Algorithmen und Datenstrukturen

SoSe 2008 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Algorithmen und Datenstrukturen

Gesamtübersicht

- Organisatorisches / Einführung
- Grundlagen: RAM, $\mathcal{O}$-Notation, Rekursion, Datenstrukturen
- Sortieren
- Wörterbücher und Mengen
- Graphen und Graphalgorithmen

Organisatorisches

Vorlesungen MI 8-10 im HS 11/12; FR 12-14 im HS 13

Übungsbetrieb Die Übungen werden wöchentlich besprochen.
BEGINN: in der zweiten Semesterwoche DO 16-18, HS 12

Dozentensprechstunde DO, 13-14 in meinem Büro H 410 (4. Stock)

Mitarbeitersprechstunde Daniel Schmitt, MO 14-16, H429

Tutorensprechstunde
Sebastian Schlecht, MO 12.45 - 13.45, H407
Alexander Woskobojnik, FR 14 - 15, H447

Teilnahme an der Abschlussklausur

Termin Abschlussklausur: Letzte Semesterwoche oder erste Woche in der vorlesungsfreien Zeit
GENAUER: MI, 9.7., vermutlich nachmittags

Nachklausur zu Begin des nächsten Semesters.

Genaueres wird noch bekanntgegeben.

Sollten Sie an einer der Klausuren nicht teilnehmen können, so legen Sie bitte ein ärztliches Attest vor; andernfalls wird die Klausur mit 0 Punkten bewertet.

Kleiner Craschkurs C++/prozedurale Sprachen

Wann und Wo ?
DO 10.04 16-18 im Cip Pool H523 und DO 17.04 14-16 Cip Pool H523

Für alle, die noch nie programmiert haben/keine Vorstellung von einer Prozedur haben.

ZIEL: Wir wollen ein Verständnis entwickeln, wie man Algorithmen in prozeduralen Sprachen korrekt ausdrückt, am Beispiel von C++. Auf dem Programm stehen:
Grundlegende Sprachelemente von C++; Übersetzen, Linken und Ausführen von Programmen; Erstellen von Pseudocode und Implementierung in C++.

PS: Die Teilnehmer benötigen einen Rechneraccount.

Scheinkriterien

Sie müssen in der Abschlussklausur oder der Nachklausur mindestens 45% der Punkte erreichen.

Zur Klausurzulassung benötigen Sie mindestens 50% der Punkte aus den Übungen.

Abgabe von Hausaufgaben

Diese sollte in Gruppen zu 2-3 Personen erfolgen.

Abgabeschluss ist in der Regel “kurz vor” Übungsbeginn, d.h., donnerstags bis 16 Uhr, im Postkasten der Vorlesung im 4. Stock vor Raum H426 oder direkt in der Übung. Da nach dem jeweils angegebenen Abgabezeitraum die Aufgaben vorgerechnet werden, besteht keine Möglichkeit einer späteren Abgabe. Mit anderen Worten: Verspätete Abgaben gelten als nicht abgegeben und werden dementsprechend mit 0 Punkten bewertet.

Die Lösungen sind handschriftlich anzufertigen; weder Schreibmaschinen- noch Computerausdrucke werden akzeptiert, erst recht keine Kopien.

Eine Woche später (in der Regel) werden die korrigierten und “bepunkteten” Hausaufgaben wieder zurückgegeben (ebenfalls vor den Übungen).

Probleme ? Fragen ?

Klären Sie bitte Schwierigkeiten mit Vorlesungen oder Übungen möglichst umgehend in den zur Verfügung gestellten Sprechzeiten.

In der Tutorensprechstunde stehen Ihnen (alternierend) Studenten höherer Semester zu Rückfragen bereit.

Wir helfen Ihnen gerne!
. . . wir sind aber keine Hellseher, die Ihnen Ihre Schwierigkeiten an der Nasenspitze ansehen. . .

Einführung 1: Einordnung

- Was heißt “Algorithmus” ?

- Was sind “Datenstrukturen”?

