cäsar-Scheibe ist ein hinreichend sicheres Verschlüsselungsverfahren!“

Methode von Huffman

Dieses Verfahren wird übrigens im Kapitel „Von Kamelen und dem Nadelöhr“ beschrieben.

Internet-Spione

Fassen wir noch einmal zusammen: Wenn Sie mit einem Freund sicher kommunizieren möchten, treffen Sie sich irgendwann einfach persönlich und machen einen Schlüssel aus, zum Beispiel „X“. Auf diesen Schlüssel stellt dann jeder seine Cäsar-Scheibe ein. Wenn Sie sich nun per Post Nachrichten schicken, kann der Bote

- diese Nachrichten nicht lesen
- diese Nachrichten nicht verfälschen (denn dazu müsste er sie ja erst einmal lesen)
- nicht eine völlig andere Nachricht schicken und so tun, als wäre sie von Ihnen (denn dazu müsste er sie korrekt verschlüsseln und das kann er ohne Schlüssel nicht)

Ich versichere Ihnen – mit modernen Verschlüsselungsverfahren unter Zuhilfenahme von Computern kann man dies auch tatsächlich gewährleisten!

Wo liegt nun aber das Problem?

Maria Stuart hatte bereits vorgesorgt, als sie um 1580 in einem englischen Kerker eingesperrt war: Sie hatte mit ihren Verbündeten einen Geheimcode ausgemacht! Allerdings handelte es sich um einen einfachen Substitutionscode, der einfach Buchstaben durch besondere Zeichen ersetzte. Unten sehen Sie den Namen „Stuart“ in dieser Geheimschrift. Die meisten der Botschaften wurden wohl trotz Codierung abgefangen...

b+LbR+

Das Szenario oben trifft schlicht und einfach auf die meisten Kommunikationen im Internet nicht zu! Stellen Sie sich vor, Sie möchten ein Buch beim neuen Online-Buchversender „Aragon“ bestellen. Dieser weiß bisher nichts von Ihnen und Sie wussten bisher nichts von ihm! Jetzt haben Sie allerdings das Buch „Abenteuer Informatik“ in seinem Angebot gesehen und wollen es sofort bestellen. Hierzu müssen Sie Ihre Kreditkartennummer übermitteln, wollen jedoch nicht, dass diese von irgendeinem Schlitzohr ausspioniert wird.

Wenn wir nach dem obigen Schema vorgehen wollten, müssten Sie sich nun erst einmal von Angesicht zu Angesicht mit einem Aragon-Vertreter treffen und einen Schlüssel übergeben. Warum?

Stellen Sie sich vor, Sie schicken den Schlüssel ebenfalls über das Internet. Diese Nachricht (mit dem Schlüssel) muss noch ohne Schutz gesendet werden, denn Sie haben ja noch keinen Schlüssel ausgetauscht. Dann kann ein Schlitzohr, das Ihre Leitung angezapft hat, zunächst den Schlüssel aufzeichnen und dann in aller Ruhe die verschlüsselten Nachrichten (mit der Kreditkartennummer) decodieren. In diesem Fall ist die Verschlüsselung vergeblich!

Tatsächlich hat man in frühen E-Banking-Varianten von der entsprechenden Bank zunächst per Einschreiben eine Diskette mit einem Schlüssel geschickt bekommen. Erst nach der Installation dieses Schlüssels konnte dann sicher mit der Bank kommuniziert werden.

Überlegen Sie, ob dieses Verfahren im Fall des Online-Buchkaufs praktikabel wäre.



Sicherlich nicht: Einerseits wäre der regelmäßige Post-Versand von Schlüsseln an jeden Kunden sehr teuer. Noch viel wesentlicher ist jedoch: Wenn Sie im Internet etwas sehen und es kaufen möchten, dann wollen Sie sicherlich nicht erst warten, bis Sie vom Versender ein paar Tage später eine Diskette per Post erhalten, nur um dann die eigentliche Bestellung erst abzuschicken. In dieser Zeit kaufen Sie das Buch sicherlich eher im kleinen Geschäft um die Ecke, der Internet-Versender macht dann keinen Umsatz.

Es sollte also die Möglichkeit geben, von Anfang an mit einem vorher unbekannten Partner sicher zu kommunizieren.

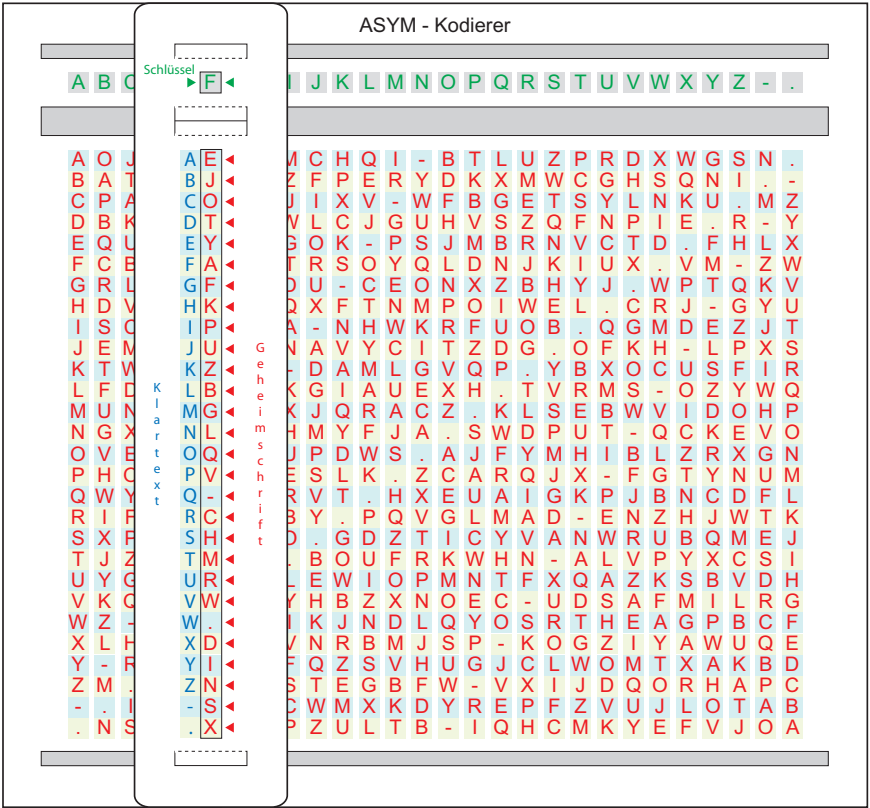
Whitfield Diffie, Martin Hellman und Ralph Merkle haben sich Anfang der 1970er Jahre mit dieser Problematik beschäftigt. Sie erkannten, dass das Hauptproblem darin besteht, dass ein und derselbe Schlüssel gleichzeitig für die Ver- und die Entschlüsselung verwendet wird.

Beispiel: Wenn Sie mit der Cäsar-Scheibe eine Nachricht schicken und dazu den Schlüssel „G“ verwenden, so stellt auch der Empfänger seine Scheibe auf „G“ ein.

Ein solches Verfahren nennt man auch „symmetrische Verschlüsselung“, denn wenn jemand den einen existierenden Schlüssel kennt, kann er damit sowohl Nachrichten codieren als auch decodieren.

Im Gegensatz dazu steht die „asymmetrische Verschlüsselung“, die zwei unabhängige Schlüssel wechselweise benutzt. Das wollen wir in einem Experiment selbst erforschen. Hierzu wird eine weitere kleine Bastelei notwendig: Schneiden Sie die drei Stücke des ASYM-Codierers an den dicken schwarzen Linien aus. Dann schneiden Sie diese an den dünnen schwarzen Linien ein bzw. aus. Der dünne Codierbalken mit den Beschriftungen „Schlüssel“, „Klartext“ und „Geheimschrift“ muss danach an den gestrichelten Linien nach hinten gefalzt werden. Diese stecken Sie dann durch die Aussparungen des großen Stücks, falzen Sie vollständig um und kleben Sie dann mit wirklich wenig Klebstoff auf die gestrichelten Bereiche des zweiten dünnen Stücks.

Nun sollten Sie in der Lage sein, den Balken frei über dem Codierer hin und her zu bewegen, so wie in Abbildung 9.5 dargestellt.



Sehr gut funktioniert auch folgender Bastelvorschlag der GI-Fachgruppe HILL: Laminieren Sie alle Teile des ASYM-Codierers und schneiden Sie die in Abbildung 9.5 grau gezeichneten Teile heraus. Statt die Falze zu benutzen, tackern Sie den Codierbalken einfach so zusammen, dass er frei über dem Feld beweglich bleibt.

Abbildung 9.5 Der ASYM-Codierer

Mit dem ASYM-Codierer können Sie Nachrichten ver- und entschlüsseln, die aus den Buchstaben „A“ bis „Z“ sowie den Zeichen „-“ und „.“ bestehen. Der Schlüssel besteht aus einem Buchstaben oder einem Wort.

Zur Verschlüsselung schreiben Sie den Schlüssel so oft wie notwendig über die Nachricht in Klartext. Nun verschlüsseln Sie jeden einzelnen Buchstaben, indem Sie den Balken des ASYM-Codierers so stellen, dass der entsprechende grüne Schlüssel im oberen Fenster sichtbar wird. Lesen Sie dann den Geheimbuchstaben im unteren Feld neben dem blauen Klartextbuchstaben ab! Leerzeichen bleiben unverändert und werden bei der Codierung nicht berücksichtigt.

Abbildung 9.6 zeigt ein Beispiel. Die Nachricht „ABENTEUER INFORMATIK“ soll mit dem Schlüssel „EMU“ verschlüsselt werden. „EMU“ wird hierfür entsprechend oft über den Klartext geschrieben. In der Abbildung werden – genau wie im Codierer – blau für den Klartext, grün für den Schlüssel und rot für die Geheimschrift verwendet.

EMUEMUEMU	EMUEMUEMU
ABENTEUER	INFORMATIK
FRCZFCJPE	YJUCQBFFQH

Abbildung 9.6 Die Codierung von „ABENTEUER INFORMATIK“ mit dem Schlüssel „EMU“

Um den ersten Buchstaben „A“ zu codieren, stellen Sie den Codierer auf den ersten Schlüsselbuchstaben „E“ ein. Lesen Sie dann den ersten Geheimbuchstaben neben dem blauen „A“ ab. Es handelt sich um ein „F“. Machen Sie nun das Gleiche für alle Klartextbuchstaben, die unter einem „E“ stehen. Danach sind die Schlüsselbuchstaben „M“ und „U“ dran, stellen Sie den Codierer auf die entsprechenden Schlüssel und codieren Sie die zugeordneten Buchstaben des Klartextes. Die resultierende Geheimbotschaft lautet „FRCZFCJPE YJUCQBFFQH“. Es ist sicherlich ziemlich schwierig, ohne den entsprechenden Schlüssel den Klartext herauszufinden. Dieser Schlüssel heißt allerdings nicht „EMU“, sondern „FIR“!

Probieren Sie es selbst aus! Gehen Sie genauso vor wie bei der Erstellung der Geheimschrift, nur „verschlüsseln“ Sie diesmal die Botschaft „FRCZFCJPE YJUCQBFFQH“ mit dem Schlüssel „FIR“.



Abbildung 9.7 zeigt das Ergebnis. Die neue „Geheimbotschaft“ ist die Nachricht, mit der Sie angefangen hatten – „ABENTEUER INFORMATIK“.

F I R F I R F I R F I R F I R F
 F R C Z F C J P E Y J U C Q B F F Q H
 A B E N T E U E R I N F O R M A T I K

Abbildung 9.7 Decodierung der Geheimbotschaft

Versuchen Sie es auch umgekehrt: Verwenden Sie zum Verschlüsseln der Nachricht den Schlüssel „FIR“ und dann zum Entschlüsseln „EMU“. Spielen Sie weiter mit dem Verfahren. Was passiert, wenn man eine Nachricht zweimal hintereinander mit dem gleichen Schlüssel codiert? Probieren Sie dies mit verschiedenen Nachrichten aus! Experimentieren Sie auch mit den beiden Schlüsseln „ROM“ und „UBI“.



Wenn Sie „ABENTEUER INFORMATIK“ mit „FIR“ codieren, kommt als Geheimbotschaft „EZRL.RRGA PHJQBLE.OZ“ heraus. Wenn Sie nun „EMU“ als Schlüssel verwenden, landen Sie wieder bei „ABENTEUER INFORMATIK“. Die erneute Anwendung des gleichen Schlüssels führt dagegen nicht zu einem Klartext: „YSABPABDU VQG-ZTYPYN“.

Das Verfahren funktioniert bei allen verwendeten Nachrichten und es funktioniert ebenfalls mit den beiden Schlüsseln „ROM“ und „UBI“ – allerdings immer nur mit zusammengehörigen Schlüsseln. Versuchen Sie zum Beispiel eine Nachricht mit „EMU“ zu verschlüsseln und dann mit „ROM“ oder „UBI“ zu entschlüsseln, kommt nur unleserlicher Kauderwelsch heraus!

Fassen wir die Erkenntnisse nochmals zusammen:

Der ASYM-Codierer erlaubt Ihnen, einen Code zu benutzen, der zwei unterschiedliche Schlüssel hat. Durch die Anwendung beider Schlüssel hintereinander kommt wieder die ursprüngliche Nachricht hervor. Dabei ist die Reihenfolge der Anwendung unerheblich.

Eine einmal verschlüsselte Nachricht ist somit ausschließlich mit dem entsprechenden Gegenschlüssel wieder zu entziffern. Kein anderer Schlüssel, auch nicht der Schlüssel, der zur Verschlüsselung genutzt wurde, kann den Klartext sonst noch hervorbringen!

Als Nächstes wollen wir dieses neue Verfahren dazu nutzen, Nachrichten sicher über das Internet zu senden. Zuvor jedoch noch ein Hinweis, wenn Sie gegenüber dem ASYM-Codierer skeptisch sind und meinen, ihn geknackt zu haben:

Lesen Sie bitte den Hinweis im Kästchen am Ende des Kapitels. Falls das nicht der Fall ist: Nehmen Sie sich nicht den Spaß und arbeiten Sie das Kapitel einfach weiter durch.

Die Schönheit der Asymmetrie

Spielen wir noch einmal ein Szenario durch, bei dem die Cäsar-Scheibe als Codierer versagt hatte:

Sie sitzen in einem Zimmer und spielen sich selbst. Sie möchten beim Online-Versender Aragon ein Buch kaufen und zu diesem Zweck die Nummer Ihrer Kreditkarte („ZWEIVIERDREIACHT“, eine kleine Zahl soll für das Experiment reichen) dorthin schicken.

Ihr Freund Otto spielt den Versender Aragon und sitzt in einem anderen Zimmer. Ein persönliches Treffen zwecks Schlüsseltausch ist ausgeschlossen. Ein weiterer Freund, Emil, spielt den Boten – er steht für alle schwarzen Schafe, die prinzipiell Ihren Internet-Verkehr mithören oder manipulieren könnten. Emil hat daher den Auftrag, Ihre Kreditkartennummer auszuspionieren, ohne Verdacht zu erregen (sonst könnten Sie die Karte ja sperren lassen).

Die Namen Otto und Emil werde ich im folgenden Teil konsequent verwenden. Sie stehen stellvertretend für „echte“ Freunde, mit denen Sie dieses Szenario durchspielen. Das ist ein großer Spaß und ich empfehle es, um das Verfahren zu verstehen. Basteln Sie für jeden Teilnehmer eine Cäsar-Scheibe (als Codierer für einen symmetrischen Code) und einen ASYM-Codierer.

Fassen wir also die Aufgabe nochmals zusammen: Sie möchten die Nachricht „ZWEIVIERDREIACHT“ an Otto schicken. Das geht nur über den Boten Emil. Dieser soll allerdings keine Chance haben, die Informationen auszuspionieren und die Kreditkartennummer selbst missbräuchlich zu verwenden.

Versuchen Sie, das Problem alleine unter Verwendung der Cäsar-Scheibe zu lösen. Beschreiben Sie, welche Nachrichten Sie und Otto mit welchen Schlüsseln codieren, um das Ziel zu erreichen.



Wie man es auch dreht und wendet – ohne zuvor einen Schlüssel ausgetauscht zu haben, gibt es keine Möglichkeit, die Nachricht sicher zu übermitteln! Entweder Sie übermitteln gleich den Klartext oder Sie müssen vorher irgendwie den Schlüssel weitergeben, was auf das Gleiche herausläuft, denn das geht auch nur über Emil, der ihn dann ebenfalls hat und jede Ihrer Nachrichten entschlüsseln kann.

Probieren wir, ob wir mit dem ASYM-Codierer mehr Erfolg haben. Wir gehen dabei davon aus, dass ein zusammengehöriges Schlüsselpaar sehr schwer zu finden ist und man von einem Schlüssel des Paares nicht auf den zugehörigen anderen schließen kann. Im Anhang finden Sie ein paar verschiedenfarbige Kärtchen mit einigen Schlüsselkombinationen. Um das Spiel authentisch zu machen, hat jeder Spieler seine eigenen Schlüsselkärtchen. Jeder kennt anfangs auch nur seine eigenen Schlüssel.

Sie als „normaler“ Kunde im Internet haben natürlich keinen Schlüssel. Otto als Internet-Händler kennt die orangenen Schlüssel. Emil als Internet-Schlitzohr hat sich ebenfalls ein paar Schlüssel errechnet, nämlich die türkisfarbenen.

Wie geht man nun vor?

Otto als Aragon AG nimmt eines seiner Schlüsselpaare, zum Beispiel „BOT“ und „OBP“. Er markiert einen Schlüssel mit einem „Ö“ und einen anderen mit „P“, zum Beispiel so wie in Abbildung 9.8. Das „Ö“ steht für „öffentlich“ – diesen Schlüssel gibt er jedem, der ihn wissen möchte. „P“ steht für „privat“, diesen Schlüssel behält er geheim! Niemand (auch nicht einer seiner Freunde und auch kein Kunde) bekommt ihn. Zumindest tun wir jetzt einmal so, als wenn er nicht hier im Buch für alle zu lesen stände...

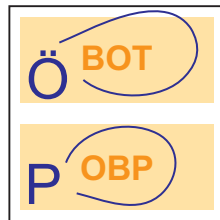


Abbildung 9.8 Markierte Schlüssel

Von alledem wissen Sie als Kunde jedoch nichts! Sie möchten Ihre Kreditkartennummer sicher an Otto übermitteln und benötigen dazu erst einmal seinen Schlüssel. Daher schicken Sie ihm die folgende Nachricht in Klartext: „WIE IST DEIN SCHLUESSEL“. Emil liefert diese aus und nimmt die Antwort von Otto entgegen: „BOT“. Auch dieser Schlüssel geht unverschlüsselt an Sie.

Nun nehmen Sie die Nachricht „ZWEIVIERDREIACHT“ und verschlüsseln Sie mit „BOT“.



Emil darf jetzt die Botschaft „MQVSO.QGFIJ.OFLJ“ übermitteln. Er kann damit nichts anfangen, denn obwohl er selbstverständlich ebenfalls im Besitz des öffentlichen Schlüssels ist, kann er sie nicht decodieren.

Otto entschlüsselt die Nachricht mit seinem privaten Schlüssel „OBP“ – nur auf diese Weise ist die Originalbotschaft wieder lesbar. Er lässt nun über Emil die Quittung „IST ANGEKOMMEN“ in Klartext zurückmelden – denn hier ist ja kein Geheimnis auszukundschaften.

Auf den ersten Blick sieht es nun so aus, als wenn Emil keine Chance hätte, Ihre geheimen Botschaften zu erkunden. Ich behaupte einfach einmal, dass diese Annahme verfrüht ist. Daher dürfen Sie als Nächstes ausnahmsweise einmal Ihre volle kriminelle Energie ausleben: Versetzen Sie sich in die Rolle von Emil. Dieser möchte Ihrem Team schaden. Natürlich könnte er einfach die Botschaften wegwerfen und gar nicht weiterleiten. Wenn jetzt jedoch gar keine Antworten kämen, würden Sie misstrauisch und würden den Boten austauschen. Überlegen Sie, welche Möglichkeiten er trotz asymmetrischer Verschlüsselung noch hat.



Sie sind jetzt sicherlich auf eine ganze Reihe diabolischer Pläne gekommen. Hier meine erste Idee, bei der Emil eine noch recht harmlose Sabotage betreibt:

Emil leitet die verschlüsselte Nachricht nie an Otto weiter, sondern wirft sie gleich weg und schreibt selbst die Antwort „IST ANGEKOMMEN“, die er Ihnen nach kurzem Warten überbringt. Auf diese Weise werden Sie nicht misstrauisch und denken, alles sei in Ordnung, während die Aragon AG Ihre Daten nie erhält und damit auch kein Kaufvertrag zustande kommt. Sie würden vergeblich auf Ihre Lieferung warten!

Überlegen Sie, wie man die Kommunikation durch eine kleine Veränderung gegen dieses Vorgehen schützen kann! Tipp: Es geht natürlich darum, die Antwort irgendwie abzusichern.



Das Problem tritt auch bei der Kommunikation im Internet auf: Einerseits möchten wir für bestimmte Nachrichten sicherstellen, dass nur der Adressat ihren Inhalt lesen kann. Das machen wir durch Verschlüsselung.

Es gibt jedoch eine weitere Situation: Hier müssen wir sicherzustellen, dass eine Nachricht auch von dem kommt, der als Absender darauf steht. Das nennt man Authentifizierung. Genau das Problem stellt sich bei der Antwort von Otto „IST ANGEKOMMEN“.

Nur wenn wir sicher sind, dass die Antwort wirklich von Otto kommt, können wir beruhigt weiterarbeiten in der Gewissheit, dass auch Otto auf dem neusten Erkenntnisstand ist. Auch das kann mit der ASYM-Codierung gewährleistet werden:

Otto schickt die Antwort „IST ANGEKOMMEN“ nicht im Klartext, sondern codiert sie mit seinem privaten Schlüssel. Machen Sie das für ihn!



Emil muss nun die Nachricht „RXW BGXJTJZUM.“ weiterleiten. Er kann diese jederzeit entschlüsseln, denn dies lässt sich mit dem öffentlichen Schlüssel sehr leicht erledigen. Allerdings schafft er es nicht, selbst eine solche Nachricht zu erzeugen: Das kann nur derjenige, der den privaten Schlüssel hat!

Wenn Sie nun eine Antwort von Otto bekommen, decodieren Sie sie als Erstes mit dessen öffentlichem Schlüssel. Falls nun etwas Sinnvolles herauskommt, können Sie sicher sein, dass der Absender tatsächlich Otto ist. Falls nicht, hat sich offenbar jemand an der Nachricht zu schaffen gemacht und Sie können Emil durch jemand Vertrauenswürdigen ersetzen.

Fassen wir bis hierher nochmals die Erkenntnisse über die ASYM-Codierung zusammen:

Wenn jemand ein Schlüsselpaar besitzt und einen Schlüssel als öffentlichen Schlüssel bekannt macht, dann kann jeder ihm geheime Nachrichten schicken (er weiß aber nicht, ob der Absender wirklich „echt“ ist). Er selbst kann niemandem eine geheime Nachricht schicken, allerdings kann er seine Nachrichten authentifizieren: Jeder kann nachvollziehen, dass die von ihm verschickte Botschaft auch wirklich von ihm stammt, denn nur seine Botschaften geben einen Sinn, wenn man sie mit seinem öffentlichen Schlüssel decodiert.

Trotz allem kann Emil nun noch Schaden anrichten! Das Problem liegt im anfänglichen Schlüsseltausch. Erinnern Sie sich? Die erste Nachricht an Otto war „WIE IST DEIN SCHLUESSEL“. Emil liefert diese Nachricht bestimmungsgemäß ab, behält allerdings die Antwort „BOT“ ein und liefert stattdessen „CLU“. Dabei handelt es sich um einen seiner eigenen öffentlichen Schlüssel.

Sie können nicht erkennen, ob die Antwort tatsächlich von Otto stammt, denn zu diesem Zeitpunkt haben Sie ja noch keinen Schlüssel, der die Authentizität ge-

währleistet. Daher codieren Sie die Nachricht mit dem öffentlichen Schlüssel „CLU“ und verschicken „NCCZQUPNF-QJV.Z“. Das kann jedoch Emil ohne Probleme mit seinem eigenen privaten Schlüssel „JQR“ entziffern. Um keinen Verdacht zu erregen, codiert er dann die Nachricht neu mit Ottos öffentlichem Schlüssel „BOT“. Auf diese Weise erkennt Otto nicht, dass jemand die Botschaft abgefangen hat und reagiert normal. Selbstverständlich muss Emil nun auch Ottos Antwort wieder mit seinem eigenen privaten Schlüssel umcodieren.

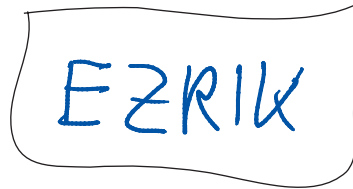
Spiele Sie diese Situation mit zwei Freunden durch. Wechseln Sie dabei in alle drei Rollen. Sie können dabei immer Ihren persönlichen Schlüssel behalten, denn Sie geben ja auf jeden Fall nur Ihren öffentlichen Schlüssel heraus.



Offenbar ist der Austausch des Schlüssels hier immer noch ein Problem! Das Gleiche gilt im Internet: Die asymmetrische Verschlüsselung erlaubt eine grundsätzlich sichere Kommunikation mit einem Partner. Nicht sicher ist es, an den Schlüssel heranzukommen.

Für unser Spiel bedeutet dies: Mit dem gegebenen Szenario gibt es für Sie keine Möglichkeit, eine geheime Nachricht an Otto zu schicken. Hierfür müssen wir eine weitere Instanz hinzunehmen, wie im Folgenden beschrieben.

Eine weitere Spielerin kommt hinzu: Ute übernimmt die Rolle der so genannten „Zertifizierungsinstanz“ oder „Certification Authority (CA)“. Zu diesem Zweck drückt sie vor dem Spiel jedem Teilnehmer persönlich einen Zettel in die Hand:



Außerdem bietet Sie gegen einen kleinen Obolus an, dass man sich persönlich mit ihr trifft und einen Schlüssel übergibt. Otto als Internet-Firma nimmt dies gerne in Anspruch und übergibt ihr persönlich ein Schriftstück mit seinem Schlüssel „OTTO: BOT“. Ute überprüft anhand offizieller Dokumente (z. B. Personalausweis), dass es wirklich Otto ist, der ihr den Schlüssel übergibt und archiviert den Schlüssel.

Seien Sie Verschlüsselungs-Detektiv! Wie könnten Sie nun sicher mit Otto alias Aragon AG eine Nachricht austauschen? Wie müssen beide vorgehen?



EZRIK ist offenbar der öffentliche Schlüssel der Zertifizierungsinstanz Ute. Zur Unterscheidung nehmen wir den längeren Schlüssel mit fünf Zeichen, das wäre jedoch nicht unbedingt notwendig. Der persönlich übergebene Zettel stellt sicher, dass alle wirklich den echten Schlüssel bekommen haben! Auf diese Weise kann also jeder von Anfang an mit Ute verschlüsselt Nachrichten austauschen! Die nun notwendige Vorgehensweise für eine sichere Nachrichtenübermittlung zwischen Ihnen und Otto habe ich im Folgenden dargestellt.

Abbildung 9.9 zeigt die Ausgangssituation schematisch: Jeder Spieler kennt sein eigenes Schlüsselpaar. Außerdem kennt jeder auch den öffentlichen Schlüssel der Zertifizierungsinstanz: EZRIK. Im Diagramm ist neben den Personen all ihr Wissen dargestellt. Dieses gilt als „sicher“ – die persönlichen Schlüssel per Definition und der Schlüssel EZRIK, weil jeder ihn persönlich von Ute erhalten hat. Ute hat zusätzlich den Schlüssel „BOT“ als gesicherte Information – auch dieser Schlüssel wurde „persönlich“ übergeben.

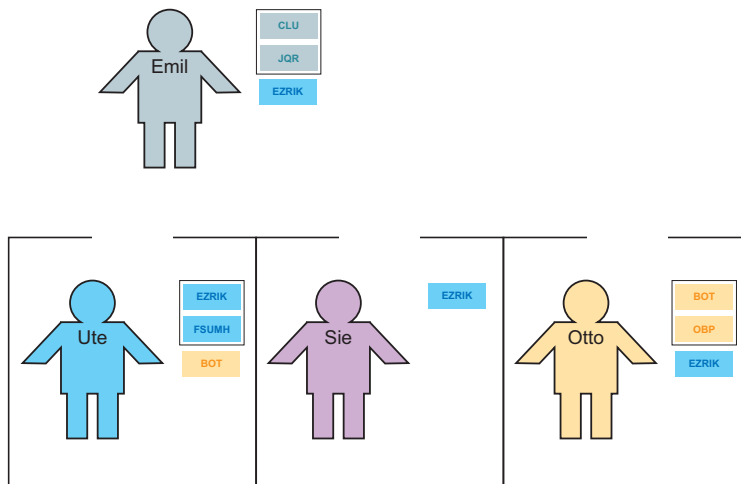


Abbildung 9.9 Die Ausgangssituation

Sie wollen nun die Nummer Ihrer Kreditkarte sicher an Otto schicken. Überlegen Sie, wie Sie hierfür vorgehen könnten. Zeichnen Sie auf ein Blatt Papier jede Nachricht mit Sender und Empfänger und wie diese verschlüsselt ist.



Es ist nun wichtig, den korrekten Schlüssel von Otto herauszufinden. Genau hier lag die Schwäche im letzten Versuch. Sie haben allerdings bereits einen vertrauenswürdigen Gesprächspartner, mit dem eine sichere Kommunikation möglich ist: Ute.

Ich mache hierfür einen Vorschlag, der noch nicht ganz korrekt ist. Sehen Sie es als Herausforderung und spielen Sie erneut Detektiv, um die Sicherheitslücke herauszufinden und zu eruieren, wie Emil diese ausnutzen könnte.

Als Erstes schicken Sie eine Anfrage an Ute mit der Frage nach Ottos Schlüssel, wie in Abbildung 9.10 dargestellt.

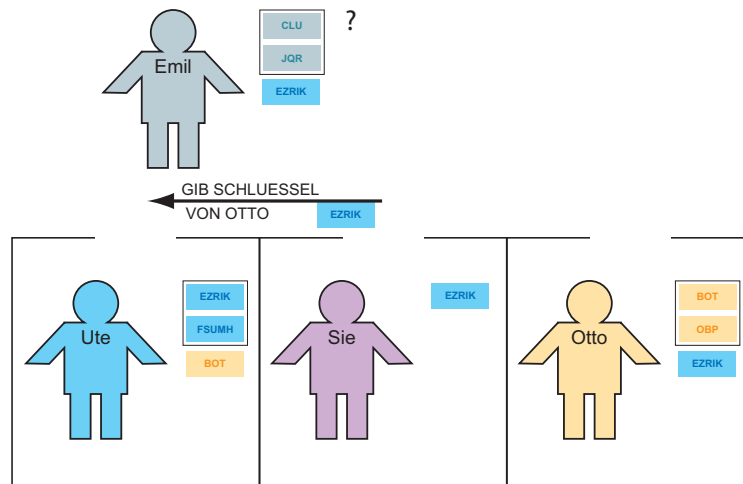


Abbildung 9.10 Frage an Ute nach Ottos Schlüssel

Emil kann diese Kommunikation nicht entschlüsseln, denn Sie nutzen Utes öffentlichen Schlüssel zur Codierung.

Abbildung 9.11 zeigt, wie Ute antwortet und Ihnen Ottos Schlüssel übermittelt. Dabei nutzt Ute ihren privaten Schlüssel zur Codierung, daher ist für Sie sicher gestellt, dass die Antwort auch von ihr kommt. Allerdings kann jeder (auch Otto) die Nachricht entschlüsseln, was aber nichts macht, denn es handelt sich ja um einen öffentlichen Schlüssel, den jeder haben darf.

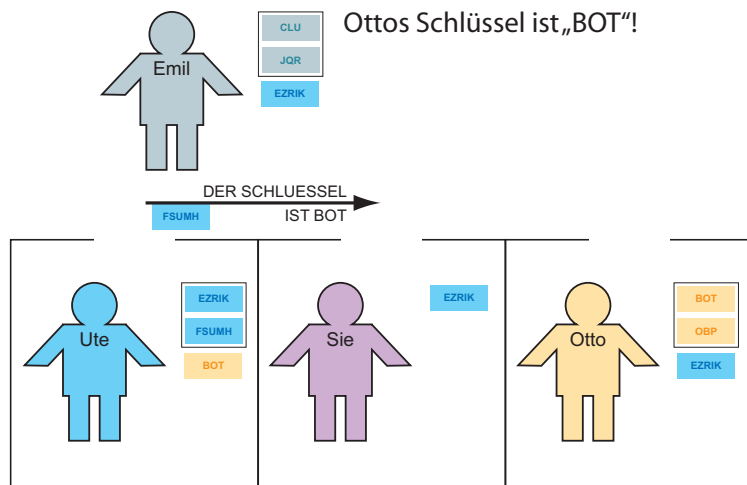


Abbildung 9.11 Übermittlung des Schlüssels

Danach kennen Sie Ottos korrekten Schlüssel und können mit ihm kommunizieren: Abbildung 9.12 zeigt, wie sie die Nummer Ihrer Kreditkarte an Otto senden können. Emil kann diese nicht entschlüsseln, nur Otto ist hierzu in der Lage. Er schickt Ihnen eine Bestätigung zurück und verwendet zur Authentifizierung seinen privaten Schlüssel, wie in Abbildung 9.13 zu sehen.

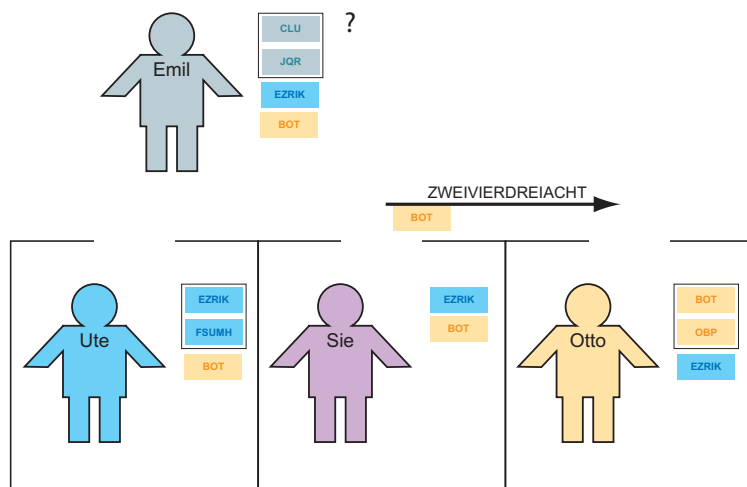


Abbildung 9.12 Die Kreditkartennummer wird übertragen

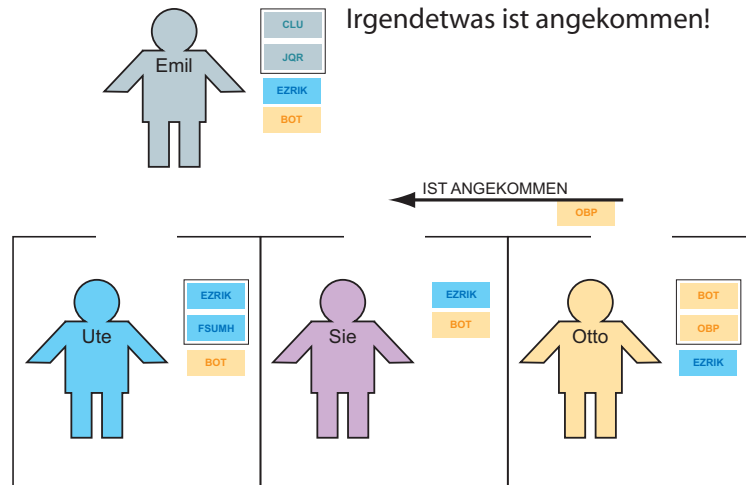


Abbildung 9.13 Bestätigung von Otto

Jetzt sind Sie an der Reihe: Spielen Sie Emil! Wie würden Sie vorgehen, um die Kreditkartennummer trotz aller Sicherheitsvorkehrungen auszuspionieren!



Tja – Emil ist immer noch der Überbringer aller Nachrichten. Er kann jede Nachricht abfangen und durch eine andere ersetzen. Letztlich möchte er Ihre Kommunikation mit Otto abhören. Er darf daher auf keinen Fall zulassen, dass Sie Ottos echten öffentlichen Schlüssel bekommen. Wenn das passiert, ist die Sache wasserdicht und er aus dem Rennen. Den Schlüssel bekommen Sie allerdings von Ute und genau hier greift Emil an.

Auch Emil kann sich zu Beginn mit Ute treffen und dort ganz legal seinen öffentlichen Schlüssel hinterlegen. Nun kennt Ute auch seinen Schlüssel und kann diesen auf Anfrage weitergeben.

Schauen Sie sich Abbildung 9.10 nochmals an. Hier fragen Sie Ute nach dem Schlüssel von Otto. Emil weiß nicht wirklich, was in der Nachricht steht, denn Sie haben die Übertragung verschlüsselt – nur Ute kann diese mit ihrem privaten Schlüssel entziffern. Aber da Ute die Zertifizierungsinstanz ist, kann Emil sich denken, dass es sich um die Frage nach einem Schlüssel handelt.

