

Studiengang: Informatik
Betreuer: Dr. Karsten Weicker
Prüfer: Dr. Volker Claus, Fakultät 5, Universität Stuttgart

Beginn am: 15. Oktober 2003
Beendet am: 14. April 2004

CR-Klassifikation: E.2, K.3.1, K.3.2

Diplomarbeit Nr. 2154

Entwurf und Implementierung eines Lernsystems für gestreute Speicherung

Zissis Siantidis

Fakultät Elektrotechnik, Informatik,
Informationstechnik
Universität Stuttgart
Institut für Formale Methoden der Informatik
Abteilung Formale Konzepte
Universitätsstr. 38
70569 Stuttgart

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS / TABELLENVERZEICHNIS.....	iii
1 Einleitung	1
1.1 Einführung und Zielsetzung	1
1.2 Übersicht über die Ausarbeitung.....	1
2 Lernsituation im Grundstudium Informatik.....	3
2.1 Die Vorlesungen.....	3
2.2 Das reale Lernverhalten	4
3 Didaktische Konzepte und Vorgehensweisen	5
3.1 Lernziele.....	5
3.2 Motivation	6
3.2.1 Gegenstandsbezogene Motivationshilfen	7
3.2.2 Leistungsbezogene Motivationshilfen	8
3.3 Erfolgskontrolle.....	8
3.4 Didaktische Prinzipien	9
3.5 Lerntheoretische Grundlagen	10
3.5.1 Behaviorismus.....	10
3.5.2 Kognitive Psychologie	11
3.5.3 Gestaltpsychologie.....	11
4 Neue Medien und E-Learning	12
4.1 Computerunterstützter Unterricht	13
4.2 Lernprogrammvarianten.....	15
4.2.1 Übungsprogramme („Drill & Practice“).....	15
4.2.2 Tutorielle Unterweisung	15
4.2.3 Simulationen	16
5 Hashing.....	18
5.1 Grundlagen	18
5.2 Hashfunktionen	19
5.2.1 Die Divisions-Rest-Methode	20
5.2.2 Die multiplikative Methode	21
5.2.3 Sonstige Methoden für Hashfunktionen	22
5.3 Adresskollisionen	23
5.3.1 Verkettung der Überläufer	23
5.3.2 Open Hashing	24
Lineares Sondieren	25
Quadratisches Sondieren.....	27
5.3.3 Hinweis zur Clusterbildung	29
5.4 Rehashing	29

6 Gestaltungsprinzipien und Anforderungen	32
6.1 Designprinzipien	32
6.2 Die Minnesota Kriterien.....	33
6.3 Ablaufsteuerung	34
6.4 Interaktionsstruktur	35
6.5 Bildschirmdesign.....	37
7 Das Lernmodul „Hashing“	39
7.1 Aufbau.....	39
7.2 Inhalt	40
7.3 Interaktive Applikationen.....	42
7.3.1 Hashing mit der Divisions-Rest-Methode (Kap. 3, S. 10).....	42
7.3.2 Lineares Sondieren mit einstellbarer Schrittweite (Kap. 4, S. 11).....	45
7.3.3 Clustering - Zufallszahlen mit linearem Sondieren (Kap. 4, S. 14).....	47
7.3.4 Quadratisches Sondieren (Kap. 4, S. 17)	49
7.3.5 Clustering - Zufallszahlen mit quadratischem Sondieren (Kap. 4, S. 19)	51
7.4 Allgemeine Merkmale.....	53
7.4.1 Lernziele	53
7.4.2 Motivation.....	53
7.4.3 Didaktische Prinzipien.....	54
7.4.4 Computerunterstützter Unterricht	55
7.4.5 Gestaltungsprinzipien	56
8 Zusammenfassung und Ausblick	59
8.1 Zusammenfassung.....	59
8.2 Vorschläge für künftige Erweiterungen	59
8.3 Ausblick	61
ANHANG	62
A BESCHREIBUNG DES LERNMODULS.....	63
Isolierung einzelner interaktiver Applikationen	68
B LERNMODUL ANPASSEN	69
B.1 Die Datei <code>lernmodul.js</code>	70
B.2 Die Datei <code>lernmodul.css</code>	73
B.3 Die HTML-Dateien <code>pageXXYY.html</code>	75
C QUELLCODE-DATEIEN	78
D PROGRAMMLISTING.....	79
E DIE CD-ROM	88
F LITERATURVERZEICHNIS.....	89

ABBILDUNGSVERZEICHNIS

Abbildung 1: Rehashing	30
Abbildung 2: Navigationsleiste	39
Abbildung 3: Interaktive Applikation 1 - Hashing mit der Divisions-Rest-Methode	42
Abbildung 4: Interaktive Applikation 2 - Lineares Sondieren mit einstellbarer Schrittweite	45
Abbildung 5: Interaktive Applikation 3 - Clustering anhand von Zufallszahlen mit linearem Sondieren	47
Abbildung 6: Interaktive Applikation 4 - Quadratisches Sondieren	49
Abbildung 7: Interaktive Applikation 5 - Clustering anhand von Zufallszahlen mit quadratischem Sondieren	51

TABELLENVERZEICHNIS

Tabelle 1: Lernziele nach [Grillenbeck, 2000]	6
Tabelle 2: Lernprogrammtypen	15
Tabelle 3: Aufwandsvergleich Open Hashing	28
Tabelle 4: Aufgabenformen in Computer-Lernprogrammen	36

1 Einleitung

1.1 Einführung und Zielsetzung

Der Einsatz von Computern im Bereich der Lehre hat sich mittlerweile – nicht nur an Universitäten – etabliert. Mit dem Internet als Treiber der vernetzten Computertechnologie hat sich in Bildungseinrichtungen ein neuer Kommunikationskanal herauskristallisiert: e-Learning. Online-Kurse, Online-Unterlagen, Online-Sitzungen (z.B. Chat) etc. sind die heutigen Mittel, mit denen innovatives Lehren und Lernen stattfindet.

Auch an der Universität Stuttgart ist man bestrebt, die Möglichkeiten des e-Learning auszuschöpfen. Im Rahmen dieses Bestrebens entstehen verschiedene elektronische Materialien, die das Lehren und Lernen unterstützen sollen.

Ziel der vorliegenden Arbeit ist es, ein Lernmodul zum Themengebiet „Hashing“ zu entwickeln, das im Grundstudium der Informatik an der Universität Stuttgart sowohl in der Lehre als auch zum Selbststudium eingesetzt werden kann. Dabei sollen interaktive Bestandteile des Lernmoduls ein besseres Verständnis für komplexe Zusammenhänge ermöglichen.

1.2 Übersicht über die Ausarbeitung

In Kapitel 2 erfolgt zunächst ein grober Überblick über die Lernsituation im Grundstudium. Hierzu werden Studienvorgaben sowie Lerngewohnheiten betrachtet.

In Kapitel 3 werden einige Grundlagen der Didaktik vorgestellt, die die Entwicklung des Lernmoduls maßgeblich beeinflusst haben. Dabei wird im Besonderen auf Lernziele, Motivation, Erfolgskontrolle, didaktische Prinzipien und lerntheoretische Grundlagen eingegangen.

Kapitel 4 befasst sich mit e-Learning und dessen Einsatz in den Neuen Medien. Hierbei wird speziell auf Lernprogrammvarianten eingegangen.

In Kapitel 5 findet sich ein grober Überblick über das Themengebiet „Hashing“, zumindest in dem Maße, wie es für das Grundstudium relevant ist. Eine dem Themengebiet angemessene Behandlung mit allen Verfahren, Einzelheiten und Sonderfällen würde den Rahmen dieser Arbeit sprengen.

Kapitel 6 befasst sich mit dem eigentlichen Lernmodul „Hashing“ und erläutert, wie es entstanden ist, auf welche Merkmale Wert gelegt wurde und warum und geht auf seinen Aufbau und Inhalt ein.

Kapitel 7 schließlich enthält eine Zusammenfassung, geht auf künftige Erweiterungsmöglichkeiten ein, gibt schließlich einen Ausblick und bildet so das Schlusskapitel dieser Arbeit.

Benutzungshinweise, Anpassung des Lernmoduls sowie Hinweise zur Füllung mit neuem Inhalt, Übersicht über die Quellcode-Dateien und ein Programmlisting sind im Anhang zu finden.

Das im Rahmen dieser Arbeit entstandene Lernmodul „Hashing“ ist auf der beiliegenden CD-ROM in Anhang E zu finden.

2 Lernsituation im Grundstudium Informatik

Dieses Kapitel soll einen Überblick über die augenblickliche Lernsituation im Grundstudium Informatik an der Universität Stuttgart vermitteln. Hierbei sollen die angebotenen Vorlesungen sowie das reale Lernverhalten der Studierenden, soweit dies objektiv beschrieben werden kann, zur Betrachtung herangezogen werden.

2.1 Die Vorlesungen

Die Vorlesungen im Grundstudium Informatik lassen sich grob in vier Kategorien aufteilen:

- Mathematik
- praktische Informatik
- technische Informatik
- theoretische Informatik

Diese vier Kategorien bilden auch die Prüfungseinheiten des Grundstudiums zur Erlangung des Vordiploms (zusammen mit einigen weiteren Vordiplomsleistungen wie Softwarepraktikum, Hardwarepraktikum etc.). Die Vorlesungen decken eine Grundmenge an Lernstoff ab, die im Hauptstudium benötigt und daher auch vorausgesetzt wird.

Erwähnenswert ist die Tatsache, dass nicht wie bei anderen Studiengängen Einzelprüfungen abgelegt werden, sondern ein gesamter Block auf einmal abgeprüft wird. Der Lernstoff umfasst dabei bis zu vier Vorlesungen, was einen immens hohen Aufwand für die Prüfungsvorbereitung erfordert.

2.2 Das reale Lernverhalten

Das reale Lernverhalten der Studierenden im Grundstudium ist leider nicht optimal. Obwohl in zahlreichen Einführungsveranstaltungen sowohl von Seiten der Universität als auch von Seiten der Fachschaft einige Lernstrategien vorgeschlagen werden, lässt sich beobachten, dass ein Großteil der Studierenden während des Vorlesungszeitraums ein eher inaktives Lernverhalten aufweist: nur die vorgegebenen Leistungen (Übungsscheine etc.) werden mit minimalem Aufwand erbracht. Der Hauptaufwand wird in der vorlesungsfreien Zeit betrieben, in der bei der Mehrzahl der Studenten mit der Prüfungsvorbereitung angefangen wird. Dies ist deshalb möglich, weil die Prüfungen meistens nicht am Anfang, sondern gegen Ende der vorlesungsfreien Zeit stattfinden. Dabei wird der erforderliche Zeitaufwand häufig unterschätzt. Das Resultat sind mangelhaft vorbereitete Studenten und teilweise hohe Durchfallquoten.

Von Seiten der Universität kann vieles getan werden, um die Studenten bei der Prüfungsvorbereitung zu unterstützen. Die vorliegende Arbeit stellt einen Ansatz dazu dar, bei dem versucht wird, Studenten ein zusätzliches Mittel zur Prüfungsvorbereitung zur Verfügung zu stellen: ein elektronisches Lernmodul. Nicht nur weil ein elektronisches Lernmodul eine willkommene Abwechslung zur doch papier- bzw. bücherlastigen Prüfungsvorbereitung darstellt, sondern auch weil durch interaktive Bestandteile viele Zusammenhänge besser verdeutlicht werden können als eine rein textuelle Beschreibung, kann ein solches Lernmodul stark motivieren und den Prüfungsvorbereitungsprozess bereichern.

3 Didaktische Konzepte und Vorgehensweisen

In diesem Kapitel sollen grundlegende didaktische Konzepte und Vorgehensweisen unter dem Gesichtspunkt der späteren Erstellung eines Lernmoduls erläutert werden

3.1 Lernziele

Vor der Entwurfsphase eines Lernmoduls sollte genau überlegt werden, welche Lernziele das Lernmodul haben soll. Eine gute Übersicht über mögliche Lernziele findet sich in [Grillenbeck, 2000]:

1.	Erwerb von Wissen (z.B. von Termini, Fakten, Methoden, Theorien)
2.	Fähigkeit zur Anwendung des erworbenen Wissens (z.B. zweckmäßige Anwendung von wissenschaftlichen Begriffen und Erkenntnissen, die bereits vermittelt wurden, auf neue Erscheinungen und Probleme)
3.	Verstehen der inhaltlichen Aussagen (Fähigkeit zur Interpretation, zu Schlussfolgerungen aus expliziten Aussagen usw.)
4.	(Methoden-)Kritische Analyse des Wissens (z.B. Fähigkeit, stillschweigende Annahmen zu erkennen; Fertigkeit, Tatsachen von Hypothesen zu unterscheiden)
5.	Schaffung von neuem Wissen durch Synthese (z.B. Fähigkeit, geeignete Hypothesen zu formulieren, die sich auf eine Analyse der damit zusammenhängenden Faktoren stützen und diese Hypothesen im Hinblick auf neue Faktoren und Betrachtungen zu modifizieren)
6.	Artikulationsfähigkeit, Kommunikationsfähigkeit

7.	Präsentationsfähigkeit
8.	Team-Fähigkeit (Partizipations-, Konflikt- u. Kooperationsfähigkeit)
9.	Evaluierung des Wissens (Befähigung zum begründeten Urteil darüber, inwieweit (Daten-)Material und Methoden bestimmten Standards, Bewertungsnormen usw. entsprechen)
10.	Fähigkeit zur Selbstorganisation und Selbststeuerung
11.	Spontaneität, Individualität (Fähigkeit, echte Empfindungen bewusst zu erleben und zum Ausdruck zu bringen)

Tabelle 1: Lernziele nach [Grillenbeck, 2000]

Im Zusammenhang mit Lernzielen sind für den Lehr-/Lernprozess noch zwei Punkte wichtig:

- Den Lernenden sollten diese Lernziele vorab mitgeteilt werden. Die Lernziele müssen für die Lernenden zu jedem Zeitpunkt transparent sein. Außerdem ist die frühe Mitteilung der Lernziele ein wesentlicher Faktor für die Motivation. Werden sie nicht genannt, so können Lernende diese schwer abschätzen und sich selbst in Bezug dazu einschätzen (siehe 3.2).
- Erfolgskontrolle: Zu geeigneten Zeitpunkten ist eine Überprüfung der Lernziele vorzunehmen (siehe 3.3).

3.2 Motivation

Der Begriff der Motivation ist in der Psychologie-Literatur ausgiebig behandelt worden. Eine gute Gliederung von Lernmotiven findet sich in [Grillenbeck, 2000]:

- *Gegenstandsbezogene Lernmotive*: Der Lerner tritt mit dem Lerngegenstand in Beziehung. Mögliche Motive (Neugier, Suche nach Wahrheit, Reiz, Erlebnis) liegen im Gegenstand selbst.
- *Leistungsbezogene Lernmotive*: Die Bedürfnisse, die eine Verbesserung der Lern- und Leistungsfähigkeit und die optimale Aufgabenbewältigung zum Inhalt haben (Kreativitätsbedürfnis, Leistungsmotivation, Kompetenzmotivation).
- *Selbstbezogene Lernmotive*, z.B. Bedürfnis nach Sicherheit, lebenspraktische Zukunftsvorstellungen.
- *Sozialbezogene Lernmotive*: Lernen aus dem Wunsch nach sozialer Integration und Anerkennung, dem Wunsch, sich abzuheben (Wettkampf, Konkurrenz) oder die aktive Mitgestaltung der soziokulturellen Umwelt.

Da es in der vorliegenden Arbeit darum geht, ein Lehr-/Lernmodul zu entwickeln, das dem Selbststudium dienen soll und wir uns fragen müssen, wie dieses Modul Motivation erreichen kann, sind die ersten beiden Kategorien für das Thema der vorliegenden Arbeit relevanter als die letzten beiden, weswegen hier exemplarisch nur auf diese eingegangen sei:

3.2.1 Gegenstandsbezogene Motivationshilfen

Unter dem Begriff der gegenstandsbezogenen Motivationshilfen sind Methoden zusammengefasst, die den Lerngegenstand selbst und von ihm ausgehende Eindrücke betrachten, um die Lernbereitschaft anzuregen. Als Beispiel sei die Motivation durch kognitive Inkongruenzen genannt: Es wird eine Diskrepanz zwischen einer Erwartung und den realen Verhältnissen aufgezeigt. Diese Inkongruenzen nennt man bei logischen Widersprüchen Anomalien.

3.2.2 Leistungsbezogene Motivationshilfen

Leistungsbezogene Motivationshilfen sind Methoden, die die Leistungsbereitschaft fördern, indem sie auf eine Angemessenheit bei der Aufgabendosierung, dem Aufgabenniveau und der Anzahl der Wiederholungen achten. Als Beispiel soll eine Lernzielanalyse dienen: wie in Abschnitt 3.1 lassen sich Lernziele in Kategorien einordnen. Die (nach [Grillenbeck, 2000]) vier wichtigsten Kategorien sind Kennen, Verstehen, Anwenden, Beurteilen. Es gibt nun zu viele Dinge, die im Grundstudium vermittelt werden, als dass man verlangen könnte, alle bis zum Stadium des Beurteilens vermitteln zu können. Bei wichtigeren Sachverhalten höher in der Lernzielhierarchie aufzusteigen und bei weniger wichtigen dementsprechend weniger kann der Motivation sehr förderlich sein. In der Praxis heißt das: der Lernende soll erkennen, welche Inhalte sehr relevant und welche weniger relevant sind. So kann er seinen Aufwand besser einteilen und wird nicht durch einen immens groß erscheinenden Stoffberg demotiviert.

3.3 Erfolgskontrolle

Selbstverständlich muss in regelmäßigen Abständen der Erfolg des Lernprozesses überwacht werden. Aus diesem Grunde haben sich in den Lernprozess eingestreute Zwischenprüfungen bei Lernmodulen in der Praxis bewährt. Dabei werden in einem Lernmodul die Lernenden regelmäßig aufgefordert, das Gelernte wiederzugeben bzw. das Verständnis der Inhalte zu belegen. Dies kann auf unterschiedliche Arten geschehen:

- Testfragen mit unterschiedlichen Antwortmöglichkeiten: Text oder Multiple-Choice.
- Interaktive Sequenzen, in denen ein Verfahren nachgeahmt werden muss (z.B. ein Sortierverfahren)

3.4 Didaktische Prinzipien

Die Effizienz eines Lernmoduls ist stark abhängig von seiner didaktischen Qualität, welche wiederum durch die Einhaltung gewisser didaktischen Prinzipien erreicht werden kann. Hierzu sollen nun einige didaktische Prinzipien, also Leitlinien didaktischen Handelns betrachtet werden, nach [Baumann, 1996].

- **Prinzip des aktiven Lernens**
Der Lernende sollte sich die Lerninhalte nicht (nur) durch passive Rezeption, sondern durch aktive Konstruktion aneignen (entdeckendes Lernen).
- **Integrationsprinzip**
Es sollen Beziehungsnetze und Sinnzusammenhänge geschaffen werden. Die Integration sollte bereits bei der Erarbeitung einer Erkenntnis einsetzen (Prinzip des Lernens in Zusammenhängen).
- **Prinzip der Veranschaulichung**
Neue Lerninhalte sollen über Lernsituationen vermittelt werden, bei denen nur einzelne Elemente oder Aspekte neu sind, die aber möglichst viele Ansatzpunkte für eine Anwendung bekannter kognitiver Schemata bieten. Neue Inhalte sollten auf das Vorwissen des Lernenden aufbauen und dieses ergänzen.
- **Prinzip der Stabilisierung**
Erworbenes Wissen sollte regelmäßig in neue Zusammenhänge eingebettet werden und dabei verallgemeinert und differenziert werden. Dieses erneute Aufgreifen dient der Stabilisierung der Lerninhalte beim Lernenden.
- **Operatives Prinzip**
Der Lernende soll zum gedanklichen Operieren mit dem Lerngegenstand angeleitet werden. Dieses Prinzip ist eng verknüpft mit dem Prinzip des aktiven Lernens. Ihm zugrunde liegt die Erkenntnis, dass für die Erkenntnisgewinnung

geistige Operationen eine große Rolle spielen. Aus diesem Grunde sollen dem Lernenden Materialien zur Verfügung gestellt werden, an denen er operieren, „forschen“ und „herumspielen“ kann. Gerade im Kontext eines Lernmoduls ist dies durch interaktive Programmmodule bzw. Animationen sehr gut möglich.

- **Prinzip der Lebensnähe und Aktualität**

Die Lerninhalte sind so zuzubereiten, dass sie Bezüge zur aktuellen Lebenswelt des Lernenden aufweisen.

- **Prinzip der Zielvorstellung**

Dem Lernenden ist das jeweilige Unterrichtsziel mitzuteilen.

3.5 Lerntheoretische Grundlagen

In diesem Abschnitt sollen die (nach [Grillenbeck, 2000]) wichtigsten Strömungen der Lernpsychologie kurz und schlaglichtartig vorgestellt werden. Eine ausführlichere Darstellung der Lernpsychologie findet sich in [Lefrancois, 1994] und [Edelmann, 1986].

3.5.1 Behaviorismus

Skinner's *operantes Konditionieren* besagt im Wesentlichen, dass man den Lernenden immer dann belohnen soll, wenn er sich dem Lernziel nähert, und immer dann bestrafen soll, wenn er sich davon entfernt. Viele Lernprogramme funktionieren nach diesem Prinzip. Dabei wird der Stoff in mehrere kleinere Einheiten aufgeteilt, die nacheinander durchlaufen werden. Nach jeder Einheit werden Fragen gestellt, die zur Wissenskontrolle dienen. Die Reaktionen des Lernenden sollen sofort, je nach Antwort, positiv oder negativ verstärkt werden.

Nachteil dieser Methode ist die Tatsache, dass außer Wissen kein anderes Lernziel erreicht wird. Verstehen, Anwenden und Beurteilen der Zusammenhänge werden nicht erreicht.