“Algorithmus” (aus einem Wörterbuch, hier Wikipedia)

Das Wort *Algorithmus* ist eine Abwandlung oder Verballhornung des Namens von Muhammed Al Chwarizmi (ca. 783-850), dessen arabisches Lehrbuch *Über das Rechnen mit indischen Ziffern* (um 825) in der mittelalterlichen lateinischen Übersetzung begann mit den Worten Dixit Algorismi („Algorismi hat gesagt“).

Durch Gräzisierung der Schreibweise des vermeintlich griechischen Wortbestandteiles rismus hat sich dann in der lateinischen Wissenschaftssprache die Schreibung algorithmus ergeben; in dieser Form hat sich das Wort später als Fachterminus für **geregelte Prozeduren zur Lösung definierter Probleme** eingebürgert.

Das ist (mathematisch) nicht sehr genau...

Eigenschaften eines Algorithmus

Finitheit Das Verfahren muss in einem endlichen Text eindeutig beschreibbar sein.

Ausführbarkeit Jeder Schritt des Verfahrens muss tatsächlich ausführbar sein.

Dynamische Finitheit Das Verfahren darf zu jedem Zeitpunkt nur endlich viel Speicherplatz benötigen.

Terminierung Das Verfahren darf nur endlich viele Schritte benötigen.

Determiniertheit Der Algorithmus muss bei denselben Voraussetzungen das gleiche Ergebnis liefern.

Determinismus Die nächste anzuwendende Regel im Verfahren ist zu jedem Zeitpunkt eindeutig definiert.

Eine mathematisch genauere Festlegung

Eine Berechnungsvorschrift zur Lösung eines Problems heißt genau dann *Algorithmus*, wenn eine zu dieser Berechnungsvorschrift äquivalente deterministische Turingmaschine existiert, die für jede Eingabe, die eine Lösung besitzt, stoppt.

Einige von Ihnen kennen möglicherweise bereits Turingmaschinen; sie werden normalerweise aber erst später, d.h. in "Berechenbarkeit und Komplexität" (Diplomstudiengang) bzw. "Grundlagen der Theoretischen Informatik II" (Bachelorstudiengang) eingeführt.

HIER: RAM-Modell

Algorithmen und Programmierung

Das *Programmieren* (das üblicherweise mit der Informatik als Erstes in Zusammenhang gebracht wird) kann man als konkretes Ausprägen / Implementieren eines vorher entwickelten Algorithmus ansehen.

$\rightsquigarrow$ Algorithmen sind möglichst programmiersprachen- und natürlich auch maschinenunabhängig zu beschreiben.

$\rightsquigarrow$ Verwendung von “Pseudocode” zur Notierung von Algorithmen sowie eines allgemeinen abstrakten Maschinenmodells (bei uns einer “RAM”) zur besseren Analyse (Vergleich) von Algorithmen

“Struktur”

Mathematik und Informatik als Strukturwissenschaften

“Softwareengineering” bedeutet immer auch:

- Beschreibe das Anwendungsproblem genau.
- Abstrahiere von unnötigen Einzelheiten.
- Erkenne statt dessen das Wesentliche (die Struktur).
- Formalisiere das Wesentliche, um die Aufgabe in ein Programm überführen zu können, das auch das leistet, was der Anwender möchte.

Datenstrukturen

Die Definition von Datenstrukturen erfolgt im Allgemeinen durch die Angabe einer konkreten Spezifikation zur Datenhaltung und der dazu nötigen Operationen (für Zugriff und Manipulation der Daten).

Diese konkrete Spezifikation legt das allgemeine Verhalten der Operationen fest und abstrahiert damit von der konkreten Implementierung der Datenstruktur.

Beispiele für Datenstrukturen: Felder (Arrays), Listen, Bäume, ...

Grundverständnis der Vorlesung

Die Vorlesung beschäftigt sich mit **Grundbegriffen der Informatik** als “Computer Science:”

- Gutes Programmieren und ebenfalls gutes Software-Entwickeln ist ohne gute Kenntnisse über Algorithmen unvorstellbar.
- “Algorithmik” zeigt Ihnen auch die Nützlichkeit und Anwendbarkeit (vorher) gelernter mathematischer Kenntnisse.
- Viele Professuren beschäftigen sich gerade in Trier mit der Entwicklung von Algorithmen.