Er fängt daher die Nachricht ab und schickt stattdessen die Frage „GIB SCHLUESSEL VON EMIL“ in Ihrem Namen an Ute. Sie besitzen selbst keinen privaten und öffentlichen Schlüssel. Daher können Sie die Nachricht zwar verschlüsseln, aber gegenüber Ute nicht authentifizieren. Ute kann daher nicht entscheiden, ob die Nachricht wirklich von Ihnen stammt oder nicht. Da die Nachricht mit dem öffentlichen Schlüssel von Ute codiert wird, ist jeder – auch Emil – hierzu in der Lage. Abbildung 9.14 zeigt diesen Schritt.

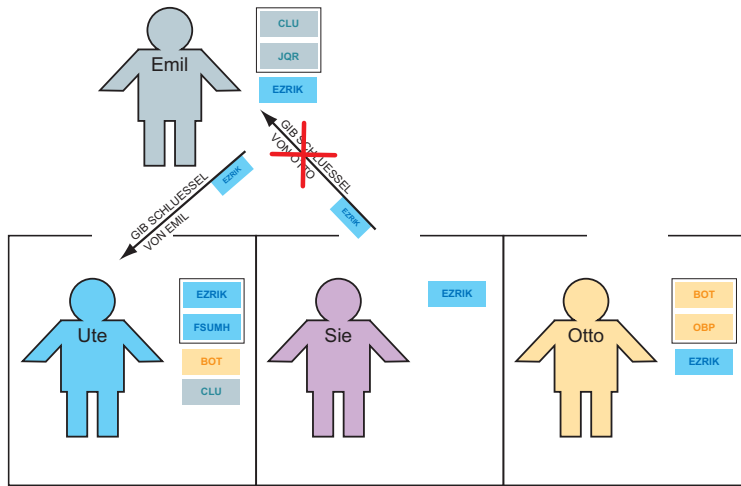


Abbildung 9.14 Emil fängt die Nachricht ab und ersetzt sie durch eine eigene

Nun antwortet Ute ganz wahrheitsgemäß auf die Anfrage mit Emils öffentlichem Schlüssel (Abbildung 9.15). Sie verschlüsselt diese Nachricht mit ihrem privaten Schlüssel und authentifiziert sie dadurch: Niemand anderes außer Ute kann diese Nachricht schicken, daher vertrauen Sie ihr. Wenn Sie nun die Kreditkartennummer an Otto übermitteln, verwenden Sie Emils Schlüssel. Emil kann die Nachricht wieder abfangen und – wie bereits oben geschildert – seine Spuren verwischen.

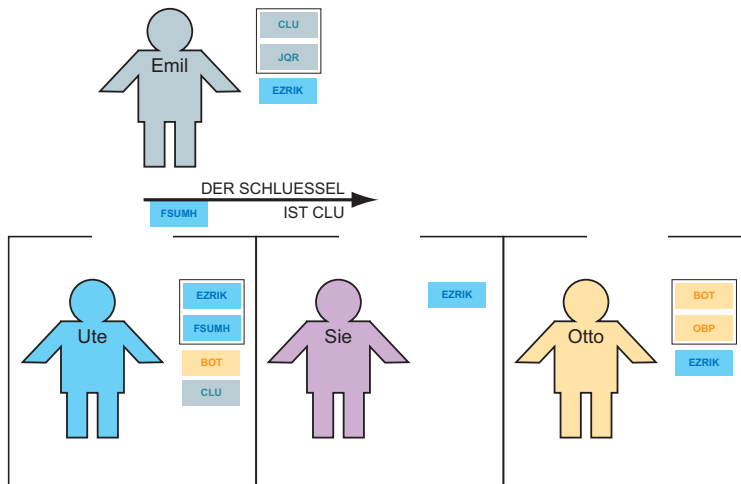


Abbildung 9.15 Sie erhalten Emils Schlüssel anstatt Ottos

Wieder dürfen Sie die Seiten wechseln: vom Schlitzohr zum Sicherheitsexperten. Welche zusätzliche Vorkehrung muss getroffen werden, um die Übermittlung von Ottos Schlüssel wirklich sicher zu machen?



Zwei Schwachstellen treten bei der Übermittlung des Schlüssels auf:

1. Ute kann nicht feststellen, dass die Anfrage nach einem Schlüssel echt ist (oder von einer dritten Partei wie Emil gefälscht wurde).
2. Sie können nicht feststellen, dass Sie Emils Schlüssel statt Ottos bekommen haben.

An der ersten Schwachstelle können wir systembedingt nichts verändern. Da Sie nicht im Besitz eines Schlüssels sind, können Sie die Nachricht nicht authentifizieren. Allerdings können wir an der zweiten drehen und dadurch die Übermittlung wasserdicht machen: Ute sendet nicht nur einfach den Schlüssel, sondern in der gleichen Nachricht auch die Information, zu wem dieser Schlüssel passt, wie in Abbildung 9.16.

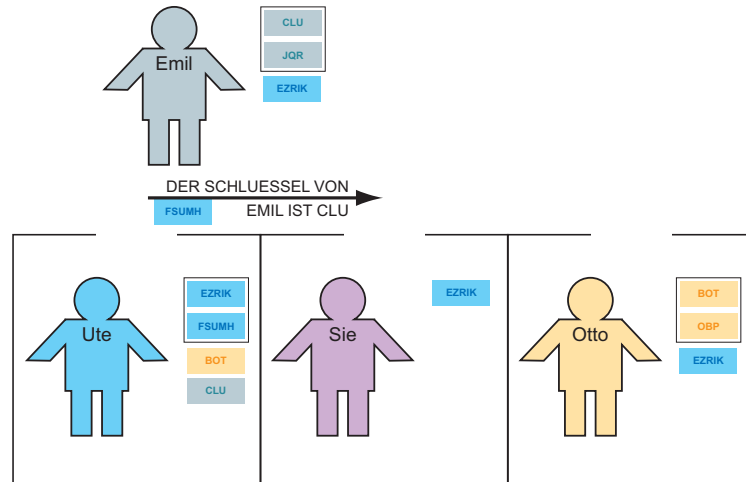


Abbildung 9.16 Ute übermittelt Schlüssel und Schlüsselpaare zusammen

Auf diese Weise registrieren Sie immer, wessen Schlüssel Sie bekommen haben. Sie können erkennen, dass offenbar mit der Kommunikation zu Ute etwas schief gegangen ist, denn sie hat nicht die Antwort auf Ihre Frage gesendet. Sie sind immer noch nicht im Besitz von Ottos Schlüssel und werden Ihre Kreditkartennummer nicht preisgeben!

Was zeigt dieses Beispiel?

Bei der Verwendung der asymmetrischen Verschlüsselung hat immer eine Partei ein Schlüsselpaar und eine andere nicht. Normalerweise haben heute nur Geschäfte, Banken oder andere Anbieter ein Schlüsselpaar, ein „normaler“ Kunde nicht. Daher spreche ich hier von „Kunde“ und „Anbieter“.

Die Kommunikation vom Kunden zum Anbieter ist immer geheim, aber nicht authentifiziert. Die Kommunikation vom Anbieter zum Kunden ist immer offen, aber authentifiziert. Diese Regel muss man jederzeit bedenken und entsprechende – vielleicht überflüssig erscheinende – Informationen zur Sicherung einfügen, wie in diesem Beispiel die Information über den Besitzer des Schlüssels.

Mit Zertifikat geht alles besser

Tatsächlich funktioniert es genau so wie beschrieben, wenn Sie im Internet mit dem sicheren Server eines Händlers verbunden sind und ihm Ihre Kreditkartendaten übermitteln: Zunächst überprüft Ihr Browser, zum Beispiel der Internet-Explorer, anhand einer Zertifizierungsinstanz den korrekten öffentlichen Schlüssel des Händlers. Daraufhin kann er die Nummer sicher senden.

Vielleicht wenden Sie nun ein, dass Sie sich nicht erinnern können, am Anfang von irgendwem einen Zettel mit der Aufschrift „EZRIK“ zugesteckt bekommen zu haben. Mit anderen Worten: Woher kennt Ihr Computer am Anfang den öffentlichen Schlüssel der Zertifizierungsinstanz?

Hier hat der Hersteller des Browsers die Arbeit für Sie bereits übernommen. Heutige Browser haben die öffentlichen Schlüssel der wichtigsten Zertifizierungsinstanzen (Certification Authorities oder CAs) bereits fest eingebaut. Auf diese Weise ist die Verbindung dorthin auch von Anfang an sicher.

Vielleicht ist Ihnen noch ein interessantes Detail aufgefallen: Bei der asymmetrischen Verschlüsselung kann jeder heimliche Mithörer alles lesen, was vom Anbieter an Sie verschickt wird. Diese Kommunikation ist lediglich authentifiziert, aber nicht geheim! Wenn also der Anbieter zur Bestätigung die Nummer Ihrer Kreditkarte nochmals wiederholte, wäre die gesamte Mühe der Verschlüsselung auf dem Hinweg vergebens gewesen. Aus diesem Grund dürfte ein Anbieter nie solche kritischen Informationen zurückschicken, sondern immer nur eine allgemeine Bestätigung (z. B. „Die Zahlung erfolgt über Ihre Kreditkarte ****-****-****-1234“). Das passiert zu Ihrer Sicherheit!

Allerdings kann sich nicht jeder Anbieter damit zufrieden geben, dass die Rückantwort an seine Kunden prinzipiell für jeden Lauscher offen erfolgt. Während es vielleicht noch tolerabel ist, dass jemand auf diese Weise den Inhalt einer Buchbestellung mitliest, gilt das sicherlich nicht für Kontoauszüge beim Online-Banking. Hier ist die Geheimhaltung in beide Richtungen von großer Wichtigkeit.

Dies wird wiederum von einem symmetrischen Verfahren gewährleistet, wie in unserem Experiment von der Cäsar-Scheibe. Wenn wir nur irgendwie sicher den Schlüssel übergeben könnten...

Genau dies ist aber dank des asymmetrischen Verfahrens möglich: Versetzen Sie sich nochmals in das Szenario von oben: Sie haben nun von der Zertifizierungsinstanz Ute Ottos öffentlichen Schlüssel „BOT“ bekommen. Sie könnten ihm nun – für niemanden lesbar – Ihre Kreditkartennummer zukommen lassen oder auch irgendeine andere Information. Diese andere Information kann also auch der Schlüssel für die Cäsar-Scheibe sein, wie in Abbildung 9.17.

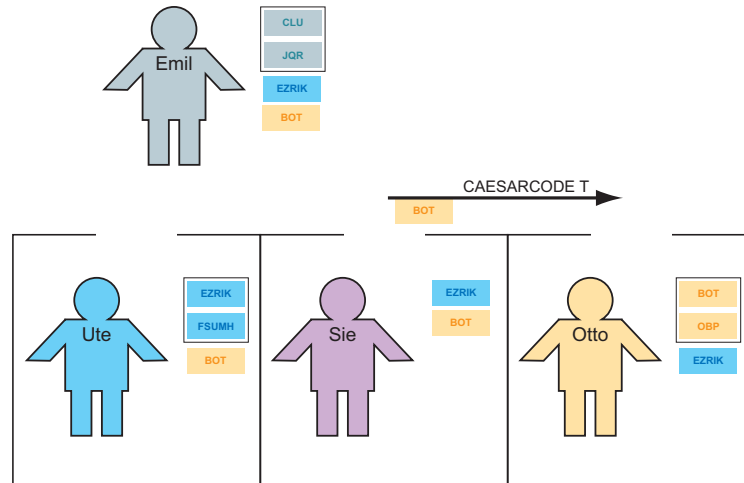


Abbildung 9.17 Sie schicken an Otto einen Schlüssel für die Cäsar-Scheibe

Emil kann diese Nachricht nicht entziffern, denn nur Otto kann das mit seinem privaten Schlüssel. Otto schickt daraufhin eine Bestätigung, die er direkt mit der Cäsar-Scheibe (Code „T“) verschlüsselt. Von diesem Moment an können Sie sicher mit Otto Nachrichten austauschen.

Überlegen wir noch kurz, was passiert, wenn Emil die Nachricht „CAESARCODE T“ abfängt und durch eine eigene mit einem eigenen Code ersetzt, zum Beispiel „CAESARCODE X“. Otto hat keine Chance, den Austausch zu bemerken und möchte mit Ihnen nun über den Code X kommunizieren. Er schickt also seine Bestätigung „IST ANGEKOMMEN“ mit diesem Code zurück und Emil kann sie sofort entziffern.

Damit ist er allerdings mit seinem Latein am Ende, denn Sie erwarten eine Rückantwort, die mit dem Code T verschlüsselt ist. Emil kann das nicht wissen, denn er konnte zwar Ihre Nachricht an Otto abfangen, jedoch nicht entschlüsseln. Daher kann er Ihnen auch keine gefälschte Antwort schicken, die mit dem Code T verschlüsselt wurde und seine Schurkerei fliegt auf. Ohne die korrekte Bestätigung von Otto werden Sie sich nicht auf eine Kommunikation einlassen.

Tatsächlich funktioniert auf diese Weise die sichere Kommunikation im Internet, wenn Sie sich als Online-Kunde bei einer Bank oder irgendeinem anderen Anbieter anmelden. Die asymmetrische Verschlüsselung wird nur zu Beginn verwendet, um einen Schlüssel für die symmetrische Verschlüsselung auszutauschen. Danach wird der so vereinbarte Schlüssel für eine Sitzung verwendet. Auf diese Weise bleiben sowohl Ihre Nachrichten an die Internet-Adresse (z. B. PINs, TANs, Überweisungsaufträge) als auch die Rückantwort (z. B. Kontoauszüge, Auftragslisten) ein Geheimnis!

Mit der Cäsar-Scheibe und dem ASYM-Codierer haben Sie übersichtliche Werkzeuge zur symmetrischen und asymmetrischen Verschlüsselung in Händen. Spielen Sie mit Freunden ruhig einmal ein paar Szenarien durch. Es macht auch wirklich Spaß, die Rolle des Schurken zu übernehmen, der versucht, die Botschaften zwischen anderen Spielern abzuhören oder irgendwie anders Kapital daraus zu ziehen.

Daran ist auch nichts Ehrenrühriges, denn auch in Realität ist ein Heer ganz legaler „Hacker“ damit beschäftigt, die Sicherheitsvorkehrungen der eigenen Organisation zu umgehen. Nur so kann sichergestellt werden, dass man auf die Angriffe der „echten“ Schlitzohren vorbereitet ist.

Was steckt dahinter?

Die durchgeführten Experimente haben die wichtigsten Prinzipien der Sicherheitstechnologie im Internet bereits veranschaulicht. Selbstverständlich sind die im letzten Abschnitt praktizierten Verschlüsselungen nicht wirklich sicher! Die statistische Analyse, wie ich Sie Ihnen zu Beginn gezeigt habe, ist nur eines der vielen Werkzeuge, die heimliche Horcher zur Verfügung haben.

Es muss also ein Verschlüsselungsverfahren her, das gegen entsprechende Angriffe gefeit ist. Auch hierfür habe ich oben bereits Möglichkeiten angedeutet. Im Wesentlichen geht es darum, die statistischen Besonderheiten der Dateien zu verschleiern.

Wenn zum Beispiel ein Hacker den Datenverkehr zu einer Bank überwacht, kann er sich erst einmal selbst völlig legal bei der Bank anmelden und kennt dadurch das Schema, in dem zum Beispiel die Eingabe der persönlichen Geheimzahl erfolgt. Dieses unterscheidet sich von Kunde zu Kunde wahrscheinlich lediglich in Bezug auf die Kontonummer und die Geheimzahl selbst. Wenn er nun eine fremde, verschlüsselte Nachricht an die Bank abfängt, kann er womöglich dessen Kontonummer auch noch irgendwie herausbekommen, denn diese ist ja kein großes Geheimnis. Auf diese Weise weiß er zu 95 %, wie die unverschlüsselte Nachricht aussieht (eben bis auf die Geheimzahl). Trotzdem darf er daraus keinen Gewinn ziehen können: Eine winzige Änderung des Klartextes muss den verschlüsselten Text komplett umstrukturieren.

Selbst wenn ein Hacker den kompletten Klartext bereits kennt, darf er daraus nicht den Schlüssel berechnen können. Stellen Sie sich als Beispiel hierfür vor, Sie fangen eine Nachricht ab, die mit der Cäsar-Scheibe codiert wurde. Diese lautet „YRCCF“. Wenn Sie nun wissen, dass der Sender jede seiner Nachrichten mit „HALLO“ beginnt, brauchen Sie nur Ihre Scheibe so einzustellen, dass irgendeiner der Klartextbuchstaben mit dem entsprechenden Geheimtextbuchstaben zur Deckung kommt und können den Schlüssel direkt ablesen: „R“.

Es würde den Rahmen dieses Buches sprengen, die genauen mathematischen Grundlagen für solche Verschlüsselungsverfahren allgemein verständlich darzulegen. Daher beschränke ich mich hier auf den Hinweis, dass es entsprechende Methoden gibt. DES (Data Encryption Standard, mit modernen Abwandlungen wie Triple-DES oder 3DES), AES (Advanced Encryption Standard), IDEA (International Data Encryption Algorithm) und RC4 (Ron’s Cipher 4 nach dem Entwickler Ronald L. Rivest) sind nur einige der tatsächlich verwendeten symmetrischen Verschlüsselungen.

Auf die Länge kommt es an

Ganz egal, welche Verschlüsselung letztlich genutzt wird – wichtig ist die Länge des Schlüssels und damit die Anzahl unterschiedlicher Codierungsmöglichkeiten. Nehmen Sie wieder die Cäsar-Scheibe. Hier gibt es nur 26 verschiedene Schlüssel (von denen einer, „A“, nicht wirklich einen Code erzeugt). Ein Hacker könnte also einfach alle Schlüssel durchprobieren und das Verfahren auf diese Weise einfach und schnell knacken.

Stellen Sie sich nun vor, dass ein Computer das Durchprobieren der möglichen Schlüssel übernimmt und damit Millionen von Kombinationen pro Sekunde abhandelt. Die Länge des Schlüssels bestimmt normalerweise die Anzahl verschiedener Möglichkeiten.

So gibt es $2^{48} = 281.474.976.710.656$ oder ca. 281 Billionen Kombinationen einer Zahl der Länge von 48 Bit. Allerdings können die meisten Verfahren nicht einfach irgendwelche Zahlen als Schlüssel verwenden. Diese Zahlen braucht ein Hacker dann selbstverständlich nicht auszuprobieren. Daher ist die Länge des Schlüssels nur ein Anhaltspunkt für die Sicherheit. Zum Beispiel entspricht ein 64 Bit langer Schlüssel mit dem RC4-Verfahren in etwa der Sicherheitsstufe eines 512 Bit langen Schlüssels im RSA-Verfahren (auf das wir später zu sprechen kommen).

Die folgende Tabelle gibt Aufschluss, wie lange ein guter Computer heute ungefähr benötigt, um einen Schlüssel zu knacken. Die Angaben sind natürlich Schätzungen, da man jetzt noch nicht weiß, wie schnell die Computertechnik sich weiterentwickelt. So könnte die Verwirklichung bestimmter neuer Konzepte die Zeiten noch deutlich reduzieren (z. B. der Quantencomputer).

Länge (symmetrisches Verfahren)	Länge (asymmetrisches Verfahren)	Dauer zum Knacken
56 Bit	384 Bit	Minuten bis Stunden
64 Bit	512 Bit	Tage bis Wochen
112 Bit	1792 Bit	mehrere Jahre
128 Bit	2304 Bit	Jahrzehnte

Sie werden jetzt vielleicht einwenden, dass die Verbindung beim Online-Banking immer nur kurz besteht und daher ein kurzer Schlüssel ausreicht. Das stimmt: Während der Sitzung haben Hacker sicherlich auch bei einem „schlechten“ 56 Bit langen Schlüssel kaum eine Chance zu intervenieren. Was hindert sie allerdings daran, die Datenpakete aufzuzeichnen und dann in aller Ruhe zu entschlüsseln, um etwa Ihren PIN-Code oder Ihre Kreditkartennummer auszuspionieren? Diese ändern sich bestimmt nicht täglich, oder?

Sie konnten sich im Experiment selbst davon überzeugen, dass für ein symmetrisches Verfahren keine Chance besteht, mit dem Gesprächspartner den Schlüssel auszutauschen, es sei denn, man trifft sich von Angesicht zu Angesicht. Die Lösung besteht in asymmetrischen Verfahren. Genau wie selbst ausprobiert, kann zunächst mit Hilfe eines asymmetrischen Verfahrens ein Schlüssel für ein symmetrisches Verfahren ausgetauscht werden. Danach erfolgt die Kommunikation über das symmetrische Verfahren. Es wird momentan im Wesentlichen nur ein einziges asymmetrisches Verfahren verwendet, weshalb ich darauf etwas näher eingehen möchte.

RSA

Das RSA-Verfahren ist nach den Entwicklern Ronald Rivest, Adi Shamir und Leonard Adleman benannt. Im Wesentlichen beruht es darauf, dass es ziemlich aufwendig ist, eine große Zahl in ihre Primfaktoren zu zerlegen. Versuchen Sie es: Ich behaupte, die Zahl 50.065.021 ist das Produkt zweier Primzahlen. Versuchen Sie herauszufinden, welche das sind.



Bis auf Tricks und Kniffe gibt es hier eigentlich nur die Möglichkeit, die Zahl durch alle möglichen Primzahlen zu dividieren, bis ein glattes Ergebnis herauskommt. Dann hat man die beiden Multiplikatoren gefunden. Im Beispiel gilt $50.065.021 = 5.003 \times 10.007$.

Selbstverständlich muss man sehr viel größere Zahlen nehmen, damit das Verfahren auch gegen einen Computer Bestand hat.

Bei RSA werden aus zwei zufällig bestimmten, sehr großen Primzahlen nach einem mathematischen Verfahren zwei Schlüssel abgeleitet, von denen man einen als öffentlichen und einen anderen als privaten Schlüssel deklariert. Nach der Vernichtung der Primzahlen und aller Zwischenberechnungen ist es sehr schwer, aus dem öffentlichen Schlüssel den privaten zu errechnen (ungefähr so schwer wie die Primfaktorzerlegung der großen Zahl).



Genau das ist einer der wesentlichen Faktoren für die Sicherheit des Verfahrens. Hier liegt übrigens auch die größte Schwachstelle des ASYM-Codierers aus dem

Experiment: Sie können zu jedem Schlüssel den „Gegenschlüssel“ erzeugen, indem Sie eine entsprechend große Nachricht codieren, die nur aus Buchstaben „A“ besteht. Wenn Sie zum Beispiel „AAA“ mit dem Schlüssel „BOT“ codieren, kommt „OBP“ heraus. Jeder, der dies einmal durchschaut hat, kann sich also ohne Probleme den privaten Schlüssel jedes Teilnehmers generieren und damit alle Nachrichten lesen oder auch fälschen.

Jeder Betreiber einer Internet-Seite, der seinen Kunden die sichere Datenübermittlung erlauben möchte, muss sich also ein RSA-Schlüsselpaar generieren. Den privaten Schlüssel behält er für sich, den öffentlichen verbreitet er im Internet.

Im Experiment haben wir festgestellt, dass dies allerdings noch nicht ausreicht, denn auf diese Weise könnte der Kunde trotzdem einen falschen öffentlichen Schlüssel untergeschoben bekommen. Irgendwie muss er die Korrektheit des Schlüssels überprüfen!

Das passiert tatsächlich, indem der Serverbetreiber seinen öffentlichen Schlüssel bei einer Certification Authority (CA) hinterlegt und ihn dort „signieren“ lässt. Was bedeutet dieses „Signieren“? Hierzu steckt die CA den öffentlichen Schlüssel zusammen mit dem Namen des Eigentümers, einer Gültigkeitsdauer und anderen Informationen in einen neuen Datensatz. Diesen codiert sie mit ihrem eigenen privaten Schlüssel und gibt ihn an jeden weiter, der ihn haben möchte: Das ganze nennt sich „Zertifikat“ (siehe Abbildung 9.18).

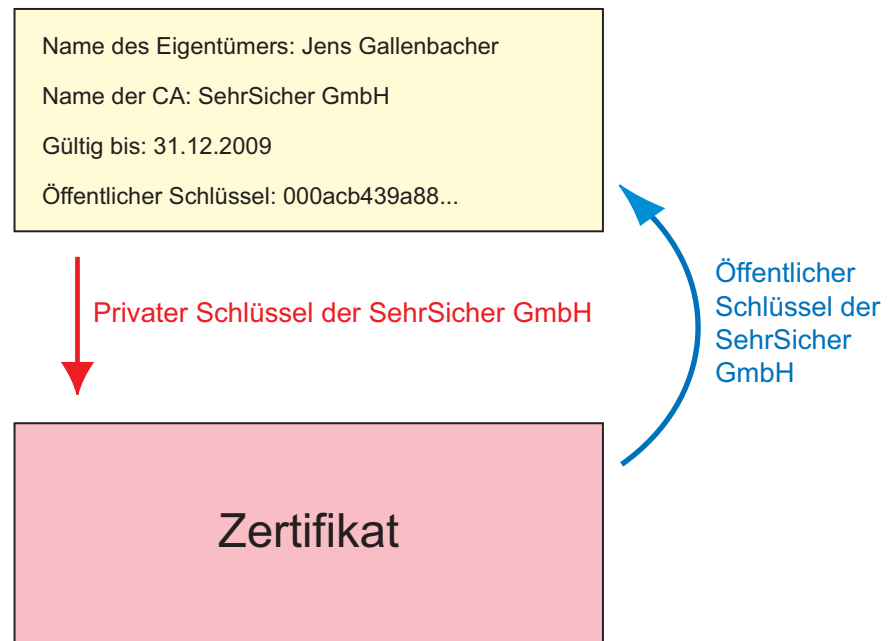


Abbildung 9.18 Zertifikatserzeugung und -benutzung

Ein Zertifikat kann von jedem gelesen werden, der im Besitz des öffentlichen Schlüssels der CA ist (also prinzipiell tatsächlich von jedem). Wenn sich das Zertifikat allerdings mit dem öffentlichen Schlüssel der CA entziffern lässt, muss es mit dem privaten Schlüssel codiert worden sein. Da wir annehmen, dass nur die CA selbst im Besitz ihres privaten Schlüssels ist, können wir dann davon ausgehen, dass das Zertifikat echt ist und seinem Inhalt trauen.

Auf diese Weise ist egal, ob der Kunde das Zertifikat wirklich von der Certification Authority bekommt (wie in unserem Experiment) oder von irgendeiner anderen Stelle, zum Beispiel dem Anbieter selbst. Er kann die Korrektheit auf jeden Fall mit dem öffentlichen Schlüssel der CA selbst überprüfen.

Auf gleiche Weise funktioniert übrigens auch die so genannte „Digitale Unterschrift“. Die Echtheit des Zertifikats wird im obigen Beispiel sichergestellt, indem es von der CA mit ihrem privaten Schlüssel codiert wurde. Auf diese Weise kann aber jeder, der im Besitz eines privaten Schlüssels ist, eine beliebige Datenquelle – zum Beispiel eine E-Mail – unterschreiben: Er verschlüsselt diese mit seinem privaten Schlüssel. Jeder, der die Nachricht abfängt, kann sie decodieren, aber nicht manipulieren, denn er kann sie ja nicht erneut verschlüsseln. Allerdings könnte er die verschlüsselte Nachricht speichern, um sie später nochmals zu senden.

Ein Beispiel hierfür ist eine Überweisung: Sie schicken Ihrer Bank eine signierte E-Mail mit der Überweisung von 1.000 EUR an die Firma „Gäuner & Co KG“. Ein Mitarbeiter dieses Syndikats spioniert die Nachricht aus und sendet sie einfach doppelt an die Bank, so dass die Überweisung nun doppelt ausgeführt würde.

Das kann jedoch leicht verhindert werden, indem der Nachricht die genaue Zeit beigefügt wird. Ist diese auf Millisekunden genau, kann die Bank erkennen, dass die zweite eingereichte Überweisung mit der ersten identisch ist und sie zurückweisen. Ein solcher Zeitstempel ist auch tatsächlich fester Bestandteil einer digital signierten Nachricht. Der Vollständigkeit halber sollte erwähnt werden, dass normalerweise nicht die gesamte Nachricht verschlüsselt wird (sie kann ohnehin von jedem gelesen werden), sondern nur eine Art Prüfsumme. Das reicht aus, um die Authentizität zu garantieren.

Sie möchten jetzt sicherlich wissen, wie Sie die Länge des verwendeten Schlüssels beim E-Banking oder Einkaufen im Internet herausbekommen. Daher werde ich an dieser Stelle die einzige Ausnahme der „Ohne Computer“-Philosophie des Buches machen. Jeder Internet-Browser lässt zu, sich die Eigenschaften einer Seite anzuschauen. Beim Internet-Explorer geht das mit „Datei/Eigenschaften“, bei Mozilla Firefox über „Extras/Seiteninformationen“ und dann „Sicherheit“. Abbildung 9.19 zeigt die Informationen eines Internet-Explorers. Rot eingekreist sehen Sie die Angaben über die verwendeten Verfahren und Schlüssellängen. RSA steht dabei immer für das asymmetrische Verfahren, mit dem der symmetrische Schlüssel ausgetauscht wurde, beim symmetrischen Verfahren gibt es dann verschiedene Möglichkeiten.

Der Quantencomputer ist der Alptraum jedes Kryptographen und der Traum jedes Kryptoanalytikers. Mit seiner Hilfe kann man Zahlen in sehr kurzer Zeit in ihre Primfaktoren zerlegen und auf diese Weise das RSA-Verfahren knacken. Der Quantencomputer nutzt den quantenmechanischen Zustand der „Superposition“ aus – eine Speicherstelle hat danach nicht einen bestimmten Wert, sondern alle Werte gleichzeitig mit bestimmten Wahrscheinlichkeiten. Die Berechnung besteht darin, dass die Wahrscheinlichkeiten sich langsam zu Gunsten des Ergebnisses verändern. Einen Haken hat die Sache mit dem Quantencomputer allerdings: Es konnte bisher niemand einen bauen ...

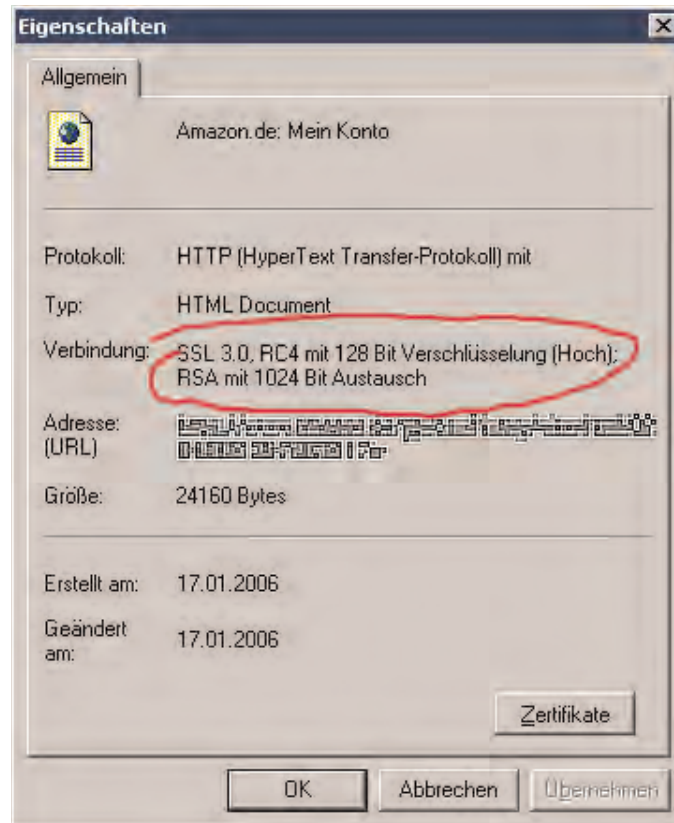


Abbildung 9.19 Sicherheitsinformationen im Internet-Explorer

Wer vertraut wem?

Jetzt ist nur noch eine Erkenntnis des Experiments auszuwerten: Wo bekommt man anfangs den öffentlichen Schlüssel der entsprechenden Certification Authority her? Wie oben bereits angedeutet besitzen heutige Browser bzw. die Betriebssysteme wie Windows bereits die wichtigsten Schlüssel fest eingebaut. Wenn also das Betriebssystem von einem Originalmedium installiert wurde, kann man diesen Schlüsseln im Allgemeinen vertrauen.

Selbstverständlich gibt es auch Zertifizierungsinstanzen, die noch nicht berücksichtigt sind. Diese haben dann aber normalerweise ihrerseits den eigenen öffentlichen Schlüssel wiederum bei einer der bekannten CAs zertifizieren lassen. Benötigt der Computer einen solchen Schlüssel, besorgt er ihn, überprüft ihn und speichert ihn dann als „vertrauenswürdig“ ab. Auf diese Weise vergrößert sich kontinuierlich der Kreis vertrauenswürdiger Kommunikationspartner.

Vielleicht widersprechen Sie, wenn ich hier von „bekannten CAs“ rede, denn die wenigsten Benutzer des Internets haben bisher von „DST“ oder „Equifax“ gehört. „Deutsche Telekom“ oder vielleicht auch „Verisign“ sind da schon verbreiteter. Allerdings ist es für Benutzer auch nicht wichtig, diese Namen zu wissen – nur der Internet-Browser bzw. das Betriebssystem muss die Daten gespeichert haben und das bezeichne ich in diesem Fall mit „bekannt“.

Vorsicht ist geboten, wenn ein Anbieter nicht das Geld ausgeben wollte, sich bei einer der großen CAs zu registrieren und seine eigene Zertifizierungsstelle benutzt. In diesem Fall werden Sie normalerweise vom Browser gefragt, ob Sie die Verbindung trotzdem eingehen möchten. Falls Sie das bejahen, erfolgt eine verschlüsselte Verbindung. Ohne korrektes Zertifikat einer vertrauenswürdigen Stelle können Sie jedoch nie sicher sein, dass nicht irgendein Hacker mithört oder gar die gesamte Kommunikation selbst bestreitet.

Ganz besonders kritisch ist, wenn Sie gefragt werden, ob Sie ein neues, eventuell unsicheres Zertifikat installieren möchten. Damit erweitern Sie das Netz Ihres Vertrauens. Alle dort ausgestellten Zertifikate gelten von diesem Moment an als sicher! Es ist daher ganz besonders wichtig, die Korrektheit zu überprüfen. Da das neue Zertifikat offenbar nicht von einer der bekannten CAs gegengezeichnet ist, müssen Sie sich auf eine andere Weise von der Vertrauenswürdigkeit überzeugen.

Ein wichtiges Mittel hierfür ist der so genannte „Fingerabdruck“. Es ist so eine Art Prüfcode, der für jeden Schlüssel nahezu eindeutig ist. Ein Anbieter könnte Ihnen einen Brief mit dem Fingerabdruck seines Zertifikats schicken. Einige Zeitschriften veröffentlichen ihren Fingerabdruck im Impressum. Durch den Vergleich des schwarz auf weiß vorliegenden Fingerabdrucks mit dem des Zertifikats können Sie dann die Echtheit prüfen.

Ist also Internet-Banking sicher?

Diese Frage muss sich jeder selbst beantworten und ich hoffe, Sie sind nach dem Durcharbeiten dieses Kapitels der Antwort ein Stück näher. Ich möchte aber trotzdem nochmals zusammenfassen, wem und was Sie alles vertrauen müssen, um dabei ein sicheres Gefühl zu haben. Selbstverständlich gelten die hier getroffenen Aussagen auch für jede andere Art von Internet-Geschäften.

Sie vertrauen der Bank?

Damit meine ich nicht das allgemeine Vertrauen in die guten Absichten – das setze ich mal voraus. Die Bank muss aber auch in der Lage sein, Angreifer aus ihrem Computersystem fernzuhalten. Wenn hier jemand eindringt, kann er nicht nur direkt alle Kundendaten ausspionieren, er könnte sich auch einfach den privaten Schlüssel der Bank aneignen, um dann zum Beispiel Überweisungen zu seinen Gunsten zu manipulieren.

Normalerweise können Sie jedoch davon ausgehen, dass Banken sehr viel Energie und Mittel in Abwehrsysteme gegen Hacker stecken. Momentan gibt es deutlich einfacherer Möglichkeiten, das System zu gefährden, als einen Angriff auf den Bankrechner.

Sie vertrauen der Zertifizierungsinstanz?

Die Zertifizierungsinstanz ist der Dreh- und Angelpunkt, wenn es um die sichere Übermittlung der öffentlichen Schlüssel geht. Wenn dort jemand in den Computer eindringt und den privaten Schlüssel stiehlt, kann er falsche Zertifikate ausstellen – zum Beispiel eines, das auf Ihre Bank lautet, aber den Schlüssel des Hackers beinhaltet.

Auch die Zertifizierungsinstanzen gelten heute als sicher. Sie werden staatlich überwacht und bisher wurden zumindest noch keine größeren Fälle von Internetkriminalität entdeckt, die auf einen Einbruch bei einer CA zurückzuführen sind. Falls die Ausspähung eines Schlüssels auffällt, kann dieser übrigens auch gesperrt werden.

Sie vertrauen dem asymmetrischen Verfahren?