3.5.2 Kognitive Psychologie

Die kognitive Psychologie interessiert sich mehr für die lernbedingten (inneren) Änderungen der Strukturen im Gehirn des Lernenden als für die Beobachtung seiner (äußeren) Verhaltensweisen. Sie „sieht den Menschen nicht als passiven Informationsempfänger, sondern als ein in seiner Interaktion mit der Umwelt schöpferisch tätiges Subjekt“ [Grillenbeck, 2000]. Lernen wird als Begriffsbildung und Einordnung von Neuem in den bestehenden Wissenskontext angesehen. Der Kognitivismus sieht Lernen als Prozess des Wissenserwerbs. Hierbei wird der Lerngegenstand als fertiges System vermittelt, dem Lehrenden fällt eine aktive, dem Lernenden eine eher passive Rolle zu. Ausserdem hat die kognitive Psychologie auch die Wichtigkeit des Zwecks von Lernen erkannt und behandelt die Frage, warum man etwas lernt. Hieraus entsprungen ist der allgemein anerkannte Grundsatz, zu Beginn einer Unterrichtseinheit Ziel und Sinn der Einheit deutlich zu machen.

In vielen Lernprogrammen sind Ideen der kognitiven Psychologie eingeflossen, z.B. lässt sich oft ein hoher Grad an Interaktivität vorfinden, um Rückmeldungen geben zu können. Querverbindungen und übergeordnete Zusammenhänge werden ebenfalls häufig aufgezeigt.

3.5.3 Gestaltpsychologie

Die Gestaltpsychologie sieht für das Lernen Konstruktion statt Instruktion vor. Sie baut auf das *Aha-Erlebnis*: Der Lernerfolg ist am größten, wenn Einsicht in die Zusammenhänge gewonnen wird. Der Lernende soll also seinen Lernweg selber bestimmen und soll selber auf die wichtigen Zusammenhänge stoßen. Dies erreicht er durch Experimentieren und Modellieren. Die Lernsysteme müssen einen hohen Freiheitsgrad zulassen und ermöglichen nach Änderung von Eigenschaften, Variablen und Elementen den Vergleich zur vorherigen Situation. Mit der Zeit werden Zusammenhänge verstanden und Gesetzmäßigkeiten entdeckt.

Ein Nachteil dieses Prinzips ist, dass sich der Lernende verlaufen kann, in eine Sackgasse gerät und dieses nicht merkt. Aus diesem Grunde ist eine betreuende Person zu Aufsicht, Lenkung und Vermeidung von Frustration notwendig.

4 Neue Medien und E-Learning

E-Learning steht für „electronic learning“ und ist ein Begriff, der in den späten 90er Jahren mit der boomenden Internettechnologie aufgekommen ist. Er beschreibt das „elektronische Lernen“, also die Nutzung von elektronischen Medien (vorzugsweise Computern) zum Erwerb von Wissen, Kenntnissen, Fähigkeiten und Kompetenzen.

Die Ursachen dieser Entwicklung sind vielfältig [Weicker, 2003]. Ein einmaliges Erwerben von Fähigkeiten genügt in der heutigen, sich (nicht nur) technologisch rasch ändernden Gesellschaft nicht mehr. Vielmehr ist ein lebenslanges Lernen Realität. Hinzu kommt, dass ein Großteil des Wissens elektronisch verfügbar ist, ja sogar manchmal nur elektronisch erstellt und abgespeichert wird. Gepaart mit der stark angepriesenen Mobilität eines Einzelnen, die durch immer kleiner werdende Geräte (Handys, PDAs) ständig perfektioniert und geradezu angetrieben wird, ist ein Trend entstanden, der Schulungen (aber auch Meetings, Besprechungen etc.) nicht mehr nur in speziellen dafür vorgesehenen Räumlichkeiten, sondern zu beliebiger Zeit an beliebigen Orten stattfinden lässt. Die schnelle Weiterentwicklung des Internets (sowohl in Sachen Funktionalität als auch in Sachen Geschwindigkeit) unterstützt diese Entwicklung.

E-Learning verschmilzt sehr stark die Internettechnologie mit der Ausbildung. Hierbei wird vom klassischen Lernszenario einer Klasse in einem Klassenzimmer mit einem Lehrer bewusst abgewichen und eine zeitliche bzw. räumliche Trennung der einzelnen Teilnehmer ermöglicht. Kommunikation bleibt trotzdem ein Kernpunkt und wird entweder synchron (z.B. durch Chat) oder asynchron (durch E-Mail, Messenger-Dienste etc.) durchgeführt.

Der wesentliche Zweck einer E-Learning Umgebung ist, dass Anwender sie benutzen können, um etwas zu lernen, ohne dass ein menschlicher „Lehrer“ anwesend sein muss, und dass sie dies zu jeder Zeit und an jedem beliebigen Ort tun können. Dies ist einer der wesentlichen Vorteile eines E-Learning Systems. Die Anwender können den Lernprozess ihrem individuellen Lernstil und ihrer individuellen Lerngeschwindigkeit anpassen.

Da E-Learning die Benutzung von Computertechnologie impliziert, kann durch geeignete Programmumgebungen und Werkzeuge ein direktes Ausprobieren und Anwenden des Gelernten ermöglicht werden.

Da mit nur einem einzigen E-Learning System (fast) beliebig viele Anwender geschult werden können, ist das Kosten-Nutzen Verhältnis extrem gut und rechtfertigt sogar höhere Investitionen.

Werden die Lerninhalte zentral gespeichert und durch Netzwerktechnologie dem Anwender zur Verfügung gestellt (wie es meistens realisiert ist), so ist sogar ein dynamisches, ständig aktuelles Informationsangebot möglich.

Bei einer Vernetzung kommen automatisch weitere Aspekte hinzu. So muss auf eine Universalität geachtet werden (das System sollte plattform- und browserunabhängig sein). Ebenfalls sind die Leistungen und Lernergebnisse mehrerer Teilnehmer bequem verwaltbar und zentral analysierbar.

Doch die Benutzung einer E-Learning Umgebung birgt auch Nachteile. Allen voran der Hauptkritikpunkt an vielen Systemen: mangelnde Qualität der Lernmaterialien. Die Erstellung von didaktisch guten Lerninhalten ist extrem schwierig und ihre Schwierigkeit wird leider oft unterschätzt.

Auch lassen sich viele Inhalte am besten von einem Menschen übermitteln, der durch seine Gestik, Mimik und allgemeiner Körpersprache sowie Artikulation die Übermittlung der Information unterstreicht und bekräftigt. Dieser Punkt führt dazu, dass E-Learning-Veranstaltungen immer mit tutorieller Betreuung durchgeführt werden, die manchmal sehr zeitaufwändig werden kann.

Ein anderer Gesichtspunkt ist die bei den Teilnehmern erforderliche Motivation, die über ein elektronisches Medium nur schwer aufgebaut werden kann.

Auch einige technische Probleme treten auf: Aufgrund der Vielzahl der verfügbaren Plattformen/Browser ist die angesprochene Universalität nicht einfach zu erreichen. Auch wirft die angesprochene Vernetzung viele technische Probleme auf: Zugriffsberechtigungen, langsame Internetverbindungen etc.

Generell muss man zwischen Computer Based Training (CBT) und Web Based Training (WBT) unterscheiden.

4.1 Computerunterstützter Unterricht

Die Entwicklung von computerunterstütztem Unterricht hat früh begonnen. Schon 1966 stellte Martens [Martens, 1966] Gestaltungsregeln auf:

1. Der Lernstoff sollte nicht in zu kleine Teile aufgeteilt werden. Zusammenhänge könnten nicht mehr erkannt werden und die Gestaltbildung wird erschwert. Ausgehend vom Gesamtbild sollten Einzelheiten gezeigt werden.
2. Die strukturgerechte Darstellung des Lerninhalts soll den Lernenden beim Lernfortschritt und bei der Aufgabenbearbeitung unterstützen. Wird der Zusammenhang des Stoffes einsichtig dargestellt, kann der Lernende den letzten Schritt eines Zusammenhangs oft selber durchführen.
3. Damit ein Erfolgserlebnis entstehen kann, sollten die Aufgaben nicht zu einfach sein.

Nach [Heinrich, 1972] lassen sich Programme zum computerunterstützten Unterricht in folgende Kategorien einteilen:

- **Drill und Übung:**
Der Computer übernimmt nur die Aufgabe des Fragenstellens und gibt Rückmeldung über die Richtigkeit der Antworten (z.B. Vokabeltrainer).
- **Tutorielle Unterweisung:**
Das Lernprogramm übernimmt auch Führungsfunktionen und gibt Erklärungen, der Lerner kann bestimmen, was wann behandelt werden soll.
- **Computergesteuertes Selbststudium:**
Die eigentlichen Lernprogramme. Das Lernprogramm verfügt Erklärungen, Übungen und Aufgaben. Der Lernende steuert den Ablauf selbst. Ein menschlicher Lehrender ist als Tutor anwesend, um schwierige Fragen zu beantworten
- **Simulation, Spiel, Offene Lernumgebungen:**
Auf dem Rechner finden Simulationen und Spiele statt. Auf diese Weise können Fragestellungen behandelt werden, die nicht berechnet werden können, weil sie Interaktionen und netzwerkartige Zusammenhänge aufzeigen wollen.

- **Problemlösen, Programmieren:**

Man sollte nicht nur fertige Programme benutzen, sondern auch eigene Lösungsvorschläge machen und diese auch implementieren.

Eine Einordnung dieser Kategorien in die Lernpsychologie findet sich bei [Schulmeister, 1994]:

Lernpsychologische Sicht	Lernprogrammtypus
behavioristisch	Drill & Practice
kognitivistisch	Tutorielle Systeme
konstruktivistisch	Offene Lernumgebungen

Tabelle 2: Lernprogrammtypen

4.2 Lernprogrammvarianten

Für alle oben genannten Kategorien (außer der letzten) gibt es bereits entsprechende Lernprogramme. Wir wollen diese nun genauer betrachten.

4.2.1 Übungsprogramme („Drill & Practice“)

Übungsprogramme ermöglichen hauptsächlich das Wiederholen und Einüben schon vorhandenen Wissens. Hierzu werden dem Lernenden Fragen gestellt, die anhand verschiedener Kriterien (z.B. steigender Schwierigkeitsgrad bei richtigen Antworten) aus einem Aufgabenpool ausgewählt werden. Der gesamte Ablauf ist beschränkt auf die Phasen Frage, Antwort und Rückmeldung.

Beispiele sind Vokabeltrainer oder Lernsoftware für die Führerscheinprüfung mit Beispielfragen aus den offiziellen Fragebögen.

4.2.2 Tutorielle Unterweisung

Bei der tutoriellen Unterweisung werden die Lerninhalte in kleinen Einheiten interaktiv präsentiert (manchmal sogar multimedial). Anschließende Verständnisfragen und

Aufgaben werden überprüft und kommentiert. Anhand der Antworten kann das Lernprogramm entweder zu weiteren Lerneinheiten verzweigen oder gewisse Einheiten zur Wiederholung anbieten.

Hauptanwendungsgebiet tutorieller Systeme ist die Vermittlung neuer Inhalte eines Fachgebietes und die Überprüfung des Lernerfolgs, weswegen sie gerade bei eindeutigem Lernstoff, der nicht dauernd geändert wird, eingesetzt werden können.

Beispiele gibt es mittlerweile sehr viele, vor allem Lernprogramme zu betriebswirtschaftlichen Grundlagen (wie Marketing oder Finanzplanung), aber auch Lernprogramme für Schüler zum Aufbessern der Deutsch-, Mathematik- oder Physikkenntnisse.

Das Lernmodul „Hashing“, das im Rahmen dieser Arbeit entsteht, ist ebenfalls dieser Lernprogrammvariante zuzuordnen.

4.2.3 Simulationen

Bei einer Simulation hat der Lernende viel Anwendungs- und Handlungsspielraum. Es wird ein Modell der Realität zur Verfügung gestellt, an dem gewisse Parameter verändert werden können. Diese Eingriffe ermöglichen ein exploratives Lernen.

Es wird zwischen drei Arten von Simulationen unterschieden:

- **Entscheidungssimulation:**

Der Lernende muss sich in eine Situation versetzen und Entscheidungen treffen. Durch Interaktion verändert sich diese Ausgangssituation und wird zur neuen Situation usw. Beispiele sind reale physikalische Umgebungen (z.B. Flugzeug-Cockpit) oder abstrakte soziale Gebilde (z.B. Unternehmen, Volkswirtschaft). Solche Simulationen haben im Prinzip kein vorgesehenes Ende.

- **Verhaltenssimulation:**

Der Lernende wird mit einer Problemsituation konfrontiert und soll angemessen reagieren. Aufgrund seiner Entscheidung wird ihm die konsequente Folgesituation präsentiert. Diese Art von Simulationsprogrammen ist sehr umstritten, da sie sehr wenig Handlungsspielraum zulässt und meist nicht

erlaubt, das eigene Temperament und die eigenen Wertvorstellungen zu benutzen.

- **Anwendungssimulation:**

Der Lernende soll die Bedienung komplexer informationstechnischer Systeme einüben. Anwendungssimulationen werden überall da eingesetzt, wo der Eingriff ins Echtssystem nicht möglich, zu komplex oder zu teuer ist. Auch mögliche Ernstfälle oder spezielle Handlungsprozeduren können so eingeübt werden (z.B. in Atomkraftwerken oder Raumkapseln usw.).

5 Hashing

Hashing ist eine Methode zur Verwaltung von Daten, wobei die Daten durch einen Suchschlüssel eindeutig gekennzeichnet sind. Sie stellt eine Alternative dar zur Benutzung von Baumstrukturen (AVL-Baum, B-Baum, optimaler Suchbaum etc.), bei denen der Aufwand (worst case und average case) bei den drei Standard-Operationen Einfügen, Suchen und Löschen („Dictionary Operations“) jedes Mal $O(\log n)$ beträgt. Hashing kann unter gewissen Annahmen einen Aufwand von $O(1)$ erreichen.

Die Idee besteht darin, dass der Speicherort eines Datenelementes nicht gesucht wird, sondern durch eine *Hashfunktion* direkt berechnet wird.

5.1 Grundlagen

Das Szenario beim Hashing besteht darin, dass die tatsächlich vorkommenden und zu verwaltenden Schlüssel eine relativ kleine Teilmenge S ausmachen, während die Menge der möglichen Schlüssel U (das „Universum“) viel größer ist.

Hashing verwendet eine *Hashfunktion* $h: U \rightarrow \{0, 1, \dots, m-1\}$, welche jeden Schlüssel $s \in U$ auf einen Speicherort $h(s)$ innerhalb einer *Hashtabelle* T abbildet, die die eigentlichen Datensätze in einer Array-ähnlichen linearen Datenstruktur speichert. Die Größe m dieser Hashtabelle wird vorher festgelegt und die Hashtabelle ist mit $0, 1, \dots, m-1$ durchindiziert.

Angenommen, diese Funktion h sei injektiv auf der Menge S , d.h. je zwei Schlüssel $s_1, s_2 \in S$ haben verschiedene Hashwerte $h(s_1) \neq h(s_2)$, dann sind die Dictionary Operations sehr einfach in der Zeit $O(1)$ implementierbar [Schöning, 2001]:

- Suchen:


```
IF (T[h(s)] != NIL) THEN . . .
```
- Einfügen:


```
T[h(s)] := /*Daten zu Schlüssel s
```
- Löschen:


```
DISPOSE(T[h(s)]); T[h(s)] := NIL;
```

In solch einem Fall spricht man von *perfektem Hashing*. Leider ist nicht jede Hashfunktion injektiv, d.h. auch die beste Hashfunktion kann nicht vermeiden, dass zwei Schlüssel denselben Hashwert haben (solche Schlüssel nennt man *Synonyme*). In solch einem Fall kommt es zu einer *Adresskollision*. Im Fall einer Adresskollision muss das Hashverfahren reagieren und diese Kollision auflösen. Hierzu gibt es mehrere Strategien, die weiter unten behandelt werden.

Ein Hashverfahren muss also zwei Forderungen genügen [Ottmann, 2002]: Erstens sollten möglichst wenige Kollisionen auftreten, was man durch die Wahl einer „guten“ Hashfunktion erreichen kann. Zweitens müssen Adresskollisionen möglichst effizient aufgelöst werden. Neben einer möglichst gut streuenden Hashfunktion ist die Behandlung von Kollisionen ein zentrales Merkmal von Hashverfahren.

Im Zusammenhang mit Aufwandsabschätzungen taucht eine weitere wichtige Größe auf: der *Belegungsfaktor* α (auch „Auslastungsfaktor“ oder „Auslastungsgrad“), definiert durch:

$$\alpha = \frac{\text{Anzahl belegter Hashpositionen}}{\text{Anzahl aller Hashpositionen}} = \frac{\text{Anzahl belegter Hashpositionen}}{m}$$

Ein Belegungsfaktor $\alpha = 0,6$ bedeutet also, dass die Hashtabelle zu 60% gefüllt ist.

5.2 Hashfunktionen

In diesem Abschnitt sollen die Möglichkeiten untersucht werden, eine geeignete Hashfunktion zu definieren. Diese Funktion sollte die Elemente von U möglichst gut in die Menge $\{0, 1, \dots, m-1\}$ zerstreuen. Hierbei gibt es nun mehrere Methoden:

- Divisions-Rest-Methode
- multiplikative Methode
- Faltung
- Mid-Square-Methode
- Truncation

Auf diese Methoden soll in den nachfolgenden Abschnitten eingegangen werden.

5.2.1 Die Divisions-Rest-Methode

Ein nahe liegendes und einfaches Verfahren zur Erzeugung einer Hashadresse $h(s)$ zu einem gegebenen Schlüssel $s \in N_0$ ist, den Rest von s bei ganzzahliger Division durch die Hashtabellengröße m zu nehmen:

$$h(s) = s \bmod m$$

So wird sichergestellt, dass $h(s)$ die Hashtabellengrenzen nicht übersteigt:
 $0 \leq h(s) \leq m - 1$.

Hier ist eine geeignete Wahl von m von Vorteil. Beispielsweise liefert $h(s)$ bei der Wahl von $m = 2^i$ nur die i niederwertigen Bits der Dualdarstellung von s , ohne die restlichen Bits bei der Hashadressenberechnung zu betrachten.

Es sei z.B. eine Hashtabelle mit einer Größe von $m = 2^8 = 256$ gegeben, um darin Variablennamen, die aus bis zu drei Buchstaben bestehen, zu verwalten. Jeder Buchstabe wird als ASCII-Zeichen gespeichert. Daher kann jeder Variablenname als 24-Bit Zahl, die sich aus drei ASCII-Zeichen (je 8 bit) zusammensetzt, repräsentiert werden.

Das Problem ist, dass durch die Hashfunktion $h(s) = s \bmod 256$ für jeden aus drei ASCII-Zeichen bestehenden Variablennamen $s = C_3C_2C_1$ nur das Zeichen C_1 für die Hashadressenberechnung herangezogen wird, weil der Drei-Zeichen-Schlüssel $C_3C_2C_1$ als 24-Bit Zahl mit dem numerischen Wert $C_3 \cdot 256^2 + C_2 \cdot 256^1 + C_1 \cdot 256^0$ betrachtet wird und durch die „modulo 256“ Berechnung auf den Wert C_1 reduziert wird.

Da durch die Hashfunktion $h(s) = s \bmod 256$ nur das letzte Zeichen ausgewählt wird, werden die Schlüssel $X1, X2, X3, Y1, Y2, Y3$ wie folgt abgebildet:

$$h(X1) = h(Y1) = '1'$$

$$h(X2) = h(Y2) = '2'$$

$$h(X3) = h(Y3) = '3'$$

Die tatsächlichen Hashtabellenpositionen ergeben sich aus den ASCII-Codes für '1', '2' und '3' und sind demnach 50, 51 bzw. 52. So werden die sechs Schlüssel $X_1, X_2, X_3, Y_1, Y_2, Y_3$ in ein gemeinsames Cluster von drei benachbarten Hashadressen abgebildet. Überdies werden fast gleiche Schlüssel auf benachbarte Hashadressen abgebildet, so dass man sagen kann, dass $h(s)$ Cluster innerhalb der Menge der Schlüssel bewahrt. Im Gegensatz dazu „bricht“ eine geeignete Hashfunktion $h(s)$ Cluster innerhalb der Menge der Schlüssel auf, ohne sie zu bewahren oder die Menge der Schlüssel noch stärker zu clustern.

Laut [Knuth, 1998] wird eine bessere Verteilung der durch $h(k)$ produzierten Hashadressen erreicht, wenn m als Primzahl gewählt wird. Allerdings müssen bei der Wahl einer Primzahl in Verbindung mit der Divisions-Rest-Methode bestimmte Vorkehrungen getroffen werden. Insbesondere soll für m keine Primzahl der Form $m = r^k \pm a$ gewählt werden, wobei r die Basis der Schlüsselmenge K ist und k und a kleine Integerzahlen sind.

5.2.2 Die multiplikative Methode

Bei der multiplikativen Methode werden Hashfunktionen der Form

$$h(s) = \lfloor m \cdot (s \cdot a \bmod 1) \rfloor$$

erzeugt, wobei sich „ $s \cdot a \bmod 1$ “ auf die Nachkommastellen von „ $s \cdot a$ “ bezieht. Ein Schlüssel s wird hier zunächst mit einer Konstanten a , $0 < a < 1$ multipliziert. Dann wird der ganzzahlige Anteil des Ergebnisses abgeschnitten. Auf diese Weise erhält man für verschiedene Schlüssel verschiedene Werte zwischen 0 und 1. Eine anschließende Multiplikation mit m verstreut die Hashwerte ziemlich gleichmäßig über die Hashtabelle. Eine „modulo m “ Berechnung ist hier nicht notwendig.

