Michael Fothe

Kunterbunte Schulinformatik

Ideen für einen kompetenzorientierten Unterricht in den Sekundarstufen I und II

LOG IN VERLAG


LOG IN
VERLAG

Michael Fothe

Kunterbunte Schulinformatik

Ideen für einen kompetenzorientierten Unterricht
in den Sekundarstufen I und II

LOG IN Verlag Berlin 2010

Fothe, Michael: Kunterbunte Schulinformatik – Ideen für einen kompetenzorientierten Unterricht in den Sekundarstufen I und II. LOG IN Verlag, Berlin, 2010.

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

ISBN 978-3-9805540-8-4

Für das Können gibt es nur einen Beweis:
das Tun.

Marie von Ebner-Eschenbach

Vorwort

Junge Menschen müssen sowohl Wissen zu Konzepten der Informatik als auch Fertigkeiten, mit deren Produkten umzugehen, erwerben. Beides gehört zusammen und macht informatische Bildung aus[1]. Das Herausbilden und weitere Einüben der Fertigkeiten kann teilweise in anderen Fächern erfolgen. Die Konzepte können jedoch nur in einer verbindlichen Lernzeit und durch qualifizierte Lehrerinnen und Lehrer vermittelt werden. Für alle Schülerinnen und Schüler sollte durchgängig eine Wochenstunde Informatik ab der Klassenstufe 5 als Minimum gelten.

Bildungsstandards sind ein wichtiger Beitrag zur inhaltlichen Positionsbestimmung und können auch der Legitimation eines Unterrichtsfaches in der Schule dienen. Deren Einführung ist mit der Erwartung verbunden, dass eine Qualitätsverbesserung eintritt und dass das Handeln der Schulen verantwortlicher, abrechenbarer und transparenter wird.

Bildungsstandards der Kultusministerkonferenz (KMK) für das Fach Informatik sind bisher nicht erschienen. Es gibt die einheitlichen Prüfungsanforderungen der KMK in der Abiturprüfung Informatik von 2004 (EPA Informatik) und die von der Gesellschaft für Informatik e.V. (GI) im Jahr 2008 veröffentlichten Empfehlungen zu Bildungsstandards Informatik. GI-Empfehlungen und EPA Informatik werden in diesem Buch vorgestellt und eingeordnet. Darüber hinaus geht es auch um das Zentralabitur im Fach Informatik. Im Mittelpunkt stehen jedoch Ideen für das Gestalten von Informatikunterricht. Das Buch richtet sich vorrangig an Personen, die Unterricht auf der Grundlage von Kompetenzbeschreibungen planen, durchführen und reflektieren wollen. Ziel ist das Entwickeln einer Kultur des sinnvollen Umgangs mit Bildungsstandards. Dafür sollen Anregungen gegeben werden – und nicht etwa verbindliche Handlungsanleitungen.

Das Wort »kunterbunt« soll für *vielfältig* und *abwechslungsreich* stehen und durch mögliche Assoziationen – beispielsweise zur »Villa Kunterbunt« – auch daran erinnern, dass es bei allem, was an Schulen geschieht, um Kinder und Jugendliche geht.

Grundsätze eines kompetenzorientierten Unterrichts sind Gegenstand von Kapitel 1. In den Kapiteln 2 bis 7 steht der Unterricht der Sekundarstufe I im Mittelpunkt, in den Kapiteln 8 bis 16 geht es vorrangig um die Sekundarstufe II. Allgemeine Aussagen werden mit (Fall-)Beispielen untersetzt. Eine Auswahl an

[1] http://www.dlgi.de/uploads/media/DLGI_Magazin_0408.pdf (S. 14 f.)

Beispielen hat stets auch subjektiven Charakter. Ich hoffe dennoch, dass die Beispiele auf das Interesse der Leserinnen und Leser treffen.

Schwerpunkte in der Sekundarstufe I sind Themen, bei denen die »Geschichte als Steinbruch« für den Informatikunterricht genutzt wird (Roland Stowasser tat dies für den Mathematikunterricht bereits in den 1970er-Jahren), Themen mit Bezügen zu anderen Fächern sowie die Nutzung eines dreistufigen Kompetenzmodells als Grundlage für einen differenzierten Unterricht. Für die Sekundarstufe II stehen solche Themen im Mittelpunkt, die nach meiner Erfahrung als ehemaliger Landesfachberater für Informatik in Thüringen als schwierig (zu unterrichten) gelten und daher eine sorgfältige Aufarbeitung benötigen. Dazu zählen Rekursion, Compilerbau, prinzipielle Grenzen der Berechenbarkeit und das Bestimmen des Zeitaufwands des Ablaufs von Algorithmen. Nach Möglichkeit werden Bezüge zu den EPA Informatik hergestellt. Die verwendeten Programmiersprachen sind ebenfalls exemplarisch. Wesentliches Kriterium für deren Auswahl sind umfassendere eigene Unterrichtserfahrungen. Oberon-2 ist ein Repräsentant der Pascal-Sprachen, Python eine objektorientierte Skriptsprache und Prolog ein Klassiker unter den logischen Programmiersprachen.

Außerdem sei auf das Web-Angebot zum Buch hingewiesen[2]. Es soll vor allem Quelltexte von Programmen beinhalten und zusätzliche Informationen geben, die für die Leserinnen und Leser nützlich sein könnten.

Vielen Dank an Bernd Bethge, Heike Eisenberg, Dr. Lutz Kohl, Prof. Dr. Rolf Niedermeier und Gabriele Rosner, die hilfreiche Hinweise gaben. In Vorträgen an der Pädagogischen Hochschule Schwäbisch Gmünd und an den Universitäten in Jena, Dresden, Potsdam und Frankfurt am Main stellte ich Teile des Buches zur Diskussion. Ich danke allen, die sich dabei aktiv einbrachten. Dem LOG IN Verlag danke ich darüber hinaus für die gute Zusammenarbeit.

Jena und Erfurt, im Juni 2010 Michael Fothe

[2] http://www.log-in-verlag.de/kunterbunte-schulinformatik/

Jede Lösung eines Problems
ist ein neues Problem.
Johann Wolfgang von Goethe

Inhalt

Drei Dinge machen einen guten Meister:
Wissen, Können und Wollen.
Deutsches Sprichwort

1 Vier Grundsätze eines kompetenzorientierten Unterrichts

Der zentrale Begriff dieses Buches ist »Kompetenz«[3]. In der »Klieme-Expertise«, der grundlegenden Arbeit zu Bildungsstandards an deutschen Schulen[4], wird der Begriff geklärt (BMBF, 2003, S. 72 f.):

> Wir verstehen unter Kompetenzen die bei Individuen verfügbaren oder von ihnen erlernbaren kognitiven Fähigkeiten und Fertigkeiten, bestimmte Probleme zu lösen, sowie die damit verbundenen motivationalen, volitionalen und sozialen Bereitschaften und Fähigkeiten, die Problemlösungen in variablen Situationen erfolgreich und verantwortungsvoll nutzen zu können. Kompetenz ist eine Disposition, die Personen befähigt, bestimmte Arten von Problemen erfolgreich zu lösen, also konkrete Anforderungssituationen eines bestimmten Typs zu bewältigen. Die individuelle Ausprägung der Kompetenz wird von verschiedenen Facetten bestimmt: Fähigkeit, Wissen, Verstehen, Können, Handeln, Erfahrung, Motivation.

In der Begriffsklärung werden motivationale, volitionale und soziale Bereitschaften und Fähigkeiten genannt. Es geht also auch darum, dass die Schülerinnen und Schüler motiviert sind, dass sie den Willen besitzen, sich anzustrengen und auch bis zum Ende durchzuhalten, und dass sie in der Lage sind, mit anderen zusammenzuarbeiten (siehe auch ISB, 2006).

Eine zweite Begriffsklärung zur Kompetenz soll angegeben werden, weil sie auf weitere Aspekte aufmerksam macht (Schott/Ghanbari, 2008, S. 30):

> Eine Kompetenz ist eine Fähigkeit. Sie wird beschrieben durch die Angabe einer bestimmten Menge von Aufgaben, die man ausführen kann, wenn man die betreffende Kompetenz besitzt; diese Aufgabenmenge kann Teilmengen verschiedener Aufgabenarten beinhalten; und von einem Kompetenzgrad oder, bei mehreren Teilmengen von Aufgaben, von mehreren Kompetenzgraden, die festlegen, wie gut man diese Aufgaben ausführen kann, wenn man die betreffende Kompetenz besitzt. Eine Kompetenz beschreibt eine Fähigkeit, die durch eine gewisse Nachhaltigkeit charakterisiert ist, d. h., sie sollte als Eigenschaft einer Person längere Zeiträume überdauern.

[3] Zum Begriff »informatische Kompetenz« siehe Friedrich, 2003.
[4] http://www.bmbf.de/pub/zur_entwicklung_nationaler_bildungsstandards.pdf

In der folgenden Tabelle 1.1 sind Merkmale guter Bildungsstandards angegeben. Diese wurden der »Klieme-Expertise« entnommen (BMBF, 2003, S. 24 f.).

Fachlichkeit	Bildungsstandards sind jeweils auf einen bestimmten Lernbereich bezogen und arbeiten die Grundprinzipien der Disziplin bzw. des Unterrichtsfachs klar heraus.
Fokussierung	Die Standards decken nicht die gesamte Breite des Lernbereiches bzw. Faches in allen Verästelungen ab, sondern konzentrieren sich auf einen Kernbereich.
Kumulativität	Bildungsstandards beziehen sich auf die Kompetenzen, die bis zu einem bestimmten Zeitpunkt im Verlauf der Lerngeschichte aufgebaut worden sind. Damit zielen sie auf kumulatives, systematisch vernetztes Lernen.
Verbindlichkeit für alle	Sie drücken die Mindestvoraussetzungen aus, die von allen Lernern erwartet werden. Diese Mindeststandards müssen schulformübergreifend für alle Schülerinnen und Schüler gelten.
Differenzierung	Die Standards legen aber nicht nur eine »Messlatte« an, sondern differenzieren zwischen Kompetenzstufen, die über und unter bzw. vor und nach dem Erreichen des Mindestniveaus liegen. Sie machen so Lernentwicklungen verstehbar und ermöglichen weitere Abstufungen und Profilbildungen, die ergänzende Anforderungen in einem Land, einer Schule, einer Schulform darstellen.
Verständlichkeit	Die Bildungsstandards sind klar, knapp und nachvollziehbar formuliert.
Realisierbarkeit	Die Anforderungen stellen eine Herausforderung für die Lernenden und die Lehrenden dar, sind aber mit realistischem Aufwand erreichbar.

Tabelle 1.1:
Merkmale guter Bildungsstandards.

In einem Unterricht, der auf Bildungsstandards beruht, besteht eine wichtige Aufgabe von Informatiklehrerinnen und -lehrern im Umsetzen der ergebnisorientierten Bildungsstandards in prozessorientierten Unterricht. Das Ziel kann ein schulinterner Lehrplan sein. Auf vier Grundsätze, die bei dieser Tätigkeit zu beachten sind, geht dieses Kapitel näher ein. Die Grundsätze werden jeweils charakterisiert und mit un-

terrichtspraktischen Beispielen untersetzt. Auf die besondere Bedeutung von Anwendungsbezug soll in diesem Zusammenhang hingewiesen werden (siehe Gallenbacher, 2007).

1. Grundsatz: Altersgemäßheit beachten

Bildungsstandards beziehen sich auf mehrere Klassenstufen. Bei der Konzeption der einzelnen Klassenstufen ist der Grundsatz der Altersgemäßheit zu beachten. Lehrerinnen und Lehrer müssen auch in der Lage sein, die Zuordnung der Kompetenzen und Inhalte zu Klassenstufen (z. B. gegenüber Eltern) zu begründen.

Kommunikationssysteme

Man überlege sich, was man Grundschulkindern antwortet, die einem bei der Arbeit am Computer zusehen und sinngemäß fragen, was eine E-Mail ist und wie das funktioniert. Was antwortet man Schülerinnen und Schülern der Sekundarstufe I oder der Sekundarstufe II? Die Behandlung eines Themas im Unterricht muss bekanntlich nicht aufgeschoben werden, bis eine endgültige abschließende Behandlung möglich erscheint, sondern kann bereits auf früheren Stufen in einfacher Form stattfinden. Die Erweiterbarkeit auf höherem Niveau ist dabei zu gewährleisten. Zu vermeiden sind Erklärungen, die ein späteres Umdenken erfordern (Baumann, 1996, S. 176). Von Bedeutung ist dabei eine didaktische Reduktion nach dem Motto: Vereinfachen, aber nicht falsch werden lassen.

Lehrpläne sind häufig nach dem Spiralprinzip aufgebaut. Dieses Konstruktionsprinzip ist auch bei schulinternen Lehrplänen von Bedeutung. So kann es z. B. in der Klassenstufe 8 heißen: Aufbau und Struktur des lokalen Rechnernetzes an der Schule. In der Klassenstufe 11 wird dies dann wieder aufgegriffen. Dann geht es jedoch um mehr: Topologien von Netzen, Beschreiben des Datenaustauschs durch ein einfaches Schichtenmodell und Kennenlernen einfacher Kommunikationsprotokolle.

Normalisierung von Datenbanken

Die Normalisierung hilft, Redundanzen und Anomalien in Datenbank-Entwürfen zu vermeiden. Im Informatik-Studium wird das Thema mit mathematischer Präzision betrieben. Im Informatikunterricht an Schulen könnten stattdessen Beispiele diskutiert werden – auch solche, die den Normalformen 1NF, 2NF oder 3NF nicht entsprechen. Die Datenbank-Entwürfe werden ggf. überarbeitet. Im Übrigen ist das Thema »Fahrradausleihe« für Schülerinnen und Schüler meistens interessanter als die »Verwaltung der Schulbibliothek« und weniger aufregend als das »Katalogisieren von Musik-CDs«.

Roulette

Casinos werben damit: Roulette ist das fairste Glücksspiel der Welt, denn immerhin rund 98 Prozent der Einsätze werden wieder ausgeschüttet[5]. Diese Aussage lässt sich sicher nachrechnen, aber tröstet sie auch den Verlierer? In Projektphasen der Sekundarstufe II können interessierte Schülerinnen und Schüler komplexe Roulette-Programme entwickeln. In der Sekundarstufe I kann eine Reduktion auf ein Programm erfolgen, in dem es nur um die einfachen Chancen geht, die 0 nicht beachtet wird und beim Spielen sogar Schulden gemacht werden dürfen. Bei den einfachen Chancen wird auf jeweils 18 Zahlen gesetzt: Rouge (alle »roten« Zahlen), Noir (alle »schwarzen« Zahlen), Pair (alle geraden Zahlen), Impair (alle ungeraden Zahlen), Manque (die Zahlen von 1 bis 18) und Passe (die Zahlen von 19 bis 36). Im folgenden Python-Programm macht der Benutzer immer wieder seinen Einsatz und setzt auf Rot oder Schwarz. Kommt die richtige Farbe, so schlägt das Programm den Einsatz dem aktuellen Besitz hinzu, ansonsten verliert er ihn. Von Bedeutung ist die Funktion zufall, die eine Pseudozufallszahl ermittelt, und zwar eine 0 oder 1.

```
import random

def zufall():
    return int(2*random.random())

besitz = 100
print "Dein Besitz zu Beginn: ", besitz

ende = "n"

while ende != "j":
    einsatz = input("Deinen Einsatz bitte: ")
    besitz = besitz - einsatz
    print "0 bedeutet Rot"
    print "1 bedeutet Schwarz"
```

[5] http://www.spielbank-wiesbaden.de/DE/275/FranzRoulette.php

```
    gesetzt = input("Gib 0 oder 1 ein: ")
    gefallen = zufall()
    if gefallen == 0:
        print "Es kam eine rote Zahl"
    else:
        print "Es kam eine schwarze Zahl"
    if gesetzt == gefallen:
        besitz = besitz + 2 * einsatz
    print "Dein aktueller Besitz: ", besitz
    ende = raw_input("Jetzt Schluss machen? Gib j oder n ein: ")

print "Dein Besitz am Ende: ", besitz
```

Die Schülerinnen und Schüler spielen fünf Minuten lang. Der Besitz beträgt zu Beginn einheitlich 100 Euro. Nach der Spielzeit nennen die Schülerinnen und Schüler ihren aktuellen Besitz, und es wird ausgewertet, wie viele Gewinner und Verlierer es gibt. Es wird ein Fazit gezogen: Über das Schicksal des Einzelnen lässt sich im Vorfeld nichts sagen. Das macht wohl den Reiz aus – und ist auch die Gefahr. Bei aller Fairness.

P=NP-Frage

Die berühmte Frage P = NP? kann auf unterschiedlichen Niveaus thematisiert werden. In Lehrveranstaltungen an Hochschulen werden die Komplexitätsklassen P und NP mithilfe von Turing-Maschinen beschrieben und ihre Relation zueinander diskutiert. Dies setzt ein erhebliches theoretisches Rüstzeug voraus. Man kann die Frage aber auch bereits in der Schule anhand eines konkreten Beispiels behandeln. Dafür eignet sich z. B. das Knotenüberdeckungsproblem, ein einfach verständliches Graph-Problem mit praktischer Relevanz (siehe Niedermeier u. a., 2007).

In der Version als Partyproblem lautet es: Nehmen wir an, Person X will eine harmonische Geburtstagsfeier gestalten. Einzuladende Kandidaten wären am liebsten n Personen. Nicht alle potenziellen Gäste harmonieren jedoch miteinander, sodass X das Ziel verfolgt, möglichst viele der n Personen so einzuladen, dass nur miteinander harmonierende Personen an der Feier teilnehmen.

Andere Versionen des Knotenüberdeckungsproblems sind ein Experimentproblem und ein Verkehrsüberwachungsproblem. Um es auf den Punkt zu bringen: Wenn es gelingen würde, für das Knotenüberdeckungsproblem einen Lösungsalgorithmus zu finden, der schnell ist (d. h., er besitzt einen polynomialen Zeitaufwand), so wüsste man von einem ganzen Universum an Problemen, dass es für sie solche schnellen Lösungsalgorithmen gibt. Bisher kennt man von den Problemen, um die es hier geht, nur Lösungsalgorithmen, die langsam sind (d. h., sie besitzen exponentiellen Zeitaufwand). Und zwischen polynomial und exponentiell liegt eine Welt. Um dies zu verstehen, müssen die Schülerinnen und

Schüler nur grundlegende Erfahrungen zum Zeitaufwand von Algorithmen besitzen. Eine wesentliche Idee im Kontext der P=NP-Frage ist die Reduzierbarkeit von Problemen: Problem A gilt als »mindestens so schwer« wie Problem B, wenn B in polynomialer Zeit auf A reduziert werden kann (siehe Abbildung 1.1). Auch diese Idee lässt sich an Beispielen klarmachen, so an der Überführung des Knotenüberdeckungsproblems in das Hitting-Set-Problem, einem NP-vollständigen Problem aus der Mengentheorie.

Abbildung 1.1:
Die Reduktion eines Problems auf ein anderes wird durch diese beiden Bilder illustriert. Auch der Scherenschnitt beinhaltet noch die wesentliche Information.

2. Grundsatz: Inhalte vernetzen

Beim Planen von Informatikunterricht sind, wo immer dies sinnvoll möglich ist, Querverbindungen innerhalb des Faches und zu anderen Fächern herzustellen. Dadurch ist dauerhafteres Wissen und Können der Schüler zu erwarten, man spart tendenziell Zeit, und nicht zuletzt erkennen die Schülerinnen und Schüler, dass es sinnvoll war, etwas ganz Bestimmtes bereits gelernt zu haben – schließlich wird es später wieder aufgegriffen. Das ist ein guter Beitrag zum Gewährleisten der Glaubwürdigkeit von Schule. Die nachfolgenden Beispiele stammen aus den Themen »Möglichkeiten und Grenzen«, »Formale Sprachen« und »Verschlüsseln«.

Möglichkeiten und Grenzen des Einsatzes von Informatiksystemen

Dieses Rahmenthema kann im Informatikunterricht unter verschiedenen Aspekten betrachtet werden (TKM, 1999, S. 22 f.). Bei manchen Fragen ist sicher eine Zusammenarbeit mit Lehrerinnen und Lehrern anderer Fächer sinnvoll (siehe Tabelle 1.2).

1. theoretischer Aspekt	Ist alles mit einem Computer berechenbar?
2. praktischer Aspekt	Ist jedes prinzipiell lösbare Problem in praktisch akzeptabler Zeit auf einem Computer bearbeitbar? Wie zuverlässig sind Informatiksysteme?
3. historischer Aspekt	Woher kommt die Informatik?
4. ökonomisch-sozialer Aspekt	Welche Wirkungen hat der Computereinsatz in der Arbeitswelt und im Freizeitbereich?
5. ethisch-philosophischer Aspekt	Kann eine Maschine denken?
6. datenschutzrechtlicher Aspekt	Darf der Einzelne durch den Einsatz von Computern zur anonymen Nummer werden?

Tabelle 1.2:
Aspekte und Fragen zum Rahmenthema »Möglichkeiten und Grenzen«.

Syntax von E-Mail- und WWW-Adressen

In den GI-Empfehlungen zu Bildungsstandards Informatik heißt es (GI, 2008, S. 34): »Schülerinnen und Schüler der Jahrgangsstufen 5 bis 7 überprüfen vorge-

gebene E-Mail- und WWW-Adressen auf Korrektheit und geben korrekte E-Mail- und WWW-Adressen an.« Die GI-Empfehlungen orientieren dabei für die Jahrgangsstufen 5 bis 7 auf verbale Beschreibungen und Visualisierungen. In höheren Jahrgangsstufen kann die Syntax dann mit der erweiterten Backus-Naur-Form (EBNF) oder mit Syntaxdiagrammen beschrieben werden. Die Syntax der E-Mail-Adressen wird nachfolgend mit der EBNF definiert. Zuerst erklären wir die Nichtterminalsymbole buchstabe, folge und zeichen:

```
buchstabe =  "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".
folge = buchstabe { buchstabe }.
zeichen = "-" | ".".
```

Die Syntax von E-Mail-Adressen lautet in vereinfachter Form[6]:

```
adresse = folge { zeichen folge } "@" folge { zeichen folge } "." folge.
```

Mit diesem Beispiel wird eine sinnvolle Verbindung zwischen der Theorie der formalen Sprachen und der Nutzung des Internets hergestellt. Die Erarbeitung einer Syntaxbeschreibung von WWW-Adressen (auch vereinfacht) auf der Basis von typischen Beispielen könnte sich anschließen.

Was Vigenère mit Cäsar zu tun hat

Die Themen »Buchstaben-Häufigkeit«, »Cäsar-Chiffre« und »Vigenère-Chiffre« kann man beziehungslos nebeneinander behandeln. Verbindungen lassen sich jedoch auch ganz bewusst herstellen. Solch ein Vorgehen wird nachfolgend genauer beschrieben (siehe auch Abbildung 1.2).

Im Informatikunterricht kann der Frage nachgegangen werden, welches der häufigste Buchstabe in deutschsprachigen Texten ist. Die Schülerinnen und Schüler entwickeln einen Algorithmus, der die Häufigkeit der einzelnen Buchstaben in einem Text auszählt und anschließend das Maximum der 26 Zahlen ermittelt. Der Text kann z. B. als Zeichenkette vorliegen. Das Ergebnis ist der Buchstabe e, sofern es sich um einen typischen Text handelt. Eine Variation der Aufgabenstellung ist das Ermitteln der häufigsten Buchstaben, wobei ein einfaches Sortierverfahren eingesetzt wird. Aus der Literatur ist z. B. die Reihenfolge e n i r s a t für die sieben häufigsten Buchstaben im Deutschen bekannt (Bauer, 1997, S. 279).

[6] Zur genauen Beschreibung siehe den Internetstandard RFC 5322: http://tools.ietf.org/html/rfc5322

Abbildung 1.2:
Julius Cäsar (100–44 v. Chr.) und Blaise de Vigenère (1523–1596).

Bei der Cäsar-Chiffre ist der Schlüssel die Zahl, die angibt, um wie viele Buchstaben das Geheimalphabet gegenüber dem Klaralphabet versetzt ist. Der Schlüssel 3 bedeutet: drei Buchstaben weiter. Beispielsweise wird der Klarbuchstabe r in den Geheimbuchstaben u überführt, das y wird in b überführt, der häufigste Buchstabe e wird zum h.

Im Informatikunterricht entwickeln die Schülerinnen und Schüler Funktionen zum Ver- und Entschlüsseln eines Zeichens und darauf aufbauend einer Zeichenkette. Eine Diskussion zur Sicherheit der Cäsar-Chiffre kann sich anschließen. Die Schülerinnen und Schüler erkennen, dass die Cäsar-Chiffre unsicher ist, da sich aus dem häufigsten Geheimbuchstaben der Schlüssel ermitteln lässt (sofern der zugrundeliegende Klartext ein längerer deutschsprachiger Text ist). Ist h der häufigste Geheimbuchstabe, so ist der Schlüssel 3. Aus i folgt 4, aus j folgt 5 usw. Das unbefugte Entschlüsseln von Geheimtexten lässt sich unter Anwendung des o. g. Algorithmus zur Ermittlung des häufigsten Buchstabens in einem Text automatisieren. Die folgende Python-Funktion ermittelt den Schlüssel:

```
def angreifen(geheimtext):
    liste = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    for zeichen in geheimtext:
        nummer = ord(zeichen) - ord("a")
        liste[nummer] = liste[nummer] + 1
    maximum = liste[0]
    stelle = 0
    for platz in range(1,26,1):
```

```
        if liste[platz] > maximum:
            maximum = liste[platz]
            stelle = platz
    differenz = stelle - (ord("e") - ord("a"))
    if differenz < 0:
        differenz = differenz + 26
    return chr(differenz + ord("a"))
```

Die Funktion angreifen gibt den Schlüssel nicht als Zahl, sondern als Buchstaben zurück. Beispielsweise wird statt des numerischen Schlüssels 3 der Schlüsselbuchstabe d geliefert (siehe Tabelle 1.3). Das Herausarbeiten dieser Gleichwertigkeit von Zahl und Buchstabe dient der Vernetzung von Cäsar- und Vigenère-Chiffre und ist daher von entscheidender Bedeutung. Ein freundlicher Hinweis sei ergänzt: Durch den Auftrag, die Funktion angreifen zu entwickeln, sollen sich Schülerinnen und Schüler nicht ermutigt fühlen, künftig Geheimnisse anderer auszuspähen. Aber eindrucksvoll ist es schon, wenn der Programmbenutzer einen Geheimtext eingibt und der Computer ohne Kenntnis des Schlüssels den Klartext liefert.

0	1	2	3	4	5	6	7	8	9	10	11	12
a	b	c	d	e	f	g	h	i	j	k	l	m

13	14	15	16	17	18	19	20	21	22	23	24	25
n	o	p	q	r	s	t	u	v	w	x	y	z

Tabelle 1.3:
Zusammenhang zwischen numerischem Schlüssel und Schlüsselbuchstaben.

Bei der Vigenère-Chiffre gibt man ein Schlüsselwort an, z. B. *duo*. Die Schlüssellänge ist in diesem Beispiel 3. Das Schlüsselwort lässt sich mithilfe der Tabelle 1.3 in die Schlüsselliste [3 , 20 , 14] überführen. Die Schlüsselliste wird beim Ver- und Entschlüsseln rotierend auf den Klar- bzw. Geheimtext angewandt. Der 1., 4., 7., … Klarbuchstabe wird wie bei der Cäsar-Chiffre verschlüsselt, und zwar mit dem Schlüssel 3. Der 2., 5., 8., … Klarbuchstabe wird mit dem Schlüssel 20 und der 3., 6., 9., … Klarbuchstabe mit dem Schlüssel 14 verschlüsselt. Beim Entschlüsseln wird entsprechend verfahren. Damit ist ein entscheidender Zusammenhang zwischen Vigenère- und Cäsar-Chiffre dargestellt. Funktio-

nen, die zum Ver- und Entschlüsseln eines Zeichens mittels Cäsar-Chiffre entwickelt wurden, können jetzt verwendet werden. Auch kann ein Angriff auf die Vigenère-Chiffre bei bekannter Schüssellänge n gut erklärt werden: Der Geheimtext wird in n Teiltexte zerlegt. Nehmen wir an, man weiß, dass die Schlüssellänge 3 ist. Das Bilden der drei Teiltexte verdeutlicht das folgende Schema:

```
Gesamter Text:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 ...
1. Teiltext:    1        4        7       10       13       16
2. Teiltext:       2        5        8       11       14       17
3. Teiltext:          3        6        9       12       15
```

Der Angriff auf die drei Teiltexte erfolgt so, wie es für die Cäsar-Chiffre beschrieben wurde. Für das Ermitteln der Schlüssellänge gibt es Verfahren; dafür soll auf die Literatur verwiesen werden (z. B. Beutelspacher, 2009). Im Übrigen erweist sich die Cäsar-Chiffre als Vigenère-Chiffre mit der Schlüssellänge 1. Wir erkennen, dass die Vernetzung zwischen Cäsar- und Vigenère-Chiffre sinnvoll ist und auf mehreren Ebenen erfolgen kann.

3. Grundsatz: Mit Unterschieden klug umgehen

Jeden Schüler und jede Schülerin ein Stück weiter bringen. Das setzt einen Unterricht voraus, von dem jeder auch etwas hat. Bekanntlich sind sowohl Unterrichtsphasen wichtig, in denen der Lehrer alle Schüler „frontal" erreicht, als auch solche, in denen die Schülerinnen und Schüler allein, zu zweit oder in Gruppen tätig sind. Für einen differenzierten Informatikunterricht benötigt man Materialien (Aufgaben u. a.), mit denen die Schülerinnen und Schüler arbeiten und die sie bearbeiten. Solche Materialien sollten verstärkt entwickelt, erprobt und veröffentlicht werden. Hieran besteht ein erheblicher Bedarf.

Binnendifferenzierung kann helfen, individuelle Defizite auszugleichen. Das kann mit dem Ziel geschehen, bei allen die Mindeststandards abzusichern (siehe Kapitel 2), um eine Basis für den weiteren Unterricht zu schaffen. Differenzierung kann aber auch die vorhandene Ungleichheit zwischen Schülern produktiv ausnutzen und gerade die Stärken fortentwickeln. Mehrstufige Kompetenzmodelle und dazugehörige Aufgabensammlungen können Grundlagen für differenziertes Arbeiten sein (siehe Kapitel 7). Gerade auch bei leistungsschwachen Schülerinnen und Schülern muss das Ziel ein sinnvolles Arbeitsergebnis sein, das korrekt ist, in sich stimmig und brauchbar für das Weiterlernen – wenn auch auf einem einfachen Niveau. Beispiele aus dem Anfangsunterricht und aus der Informationstechnik werden nachfolgend angegeben.

Tausch zweier Werte

In vielen Anwendungen müssen Variablen ihren Wert tauschen (z. B. beim Sortieren). Gegeben sind z. B. die Variablen x und y mit ihren Werten x = "Erik" und y = "Julia". Im Unterricht kann die Standard-Lösung anhand einer Wertbelegungstabelle erarbeitet werden. Die Hilfsvariable für den Ringtausch nennen wir hilf. Das relevante Programmstück lautet:

```
hilf := x
x := y
y := hilf
```

Die Standard-Lösung kann erstaunlicherweise auch von Papageien gefunden werden, wie ein Fernsehbeitrag eindrucksvoll zeigte[7]. Dabei waren zwei Gegenstände zu tauschen, was dem Papagei dadurch gelang, dass er einen der beiden Gegenstände zeitweise auf einem »Hilfsplatz« ablegte (siehe auch Abbildung 1.3).

Die Schülerinnen und Schüler können in individueller Arbeit die folgenden Programmstücke hinsichtlich ihrer Korrektheit untersuchen:

hilf := y x := y y := hilf	hilf := y y := x x := hilf	(x, y) := (y, x)	x := y y := x

Leistungsstarke Schülerinnen und Schüler können stattdessen die Aufgabe erhalten, nach einer Lösung zu suchen, bei der zwei Integer-Variablen ihren Wert tauschen sollen und bei der keine Hilfsvariable benötigt wird. Nachdem sich die Aufregung gelegt hat (»Das geht doch gar nicht!«), kann der folgende Impuls hilfreich sein: Aus der Kenntnis der Summe zweier Zahlen und einer der beiden Zahlen lässt sich die andere Zahl ermitteln. Beispiel: Ist 9 die Summe und 2 die eine, so ist 7 die andere Zahl. Das kann zum folgenden Algorithmus führen (Fothe, 2002):

```
x := x + y
y := x – y
x := x – y
```

[7] Kluge Vögel / Die Werkzeugmacher / WDR / 20. Oktober 2009

Abbildung 1.3:
Kluge Vögel.

Addition von Zeiten

Entwickeln Sie ein Programm, das Zeitangaben addiert.

Diese Aufgabe aus dem täglichen Leben ist durch ein einfaches Computerprogramm ohne Verzweigungen und Schleifen lösbar. Sie eignet sich für ein differenziertes Arbeiten. Der Leistungsumfang des Programms kann nämlich recht unterschiedlich sein. Vom Addieren (nur) von Sekunden bis hin zu Zeitangaben mit Sekunden, Minuten, Stunden und Tagen (evtl. auch Monaten und Jahren) ist vieles möglich. Vor allem ist das Problem des Übertrags zu bewältigen. Die Aufgabe ist auch kulturhistorisch interessant. Gerade die Astronomie hat sich in ihrer Geschichte mit Zeitangaben akribisch befasst. Ein Beispielprogramm in Python, das 2 h 40 min 50 s + 7 h 35 min 20 s ausrechnet, ist nachfolgend angegeben (Ergebnis: 10 h 16 min 10 s):

```
stunden1 = 2
minuten1 = 40
sekunden1 = 50
```

```
stunden2 = 7
minuten2 = 35
sekunden2 = 20

summe = sekunden1 + sekunden2
sekunden = summe % 60        # Rest bei der Division
uebertrag = summe / 60       # ganzzahliger Anteil bei der Division

summe = minuten1 + minuten2 + uebertrag
minuten = summe % 60
uebertrag = summe / 60

stunden = stunden1 + stunden2 + uebertrag
print stunden, minuten, sekunden
```

Vier Aufgaben mit steigendem Schwierigkeitsgrad:

- Erhöhen Sie die Flexibilität des gegebenen Computerprogramms. Wandeln Sie dazu die ersten sechs Wertzuweisungen in input-Anweisungen um.
- Erweitern Sie das vorgegebene Programm so, dass die Zeitangaben auch Tage enthalten dürfen.
- Erarbeiten Sie ein Programm, das eine beliebige Anzahl von Zeitangaben addiert.
- Erarbeiten Sie Programme, die zwei Zeitangaben voneinander subtrahieren und die Zeitangaben vervielfachen (z. B. dreimal 2 h 45 min).

Schaltnetze und Schaltwerke[8]

Im Informatikunterricht mit einer informationstechnischen Ausrichtung können Schaltnetze und Schaltwerke thematisiert werden (siehe Tabelle 1.4).

Das setzt voraus, dass die Schülerinnen und Schüler über die entsprechenden mathematischen Grundlagen verfügen (insbesondere zur Logik).

Die Arbeitsaufträge an die Schüler können sich entsprechend ihrem Leistungsvermögen in Umfang und Komplexität der Anforderungen sowie im Grad der Selbstständigkeit bei der Bearbeitung unterscheiden (siehe Tabelle 1.5). Leistungsstarke Schülerinnen und Schüler erweitern den Leistungsumfang ihrer Lösungen oder bearbeiten Zusatzaufgaben. Sie können auch als Experten eingesetzt werden, an die sich Mitschüler mit Fragen wenden können.

[8] Diesen Abschnitt erprobten Detlef Flock und Michael Fothe in den Jahren 2001 bis 2003.

Schaltnetze (kombinatorische Schaltungen)	*Schaltwerke (sequenzielle Schaltungen)*
Vergleicher	RS-Flipflop
Multiplexer und Demultiplexer	Speicherregister aus RS-Flipflops
Codeschloss	Getaktetes RS-Flipflop
Halb- und Volladdierer	RS-Master-Slave-Flipflop
	Zähler
	Schieberegister
	Ringzähler

Tabelle 1.4:
Übersicht zu Schaltnetzen und Schaltwerken.

Normale Anforderungen	*Erhöhte Anforderungen*
Aufbau einer Schaltung unter Verwendung beliebiger Gatter (NOT, AND, OR, NAND, NOR)	ausschließliche Verwendung von NAND-Gattern
Umsetzen eines vorgegebenen Logikplans für einen Volladdierer (4 Bit) mit einem Simulationsprogramm	eigenes Erarbeiten eines Logikplans
Umsetzen eines vorgegebenen Logikplans für ein Schieberegister	selbstständiges Erarbeiten eines Ringzählers als Zusatzaufgabe

Tabelle 1.5:
Beispiele für differenziertes Arbeiten.

4. Grundsatz: Methodische Vielfalt anstreben

Vielfalt ist in vielen Lebensbereichen sinnvoll, so auch beim Lehren und Lernen. Ganz allgemein soll dieser Abschnitt die Leserin und den Leser anregen, ihr vorhandenes methodisches Repertoire zu erweitern. Speziell geht es nachfolgend um Rollenspiele. Man ist mitunter geneigt, unserer Zeit, allem medialen Geplapper zum Trotz, eine gewisse Sprachlosigkeit zu bescheinigen. Rollenspiele können helfen, diese zu überwinden. Sie können Kommunikation über fachliche Themen anbahnen. Dazu gehört das gegenseitige Hinweisen auf Fehler, Ungenauigkeiten und Verbesserungsmöglichkeiten.

Mit Rollenspielen können Vorgänge/Algorithmen veranschaulicht werden. Die Mitwirkenden stellen dabei z. B. Elemente einer Liste dar, die zu sortieren ist, oder Bits aus einer Bitfolge, die komprimiert werden soll. Sie verkörpern ein Mobiltelefon, den Sender oder den Empfänger einer Nachricht bzw. Personen, die Nachrichten verschlüsseln, entschlüsseln oder signieren (Fothe, 2006). Sie bearbeiten einen Prozeduraufruf für einen bestimmten Parameter und bauen einen binären Baum auf (siehe S. 102 ff.). Die Regeln für das Rollenspiel können unterschiedlich strikt formuliert sein. Strikte Regeln haben ein determiniertes Handeln des Menschen zur Folge. Weniger strikte Regeln ermöglichen auch autonomes und nicht determiniertes Handeln, was ein Fortentwickeln der Regeln einschließt. Rollenspiele sollen möglichst einfach gestaltet werden. Dadurch können die Lernenden unmittelbar im Spiel mitagieren und werden dann sowohl zu Zuhörern und Betrachtern der Mitspieler als auch selbst zu einem Bestandteil des dargestellten Systems. Solche handlungsorientierte Gruppenarbeit kann gruppendynamische Lernprozesse und kooperatives Lernen initiieren. Rollenspiele gibt es mittlerweile in vielen Unterrichtsfächern – wenn auch mit unterschiedlichen Zielsetzungen. Dadurch sind die Schülerinnen und Schüler an diese Methode durchaus gewöhnt, was deren Einsatz im Informatikunterricht erleichtert (siehe auch Dißmann, 2003).

Als inhaltlicher Rahmen wird nachfolgend das Thema »Mobiltelefon« gewählt. Bei der Konzeption der Rollenspiele wurden als Kriterien herangezogen: Relevanz, Anwendbarkeit, Altersgemäßheit, Umsetzbarkeit in eine spielerische Form, Eigenaktivität der Kinder und Sichtbarkeit für die Zuschauer (Fothe, 2007). Zum Umsetzen von Rollenspielen in ein Computerprogramm siehe S. 104.

Sortieren durch Minimumsuche

Für das Erzeugen eines sortierten Telefonbuchs im Mobiltelefon wird ein Algorithmus zum Sortieren benötigt. Ein Beispiel ist das Sortieren durch Minimumsuche. Einige Schülerinnen und Schüler erhalten eine Karte mit einem Vornamen. Als Grundlage für das Herausfinden des jeweiligen Minimums ist es erforderlich, die Vornamen in eine Ordnung zu bringen. Dazu wird festgelegt, dass der Vorname a »kleiner« als der Vorname b ist, wenn er weiter vorn in einem Wörterbuch steht. Als »kleinster« Vorname wird in unserem Beispiel Erik ermittelt. Erik und Julia tauschen ihre Plätze. Abbildung 1.4 verdeutlicht den Verlauf der Abarbeitung des Algorithmus.

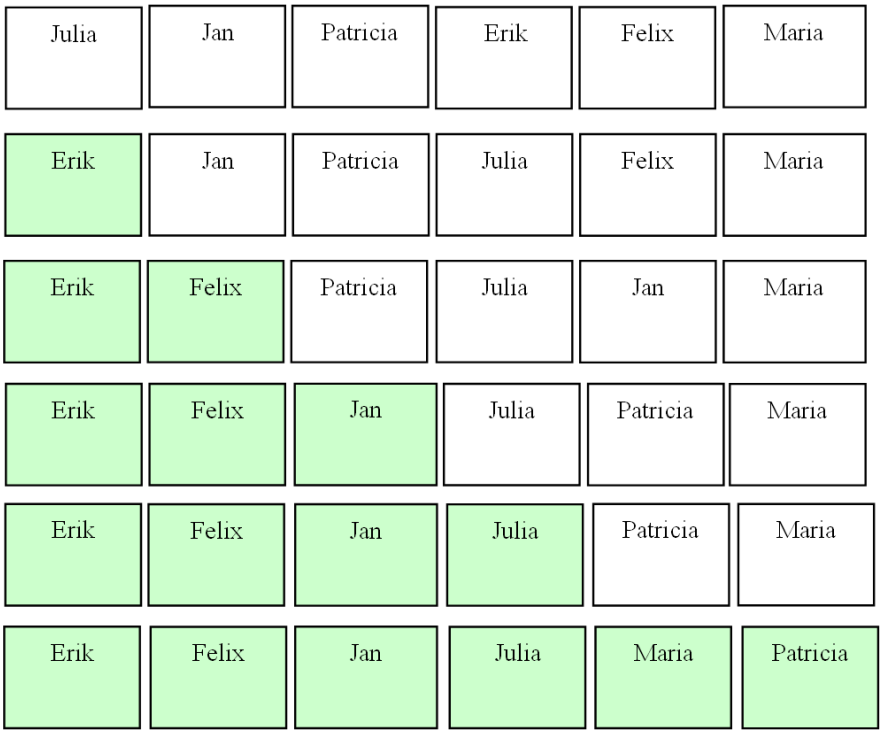


Abbildung 1.4:
Sortieren durch Minimumsuche (bei Mitwirkung von sechs Personen).
Die Schülerinnen und Schüler, die an die richtige Position gebracht wurden, sind farbig markiert.

Datenkompression

Dateien, die von einem Mobiltelefon versandt werden, bestehen aus Folgen von Nullen und Einsen. Ein Beispiel ist eine Bilddatei. Wie kann eine solche Datei möglichst schnell und eventuell auch kostengünstig übertragen werden? Als Lösung bietet es sich an, die Datei zu komprimieren (packen, verdichten). Den Schülerinnen und Schülern wird der folgende einfache Algorithmus zum Komprimieren einer Datei vorgestellt: Die Datei wird von vorn nach hinten durchlaufen. Fünf aufeinander folgende Nullen werden durch ein »N«, fünf aufeinander folgende Einsen durch ein »E« ersetzt. Beispielsweise wird aus der Bitfolge 00111111111110010000011 die Zeichenfolge 00EE1001N11.

Jeder beteiligte Schüler bekommt ein Blatt mit einer Null oder Eins in die Hand. Die Schüler stellen sich in einer Reihe auf und halten das Blatt vor sich.

Dann wird die Reihe von vorn nach hinten durchlaufen und es wird »komprimiert«: Fünf Schüler, die nebeneinander stehen und die gleiche Zahl in der Hand halten, treten aus der Reihe heraus. Ein Schüler mit einem Blatt mit einem »N« bzw. »E« tritt für sie in die Reihe hinein. Nach dem vollständigen Packen wird der Umkehrvorgang, das Entpacken, durchgeführt. Das Komprimieren erweist sich (glücklicherweise!) als verlustfrei.

Die Schüler erhalten anschließend die Aufgabe, sich so hinzustellen, dass ein Komprimieren nach dem vorgegebenen Algorithmus nicht möglich ist. Dazu nehmen sie, wie bereits zu Beginn, ein Blatt mit einer Null oder Eins in die Hand. Die Schüler sollen erkennen, dass höchstens vier Nullen bzw. vier Einsen benachbart sein dürfen.

Ergänzend kann der Hinweis gegeben werden, dass die Zeichen »N« und »E« bei der Programmierung durch Nullen und Einsen zu codieren wären. Es liegen insgesamt vier Zeichen vor. Zum Codieren eines Zeichens würden daher 2 Bit benötigt. Eine Null könnte z. B. durch 00, eine Eins durch 11, ein »N« durch 01 und ein »E« durch 10 codiert werden. Das schmälert den Gewinn beim Komprimieren. Der Algorithmus wird (etwas verändert) auf S. 51 f. in ein Computerprogramm umgesetzt.

Verschlüsseln

Wie können Daten zwischen einem Mobiltelefon und der Basisstation so übertragen werden, dass kein Unbefugter mithören kann? Die Daten müssen verschlüsselt werden. Das Ver- und Entschlüsseln kann mithilfe der XOR-Operation geschehen (siehe Abbildung 1.5). 0 steht für FALSCH, 1 für WAHR. Für die Operation gilt: a XOR b = (a + b) MOD 2. Ein wesentlicher Zusammenhang ist (a XOR b) XOR b = a XOR (b XOR b) = a XOR 0 = a.

XOR	0	1
0	0	1
1	1	0

Abbildung 1.5:
Wertetabelle für die XOR-Operation (entweder-oder).

Den Schülerinnen und Schülern wird erläutert, dass die in das Mobiltelefon gesprochenen Töne digitalisiert werden. Dabei entsteht eine Bitfolge. Die einzelnen Bits dieser Bitfolge werden im Mobiltelefon verschlüsselt. Es sei a ein Bit aus der Bitfolge der digitalisierten Töne. Es sei b ein Bit, das zum Verschlüsseln verwendet wird. Im Mobiltelefon wird zum Verschlüsseln c = a XOR b gerechnet. Das Bit c wird mit Funk übertragen. In der Basisstation wird das Bit b zum

Entschlüsseln verwendet. Dazu wird d = c XOR b gerechnet. Wegen des dargestellten wesentlichen Zusammenhangs gilt d = a, d. h., nach dem Ver- und Entschlüsseln erhält man wieder das Bit a. Ein Bösewicht, der das Bit c abhört, kann damit nichts anfangen, da er nicht über das Bit b zum Entschlüsseln verfügt. Im Mobiltelefon und in der Basisstation wird mithilfe eines geheimen Algorithmus die gleiche Bitfolge zum Ver- bzw. Entschlüsseln erzeugt.

Für das Rollenspiel werden grüne, blaue, gelbe und rote Karten benötigt. Auf den Karten steht jeweils eine Null oder eine Eins. Die grünen Karten stellen die Bitfolge dar, die verschlüsselt werden soll. Die blauen Karten repräsentieren die Bitfolge zum Ver- und Entschlüsseln. Der Satz blauer Karten liegt zweifach mit der genau gleichen Kartenreihenfolge vor. Gelbe Karten stellen das Ergebnis des Verschlüsselns, rote Karten das Ergebnis des Entschlüsselns dar. Das Verschlüsseln im Mobiltelefon verdeutlicht Abbildung 1.6. Die beiden Bits, die auf der vorn anstehenden grünen und auf der vorn anstehenden blauen Karte stehen (in unserem Beispiel 0 und 1), werden verknüpft. Die gelbe »Resultatkarte« (mit einer 1) wird zur Basisstation transportiert. Die bearbeitete grüne und blaue Karte werden zur Seite gelegt.

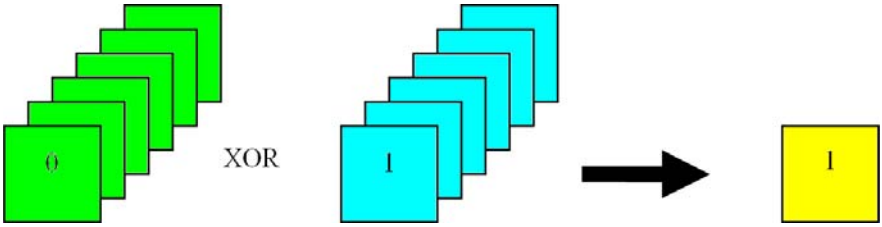


Abbildung 1.6:
Verschlüsseln im Mobiltelefon.

Das sich anschließende Entschlüsseln in der Basisstation zeigt Abbildung 1.7. Die beiden Bits, die auf der eben transportierten gelben Karte und auf der vorn anstehenden blauen Karte stehen (in unserem Beispiel 1 und 1), werden verknüpft. Das Ergebnis 0 wird durch eine rote »Resultatkarte« dargestellt. Zum Schluss wird die rote mit der grünen Karte vom Anfang verglichen. Wenn die Vorgänge für das Ver- und Entschlüsseln korrekt vorgenommen wurden, tragen beide Karten die gleiche Aufschrift. Am Ver- und Entschlüsseln wirken jeweils drei Schüler mit, die die Karten bereitstellen, die Rechnungen ausführen und die Karten transportieren.