Lernziele

- Gute Algorithmen für wichtige Probleme kennenlernen
- Paradigmen für Algorithmen(entwurf) (Klassen von Algorithmen)
- Korrektheitsbeweise führen können
- Laufzeit- und Speicherplatzabschätzungen (asymptotisch) durchführen können

Einführung 2: Literatur

Begleitend zur Vorlesung empfehlen wir:

Ralf H. Güting, Stefan Dieker: Algorithmen und Datenstrukturen:

Dieses Buch bietet einen einfachen Einstieg in grundlegende Algorithmen und Datenstrukturen, ebenso wie in (abstrakte) Datentypen.

Die Inhalte werden mit Pseudocode anschaulich vermittelt und in Java Codebeispielen umgesetzt.

ISBN 3-519-12121-2

Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms Standardwerk.

Diese Buch ist “leicht“ geschrieben dabei aber exakt in mathematischem Stil.

Es beginnt bei den Grundlagen und schließt sehr umfangreich.

Einerseits arbeitet es konstruktiv, andererseits finden sich eine Vielzahl von Korrektheits- und Laufzeitbeweisen.

Viele Beispiele in Pseudocode und Veranschaulichungen.

Ein ”Muss“ für alle, die sich für die Entwicklung von Algorithmen interessieren.

ISBN 0-262-53196-8

Sedgewick: Algorithmen Ebenfalls ein Standardwerk, das Ihnen auch ein guter Begleiter sein kann im weiteren Studium.

Es ist sehr dick; von den 45 Kapiteln werden wir hier bestenfalls 20 anschneiden. . .

ISBN 3-8273-7032-9

Als Varianten bieten sich an: ”Algorithmen in JAVA“ bzw. ”Algorithmen in C(++)“

Kleinberg, Tardos: Algorithm Design ist weniger ”elementar“ als die vorigen beiden Standardwerke, aber liefert vielleicht den besten Überblick, was ”Algorithmik“ insgesamt bedeutet (und Sie erkennen an den Kapitelüberschriften auch viele Vorlesungstitel aus dem Masterprogramm wieder).

ISBN 0-321-29535-8

Mehlhorn: Datenstrukturen und Algorithmen Band 1 Sortieren und Suchen; Band 2 Graphenalgorithmen sind besonders relevant.

Das war mal das deutschsprachige Standardwerk, ist aber etwas in die Jahre gekommen.

ISBN 3-519-02255-9

Ein Beispiel Lineare Suche nach Wert `x` im Array `A[0,n-1]`:

```
i = 0;
while ((i < n) && (A[i] != x))
   i++;
```

In welchem Sinne wird hier nach `x` gesucht ?

Was passiert, wenn `x` gefunden wird ?

Was passiert, wenn `x` nicht gefunden wird ?

Arbeitet das Programm(fragment) stets “wie beabsichtigt” ?

Ein Beispiel Lineare Suche nach Wert `x` im Array `A[0,n-1]`:

```
i = 0;
while ((i < n) && (A[i] != x))
   i++;
```

In welchem Sinne wird hier nach `x` gesucht ?!

Wenn `x` gefunden wird, so bricht die Schleife ab, denn die zweite Bedingung `(A[i] != x)` ist verletzt, sobald ein Index `i` gefunden wurde, an dessen Stelle im Feld `A` der Wert `x` steht.

Da der Laufindex `i` innerhalb des Arrays nur Werte kleiner `n` annehmen kann, hat `i` in diesem Fall auch nach Verlassen der Schleife einen Wert kleiner `n`, und dort steht im Feld `A` der Wert `x`.

Was passiert, wenn `x` nicht gefunden wird ?

Die zweite Bedingung `(A[i] != x)` ist nie verletzt, also kann ein Verlassen der Schleife nur durch Verletzen der ersten Bedingung geschehen, d.h., der Laufindex `i` hat beim Verlassen der Schleife einen Wert von mindestens `n`.

Arbeitet das Programm(fragment) stets “wie beabsichtigt” ?

Ein Beispiel Lineare Suche nach Wert `x` im Array `A[0,n-1]`:

```
i = 0;
while ((i < n) && (A[i] != x))
   i++;
```

Arbeitet das Programm(fragment) stets “wie beabsichtigt” ?