Nach [Knuth, 1998] und [Standish, 1980] hat sich der „goldene Schnitt“ ϕ^{-1} als besonders geeignete Zahl a erwiesen:

$$a = \phi^{-1} = \frac{\sqrt{5}-1}{2} \approx 0,6180339887$$

Als Kriterium für die „Güte“ wird dabei folgendes, interessantes Phänomen genannt: Markiert man die Punkte $\{i\phi^{-1}\}$, $i > 0$ (mit $\{x\}$ = Nachkommaanteil von x) im Intervall $[0,1)$, so fällt auf, dass diese Punkte schön weit voneinander entfernt bleiben, und jeder neue Punkt fällt in das größte verbleibende Intervall und teilt dieses genau im Verhältnis des goldenen Schnittes! Dieses Verhalten macht man sich nun zunutze, bei einer Multiplikation bleibt diese Eigenschaft erhalten und man erhält eine wohlverteilende Hashfunktion.

5.2.3 Sonstige Methoden für Hashfunktionen

Bei der *Faltung* werden die Schlüssel in mehrere Abschnitte zerlegt. Die einzelnen Abschnitte werden dann z.B. aufaddiert (oder subtrahiert, multipliziert, etc.), um die Hashadresse zu erhalten. Ein 9-stelliger Schlüssel $s = 251286334$ wird z.B. in die drei Abschnitte 251, 286 und 334 zerlegt werden, die daraufhin zur Hashadresse 871 aufaddiert werden. Ist die Größe der Hashtabelle mit $m = 1000$ beispielsweise eine vierstellige Zahl, so kann die *modulo m* Berechnung für $h(s)$ eingespart werden, wenn sichergestellt wird, dass nur dreistellige Hashadressen berechnet werden.

Bei der *Mid-Square-Methode* wird ein Ausschnitt aus der Mitte eines Schlüssels entnommen, als Zahl interpretiert und quadriert, um die Hashadresse zu berechnen. Aus dem 9-stelligen Schlüssel $s = 013402122$ kann man z.B. die mittleren Ziffern 402 herausnehmen und quadrieren, um $h(s) = 402^2 = 161604$ zu erhalten. Wenn die Hashadresse 161604 die Hashtabellengröße m übersteigt, so können z.B. die mittleren vier Ziffern des Ergebnisses als Hashadresse $h(s) = 6160$ interpretiert werden, oder die Hashadresse ergibt sich zu $h(s) = 161604 \bmod m$.

Bei der *Truncation* werden einfach Teile des Schlüssels gelöscht und die übrigen Ziffern (oder Bits oder Zeichen) zur Berechnung der Hashadresse benutzt. Wenn z.B. der Schlüssel $s = 013402122$ gegeben ist, können bis auf die letzten drei alle Ziffern ignoriert werden. So erhält man $h(s) = 122$. Eine modulo m Berechnung für $h(s)$ kann dann eingespart werden, wenn sichergestellt werden kann, dass $h(s)$ für jedes s kleiner als m ist. Während die Truncation kaum Berechnungszeit erfordert, neigt sie dazu, die Schlüssel nicht willkürlich und gleichmäßig über die Positionen der Hashtabelle zu

verteilen. Deshalb wird sie oft in Verbindung mit den anderen oben genannten Methoden eingesetzt und selten alleine verwendet.

In [Lum et al., 1971] wird das Verhalten verschiedener Typen von Hashfunktionen, wie das Divisions-Rest-Verfahren, die multiplikative Methode, die Mid-Square-Methode, die Faltung und die algebraische Verschlüsselung untersucht und miteinander verglichen. Das Ergebnis dieser Untersuchung ist, dass das Divisions-Rest-Verfahren insgesamt die beste Performanz vorweisen kann.

5.3 Adresskollisionen

Auch mit einer sehr guten Hashfunktion lassen sich Adresskollisionen nicht vermeiden, da die Anzahl möglicher Elemente viel größer ist als die Größe der Hashtabelle. Zur Auflösung solcher Kollisionen gibt es nun mehrere Verfahren:

- Verkettung der Überläufer
- Open Hashing

5.3.1 Verkettung der Überläufer

Bei der *Verkettung der Überläufer* wird bei Kollisionen eine Liste mit den Elementen erzeugt, die dieselbe Hashtabellenposition belegen würden. Ein „Überläufer“ ist ein Element, das eine bereits gefüllte Position in einer Hashtabelle zum „Überlaufen“ bringt. Alle Überläufer werden in einer dynamisch veränderbaren Struktur gespeichert, etwa in einer verketteten Liste. Dabei wird unterschieden, ob jedes Element der Hashtabelle auch Anfangselement einer verketteten Liste ist (*separate Verkettung*) oder ob es nur einen Zeiger auf den Listenanfang darstellt (*direkte Verkettung*).

Bei diesem Verfahren besteht die Gefahr, dass es zu einer Abspeicherung in einer linearen Liste *entarten* kann, wenn (fast) alle Elemente auf dieselbe Position abgebildet werden oder wenn die Anzahl der zu speichernden Elemente sehr viel größer ist als die Länge der Hashtabelle und so an jeder Position lange Überlauf Listen hängen. Der Belegungsfaktor kann somit größer als 1 sein. In diesem Fall könnte man für die

Überlauflisten Suchbäume aufbauen, um wenigstens logarithmischen Aufwand zu garantieren.

Zur Aufwandsabschätzung definieren wir zwei Erwartungswerte C_n und C'_n . Hierbei ist C_n der Erwartungswert für die Anzahl der betrachteten Einträge der Hashtabelle bei erfolgreicher Suche und C'_n der Erwartungswert bei erfolgloser Suche. Die anderen zwei Operationen – Einfügen und Löschen – sind bezüglich des Aufwands analog zu diesen beiden Erwartungswerten: Dem Einfügen eines Elements geht stets eine erfolglose Suche voraus, das Löschen eines Elements erfolgt stets nach einer erfolgreichen Suche. Demnach ist der Einfüge-Aufwand gerade C'_n , der Lösch-Aufwand gerade C_n . In folgenden Aufwandsabschätzungen werden wir deshalb stets nur auf diese beiden Erwartungswerte eingehen.

Zur genauen Analyse des Aufwands dieses Verfahrens siehe [Ottmann, 2002] und [Schöning, 2001]. Es seien hier nur die Resultate genannt:

Separate Verkettung:

$$C'_n \approx \alpha + e^{-\alpha} \qquad C_n \approx 1 + \frac{\alpha}{2}$$

Direkte Verkettung:

$$C'_n \approx \alpha \qquad C_n \approx 1 + \frac{\alpha}{2}$$

5.3.2 Open Hashing

Im Gegensatz zur Verkettung der Überläufer außerhalb der Hashtabelle wird bei Open Hashing im Falle einer Adresskollision nach einer alternativen („offenen“) Position in der Hashtabelle gesucht und das Synonym dort gespeichert. Natürlich heißt das, dass maximal m Elemente gespeichert werden können und der Belegungsfaktor somit höchstens 1 sein kann.

Ein sehr gängiges Verfahren ist das *Sondieren*: Man definiert eine Reihenfolge, in der alle Speicherplätze einer nach dem anderen betrachtet werden, bis ein freier Speicherplatz gefunden wird. Diese Reihenfolge nennt man *Sondierungsfolge* (oder *Sondierreihenfolge*). Hierbei gibt es zwei grundlegende Verfahren: *lineares Sondieren* und *quadratisches Sondieren*.

Andere Verfahren sind *uniformes und zufälliges Sondieren*, *Double Hashing*, *Ordered Hashing*, *Robin-Hood-Hashing*, *Coalesced Hashing*. Diese Verfahren werden in [Ottmann, 2002] beschrieben.

Lineares Sondieren

Beim linearen Sondieren wird im Falle einer besetzten Hashtabellenposition einfach mit einer konstanten Schrittweite w (meistens 1) zu einer anderen Hashtabellenposition gesprungen. Wenn diese leer ist, wird dort gespeichert und das Verfahren wird beendet, ansonsten wird wieder gesprungen etc. Natürlich wird dabei stets modulo m gerechnet, damit die Hashtabellengrenzen nicht überschritten werden.

Als Ergebnis kann es vorkommen, dass ein Element s nun nicht an seiner Position $h(s)$ gespeichert ist sondern in einer entlang der Sondierfolge späteren Position. Dies muss beim den Operationen Einfügen, Suchen und Löschen beachtet werden (s.u.).

Ein zentrales Manko des linearen Sondierens ist das sog. *Clustering*. Dieses Phänomen beschreibt den Umstand, dass eine Folge aufeinander folgender besetzter Hashtabellenpositionen dazu neigt, immer größer zu werden. Zur Verdeutlichung mache man sich folgendes Beispiel klar:

Eine Hashtabelle mit zehn Einträgen hat vier besetzte und sechs freie Positionen. Es liegt folgende Hashfunktion zugrunde: $h(s) = s \bmod 10$. Schrittweite w des linearen Sondierens sei 1.

0	
1	
2	12
3	33
4	53
5	74
6	
7	
8	
9	

Betrachten wir nun für $x = 0..9$ die Wahrscheinlichkeiten, dass das nächste Element in Position x gespeichert wird:

$$P(0) = P(1) = P(7) = P(8) = P(9) = 0,1$$

Für jede dieser leeren Position beträgt die Wahrscheinlichkeit genau $1/10$, nämlich genau der Häufigkeit des Vorkommens jeden einzelnen Wertes zwischen 0 und 9.

$$P(2) = P(3) = P(4) = P(5) = 0$$

Für jede dieser vollen Positionen beträgt die Wahrscheinlichkeit natürlich 0, da dort keine weiteren Elemente gespeichert werden können.

$$P(6) = 0,5$$

Für die Position 6 besteht die Wahrscheinlichkeit 0,5!

Alle nachfolgenden Elemente, die in Position 2,3,4,5 oder 6 gespeichert werden würden, werden in dieser Position gespeichert. Addiert man die Teilwahrscheinlichkeiten, so erhält man 0,5.

Es ist also wahrscheinlicher, dass ein nachfolgendes Element an das Ende eines solchen „Clusters“ gespeichert wird, als dass es „allein“ irgendwo entfernt gespeichert wird. Dieser Effekt wird noch verstärkt, wenn solche belegte Teilstücke zusammenwachsen. Die Effizienz des linearen Sondierens verschlechtert sich drastisch, sobald sich der Belegungsfaktor dem Wert 1 nähert.

Der Aufwand für erfolgreiche bzw. erfolglose Suche beträgt [Ottmann, 2002]:

$$C_n \approx \frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right) \qquad C'_n \approx \frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$$

Quadratisches Sondieren

Um dieses Clustering zu vermeiden, wird beim quadratischen Sondieren die Folge der Quadratzahlen für die Schrittweite w eingesetzt. Die Sondierfolge ist somit:

$$T[h(s) + 1], T[h(s) + 4], T[h(s) + 9], T[h(s) + 16], \text{ usw}$$

Natürlich wird auch hier wieder nach jedem Schritt modulo m gerechnet.

Beim quadratischen Sondieren sollte darauf geachtet werden, wie die Hashtabellengröße m gewählt wird. Ist m beispielsweise 16, so ergibt sich bei der Sondierfolge:

...

$$16 \bmod 16 = 0$$

$$25 \bmod 16 = 9$$

$$36 \bmod 16 = 4$$

$$49 \bmod 16 = 1$$

...

Es ergeben sich also bereits benutzte Quadratzahlen, was zu einer unendlichen Schleife führt. Vermeiden lässt sich das durch die Wahl einer Primzahl für m .

Zum Aufwand des quadratischen Sondierens sei auf [Ottmann, 2002] verwiesen. Hier nur das Ergebnis:

$$C_n \approx 1 + \ln\left(\frac{1}{1-\alpha}\right) - \frac{\alpha}{2}$$

$$C'_n \approx \frac{1}{1-\alpha} - \alpha + \ln\left(\frac{1}{1-\alpha}\right)$$

Wie weiter oben erwähnt, müssen die Operationen Einfügen, Suchen und Löschen die Sondierreihenfolge und ihre Besonderheiten beachten. Die Operationen Einfügen, Suchen und Löschen eines Schlüssels s funktionieren folgendermaßen:

- **Einfügen:**

Betrachte $h(s)$. Wenn die Position schon ein Element enthält, betrachte die gemäß der Sondierungsfolge nächste Position. Fahre fort bis eine leere Position gefunden ist, speichere den Schlüssel dort ab.

- **Suchen:**

Betrachte $h(s)$. Ist der dortige Schlüssel identisch mit s , so ist er gefunden. Falls nicht, fahre gemäß der Sondierungsfolge fort und betrachte den Schlüssel an der nächsten Position. Vergleiche auch dort wieder und fahre fort bis s gefunden ist oder die betrachtete Position leer ist; in diesem Fall ist die Suche erfolglos.

Bei all diesen Verfahren stellt das Löschen eines Elements ein Problem dar. Man kann einen Schlüssel nicht einfach löschen, da dann die Sondierungsfolge gestört ist. Suchoperationen würden frühzeitig abbrechen, weil sie auf leere Positionen treffen und so später eingefügte Elemente nicht finden. Aus diesem Grund wird ein Element nicht einfach gelöscht, sondern nur als entfernt markiert. Einfügeoperationen betrachten diese Position dann als leer, während Suchoperationen diese Position als belegt betrachten. Dies ist nicht nur mit einem erhöhten Speicherplatzbedarf sondern auch mit einem höheren Aufwand für die Löschoperation verbunden. Daher werden offene Hashverfahren meist nur dann eingesetzt, wenn hauptsächlich eingefügt und gesucht wird.

Zur Verdeutlichung des Aufwandes der einzelnen Verfahren folgt exemplarisch eine Tabelle mit einigen Messwerten [Ottman, 2002]:

α	Direkte Verkettung		Lineares Sondieren		Quadratisches Sondieren	
	erfolgreich	erfolglos	erfolgreich	erfolglos	erfolgreich	erfolglos
0.5	1.25	0.5	1.5	2.5	1.44	2.19
0.9	1.45	0.9	5.5	50.5	2.85	11.4
0.95	1.475	0.95	10.5	200.5	3.52	22.05
1	1.5	1	-	-	-	-

Tabelle 3: Aufwandsvergleich Open Hashing

5.3.3 Hinweis zur Clusterbildung

Eine interaktive Applikation des Lernmoduls (siehe Abschnitt 7.3.5) zeigt einige interessante Aspekte der Clusterbildung auf.

So lässt sich beim linearen Sondieren etwa schon bei einem Belegungsfaktor von etwa 50% Clustering feststellen. Aus diesem Grunde sind die Felder der Interaktion mit entsprechenden Werten vorbelegt.

Vergleicht man die grafischen Belegungsmuster von Hashtabellen mit linearem bzw. quadratischem Sondieren, so fällt auf, dass das Muster beim quadratischen Sondieren keinesfalls besser, d.h. gleichmäßiger verteilt, aussieht. Die Ursache dafür liegt darin, dass man die Cluster beim quadratischen Sondieren mit bloßem Auge nicht erkennen kann. Lückenlose Bereiche stellen beim quadratischen Sondieren keine Cluster dar.

Der Unterschied liegt beim quadratischen Sondieren eben darin, dass insgesamt weniger Vergleiche notwendig sind, um einen Schlüssel einzufügen bzw. zu finden. Dies wurde in der interaktiven Applikation insofern berücksichtigt, als zusätzlich zu den grafischen Belegungsmustern statistische Informationen ausgegeben werden, die über die Gesamtanzahl und mittlere Anzahl der Vergleiche Auskunft geben. Auch die Anzahl der Vergleiche, die für die letzten zehn einzufügenden Schlüssel notwendig waren, wird angezeigt, da dieser Wert für weitere Einfügeoperationen ausschlaggebender ist. Für weitere Informationen siehe Abschnitt 7.3.5.

5.4 Rehashing

Wächst der Belegungsfaktor auf einen ungünstigen Wert nahe 1 an, so bietet es sich an, die Hashtabelle zu vergrößern. Es entsteht eine neue Hashtabelle T' mit Hashtabellengröße $m' > m$ und neuer Hashfunktion f' , da die größere Hashtabelle eine neue Hashfunktion erforderlich macht. Aus diesem Grunde müssen alle schon gespeicherten Elemente anhand der geänderten Hashfunktion f' neu gespeichert werden. Diese Umorganisation der Hashtabelle heißt *Rehashing*.

Hierzu gibt es nun zwei Möglichkeiten. Entweder man reserviert zusätzlich den kompletten Speicherplatz für die neue Hashtabelle T' und hat vorübergehend zwei

Hashtabellen im Speicher, was sehr speicherintensiv werden kann, oder man verlängert die erste Hashtabelle und reorganisiert innerhalb der neuen Tabelle.

Das Verfahren zur Umorganisation ist aus folgendem Grunde nicht trivial: Befindet sich auf dem neuen Platz für einen Schlüssel bereits ein anderer Schlüssel, darf dieser nicht einfach überschrieben werden, da er sonst verloren geht. Daher werden zwei Puffer benötigt, die jeweils gerade einen Schlüssel speichern können.

Der Algorithmus arbeitet folgendermaßen: Es werden alle Positionen $0..m$ durchlaufen, also die „alte Tabelle“ innerhalb der neuen Tabelle. Jeder Schlüssel der dabei gefunden wird, wird an seinen neuen Platz gespeichert. Nun kann es aber sein, dass an diesem neuen Platz schon ein Schlüssel gespeichert ist, weswegen dieser dann in Puffer 1 zwischengespeichert wird, bevor die Speicherung stattfindet. Nun muss der Schlüssel aus diesem Puffer an seine neue Position gespeichert werden. Falls sich dort wieder bereits ein Schlüssel befindet, so muss dieser auch irgendwo zwischengespeichert werden; daher die Notwendigkeit eines zweiten Puffers. Jetzt wird Schlüssel aus Puffer 1 abgelegt, Inhalt des Puffers 2 kommt in Puffer 1 und das ganze geht von vorne los, bis der Schlüssel in Puffer 1 endlich problemlos abgespeichert werden kann. Dann erst wird mit dem nächsten Schlüssel der „alten Tabelle“ weitergemacht.

Andere Rehashing-Verfahren finden sich in [Standish, 1980].

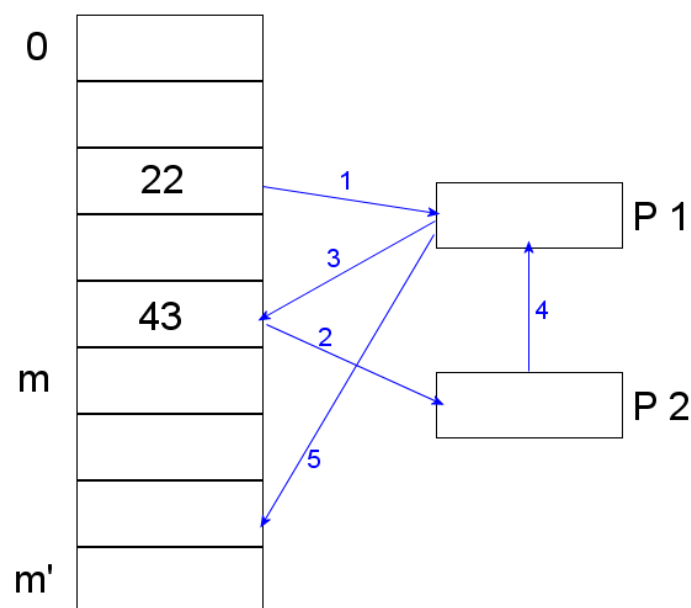


Abbildung 1: Rehashing

Der Algorithmus sieht so aus:

- Durchlaufe alle Positionen $0..m$ (also die „alte Tabelle“)
- Für jeden Schlüssel s , der gefunden wird:
 - Berechne die neue Position $h'(s)$
 - Befindet sich auf der neuen Position $h'(s)$ bereits ein Schlüssel r , dann
 - speichere ihn in den Puffer 1: $p1 := r;$
 - speichere s an der Position $h'(s)$ ab
 - solange der Puffer 1 voll ist:
 - Berechne die neue Position des Schlüssels in Puffer 1
 - Befindet sich dort ein Schlüssel q , dann
 - speichere ihn in Puffer 2: $p2 := q;$
 - speichere Schlüssel aus Puffer 1 an seiner neuen Position ab
 - speichere Schlüssel aus Puffer 2 in Puffer 1:
 $p1 := p2;$
 - ansonsten speichere Schlüssel aus Puffer 1 an seiner neuen Position ab
 - ansonsten speichere s an der Position $h'(s)$ ab
 - Fahre fort mit dem nächsten Schlüssel

Natürlich muss der Status des Puffers 1 (voll, leer) durch ein entsprechendes Flag (Boolean-Variable) mitverwaltet werden. Auch müssen Kollisionen behandelt werden, da ansonsten im Falle einer Kollision die Puffer 1 und 2 in einer Endlosschleife immer wieder mit denselben beiden Schlüsseln (nämlich Synonyme) belegt werden. Auf beides wurde hier aus Gründen der Übersichtlichkeit verzichtet.

6 Gestaltungsprinzipien und Anforderungen

In diesem Kapitel soll auf Gestaltungsprinzipien für die Entwicklung von Lernmodulen eingegangen werden. Der Autor hat versucht, möglichst viele der hier genannten Prinzipien in der Implementierung des Lernmoduls umzusetzen. Welche Prinzipien beachtet wurden und welche nicht, ist (mit Begründung) in Kapitel 7.3 nachzulesen.