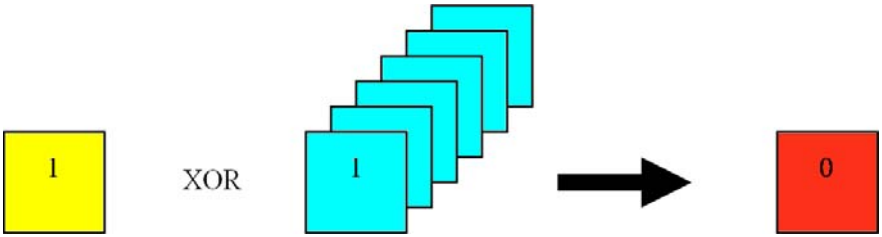


Abbildung 1.7:
Entschlüsseln in der Basisstation.

Lagebeziehung von Punkt und Vieleck

Die aktuelle Position eines Mobiltelefons sei bekannt. Es soll herausgefunden werden, ob sich das Mobiltelefon in einem bestimmten Gebiet befindet. Die Problemstellung wird genauer formuliert: Für ein gegebenes Vieleck und einen gegebenen Punkt soll ermittelt werden, ob der Punkt im Vieleck liegt. Das Vieleck darf konvex oder konkav sein, nicht jedoch überschlagen[9]. Als Material werden zwei Schnüre bereitgestellt (mit 10 m und 20 m Länge). Zuerst spannen die Schüler mit der langen Schnur ein Rechteck auf (siehe Abbildung 1.8). Zusätzlich repräsentiert ein Schüler das Mobiltelefon (in den Abbildungen als M dargestellt), und ein anderer Schüler stellt einen Punkt dar, der sich garantiert außerhalb des Vielecks befindet (in den Abbildungen als A dargestellt). Die Strecke MA wird durch die kurze Schnur gebildet. Diese Strecke darf nicht durch eine Ecke des Vielecks gehen. Um dies zu erreichen, muss sich A eventuell etwas bewegen. Als Kriterium wird herausgearbeitet: Eine ungerade Anzahl an Seiten des Vielecks, die von der Strecke MA geschnitten werden, bedeutet, dass M im Vieleck liegt. Andernfalls liegt M außerhalb des Vielecks. Die Vielecke können schrittweise immer komplexer werden, bis vielleicht am Ende eine zugegebenermaßen bizarre Figur entsteht, für deren Aufbau mehrere Schüler benötigt werden (siehe Abbildung 1.9).

[9] In einem Funktelefonnetz ist jede Funkzelle von sechs Zellen umgeben; aus diesem Grund stellt man eine Funkzelle in Form eines Sechsecks dar.

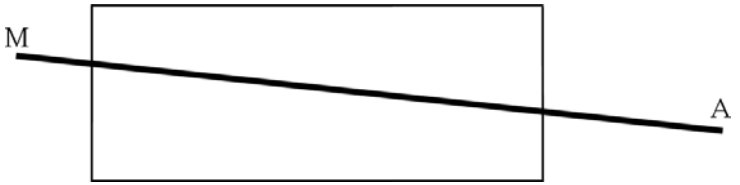


Abbildung 1.8:
Das Mobiltelefon M und der Punkt A befinden sich außerhalb des Rechtecks.

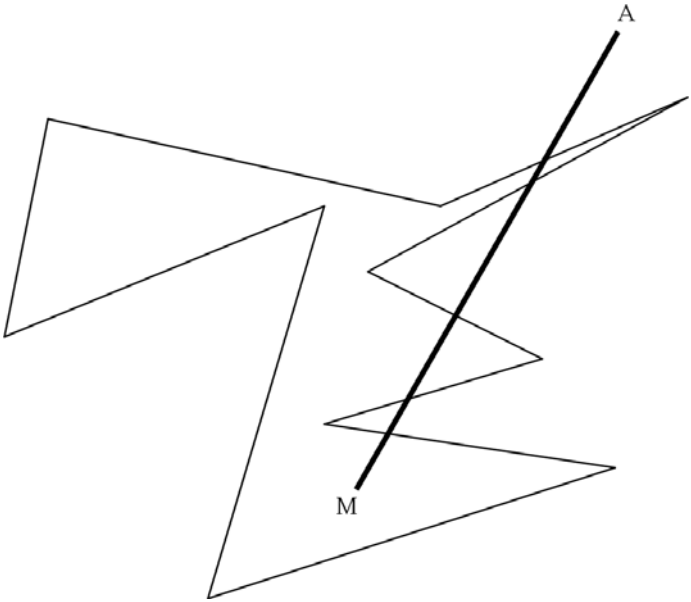


Abbildung 1.9:
Der Punkt A befindet sich außerhalb des Vielecks.
Befindet sich das Mobiltelefon M innerhalb oder außerhalb?

Informatikunterricht in der Sekundarstufe I

Wer das Ziel nicht kennt,
wird den Weg nicht finden.
Christian Morgenstern

2 Bildungsstandards Informatik für die Sekundarstufe I

Bei den GI-Empfehlungen zu Bildungsstands Informatik[10] handelt es sich um Mindeststandards (GI, 2008). Sie beschreiben Grundsätze eines guten Informatikunterrichts und geben an, über welche Kompetenzen jede Schülerin und jeder Schüler – und zwar unabhängig von der Schulform – auf dem Gebiet der informatischen Bildung am Ende der 7. und am Ende der 10. Jahrgangsstufe verfügen sollte. An den Schulen, an denen die GI-Empfehlungen umgesetzt sind, ist den Lehrerinnen und Lehrern aller Unterrichtsfächer klar, was sie bei ihren Schülern ab der 8. und ggf. ab der 11. Jahrgangsstufe auf dem Gebiet der informatischen Bildung voraussetzen können. Das kann dem Computereinsatz im Fachunterricht ein ganz neues Fundament geben. Absolventen allgemeinbildender Schulen würden über definierte Kompetenzen verfügen, was sicher auch für berufsbildende Schulen, Ausbildungsbetriebe und Hochschulen von Interesse ist. Sehen wir uns die GI-Empfehlungen etwas genauer an: In ihnen sind fünf Inhaltsbereiche (Themenfelder) und fünf Prozessbereiche (Arten des Arbeitens mit informatischen Inhalten) angegeben (siehe Tabelle 2.1).

Inhaltsbereiche (Inhaltsdimension[11])	Information und Daten Algorithmen Sprachen und Automaten Informatiksysteme Informatik, Mensch und Gesellschaft
Prozessbereiche (Handlungsdimension)	Modellieren und Implementieren Begründen und Bewerten Strukturieren und Vernetzen Kommunizieren und Kooperieren Darstellen und Interpretieren

Tabelle 2.1:
Inhalts- und Prozessbereiche der GI-Empfehlungen.

[10] http://www.informatikstandards.de/
[11] Eine dritte Dimension ist die Komplexitätsdimension. Sie betrifft Anspruch, Schwierigkeitsgrad und Neuigkeitsgehalt von zu bearbeitenden Aufgaben (siehe Baumann, 2008).

Die einzelnen Bereiche werden in den GI-Empfehlungen durch konkrete Kompetenzangaben untersetzt. Die Darstellung erfolgt auf drei Ebenen:
1. Gesamtsicht,
2. Bezugnahme auf die Jahrgangsstufen 5 bis 7 sowie 8 bis 10 und
3. Hinzufügung von Erläuterungen und Beispielaufgaben.

Die im Jahr 2008 veröffentlichten GI-Empfehlungen haben begonnen, Wirkung zu entfalten. Dies wird z. B. bei der Durchsicht des Tagungsbandes der GI-Fachtagung Informatik und Schule INFOS 2009 deutlich. Immerhin mehr als die Hälfte der Langfassungen nehmen Bezug auf die GI-Empfehlungen (Koerber, 2009). Ein anderes Beispiel: In Thüringen wurde das Medienkunde-Konzept überarbeitet[12]; zahlreiche Kompetenzen, die in den GI-Empfehlungen aufgeführt sind, finden sich in der neuen Konzeption wieder (TMBWK, 2009).

[12] http://www.medienkunde.de/

Vergesst nur die alten Meister nicht,
darauf wird aufgebaut.
Nikolaus Joachim Lehmann, 1. Juli 1990

3 Fallbeispiel: Linienrechnen

Unterricht wird in Themen gegliedert. Innerhalb eines Themas lösen die Schülerinnen und Schüler Aufgaben, sie bearbeiten Probleme. Dabei bauen sie Prozess- und Inhaltskompetenzen auf und wenden diese auch an. In diesem Kapitel erfolgt eine fachdidaktische Aufbereitung des Themas »Adam Ries und das Linienrechnen«, das für die Unterrichtsfächer Mathematik, Informatik und Geschichte in der Sekundarstufe I von Interesse sein kann (vgl. den 2. Grundsatz, S. 13 ff.). Im Informatikunterricht bietet das Thema Möglichkeiten zum Gewinnen von (Vor-)Erfahrungen zu Algorithmen, Nichtdeterminismus, Syntax, Semantik und zur Mechanisierung von Rechenvorgängen. Die Bedeutung des Rechnens auf den Linien im öffentlichen Leben ab dem 13. Jahrhundert kann herausgearbeitet werden.

Relevante informatische Kompetenzen mit Bezug zum Thema »Adam Ries und das Linienrechnen« sind (GI, 2008):

- Schülerinnen und Schüler der Jahrgangsstufen 5 bis 7 interpretieren Handlungsvorschriften korrekt und führen sie schrittweise aus.
- Schülerinnen und Schüler der Jahrgangsstufen 8 bis 10 überprüfen die wesentlichen Eigenschaften von Algorithmen.
- Schülerinnen und Schüler der Jahrgangsstufen 8 bis 10 unterscheiden die Begriffe »Syntax« und »Semantik« und erläutern sie an Beispielen.
- Schülerinnen und Schüler der Jahrgangsstufen 5 bis 7 erläutern das Prinzip der Eingabe, Verarbeitung und Ausgabe von Daten (EVA-Prinzip) als grundlegendes Arbeitsprinzip von Informatiksystemen.
- Schülerinnen und Schüler der Jahrgangsstufen 5 bis 7 bewerten Informationsdarstellungen hinsichtlich ihrer Eignung.
- Schülerinnen und Schüler der Jahrgangsstufen 8 bis 10 analysieren Sachverhalte und erarbeiten angemessene Modelle.
- Schülerinnen und Schüler der Jahrgangsstufen 5 bis 7 untersuchen bereits implementierte Systeme.
- Schülerinnen und Schüler der Jahrgangsstufen 8 bis 10 kennen und beachten grundlegende Aspekte des Urheberrechts.

Die Liste ist sicher noch erweiterbar. In dem Thema steckt also Potenzial auch für den Informatikunterricht (Fothe, 2009). Das Rechnen auf den Linien wird z. B. in den Museumsführern durch die Informatik-Ausstellung im Deutschen Museum München und durch das Heinz Nixdorf MuseumsForum (HNF) Paderborn erwähnt. In einem Plädoyer, die Geschichte der Informatik und der Informationsverarbeitung stärker in den Informatikunterricht zu integrieren, wird

Adam Ries bei den potenziellen Schlüsselstellen in der Geschichte der Informatik unter der Überschrift »Entstehung von Zahlen, Zahlensysteme und Rechenregeln« aufgeführt (siehe Thomas, 2005). Die Kompetenzen, die in einem dreistufigen Kompetenzmodell zum Thema »Algorithmen« angegeben sind, könnten teilweise anhand von Algorithmen zum Rechnen auf den Linien entwickelt werden (siehe Kapitel 7).

Linienrechnen im Grundsatz

Mithilfe eines Abakus oder mit Linienrechnen kann produktiv mit römischen Zahlen gerechnet werden. Das Linienrechnen erfolgt auf einem Rechenbrett, Rechentuch oder Rechentisch und mit Rechenpfennigen. Wir nehmen ein Rechenbrett mit vier Linien, vier Zwischenräumen und drei Feldern (siehe Abbildung 3.1). Möglich wären auch andere Anzahlen.

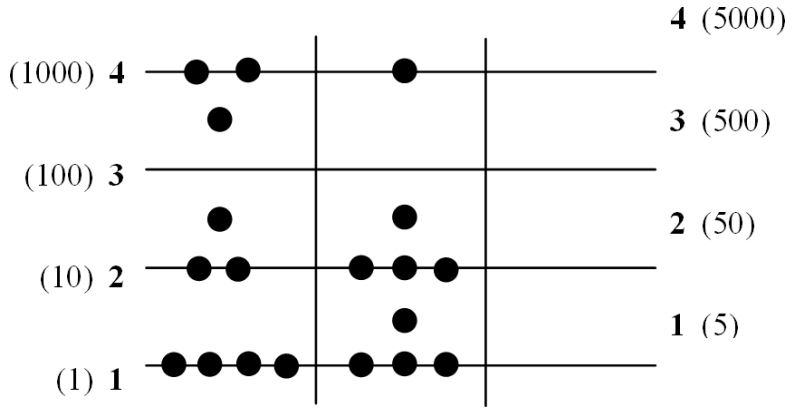


Abbildung 3.1:
Die Linien und Zwischenräume des Rechenbretts sind nummeriert.
In Klammern ist der jeweilige Wert eines Rechenpfennigs angegeben.
Im ersten Feld liegt die Zahl 2574, im zweiten Feld die Zahl 1088.
Im Ergebnisfeld liegt kein Rechenpfennig, was der Zahl 0 entspricht.

Die beiden Zahlen, die addiert, multipliziert, subtrahiert oder dividiert werden sollen, werden in das erste und zweite Feld gelegt. Beim Legen einer Zahl gibt deren Einerstelle an, wie viele Rechenpfennige auf die Linie 1 und in den Zwischenraum 1 zu legen sind. Die Zehnerstelle der Zahl beschreibt das Belegen der Linie 2 und des Zwischenraums 2, die Hunderterstelle das Belegen der Linie 3 und des Zwischenraums 3 usw. Das Ergebnisfeld ist zu Beginn leer. Die Rechenpfennige werden nach speziellen Regeln für jede Rechenoperation verschoben,

ersetzt, weggenommen oder hingelegt (siehe Deschauer, 1992). Von grundlegender Bedeutung für das Rechnen auf den Linien sind die Ersetzungsregeln, die das Höherlegen (Elevieren) und Tieferlegen (Resolvieren) von Rechenpfennigen beschreiben (siehe Tabelle 3.1). Höher- oder Tieferlegen verändern die Summe der Werte aller Rechenpfennige in einem Feld nicht. Das Rechnen ist beendet, wenn sich im Ergebnisfeld auf den Linien jeweils höchstens vier Rechenpfennige befinden und in den Zwischenräumen jeweils höchstens ein Rechenpfennig liegt. Überzählige Rechenpfennige sind höherzulegen (siehe Abbildung 3.2). Das Höherlegen verändert, wie bereits dargelegt, den Gesamtwert der Rechenpfennige im Ergebnisfeld nicht.

H1	Zwei Rechenpfennige, die in einem Zwischenraum liegen, können weggenommen werden. Dafür wird dann ein Rechenpfennig auf die nächsthöhere Linie gelegt.
H2	Fünf Rechenpfennige, die auf einer Linie liegen, können weggenommen werden. Dafür wird dann ein Rechenpfennig in den nächsthöheren Zwischenraum gelegt.
T1	Ein Rechenpfennig, der auf einer Linie liegt, kann weggenommen werden. Dafür werden dann zwei Rechenpfennige in den nächsttieferen Zwischenraum gelegt.
T2	Ein Rechenpfennig, der in einem Zwischenraum liegt, kann weggenommen werden. Dafür werden dann fünf Rechenpfennige auf die nächsttiefere Linie gelegt.
T3	Ein Rechenpfennig, der auf einer Linie liegt, kann weggenommen werden. Dafür werden dann ein Rechenpfennig in den nächsttieferen Zwischenraum und fünf Rechenpfennige auf die nächsttiefere Linie gelegt.

Tabelle 3.1:
Die fünf Ersetzungsregeln, nach denen Rechenpfennige höher- oder tiefergelegt werden können. Die Ersetzungsregel T3 ist speziell für das Subtrahieren vorgesehen.

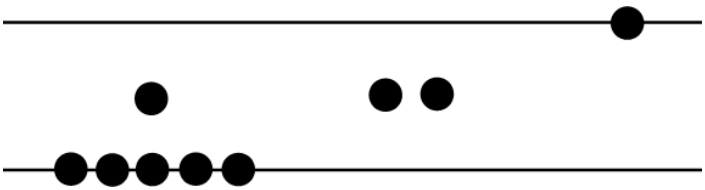


Abbildung 3.2:
Beispiel für das Höherlegen von Rechenpfennigen. Die Ersetzungsregeln H2 und H1 werden nacheinander angewandt. Beim Höherlegen der fünf Rechenpfennige entsteht ein primärer Übertrag, der einen sekundären Übertrag zur Folge hat.

Algorithmen zum Linienrechnen

Auch bei historischen Themen sollten die Schülerinnen und Schüler etwas aktiv tun. Beim Linienrechnen lernen sie *konkrete Algorithmen* kennen und können diese ohne Computer zur Abarbeitung bringen. Zumindest das Addieren und Subtrahieren auf den Linien sollten sie auch wirklich ausführen und die Rechenregeln nach Möglichkeit auch selbst entdecken und begründen[13]. Die Schülerinnen und Schüler können überprüfen, ob die Beschreibungen für das Rechnen auf den Linien die wesentlichen *Eigenschaften von Algorithmen* erfüllen (eindeutig, ausführbar, allgemein, endlich). Die Schülerinnen und Schüler können erkennen, dass an gewissen Stellen mehrere Möglichkeiten der Fortsetzung bestehen, von denen sie nach Belieben eine auswählen können, dass jedoch beim Berechnen einer ganz bestimmten Aufgabe stets das gleiche Ergebnis herauskommt – vorausgesetzt natürlich, sie verrechnen sich nicht. Beim Linienrechnen kommt es häufig nicht auf die Reihenfolge von Handlungen an. Das kann geradezu als ein Kennzeichen dieser Methode angesehen werden. Anhand des Linienrechnens können die Schülerinnen und Schüler somit Vorerfahrungen zum *Konzept des Nichtdeterminismus* gewinnen (Claus/Schwill, 2006, S. 456 f.).

Ein Beispiel ist das Höherlegen im Ergebnisfeld, das ständig erfolgen kann oder zwischendurch immer mal wieder oder nur ganz zum Schluss; der Bedarf an eingesetzten Rechenpfennigen kann dabei sehr unterschiedlich sein. Ein anderes Beispiel ist das Multiplizieren, das in unterschiedlichen Varianten durchgeführt werden kann. Es gibt Varianten, bei denen das Beherrschen des Einmaleins nicht erforderlich ist, und es gibt Varianten, bei denen man das Einmaleins zumindest teilweise beherrschen muss. Das Verwenden des Einmaleins macht das Multiplizieren häufig schneller, weil vergleichsweise wenige Rechenpfennige zu bewegen sind.

Das Rechnen auf den Linien lässt sich als *Musterverarbeitung* charakterisieren. Dabei wird das Linienrechnen aus der Perspektive der Informatik betrachtet und es wird eine Abstraktion vorgenommen, um das Wesentliche des Rechenverfahrens zu verdeutlichen (Fothe, 2004). Diese Variante zum Multiplizieren soll am Beispiel 125×481 erläutert werden (siehe Abbildung 3.3). Das linke Rechenbrett stellt die Ausgangssituation dar. Im rechten Rechenbrett ist die Situation nach dem Bearbeiten eines der beiden Rechenpfennige von Linie 2 dargestellt. Dieser Rechenpfennig ist weiß hervorgehoben und wird dann weggenommen. Die Anordnung vom zweiten Feld wurde vollständig in das Ergebnisfeld gelegt – und zwar um eine Linie nach oben verschoben. Das Multiplizieren wird also auf wiederholtes Addieren zurückgeführt. Die Rechenpfennige, die sich im ersten Feld auf Linien befinden, können in beliebiger Reihenfolge bearbeitet werden. Jedes Mal wird die Anordnung aus dem zweiten Feld in das Ergebnisfeld gelegt. Der Rechenpfennig im Zwischenraum 1 ist vor der Bearbeitung tieferzulegen.

[13] http://www.minet.uni-jena.de/preprints/fothe_09/Fothe-Linienrechnen.pdf

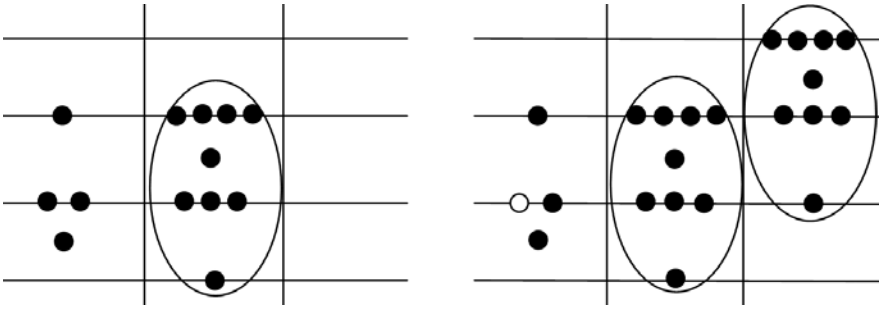


Abbildung 3.3:
Berechnung von 125 × 481.

Wenn beim Multiplizieren der gerade bearbeitete Rechenpfennig im ersten Feld weggenommen wurde (sozusagen ein Arbeitstakt abgeschlossen wurde), gilt

ErstesFeld × ZweitesFeld + ErgebnisFeld = Resultat.

Aus dieser Gleichung folgt, dass jetzt die Belegungen des ersten und zweiten Feldes getauscht werden dürfen, ohne dass das Resultat falsch wird (siehe Abbildung 3.4). Man kann sich fragen, ob sich diese Eigenschaft zur Erlangung von Rechenvorteilen ausnutzen lässt. Im Übrigen muss das Tauschen nicht »physisch« erfolgen; vielmehr kann das erste Feld (evtl. zeitweise) die Nummer 2 erhalten, das zweite Feld die Nummer 1.

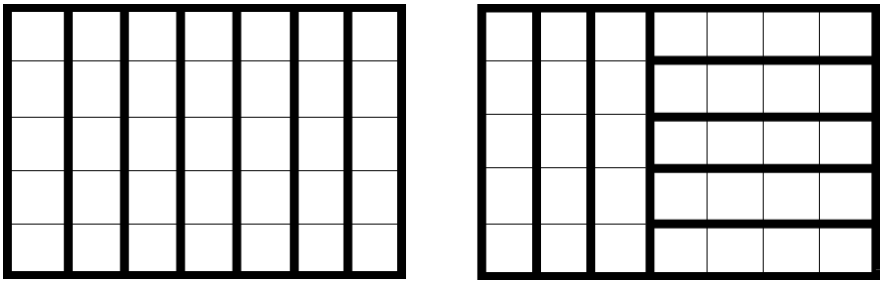


Abbildung 3.4:
Links ist für das Beispiel 7 × 5 das übliche Zurückführen der Multiplikation auf die Addition visualisiert. Rechts ist die Situation dargestellt, dass die Belegungen des ersten und zweiten Feldes mitten in der Berechnung (einmal) getauscht werden.
Es wird 3 × 5 + 5 × 4 gerechnet.

Zu Beginn des Addierens, Multiplizierens, Subtrahierens oder Dividierens müssen keine syntaktisch korrekten Ziffern vorliegen (siehe S. 39). Bei Aufgaben wie (36 + 81) × 4 ist es daher nicht erforderlich, nach der Berechnung von 36 + 81 überzählige Rechenpfennige höherzulegen. Man kann »einfach« weiterrechnen und mit 4 multiplizieren. Die *Robustheit* des Linienrechnens ist schon beträchtlich. Nachteilig beim Linienrechnen gegenüber dem Ziffernrechnen ist, dass Zwischenergebnisse nicht von Natur aus dokumentiert werden[14]. Diese Tatsache war sicher ein Hauptgrund dafür, dass das Linienrechnen durch das uns bekannte Ziffernrechnen abgelöst wurde. Im Mittelalter wurden Rechnungen auf den Linien, um Rechenfehler und Betrug weitgehend auszuschließen, mitunter von zwei Personen gleichzeitig ausgeführt. Dieses Vorgehen zum Erreichen der *Korrektheit* gab es auch in der Frühphase der Computertechnik, und zwar bei der OPREMA, die 1954/55 im Zeisswerk Jena entwickelt wurde[15]. Die OPREMA war als Zwillingsmaschine konzipiert, die aus zwei baugleichen Anlagen besteht. Alle Rechenschritte sollten redundant in beiden Anlagen ausgeführt werden. Nachdem sich aber gezeigt hatte, dass beide Anlagen zuverlässig arbeiten, wurde die ursprüngliche Idee fallengelassen und man hatte zwei Computer zur Verfügung.

Anhand des Linienrechnens können Vorerfahrungen zur Problematik des *Zeitaufwands von Algorithmen* gewonnen werden. Beim Linienrechnen kann man Umwege gehen, die zwar zu mehr Aufwand, nicht jedoch zu Rechenfehlern führen. Ein Beispiel ist das Dividieren, bei dem Rechenpfennige im ersten Feld unnötig tief gelegt werden können; dafür sind dann später im Ergebnisfeld Rechenpfennige höherzulegen. Wie lässt sich der Aufwand bestimmen? Eine Möglichkeit ist das Ermitteln der Anzahl an Schritten, die auszuführen sind. Beim Dividieren auf den Linien können die folgenden Schritte betrachtet werden:

- Tieferlegen im ersten Feld,
- Wegnehmen im ersten Feld/Legen im Ergebnisfeld und
- Höherlegen im Ergebnisfeld.

Die Rechenregeln sind arithmetisch begründet. Diese Tatsache ist beim Ausführen jedoch nebensächlich. Ein gutes Resümee zu den Algorithmen gibt Menninger: »Ich empfehle dem Leser, unsere Aufgabe wirklich auf den Linien durchzuführen; er wird erstaunt sein über die Anschaulichkeit des Verfahrens, das ohne eigentliches Rechnen vor sich geht« (Menninger, 1979, S. 165).

Syntax und Semantik

Das Höherlegen im Ergebnisfeld führt zu syntaktisch korrekten Ziffern, sodass das unmittelbare Ablesen des Resultats möglich wird (siehe Abbildung 3.5).

[14] Beim Rechnen mit dem Taschenrechner ist dies auch so.
[15] http://idw-online.de/pages/de/news284008

Anhand des Linienrechnens können die Begriffe »Syntax« (»Wie ist etwas korrekt aufzuschreiben?«) und »Semantik« (»Was bedeutet das Aufgeschriebene?«) im Informatikunterricht thematisiert werden. Dabei werden grafische Objekte und nicht wie meistens Zeichen betrachtet.

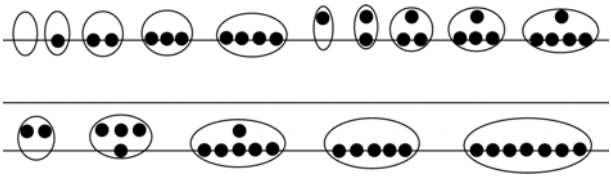


Abbildung 3.5:
Die obere Zeile enthält die syntaktisch korrekten Ziffern.
In der unteren Zeile liegen Syntaxfehler vor.

Die Ersetzungsregeln können als grafikorientierte Produktionsregeln einer formalen Sprache formuliert werden (siehe Abbildung 3.6).

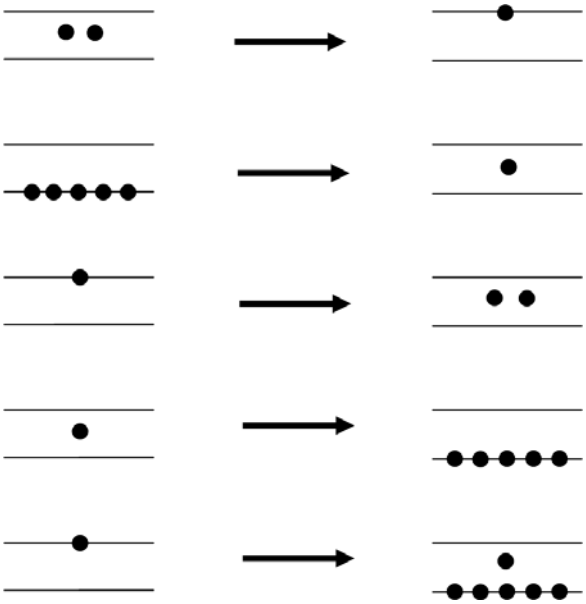


Abbildung 3.6:
Formale Darstellung der Ersetzungsregeln (siehe Tabelle 3.1).

Das EVA-Prinzip und das Linienrechnen

Den Schülerinnen und Schülern kann auch am Linienrechnen deutlich werden, dass das EVA-Prinzip ein wirklich grundlegendes Prinzip ist. Beim Rechnen auf den Linien werden indisch-arabische oder ursprüngliche römische Zahlen ein- und ausgegeben. Beispiele für solche römischen Zahlen[16] sind MMDLXXIIII und MLXXXVIII von Abbildung 3.1. Die Verarbeitung geschieht durch das Operieren mit Rechenpfennigen. Das Rechnen auf den Linien kann als Vorstufe zu mechanischen Rechenmaschinen betrachtet werden. Wenn Schülerinnen und Schüler auf den Linien rechnen, simulieren sie in gewisser Weise eine mechanische Rechenmaschine. Die folgenden Erkenntnisse können von den Schülerinnen und Schülern gewonnen werden:

- Das Multiplizieren auf den Linien wird auf das Addieren, das Dividieren auf das Subtrahieren zurückgeführt. Das ist auch bei mechanischen Rechenmaschinen so.
- Bei mechanischen Rechenmaschinen gibt es in der Regel den Zehnerübertrag. Einen direkten Zehnerübertrag gibt es beim Rechnen auf den Linien nicht. Stattdessen liegen Fünfer- und Zweierüberträge vor.
- Bei mechanischen Rechenmaschinen werden Überträge »von unten nach oben« – also ausgehend von der Einerstelle – bearbeitet. Beim Linienrechnen kann man das so machen, man muss es aber nicht.
- Primäre und sekundäre Überträge werden beim Linienrechnen gleichartig behandelt. Das trifft z. B. auch auf die Leibniz-Rechenmaschine zu (siehe Abbildung 3.7).

Abbildung 3.7:
Rechenmaschine von Gottfried Wilhelm Leibniz (1646–1716).
Es handelt sich um die erste Vier-Spezies-Rechenmaschine.

[16] Bei den ursprünglichen römischen Zahlen gab es keine verkürzenden Schreibweisen wie beispielsweise IV statt IIII. Diese bürgerten sich erst zu Beginn des 16. Jahrhunderts ein.

Objektorientierung als Weltsicht

Bereits in der Sekundarstufe I kann das Modellieren von Rechenbrett und Rechenpfennig aus objektorientierter Perspektive erfolgen (siehe Hubwieser, 2007, S. 209–223). Zwei Klassen mit relevanten Attributen und Methoden lassen sich angeben. Polymorphie kann erkannt werden: Ein Rechenpfennig reagiert je nach seiner Position – also dem Kontext, in dem er auftritt – unterschiedlich. Mögliche Positionen sind Linie und Zwischenraum. Kommunikation zwischen Rechenpfennigen kann beschrieben werden: Fünf Rechenpfennige auf einer Linie können sich wegnehmen, und ein Rechenpfennig legt sich in den nächsthöheren Zwischenraum. Zwei Rechenpfennige in einem Zwischenraum können sich wegnehmen, und ein Rechenpfennig legt sich auf die nächsthöhere Linie.

Modellieren und Programmieren

Computerprogramme zum grafischen Darstellen von syntaktisch korrekten Ziffern, zum Legen einer Zahl auf die Linien und Zwischenräume und zum Höher- oder Tieferlegen von Rechenpfennigen können von den Schülerinnen und Schülern analysiert bzw. entworfen und implementiert werden. Das Rechnen auf den Linien wurde in Spielform beschrieben (Fothe, 2001). Zu den Einpersonenspielen wurde ein Computerprogramm entwickelt. Der Spieler teilt dem Computer die Arbeitsschritte für das Addieren, Multiplizieren, Subtrahieren und Dividieren mit. Ein Beispiel für einen Arbeitsschritt, der beim Multiplizieren auszuführen ist, ist die Angabe, welcher Rechenpfennig im ersten Feld als nächstes bearbeitet werden soll. Der Computer führt den Arbeitsschritt aus und stellt das jeweilige (Zwischen-)Resultat grafisch dar. Das Computerprogramm lässt Regelverstöße durch den Spieler (den Rechnenden) nicht zu. Dadurch kann er sich auf das Umsetzen einer Spielstrategie (Rechenstrategie) konzentrieren. Wesentliches Ziel der Strategie ist das schnelle Gewinnen des Resultats.

Im Informatikunterricht der Sekundarstufe II können die Schülerinnen und Schüler vorgegebene Computerprogramme zum Linienrechnen analysieren, um die zugrunde liegenden Algorithmen zu verstehen. Anschließend erhöhen sie die Benutzungsfreundlichkeit der Programme, verbessern die Bildschirmgestaltung oder erweitern den Leistungsumfang. Möglich wäre auch ein Umsetzen der vorgegebenen Programme in andere Programmiersprachen. Werkzeuge wie Tabellenkalkulationsprogramme könnten in dem Zusammenhang auf ihre Verwendbarkeit hin überprüft werden. In Projektarbeiten werden die vollständigen Regeln für das Linienrechnen mit dem Ziel der Visualisierung der Rechenvorgänge in Computerprogramme überführt[17].

[17] http://users.minet.uni-jena.de/~infotest/ries/applet/AdamRies-start.html

Adam Ries und das Urheberrecht

Adam Ries schrieb drei Rechenbücher (siehe Abbildung 3.8). Im Jahr 1550 erschien sein drittes Rechenbuch in Leipzig. Es gilt als die beste deutsche Arithmetik in der Mitte des 16. Jahrhunderts. Das Manuskript war schon seit den 1520er-Jahren fertig, jedoch behinderten hohe Druckkosten die Veröffentlichung. Für das dritte Rechenbuch erhielt er auf Antrag ein zeitlich begrenztes Privileg des Kaisers Karl V., um den damals üblichen Raubdrucken zu begegnen. Durch dieses Privileg, vermerkt im Rechenbuch, war es Dritten für einige Jahre untersagt, das Buch selbst herauszugeben. Den Schülerinnen und Schülern kann an diesem Beispiel deutlich werden, dass das Urheberrecht und dessen Sicherung bereits vor Jahrhunderten bedeutsam waren. Die Diskussionen können zum Ausgangspunkt für das Thematisieren des Schutzes kreativer Leistungen in der Gegenwart werden. Für weitere Informationen zu Adam Ries wird auf die Literatur verwiesen (z. B. Rochhaus, 2008; Wußing, 2009).

Abbildung 3.8:
Diese Texttafel ist Bestandteil des dreiteiligen Adam-Ries-Denkmals in Erfurt (Michaelisstraße 48).

Dem Menschen überlegen
Leibniz' Wahlspruch für seine Rechenmaschine

4 Fallbeispiel: Informatiksystem Taschenrechner

Aus dem Alltag weiß man, dass es sinnvoll ist, Aufbau und Arbeitsweise von Werkzeugen zumindest im Prinzip zu kennen. Als Beispiel sei das Auto genannt. Folgerichtig sind Motor und Getriebe Gegenstand von Physik- oder Technikunterricht[18]*. In diesem Kapitel wird hinter die Kulissen des Informatiksystems Taschenrechner geschaut. Taschenrechner sind Werkzeuge, mit denen Schülerinnen und Schüler jahrelang zu tun haben. Im Informatikunterricht realisieren sie relevante Taschenrechner-Funktionalität mithilfe von Computerprogrammen selbst. Die black box wird dadurch zur grey box. Das Thema „Taschenrechner" eignet sich für differenziertes Arbeiten (vgl. den 3. Grundsatz, S. 17 ff.).*

Konkret wird in diesem Kapitel der Frage nachgegangen, wie Taschenrechner das Rechnen mit gemeinen Brüchen ermöglichen (siehe Abbildung 4.1), und es wird die Arbeitsweise von UPN-Taschenrechnern[19] genauer betrachtet.

Taschenrechner mit gemeinen Brüchen

Die Entscheidung, welcher Schüler oder welche Schülergruppe welche Variante bearbeitet, sollte nach ausführlicher Diskussion zu Beginn der Arbeiten getroffen werden. Im Mittelpunkt der Überlegungen können die beiden folgenden Fragen stehen:

- Werden Zähler und Nenner eines gemeinen Bruchs in Variablen einzeln verwaltet oder werden sie zu einer Datenstruktur zusammengefasst?
- Sind Operationen für zwei oder für mehr Brüche vorgesehen?

Am einfachsten ist die Variante, bei der genau zwei Brüche addiert, subtrahiert, multipliziert oder dividiert werden sollen und bei der Zähler und Nenner in Integer-Variablen gespeichert werden. Das Zusammenfassen von Zähler und Nenner zu einem Record oder Tupel kommt der mathematischen Notation näher; der Zugriff auf den Bruch wird dadurch jedoch komplizierter. Soll die Anzahl an Brüchen flexibel sein, so eignet sich z. B. eine Liste von Tupeln zum Speichern der Operanden. Diese Variante ist anspruchsvoll. Jede Variante kann in unterschiedlicher Qualität bearbeitet werden. Anregungen dafür bieten die folgenden Fragen:

[18] http://www.informatica-didactica.de/cmsmadesimple/index.php?page=Fothe2006
[19] umgekehrte polnische Notation

- Werden die Ergebnis-Brüche gekürzt?
- Werden Unterprogramme verwendet?
- Wird als Hauptnenner beim Addieren und Subtrahieren das Produkt der Nenner genommen (das ist einfach) oder deren kleinstes gemeinsames Vielfaches, wie in der Bruchrechnung üblich (das ist komplizierter)?
- Sind gemischte Zahlen zugelassen (z. B. 2 ½)?

Abbildung 4.1:
Beispiel für einen Taschenrechner, der das Rechnen mit gemeinen Brüchen ermöglicht.

Eine einfache Form der Addition, formuliert als Python-Programm:

```
z1 = 2                      # Zaehler des ersten Bruchs
n1 = 3                      # Nenner des ersten Bruchs
z2 = 7                      # Zaehler des zweiten Bruchs
n2 = 8                      # Nenner des zweiten Bruchs
ns = n1 * n2                # Nenner der Summe (Hauptnenner)
zs = z1 * n2 + z2 * n1      # Zaehler der Summe
print zs, ns                # Ausgabe der Summe
```

Zum Kürzen des Ergebnis-Bruchs wird der größte gemeinsame Teiler (ggT) zweier Integer-Zahlen a und b benötigt, der sich z. B. durch Wechselwegnahme berechnen lässt:

```
a = input("a=")
b = input("b=")
while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a
ggt = a
print ggt
```

Beim Kürzen werden Zähler und Nenner des Ergebnis-Bruchs durch den ggT von Zähler und Nenner geteilt. Aus dem ggT lässt sich das kleinste gemeinsame Vielfache (kgV) berechnen: `kgv = a * b / ggt`

Die Schülerinnen und Schüler können Computerprogramme in unterschiedlichen Qualitäten nachweisen (siehe Tabelle 4.1). Stets ist das Ziel ein funktionierendes Computerprogramm, das gemeine Brüche addiert, subtrahiert, multipliziert und dividiert, also ein sinnvolles Arbeitsergebnis.

- Addition, Subtraktion, Multiplikation und Division werden realisiert
- Operationen sind für zwei Brüche vorgesehen
- Zähler und Nenner werden in Variablen einzeln verwaltet
- Hauptnenner ist das Produkt der beiden Nenner

- Addition, Subtraktion, Multiplikation und Division werden realisiert
- Operationen sind für zwei Brüche vorgesehen
- Zähler und Nenner werden in einer Datenstruktur zusammengefasst
- Hauptnenner ist das Produkt der beiden Nenner
- Ergebnis-Brüche werden gekürzt; für die ggT-Berechnung wird ein Unterprogramm angegeben

- Addition, Subtraktion, Multiplikation und Division werden in Unterprogrammen realisiert
- Operationen sind für beliebig viele Brüche vorgesehen
- Zähler und Nenner werden in einer Datenstruktur zusammengefasst
- Hauptnenner ist das kgV der beiden Nenner
- Ergebnis-Brüche werden gekürzt; für die ggT-Berechnung wird ein Unterprogramm angegeben
- gemischte Zahlen sind zugelassen

Tabelle 4.1:
Drei Qualitäten von Computerprogrammen,
die das Rechnen mit gemeinen Brüchen realisieren.

Weitergehende Fragen befassen sich mit dem Design von Taschenrechnern. Die Schülerinnen und Schüler analysieren beispielsweise, wie gemeine Brüche in ihren Taschenrechner einzugeben sind und wie sie auf dem Display angezeigt werden. Dazu entwickeln sie dann auch eigene Ideen (z. B. zur Ein- und Ausgabe von gemischten Zahlen). Das im Kapitel 12 dargestellte Verfahren könnte auf das Rechnen mit gemeinen Brüchen erweitert werden. Damit wäre es möglich, Taschenrechner-Funktionalität einschließlich solcher Vorrangregeln wie »Punktrechnung kommt vor Strichrechnung« nachzubauen.

UPN-Taschenrechner

UPN-Taschenrechner kann man in den USA im Superstore kaufen. Im Internet gibt es Emulationen solcher Taschenrechner[20]. Wie funktionieren diese Rechenhilfsmittel eigentlich? Es wird die umgekehrte polnische Notation, auch Postfix-Notation genannt, verwendet (siehe S. 86 und Kapitel 12). Deren wichtigste Regel lautet: *Operatoren wirken auf zwei vorausgehende Operanden oder Zwischenergebnisse.* Nehmen wir den UPN-Ausdruck 75 2 * 10 35 + − 5 / als Beispiel. Wir durchlaufen ihn von links nach rechts und rechnen:

75 * 2 = 150, 10 + 35 = 45, 150 − 45 = 105, 105 / 5 = 21

In der üblichen Notation lautet der Ausdruck: (75 * 2 − (10 + 35)) / 5

Von Vorteil gegenüber den bei uns üblichen Taschenrechnern ist, dass es keine Vorrangregeln gibt. Klammern zum Durchbrechen von Vorrangregeln sind daher nicht erforderlich. Ein Operator wird sofort ausgeführt, wenn er beim Durchlaufen von links nach rechts anliegt. Eine Taste mit dem Gleichheitszeichen ist nicht vorhanden. Dafür gibt es eine ENTER-Taste, die bei der Eingabe zum

[20] http://epx.com.br/ctb/hp12c.php

Trennen zweier Zahlen verwendet wird. In unserem Beispiel müsste man also eintippen:

75 ENTER 2 * 10 ENTER 35 + − 5 /

Von Nachteil beim Einsatz eines UPN-Taschenrechners ist sicher die Notwendigkeit, die Ausdrücke vor dem Eintippen in die Postfix-Notation zu überführen (siehe Tabelle 4.2).

Übliche Notation	*UPN*
a * b + c	a b * c +
a + b * c	a b c * +
(a + b) * c	a b + c *
a * b − c * d	a b * c d * −
a * b * c * d	a b * c * d *

Tabelle 4.2:
Zu fünf Ausdrücken, die in der üblichen Notation gegeben sind, ist der gleichwertige UPN-Ausdruck angegeben.

Schauen wir uns nun die interne Arbeit eines UPN-Taschenrechners etwas genauer an. Die mir bekannten UPN-Taschenrechner verfügen über drei oder vier Register. Wir nehmen vier und nennen sie x, y, z und t. Die Register sind wie ein Stack angeordnet (siehe Abbildung 4.2). Ein Stack arbeitet nach dem LIFO-Prinzip (last in – first out). Das Element, das als letztes in den Stack eingebracht wurde, ist das erste, das aus dem Stack zu entnehmen ist. Eine andere wichtige Datenstruktur ist die Schlange; sie arbeitet nach dem FIFO-Prinzip (first in – first out). Elemente werden hinten angefügt und vorn weggenommen. Eine Anwendung ist die Suche in Graphen: Tiefensuche kann mit einem Stack, Breitensuche mit einer Schlange realisiert werden (Baumann, 1983, S. 132-152).

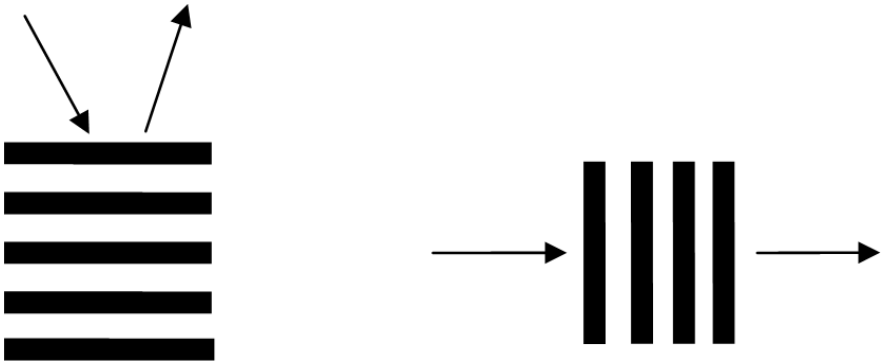


Abbildung 4.2:
Stack (Stapel, Keller) und Schlange (Warteschlange).

t	0	0	0	0	0	0	0	0	0	0	0	0
z	0	0	0	0	0	0	150	150	0	0	0	0
y	0	0	75	75	0	150	10	10	150	0	105	0
x	0	75	75	2	150	10	10	35	45	105	5	21
Eingaben:	CA	75	ET	2	*	10	ET	35	+	–	5	/

Tabelle 4.3:
Beispiel für die interne Arbeit eines UPN-Taschenrechners.
Die Taste CLEAR ALL (CA) führt dazu, dass anschließend alle Register den Inhalt 0 besitzen. ET steht für ENTER.

Zur Abarbeitung des obigen Beispiels siehe Tabelle 4.3. Die Berechnung wird erläutert:

- *Stack-Anweisung:* Das Betätigen der ENTER-Taste löst das Kopieren des Inhalts des x-Registers in das y-Register aus (das x-Register behält also seinen Inhalt). Der bisherige Inhalt des y-Registers wird in das z-Register verschoben, der Inhalt des z-Registers kommt ins t-Register. Der bisher im t-Register enthaltene Inhalt geht verloren.
- *Automatisches Stack-ab:* Das Betätigen einer Taste für einen zweistelligen Operator (zum Beispiel –) führt zum Entstapeln der Inhalte des x- und y-Registers, zum Verknüpfen der beiden Werte (y – x) und zum Stapeln des Ergebnisses. Die Zwischenschritte sind in der Tabelle nicht dargestellt. Der Inhalt des t-Registers wird in das z-Register kopiert.
- *Automatisches Stack-auf:* Wird nach einer Berechnung eine Zahl eingegeben, so wird das bei der Stack-Anweisung beschriebene Hochschieben automatisch ausgelöst.

Die Schülerinnen und Schüler können sich auch mit dem Design von UPN-Taschenrechnern befassen, so mit dem Realisieren einstelliger Operatoren (wie dem »Vorzeichen-Minus«) und mit Möglichkeiten, alle Registerinhalte zu inspizieren. Standardmäßig wird auf dem Display der Inhalt des x-Registers angezeigt (Schärf/Strecha, 1977, S. 18-32).

Wenn man keinen Durchblick hat,
ist man froh, wenn's weitergeht.
Klaus Brunnstein, 30. November 2001

5 Fallbeispiel: Suchen in Texten

Die Operationen Suchen und Ersetzen sind z. B. aus der Arbeit mit Textsystemen allgemein bekannt. Mit diesen Operationen kann man auch unabhängig von Anwendungssystemen etwas anfangen. In diesem Kapitel werden einfache Algorithmen zum Suchen und Ersetzen vorgestellt und in verschiedenen inhaltlichen Kontexten angewandt (vgl. den 2. Grundsatz, S. 13 ff.).

Die Python-Funktion suche1 erledigt das Suchen eines Musters in einem Text:

```
def suche1(text, muster):
    textlaenge = len(text)
    musterlaenge = len(muster)
    for index in range(0, textlaenge-musterlaenge+1, 1):
        if text [index : (index+musterlaenge)] == muster:
            return index
    return -1
```

Diese Funktion wird aufgerufen:

```
t = "ersagteabernichts"
m = "aber"
print suche1(t, m)

t = "ersagteaberaberabernichts"
m = "aber"
print suche1(t, m)
```

Das Muster wird so lange von links nach rechts unter dem Text verschoben, bis sich eine Übereinstimmung ergibt. Es wird ein Index zurückgegeben (Position des ersten Zeichens bei der ersten Übereinstimmung). Ist das Muster im Text nicht enthalten, wird der Wert −1 zurückgegeben. Die Idee ist einfach, die präzise Formulierung des Algorithmus, insbesondere der range-Anweisung und der Indizes, ist jedoch fehleranfällig. Das Programm sollte daher vorgegeben und mit den Schülerinnen und Schülern besprochen werden[21].

[21] Es gibt schnellere Algorithmen (z. B. das Verfahren von Boyer-Moore); diese sind jedoch vergleichsweise kompliziert (siehe z. B. Ottmann/Widmayer, 2002, S. 613–652).