Mögliches Problem: Der Wert `x` ist nicht im Array `A[0,n-1]`:

$\rightsquigarrow$ Der Laufindex `i` wird bis `n` hochgezählt.

$\rightsquigarrow$ “Eigentlich” müsste auch noch die zweite Bedingung `(A[i] != x)` getestet werden.

$\rightsquigarrow$ Es wird auf ein Element im Speicher zugegriffen, nämlich `A[n]`

Dieser Zugriff könnte auf einen geschützten Speicherbereich erfolgen und somit zu einem Programmfehler führen !

Alles halb so schlimm ?

Warum “klappt” denn das gezeigte Programm “tatsächlich” ?
(und man findet diesen Code in vielen Stellen im Internet...)

In einer Beschreibung der Programmiersprache C (an der sich das Fragment orientiert) finden wir `http://docs.hp.com/en/B3901-90014/ch05s10.html`:

Logical operators (...) are the only operators for which the order of evaluation of the operands is defined. The compiler must evaluate operands from left to right. Moreover, the compiler is guaranteed not to evaluate an operand if it is unnecessary. For example, in the expression

```
if ((a != 0) && (b/a == 6.0))
```

if a equals 0, the expression `(b/a == 6)` will not be evaluated. This rule can have unexpected consequences when one of the expressions contains side effects.

Ähnliches gilt auch bei der Programmiersprache C++:
`http://msdn2.microsoft.com/en-us/library/c6s3h5a7(VS.80).aspx`

Rückschlüsse

Wir werden im Folgenden versuchen zu vermeiden, Spitzfindigkeiten wie den Unterschied zwischen AND und ANDTHEN (in ADA) auszunutzen.

Wir streben lieber Klarheit an.

Betrachten wir unser Beispiel aber noch etwas weiter. . .

Ein Beispiel Lineare Suche nach Wert `x` im Array `A[0,n-1]`:

```
i = 0;
while ((i < n) && (A[i] != x))
   i++;
```

Wieviele “Schritte” führt der Algorithmus im schlimmsten Falle aus ?
Eingangs erfolgt eine Zuweisung.
Jedesmal, wenn die Schleife betreten werden soll, werden zwei Tests durchgeführt und die (Booleschen) Ergebnisse per Konjunktion verknüpft.
Mit dem nachfolgenden Inkrement (im Schleifenrumpf) werden “im negativen Fall” (`x` wird nicht gefunden) je Schleifeniteration vier “Operationen” durchgeführt.
Im schlimmsten Fall wird `x` nicht gefunden, es werden also `n` (vergebliche) Schleifendurchläufe durchgeführt sowie noch ein abschließender negativer Test der ersten Bedingung `(i < n)`.
Insgesamt sind das also `4n+2` Operationen.
Aufgabe: Haben wir wirklich alles “richtig gezählt” ?
Welcher Annahmen haben wir bei dieser Art von Zählung (implizit) gemacht ?

Was waren unsere impliziten Annahmen bei der Analyse ?

Wir hatten fünf unterschiedliche Operationen “einfach gleich” gezählt:
`i = 0;` ist eine *Zuweisung* einer Konstanten zu einer Variablen.
`i < n` ist ein *Vergleich* zweier Variablenwerte.
`A[i] != x` ist ebenfalle ein Vergleich; er beinhaltet aber darüber hinaus den *Zugriff auf ein Element in einem Feld*.
`i++` ist ein *Inkrement* (Hochzählen) einer Variablen.

Je nach konkreter Maschine und je nach konkretem Compiler sind die genannten Operationen “tatsächlich” unterschiedlich schnell und auch untereinander verschieden.

Das RAM-Modell Eigenschaften

Die Maschine hat einen (zentralen) Prozessor.
Eine RAM arbeitet sequentiell (keine Parallelverarbeitung von Befehlen).
Jeder Schritt der Maschine verursacht Kosten 1, unabhängig von der Größe der Operanden (*Einheitskostenmaß*); nur manchmal werden wir davon abweichen...
Zu den "Schritten" der Maschine zählen neben direkten ebenfalls indirekte Speicherzugriffe.
Indirekte Speicherzugriffe gestatten die effiziente "RAM-Implementierung" von Feldern.