6.1 Designprinzipien

Für die Entwicklung von Lernprogrammen sind einige Designprinzipien besonders bedeutsam, wie z.B.:

- **Einfachheit:**
Gerade die Zielgruppe unseres Lernmoduls (Studierende im Grundstudium) erfordert eine gewisse Einfachheit in Sprache und Bilder.
- **Verständlichkeit:**
Hierfür gibt es wenig konkrete Richtlinien, vielmehr ist es eine subjektive Eigenschaft jeden Autors.
- **Gleichbleibendes Aussehen:**
Alle Designelemente und Interaktionsobjekte sollten ein einheitliches Aussehen haben und wieder verwendet werden.
- **Gleichbleibende räumliche Anordnung räumlich wiederkehrender Elemente:**
Hierfür wurde der Bildschirm in Segmente aufgeteilt. Diese Aufteilung bleibt immer gleich.

- **Orientierungshilfen:**
Der Benutzer kann ständig sehen, wo er gerade ist und wie viel er noch vor sich hat.
- **Einfache Fehlermeldungen und Hilfe:**
Es gibt nur eine Hilfeseite, die alles zur Bedienung und Navigation des Lernprogramms beinhaltet und jederzeit von überall aufrufbar ist.

6.2 Die Minnesota Kriterien

Am Minnesota Research and Development Center, wo man sich mit der Gestaltung von Ausbildung beschäftigt, hat eine Forschergruppe ein „Instructional-Design-Model“ für Computerlernprogramme entwickelt, welches bei der Konzeption des Lernmoduls berücksichtigt wurde. Die vier Hauptprinzipien sind [Grillenbeck, 2000]:

- **Fidelity:**
realitätsnahe und problembezogene Situationsdarstellungen, auch mit Details, die den Lernenden Merkmale des Arbeitsablaufes oder der Arbeitsumgebung wieder erkennen lassen.
- **Visualization:**
bildhafte Darstellung der Sachverhalte mit möglichst konkretem Bezug auf die Lernenden und das Thema. Einfache und klare Bilder.
- **Range and Depth of Experience:**
Zugriff auf reichhaltige Erfahrungen aus verschiedenen Perspektiven. Beispiele und Aufgaben sollen auf unterschiedlichen Ebenen den Stoff vermitteln.
- **Mediation of Learner's Experience:**
Unterstützung des Verständnisses durch Hervorhebung wichtiger Aspekte. Dadurch bedeutet „Mediation“ auch Führung. Es muss darauf geachtet werden, auf welcher Lernstufe sich der Lernende befindet. Anfänger müssen anders

geleitet werden als Experten. Das Lernmodul „Hashing“ ist vom Typ her ein tutorielles System mit freier Lernersteuerung. So ist es möglich, dass ein Anfänger sich vom Programm leiten lässt, während der Fortgeschrittene die Kapitel und Aufgaben frei wählen kann, die er wiederholen möchte.

6.3 Ablaufsteuerung

Beim Entwurf eines Lernprogramms muss die sehr wichtige Entscheidung gefällt werden, ob der Benutzer oder das Programm den Ablauf vorgibt. Hierbei spielt die Zielgruppe eine wichtige Rolle: Kinder, Jugendliche, Studenten und Erwachsene erfordern unterschiedliche Vorgehensmuster. Im Falle des Lernmoduls „Hashing“ werden die beiden letztgenannten Zielgruppen adressiert. Aus diesem Grunde wird den Benutzern eine hohe Lernkompetenz zugesprochen und eine selbständige Arbeitsweise vorausgesetzt und so eine individuelle Abarbeitung zur Verfügung gestellt. Der Benutzer hat dabei zu jedem Zeitpunkt folgende vier Möglichkeiten:

- **Menüsteuerung:**
Über das Inhaltsverzeichnis kann zu beliebigen Kapiteln gesprungen werden
- **Rücksprung und Weiterblättern:**
Über ein Navigationssteuerkreuz kann vorwärts / rückwärts / zum vorigen Kapitel / zum nächsten Kapitel gesprungen werden
- **Abbruch:**
Durch das Schließen des Browserfensters kann das Programm jederzeit beendet werden.
- **Unterbrechung:**
Da das Lernmodul aus statischen Seiten besteht, kann der Lernprozess beliebig unterbrochen und zu einem späteren Zeitpunkt fortgesetzt werden, ohne dafür eine Funktion in Anspruch nehmen zu müssen.

6.4 Interaktionsstruktur

Im Zusammenhang mit Lernprogrammen bedeutet Interaktion die Gesamtheit aller Möglichkeiten, mit dem Lernprogramm zu interagieren, durch Aufgaben, Übungen aber auch interaktive Programmbestandteile sowie Bedienungselemente des Programms.

Neben den genannten Möglichkeiten zur Ablaufsteuerung können am Ende jeder Lektion Aufgaben zur Lernzielkontrolle eingefügt werden, Übungen zur Erarbeitung von Lehrstoff, ein Glossar zum Nachschlagen von Fachbegriffen sowie eine Hilfefunktion.

- **Aufgaben:**

Nachfolgend eine Tabelle mit den wichtigsten Aufgabenformen und ihrer didaktischen Einsatzmöglichkeiten in Lernprogrammen [Grillenbeck, 2000]:

Aufgabenform	Didaktische Eignung	Bemerkungen
<i>Auswahlaufgaben</i> (Einfachauswahl, Multiple Choice)	<ul style="list-style-type: none"> • Einfachauswahl nur zur Überprüfung der Lernziele • Multiple-Choice Aufgaben ermöglichen die Überprüfung relativ komplexer Lernleistungen, vor allem in der Form „n aus x“ • Anwendungsbezogenes Wissen, nicht Faktenwissen, kann damit gut überprüft werden 	<ul style="list-style-type: none"> • Gefahr des Ratens • z.B. Unterscheidung geeignete oder ungeeignete Verhaltensweisen in der Praxis • Anbieten sinnvoller Distraktoren
<i>Zuordnungsaufgaben</i> (Objekte bewegen und gruppieren)	<ul style="list-style-type: none"> • Auswahlidee in graphischer Form • Überprüfung komplexer Lernziele 	<ul style="list-style-type: none"> • Einfachere Varianten durch Ankreuzen „x gehört zu y“ denkbar
<i>Reihenfolgeaufgaben</i> (Objekte richtig	<ul style="list-style-type: none"> • Anzahl von auszuwählenden Objekten graphisch 	<ul style="list-style-type: none"> • Praxisbezug für eine Ablaufbeschreibung:

anordnen)	aufbereitet in eine sinnvolle Reihenfolge bringen	„Zuerst... dann...“
Einschätzungsaufgaben (z.B. Prozentsatz einer Grundgesamtheit abschätzen)	<ul style="list-style-type: none"> • Aktivierung des Lernenden • Keine komplexen Lernleistungen überprüfbar 	<ul style="list-style-type: none"> • Innerhalb einer Skala soll ein Bereich/Punkt angegeben werden, um eine Einschätzung abzugeben
Zahleneingaben (Ergebnisse eintippen)	<ul style="list-style-type: none"> • Ist die Zahleneingabe Ergebnis einer komplexen Berechnung, kann sie anspruchsvolle Lerninhalte abfragen 	<ul style="list-style-type: none"> • Schwierige Rückmeldung, da der Denkweg des Lernenden nicht nachvollzogen werden kann
Texteingaben (Lückentexte, komplette Eingaben)	<ul style="list-style-type: none"> • Im Prinzip beliebige Möglichkeiten der Lernzielkontrolle • Vergleich mit Musterlösung 	<ul style="list-style-type: none"> • Technische Werkzeuge für „intelligente“ Rückmeldungen und Überprüfungen überfordert, da das System im Idealfall die Antwort syntaktisch und semantisch testen müsste

Tabelle 4: Aufgabenformen in Computer-Lernprogrammen

- **Glossar:**

Über ein Glossar können Fachbegriffe nachgeschlagen werden. Diese sind im Text farblich hervorgehoben und können durch Klick nachgeschlagen werden. Das Layout und die Interaktionskomponenten entsprechen dabei den WWW-Standards (Hyperlinks, etc.) und sollten in ihrer Bedienung daher keine Schwierigkeiten bereiten.

- **Hilfefunktion:**

Die Programmbedienung wird in einer Einleitungssequenz auf einer Seite erläutert. Diese Seite kann über das Inhaltsverzeichnis von überall aus aufgerufen werden. Zusätzlich wird empfohlen, die Lektionen des Lernmoduls in der vorgegebenen Reihenfolge zu bearbeiten, da sie inhaltlich aufeinander aufbauen.

6.5 Bildschirmdesign

Es existiert eine Fülle von Regeln für das Bildschirmdesign und die Gestaltung von Benutzeroberflächen. Im Wesentlichen befassen sie sich mit Text- und Graphiklayout, Aufteilung des Bildschirms für Steuerungsfunktionen und Lerninhalte und Integration von Orientierungshilfen.

Für das Textlayout gelten folgende Grundregeln:

- Der Bildschirm soll gleichmäßig gefüllt werden.
- Schriftgrößen wählen, die aus einem Betrachtungsabstand von 50-100 cm mühelos zu erkennen sind. Außerdem Schriftarten ohne Serifen wählen, da Serifenschriften auf Bildschirmen leicht unscharf wirken.
- Eine zu große Vielfalt an Schrifttypen erzeugt Unruhe im Bildaufbau und beim Betrachten und erschwert die Lesbarkeit.
- Zeichen- und Zeilenabstand sollten eine optische Verschmelzung der Wörter verhindern.
- Aus ergonomischer Sicht sind „dunkel auf hell“ Darstellungen vorzuziehen.
- Keine langen Fließtextpassagen verwenden.

Für das Bildschirmlayout sollte folgendes beachtet werden:

- Einheitliches, stringentes Layoutkonzept verwenden.
- Die Orientierung über Herkunft, momentane Position und Fortsetzungsmöglichkeiten im Programmablauf sollte statisch in einem festen Bildschirmbereich realisiert sein, z.B. in einer Steuer- bzw. Navigationsleiste.

- Verzögerungen beim Bildaufbau sollten eingebaut werden, um die Aufnahmefähigkeit der Lernenden nicht zu überfordern. Dynamik in einen statischen Prozess zu bringen und dramaturgische Effekte zu erzielen.

7 Das Lernmodul „Hashing“

In diesem Kapitel wird das konkrete Lernmodul „Hashing“, das begleitend zu dieser Diplomarbeit entstanden ist, näher erläutert. Nach der Betrachtung einiger grundlegender Gestaltungsprinzipien gehen wir auf den Aufbau, den Inhalt, die enthaltenen interaktiven Applikationen und die Merkmale des Lernmoduls ein.

7.1 Aufbau

Das Lernmodul „Hashing“ ist in Kapitel und Seiten gegliedert. Jedes Kapitel besteht aus einer unterschiedlichen Anzahl von Seiten und behandelt einen Teilaspekt des Themengebiets „Hashing“. Es gibt sechs Kapitel:

1. Wieso Hashing?
2. Hashing - Grundlagen
3. Hashfunktionen
4. Adresskollisionen
5. Rehashing
6. Literatur

Diese Kapitel sind wiederum in Seiten aufgeteilt, die jeweils eine Lerneinheit darstellen. Eine Lernmodulseite umfasst dabei ca. eine Bildschirmseite.

Der Benutzer liest sich eine Seite durch und blättert dann weiter zur nächsten Seite oder noch mal zurück zur vorherigen Seite. Dies geschieht mit Hilfe der Navigationsleiste, die sich am unteren Bildschirmrand befindet:



Abbildung 2: Navigationsleiste

Zusätzlich kann er Kapitel vor-/zurückblättern, das Inhaltsverzeichnis, das Glossar oder den Index aufrufen. Außerdem kann er mittels des „Home“-Buttons zur allerersten Startseite des Lernmoduls gelangen.

In den Text sind kleinere interaktive Applikationen eingestreut. Dadurch werden gewisse Zusammenhänge vertieft. Siehe dazu Abschnitt 7.3.

Auf diese Weise bearbeitet der Benutzer das Lernmodul linear durch, bis er ans Ende gelangt. Diese lineare Vorgehensweise wird im einleitenden Kapitel 1 vorgeschlagen, jedoch ist sie nicht zwingend notwendig. Erfahrenere Benutzer können durchaus Lerneinheiten überspringen bzw. gezielt Inhalte aufrufen. Dies wird durch das Inhaltsverzeichnis mit seinen Kapitel-Links und auch Seiten-Links unterstützt. Wünschenswert wäre hier eine Art Lesezeichenverwaltung (siehe Abschnitt 8.2).

Im gesamten Text sind Fachbegriffe eingestreut, die farblich hervorgehoben sind. Klickt man auf einen, so öffnet sich ein kleines Fenster mit der Erklärung des Fachbegriffs und evtl. Verweisen auf andere verwandte Fachbegriffe. Es ist ebenfalls ein Glossar vorhanden, in dem sämtliche Fachbegriffe mitsamt Erklärung aufgelistet sind. Man erreicht dieses über den Button „Glossar“ in der Navigationsleiste.

Da die Fachbegriffe im Text sehr häufig vorkommen und zu viele farblich hervorgehobene Wörter die Lesbarkeit drastisch verschlechtern, wurde diesbezüglich eine Kompromisslösung gewählt: Nur das erste Vorkommen eines Fachbegriffes in einem Kapitel wird farblich markiert. Alle anderen Vorkommen bleiben in der Textfarbe und können auch nicht angeklickt werden. Es wird davon ausgegangen, dass innerhalb eines Kapitels die Seiten linear bearbeitet werden.

Der Index, den man über den Button „Index“ in der Navigationsleiste erreicht, listet alle Fachbegriffe mitsamt Hypertext-Links zu ihren Vorkommen im Text auf.

Die Menge der Fachbegriffe sowie die Erklärungen können vom Benutzer angepasst werden. Die Verwaltung der Fachbegriffe findet in der Datei `lernmodul.js` statt (mehr dazu in Anhang B).

7.2 Inhalt

Der Inhalt des Lernmoduls „Hashing“, also der Lernstoff, richtet sich nach dem Inhalt des Kapitels 5, umfasst also eine grundlegende Einführung in das Thema „Hashing“

sowie einige besondere Verfahren, wie etwa das „Rehashing“. Selbstverständlich gibt es viel mehr Erkenntnisse, Methoden und Verfahren in diesem Gebiet, jedoch würde eine ausgiebige Behandlung all dieser nicht nur den Rahmen dieser Arbeit sprengen, sondern auch eine unangemessen hohe Konzentration auf dieses Thema im Grundstudium Informatik darstellen.

Somit gliedert sich der Inhalt des Lernmoduls in folgende Kapitel:

1. Wieso Hashing?

Abgrenzung des Themas: Warum ist Hashing notwendig/gut/beliebt?

2. Hashing – Grundlagen

Grundlagen und Grundbegriffe, die später ständig auftauchen, werden in diesem Kapitel erläutert. In dieser Arbeit wurden diese in Kapitel 5.1 behandelt.

3. Hashfunktionen

Verschiedene Möglichkeiten für Hashfunktionen sind Thema dieses Kapitels. Ausgehend von Abschnitt 5.2 dieser Arbeit werden Verfahren wie die Divisions-Rest-Methode, die multiplikative Methode etc. vorgestellt.

4. Adresskollisionen

Adresskollisionen und mögliche Strategien zu ihrer Auflösung sind Thema in diesem Kapitel des Lernmoduls. Hierbei werden Verkettung der Überläufer und Open Hashing näher betrachtet.

5. Rehashing

Ein besonderes Verfahren wird in diesem Kapitel vorgestellt: das Rehashing, das bei Vergrößern der Hashtabelle durchgeführt werden muss.

6. Literatur

Eine Auswahl an weiterführender Literatur zum Thema „Hashing“.

Ein Glossar für die Fachbegriffe und ein Index runden den Inhalt des Lernmoduls ab. Zum besseren Verständnis wurde für die Hashverfahren ein durchgehendes Beispiel

benutzt: die Namen der zwölf griechischen Götter des Olymp: Aphrodite, Apollo, Ares, Artemis, Athene, Demeter, Dionysos, Hephaistos, Hera, Hermes, Poseidon und Zeus.

7.3 Interaktive Applikationen

Ein wesentliches Merkmal des Lernmoduls „Hashing“ sind die darin enthaltenen interaktiven Module, nachfolgend interaktive Applikationen genannt. Sie sollen dem Benutzer gewisse Zusammenhänge bzw. Verfahrensweisen verdeutlichen. Außerdem können sie zu Lehrzwecken in einer Vorlesungsveranstaltung verwendet werden.

In diesem Abschnitt sollen die Funktionsweise, die Programmierung sowie Probleme bei der Realisierung der einzelnen interaktiven Applikationen beschrieben werden.

7.3.1 Hashing mit der Divisions-Rest-Methode (Kap. 3, S. 10)

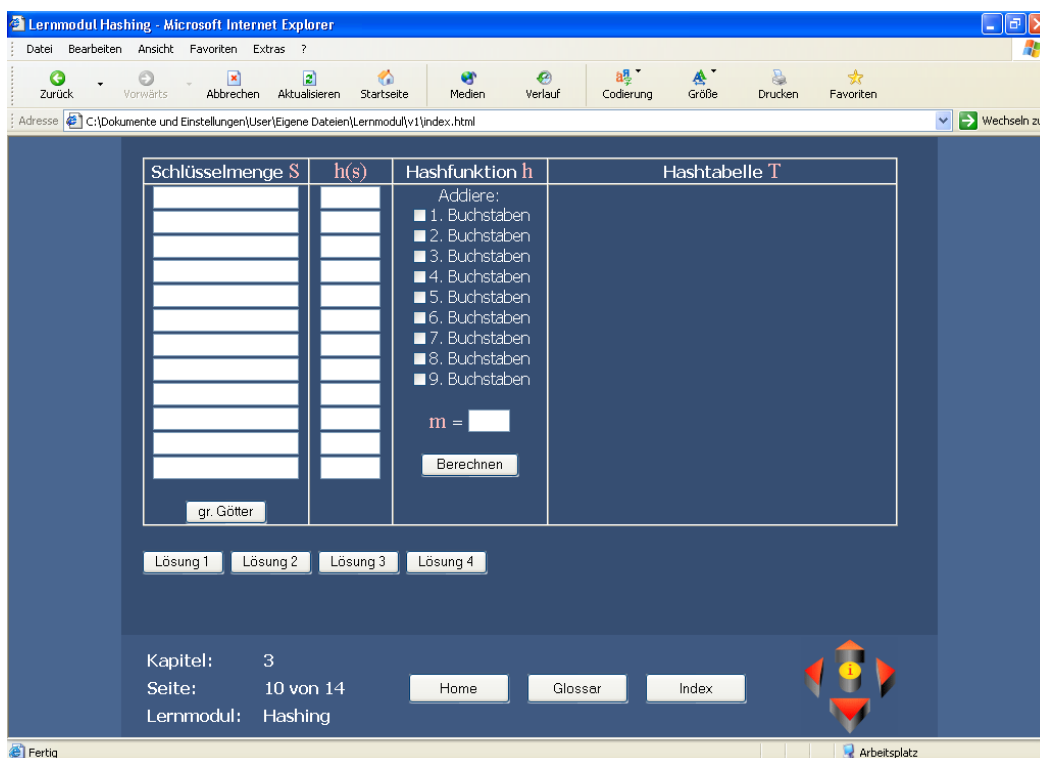


Abbildung 3: Interaktive Applikation 1 - Hashing mit der Divisions-Rest-Methode

- Funktionsweise:

Auf der linken Seite gibt der Benutzer bis zu zwölf Schlüssel ein. Mit einem Klick auf den Button „gr. Götter“ werden automatisch die Namen der zwölf Götter in die Felder eingetragen (damit Tipparbeit gespart wird). In der Mitte spezifiziert man nun die Hashfunktion genauer, indem man zum einen die Buchstaben wählt, die bei der Summierung berücksichtigt werden sollen, und zum anderen die Hashtabellengröße m angibt. Ein Klick auf „Berechnen“ startet nun das Hashverfahren, und es werden die Hashwerte der Schlüssel berechnet und in die dafür vorgesehene Spalte eingetragen. Gleichzeitig wird eine Hashtabelle erstellt und auf der rechten Seite angezeigt. Da hier noch keine Kollisionsauflösungsstrategie benutzt wird, kann es dazu kommen, dass sich mehrere Schlüssel dieselbe Tabellenposition teilen. Dies wird durch ein grafisches Symbol in Form eines gelben Dreiecks angezeigt.

Diese Applikation soll die Häufigkeit von Adresskollisionen visualisieren und ein Gespür für die Schwierigkeit vermitteln, eine injektive Hashfunktion zu finden.

- Programmierung:

Die Applikation ist – wie jede interaktive Applikation dieses Lernmoduls – in Javascript programmiert. Javascript ist eine ziemlich mächtige Scriptsprache, die vom Browser selbst (zumindest von den häufigsten: Internet Explorer, Netscape Navigator und Opera) unterstützt wird. Es muss kein Zusatzmodul installiert werden, was die Portabilität sehr fördert. Es sollte jedoch darauf geachtet werden, dass Javascript im Browser nicht deaktiviert sein darf.

Javascript bietet (fast) alles, was eine Hochsprache wie z.B. C oder Java auch bietet: Funktionenkonzept, Kontrollstrukturen, eingebaute Funktionen, Objektmodell etc. Lediglich einige Sicherheitsaspekte wurden wegen der Hauptverwendung im Internet berücksichtigt: So kann man mit Javascript nicht auf einen Client-Rechner zugreifen und Dateien erstellen bzw. ändern, sondern nur die angezeigte HTML-Datei beeinflussen.

Mit Javascript lassen sich dynamische Websites programmieren, also Websites, die kein statisches Erscheinungsbild haben, sondern abhängig von Benutzereingaben bzw. Ereignissen (z.B. Mausklick, Mausbewegung etc.) bestimmte Aktionen durchführen, z.B. ändert ein Button seine Farbe, wenn der Benutzer mit der Maus darüberfährt. Auch zur Verarbeitung von Formularen ist Javascript sehr nützlich.