Das Muster kann mehrfach im Text vorkommen. Die Schülerinnen und Schüler erhalten die Aufgabe, ein Programm zu erarbeiten, das alle Übereinstimmungen von Muster und Text ermittelt. Nachfolgend ist eine Musterlösung angegeben. Die Funktion suche2 liefert als Funktionswert ein Tupel mit allen Positionen:

```
def suche2(text, muster):
    textlaenge = len(text)
    musterlaenge = len(muster)
    loesung = ()
    for index in range(0, textlaenge-musterlaenge+1, 1):
        if text [index : (index+musterlaenge)] == muster:
            loesung = loesung + (index,)
    return loesung
```

Aufrufe der Funktion sind z. B.:

```
t = "ersagteaberaberabernichts"
m = "aber"
print suche2(t, m)

t = "ersagteabrnichts"
m = "aber"
print suche2(t, m)
```

Die Funktionen suche1 und suche2 werden am besten in das Modul SIT.pyw überführt und anschließend stets importiert.

Datenkompression

Der bereits im Kapitel 1 (S. 23 f.) beschriebene einfache Kompressionsalgorithmus wird von den Schülerinnen und Schülern (etwas verändert) auf dem PC umgesetzt: In einer Zeichenkette, die nur aus Nullen und Einsen besteht, werden zuerst jeweils fünf aufeinander folgende Nullen durch ein »N« ersetzt. Anschließend werden jeweils fünf aufeinander folgende Einsen durch ein »E« ersetzt. Das folgende Python-Programm erledigt das Packen; es kann im Unterrichtsgespräch erarbeitet werden:

```
import SIT

def packen(z):
    position = SIT.suche1(z,"00000")
```

```
    while position != -1:
        z = z[0:position] + "N" + z[(position+5):]
        position = SIT.suche1(z,"00000")

    position = SIT.suche1(z,"11111")
    while position != -1:
        z = z[0:position] + "E" + z[(position+5):]
        position = SIT.suche1(z,"11111")

    return z

zeichenkette= "00111111111110010000011"
print packen(zeichenkette)
```

Die folgenden Aufgaben besitzen steigenden Schwierigkeitsgrad:

- Erarbeiten einer Funktion zum Entpacken einer Zeichenkette,
- Erarbeiten einer Funktion, die das Suchen und Ersetzen ganz allgemein realisiert (Grundlage dafür kann das Programm zur Datenkompression sein; ein Aufruf der Funktion könnte lauten: `print sue(z, "11111", "E")` und
- Erarbeiten einer Funktion, die das kontextabhängige Suchen und Ersetzen ausführt (der Ersetzungsvorgang wird also nur dann ausgeführt, »wenn die Umgebung stimmt«).

DNS-Sequenzierung

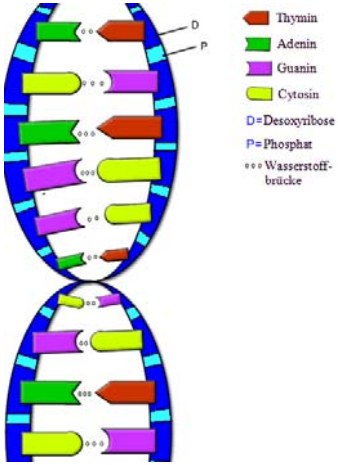


Abbildung 5.1:
Grundsätzliche Struktur der Desoxyribonukleinsäure (DNS).

Das Modell eines DNS-Moleküls (oder, je nach Sichtweise, eines Stranges des DNS-Moleküls) ist eine Zeichenkette, die aus den Buchstaben A, C, G und T zusammengesetzt ist[22] (siehe Abbildung 5.1).

Beispiel: "AACCTGATGGGTACTAAGTCGTAACCGTTGCATGTACGTTA"

DNS-Moleküle können durch Enzyme aufgebrochen werden. Ein Enzym-Molekül kann immer nur an einer für das Enzym charakteristischen Nukleotidfolge »andocken« (beispielsweise ATG). Die DNS wird dann an einer bestimmten Stelle innerhalb dieser Folge getrennt (etwa zwischen A und T). Dieses Beispiel-Enzym könnte durch die Zeichenkette "A|TG" beschrieben werden. Die vollständige Zerlegung der DNS-Zeichenkette durch dieses Enzym liefert die Teil-Zeichenketten:

"AACCTGA", "TGGGTACTAAGTCGTAACCGTTGCA" und
"TGTACGTTA".

Unter Nutzung der Funktion suche1 erarbeiten die Schüler ein Programm, das solche Zerlegungen durchführt. Ein Beispielprogramm ist nachfolgend angegeben:

```
import SIT

def sequenzieren(dns,enzym):

    enzym1 = ""
    for zeichen in enzym:
        if zeichen != "|":
            enzym1 = enzym1 + zeichen
    laenge = len(enzym1)

    position = SIT.suche1(dns,enzym1)
    while position != -1:
        dns = dns[0:position] + enzym + dns[(position+laenge):]
        position = SIT.suche1(dns,enzym1)

    return dns

dns = "AACCTGATGGGTACTAAGTCGTAACCGTTGCATGTACGTTA"
enzym = "A|TG"
print sequenzieren(dns,enzym)
```

[22] http://www.bwinf.de/uploads/media/download/192aufgaben.pdf

Was passiert bei extremen Eingaben? Was ist überhaupt biologisch relevant? Ein sicher fragwürdiges Beispiel:

```
dns = "AATTTTTTTTTTGG"
enzym = "T|TT"
```

Vieles spricht dafür, das Erarbeiten des Computerprogramms gemeinsam mit einem Biologielehrer in Angriff zu nehmen.

Anstößige E-Mail-Adressen?

Bei dem Versuch, Unschönes im Internet zurückzudrängen, kann man auch über das Ziel hinausschießen. In der Computer Zeitung Nr. 4 vom 21. Januar 2002 wurde berichtet, dass ein Internet-Provider anstößige Ausdrücke wie zum Beispiel »Arsch« oder »arsch« aus seinen E-Mail-Adressen verbannt. Die Schülerinnen und Schüler sollen ein Programm erarbeiten, das feststellt, ob eine E-Mail-Adresse zulässig ist. Der Funktionswert 1 bedeutet, dass die E-Mail-Adresse zulässig ist, eine 0, dass sie es nicht ist. Das System des Internet-Providers unterschied nicht zwischen Namen und Namensteilen. Diese Einschränkung soll auch für das von den Schülern zu erarbeitende Programm gelten. Pech für alle Arschaks, Barschels, Darscheids, Klarschinskys, Zarschitzkys, …

Ein Beispielprogramm

```
import SIT

def korrekt(adresse):
    liste = ["Arsch", "arsch"]
    for ausdruck in liste:
        if SIT.suche1(adresse, ausdruck) >= 0:
            return 0
    return 1

print korrekt("Barschel")
```

Im Übrigen wurde das System nach Protesten schnell vom Netz genommen. Man sollte sich genau überlegen, ob man die Aufgabe wirklich in seinem Unterricht bearbeiten lässt. Es kommt schon sehr auf die Schüler an.

Norbert Wiener meinte:
Information = Welt – Energie – Stoff
Hartmut Wedekind, 6. Mai 2004

6 Fallbeispiel: Optische Telegrafie

Lernen soll vom Leichten zum Schweren erfolgen (Comenius). Eine optische Telegrafenlinie stellt ein einfaches Kommunikationssystem dar. Das Internet ist erheblich komplizierter, selbst wenn man sich auf Wesentliches konzentriert. Dieses Kapitel thematisiert die optisch-mechanische Telegrafenlinie zwischen Berlin und der Rheinprovinz, die das Königreich Preußen von 1833 bis 1849 unterhielt, und es geht um zwei Fragen: Ist es sinnvoll, sich zuerst mit der optischen Telegrafie zu befassen und dann erst mit dem Internet? Kann man beim Auseinandersetzen mit der optischen Telegrafie etwas lernen, was auch heute noch brauchbar und anwendbar ist?

Abbildung 6.1:
Briefmarke von 1983 mit der St. Annenkirche in Berlin-Dahlem, die als Telegrafenstation diente (Station 2), und zwei Inspektoren, die die Telegrafenlinie beaufsichtigten[23].

Voll ausgebaut umfasste die Telegrafenlinie 62 Stationen (siehe Abbildungen 6.1 und 6.4). Sie wurde ausschließlich für staatliche und militärische Nachrichten genutzt. Vor 1833 wurden diese Nachrichten mit reitenden Boten transportiert; insofern stellte die optische Telegrafie einen bedeutenden Fortschritt dar. Die optische Telegrafie wurde schließlich von der elektromagnetischen Telegrafie abgelöst. Nachfolgend werden Fragen aufgeworfen und Antworten skizziert (Herbarth, 1978; Beyrer/Mathis, 1995).

[23] Eine weitere Briefmarke mit Bezug zum Thema: BRD Mi.-Nr. 471.
http://de.wikipedia.org/wiki/Briefmarken-Jahrgang_1965_der_Deutschen_Bundespost

Wie ist ein Zeichen aufgebaut? Ein Zeichen wird durch sechs Telegrafenflügel (drei Flügelpaare) dargestellt. Jeder Telegrafenflügel kann vier verschiedene Stellungen einnehmen (siehe Abbildung 6.2).

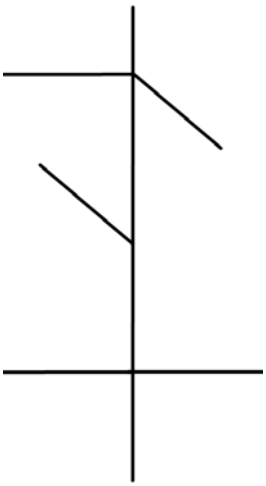


Abbildung 6.2:
Dieses Zeichen bedeutet »pünktlich«.

Wie lässt sich ein Zeichen beschreiben? Die historische Beschreibung der Zeichen ist nicht ganz einfach. Beispielsweise lautet sie für das Zeichen »pünktlich«: (A 5.2, B 6, C 5.3). Das soll hier nicht näher geklärt werden. Wir können stattdessen auch das Tripel (22, 10, 23) angeben. Die drei ganzen Zahlen beschreiben die Flügelpaare von unten nach oben (siehe Abbildung 6.3).

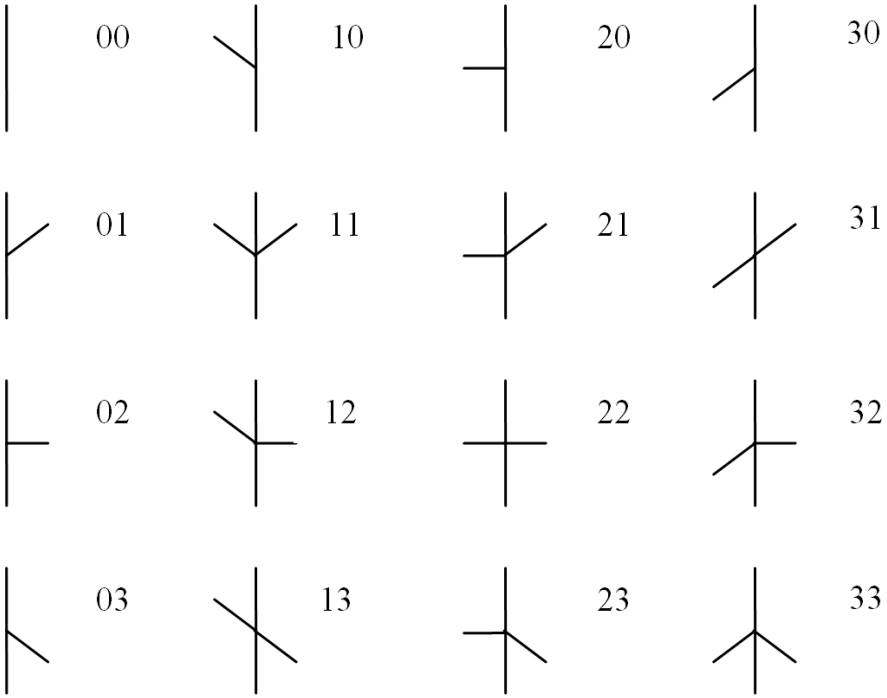


Abbildung 6.3:
Zur Beschreibung der Stellungen der Telegrafenflügel eines Flügelpaares kann eine zweistellige Zahl angegeben werden. Deren Zehnerstelle beschreibt den linken, die Einerstelle den rechten Telegrafenflügel.

Wie viele Zeichen gibt es? Jeder Telegrafenflügel kann vier Stellungen einnehmen. Bei sechs Flügeln sind 4^6 verschiedene Zeichen möglich, also 4.096. Unter Bezugnahme auf die Abbildung 6.3 kann man auch 16^3 rechnen.

Wie wird eine Nachricht codiert? Das Codieren erfolgt durch Umsetzen einer Depesche in Zeichen. Es gab Zeichen z. B. für die einzelnen Buchstaben, für Ortsnamen, für Telegrafenteile, für die Monate und für ganze Redesätze. Beim Codieren wurde also gleichzeitig komprimiert. Das Codieren erfolgte durch einen ausgesuchten Personenkreis, der im Besitz des Code-Buches war. Das Code-Buch war streng geheim (es existiert heute keine Ausfertigung mehr). Beim Codieren wurde also auch verschlüsselt. Die Telegrafisten in den Stationen kannten den Inhalt der Depeschen nicht, die sie weiterleiteten. Neben dem Code-Buch gab es das Telegrafistenwörterbuch (das ist heute noch verfügbar). Es enthält 2250 Zeichen und diente der dienstlichen Kommunikation zwischen den Stationen. Einer dienstlichen Depesche wurde ein spezielles Zeichen vorange-

stellt, sodass dem Empfänger klar war, dass die nachfolgende Depesche mit dem Telegrafistenwörterbuch (und nicht mit dem Code-Buch) codiert worden war.

Welchen Informationsgehalt besitzt ein Zeichen? Für das Codieren der vier Stellungen eines Telegrafenflügels benötigt man 2 Bit. Die Codewörter sind 00, 01, 10 und 11. Bei sechs Flügeln braucht man 6 × 2 Bit. Ein Zeichen entspricht also 12 Bit.

Wie groß war die Schrittgeschwindigkeit? Welches Datenvolumen wurde übertragen? Man kann von ungefähr 1 ½ Zeichen pro Minute ausgehen, also 90 Zeichen pro Stunde. Daraus folgt bei einer angenommenen täglichen Übertragungszeit von 10 Stunden[24]: 900 Zeichen pro Tag, 328.725 Zeichen pro Jahr (365,25 Tage). Als gesamtes Datenvolumen eines Jahres berechnen wir 3.944.700 Bit, was die Größenordnung 0,5 MByte besitzt. Da der Vorgang des Komprimierens vorgelagert war, wurde jedoch ein Mehrfaches an Information übertragen (z. B. wurde das gesamte Wort »pünktlich« auch nur mit einem Zeichen übertragen).

Wie lange war ein Zeichen von Berlin nach Koblenz unterwegs? Bei gutem Wetter benötigte ein Zeichen zum Durchlaufen der ganzen Strecke rund 7 ½ Minuten.

Wie wurden Übertragungsfehler festgestellt? Wie wurden diese korrigiert? Die Telegrafisten beobachteten mittels Fernrohr die Vorgängerstation, stellten das dort beobachtete Zeichen auf der eigenen Station ein und überprüften, ob die Nachfolgerstation das Zeichen auch korrekt einstellt. War das nicht der Fall, wurde eine dienstliche Depesche mit Korrekturangaben losgeschickt.

Welche Uhrzeit galt auf der Telegrafenlinie? Die Ortszeit – bezogen auf den Sonnenstand – unterscheidet sich zwischen Berlin und dem Rheinland fast um eine halbe Stunde, sodass es sich erforderlich machte, eine bestimmte Uhrzeit festzulegen. Die Berliner Zeit galt als Einheitszeit auf der gesamten Telegrafenlinie. Diese Vereinheitlichung war eine bedeutende kulturhistorische Leistung. Alle drei Tage erfolgte eine besonders vorbereitete Zeitsynchronisation. Ein bestimmtes Zeichen (das »Zeitzeichen«) durchlief die gesamte Strecke von Berlin nach Koblenz und zurück in zwei Minuten, sodass der maximale Fehler an den Stationen nur eine Minute betrug.

[24] Das ist eine grobe Annahme.

Abbildung 6.4:
Die historische Telegrafenlinie verläuft heute durch sechs Bundesländer.
Vor allem engagierte Amateurhistoriker setzen sich für den Erhalt von noch vorhandenen Gebäuden ein und machen auf die optische Telegrafie in ihrer Region aufmerksam.

Wie wurde eine Depesche von Paris nach Berlin transportiert? Es wird von einer Depesche berichtet, für deren Transport zuerst der französische Telegraf von Paris nach Metz genutzt wurde. Von dort aus ging es mit einer Eilstafette über Saarbrücken nach Koblenz weiter (also mit einem reitenden Boten, der regelmäßig die Pferde wechselte). Ab Koblenz wurde die Depesche auf der preußischen Telegrafenlinie nach Berlin weitertransportiert. Sie war insgesamt 30 Stunden unterwegs.

Nun zu unseren Fragen vom Beginn dieses Kapitels: Offensichtlich können anhand der optischen Telegrafie im Informatikunterricht Sachverhalte besprochen werden, die auch bei modernen Kommunikationssystemen relevant sind. Man denke an das Codieren, Komprimieren, Verschlüsseln, an das Erkennen und Korrigieren von Übertragungsfehlern, an das Nutzen von verschiedenen Kanälen beim Versenden einer E-Mail und an die Zeitsynchronisation von Rechnern über das Internet. Die Schülerinnen und Schüler können bereits in der Sekundarstufe I auf altersgemäßem Niveau für Fragestellungen sensibilisiert werden, die auch im Internet auftreten (vgl. den 1. Grundsatz, S. 9 ff.).

Die optische Telegrafie eignet sich als Gegenstand von Projektarbeiten. Die Schülerinnen und Schüler erhalten z. B. die Aufgabe, ein Handbuch für die Kommunikation zu erarbeiten. Wesentlicher Bestandteil des Handbuchs wäre die Beschreibung des Protokolls für die Kommunikation. Viele Fragen wären zu klären: Wie ist mit Eildepeschen umzugehen? Wie ist in einer Station zu verfahren, wenn sich zwei Depeschen begegnen – die eine kommt aus Berlin, die andere aus Koblenz? Wie ist vorzugehen, wenn eine normale Depesche oder sogar eine Eildepesche wegen Nebels nicht weitergeleitet werden kann? Wie erfolgt die Adressierung einer dienstlichen Depesche an eine andere Station? Das Umsetzen der Vorgänge könnte in Rollenspielen erfolgen.

> Ich bin auf Wort, Sprache und Bild im eigentlichsten Sinne angewiesen und völlig unfähig durch Zeichen und Zahlen, mit welchen sich höchst begabte Geister leicht verständigen, auf irgend eine Weise zu operieren.
>
> *Johann Wolfgang von Goethe*

7 Fallbeispiel: Algorithmen

Informatiklehrerinnen und -lehrer erhalten ein Arbeitsmaterial zum Thema »Algorithmen« als Grundlage für die Unterrichtsgestaltung in einer 8., 9. oder 10. Klasse im Umfang von ca. 40 Unterrichtsstunden. Das Arbeitsmaterial beinhaltet ein dreistufiges Kompetenzmodell, Beispielaufgaben zur Illustration der Anforderungen des Kompetenzmodells und eine umfangreiche Sammlung von Übungsaufgaben[25]*. Die Lehrerinnen und Lehrer legen selbst fest, welche Stufe des Kompetenzmodells welche Schülerinnen und Schüler wann im Unterricht erreichen sollen und welche Übungsaufgaben bearbeitet werden. Am Ende des Themas setzen die Informatiklehrerinnen und -lehrer einen Kompetenztest in ihrem Unterricht ein und werten ihn aus. Die bei der Auswertung gewonnenen Ergebnisse sollen den Lehrerinnen und Lehrern helfen, die Schülerleistungen einzuordnen. Mit dem Forschungsprojekt »Kompetenzorientierter Informatikunterricht in der Sekundarstufe I unter Verwendung der visuellen Programmiersprache Puck« wurde dieses Szenarium an der Universität Jena untersucht*[26]*. Wesentliche Arbeitsergebnisse sind in diesem Kapitel zusammengestellt.*

Mit Algorithmen wurde ein traditionelles Thema der Schulinformatik gewählt, mit dem Lehrer vertraut sind. Algorithmen besitzen in der Informatik einen besonderen Stellenwert, was z. B. daran deutlich wird, dass Algorithmisierung als eine von drei fundamentalen Masterideen charakterisiert wird (siehe Schubert/Schwill, 2004, S. 96; Vöcking u. a., 2008).

Kompetenzmodell zum Thema »Algorithmen«

Im Rahmen des Forschungsprojekts wurde ein dreistufiges Kompetenzmodell zum Thema »Algorithmen« entwickelt und veröffentlicht (Kohl/Fothe, 2007). In Stufe I besitzen die Schülerinnen und Schüler grundlegende Kompetenzen zu diesem Thema, in Stufe II geht es um vertiefte Kompetenzen und in Stufe III dann um umfassendere Kompetenzen. Der Begriff »umfassende Kompetenzen« wurde, weil für die Jahrgangsstufen 8 bis 10 wohl unrealistisch, vermieden (siehe Tabelle 7.1).

[25] Das Arbeitsmaterial ist im Internet verfügbar: http://www.minet.uni-jena.de/preprints/kohl_08/KohlArbeitsmaterialUntersuchung.pdf

[26] http://www.db-thueringen.de/servlets/DocumentServlet?id=13117

Stufen / Komponenten	Stufe I *Die Schülerinnen und Schüler haben grundlegende Kompetenzen zu Algorithmen.* *Die Schülerinnen und Schüler …*	Stufe II *Die Schülerinnen und Schüler haben vertiefte Kompetenzen zu Algorithmen.* *Die Schülerinnen und Schüler …*	Stufe III *Die Schülerinnen und Schüler haben umfassendere Kompetenzen zu Algorithmen.* *Die Schülerinnen und Schüler …*
A Eigenschaften von Algorithmen	▷erklären den Algorithmusbegriff und die wesentlichen Eigenschaften von Algorithmen ▷überprüfen die wesentlichen Eigenschaften von Algorithmen in einfachen Fällen ▷nennen Probleme, die mithilfe von Algorithmen lösbar bzw. nicht lösbar sind	▷erklären den Algorithmusbegriff und die wesentlichen Eigenschaften von Algorithmen an bekannten Beispielen ▷begründen anhand dieser Eigenschaften, ob gegebene Handlungsabläufe Algorithmen sind ▷nennen Probleme, die mithilfe von Algorithmen lösbar bzw. nicht lösbar sind	▷erklären den Algorithmusbegriff und die wesentlichen Eigenschaften von Algorithmen an selbst konstruierten Beispielen ▷begründen anhand dieser Eigenschaften, ob gegebene Handlungsabläufe Algorithmen sind ▷nennen Probleme, die mithilfe von Algorithmen lösbar bzw. nicht lösbar sind
B Algorithmische Grundbausteine und Datentypen	▷erklären die algorithmischen Grundbausteine wie Variablen, Wertzuweisungen, Verzweigungen und Wiederholungen und wenden diese Erklärungen an ▷stellen die algorithmischen Grundbausteine als Pseudocode dar ▷verwenden einen numerischen Datentyp	▷erklären die algorithmischen Grundbausteine wie Variablen, Wertzuweisungen, Verzweigungen und Wiederholungen und wenden diese Erklärungen an ▷stellen die algorithmischen Grundbausteine in verschiedenen Darstellungsformen dar ▷verwenden verschiedene Datentypen	▷erklären die algorithmischen Grundbausteine wie Variablen, Wertzuweisungen, Verzweigungen, Wiederholungen und Unterprogramme mit Parametern und wenden diese Erklärungen an ▷stellen die algorithmischen Grundbausteine in verschiedenen Darstellungsformen dar und wechseln zwischen Darstellungsformen ▷verwenden verschiedene Datentypen
C Arbeit mit Algorithmen	▷lesen in Pseudocode gegebene einfache Algorithmen ▷prüfen schrittweise einfache Algorithmen mit gegebenen Beispielen ▷setzen gegebene einfache Algorithmen in Programme um ▷modifizieren und ergänzen einfache Algorithmen bzw. Programme nach Vorgaben	▷analysieren die Funktionsweise und den Leistungsumfang gegebener Algorithmen ▷prüfen Algorithmen mit gegebenen Beispielen mithilfe von Durchlauftabellen (Schreibtischtest) ▷setzen gegebene Algorithmen in Programme um ▷modifizieren und ergänzen Algorithmen bzw. Programme nach Vorgaben	▷analysieren die Funktionsweise und den Leistungsumfang gegebener komplexer Algorithmen ▷prüfen Algorithmen mithilfe von Durchlauftabellen (Schreibtischtest) und wählen dazu typische und untypische Beispiele selbst aus ▷setzen gegebene komplexe Algorithmen in Programme um ▷modifizieren und ergänzen komplexe Algorithmen bzw. Programme nach Vorgaben und nach selbst gesetzten Zielen ▷korrigieren gegebene fehlerhafte Algorithmen bzw. Programme
D Programmentwicklung	▷entwerfen einfache Programme skizzenhaft ▷implementieren einfache Programme mit einem Programmiersystem ▷testen einfache Programme anhand gegebener Eingaben auf ihre Grundfunktionalität	▷fertigen einen schriftlichen Entwurf für Programme an ▷implementieren Programme mit einem Programmiersystem benutzungsfreundlich ▷testen Programme anhand gegebener Eingaben auf ihre Funktionalität ▷reflektieren über den Lösungsweg	▷fertigen einen schriftlichen Entwurf für komplexe Programme an ▷implementieren komplexe Programme mit einem Programmiersystem benutzungsfreundlich ▷testen komplexe Programme anhand selbst gewählter Eingaben auf ihre Funktionalität ▷reflektieren über den Lösungsweg sowie über Vor- und Nachteile der Lösung ▷verbessern Programme eigenständig

Tabelle 7.1:
Kompetenzmodell zum Thema »Algorithmen«.

Die Stufe I des Modells orientiert sich an den GI-Empfehlungen zu Bildungsstandards Informatik (Jahrgangsstufen 8 bis 10) und entspricht daher Mindeststandards (siehe Kapitel 2). Die anderen Stufen beschreiben Kompetenzen, die über Mindeststandards hinausgehen. Das Modell bezieht sich insbesondere

auf den Inhaltsbereich Algorithmen, jedoch bestehen auch Bezüge zu anderen Inhalts- sowie zu Prozessbereichen der GI-Empfehlungen. Typografische Festlegungen sollen die Lesbarkeit der Tabelle unterstützen. So sind in dem Modell die Kompetenzen der Stufe II, die bereits in der Stufe I genannt sind, und die Kompetenzen der Stufe III, die bereits in der Stufe II genannt sind, grau gedruckt. Kompetenzen, die in einer Stufe neu hinzukommen, und graduelle Unterschiede werden jeweils schwarz gedruckt.

Bei der Erarbeitung der Aufgaben wurden folgende Prinzipien beachtet:

- Ein Algorithmus, der nur eine Verzweigung oder eine Wiederholung enthält, wird im Allgemeinen der Stufe I zugeordnet.
- Ein Algorithmus, der mehrere, auch ineinander verschachtelte Verzweigungen und Wiederholungen enthält, wird im Allgemeinen der Stufe II oder III zugeordnet.
- Ein Algorithmus, der ein oder mehrere Unterprogramme mit Parametern enthält, wird im Allgemeinen der Stufe III zugeordnet.

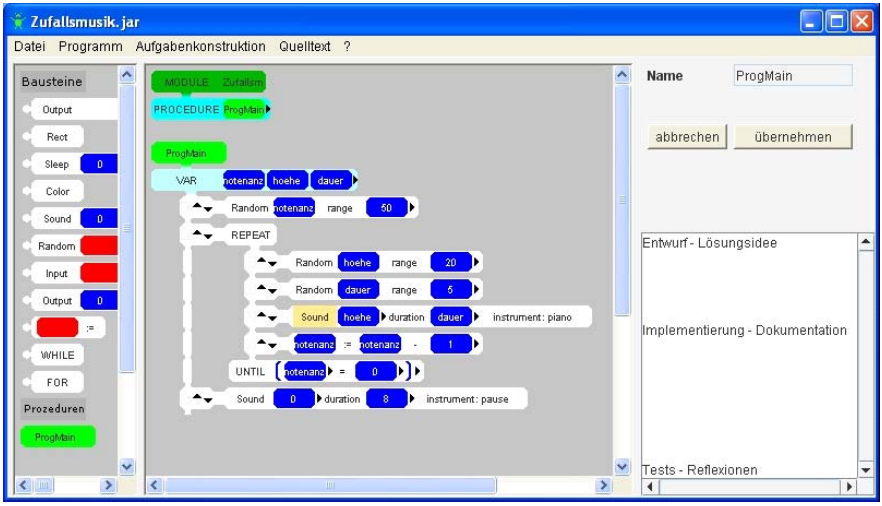


Abbildung 7.1:
Puck-Programm, das Musik erzeugt.

Die Kompetenzen wurden in die vier Komponenten A, B, C und D aufgeteilt. Diese Komponenten stellen keine Reihenfolge oder Gewichtung dar. Vielmehr geht es um eine Systematisierung des Modells und um eine begründete Vorwegnahme der Struktur eines Kompetenztests. In einem Kompetenztest kann dann jeder Komponente eine Aufgabe auf der jeweiligen Stufe zugeordnet werden. Das Kompetenzmodell wurde ohne Bezugnahme auf eine bestimmte Programmier-

sprache entwickelt. In dem Forschungsprojekt wurde die visuelle Programmiersprache Puck eingesetzt; siehe Abbildung 7.1 (Kohl u. a., 2007). Möglich wäre auch der Einsatz eines anderen Werkzeugs gewesen.

Kompetenztest

Der Kompetenztest orientiert sich an den geforderten Kompetenzen des Kompetenzmodells. Zu jeder Komponente und zu jeder Stufe gibt es eine Aufgabe. Die Lehrerinnen und Lehrer legen selbst fest, wie viel Zeit für die Bearbeitung vorgesehen wird. Empfohlen sind 90 Minuten. Sie entscheiden auch, ob und ggf. wie der Test benotet werden soll. Zur Auswahl der Aufgaben des Kompetenztests wurden vier Varianten vorgegeben:

1. Die Lehrerinnen und Lehrer legen eine Stufe fest, die von der ganzen Klasse einheitlich bearbeitet werden soll.
2. Die Lehrerinnen und Lehrer legen individuell für jede Schülerin und jeden Schüler (oder für Schülergruppen) fest, welche Stufe bearbeitet werden soll.
3. Jede Schülerin und jeder Schüler legt für sich fest, welche Stufe bearbeitet werden soll.
4. Jede Schülerin und jeder Schüler legt für sich für jede der Komponenten die Stufe fest, die bearbeitet werden soll.

Es wurde vorgeschlagen, dass die Schülerinnen und Schüler, die vor Ablauf der Zeit fertig sind, auch Aufgaben einer weiteren Stufe bearbeiten können. In die Bewertung sollten dann bei jeder der vier Komponenten A, B, C und D jedoch nur die Aufgaben einer Stufe einfließen (siehe Tabelle 7.2).

	Stufe I	*Stufe II*	*Stufe III*
A	4	8	12
B	4	8	12
C	8	16	24
D	8	16	24
Summe	24	48	72

Tabelle 7.2:
Punkteverteilung beim Kompetenztest.

Die Einschätzung sollte nach folgenden Kriterien erfolgen:

- Um Stufe I zu erreichen, müssen mindestens 18 Punkte erreicht werden.
- Um Stufe II zu erreichen, müssen mindestens 36 Punkte erreicht werden.
- Um Stufe III zu erreichen, müssen mindestens 54 Punkte erreicht werden.

Begleituntersuchung

Nach dem Einsatz des Kompetenztests wurden die beteiligten Lehrerinnen und Lehrer gebeten, einen Fragebogen zu ihren Erfahrungen auszufüllen. Nachfolgend werden einige Ergebnisse der Befragung zusammengestellt (siehe Kohl, 2009).

- Drei Lehrerinnen und Lehrer setzten die Materialien bei Hauptschülern, acht bei Realschülern und 29 bei Gymnasiasten in insgesamt elf deutschen Ländern und der Schweiz ein.
- Ein Großteil der Lehrerinnen und Lehrer hat den Unterricht oft (60,0 %) bzw. immer (12,5 %) am Kompetenzmodell ausgerichtet. Die restlichen Lehrerinnen und Lehrer richteten den Unterricht gelegentlich (17,5 %) bzw. selten (10,0 %) am Kompetenzmodell aus.
- Insgesamt konnte festgestellt werden, dass das entwickelte Kompetenzmodell von 97,5 % der Lehrerinnen und Lehrer in irgendeiner Weise zur Planung und von 90,0 % in irgendeiner Weise zur Einschätzung von Unterricht eingesetzt wurde.
- 47,5 % der Lehrerinnen und Lehrer stellten den Schülerinnen und Schülern das Kompetenzmodell im Unterricht kurz vor, zwei Lehrpersonen (5,0 %) taten dies sogar ausführlich.
- 60,0 % der Lehrerinnen und Lehrer sprachen sich für Fortbildungen zum Einsatz von Kompetenzmodellen im Informatikunterricht aus.
- 65,0 % der Lehrerinnen und Lehrer gaben an, dass sie Kompetenzmodelle auch in anderen Bereichen des Informatikunterrichts einsetzen würden.
- Die bereitgestellten Aufgaben wurden von 60,0 % der Lehrerinnen und Lehrer oft und von 15,0 % immer zur Unterrichtsvorbereitung eingesetzt. 20,0 % der Lehrerinnen und Lehrer setzten die Aufgaben gelegentlich, jeweils eine Lehrperson setzte die Aufgaben selten (2,5 %) bzw. nie (2,5 %) ein.
- 82,5 % der Lehrerinnen und Lehrer stimmten der Aussage zu, dass die Einordnung der Aufgaben in die Komponenten und Stufen des Kompetenzmodells eine wichtige Hilfe war.
- Die bereitgestellten Aufgaben wurden zu 15,0 % oft, zu 30,0 % gelegentlich und zu 35,0 % selten zur (Binnen-)Differenzierung eingesetzt.
- Im Unterricht der Lehrpersonen, bei denen Schülerinnen und Schüler im Kompetenztest auch Kompetenzen der Stufen II bzw. III nachweisen konnten, wurde häufiger mithilfe von Aufgaben, die unterschiedlichen Stufen zugeordnet waren, differenziert.
- Insgesamt werden die dargestellten Ergebnisse in der Dissertation so interpretiert, dass ein Großteil der Lehrerinnen und Lehrer die bereitgestellten

Aufgaben sinnvoll als Grundlage von kompetenzorientiertem Unterricht einsetzen konnte.

- 57,5 % der Lehrpersonen gaben an, den Kompetenztest im Unterricht eingesetzt zu haben.
- Von den Lehrpersonen, die die Fragen zum Kompetenztest beantworteten, gaben 50,0 % an, dass der Kompetenztest benotet wurde und 50,0 %, dass er nicht benotet wurde.
- Nachfolgend wird angegeben, wie häufig die o. g. Varianten zur Auswahl der Aufgaben des Kompetenztests genutzt wurden: 19 Lehrerinnen und Lehrer legten eine Stufe fest, die von der ganzen Klasse einheitlich bearbeitet wurde. Zwölfmal wurde von allen Schülerinnen und Schülern Stufe I des Kompetenztests bearbeitet, Stufe II wurde viermal bearbeitet, Stufe III wurde dreimal bearbeitet. Keine Lehrperson hat individuell für jede Schülerin und jeden Schüler (oder für Schülergruppen) festgelegt, welche Stufe bearbeitet werden soll. Bei einer Lehrerin haben die Schülerinnen und Schüler selbstständig eine zu bearbeitende Stufe gewählt. Zwei Lehrpersonen stellten ihren Schülerinnen und Schülern frei, bei jeder Komponente eine Stufe zu wählen. Dass den Schülerinnen und Schülern zumindest von drei Lehrpersonen die Möglichkeit gegeben wurde, die zu bearbeitende Stufe selbst zu wählen, wird in der Dissertation als Indiz für die Durchführbarkeit der beiden letztgenannten Varianten gedeutet.
- Da von den meisten Lehrerinnen und Lehrern einheitlich eine von allen Lernenden zu bearbeitende Stufe festgelegt wurde, wäre es sinnvoll, einen einzigen Kompetenztest zu entwickeln, mit dem die Kompetenzen mehrerer Stufen überprüft werden können. Ob das beim Thema »Algorithmen« möglich ist und wie ein solcher Kompetenztest strukturiert werden könnte, ist offen.
- Von keinem einzigen Lehrer wurde beanstandet, dass kein einheitlicher Erwartungshorizont bereitgestellt wurde.
- Das Bewertungskriterium, nach dem mindestens 75 % der auf der jeweiligen Stufe möglichen Punkte erreicht werden mussten, wurde größtenteils als »genau richtig« eingeschätzt.
- Es gab keine Anmerkungen zu Problemen bei der Korrektur des Tests.
- 87,5 % der Lehrerinnen und Lehrer gaben an, dass Kompetenztests hilfreich sind, um die Kompetenzen der Schülerinnen und Schüler einzuschätzen.

Fazit

Zumindest bei traditionellen Themen der Schulinformatik kann es durch Bereitstellung von Kompetenzmodell und Übungsaufgaben gelingen, Veränderungen hin zu einem kompetenzorientierten Informatikunterricht anzubahnen. Die Unterrichtsgestaltung erfolgt auf der Grundlage des Kompetenzmodells. Die Übungsaufgaben werden regelmäßig im Unterricht eingesetzt. Zwei Drittel der an der Untersuchung beteiligten Lehrerinnen und Lehrer würden Kompetenzmodelle auch in anderen Bereichen des Informatikunterrichts einsetzen. Viele

Informatiklehrerinnen und -lehrer sind also zu Veränderungen in ihrem Unterricht bereit, formulieren in dem Zusammenhang aber auch die Notwendigkeit von Fortbildung.

Das im Rahmen des Forschungsprojekts entwickelte Kompetenzmodell zum Thema »Algorithmen« ist sehr systematisch aufgebaut. Es gibt drei Niveaustufen und vier inhaltliche Komponenten. Die Übungsaufgaben wurden passend dazu entwickelt. Der Kompetenztest bezieht sich ebenfalls auf das Kompetenzmodell. Diese Durchgängigkeit hat zur Folge, dass auf Grundlage der vorliegenden Materialien ein Unterricht möglich ist, der sich durch (Binnen-)Differenzierung auszeichnet. Lernende könnten im Unterricht und im Kompetenztest solche Aufgaben bearbeiten, die hinsichtlich Schwierigkeit und Komplexität auf sie zugeschnitten sind. Dies wurde im Rahmen des Forschungsprojekts oft oder zumindest gelegentlich von 45 % der beteiligten Lehrerinnen und Lehrern auch so praktiziert.

Der relativ hohe Anteil der benoteten Tests sowie die große Anzahl an Klassen, die einheitlich eine Stufe bearbeiteten, deuten darauf hin, dass der von der Universität Jena bereitgestellte Kompetenztest von vielen Lehrerinnen und Lehrern als eine normale Klassenarbeit angesehen wurde. Das war vom Forschungsdesign her möglich. Hiermit wird vorgeschlagen, *Kompetenztest* und *kompetenzorientierte Klassenarbeit* künftig deutlicher voneinander zu trennen. Die Aufgaben eines Kompetenztests werden von dritter Seite bereitgestellt. Bei einem Kompetenztest geht es vorrangig um die Qualitätssicherung von Lehren und Lernen, um den Vergleich von Klassen, Schulen und Bildungssystemen sowie ggf. auf der individuellen Ebene um Individualdiagnostik und -förderung. Ziel ist nicht das Erteilen einer Note. In einem kompetenzorientierten Unterricht werden Klassenarbeiten auf der Grundlage eines (evtl. schulformspezifischen) Kompetenzmodells vom Lehrer selbst erarbeitet und von ihm auch verantwortet. Der Lehrer beurteilt und erteilt Noten. Es ist seine pädagogische Aufgabe, sich das präzise Anspruchsniveau, die inhaltliche Schwerpunktsetzung und auch die Formulierungen in der Klassenarbeit auf Basis eines Kompetenzmodells und des erteilten Unterrichts genau zu überlegen.

In kompetenzorientierten Klassenarbeiten können mehrere Aufgabenserien mit unterschiedlichem Schwierigkeitsgrad bereitgestellt werden. Jeder Schüler entscheidet nach vorheriger Beratung durch den Lehrer letztendlich selbst, welche Aufgabenserie von ihm bearbeitet wird. Die Benotung könnte unter den derzeitigen Rahmenbedingungen vielleicht so aussehen: Bearbeitet der Schüler die Aufgabenserie der Stufe III, dann kann er, wenn alles perfekt ist, die Note 1 erhalten. Bearbeitet er die Aufgabenserie der Stufe II, dann kann er bestenfalls die Note 2 erhalten. Bei der Bearbeitung der Aufgabenserie der Stufe I ist bestenfalls die Note 3 möglich. Das ist ein anderer Ansatz als die Leistungsbewertung über Defizite, bei der sich die Note aus dem Abstand zwischen der Schülerlösung zum Erwartungshorizont ergibt. Dies wird mit dem beschriebenen Vorgehen zumindest gemildert.

Informatikunterricht in der Sekundarstufe II

Die höchste Einheit muss sein, aber sie darf
der Mannigfaltigkeit nichts nehmen.
Friedrich von Schiller

8 Bildungsstandards Informatik für die Sekundarstufe II

Die einheitlichen Prüfungsanforderungen[27] der KMK in der Abiturprüfung Informatik (EPA Informatik) bilden den bundesweiten Maßstab für die Abiturprüfung im Unterrichtsfach Informatik und dienen der gegenseitigen Anerkennung der Abiturprüfungen der einzelnen Länder (KMK, 2004; Fothe, 2008).

In den EPA Informatik sind die fachlichen und methodischen Kompetenzen in folgende *Kompetenzbereiche* eingeordnet:

- Erwerb und Strukturierung informatischer Kenntnisse,
- Kennen und Anwenden informatischer Methoden,
- Kommunizieren und Kooperieren und
- Anwenden informatischer Kenntnisse, Bewerten von Sachverhalten und Reflexion von Zusammenhängen.

Die fachlichen Inhalte der EPA Informatik sind in folgende *Lern- und Prüfungsbereiche* eingeteilt:

- Grundlegende Modellierungstechniken[28],
- Interaktion mit und von Informatiksystemen sowie
- Möglichkeiten und Grenzen informatischer Verfahren.

Von zentraler Bedeutung ist die fachspezifische Modellierung. Im Informatikunterricht einiger Länder gehört das Umsetzen mit zum Modellbildungszyklus, in anderen Ländern wird die Modellbildung als Voraussetzung für das Umsetzen angesehen. Beide Auffassungen finden sich in der Beschreibung des Modellbildungszyklus in den EPA Informatik wieder (KMK, 2004, S. 7):

> Die Prüflinge […] sind insbesondere mit dem Modellbildungszyklus vertraut; dazu gehören in problemadäquater Auswahl und Reihenfolge: Problemanalyse und Problemspezifikation, Abgrenzen des Problems, Abstraktion, Idealisierung, Strukturieren und Zerlegen in Teilprobleme (Modularisieren), Formalisieren, Umsetzen unter Berücksichtigung der zur Verfügung stehenden Werkzeuge und Hilfsmittel, Testen der Lösung, kritisches Reflektieren der Ergebnisse und der Lösung allgemein, Überarbeitung des Modells, Optimierung der Lösung.

[27] http://www.kmk.org/fileadmin/veroeffentlichungen_beschluesse/1989/1989_12_01_EPA_Informatik.pdf
[28] im Sinne von Techniken der geistigen Arbeit

Die EPA Informatik wurden auf der Grundlage der im Jahr 2003 geltenden Informatiklehrpläne der Länder erarbeitet. Sie setzen dennoch inhaltliche Schwerpunkte. Themen, die den Lern- und Prüfungsbereichen der EPA Informatik nicht zuzuordnen sind, können bis zu einem Drittel einer Prüfungsaufgabe ausmachen; das zugehörige Anforderungsniveau muss dem der anderen Aufgaben entsprechen. Durch diese Festlegung war es nicht notwendig, alle Themen, die nach den einzelnen Landesregelungen in den Schulen unterrichtet werden und Gegenstand der Abiturprüfung in den jeweiligen Ländern sein können, in die EPA aufzunehmen. Die Länder können im Sinne einer Profilbildung ergänzende Anforderungen mit Prüfungsrelevanz formulieren.

Zu den EPA Informatik gehören zahlreiche Aufgabenbeispiele aus traditionellen und modernen Bereichen der Schulinformatik, deren Vielfalt sicher die konkrete Gestaltung von Informatikunterricht positiv beeinflusst.

Die EPA Informatik legen fest, welchen Schwierigkeits- und Komplexitätsgrad Abituraufgaben haben sollen. In den EPA Informatik werden dazu drei Anforderungsbereiche gekennzeichnet, und es werden Beispielanforderungen angegeben. Ein Teil der Beispiele findet sich in der Tabelle 8.1 wieder; sie sind hier nach Themen und Anforderungsbereichen geordnet.

AB	*Beispielanforderungen*
	Anforderungen zum Modellieren:
I	Verwenden einfacher Modellierungen […]
I	Wiedergeben eines bekannten Modells in geübter Darstellung
II	Erstellen eines Modells zu einem Problem mit bekannten Verfahren
II	Überprüfen der Eignung eines bekannten informatischen Modells für die Lösung einer neuen Problemstellung
III	Beurteilen der eigenen Modellierung […] im Anwendungskontext
	Anforderungen zu Möglichkeiten und Grenzen des Einsatzes von Informatiksystemen:
I	Beschreiben von Anwendungsmöglichkeiten der Informations- und Kommunikationstechniken und deren Wechselwirkungen mit Individuen und Gesellschaft
II	Analysieren eines Fallbeispiels (z. B. Datenschutz, Auswirkungen der neuen Informations- und Kommunikationstechniken)
III	Formulieren einer begründeten Stellungnahme zu einem authentischen Text in Bezug auf Möglichkeiten, Angemessenheit und Grenzen des Einsatzes von Informatiksystemen
	Anforderungen zu Algorithmen:
I	Wiedergeben von […] einfachen Algorithmen […] in einer im Unterricht behandelten Darstellungsform
I	Verwenden […] bekannter einfacher Algorithmen
II	Analysieren eines gegebenen Algorithmus

AB	*Beispielanforderungen*
II	Übertragen von Aufwandsbetrachtungen auf einen vergleichbaren, aber nicht bekannten Algorithmus
II	Begründen von bestimmten Eigenschaften (z. B. Terminierung, Zeit- und Speicheraufwand) eines gegebenen Algorithmus durch nicht formale Überlegungen
III	Entwickeln eines […] Algorithmus zur Lösung eines neuen Problems
	Anforderungen zum objektorientierten Modellieren:
I	Identifizieren von Objekten und ihren Beziehungen in einem bekannten Sachzusammenhang
II	Durchführen einer objektorientierten Analyse und Entwickeln eines objektorientierten Designs für eine vergleichbare neue Problemstellung
	Anforderungen zum Problemlösen:
I	Einfaches Erweitern einer vorgegebenen Problemlösung in geübtem Zusammenhang
II	Planvolles Einsetzen bekannter Informatiksysteme zur Lösung einer neuen Problemstellung aus einem bekannten Bereich
II	Nutzen vorhandener Programmbibliotheken für die eigene Problemlösung
II	Dokumentieren einer Problemlösung mit angemessenen Darstellungsmitteln
III	Durchführen einer komplexen Problemanalyse
III	Zerlegen eines gegebenen anspruchsvollen Problems in geeignete Teilprobleme
III	Beurteilen der eigenen […] Problemlösung im Anwendungskontext
	Anforderungen zu formalen Sprachen und Automaten:
I	Beschreiben und Darstellen bekannter Automaten […]
II	Entwickeln eines einfachen Automaten
III	Entwickeln einer Sprache (z. B. Angabe der Syntax und Semantik einer einfachen Steuersprache für einen Roboter)

Tabelle 8.1:
Beispielanforderungen der EPA Informatik zu relevanten Themen der Schulinformatik (AB = Anforderungsbereiche).

Zu relevanten Themen der Schulinformatik sind in den EPA also Anforderungen in unterschiedlichen Niveaus angegeben. Die EPA Informatik beinhalten daher in Ansätzen ein dreistufiges Kompetenzmodell. Wesentliches Ziel des Modells ist jedoch nicht das Zuordnen von Prüflingen zu Aufgaben, die diese in der Abiturprüfung bearbeiten sollen, oder das Zuordnen von Prüflingen zu einer Kompetenzstufe, die diese im Ergebnis der Prüfung nachgewiesen haben. Vielmehr geht es um das Konfigurieren von Abituraufgaben. In den EPA Informatik ist dazu festgelegt: Das Schwergewicht der zu erbringenden

Prüfungsleistungen liegt im Anforderungsbereich II (»Transferleistung«). Daneben sind der Anforderungsbereich I (»Wiedergabeleistung«) und der Anforderungsbereich III (»schöpferische Leistung«) zu berücksichtigen, und zwar Anforderungsbereich I in höherem Maße als Anforderungsbereich III.