Die erste sichere Kommunikation erfolgt mit RSA. Bisher ist niemand auf ein Verfahren gekommen, wie man aus dem öffentlichen Schlüssel in kürzerer Zeit den privaten Schlüssel errechnen kann. Es ist daher sehr unwahrscheinlich, dass es ein solches Verfahren gibt. Gleichzeitig konnte man allerdings bis heute auch nicht beweisen, dass ein solches Verfahren nicht existiert!

Allerdings: Ein großer Teil der Finanz- und Geschäftswelt setzt bei vertraulicher Kommunikation momentan auf RSA. Falls sich das Verfahren irgendwann als nicht sicher herausstellen sollte, haben Sie als Bankkunde ein Problem – ob Sie dann persönlich E-Banking genutzt haben oder nicht!

Es gibt daher meiner Meinung nach keinen Grund, dem RSA-Verfahren zu misstrauen. Allerdings müssen Sie darauf achten, dass die Verbindung mit einer ausreichend großen Schlüssellänge erfolgt (1.024 Bit und mehr).

Sie vertrauen dem symmetrischen Verfahren?

Hier gilt im Prinzip das Gleiche wie für das asymmetrische Verfahren: Bei großen Schlüssellängen kann man von der sicheren Funktion ausgehen. 128 Bit und mehr sollten ausreichen.

Einen Schlüssel mit 56 Bit Länge kann man demgegenüber mit einem normalen Rechner schon in einem Tag herausrechnen. Hier kommt es einfach darauf an, welche Informationen Sie über das Internet weitergeben. Ist Ihr Geheimnis zum Beispiel nur das Passwort zu einem Forum, wird sich wohl niemand die Mühe machen, einen Computer einen ganzen Tag lang zu beschäftigen, um es zu knacken, nur um dann in Ihrem Namen einen Beitrag zu schreiben. Handelte es sich dagegen um Ihren Zugangscode zu Dagobert Ducks Geldspeicher, sähe die Sache sicherlich anders aus...

Sie vertrauen Ihrem Computer?

Wir nähern uns langsam den relevanteren Fragen: Anders als Banken besitzen Sie sicherlich für den heimischen Computer keine Expertengruppe, die Hacker fernhält.

Wenn es jemandem gelingt, auf Ihrem Computer irgendwelche schädliche Software zu installieren, kann diese zum Beispiel ohne weiteres die vorgespeicherten öffentlichen Schlüssel der Certification Authorities verändern, ohne dass Sie es mitbekommen. Wenn dann auch noch der Internetverkehr – zumindest während E-Banking-Sitzungen – an die Adresse des Hackers umdirigiert wird, kann dieser alle Ihre Geheimnummern, Transaktionsnummern, Buchungen usw. mitlesen oder ggf. auch fälschen. Der Verlust von Daten ist also nicht das Schlimmste, was ein Computervirus oder ein so genannter Trojaner anrichten kann! Gegebenenfalls öffnet er auch die Tür in Ihr Bankkonto...

Leider bieten heutige Betriebssysteme immer noch nur unzureichenden Schutz vor solchen Einflüssen, wenn sie nicht professionell administriert werden. Daher ist es besonders für die Nutzer von Onlinediensten und Internet-Banking wichtig, Sicherheitshinweise ernst zu nehmen und Ihr Betriebssystem immer auf dem neuesten Stand zu halten, aktuelle Virenabwehr zu installieren usw.

Sie vertrauen sich selbst?

Die weitaus größte Gefahr besteht gar nicht wegen technischer Mängel oder Hackern, die sich sozusagen gewaltsam Einlass in Ihren Computer oder den der Bank verschaffen. Auch bei der Computerkriminalität hat sich etwas durchgesetzt, das am ehesten mit Trickdiebstahl vergleichbar ist.

Was tun Sie wenn Sie folgende E-Mail bekommen (setzen Sie als Bank Ihre eigene Hausbank ein)?

Von: Deutsche Finanzbank Kundenkontakte

An: Sie@IhreEmail.de

Betreff: Ihr Konto (hier: Online-Banking)

Sehr geehrter Geschäftspartner,
durch vermehrt auftretende Probleme im Zusammenhang mit Angriffen auf unser Online-Banking-System sind wir gezwungen, unsere Sicherheitsmaßnahmen zu verstärken. Wir überprüfen in diesem Zusammenhang unseren Kundenstamm und möchten Sie bitten, sich innerhalb der nächsten 7 Tage (zum Beispiel durch Click auf die unten stehende Adresse) bei unserem System normal anzumelden. Sonst entstehen für Sie keine weiteren Unannehmlichkeiten.

Vielen Dank für Ihr Verständnis.

Mit freundlichen Grüßen

Deutsche Finanzbank Kundencenter

Direkter Link zum Online-Banking:

Deutsche Finanzbank Online: <http://www.deutsche-finanzbank.de/>

Würden Sie den Instruktionen folgen? Schließlich ist ja nichts dabei, sich bei seiner Bank nur einzuloggen – die Übertragung ist doch sicher...

Wenn Sie das tun, indem Sie Ihrem gewohnten Verweis unter „Favoriten“ folgen oder manuell die Adresse in den Browser eingeben, haben Sie da auch völlig Recht. Problematisch wird es, wenn man der Bequemlichkeit halber den Verweis aus der E-Mail wählt.

Wenn Sie genau hinschauen, ist dort nämlich ein Schreibfehler drin. Jeder kann ganz legal diesen „verdrehten“ Namen auf sich registrieren lassen, ein Zertifikat beantragen und damit einen sicheren WWW-Service betreiben. Kriminell wird es dann, wenn die Ähnlichkeit zum echten Namen ausgenutzt wird und der WWW-Service dem ahnungslosen Internet-Benutzer vorspiegelt, sich tatsächlich auf den Seiten der Deutschen Finanzbank zu befinden.

Wenn Sie hier Ihre Geheimzahlen usw. eingeben, kann das zwar wirklich nur der Besitzer der Webseite lesen – der ist allerdings in diesem Fall ein ausgekochtes Schlitzohr und könnte mit den Daten in Ruhe Ihr Konto plündern!

So genannte „Phishing-Mails“, die ähnlich wie die oben abgedruckte aussehen, überschwemmen leider immer noch die Mailboxen von Leuten, die ein Online-Konto führen und solchen, die das nicht tun – die Betrüger habe ja keinen Zugriff auf die Kundendateien. Leider fallen tausende von Internet-Nutzern immer wieder darauf herein, selbst wenn die Original-Mails oft vor Rechtschreibfehlern und grammatikalischen Grausamkeiten strotzen – so dass eigentlich schon beim Lesen klar sein müsste, dass eine echte Bank eine solche Mail nicht schreibt.

Eine andere Gefahr stellen die doch sehr häufigen Fragen des Browser dar, ob man nicht dies oder das installieren wolle. Viele Benutzer betätigen da schon aus Gewohnheit den „OK“-Knopf. Falls es sich aber um die Nachfrage handelt, ob man ein unsicheres Zertifikat installieren wolle, sollte man das im Normalfall verneinen (wie bereits oben erklärt). Aber Hand aufs Herz: Sind Sie sicher, dass Sie wirklich immer alle Meldungen lesen, bevor Sie „OK“ bestätigen?

Die Banken versuchen, den Schaden abzumildern, der mit dem Ausspähen der Daten auf diese Weise entstehen kann – zum Beispiel durch die iTAN, bei der der Kunde zur Bestätigung eine ganz bestimmte Nummer aus einer ganzen Liste von Zahlen zur Bestätigung seiner Transaktion eingeben muss. So soll verhindert werden, dass ein Internet-Spion mit nur einer einzelnen ausgespähten TAN bereits viel Unheil anrichten kann. Trotzdem ist es natürlich besser, wenn Ihre Transaktionsnummern erst gar nicht in falsche Hände geraten.

Die hier aufgeführten Fälle sind nur einige Beispiele für Internet-Kriminalität, in denen die Unwissenheit und die Unsicherheit der Benutzer ausgebeutet werden. Spätestens nach der Lektüre dieses Kapitels kann Ihnen das aber ja nichts mehr anhaben!

Übrigens (weil ich das oft nach einem Vortrag über das Thema gefragt werde): Ich selbst nutze Internet-Banking ...

Was man nicht weiß...

Jetzt haben wir ausführlich darüber geredet, wie man seine Geheimnisse durch Verschlüsselung schützt. Die alten Griechen hatten da noch eine andere Methode: Die so genannte Steganografie. „Steganós“ ist griechisch und heißt einfach „versteckt“. Die zugrunde liegende Philosophie: Ein Geheimnis, das nicht entdeckt wird, ist sicherer als ein Geheimnis, das ein Unberechtigter nur nicht versteht.

Und so wurden wichtige Nachrichten einfach vor den Blicken anderer verborgen. Eine Methode war zum Beispiel, einem Sklaven den Kopf zu rasieren und dort die Nachricht zu tätowieren. Nachdem die Haare wieder gewachsen waren, schickte man ihn zum Adressaten.

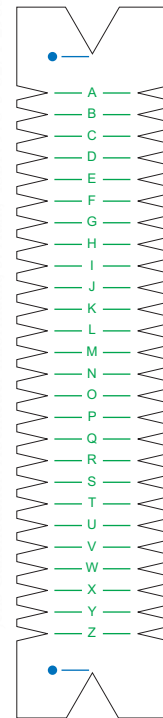
Heute geht man glücklicherweise nicht mehr ganz so martialisch vor, aber die Kunst des Versteckens ist nicht verloren gegangen, sondern wird noch fleißig in der Computertechnik verwendet. Die Datenflut nimmt immer mehr zu – im Internet gibt es Unmengen an Grafiken und Texten. Auf diese Weise lässt sich immer besser etwas verstecken, denn für jemanden, der nicht genau weiß, wo er suchen muss, wird es bei einem wachsenden Heuhaufen immer schwieriger, die Nadel zu finden.

Nachrichten kann man eigentlich überall verstecken. Um das zu demonstrieren, habe ich eine kleine Nachricht hier im Buch versteckt und zwar in der Randspalte dieses Kapitels. Suchen Sie oben und unten im Bereich dieser Spalte jeweils einen kleinen, blauen Punkt. Dies sind sozusagen die „Anker“ für Ihre Suchschablone aus dem Anhang. Wenn Sie die fertig gebastelt haben (bitte sehr präzise schneiden), sagt sie Ihnen, wo Sie die einzelnen Buchstaben der Nachricht finden können. Legen Sie die Schablone so an, dass Sie in den Aussparungen oben und unten genau die kleinen blauen Punkte sehen. Danach suchen Sie den kleinen grünen Punkt in den Aussparungen, die zum Rand der Seite weisen (also links für linke Seiten und rechts für rechte Seiten). Wenn Sie dies für jede Seite gemacht haben, ergibt sich eine kleine Botschaft.

Haben Sie vorher die kleinen Punkte bemerkt oder sich etwas dabei gedacht? Auf folgende Weise kann man in allen Sorten von Dokumenten auch unbemerkt Botschaften unterbringen:

- In Texten steht in definierten Abständen mal ein Leerzeichen mehr zwischen zwei Worten.
- Bei Bildern befinden an ganz bestimmten Stellen fast unsichtbar kleine Punkte.
- In Audiodateien (z. B. MP3) gibt es an leisen Stellen zusätzlich eine Millisekunde Pause.

Niemand kann diese Botschaften lesen, wenn er nicht weiß, wo er suchen muss, also das digitale Pendant Ihrer Schablone zu diesem Buch besitzt. So verschicken Sie zum Beispiel ein harmloses Urlaubsfoto an einen Freund und verstecken die eigentliche Nachricht darin.



Die Stegano-Schablone

Während sich dieses Verfahren tatsächlich sehr gut zum Verschicken geheimer Botschaften eignet, wird es in der Praxis doch noch häufiger für einen ganz anderen Zweck eingesetzt:

Im Zeitalter des Internet wird die Sicherung geistigen Eigentums immer schwieriger. Wie beweist ein Fotograf, dass ein Bild, welches im World Wide Web auftaucht, eigentlich von ihm ist? Wie stellt ein Filmverleih fest, welcher Vorführer die Vorschau-DVD illegal im Internet verbreitet?

Indem vor der Verbreitung unsichtbare Markierungen in dem Material untergebracht werden. Eine Botschaft wird versteckt. Das nennt man auch digitales Wasserzeichen, weil es wie das Wasserzeichen im Papier für den normalen Gebrauch unsichtbar bleibt. Nur wer weiß, wonach er suchen muss, kann die Botschaft lesen.

Auf diese Weise kann der Fotograf seine Urheberschaft nachweisen: Nur er kann zum Beispiel unter Aufsicht eines Notars seinen Namen aus dem Bild extrahieren, den er vorher steganografisch dort angebracht hat. Es gibt sogar Dienstleister, die man beauftragen kann, in regelmäßigen Abständen das Internet nach den eigenen, per Wasserzeichen markieren Bildern und anderen Dokumenten zu durchforsten.

Auch ein Filmverleih kann jedes verschickte Exemplar einer Vorschau-DVD personalisieren, also an versteckten Stellen Hinweise anbringen. Wenn dann eine Kopie irgendwo im Netz auftaucht, kann man diese Hinweise analysieren und weiß, wer sie in Umlauf gebracht hat.

Eine wesentliche Frage bleibt: Warum müssen diese Markierungen eigentlich versteckt werden? Eigentlich darf doch jeder lesen, wer der Urheber eines Bildes ist oder um wessen legale Filmkopie es sich bei einer DVD handelt! Das stimmt zwar – der Vorteil unsichtbarer Kennzeichnungen ist allerdings, dass man sie erst finden muss, bevor man sie entfernen kann.

Ein ganzer Forschungszweig beschäftigt sich damit, solche Wasserzeichen so prägnant anzubringen, dass sie nie rückstandsfrei entfernt werden können. So sind die Urheberrechtsinformationen von Bildern auch dann noch vorhanden, wenn man sie ausdruckt und danach mit einem Kopierer vervielfältigt.

Jemand, der sich das Material aneignen möchte und sich daher bemüht, die Wasserzeichen zu entfernen, kann dank Steganografie nie sicher sein, dabei wirklich alle erwischt zu haben.

Resümee

In diesem Kapitel haben Sie selbst erforschen können, welche Probleme bestehen, um sicher im Internet zu kommunizieren. Wesentlich ist dabei der erstmalige Austausch eines Schlüssels. Asymmetrische Verschlüsselungsverfahren ermöglichen diesen Austausch auch mit Kommunikationspartnern, die man vorher nicht kennt.

Zusammenfassend habe ich die notwendigen Schritte für einen sicheren Datenaustausch nochmals in einer Grafik zusammengefasst, diesmal mit der Deutschen Finanzbank als Anbieter und Jennifer als Kunden. Abbildung 9.20 zeigt, wie die Bank zunächst zu ihrem Zertifikat kommt, das die Voraussetzung für die sichere Kommunikation mit einem Kunden ist.

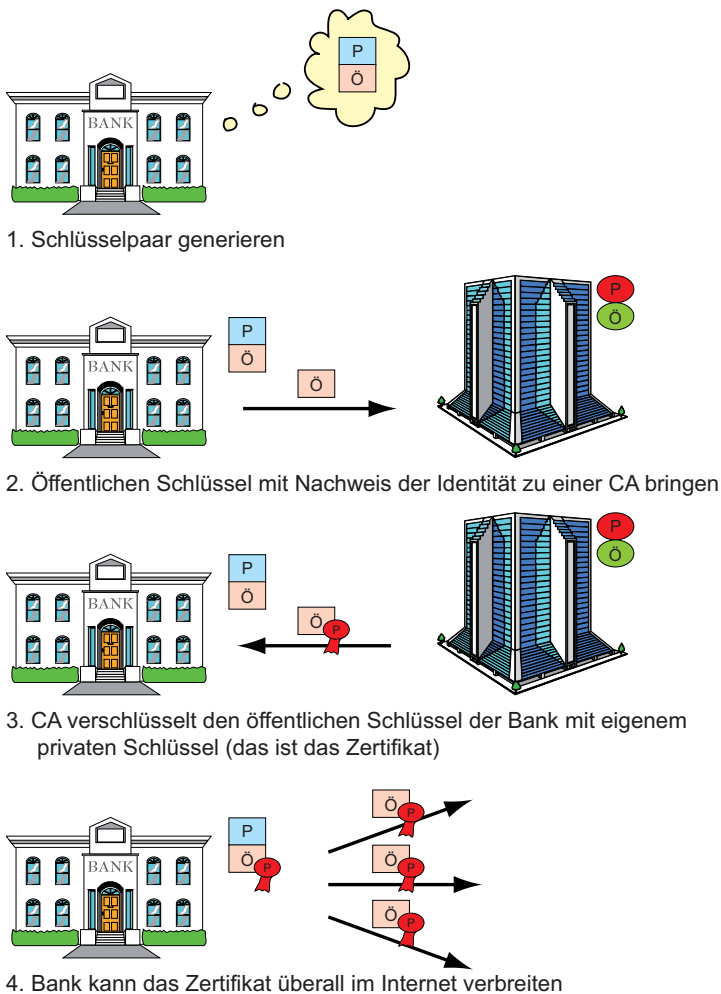
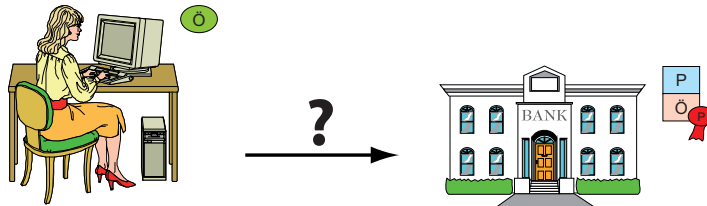


Abbildung 9.20 Die Bank bekommt ihr Zertifikat

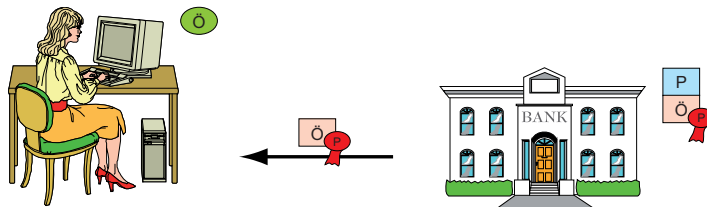
Abbildung 9.21 zeigt schließlich, wie ein symmetrischer Schlüssel ausgetauscht wird, damit Jennifer sicher mit ihrer Bank Überweisungen und andere Nachrichten austauschen kann.



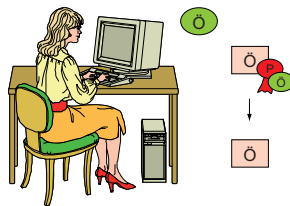
1. Jennifer ist lediglich in Besitz des öffentlichen CA-Schlüssels



2. Jennifer fragt die Bank (oder auch eine CA) nach dem Bank-Zertifikat



3. Die Bank schickt Jennifer ihr Zertifikat

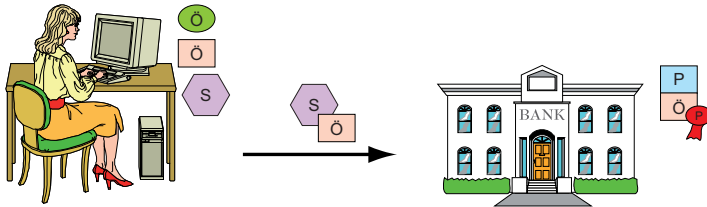


4. Jennifer nutzt den öffentlichen Schlüssel der CA, um das Zertifikat zu entschlüsseln und damit die Echtheit zu überprüfen



5. Jennifer erzeugt zufällig einen Schlüssel für ein symmetrisches Verfahren

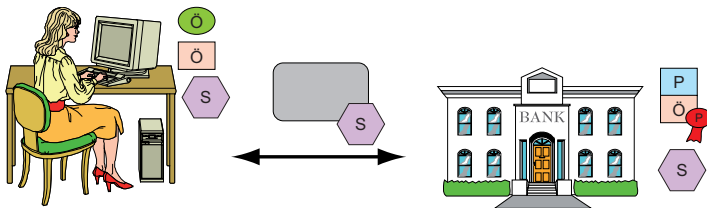
Abbildung 9.21 Aufnahme einer sicheren Kommunikation zwischen Jennifer und der Finanzbank (Fortsetzung auf der nächsten Seite)



6. Jennifer verschlüsselt den symmetrischen Schlüssel mit dem öffentlichen Schlüssel der Bank und verschickt ihn



7. Nur die Bank kann die Nachricht mit ihrem privaten Schlüssel entziffern und damit den symmetrischen Schlüssel akquirieren



8. Da beide Parteien jetzt in Besitz des symmetrischen Schlüssels sind, können sie nun über diesen sicher Nachrichten (zum Beispiel Überweisungen und Kontoauszüge) austauschen

Abbildung 9.21 (Fortsetzung)
Aufnahme einer sicheren Kommunikation zwischen Jennifer und der Finanzbank

Die hier beschriebenen technischen Verfahren helfen jedoch auch nicht weiter, wenn ein Benutzer auf den Internet-Trickbetrug mit gefälschten E-Mails oder Web-Adressen hereinfällt. Glücklicherweise kann Ihnen das ja jetzt nicht mehr passieren...

Literatur

Buch: Claudia Eckert „**IT-Sicherheit**“; Oldenbourg Verlag; 3. Auflage 2004; ISBN 3-486-20000-3



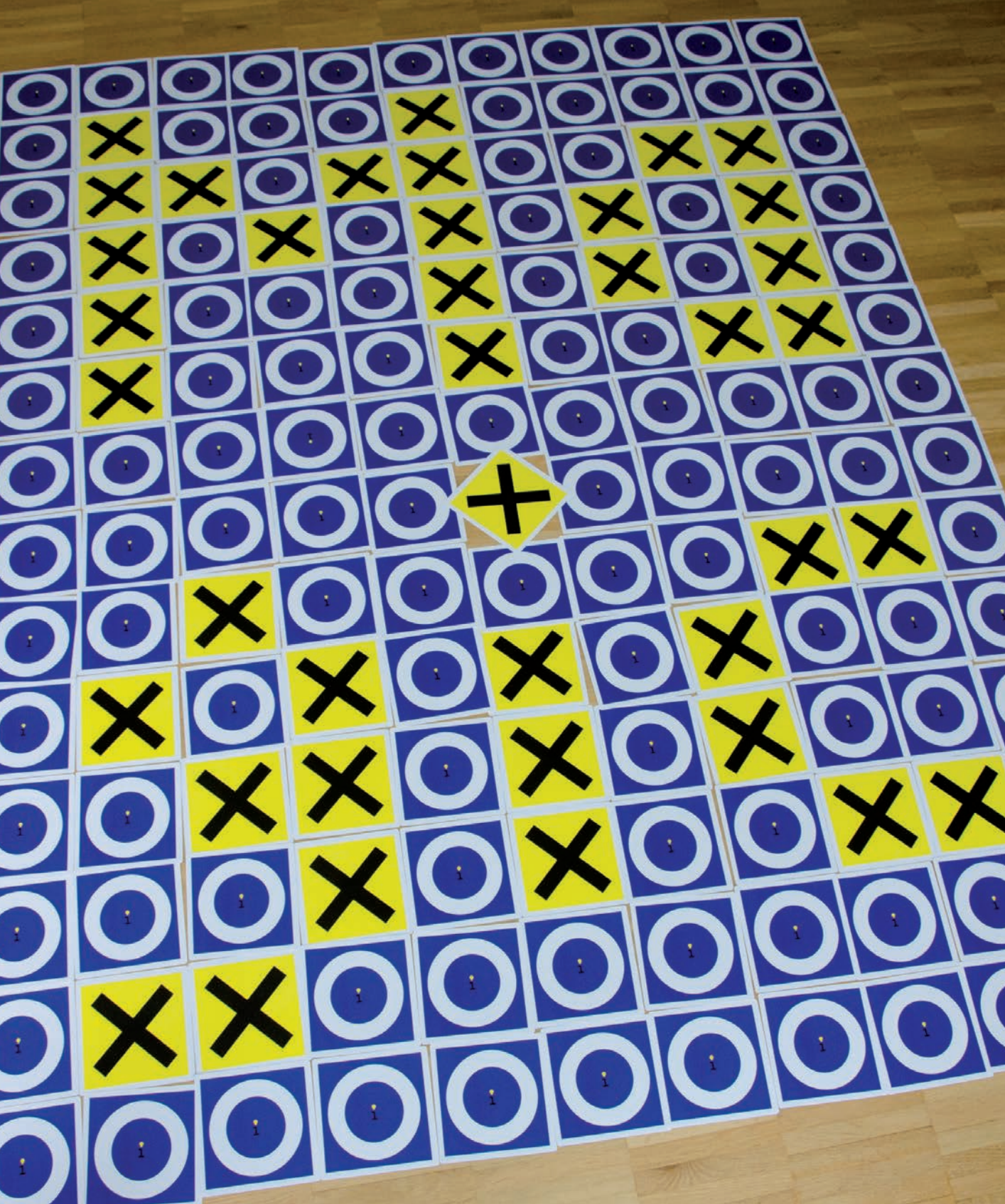
Hier erfahren Sie alles über die neuesten Erkenntnisse zur Sicherheit von Informationssystemen. Verschlüsselungstechnologien spielen hier eine große Rolle. Zum Beispiel steht im Buch eine Beschreibung der genauen Funktionsweise von RSA sowie ein kurzer Abriss über die mathematischen Hintergründe.

Hinweis (für Skeptiker des ASYM-Codierers):

Vielleicht haben Sie jetzt bereits erkannt, dass der ASYM-Code genau wie der Cäsar-Code vom Anfang dieses Kapitels sehr schwach ist. Man kann ihn leicht knacken und mit etwas mehr Aufwand kann man einen verschlüsselten Text auch anhand des ursprünglichen Schlüssels wieder entziffern. Genau genommen sind sogar beide Verschlüsselungen symmetrisch.

Später zeige ich Ihnen im Abschnitt „Was steckt dahinter“ auch wirklich sichere Verfahren. Diese sind allerdings so aufwendig, dass man mit ihnen kaum experimentieren kann. Daher gehen wir für unsere eigenen Versuche einfach davon aus, der ASYM-Code sei sicher! Auf diese Weise können Sie den Experimenten mit Spaß folgen und durch sie auch erkennen, warum die „echten“ Verschlüsselungsverfahren so gut im Internet funktionieren.

Lesen Sie jetzt das Kapitel dort weiter, wo Sie zu diesem Hinweis gesprungen sind.



11. InformaGik

Einführung

Zum Einstieg in dieses Kapitel würde ich Ihnen gerne ein kleines Zauberkunststück vorführen. Für einen solchen Trick benötigen wir allerdings noch einen Zauberkünstler, der in die Geheimnisse eingeweiht ist. Bevor Sie weiterlesen, sollten Sie sich daher entscheiden, ob Sie sich den Trick von jemand anderem vorführen lassen möchten. In diesem Fall geben Sie dem designierten Zauberkünstler das Buch und bitten ihn, die Vorführanleitung am Ende dieses Kapitels zu lesen. Der Trick ist recht einfach und kann auch von in der Magie ungeübten Menschen durchgeführt werden.

Nachdem Sie selbst den Aha-Effekt erlebt haben, können Sie die Anleitung lesen und Zauberer spielen. Bitte denken Sie daran, im Anschluss immer zusammen mit Ihrem Publikum zu überlegen, wo in Ihrem Alltag eine Anwendung zu finden ist, die eine ganz ähnliche „InformaGik“ verwendet.

Information

Erinnern Sie sich? Schon in Kapitel 5 haben wir uns mit Information und deren Codierung beschäftigt. Dort konnten wir die Häufigkeit einzelner Informationseinheiten (Buchstaben) ausnutzen, um diese so sparsam wie möglich durch Folgen von 0 und 1 auszudrücken. Auch in diesem Kapitel geht es um Information: Ein Feld aus X und O enthält 25 unabhängige Informationseinheiten. In diesem Fall sind sie zufällig von Freiwilligen als Muster gelegt worden, aber es könnte sich durchaus auch um einen kleinen Text in binärer Codierung handeln.

Diese Informationen werden vom Zauberkünstler ergänzt – allerdings nur scheinbar zufällig: Er folgt einer festen Regel, so dass danach in jeder Zeile und Spalte immer eine gerade Anzahl von X liegt. Das Legen wird auf diese Weise eindeutig, man hat keinerlei Wahlfreiheit.

Was bedeutet das für den Informationsgehalt des Spielfelds? Die Zahl der Karten ist größer geworden. Bedeutet das, dass nun auch der Informationsgehalt von 25 auf 36 „Einheiten“ gestiegen ist?

Denken Sie bei der Beantwortung an Ihre Erfahrungen aus Kapitel 5: Einen Text können wir durch geschickte Codierung nach dem Verfahren von Huffman deutlich im Umfang reduzieren, ihn aber jederzeit wieder in die ursprüngliche Form bringen. Der Informationsgehalt ändert sich durch diese Codierung daher sicherlich nicht.

Können Sie aus dem 6×6 -Feld aus X und O ebenfalls Informationen wegnehmen, so dass dieses trotzdem wieder komplett rekonstruierbar ist?



Selbstverständlich lassen sich die zusätzlich gelegte Zeile und Spalte wieder entfernen, denn diese können wir jederzeit nach der Regel wieder neu legen, nach der sie ursprünglich entstanden sind. Wenn wir genau darüber nachdenken, können wir sogar eine beliebige Zeile und eine beliebige Spalte aus dem Feld entfernen! Mit der Regel, Zeilen und Spalten auf eine gerade Zahl von X zu ergänzen, sind wir auch jetzt noch in der Lage, das Muster wieder herzustellen. Versuchen Sie es in Abbildung 11.1.

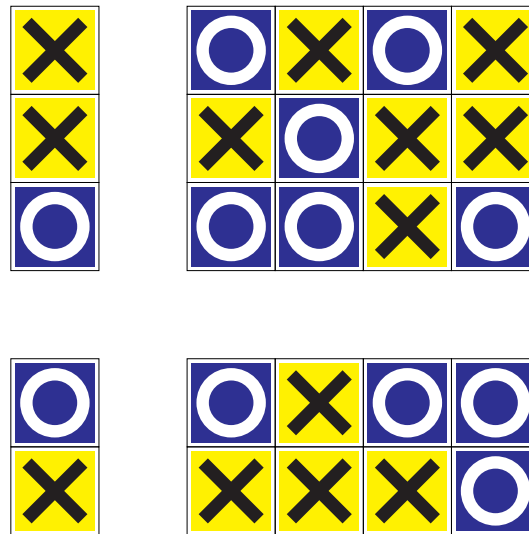


Abbildung 11.1 In dem 6×6 -Muster fehlt die zweite Spalte und die vierte Zeile. Können Sie diese wieder herstellen?

Wenn man aber etwas weglassen kann und danach immer noch keine Information verloren ist, kann das Weggelassene ganz offensichtlich keine Information enthalten. Wozu soll es also dienen, so etwas hinzuzufügen?

In unserem Zaubertrick ermöglichten uns die zusätzlichen Karten die Identifikation eines Fehlers, den in diesem Fall jemand absichtlich durch Umdrehen einer Karte erzeugt hat. Fehler passieren aber in unserer Umgebung auch unbeabsichtigt:

- Beim Abschreiben von Zahlenkolonnen verwechseln Sie eine Ziffer oder vertauschen zwei benachbarte Stellen.
- Durch Nebengeräusche im Telefon notiert jemand Ihren Namen falsch.
- Ein Autokennzeichen ist verschmutzt und so wird aus KU für Kulmbach KO für Koblenz.

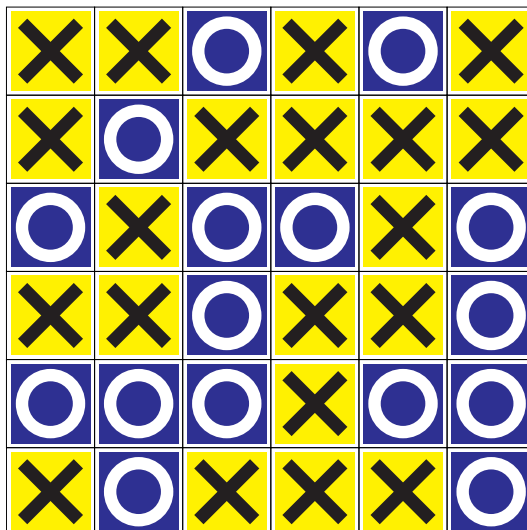
Während solche Fehler durch höhere Sorgfalt reduziert werden könnten, tun wir uns besonders in Systemen, die sehr große Datenmengen auf engstem Raum speichern, damit sehr schwer: Die sogenannten Pits auf einer Blu-Ray-Disc haben einen Durchmesser von $0,15\text{ }\mu\text{m}$, die Spuren einen Abstand von $0,32\text{ }\mu\text{m}$. Auf eine Fläche von einem Millimeter mal einem Millimeter kommen also pro Lage etwa 20 Millionen Informationseinheiten oder 2,5 MByte.

Bei einer zweilagigen Scheibe macht also selbst ein winziges Staubkorn mit $10\text{ }\mu\text{m}$ Durchmesser (auf einem Quadratmillimeter haben 10.000 dieser Teilchen an der Grenze zum Feinstaub Platz) bereits etwa 4.200 Pits unlesbar, was immerhin etwa dem Informationsgehalt des Texts auf einer Buchseite entspricht. Da kaum jemand im Wohnzimmer einen staubfreien Reinraum unterhält, sind solche Partikel unvermeidbar. Es ist daher notwendig, den Staub zu akzeptieren: Die Fehler müssen irgendwie erkannt und korrigiert werden.

Fehlerkorrektur

Beim Vorführen des Zaubertricks vom Anfang des Kapitels haben wir uns daher implizit mit einer weiteren Bedeutung des Begriffs „Datensicherheit“ beschäftigt: der Sicherung von Daten gegen unbeabsichtigtes Verfälschen. Zusätzliche – fachsprachlich „redundante“ – Daten erhöhen den Informationsgehalt eines Datensatzes nicht, aber sie ermöglichen, dass eine gewisse Zahl von Fehlern erkannt und unter Umständen auch korrigiert werden kann.

Das wollen wir gleich an unserem Zauberquadrat ausprobieren: Wenn Sie von einer korrekten 6×6 -Anordnung ein Kärtchen umdrehen, können Sie sofort erkennen, welches das war, und es wieder korrigieren.



Redundanz

In der römischen Antike sprach man hauptsächlich von „redundare, wenn Flüsse über die Ufer getreten waren. In der Technik hat sich der Begriff heute für etwas etabliert, was über das absolut Notwendige hinaus vorhanden ist. In Flugzeugen sorgen etwa redundante Systeme dafür, dass ein abstürzender Computer nicht zum Absturz der Maschine führt. Redundante Daten sorgen für Fehlertoleranz.

Abbildung 11.2 Finden Sie Fehler im Muster?

Nun drehen Sie einmal zwei unterschiedliche Kärtchen um. Versuchen Sie, herauszufinden, ob Fehler in Abbildung 11.2 vorliegen und wenn ja, diese zu korrigieren – jeweils ohne die Abbildung mit dem „Original“ vom Ende des Kapitels zu vergleichen.



Wir stellen fest, dass die zweite und die fünfte Zeile sowie die zweite und die vierte Spalte nicht der Regel entsprechen. Allerdings könnten wir nicht mit Gewissheit sagen, welche Fehler zu dieser Konstellation geführt haben: Es wäre möglich, dass entweder die in Abbildung 11.3 grün markierten oder die rot markierten Karten umgedreht wurden.

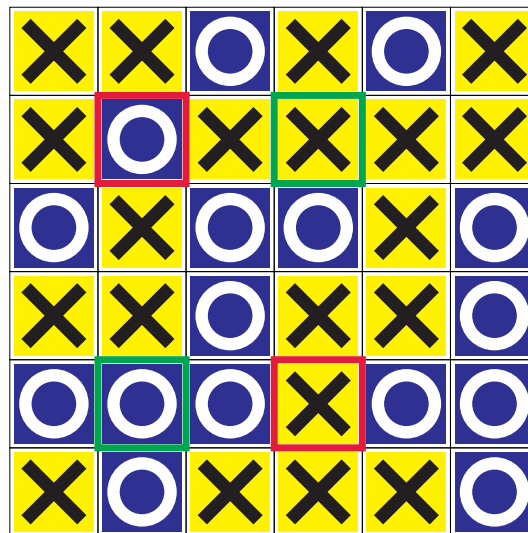


Abbildung 11.3 Entweder die grün oder die rot markierten Karten sind herumgedreht worden.

Versuchten Sie, den Fehler zu korrigieren, hätten sie mit gleicher Wahrscheinlichkeit den Fehler beseitigt oder verschlimmert, indem dann nicht zwei, sondern vier Kärtchen falsch lägen. In diesem Fall würde man sich also eher dafür entscheiden, gar nicht zu korrigieren. Ein Computersystem könnte etwa auf den Fehler aufmerksam machen oder – falls es sich etwa um ein Datenpaket aus dem Internet handelt – die Daten erneut anfordern.

Noch schlechter sieht die Sache aus, wenn wir drei Kärtchen umdrehen, die drei Ecken eines Rechtecks beschreiben. In Abbildung 11.4 sind das die drei rot markierten Kärtchen. Folgen wir dem Fehlerkorrekturschema aus dem Zaubertrick, würde wir die rot markierte Karte umdrehen und damit die Fehlerzahl auf vier erhöhen, statt sie zu senken.