Aufgrund dieser Eigenschaften ist Javascript für das Lernmodul „Hashing“ ideal. In dieser Applikation werden die Benutzereingaben in einem Formular erfasst und danach mit Javascript bearbeitet, um danach wieder in Formularfeldern als Ausgabedaten ausgegeben zu werden.

- Probleme:

Größere Probleme gab es bei der Programmierung dieser Applikation nicht. Eine Hürde stellte die Behandlung der vielen Formularfelder dar, die nur mittels ihres Namens angesprochen werden können, der bei vielen Feldern bis auf die letzten zwei Ziffern ähnlich ist. Um auf alle systematisch zuzugreifen, bietet sich die Verwendung der EVAL-Funktion an, der man einen STRING übergeben kann, der Programmcode enthält, worauf dieser STRING einfach ausgewertet und ausgeführt wird (!). Die EVAL-Funktion wurde ebenfalls zur Summenbildung benutzt, in der vom Benutzer spezifizierte Buchstaben berücksichtigt werden sollen, aber für jeden Schlüssel immer dieselben. Hier wurde der Summenstring mit den entsprechenden Buchstaben einmal erzeugt und mittels der EVAL-Funktion für jeden Schlüssel ausgewertet.

Eine andere Hürde stellte die Summierung der ASCII-Werte dar, da bei zu kurzen Wörtern nicht vorhandenen Buchstaben nicht etwa zu null, sondern zu „NaN“ ausgewertet wurden. „NaN“ ist ein Javascript-spezifischer Wert, der „not a number“ bedeutet und einen eigenen Datentyp darstellt, weswegen er mit keinem anderen verglichen bzw. verrechnet werden kann. Die Lösung dieses Problems war jedoch ziemlich einfach: eine eigene Funktion zur ASCII-Code-Ermittlung musste definiert werden, die bei vorhandenen Buchstaben die Standardfunktion benutzt und bei nichtvorhandenen Buchstaben einfach null zurückgibt.

Eine letzte Hürde stellte das Erscheinen der fertig berechneten Hashtabelle dar. Die Standardfunktion `document.write()` schreibt nämlich immer in eine neue Datei. Die alte, gerade angezeigte Datei verschwindet dann einfach, was in diesem Fall nicht beabsichtigt war. Auch hier war die Lösung relativ einfach, nicht jedoch ihre Auffindung. Es muss ein Bereich im HTML-Code definiert werden, der eine ID bekommt und leer ist. Später kann mittels der ID dieser Bereich angesprochen werden und mittels der Eigenschaft `innerHTML` weiterer HTML-Code in diesen Bereich hinzugefügt werden. Die Schwierigkeit lag darin, dass HTML-Elemente normalerweise

nicht über ihre ID, sondern über ihren Namen (`name`) angesprochen werden. In diesem Falle musste also vom üblichen Javascript-Programmierstil abgewichen werden.

7.3.2 Lineares Sondieren mit einstellbarer Schrittweite (Kap. 4, S. 11)

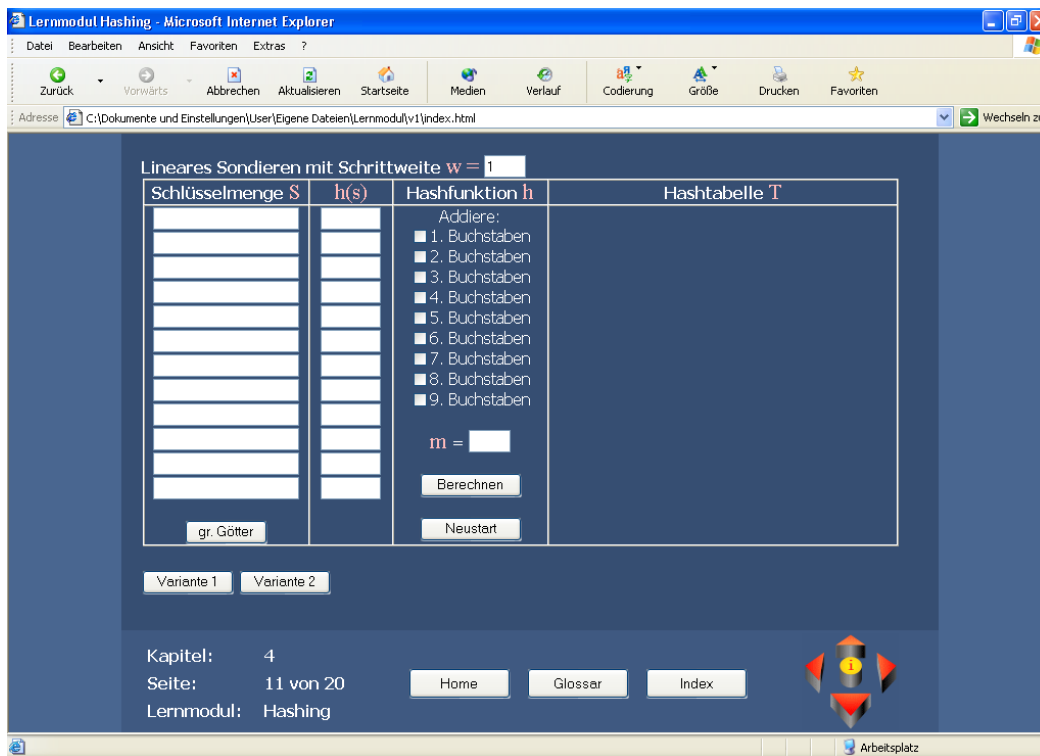


Abbildung 4: Interaktive Applikation 2 - Lineares Sondieren mit einstellbarer Schrittweite

- Funktionsweise:

Die Funktionsweise dieser Applikation ist ähnlich der Funktionsweise des vorherigen Moduls. Der Benutzer gibt auch hier wieder Schlüssel ein oder bedient sich der vorgegebenen Götternamen. Nach der Spezifizierung der Hashfunktion wird wieder die berechnete Hashtabelle angezeigt. Allerdings findet diesmal eine Einzelschrittberechnung statt. Bei jedem Klick auf den Button „Berechnen“ wird die Hashtabellenposition des nächsten Schlüssels berechnet, dieser eingetragen und seine Position mit einem gelben Pfeil angezeigt. Auf diese Weise klickt sich der Benutzer durch und kann das Verfahren nachvollziehen. Kommt es zu einer Adresskollision, so wird dies durch eine Meldung angezeigt, in der auf die Adresskollision hingewiesen

wird und auch schon die alternative Position des Elementes genannt wird. Gibt es keine alternative Position (z.B. wenn das Sondierverfahren in einen Zyklus gerät), so wird dies ebenfalls angezeigt und das Hashverfahren abgeschlossen.

Ist das Hashverfahren beendet, so bewirkt ein weiterer Klick auf den Button „Berechnen“ nichts mehr. Der Benutzer *muss* auf den Button „Neustart“ klicken, bevor eine neues Hasverfahren gestartet werden kann. Dies löscht alle Formulareingaben und den internen Zähler und wirkt also wie ein „Reset“.

- Programmierung:

Diese Applikation wurde ebenfalls in Javascript programmiert. Da sie zu einem großen Teil der vorherigen Applikation sehr ähnlich ist, konnte ein Großteil des Codes übernommen werden. Unterschiede ergaben sich nur in der Implementierung der Einzelschrittberechnung, beim Sondieren sowie beim Anzeigen des Pfeils.

- Probleme:

Die Einzelschrittberechnung war nur mit einem kleinen Trick zu realisieren. So musste bei jedem Klick auf „Berechnen“ die Hashfunktion wissen, in welchem Schritt sie sich gerade befindet. Ihre Informationen kann die Funktion aber nur aus Formularfeldern beziehen. Also war es notwendig, ein verstecktes Formularfeld (`input type="hidden"`) zu definieren, in dem die aktuelle Schrittzahl gespeichert wird. Diese muss natürlich nach jedem Schritt entsprechend inkrementiert werden. Das Ende des Verfahrens wird dabei durch den Wert 99 angezeigt.

Das Sondieren war kein größeres Problem. Mittels einer `while`-Schleife wird die neue Position ermittelt:

```
i=0;
while ((belegt[(oldplace + i*w)%m]) && (i<m)) {
    i++;
}
newplace = (oldplace + i*w)%m;
```

Es musste lediglich darauf geachtet werden, dass kein Zyklus auftritt, dass das Verfahren nicht bereits abgeschlossen ist etc. Dies wurde durch entsprechende `if`-Statements erreicht.

7.3.3 Clustering anhand von Zufallszahlen mit linearem Sondieren (Kap. 4, S. 14)

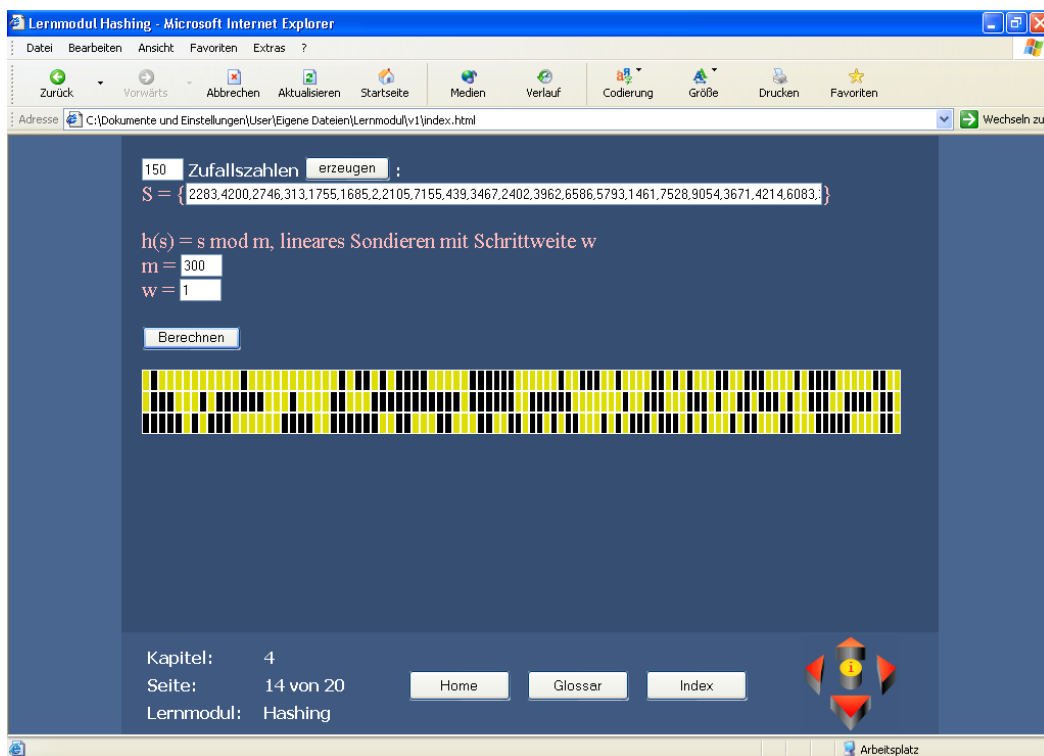


Abbildung 5: Interaktive Applikation 3 - Clustering anhand von Zufallszahlen mit linearem Sondieren

- Funktionsweise:

Um das Clustering bei linearem Sondieren sichtbar zu machen, wurde diese Applikation entworfen, welche eine vorgegebene Anzahl von Zufallszahlen erzeugt, diese in eine Hashtabelle einträgt und das Ergebnis visualisiert.

Der Benutzer kann oben links zunächst angeben, wie viele Zufallszahlen erzeugt werden sollen. Diese werden dann durch einen Klick auf den Button „Erzeugen“ schließlich generiert und in das darunterliegende Formularfeld eingetragen. Alternativ

kann der Benutzer auch selbst einige Zahlen angeben und diese in die Zeile eintippen (durch Kommata getrennt).

Als nächstes spezifiziert man die Hashtabellengröße und die Schrittweite für das lineare Sondieren. Ein Klick auf „Berechnen“ startet nun das Hashverfahren und speichert jede Zufallszahl in die Hashtabelle ab. Gleich danach (je nach eingegebenen Werten und Rechnerleistung können mehrere Sekunden vergehen) wird eine grafische Übersicht über die Hashtabelle ausgegeben. Dabei werden alle belegten Positionen gelb, alle leeren Positionen schwarz dargestellt, jeweils 100 Positionen in einer Reihe. Bewegt man den Mauszeiger auf ein Feld, so wird der Index des Feldes gefolgt von seinem Inhalt (der Zufallszahl) angezeigt. Auf diese Weise kann man stichprobenartig feststellen, welche Zahlen wo abgespeichert wurden, und welche Zahlen z.B. auf ihren „rechtmäßigen“ Platz gekommen sind und welche nicht.

Experimentiert man mit verschiedenen Belegungsfaktoren, so wird der Zusammenhang zwischen Belegungsfaktor und Auftreten von Clustern klar, die durch aneinanderhängende gelbe Felder sehr schön sichtbar sind (allerdings nur bei Schrittweite gleich eins, bei höheren Schrittweiten sind die Cluster mit bloßem Auge kaum wahrnehmbar).

- Programmierung:

Diese Applikation hat mit den vorherigen Applikationen nicht sehr viel gemeinsam, weswegen sie komplett neu entworfen und programmiert werden musste. Das Formular hat ein gänzlich anderes Erscheinungsbild, und die Ausgabe unterscheidet sich auch sehr stark.

Übernommen werden konnte der Code für das lineare Sondieren sowie für die spätere Anzeige von HTML-Code in der vorhandenen und gerade angezeigten Seite.

- Probleme:

Auch hier gab es keine Probleme, die sich nicht durch kurzes Nachdenken bzw. Recherchieren lösen ließen.

Ein Punkt sind die „Tooltips“, also die kleinen Hinweise auf Index und Inhalt einer Tabellenposition, die beim Drübergehen mit dem Mauszeiger angezeigt werden. Eine Recherche ergab, dass HTML hierzu ein Attribut `title` bereitstellt, das diese Tooltips

enthält. Dieses Attribut muss natürlich bei der Generierung des HTML-Codes für die Hashtabelle mitgeführt werden und entsprechend gesetzt werden.

Ein anderer Punkt war die Generierung der Hashtabelle, implementiert als HTML-Tabelle mit Tabellenzellen verschiedener Hintergrundfarbe. Hier mussten die Schleifenbedingungen anhand der Tabellengröße m sorgfältig gesetzt werden, um auch nicht durch hundert teilbare Tabellengrößen adäquat darzustellen.

7.3.4 Quadratisches Sondieren (Kap. 4, S. 17)

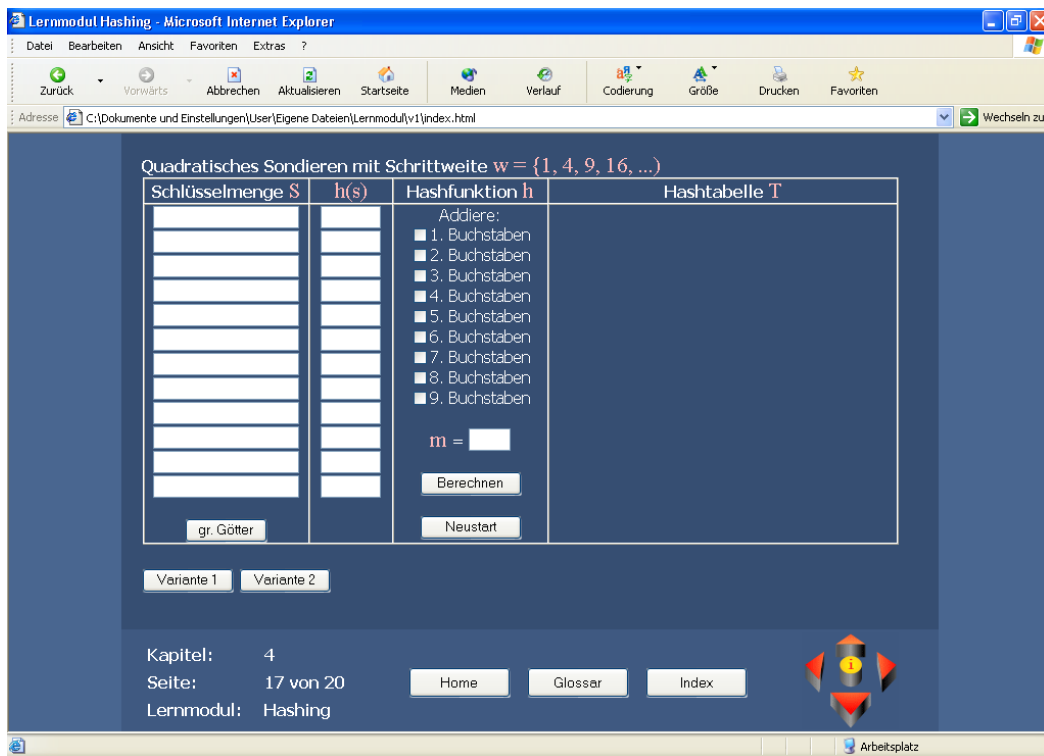


Abbildung 6: Interaktive Applikation 4 - Quadratisches Sondieren

- Funktionsweise:

Diese Applikation ist der zweiten (Abschnitt 7.3.2) sehr ähnlich. Auch hier werden Schlüssel eingegeben und danach schrittweise die Hashwerte berechnet und die Hashtabelle angezeigt. Der einzige Unterschied besteht in der Wahl des Sondierverfahrens: es liegt quadratisches Sondieren zugrunde, was auch in der obersten Zeile angezeigt wird. Am besten nachvollziehen kann man das im Falle einer

Adresskollision und der danach vorgeschlagenen alternativen Speicherposition, die entweder 1, 4, 9 etc. Plätze vom eigentlichen Funktionswert entfernt liegt.

- Programmierung:

Zur Programmierung gibt es außer den in Abschnitt 7.3.2 genannten Aspekten nichts hinzuzufügen, außer dass beim Sondierverfahren nicht linear sondern quadratisch nach weiteren Tabellenpositionen gesucht wird. Hierdurch ergibt sich folgender Programmcode:

```
i=0;
while ((belegt[(oldplace+i*i)%m]) && (i<m)) {
    i++;
}
newplace= (oldplace+i*i)%m;
```

Der einzige Unterschied ist also in der while-Bedingung zu finden, in der nicht $i*w$, sondern $i*i$ gerechnet wird, um die Sprungweite zur nächsten abzu prüfenden Tabellenposition zu berechnen.

- Probleme:

Nennenswerte Probleme gab es bei der Implementierung dieser interaktiven Applikation nicht.

7.3.5 Clustering anhand von Zufallszahlen mit quadratischem Sondieren (Kap. 4, S. 19)

250 Zufallszahlen erzeugen :
 $S = \{ 8862, 6974, 9616, 3561, 1268, 4479, 6712, 7089, 2412, 6356, 1300, 9207, 1938, 4297, 9776, 9726, 9701, 695, 8108, 7299, 1 \}$

$h(s) = s \bmod m$, lineares Sondieren mit Schrittweite w bzw. quadratisches Sondieren
 $m = 300$
 $w = 1$

Berechnen

Belegungsfaktor $\alpha = 0.83$

quadratisches Sondieren

Anzahl der Vergleiche: 515
 Letzte 10 Elemente: 53 (10.3%)
 Mittl. Anzahl der Vergleiche: 2.06

lineares Sondieren

Anzahl der Vergleiche: 823
 Letzte 10 Elemente: 164 (19.9%)
 Mittl. Anzahl der Vergleiche: 3.29

Kapitel: 4
 Seite: 19 von 20
 Lernmodul: Hashing

Home Glossar Index

Abbildung 7: Interaktive Applikation 5 - Clustering anhand von Zufallszahlen mit quadratischem Sondieren

- Funktionsweise:

Diese interaktive Applikation soll in Anlehnung an die Applikation aus 7.3.3 den Unterschied zwischen linearem und quadratischem Sondieren vermitteln. Auch hier werden auf dieselbe Weise Zufallszahlen generiert, die Hashtabellengröße m mitsamt Schrittweite w spezifiziert. Diesmal werden jedoch zwei Hashtabellen generiert und angezeigt: eine mit linearem Sondieren und eine mit quadratischem Sondieren.

Auf diese Weise soll schon optisch sofort ein erster Eindruck über den Unterschied dieser beiden Verfahren vermittelt werden. Zusätzlich werden einige statistische Informationen angezeigt:

- Anzahl der Vergleiche: Die Gesamtanzahl aller nötigen Vergleiche, um für alle Zufallszahlen eine leere Tabellenposition zu finden.
- Mittlere Anzahl der Vergleiche: Die mittlere Anzahl der Vergleiche pro Zufallszahl, berechnet als der Quotient aus Anzahl der Vergleiche und Anzahl der Zufallszahlen
- Letzte 10 Elemente: Falls noch zusätzliche Zufallszahlen eingefügt werden würden, so wäre für diesen Aufwand die Zahl der letzten Vergleiche ausschlaggebend. Deshalb wird die Anzahl der Vergleiche der letzten zehn Zufallszahlen gesondert addiert und dieser Wert hier ausgegeben. Außerdem wird noch der Anteil an den Gesamtvergleichen prozentual angegeben.

Ein weiterer Zusatz ist noch die Anzeige des Belegungsfaktors ganz oben, der als Quotient zwischen Anzahl der Zufallszahlen und Hashtabellengröße m berechnet wird.

- Programmierung:

Die Programmierung entsprach im Großen und Ganzen der Programmierung der interaktiven Applikation aus Abschnitt 7.3.3, weswegen große Teile des Programmcodes wieder verwendet werden konnten. Der einzige Unterschied bestand darin, dass nun zwei Hashtabellen berechnet werden und ausgegeben werden müssen und dass zusätzlich einige statistische Informationen berechnet und angezeigt werden sollen.