Wir schlagen vor, die vorliegenden EPA Informatik zu Bildungsstandards für die Sekundarstufe II weiterzuentwickeln. Für diese Evolution spricht u. a. die langjährige Erfahrung von Lehrerinnen und Lehrern in der Arbeit mit EPA. Die Bildungsstandards wären Grundlage sowohl für den Informatikunterricht in der gymnasialen Oberstufe in allen 16 Ländern als auch für die Abiturprüfung. Wir empfehlen ein *dreistufiges* Kompetenzmodell. Dadurch könnten die mit den Anforderungsbereichen der EPA Informatik gewonnenen Erfahrungen genutzt und fortentwickelt werden. Bei der Entwicklung eines Kompetenzmodells zum Thema »Algorithmen« für die Sekundarstufe I deutete einiges darauf hin, dass es bei drei Stufen am besten gelingt, eine sachlich begründete und gleichzeitig lesbare Stufung der Kompetenzen vorzunehmen (siehe Kapitel 7). Von Relevanz ist die beabsichtigte Nutzung des Kompetenzmodells im Unterricht, bei Leistungsbewertungen (z. B. in Kursarbeiten) und in der Abiturprüfung. Wird es in der Abiturprüfung nur zum Konfigurieren von Aufgaben genutzt oder ist z. B. vorgesehen, den Prüflingen vollständige Abituraufgaben für alle drei Kompetenzstufen zur Auswahl zu übergeben? Dazu müsste evtl. das System der Notengebung grundlegend verändert werden. Möglicherweise wird sich dies als zwingende Konsequenz aus der allgemeinen Entwicklung hin zu einem kompetenzorientierten Unterricht erweisen, der auf mehrstufigen Kompetenzmodellen beruht.

Falls die Bildungsstandards als unmittelbare Grundlage für das Erarbeiten von Abituraufgaben verwendet werden sollen, sind die Kompetenzen sehr konkret auszuarbeiten. Ein Beispiel dafür sei an dieser Stelle angegeben (zu einem weiteren Beispiel siehe S. 77). Bei den fachlichen Inhalten der EPA Informatik wird speziell zur objektorientierten Modellierung angegeben: »Objekt, Klasse, Beziehungen zwischen Klassen, Interaktion von Objekten, Klassendiagramm (z. B. mit UML)«. Diese Inhalte werden folgendermaßen konkretisiert:

Die Schülerinnen und Schüler

a) erläutern Grundkonzepte der objektorientierten Modellierung (Objekt, Klasse, Vererbung, Polymorphie, Datenkapselung, Wiederverwendbarkeit),
b) modellieren Probleme, dokumentieren die Modelle und stellen die Modelle mit grafischen Mitteln dar,
c) analysieren, modifizieren und überprüfen eigene oder gegebene Modellierungen und
d) implementieren objektorientierte Modelle[29].

Mit den Freiheiten, die sich aus den Festlegungen in den Bildungsstandards ergeben, muss man anschließend beim Erarbeiten von Abituraufgaben umgehen können. Von sechs grundlegenden Modellierungstechniken sind nach den Fest-

[29] http://www.informatiktest.de/

legungen in den EPA Informatik mindestens zwei im Grundkursfach und mindestens drei im Leistungskursfach zu thematisieren. Beim Erarbeiten von Bildungsstandards Informatik für die Sekundarstufe II wäre zu prüfen, ob diese erhebliche Wahlmöglichkeit weiterhin Bestand haben sollte. Bei einer Reihe von Fragen sind Erfahrungen aus den Fächern sicher von Interesse, für die Bildungsstandards für die Sekundarstufe II bereits entwickelt werden. Relevante Forschungsergebnisse der Fachdidaktik Informatik sind in geeigneter Weise heranzuziehen (z. B. Nelles u. a., 2010).

Mit Blick auf die Zielstellung eines Gesamtkonzepts zur informatischen Bildung wird ein möglicher Zusammenhang zwischen Bildungsstandards der Sekundarstufen I und II aufgezeigt: Die fachlichen und methodischen Kompetenzen sowie die fachlichen Inhalte der EPA Informatik lassen sich ohne große Schwierigkeiten den Inhalts- und Prozessbereichen der GI-Empfehlungen zuordnen. Die Inhalts- und Prozessbereiche der GI-Empfehlungen könnten also auch zur Strukturierung der Kompetenzen von Bildungsstandards Informatik für die Sekundarstufe II genutzt werden. Bildungsstandards Informatik für die Sekundarstufen I und II könnten also die gleiche Grobstruktur besitzen (Fothe, 2008). Hinzuweisen ist darauf, dass in den EPA Informatik die grundlegenden Modellierungstechniken den fachlichen Inhalten zugeordnet sind; in den GI-Empfehlungen gehört das Modellieren zu den Prozessbereichen.

Das Merkmal *Verbindlichkeit für alle* aus der »Klieme-Expertise« scheint für Bildungsstandards Informatik für die Sekundarstufe II nicht sinnvoll zu sein, denn es würde auf Mindeststandards hindeuten. An der Abiturprüfung nehmen jedoch sowohl leistungsstarke, durchschnittliche als auch leistungsschwache Schülerinnen und Schüler teil. Auch für leistungsstarke Schülerinnen und Schüler müssen die Prüfungsanforderungen klar sein. Das spricht für das Erarbeiten von Regel- oder Maximalstandards.

Das Bearbeiten von Problemen aus unterschiedlichen Fachgebieten ist für die Informatik und den Informatikunterricht charakteristisch und sollte es daher auch für die Abiturprüfung sein (siehe Kapitel 15). Sind künftig Abiturprüfungen realistisch, die Informatik und einem anderen Unterrichtsfach gleichermaßen zugeordnet werden?

Ein Beispiel wäre eine gemeinsame Abiturprüfung von Informatik und Physik. Dies wäre ein Beitrag für Interdisziplinarität und würde auch Komplexaufgaben ermöglichen, wie sie für die Sekundarstufe II typisch sein sollten. Als Voraussetzung dafür müsste das System der Abiturprüfung grundlegend verändert werden; die Bildungsstandards sollten dazu fachspezifische Festlegungen treffen.

In den Kapiteln 15 und 16 wird deutlich werden, dass zu den Abiturprüfungen einiges zu bedenken ist. Dabei muss unterschieden werden,

1. ob man eine dezentrale Abiturprüfung zu erarbeiten hat,
2. ob zentrale Landesprüfungen vorgesehen sind, bei denen die Länder so wie bisher eigene Aufgaben entwickeln,
3. ob zentrale Landesprüfungen vorgesehen sind, bei denen sich die Länder aus einem nationalen Aufgabenpool bedienen, oder

4. ob eine deutschlandweite Abiturprüfung durchgeführt werden soll, die gleichzeitig an allen Schulen mit gymnasialer Oberstufe geschrieben wird.

Die 3. und 4. Variante sind bisher keine schulische Realität.

Ich kann der Vision einer deutschlandweiten Abiturprüfung im Fach Informatik einiges abgewinnen. Die Länder (oder evtl. sogar Schulen) sollten dabei jedoch in einem definierten Umfang eigene Teilaufgaben stellen können. Für die eigenen Teilaufgaben könnte insgesamt ein Drittel der Punkte (Bewertungseinheiten) der gesamten Prüfungsaufgabe vorgesehen werden. Damit wäre es den Ländern (oder Schulen) möglich, eigene Schwerpunkte zu setzen. Dies würde einer unnötigen Uniformierung des Unterrichts entgegen wirken, was gerade für ein Unterrichtsfach wie Informatik sinnvoll ist, das sich durchaus dynamisch entwickelt hat und sich auch weiterhin dynamisch entwickeln wird. Die Größenordnung von einem Drittel würde die bisherigen Regelungen in den EPA Informatik in gewisser Weise fortschreiben.

Wenn du es nicht ausführen kannst,
dann tu es in den Keller!
Friedrich L. Bauer & Klaus Samelson, 1955

9 Fallbeispiel: Rekursion und Iteration

Bei Rekursion und Iteration handelt es sich um fundamentale Ideen der Informatik. Rekursion und Iteration treten bei vielen theoretischen und praktischen Fragestellungen in der Informatik (siehe bereits bei Barron, 1971) und in anderen Fachgebieten wie z. B. Mathematik und Biologie auf. In den EPA Informatik wird gefordert: »Die Prüflinge […] können verschiedene Problemlösungsstrategien und Techniken wie Iteration, Rekursion und Klassenbildung einsetzen.« In diesem Kapitel wird diese Festlegung zu Rekursion und Iteration konkretisiert, und es werden Materialien angegeben, die zur Kompetenzentwicklung genutzt werden können. Lehrerinnen und Lehrer wählen auf der Grundlage der eigenen Vorstellungen Materialien aus und bringen diese in die gewünschte zeitliche Reihenfolge. Dieses Kapitel ist die inhaltliche Grundlage von Kapitel 10.

Teilkompetenzen zu Rekursion und Iteration

Die Schülerinnen und Schüler
a) erläutern die Grundlagen von Rekursion und Iteration (Vergleichen von Rekursion und Iteration, Äquivalenz von Rekursion und Iteration sowie Prinzip der Abarbeitung eines rekursiven Algorithmus auf einem iterativ arbeitenden Computer),
b) definieren informatische Begriffe auf rekursive Art,
c) verwenden die Syntaxdefinition einer Programmiersprache sachgemäß,
d) analysieren und erläutern exemplarisch Computerprogramme, denen rekursive oder iterative Algorithmen zugrunde liegen, und
e) entwerfen und implementieren solche Computerprogramme.

Rekursion in Texten

In natürlichen Sprachen lassen sich Strukturen finden, mit denen sich Vor-Erfahrungen zur Rekursion gewinnen lassen. Der Satz »Dass dieser Wurm an Würmern litt, die wiederum an Würmern litten …« (Joachim Ringelnatz) und die Ankündigung »Ich pflege jedem Sonntagskind, das sich zu mir zu finden weiß, drei Wünsche zu gewähren. Die ersten zwei sind frei, den dritten kann ich verweigern, wenn er töricht ist.« (Wilhelm Hauff: Das kalte Herz) illustrieren einen Spezialfall der Rekursion, die Endrekursion, sofern der dritte Wunsch

wieder drei Wünsche sind. Ein weiteres Beispiel für Rekursion sind Schachtelsätze, wie ein Eintrag in einem Lokalanzeiger:

> Derjenige der den Täter der den Pfahl der an der Brücke die an dem Weg der nach Jena führt liegt steht umgeworfen hat anzeigt erhält eine Belohnung.

Die Strukturierung kann mit Kommas erfolgen:

> Derjenige, der den Täter, der den Pfahl, der an der Brücke, die an dem Weg, der nach Jena führt, liegt, steht, umgeworfen hat, anzeigt, erhält eine Belohnung.

Möglich ist auch eine Strukturierung durch Ebenen:

> Derjenige erhält eine Belohnung.
> der den Täter anzeigt
> der den Pfahl umgeworfen hat
> der an der Brücke steht
> die an dem Weg liegt
> der nach Jena führt

Auf jeder Ebene, bis auf die unterste, wird der Textfluss unterbrochen und später fortgesetzt. Die Rekursion wird auf syntaktischer (oder Satzglied-)Ebene deutlich. Hier gilt für das beschriebene Beispiel (ohne Satzzeichen):

Satz = Subjekt [Attributsatz] Prädikat Objekt.
Attributsatz = Subjekt Objekt [Attributsatz] Prädikat.

Struktur mathematischer Kurven

Hilbert-Kurven besitzen interessante mathematische Eigenschaften und werden auch praktisch angewandt, so z. B. bei der Bestimmung der Lastverteilung der einzelnen Prozessoren von Parallelrechnern.

Im Unterricht können die Hilbert-Kurven der Ordnung 1, 2 und 3 vorgegeben werden (siehe Abbildung 9.1). H_1 ist die elementare Hilbert-Kurve. Es werden die Konstruktion von H_2 aus H_1 und von H_3 aus H_2 nachvollzogen. Die Schülerinnen und Schüler erkennen, dass jeweils vier Hilbert-Kurven H_k zur Hilbert-Kurve H_{k+1} zusammengesetzt werden. Die vier Hilbert-Kurven werden jeweils in die gleiche Position zueinander gebracht und mit »Verbindungsstrecken«, die in der Abbildung aus Verständnisgründen hervorgehoben sind, verbunden.

Im Unterricht kann man auch umgekehrt vorgehen und nur die Hilbert-Kurve H_3 vorgeben. Dann wird herausgearbeitet, dass die Hilbert-Kurve H_3 aus vier

Hilbert-Kurven H_2 und dass die Hilbert-Kurve H_2 aus vier Hilbert-Kurven H_1 besteht. Die Schülerinnen und Schüler erkennen ganz allgemein, dass eine Hilbert-Kurve H_k aus vier Hilbert-Kurven H_{k-1} besteht. Diese Analyse bereitet das Umsetzen in ein Computerprogramm vor.

Andere Kurven, die analysiert werden können, sind z. B. die Sierpinski-Kurven, der Baum des Pythagoras und die Schneeflockenkurve.

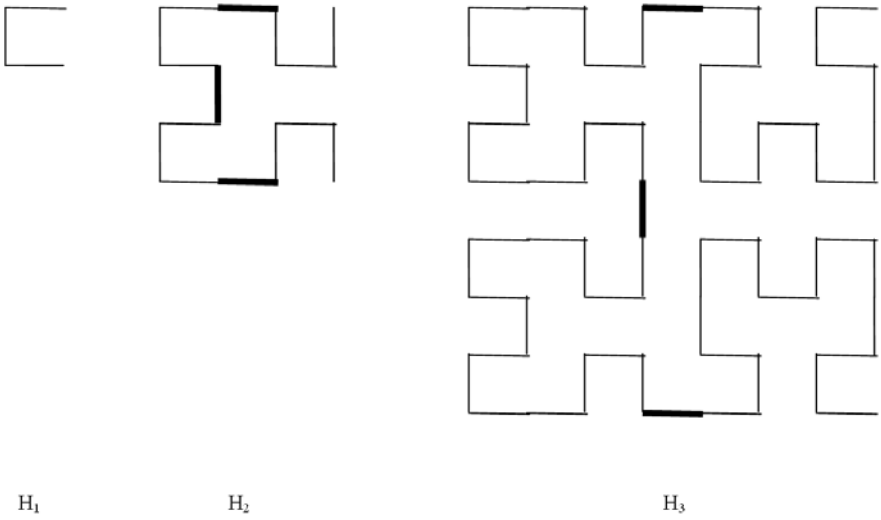


Abbildung 9.1:
Hilbert-Kurven der Ordnung 1, 2 und 3.

Grundlagen von Rekursion und Iteration

Bei der Betrachtung der Bearbeitung per Computer oder Hand kann Iteration als Wiederholung strukturgleicher Blöcke durch Aneinanderreihung, Rekursion als Wiederholung strukturgleicher Blöcke durch Schachtelung gekennzeichnet werden (siehe Abbildungen 9.2 und 9.3).

Iterative Problemlösungen können in rekursive umgesetzt werden. Dabei entsteht Endrekursion. Rekursive Problemlösungen können auf einem Computer nach dem von-Neumann-Rechnermodell – solche Computer arbeiten iterativ – zur Abarbeitung gebracht werden. Damit erweisen sich Rekursion und Iteration als äquivalent.

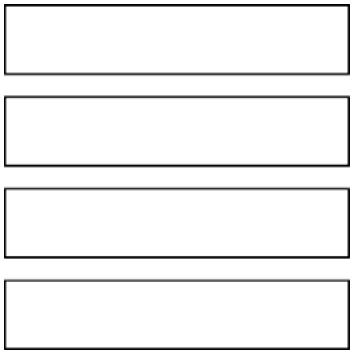


Abbildung 9.2:
Iteration als Aneinanderreihung strukturgleicher Blöcke.

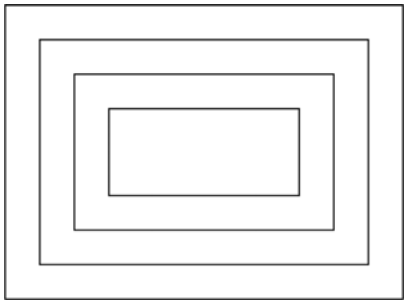


Abbildung 9.3:
Rekursion als Schachtelung strukturgleicher Blöcke.

Überführen von Iteration in Rekursion

Dieser Vorgang soll am Beispiel der Berechnung des größten gemeinsamen Teilers (ggT) zweier positiver ganzer Zahlen x und y nach Euklid verdeutlicht werden (siehe auch S. 45). Die Berechnung kann iterativ mit einer Schleife erfolgen (Python-Programm):

```
def ggt1(x, y):
    while y > 0:
        rest = x % y
        x = y
        y = rest
    return x
```

```
a = 666
b = 306
g = ggt1(a, b)
print g
```

Die ggT-Berechnung lässt sich auch rekursiv formulieren:

```
def ggt2(x, y):
    if y == 0:
        return x
    else:
        return ggt2(y, x % y)

a = 666
b = 306
g = ggt2(a, b)
print g
```

Die Funktion ggt2 ist ein Beispiel für Endrekursion. Beim rekursiven Aufstieg sind keine Operationen mehr auszuführen. Die Funktion ggt2 stellt die Beschreibung eines mathematischen Sachverhalts dar. Die Beschreibung lässt die Form der internen Abarbeitung offen.

Überführen von Rekursion in Iteration

Am Beispiel von Quicksort wird gezeigt, wie rekursive Prozeduren mithilfe eines Stapels auf einem iterativ arbeitenden Computer zur Abarbeitung gebracht werden. Quicksort besitzt eine rekursive Struktur (siehe S. 146). Das Array a, das zu sortieren ist, liegt in einer globalen Variablen vor. Beim Abarbeiten von Sort(l, r) werden die Elemente von a[l] bis a[r] in die richtige Reihenfolge gebracht. Die Prozedur Sort enthält zwei rekursive Aufrufe, und zwar für die linke und die rechte Teilfolge (Methode »Teile und beherrsche«). Der rekursive Abbruch (Basisfall) liegt vor, wenn eine Teilfolge nur aus einem Element besteht. Nun kommen wir zur internen Abarbeitung: Bevor ein rekursiver Aufruf der Prozedur Sort abgearbeitet wird, werden die Werte der lokalen Variablen und der Wertparameter sowie die Rückkehradresse gespeichert. Die Speicherung erfolgt in einem Stapel (»Rekursionsstapel«), der vom Laufzeitsystem vieler Programmiersprachen automatisch verwaltet wird. Zur Illustration der Arbeit des Rekursionsstapels wird die folgende Geschichte erzählt:

Ein zerstreuter Mensch will aus irgendeinem Grund von einem Hotelzimmer in ein anderes umziehen. Im bisherigen Hotelzimmer bleiben dabei verschiedene Sachen liegen, darunter mehrere Paar Schuhe, die teilweise geputzt sind. Das Zimmermädchen fotografiert das Zimmer, räumt alles zusammen und legt es ins

Lager. Dann verbringt der zerstreute Mensch herrliche Tage im nächsten Zimmer, im übernächsten Zimmer, im überübernächsten Zimmer usw. Bei jedem Umzug lässt er etwas liegen.

Nach Abarbeitung der Prozedur Sort werden die Werte der lokalen Variablen und der Wertparameter sowie die Rückkehradresse vom Stapel geholt. Sie haben sich in der Zwischenzeit nicht verändert. Sie stehen an der Stapelspitze und damit an der richtigen Position. Die lokalen Variablen und die Wertparameter erhalten ihre ehemaligen Werte zurück – und zwar so, als hätte es gar keinen Aufruf der Prozedur Sort gegeben. Die Rückkehradresse gibt an, welcher Befehl als nächstes abzuarbeiten ist. Die Arbeit des Stapels lässt sich innerhalb einer Wiederholungsanweisung organisieren. Damit wird die Iteration deutlich. Wir erzählen die Geschichte nun weiter (siehe Abbildung 9.4):

Irgendwann will der zerstreute Mensch in das vorhin verlassene Hotelzimmer zurückkehren. Das Zimmermädchen richtet es genauso ein, wie es vor dem Verlassen aussah und legt auch einen Zettel hin. Auf dem Zettel steht, welcher Schuh als nächstes zu putzen ist. Das macht er dann auch.

Abbildung 9.4:
Die Hauptperson in einer rekursiven Geschichte.

Rekursive Begriffe

Das Definieren von rekursiven Begriffen – auch das ist eine Form des Modellierens – kann dem Erarbeiten rekursiver Computerprogramme vorausgehen. Nachfolgend geht es um grundlegende Datenstrukturen und um den Begriff *Turm*, der später wieder aufgegriffen wird (siehe S. 94 ff.).

Das erste Beispiel ist der Begriff *Liste*:

> Eine Liste besteht aus dem Erst-Element und der Rest-Liste, oder es ist die leere Liste. Eine Liste besteht aus endlich vielen Elementen.

Das zweite Beispiel ist der Begriff *binärer Baum* (siehe Abbildung 9.5):

> Ein binärer Baum besteht aus einem Element (der Wurzel) und zwei binären Bäumen (dem linken und dem rechten Teilbaum), oder es ist der leere binäre Baum. Ein binärer Baum besteht aus endlich vielen Elementen.

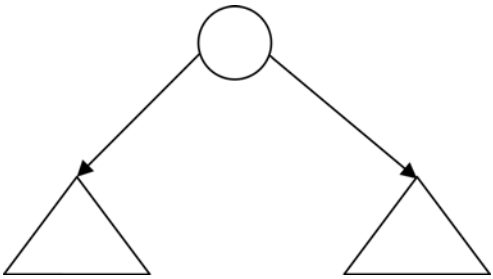


Abbildung 9.5:
Ein binärer Baum ist aus der Wurzel (Kreis) und
dem linken und dem rechten Teilbaum (Dreiecke) aufgebaut.

Anwendungen sind z. B. das Visualisieren von Ja-Nein-Entscheidungen, Ausdrucksbäume und der Stammbaum einer Person (Ahnentafel), wenn nur wenige Generationen zu betrachten sind. Aus biologischen Gründen eignen sich binäre Bäume praktisch nicht als Modell eines Stammbaums, wenn es um 1000 oder mehr Jahre geht und Redundanzen vermieden werden sollen (siehe Sykes, 2003).

Die Schülerinnen und Schüler sollen sich mit der Definition des Begriffes *binärer Baum* aktiv auseinander setzen. Mögliche Aufgaben sind das Begründen, dass es sich bei der in Abbildung 9.6 dargestellten Struktur wirklich um einen binären Baum handelt, und das Angeben korrekter binärer Bäume.

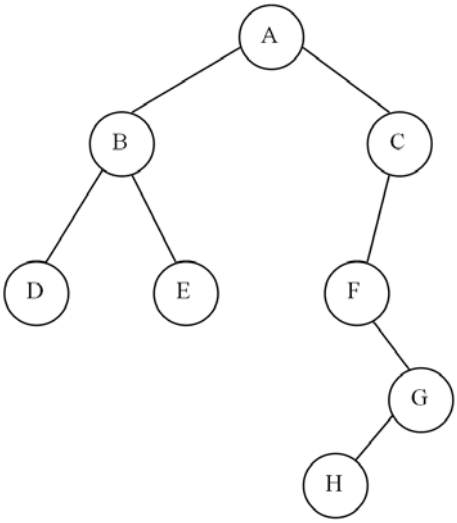


Abbildung 9.6:
Handelt es sich um einen binären Baum? Begründen Sie Ihre Antwort.

Das dritte Beispiel *Suchbaum* ist komplizierter:

Ein Suchbaum ist ein binärer Baum, für den zusätzlich die Suchbaum-Eigenschaft gilt. Diese besagt, dass für den gesamten Baum und für alle seine Teilbäume gilt:

- Alle Schlüssel im linken Teilbaum sind kleiner als der Schlüssel der Wurzel.
- Alle Schlüssel im rechten Teilbaum sind größer als der Schlüssel der Wurzel.

In einem Suchbaum kann ein gespeichertes Objekt leicht gefunden werden, und zwar in einer Folge von Links-Rechts-Entscheidungen (zu einer Aufgabe siehe Abbildung 9.7).

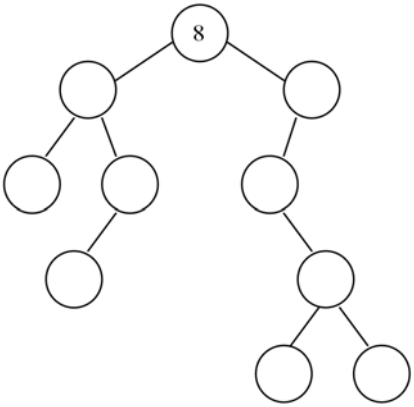


Abbildung 9.7:
Ergänzen Sie die fehlenden Schlüssel so, dass ein Suchbaum entsteht.
Nehmen Sie ausschließlich ganze Zahlen. Kein Schlüssel soll mehr als einmal vorkommen.

Das vierte Beispiel ist das Definieren von *inorder*, *preorder* und *postorder*. Dabei handelt es sich um Reihenfolgen von Knoten, die sich beim Durchlaufen eines binären Baumes ergeben. Um die Reihenfolgen zu erzeugen, werden die rekursiven Vorschriften LWR, WLR und LRW auf den gesamten Baum und auf alle seine Teilbäume angewandt. Das Zeichen L steht für linker Teilbaum, R für rechter Teilbaum und W für Wurzel.

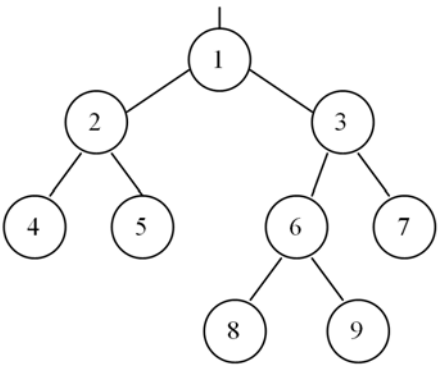


Abbildung 9.8:
Gegebener Baum.

Für den binären Baum aus der Abbildung 9.8 ergeben sich unter Anwendung der rekursiven Vorschriften die folgenden Reihenfolgen der Knoten:

inorder	L W R	4 2 5 1 8 6 9 3 7
preorder	W L R	1 2 4 5 3 6 8 9 7
postorder	L R W	4 5 2 8 9 6 7 3 1

Werden die rekursiven Vorschriften auf einen Ausdrucksbaum angewandt, so erhält man die Ausdrücke in der Infix-Notation (ohne Klammern), Prefix-Notation und Postfix-Notation (siehe S. 46 ff.).

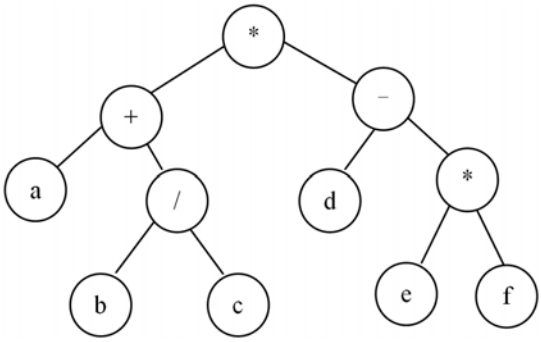


Abbildung 9.9:
Gegebener Ausdrucksbaum.

Für den Ausdrucksbaum aus Abbildung 9.9 ergibt sich:

inorder	Infix-Notation (ohne Klammern)[30]	a + b / c * d – e * f
preorder	Prefix-Notation	* + a / b c – d * e f
postorder	Postfix-Notation	a b c / + d e f * – *

Das fünfte Beispiel ist die Definition von *Turm* (siehe Abbildung 9.10):

Ein Turm besteht aus seiner größten Scheibe und dem Rest-Turm, oder es ist der leere Turm.

[30] vollständig mit Klammern: (a + b / c) * (d – e * f)

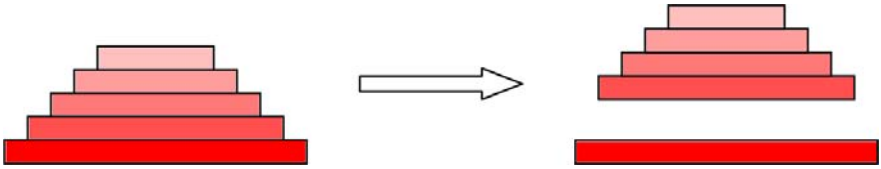


Abbildung 9.10:
Ein Turm der Höhe 5 wird in seine größte Scheibe und
den Rest-Turm der Höhe 4 zerlegt.

Syntaxdefinition

Die syntaktische Beschreibung einer Programmiersprache erfolgt meist durch Angabe einer Chomsky-2-Grammatik. Bei der Syntaxbeschreibung wird häufig Rekursion verwendet, um eine kompakte Beschreibung zu erhalten. Ein Beispiel ist die syntaktische Definition von Anweisung in der Programmiersprache Java, die nachfolgend auszugsweise angegeben ist. Die Definition erfolgt mithilfe der EBNF:

```
Anweisung = … ( …
                | "if" "(" Ausdruck ")" Anweisung [ "else" Anweisung ]
                | "while" "(" Ausdruck ")" Anweisung
                | "return" [ Ausdruck ] ";"
                | … ).
```

Das zu definierende Nichtterminalsymbol Anweisung ist mehrfach auf der rechten Seite angegeben, d.h., bei der Definition von Anweisung wird Anweisung verwendet.

Ein weiteres Beispiel ist die syntaktische Definition von Ausdruck in der Programmiersprache Oberon-2 (siehe Mössenböck, 1998):

```
Relation = "=" | "#" | "<" | "<=" | ">" | ">=" | "IN" | "IS".
AddOperator = "+" | "–" | "OR".
MulOperator = "*" | "/" | "DIV" | "MOD" | "&".
Ausdruck = EinfacherAusdruck [ Relation EinfacherAusdruck ].
EinfacherAusdruck = [ "+" | "–" ] Term { AddOperator Term }.
Term = Faktor { MulOperator Faktor }.
Faktor = Zahl | ZeichenKonstante | Zeichenkette | "NIL" | Menge |
      Bezeichner [ AktuelleParameter ] | "(" Ausdruck ")" | "~" Faktor.
```

Der syntaktische Aufbau eines Ausdrucks wird durch ein System von Nichtterminalsymbolen rekursiv beschrieben (Ausdruck, EinfacherAusdruck, Term

und Faktor). Ein Faktor kann z. B. eine Zahl, ein Variablenbezeichner oder – und damit wird die Rekursion deutlich – wieder ein Ausdruck (eingeschlossen in runde Klammern) sein. Nach der Beschreibung stellen z. B. die folgenden Zeichenfolgen syntaktisch korrekte Ausdrücke dar:

```
2 * a + b = c
1.0 – ( x – 1.0 ) / ( y – 1.0 )
a + b * ( c + d * ( e + f * ( g + h * i ) ) )
```

Beim Erzeugen von Ausdrücken sind Kontextbedingungen zu beachten. Diese betreffen die Verträglichkeit von Operanden und Operatoren zueinander. Man nennt dies Ausdruckskompatibilität. Die Schülerinnen und Schüler überprüfen gegebene Zeichenfolgen auf Korrektheit und erzeugen zielgerichtet korrekte Zeichenfolgen.

Algorithmen und Computerprogramme

Nachfolgend sind exemplarisch einige Themen aufbereitet, an denen sich die Teilkompetenzen d) und e) entwickeln lassen (siehe S. 77). Die Themen besitzen unterschiedlichen Schwierigkeitsgrad und auch verschiedene praktische Relevanz. Generell wird in diesem Buch von der These ausgegangen, dass ein Verständnis von Rekursion eher dann erzielt wird, wenn der Bezug zum Ablauf deutlich gemacht wird.

Einstiegsbeispiel für Iteration: Heron-Verfahren

In diesem Abschnitt geht es wieder um das Informatiksystem Taschenrechner (siehe Kapitel 4). Jetzt wird der Frage nachgegangen, wie ein Taschenrechner die 3. Wurzel aus einer Zahl näherungsweise berechnen kann. Wir betrachten dazu ein geometrisches Modell: Gegeben ist das Volumen a eines Würfels, gesucht ist dessen Seitenlänge. Wir entwickeln für die Berechnung einen iterativen Algorithmus und gehen dabei von einem Quader mit quadratischer Grundfläche aus. Die Seitenlänge der Grundfläche sei x0. Der Quader soll das gleiche Volumen wie der Würfel besitzen, also a. Die drei Seitenlängen des Quaders sind also x0, x0 und a / x0 / x0. (Wir machen die Probe: Das Produkt der drei Seitenlängen ist gleich dem Volumen a.) Wir ermitteln nun das arithmetische Mittel x1 der drei Seitenlängen und betrachten einen neuen Quader mit quadratischer Grundfläche. Deren Seitenlänge ist x1. Wir weisen nun den Wert von x1 der Variablen x0 zu und rechnen erneut so, wie eben beschrieben. In jedem Schritt nähert sich der Quader der Würfelform an. Den Prozess brechen wir ab, wenn sich die Seitenlänge in einem Berechnungsschritt nur noch wenig ändert. Der zuletzt berechnete Wert x1 ist das Resultat der Berechnung. Ein Struktogramm zur Beschreibung des Algorithmus stellt die Abbildung 9.11 bereit.

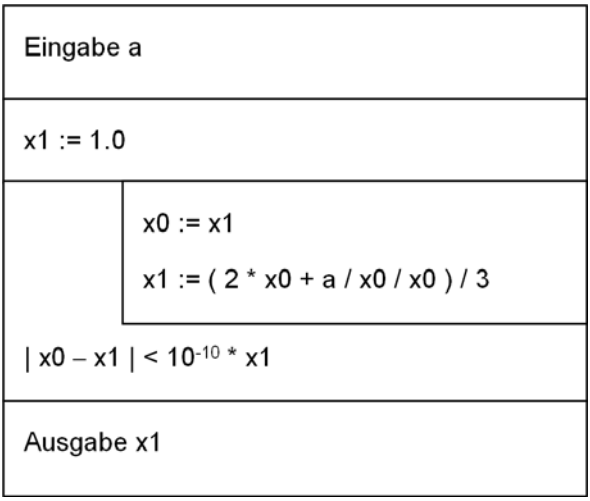


Abbildung 9.11:
Iterative Berechnung der 3. Wurzel aus a.

Einstiegsbeispiel für Rekursion: Umdrehen einer Zeichenfolge

Ein Beispiel ist ein Python-Programm, das eine Zeichenfolge auf rekursive Art umdreht:

- Erstes Zeichen abspalten,
- Rest-Zeichenfolge umdrehen und
- abgespaltenes Zeichen an die umgedrehte Rest-Zeichenfolge anhängen.

```
def umdrehen(zeichenfolge):
    if zeichenfolge == "":
        return ""
    else:
        erstesZeichen = zeichenfolge[0]
        restzeichenfolge = zeichenfolge[1:]
        return umdrehen(restzeichenfolge) + erstesZeichen

print umdrehen("abcd")
print umdrehen("nebelbeisiebleben")
```

An dem Programm wird deutlich, dass Rekursion durch einen Selbstaufruf von Unterprogrammen gekennzeichnet ist. In der zweiten Zeile von unten wird die

Funktion umdrehen das erste Mal aufgerufen. Der Wert "abcd" wird an den Wertparameter zeichenfolge der Funktion umdrehen übergeben. Da die Zeichenfolge nicht leer ist, werden die Werte für die lokalen Variablen berechnet: erstesZeichen="a" und restzeichenfolge="bcd". Mit umdrehen("bcd") ruft sich die Funktion umdrehen rekursiv auf. Wir nennen dies den rekursiven Abstieg. Weitere Selbstaufrufe sind umdrehen("cd"), umdrehen("d") und umdrehen(""). Mit der Abarbeitung von umdrehen("") ist der rekursive Abbruch (Basisfall) erreicht und die Funktion umdrehen liefert die leere Zeichenfolge als Funktionswert. An die leere Zeichenfolge werden beim rekursiven Aufstieg nacheinander die Zeichen "d", "c", "b" und "a" angehangen. Als Ergebnis entsteht die Zeichenfolge "dcba". Jedes Mal, wenn die Funktion umdrehen zur Abarbeitung gebracht wird (bis auf das letzte Mal), werden die lokalen Variablen erstesZeichen und restzeichenfolge erzeugt.

Einstiegsbeispiel für Rekursion: Überführen von Dezimal- in Dualzahlen

Die Schülerinnen und Schüler wissen, dass Computer intern im Dualsystem arbeiten. Eine Konsequenz davon ist, dass eine Dezimalzahl, die vom Computer eingelesen wird, vor der Verarbeitung in die entsprechende Dualzahl zu überführen ist. Das folgende Python-Programm löst diese Aufgabe:

```
def dual(zahl):
    ganz = zahl / 2
    rest = zahl % 2
    if rest == 0:
        zeichen = '0'
    else:
        zeichen = '1'
    if ganz == 0:
        return zeichen
    else:
        return dual(ganz) + zeichen

n = input("ganze Zahl: ")
print dual(n)
```

Rekursion in der logischen Programmiersprache Prolog

Rekursion ist eines der tragenden Konzepte beim Programmieren in Prolog. Ein Beispiel ist das Prädikat append, das in vielen Prolog-Systemen ein Standard-Prädikat ist. Es besitzt eine rekursive Struktur und kann u. a. zum Zusammenfügen zweier Listen verwendet werden; zum Programmieren mit Prolog siehe Clocksin/Mellish, 2003; Röhner, 2007; Thiele u. a., 2001.

```
append ( [ ] , Liste , Liste ).
append ( [ E | L1 ] , L2 , [ E | L3 ] ) :- append ( L1 , L2 , L3 ).
```

Die Anfrage ?- append ([1 , 2 , 3] , [4 , 5] , L). führt dazu, dass die Listen [1 , 2 , 3] und [4 , 5] zur Liste L zusammengefügt werden. E ist das Erst-Element der jeweiligen Liste. Der Separator | dient sowohl dazu, eine Liste in Erst-Element und Rest-Liste zu zerlegen, als auch zum Anfügen eines Elements am Kopf einer Liste. Beide Funktionalitäten werden in dem gegebenen Programm genutzt. Tabelle 9.1 beschreibt, wie das Prolog-Programm arbeitet. Die Zeilen der Tabelle sind farbig gestaltet, um den Verständnisprozess zu unterstützen.

Aufruf	*E*	*L1*	*L2*	*L*	*L3*
append([1 , 2 , 3] , [4 , 5] , L)	1	[2 , 3]	[4 , 5]	[1 | L3]	
append([2 , 3] , [4 , 5] , L3)	2	[3]	[4 , 5]		[2 | L3]
append([3] , [4 , 5] , L3)	3	[ ]	[4 , 5]		[3 | L3]
append([ ] , [4 , 5] , L3)			[4 , 5]		[4 , 5]
					[3 , 4 , 5]
					[2 , 3 , 4 , 5]
				[1 , 2 , 3 , 4 , 5]	

Tabelle 9.1:
Abarbeitung der Anfrage ?- append ([1 , 2 , 3] , [4 , 5] , L).

Rekursion vs. Iteration

Leonardo Fibonacci befasste sich im Jahr 1202 mit einer Kaninchenvermehrungsaufgabe, die auf die folgende Zahlenfolge führt: Beginnend mit 0 und 1 wird das jeweils nächste Glied der Zahlenfolge als Summe der beiden vorausgehenden Glieder ermittelt. Die nächsten zehn Glieder der Fibonacci-Zahlenfolge sind also 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. Bis heute werden immer wieder Gesetzmäßigkeiten und Anwendungen entdeckt, die mit dieser berühmten Zahlenfolge zu tun haben. Verwiesen sei z. B. auf den Zusammenhang mit dem goldenen Schnitt[31]. Glieder der Zahlenfolge lassen sich in einer Schleife berechnen. Dabei handelt es sich um *Iteration*.

Das *rekursive* Bildungsgesetz der Zahlenfolge lautet:

$a_0 = 0$
$a_1 = 1$
$a_n = a_{n-1} + a_{n-2}$, falls $n \geq 2$

[31] http://www.bezreg-duesseldorf.nrw.de/lerntreffs/mathe/pages/magazin/leute/fibonnacci.pdf

Das Bildungsgesetz lässt sich in ein rekursives Programm umsetzen. Das Berechnen von Gliedern der Zahlenfolge kann also iterativ, rekursiv und auch explizit mit einer Formel erfolgen (Formel von Moivre-Binet). Arbeitsteilig können die Schülerinnen und Schüler die drei Möglichkeiten auf einem Computer realisieren und es werden die Vorteile und Nachteile der erarbeiteten Programme diskutiert. Ein wesentlicher Unterschied liegt in der benötigten Rechenzeit. Bei weitem am schlechtesten schneidet das rekursiv arbeitende Programm ab. Die Ursachen werden diskutiert. Als Konsequenz wird herausgearbeitet, dass Glieder von rekursiv definierten Zahlenfolgen am besten iterativ oder mit einer Formel berechnet werden.

Beispiel für Rekursion: Unterbrechen einer Tätigkeit

Das folgende Beispiel knüpft an die zweite Strukturierung des Schachtelsatzes an (siehe S. 78).

Es klingelt. Ich lasse Anna herein ▼ und sage Anna guten Tag.
Es klingelt. Ich lasse Bert herein ▼ und sage Bert guten Tag. ▲
Es klingelt. Ich lasse Clara herein ▼ und sage Clara guten Tag. ▲
Es klingelt. Ich lasse Dirk herein und sage Dirk guten Tag. ▲

Bevor ich Anna, Bert und Clara begrüßen kann, klingelt es wieder. Meine Tätigkeit wird von einer gleichartigen Tätigkeit unterbrochen. Ich begrüße erst Dirk, dann Clara, Bert und Anna (vgl. Thüringer Abiturprüfung, Grundfach Informatik, 2001).

Beispiel für Rekursion: Permutationen

Ein rekursiver Algorithmus soll alle Anordnungen (Permutationen) von gegebenen Elementen ermitteln. Der Algorithmus wird nachfolgend an den Elementen A, B, C und D erläutert. Mithilfe von Tauschoperationen werden die folgenden Viererfolgen erzeugt:

A	B	C	D

A	B	D	C

A	D	C	B

D	B	C	A

Jedes der vier Elemente steht einmal an der vierten Position. Nach dem Erzeugen einer jeden Viererfolge werden mithilfe von Tauschoperationen Dreierfolgen erzeugt. Das Element an der vierten Position bleibt dabei fest. Zum Beispiel werden aus der zweiten Viererfolge die folgenden Dreierfolgen erzeugt:

A	B	D	C

A	D	B	C

D	B	A	C

Jedes der drei Elemente steht einmal an der dritten Position. Nach dem Erzeugen einer jeden Dreierfolge werden mithilfe von Tauschoperationen Zweierfolgen erzeugt. Die Elemente an der dritten und vierten Position bleiben dabei fest. Zum Beispiel werden aus der dritten Dreierfolge die folgenden Zweierfolgen erzeugt:

D	B	A	C

B	D	A	C

Jedes der beiden Elemente steht einmal an der zweiten Position.

Das beschriebene Handlungsmuster ist unabhängig von der Anzahl an Elementen einer Folge (ob 4, 3 oder 2). Mit dem rekursiven Vorgehen werden die 24 Permutationen für die gegebenen vier Elemente ermittelt.

Der Algorithmus kann z. B. als Oberon-Programm angegeben werden. Entscheidend ist die Prozedur Perm. Bei vier Elementen ist Perm(3) der erste Aufruf dieser Prozedur (die Nummerierung der Elemente beginnt mit 0). Der rekursive Abbruch (Basisfall) liegt vor, wenn eine Folge nur aus einem Element besteht.

```
PROCEDURE Perm(n: INTEGER);
VAR k: INTEGER;
BEGIN
  IF n = 0 THEN LoesungAusgeben ELSE
    FOR k := n TO 0 BY -1 DO
       Tausch(a[k], a[n]);
       Perm(n – 1);                    (* rekursiver Aufruf *)
       Tausch(a[k], a[n])
    END
  END
END Perm;
```

Das Programm kann im Informatikunterricht beim Lösen von verschiedenen Problemen genutzt werden. Ein Beispiel ist das Problem des Handlungsreisenden: Ein Handlungsreisender will nacheinander n Städte besuchen (siehe Abbildung 9.12). Gesucht ist eine kürzeste Rundreise (vgl. Thüringer Abiturprüfung, Leistungsfach Informatik, 2001). Die Permutation 1 4 3 2 beschreibt die folgende Rundreise: Beginnend in der Stadt 1 geht es nacheinander in die Städte

4, 3 und 2 und dann wieder zur Ausgangsstadt 1. Zur Lösung werden drei Teillösungen erarbeitet:
1. Ermitteln aller Permutationen,
2. Ermitteln der Länge einer jeden Rundreise und
3. Heraussuchen einer kürzesten Rundreise.

Die Methode des systematischen Durchprobierens aller Varianten ist nur für wenige Städte anwendbar, da die benötigte Rechenzeit exponentiell mit der Problemgröße, der Anzahl an Städten, ansteigt.

Abbildung 9.12:
Rundreise durch n Städte.

Klassiker: Türme von Hanoi

Bei dem Spiel »Türme von Hanoi« sind Scheiben von einem Feld auf ein anderes in möglichst wenigen Schritten zu bewegen. Auf einem dritten Feld können Scheiben zwischengelagert werden (siehe Abbildung 9.13). Zwei Spielregeln sind

zu beachten: Bewege immer nur eine Scheibe, und lege eine Scheibe stets auf eine größere oder auf ein leeres Feld.

Beim Thema »Rekursion und Iteration« kommt man an den Türmen von Hanoi eigentlich nicht vorbei. Folgende Gründe für deren Beliebtheit sollen genannt werden:

- Es handelt sich um ein Spiel, das beim ersten Kontakt durchaus eine Herausforderung darstellt.
- Computerprogramme zur rekursiven Problemlösung sind einfach aufgebaut und dennoch leistungsstark.
- Es lassen sich bemerkenswerte iterative Problemlösungen finden.

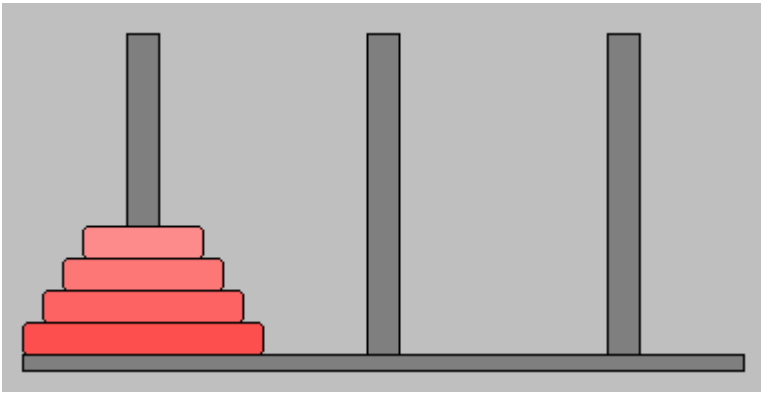


Abbildung 9.13:
Türme von Hanoi.

Wir formulieren einen Lösungsalgorithmus unter Verwendung des rekursiven Begriffs *Turm* (siehe S. 86 f.):

> Die Aufgabe, einen *Turm* vom Startfeld a zum Zielfeld b unter Verwendung eines Hilfsfeldes c zu versetzen, wird folgendermaßen gelöst: Ist der Turm leer, so ist nichts zu tun. Andernfalls versetze den *Rest-Turm* von a nach c, bewege die größte Scheibe des Turms von a nach b und versetze anschließend den *Rest-Turm* von c nach b. Die Nummer des jeweiligen Hilfsfeldes kann mit der Formel c = 6 − a − b berechnet werden. Die Vorschrift wird gestartet mit: Versetze den gesamten *Turm* von a = 1 nach b = 2.

Das Versetzen eines Turmes der Höhe k wird auf das zweimalige Versetzen eines Turmes der Höhe k–1 zurückgeführt. Der rekursive Abbruch (Basisfall) liegt vor, wenn ein leerer Turm zu versetzen ist. Nach dem Lösungsalgorithmus

sind Türme zu versetzen, obwohl nach den Spielregeln nur einzelne Scheiben bewegt werden dürfen. Dieser Widerspruch wird durch wiederholtes Anwenden der rekursiven Vorschrift aufgelöst. Tabelle 9.2 illustriert dies für einen Turm, der aus drei Scheiben besteht. Für den gesamten Turm hat das Startfeld die Nr. 1, das Zielfeld die Nr. 2 und das Hilfsfeld die Nr. 3. Die Scheiben bekommen der Größe nach die Nummern von 1 bis n. Die kleinste Scheibe trägt die Nummer 1.

			Turm(0, 1, 3)	
		Turm(1, 1, 2)	Scheibe(1, 1, 2)	Scheibe(1, 1, 2)
			Turm(0, 3, 2)	
	Turm(2, 1, 3)	Scheibe(2, 1, 3)	Scheibe(2, 1, 3)	Scheibe(2, 1, 3)
			Turm(0, 2, 1)	
		Turm(1, 2, 3)	Scheibe(1, 2, 3)	Scheibe(1, 2, 3)
			Turm(0, 1, 3)	
Turm(3, 1, 2)	Scheibe(3, 1, 2)	Scheibe(3, 1, 2)	Scheibe(3, 1, 2)	Scheibe(3, 1, 2)
			Turm(0, 3, 2)	
		Turm(1, 3, 1)	Scheibe(1, 3, 1)	Scheibe(1, 3, 1)
			Turm(0, 2, 1)	
	Turm(2, 3, 2)	Scheibe(2, 3, 2)	Scheibe(2, 3, 2)	Scheibe(2, 3, 2)
			Turm(0, 1, 3)	
		Turm(1, 1, 2)	Scheibe(1, 1, 2)	Scheibe(1, 1, 2)
			Turm(0, 3, 2)	

Tabelle 9.2:
Abarbeitung des Lösungsalgorithmus für einen Turm der Höhe 3.
Turm(k, a, b) bedeutet: Versetze den Turm der Höhe k von a nach b.
Scheibe(k, a, b) bedeutet: Bewege Scheibe k von a nach b.
In jeder Spalte der Tabelle steht im Grunde genommen das Gleiche. Die rechte Spalte ist jedoch die einzige, die ausschließlich direkt ausführbare Handlungen enthält.