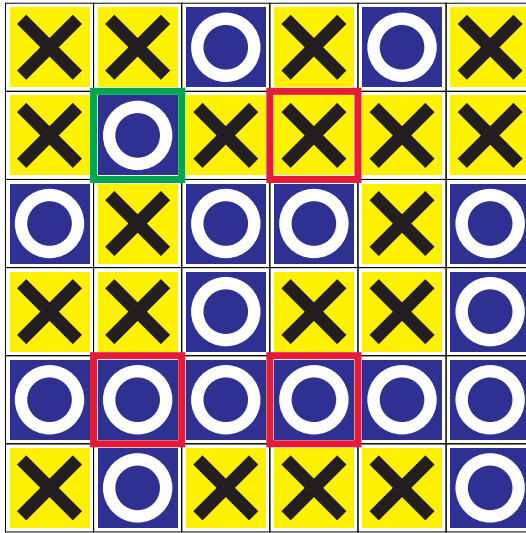


Abbildung 11.4 Statt die drei Fehler zu korrigieren, würde ein Korrekturschema die grün umrandete Karte umdrehen

Spielen Sie eine Weile selbst mit den Karten und untersuchen Sie, wie viele Fehler zu einem erkennbaren Regelverstoß führen und in welchen Fällen Sie die Fehler mit Gewissheit korrigieren können.



Vielleicht kommen Sie auf eine Antwort wie

„Ein Fehler lässt sich erkennen und korrigieren, zwei Fehler lassen sich erkennen und drei Fehler lassen sich ebenfalls erkennen.“

Das stimmt leider nur fast. Mit dem letzten Beispiel habe ich demonstriert, wie die Fehlerkorrektur die Situation sogar verschlimmert, wenn aus drei Fehlern vier werden. Andererseits kann man den Fehler doch erkennen, aber nicht korrigieren ...

Richtig wäre daher eine Antwort wie

„Man kann sich entscheiden, ob man Fehler korrigiert oder nicht.“

- Korrigiert man Fehler, kann man einen Fehler korrigieren, zwei Fehler noch erkennen.
- Verzichtet man auf die Korrektur, kann man bis zu drei Fehler erkennen.“

Genau das gilt prinzipiell für alle Verfahren, die mit Hilfe zusätzlicher Daten Informationen toleranter gegenüber Fehlern machen: Man muss sich jeweils entscheiden, ob man Fehler nur erkennen oder auch korrigieren möchte. Die Korrektur funktioniert nur mit einer kleineren Anzahl von Fehlern und man muss bei mehr Fehlern eine falsche Korrektur in Kauf nehmen.

Die Zahl korrigierbarer bzw. erkennbarer Fehler lässt sich durch mehr zusätzliche Informationen steigern. So könnte man auf die Idee kommen, zwei Zeilen und

Spalten zu ergänzen und etwa die Zahl der X für die 1., 3., 5. und 7. Zeile bzw. Spalte und für die anderen getrennt zu berechnen, wie in Abbildung 11.5 gezeigt.

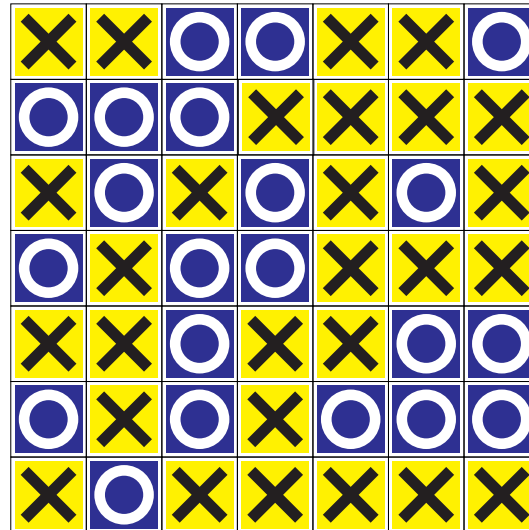


Abbildung 11.5 Mehr Fehler können mit mehr zusätzlichen Daten korrigiert werden

Schleichen sich nun zwei Fehler ein, können Sie diese meistens erkennen und korrigieren. Abbildung 11.6 zeigt ein auf diese Weise entstandenes Quadrat (mit einer anderen Information als oben), in dem zwei Fehler versteckt sind. Finden Sie diese?

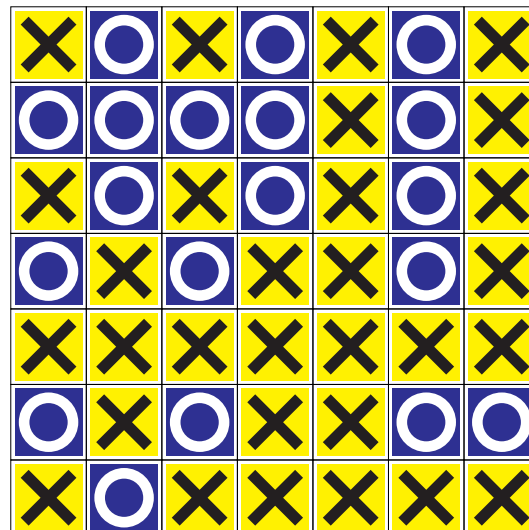


Abbildung 11.6 Finden Sie die beiden Fehler!



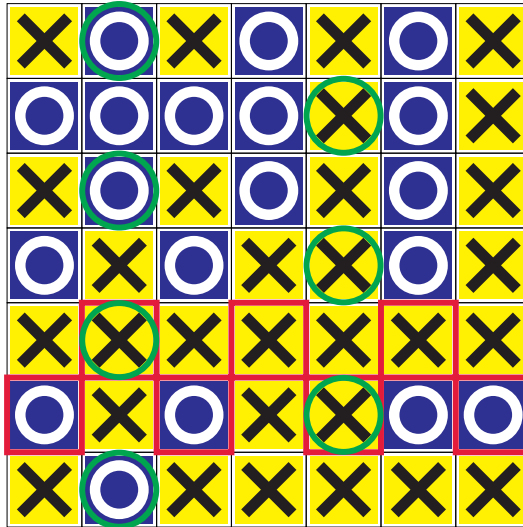


Abbildung 11.7 Potenzielle Fehlerquellen

Wieder halten wir nach Fehlern in den Zeilen und in den Spalten getrennt Ausschau. In Abbildung 11.7 habe ich die potenziell fehlerhaften Felder mit roten Quadraten (für fehlerhafte Zeilen) und grünen Kreisen (für fehlerhafte Spalten) markiert. Dort, wo ein Feld sowohl für einen Spalten- als auch für einen Zeilenfehler verantwortlich ist (und daher mit Quadrat und Kreis markiert ist), ist der Fehler am wahrscheinlichsten – wir sollten daher genau diese Felder korrigieren.

Auf diese Weise lassen sich nun auch zwei Fehler erkennen. Überlegen Sie, ob dies für alle möglichen Fehlerfälle so ist.

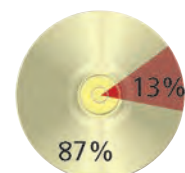


Leider gilt das nicht immer! Durch die Trennung in gerade und ungerade Zeilen bzw. Spalten haben wir lediglich unser Quadrat mit der Information sozusagen in zwei kleinere Teilquadrate aufgeteilt – so als würden Sie bei einem Schachbrett hintereinander zuerst nur die schwarzen und danach die weißen Felder betrachten. Auf diese Weise wären zwei Fehler dann korrigierbar, wenn sie sich auf unterschiedlichen Feldfarben befänden – zwei Fehler, die beide auf den „schwarzen“ Feldern wären, könnten wir immer noch nicht einordnen. Unser Muster ist noch etwas komplizierter als das Schachbrett, Sie können das aber trotzdem ausprobieren, indem Sie zwei Karten umdrehen, die auf einer horizontalen, vertikalen und diagonalen Gerade liegen und zwischen denen sich noch genau eine weitere Karte befindet.

Auf jeden Fall ist unser neues Verfahren aber gewappnet gegen einen häufigen, menschlichen Fehler: das Vertauschen zweier nebeneinanderliegender Karten.

Fehlerkorrektur von DVD oder Blu-Ray

Bei DVDs oder Blu-Ray-Discs wird von der gesamten möglichen Speicherkapazität nur etwa 87 % für „echte Daten“ ausgenutzt, der Rest wird für redundante Daten investiert, die eine umfangreiche Fehlerkorrektur ermöglichen. Diese Korrektur ist darüber hinaus auf die Besonderheit angepasst, dass Fehler wahrscheinlich lokal sehr konzentriert auftreten (durch ein Staubkorn oder eine Beschädigung).



Diese befinden sich sowohl horizontal als auch vertikal immer in unterschiedlichen „Bereichen“ (geraden oder ungeraden Zeilen oder Spalten).

Auch bei den wirklich eingesetzten Fehlerkorrekturverfahren lassen sich bestimmte Fehlerarten besser korrigieren als andere und auch dort liegt ein besonderes Augenmerk auf dem Vertauschen von Stellen, den beliebten „Zahlendreher“. Das können Sie anhand Ihrer Kreditkarte oder Ihres Personalausweises leicht überprüfen: Die Nummern werden öfters auch einmal per Telefon weitergegeben oder abgetippt (z. B. wenn Sie online etwas damit bestellen). Dabei passiert es leicht, dass zwei Ziffern in die falsche Reihenfolge kommen. Aus diesem Grund gibt es auch hier die sogenannte „Prüfziffer“, die so ergänzt wird, dass die gesamte Zahlenkolonne eine festgelegte Regel einhält, die durch Vertauschen oder Vertippen gestört wird.

Bei der Kreditkarte wird einfach abwechselnd jede Ziffer mit 2 und mit 1 multipliziert. Von den Ergebnissen addiert man wiederum die einzelnen Ziffern. Diese Summe muss ein Vielfaches von 10 sein.



Für unsere nebenstehende Beispiel-Kreditkarte lautet die Rechnung: $4 \cdot 2 = 8$, $0 \cdot 1 = 0$, $5 \cdot 2 = 10$, $6 \cdot 1 = 6$, $4 \cdot 2 = 8$, $5 \cdot 1 = 5$, $8 \cdot 2 = 16$, $7 \cdot 1 = 7$, $8 \cdot 2 = 16$, $9 \cdot 1 = 9$, $7 \cdot 2 = 14$, $5 \cdot 1 = 5$, $1 \cdot 2 = 2$, $2 \cdot 1 = 2$, $3 \cdot 2 = 6$, $2 \cdot 1 = 2$. Alle Ziffern der Resultate werden addiert: $8 + 0 + 1 + 0 + 6 + 8 + 5 + 1 + 6 + 7 + 1 + 6 + 9 + 1 + 4 + 5 + 2 + 2 + 6 + 2 = 80$. 80 ist ein Vielfaches von 10, daher hat diese Kreditkarte zumindest eine gültige Nummer.

Probieren Sie das doch gleich einmal mit Ihren Kreditkarten aus. Wie verändert sich die Summe, wenn Sie zwei Ziffern vertauschen?

Die Nummern verraten noch mehr:

Wussten Sie, dass man aus der Nummer auch ermitteln kann, um welche Art Kreditkarte es sich handelt? So beginnt zum Beispiel die Nummer einer MasterCard immer mit 51 bis 55, eine Visa Karte mit 4 und eine BahnCard mit 70.



Durch das abwechselnde Multiplizieren mit 1 und 2 fallen Zahlendreher in den allermeisten Fällen auf. Gegenüber bestimmten Vertauschungen ist das Verfahren allerdings blind. Erkennen Sie, welche das sind?



Zahlendreher bei den Ziffern 0 und 9 fallen nicht auf, da beide Ziffern auf jeder Position den gleichen Teil zur Prüfsumme beitragen: 0 ergibt multipliziert mit 1 und 2 ohnehin die gleiche Zahl. $9 \cdot 1 = 9$ und $9 \cdot 2 = 18$. Da aber die Ziffern einzeln in die Prüfsumme eingehen, ergibt sich $1 + 8 = 9$.

Nicht besser ist das Verfahren beim EU-Personalausweis oder Reisepass: Der nebenstehende Ausweis einer berühmten Persönlichkeit dient uns als Beispiel. Hier ist explizit festgelegt, dass immer die letzte Ziffer eines Blocks (Ausweisnummer, Geburtsdatum, Ablaufdatum) eine Prüfziffer darstellt, die folgendermaßen berechnet wird:



Die Ziffern werden abwechselnd nacheinander mit 7, 3 und 1 multipliziert. Die Prüfziffer ist die letzte Ziffer der Summe. Buchstaben (wie das „M“) spielen dabei keine Rolle. Für die Ausweisnummer gilt daher die Rechnung $5 \cdot 7 + 0 \cdot 3 + 0 \cdot 1 + 4 \cdot 7 + 5 \cdot 3 + 6 \cdot 1 + 7 \cdot 7 + 8 \cdot 3 + 9 \cdot 1 = 35 + 0 + 0 + 28 + 15 + 6 + 49 + 24 + 9 = 166$. Die Prüfziffer ist also 6.

Am Ende wird dann auf die gleiche Weise noch eine Gesamtprüfziffer gebildet, für die alle Ziffern (einschließlich der drei Einzelprüfziffern) berücksichtigt werden.

Wollen Sie das Verfahren mit den Daten auf dem Beispielausweis ausprobieren? Oder Sie überprüfen gleich den eigenen! Welche Zahlendreher fallen hier nicht auf?



Zum Beispiel sind 0 und 5 vertauschbar (da die 5 auf jeder Position wieder eine 5 zur Prüfsumme beiträgt). Genauso aber etwa 1 und 6, da in jeder möglichen Position beim Vertauschen die gleichen Anteile herauskommen.

Beheben könnte man das Problem durch Kombination der Verfahren: Für die einzelnen Prüfziffern bei Ausweisnummer, Geburtsdatum und Verfallsdatum würde das bisherige Verfahren angewendet, für die Gesamtprüfziffer aber das Verfahren der Kreditkarte. Da beide Verfahren unterschiedliche Anfälligkeiten bezüglich Vertauschungen haben, würde eine solche in jedem Fall auffallen. Offenbar hätte man bei der Einführung besser einen Informatiker zurate gezogen ...

Was steckt dahinter?

Sie haben es bereits bemerkt: Alles dreht sich in diesem Kapitel wieder um Codierung: die Darstellung einer Informationseinheit durch eine Folge von Symbolen – in der Informatik meistens aus dem sehr eingeschränkten Alphabet „0“ und „1“.

Man kann nun jede mögliche Folge von Symbolen (dem Code) einer bestimmten Informationseinheit zuordnen. Auf diese Weise können wir etwa ein Alphabet aus 8 unterschiedlichen Zeichen durch je 3 Bit codieren:

A	B	C	D	E	F	G	H
000	001	010	011	100	101	110	111

Die Folge 011000010111 lässt sich so in „DACH“ übersetzen. Jedes einzelne Bit ist jedoch relevant – wird es falsch übertragen, so kommt eine andere Nachricht heraus. 011001010111 steht zum Beispiel für „DBCH“.

Das kennen wir auch in unserer Sprache: sichtbar gemacht oft von unseren jüngsten Erdenbewohnern, die neue Begriffe mit dem ihnen verfügbaren Sprachschatz erschließen. So merkt sich der eine oder andere, dass ein Walzwerk Lieferant für Eisenplatten ist, weil Papa 'mal so etwas vom Walzwerk geholt hat. Andere wundern sich, warum Apfelschalen einerseits fest sind, es sie aber andererseits auch als Getränk gibt („Apfelschorle“) ...

Andere Worte würde man nie verwechseln: „Stoppschild“ oder „Boot“ oder „Haus“ – es gibt keine anderen Worte, die ähnlich klingen, aber eine unterschiedliche Bedeutung haben. Richard Hamming hat diese Erkenntnis auf Codes in der Informatik übertragen: Ein Code ist gegen Fehler sicherer, wenn die verwendeten „korrekten“ Symbolfolgen möglichst unterschiedlich sind.

Versuchen wir unser Glück mit folgender Codetabelle:

A	B	C	D	E	F	G	H
000000	000111	011001	011110	101010	101101	110011	110100

Das Wort „DACH“ benötigt nun doppelt so viele Bits wie vorher:

011110000000011001110100.

Wenn sich nun ein Fehler einschleicht, ergibt sich vielleicht

011110001000011001110100.

Versucht nun jemand, die Nachricht zu lesen, kann er das „D“ vorne und „CH“ hinten identifizieren. Das zweite Zeichen findet er aber nicht in der Zeichentabelle: 001000 gibt es nicht. Nun kann man raten, welches Zeichen wohl gemeint war. Dazu vergleichen wir die empfangene Zeichenfolge mit allen möglichen korrekten Zeichenfolgen:

A	B	C	D	E	F	G	H
000000	000111	011001	011110	101010	101101	110011	110100
001000	001000	001000	001000	001000	001000	001000	001000

Dabei gibt es einen eindeutigen Sieger – das A. Der Code unterscheidet sich nur an einer Stelle vom empfangenen Code und wenn wir davon ausgehen, dass es deutlich wahrscheinlicher ist, dass nur ein Bit verfälscht wurde als mehrere Bits, können wir die Nachricht korrekt zu „DACH“ korrigieren.

In der verwendeten Code-Tabelle unterscheiden sich zwei beliebige Codes immer an mindestens drei Stellen. Man spricht hier von einer Hamming-Distanz von 3. Auf diese Weise gibt es bei einem einzelnen Bitfehler immer einen einzelnen Code, der die höchste Übereinstimmung mit der empfangenen Nachricht hat. Die kleinste Hamming-Distanz, die zwei beliebige Codes einer Code-Tabelle haben, bestimmt die Hamming-Distanz der Tabelle.

Codes mit einer höheren Hamming-Distanz sind also noch robuster gegenüber Fehlern, da die korrekten Codes so unterschiedlich sind, dass ggf. auch mehrere Fehler erkannt und korrigiert werden können.

Versuchen Sie doch einmal, die Hamming-Distanz der folgenden Code-Tabelle herauszuknobeln. Wie viele Fehler können korrigiert werden?

A	B	C	D	E	F	G	H
00000000000	00000111111	00111000111	00111111000	11001001011	11001110100	11110001100	11110110011



Die Codes haben untereinander mindestens die Hamming-Distanz 6. In einer auf diese Weise codierten Nachricht dürfen pro Zeichen 2 Bits falsch übertragen werden. Drei Fehler können immer noch erkannt, aber eventuell nicht mehr korrekt korrigiert werden: Empfängt man die Bitfolge 11010001111, kann hier mit gleicher Wahrscheinlichkeit ein „E“ oder ein „G“ gesendet worden sein – zu beiden hat die empfangene Nachricht die gleiche Hamming-Distanz von 3.

Wenn sich mehr als 3 Fehler einschleichen, würde man allerdings falsch korrigieren. Werden zum Beispiel die letzten 4 Bits für „B“ falsch übertragen und es kommt 00000110000 an, würde man am ehesten ein „A“ annehmen.

So wie bei der binären Magie kann man sich aber auch hier entscheiden:

Entweder

- 2 Fehler erkennen und korrigieren und 3 Fehler noch erkennen

oder gleich

- 5 Fehler nur erkennen, dafür aber keinerlei Korrekturversuche unternehmen.

Sie sehen, dass unser Experiment vom Anfang des Kapitels durchaus einen guten Einblick in die echte Magie der Informatik ermöglichte: Indem wir die Anzahl der X in einer Zeile oder Spalte gerade hielten, haben wir dafür gesorgt, dass alle gültigen Zeilen und Spalten die Hamming-Distanz 2 besitzen. Für ganze Quadrate gilt (wegen der Kombination aus Zeilen und Spalten) sogar die Hamming-Distanz von 3, wodurch ein einzelner Fehler korrigierbar ist.

Die Wahl der richtigen Hamming-Distanz und damit Fehlertoleranz eines Codes kann individuell je nach erwarteter Fehlerwahrscheinlichkeit und der Fatalität eines Fehlers angepasst werden:

Bei der Übertragung von Sprache über das Haustelefon ins Nachbarzimmer ist kaum ein Fehler zu erwarten und wenn er doch auftritt, knackt die Leitung allenfalls – man wird daher auf eine Korrektur wahrscheinlich komplett verzichten.

Fehlerkorrektur von Audio-CDs

Das Prinzip, dass die Fehlerkorrektur so gut sein muss, wie die Anwendung es erfordert, kann man auch bei Audio-CDs erkennen: Dort ist (zumindest im Musikeil) keinerlei Korrektur vorgesehen, denn im Zweifel wurde von den Entwicklern eine kurze Unterbrechung oder ein winziges Knistern als akzeptabel angesehen. Daher kann es beim Einlesen oder „Grabben“ einer Audio-CD auch zu unterschiedlichen Ergebnissen kommen – auch wenn die Information eigentlich digital gespeichert ist.

Die Signalisierung zu einer mehrere Lichtminuten entfernten Raumsonde muss dagegen mit immensen Störquellen kämpfen. Außerdem kann ein falsches Signal unter Umständen dazu führen, dass die Antenne falsch ausgerichtet wird und die Sonde damit verloren geht. Hier wird man ein Maximum an Sicherungen einfügen.

Resümee

Im Kapitel 5 haben wir Informationen so codiert, dass sie möglichst kompakt übertragen oder gespeichert werden konnten. Wir haben zu diesem Zweck die Redundanz vermindert.

In diesem Kapitel haben wir erforscht, dass Redundanz andererseits sinnvoll und manchmal sogar notwendig für eine sichere Übermittlung und Speicherung von Daten ist. Dazu ist allerdings die „natürliche“ Redundanz meistens ungeeignet – da sie nicht konsequent umgesetzt ist, wie man an sehr ähnlichen Worten in unserem Sprachschatz erkennen kann.

Als Informatiker versucht man daher am besten, zunächst die Redundanz zu eliminieren (z. B. mit dem Huffman-Verfahren) und danach wieder gezielt hinzuzufügen, um die maximal mögliche Sicherheit für die zur Verfügung stehende Datenmenge zu erreichen.

Zauberanleitung: So führen Sie das Kunststück vor

Sie benötigen 36 der Karten mit X auf einer und O auf der anderen Seite aus den Bastelbögen am Ende des Buches.

Suchen Sie Freiwillige unter Ihren Zuschauern.

Geben Sie diesen 25 Karten und fordern Sie sie auf, ein möglichst kompliziertes Quadrat aus 5 mal 5 Karten zu legen, etwa auf einem Tisch. Achten Sie aber darauf, dass auf der Unterlage noch Platz für eine sechste Zeile und Spalte bleibt.

Sagen Sie dann etwas wie: „Es sind ja noch Karten übrig – ich werde die Sache noch etwas schwerer machen, indem ich zufällig noch eine sechste Zeile und Spalte dazulege.“

Legen Sie dann tatsächlich eine 6. Zeile und Spalte. Dabei gehen Sie aber nur scheinbar zufällig vor: Sie achten darauf, dass die Anzahl von „X“ in jeder Zeile und in jeder Spalte geradzahlig ist (0, 2, 4 oder 6). Wenn also in einer Zeile zum Beispiel 3 X und 2 O liegen, ergänzen Sie ein X. Wenn 2 X und 3 O liegen, ergänzen Sie ein O.

Wenden Sie sich nun ab und fordern Sie einen Freiwilligen auf, eine Spielkarte möglichst unauffällig umzudrehen.

Danach identifizieren Sie diese augenblicklich: Es gibt nun genau eine Zeile und eine Spalte mit einer ungeraden Anzahl von X. Die gesuchte Karte ist diejenige im Schnittpunkt. Im Beispiel befindet sie sich in der zweiten Zeile und dritten Spalte.

Abbildung 11.8 verdeutlicht das grafisch.

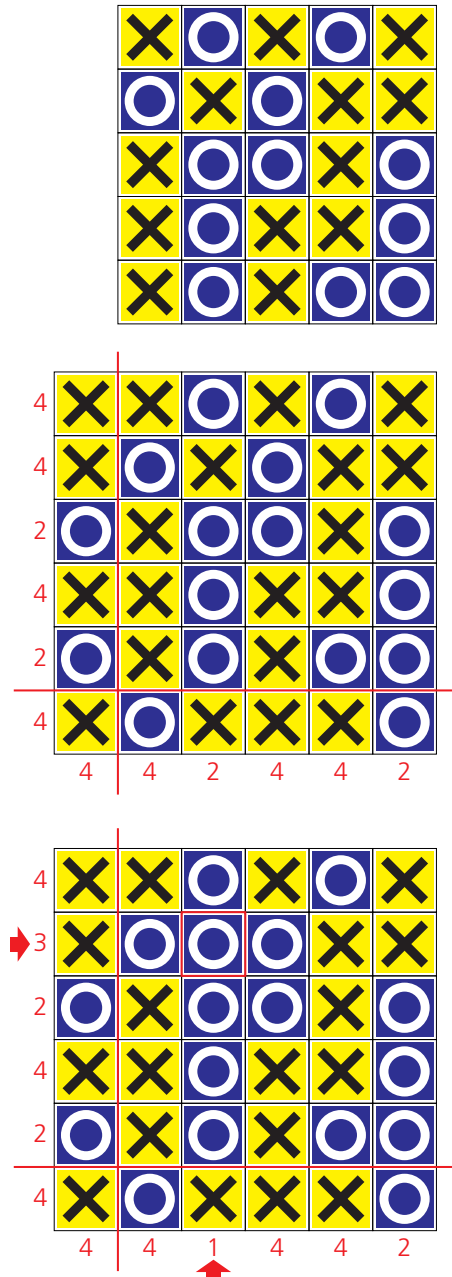


Abbildung 11.8 Lösungsschema für den Zaubertrick

Literatur



Tim Bell, Ian H. Witten, Mike Fellows „**Computer Science Unplugged**“; <http://www.csunplugged.org/>

In diesem online verfügbaren Buch finden Sie das Zauberkunststück aus diesem Kapitel sowie einige weitere interessante Experimente mit Informatik. Inzwischen existiert sogar eine deutsche Übersetzung.



12. Allmächtiger Computer!?

Einführung

Edsger W. Dijkstra ist uns bereits im ersten Kapitel begegnet. Neben brillanten algorithmischen Ideen hatte er auch eine sehr moderne Sicht auf die Wissenschaft Informatik. So prägte er den Satz: „Informatik hat ungefähr so viel mit Computern zu tun wie Astronomie mit Teleskopen.“ Diese Analogie spiegelt natürlich einerseits ein Leitthema dieses Buches wider: Informatik funktioniert auch ohne Computer. Andererseits zeigt sie auch die Grenzen des Werkzeugs Computer auf: Auch mit dem bestmöglichen Teleskop können Astronomen nicht alles herausbekommen, was sie über die Himmelskörper wissen möchten.

Für den Computer gilt allerdings die weitverbreitete Meinung, das Gerät könne prinzipiell alles, die Limitierung liege allenfalls beim Programmierer. Diesen Widerspruch wollen wir hier aufgreifen und die Frage klären, ob Computer allmächtig sind, wenn sie nur richtig programmiert werden.

Das Affenpuzzle

Auch ein so schwieriges Thema sollte Spaß machen und daher beginnen wir unsere Forschung mit einem kleinen Spielchen: Schneiden Sie die Spielsteine mit den Affen hinten im Buch aus und beginnen Sie zu puzzeln. Die Steine sind mit unterschiedlichen Rückseiten markiert. Das einfachste Puzzle besteht lediglich aus den vier Teilen mit gelben und dunkelroten Punkten. Sie dürfen die Puzzlesteine beliebig drehen. Ziel ist ein Quadrat, das zwei Puzzlesteine hoch und breit ist und bei dem die Affen im mittleren Bereich sowohl in der Farbe zusammenpassen als auch jeweils vollständig aus Ober- und Unterteil bestehen.

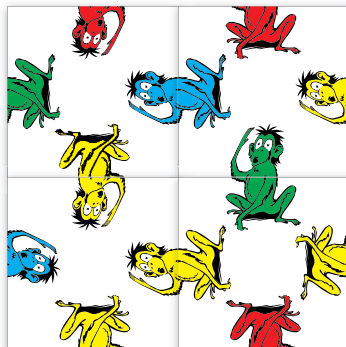


Abbildung 12.1 Gelöstes
2 × 2-Puzzle

Abbildung 12.1 zeigt ein korrekt gelöstes Puzzle (allerdings mit anderen Spielsteinen, um Ihnen den Spaß nicht zu nehmen). Achtung: Bitte schauen Sie vor und nach dem Puzzeln auf die Uhr und schreiben Sie sich die für die Lösung benötigte Zeit auf!



Sie haben ganz sicher nicht allzu lange gebraucht, daher wollen wir Sie ein klein wenig mehr herausfordern: Nehmen Sie jetzt **zusätzlich** die fünf Karten mit gelben und grünen Punkten.

Wenn Sie nun die neun Puzzleteile (also gut doppelt so viele wie im 2×2 -Fall) wiederum korrekt zusammenfügen möchten – wie lange wird das wohl dauern?

Doppelt so lange? Oder vier Mal? Sechzehn Mal?

Nachdem Sie eine Zeit geraten haben, fangen Sie an zu Puzzeln, und notieren Sie wieder die tatsächlich benötigte Zeit. Kleiner Tipp: Das soeben fertiggestellte 2×2 -Puzzle ist nicht Bestandteil der 3×3 -Version.



Sicherlich haben Sie deutlich länger als die 16-fache Zeit benötigt – falls nicht, lassen Sie auch Freunde oder Familienmitglieder auf Zeit puzzeln, um einen zufälligen „Treffer“ auszuschließen.

Geheimnisvolle Rückseiten

Ein weiteres kleines Rätsel ist in den Rückseiten der Affenpuzzleteile versteckt: Es ist kein Zufall, wo sich die gelben und wo die andersfarbigen Punkte befinden. Können Sie herausknobeln, nach welchem Schema sie angeordnet sind? Das Muster der zehn unterschiedlichen Farben für die Affenpuzzleteile mit Schildkröte ist ebenfalls nicht zufällig entstanden – auch hier sind Sie gefragt! Lösungsansätze können im Abenteuer-Informatik-Forum diskutiert werden.

Es ist bemerkenswert, wie viel schwerer ein solches Puzzle wird, wenn man die Zahl der Teile vergrößert. Wenn sie bereit sind, führen wir dieses Experiment noch weiter: Nehmen Sie nun zusätzlich die sieben Puzzleteile mit gelben und blauen Punkten und versuchen Sie, aus den insgesamt 16 Feldern ein korrektes 4×4 -Quadrat zu legen. Bitte notieren Sie sich wiederum die Zeit, auch wenn Sie ggf. das Puzzeln unterbrechen.

Auch wenn die Zahl der Teile nun noch nicht einmal verdoppelt wurde, benötigen Sie wahrscheinlich noch einmal deutlich länger. Nehmen Sie noch die Spielsteine mit den gelben und hellroten Punkten hinzu, können Sie zur Krönung auch noch ein 5×5 -Affenpuzzle lösen. Das schaffen nur die allerwenigsten Menschen ohne Computerunterstützung!

Die Zahlen auf den Rückseiten geben Ihnen keinen Hinweis, aber damit können Sie sich ggf. im Forum leichter über Lösungsansätze verständigen.

Sie überlegen vielleicht, ob man nicht durch geschickteres Vorgehen Zeit sparen könnte. Auch in den vorangegangenen Kapiteln waren einige Probleme mit reinem Ausprobieren kaum lösbar, aber unter Anwendung eines pfiffigen Algorithmus schon.

Auch in diesem Fall könnte es einen solchen guten Lösungsalgorithmus geben – nur hat ihn bis heute niemand gefunden! Das Affenpuzzle gehört zur Gruppe der nicht praktisch berechenbaren Probleme, die ich bereits im ersten Kapitel erwähnt hatte. Da sich schon die klügsten Köpfe der Menschheit vergeblich an einer guten Lösung versucht haben, also einer Lösung, die mehr macht als nur geschickt auszuprobieren, geht man davon aus, dass es tatsächlich keine solche Lösung gibt. Allerdings konnte diese Aussage bisher auch noch nie jemand beweisen.

In der Informatik gibt es immer wieder Probleme, die „nicht skalieren“. Das bedeutet, zur Lösung einer doppelt so großen Aufgabe benötigt man weit mehr als doppelt so viel Zeit. So etwas haben wir zum Beispiel im zweiten Kapitel für das Sortieren von Karten herausgefunden, genauso wie im ersten Kapitel für die Berechnung des kürzesten Wegs von Berlin nach München. In vielen Anwendungen muss die Aufgabengröße zum Quadrat genommen werden, um die zur Lösung benötigte Zeit abzuschätzen. Trotzdem sind diese Aufgaben immer noch praktisch lösbar.

Einige Aufgaben sind aber so schwer, dass sie mit Verdopplung der Aufgabengröße gleich unermesslich viel mehr Lösungszeit beanspruchen. Das gilt sowohl für das Lösen von Hand als auch für die Verwendung eines Computers: Das Hinzufügen eines einzigen Puzzlesteins zu einer lösbaren Aufgabe kann dann bedeuten, dass das Problem gar nicht mehr lösbar ist, weil man eine absurd lange Zeit mit der Lösung beschäftigt wäre! Hierbei spricht man dann von nicht praktisch lösbaren Problemen.

Auch andere Disziplinen kennen Probleme, die nicht skalieren: Planen Sie eine Brücke über einen Graben von einem Meter Breite, legen Sie einfach eine Platte aus Metall oder Stein darüber – fertig. Eine Brücke über eine 10 Meter breite Kluft kann aber nicht einfach mit 10 solcher Platten gebaut werden. Hier ist eine statisch genau berechnete Konstruktion erforderlich mit deutlich höherem als dem 10-fachen Aufwand.

Wenn Sie nun per Brücke mehr als 100 Meter überwinden möchten, ist der erforderliche Aufwand wiederum um ein Vielfaches höher. Die größte bisher erreichte Spannweite einer Brücke ist ca. 2.000 Meter. Jeder weitere Meter hat enormen Planungsaufwand und immense Material- und Baukosten zur Folge.

Eine Million Dollar Preisgeld ...

... bekommt derjenige, der für unser Affenpuzzle oder ein beliebiges anderes nicht praktisch berechenbares Problem (Fachbegriff: Problem aus der Klasse NPC oder „nicht deterministisch polynomiell berechenbar“) ein Verfahren herausfindet, mit dem es doch praktisch berechenbar wäre. Alternativ dürfen Sie auch mathematisch beweisen, dass es kein solches Verfahren geben kann. Damit hätten Sie nämlich eines der sieben sogenannten „Millennium Problems“ gelöst. Das Clay Mathematics Institute in Cambridge hat auf die Lösung ein Preisgeld von 1.000.000 US-Dollar ausgesetzt. Inzwischen gibt es über 70 registrierte Versuche, das Geld einzustreichen, es wurde aber bisher kein einziger der eingereichten und veröffentlichten Beweise als stichhaltig akzeptiert. Das Rennen ist also noch offen ...

Für Experten: Natürlich steigert sich der Aufwand beim Brückenbau immer noch nicht so stark wie bei den hier beschriebenen Problemen der Informatik, insbesondere nicht exponentiell. Allerdings verbilligt sich das Material dort auch nicht in dem Maße wie bei Computern ...



Abbildung 12.2 Verschiedene Brückenkonstruktionen

Was steckt dahinter: Wie schwer ist das Affenpuzzle?

Bevor wir wieder mit einfacheren Experimenten fortfahren, wollen wir ergründen, wie schwer das Affenpuzzle wirklich ist. Diese Analyse gehört in der Informatik zum normalen Geschäft.

Gehen wir erst einmal davon aus, dass wir einfach stur alle Möglichkeiten durchgehen, die Affenpuzzleteile zu legen, bis endlich alle zueinanderpassen. Die n Puzzleteile können in n! Permutationen gelegt werden. Jedes Teil passt dann nur in einer von vier möglichen Drehrichtungen. Insgesamt gibt es also $n! \cdot 4^n$ Möglichkeiten.

Wenn wir davon ausgehen, dass es sich bei den Puzzles um schwierige Exemplare handelt, also solche, für die es nur vier Lösungen gibt (das ist das Minimum, da ein korrektes Puzzle auch um je 90 ° gedreht noch korrekt ist), müssen wir im Durchschnitt ein Achtel der Anordnungen durchprobieren, bis wir eine korrekte gefunden haben. Für das Probieren einer Lösung müssen jeweils n Karten gelegt werden. Daher kommen wir auf einen Gesamtaufwand von

$$\frac{n! \cdot 4^n \cdot n}{8} \text{ Kartenbewegungen.}$$

Anhand einer Tabelle können wir uns das Ausmaß dieser Formel klarmachen:

Kantenlänge Puzzle	Puzzle- teile	Aufwand nach Formel	Dauer beim Kartenlegen per Hand in 0,5 s	Dauer beim Kartenlegen per Computer in 0,01 ns
2	4	3072	25 Minuten	< 1 s
3	9	47563407360	754 Jahre	< 1 s
4	16	44931349155019751424000	59.365 mal Universumsalter	14.247 Jahre
5	25	8732034971401365448912323118891008000000		
6	36	878344409141902190815781553750844988457254774772439449600000000		
7	49	96386108756373187 483760857339381516779271342702313473398026579323858050146925281280000000000		
8	64	21588692807774635761949078610839551766428391804935978 093730485270657051519075103281117627954516958437175922471731200000000000000		
9	81	16945018342271720384 528887431389738998662893500484164086563712258501015982964610508987534926237 09694631699582022417995579566498653513346428195827613696000000000000000000		
10	100	74984723061792395304341371831967689215252543707966986354013192594178 138233002394645831845136345079452453791988578895816786883974949629650018907 83478231601490761199317313797785217305699736040243200000000000000000000000		

Nach diesen Zahlen müsste bereits das 4×4 -Puzzle als nicht lösbar gelten, weil bereits ein Computer länger dafür benötigte als wir warten möchten ...