- Probleme:

Das Problem der zwei Hashtabellen wurde trivial gelöst: Der Programmcode wurde einfach kopiert und die entsprechende Änderung für das Sondierverfahren vorgenommen. Einziger Punkt: Die beiden Hashtabellen sollen erst nach der Berechnung beider ausgegeben werden. Da aber die Hashtabelle als HTML-Code in einer STRING-Variable erzeugt wird, wird der HTML-Code der zweiten Hashtabelle

einfach ebenfalls an diese STRING-Variable gehängt, bevor diese Variable ausgegeben wird.

Die Berechnung der statistischen Werte stellt aufgrund der ausreichend vorhandenen Bezugsgrößen kein nennenswertes Problem dar.

7.4 Allgemeine Merkmale

In diesem Abschnitt sollen nun einige Merkmale des Lernmoduls „Hashing“ näher betrachtet werden und die Entscheidung ihrer Verwendung begründet werden. Ausgangspunkt für diese Betrachtungen sind die Feststellungen in Kapitel 3, 4 und 6.

7.4.1 Lernziele

In Kapitel 3.1 wurden einige Lernziele genannt. Von diesen hat das Lernmodul „Hashing“ folgende Lernziele:

- Erwerb von Wissen
- Fähigkeit zur Anwendung des erworbenen Wissens
- Verstehen der inhaltlichen Aussagen

Die Erreichung weiterer erwähnter Lernziele (Schaffung von neuem Wissen durch Synthese, Artikulationsfähigkeit, Kommunikationsfähigkeit, etc.) ist im Rahmen eines Lernmoduls entweder nicht realisierbar oder nicht erwünscht.

7.4.2 Motivation

In Abschnitt 3.2 wurden einige Lernmotive genannt, von denen für das Lernmodul „Hashing“ die gegenstandsbezogenen und leistungsbezogenen Lernmotive eine Rolle spielen.

Gegenstandsbezogene Motivationshilfen sind in Kapitel 1 des Lernmoduls zu finden, in dem die Gründe für die Notwendigkeit bzw. Vorteile des Hashing aufgezeigt werden.

Leistungsbezogene Motivationshilfen sind ebenfalls zu finden. Viele Dinge, vor allem wichtige Aspekte, werden öfter genannt, damit sie besser im Gedächtnis haften bleiben. Außerdem gibt es ab und zu ein paar sehr kurze Seiten, die nur aus ein paar Sätzen bestehen, um den Lernfluss zu unterstützen. Zu viele lange Seiten mit viel Text demotivieren den Benutzer ziemlich stark.

7.4.3 Didaktische Prinzipien

Auch die didaktischen Prinzipien, die in Abschnitt 3.4 behandelt wurden, wurden beachtet:

- **Prinzip des aktiven Lernens:**
Die aktive Teilnahme des Lernalers wird durch die eingebauten interaktiven Applikationen sichergestellt.
- **Integrationsprinzip:**
Durch Rückgriffe auf bereits Erwähntes und Referenzen auf andere Kapitel werden ständig Sinnzusammenhänge geschaffen. Auf manche wichtige Lösungsmöglichkeiten wird sogar vorgegriffen und so zusätzlich motiviert.
- **Prinzip der Veranschaulichung:**
Durch eine geeignete Gliederung des Lernmoduls werden immer nur wenige neue Lerninhalte hinzugefügt.
- **Prinzip der Stabilisierung:**
Dieses Prinzip ist innerhalb eines einzelnen Lernmoduls schwer umzusetzen und wurde daher in diesem Lernmodul nicht berücksichtigt. Es ist Aufgabe einer Lehrveranstaltung, in der u.a. Hashing gelehrt wird, dieses Wissen später in neue Zusammenhänge einzubetten, etwa im Rahmen eines Vergleichs mit anderen Verfahren wie z.B. binäre Suchbäume, AVL-Bäume etc.

- **Operatives Prinzip:**
Dieses Prinzip steht in engem Zusammenhang mit dem Prinzip des aktiven Lernens und wird daher auf dieselbe Weise unterstützt: interaktive Applikationen innerhalb des Lernmoduls.
- **Prinzip der Lebensnähe und Aktualität:**
Durch bekannte Beispiele aus dem Alltag (etwa das Telefonbuchbeispiel) werden Bezüge zur aktuellen Lebenswelt hergestellt.
- **Prinzip der Zielvorstellung:**
Am Anfang jeden Kapitels wird dem Lernenden das jeweilige Unterrichtsziel mitgeteilt.

7.4.4 Computerunterstützter Unterricht

In Abschnitt 4.1 wurden verschiedene Formen des computerunterstützten Unterrichts erwähnt:

- Drill & Practice
- Tutorielle Unterweisung
- Computergesteuertes Selbststudium
- Simulation, offene Lernumgebungen

Das Lernmodul „Hashing“ lässt sich aufgrund seiner Beschaffenheit in die Kategorie „Tutorielle Unterweisung“ einordnen, denn der Computer übernimmt nicht nur die Aufgabe des Fragenstellens, sondern übernimmt auch Führungsfunktionen und gibt Erklärungen, bietet eine schrittweise Annäherung an das Thema mit vielen Beispielen. Die Lerninhalte werden in Einheiten präsentiert. Außerdem kann der Lernende frei bestimmen, welche Lerneinheit er wann durcharbeiten möchte.

Das computergesteuerte Selbststudium kommt unter gewissen Annahmen auch in Frage, da das Lernmodul ja im universitären Grundstudium eingesetzt werden soll, wo ein menschlicher Tutor (Professor, Dozent, etc.) anwesend ist.

Auf keinen Fall ist das Lernmodul ein „Drill & Practice“-Programm, da nicht die Aufgabenstellungen im Vordergrund stehen.

Ebenfalls ist es keine Simulation bzw. offene Lernumgebung, da keine Simulationen im Rechner stattfinden, die netzwerkartige Zusammenhänge aufzeigen wollen oder in die sich der Benutzer hineinversetzen muss. Die interaktiven Applikationen sind nicht mit Simulationen zu verwechseln, da sie nur zuvor erwähnte, genau definierbare Zusammenhänge verdeutlichen wollen und in ihrer Anwendungsfreiheit stark eingeschränkt sind.

7.4.5 Gestaltungsprinzipien

Auf die in Kapitel 6 erwähnten Gestaltungsprinzipien wurde bei der Erstellung des Lernmoduls „Hashing“ besonders Wert gelegt.

So wurden die in Abschnitt 6.1 behandelten Designprinzipien in besonderem Maße berücksichtigt:

- **Einfachheit:**
Die Sprache ist bewusst einfach gehalten, nicht zu mathematisch-wissenschaftlich, da dies gerade im Grundstudium eine willkommene Abwechslung zu den sonst eher trocken formulierten Lehrbüchern darstellt.
- **Verständlichkeit:**
Aufgrund der Beachtung des ersten Punktes, hofft der Autor, eine gewisse Verständlichkeit erreicht zu haben.
- **Gleich bleibendes Aussehen, gleich bleibende räumliche Anordnung:**
Der Bildschirm wurde in Segmente aufgeteilt: Links und rechts einfarbige Ränder, um das Lernmodul abzugrenzen. Das Lernmodul hat eine breite von 800 Pixeln, das entspricht der Breite der am weitesten verbreiteten Bildschirmauflösung. Oben befindet sich das Hauptfenster mit dem Inhalt und unten die Navigationsleiste.

Die Navigationsleiste bleibt in ihrem Aussehen immer gleich. Auch ihre Position am unteren Bildschirmrand ändert sich nie. Die Designelemente der interaktiven Applikationen sind ebenfalls immer gleich.

Auch die vereinbarten Schriftarten und -farben werden im gesamten Lernmodul eingehalten. So ist der Standardtext weiß, Fachbegriffe gelb, mathematische Teile in einem leicht rötlichen Ton und einer Serifenschrift gehalten.

- **Orientierungshilfen:**

Im linken Teil der Navigationsleiste am unteren Bildschirmrand sieht der Benutzer ständig, in welchem Kapitel und auf welcher Seite er sich gerade befindet. Das ständig sichtbare Navigationssteuerkreuz im rechten Teil zeigt die möglichen Navigationsmöglichkeiten auf.

- **Einfache Fehlermeldungen und Hilfe:**

Es gibt so gut wie keine Fehlermeldungen im Lernmodul „Hashing“, außer bei den interaktiven Applikationen, wo auf Fehleingaben aufmerksam gemacht wird. Das Hilfesystem beschränkt sich auf eine Seite, auf der die Bedienung des Lernmoduls erläutert wird.

Abschließend soll noch auf das Textlayout und Bildschirmdesign des Lernmoduls „Hashing“ eingegangen werden:

Textlayout:

- Bis auf die Ränder am linken und rechten Bildschirmrand, die notwendig sind, damit das Lernmodul die Breite von 800 Pixeln nicht überschreitet, ist der Bildschirm immer gleichmäßig gefüllt. Im oberen und unteren Bereich wurden gleiche Schriftarten, -farben und Zeilenabstände gewählt.
- Es wurde eine sehr leserliche Windows-Schrift gewählt („Tahoma“), die keine Serifen enthält auf jedem Rechner installiert ist.
- Es gibt nur zwei verschiedene Schriftarten im Lernmodul „Hashing“, „Tahoma“ für den Volltext und „Times New Roman“ für alle mathematischen Ausdrücke.

- Der Zeilenabstand wurde bewusst ein wenig vergrößert, um etwas vollere Textseiten nicht zu textlastig erscheinen zu lassen und lesbarer zu machen.
- Es wurde bewusst vom Layoutprinzip „Dunkel auf Hell“ abgewichen, da eine zu lange Betrachtung heller Hintergründe Müdigkeit hervorruft. Der Autor hat eine kontrastreiche „Hell auf Dunkel“ - Darstellung vorgezogen (weiße Schrift auf dunkelblauem Hintergrund).
- Es wurden keine langen Fließtextpassagen verwendet. Mathematische Ausdrücke, Beispiele, Abbildungen und interaktive Applikationen lockern die Darstellung auf.

Bildschirmlayout:

- Ein einheitliches Layoutkonzept wurde verwendet, welches sich im gesamten Lernmodul wieder findet.
- Für die Navigationsleiste ist ein fester Bildschirmbereich reserviert worden (unten).
- Der Verzögerungseffekt ist bei den kurzen Seiten nicht notwendig. Bewusst wurde er jedoch bei den interaktiven Applikationen verwendet, um die Vorgehensweise bestimmter Verfahren nachvollziehbarer zu machen.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Das im Rahmen dieser Arbeit entstandene Lernmodul „Hashing“ bietet sowohl Dozenten als auch Studenten ein leistungsfähiges Werkzeug.

Dozenten können die interaktiven Bestandteile des Lernmoduls dazu nutzen, um in einer Vorlesung gewisse Sachverhalte zu visualisieren. Dies führt nicht nur zu einem weiteren Kommunikationskanal und damit besseren Verständnis für viele Zuhörer, sondern fördert durch die Abwechslung, die ein Einsatz in der Vorlesung bringt, bzw. durch die Anpassbarkeit (etwa mit Schlüsseln, die von den Studenten zugerufen werden) die Motivation der Zuhörer.

Studenten können das Lernmodul in ihrer Prüfungsvorbereitung nutzen.

Bei der Entwicklung des Lernmoduls wurde neben der ergonomischen und einfachen Bedienung darauf Wert gelegt, dass dessen Komponenten wieder verwendbar sind und sehr leicht ein neues Lernmodul mit anderem Inhalt erstellt werden kann, ohne die Implementierung des Lernmoduls von Grund auf verstehen zu müssen. Hinweise dazu finden sich in Anhang B.

8.2 Vorschläge für künftige Erweiterungen

Ausgehend vom aktuellen Stand des Lernmoduls lassen sich viele Erweiterungsmöglichkeiten nennen.

Zunächst einmal gäbe es einige inhaltliche Erweiterungsmöglichkeiten: So könnte man etwas detaillierter auf die genannten Formeln eingehen, die Herleitung dieser betrachten etc. Es müsste nur ein geeignetes Konzept gefunden werden, diese Dinge einzubringen, da sie vielleicht nicht unbedingt zum Kernstoff gehören und nur für manche Benutzer relevant sind. Eine Möglichkeit hierzu wären weitere Exkurse.

Ein anderer inhaltlicher Aspekt wäre eine abschließende historische Zusammenfassung mit einer Literaturliste, wie sie schon jetzt am Ende des Lernmoduls angedeutet wurde. Auch könnte man etwas mehr auf typische Anwendungsmöglichkeiten des Hashing eingehen, welches ein ziemlich interessanter Aspekt wäre, da damit ein Bezug zur Realität hergestellt werden könnte.

Schließlich könnte man sich beim konkreten Lernmodul „Hashing“ überlegen, ob noch weitere Verfahren bzw. neue Erkenntnisse aus dem Bereich Hashing berücksichtigt werden sollen, etwa das Double Hashing, was in [Standish, 1994] ergiebig behandelt wird. Zusätzliche Beispiele finden sich in [Appelrath, 2000].

Ein für Informatiker nicht zu vernachlässigender Gesichtspunkt ist die konkrete Implementierung erlernter Verfahren. Auf die Programmierung mit Java wird in [Saake, 2002] eingegangen.

Auch technisch gibt es einige Erweiterungsmöglichkeiten: Ein erster Vorschlag wäre, das Lernmodul nicht auf Datenträgern (z.B. CD), sondern online im Internet auf einem PHP-fähigen Webserver zur Verfügung zu stellen. Dies würde – im Zusammenhang mit einer Datenbank (die meistens als MySQL Datenbank ebenfalls zur Verfügung steht bzw. leicht realisiert werden kann) sehr viele Möglichkeiten eröffnen¹:

Lesezeichen:

Der Benutzer soll Lesezeichen setzen können, um später ohne zu suchen an derselben Stelle wieder fortfahren zu können.

Notizen:

Der Benutzer soll zu jeder Seite Notizen machen können, die abgespeichert werden und bei jedem Anzeigen der Seite ebenfalls angezeigt werden. Diese sollen nur ihm zugänglich sein.

Aufgaben:

Am Ende jeden Kapitels sollen Fragen gestellt werden, die der Benutzer beantworten muss. Hierbei gibt es viele Möglichkeiten: in den Text eingestreute Fragen,

¹ PHP und MySQL sind frei erhältliche Zusatzmodule für Webserver. Mehr Informationen auf den offiziellen Webseiten <http://www.php.net> bzw. <http://www.mysql.org>

Aufgabenblöcke am Ende jeder Lerneinheit, etc. Auch hinsichtlich der Aufgabentypen gibt es unterschiedliche Formen der Aufgabenstellung (siehe Abschnitt 6.4).

Aufgabenstatistik:

Die Beantwortung der Fragen wird protokolliert und darüber eine Statistik geführt. So kann dem Benutzer der Lernfortschritt angezeigt werden, Kapitel zur Wiederholung empfohlen werden, Schwachstellen aufgezeigt werden etc.

All das erfordert eine Benutzerverwaltung, die jeden Benutzer eindeutig identifiziert und benutzerspezifische Daten abspeichert und verwaltet. Eine solche Benutzerverwaltung eröffnet auch andere Möglichkeiten: Kommunikation zwischen Benutzern, Chat etc. Mit PHP als einfach zu erlernende, mächtige Programmiersprache und MySQL als leistungsfähiges Datenbanksystem sind der Phantasie hierbei keine Grenzen gesetzt.

8.3 Ausblick

Das Lernmodul „Hashing“ bietet eine ergonomische, leicht bedienbare Oberfläche und ist leicht erweiterbar. Da das Themengebiet „Hashing“ nur ein kleines Teilgebiet des Lernstoffes des Grundstudiums Informatik ist, wäre es wünschenswert, wenn noch weitere Lernmodule auf Basis des in dieser Arbeit entstandenen Lernmoduls entstehen würden. Eine ausreichende Dokumentation, die zudem schnell gelesen werden kann, liegt in dieser Arbeit vor. Man könnte durch den Einsatz einiger weniger Mitarbeiter bzw. Studenten (evtl. wissenschaftliche Hilfskräfte) in kürzester Zeit Lernmodule für verschiedene Teilgebiete der Informatik schreiben und so evtl. eine komplette Vorlesung abdecken.

Mit einer universitätsweiten einheitlichen Benutzung desselben Lernmodulsystems wäre dann ein entscheidender Schritt in Richtung effektives e-Learning und innovative Lehr- und Lernmöglichkeiten an einer zukunftsweisenden Bildungseinrichtung getan.

ANHANG

A Beschreibung des Lernmoduls

B Lernmodul anpassen

C Quellcode-Dateien

D Programmlisting

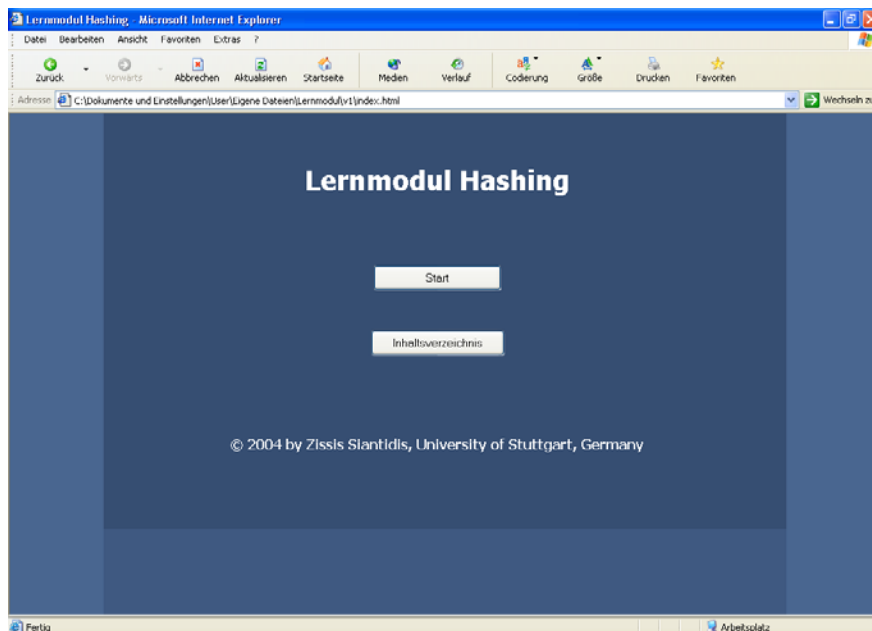
E Die CD-ROM

A BESCHREIBUNG DES LERNMODULS

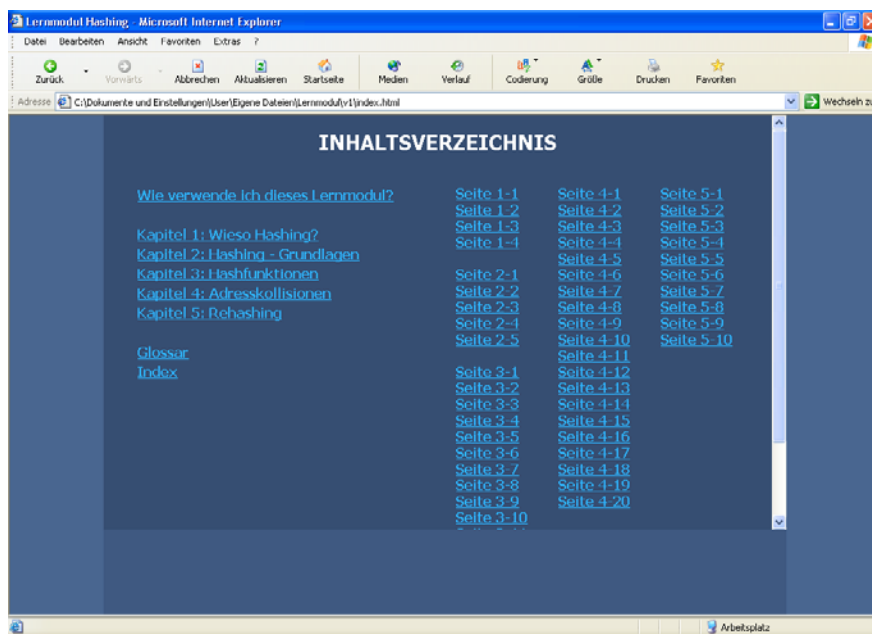
Das Lernmodul „Hashing“ ist ein HTML-basiertes Lernmodul und wird innerhalb eines Browsers aufgerufen. Es ist in Kapitel und Seiten gegliedert. Jedes Kapitel besteht aus einer unterschiedlichen Anzahl von Seiten und behandelt einen Teilaspekt des Themengebiets „Hashing“. Es gibt fünf Kapitel:

1. Wieso Hashing?
2. Hashing - Grundlagen
3. Hashfunktionen
4. Adresskollisionen
5. Rehashing
6. Literatur

Gestartet wird es durch Aufruf der Datei `index.html` im Stammverzeichnis des Lernmoduls. Es erscheint das Titelbild:



Durch einen Klick auf „Start“ erscheint eine Willkommenseite, auf der die Bedienelemente erklärt werden. Für Benutzer, die mit der Bedienung vertraut sind und das Lernmodul nicht zum ersten mal benutzen, gibt es den Button „Inhaltsverzeichnis“, mit dem man sofort zum Inhaltsverzeichnis gelangt, welches so aussieht:



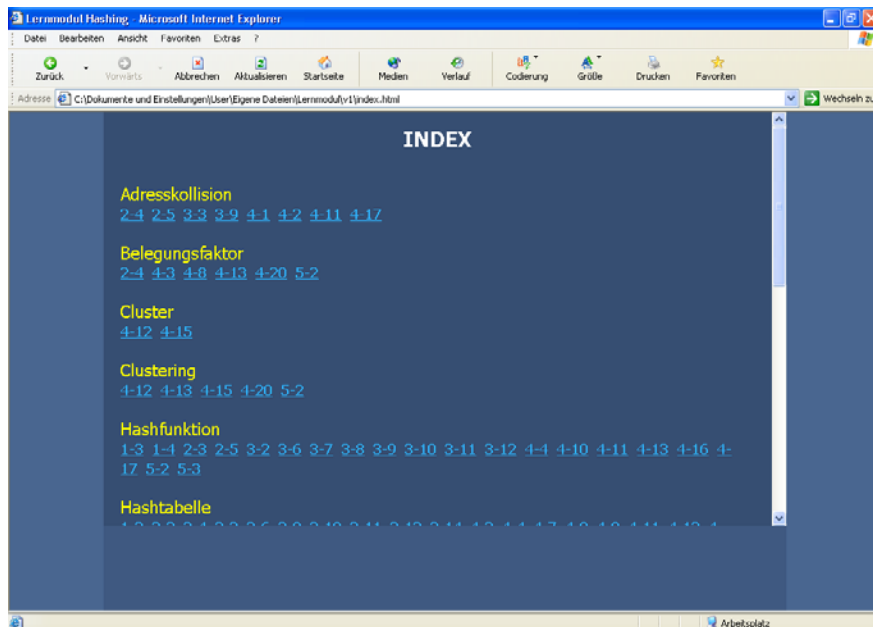
INHALTSVERZEICHNIS			
Wie verwende ich dieses Lernmodul?	Seite 1-1	Seite 4-1	Seite 5-1
	Seite 1-2	Seite 4-2	Seite 5-2
	Seite 1-3	Seite 4-3	Seite 5-3
Kapitel 1: Wieso Hashing?	Seite 1-4	Seite 4-4	Seite 5-4
Kapitel 2: Hashing - Grundlagen		Seite 4-5	Seite 5-5
Kapitel 3: Hashfunktionen	Seite 2-1	Seite 4-6	Seite 5-6
Kapitel 4: Adresskollisionen	Seite 2-2	Seite 4-7	Seite 5-7
Kapitel 5: Rehashing	Seite 2-3	Seite 4-8	Seite 5-8
	Seite 2-4	Seite 4-9	Seite 5-9
	Seite 2-5	Seite 4-10	Seite 5-10
Glossar		Seite 4-11	
Index	Seite 3-1	Seite 4-12	
	Seite 3-2	Seite 4-13	
	Seite 3-3	Seite 4-14	
	Seite 3-4	Seite 4-15	
	Seite 3-5	Seite 4-16	
	Seite 3-6	Seite 4-17	
	Seite 3-7	Seite 4-18	
	Seite 3-8	Seite 4-19	
	Seite 3-9	Seite 4-20	
	Seite 3-10		

Wie man sieht, besteht nicht nur die Möglichkeit, einzelne Kapitel auszuwählen, sondern man kann auch direkt zu jeder einzelnen Seite des Lernmoduls springen. Es gibt ebenfalls ein Glossar mit Erklärungen zu den wichtigsten Fachbegriffen sowie einen Index mit Verweisen zu deren Vorkommen im gesamten Lernmodul.