Mit dem Lösungsalgorithmus kann im Unterricht in unterschiedlicher Weise weitergearbeitet werden. Er kann in ein Struktogramm oder in ein Computerprogramm überführt werden. Das folgende Python-Programm enthält zwei rekursive Unterprogrammaufrufe. Beim ersten Aufruf handelt es sich um echte Rekursion, beim zweiten um Endrekursion.

```
def Turm(k, a, b):
    if k != 0:
        c =  6 - a - b
        Turm(k - 1, a, c)
        print (k, a, b)
        Turm(k - 1, c, b)
    return
Turm(3, 1, 2)
```

Eine Aufgabe für interessierte Schülerinnen und Schüler kann darin bestehen, das Versetzen eines Turmes in grafischer Form auf dem Monitor darzustellen.

Aus der rekursiven Problemlösung lassen sich iterative herleiten. Eine Variante ist das wechselweise Anwenden der beiden folgenden Regeln:

- Bewege die kleinste Scheibe eins weiter (entweder stets rechts herum oder stets links herum).
- Bewege eine andere Scheibe. (Das sieht nur großzügig aus.)

Im Unterricht kann thematisiert werden, dass sich mit jeder um 1 gewachsenen Scheibenanzahl die Rechenzeit verdoppelt. Die Zeit wächst also exponentiell mit der Problemgröße. Diese Erkenntnis hat Konsequenzen: Bei 64 Scheiben werden für das Versetzen des Turmes fast 600.000 Jahre benötigt, wenn das Bewegen einer Scheibe eine Mikrosekunde dauert (10^{-6} s).

Zeichnen von Hilbert-Kurven

Dieser Abschnitt knüpft an die Analyse von Hilbert-Kurven an (siehe S. 78 f.). Das Zeichnen von Hilbert-Kurven kann in zwei Teilprogrammen erfolgen. Das erste Teilprogramm liest die Ordnung der Hilbert-Kurve ein und gibt eine Folge von Zeichenanweisungen aus. Für die Hilbert-Kurve H_2 ergibt sich die Folge SWNWWSESWSEENES. Die Zeichenanweisungen S / N / E / W bedeuten: ein Schritt in Richtung Süden / Norden / Osten / Westen. Das erste Teilprogramm enthält die vier rekursiv arbeitenden Unterprogramme A, B, C und D. Es ist nachfolgend in Python angegeben:

```
def A(k):
    if k > 0:
        D(k-1)
        print "W",
        A(k-1)
        print "S",
        A(k-1)
        print "E",
        B(k-1)
    return

def B(k):
    if k > 0:
        C(k-1)
        print "N",
        B(k-1)
        print "E",
        B(k-1)
        print "S",
        A(k-1)
    return
```

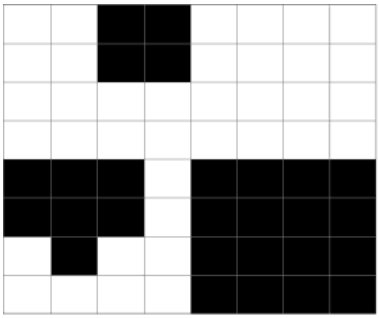
```
def C(k):
    if k > 0:
        B(k-1)
        print "E",
        C(k-1)
        print "N",
        C(k-1)
        print "W",
        D(k-1)
    return

def D(k):
    if k > 0:
        A(k-1)
        print "S",
        D(k-1)
        print "W",
        D(k-1)
        print "N",
        C(k-1)
    return

ordnung = input("Ordnung: ")
A(ordnung)
```

Das zweite Teilprogramm liest eine Folge von Zeichenanweisungen ein, interpretiert diese und gibt eine Folge von gleich langen Strecken auf dem Monitor aus. Beide Teilprogramme können gleichzeitig erarbeitet werden. Damit eignet sich die Aufgabe für ein arbeitsteiliges Vorgehen: Ein Schüler erarbeitet das Teilprogramm für das Erzeugen der Zeichenanweisungen; er befasst sich mit Rekursion. Ein anderer Schüler realisiert das Interpretieren von Zeichenanweisungen; er hat Grafik als inhaltlichen Schwerpunkt. Beide Schüler können weitgehend unabhängig voneinander arbeiten.

Anwendung von Rekursion: Speichern von Bildern



0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1
1	1	1	0	1	1	1	1
0	1	0	0	1	1	1	1
0	0	0	0	1	1	1	1

Abbildung 9.14:
Links ist eine Rastergrafik angegeben; die einzelnen Pixel sind weiß oder schwarz.
Rechts liegt ein Modell der Grafik vor.

Die Rastergrafik von Abbildung 9.14 wird in vier Quadranten aufgeteilt:

- Der erste Quadrant (rechts oben) besteht ausschließlich aus Nullen. Daher wird für ihn eine 0 gespeichert.
- Der zweite Quadrant (links oben) besteht aus Nullen und Einsen. Er wird wiederum in Quadranten zerlegt. Diese vier Quadranten bestehen einheitlich aus jeweils vier Nullen oder aus vier Einsen. Für den zweiten Quadranten wird (1,0,0,0) gespeichert.
- Der dritte Quadrant (links unten) besteht aus Nullen und Einsen. Er wird weiter zerlegt und es zeigt sich, dass zwei dieser Quadranten nur aus Nullen bzw. nur aus Einsen bestehen und dass die beiden anderen Quadranten gemischt sind. Es ergibt sich ((0,1,1,0),1,(1,0,0,0),0).
- Der vierte Quadrant (rechts unten) besteht einheitlich aus Einsen. Daher wird für ihn eine 1 gespeichert.

Die gesamte Rastergrafik kann mit (0,(1,0,0,0),((0,1,1,0),1,(1,0,0,0),0),1) gespeichert werden. Diese Zeichenfolge lässt sich in einen Viererbaum (Quadtree) überführen, und es können rekursive Unterprogramme zur Arbeit mit dem Baum angegeben werden. Ein Beispiel dafür ist das Drehen des Bildes um 90°, was durch Rotation der Knoten auf allen Ebenen des Baumes realisiert wird[32].

[32] siehe Aufgabenbeispiel 2.3 der EPA Informatik

Anwendung von Rekursion: Suche in einem Labyrinth

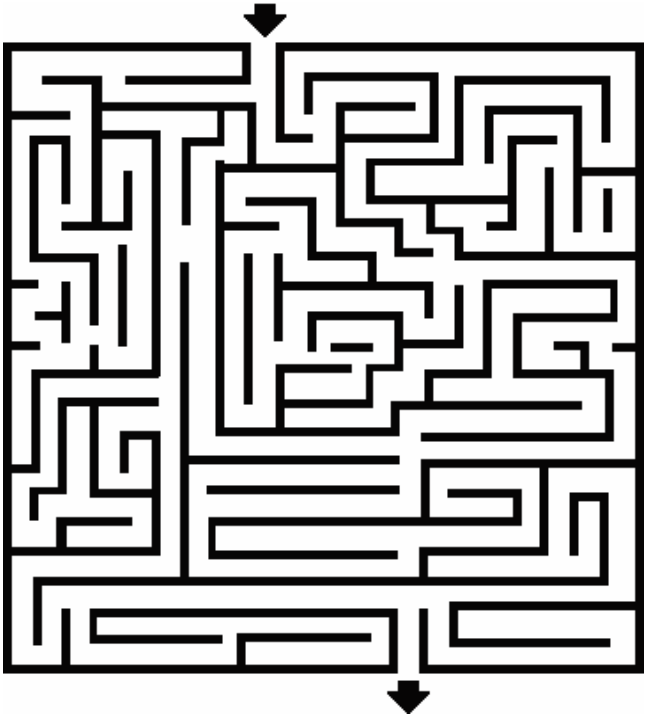


Abbildung 9.15:
Beispiel für ein rechteckiges Labyrinth.

Die Suche in einem Labyrinth ist ein Beispiel für das Problemlösungsverfahren Backtracking. In einem rechteckigen Labyrinth werden alle Wege von einem Start- zu einem Zielfeld gesucht (siehe Abbildung 9.15). Als Modell für das Labyrinth eignet sich ein zweidimensionales Array, dessen Elemente Zeichen sind. Nullen stellen die Wege, Einsen die Mauern oder Hecken dar. Das Zeichen A symbolisiert das Start- und das Zeichen B das Zielfeld. Die Prozedur Schritt führt einen zulässigen Schritt in dem Labyrinth aus und garantiert, dass im Suchprozess alle Varianten ausprobiert und dass Zyklen bei der Suche ausgeschlossen werden. Nach einem zulässigen Schritt erfolgt ein rekursiver Aufruf der Prozedur Schritt. In einer Projektarbeit können die Schülerinnen und Schüler ein vorgegebenes Programm für rechteckige Labyrinthe analysieren und es dann z. B. so modifizieren, dass es bei räumlichen Labyrinthen funktioniert. Bei einem räumlichen Labyrinth gibt es mehrere ebene Labyrinthe übereinander, die durch Leitern an einem oder an mehreren Feldern miteinander verbunden sind.

Rollenspiele zur Rekursion

Rollenspiele können dabei unterstützen, die im Kompetenzbereich »Kommunizieren und Kooperieren« der EPA Informatik angegebenen Kompetenzen herauszubilden (vgl. den 4. Grundsatz, S. 22 ff.).

In diesem Abschnitt werden zwei Rollenspiele beschrieben, die im Informatikunterricht an allgemeinbildenden Gymnasien im Rahmen eines Thüringer Projekts zur Schulentwicklung erprobt wurden (Fothe, 2007). In den Rollenspielen »Fakultätsberechnung« und »Baum minimaler Höhe« geht es um rekursive Algorithmen.

Fakultätsberechnung

Die rekursive Fakultätsberechnung gilt mitunter als Repräsentant eines weltfremden Informatikunterrichts, denn Funktionswerte der Fakultätsfunktion sollten aus Effizienzgründen stets iterativ oder mit einer Tabelle ermittelt werden. Man kann an der Berechnung jedoch sehr schön sehen, wie Rekursion funktioniert. In dem Rollenspiel wird die Formel n! = n * (n-1)! mit dem Basisfall 0! = 1 verwendet. An der Berechnung von 4! wirken die fünf Schüler A, B, C, D und E mit. Jeder ist für einen Funktionsaufruf zuständig und arbeitet ihn ab. Der Lehrer gibt Schüler A den Auftrag zur Berechnung von 4!. Schüler A beauftragt Schüler B mit der Berechnung von 3!. B liefert später das Ergebnis 6. A rechnet dann noch 4 * 6 und teilt das Ergebnis 24 dem Lehrer mit.

- A soll 4! berechnen. A weiß, dass 4! = 4 * 3! gilt und kennt jemanden, der 3! berechnen kann.
- B soll 3! berechnen. B weiß, dass 3! = 3 * 2! gilt und kennt jemanden, der 2! berechnen kann.
- C soll 2! berechnen. C weiß, dass 2! = 2 * 1! gilt und kennt jemanden, der 1! berechnen kann.
- D soll 1! berechnen. D weiß, dass 1! = 1 * 0! gilt und kennt jemanden, der 0! berechnen kann.
- E weiß, dass 0! = 1 gilt und teilt dies D mit.
- D rechnet 1 * 1 = 1 und teilt das Ergebnis C mit.
- C rechnet 2 * 1 = 2 und teilt das Ergebnis B mit.
- B rechnet 3 * 2 = 6 und teilt das Ergebnis A mit.
- A rechnet 4 * 6 = 24 und ist fertig.

Das Rollenspiel kann modifiziert werden. Möglichkeiten dafür sind das Einbeziehen von mehr Schülern, als für die Arbeit benötigt werden, und das Abarbeiten des Rollenspiels ohne Abbruchbedingung. Der Basisfall wird also außer Acht gelassen und es wird stur eine Weile weiter abgearbeitet.

Baum minimaler Höhe

Nach Jerome Bruner gibt es drei Darstellungsformen des Wissens und Könnens (Hartmann u. a., 2006, S. 116 ff.):
- die enaktive Repräsentation (Darstellung durch Handlungen),
- die ikonische Repräsentation (Darstellung durch bildliche Mittel) und
- die symbolische Repräsentation (Darstellung durch Sprache und Zeichen).

Dabei handelt es sich nicht um nacheinander zu durchlaufende Stufen, sondern um Darstellungsformen, die wechselseitig aufeinander bezogen sind. Anhand des Rollenspiels »Baum minimaler Höhe« wird nachfolgend analysiert, welche Bedeutung die drei Darstellungsformen bei Rollenspielen im Informatikunterricht besitzen. Dabei wird von der These ausgegangen, dass alle drei Darstellungsformen von Relevanz sind, auch wenn für Rollenspiele die *enaktive* Repräsentation kennzeichnend ist. Man ist geradezu geneigt, Rollenspiele als *das* Beispiel für diese Darstellungsform anzusehen.

Eine Standardaufgabe bei der Arbeit mit Bäumen lautet (siehe z. B. Wirth, 1979, S. 263 ff.):

Ein binärer Baum minimaler Höhe soll aus einer vorgegebenen Anzahl an Knoten erstellt werden. Die Suchbaum-Eigenschaft muss nicht erfüllt sein (siehe Abbildung 9.16).

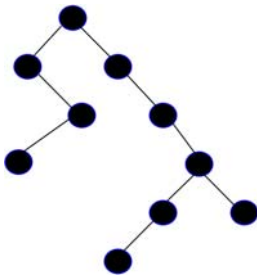


Abbildung 9.16:
Dieser binäre Baum mit 10 Knoten besitzt die Höhe 6.
Der Baum ist nicht von minimaler Höhe.

Vorschrift für das Verteilen von n Knoten:
- Man verwende einen Knoten für die Wurzel.
- Man erzeuge den linken Teilbaum mit nL = n DIV 2 Knoten nach dieser Vorschrift.
- Man erzeuge den rechten Teilbaum mit nR = n − nL − 1 Knoten nach dieser Vorschrift.

Nachfolgend sind die Regeln für das Rollenspiel angegeben:

- Schüler A erhält von seinem Auftraggeber den Auftrag[33]: »Erstelle Baum mit n Knoten.« Schüler A sagt: »Ich bin die Wurzel« und rechnet nL = n DIV 2. Dann gibt er Schüler B den Auftrag: »Erstelle linken Teilbaum mit nL Knoten.«
- Nachdem Schüler B seinen Auftrag erledigt hat, teilt er dem Schüler A mit: »Auftrag erledigt.« Schüler A rechnet dann nR = n − nL − 1 und gibt Schüler C den Auftrag: »Erstelle rechten Teilbaum mit nR Knoten.«
- Nachdem Schüler C seinen Auftrag erledigt hat, teilt er dem Schüler A mit: »Auftrag erledigt.« Schüler A informiert nun seinen Auftraggeber über die Erledigung des Auftrags.
- Die ersten drei Regeln werden rekursiv angewandt.
- Beim Auftrag »Erstelle linken/rechten Teilbaum mit 0 Knoten« gibt der beauftragte Schüler NIL zurück. Der leere Teilbaum entspricht dem Basisfall.
- Die Schüler, die in die Erstellung des Baumes einbezogen sind, stellen sich in Form eines binären Baumes auf.
- Der Lehrer löst den gesamten Prozess aus, indem er einem Schüler z. B. den Auftrag gibt: »Erstelle Baum mit 10 Knoten« (siehe Abbildung 9.17).

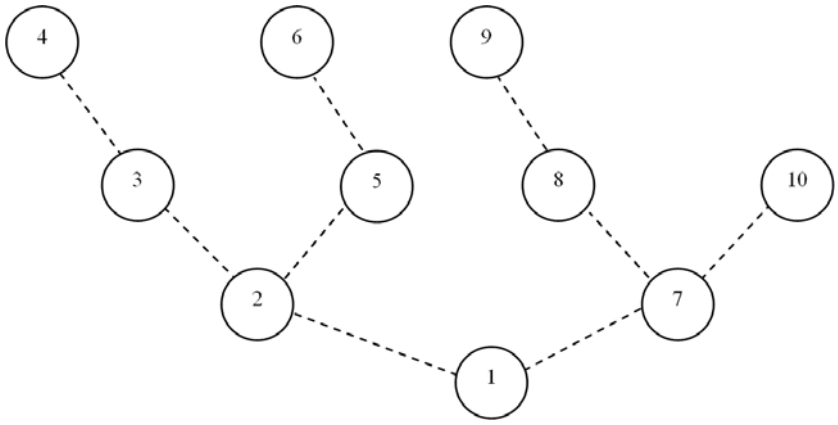


Abbildung 9.17:
Binärer Baum minimaler Höhe, der bei Mitwirkung von zehn Schülern entsteht. Die Schüler werden in der Reihenfolge von 1 bis 10 Bestandteil des Baums. Schüler sehen stets dorthin, wo etwas passiert. Aus Sicht des Schülers 1 ist daher Schüler 2 die Wurzel des linken und Schüler 7 die Wurzel des rechten Teilbaums.

[33] Das Auslösen eines Auftrags kann durch Übergabe eines Blattes geschehen, auf dem die Anzahl n steht. Damit wird das Vergessen dieses Wertes im Verlaufe des Rollenspiels verhindert.

Die Regeln für das Rollenspiel sind in Textform gegeben, was der *symbolischen* Repräsentation entspricht. Beim Rollenspiel arbeitet jeder Schüler einen (rekursiven) Prozeduraufruf ab. Am Ende verkörpert er einen Knoten des binären Baums. Die Schüler stellen sich in Form eines binären Baumes auf. Sie stellen den Baum dar. Sie sind der Baum. Das Ergebnis des Rollenspiels ist somit ein Bild, das durch die Handelnden aufgebaut wird. Beim Aufführen des Rollenspiels geht die *enaktive* in die *ikonische* Repräsentation über. Der übliche binäre Baum lässt sich dann wie in Abbildung 9.18 darstellen.

Nach dem Rollenspiel kann ein entsprechendes Computerprogramm entworfen und implementiert werden. Beim Entwerfen des Programms können die Schülerinnen und Schüler auf die Erfahrungen zurückgreifen, die sie im Rollenspiel gewonnen haben. Dabei handelt es sich um den Übergang von der *enaktiven* in die *symbolische* Repräsentation. Auf einen wesentlichen Unterschied zwischen Rollenspiel und Computerprogramm soll hingewiesen werden: Ein Schüler, der seinen Auftrag erledigt hat, hat seinem Auftraggeber nicht nur »Auftrag erledigt«, sondern zusätzlich einen Zeiger auf sich selbst (genauer: auf den von ihm erstellten Teilbaum) zurückzugeben. Mithilfe dieser Zeiger werden die Knoten des binären Baumes in der richtigen Art und Weise miteinander verbunden.

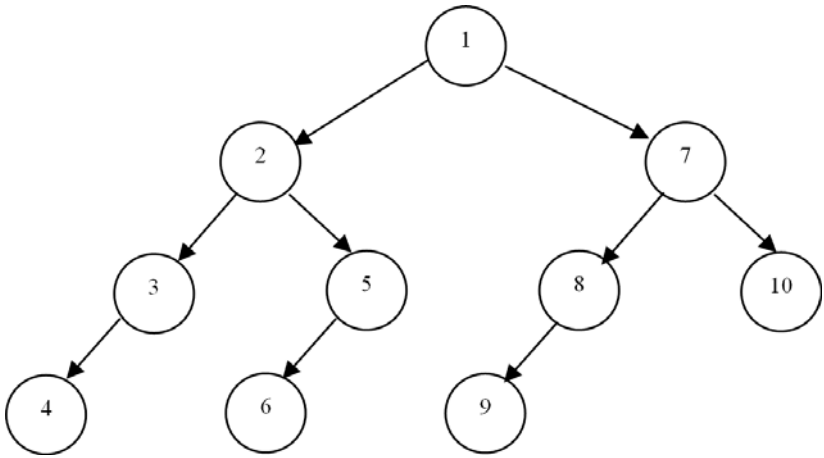


Abbildung 9.18
Der binäre Baum aus Abbildung 9.17 in üblicher Notation.
Die linken und rechten Teilbäume sind jeweils vertauscht.

Als Fazit kann festgestellt werden, dass die auf S. 102 genannte These durch die Analyse des Rollenspiels »Baum minimaler Höhe« gestützt wird. Gerade ein Wechsel zwischen den drei Darstellungsformen bietet interessante Möglichkeiten für die Unterrichtsgestaltung.

Was haben wir getan?
Was tun wir jetzt?
Was sollten wir noch tun?
Georg Christoph Lichtenberg

10 Reflektieren von Informatikunterricht

Informatiklehrerinnen und -lehrer erarbeiten mit ihren Schülerinnen und Schülern ein Thema und möchten anschließend wissen, ob ihr Unterricht die notwendige inhaltliche Bandbreite hatte und ob er den EPA Informatik entsprach. Auch möchten sie nähere Informationen zum Lernstand und -verhalten ihrer Schüler erhalten. Sie laden dazu Testaufgaben aus dem Internet herunter, lassen ihre Schüler die Aufgaben bearbeiten, korrigieren deren Antworten, befragen zusätzlich einzelne Schüler zu ihren Antworten und ziehen Schlussfolgerungen für ihren Unterricht und dessen Fortentwicklung. Dieses Szenarium wurde in den Jahren 2004/2005 an der Universität Jena in einem Forschungsprojekt untersucht. In dem Test ging es um das Thema »Rekursion und Iteration«. Dieses Kapitel soll Lehrerinnen und Lehrer anregen, Tests und Interviews im eigenen Unterricht einzusetzen[34].

Ziel des Forschungsprojekts war die Unterstützung von Lehrerinnen und Lehrern und nicht eine von außen bewertete Kontrolle der Wirksamkeit pädagogischen Handelns. Es ging auch nicht um einen Vergleich mit anderen Klassen, Schulen, Ländern oder Staaten. Vielmehr sollte auf der Grundlage der EPA Informatik ein relativ objektives Instrument bereitgestellt werden, das Lehrerinnen und Lehrern hilft, ihre Arbeit und ihre Schülerinnen und Schüler besser einzuschätzen. Lehrer verfügen zu ihrem eigenen Missbehagen häufig über kein Kriterium für die Beurteilung der Qualität ihrer Arbeit (Herrmann, 2002, S. 116). In dem Forschungsprojekt wurde der Frage nachgegangen, ob das beschriebene Szenarium geeignet ist, dieses Defizit zu verringern. Die Studie sollte nicht zuletzt auch einen Beitrag in der Diskussion zur sinnvollen Arbeit mit Bildungsstandards liefern. Aktueller Anlass für die Untersuchung war die Einführung der EPA Informatik und der Wechsel vom dezentralen zum zentralen Informatikabitur in einigen Ländern.

An der Untersuchung beteiligten sich 57 Informatiklehrerinnen und -lehrer aus zehn Ländern. Die Auswertung lieferte Hinweise darauf, dass extern bereitgestellte Tests nicht nur „im Prinzip" eine gute Sache sind, sondern dass sie eine begründete Chance haben, in der Schulpraxis auch wirklich gewinnbringend eingesetzt zu werden. Es konnte nachgewiesen werden, dass Test und Interviews als Bestandteile des Verfahrens einzeln, aber auch in der Kombination wirkungsvoll sind. Informatiklehrerinnen und -lehrer können einen Blick nach außen – darunter sollen die verbindlichen Vorgaben für den Informatikunterricht verstan-

[34] http://www.informatiktest.de/

den werden – und gleichzeitig einen Blick nach innen – also auf die Wirksamkeit des eigenen Unterrichts und das Lernen der Schülerinnen und Schüler – werfen (Fothe, 2005; Fothe/Ludwig 2008).

Vorbereitung, Durchführung und Auswertung des Tests

Die Kompetenzen zu Rekursion und Iteration lauten (siehe S. 77):
Die Schülerinnen und Schüler
a) erläutern die Grundlagen von Rekursion und Iteration (Vergleichen von Rekursion und Iteration, Äquivalenz von Rekursion und Iteration sowie Prinzip der Abarbeitung eines rekursiven Algorithmus auf einem iterativ arbeitenden Computer),
b) definieren informatische Begriffe auf rekursive Art,
c) verwenden die Syntaxdefinition einer Programmiersprache sachgemäß,
d) analysieren und erläutern exemplarisch Computerprogramme, denen rekursive oder iterative Algorithmen zugrunde liegen, und
e) entwerfen und implementieren solche Computerprogramme.

Im nachfolgenden Test werden alle beschriebenen Kompetenzen mit folgender Ausnahme abgeprüft: Quelltexte von Programmen sind weder zu analysieren noch zu entwickeln. In der folgenden Zusammenstellung ist dargestellt, welche Kompetenzen zu Rekursion und Iteration mit den einzelnen Aufgaben überprüft werden:

Kompetenzen	a)	b)	c)	d)	e)
Aufgaben	1, 2, 6	3, 4, 8	5	1, 6, 7, 8, 9, 10	6, 7

Vor dem Bearbeiten der Testaufgaben stellen die Lehrerinnen und Lehrer ihren Erwartungshorizont für jede Aufgabe auf, und zwar explizit auf ihren Kurs abgestimmt. Die nachfolgend bereitgestellten Musterlösungen und Erläuterungen zu den Aufgaben sollen dazu Anregungen geben. Maßgebend ist jedoch der erteilte Unterricht. Die Bearbeitungszeit für den Test beträgt 45 Minuten. Computer sind als Hilfsmittel nicht zugelassen. Den Lehrerinnen und Lehrern ist es freigestellt, im Bedarfsfall einzelne Aufgaben von der Bearbeitung auszuschließen. Die unterrichtenden Lehrer korrigieren jeweils die Antworten ihrer Schüler. Sie ermitteln für jede Aufgabe die Anzahl an Schülern, die die Aufgabe gelöst, teilweise gelöst bzw. nicht gelöst haben. Auch setzen sie für jede Aufgabe das Korrekturergebnis in Beziehung zum Erwartungshorizont und richten ihr Augenmerk auf Auffälligkeiten wie z. B. typische Fehler. Schließlich ziehen die Lehrerinnen und Lehrer Schlussfolgerungen zum Lernstand ihrer Schülerinnen und Schüler sowie zu ihrem Unterricht und dessen Fortentwicklung. Die Schülerinnen und Schüler erfahren im Vorfeld, dass ein unbenoteter Test zum Thema durchgeführt wird. Eine besondere Vorbereitung auf den Test ist weder nötig

noch erwünscht. Die Schülerantworten sollen nicht benotet werden, da leistungsbeeinträchtigende Einflüsse wie beispielsweise Aufregung und Versagensangst vermieden werden sollen.

Test zum Thema »Rekursion und Iteration«

1. Aufgabe:

Gegeben ist der folgende Algorithmus: In einer Schleife werden nacheinander alle Zeichen eines Textes eingelesen, verschlüsselt und ausgegeben. Ist der Algorithmus ein Beispiel für Rekursion oder Iteration?

Gegeben ist nun der folgende Algorithmus: Aus der Folge von Elementen, die zu sortieren ist, wird ein Element X ausgewählt. Die Elemente, die kleiner sind als das Element X, werden links von ihm angeordnet. Die anderen Elemente werden rechts von ihm angeordnet. Nun wird in gleicher Weise mit der linken und der rechten Teilfolge verfahren – aber nur dann, wenn die jeweilige Teilfolge aus mindestens zwei Elementen besteht. Zur linken Teilfolge gehören alle Elemente vom ersten Element bis zum Element X. Zur rechten Teilfolge gehören alle anderen Elemente. Ist der Algorithmus ein Beispiel für Rekursion oder Iteration?

Musterlösung:

Der erste Algorithmus ist ein Beispiel für Iteration. Der zweite Algorithmus ist ein Beispiel für Rekursion.

Erläuterung: Mit dieser Aufgabe soll überprüft werden, ob die Schüler herausfinden können, inwieweit gegebene Algorithmen iterativ oder rekursiv arbeiten. Beim ersten Algorithmus ist der Begriff »Schleife« wesentlich. Wiederholungsanweisungen sind kennzeichnend für Iteration. Beim zweiten Algorithmus ist zu ermitteln, dass die Kriterien für Rekursion erfüllt sind: Mit der linken und der rechten Teilfolge wird in gleicher Weise verfahren wie mit der gesamten Folge (»rekursiver Abstieg«) und eine Teilfolge muss aus mindestens zwei Elementen bestehen (»rekursiver Abbruch«).

2. Aufgabe:

Beschreiben Sie, was man unter Rekursion und Iteration versteht und welche Zusammenhänge es zwischen ihnen gibt.

Musterlösung:

Rekursion und Iteration sind Formen der Wiederholung. Rekursion kann als Wiederholung strukturgleicher Blöcke durch Schachtelung, Iteration als Wiederholung strukturgleicher Blöcke durch Aneinanderreihung gekennzeichnet werden.

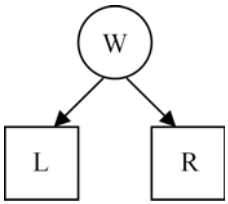
Mit einem Stack (Stapel) kann ein rekursives Programm auf einem iterativ arbeitenden Computer zur Abarbeitung gebracht werden. Das Umsetzen eines iterativ arbeitenden Programms in ein rekursiv arbeitendes führt zur Endrekursion.

Erläuterung: Mit dieser Aufgabe soll ermittelt werden, ob die Schüler die Grundlagen von Rekursion und Iteration kennen und beschreiben können.

3. Aufgabe:

Gegeben ist die folgende rekursive Definition von »binärer Baum«:

Ein binärer Baum besteht aus einem Element (der Wurzel W) und zwei binären Bäumen (dem linken Teilbaum L und rechten Teilbaum R).



Die Definition ist unvollständig. Was fehlt?

Musterlösung:

Es fehlt der Abbruch (leerer binärer Baum).

Erläuterung: Die Schüler sollen erkennen, dass ein Abbruchkriterium (Angabe des Basisfalls) erforderlich ist, da der binäre Baum ansonsten unendlich groß ist.

4. Aufgabe:

Ihre Lehrerin bzw. Ihr Lehrer zeigt Ihnen eine Matroschka. Definieren Sie den Begriff »Matroschka« auf rekursive Art.

Musterlösung:

Eine Matroschka besteht aus einer Puppe, die eine Matroschka enthält, oder es ist die kleinste Matroschka.

Erläuterung: Anhand eines Begriffs, der nichts mit der Informatik zu tun hat, soll festgestellt werden, ob die Schüler in der Lage sind, die ihnen bekannte Methode des rekursiven Definierens eines Begriffs auf einen neuen Kontext zu übertragen. Die Schüler müssen den Aufbau einer rekursiven Definition kennen (vgl. auch die 3. Aufgabe) und in der Lage sein, ihre Lösung geeignet zu verbalisieren.

5. Aufgabe:

Gegeben ist das Zeichen **A**. Das Zeichen **A** kann durch das Zeichen **0** oder durch die Zeichenfolge **1A** ersetzt werden. Zeichenfolgen, die kein **A** enthalten, werden Ergebnis-Zeichenfolgen genannt. Geben Sie vier Ergebnis-Zeichenfolgen an, die nach diesen Regeln gebildet wurden.

Musterlösung:

Zum Beispiel **0**, **10**, **110** und **1111111110**.

Erläuterung: Bei dieser Aufgabe ist eine Syntaxdefinition in verbaler Form gegeben, also nicht durch EBNF oder durch Syntaxdiagramme. Mit dem Beispiel soll festgestellt werden, ob die Schüler das Prinzip eines rekursiven Ersetzungsmechanismus verstanden haben, der häufig in Sprachdefinitionen von Programmiersprachen angewandt wird.

6. Aufgabe:

Nacheinander werden ein roter, grüner, gelber und blauer Kreis mit den angegebenen Radien gezeichnet. Die vier Kreise werden später wieder gelöscht.

Ein roter Kreis mit r = 1 wird gezeichnet ▼ und wieder gelöscht.
Ein grüner Kreis mit r = 2 wird gezeichnet ▼ und wieder gelöscht.
Ein gelber Kreis mit r = 3 wird gezeichnet ▼ und wieder gelöscht.
Ein blauer Kreis mit r = 4 wird gezeichnet und wieder gelöscht.

Das Zeichen ▼ bedeutet, dass die Abarbeitung unterbrochen und in der darunter liegenden Zeile fortgesetzt wird. Die Handlungen werden von einem rekursiv arbeitenden Computerprogramm ausgeführt. In welcher Reihenfolge werden die vier Kreise gelöscht?

Musterlösung:

Die Kreise werden in der Reihenfolge 1. blau, 2. gelb, 3. grün und 4. rot gelöscht.

Erläuterung: Die vier Kreise werden in der Reihenfolge rot – grün – gelb – blau gezeichnet und in der umgekehrten Reihenfolge gelöscht. Die Schüler sollen zeigen, dass sie wissen, wie ein rekursives Computerprogramm zur Abarbeitung gebracht wird. Das Dreieck steht für einen rekursiven Unterprogramm-Aufruf. Beim rekursiven Aufstieg werden die unerledigten Operationen – hier das Löschen – ausgeführt.

7. Aufgabe:

Im Unterricht haben Sie das Spiel »Türme von Hanoi« kennen gelernt. Bei dem Spiel sind Scheiben von einem Feld auf ein anderes in möglichst wenigen Schritten zu versetzen. Auf einem dritten Feld können Scheiben zwischengelagert werden. Beim Versetzen sind zwei Spielregeln zu beachten: Versetze immer nur eine Scheibe und lege eine Scheibe stets auf eine größere oder auf ein leeres Feld. Beschreiben Sie einen Algorithmus zur Lösung des Problems »Türme von Hanoi«.

Musterlösung:

Es wird der rekursive Begriff »Turm« verwendet: Ein Turm besteht aus der größten Scheibe und dem Rest-Turm, oder es ist der leere Turm.

Das Versetzen eines Turmes vom Startfeld zum Zielfeld erfolgt in drei Schritten:

1. Schritt: Der Rest-Turm wird vom Startfeld zum Hilfsfeld versetzt.
2. Schritt: Die größte Scheibe wird vom Startfeld zum Zielfeld versetzt.
3. Schritt: Der Rest-Turm (siehe 1. Schritt) wird nun vom Hilfsfeld zum Zielfeld versetzt.

Die drei Schritte werden rekursiv auf jeden Turm, der zu versetzen ist, angewandt. Bei einem leeren Turm ist nichts zu tun.

Erläuterung: In dieser Aufgabe sollen die Schüler einen Algorithmus wiedergeben, der im Unterricht behandelt wurde (sofern dies der Fall war).

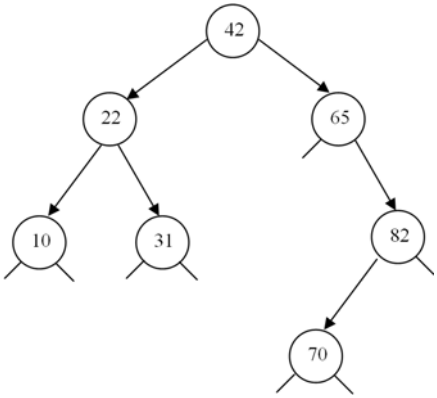
8. Aufgabe:

Methode *inorder*:
Zuerst wird der linke Teilbaum bearbeitet, dann wird die Zahl in der Wurzel ausgegeben und anschließend wird der rechte Teilbaum bearbeitet.

Diese Vorschrift wird auf den gesamten Baum und auf alle Teilbäume angewandt.

Bei einem leeren Teilbaum wird nichts ausgegeben.

a) Gegeben ist der folgende binäre Baum:



Die Methode *inorder* (siehe Kasten) wird auf diesen binären Baum angewandt. In welcher Reihenfolge werden die Zahlen ausgegeben?

b) Bei einem Suchbaum gilt für den gesamten Baum und für alle Teilbäume: Alle Zahlen im jeweiligen linken Teilbaum sind kleiner als die Zahl in der Wurzel. Alle Zahlen im jeweiligen rechten Teilbaum sind größer als die Zahl in der Wurzel.

Ist der in Teilaufgabe a) angegebene binäre Baum ein Suchbaum?

c) Ein beliebiger Suchbaum wird mithilfe der Methode *inorder* ausgegeben. In welcher Reihenfolge werden die Zahlen ausgegeben? Begründen Sie Ihre Antwort.

Musterlösung:

a) Die Ausgabe-Reihenfolge ist 10, 22, 31, 42, 65, 70, 82.
b) Der angegebene binäre Baum ist ein Suchbaum.
c) Die Zahlen werden in sortierter Reihenfolge von der kleinsten bis zur größten Zahl ausgegeben. *Begründung:* Bei *inorder* werden immer zuerst die Zahlen im linken Teilbaum, dann die Zahl in der Wurzel und anschließend die Zahlen im rechten Teilbaum ausgegeben. Bei einem Suchbaum sind immer alle Zahlen im linken Teilbaum kleiner als die Zahl in der Wurzel und alle Zahlen im rechten Teilbaum größer als die Zahl in der Wurzel. Beide Erläuterungen besitzen die gleiche Struktur (*linker Teilbaum – Wurzel – rechter Teilbaum*) und beziehen sich auf den gesamten Baum und auf alle Teilbäume. Daraus ergibt sich die sortierte Reihenfolge.

Erläuterung: Die Schüler sollen darstellen, welche Wirkung die rekursive Methode *inorder*, die in verbaler Form gegeben ist, auf Suchbäume hat. In Teilaufgabe a) wenden die Schüler *inorder* auf einen vorgegebenen binären Baum an. In

Teilaufgabe b) sollen die Schüler anhand der gegebenen Definition von »Suchbaum« entscheiden, ob der in Teilaufgabe a) angegebene binäre Baum ein Suchbaum ist. In Teilaufgabe c) wird dann der Zusammenhang zwischen Suchbaum und *inorder* hergestellt und begründet.

9. Aufgabe:

Gegeben ist das folgende Quadrat:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Das Quadrat wird in den 1., 2., 3. und 4. Teil zerlegt:

2. Teil	1. Teil
3. Teil	4. Teil

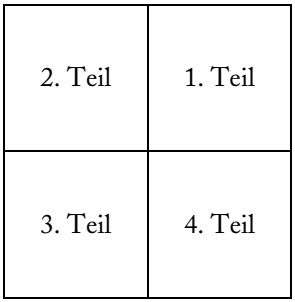
Die vier Teile werden in gleicher Weise wiederum in den 1., 2., 3. und 4. Teil zerlegt. Ausgaben erfolgen stets in der Reihenfolge 1. Teil – 2. Teil – 3. Teil – 4. Teil.

Geben Sie an, in welcher Reihenfolge die Zahlen von 1 bis 16 nach diesen Erläuterungen ausgegeben werden.

10. Aufgabe:

Die 9. Aufgabe enthielt zwei Abbildungen:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



Die Lösung der Aufgabe lautet:

4	3	7	8	2	1	5	6	10	9	13	14	12	11	15	16

Bitte schildern Sie, was Sie sich bei der Lösung der 9. Aufgabe überlegt haben. Falls Sie die 9. Aufgabe falsch gelöst haben, interessiert auch, wie der Fehler zustande kam.

Erläuterung: Die Schüler sollen in der 9. Aufgabe zeigen, dass sie ein vorgegebenes rekursives Zerlegungsschema in einer konkreten Situation anwenden können. Die beschriebene Art der Zerlegung ist eine Möglichkeit zum Speichern von Bildern in Grafikprogrammen.

Die 10. Aufgabe bezieht sich auf die 9. Aufgabe. Sie wird erst bearbeitet, nachdem die Schülerantworten zu den ersten neun Aufgaben eingesammelt wurden. Die Schüler notieren sich ihre Lösung zur 9. Aufgabe zusätzlich auf einem Extrablatt. Sie sollen sich mit ihrer Antwort zur 9. Aufgabe anhand der korrekten Lösung auseinandersetzen. Damit soll ein Reflexionsprozess initiiert und für die Lehrperson transparent werden.

Durchführung der Interviews

Unmittelbar nach der Bearbeitung des Tests werden ein oder drei Schüler von ihrem Lehrer zu ihren Antworten befragt, um mehr über deren konkretes Herangehen an die Bearbeitung der Aufgaben und über ihre kognitiven Strategien und Hilfsmittel zu erfahren (z. B. Visualisierungen, Nutzen von »Eselsbrücken«). Die Lehrer sollen typische Fehlerquellen und Denkfehler erkennen, eventuelle Über- oder Unterforderung ihrer Schülerinnen und Schüler ausmachen und Rückschlüsse auf die didaktisch-methodische Herangehensweise und den Einsatz geeigneter Materialien im eigenen Unterricht ziehen. Wird nur ein Schüler interviewt, sollte dieser bisher durchschnittliche Leistungen gezeigt haben. Falls drei Schüler interviewt werden, sollte jeweils einer von ihnen leistungsstark, durchschnittlich und leistungsschwach sein.

Ein Interview sollte aus schulorganisatorischen Gründen maximal ca. 10 Minuten dauern. An den folgenden Fragen können die Interviews ausgerichtet werden:

- Wie sind Sie vorgegangen, um die wesentlichen Informationen in der Aufgabenstellung herauszufinden (z. B. anhand der 1. Aufgabe)?
- Wie sind Sie ganz genau bei der Lösung einer Aufgabe vorgegangen (z. B. anhand der 4., 5., 6. oder 8. Aufgabe)?
- Bei welchen Aufgaben hatten Sie Schwierigkeiten zu überwinden? Was waren das für Schwierigkeiten? Wie haben Sie diese gemeistert?
- Gab es Aufgaben, die Sie nicht lösen konnten? Woran lag das?
- Welche Aufgaben fielen Ihnen sehr leicht? Woran lag das?
- Wie haben Sie den inhaltlichen Anspruch der Testaufgaben im Vergleich zu bisherigen Arbeiten im Informatikunterricht wahrgenommen?

Was man erfindet, tut man mit Liebe,
was man gelernt hat, mit Sicherheit.
Johann Wolfgang von Goethe

11 Fallbeispiel: Entwickeln von Computerprogrammen

Das Lösen von Aufgaben und Problemen mithilfe von Informatiksystemen kann in den Hauptphasen Entwurf, Implementierung und Reflexion erfolgen[35]*. In diesem Kapitel werden die genannten Phasen charakterisiert und es wird ein Beispiel-Entwurf angegeben, der sich mit einem alltäglichen Thema befasst: dem Kalender. Das Thema ist aus der Perspektive der Informatik interessant. Man denke an das Jahr-2000-Problem (Gruhn/Chavan, 1997) und an die Tatsache, dass aufgrund eines Problems der Chipsoftware bei einer Reihe von Kreditkarten die Jahreszahl 2010 nicht korrekt verarbeitet wird, und setze sich in diesem Zusammenhang mit der Frage auseinander »Wie zuverlässig sind Informatiksysteme?« (siehe Tabelle 1.2, S. 13).*

Man muss erwarten können, dass Schülerinnen und Schüler über das Lösen einer Aufgabe oder eines Problems erst einmal nachdenken, ehe sie mit dem Tippen beginnen. Eine Aufgabe, die von der Mehrheit der Schülerinnen und Schüler sofort und ohne weiteres Nachdenken gelöst werden kann, ist zumindest in dieser Klasse zu leicht. Um möglichen Missverständnissen vorzubeugen: Entwurf bedeutet nicht, von Schülerinnen und Schülern stets einen ausführlichen schriftlichen Entwurf abzuverlangen.

Die Aufgabe lautet:

Entwerfen Sie ein Python-Programm, das für ein beliebiges Jahr aus dem Zeitraum von 1701 bis 2100 einen Jahreskalender erstellt.

Zum *Entwurf* gehört die Analyse der gegebenen Problemstellung. Der Entwurf besteht im Erstellen einer Lösungsstrategie, die auf ein Abbild des problembezogenen Realitätsausschnittes angewandt wird. Der Realitätsausschnitt wird mithilfe von Datenstrukturen modelliert. Die Lösungsstrategie wird in einen Algorithmus umgesetzt, der auf die Datenstrukturen angewandt wird. Der Entwurf lässt sich in drei Phasen gliedern. In der Tabelle 11.1 ist eine Maximalvariante angegeben (Fothe, 2002).

Der Entwurf ist auch eine Grundlage für die Arbeit im Team (Aufgabenzuweisung, Koordinierung, Planung von Schnittstellen).

Die *Implementierung* besteht in der Umsetzung des Entwurfs in ein vom Computer ausführbares Programm, im Testen des Programms und im Ergänzen von Kommentaren. Kommentare unterstützen die Lesbarkeit des Quelltextes. Sie nehmen Bezug auf den Entwurf.

[35] siehe Aufgabenbeispiel 1.2.1 der EPA Informatik

Die *Reflexion* erfolgt in zwei Richtungen. Zum einen wird überprüft, ob das am Anfang formulierte Problem gelöst wurde. Dabei werden die Qualität sowie Einsatzmöglichkeiten und Grenzen des entwickelten Produkts eingeschätzt. Zum anderen werden Erfahrungen herausgearbeitet, die bei der Entwicklung des Programms gewonnen wurden. Aus Fehlern soll man bekanntlich lernen. Die Schülerinnen und Schüler befassen sich also mit metakognitiven Strategien der Problemlösung. Dies entspricht den EPA Informatik, in denen es bei den fachlichen und methodischen Kompetenzen heißt (KMK, 2004, S. 8): »Die Prüflinge […] sind in der Lage, die eigene Arbeit und die Arbeit Anderer kritisch zu reflektieren […].«

Häufig ist das Erarbeiten eines Entwurfs der eigentliche schöpferische Prozess. Daher sollten die Schülerinnen und Schüler zuerst ihre Entwürfe in der Klasse vor- und zur Diskussion stellen, ehe es ans Umsetzen geht.

Beschreibungsphase
▪ Beschreibung der Aufgaben, die zu lösen sind (Wiedergabe der zu lösenden Probleme, Verstehen der Probleme, Angabe der Hilfsmittel) ▪ Konkretisierungen und begründete Festlegungen (eigene Abgrenzungen der Problematik, eigene Festlegungen mit Begründung, Vorgabe von Möglichkeiten und Grenzen)
Strukturierungsphase
▪ Welche Module sind vorgesehen? Was sollen sie leisten? ▪ Welche grundlegenden Datenstrukturen sind zur Lösung welcher Probleme vorgesehen? ▪ Welche Unterprogramme sind vorgesehen? Welche Aufgabe haben sie zu erfüllen? (verbale Beschreibung)
Algorithmische Phase
▪ Für jedes Unterprogramm sind dessen Name und die formalen Parameter anzugeben. ▪ Es ist zu beschreiben, wie das Unterprogramm die Aufgabe lösen soll. ▪ Für wesentliche Teile ist der Algorithmus detailliert anzugeben (z. B. als verbale Beschreibung oder Struktogramm).

Tabelle 11.1:
Phasen eines Programmentwurfs aus schulischer Sicht.

Kommen wir nun zum Beispiel-Entwurf zum Thema »Jahreskalender«. In der *Beschreibungsphase* machen sich die Schülerinnen und Schüler die Kernfunktionalität des Produkts klar. Verschiedene Varianten werden diskutiert, so auch die folgende Anordnung der Monate, die bei professionellen Kalendern mitunter vorliegt:

Januar	Februar	März
April	Mai	Juni
Juli	August	September
Oktober	November	Dezember

Diese Anordnung wird jedoch verworfen, da die zeilenweise Ausgabe auf dem Monitor oder auf dem Drucker wegen der Verzahnung von jeweils drei Monaten schwer zu realisieren ist. Vielmehr wird ein einfacher Aufbau des Kalenders bevorzugt, bei dem die Monate untereinander angeordnet sind:

Januar
Februar
März
April
Mai
Juni
Juli
August
September
Oktober
November
Dezember

Anschließend wird die Struktur des Jahreskalenders genauer betrachtet. Ein Beispiel ist der Kalender für 2011. Er beginnt wie folgt:

```
Januar
Mo      Di      Mi      Do      Fr      Sa      So
                                         1       2
 3       4       5       6       7       8       9
10      11      12      13      14      15      16
17      18      19      20      21      22      23
24      25      26      27      28      29      30
31

Februar
Mo      Di      Mi      Do      Fr      Sa      So
         1       2       3       4       5       6
 7       8       9      10      ...
```

Die Schülerinnen und Schüler recherchieren zum Thema »Kalender« und können dabei u. a. erfahren: Im Jahr 1582 wurde von Papst Gregor XIII. eine Kalenderreform angeordnet. Nach den Festlegungen der Reform folgte auf den 4. Oktober 1582 der 15. Oktober. Es wurden also die Bezeichnungen von 10 Tagen ausgelassen[36]. Außerdem wurde eine neue Regelung für die Festlegung eines Schaltjahres eingeführt. Die Regelung ist auch heute noch gültig. Die meisten katholischen Länder Europas führten den gregorianischen Kalender noch 1582 oder in den darauffolgenden Jahren ein, die protestantischen Länder Deutschlands erst im Jahr 1700. Auf den 18. Februar folgte dort unmittelbar der 1. März 1700. Für die einheitliche Regelung im Jahr 1700 machte sich der Jenaer Mathematikprofessor Erhard Weigel (1625–1699) stark (zur Geschichte der Mathematik in Jena siehe z. B. Fothe/Zimmermann, 2009). Der gregorianische Kalender wird heute weltweit eingesetzt.