Dass dies allerdings nach unten korrigiert werden muss, wird bereits daran ersichtlich, dass Sie bei den kleineren Puzzles für die Lösung von Hand deutlich weniger Zeit benötigt haben als veranschlagt wird.

Überlegen wir daher also, wie man geschickter vorgehen könnte. Als Informatiker kann man recht einfach formal beweisen, dass das Affenpuzzle zu den nicht praktisch lösbaren Problemen gehört, daher wissen wir, dass wahrscheinlich jedes Verfahren etwas mit Ausprobieren zu tun hat. Allerdings kann man geschickt ausprobieren.

Eine Technik dafür nennt sich „Branch and Bound“. „Branch“ (englisch „Verzweigung“) bezieht sich auf das Ausprobieren: Wie in einem Baum, in dem man aus allen Blättern das mit einer bestimmten Herbstfärbung finden muss, verzweigt man zu allen potenziellen Lösungen, um die korrekte zu finden. „Bound“ ist die Schranke: Man versucht möglichst frühzeitig zu erkennen, dass man beim Ausprobieren einen definitiv falschen Zweig erreicht hat und umkehren muss.

Wie könnten wir das beim Affenpuzzle umsetzen? Schauen wir uns dafür doch einfach beim Puzzeln zu! Wir legen nicht alle Puzzleile vollständig hin, sondern erst eines, dann ein weiteres passendes usw. Falls es kein passendes Teil mehr gibt, wissen wir, dass wir auf dem Holzweg sind und versuchen, die bereits gelegten Teile zu verändern. Das passiert oft auch schon bei sehr wenigen gelegten Teilen.

Beim Konstruieren der Ausstellung „Abenteuer Informatik“ wurde ich von Gero Scholz daher auf folgende Lösung hingewiesen:

Zuerst wird das Teil oben links gelegt, hierfür gibt es alle Möglichkeiten (Anzahl der Teile mal vier Ausrichtungen). Danach legen wir das Puzzleile rechts davon. Dieses muss ja an das vorhandene Affensymbol angelegt werden. Nur eines von acht Symbolen passt und daher schränkt sich die Zahl der Züge auch auf etwa $1/8$ der noch vorhandenen Kombinationen ein.

Nun könnten wir wieder ein weiteres Teil rechts anlegen, stattdessen puzzeln wir vom Startteil nach unten weiter. Auch hier gilt, dass nur etwa $1/8$ der noch vorhandenen Teile passt.

Abbildung 12.3 zeigt die Situation. Legen wir als nächstes die rot markierte Position, werden die dort möglichen Teile durch das Symbol nach links und das Symbol nach oben jeweils auf $1/8$ begrenzt, insgesamt also auf $1/64$. Beim 4×4 -Puzzle besteht daher schon eine hohe Wahrscheinlichkeit, dass bereits jetzt kein Teil mehr passt und wir den Versuch bereits nach drei gelegten Teilen als Sackgasse identifizieren können.

In der Abbildung muss auf die Position eine Karte kommen, die einen grünen Affenkopf und im Uhrzeigersinn daneben einen roten Affenhintern aufweist. Davon gibt es im Puzzle lediglich ein Exemplar, das in einer bestimmten Ausrichtung gelegt werden muss. Statt also die restlichen 22 Karten in je 4 Positionen auszuprobieren, beschränken wir uns auf das Legen dieses einen Teils.

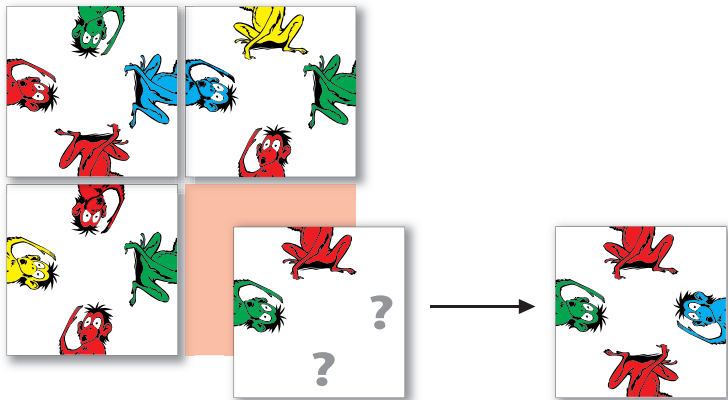


Abbildung 12.3 Es gibt nur eine passende Karte für die rot markierte Position

Im Kapitel 8 haben Sie sogar bereits eine Technik kennengelernt, mit der man prinzipiell die Puzzleteile im Computer so ablegt, dass man mit einem einzigen Zugriff identifizieren kann, ob ein passendes Teil existiert oder nicht: Hashing. Auch im weiteren Verlauf sorgt man nun dafür, recht früh immer Situationen zu erzeugen, in denen mit hoher Wahrscheinlichkeit eine Sackgasse erkannt wird. Mit „Branch and Bound“ sieht unsere Tabelle schon deutlich besser aus (die Zahlen entstehen aus einer komplizierteren statistischen Berechnung und keiner einfachen Formel):

Kantenlänge Puzzle	Puzzle-teile	Aufwand mit „Branch and Bound“	Dauer beim Kartenlegen per Hand in 0,5 s	Dauer beim Kartenlegen per Computer in 0,01 ns
2	4	129	1 Minute	< < 1 s
3	9	6773	56 Minuten	< < 1 s
4	16	1965501	11 Tage	< < 1 s
5	25	11385972942	180 Jahre	0,1 s
6	36	4482592451432549	0,005 mal Alter des Universums	12 Minuten
7	49	319714732333844702623086	422.420 mal Alter des Universums	101.380 Jahre
8	64	8846020083295533725639427309614807		
9	81	175951318252867500823848449208798428774205194798		
10	100	4265326330573335142293854240283428951958671066972613909968375819		

Der Aufwand für das manuelle Lösen des 2 × 2- und des 3 × 3-Puzzles entspricht ungefähr wirklich der Zeit, die man benötigt. Offenbar haben wir sogar noch eine bessere Strategie, denn ich behaupte, dass man das 4 × 4-Puzzle in unter elf Tagen schafft und sogar eine Chance hat, das 5 × 5-Puzzle in erlebbarer Zeit zu lösen.

Mit einem Supercomputer könnten wir sogar noch bis 6×6 gehen, bereits bei 7×7 streikt jedoch auch dieser: Unser Affenpuzzle bleibt ein nicht praktisch lösbares Problem.

Das Problem des Fliesenlegers

Noch sind wir mit dem Affenpuzzle nicht fertig. Einem Hersteller von Fliesen gefällt das Dekor so gut, dass er die „blaue Serie“ herstellt: Drei Fliesen mit den typischen Affen an den Rändern und blauem Hintergrund. Ein Fliesenleger könnte sich nun die Frage stellen, ob er mit diesen drei Sorten prinzipiell alle möglichen Flächen ausfüllen könnte, wenn er nur genügend davon bestellte.

Probieren Sie es aus: Versuchen Sie, eine möglichst große Fläche mit den ausgeschnittenen Puzzleteilen zu legen. Können Sie außerdem irgendwie herausfinden, ob man mit mehr Teilen immer größere Flächen ausfüllen könnte?



Zur Antwort reicht schon eine 3×3 Felder große Fläche. Abbildung 12.4 zeigt sie.

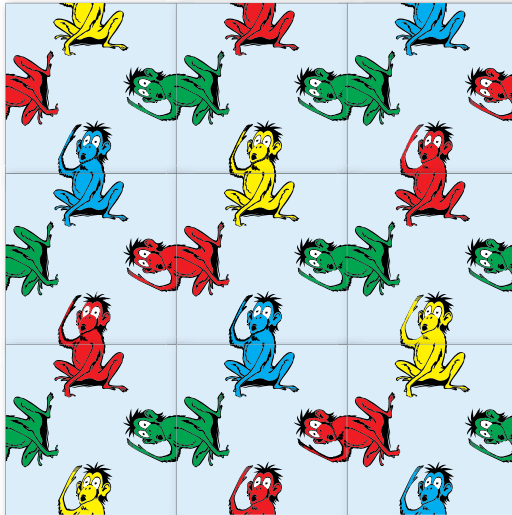


Abbildung 12.4 3×3 -“Multikachel“

Was ist so bemerkenswert an dieser Anordnung? Die obere Kante passt genau an die untere und die linke an die rechte Kante. Auf diese Weise können Sie beliebig viele dieser 3×3 -Blöcke aneinanderlegen und auf diese Weise in alle Richtungen beliebig große Flächen ausfüllen.

Vom großen Erfolg der blauen Serie angestachelt, legt der Fliesenhersteller die „rosa Serie“ auf. Versuchen Sie auch hier herauszufinden, wie flexibel man damit beim Auslegen ist.



Sie sehen, dass es nur jeweils ein gelbes und ein blaues Affenvorder- und Hinterteil gibt. Daher ist es ziemlich klar, dass die Affen in einer Reihe, wie in Abbildung 12.5, gelegt werden müssen – es gibt keine andere Möglichkeit.



Abbildung 12.5 Die rosa Affen können nur in dieser Weise gelegt werden

Da nur ein grünes Hinterteil zur Verfügung steht, legen wir es rechts an die beiden grünen Köpfe. Abbildung 12.6 zeigt dies.

Obwohl wir nur Puzzleteile gelegt haben, die wir aufgrund der zur Verfügung stehenden Teile legen mussten, passen nun zwei Teile nicht zueinander. Für die rosa Serie können wir daher mit Bestimmtheit sagen, dass man damit auf keinen Fall beliebig große Flächen ausfüllen kann.

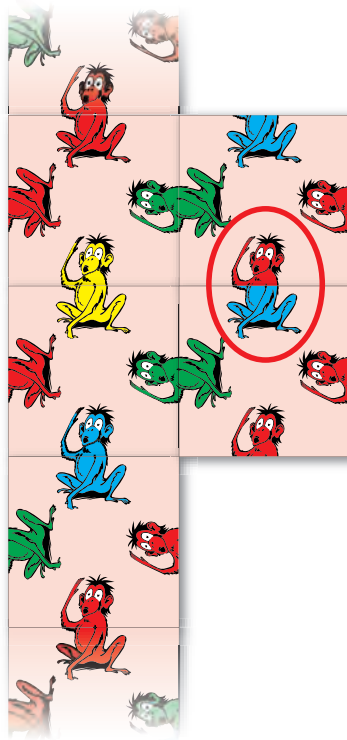


Abbildung 12.6 Widerspruch mit den rosa Affen

Nun könnte man die gewonnenen Erfahrungen ja als Grundlage für ein Verfahren nehmen, das für beliebige Sätze von Puzzleteilen bestimmt, ob man damit beliebig große Flächen auslegen kann:

Versuche, die Kacheln regelkonform aneinander zu legen.

Falls dabei ein Rechteck entsteht, bei dem die obere und untere sowie die linke und rechte Kante aneinanderpassen

→ Antwort JA.

Falls trotz Ausreizung aller Möglichkeiten eine Karte nicht passt

→ Antwort NEIN.

Es scheint so, als ob man mit genügend Geduld immer auf eine der beiden Alternativen stößt: Ein Programm, das auf diese Weise vorgeht, würde ggf. sehr lange brauchen, aber irgendwann entweder das Ergebnis „JA“ oder „NEIN“ hervorbringen ... oder doch nicht?



Abbildung 12.7 Ausschnitt aus einem Affenpuzzle mit Schildkröten

Um das zu ergründen, habe ich eine weitere Sorte von Affenpuzzleteilen für Sie vorbereitet: mit Schildkröten in den Ecken. Die zusätzliche Regel beim Legen ist nun, dass jede Schildkröte genau einen Kopf haben muss. Einen korrekten Ansatz sehen Sie in Abbildung 12.7.

Versuchen Sie, nach dem soeben bestimmten Verfahren vorzugehen, um zu entscheiden, ob die Teile tauglich für beliebig große Flächen sind.



Nach einigem Puzzeln haben Sie es sicherlich geschafft, alle Teile irgendwie zu verwenden und trotzdem noch nicht auf einen Widerspruch zu stoßen. Ein zueinanderpassendes Rechteck haben Sie aber sicherlich trotzdem nicht gefunden.

Wenn Sie Lust haben: Kopieren Sie die Puzzleteile und erzeugen Sie auf diese Weise ein riesiges Affenpuzzle. Ich garantiere Ihnen: Selbst wenn Sie von nun an jede freie Minute in die Erstellung des Affenpuzzle stecken – mit dem gerade herausgeknobelten Verfahren kommen Sie weder auf die Antwort „JA“ noch auf „NEIN“.

Wenn Sie mir das nicht glauben, sehen Sie sich Abbildung 12.8 an. Sie zeigt einen Ausschnitt aus einem solchen Puzzle. Versuchen Sie, in diesem Puzzle ein entsprechendes Rechteck zu finden.



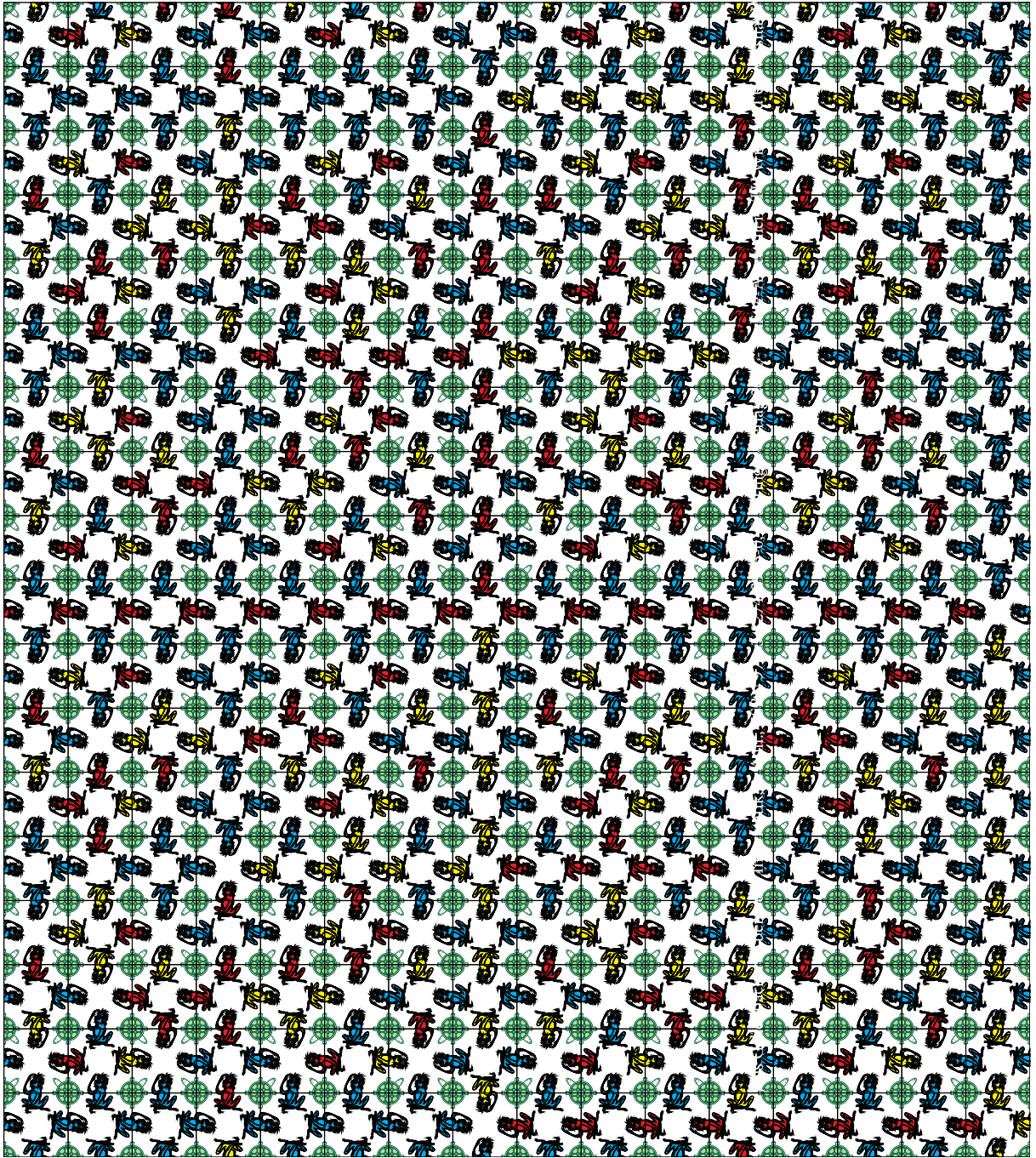


Abbildung 12.8 Riesen-Affenpuzzle

Die Terminierung

Computerprogramme sollen natürlich im Normalfall irgendwann einmal mit ihrer Berechnung fertig werden und mit einem Ergebnis aufwarten. In der Fachsprache heißt das Terminierung. Ihnen bekannte Formen von nicht terminierenden Programmen sind die berühmten „Abstürze“, bei denen ein Computer „gar nichts mehr macht“. Effektiv macht der Computer doch noch etwas, aber er führt eventuell immer wieder den gleichen Code in einer Endlosschleife aus und reagiert nicht mehr auf Ihre Eingaben – das Programm bzw. Betriebssystem terminiert nie.

Ein wichtiger Aspekt der theoretischen Informatik ist die Betrachtung, ob ein Programm immer (also für jede mögliche Eingabe) terminiert.

Vielleicht glauben Sie es mir auch einfach: Mit den acht neuen Puzzleteilen können Sie prinzipiell unendlich weiterpuzzeln, werden aber definitiv nie Rechtecke finden, die sich periodisch wiederholen. Ein Programm, das nach dem oben aufgestellten Verfahren vorgeht, würde nie fertig werden und nie auch nur das Ergebnis „JA“ oder „NEIN“ hervorbringen.

Effektiv hat sich mit dem speziellen Affenpuzzle-Satz (bzw. einem geometrischen Pendant) der Mathematiker Raphael M. Robinson auseinandergesetzt und bewiesen, dass es sich um eine unendliche, nichtperiodische Kachelung handelt. Schlauberger können nun einwenden, dass man doch einem Computer auch den Beweis des Herrn Robinson beibringen könnte, so dass der das Muster erkennt und auch bei den Affenpuzzlesteinen mit Schildkröte mit „JA“ antwortet.

Tatsächlich ist das auch richtig! Allerdings gibt es quasi unendlich viele Kombinationsmöglichkeiten für Affenpuzzle-Fliesensets, die wieder andere Wiederholungsmuster aufweisen als das nach Robinson, und es ist unmöglich, diese alle vorausszusehen und dem Computer beizubringen. Nur ein Mensch kann sich mit seiner Kreativität immer wieder auf neue Sets einstellen und für diese eine Antwort finden.

Die hier geschilderte Problemstellung kann also von keinem Computer der Welt gelöst werden und es wird auch nie einen Computer geben, der sie lösen kann! Ist es nicht beruhigend zu wissen, dass manche Dinge noch dem Menschen vorbehalten sind?

Was steckt dahinter: Nichtberechenbare Probleme

Computer können nicht alles: Das haben wir mit dem Affenpuzzle anschaulich gemacht. Die Frage des Fliesenlegers nach der Kachelung beliebig großer Flächen ist in der theoretischen Informatik als Domino-Problem bekannt. Es kann von Computern nicht gelöst werden. Besser gesagt: Es gibt immer Eingaben (in diesem Fall Sätze von Fliesen), für die der Computer keine Entscheidung „JA“ oder „NEIN“ treffen kann. Man sagt dann, das Problem ist nicht entscheidbar.

Beweisen wollen wir diese Unentscheidbarkeit allerdings für eine andere Fragestellung: das sogenannte Halteproblem. Wie oben bereits in der Marginalie erwähnt, ist es bei Software sehr wichtig, dass diese irgendwann anhält bzw. terminiert. Es wäre nun schön, wenn man die Entscheidung „hält irgendwann“ oder „hält nicht immer“ automatisch treffen lassen könnte.

Wir nehmen daher an, unter Aufbietung aller unserer Talente schreiben wir ein Programm HÄLT?, dem man ein anderes Programm zum Testen übergibt. HÄLT? ermittelt daraufhin, ob dieses übergebene Programm immer anhält und antwortet „JA“, wenn das der Fall ist, und sonst „NEIN“.

Zur Veranschaulichung verwenden wir zwei einfache Programme:

PROGRAMM A

```
Zeile 10: FALLS X=U GEHE ZU Zeile 10  
Zeile 20: ENDE
```

PROGRAMM B

```
Zeile 10: FALLS X=X GEHE ZU Zeile 10  
Zeile 20: ENDE
```

Sie sehen, dass Programm A testet, ob X und U identisch sind. Nur falls das zutrifft, springt die bedingte Ausführung wieder zur Zeile 10. Da sich der Computer kein X für ein U vormachen lässt, geht er im Programm einfach weiter und führt Zeile 20 aus, die das Programm beendet. Ein Aufruf von HÄLTS? mit diesem Programm führt also zu:

HÄLTS? (PROGRAMM A) =JA

Anders sieht die Sache beim zweiten Programm aus. Hier wird in Zeile 10 wieder zu Zeile 10 verzweigt, falls X identisch mit X ist (was natürlich immer stimmt). Damit verliert sich das Programm in einer Endlosschleife.

HÄLTS? (PROGRAMM B) =NEIN

Der Trick ist nun, unser eigenes Superprogramm in diese bedingte Ausführung einzubauen:

PROGRAMM C

```
Zeile 10: FALLS HÄLTS? (PROGRAMM C) =JA GEHE ZU Zeile 10  
Zeile 20: ENDE
```

Wir haben ja bereits festgestellt, dass in Zeile 10 die entscheidende Bedingung steckt: Ist sie wahr, verliert sich das Programm in einer Endlosschleife, ist sie falsch, terminiert es.

Andere Möglichkeiten gibt es auf die einfache Frage „Hält das Programm an?“ nicht. Da die Bedingung bei Programm C etwas knifflig ist, probieren wir die beiden möglichen Thesen durch:

These 1: PROGRAMM C hält an.

- Folge: Die Bedingung „HÄLTS? (PROGRAMM C) = JA“ ist immer wahr. Damit geht das Programm C bei Zeile 10 in eine Endlosschleife.
- Konsequenz: Das Programm C hält nicht an, was der These 1 widerspricht, die daher falsch sein muss.

These 2: PROGRAMM C hält nicht an.

- Folge: Die Bedingung „HÄLTS? (PROGRAMM C) = JA“ ist immer falsch. Damit überspringt das Programm C die Zeile 10 und hält bei Zeile 20 an.
- Konsequenz: Das Programm C hält an, was der These 2 widerspricht, die daher falsch sein muss.

Beide Thesen sind falsch, weitere kann es nicht geben. Es muss also irgendeine andere Voraussetzung falsch sein. Die einzige Möglichkeit dafür ist die Voraussetzung, dass wir ein Programm HÄLTS? entwickeln konnten.

Die einzig mögliche Folgerung aus unseren Schlüssen ist daher, dass es so ein Programm HÄLTS? nicht gibt und auch nie geben kann.

Herzlichen Glückwunsch! Sie haben nun eine ganz wichtige Erkenntnis der theoretischen Informatik nachvollzogen: Ein Computer kann für beliebige Software noch nicht einmal entscheiden, ob diese immer anhält oder nicht.

Noch viel schwerer ist zum Beispiel der Nachweis, ob ein bestimmter Teil eines Computerprogramms das macht, was es soll. So gibt es bei kritischen Softwaresystemen, etwa in Flugzeugen oder Atomkraftwerken, die Pflicht, sie zu validieren, was so viel bedeutet wie ihre Fehlerfreiheit zu beweisen.

Da ein Computer noch nicht einmal potenzielle Endlosschleifen herausfinden kann, ist er mit der Validierung noch viel mehr überfordert. Hier müssen immer Menschen – meistens Informatiker oder Mathematiker – heran.

Um Missverständnisse auszuschließen: Auch ein Mensch kann für diese Aufgabenstellungen kein allgemeines Lösungsverfahren angeben – das geht nicht! Und wenn es ginge, könnte man die Durchführung auch einem Computer übertragen. Der Mensch ist aber in der Lage, für die meisten speziellen Eingaben (also z. B. Programme beim Halteproblem) durch Kreativität herauszufinden und zu beweisen, ob diese anhalten oder nicht.

Es soll nicht verschwiegen werden, dass einige – fast immer konstruierte – Aufgabenstellungen bisher auch von Menschen nicht gelöst werden konnten. Für einen Teil davon wurde sogar bewiesen, dass sie überhaupt nicht lösbar sind.

Resümee

Wir wissen nun, dass der Computer nicht allmächtig ist. Einige Problemlösungen bleiben dem Menschen vorbehalten, weil Computer einfach zu lange benötigen würden, und andere Probleme bedürfen der menschlichen Lösung, weil Computer sie aus Prinzip nicht lösen können.

Was bringt uns diese Erkenntnis außer der tröstlichen Botschaft, dass der Informatiker sich nicht selbst überflüssig macht? Natürlich: Wenn wir wissen, dass ein Problem mit dem Computer nicht lösbar ist, versuchen wir das auch erst gar nicht. Wir könnten dafür versuchen, das Problem etwas umzuformulieren, so

dass es einerseits mit dem Computer lösbar wird und andererseits uns die Lösung trotzdem weiterhilft.

So könnte man beim Affenpuzzle über Jokerkarten nachdenken, die universell einsetzbar sind. Bereits zwei Jokerkarten machen auch ein 6×6 -Puzzle in Sekunden lösbar.

An der Front der automatischen Programmbeweise gibt es teilautomatische Beweiser: Immerhin ist der Computer ein sehr schnelles und konsequentes Werkzeug. Ein teilautomatischer Beweiser kann daher über lange Strecken Software auf Fehler hin analysieren, muss allerdings ab und zu ein Teilproblem an den Bediener weitergeben. Auf diese Weise schafft man den gesamten Beweis sozusagen im Zusammenspiel von Intelligenz auf der einen Seite und schierer Rechenleistung auf der anderen Seite.

Dijkstra hatte also recht ...

Literatur

Juraj Hromkovic „**Sieben Wunder der Informatik**“; Vieweg+Teubner; 2. Auflage 2009; ISBN 978-3-8351-0172-2



Nicht nur die Berechenbarkeit und Komplexitätstheorie wird in diesem Buch thematisiert, sondern der Leser begibt sich auf eine unterhaltsame Rundreise durch die Informatik. Da das Buch auf Vorlesungen basiert, werden viele bekannte und neue Themen aus einem anderen, etwas formaleren Blickwinkel geschildert. Auf jeden Fall eine ideale Ergänzung zu „Abenteuer Informatik“!

David Harel „**Das Affenpuzzle und weitere bad news aus der Computerwelt**“; Springer; ISBN 978-3-540-42307-2



David Harel hat ein ganzes Buch dem Thema dieses Kapitels gewidmet und erklärt an vielen weiteren Beispielen, warum der Computer manche Dinge nicht lösen kann. Auch wenn Harel bereits beim 5×5 -Affenpuzzle die Flagge streckt (was mit dem hier beschriebenen Verfahren in ein paar Sekunden gelöst ist), ein lesenswertes Buch.

Raphael M. Robinson „**Undecidability and Nonperiodicity for Tilings of the Plane**“ in *Inventiones math.* 12, 177-209 (1971), Springer



Falls Sie dem Geheimnis des Affenpuzzles mit Schildkröten auf die Spur kommen möchten, besorgen Sie sich diesen 32-seitigen Beweis in einer Bibliothek oder im Internet. Er ist schon anspruchsvoll, aber man kann sich durchbeißen. Auf jeden Fall lernen Sie dort, wie man die Puzzleteile konsequent legen muss, um beliebige große Flächen damit zu füllen.

Glossar

Algorithmus (Seite 14)

Ein Algorithmus ist eine Handlungsvorschrift zur Lösung eines Problems bzw. einer Kategorie von Problemen. Diese Handlungsvorschriften lassen sich im Allgemeinen in ein Computerprogramm umsetzen. Hierfür müssen sie hinreichend genau formuliert sein.

Aufwandsabschätzung (Seite 61)

Mit der Aufwandsabschätzung wird in der Informatik oft die Qualität von Algorithmen bestimmt. Diese Abschätzung gibt an, wie stark die notwendige Rechenzeit in Bezug zur Problemgröße anwächst. Konstante Faktoren werden hier nicht berücksichtigt. Auf diese Weise kann für kleine Problemgrößen die tatsächliche Rechenzeit eines „schlechteren“ Algorithmus unter der des besseren liegen.

Binärcode (Seite 122)

Ein Code, der nur aus zwei Zeichen besteht. Diese werden üblicherweise durch „0“ und „1“ repräsentiert. Binärcodes sind im Computerbereich üblich, weil sie sich direkt durch eine Elektronik verarbeiten lassen, die zwei Zustände („Eingeschaltet“ und „Ausgeschaltet“) versteht.

Codierung (Seite 121)

Verfahren, die Symbole einer Nachricht in eine andere Form zu wandeln, ohne den Informationsgehalt der Nachricht einzuschränken. Die Codierung impliziert, dass eine entsprechende Decodierung möglich ist, um die Nachricht wieder in den Originalzustand zu versetzen.

Codierung bedeutet in der Informationstechnik meistens die Umwandlung von der für den Menschen verständlichen Form in eine für Maschinen verarbeitbare und über Netzwerke kommunizierbare Version.

Dynamische Programmierung (Seite 85)

Strategie zur Lösung komplexer Probleme, besonders zur Lösung von Optimierungsaufgaben. Man löst zunächst viele Teilprobleme und verwendet diese Bausteine, um die nächstgrößeren Probleme zu lösen usw. bis das Gesamtproblem gelöst ist.

Elementaroperationen (Seite 56)

Operationen bzw. Arbeitsschritte, die wir bei der Bestimmung der Laufzeit eines Algorithmus für wichtig bzw. zeitkritisch erachten. Welche Operationen hier ausgewählt werden, ist nicht global definiert, sondern ist vom jeweiligen Anwendungsfall abhängig. Oft handelt es sich um Speichervorgänge (Lesen und Schreiben im Speicher).

Invariante (Seite 188)

Eine Invariante ist eine Größe oder Gesetzmäßigkeit, die während der Lösung eines Problems bzw. während der Durchführung eines Algorithmus immer gleich bleibt.

Komprimierung (Seite 157)

Bei der Komprimierung wird eine Datenmenge reduziert, indem zwischen relevanten und irrelevanten Informationen unterschieden wird und lediglich die relevanten Informationen beibehalten werden. Diese Unterscheidung wird oft im Medienbereich anhand physiologischer Studien vorgenommen, nach denen bestimmte Informationsteile nicht oder nur von wenigen Menschen wahrgenommen werden können.

Laufängencodierung (Seite 136)

Bei der Laufängencodierung werden nicht einzelne Bildpunkte gespeichert bzw. übermittelt, sondern die Anzahl gleicher Bildpunkte hintereinander. Englisch ist der entsprechende Fachausdruck run-length. In manchen Bildverarbeitungsprogrammen bekommen Sie beim Speichern die Möglichkeit, „RLC“ auszuwählen, was ausgeschrieben run-length-coding bzw. Laufängencodierung bedeutet.

Methode der Abstraktion (Seite 3)

In zur Verfügung stehender Information stecken sowohl relevante als auch unwesentliche Anteile. Durch Abstraktion reduzieren Sie die Information auf das für die aktuelle Problemlösung Wesentliche: Dadurch können Sie sich besser auf Ihre Aufgabe konzentrieren.

Methode der Gleichformung (Seite 6)

Versuchen Sie, die verschiedenen Facetten eines Problems auf die gleichen Grundelemente zurückzuführen. Dadurch wird einerseits das Problem übersichtlicher und andererseits benötigt man weniger Lösungsansätze: Für gleichförmige Teilprobleme kann der gleiche Lösungsansatz verwendet werden.

Paradigma (Seite 182)

Das Paradigma ist ein (Denk-)Muster oder Vorbild, anhand dessen ein technisches System oder ein abstraktes Konzept konstruiert wird. Dadurch wird es für die Anwender leichter, sich im System zurechtzufinden.

Präfix (Seite 129)

Ein Präfix ist ein Wort oder eine Zeichenfolge, die mit *dem Anfang* einer anderen Zeichenfolge identisch ist. In der Codierung versucht man Präfixe zu vermeiden, da man mit Präfix codierte Nachrichten nicht mehr direkt decodieren kann.

Prinzip „divide et impera“ (Seite 45)

Sehr häufig hat man im Leben Probleme zu lösen, die zu unüberschaubar und groß sind, um sie in einem Ansatz zu lösen. Vielmehr teilen wir das Gesamtproblem auf in mehrere, handhabbare Stücke, die wir lösen. Die Teillösungen werden danach nur noch zusammengefasst. Dieses Prinzip wird auch in der Informatik sehr stark verwendet: Ein Programm zerlegt die gestellte Aufgabe zunächst in mehrere kleinere Einheiten, genannt Teilprobleme (divide = teile) und weist danach andere Programme an, diese zu lösen (impera = herrsche, befähige). Dabei ist sehr wichtig, dass die Teilprobleme unabhängig voneinander gelöst werden können, denn sonst müssten die Programme miteinander kommunizieren, unter Umständen auf Lösungen voneinander warten, was den Aufwand wiederum sehr erhöht.

Problemgröße (Seite 43)

Ein wichtiger Vorgang bei der Lösung von Aufgaben aus der Informationstechnik ist das Bestimmen der so genannten Problemgröße. Sie stellt ein Maß dar, wie schwierig bzw. umfangreich die Aufgabe bzw. das Problem ist. Daher hängt von der Problemgröße entscheidend der zu leistende Aufwand ab.

Verlustfreie Komprimierung (Seite 132)

Dies ist ein populäres Synonym für eine Codierung, bei der die Datenmenge im durchschnittlichen Fall reduziert wird. Da es sich um eine Codierung handelt, ist gewährleistet, dass die Originaldaten aus dem Code wieder hergestellt werden können.

Vollständige Induktion (Seite 101)

Die vollständige Induktion ist ein zweistufiges mathematisches Beweisverfahren. Es funktioniert insbesondere für Aussagen in Bezug auf ganze Zahlen. Man beweist, dass die Aussage für eine bestimmte, kleine Zahl a gilt. Ferner beweise man: Falls die Aussage für die Zahl $n-1$ gilt, dann gilt sie auch für die Zahl n .

Damit hat man dann bewiesen, dass die Aussage für alle ganzen Zahlen größer oder gleich a gilt.

Bildnachweis

Seite 3, Marginalienspalte, Das blaue Pferd von Franz Marc, 1911, aus *The Yorck Project, 10.000 Meisterwerke der Malerei*, DVD-ROM, 2002

Seite 7, Marginalienspalte, Leonhard Euler, Pastell-Portrait von Emanuel Handmann, 1753, Öffentliche Kunstsammlung Basel

Seite 15, Marginalienspalte, E. W. Dijkstra, Foto: Hamilton Richards,
<hr60@post.harvard.edu>

Seite 85, Marginalienspalte, Richard Ernest Bellman, Foto: University of Southern California

Seite 107, Marginalienspalte, Tally Stick aus dem späten 13. Jahrhundert, Herkunft unbekannt, Museum of Archaeology, Southampton

Seite 108, Abbildung 4.1, Abakus, aus Georges Ifrah, *Universalgeschichte der Zahlen*, Sonderausgabe, Campus-Verlag, 1991, Originaltitel: Georges Ifrah, *Les chiffres ou l'histoire d'une grande invention*, Éditions Robert Laffont / Éditions Seghers

Seite 108, Marginalienspalte, Portraitbildnis von Wilhelm Schickard, Maler unbekannt, Bildnissammlung im Senatssaal in der Universität Tübingen, Bühler, Pressestelle der Universität Tübingen

Seite 108, Abbildung 4.2, Rechenmaschine von Wilhelm Schickard, Tübingen, 1623, Rekonstruktion von Prof. Sr. Baron B. von Freytag-Löringhoff, Tübingen, um 1960, nach Zeichnungen aus den Nachlässen von Kepler und Schickard, Foto: Deutsches Museum

Seite 108, Marginalienspalte, Bildnis des Philosophen Gottfried Wilhelm von Leibniz, Gemälde, Öl auf Leinwand von Bernhard Francke, o. J., 81 x 66 cm, Original: Braunschweig, Herzog Anton Ulrich-Museum, bpk Berlin

Seite 109, Marginalienspalte, Blaise Pascal, Kupferstich von Gérard Edelinck, um 1660, bpk Berlin

Seite 109, Marginalienspalte, Charles Babbage, Holzstich, um 1871, bpk Berlin

Seite 110, Marginalienspalte, Lochkarten, aus Douglas W. Jones, *Punched cards*, University of Iowa

Seite 110, Abbildung 4.3, Babbages Analytical Engine, 1842, aus Alexandre Faribault, *Évolution des machines à calculer*, Université de Sherbrooke, Québec

Seite 111, Abbildung 4.4, Ada Countess of Lovelace, nach einem Stich von W. H. Mote, 1838

Seite 112, Abbildung 4.5, Konrad Zuse, 1988, Foto: Horst Zuse

Seite 113, Abbildung 4.6, ENIAC, U. S. Army Photo, aus K. Kempf, *Historical Monograph: Electronic Computers Within the Ordnance Corps*, The ENIAC, in BRL building 328

Seite 113, Abbildung 4.7, UNIVAC, Werbeanzeige, ca. 1956

Seite 117, Abbildung 4.9, Erste Computermaus, Courtesy of SRI International, Menlo Park, CA.