Das Glossar bietet kurze Erklärungen der Fachbegriffe mit Querbezügen zu anderen Glossareinträgen:

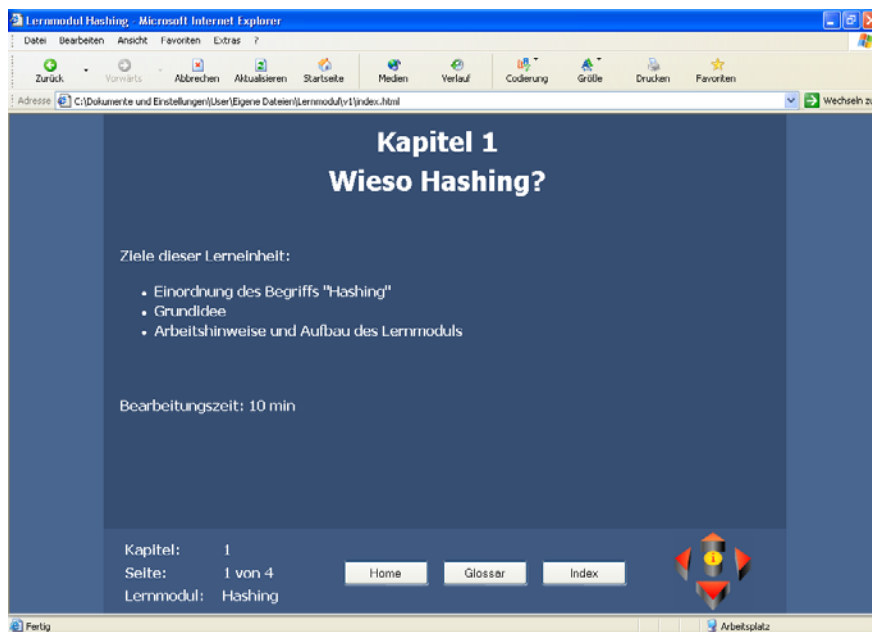


Der Index bietet Hyperlinks zu den Seiten, wo der jeweilige Fachbegriff vorkommt. Die Hyperlinks haben die Form „Kapitelnr. - Seitennr.“:



Die Fachbegriffe sind im Volltext des Lernmoduls farblich hervorgehoben und anklickbar. Bei Anklicken erscheint ein kleines Fenster mit der Erklärung des Fachbegriffes. Bezüglich farblicher Hervorhebung und damit schlechterer Lesbarkeit des Fließtextes wurde ein Kompromiss gewählt: Nur das erste Vorkommen eines Fachbegriffes in einem Kapitel wird farblich markiert, alle restlichen Vorkommen erscheinen in der normalen Textfarbe. Hierbei wird davon ausgegangen, dass zumindest innerhalb eines Kapitels linear vorgegangen wird.

Damit sind alle „Non-Content“-Seiten des Lernmoduls erwähnt. Alle anderen Seiten gehören zu Kapiteln und vermitteln Lernstoff. Jedes Kapitel beginnt mit einer Titelseite, in der die Zielsetzung des Kapitels und die ungefähre Bearbeitungszeit genannt werden:



Der untere Bereich des Fensters stellt die Navigationsleiste dar und bleibt während der ganzen Durcharbeitung unverändert. Sie gibt sowohl Auskunft über den aktuellen Standort des Lernenden innerhalb des Lernmoduls (links: Kapitelnummer, aktuelle Seitennummer, Gesamtseitenzahl in diesem Kapitel, Lernmodulname) als auch über die aktuellen Navigationsmöglichkeiten: Mit Hilfe der drei Buttons in der Mitte gelangt man auf die Startseite des Moduls, ruft das Glossar bzw. den Index auf, und das Navigationssteuerkreuz ganz rechts ermöglicht das Vor-/Zurückblättern der Seiten und

der Kapitel. Mit dem Knopf „i“ in der Mitte des Steuerkreuzes wird das Inhaltsverzeichnis aufgerufen.

An manchen Stellen im Lernmodul sind interaktive Applikationen eingebaut, anhand derer gewisse Sachverhalte experimentell herausgefunden bzw. visuell vermittelt werden sollen. Im konkreten Lernmodul sind es Applikationen zum Divisions-Rest-Verfahren bei Hashfunktionen, linearen Sondieren mit alphanumerischen Schlüsselwörtern, linearen Sondieren mit Zufallszahlen, quadratischen Sondieren mit alphanumerischen Schlüsselwörtern und quadratischen Sondieren mit Zufallszahlen. Die Bedienung einer solchen Applikation sei exemplarisch an einem Beispiel erklärt (die int. Applikationen sind genauer in Abschnitt 7.3 beschrieben):

Schlüsselmenge S	h(s)	Hashfunktion h	Hashtabelle T
Aphrodite	3	Addiere: <input type="checkbox"/> 1. Buchstaben <input checked="" type="checkbox"/> 2. Buchstaben <input checked="" type="checkbox"/> 3. Buchstaben <input checked="" type="checkbox"/> 4. Buchstaben <input type="checkbox"/> 5. Buchstaben <input type="checkbox"/> 6. Buchstaben <input checked="" type="checkbox"/> 7. Buchstaben <input type="checkbox"/> 8. Buchstaben <input type="checkbox"/> 9. Buchstaben $m = 16$ <input type="button" value="Berechnen"/>	0: Hephaistos
Apollo	11		1: Athene
Ares	10		2:
Artemis	14		3: Aphrodite
Athene	1		4: Hermes
Demeter	9		5: Dionysos
Dionysos	5		6: Poseidon
Hephaistos	0		7:
Hera	8		8: Hera
Hermes	4		9: Demeter
Poseidon	6		10: Ares
Zeus	13		11: Apollo
			12: Zeus
			14: Artemis
			15:

gr. Götter

Lösung 1 Lösung 2 Lösung 3 Lösung 4

Kapitel: 3
Seite: 10 von 14
Lernmodul: Hashing

Home Glossar Index

Arbeitsplatz

Gezeigt ist hier die Applikation für das Divisions-Rest-Verfahren bei Hashfunktionen. Alle Elemente sind Standard-Formularelemente, wie sie im WWW bekannt sind. Auf der linken Seite kann der Benutzer eine Schlüsselmenge eingeben oder die im Lernmodul standardmäßige Menge der Namen der griechischen Götter auswählen. In der Mitte kann die Hashfunktion spezifiziert werden, indem die Buchstaben ausgewählt werden, deren ASCII-Code addiert werden soll, sowie die Hashtabellengröße m

angegeben werden. Sind diese Schritte erfolgt, so werden mit einem Klick auf den Button „Berechnen“ die Funktionswerte der einzelnen Schlüssel berechnet und die sich ergebende Hashtabelle dargestellt. Durch Manipulation der Parameter ergeben sich verschiedene Hashtabellen, mit oder ohne Adresskollisionen.

Isolierung einzelner interaktiver Applikationen

Sollen einzelne interaktive Applikationen oder andere Inhaltsseiten isoliert werden (etwa zur Vorführung in einer Vorlesung) so lässt sich das sehr einfach realisieren.

Lokalisieren Sie die entsprechende HTML-Datei (z.B. `page0313.html` für die 13. Seite des dritten Kapitels) und speichern Sie sie extern irgendwo ab. Durch einen Aufruf dieser Datei öffnet sich ein Browserfenster mit dem Inhalt dieser Datei.

Es müssen zusätzlich auch zwei weitere Dateien isoliert werden, die sich im selben Verzeichnis befinden müssen. Achten Sie also darauf, dass Sie auch die Dateien `lernmodul.js` und `lernmodul.css` im selben Verzeichnis speichern, damit die gewünschte Funktionalität sowie das definierte Layout erhalten bleibt.

B LERNMODUL ANPASSEN

Das Lernmodul „Hashing“ wurde von Anfang an so konzipiert, dass es leicht anpassbar und erweiterbar ist und mit neuem Inhalt versehen werden kann. Auf diese Weise soll es möglich sein, Lernmodule für verschiedene Lerninhalte anzubieten, die alle ähnlich aussehen, ähnlich zu bedienen sind und nach außen hin „aus einem Guss“ erscheinen, sinnvoll etwa für verschiedene Kapitel einer Vorlesungsveranstaltung. So könnte es im Grundstudium neben einem Lernmodul „Hashing“ etwa ein Lernmodul „binäre Bäume“ oder ein Lernmodul „Sortieralgorithmen“ geben, welche alle in ähnlicher Art und Weise sowohl von den Studenten zur Prüfungsvorbereitung als auch von Professoren in der Vorlesung genutzt werden können.

Bevor wir mit genauen Durchführungsschritten beginnen, sollte die Funktionsweise des Lernmoduls erläutert werden. Das Lernmodul „Hashing“ ist ein auf HTML basierendes Lernmodul, das Teile in Javascript enthält. Es kann von jedem Javascript-fähigen Browser aufgerufen werden.

Im Auslieferungszustand besteht es aus einer Datei `index.html` und zwei Unterverzeichnissen `html` und `images`:

`index.html`:

Mit einem Aufruf dieser Datei startet das Lernmodul. Von hier werden alle weiteren benötigten Dateien aufgerufen.

`html`:

In diesem Verzeichnis befinden sich alle HTML-Dateien sowie die Dateien `lernmodul.js` und `lernmodul.css`, die grundlegende Navigationsfunktionalität bzw. Layoutspezifische Aspekte beinhalten.

`images`:

In diesem Unterverzeichnis befinden sich alle Bilddateien, sowohl Abbildungen als auch grafische Symbole, die vom Lernmodul benötigt werden.

Die Realisierung einer Anpassung bzw. Neuerstellung eines Lernmoduls erfordert grundsätzlich Arbeit an drei Orten: Zum einen an der Datei `lernmodul.js`, welche allgemeine Einstellungen, Kapitelzahl, Seitenzahl, Fachvokabular etc. umfasst, zum anderen an den Dateien `pageXXYY.html`, welche den eigentlichen Inhalt des Lernmoduls darstellen (XX bedeutet hier die zweistellige Kapitelnummer, YY die zweistellige Seitenzahl innerhalb des Kapitels, immer beginnend mit „01“). Als drittes sollte das Inhaltsverzeichnis in der Datei `inhalt.html` geschrieben werden. Zusätzlich kann auf Wunsch noch die Datei `lernmodul.css` den eigenen optischen Vorstellungen angepasst werden.

Im Folgenden wollen wir nun auf die Anpassungsmöglichkeiten der einzelnen Dateien eingehen. Hierbei werden wir die Dateien `lernmodul.js`, `lernmodul.css` und `pageXXYY.html` behandeln.

B.1 Die Datei `lernmodul.js`

Die Datei `lernmodul.js` ist eine Javascript Datei und stellt die Funktionen bereit, die vom Lernmodul u.a. zur Navigation benötigt werden. Sie besteht aus zwei Abschnitten:

- der Definitionsteil, in dem Änderungen an Variablen vorgenommen werden können und sollen und
- der Implementationsteil, der Funktionen enthält, die nicht geändert werden sollten.

Die zwei Abschnitte sind durch Kommentare eindeutig markiert. Nachfolgend soll auf die Änderungsmöglichkeiten der Variablen bzw. auf die zur Verfügung gestellten Funktionen eingegangen werden.

Zunächst finden sich die Variablen `lernmodulname`, `maxchapters` und das Array `maxpages`:

- `lernmodulname`: speichert den Namen des Lernmoduls. Wird unten in der Navigationsleiste ständig angezeigt. Sinnvoll zur Unterscheidung zwischen verschiedenen Lernmodulen
- `maxchapters`: speichert die maximale Anzahl der Kapitel. Notwendig zum Blättern zwischen Kapiteln
- `maxpages[i]`: speichert die maximale Seitenzahl des Kapitels mit der Nummer `i`. Notwendig zum Blättern innerhalb eines Kapitels.

Es folgen die Definitionen der Fachbegriffe. Das sind Wörter, die im laufenden Text auf Wunsch andersfarbig und anklickbar dargestellt werden. Ihre Definition soll anhand eines Beispiels erläutert werden.

Beispiel:

```
definitions['Adresskollision'] = new Definition();
definitions['Adresskollision'].locations = "[2,4] [2,5]
[3,3] [3,9] [4,1] ";
definitions['Adresskollision'].description = "Eine
Adresskollision tritt auf, wenn zwei verschiedene
[Schlüssel] auf dieselbe Hashposition abgebildet werden,
wenn also der Funktionswert der [Hashfunktion] für beide
Schlüssel gleich ist. Solche Schlüssel nennt man
auch [Synonyme]. Adresskollisionen müssen
aufgelöst und entsprechend behandelt werden.";
```

Zuerst wird die Zeile `definitions['Adresskollision'] = new Definition();` benötigt. Hier wird ein neuer Fachbegriff angelegt. Danach müssen die Vorkommen des Fachbegriffs für den Index angegeben werden. Dies geschieht durch eckig geklammerte Zahlenpaare nach der Form „[Kapitelnummer, Seitennummer]“ in der Variablen `locations`. Es können beliebig viele Zahlenpaare angegeben werden. Diese erscheinen dann im Index des Lernmoduls als Links auf die jeweilige Seite. Zuletzt muss natürlich noch die Erklärung des Fachbegriffs in der

Variablen `description` angegeben werden. Auch hier werden eckige Klammern benutzt, um Links zu erzeugen, diesmal auf andere Fachbegriffe. Auf diese Weise können Querbezüge zwischen verschiedenen Fachbegriffen realisiert werden. Wichtiger Hinweis: Während in der Erklärung HTML-Sonderzeichen der Form `ü` etc. erlaubt sind, dürfen sie im Array-Index NICHT verwendet werden. Es sollte also `definitions[,Verkettung der Überläufer']` heißen, und nicht `definitions[,Verkettung der Überläufer']`.

Nachdem die anzupassenden Variablen erläutert wurden, folgt eine Beschreibung der wichtigsten Funktionen, die in dieser Datei ebenfalls zur Verfügung gestellt werden. Diese können und sollen in den HTML-Dateien, die den eigentlichen Inhalt des Lernmoduls darstellen, verwendet werden.

```
function updateNavpanel()
```

Diese Funktion muss am Anfang jeder HTML-Datei aufgerufen werden. Sie liest aus dem Dateinamen die Kapitelnummer und Seitennummer und gibt diese unten in der Navigationsleiste aus.

```
function loadpage(chapter, page)
```

Zeigt eine durch `chapter` und `page` bestimmte Seite im Hauptfenster an. Mit Hilfe dieser Funktion lassen sich Links zu anderen Seiten generieren.

```
function insertkeyword(keyword, text)
```

Fügt einen andersfarbigen, anklickbaren Fachbegriff hinzu. Bei Anklicken erscheint ein kleines Fenster mit der Beschreibung. Die Farbe des Fachbegriffes kann in der Datei `lernmodul.css` spezifiziert werden. Benötigt mindestens den Parameter `keyword` (den Fachbegriff, genau so wie er im Array `definitions` angegeben ist. Optional kann ein zweiter Parameter angegeben, wenn der im Fließtext erscheinende Text von Fachbegriff abweicht (z.B. „... es werden Adresskollisionen verursacht ...“ , bei Klick soll die Beschreibung von „Adresskollision“ aufgerufen werden).

```
function insertexkurs(filename, linktext, width)
```

Fügt ein Symbol zum Besuchen eines Exkurses hinzu. Bei Anklicken öffnet sich ein kleines Fenster mit einer HTML-Seite mit Inhalt. Ein Exkurs ist eine Seite mit weiterführendem Inhalt oder Beleuchtung eines Aspektes, der mit dem eigentlichen Lerninhalt des Lernmoduls nichts zu tun hat.

Erwartet drei Parameter: `filename` (die aufzurufende HTML-Datei), `linktext` (die kurze Beschreibung des Exkurses, die als Link erscheint) sowie `width` (die Breite des Symbols). Wird rechtsbündig dargestellt und sollte vor dem Text, der links davon erscheinen soll, aufgerufen werden.

```
function insertpages(chapter)
```

Diese Funktion soll im Inhaltsverzeichnis aufgerufen werden, um eine Liste mit Links zu allen Seiten des Lernmoduls zu realisieren, damit man schneller eine bestimmte, bekannte Seite aufrufen kann. Listet alle Seiten des Kapitels `chapter` (integer-Wert) als Links untereinander auf.

B.2 Die Datei `lernmodul.css`

Die Datei `lernmodul.css` ist eine im WWW übliche Cascading-Style-Sheet-Datei, die Layoutinformationen enthält. In ihr sind Farben, Schriftarten, Schriftgrößen etc. definiert. Durch Ändern der Werte kann das Aussehen den eigenen Vorstellungen angepasst werden. Nachfolgend eine Erklärung der benutzten Klassen:

```
body.background
```

Hier wird die Hintergrundfarbe für den Bereich definiert, der das Lernmodul links und rechts umschließt, falls das Anzeigefenster breiter als 800 Pixel ist.

```
body.borders
```

Hier werden die Ränder sowie die Hintergrundfarbe für die Navigationsleiste festgelegt.

`body.main`

Hier werden die Einstellungen für das Hauptfenster in der Mitte vorgenommen, wo der eigentliche Inhalt erscheint.

`table.main`

Hier wird das Layout für die Navigationsleiste festgelegt.

`table.exkurs`

`th.exkurs`

Hier wird das Aussehen der „Exkurs“-Schaltflächen definiert, welche auf einen Exkurs verweisen.

`.inhalt`

Hier werden Schrifteigenschaften für das Inhaltsverzeichnis definiert.

`.definition`

Das Aussehen der andersfarbigen, anklickbaren Fachbegriffe im Text wird hier definiert.

`.defKeyword`

`.defDefinition`

Wurde ein Fachbegriff angeklickt, so erscheint ein kleines Fenster mit dem Fachbegriff und darunter seiner Erklärung. Diese beiden Klassen enthalten das Layout für diese beiden Elemente. Dasselbe Layout wird auch für das Glossar und für den Index benutzt.

Es folgen einige für dieses Lernmodul spezifische Definitionen:

`td.hashtable`

Hier wird das Aussehen der Hashtabellen in den interaktiven Applikationen festgelegt.

`.math`

Mathematische Ausdrücke im Text sollen der besseren Lesbarkeit halber in einer anderen Farbe dargestellt werden. Solche Ausdrücke werden daher in ``-Tags

eingeschlossen und der Klasse `.math` zugeordnet, welche die unterschiedliche Darstellung realisiert.

Durch eine Anpassung dieser Datei wird eine lernmodulweite Änderung des Elementes bewirkt, wenn sichergestellt wird, dass bei der Erstellung der HTML-Seiten nur diese Klassen zur Layoutdefinition verwendet wurden. Auf die Erstellung der HTML-Inhaltsseiten wird im nachfolgenden Abschnitt eingegangen.