In der *Strukturierungsphase* wird festgelegt, dass ein Datum als Tripel (jahr, monat, tag) modelliert werden soll. Drei Funktionen werden als grundlegend herausgearbeitet:

1. Funktion, die ermittelt, ob ein Jahr ein Schaltjahr ist,
2. Funktion, die die Monatslänge ermittelt, und
3. Funktion, die den Wochentag berechnet.

Die drei Funktionen werden in der *algorithmischen Phase* genauer betrachtet. Die logische Funktion schaltjahr(jahr) ermittelt, ob ein Jahr ein Schaltjahr ist. Ein Jahr ist ein Schaltjahr, wenn die Jahreszahl ohne Rest durch 400 teilbar ist oder wenn die Jahreszahl ohne Rest durch 4 und nicht durch 100 teilbar ist.

- Beispiele für Schaltjahre: 1996, 2000, 2012
- Beispiele für normale Jahre: 1997, 2014, 2100

Die Funktion anzahl(jahr, monat) ermittelt die Monatslänge.

- Die Monate 1, 3, 5, 7, 8, 10 und 12 haben 31 Tage.
- Die Monate 4, 6, 9 und 11 haben 30 Tage.
- Der Monat 2 hat 29 Tage, wenn er in einem Schaltjahr liegt. Ansonsten hat er 28 Tage.

Die Funktion wochentag(jahr, monat, tag) berechnet die Nummer des Wochentags für ein gegebenes Datum. Die Nummer 0 steht für Sonntag, 1 für Montag, 2 für Dienstag usw. Der Wochentag für das Datum (jahr, monat, tag) lässt sich auf einfache Art mit der Kalenderformel von Jacobsthal ermitteln. Sie lautet in Python:

```
d = tag + k + j + j / 4 - 2 * (c % 4)
```

[36] Mitunter liest man durchaus missverständlich, dass 10 Tage ausgelassen wurden.

monat	1	2	3	4	5	6	7	8	9	10	11	12
k	6(5)	2(1)	2	5	0	3	5	1	4	6	2	4

Tabelle 11.2:
Monatskennzahlen k
(im Januar und Februar ist bei Schaltjahren die Zahl in Klammern zu nehmen).

j ist die Jahreszahl innerhalb eines Jahrhunderts und c ist die Anzahl der vollen (vergangenen) Jahrhunderte. Es gilt also j = jahr % 100 und c = jahr / 100. Der Rest der Division des Zwischenergebnisses d durch 7 ergibt die Nummer des Wochentags. Die sogenannten Monatskennzahlen k sind in der Tabelle 11.2 zusammengestellt.

Beispiel (9. November 1918):

```
d = tag + k + j + j / 4 - 2 * (c % 4)
d = 9 + 2 + 18 + 18 / 4 - 2 * (19 % 4)
d = 27
```

27 lässt bei der Division durch 7 den Rest 6. Die deutsche Republik wurde also an einem Samstag ausgerufen. Auch der Revolutionstag 18. März 1848 war ein Samstag[37].

Die o. g. Funktionen werden beim Erstellen des Jahreskalenders genutzt. In einer Wiederholungsanweisung werden die Monate bearbeitet. Für jeden Monat wird die Nummer des Wochentags des Monatsersten ermittelt. Aus dieser Nummer ergibt sich, wie weit vor der ersten Ausgabe einzurücken ist. Dann werden nacheinander die Zahlen von 1 bis zum Monatsletzten ausgegeben und jedes Mal, wenn ein Sonntag ausgegeben wurde, wird eine neue Zeile begonnen.

Den Schülerinnen und Schülern soll deutlich werden: Die Qualität des Entwurfs bestimmt die Qualität des Produkts. Schüler, die davon nicht zu überzeugen sind, verzichten gern auf genaue Vorüberlegungen und man kann ein Programm zum Jahreskalender erwarten, das keine Unterprogramme besitzt und das mit vertretbarem Aufwand nicht zu durchschauen ist. Immer wieder tauchen die Zahlen 31, 30, 29 und 28 und die Überprüfung auf Schaltjahr auf, das Ganze macht mitunter einen recht wirren Eindruck und ist fehlerhaft.

Im Sinne eines evolutionären Herangehens können leistungsstarke Schülerinnen und Schüler ihr Programm so erweitern, dass Feiertage im Jahreskalender hervorgehoben werden. Das kann für verschiedene Religionen geschehen, was garantiert interessant ist. Für die beweglichen christlichen Feiertage gilt die Devise: Wer das Osterdatum hat, hat alles. So ist z. B. Aschermittwoch der 46. Tag

[37] Man hüte sich jedoch vor unzulässigen Verallgemeinerungen. So war der 25. Februar 1926 ein Donnerstag.
http://www.finanzamt-bernkastel-wittlich.de/wir_ueber_uns/chro_fa_bk.htm

vor Ostersonntag. Zwischen dem Oster- und dem Pfingstsonntag liegen sieben Wochen. Zur Berechnung des Osterdatums kann man den Algorithmus des deutschen Mathematikers, Astronomen und Physikers Carl Friedrich Gauß (1777–1855) verwenden. Der Algorithmus lautet in Python (der Algorithmus wird hier nicht näher erläutert):

```
# Die Variable x enthaelt als Wert eine Jahreszahl.
k = x / 100
m = 15 + (3 * k + 3) / 4 - (8 * k + 13) / 25
s = 2 - (3 * k + 3) / 4
a = x % 19
d = (19 * a + m) % 30
r = d / 29 + (d / 28 - d / 29) * (a / 11)
og = 21 + d - r
sz = 7 - (x + x / 4 + s) % 7
oe = 7 - (og - sz) % 7
os = og + oe
```

Das Datum des Ostersonntags ergibt sich aus der Variablen os wie folgt:

Wenn $1 \leq$ os ≤ 31 gilt, so ist es der os. März.
Wenn os > 31 gilt, so ist es der (os $- 31$). April.

Eine weitergehende Aufgabe kann im Erstellen einer Osterstatistik bestehen. Konkret: Wie häufig fällt der Ostersonntag auf ein bestimmtes Datum? Als Zeitraum kann z. B. 1701–2100 genommen werden.

An einem Programm zur Berechnung des Ostersonntags kann das Testen von Computerprogrammen thematisiert werden. Die Schülerinnen und Schüler erkennen dabei, dass das Testen eines Programms wichtig ist, dass dadurch dessen vollständige Korrektheit jedoch nicht bewiesen werden kann. Testen stellt letztlich nur die Anwesenheit von Fehlern fest. Für den Anfangsunterricht hat es sich bewährt, den Schülern Beispiele, mit denen sie ihre Programme testen sollen, vorzugeben. Später konstruieren die Schüler Testbeispiele selbst. Grafische Ausgaben lassen sich häufig leichter auf Korrektheit überprüfen als z. B. Folgen von Zahlen. Für das Testen muss man sich mitunter eine regelrechte Strategie einfallen lassen. Ein Beispiel aus der eigenen Tätigkeit soll angegeben werden: Ich erarbeitete eine Python-Funktion zum Berechnen des Datums des Ostersonntags. Der Test mit einigen Testbeispielen ließ ein fehlerfreies Programm vermuten. Mir kamen jedoch Zweifel. Daher setzte ich zusätzlich eine ganz andere Formelgruppe als Python-Funktion um und es ergaben sich für fünf Jahre unterschiedliche Ausgaben (siehe Abbildung 11.1). Durch genaue Analyse der Quelltexte beider Funktionen fand ich an einer Stelle einen Tippfehler. Nach dessen Korrektur lieferten beide Funktionen im gesamten geprüften Zeitraum jeweils

das gleiche Datum des Ostersonntags. Nun ging ich davon aus, dass beide Funktionen korrekt sind.

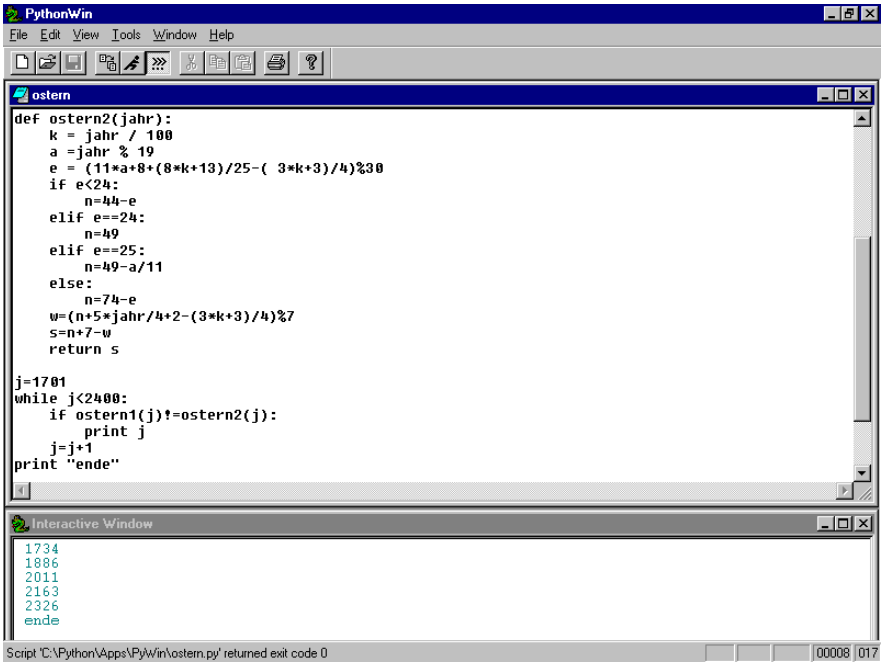


Abbildung 11.1:
Test mithilfe der Osterregel nach dem »Immerwährenden Kalender« (Graßl, 1998).

Das Modularisieren ist in den EPA Informatik bei den fachlichen und methodischen Kompetenzen aufgeführt. Beim Modularisieren erfolgt ein Strukturieren und Zerlegen in Teilprobleme. Eine interessante Aufgabe für den Informatikunterricht in der Sekundarstufe II ist das Entwerfen und Realisieren von Modulen und deren anschließende Nutzung.

Ein Beispiel ist das Modul DatumOperationen mit der folgenden Schnittstelle (in Oberon-2):

- Datentyp:
  ```
  TYPE Datum = RECORD
                  jahr, monat, tag: INTEGER;
                  END;
  ```

- Operationen:

```
PROCEDURE Abstand (a, b: Datum): LONGINT;
PROCEDURE Existenz (d: Datum): BOOLEAN;
PROCEDURE IstGleich (a, b: Datum): BOOLEAN;
PROCEDURE LiegtVor (a, b: Datum): BOOLEAN;
PROCEDURE Nachfolger (VAR d: Datum);
PROCEDURE Ostersonntag (jahr: INTEGER; VAR d: Datum);
PROCEDURE Schaltjahr (jahr: INTEGER): BOOLEAN;
PROCEDURE Summe (a: Datum; anzahl: LONGINT; VAR b: Datum);
PROCEDURE TageImJahr (jahr: INTEGER): INTEGER;
PROCEDURE TageImMonat (jahr, monat: INTEGER): INTEGER;
PROCEDURE Vorgaenger (VAR d: Datum);
PROCEDURE Wochentag (d: Datum): INTEGER;
```

Das Realisieren der Operationen kann im Unterricht arbeitsteilig erfolgen. Einige Beispiele werden nachfolgend angegeben.

Die Funktionsprozedur LiegtVor stellt fest, ob das Datum a vor dem Datum b liegt. Sind die Jahre unterschiedlich, so erfolgt die Feststellung nur unter Heranziehung der beiden Jahreszahlen. Liegen beide Kalenderdaten im gleichen Jahr, so müssen die Monate betrachtet werden. Liegen beide Kalenderdaten sogar im gleichen Monat, so müssen die Tage betrachtet werden.

```
PROCEDURE LiegtVor*(a, b: Datum): BOOLEAN;
BEGIN
  IF a.jahr < b.jahr THEN
    RETURN TRUE
  ELSIF a.jahr > b.jahr THEN
    RETURN FALSE
  ELSIF a.monat < b.monat THEN
    RETURN TRUE
  ELSIF a.monat > b.monat THEN
    RETURN FALSE
  ELSE RETURN a.tag < b.tag
  END
END LiegtVor;
```

Die Prozedur Nachfolger ermittelt das Datum des nächsten Tages. Dabei werden drei Fälle unterschieden:

- Es handelt sich nicht um einen Monatsletzten.
- Es handelt sich um einen Monatsletzten, nicht jedoch um den 31. Dezember.
- Es handelt sich um den 31. Dezember.

```
PROCEDURE Nachfolger*(VAR d: Datum);
VAR r: Datum;
BEGIN
  IF d.tag < TageImMonat(d.jahr, d.monat) THEN
    r.jahr := d.jahr;
    r.monat := d.monat;
    r.tag := d.tag + 1
  ELSIF d.monat < 12 THEN
    r.jahr := d.jahr;
    r.monat := d.monat + 1;
    r.tag := 1
  ELSE
    r.jahr := d.jahr + 1;
    r.monat := 1;
    r.tag := 1
  END;
  d := r
END Nachfolger;
```

Die Prozedur Summe ermittelt, ausgehend von einem bestimmten Datum, das Datum eines ganz bestimmten nachfolgenden Tages (z. B. des 1000. Tages). Bei der Realisierung wird die Prozedur Nachfolger (bzw. die Prozedur Vorgaenger) genutzt, was zu einem einfachen Algorithmus führt.

```
PROCEDURE Summe*(a: Datum; anzahl: LONGINT; VAR b: Datum);
VAR z: LONGINT;
BEGIN
  IF anzahl > 0 THEN
    FOR z := 1 TO anzahl DO Nachfolger(a) END
  ELSIF anzahl < 0 THEN
    FOR z := 1 TO -anzahl DO Vorgaenger(a) END
  END;
  b := a
END Summe;
```

Die Funktionsprozedur Abstand ermittelt, wie viele Tage es vom Datum a zum Datum b sind. Möglich wäre auch hier die Verwendung der Prozeduren Nachfolger bzw. Vorgaenger, was eine einfach aufgebaute Funktionsprozedur erwarten ließe. Ein komplizierterer Algorithmus ist der Folgende, der gerade bei weit auseinanderliegenden Daten vergleichsweise schnell arbeitet. Es werden drei Fälle unterschieden:

- Die beiden Kalenderdaten liegen in unterschiedlichen Jahren.
- Die beiden Kalenderdaten liegen im gleichen Jahr, jedoch in unterschiedlichen Monaten.
- Die beiden Kalenderdaten liegen sogar im gleichen Monat.

Sehen wir uns z. B. den ersten Fall genauer an: Zuerst wird die Anzahl an Tagen vom Datum a bis zum Monatsletzten ermittelt. Dann werden die vollständigen Monate bis zum Jahresende addiert. Es werden die Jahre bis zum Vorgängerjahr vom Datum b addiert, dann die vollständigen Monate bis zum Vorgängermonat vom Datum b und abschließend die Tage im Zielmonat.

```
PROCEDURE Abstand*(a, b: Datum): LONGINT;
(* Voraussetzung: a liegt vor b *)
VAR i: INTEGER;
      anzahl: LONGINT;
BEGIN
  IF a.jahr # b.jahr THEN
     anzahl := TageImMonat(a.jahr, a.monat) – a.tag;
     FOR i := a.monat + 1 TO 12 DO
        anzahl := anzahl + TageImMonat(a.jahr, i)
     END;
     FOR i := a.jahr + 1 TO b.jahr – 1 DO
        anzahl := anzahl + TageImJahr(i)
     END;
     FOR i := 1 TO b.monat – 1 DO
        anzahl := anzahl + TageImMonat(b.jahr, i)
     END;
     anzahl := anzahl + b.tag
  ELSIF a.monat # b.monat THEN
     anzahl := TageImMonat(a.jahr, a.monat) – a.tag;
     FOR i := a.monat + 1 TO b.monat – 1 DO
        anzahl := anzahl + TageImMonat(b.jahr, i)
     END;
     anzahl := anzahl + b.tag
  ELSE anzahl := b.tag – a.tag
  END;
  RETURN anzahl
END Abstand;
```

Das Modul kann z. B. für das Planen von Baumaßnahmen eingesetzt werden. Die wichtigste Frage: Wann muss ich anfangen, um pünktlich fertig zu werden?

Die Sprache ist der Papagei des Gedankens,
und ein schwer gelehriger, nichts weiter.
Friedrich Hebbel

12 Fallbeispiel: Sich selbst aus dem Sumpf ziehen

In diesem Kapitel geht es um Aspekte formaler Sprachen, um Basiskonzepte der Informatik und um das Lösen einer Komplexaufgabe mithilfe eines Informatiksystems. Der Prozessor eines Computers »versteht« nur Maschinensprache – also Folgen von Nullen und Einsen. Programmiersprachen wie Python oder Oberon versteht er nicht. Trotzdem kann man in Python und Oberon programmieren – und solche Programme werden vom Computer auch ausgeführt. Das ist erstaunlich und man kann sich fragen: Wie geht das?

Die Lösungen heißen Assembler, Compiler und Interpreter. Diese Werkzeuge übersetzen Assembler-Programme und Programme, die in einer in einer höheren Programmiersprache vorliegen, in Maschinensprache. Die erzeugten Maschinenprogramme sind auf dem Computer »lauffähig«. Bei einem Compiler liegen die Phasen der Programm-Übersetzung und der Programm-Ausführung nacheinander. Man denke an einen Buchübersetzer. Bei einem Interpreter sind die Phasen der Programm-Übersetzung und der Programm-Ausführung verzahnt. Man denke an einen Simultan-Dolmetscher. Das Übersetzen aus einer Quell- in eine Zielsprache lässt sich mithilfe von T-Diagrammen darstellen (siehe Abbildung 12.1).

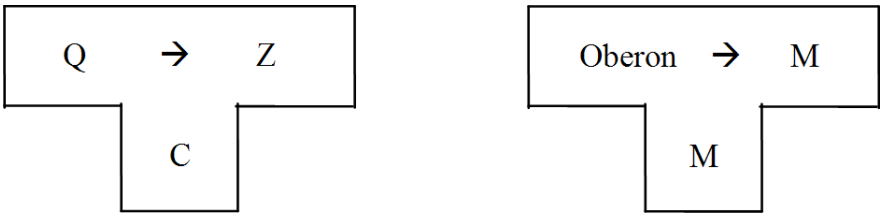


Abbildung 12.1:
Das linke T-Diagramm stellt die Arbeitsweise eines Compilers allgemein dar. Q ist die Quellsprache, Z die Zielsprache und C die Sprache, in der der Compiler vorliegt. Das rechte T-Diagramm stellt die Arbeitsweise eines Oberon-Compilers dar. Oberon wird in die Maschinensprache M übersetzt. Der Compiler selbst liegt in Maschinensprache M vor; er ist auf dem Computer also lauffähig.
Der Name dieser Diagramme ergibt sich aus deren Form.

In diesem Kapitel wird am Beispiel der Ausdrücke gezeigt, wie das Übersetzen aus einer formalen Sprache in eine andere Sprache im Informatikunterricht thematisiert werden kann. Die gegebenen Ausdrücke liegen in der Infix-Notation vor (siehe S. 86).

Mögliche Bestandteile der Ausdrücke:

- die Variablenbezeichner von a bis z,
- die Additionsoperatoren + und −,
- die Multiplikationsoperatoren * und /,
- der Operator für das Potenzieren ↑ (oder ^) und
- runde Klammern.

Zugelassen sind keine Zahlenkonstanten, keine einstelligen Operatoren, keine Funktionsaufrufe, keine logischen Operatoren und keine Variablenbezeichner, die aus mehr als einem Kleinbuchstaben bestehen. Des Weiteren wird festgelegt, dass ein Ausdruck bei Operatoren gleicher Priorität von links nach rechts ausgewertet wird. Der Ausdruck a * b * c ist also gleichwertig zu (a * b) * c, der Ausdruck a ↑ b ↑ c gleichwertig[38] zu (a ↑ b) ↑ c. Das in diesem Kapitel beschriebene Verfahren ist einfach, weil Baumstrukturen vermieden werden, und es ist dennoch lehrreich, weil Ausdrücke eine komplexe Syntax besitzen (siehe S. 87 f.).

Wir nehmen nachfolgend folgende Aufgabe in Angriff:

> Gegeben sind eine Zeichenkette mit einem Ausdruck in Infix-Notation und die Werte der in dem Ausdruck vorkommenden Variablen. Entwerfen Sie ein Computerprogramm, das den Wert des Ausdrucks berechnet.

Die Aufgabe wird in drei Schritten bearbeitet. Im ersten Schritt wird der Ausdruck aus der Infix-Notation in die Postfix-Notation überführt. Dabei wird die *Begrenzerpaarmethode mit Gewichtsvektor und einem Keller nach Dijkstra*[39] angewandt (siehe Kerner, 1973, S. 455–459). Das Ergebnis des zweiten Schrittes ist ein Programm in einer einfachen Assembler-Sprache. Im dritten Schritt wird das Assembler-Programm zur Abarbeitung gebracht.

1. Schritt: Überführen eines Ausdrucks aus der Infix-Notation in die Postfix-Notation

Die folgenden Regeln beschreiben die Funktionalität des Kellers:

1. Das oberste Symbol aus dem Keller wird zum Ausgang überführt, wenn sein Gewicht größer oder gleich dem Gewicht des neuen Symbols ist (zu den Gewichten siehe Tabelle 12.1).
2. Die schließende Klammer wird nicht gekellert.

[38] Auch wenn in vielen Programmiersprachen a ↑ (b ↑ c) gerechnet wird.
[39] Edsger Wybe Dijkstra (1930–2002): niederländischer Informatiker.

3. Die öffnende Klammer wird stets gekellert und beginnt ein neues Niveau der Gewichte. Sie wird durch die schließende Klammer im Keller gelöscht.
4. Variablenbezeichner werden sofort zum Ausgang überführt.

Zeichen	*Gewicht*
(	0
) ;	1
+ −	2
* /	3
↑	4

Tabelle 12.1:
Gewichte, die allen Zeichen außer den Variablenbezeichnern zugeordnet werden.

Die Tabellen 12.2 und 12.3 enthalten zwei Beispiele, die die eingesetzte Methode veranschaulichen sollen. Die gegebenen Ausdrücke werden von links nach rechts durchlaufen. Das Semikolon ist ein Abschlusszeichen.

Zeichen	*Wirkung*	*Keller*	*Ausgang*
x	wird sofort zum Ausgang überführt		x
+	wird gekellert (Gewicht 2)	+	x
r	wird sofort zum Ausgang überführt	+	x r
*	wird gekellert (Gewicht 3)	* +	x r
s	wird sofort zum Ausgang überführt	* +	x r s
;	erst wird der »*« und dann das »+« entkellert und zum Ausgang überführt, das »;« wird zum Ausgang überführt		x r s * + ;

Tabelle 12.2:
Bearbeitung des Beispiels x + r * s ;
Das Ergebnis ist: x r s * + ;

Zeichen	*Wirkung*	*Keller*	*Ausgang*
a	wird sofort zum Ausgang überführt		a
+	wird gekellert (Gewicht 2)	+	a
b	wird sofort zum Ausgang überführt	+	a b
*	wird gekellert (Gewicht 3)	* +	a b
(	wird gekellert (Gewicht 0) und beginnt ein neues Niveau der Gewichte	(* +	a b
c	wird sofort zum Ausgang überführt	(* +	a b c
–	wird gekellert (Gewicht 2)	– (* +	a b c
d	wird sofort zum Ausgang überführt	– (* +	a b c d
)	erst wird das »–« entkellert und zum Ausgang überführt, dann wird die Klammer »(« im Keller gelöscht	* +	a b c d –
↑	wird gekellert (Gewicht 4)	↑ * +	a b c d –
e	wird sofort zum Ausgang überführt	↑ * +	a b c d – e
–	↑ * + werden entkellert und zum Ausgang überführt, dann wird das »–« gekellert (Gewicht 2)	–	a b c d – e ↑ * +
f	wird sofort zum Ausgang überführt	–	a b c d – e ↑ * + f
;	das »–« wird entkellert und zum Ausgang überführt, das »;« wird zum Ausgang überführt		a b c d – e ↑ * + f – ;

Tabelle 12.3:
Bearbeitung des Beispiels a + b * (c – d) ↑ e – f ;
Das Ergebnis ist: a b c d – e ↑ * + f – ;

Das folgende Programmstück in Python realisiert den beschriebenen Keller. Dazu einige Erläuterungen: Die Funktion kellern(zeichen) fügt das Zeichen dem Keller an der Kellerspitze hinzu. Die Funktion entkellern() holt und entfernt das Zeichen, das an der Kellerspitze steht. Ist der Keller leer, so wird das sogenannte Kellersymbol & zurückgegeben. Die Funktion gewicht(zeichen) liefert das Gewicht des Zeichens (siehe Tabelle 12.1). Das Kellersymbol & besitzt

das Gewicht −1. Wesentliche Variablen sind eingabe, ausgabe und keller. Die Variable keller ist eine globale Variable.

```
eingabe = raw_input("Eingabe: ")
ausgabe = ""
keller = ""
for neuesZeichen in eingabe:
    if "a" <= neuesZeichen <= "z":
        ausgabe = ausgabe + neuesZeichen
    elif neuesZeichen == "(":
        kellern("(")
    else:
        altesZeichen = entkellern()
        while gewicht(altesZeichen) >= gewicht(neuesZeichen):
            ausgabe = ausgabe + altesZeichen
            altesZeichen = entkellern()
        if (altesZeichen != "&") and \
        not((altesZeichen == "(") and (neuesZeichen == ")")):
            kellern(altesZeichen)
        if neuesZeichen != ")":
            kellern(neuesZeichen)
ausgabe = ausgabe + ";"
print ausgabe
```

2. Schritt: Überführen eines Ausdrucks aus der Postfix-Notation in eine Assembler-Sprache

Befehl	*Semantik*
MOVE Variable, Register	Register := Variable
ADD RegisterB, RegisterA	RegisterA := RegisterA + RegisterB
SUB RegisterB, RegisterA	RegisterA := RegisterA − RegisterB
MUL RegisterB, RegisterA	RegisterA := RegisterA * RegisterB
DIV RegisterB, RegisterA	RegisterA := RegisterA / RegisterB
POT RegisterB, RegisterA	RegisterA := RegisterA ↑ RegisterB

Tabelle 12.4:
Assembler-Befehle.

Zu den Befehlen der Assembler-Sprache siehe Tabelle 12.4. Es werden nur Integer-Variablen verwendet. Der Operator / führt die ganzzahlige Division aus. Es gibt die zehn Datenregister D0, D1, D2, D3, D4, D5, D6, D7, D8 und D9.

Das Assembler-Programm wird mit folgender Vorschrift erzeugt:

Der Variablen k wird der Wert 0 zugewiesen.
Wiederhole die folgenden Schritte, bis alle Zeichen des Ausdrucks bearbeitet sind:

- Hole das nächste Zeichen.
- Handelt es sich um eine Variable v, so lautet der Befehl MOVE v, D k. Erhöhe den Wert der Variablen k um 1.
- Handelt es sich um einen Operator für die Operation OPE, so lautet der Befehl OPE D k–1, D k–2. Verringere den Wert der Variablen k um 1.

OPE steht stellvertretend für ADD, SUB, MUL, DIV bzw. POT. Am Ende besitzt die Variable k, wenn alles korrekt verlaufen ist, den Wert 1. In den Tabellen 12.5 und 12.6 werden die beiden Beispiele aus dem 1. Schritt fortgeführt.

k	*Zeichen*	*Art*	*Befehl*
0	x	Variable	MOVE x,D0
1	r	Variable	MOVE r,D1
2	s	Variable	MOVE s,D2
3	*	Operator	MUL D2,D1
2	+	Operator	ADD D1,D0
1	;	Abschlusszeichen	

Tabelle 12.5:
Bearbeitung des Beispiels x r s * + ;

Das Assembler-Programm dazu lautet:

```
MOVE x,D0
MOVE r,D1
MOVE s,D2
MUL D2,D1
ADD D1,D0
```

k	*Zeichen*	*Art*	*Befehl*
0	a	Variable	MOVE a,D0
1	b	Variable	MOVE b,D1
2	c	Variable	MOVE c,D2
3	d	Variable	MOVE d,D3
4	–	Operator	SUB D3,D2
3	e	Variable	MOVE e,D3
4	↑	Operator	POT D3,D2
3	*	Operator	MUL D2,D1
2	+	Operator	ADD D1,D0
1	f	Variable	MOVE f,D1
2	–	Operator	SUB D1,D0
1	;	Abschlusszeichen	

Tabelle 12.6:
Bearbeitung des Beispiels a b c d – e ↑ * + f – ;

Das Assembler-Programm dazu lautet:

```
MOVE a,D0
MOVE b,D1
MOVE c,D2
MOVE d,D3
SUB D3,D2
MOVE e,D3
POT D3,D2
MUL D2,D1
ADD D1,D0
MOVE f,D1
SUB D1,D0
```

Die mithilfe der angegeben Vorschrift erzeugten Programme gehen sparsam mit den Datenregistern um. Ein Assembler-Programm wird in einer Zeichenkette z gespeichert. Für das erste Assembler-Programm ergibt sich unter Weglassung aller Leerzeichen:

```
z = "MOVEx,D0;MOVEr,D1;MOVEs,D2;MULD2,D1;ADDD1,D0;"
```

3. Schritt: Abarbeiten des Assembler-Programms

Die Zeichenkette mit dem Assembler-Programm wird von vorn nach hinten durchlaufen und die entsprechenden Befehle werden ausgeführt. Für das Erkennen eines Befehls reicht meistens das erste Zeichen aus (A, S, D bzw. P); nur im Falle von MOVE und MUL sind die ersten beiden Zeichen zu betrachten (MO bzw. MU). Nach dem Erkennen eines Befehls wird die relevante Information aus der Zeichenkette herausgelesen. Bei MOVE steht die Variable 4 Zeichen hinter dem M und das Datenregister 7 Zeichen hinter dem M. Bei den anderen Befehlen ist das erste Datenregister 4 Zeichen hinter dem ersten Buchstaben angegeben; aus diesem ergibt sich auch das zweite Datenregister. Möglich ist auch das Verwenden der Funktion suche1(text, muster) bzw. suche2(text, muster) aus dem Kapitel 5. Die Werte der verwendeten Variablen werden am einfachsten als Wertzuweisungen angegeben. Konsequenter wäre das Speichern der Variablenbezeichner mit ihren Werten in einer Zeichenkette, die zu Beginn der Programmabarbeitung ausgewertet wird. Nach dem Abarbeiten des Assembler-Programms enthält das Datenregister D0 den gesuchten Wert des Ausdrucks.

> Jedermann, der noch bedingungslos an unbegrenzte Möglichkeiten der Computer glaubt, ist ebenso ungebildet wie ein Winkeldreiteiler oder Würfelverdoppler.
>
> *Immo O. Kerner*

13 Fallbeispiel: Alan Turing und die Endlosschleifen

Prinzipielle Grenzen der Berechenbarkeit sind nach den EPA Informatik ein fachlicher Inhalt der Abiturprüfung. Das Thematisieren des „Geht oder geht nicht" im Informatikunterricht wird in diesem Kapitel beschrieben (siehe auch den theoretischen Aspekt in der Tabelle 1.2, S. 13). Es geht um das Weltbild von Lernenden zu einem Gerät, das sie praktisch in jedem Lebensbereich umgibt, das vielen von ihnen wichtig ist und dessen Leistungen explosionsartig wachsen. Ziel ist eine fachlich begründete Positionsbestimmung zwischen den Polen »Technikfeindlichkeit« und »Computergläubigkeit«. Auf wissenschaftlicher Grundlage soll der Auffassung entgegengetreten werden, dass sich mit einem hinreichend leistungsfähigen Computer eigentlich alle Probleme lösen lassen, sofern man sie präzise beschreiben kann.

Das Modell »Turing-Maschine« und dessen Einordnung

Den Schülerinnen und Schülern soll deutlich werden, dass es Aufgaben gibt, die sich einer Lösung mithilfe von Computern entziehen. Zwei Negativ-Beispiele mit praktischem Hintergrund sollen genannt werden: Wünschenswert wäre ein Computerprogramm, das gegebene Quelltexte auf das Vorhandensein von Endlosschleifen (»unendliche Schleifen«) überprüft. Von großem Interesse wäre auch ein Computerprogramm, das für zwei *beliebige* Sprachbeschreibungen entscheidet, ob sie die gleiche Sprache – also die gleiche Menge an korrekten Wörtern – beschreiben. Beide Computerprogramme gibt es nicht – und wird es bei Nutzung der uns bekannten Computer-Architekturen auch nicht geben. Mit dem Werkzeug Computer kann man also vieles tun – aber eben nicht alles.

Als Einstieg in das Thema können die Schülerinnen und Schüler Material zu Leben und Werk des britischen Mathematikers und Informatikers Alan Turing (1912–1954) recherchieren und aufbereiten[40] (siehe auch den historischen Aspekt in der Tabelle 1.2, S. 13). Turing löste im Jahr 1936 ein ganz bestimmtes Problem[41] dadurch, indem er nachwies, dass das Problem unlösbar ist. Diese Art der Lösung – vielleicht sollten wir besser sogenannte Lösung schreiben – war überraschend. Turings Idee besteht darin, den Begriff »Algorithmus« mithilfe einer idealen Maschine zu erfassen – und das zu einer Zeit, als es noch keine

[40] http://www.telegraph.co.uk/news/newstopics/politics/gordon-brown/6170112/Gordon-Brown-Im-proud-to-say-sorry-to-a-real-war-hero.html

[41] das Entscheidungsproblem

Computer gab. Es handelt sich um die später nach ihm benannte Turing-Maschine. Die Berechnung wird in eine Folge kleinster und einfachster Schritte zerlegt. Der Lösungsweg einer jeden Aufgabe, die berechenbar ist, muss sich als eine Folge solcher elementaren Berechnungsschritte angeben lassen. Wenn nachgewiesen werden kann, dass eine solche Folge nicht existiert, so ist die Aufgabe nicht lösbar. Zu klären ist, was ein elementarer Berechnungsschritt ist. Sehen wir uns dazu Aufbau und Arbeitsweise einer Turing-Maschine genauer an (zum Aufbau siehe Abbildung 13.1).

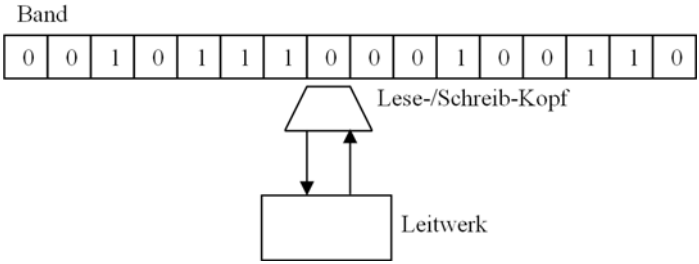


Abbildung 13.1:
Aufbau einer Turing-Maschine.
Eine Turing-Maschine besitzt als Speicher ein unendlich langes Band, das aus Zellen aufgebaut ist. Jede Zelle enthält ein Zeichen. In unserem Beispiel ist {0, 1} das Alphabet der Turing-Maschine. In jeder Zelle steht also eine 0 oder 1.

Auf dem Band kann sich der Lese-/Schreib-Kopf hin und her bewegen. Der Lese-/Schreib-Kopf liest aus der Zelle, bei der er sich gerade befindet, das dort gespeicherte Zeichen. Er schreibt anschließend in diese Zelle ein Zeichen. Das bisherige Zeichen wird dabei überschrieben. Anschließend bewegt er sich eine Zelle nach links oder eine Zelle nach rechts oder er verharrt an seiner Position. Die Turing-Maschine wird von einem Leitwerk aus gesteuert. Das Leitwerk enthält das Programm der Turing-Maschine und kommuniziert mit dem Lese-/Schreib-Kopf. Das Leitwerk befindet sich stets in einem bestimmten Zustand. Es kann seinen Zustand wechseln[42]. Das Programm der Turing-Maschine gibt in Abhängigkeit vom gelesenen Zeichen und vom aktuellen Zustand des Leitwerks an, welches Zeichen geschrieben werden soll, welche Bewegung der Lese-/Schreib-Kopf ausführen soll und welches der neue Zustand des Leitwerks ist.

[42] Ein Vergleich zwischen dem Leitwerk einer Turing-Maschine und der Tastatur eines PC kann für die Schüler hilfreich sein. Die Tastatur befindet sich immer in einem bestimmten Zustand und reagiert in Abhängigkeit von diesem unterschiedlich. Der Zustand der Tastatur kann wechseln. Konkret: Nach dem Einschalten des PC befindet sich die Tastatur in ihrem Anfangszustand. In diesem führt z. B. das Drücken der Taste A zur Ausgabe des Zeichens »a« auf dem Monitor. Das zusätzliche Drücken der Hoch-Taste führt zu einem Zustandswechsel der Tastatur. Nun wird das Zeichen »A« auf dem Monitor ausgegeben. Weitere Zustandswechsel sind möglich.

Wesentliche Grundlage für alles Weitere ist die Einsicht, dass es sich dabei wirklich um elementare Berechnungsschritte handelt und dass aus solchen Schritten jeder Lösungsweg zusammengesetzt werden kann. Was sollte denn auch einfacher sein, als das Lesen und Schreiben eines Zeichens, das eventuelle Aufsuchen einer benachbarten Zelle im Speicher und ein Zustandswechsel?[43].

Eine Turing-Maschine verwirklicht das EVA-Prinzip. Zu Beginn steht in einem Abschnitt des Bandes das Eingabewort. Die Turing-Maschine arbeitet so lange, bis der Haltezustand erreicht ist. Dann wird aus einem Abschnitt des Bandes das Ausgabewort ausgelesen. Der Haltezustand wird jedoch nicht immer erreicht.

Den Brückenschlag von der Turing-Maschine zum realen Computer sollte man im Unterricht gründlich besprechen. Dabei sollte klar werden:

- Das unendlich lange Band verschafft der Turing-Maschine gegenüber einem realen Computer mit seinem endlichen Speicher nur die Vorteile, dass für jedes Programm, das terminiert, genug Speicher vorhanden ist und dass Programme, die nicht terminieren, nicht zu einem »Absturz« der Turing-Maschine wegen »Speicherüberlauf« führen können.
- Eine Turing-Maschine kann in einem genügend großen endlichen Zeitraum jede Berechnung durchführen, die ein noch so weit entwickelter Einzelplatz-Computer bewältigt[44].
- Eine Turing-Maschine ist jedoch kein Computer. Jeder reale Computer arbeitet viel schneller als eine Turing-Maschine. Eine Turing-Maschine zeichnet sich dagegen durch einfachen Aufbau und einfache Arbeitsweise aus.
- Mithilfe von Turing-Maschinen kann gezeigt werden, dass es viele Probleme gibt, die sich mit keinem noch so schnellen oder leistungsfähigen Computer lösen lassen.

Eine spannende Frage ist sicher: Welche *mentalen* Modelle entwickeln Schülerinnen und Schüler zum *fachlichen* Modell »Turing-Maschine«?

Aufbereitung eines Beweises aus schulischer Sicht

Mit mathematischen Methoden kann nachgewiesen werden, dass es keinen Algorithmus gibt, der entscheidet, ob eine gegebene Turing-Maschine bei gegebenem Eingabewort die Bearbeitung beendet. Auch hier wieder ein Brückenschlag zum realen Computer: Damit ist dann auch bewiesen, dass es kein Computerprogramm gibt, das beliebige Quelltexte mit beliebigen Eingaben auf das Vorhandensein von Endlosschleifen überprüfen kann. Das Besprechen des Beweises im

[43] Es gibt andere elementare Berechnungsschritte im Kontext anderer Modelle (siehe z. B. Breier, 1989). Die Modelle erwiesen sich jedoch stets als gleichwertig zum Modell »Turing-Maschine«. Es konnte also immer genau das Gleiche berechnet werden.

[44] Kommunikation zwischen Computern über Computernetze, ständige Programmänderungen durch Software-Upgrades und permanentes Strömen von Input- und Outputdaten werden vom Modell »Turing-Maschine« nicht erfasst.

Informatikunterricht hat nur dann Sinn, wenn die Schülerinnen und Schüler die erforderlichen mathematischen Voraussetzungen beherrschen. Die nachfolgend vorgestellte Version des Beweises ist das Ergebnis eines mehrjährigen Erprobungsprozesses in Leistungskursen Informatik. So wurde in der ersten Version des Beweises der Begriff »Selbstanwendbarkeit« verwendet; er erwies sich als entbehrlich. Wegen festgestellter Verständnisprobleme wurde stets darauf geachtet, dass kein Schüler davon redet, dass eine Turing-Maschine eine *andere* Turing-Maschine bewertet, denn letztendlich bewertet eine Turing-Maschine ihr *eigenes* Programm. Nachfolgend sind auch einige Fragen von Schülern angegeben und mögliche Antworten darauf genannt. Die Fragen traten sinngemäß im eigenen Unterricht auf.

Der Beweis beginnt mit der Festlegung, dass nur Turing-Maschinen T betrachtet werden, bei denen als spezielles Eingabewort ihr Programm t genommen wird (siehe Abbildung 13.2).

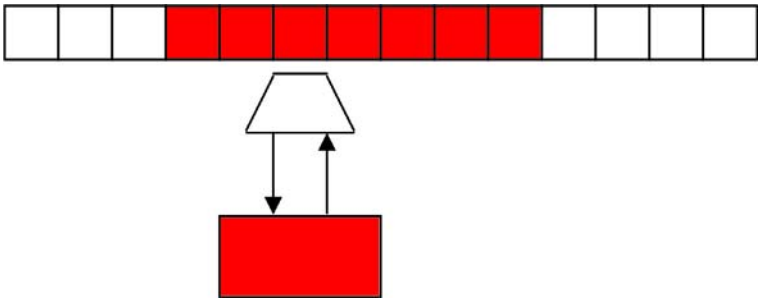


Abbildung 13.2:
Bei dieser Turing-Maschine stimmen Programm und Eingabewort überein.
Daher wurden sowohl das Leitwerk (beinhaltet das Programm) als auch ein Abschnitt des Bandes mit dem Eingabewort in der gleichen Farbe Rot gezeichnet.

Schülerfrage: Haben solche Programme denn eine praktische Bedeutung?
Mögliche Antwort: Ja. Ein Beispiel ist ein Computerprogramm, das Wörter zählt. Dieses Programm kann die Wörter im eigenen Quelltext zählen.

Der gesuchte Algorithmus soll entscheiden, ob eine gegebene Turing-Maschine T die Bearbeitung beendet. Die Annahme ist, dass dieser Algorithmus existiert. Dann gibt es auch eine Turing-Maschine, die die Entscheidung trifft. Diese Turing-Maschine nennen wir A (siehe Abbildung 13.3).

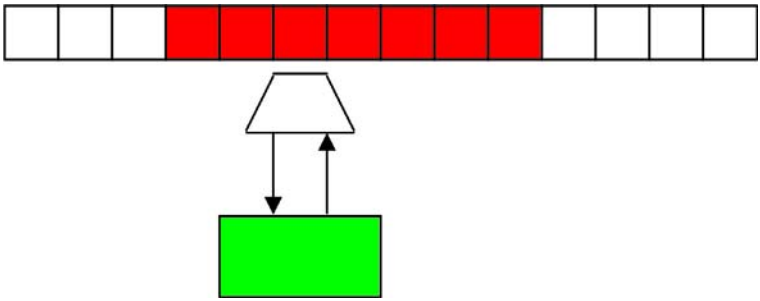


Abbildung 13.3:
Die Turing-Maschine A (grün) bewertet das Programm t der Turing-Maschine T (rot).
Bei der Turing-Maschine T sind Programm und Eingabewort gleich.

Diese Turing-Maschine A bewertet das Programm t der Turing-Maschine T. Dabei gibt es zwei Möglichkeiten (siehe Tabelle 13.1).

Möglichkeit	*Bewertung*	*Reaktion*
1	A stellt fest, dass T die Abarbeitung beendet.	A schreibt 1 auf ihr Band und beendet die Abarbeitung.
2	A stellt fest, dass T die Abarbeitung *nicht* beendet.	A schreibt 0 auf ihr Band und beendet die Abarbeitung.

Tabelle 13.1:
Bewertungen und Reaktionen der Turing-Maschine A.

Die Turing-Maschine A wird nun modifiziert. Die dabei entstehende Turing-Maschine B bewertet das Programm t der Turing-Maschine T genauso wie A, die Reaktion ist jedoch eine andere (siehe Tabelle 13.2 und Abbildung 13.4).

Möglichkeit	*Bewertung*	*Reaktion*
1	B stellt fest, dass T die Abarbeitung beendet.	B schreibt 1 1 1 … auf ihr Band und beendet die Abarbeitung *nicht*.
2	B stellt fest, dass T die Abarbeitung *nicht* beendet.	B schreibt 0 auf ihr Band und beendet die Abarbeitung.

Tabelle 13.2:
Bewertungen und Reaktionen der Turing-Maschine B.

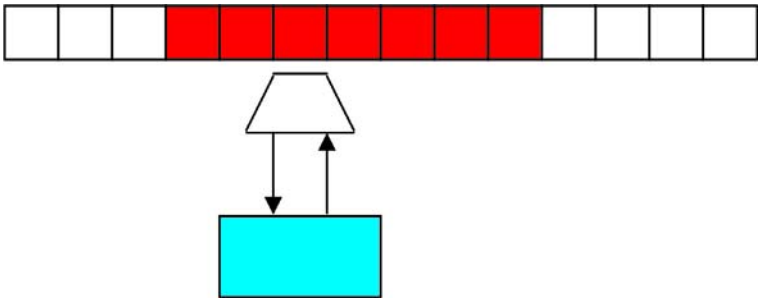


Abbildung 13.4:
Die Turing-Maschine B (blau) bewertet das Programm t der Turing-Maschine T (rot).
Bei der Turing-Maschine T sind Programm und Eingabewort gleich.

Wenn es die Turing-Maschine A gibt, so gibt es auch die Turing-Maschine B. Dann gilt auch: Wenn es die Turing-Maschine B *nicht* gibt, so gibt es auch die Turing-Maschine A *nicht*.

Schülerfrage: Stimmt das wirklich?
Mögliche Antwort: Ja. Diese Art der Umkehrung nennt man Kontraposition. Man kann sich deren Korrektheit an einem Beispiel aus dem täglichen Leben klar machen: Wenn es regnet, wird die Straße nass. Daraus folgt: Wenn die Straße nicht nass wird, regnet es nicht.

Nun soll die Turing-Maschine B das Programm b der Turing-Maschine B bewerten. Es soll also B(b) entschieden werden (siehe Abbildung 13.5).

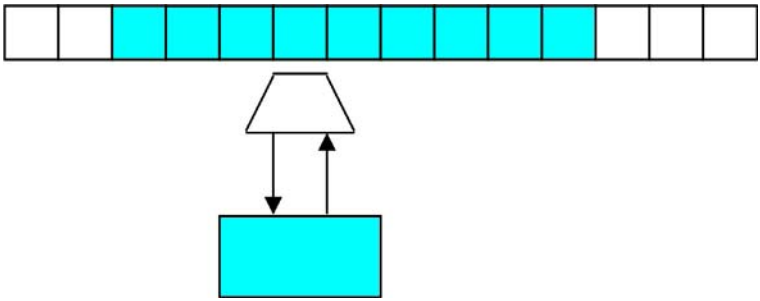


Abbildung 13.5:
Die Turing-Maschine B (blau) bewertet ihr eigenes Programm b (blau).

Schülerfrage: Warum nehmen wir gerade diese Turing-Maschine?
Mögliche Antwort: Zugegeben, das ist trickreich. Aber es ist immerhin eine Turing-Maschine von der Art, die wir betrachten wollen: Programm und Eingabewort stimmen überein.

Zwei mögliche Bewertungen sind zu untersuchen:

1. Bewertung: Die Turing-Maschine B beendet die Abarbeitung.
   Reaktion: Die Turing-Maschine B schreibt 1 1 1 … auf ihr Band und beendet die Abarbeitung *nicht*.
2. Bewertung: Die Turing-Maschine B beendet die Abarbeitung *nicht*.
   Reaktion: Die Turing-Maschine B schreibt 0 auf ihr Band und beendet die Abarbeitung.

In beiden Fällen ergibt sich ein Widerspruch. Daher kann es die Turing-Maschine B nicht geben. Sie wäre ja ein Widerspruch in sich. Wegen der Kontraposition folgt, dass es die Turing-Maschine A auch nicht gibt. Der gesuchte Algorithmus, der eine Entscheidung für beliebige Turing-Maschinen treffen soll, existiert daher nicht. Daraus können wir schlussfolgern, dass die Annahme falsch und die Behauptung wahr ist.

Schülerfrage: Wir haben nur ein Gegenbeispiel angegeben. Reicht denn das?
Mögliche Antwort: Im Alltag reicht ein Gegenbeispiel häufig nicht aus, um eine Aussage zu erschüttern. Man operiert in vielen Fällen mit Wahrscheinlichkeitsaussagen. Davon zu unterscheiden ist die formale Logik. In ihr reicht ein Gegenbeispiel wirklich aus, um eine Allaussage zu widerlegen. Wir haben jedoch nicht nur ein Gegenbeispiel angegeben, sondern unendlich viele. Schließlich haben wir von der Turing-Maschine B nur festgelegt, *was* sie leisten soll und nicht, *wie* sie es tut. Es gibt stets beliebig viele verschiedene Algorithmen/Computerprogramme, die eine bestimmte Aufgabe lösen. Man kann umständlich oder clever programmieren. Der zugrunde liegende Algorithmus kann einen guten oder schlechten Zeitaufwand besitzen. Man kann ein Programm beliebig lange mit vollkommen unnötigen Schnörkeln aufblasen – bis man am Ende gar nicht mehr sieht, was es eigentlich leistet.