Seiten 149ff, Abbildungen 5.16–5.19 und 5.21–5.22, aus Corel Professional Photos Collection, Corel Draw

Seite 154, Abbildung 5.23, aus Gerolf Steiner, *Tierzeichnungen in Kürzeln*, Spektrum Akademischer Verlag, 2006

Seite 155, Abbildung 5.24, ZDF Bilderdienst, Abbildungen 5.25 und 5.26 darauf aufbauend, bearbeitet von Dr. Jens Gallenbacher

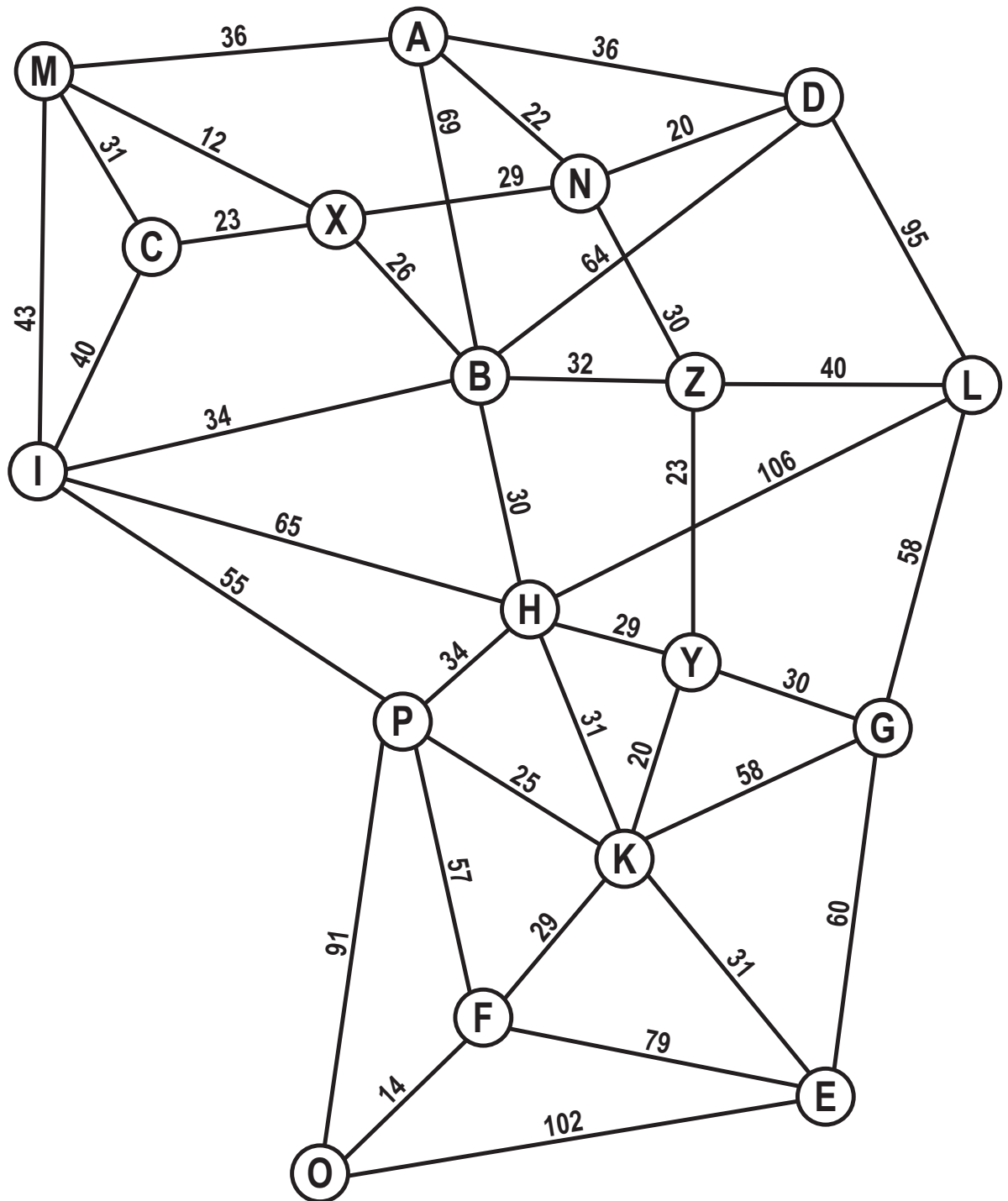
Seite 330, Marginalienspalte oben, Kreditkarte, iStockphoto

Seite 330, Marginalienspalte unten, Ausweis, Fotomontage, Leonhard Euler: Ausschnitt aus Pastell-Portrait von Emanuel Handmann, 1753, Öffentliche Kunstsammlung Basel

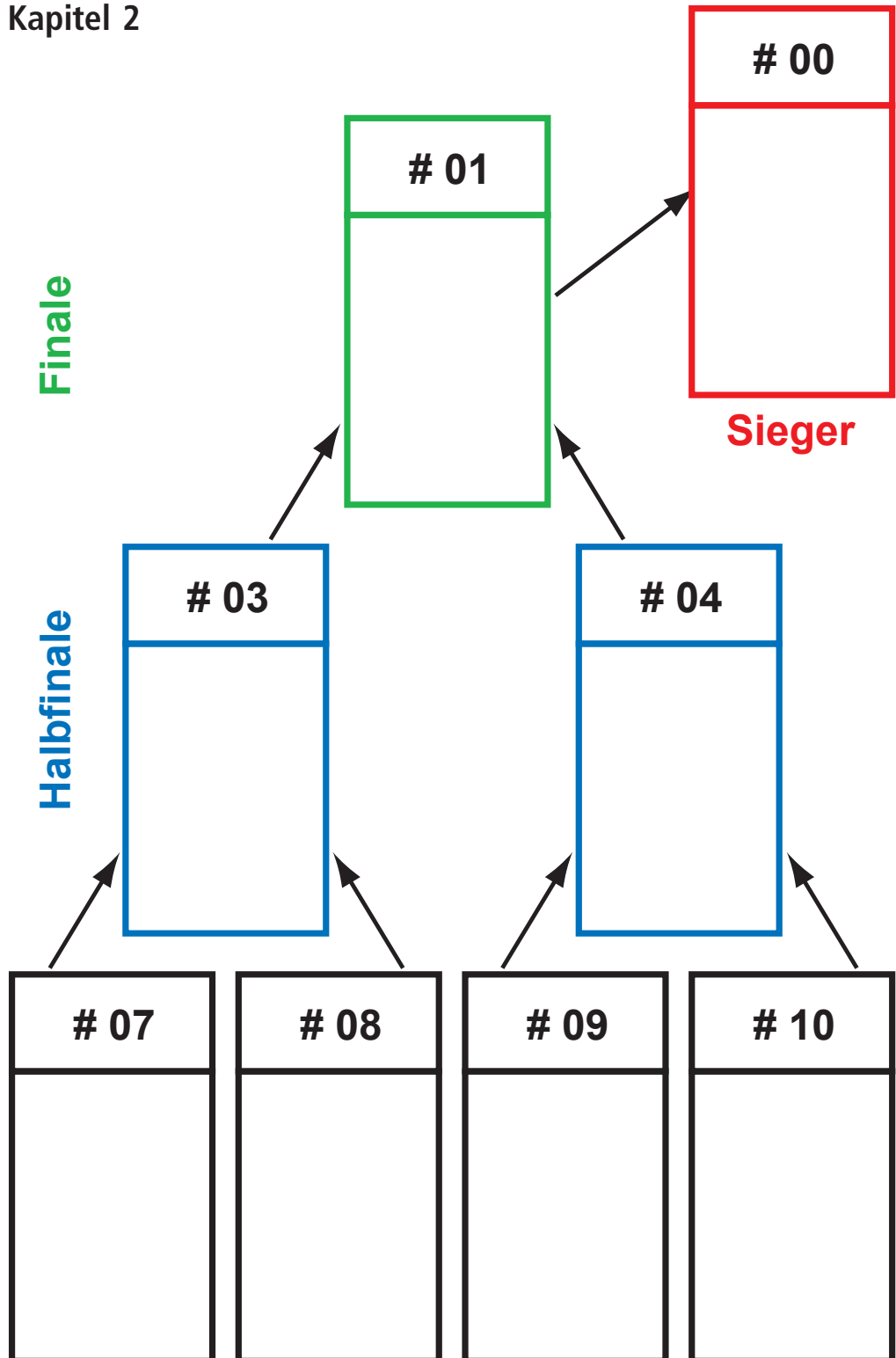
Alle weiteren Abbildungen einschließlich der Kapiteleingangsbilder wurden von Dr. Jens Gallenbacher erstellt. Teilweise wurden hierfür IMSI Masterclips (www.imsi.de) verwendet.

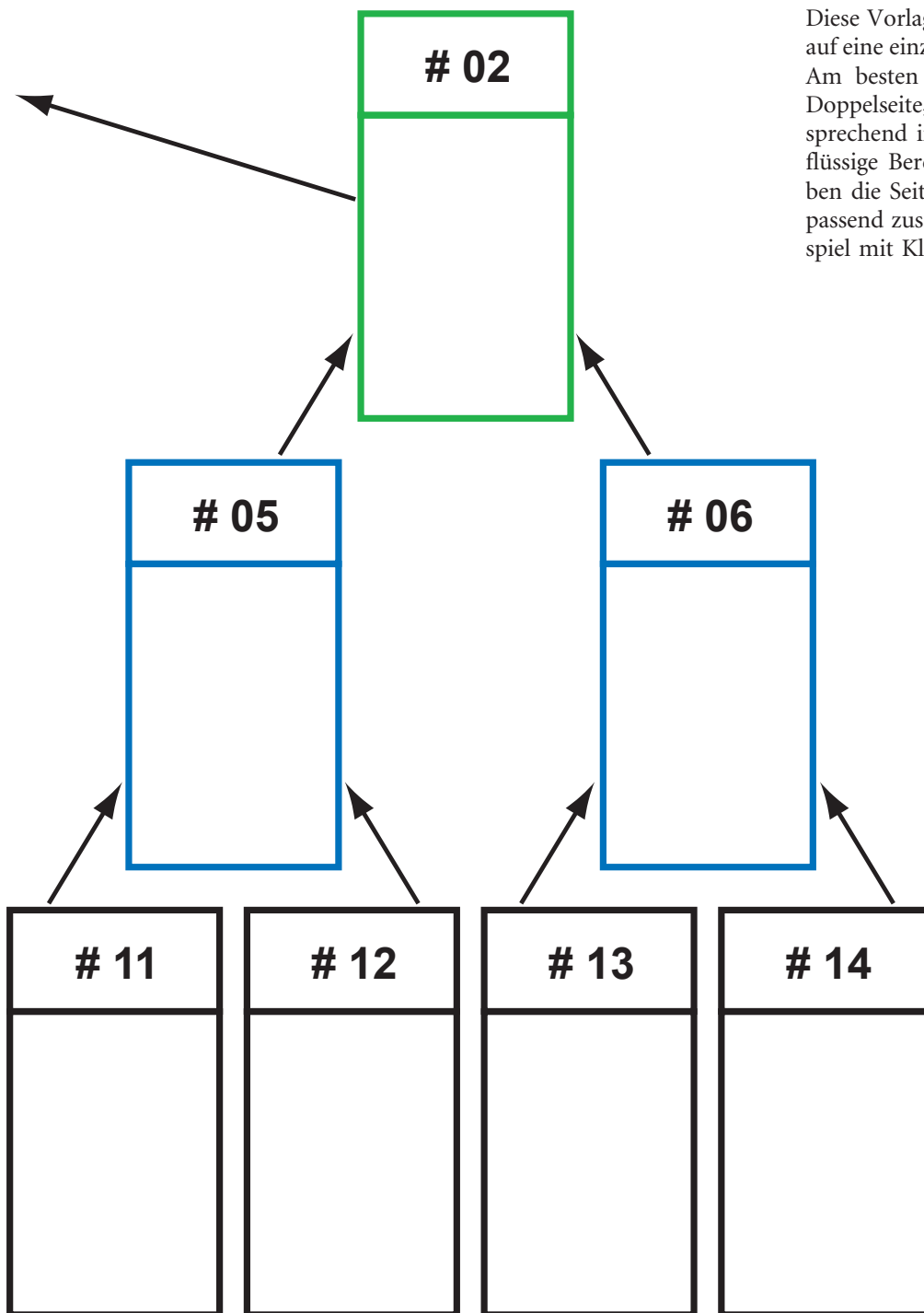
Anhang A: Kopiervorlagen

Für Lehrer, die einen ganzen Satz Kopiervorlagen und Bastelbögen benötigen, gibt es die Vorlagen auf DVD zum Selbstaussdrucken (zusammen mit der Lizenz, das für Lehrzwecke zu tun) unter der ISBN ■■■■■ zu erwerben.



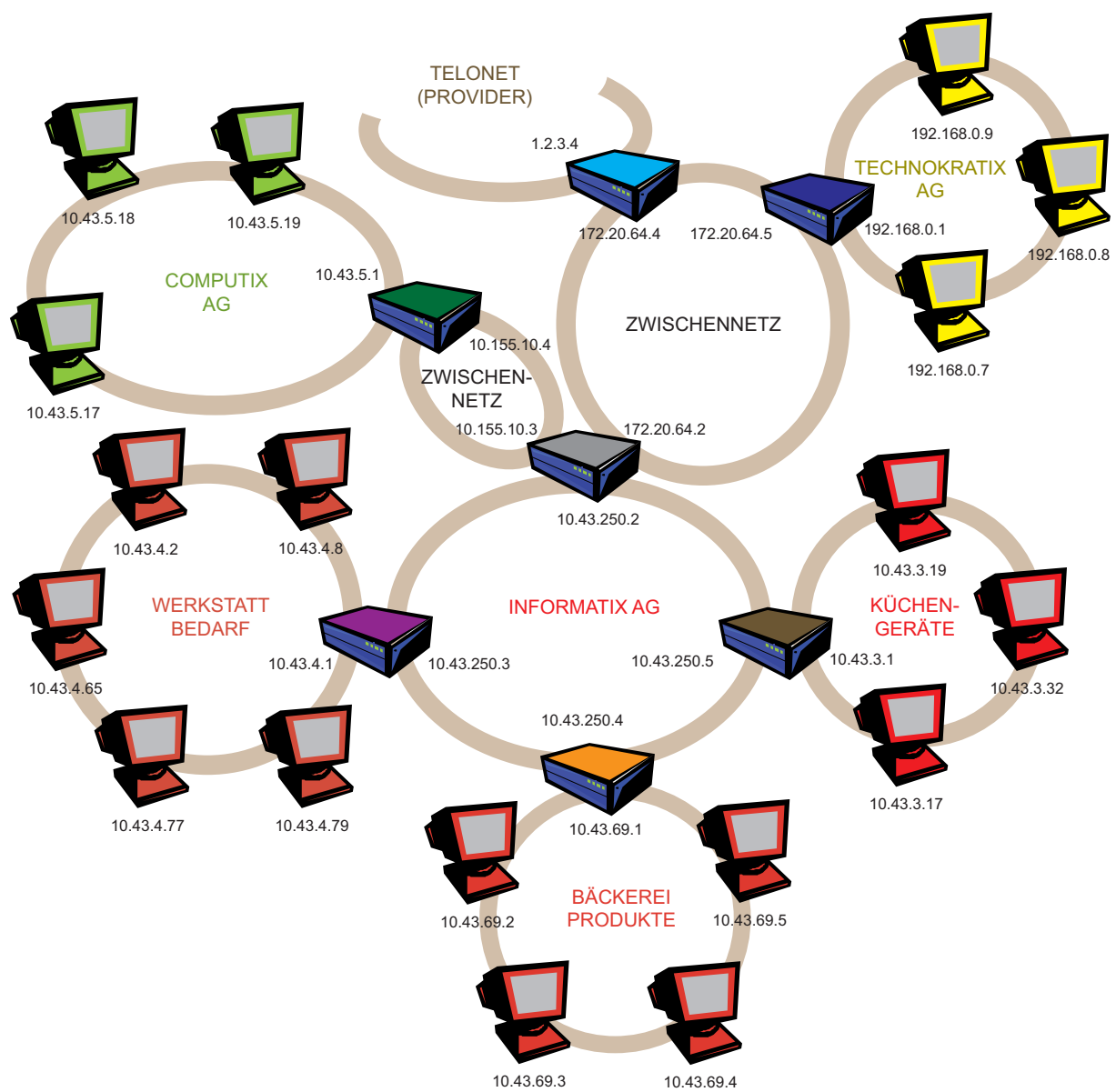
Kapitel 2





Diese Vorlage hat leider nicht auf eine einzelne Seite gepasst. Am besten kopieren Sie die Doppelseite, schneiden entsprechend in der Mitte überflüssige Bereiche ab und kleben die Seiten danach wieder passend zusammen, zum Beispiel mit Klebefilm.

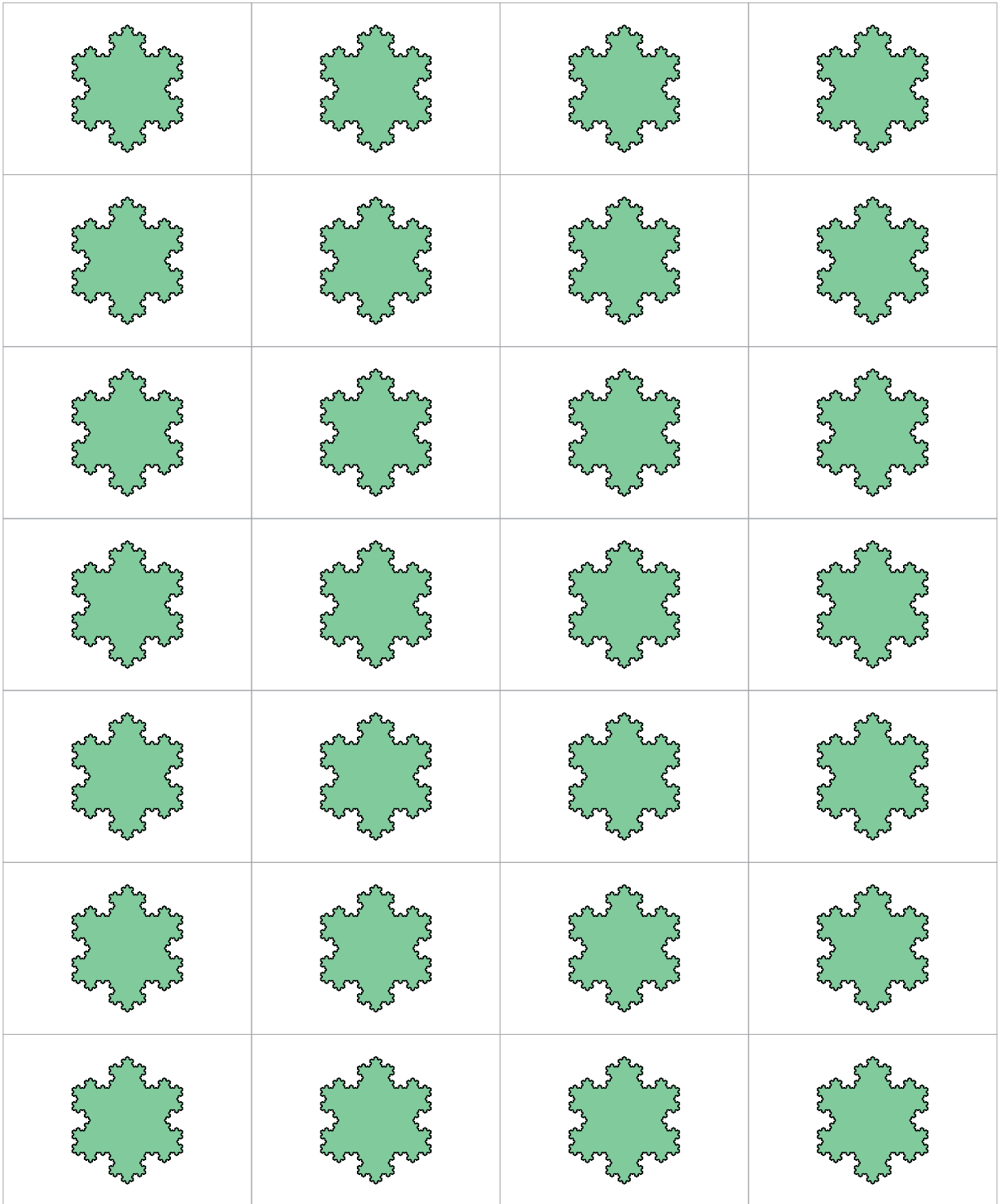
Kapitel 6



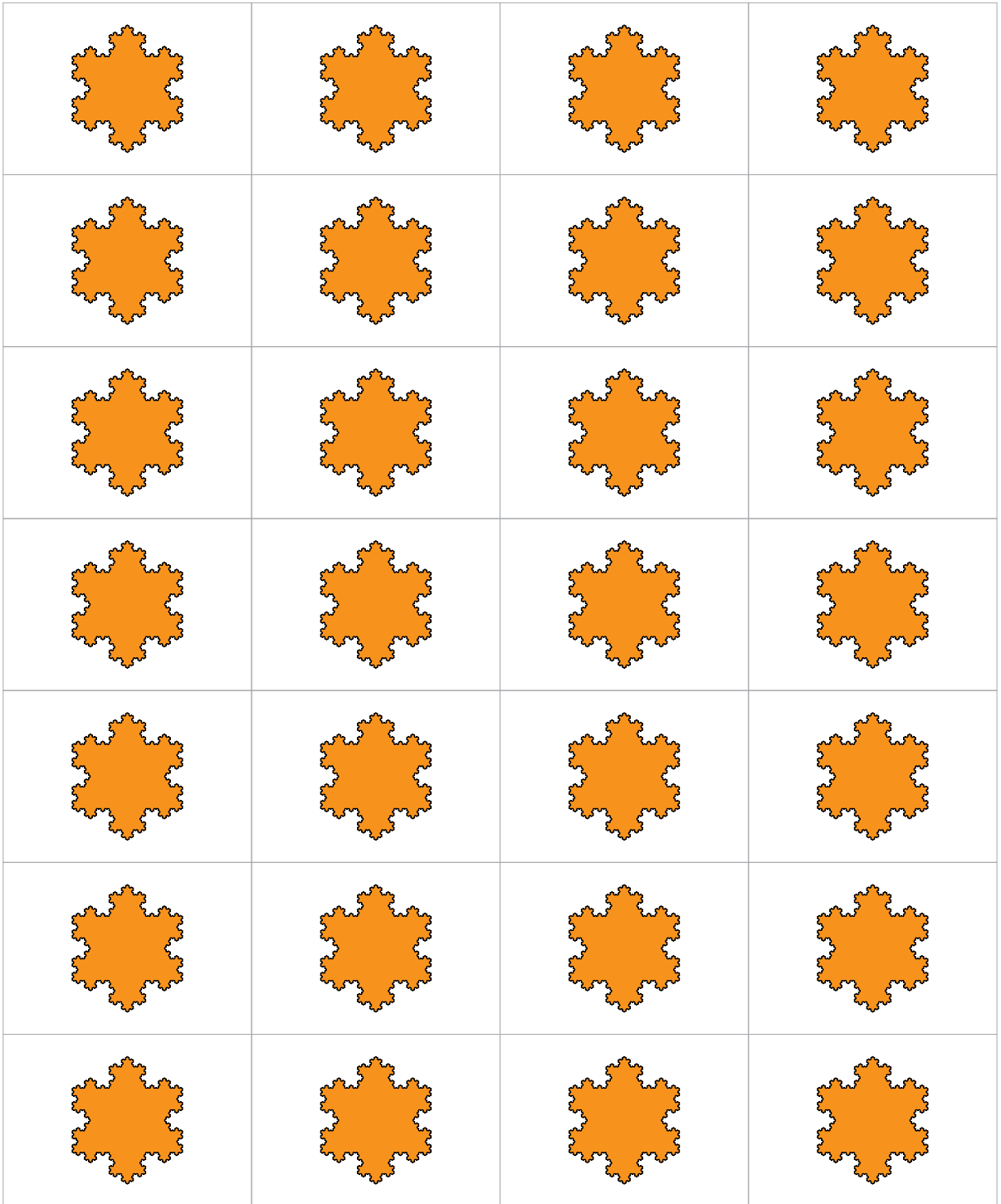
Anhang B: Bastelbögen

Kapitel 2

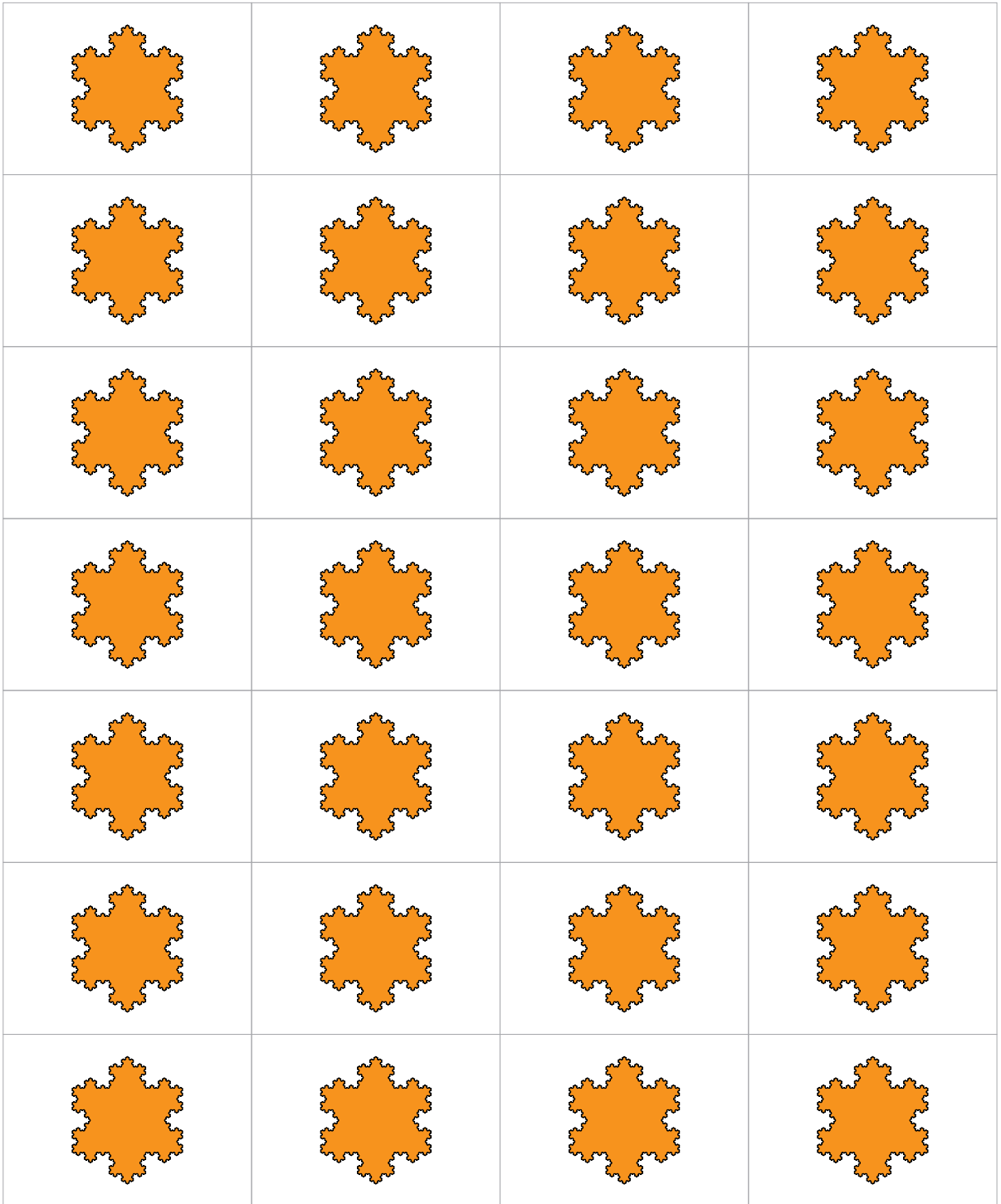
G	N	U	
F	M	T	
E	L	S	Z
D	K	R	Y
C	J	Q	X
B	I	P	W
A	H	O	V



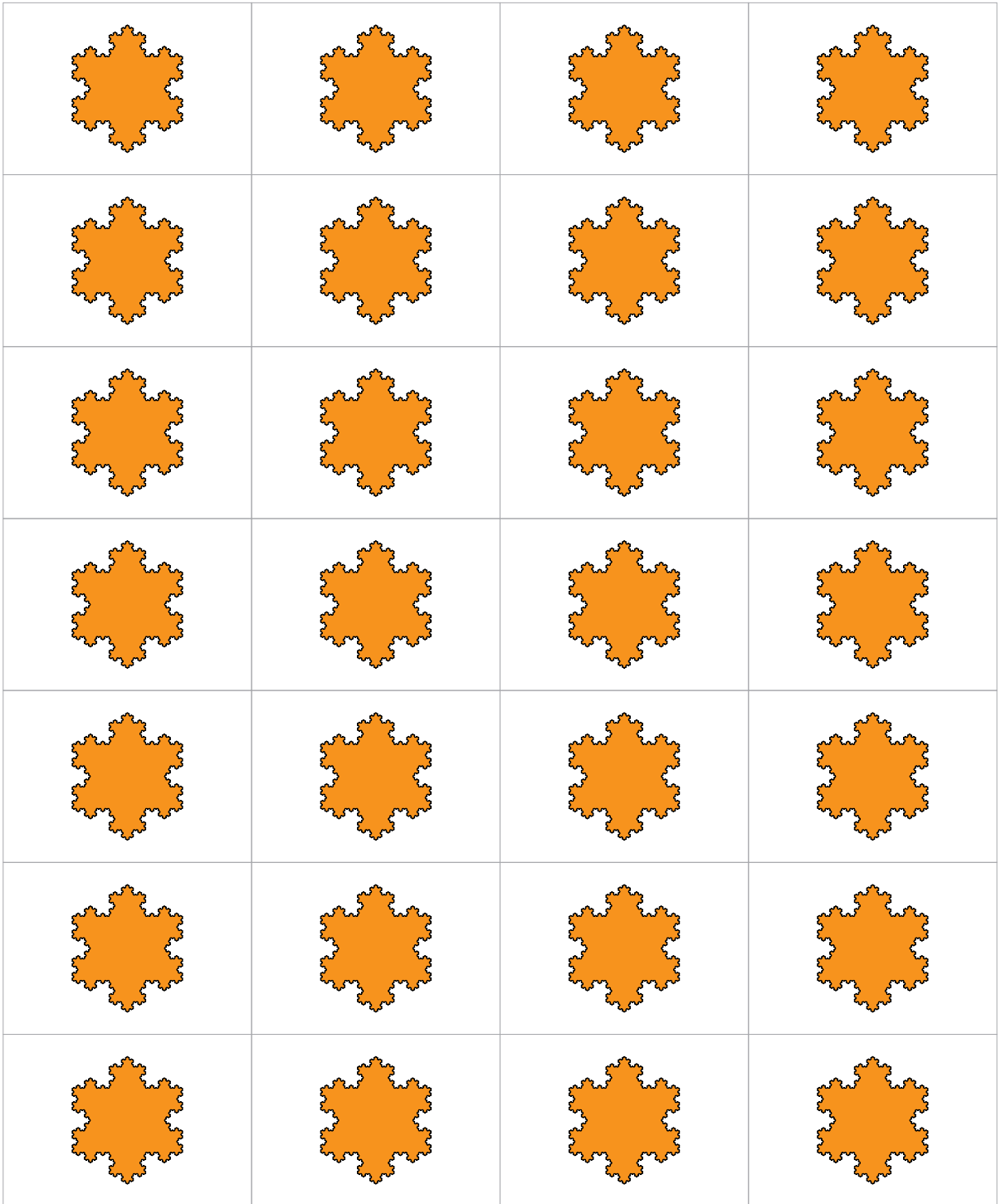
187650	aber	302782	503710	861867	212088	522107	38897
276637	Ablage	338210	absurd	Acker	Anbau	Anlage	Anteil
260249		425565	471452	544520	582278	260249	
955099	Arbeit	96588	424261	747690	44948	581176	133986
562184	Aufruf	444082	auftun	Ausland	Auslauf	Auster	Ausweg
124890		409121	559505	495493	163105	124890	
467199	Bank	145423	488893	968277	154571	165413	9859
448463	Beere	677562	Bein	bereit	Besitz	Beule	Birne
310313		551904	578884	708129	343721	310313	
112092	Boden	553475	869328	885439	475340	598622	391932
792094	Brot	497819	Busch	Dank	denn	Docht	Duft
551017		762015	328439	596164	578754	551017	



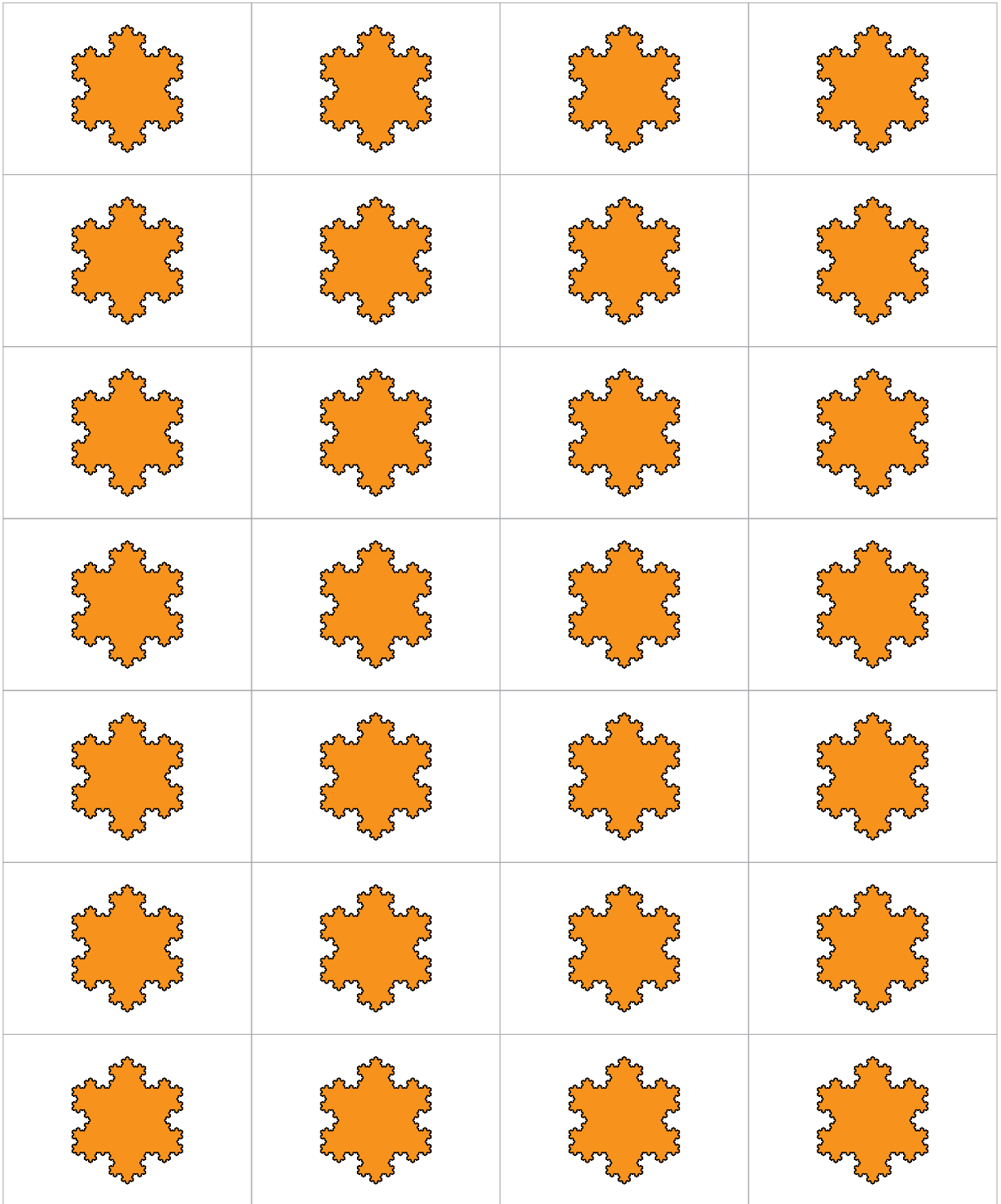
539477 Durst 303040	54218 ein 825829	369430 einmal 431497	570546 einzeln 686943	495973 Ente 313203	501884 Erde 545038	737014 erneut 511851
92280 Erz 416671	251907 falsch 196973	386655 Fell 348971	903955 Filter 541849	272409 Flug 456940	902289 frei 206299	761556 Fuhre 511274
245698 Garn 472166	79474 gefeit 541510	666701 Geist 527355	190206 genehm 501389	525643 Gesetz 454812	555035 gewagt 533659	417230 gleich 503130
770853 Graf 493004	122091 Gruppe 469414	220073 halb 502542	799190 Happen 373529	316190 Hefe 286658	323682 herb 567975	791873 Heu 411534