B.3 Die HTML-Dateien `pageXXYY.html`

Die HTML-Dateien, die mit dem Präfix `page` beginnen, stellen den eigentlichen Inhalt des Lernmoduls dar. Jede solche Datei steht für eine Seite in einem Kapitel. Die genaue Position der Seite im Lernmodul wird im Dateinamen auf folgende Weise verewigt:

XX = zweistellige Kapitelnummer

YY = zweistellige Seitennummer innerhalb des Kapitels, beginnend bei „01“

Beispiel:

`page0309.html` = neunte Seite des dritten Kapitels

Hierbei sollte erwähnt werden, dass die einleitende Willkommenseite, in der die Bedienelemente des Lernmoduls erläutert werden, als erste Seite des „nullten“ Kapitels zählt (`page0001.html`). Sollte man der Beschreibung weitere Seiten hinzufügen wollen, so würde man analog fortfahren: `page0002.html`, `page0003.html`, etc. Erst mit der Seite `page0101.html` geht das Lernmodul richtig los.

Der Aufbau einer solchen HTML-Datei sieht folgendermaßen aus:

```
<html>
<head>
  <title>Lernmodul Hashing</title>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
  <link rel="stylesheet" href="lernmodul.css">
</head>
```

```
<body class="main">
<script language="javascript" src="lernmodul.js"></script>
<script language="javascript">updateNavpanel()</script>

<h2>Wieso Hashing?</h2>
Hashing ist eine Methode, um große Datenmengen abzuspeichern und
zu verwalten.

. . .

</body>
</html>
```

Zu beachten sind folgende Aspekte:

- Durch die Zeile `<link rel="stylesheet" href="lernmodul.css">` im Header wird die weiter oben beschriebene Datei `lernmodul.css` eingebunden.
- Dem „body“-tag sollte die Klasse „main“ zugewiesen werden (`<body class="main">`), damit sichergestellt wird, dass die in der Datei `lernmodul.css` vorgegebenen Farben und Schriftarten verwendet werden.
- Die ersten beiden Zeilen nach dem “body”-tag sollten lauten:

```
<script language="javascript" src="lernmodul.js"> </script>
<script language="javascript"> updateNavpanel() </script>
```

Damit wird sichergestellt, dass alle in der Datei `lernmodul.js` definierten Javascript-Funktionen verfügbar sind und dass die Navigationsleiste upgedatet wird und die aktuelle Seitenzahl korrekt anzeigt.
- Ansonsten kann ganz normaler HTML-Code benutzt werden. Auch zusätzlich definierte StyleSheet-Definitionen können verwendet werden, solange sie in der Datei `lernmodul.css` definiert wurden.

Diese Punkte werden sehr schnell klar, wenn man sich einige `pageXXYY.html` Dateien anschaut. Zur Erstellung neuen Inhalts kopiert man sich am besten eine solche Datei und ändert nur den Inhalt innerhalb der `body`-Tags.

Hiermit sind alle wichtigen Änderungsmöglichkeiten des Lernmoduls beschrieben. Abschließend sei noch erwähnt, welche Dateien im Verzeichnis `html` wofür stehen:

<code>inhalt.html</code>	Enthält das Inhaltsverzeichnis, sollte auch angepasst werden.
<code>glossar.html</code>	Enthält den Code für das Glossar, welches alle Fachbegriffe alphabetisch sortiert und untereinander mitsamt Erklärung auflistet. Kann einfach übernommen werden.
<code>index.html</code>	Enthält den Code für den Index, welcher alle Fachbegriffe alphabetisch sortiert und untereinander mitsamt Links zu ihren Vorkommen auflistet. Kann ebenfalls einfach übernommen werden.

C QUELLCODE-DATEIEN

Die Quellcode-Dateien befinden sich allesamt im Unterverzeichnis `html`. Nachfolgend eine Übersicht über alle Dateien und ihre Aufgaben.

<code>lernmodul.js</code>	Zentrale Programmcode-Datei. Hier sind alle Funktionen zur Navigation, Fachbegriffsverwaltung etc. definiert. Genaue Beschreibung siehe Anhang B.1.
<code>lernmodul.css</code>	Cascading-Style-Sheet-Datei mit allen Layout-Informationen (Schriftarten, Schriftgrößen, Farben, etc.). Genaue Beschreibung siehe Anhang B.2.
<code>pageXXYY.html</code>	HTML-Seiten mit dem eigentlichen Inhalt des Lernmoduls. XX steht dabei für die Kapitelnummer, YY für die Seitennummer innerhalb des Kapitels. Genaue Beschreibung siehe Anhang B.3.
<code>background.html</code> <code>bottom.html</code>	Leere HTML-Dateien zum Ausfüllen der Bildschirmränder mit einer bestimmten Hintergrundfarbe.
<code>definition.html</code>	HTML-Vorlage für das Fenster mit der Erklärung eines Fachbegriffs.
<code>glossar.html</code> <code>index.html</code>	Glossar und Index. Verwenden die in <code>lernmodul.js</code> definierten Fachbegriffe.
<code>inhalt.html</code>	Das Inhaltsverzeichnis. Sollte bei einer Anpassung berücksichtigt werden
<code>home.html</code>	Die Startseite mit den zwei Buttons „Start“ und „Inhaltsverzeichnis“.

D PROGRAMMLISTING

Beispielhaft sei hier die komplette Datei `lernmodul.js` abgedruckt, welche das programmtechnische Grundgerüst darstellt, damit der Leser sich klar machen kann, welcher Programmieraufwand hinter dem Lernmodul „Hashing“ steckt.

```
// lernmodul.js
// programmed 2003, 2004 by Zissis Siantidis
// for his master thesis at the University of Stuttgart, Germany

/*****
ADJUST THE FOLLOWING VARIABLES ACCORDING TO YOUR HTML FILES
FOR FURTHER INFORMATION ABOUT CONFIGURATION CHECK CHAPTER "ANHANG B"
IN THE MASTER THESIS
*****/

// Name of learn module, is displayed in the navpanel at the bottom
lernmodulname = "Hashing"

// max number of chapters
maxchapters = 7;

// this array contains the max pages per chapter. necessary for
navigation.
var maxpages = new Array();
maxpages[0] = 1;
maxpages[1] = 4;
maxpages[2] = 5;
maxpages[3] = 14;
maxpages[4] = 20;
maxpages[5] = 9;
maxpages[6] = 1;

// Class definition for definitions array. DO NOT CHANGE!!
function Definition(description, locations) {
    this.description = description;
    this.locations = locations;
```

```

}

// this array contains the definitions for the desired keywords in the
// text that can be looked up.
// no alphabetical sorting needed here. it is sorted later by the
// program.
// Please enter your keywords here
var definitions = new Array();

definitions['Adresskollision'] = new Definition();
definitions['Adresskollision'].locations = "[2,4] [2,5] [3,3] [3,9]
[4,1] [4,2] [4,11] [4,17]";
definitions['Adresskollision'].description = "Eine Adresskollision
tritt auf, wenn zwei verschiedene \
    [Schlüssel] auf dieselbe Hashposition abgebildet werden, wenn
also der Funktionswert der \
    [Hashfunktion] für beide Schl&uuml;ssel gleich ist. Solche
Schl&uuml;ssel nennt man auch [Synonyme]. \
    Adresskollisionen m&uuml;ssen aufgel&ouml;st und entsprechend
behandelt werden.";

definitions['Belegungsfaktor'] = new Definition();
definitions['Belegungsfaktor'].locations = "[2,4] [4,3] [4,8] [4,13]
[4,20] [5,2]";
definitions['Belegungsfaktor'].description = "Der Belegungsfaktor
<span class='math'>&alpha;</span> ist der Quotient aus den belegten \
    Hashtabellenpl&auml;tzen zu den insgesamt existierenden
Tabellenpl&auml;tzen, und ist also eine Zahl, \
    die angibt, wie voll die [Hashtabelle] ist.<br><br> \
    <span class='math'>&alpha;</span> = belegte Pl&auml;tze / insg.
Pl&auml;tze</span> , also <span class='math'>0 &le; &alpha; &le;
1</span>";

definitions['Synonyme'] = new Definition();
definitions['Synonyme'].locations = "[2,4]";
definitions['Synonyme'].description = "Synonyme sind zwei
verschiedene Schl&uuml;ssel, deren Hashfunktionswert \
    identisch ist. Sie verursachen eine [Adresskollision].";

definitions['Hashtabelle'] = new Definition();

```

```
definitions['Hashtabelle'].locations = "[1,3] [2,3] [2,4] [3,2] [3,6]
[3,9] [3,10] [3,11] [3,12] [3,14] [4,3] [4,4] \
    [4,7] [4,8] [4,9] [4,11] [4,13] [4,14] [4,15] [4,17] [4,18]
[4,19] [4,20] [5,2] [5,3]";
definitions['Hashtabelle'].description = "Ein Speicherort f&uuml;r
die zu speichernden Elemente. \
    Meist eine Array-&uuml;hnliche Datenstruktur. Zusammen mit
einer \
    [Hashfunktion] zentraler Bestandteil jedes Hashverfahrens. ";

definitions['Hashfunktion'] = new Definition();
definitions['Hashfunktion'].locations = "[1,3] [1,4] [2,3] [2,5] [3,2]
[3,6] [3,7] [3,8] [3,9] [3,10] [3,11] [3,12] \
    [4,4] [4,10] [4,11] [4,13] [4,16] [4,17] [5,2] [5,3]";
definitions['Hashfunktion'].description = "Eine Funktion, die die
Menge der [Schl&uuml;ssel] auf die Menge der \
    Hashtabellenpositionen abbildet. Sie weist jedem
Schl&uuml;ssel einen Platz in der [Hashtabelle] zu. \
    Zusammen mit einer Hashtabelle zentraler Bestandteil jedes
Hashverfahrens.";

definitions['Verkettung der Überläufer'] = new Definition();
definitions['Verkettung der Überläufer'].locations = "[4,2] [4,3]
[4,4] [4,8]";
definitions['Verkettung der Überläufer'].description = "Verfahren zur
Aufl&ouml;sung von [Adresskollision]en, bei dem \
    [Synonyme] in verketteten Listen gespeichert werden.";

definitions['Open Hashing'] = new Definition();
definitions['Open Hashing'].locations = "[4,2] [4,8]";
definitions['Open Hashing'].description = "Verfahren zur
Aufl&ouml;sung von [Adresskollision]en, bei dem \
    innerhalb der [Hashtabelle] nach alternativen
Speicherpositionen f&uuml;r [Synonyme] gesucht wird.";

definitions['Sondieren'] = new Definition();
definitions['Sondieren'].locations = "[4,8] [4,9] [4,11] [4,13] [4,14]
[4,15] [4,17] [4,18] [4,19]";
```

```
definitions['Sondieren'].description = "Sondieren bezeichnet die
Methode, nach der beim [Open Hashing] nach einer \
    alternativen Speicherposition f&uuml;r [Synonyme] gesucht
wird. Bekannteste Vertreter sind: \
    [lineares Sondieren] und [quadratisches Sondieren].";

definitions['Sondierungsfolge'] = new Definition();
definitions['Sondierungsfolge'].locations = "[4,8] [4,11] [4,17]";
definitions['Sondierungsfolge'].description = "Die Reihenfolge, nach
der beim [Sondieren] alternative Positionen in der Hashtabelle gesucht
werden. Bekannteste Vertreter sind: \
    [lineares Sondieren] und [quadratisches Sondieren].";

definitions['lineares Sondieren'] = new Definition();
definitions['lineares Sondieren'].locations = "[4,8] [4,9] [4,11]
[4,14] [4,19]";
definitions['lineares Sondieren'].description = "Sondierverfahren,
nach dem zum Auffinden einer alternativen \
    Speicherposition in der [Hashtabelle] linear mit einer
konstanten Schrittweite immer weitergesprungen wird, bis \
    eine leere Position zum Speichern gefunden wird. Verursacht
leider [Clustering].";

definitions['quadratisches Sondieren'] = new Definition();
definitions['quadratisches Sondieren'].locations = "[4,8] [4,15]
[4,17] [4,19]";
definitions['quadratisches Sondieren'].description =
"Sondierverfahren, nach dem zum Auffinden einer alternativen \
    Speicherposition in der [Hashtabelle] nicht linear, sondern
mit sich &auuml;ndernder Schrittweite immer \
    weitergesprungen wird, bis eine leere Position zum Speichern
gefunden wird. F&uuml;r die Schrittweite wird dabei \
    die Folge der Quadratzahlen eingesetzt. Versucht, [Clustering]
zu vermeiden.";

definitions['Rehashing'] = new Definition();
definitions['Rehashing'].locations = "[4,20] [5,1] [5,2] [5,4]";
definitions['Rehashing'].description = "Verfahren zur Umorganisation
einer [Hashtabelle], nachdem sie vergr&ouml;ssert wurde.";
```

```
definitions['Schlüssel'] = new Definition();
definitions['Schlüssel'].locations = "[1,2] [1,3] [2,2] [2,3] [2,4]
[3,2] [3,3] [3,4] [3,5] [3,6] [4,4] [4,5] \
    [4,7] [4,8] [5,3] [5,4] [5,5] [5,6]";
definitions['Schlüssel'].description = "Ein Schlüssel ist
Bestandteil eines abzuspeichernden Datensatzes. Anhand \
    des Schlüssels kann ein Datensatz eindeutig identifiziert
werden, d.h. es gibt keine zwei gleichen Schlüssel. \
    Hashingverfahren benutzen Schlüssel um Speicherpositionen
für Datensätze zu ermitteln.";

definitions['Clustering'] = new Definition();
definitions['Clustering'].locations = "[4,12] [4,13] [4,15] [4,20]
[5,2]";
definitions['Clustering'].description = "Ein Phänomen, nach dem
sich viele Schlüssel eng \
    beieinander "tummeln". Es entstehen mehrere Bereiche in der
[Hashtabelle], in denen keine leeren Positionen \
    zu finden sind: [Cluster]. Zu Beobachten vorwiegend bei:
[lineares Sondieren].";

definitions['Cluster'] = new Definition();
definitions['Cluster'].locations = "[4,12] [4,15]";
definitions['Cluster'].description = "Ein Cluster ist ein Bereich in
einer [Hashtabelle], der bezüglich \
der Verschiebung beim Sondieren, also bezüglich der
[Sondierungsfolge], vollständig ausgefüllt ist. \
Diese Bereiche sind nicht immer erkennbar, da auch komplexere
Sondierfolgen möglich sind, etwa [quadratisches Sondieren] \
oder eine Schrittweite größer 1. Gut sichtbar sind die
Bereiche eigentlich nur, wenn die Schrittweite beim Sondieren \
konstant 1 ist; man erkennt sie dann als zusammenhängende,
vollständig ausgefüllte Bereiche in der Hashtabelle.";
```

```
/*
DO NOT CHANGE ANYTHING IN THE FOLLOWING CODE
*/

// the following variables contain information about the current
// position in the module
var parameter = document.location.search.substr(1);
var chapterstring = parameter.substr(0,2);
var pagestring = parameter.substr(2,2);
var chapter = parseInt(chapterstring,10);
var page = parseInt(pagestring,10);

// sends chapter and page number to the navigation panel
function updateNavpanel() {
    var pos=location.pathname;
    pos=pos.substr(pos.length-9,4);
    if (top.bottomframe)
top.bottomframe.location.href="navpanel.html?" +pos;
}

// writes the chapter number into the document
function writechapter() {
    document.write(chapter);
}

// writes the page number into the document
function writepage() {
    document.write(page+" von "+maxpages[chapter]);
}

// loads the page into the mainframe
function loadpage(chapter,page) {
    chapterstring=String(chapter);
    if (chapter<10) chapterstring="0"+chapterstring;
    pagestring=String(page);
    if (page<10) pagestring="0"+pagestring;
    parent.mainframe.location="page"+chapterstring+pagestring+".html";
}
}
```

```
// jumps to the beginning of the current chapter
function kapitelanfang() {
    page=1;
    loadpage(chapter,page);
}

// jumps to the end of the current chapter
function kapitelende() {
    page=maxpages[chapter];
    loadpage(chapter,page);
}

// jumps to the next page
function nextpage() {
    page=page+1;
    if (page>maxpages[chapter]) {
        chapter=chapter+1;
        if (chapter>=maxchapters) chapter=1;
        page=1;
    }
    loadpage(chapter,page);
}

// jumps to the previous page
function prevpage() {
    page=page-1;
    if (page==0) {
        chapter=chapter-1;
        if (chapter<0) chapter=0;
        page=maxpages[chapter];
    }
    loadpage(chapter,page);
}

// jumps to the next chapter
function nextchapter() {
    chapter=chapter+1;
    if (chapter>=maxchapters) {
        chapter=1;
    }
}
```

```
    loadpage (chapter,1);
}

// jumps to the previous chapter
function prevchapter() {
    chapter = chapter-1;
    if (chapter == -1) {
        chapter = maxchapters-1;
    }
    loadpage (chapter,1);
}

// opens a small window and shows the definition for the word
'keyword', which is looked up
// in the array definitions[], which is defined in the beginning of
this file
function showdefinition(keyword) {
    keyword = encodeURI(keyword);
    window1 =
window.open ("definition.html?" + keyword, "window1", "width=500,height=400
, top=50, left=50, resizable=yes");
    window1.focus();
}

// inserts a colored keyword in the text
// first parameter keyword is the keyword in the keyword list
// second parameter text is optional: the word that appears in the
text (could be different from keyword)
function insertkeyword(keyword, text) {
    if (! text) text=keyword;
    document.write("<span class='definition'
onClick=\"showdefinition('"+keyword+"')\" \
                title='Klicken Sie, um eine Beschreibung zu
erhalten.'>" + text + "</span>");
}

function showglossar() {
    window2 =
window.open ("glossar.html", "window2", "width=700,height=600, left=70, top
=70, resizable=yes, scrollbars=yes");
}
```

```
        window2.focus();
    }

function openexkurs(filename) {
    window4 =
window.open(filename,"window4","width=840,height=600,left=10,top=50,re
sizable=yes,scrollbars=yes");
    window4.focus();
}

function insertexkurs(filename, linktext, width) {
    document.write("<table class='exkurs' width='"+width+"'
align='right'> \
    <tr><th class='exkurs'>EXKURS</th></tr> \
    <tr><td align='center'> \
    <a href='#' onclick=\"openexkurs('"+filename+"')\"><img
src='../images/explorers.gif' alt='Exkurs: "+linktext+"' border='0'
align='absmiddle'></a> \
    <br><a href='#'
onclick=\"openexkurs('"+filename+"')\">"+linktext+"</a>
</td></tr></table>");
}

// inserts a list of links to all pages in chapter "chapter". used in
the contents-page for the quick links to the pages
function insertpages(chapter) {
    for (p=1;p<=maxpages[chapter];p++) {
        document.write("<a
href='javascript:loadpage('"+chapter+", '"+p+"')'>Seite '"+chapter+"-
"+p+"</a><br>");
    }
}
```

E DIE CD-ROM

Auf dieser Seite finden Sie die CD-ROM mit dem Lernmodul „Hashing“, das im Rahmen dieser Arbeit entstanden ist.

F LITERATURVERZEICHNIS

- [Appelrath, 2000] **Appelrath, H.-J., Ludewig, J.:** *Skriptum Informatik – eine konventionelle Einführung*, B.G. Teubner, Stuttgart/Leipzig, 2000
- [Baumann, 1996] **Baumann, R.:** *Didaktik der Informatik*, Ernst Klett Verlag, Stuttgart, 1996
- [Edelmann, 1986] **Edelmann, W.:** *Lernpsychologie – Eine Einführung*, Urban & Schwarzenberg, München/Weinheim, 2. Aufl. 1986
- [Grillenbeck, 2000] **Grillenbeck, R.:** *Didaktik und Methodik der Theoretischen Informatik: Motivation und computerunterstütztes Lernen*, Universität Erlangen-Nürnberg, Dissertation 2000 / 3953, Oktober 1999
- [Heinrich, 1972] **Heinrich, P.B.:** *Fachdidaktik und Lehrobjektivierung in den Naturwissenschaften. Ergebnisse des 1. BTZ-Symposiums*, Wiesbaden, Bildungstechnologisches Zentrum, 1972
- [Knuth, 1998] **Knuth, D.:** *The Art of Computer Programming, Vol.3: Sorting and Searching*, Addison-Wesley, Massachusetts, 2. Aufl. 1998
- [Lefrancois, 1994] **Lefrancois, G.R.:** *Psychologie des Lernens*, Springer, Berlin, 3. Aufl. 1994
- [Lum et al., 1971] **Lum, V. Y., Yuen, P. S. T., und Dodd, M.:** *Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files*, Communications of the ACM, 14(4):228{239, 1971
- [Martens, 1966] **Martens, Jens U.:** *Gestalttheorie in der Programmierten Unterweisung*, In: *Lehrprogramme*, 1966

-
- [Ottmann, 2002] **Ottman, T., Widmayer, P.:** *Algorithmen und Datenstrukturen*, Spektrum Akademischer Verlag GmbH, Heidelberg/Berlin, 4. Aufl. 2002
- [Saake, 2002] **Saake, G., Sattler, K.-U.:** *Algorithmen & Datenstrukturen, Eine Einführung mit Java*, dpunkt-Verlag, Heidelberg, 2002
- [Schöning, 2001] **Schöning, Uwe:** *Algorithmik*, Spektrum Akademischer Verlag GmbH, Heidelberg/Berlin, 2001
- [Standish, 1980] **Standish, T.:** *Data Structure Techniques*, Addison-Wesley, Massachusetts, 1980
- [Standish, 1994] **Standish, T.:** *Data Structures, Algorithms, and Software Principles*, Addison-Wesley, Massachusetts, 1994
- [Weicker, 2003] **Weicker, N.:** *Didaktik der Informatik, Vorlesungsunterlagen*, <http://www.fmi.uni-stuttgart.de/fk/lehre/ss03/didaktik.html>, Universität Stuttgart, Zugriff am 06.01.2004

ERKLÄRUNG

Ich versichere, dass ich diese Arbeit selbständig verfasst und nur die angegebenen Hilfsmittel verwendet habe.

Stuttgart, den 14. April 2004

Zissis Siantidis