Die grundlegende Idee des Beweises kann auf einfachem Niveau mit einer Plausibilitätsbetrachtung verdeutlicht werden. Man sucht eine Funktion `test`, die Folgendes leistet: Der Aufruf `test(x)` liefert den Funktionswert `TRUE`, falls der Algorithmus `x` bei seiner Abarbeitung zu einem Ende kommt. Ansonsten liefert der Aufruf den Funktionswert `FALSE`. Der Parameter `x` bezeichnet einen beliebigen Algorithmus. Wir betrachten nun den Algorithmus `a` (siehe Abbildung 13.6).

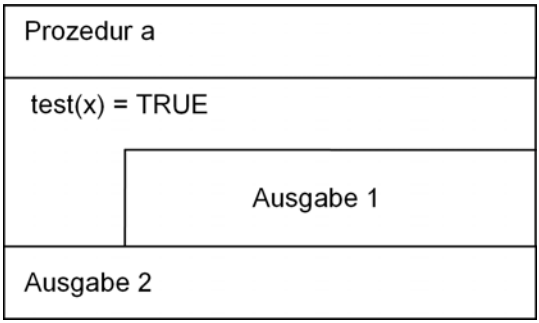


Abbildung 13.6:
Struktogramm für den Algorithmus a.

Jetzt soll test(a) ermittelt werden. Die zwei möglichen Antworten werden untersucht (siehe Abbildung 13.7). In beiden Fällen ergibt sich ein Widerspruch. Daher kann es die Funktion test nicht geben.

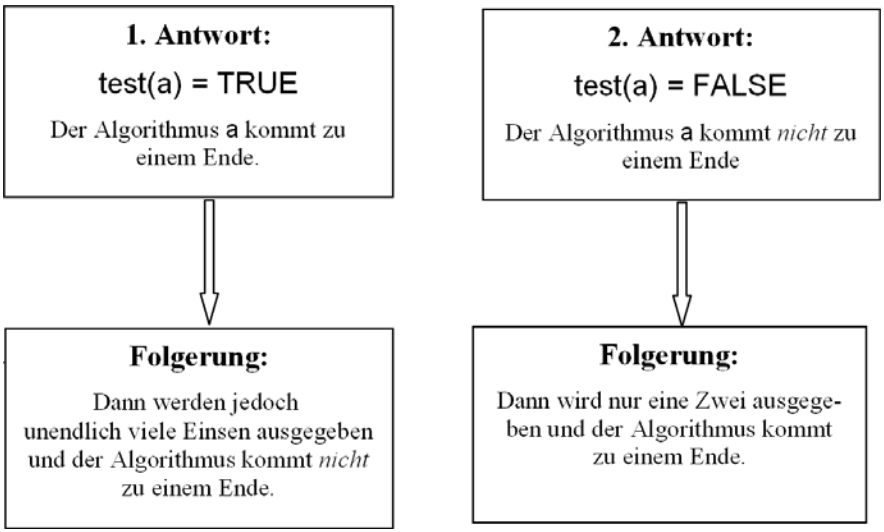


Abbildung 13.7:
Antwortmöglichkeiten und Folgerungen.

Vom Schauen will ich ausgehen.
Heinz Zemanek

14 Fallbeispiel: Zeitaufwand von Sortierverfahren

In den EPA Informatik heißt es bei den Beispielanforderungen für den Anforderungsbereich II (KMK, 2004, S. 16): »Begründen von bestimmten Eigenschaften (z. B. Terminierung, Zeit- und Speicheraufwand) eines gegebenen Algorithmus durch nicht formale Überlegungen« (siehe auch den praktischen Aspekt in der Tabelle 1.2, S. 13). In diesem Kapitel wird anhand von vier Sortierverfahren gezeigt, wie auf diese Prüfungsanforderung im Unterricht vorbereitet werden kann. Nach Möglichkeit wird der Computer nachfolgend als Werkzeug zum Experimentieren eingesetzt.

Jedes Sortierverfahren besitzt spezifische Eigenschaften, aus denen sich besonders geeignete Einsatzgebiete ableiten lassen. Eine wesentliche Eigenschaft ist die *Qualität des funktionalen Zusammenhangs* zwischen der Problemgröße – das ist die Anzahl n der Elemente a_1, a_2, a_3, ..., a_n – und der für das Sortieren benötigten Zeit t. Hat man nur wenige Elemente, so reicht ein Algorithmus mit schlechtem Zeitaufwand. Diese Algorithmen sind meist einfach zu verstehen und zu implementieren. Bei einer großen Anzahl von Elementen verwendet man einen Algorithmus mit gutem Zeitaufwand. Diese sind vergleichsweise kompliziert. Aus Gründen der Konzentration auf das Wesentliche werden in diesem Kapitel stets nur Zahlen sortiert. Es wird vorausgesetzt, dass der Leser weiß, wie die thematisierten Sortierverfahren funktionieren (Fothe, 2003). Auf diverse Animationen im Internet, die sich mit dem Sortieren und Suchen befassen, soll an dieser Stelle zumindest hingewiesen werden.

Naives Sortieren

Ein eher spielerisches Sortierverfahren ist das naive Sortieren. Bei diesem Verfahren werden nach und nach alle möglichen Reihenfolgen (Permutationen) der zu sortierenden Zahlen erzeugt und jedes Mal wird überprüft, ob die Zahlen in der sortierten Reihenfolge vorliegen (siehe S. 92 f.). Ist dies der Fall, so wird das Erzeugen und Überprüfen beendet. Ein Gefühl für den Zeitaufwand des Algorithmus lässt sich z. B. durch Abarbeiten eines Prolog-Programms (Sortieren einer Liste) oder Oberon-Programms (Sortieren eines Arrays) gewinnen. Die Rechenzeit steigt sehr schnell an. Schon für kleine n dauert es nahezu ewig, bis das Ergebnis feststeht. Man ist geneigt, von Slowsort zu sprechen, denn es gibt n! Permutationen von n verschiedenen Zahlen, von denen genau eine die gesuchte ist. Im Mittel sind daher n!/2 Permutationen zu erzeugen und zu überprüfen.

Sortieren durch Auswählen

```
PROCEDURE Auswahl*;
VAR r, s, t, x: INTEGER;
BEGIN
  FOR r := 1 TO n-1 DO
    t := r;  x := a[r];
    FOR s := r+1 TO n DO
      IF a[s] < x THEN
        t := s;  x := a[s]
      END
    END;
    a[t] := a[r];  a[r] := x
  END
END Auswahl;
```

Die Oberon-Prozedur Auswahl realisiert das Sortieren durch Auswählen (zum Sortieren von n Zahlen in aufsteigender Reihenfolge). Das Untersuchen des Zeitaufwands erfolgt als Erstes mittels Zeitmessungen. Dabei werden drei Eigenschaften der gegebenen Zahlen unterschieden: Die Zahlen liegen bereits aufsteigend sortiert vor. Die Zahlen sind Zufallszahlen. Die Zahlen liegen falsch herum, also absteigend sortiert vor (siehe Abbildung 14.1). Aus den Messergebnissen lässt sich die Vermutung ableiten, dass das Sortieren durch Auswählen quadratischen Zeitaufwand besitzt. Die Vermutung soll durch Analyse des Quelltextes bestätigt werden. Modellartig werden dabei nur die zeitaufwendigen Operationen betrachtet, also die Operationen, bei denen (mindestens) ein Zugriff auf das Array a erfolgt. Das sind der Vergleich a[s] < x und die Bewegungen x := a[r], x := a[s], a[t] := a[r] und a[r] := x. Für jede dieser Operationen wird vereinfachend jeweils die gleiche Rechenzeit angenommen. Die schnelleren Integer-Wertzuweisungen wie t := s und das Hochzählen der Laufvariablen bleiben unberücksichtigt. Die Analyse des Quelltextes soll computerunterstützt erfolgen. In dem Programm zum Sortieren durch Auswählen werden zu diesem Zweck zusätzliche Ausgabeanweisungen aufgenommen. Nach jedem Vergleich bzw. nach jeder Bewegung gibt das Programm ein Zeichen aus. Jedes Mal, wenn ein Element an die richtige Position gebracht wurde, wird eine neue Zeile begonnen. Veränderliche Parameter sind die Festlegung, ob Vergleiche oder Bewegungen ausgegeben werden sollen, die Anzahl zu sortierender Zahlen und die Eigenschaften dieser Zahlen.

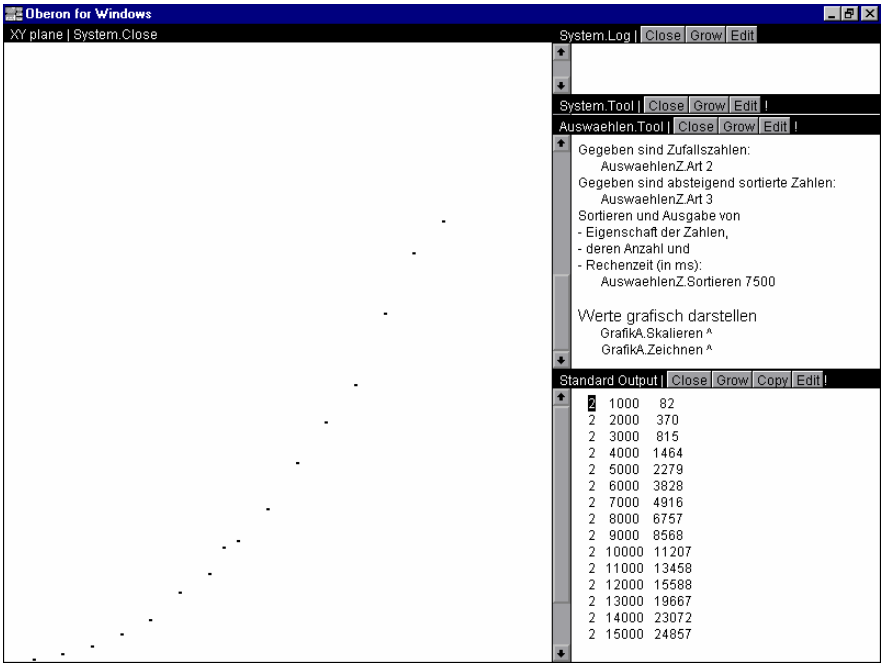


Abbildung 14.1:
Zeitmessungen beim Sortieren durch Auswählen
(x-Koordinate: Anzahl an Zahlen; y-Koordinate: Zeit für das Sortieren).

Im Einzelnen werden die folgenden Experimente durchgeführt:

1. Sortieren von 10 Zahlen mit unterschiedlichen Eigenschaften, dann 20 und 50 Zahlen: Die Schüler erkennen, dass die Anzahl V der Vergleiche unabhängig von den Eigenschaften der gegebenen Zahlen ist. Diese Aussage wird anhand des Quelltextes bestätigt.
2. Sortieren von 10, 20 und 50 Zahlen: Die Zeichen sind jeweils als Dreiecksfigur angeordnet (siehe Abbildung 14.2). Aus der Größe der Dreiecke ermitteln die Schüler die Formel $V = (n^2-n) / 2$ für die Anzahl V der Vergleiche.
3. Sortieren von 10, 20 und 50 Zahlen, die jeweils aufsteigend sortiert sind: Die Schüler ermitteln die Formel $B = 3 * (n-1)$ für die Anzahl B der Bewegungen. Die Formel wird anhand des Quelltextes bestätigt.
4. Sortieren von 10, 20 und 50 Zahlen, die jeweils absteigend sortiert sind: Die Schüler ermitteln die Formel $B \approx 3 * (n-1) + n^2 / 4$ durch Zerlegen der Figur in ein Rechteck und ein Dreieck (siehe Abbildung 14.2). Die Trichterform entsteht, weil in jedem Durchlauf der inneren for-Anweisung zwei Elemente an die richtigen Positionen gebracht werden.

Ausgabe der Vergleiche	Ausgabe der Bewegungen
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
**********	**********
190 Vergleiche	157 Bewegungen

Abbildung 14.2:
Ausgabe der ausgeführten Operationen beim Sortieren durch Auswählen.
Gegeben sind 20 Zahlen absteigend sortiert.

5. Sortieren von 10, 20 und 50 Zufallszahlen: Die Schüler vermuten, dass die Anzahl der Bewegungen zwischen den beiden Fällen aufsteigend sortiert / absteigend sortiert liegt. Sie können beobachten, dass die Anzahl der Bewegungen beim Sortieren von Zufallszahlen schneller als linear ansteigt. Genauere Erkenntnisse werden durch Experimente nicht gewonnen. Die mittlere Anzahl an Bewegungen ist schwierig zu bestimmen. Das theoretisch gewonnene Ergebnis ist $B \approx n * (\ln n + \gamma)$, wobei γ die Euler'sche Konstante 0,577216 ist (siehe Wirth, 1979, S. 96 ff.).

Aus den Experimenten lassen sich die folgenden Erkenntnisse ableiten:

- Die Anzahl der Vergleiche wächst stets quadratisch mit der Problemgröße.
- Sind die gegebenen Zahlen bereits aufsteigend sortiert, so wächst die Anzahl der Bewegungen linear. Sind die gegebenen Zahlen absteigend sortiert, so wächst die Anzahl der Bewegungen quadratisch. Werden Zufallszahlen sortiert, so wächst die Anzahl der Bewegungen zwischen linear und quadratisch.
- Die minimale Anzahl an Operationen ist dann auszuführen, wenn die gegebenen Zahlen bereits aufsteigend sortiert sind. Die maximale Anzahl an Operati-

onen ist vermutlich dann auszuführen, wenn die gegebenen Zahlen absteigend sortiert sind.

Quicksort

Die Oberon-Prozedur Sortieren sortiert n Zahlen mit dem Verfahren Quicksort. Der Algorithmus wurde gründlich untersucht. Bei der Untersuchung wurden mathematische Kenntnisse und Fähigkeiten in einem Umfang verwendet, der für den Unterricht an Hochschulen, jedoch nicht oder nur eingeschränkt an Schulen vorausgesetzt werden kann.

Nachfolgend wird ein Weg dargestellt, der mit Schülerinnen und Schülern beschritten werden kann. Dabei wird die experimentelle Methode angewandt. Die folgenreichen Auswirkungen der Wahl des Trennelements werden beachtet. Im *ersten* Schritt werden Hypothesen für den Zeitaufwand von Quicksort im besten und schlechtesten Fall mithilfe von Plausibilitätsbetrachtungen erarbeitet.

```
PROCEDURE Sortieren*;
  PROCEDURE Sort(l, r: INTEGER);
  VAR i, j, x, w: INTEGER;
  BEGIN
    i := l;  j := r;
    x := a[(l + r) DIV 2];
    REPEAT
      WHILE a[i] < x DO INC(i) END;
      WHILE x < a[j] DO DEC(j) END;
      IF i <= j THEN
        w := a[i];  a[i] := a[j];  a[j] := w;
        INC(i);  DEC(j)
      END
    UNTIL i > j;
    IF l < j THEN Sort(l, j) END;
    IF i < r THEN Sort(i, r) END
  END Sort;
BEGIN
  Sort(1, n)
END Sortieren;
```

Für das Erschließen des Zeitaufwands eignet sich ein einfaches Modell, das nachfolgend zuerst für den *besten Fall* beschrieben wird. Wir nehmen an, dass k Zahlen zu sortieren sind. Für das Abarbeiten der repeat-Anweisung werden näherungsweise k Zeiteinheiten benötigt. (Jede Zahl wird mit dem Trennelement verglichen. Manchmal werden zusätzlich zwei Zahlen getauscht.) Dann werden die linke und die rechte Teilfolge in gleicher Weise bearbeitet. Im besten Fall

wird die Teilfolge beim Sortieren stets in zwei gleich große Teilfolgen zerlegt. In der Abbildung 14.3 wird die Situation für 16 Zahlen, die zu sortieren sind, dargestellt. Das Zeichen x steht für eine Zahl und damit nach den Erläuterungen für eine Zeiteinheit. Die Zeichen x lassen sich zu einem Rechteck zusammenschieben. Es passt alles gut zusammen. Die Fläche als Maß für die benötigte Zeit zum Sortieren von 16 Zahlen hat den Inhalt 16 * 4. Durch weitere Beispiele (z. B. für 32 und 8 Zahlen) lässt sich der Zusammenhang finden, dass die benötigte Zeit proportional zu n * ld n ist.

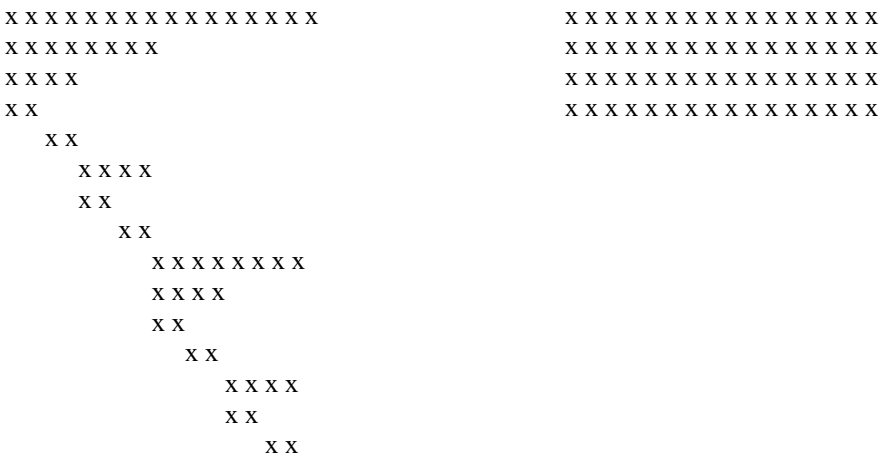


Abbildung 14.3:
Quicksort im besten Fall[45].

Der *schlechteste Fall* liegt zum Beispiel dann vor, wenn als Trennelement stets die größte Zahl einer Teilfolge genommen wird. Es entsteht ein Dreieck, das den Flächeninhalt 135 besitzt (siehe Abbildung 14.4). Allgemein ist der Flächeninhalt $(n^2+n-2)/2$. Damit ist der quadratische Zeitaufwand plausibel gemacht.

[45] Die Abbildungen 14.3 und 14.4 wurden nicht durch Software erzeugt.

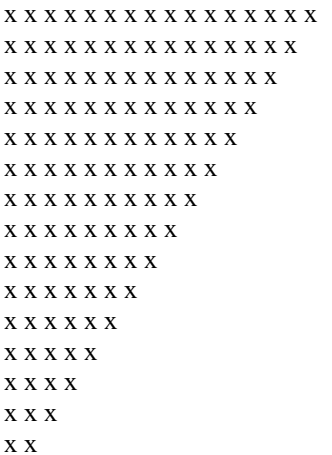


Abbildung 14.4:
Quicksort im schlechtesten Fall.

Im Modell werden die Vergleiche präzise berücksichtigt. Die Bewegungen werden dagegen sehr großzügig betrachtet. Diese Vereinfachung ist für eine Plausibilitätsbetrachtung akzeptabel. Das Modell eignet sich auch für das Ermitteln des Speicheraufwands, denn jede Folge von Zeichen »x« entspricht einem Prozeduraufruf. Im besten Fall liegen gleichzeitig bis zu ld n Einträge im Stack vor. Im schlechtesten Fall sind es n−1 Einträge.

Im *zweiten* Schritt werden die Funktionen f und g mit $f(x) = x^2$ und $g(x) = x * \mathrm{ld}\, x$ näher analysiert. Was passiert, wenn sich der x-Wert verdoppelt? Bei der Funktion f ergibt sich die Vervierfachung des y-Wertes. Für die Funktion g gilt, dass sich der y-Wert nur auf das (2 * (1 + 1 / ld x))-fache vergrößert. Diese Funktion wollen wir daher als »fast linear« bezeichnen. Die beiden Hypothesen für den Zeitaufwand im besten und schlechtesten Fall werden nun experimentell mithilfe von Zeitmessungen[46] überprüft, was auch gelingt (siehe Abbildung 14.5).

[46] Alle Zeitmessungen in diesem Kapitel erfolgten auf einem langsamen Notebook von 1997.

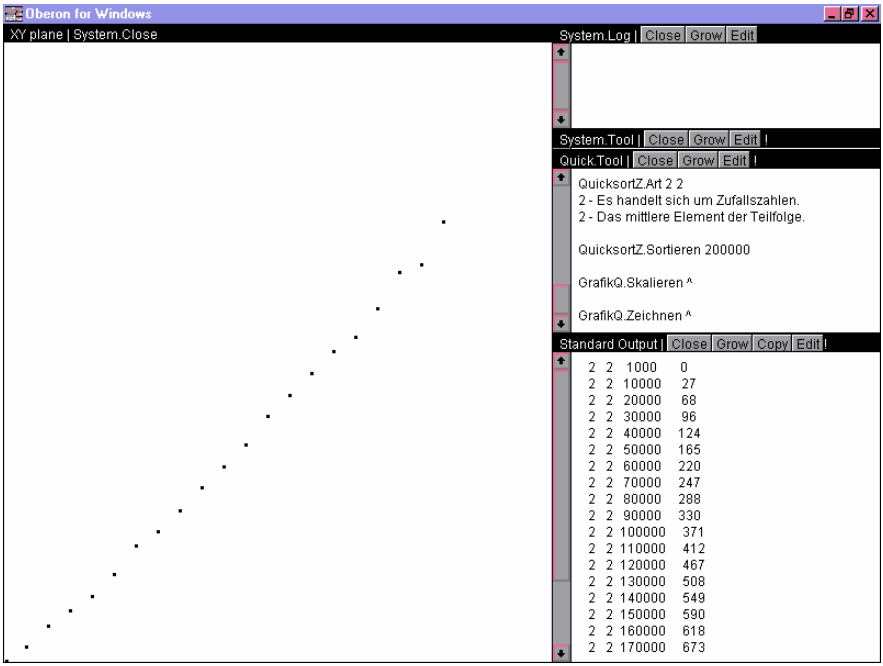


Abbildung 14.5:
Zeitmessungen bei Quicksort
(x-Koordinate: Anzahl an Zahlen; y-Koordinate: Zeit für das Sortieren).

Für unterschiedliche Anzahlen an Real-Zufallszahlen werden die verschiedenen Arten von Operationen gezählt (als Trennelement wird stets das mittlere Element einer Teilfolge genommen). Auf jeweils 5 bis 6 Vergleiche kommt dabei ein Tausch zweier Zahlen. Ein Tausch ist aus drei Bewegungen zusammengesetzt. Auch die rekursiven Aufrufe der Prozedur Sort haben Einfluss auf die Zeit, die zum Sortieren benötigt wird. Die Anzahl an Aufrufen der Prozedur – und damit der Zeitaufwand zum internen Verwalten des Rekursionsstapels – steigt ungefähr linear mit der Problemgröße an (siehe Tabelle 14.1).

Zahlen	*Vergleiche*	*Bewegungen*	*Tausch zweier Zahlen*	*Aufrufe der Prozedur* Sort
20.000	371.583	234.688	72.279	17.851
50.000	1.036.438	630.078	195.219	44.421
100.000	2.344.248	1.317.376	409.464	88.984
150.000	3.573.756	2.028.153	631.559	133.476

Tabelle 14.1:
Anzahl an verschiedenen Operationen bei Quicksort.

Im *dritten* Schritt werden zwölf Fälle unterschieden, und zwar drei mögliche Eigenschaften der gegebenen Zahlen:
1. Die Zahlen liegen bereits aufsteigend sortiert vor.
2. Die Zahlen sind Zufallszahlen.
3. Die Zahlen liegen falsch herum, also absteigend sortiert vor.

und vier Möglichkeiten für die Wahl des Trennelements:
1. Das erste Element der Teilfolge.
2. Das mittlere Element der Teilfolge.
3. Das letzte Element der Teilfolge.
4. Ein zufälliges Element der Teilfolge.

Jedes Mal sind z. B. 10.000 Zahlen zu sortieren. Vermutungen über die benötigte Rechenzeit werden aufgestellt und anschließend experimentell überprüft (siehe Tabelle 14.2).

```
1  1  10000   6834
1  2  10000     18
1  3  10000   6927
1  4  10000     21
2  1  10000     41
2  2  10000     39
2  3  10000     39
2  4  10000     27
3  1  10000   6767
3  2  10000      9
3  3  10000   6372
3  4  10000     28
```

Tabelle 14.2:
Zeitmessungen beim Sortieren mit Quicksort.
Nach jeder Zeitmessung werden vier Zahlen ausgegeben: Eigenschaft der gegebenen Zahlen, Wahl des Trennelementes, Anzahl an Zahlen und benötigte Zeit in ms.

Aus den Untersuchungen lässt sich ableiten, dass Quicksort meist sehr schnell ist. Langsam ist das Verfahren, wenn die gegebenen Zahlen aufsteigend oder absteigend sortiert sind und wenn als Trennelement stets das erste oder das letzte Element einer Teilfolge genommen wird. Es lässt sich z. B. herausarbeiten, dass der schlechteste Fall auch beim Sortieren von Zufallszahlen und Auswählen von zufälligen Trennelementen eintreten kann, dass dies jedoch sehr unwahrscheinlich ist. Die Ursachen für den Zeitaufwand werden diskutiert.

Spaghetti-Computer

Der Funktionalität eines Analogcomputers liegt eine physikalische Größe oder Eigenschaft zugrunde. Beim *Spaghetti-Computer* ist dies die Länge (Dewdney, 1984). Für jede Zahl wird eine Spaghetti-Stange entsprechender Länge hergestellt. Zum Beispiel ist für die Zahl 85 eine Spaghetti-Stange der Länge 85 cm herzustellen. Zum Sortieren wird das Bündel an Spaghetti-Stangen auf einer Ebene kräftig aufgestoßen und dann werden die Spaghetti-Stangen der Länge nach von oben nach unten weggenommen und gemessen. Die Zahlen werden aufgeschrieben. Dieser Analogcomputer besitzt *linearen* Zeitaufwand, denn bei der doppelten Anzahl an Zahlen verdoppelt sich die Zeit für das Herstellen der Spaghetti-Stangen (Vorbereitungsphase) und das Wegnehmen und Messen der Länge (Nachbereitungsphase). Der Augenblick für das Aufstoßen des Bündels kann vernachlässigt werden (Analogphase). Das Sortieren mit dem Spaghetti-Computer wird im Unterricht in einem Gedankenexperiment unter idealen Annahmen durchgespielt. Die praktische Ausführbarkeit ist natürlich nur für wenige Zahlen gegeben. Der interessante Ansatz scheitert leider bei einer größeren Anzahl von zu sortierenden Zahlen.

Resümee

Wichtige Aussagen zum Zeitaufwand von Algorithmen können über Experimente mit dem Computer gewonnen werden. Diese Methode besitzt jedoch Grenzen. Mit Plausibilitätsbetrachtungen, der experimentellen Methode und Gedankenexperimenten kann das Verständnis der Lernenden vertieft werden. Die Beispiele verdeutlichen die Möglichkeit, das Thema »Zeitaufwand von Algorithmen« weder zu theoretisch noch zu vereinfachend im Informatikunterricht zu behandeln.

Es ist immer schön zu sehen,
wie verschiedene Geister
denselben Stoff formen.
Friedrich von Schiller

15 Zentralabitur im Fach Informatik

Seit einigen Jahren ist in den Ländern eine bildungspolitische Entwicklung von dezentralen zu zentralen Abiturprüfungen festzustellen. Zentrale Abiturprüfungen sind nicht prinzipiell besser oder schlechter als dezentrale. Für beide Varianten können Vor- und Nachteile angegeben werden. So unter anderem: Das Zentralabitur ist eher geeignet zum Absichern eines einheitlichen Niveaus und zum Erhöhen der Transparenz der Arbeit von Schulen. Bei einem dezentralen Abitur fällt es dagegen leichter, neue Inhalte und Werkzeuge im Unterricht zu erproben. Die Gesamtmenge an »Stoff«, den die Prüfungsteilnehmer zu lernen haben, dürfte beim zentralen und dezentralen Abitur ähnlich sein. Im zentralen Abitur kann es dabei jedoch mehr um die Breite, beim dezentralen Abitur mehr um die Tiefe der nachzuweisenden Kompetenzen gehen. Manche Aufgabenbeispiele der EPA Informatik sind daher mehr für das dezentrale, andere mehr für das zentrale Abitur geeignet.

In diesem Kapitel werden Grundsätze für zentrale Abiturprüfungen im Fach Informatik zusammengestellt und am Beispiel von Thüringen untersetzt. Thüringen führte die gymnasiale Oberstufe mit dem Kurssystem im Jahr 1992 ein. Ein Zentralabitur im Grund- und Leistungsfach Informatik gibt es seit Sommer 1994. Aktuelle Veränderungen an der gymnasialen Oberstufe werden dazu führen, dass künftig in Thüringen zentrale Abiturprüfungen nur noch im Fach Informatik mit erhöhtem Anforderungsniveau stattfinden. Die nachfolgende Darstellung beruht auf den bisherigen Regelungen.

EPA Informatik als Grundlage

Die Prüfungsaufgaben müssen den EPA Informatik entsprechen. Was ergibt sich aus dieser Forderung für das Erarbeiten von Abituraufgaben? Denkbar sind u. a. vier Varianten, die nachfolgend charakterisiert werden. Abbildung 15.1 entspricht der Situation im dezentralen Abitur. Der Unterricht wird auf der Grundlage des Informatiklehrplans und der EPA Informatik geplant und durchgeführt. Die Prüfungsaufgaben entstehen dann auf der Basis von EPA, Lehrplan und realisiertem Unterricht.

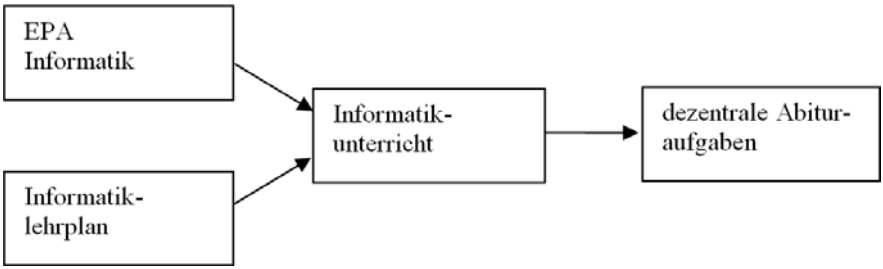


Abbildung 15.1

In der Variante von Abbildung 15.2 werden die zentralen Abituraufgaben unmittelbar auf der Grundlage der EPA Informatik erstellt. Dieses Vorgehen wird nicht als geeignet bewertet, da die EPA nicht für diesen Zweck erarbeitet wurden (siehe Kapitel 8).

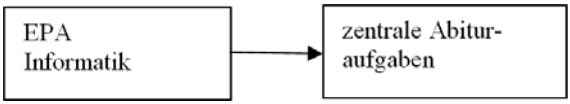


Abbildung 15.2

Die Variante von Abbildung 15.3 stellt die Situation dar, dass auf der Grundlage der EPA Informatik von 2004 landesspezifische Regelungen erarbeitet wurden (z. B. Lehrplan, Kerncurriculum, »Landes-EPA«). Diese Regelungen setzen die länderübergreifenden Vorgaben in Festlegungen für die Schulen im jeweiligen Land um. Sie enthalten z. B. Konkretisierungen, Schwerpunktsetzungen und Wahlmöglichkeiten und sind damit zum Erstellen der zentralen Abituraufgaben geeignet.

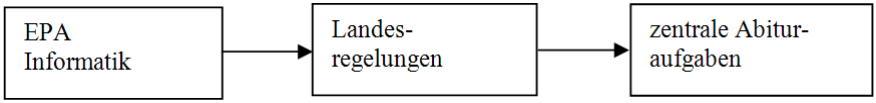


Abbildung 15.3

Die Variante aus der Abbildung 15.4 stellt die Situation dar, dass neben den EPA Informatik der bisherige Informatiklehrplan weiterhin gültig ist. Die Autoren der Abituraufgaben müssen dann sowohl den Informatiklehrplan als auch

die EPA Informatik beachten, wenn sie Aufgaben erstellen. Diese Situation ist in Thüringen gegeben.

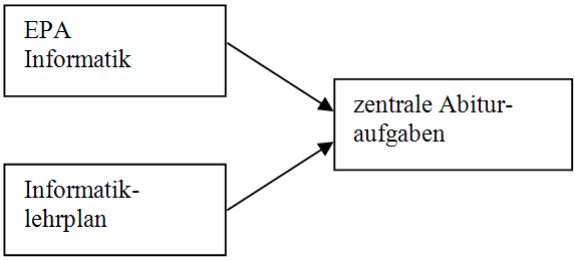


Abbildung 15.4

Inhalt und Struktur der Abiturprüfung

In Thüringen gibt es im Grundfach Informatik keine Pflichtaufgaben, sondern vier Wahlaufgaben, von denen drei zu bearbeiten sind. Die Wahlaufgaben sind Themenbereichen nicht fest zugeordnet. Im Leistungsfach Informatik bearbeiten die Prüfungsteilnehmer die Aufgaben 1 und 2 und wählen von den Komplexaufgaben 3.1 und 3.2 eine aus. Die Aufgabe 2 ist dem Themenbereich 9 des Informatiklehrplans entnommen (Logik-orientiertes Programmieren). Als Inhalte sind sowohl im Grund- als auch im Leistungsfach Informatik alle Themen des Lehrplans mit Ausnahme der Wahl-Themenbereiche festgelegt. Die Wahl-Themenbereiche können im Grundfach in der mündlichen Prüfung geprüft werden.

Computereinsatz in der Abiturprüfung

Ein sinnvoller Grundsatz ist es, dass die Prüfungsteilnehmer diejenigen Hilfsmittel in der Abiturprüfung verwenden dürfen, an die sie im Unterricht gewöhnt sind. Dies spricht für die unbedingte Zulassung von Computern in der Abiturprüfung. Eine genauere Analyse zeigt jedoch, dass zum Computereinsatz diverse Pro- und Contra-Argumente abzuwägen sind (siehe Fothe u. a., 2007). Falls Computer zugelassen sind, ist zu regeln, welche Programmierumgebungen zur Verfügung gestellt werden. Die Abituraufgaben müssen sich mit den zugelassenen Programmiersprachen gleich gut lösen lassen.

In Thüringen ist im Grundfach Informatik nur die Wahlaufgabe 4 mithilfe des Computers zu lösen (mit Pascal, Oberon bzw. Java). Der Computereinsatz ist aus Sicht der Prüfungsteilnehmer daher nicht verpflichtend, denn sie können die drei anderen Wahlaufgaben ohne Computer bearbeiten. Im Leistungsfach Informatik ist der Computereinsatz verpflichtend, und zwar beim Lösen der Aufgabe 2 (mit Prolog) und der Komplexaufgabe 3.1 oder 3.2 (mit Pascal, Oberon bzw. Java).

Alle Hilfsmittel, also auch die Computer, stehen während der gesamten Prüfungszeit zur Verfügung. Festzulegen ist z. B., wie damit umzugehen ist, wenn ein Prüfungsteilnehmer ein Computerprogramm im Prinzip fertig (eingetippt) hat, es jedoch wegen geringfügiger syntaktischer Probleme nicht zum Laufen bringt, und wie bei technisch bedingten Problemen verfahren wird (siehe Fothe u. a., 2007).

Rolle des Programmentwurfs

Eine Abituraufgabe könnte lauten:

> Schreiben Sie ein Programm, das ein zweistufiges Speichersystem simuliert.

Die Prüfungsteilnehmer erarbeiten einen Programmtext, der, falls er korrekt ist, mit der vollen Anzahl an Bewertungseinheiten versehen wird. Ist er jedoch fehlerhaft, so ist das Bewerten schwierig, da die Gedanken des Programmierers häufig nur schwer nachzuvollziehen sind. Daher ist es sinnvoll, die Aufgabe in Teile zu gliedern (siehe Kapitel 11):

> - Entwerfen Sie ein Programm, das ein zweistufiges Speichersystem simuliert.
> - Implementieren Sie das Programm.
> - Erläutern Sie, welche Methoden der Softwareentwicklung von Ihnen verwendet wurden.
>
> (vgl. Thüringer Abiturprüfung, Leistungsfach Informatik, 2001)

In Thüringen haben die Prüfungsteilnehmer einen Programmentwurf und einen kommentierten Quelltext abzugeben. Mitunter (wie auch hier) gibt es eine zusätzliche Aufgabe, die eine Reflexion zur Lösung oder zum Lösungsweg beinhaltet.

Operatoren

Operatoren sind für einen kompetenzorientierten Informatikunterricht von Bedeutung. Durch diese Aufforderungsverben soll Schülerinnen und Schülern beim Bearbeiten von Aufgaben klar werden, welche Tätigkeiten und welche Lösungsdarstellung von ihnen erwartet werden. Damit soll möglichen Missdeutungen von Aufgabenstellungen entgegengewirkt werden.

Nachfolgend wird eine Statistik zu den in den Abiturprüfungen von 2001 bis 2007 in Thüringen verwendeten Operatoren angegeben (Grund- und Leistungsfach Informatik, jeweils Haupttermin). In Thüringen gibt es (bisher) keine verbindliche Operatorenliste. Im genannten Zeitraum wurden insgesamt 44 verschiedene Operatoren verwendet (häufig verwendete Operatoren wurden hervorgehoben):

abschätzen, **angeben**, auseinandersetzen,
auswählen, beachten, **begründen**, berechnen,
beschreiben, **beurteilen**, codieren, darstellen,
definieren, diskutieren, **dokumentieren**, eingehen,
entscheiden, entschlüsseln, **entwerfen**,
erarbeiten, erklären, **erläutern**, erstellen, ermitteln,
erweitern, erzeugen, herstellen, **implementieren**,
kennzeichnen, **kommentieren**, konstruieren,
messen, **nennen**, notieren, realisieren, **testen**,
überführen, überprüfen, übertragen, verändern,
vergleichen, verschlüsseln, verwenden, würdigen,
zeichnen

Manche dieser Operatoren wurden häufig, andere nur selten verwendet (einige nur einmal). Im Grundfach Informatik sind die zehn Operatoren, die in den Abituraufgaben am häufigsten verwendet wurden: angeben/nennen, erläutern, entwerfen, begründen, implementieren, kommentieren, beschreiben, beurteilen und entscheiden (in dieser Reihenfolge). Sie machen insgesamt etwa 70 % aller Verwendungen von Operatoren aus. Im Leistungsfach Informatik sind die zehn häufigsten Operatoren: erläutern, entwerfen, implementieren, kommentieren, angeben/nennen, dokumentieren, beschreiben, begründen und testen (in dieser Reihenfolge). Sie machen insgesamt etwa 80 % aller Verwendungen aus. In einer Prüfung im Grundfach Informatik (die vier Wahlaufgaben zusammengenommen) konnten 9 bis 22 verschiedene Operatoren gezählt werden. Im Leistungsfach Informatik (die Pflicht- und Wahlaufgaben zusammengenommen) waren es 11 bis 15.

Zu der Statistik können Fragen formuliert werden, so unter anderem: Ist ein höherer Grad an Standardisierung der Operatoren sinnvoll? Sollte dazu eine verbindliche Operatorenliste erarbeitet werden? Ist die Anzahl verschiedener Operatoren, die in einer Prüfungsaufgabe verwendet werden, angemessen? Repräsentieren die häufig verwendeten Operatoren wirklich die wesentlichen Tätigkeiten im Informatikunterricht (Grund- und Leistungsfach)?

Verabredete Listen aller zulässigen Operatoren können unterschiedlich umfangreich sein. Aus einer Liste mit vergleichsweise vielen Operatoren ergäbe sich als Konsequenz, dass die Schülerinnen und Schüler im Unterricht auf die Lösung von vielfältigen Aufgaben mit vielfältigen Operatoren für die Abiturprüfung und evtl. bereits für Kursarbeiten vorzubereiten wären. Sind Operatoren jedoch wirklich ein so zentrales Bildungsgut, dass diese Schwerpunktsetzung gerechtfertigt wäre? Bei einer Liste mit vergleichsweise wenigen Operatoren ließe sich vielleicht manche sinnvolle Prüfungsaufgabe gar nicht mehr stellen. Es wäre doppelplusungut[47], wenn am Ende Operatorenlisten bestimmten, welche Aufgaben in Abiturprüfungen überhaupt gestellt werden dürfen. Es spricht also einiges für die

[47] Das Wort auf Neusprech (aus dem Roman »1984« von George Orwell) bedeutet in etwa: äußerst schlecht.

Verabredung einer reduzierten Liste, in der man sich auf die Operatoren konzentriert, denen eine besondere fachspezifische Relevanz zukommt. Die überwiegende Anzahl der Aufgaben sollte an diesen Operatoren orientiert sein. In der Liste nicht enthaltene Operatoren dürfen dennoch in Prüfungsaufgaben verwendet werden, wenn sie allgemeinverständlich sind, sie sozusagen zum »Weltwissen« von Schülerinnen und Schülern gehören.

Ein pragmatischer Lösungsansatz (dies wird auch so praktiziert) ist das Erarbeiten einer gemeinsamen Operatorenliste für alle Fächer des mathematisch-naturwissenschaftlich-technischen Aufgabenfeldes (also auch für Informatik). Fachspezifische Ergänzungen für die einzelnen Fächer können vorgenommen werden.

Man sollte sich verstärkt mit dem Formulieren von Aufgaben befassen und in diesem Kontext auch die Verwendung von Operatoren empirisch untersuchen. Ein spannendes Experiment wäre das Bearbeiten von real gelaufenen Abituraufgaben. Dabei erhält eine Hälfte der Probanden die vollständigen Aufgaben, bei der anderen Hälfte sind die Operatoren unlesbar durchgestrichen. Wie gehen die jeweiligen Probanden mit den Aufgaben um?

Interdisziplinarität in der Abiturprüfung

Die Bearbeitung von Problemen aus unterschiedlichen Fachgebieten ist typisch für die Informatik im Allgemeinen und den Informatikunterricht im Besonderen und sollte es daher auch für die Abiturprüfung sein. Mehrere Aufgabenbeispiele in den EPA Informatik verdeutlichen diesen Anspruch (siehe Tabelle 15.1).

Bezugsfächer	*Aufgaben*
Mathematik	1.2.2 Kryptographie
Physik	1.4.2 Mobile
Biologie	1.2.7 DNS-Replikation
Wirtschaft/Recht	1.1.2 Immobilien
Englisch	1.2.6 Zeitsynchronisation

Tabelle 15.1:
Aufgabenbeispiele in den EPA Informatik mit interdisziplinärem Charakter.

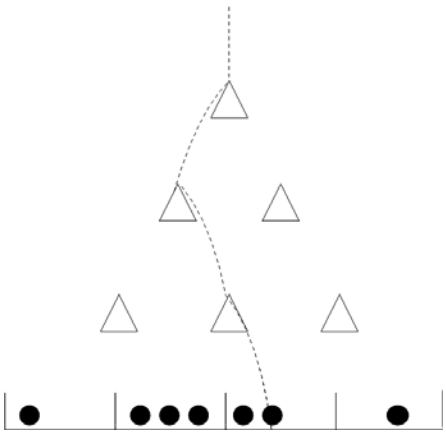
Bei Aufgaben dieser Art muss im Aufgabentext problemspezifisches Wissen bereitgestellt werden. Dabei können Vereinfachungen sinnvoll sein, um einen kurzen und prägnanten Text zu erhalten und um die Schwierigkeit oder Komplexität der zu lösenden Teilprobleme zu verringern. Mitunter lässt sich durchaus

darüber diskutieren, wie weit man beim Einbeziehen fachspezifischen Wissens gehen sollte. Ein Beispiel ist eine Aufgabe aus dem Thüringer Zentralabitur (3. Aufgabe »Tonleiter«, S. 164). War es bei dieser Aufgabe wirklich erforderlich, das »g« in »g-Dur-Tonleiter« klein zu schreiben? Damit wurde den Prüfungsteilnehmern zwar deutlich, dass sie beim Lösen der Aufgabe ausschließlich mit Kleinbuchstaben operieren sollten. Musiktheoretisch korrekt ist jedoch ein großes »G«. Die Forderung nach korrekter Verwendung der Begriffe aus anderen Fachgebieten ist unter dem Aspekt des fächerübergreifenden Unterrichtens und der grundsätzlichen Interdisziplinarität der Informatik durchaus berechtigt (GI, 2008, S. 10).

Die folgende Beispielaufgabe besitzt Bezüge zur Mathematik (vgl. Thüringer Abiturprüfung, Leistungsfach Informatik, 2003):

a) In Programmiersprachen wie Oberon oder Turbo Pascal ist der Begriff des Datentyps von grundlegender Bedeutung. Erläutern Sie, was man unter einem Datentyp versteht. Wählen Sie von den einfachen Datentypen INTEGER, REAL, CHAR und BOOLEAN zwei aus und geben Sie für diese Datentypen wesentliche Operationen und Relationen sowie das Prinzip der internen Realisation an.

b) Ein Galtonbrett besteht aus mehreren Reihen Hindernissen, die versetzt untereinander angeordnet sind. In jeder Reihe kommt ein Hindernis dazu. Eine Kugel fällt von oben nach unten durch das Galtonbrett. Bei jedem Hindernis setzt die Kugel mit einer Wahrscheinlichkeit von jeweils 0,5 ihren Weg links bzw. rechts fort. Unter den Zwischenräumen der letzten Reihe befinden sich Behälter. Die Kugel wird in einem der Behälter aufgefangen. Die folgende Abbildung zeigt den Weg einer Kugel durch ein Galtonbrett mit drei Reihen.



Entwerfen Sie eine Funktion, die eine Zufallszahl aus der Menge {0, 1} ermittelt. Beide Zahlen sollen mit gleicher Wahrscheinlichkeit gezogen werden. Implementieren Sie die Funktion in Oberon oder Turbo Pascal und testen Sie die Funktion. Dokumentieren Sie die Tests.

Entwerfen Sie ein Programm, das die Arbeitsweise eines Galtonbretts simuliert. Durch das Galtonbrett sollen n Kugeln fallen. Das Programm liest die Anzahl r an Reihen und die Anzahl n an Kugeln ein und gibt aus, wie viele Kugeln in den verschiedenen Behältern aufgefangen wurden. Es gilt $1 \leq r \leq 10$ und $1 \leq n \leq 20000$. Implementieren Sie das Programm in Oberon oder Turbo Pascal. Dokumentieren Sie die Ergebnisse der Programmabarbeitung für mehrere Werte von r und n.

c) Entwerfen Sie ein Programm, das Binomialkoeffizienten bino(n, k) mithilfe der folgenden Erläuterungen berechnet. n und k sind ganze Zahlen mit $n \geq k \geq 0$. Sind $k = 0$ oder $n = k$, so gilt $bino(n,k) = 1$. Ansonsten gilt $bino(n,k) = bino(n-1, k-1) + bino(n-1, k)$. Implementieren Sie das Programm in Oberon oder Turbo Pascal. Testen Sie das Programm für mehrere Werte von n und k. Dokumentieren Sie die Tests. Erläutern Sie an dem implementierten Programm, was man unter rekursiven Unterprogrammen versteht.

Die Komplexaufgabe beinhaltet theoretische (Datentypen und Rekursion) und praktische Aspekte (Entwerfen und Implementieren von Computerprogrammen). Schwerpunktmäßig geht es um das Modellieren von Abläufen mit Algorithmen. Der Prüfungsteilnehmer wird durch die Aufgabenstellung explizit aufgefordert, die Programme zu testen und die Tests zu dokumentieren. Er muss die Tests z. T. selbst konstruieren. So könnte er die Funktion zum Ermitteln einer Zufallszahl 10.000-mal aufrufen lassen und mitzählen, wie häufig eine 0 bzw. 1 von der Funktion zurückgeliefert wird.

Vorbereitung der Prüfungsteilnehmer auf die Abiturprüfung

Damit die Prüfung nicht zum Glücksspiel wird, sollten die Anforderungen für die Schülerinnen und Schüler transparent sein. Schüler und Lehrer müssen rechtzeitig vor der Abiturprüfung wissen:

- welche Inhalts- und Prozesskompetenzen geprüft werden,
- welchen Schwierigkeits- und Komplexitätsgrad die Abituraufgaben besitzen,
- ob es Pflicht- oder Wahlaufgaben gibt und
- welche Hilfsmittel zugelassen sind.

Es empfiehlt sich, die potenziellen Prüfungsteilnehmer längerfristig und regelmäßig auf die Prüfungssituation vorzubereiten. Dafür einige Möglichkeiten:

- Im Unterricht werden Abituraufgaben der Vorjahre bearbeitet. Das Lesen und Verstehen zentral gestellter Aufgaben wird geübt. Die Schülerinnen und Schüler müssen dabei erfassen, was wirklich von ihnen verlangt wird.

- Es wird besprochen, welche Angaben ein Entwurf enthalten muss und wie die Bewertungseinheiten auf Entwurf und Implementierung aufgeteilt werden.
- Kursarbeiten werden unter prüfungsähnlichen Bedingungen durchgeführt.
- Operatoren für Schülertätigkeiten werden thematisiert und in Kursarbeiten verwendet.
- Strategien zur Auswahl der zu bearbeitenden Wahlaufgaben werden entwickelt und in Kursarbeiten geübt (Beitrag zur Selbstkompetenz).
- Den Schülerinnen und Schülern wird vor der Abiturprüfung unter Bezugnahme zum Informatiklehrplan eine Liste von Fachbegriffen übergeben. Oder besser noch: Die Schüler legen sich eine solche Liste selbst an.