801774	hinter	645869	727378	242709	285853	468831	433707
769872	hold	587632	Idee	intern	jung	Karte	Kind
			365762	500126	454879	504286	981432
822890	Klette	47703	689770	494105	338328	10320	230696
656819	Komma	387491	Korb	Kreuz	Kur	lang	Leber
			485489	563432	716338	628007	657758
853090	Leute	999775	42536	601352	840530	344842	80693
456570	Los	261525	Mais	Maus	mild	Mohn	nach
			655509	557321	484208	408488	649042
173243	Narr	938664	442995	737455	825946	407769	194723
593174	nie	443699	Oase	Ozon	Perle	Platte	Presse
			955147	635066	616474	356123	394035

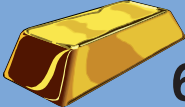


101144	Qualm	64452	889619	179025	298597	614559	693337
627000	Raub	33834	Regel	Report	Rost	Saal	schade
			299566	429514	434058	871139	536598
601096	Schein	26685	569868	663440	544775	621512	512644
746586	Schlag	532031	Schliff	Schrank	Schutz	schwer	selbst
			180319	356039	285673	570621	304128
265569	Sinn	1546	633803	875185	341378	922111	658614
534281	sozial	685547	Spitze	Stadt	Stein	Stock	streng
			511749	437755	819903	622771	791314
379917	Suppe	716582	208537	704238	919143	832882	406012
345638	Tausch	608609	toll	treu	Uhr	Umkehr	Umzug
			293607	415283	739765	534179	525784

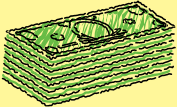


Kapitel 2 und 3

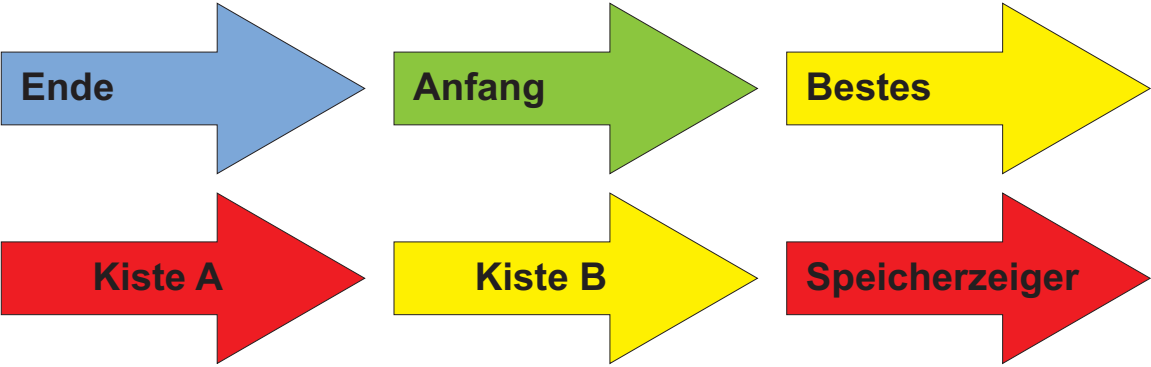
783299 Unheil 460043	813083 untreu 520134	715258 Vehikel 473739	229762 Verein 491187
532702 Verleih 308833	450663 Vers 473574	759881 Verweis 482518	626457 voll 468563
67142 Vorteil 409135	585876 Wand 430459	890751 Weide 66580	971956 Wert 715557
987582 Winter 642427	371795 Wurm 356724	941393 Zeit 381937	352600 Zirkus 491846

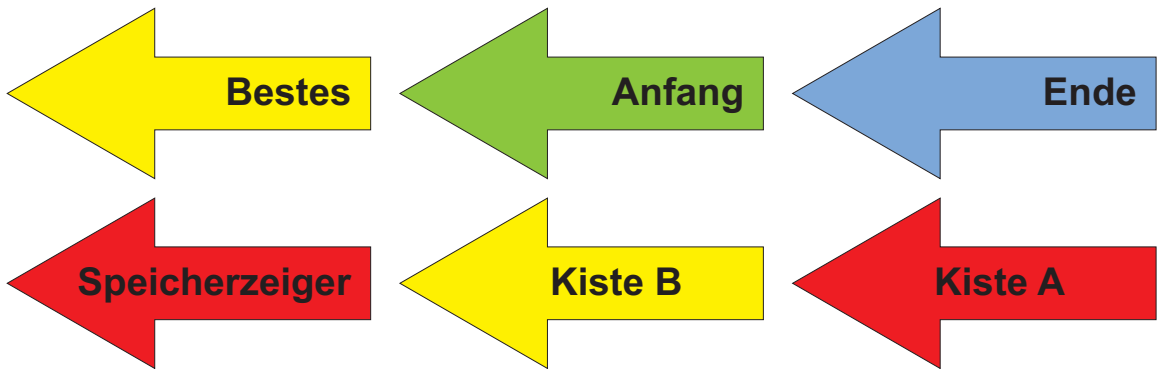
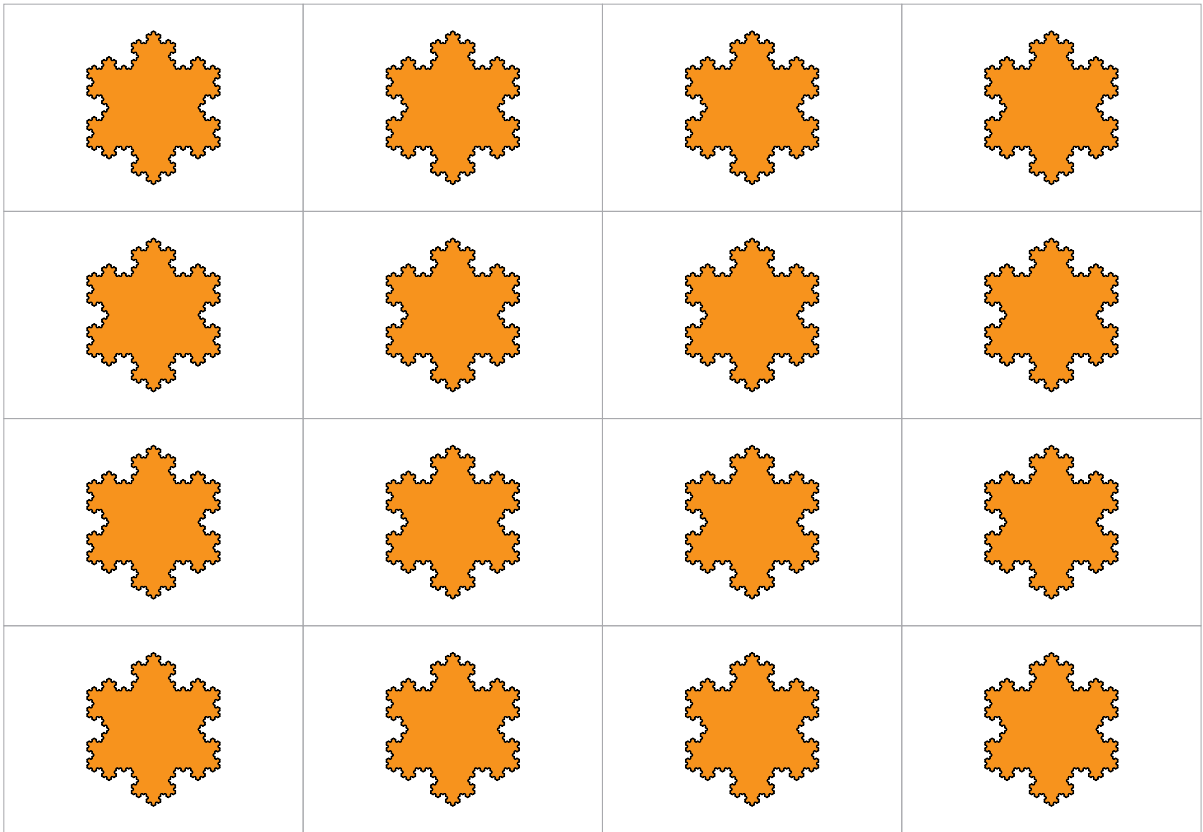


6



11

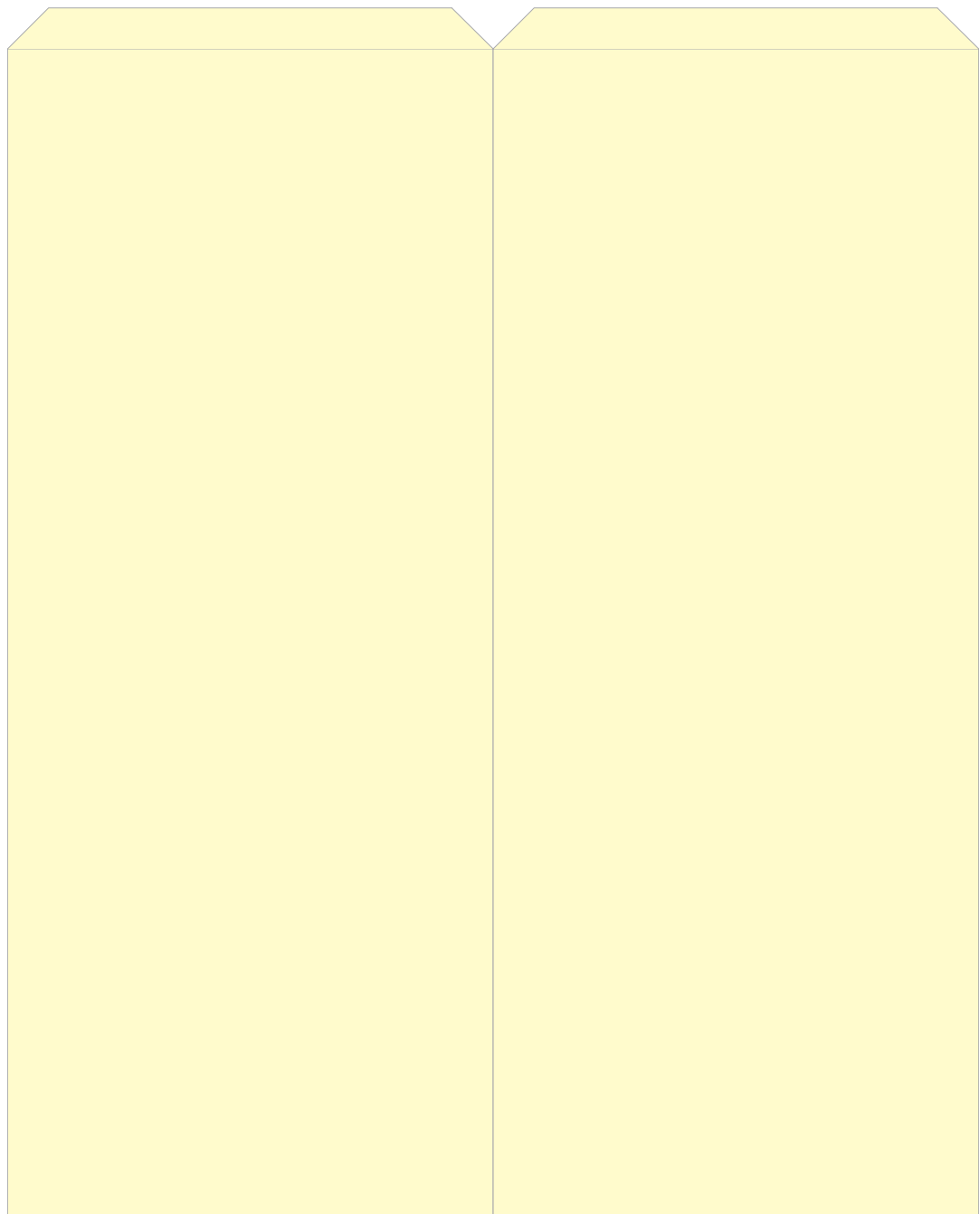




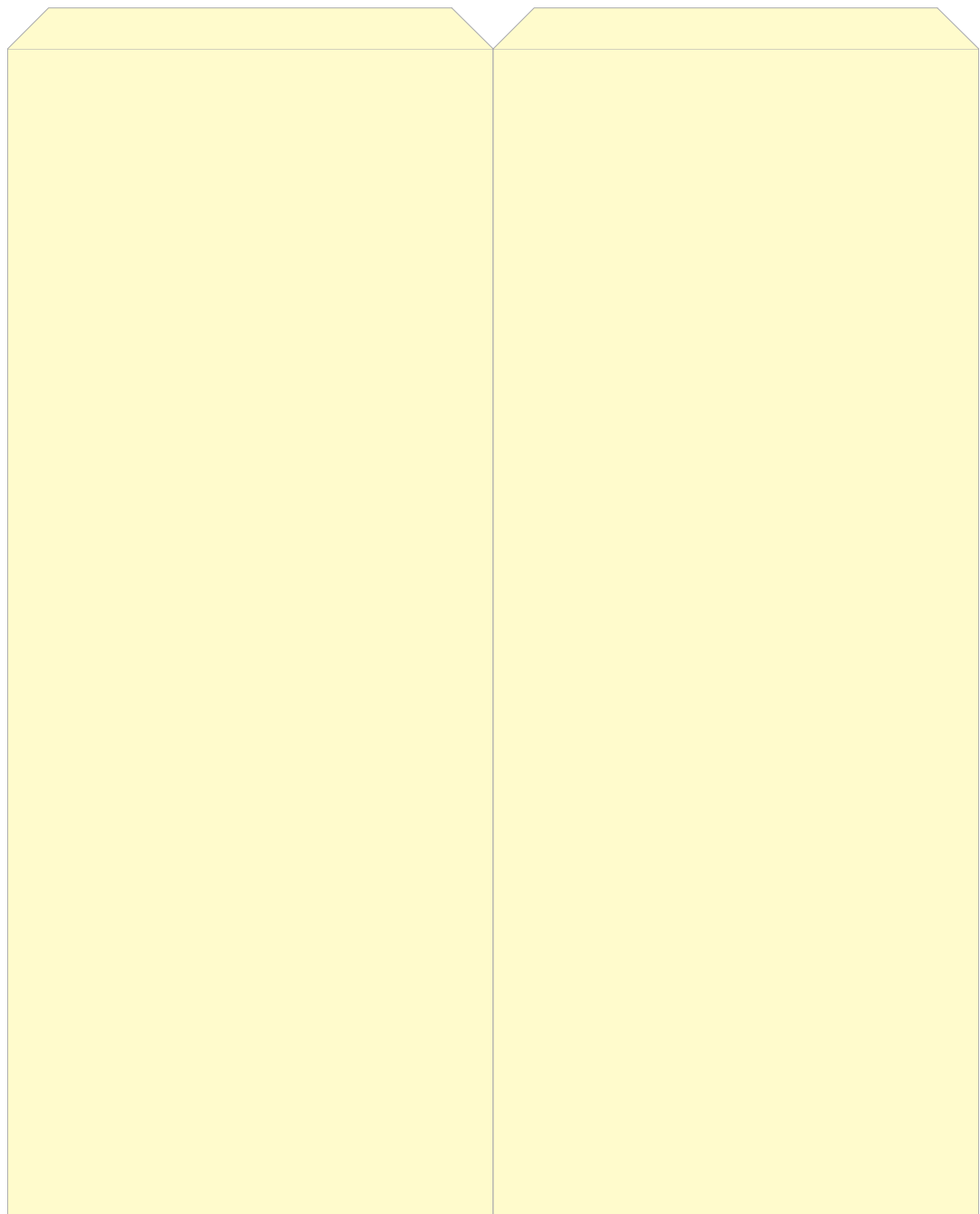
Kapitel 2 (Fortsetzung)

0-20 # 00 A	
21-29 # 01 B	
30-32 # 02 C	
33-35 # 03 D	
36-39 # 04 E	

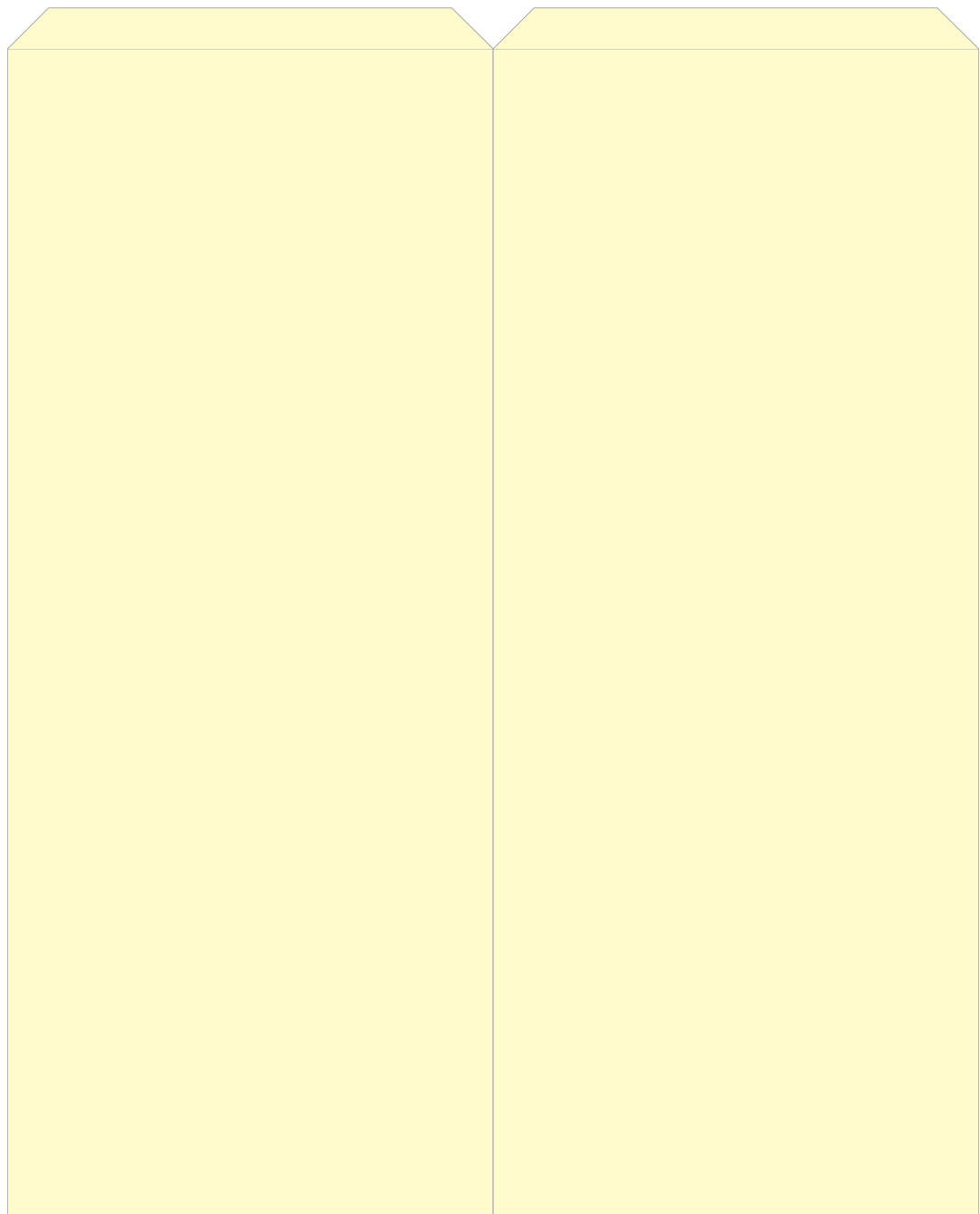
40-42 # 05 F	
43-44 # 06 G	
45-46 # 07 H	
47-48 # 08 I	
49 # 09 J	



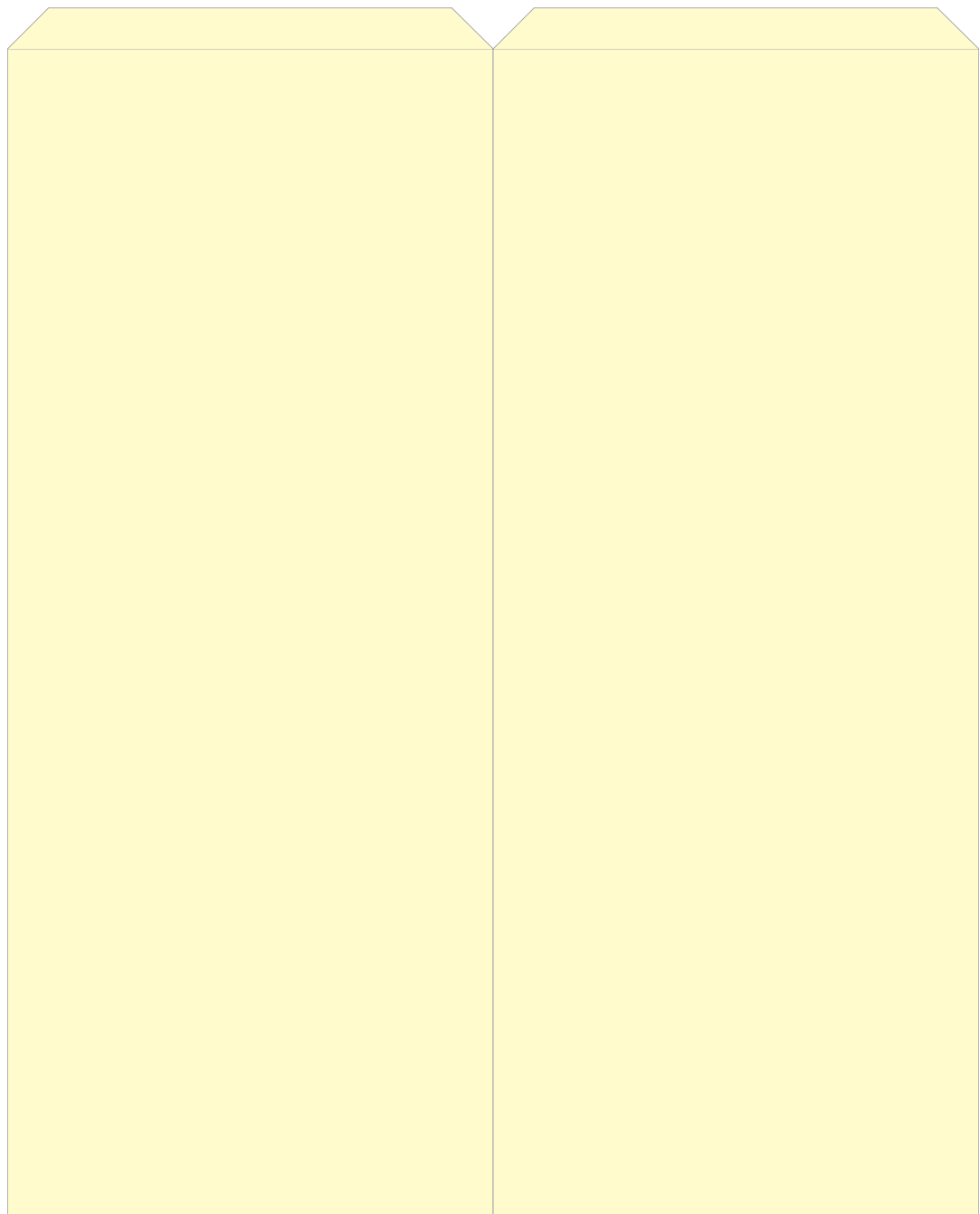
50 # 10 K	
51-52 # 11 L	
53-54 # 12 M	
55-56 # 13 N	
57-58 # 14 O	
59-61 # 15 P	
62-64 # 16 Q	
65-70 # 17 R	
71-78 # 18 S	
79-99 # 19 T	



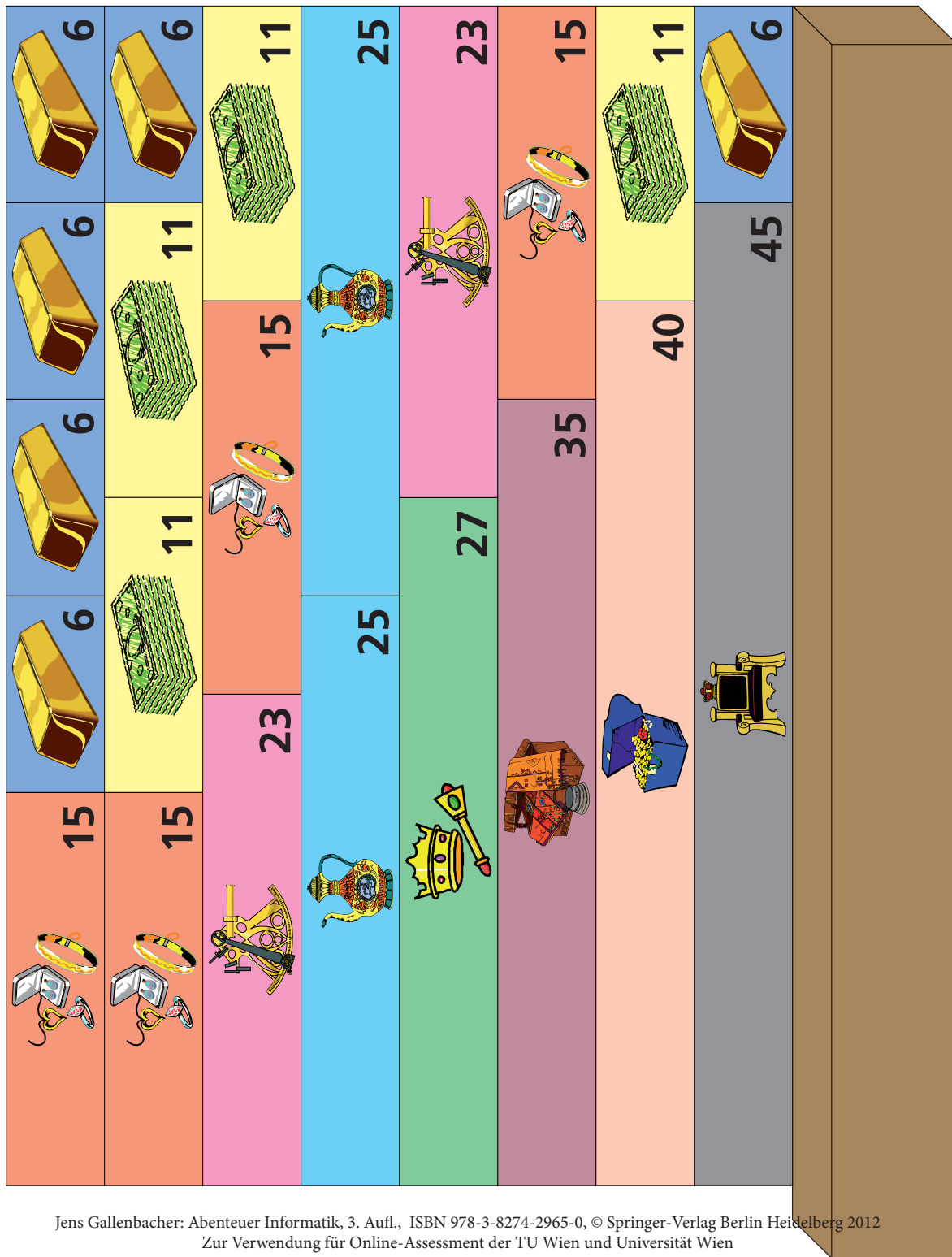
# 24 Y	
# 23 X	
# 22 W	
# 21 V	
# 20 U	
# 29	
# 28	
# 27	
# 26	
# 25 Z	

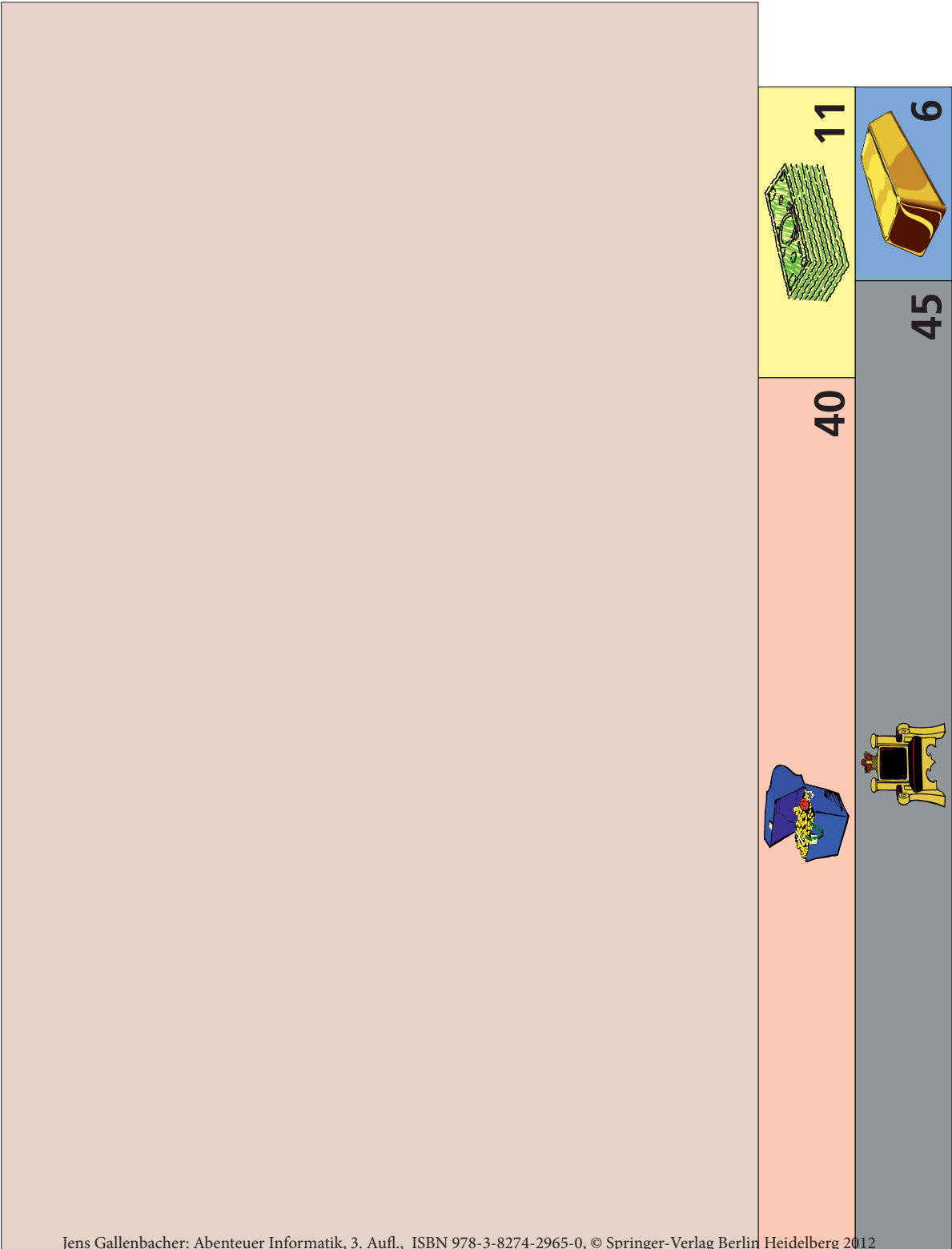


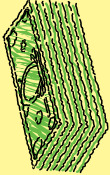
#	
#	
#	
#	
#	
#	

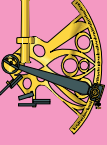
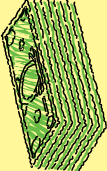
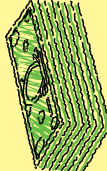
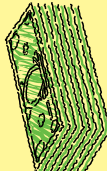


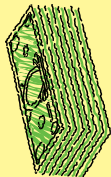
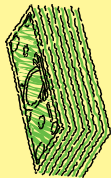
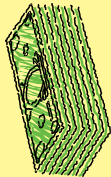
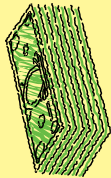
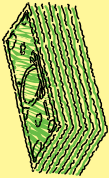
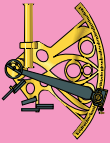
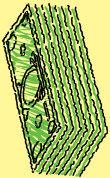
[illegible]

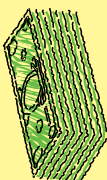
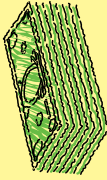
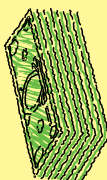
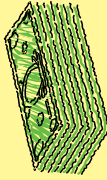


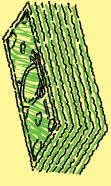
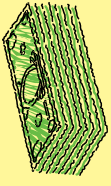
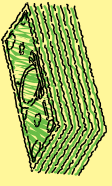
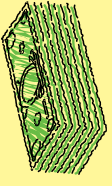
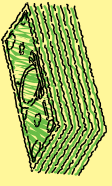
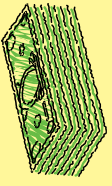
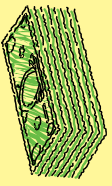
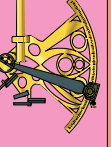
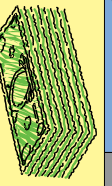
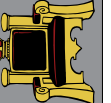


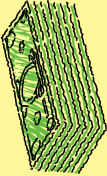
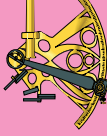
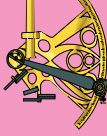
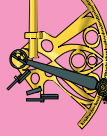
40	11	6
		
	45	

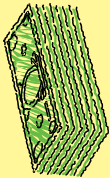
		35				15
		27				23
		25				6
		23				11
		15				6
		15				6
		27				23
		6				23
		11				11

11	11	11	11	11	11
					
6	6	11	23		
					
27			23		
					
15	6	6	6	6	6
					
15	11	11	11		
					
23	15	15	11		
					
25		27	23	6	
					
35			15	15	15
					

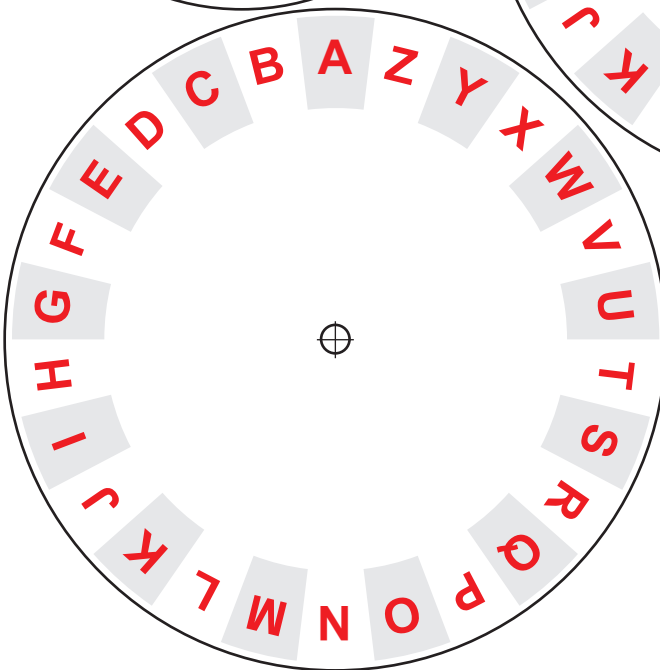
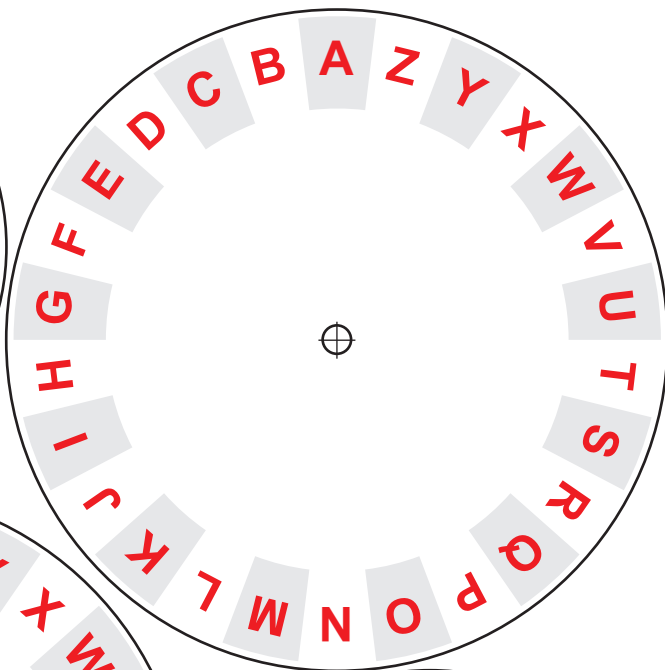
		45			6
		40			11
		35			15
		27			23
		25			25
					
		11		11	
		11			
					

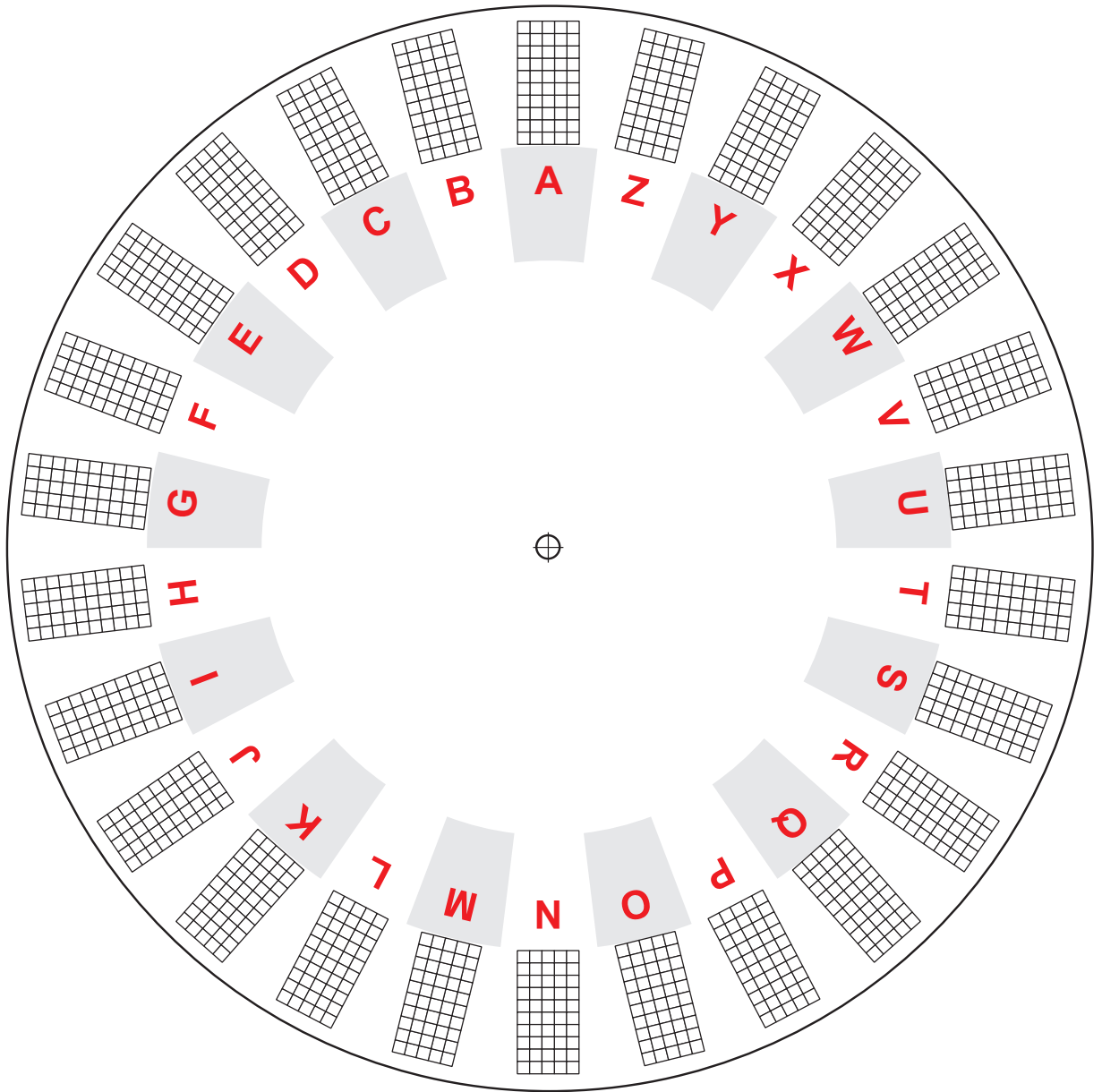
 6	 6	 6	 6	 6	 6
 11	 11	 11		 6	 6
 11	 11	 11	 11		
 6	 6	 6	 6	 6	 6
 25			 25		
 27			 23		
 35			 15		
 40			 11		
 45			 6		

		40			11
		35			15
		35			15
		27			23
		27			23
		25			25
	6		6		25
	6				23
		15			15

					
15		15		15	
	6		23		
			6	25	
	6			6	
	25		25		
			27		
			27		
			35		
			35		
			35		
		35		15	
		40			
				11	

Kapitel 9



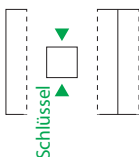


ASYM - Kodierer

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z - .