Zur dezentralen Dimension eines Zentralabiturs

Dezentrale und zentrale Abiturprüfungen besitzen jeweils spezifische Eigenschaften. Sinnvolle Kompromisse zwischen beiden Formen mit dem Ziel einer dezentralen Dimension des Zentralabiturs sind denkbar. Die folgenden konkreten Ideen werden genannt:

1. *Vergleichsweise offene Aufgaben*
   Der Einsatz von Erst- und Zweitkorrektor soll eine korrekte Bewertung der Prüfungsleistung gewährleisten.
2. *Freiheiten bei der Wahl der Werkzeuge*
   Um ein Beispiel zu nennen: Prüfungsteilnehmern kann durchaus freigestellt werden, ob eine Wertetabelle von einem selbst entwickelten Computerprogramm oder von einer Tabellenkalkulation erzeugt wird.
3. *Teilaufgaben, die von dem Land oder der Schule selbst gestellt werden*
   siehe Kapitel 8.
4. *Knappe Hinweise zur Korrektur und Bewertung*
   Dies kann sinnvolle Spielräume beim Korrigieren und Bewerten der Schülerantworten auf der Grundlage des konkret realisierten Unterrichts eröffnen.
5. *Maximalpunkte eher „großschrittig" vorgeben*
   Also besser 8 Bewertungseinheiten für die gesamte Aufgabe vorgeben als viermal 2 Punkte für die einzelnen Teilaufgaben. Auch dies kann sinnvolle Spielräume eröffnen.
6. *Abstimmung zwischen Erst- und Zweitkorrektor*
   Dadurch ist gewährleistet, dass der konkret realisierte Unterricht bei der Korrektur und Bewertung angemessen berücksichtigt werden kann.

Dies stellt kein Gesamtpaket dar, das nur insgesamt umzusetzen ginge. Über die einzelnen Punkte ist sicher im Detail zu diskutieren. Beginnen wir mit dem ersten: Es darf nicht so weit kommen, in Abiturprüfungen vorwiegend offene Aufgaben stellen zu wollen, denn diese Aufgaben sind häufig den EPA-Anforderungsbereichen II oder sogar III zuzuordnen. Ein ausgewogenes Verhältnis der Anforderungsbereiche I, II und III in den Aufgabenstellungen ist auch weiterhin

zu gewährleisten (siehe Kapitel 8). Nachfolgend sind einige Aufgaben aus Thüringer Abiturprüfungen angegeben, für die Offenheit kennzeichnend ist.

Eine Aufgabe aus dem Grundfach 2006:

> Sie sollen 20 Computerarbeitsplätze miteinander vernetzen. Entscheiden Sie sich für eine Topologie und ein Konzept des Netzes. Begründen Sie Ihre Entscheidungen.

Eine Aufgabe aus dem Grundfach 2007:

> Diskutieren Sie eine Möglichkeit, mit deren Hilfe die Gültigkeit eines empfangenen RC5-Codes überprüft werden kann.

Eine Aufgabe aus dem Grundfach 2008:

> Entscheiden Sie sich für eine geeignete Datenstruktur. Geben Sie für die Datenstruktur eine Typvereinbarung an. Begründen Sie Ihre Entscheidung.

Eine Aufgabe aus dem Grundfach 2009:

> Geben Sie einen Algorithmus an, der bei einem Rechnernetz die kürzeste Gesamtlänge der Netzkabel ermittelt.

Eine Aufgabe aus dem Grundfach 2010:

> Setzen Sie sich mit dem folgenden Zitat von Pablo Picasso auseinander: »Computer sind nutzlos, sie können uns nur Antworten geben.«

Diese Aufgaben sind jeweils in umfassendere Aufgabenstellungen eingebettet.

Wir lernen, was wir vergessen sollten,
und vergessen, was wir lernen sollen.
Karl Friedrich Wilhelm Wander

16 Erfahrungen aus dem Thüringer Zentralabitur

Fragestellungen

In einer Befragung von Informatiklehrerinnen und -lehrern, die vom Thüringer Kultusministerium (jetzt Thüringer Ministerium für Bildung, Wissenschaft und Kultur) genehmigt wurde, ging es um das Erschließen von Erfahrungen, die mit dem Zentralabitur im Grund- und Leistungsfach Informatik in Thüringen gewonnen wurden. Die Ergebnisse der Befragung sollen in die länderübergreifend geführte Diskussion zum Zentralabitur einfließen. Konkret wurden der Computereinsatz in der Abiturprüfung, das Niveau von Grundkurs-Abituraufgaben, die Vorbereitung der Prüfungsteilnehmer auf die Abiturprüfung sowie die Korrektur und Bewertung von Abiturarbeiten thematisiert. Am Rande wurde der Frage nachgegangen, ob der Zugang zum Internet und die Nutzung von Mobiltelefonen in Prüfungen künftig erlaubt sein sollten. In der Befragung wurden auch Regelungen, wie sie für das Thüringer Zentralabitur im Grund- und Leistungsfach Informatik gelten, hinterfragt.

Alle Thüringer Gymnasien und Gesamtschulen (insgesamt ca. 100) wurden von der Universität Jena angeschrieben. Die Informatiklehrerinnen und -lehrer an den Schulen wurden gebeten, im Februar/März 2009 einen Fragebogen auszufüllen. Es beteiligten sich 57 Informatiklehrerinnen und -lehrer aus 39 Schulen. Bestimmungen des Datenschutzes wurden von der Universität Jena selbstverständlich beachtet, Rückschlüsse auf die einzelne Person oder Schule wurden also nicht gezogen. In diesem Kapitel wird eine vorläufige Auswertung der Befragung dargestellt.

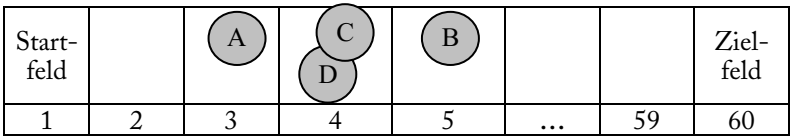
Abituraufgaben aus dem Grundfach Informatik

Dem Fragebogen an die Thüringer Gymnasien und Gesamtschulen wurde eine Zusammenstellung von Thüringer Abituraufgaben aus den Jahren von 2003 bis 2008 beigefügt[48]. Bei den Aufgaben handelt sich um Wahlaufgaben im Grundfach Informatik, und zwar um die Wahlaufgabe 4, die mithilfe des Computers zu lösen ist. Bei jeder Wahlaufgabe konnten 20 Bewertungseinheiten erreicht werden (von jeweils 60 BE für die gesamte Abiturprüfung).

[48] Die Texte der Aufgaben wurden redaktionell bearbeitet (konkrete Programmiersprachen weggelassen, Überschriften zu den Aufgaben ergänzt u. a.).

1. Aufgabe: Spiel für vier Personen

An einem Spiel nehmen vier Personen teil. Wir nennen diese Personen A, B, C und D. Für das Spiel werden ein Spielbrett, eine Münze und für jeden Spieler ein Spielstein benötigt. Das Spielbrett ist eine lineare Anordnung von 60 Feldern. Die Felder tragen die Nummern von 1 bis 60. Feld Nr. 1 ist das Startfeld, Feld Nr. 60 das Zielfeld. Die Münze hat eine Vorder- und eine Rückseite.



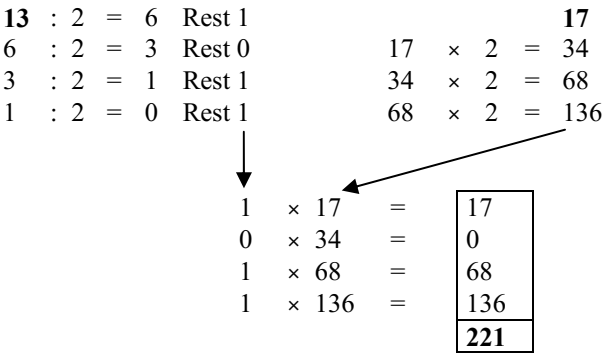
Zu Beginn des Spiels stehen alle Spielsteine auf dem Startfeld.

In jeder Spielrunde wird die Münze von den Spielern in der Reihenfolge A, B, C und D geworfen. Liegt die Vorderseite der Münze oben, so wird der Spielstein des jeweiligen Spielers um zwei Felder in Richtung Zielfeld gerückt. Liegt die Rückseite oben, wird der Spielstein um ein Feld in Richtung Startfeld gerückt. Der Spielstein wird nicht gerückt, wenn er sich auf dem Startfeld befindet und die Rückseite der Münze oben liegt oder wenn er sich auf dem Feld Nr. 59 befindet und die Vorderseite oben liegt. Das Spiel wird beendet, wenn in einer Spielrunde mindestens ein Spieler das Zielfeld erreicht. Gewinner des Spiels sind alle Spieler, deren Spielsteine am Ende dieser Spielrunde auf dem Zielfeld stehen.

Entwerfen und implementieren Sie ein Programm, das dieses Spiel für vier Personen simuliert. Das Rücken der Spielsteine soll während des gesamten Spiels auf dem Monitor dargestellt werden. Der Zufallszahlengenerator des Programmiersystems ist zur Simulation des Werfens der Münze zu verwenden.

2. Aufgabe: Bauernmultiplikation

Am Beispiel 13 × 17 = 221 wird im folgenden Schema die russische Bauernmultiplikation dargestellt:



Geben Sie zur Lösung der Multiplikationsaufgabe 166 × 125 alle Lösungsschritte an. Verwenden Sie das dargestellte Schema.

Entwerfen Sie ein Programm, das zwei Faktoren einliest und das Produkt nach der russischen Bauernmultiplikation berechnet und ausgibt. Implementieren Sie das Programm.

3. Aufgabe: Tonleiter

Die chromatische Tonleiter besteht aus zwölf Tönen. Schreibt man diese Tonleiter zweimal hintereinander auf, so entsteht die aufsteigende Tonfolge:

c cis d dis e f fis g gis a ais h c cis d dis e f fis g gis a ais h

Von einem Ton dieser Tonfolge zum nächsten führt ein Halbtonschritt und zum übernächsten Ton ein Ganztonschritt.

Jede Dur- bzw. Moll-Tonleiter besteht aus acht aufsteigend geordneten Tönen der gegebenen Tonfolge. Der erste Ton ist der Grundton.

Für eine Dur-Tonleiter gilt:

- Vom ersten Ton zum zweiten, vom zweiten zum dritten, vom vierten zum fünften, vom fünften zum sechsten und vom sechsten zum siebenten führt je ein Ganztonschritt.
- Vom dritten Ton zum vierten und vom siebenten zum achten führt je ein Halbtonschritt.

Für eine Moll-Tonleiter gilt:

- Vom ersten Ton zum zweiten, vom dritten zum vierten, vom vierten zum fünften, vom sechsten zum siebenten und vom siebenten zum achten führt je ein Ganztonschritt.
- Vom zweiten Ton zum dritten und vom fünften zum sechsten führt je ein Halbtonschritt.

Beispiele:

- Die g-Dur-Tonleiter besteht aus den Tönen g a h c d e fis g.
- Die h-Moll-Tonleiter besteht aus den Tönen h cis d e fis g a h.

Entwerfen und implementieren Sie ein Programm, das für jeden der Grundtöne c, g, d, a, e, h, fis die Dur-Tonleiter und für jeden der Grundtöne a, e, h, fis, cis, gis die Moll-Tonleiter ermittelt und ausgibt.

4. Aufgabe: Daten-Auswertung

An einem Minigolf-Turnier nehmen maximal 50 Spieler teil. Zu Beginn des Turniers erhält jeder Spieler einen Spielschein, auf dem sein Vor- und Nachname stehen. Danach muss jeder Spieler auf 18 verschiedenen Bahnen spielen. Pro

Bahn stehen dem Spieler bis zu sechs Schläge zur Verfügung. Für jeden ausgeführten Schlag auf einer Bahn erhält der Spieler einen Punkt. Hat der Spieler den Ball nach dem sechsten Schlag nicht in das Ziel der jeweiligen Bahn bekommen, erhält er einen zusätzlichen Punkt. Die auf jeder Bahn erzielten Punkte werden auf seinem Spielschein notiert. Nachdem der Spieler die 18 Bahnen bespielt hat, gibt er seinen Spielschein beim Turnierleiter ab. Der Turnierleiter gibt den Vor- und Nachnamen sowie die auf den Bahnen erzielten Punkte des Spielers in einen Computer ein.

In der Tabelle sind die Spielscheine von zehn Spielern enthalten.

Beispiel:

	10 Spielerscheine									
	Eva Sauer	Anna Kummer	Bernd Winter	Kai Hecht	Lutz Jahn	Fred Ammer	Susi Reuter	Karl Meier	Hans Beyer	Dirk Hunger
Bahn 1	3	5	4	3	3	7	4	5	4	3
Bahn 2	4	5	5	4	3	6	5	4	5	6
Bahn 3	3	4	4	4	4	5	5	4	6	4
Bahn 4	5	3	5	7	5	5	4	6	4	5
Bahn 5	6	4	3	5	6	4	3	6	4	5
Bahn 6	7	3	5	6	6	4	3	3	5	4
Bahn 7	3	5	6	3	4	5	6	4	3	4
Bahn 8	4	7	5	4	5	5	6	7	3	5
Bahn 9	5	4	5	4	5	6	7	5	5	6
Bahn 10	3	4	4	5	3	7	4	6	4	4
Bahn 11	5	3	4	6	6	5	4	5	6	4
Bahn 12	6	5	6	6	3	5	6	7	5	7
Bahn 13	7	7	6	5	3	7	6	5	7	6
Bahn 14	3	4	3	4	4	6	4	4	5	6
Bahn 15	4	4	3	5	5	6	5	4	4	4
Bahn 16	3	5	4	4	4	5	5	3	3	4
Bahn 17	4	3	6	6	5	5	4	7	5	5
Bahn 18	5	5	5	5	6	3	6	5	4	6

Nach dem Abschluss der Eingabe aller Daten wertet der Computer die Daten wie folgt aus:

- Für jeden Spieler wird die Gesamtpunktzahl errechnet. Die Gesamtpunktzahl ergibt sich als Summe der auf den 18 Bahnen erzielten Punkte.

- Auf dem Monitor des Computers werden die Gesamtpunktzahlen der Spieler in aufsteigender Reihenfolge sortiert untereinander ausgegeben. Neben jeder Gesamtpunktzahl stehen Vor- und Nachname des Spielers.

Entwerfen und implementieren Sie ein Programm, mit dem ein Computer die Daten der Spieler wie beschrieben auswertet.

Testen Sie Ihr Programm. Dokumentieren Sie den Test.

5. Aufgabe: Ver- und Entschlüsseln

Gegeben ist der Geheimtext 122434243422454244. Dieser Geheimtext wurde mithilfe der Polybios-Tafel erzeugt:

	1	2	3	4	5
1	a	b	c	d	e
2	f	g	h	i/j	k
3	l	m	n	o	p
4	q	r	s	t	u
5	v	w	x	y	z

Beachten Sie, dass sich die Buchstaben i und j in der Polybios-Tafel ein Feld teilen.

Entschlüsseln Sie den Geheimtext mithilfe der Polybios-Tafel. Geben Sie den Klartext an.

Können Nachrichten, die mithilfe der Polybios-Tafel verschlüsselt wurden, ohne Kenntnis der Polybios-Tafel entschlüsselt werden? Begründen Sie Ihre Antwort.

Verschlüsseln Sie den Klartext »abiturjahrgang« mithilfe der Polybios-Tafel. Geben Sie den Geheimtext an.

Entwerfen und implementieren Sie ein Programm, das Folgendes leistet:

- Einlesen eines Klartextes
- Verschlüsseln des Klartextes mithilfe der Polybios-Tafel
- Ausgabe des Geheimtextes

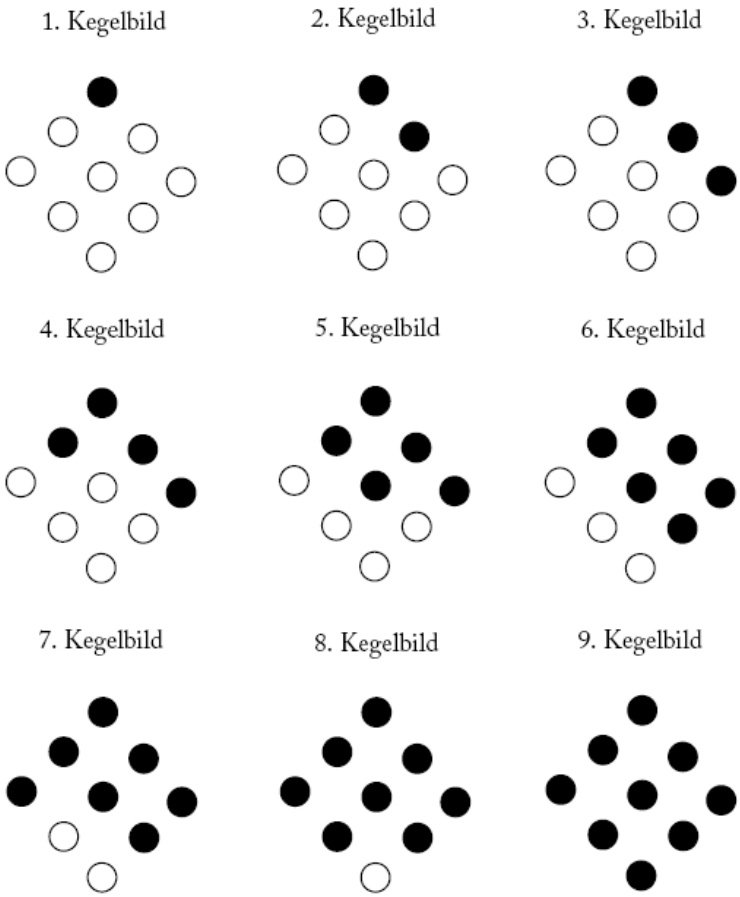
Beachten Sie folgenden Hinweis: Sie können beim Lösen der Aufgabe davon ausgehen, dass ein Klartext nur aus den in der Polybios-Tafel angegebenen Kleinbuchstaben besteht.

Testen Sie das Programm mit dem Klartext »abiturjahrgang«. Dokumentieren Sie den Test.

6. Aufgabe: Kegeln

Beim Abi-Kegeln gibt es folgende Spielvariante:
- Das Spiel hat mindestens zwei und höchstens fünf Teilnehmer.
- Die Reihenfolge, in der die Teilnehmer spielen, wird vor Beginn des Spiels festgelegt.
- Die Teilnehmer spielen nacheinander auf einer Kegelbahn.
- Es werden insgesamt neun verschiedene Kegelbilder aufgestellt.
- Jeder Teilnehmer muss auf jedes Kegelbild genau einmal spielen.
- Jeder Teilnehmer spielt genau eine Kugel pro Kegelbild.

In der folgenden Abbildung sind die neun verschiedenen Kegelbilder dargestellt:



Im ersten Kegelbild soll ein Kegel, im zweiten sollen zwei, im dritten drei, im vierten vier, im fünften fünf, im sechsten sechs, im siebenten sieben, im achten acht und im neunten neun Kegel abgeräumt werden.

Der Teilnehmer erhält nach seinem Wurf eine Punktzahl. Die Punktzahl errechnet sich aus der Anzahl der gefallenen Kegel multipliziert mit einem Faktor. Dieser Faktor ist im ersten Kegelbild 9, im zweiten 8, im dritten 7, im vierten 6, im fünften 5, im sechsten 4, im siebten 3, im achten 2 und im neunten 1.

Beispiel:

Beim vierten Kegelbild kann ein Teilnehmer 24 Punkte erhalten, wenn bei seinem Wurf alle vier Kegel gefallen sind.

Entwerfen und implementieren Sie ein Programm, das Folgendes leistet:

- Die Anzahl der Teilnehmer am Spiel und die Namen der Teilnehmer werden eingelesen.
- Für jedes Kegelbild werden die Nummer des Kegelbildes, der Name jedes Teilnehmers, die Anzahl seiner gefallenen Kegel und seine Punktzahl ausgegeben.
- Zum Abschluss des Spiels werden die Gesamtpunktzahl und der Name jedes Teilnehmers ausgegeben.

Beachten Sie folgende Festlegung:

- Jede Anzahl gefallener Kegel ist mithilfe des Zufallsgenerators zu ermitteln, wobei jedes Ereignis die gleiche Wahrscheinlichkeit haben soll.

Der Fragebogen und dessen Auswertung

Schwierigkeitsgrad der Aufgaben

Sind die beiliegenden Prüfungsaufgaben vom Schwierigkeitsgrad her für eine Abiturprüfung im Grundfach Informatik angemessen?

Die Antworten auf diese Frage sind in der Tabelle 16.1 zusammengefasst. Alle Zahlenangaben erfolgen in Prozent.

Die Aufgabe ist	*zu leicht*	*angemessen*	*zu schwer*
1. Aufgabe – Spiel für vier Personen	0,0	77,2	22,8
2. Aufgabe – Bauernmultiplikation	7,0	84,2	8,8
3. Aufgabe – Tonleiter	7,0	54,4	38,6
4. Aufgabe – Daten-Auswertung	8,8	59,6	31,6
5. Aufgabe – Ver- und Entschlüsseln	3,5	82,5	14,0
6. Aufgabe – Kegeln	1,8	63,1	35,1

Tabelle 16.1:
Einschätzung des Schwierigkeitsgrades von Abituraufgaben im Grundfach Informatik.

Die sechs Aufgaben werden also von einer (einfachen) Mehrheit der Teilnehmer an der Befragung vom Schwierigkeitsgrad her als angemessen eingeschätzt. Die Zustimmung fällt jedoch unterschiedlich aus. Sie reicht von nur 54,4 % bei der 3. Aufgabe bis immerhin 84,2 % bei der 2. Aufgabe. Die 3., 4. und 6. Aufgabe werden von 31,6 bis 38,6 % der Teilnehmer, also von rund einem Drittel, als zu schwer betrachtet. Bei der 3. und 4. Aufgabe ist die Bewertung vergleichsweise breit gestreut.

Eine Interpretation der Ergebnisse ist nicht einfach, was die folgenden Argumente deutlich machen:

- Das Einschätzen des Schwierigkeitsgrades von Aufgaben, die mithilfe von Computern zu bearbeiteten sind, ist prinzipiell nicht einfach – insbesondere dann, wenn man die Aufgabe selbst (noch) nicht gelöst hat.
- Nach dem Lehrplan legen die Lehrerinnen und Lehrer die konkret mit einem Informatiksystem zu bearbeitenden Aufgaben und Probleme weitgehend selbst fest (TKM, 1999, S. 18-20). Die Einschätzung des Schwierigkeitsgrades einer Abituraufgabe erfolgt sicher aus der Perspektive des eigenen Unterrichts mit seinen inhaltlichen Schwerpunktsetzungen, sodass eine gewisse Bandbreite in der Einschätzung verständlich wird.

- Einige Teilnehmer ergänzten von sich aus, also ohne besondere Aufforderung dazu, auf dem Fragebogen Bemerkungen zu einzelnen Aufgaben. 12,3 % der Teilnehmer gaben an (als zusätzliche Bemerkung), dass die 4. Aufgabe zu umfangreich war. Auch bei der 6. Aufgabe gaben 7,0 % einen solchen Kommentar ab. Es ist zu vermuten, dass beide Aufgaben eigentlich nicht für zu schwer, sondern eher für zu aufwendig gehalten werden. Als Konsequenz sollte künftig bei der Einschätzung von Prüfungsaufgaben nicht nur isoliert nach deren Schwierigkeit, sondern zumindest auch nach dem Arbeitsumfang zur Bearbeitung gefragt werden (siehe auch Schlüter, 2009).
- Die an Prüfungen unmittelbar und mittelbar Beteiligten besitzen vermutlich eine gewisse Tendenz, Prüfungsaufgaben im Zweifelsfall als zu schwierig zu charakterisieren.

Ein vorläufiges Resümee wird formuliert: Die sechs Prüfungsaufgaben sind vom Schwierigkeitsgrad her als Wahlaufgaben in einer schriftlichen Abiturprüfung im Grundfach Informatik als angemessen anzusehen.

In Interviews mit Informatiklehrerinnen und -lehrern könnte diese Position einer Überprüfung unterzogen werden. Im Mittelpunkt der Interviews könnten Programmentwürfe und Musterlösungen zu den sechs Aufgaben stehen. Nachteilig bei Interviews ist die nicht vorhandene Anonymität der Gesprächsteilnehmer. Hinzuweisen ist auch darauf, dass der Schwierigkeitsgrad einer Aufgabe im Kontext der gesamten Prüfungsaufgabe zu betrachten ist. Für den Prüfungsteilnehmer ist vor allem von Bedeutung, ob die Prüfung als Ganzes einen angemessenen Schwierigkeitsgrad hatte.

In den nachfolgenden Tabellen haben die verwendeten Symbole die folgende Bedeutung:

- - ich stimme überhaupt nicht zu
- ich stimme nicht zu
0 unentschlossen
+ ich stimme etwas zu
+ + ich stimme voll zu

Computereinsatz in der Abiturprüfung

Nr.	*Aussage*	- -	-	0	+	+ +
1	Der Einsatz von Computern in der schriftlichen Abiturprüfung im Grundfach Informatik ist wichtig.	0,0	3,5	8,8	40,3	47,4
2	Der Einsatz von Computern in der schriftlichen Abiturprüfung im Leistungsfach Informatik ist wichtig.	0,0	0,0	3,8	13,5	82,7
12	Der Computereinsatz stellt in einer Prüfungssituation ein zu großes Risiko dar, da kleine Fehler zu einem großen Zeitverlust führen können.	14,2	41,1	17,9	21,4	5,4

Tabelle 16.2:
Umfrageergebnisse zu den Aussagen 1, 2 und 12.

Immerhin 96,2 % der Teilnehmer an der Befragung betrachten den Computereinsatz in einer schriftlichen Abiturprüfung im Leistungsfach Informatik für wichtig oder sogar für sehr wichtig (siehe Tabelle 16.2). Für das Grundfach Informatik sind dies immerhin noch 87,7 %. Aussage 12 konfrontiert durchaus suggestiv mit einer möglichen Auswirkung, dass nämlich bei der Arbeit am Computer kleine Fehler zu einem großen Zeitverlust führen können. 55,3 % der Teilnehmer widersprechen dennoch der Behauptung, dass der Computereinsatz in der Prüfung daher zu risikobehaftet wäre.

Nr.	*Aussage*	- -	-	0	+	+ +
4	Die beiliegenden Abituraufgaben sind als Übungsaufgaben für den Unterricht im Grundfach Informatik geeignet.	3,6	3,6	12,8	40,0	40,0
8	Die beiliegenden Abituraufgaben sind für die Verwendung in Kursarbeiten im Grundfach Informatik geeignet.	8,9	35,7	16,1	28,6	10,7
11	Zur Vorbereitung auf die schriftliche Abiturprüfung muss das Anfertigen von Programmentwürfen regelmäßig im Unterricht geübt werden. Ansonsten geht das in der Abiturprüfung schief.	1,8	0,0	1,8	25,0	71,4

Tabelle 16.3:
Umfrageergebnisse zu den Aussagen 4, 8 und 11.

Die Eignung der beigefügten Abituraufgaben als Übungsaufgaben für den Unterricht im Grundfach Informatik sehen 80,0 % der Teilnehmer an der Befragung zumindest teilweise als gegeben an (siehe Tabelle 16.3). Anders wird die Einsatzmöglichkeit der Aufgaben in Kursarbeiten betrachtet: 44,6 % der Teilnehmer sehen dies (eher) negativ und nur 39,3 % (eher) positiv. Eine deutliche Mehrheit von immerhin 71,4 % der Teilnehmer ist unbedingt der Auffassung, dass das Anfertigen von Programmentwürfen als regelmäßige Prüfungsvorbereitung zu üben ist.

Nr.	*Aussage*	- -	-	0	+	+ +
3	Es ist sinnvoll, dass die Hinweise zur Korrektur und Bewertung im Fach Informatik keine Musterlösungen enthalten.	24,5	21,1	28,1	14,0	12,3
5	Es ist sinnvoll, dass bei den beiliegenden Abituraufgaben nur vorgegeben ist, dass für die gesamte Aufgabe 20 Bewertungseinheiten (BE) erreicht werden können. Eine verbindliche Aufteilung der BE auf Teilaufgaben wäre nicht sinnvoll.	7,1	12,5	16,1	26,8	37,5
6	Vom Prüfungsteilnehmer muss die Abgabe eines Programmentwurfes verlangt werden, damit Korrektur und Bewertung seiner Lösung präzise erfolgen können.	1,8	10,5	12,3	31,6	43,8
7	Bei Aufgaben, die mit dem Computer zu bearbeiten sind, sollte vom Prüfungsteilnehmer stets auch verlangt werden, über den Lösungsweg zu reflektieren (z. B. mit der Aufforderung, die bei der Arbeit verwendeten Methoden der Softwareentwicklung anzugeben).	3,5	14,3	28,6	42,8	10,8
9	Die Hinweise zur Korrektur und Bewertung enthalten bisher keine Angaben darüber, wie die Bewertungseinheiten auf Entwurf, Implementierung und ggf. Reflexion aufgeteilt werden. Das ist sinnvoll, denn dadurch können die Regelungen, die für die Kursarbeiten im jeweiligen Kurs galten, auch in der Abiturprüfung angewandt werden.	3,6	7,3	14,5	40,0	34,6

Nr.	*Aussage*	- -	-	0	+	+ +
10	Kurz gefasste Hinweise zur Korrektur und Bewertung eröffnen sinnvolle Spielräume bei der Korrektur und Bewertung auf der Grundlage des wirklich stattgefundenen Unterrichts.	1,8	5,5	10,9	34,5	47,3

Tabelle 16.4:
Umfrageergebnisse zu den Aussagen 3, 5-7, 9, 10.

Die folgenden Positionen aus dem Thüringer Zentralabitur (Grund- und Leistungsfach Informatik) werden von den Teilnehmern an der Befragung mehrheitlich mitgetragen (in Klammern ist jeweils die Nr. der Aussage angegeben; siehe Tabelle 16.4):

- Kurz gefasste Hinweise zur Korrektur und Bewertung (Aussage 10),
- keine verbindliche Festlegung, wie die Bewertungseinheiten auf Entwurf, Implementierung und ggf. Reflexion aufgeteilt werden (Aussage 9), und
- keine verbindliche Aufteilung der Bewertungseinheiten auf Teilaufgaben (Aussage 5).

Die Bereitstellung von Musterlösungen wird jedoch erwartet (Aussage 3). Es ist zu vermuten, dass die Teilnehmer Musterlösungen als wesentliche Hilfe zur Charakterisierung des Anforderungsniveaus ansehen. Die Notwendigkeit der Abgabe eines Programmentwurfs wird mitgetragen (Aussage 6). Kein so eindeutiges Bild ergibt sich bei der Forderung, dass der Prüfungsteilnehmer bei Aufgaben, die mit dem Computer zu bearbeiten sind, stets auch über den Lösungsweg reflektieren sollte (Aussage 7).

Zugang zum Internet, Nutzung von Mobiltelefonen in Prüfungen

In einem Modellversuch an einer Schule in Australien sind in Prüfungen auch der Zugang zum Internet und die Nutzung von Mobiltelefonen erlaubt[49]*. Würden Sie sich dies auch für das Thüringer Zentralabitur im Fach Informatik wünschen? Bitte begründen Sie Ihre Antwort.*

50 Teilnehmer waren dagegen, sechs Teilnehmer waren (zumindest teilweise) dafür, einer kreuzte nichts an. Sicher sind die Begründungen gerade der Minorität an Teilnehmern von besonderem Interesse, die zumindest teilweise dafür waren. Daher sind diese nachfolgend zitiert:

- Internet ja (modernes Hilfsmittel in der heutigen Zeit), Mobiltelefon nein (es ist nicht nachvollziehbar, welcher »Spezialist« angerufen wurde).

[49] DER SPIEGEL 36/2008, S. 53

- Für einen guten Softwareentwurf und für eine angemessene Implementierung bedarf es einiger Übung, beides ist nicht so leicht aus dem Internet im Prüfungsfall zu holen. Als Nachschlagewerk für eine korrekte Syntax bzw. zur Auswahl geeigneter Methoden, Klassen usw. halte ich das Internet allerdings für geeignet und sinnvoll. Ebenso als Nachschlagewerk der korrekten grafischen Notation der für den Softwareentwurf verwendeten Modelle.
- Für einen Teil des Abiturs sinnvoll (ähnlich wie ein CAS in Mathematik), aber auch ein Teil ohne Hilfsmittel ist wichtig (Grundwissen, Algorithmen). Also ca. 50 % mit und 50 % ohne PC/Handy.
- Es kommt heute und in Zukunft eher darauf an, sich neues Wissen aus Quellen zu erschließen. Für eine Argumentation zum Thema »Computer und Arbeitswelt« könnten z. B. neue Statistiken gesucht und verwendet werden.
- Internet ja – warum nicht? Mobiltelefon nein – stört zu sehr!
- Bei entsprechender Aufgabenstellung ist das möglich. Das Ziel ist es ja, ein Problem zu lösen. Dafür benutzt man außerhalb der Prüfung auch verschiedene »Medien«.

Am 23. November 2009 berichtete die Süddeutsche Zeitung, dass die Schüler in Dänemark in ihren Abschlussprüfungen das Internet zu Rate ziehen dürfen[50].

Resümee

Die Ergebnisse der Befragung geben einige konkrete Hinweise zur sinnvollen Ausgestaltung von zentralen Abiturprüfungen im Fach Informatik und zu Konsequenzen für die Unterrichtsgestaltung. Das in Thüringen im Fach Informatik praktizierte Vorgehen im Sinne einer gewissen dezentralen Dimension des Zentralabiturs erfährt insgesamt deutlich mehr Zustimmung als Ablehnung. Dies betrifft die folgenden Positionen: Knappe Hinweise zur Korrektur und Bewertung; Maximalpunkte eher »großschrittig« vorgeben (siehe S. 160 f.). Die Einführung von zentralen Abiturprüfungen sollte nach Möglichkeit wissenschaftlich begleitet werden. Dabei kann u. a. herausgefunden werden, welche Akzeptanz normative Setzungen besitzen und wie sich diese mit der Zeit aufgrund von Erfahrung und Gewöhnung (und damit einhergehender Sicherheit im Handeln) verändert. Es kann auch festgestellt werden, welche Unterstützung Lehrerinnen und Lehrer in ihrer Tätigkeit benötigen.

[50] http://www.sueddeutsche.de/jobkarriere/918/495246/text/

Referenzen

Barron, D. W.: Rekursive Techniken in der Programmierung. Leipzig: BSB B. G. Teubner Verlagsgesellschaft, 1971.

BMBF – Bundesministerium für Bildung und Forschung (Hrsg.); Klieme, E. u. a.: Zur Entwicklung nationaler Bildungsstandards – Expertise. Reihe »Bildungsreform«, Bd. 1. Berlin; Bonn: BMBF, 2003.

Bauer, F. L.: Entzifferte Geheimnisse – Methoden und Maximen der Kryptologie. Berlin; Heidelberg; New York: Springer, ^{2}1997.

Baumann, R.: Spiel, Idee und Strategie programmiert in Pascal. Reihe »CHIP-Wissen«. Würzburg: Vogel, 1983.

Baumann, R.: Didaktik der Informatik. Stuttgart; München; Düsseldorf; Leipzig: Klett, ^{2}1996.

Baumann, R.: Probleme der Aufgabenkonstruktion gemäß Bildungsstandards – Überlegungen zu Kompetenzstufen und Operatoren. In: LOG IN, 28. Jg. (2008), H. 153, S. 54–59.

Beutelspacher, A.: Kryptologie. Wiesbaden: Vieweg, ^{9}2009.

Beyrer, K.; Mathis, B.-S. (Hrsg.): So weit das Auge reicht – Die Geschichte der optischen Telegrafie. Eine Publikation des Museums für Post und Kommunikation, Frankfurt am Main, anlässlich der gleichnamigen Ausstellung. Karlsruhe: Braun, 1995.

Breier, N.: Ein Plädoyer für die Registermaschine. In: LOG IN, 9. Jg. (1989), H. 1, S. 29–32.

Claus, V.; Schwill, A.: Duden Informatik A–Z. Mannheim; Leipzig; Wien; Zürich: Dudenverlag, ^{4}2006.

Clocksin, W. F.; Mellish, C. S.: Programming in Prolog. Berlin; Heidelberg; New York: Springer, 2003.

Deschauer, S.: Das zweite Rechenbuch von Adam Ries. Braunschweig; Wiesbaden: Vieweg, 1992.

Dewdney, A. K.: Computer-Kurzweil. In: Spektrum der Wissenschaft, 7. Jg. (1984), H. 9, S. 8–13.

Dißmann, S.: Handlungsorientiertes Erlernen von Programmkonstruktionen anhand von Rollenspielen. In: Hubwieser (Hrsg.), 2003, S. 249–260.

Fothe, M.: Rechnen auf den Linien – Vier Spiele mit dem Computer. In: LOG IN, 21. Jg. (2001), H. 3/4, S. 48–53.

Fothe, M.: Problemlösen mit Python. Reihe »Materialien«, H. 72. Bad Berka: Thüringer Institut für Lehrerfortbildung, Lehrplanentwicklung und Medien, 2002.

Fothe, M.: Zeitverhalten von Sortierverfahren – Beispiele für experimentelles Arbeiten im Informatikunterricht. In: Hubwieser (Hrsg.), 2003, S. 111–120.

Fothe, M.: Rechnen auf den Linien – eine historische Betrachtung aus der Sicht der modernen Informatik. In: Roloff, H.; Weidauer, M. (Hrsg.): Wege zu

Adam Ries – Tagung zur Geschichte der Mathematik, Erfurt 2002. Augsburg: Rauner, 2004, S. 87–91.
Fothe, M.: Rekursion und Iteration – Voruntersuchung zu einem Test. In: Friedrich (Hrsg.), 2005, S. 207–218.
Fothe, M.: Ein Rollenspiel zum Verschlüsseln. In: LOG IN, 26. Jg. (2006), H. 138/139, S. 82–85.
Fothe, M.: Algorithmen in spielerischer Form. In: Stechert, P. (Hrsg.): Informatische Bildung in der Wissensgesellschaft. Reihe »Medienwissenschaften«, Bd. 6. Siegen: Universitätsverlag, 2007, S. 31–42.
Fothe, M.: Bildungsstandards Informatik für die Sekundarstufe II – Vorüberlegungen zur Entwicklung. In: Brinda, T. u. a. (Hrsg.): Didaktik der Informatik – Aktuelle Forschungsergebnisse. Reihe »GI-Edition LNI – Lecture Notes in Informatics«, Bd. P-135. Bonn: Köllen Verlag, 2008, S. 107–116.
Fothe, M.: Adam Ries und das Linienrechnen – ein historisches Thema für den Informatikunterricht. In: Koerber (Hrsg.), 2009, S. 330–339.
Fothe, M.; Ludwig, H.: Zu Möglichkeiten der externen Unterstützung von Informatiklehrerinnen und -lehrern in der gymnasialen Oberstufe. In: informatica didactica, Ausgabe Nr. 8 (2008).
Fothe, M.; Moldenhauer, W.; Thiele, O.: Von der Komplexität eines Zentralabiturs – Thüringer Erfahrungen im Grund- und Leistungsfach Informatik. In: LOG IN, 27. Jg. (2007), H. 148/149, S. 24–31.
Fothe, M.; Zimmermann, B. (Hrsg.): Zur Geschichte der Mathematik in Jena – Wurzeln strukturwissenschaftlichen Denkens. Schriftenreihe des Frege Centre for Structural Sciences; Bd. Nr. 1. Jena: IKS Garamond, 2009.
Friedrich, S.: Informatik und PISA – vom Wehe zum Wohl der Schulinformatik. In: Hubwieser (Hrsg.), 2003, S. 133–144.
Friedrich, S. (Hrsg.): Unterrichtskonzepte für informatische Bildung. INFOS 2005 – 11. GI-Fachtagung Informatik und Schule. Reihe »GI-Edition LNI – Lecture Notes in Informatics«, Bd. P-60. Bonn: Köllen Verlag, 2005.
Gallenbacher, J.: Abenteuer Informatik – IT zum Anfassen von Routenplaner bis Online-Banking. Heidelberg: Spektrum Akademischer Verlag, 2007.
GI – Gesellschaft für Informatik (Hrsg.): Grundsätze und Standards für die Informatik in der Schule: Bildungsstandards Informatik für die Sekundarstufe I. In: Beilage zu LOG IN, 28. Jg. (2008), H. 150/151.
Graßl, A.: Eine Osterregel nach dem »Immerwährenden Kalender«. In: MNU, 51. Jg. (1998), H. 3 (15.4.1998), S. 141–144.
Gruhn, V.; Chavan, S.: Das Jahr-2000-Problem – Projektmanagement von Datumsumstellungen in der Software. Bonn u. a.: Addison-Wesley, 1997.
Hartmann, W.; Näf, M.; Reichert, R.: Informatikunterricht planen und durchführen. Reihe »eXamen.press«. Berlin; Heidelberg; New York: Springer, 2006.
Herbarth, D.: Die Entwicklung der optischen Telegrafie in Preussen. Landeskonservator Rheinland, Arbeitsheft 15. Köln: Rheinland-Verlag, 1978.
Herrmann, U.: Wie lernen Lehrer ihren Beruf? Empirische Befunde und praktische Vorschläge. Weinheim; Basel: Beltz, 2002.

Hubwieser, P. (Hrsg.): Informatische Fachkonzepte im Unterricht. INFOS 2003 – 10. GI-Fachtagung Informatik und Schule. Reihe »GI-Edition LNI – Lecture Notes in Informatics«, Bd. P-32. Bonn: Köllen Verlag, 2003.
Hubwieser, P.: Didaktik der Informatik – Grundlagen, Konzepte, Beispiele. Reihe »eXamen.press«. Berlin; Heidelberg; New York: Springer, [3]2007.
ISB – Staatsinstitut für Schulqualität und Bildungsforschung München (Hrsg.): Glossar – Begriffe im Kontext von Lehrplänen und Bildungsstandards. München: ISB, 2006.
Kerner, I. O.: Numerische Mathematik und Rechentechnik. Teil II/2. Reihe »Mathematisch-Naturwissenschaftliche Bibliothek«, Bd. 47/2. Leipzig: BSB B. G. Teubner Verlagsgesellschaft, 1973.
Kerner, I. O.: Informatik. Reihe »Mathematik für Lehrer«, Bd. 9. Berlin: Deutscher Verlag der Wissenschaften, 1990.
KMK – Ständige Konferenz der Kultusminister der Länder in der Bundesrepublik Deutschland: Einheitliche Prüfungsanforderungen in der Abiturprüfung Informatik. Beschluss vom 1.12.1989 i. d. F. vom 5.2.2004. München; Neuwied: Wolters Kluwer Deutschland – Luchterhand, 2004.
Koerber, B. (Hrsg.): Zukunft braucht Herkunft – 25 Jahre »INFOS – Informatik und Schule«. INFOS 2009 – 13. GI-Fachtagung Informatik und Schule. Reihe »GI-Edition LNI – Lecture Notes in Informatics«, Bd. P-156. Bonn: Köllen Verlag, 2009.
Kohl, L.: Kompetenzorientierter Informatikunterricht in der Sekundarstufe I unter Verwendung der visuellen Programmiersprache Puck. Friedrich-Schiller-Universität Jena, Dissertation, 2009.
Kohl, L.; Fothe, M.: Algorithmen aus einer anderen Perspektive – Ein Vorschlag für ein Kompetenzmodell zum Inhaltsbereich »Algorithmen« der Bildungsstandards Informatik. In: LOG IN, 27. Jg. (2007), H. 146/147, S. 20–22.
Kohl, L.; Meißner, G.; Schmidt, H.: Puck – ein Sommernachtstraum. In: LOG IN 27. Jg. (2007), H. 144, S. 39–47.
Menninger, K.: Zahlwort und Ziffer – Eine Kulturgeschichte der Zahl. Bd. II. Göttingen: Vandenhoeck und Ruprecht, [3]1979.
Mössenböck, H.: Objektorientierte Programmierung in Oberon-2. Berlin; Heidelberg; New York: Springer, [3]1998.
Nelles, W.; Rhode, T.; Stechert, P.: Entwicklung eines Kompetenzrahmenmodells – Informatisches Modellieren und Systemverständnis. In: Informatik Spektrum, 33. Jg. (2010), H. 1, S. 45–53.
Niedermeier, R.; Vogel, J.; Fothe, M.; König, M. und (im Teil 2) Baumann, R.: Das Knotenüberdeckungsproblem – Eine Fallstudie zur Didaktik NP-schwerer Probleme. Teile 1 und 2. In: LOG IN, 27. Jg. (2007), H. 146/147, S. 53–59; H. 148/149, S. 81–89.
Ottmann, T.; Widmayer, P.: Algorithmen und Datenstrukturen. Heidelberg; Berlin: Spektrum Akademischer Verlag, [4]2002.
Rochhaus, P.: Adam Ries – Vater des modernen Rechnens. Erfurt: Sutton Verlag, 2008.

Röhner, G.: Informatik mit Prolog – Materialien zum Unterricht, Sekundarstufe II. Frankfurt am Main: Amt für Lehrerbildung, ^{3}2007.
Schärf, J.; Strecha, R.: UPN-Taschenrechner in Schule und Praxis. Wien; München: Oldenbourg, 1977.
Schlüter, K.: Eine Studie zu den Merkmalen der Aufgabenschwierigkeit am Beispiel eines Informatik-Schülerwettbewerbs. Erster Teil: Aufgabenklassifizierung. In: Koerber (Hrsg.), 2009, S. 181–192.
Schott, F.; Ghanbari, S. A.: Kompetenzdiagnostik, Kompetenzmodelle, kompetenzorientierter Unterricht. Zur Theorie und Praxis überprüfbarer Bildungsstandards; ComTrans – ein theoriegeleiteter Ansatz zum Kompetenztransfer als Diskussionsvorlage. Münster; New York; München; Berlin: Waxmann, 2008.
Schubert, S.; Schwill, A.: Didaktik der Informatik. Heidelberg; Berlin: Spektrum Akademischer Verlag, 2004.
Sykes, B.: Die sieben Töchter Evas – Warum wir alle von sieben Frauen abstammen: revolutionäre Erkenntnisse der Gen-Forschung. Bergisch Gladbach: Bastei Lübbe, 2003.
Thiele, O.; Kämmerer, M.; Helbig, J.; Merten, B.: Logik-orientiertes Programmieren mit Prolog – Unterricht, Übungen und Abitur. Reihe »Materialien«, H. 63. Bad Berka: Thüringer Institut für Lehrerfortbildung, Lehrplanentwicklung und Medien, 2001.
Thomas, M.: Vom Abakus bis Zuse. In: Friedrich (Hrsg.), 2005, S. 185–196.
TKM – Thüringer Kultusministerium (Hrsg.): Lehrplan für das Gymnasium Informatik. Erfurt: TKM, 1999.
TMBWK – Thüringer Ministerium für Bildung, Wissenschaft und Kultur (Hrsg.): Medienkunde. Erfurt: TMBWK, 2009.
Vöcking, B. u. a. (Hrsg.): Taschenbuch der Algorithmen. Reihe »eXamen.press«. Berlin; Heidelberg: Springer, 2008.
Wirth, N.: Algorithmen und Datenstrukturen. Reihe »Teubner-Studienbücher Informatik«, Bd. 31. Stuttgart: Teubner, ^{2}1979.
Wußing, H.: Adam Ries. Leipzig: EAGLE Edition am Gutenbergplatz, ^{3}2009.

Alle in diesem Buch angegebenen Internetquellen wurden zuletzt am 31. August 2010 geprüft.

Index

Abbildungsnachweis

Abbildung auf S. 10:
http://www.spielen-mit-verantwortung.de/startseite/index.php
Abbildungen 1.1, 1.3, 3.7, 3.8, 5.1, 6.1, 9.4, 9.13, 9.15 und S. 109:
LOG-IN-Archiv
Abbildung 1.2:
Kunsthistorisches Museum Wien (links),
http://commons.wikimedia.org/wiki/File:Vigenere.jpg (rechts)
Abbildung 4.1:
CASIO Europe GmbH
Abbildung 6.4:
Interessengemeinschaft Optische Telegrafie in Preußen – Andreas Hendrich, Reinhold Zabel
Abbildung 7.1:
Lutz Kohl
Abbildung 9.12:
http://upload.wikimedia.org/wikipedia/commons/c/c4/TSP_Deutschland_3.png

Alle anderen Abbildungen sind vom Verfasser.

Die Gefahr, dass der Computer so wird
wie der Mensch, ist nicht so groß, wie die Gefahr,
dass der Mensch so wird wie der Computer.

Konrad Zuse

Im Mittelpunkt der »Kunterbunten Schulinformatik« stehen Ideen für das Gestalten von Informatikunterricht.

Das Buch richtet sich vorrangig an Personen, die Unterricht auf der Grundlage von Kompetenzbeschreibungen planen, durchführen und reflektieren wollen. Ziel ist letztlich das Entwickeln einer Kultur des sinnvollen Umgangs mit Bildungsstandards.

Das Wort »kunterbunt« soll für vielfältig und abwechslungsreich stehen und durch mögliche Assoziationen auch daran erinnern, dass es bei allem, was an Schulen geschieht, um Kinder und Jugendliche geht.

LOG IN VERLAG