A O J V F E Y K M C H Q I - B T L U Z P R D X W G S N .
 B A T O L J U V Z F P E R Y D K X M W C G H S Q N I . -
 C P A H R O Q D J I X V - W F B G E T Q C S Y L N K U . M -
 D B K A X Y M O Z W L C J G U H V S B R N C T D . F H - L X
 E F C B U W A Y I H T R S - O Y Q L D N X Z B H Y J . W P T - Q K V
 G R L I M F A S D X F T N M P O I W E L . C R J - G Y U
 H D V B S K Z A Q X - N H W K R F U O B . Q G M D E Z J T
 I S C X Y P V L A - N H W K R F U O B . Q G M D E Z J T
 J E M Q B U R W N A V Y C I T Z D G . O F K H - L P X S
 K T W J H Z N E - D A M L G V Q P . Y B X O C - U S F I R
 L F D C N B J P K G I A U E X H . T V R M S - O Z Y W Q
 M U N Y T G F - X J Q R A C Z . K L S E B W - V I D O H P
 N G X R Z L B I H M Y F J A . S W D P U T - Q C K E V O
 O V E K C Q - T U P D W S . A J F Y M H I B L Z R X G N
 P H O D I V W B E S L K . Z C A R Q J X - F G T Y N U M
 Q W Y Z O - S M R V T . H X E U A I G K P J B N C D F L
 R I F S U C O X B Y . P Q V G L M A D - E N Z H J W T K
 S X P L - H K F O . G D Z T I C Y V A N W R U B Q M E J
 T J Z E D M G Q . B O U F R K W H N - A L V P Y X C S I
 U Y Q T P W . J U I K J N D L Q Y O S R D T H E A G M I B C F
 V K G - M V . X U I K J N D L Q Y O S R D T H E A G M I B C F
 W Z - M V . X U I K J N D L Q Y O S R D T H E A G M I B C F
 X L H F . D T C V N R B M J S P - K O G Z I Y A W Q E
 Y - R . E I P N F Q Z S V H U G J C L W O M T X A K B D
 Z M . U K N L Y S T E G B F W - V X I J D Q O R H A P C
 - . I N Q S H G C W M X K D Y R E P F Z V U J L O T A B
 . N S G W X D R P Z U L T B - I Q H C M K Y E F V J O A

G e h e - E s c h e f f



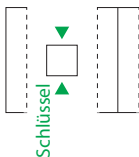
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z - .

K l a r t e x t

ASYM - Kodierer

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z - .

A	O	J	V	F	E	Y	K	M	C	H	Q	I	-	B	T	L	U	Z	P	R	D	X	W	G	S	N	.
B	A	T	O	L	J	U	V	Z	F	P	E	R	Y	D	K	X	M	W	C	G	H	S	Q	N	I	.	-
C	P	A	H	R	O	Q	D	J	I	X	V	-	W	F	B	G	E	T	S	Y	L	N	K	U	.	M	Z
D	B	K	A	X	T	M	O	W	L	C	J	G	U	H	V	S	Z	Q	F	N	P	I	E	.	R	-	Y
E	Q	U	W	A	Y	I	Z	G	O	K	-	P	S	J	M	B	R	N	V	C	T	D	.	F	H	L	X
F	C	B	P	G	A	E	H	T	R	S	-	O	Y	Q	L	D	N	J	K	I	X	.	V	M	-	Z	W
G	R	L	I	M	F	A	S	D	Q	X	-	C	E	O	N	P	O	I	W	E	L	.	C	R	J	-	Q
H	D	V	B	S	K	Z	A	A	Q	-	F	N	H	W	K	R	F	U	O	B	.	Q	G	M	D	-	E
I	S	C	X	Y	P	V	L	A	-	N	H	W	K	R	F	U	O	B	.	Q	G	M	D	-	E	Z	J
J	E	M	Q	B	U	R	W	N	A	V	Y	C	I	T	Z	D	G	.	O	F	K	H	-	L	P	X	S
K	T	W	J	H	Z	N	E	-	D	A	M	L	G	V	Q	P	.	Y	B	X	O	C	U	S	F	I	R
L	F	D	C	N	B	J	P	K	G	I	A	U	E	X	H	.	T	V	R	M	S	-	O	Z	Y	W	Q
M	U	N	Y	T	G	F	-	X	J	Q	R	A	C	Z	.	K	L	S	E	B	W	V	I	D	O	H	P
N	G	X	R	Z	L	B	I	H	M	Y	F	J	A	.	S	W	D	P	U	T	-	Q	C	K	E	V	O
O	V	E	K	C	Q	-	T	U	P	D	W	S	.	A	J	F	Y	M	H	I	B	L	Z	R	X	G	N
P	H	O	D	I	V	W	B	E	S	L	K	.	Z	C	A	R	Q	J	X	-	F	G	T	Y	N	U	M
Q	W	Y	Z	O	-	S	M	R	V	T	.	H	X	E	U	A	I	G	K	P	J	B	N	C	D	F	L
R	I	F	S	U	C	O	X	B	Y	.	P	Q	V	G	L	M	A	D	-	E	N	Z	H	J	W	T	K
S	X	P	L	-	H	K	F	O	.	G	D	Z	T	I	C	Y	V	A	N	W	R	U	B	Q	M	E	J
T	J	Z	E	D	M	G	Q	.	B	O	U	F	R	K	W	H	N	-	A	L	V	P	Y	X	C	S	I
U	Y	G	-	J	R	C	.	L	E	W	I	O	P	M	N	T	F	X	Q	A	Z	K	S	B	V	D	H
V	K	Q	T	P	W	.	J	Y	H	B	Z	X	N	O	E	C	-	U	D	S	A	F	M	I	L	R	G
W	Z	-	M	V	.	X	U	I	K	J	N	D	M	L	Q	S	P	-	S	R	T	H	E	A	G	P	B
X	L	H	F	.	D	T	C	V	N	R	B	J	S	P	-	K	O	G	Z	I	Y	A	W	U	Q	E	F
Y	-	R	.	E	I	P	N	F	Q	Z	S	V	H	U	G	J	C	L	W	O	M	T	X	A	K	B	D
Z	M	.	U	K	N	L	Y	S	T	E	G	B	F	W	-	V	X	I	J	D	Q	O	R	H	A	P	C
-	.	I	N	Q	S	H	G	C	W	M	X	K	D	Y	R	E	P	F	Z	V	U	J	L	O	T	A	B
.	N	S	G	W	X	D	R	P	Z	U	L	T	B	-	I	Q	H	C	M	K	Y	E	F	V	J	O	A

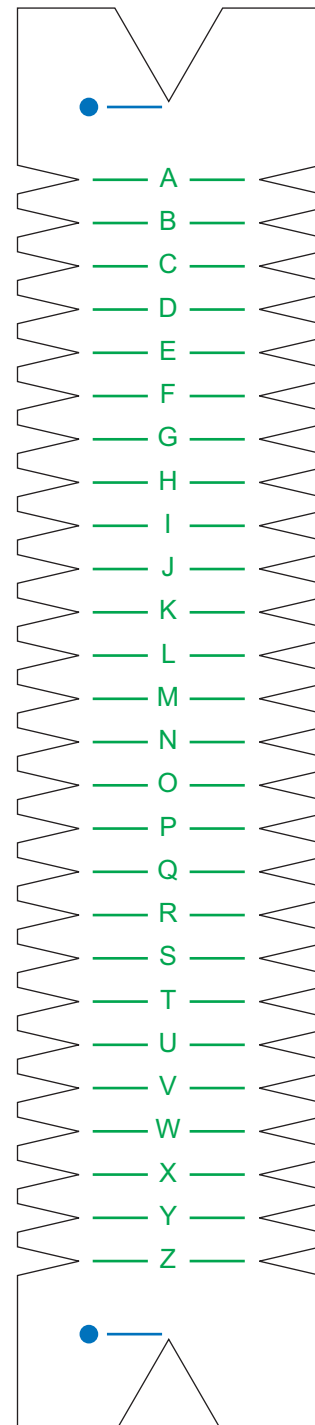


G e h e i m s c h r i f f

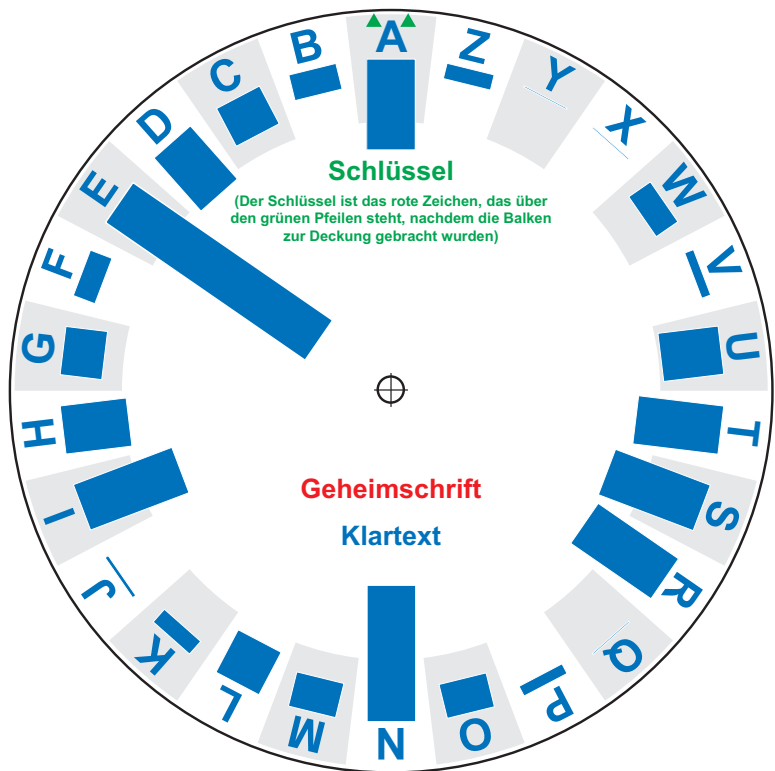
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z - .

K l a r t e x t

PUT	WYG	IGOR	GECKO
TRP	XGY	MYBU	YFJHB
QST	BOT	JILL	RIXEL
LZP	OBP	CMQQ	UMWFQ
CLU	KVM	PROX	QUYOF
JQR	HDI	TUBW	LRGBE
HRP	CIJ	IDEE	PROTO
KUT	JMC	MVFF	TUBPB
OLE	XUG	STYB	BLUME
BQF	WRY	ZPGO	OQRIF
KLE	ITZ	GOHS	EZRIK
HQF	MPS	YBKZ	FSUMH



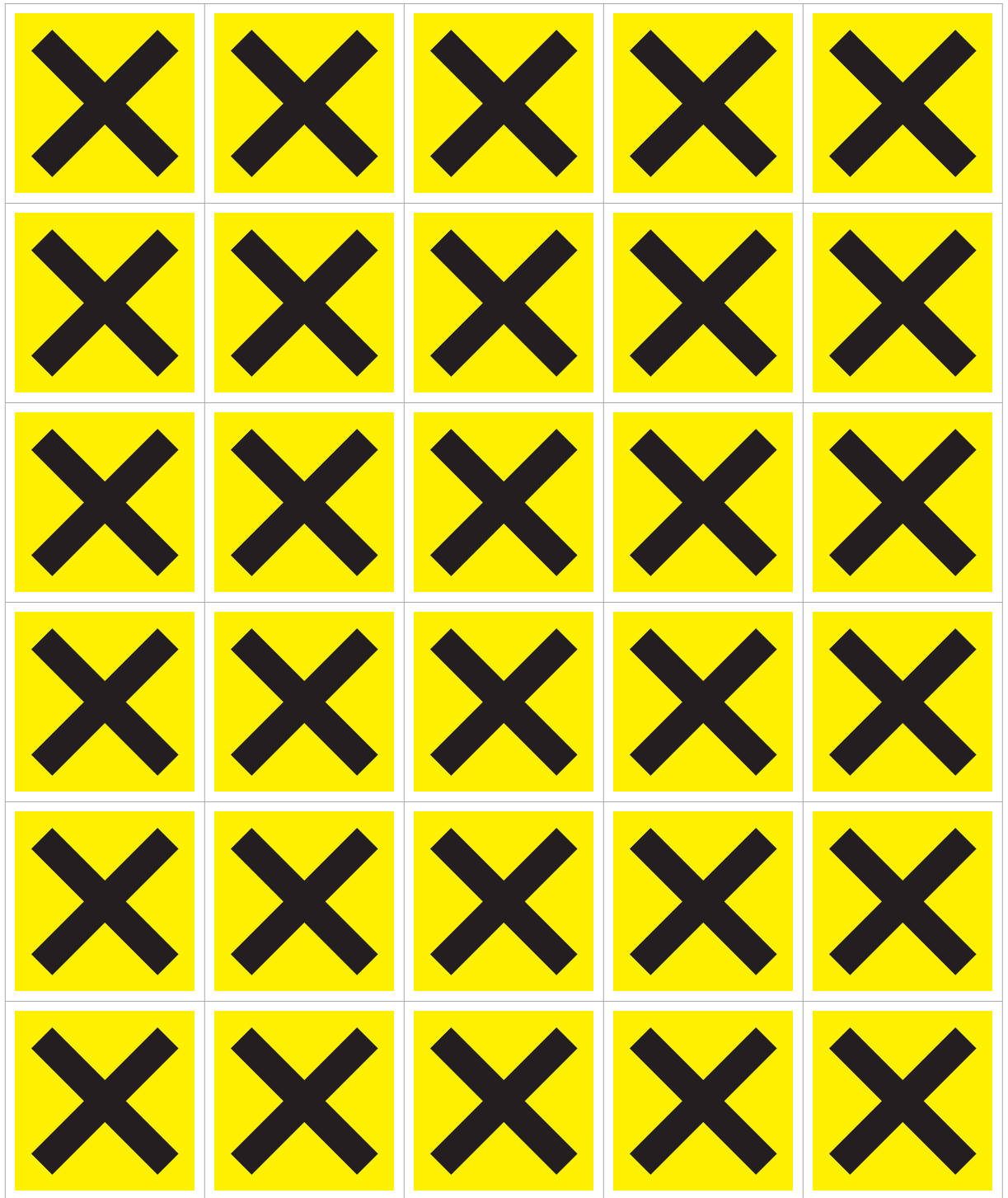
Kapitel 5 und 9

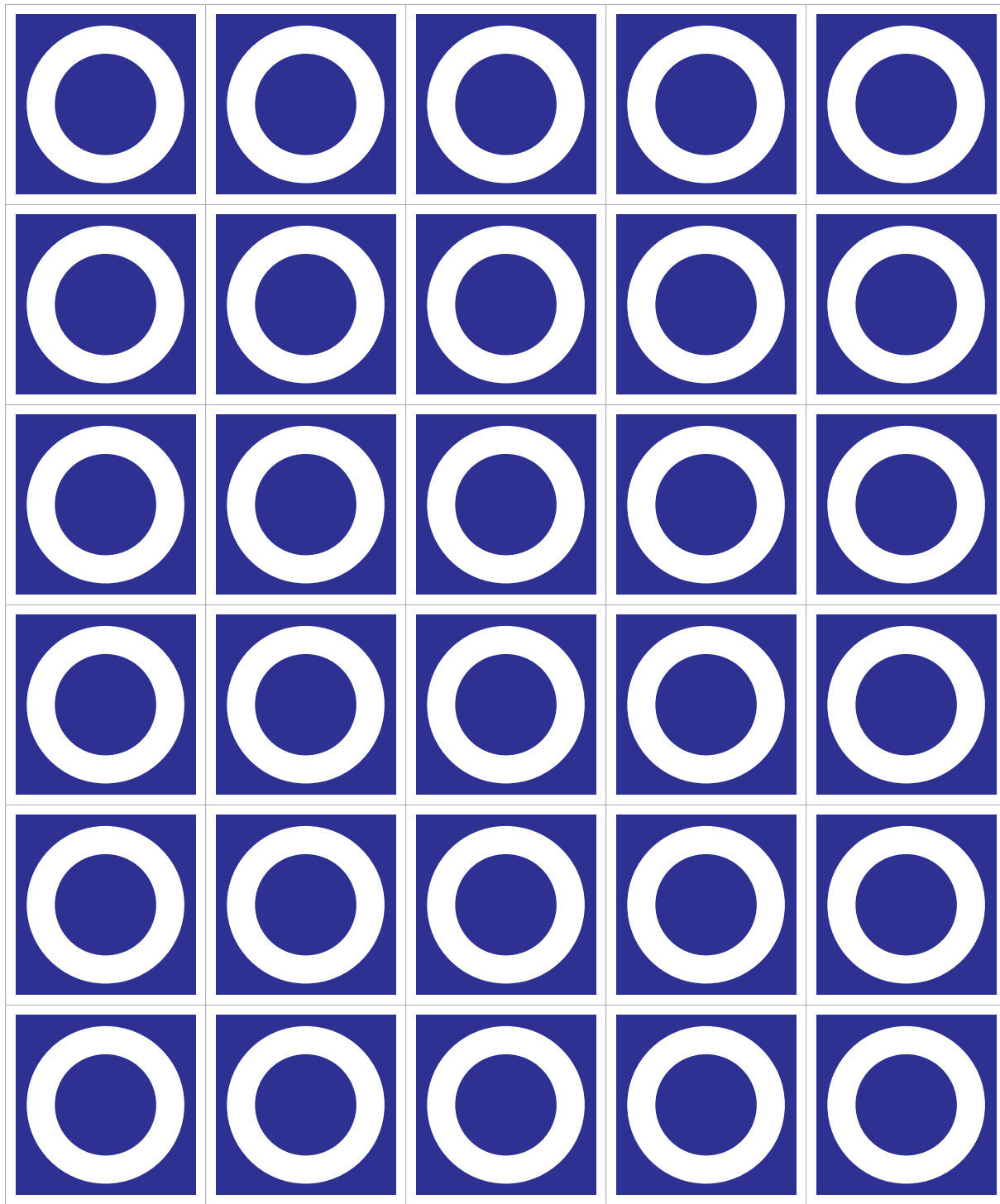


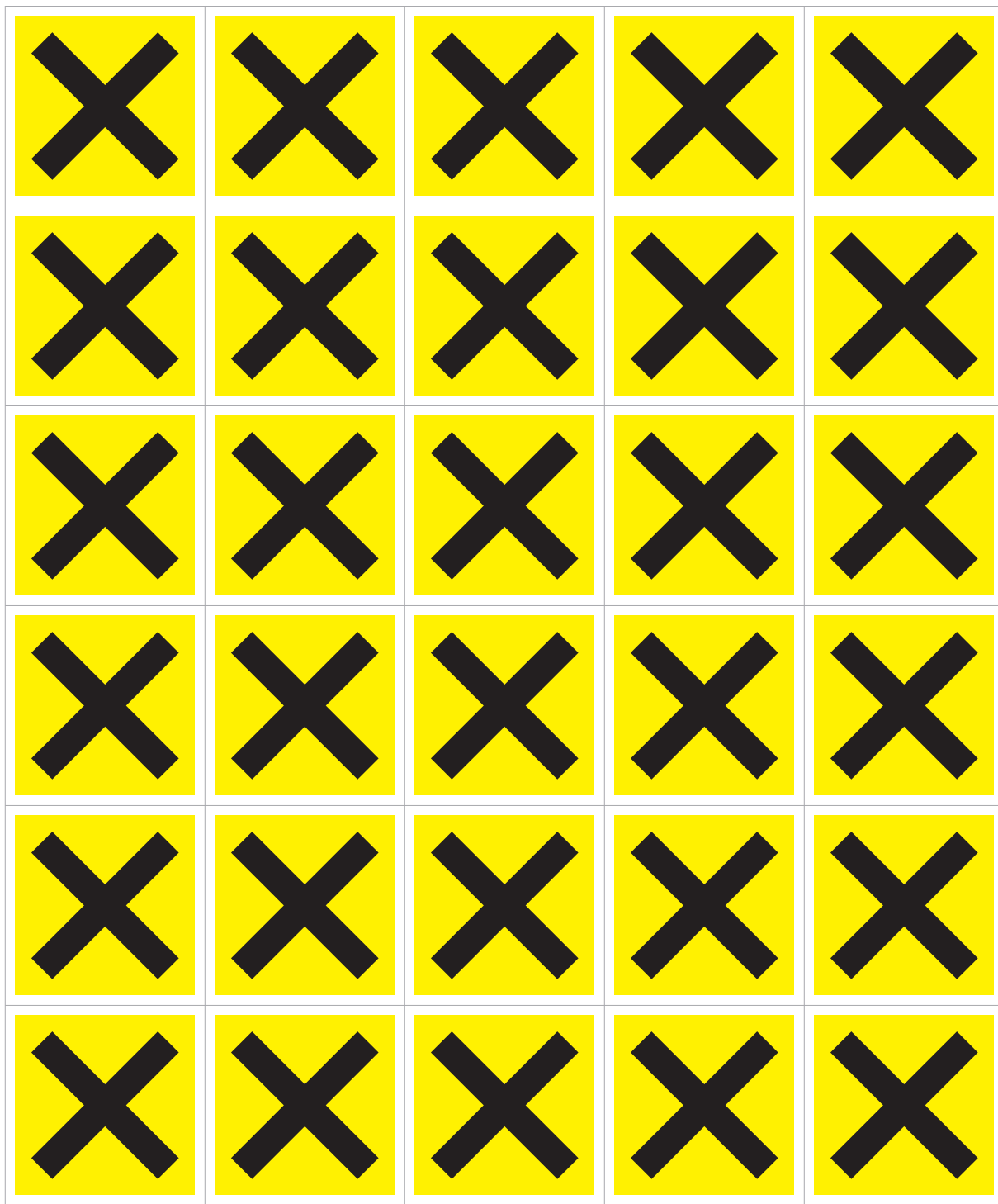
	—	1850
	A	530
	B	154
	C	249
	D	414
	E	1417
	F	135
	G	245
	H	388
	I	615
	J	22
	K	99
	L	280
	M	206
	N	796
	O	204
	P	64
	Q	2
	R	570
	S	592
	T	501
	U	354
	V	55
	W	154
	X	2
	Y	3
	Z	92
	.	2
	,	2
	:	1
	-	1
	#	1

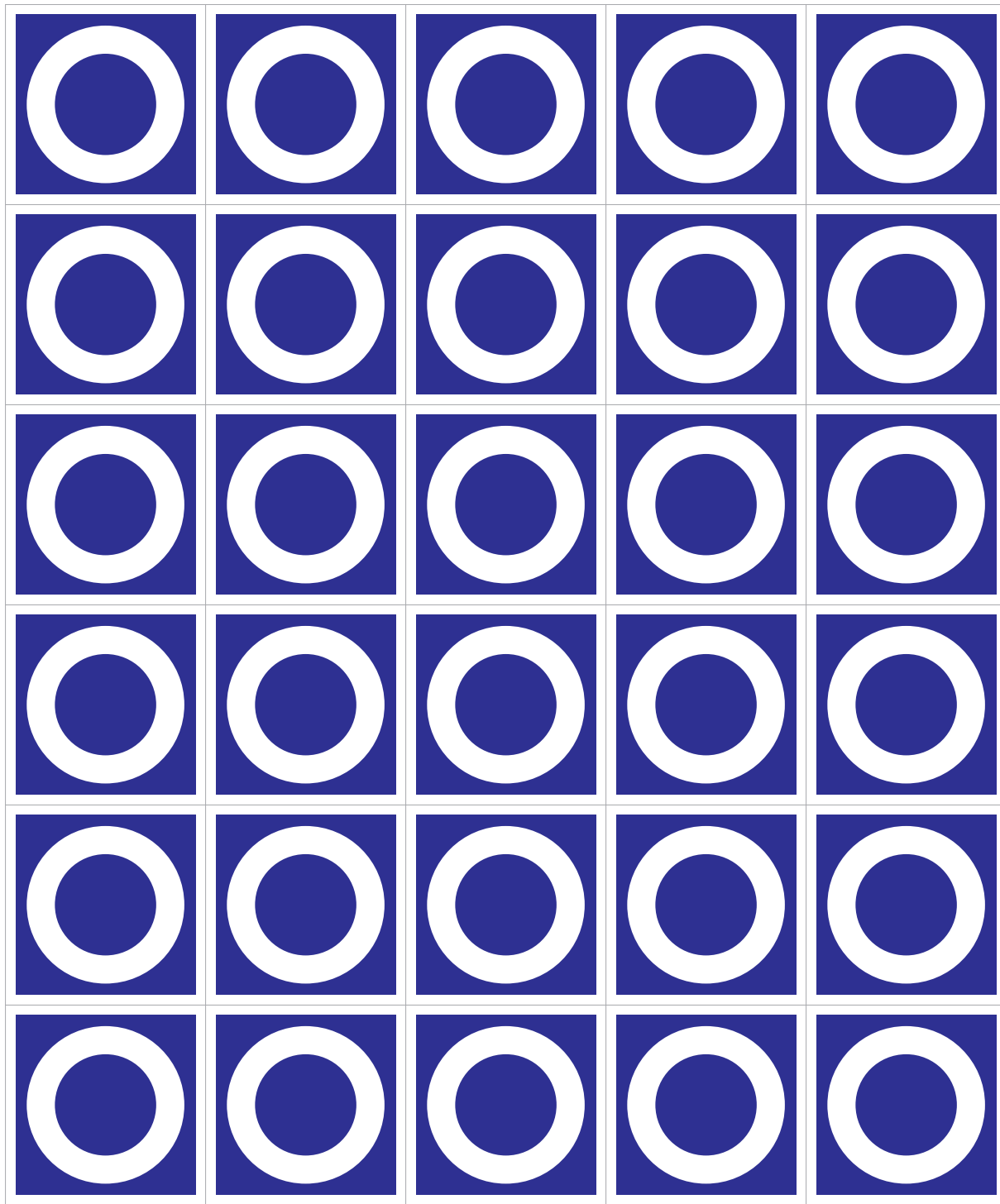
	—	H	P	X
	A	I	Q	Y
	B	J	R	Z
	C	K	S	.
	D	L	T	,
	E	M	U	:
	F	N	V	-
	G	O	W	#

Kapitel 11

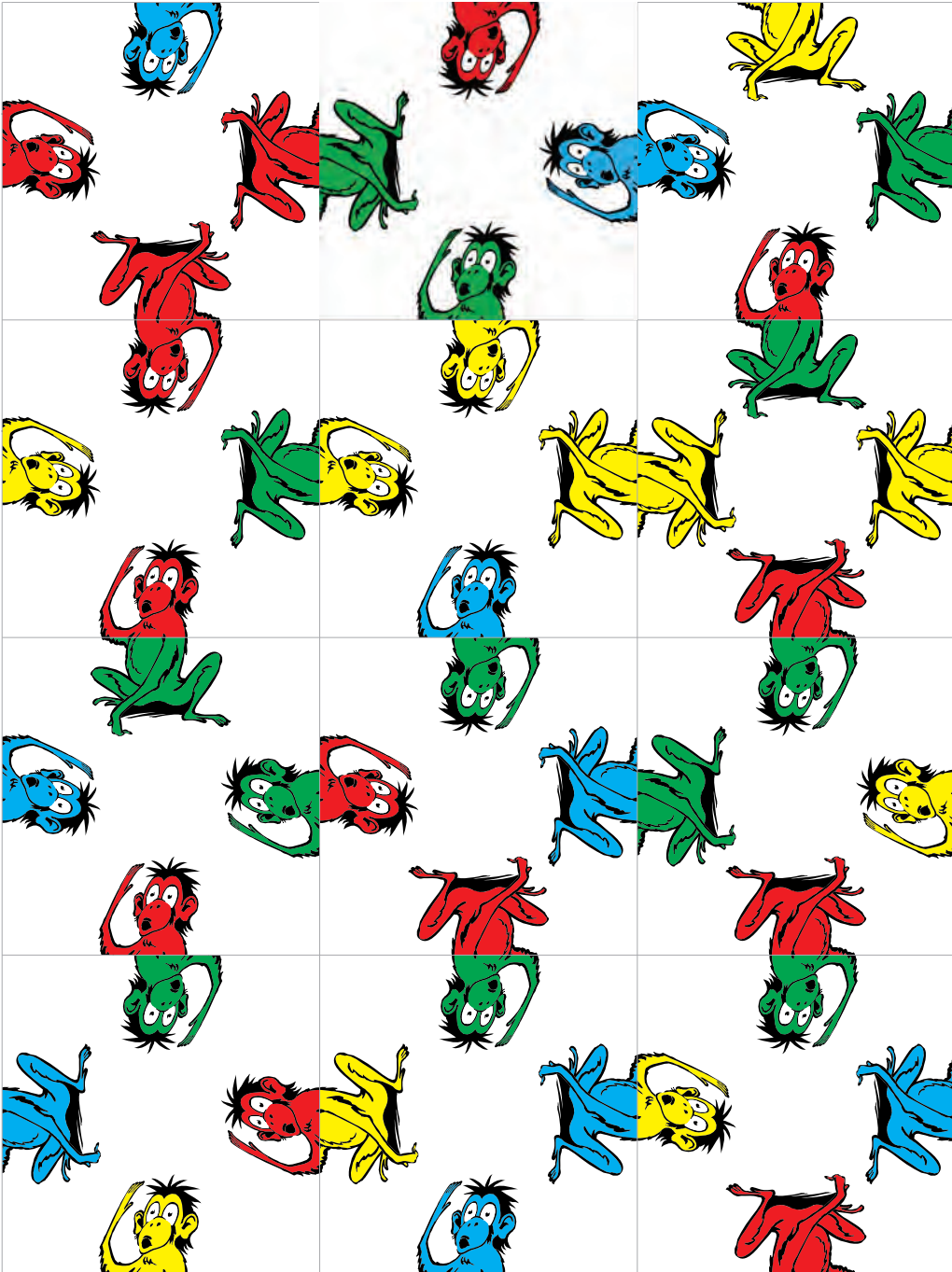




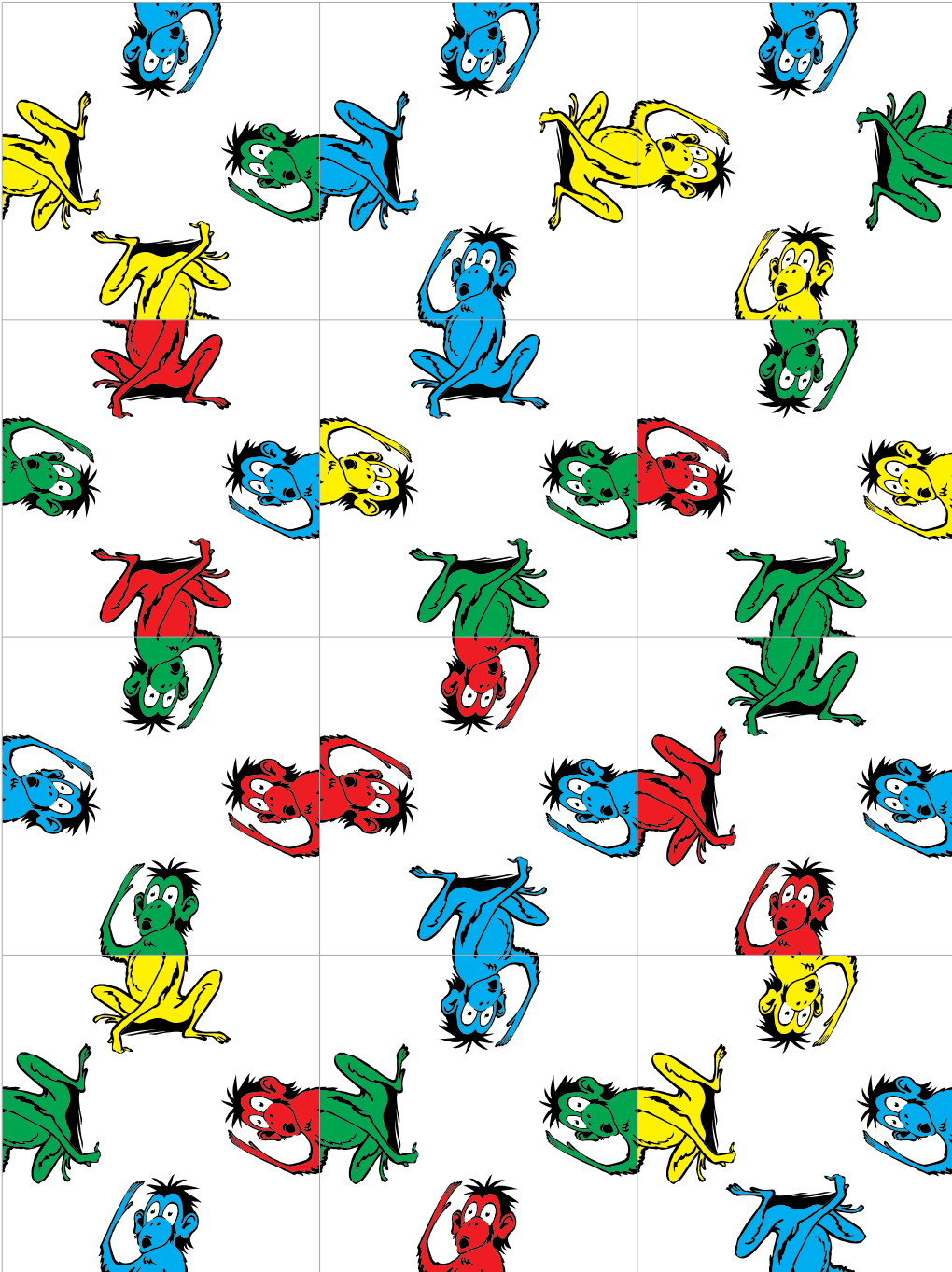




Kapitel 12



97	89	26
28	10	20
19	79	14
39	71	39



84	69	46
65	50	92
43	15	93
35	32	75

