Karlsruher Institut für Technologie

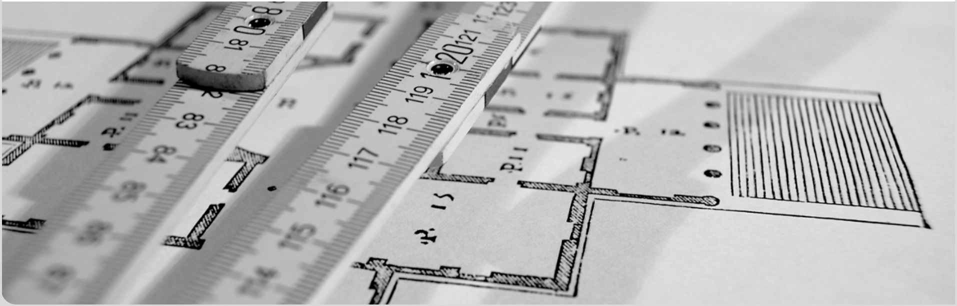
Karlsruher Institut für Technologie

Algorithmen I
Tutorium 32

Eine Lehrveranstaltung im SS 2017

Daniel Jungkind (ufesa@kit.edu) | 05. Mai 2017

INSTITUT FÜR THEORETISCHE INFORMATIK



Folien partiell geklaut von:

Christopher Hommel

Tutorium 16

Herzlichen Dank! :)

Kennenlernen 2.0

1	Programmieren ist noch ziemlich neu für mich.
2	Vor dem Studium habe ich nicht viel mit Programmieren zu tun gehabt, komme aber gut damit zurecht.
3	Ich konnte schon vor dem Studium einigermaßen programmieren und kannte daher vieles aus der Vorlesung schon.
4	In der Programmieren-Vorlesung habe ich eigentlich nichts neues gelernt, da ich auch so schon gut programmieren konnte
5	Im Programmieren habe ich langjährige Erfahrung.
$\perp$	Nieder mit Pauschalantworten! Ich formuliere selbst!

Kennenlernen 2.0

1	Ich habe mich noch nicht an eigenen Projekten versucht.
2	Ich habe ein paar kleine eigene Projekte geschrieben.
3	Ich habe bereits ein paar Projekte durchgeführt, die von Umfang her schon fast mit den Abschlussaufgaben vergleichbar waren.
4	Ich habe schon ein paar eigene größere Projekte realisiert.
5	Ich habe bereits einige ziemlich große Projekte umgesetzt.
$\perp$	Nieder mit Pauschalantworten! Ich formuliere selbst!

Wozu das Tutorium?

- **Für Eure Fragen**
- Zur Vorbereitung auf die Übungsblätter
- ...und damit zur Vorbereitung auf die Klausur! :)
- Wiederholung und Vertiefung klausurrelevanter Themen
- Und vor allem: Spaß!

Orga-Kram – Übungsblätter

- **Freiwillig**
- **Ausgabe**: Mo nach der Vorlesung,
- **Abgabe**: Di nächster Woche, 12.45 Uhr (das sind 9 Tage Zeit) im Kasten im Untergeschoss des Infobaus
- Abgabe **schwerstens empfohlen**, gibt nämlich einen Klausurbonus:
  $\geqslant$ 25% der Gesamtpunkte $\Rightarrow$ 1 Bonuspunkt
  $\geqslant$ 50% der Gesamtpunkte $\Rightarrow$ 2 Bonuspunkte
  $\geqslant$ 75% der Gesamtpunkte $\Rightarrow$ 3 Bonuspunkte
  (Die Bonuspunkte helfen nicht beim Bestehen, verbessern aber die Note meistens um eine Stufe.)
- Korrigierte Übungsblätter werden einige Male ins Tutorium mitgebracht; zu lange danach $\Rightarrow$ bei den Übungsleitern abholen
- Abschreiben: Böhse™. Wird geahndet.

Orga-Kram – der Kummerkasten

- Offizielle Evaluation 1$\times$ pro Semester: *bisschen* zu wenig Feedback
- Daher: Kasten auf der Vorlesungshomepage (hier klicken!) für all eure Meinungen und Anregungen (**anonym**!)
- **Keine Stofffragen!** Anonym $\Rightarrow$ Antwort/Rückfragen **nicht möglich!** (Besser: ILIAS oder **hier stellen**!)
- Gerne auch Feedback zu Tutorien, dann aber bitte mit **Nummer des Tutoriums** bzw. **Name des Tutors** (wie gesagt, da anonym!)
- Also: Etwas läuft furchtbar schief (oder vielleicht auch nur besonders gut?) $\Rightarrow$ Meldet es **zeitnah**! (Kurz vor Ende der Vorlesungszeit ist meistens zu spät)

Pseudocode

- ✚ Es gibt keine exakte Sprachdefinition
- − Es gibt keine exakte Sprachdefinition
- Daher im Folgenden: Grobe *Richtlinie* für Pseudocode (ohne Anspruch auf Vollständigkeit)
- Das **Wichtigste**: Es sollte **klar** werden, was **gemeint** ist, d.h. Pseudocode soll vor allem **übersichtlich** und **verständlich** sein
- Kommentare mit // (wie in Java), #, o.ä.
- Semikolon am Ende eines Befehls unnötig

Pseudocode

Variablendeklaration und -initialisierung

$Variablenname = Wert : Typ$

$a := 3$

b : **array**$[Von..Bis]$ **of** Int // *Erlaubte Indizes für Zugriff: Von..Bis*

$c =$ **new** $XYZ(Konstruktorparameter)$

$d =$ "leet" : $String$

- Typ kann weggelassen werden, wenn offensichtlich
- Mögliche Typen sind zum Beispiel
  - $\mathbb{R}$, $\mathbb{N}$, $\mathbb{Z}$ / Int(eger), Bool(ean), ...
  - **array** (als im Speicher zusammenhängender „Datenblock“)
  - *String*
  - Weitere Datenstrukturen aus der VL (more to come!)
  - **Pointer to** *T* als *„Zeiger“* (engl. *handle*) für Objekte vom Typ T
  - *Element* ist Platzhalter für beliebigen Typ (so wie *Object* in Java)

Pseudocode

Besondere Werte

- $+\infty, -\infty$
- $\perp$ als Nullobjekt mit undefiniertem Wert

Pseudocode

Kontrollstrukturen

while $x \neq y$ **and** $y \neq z$ **do**
    // Anweisungen

repeat
    // fußgesteuert
until $x = y$ **or** $y = z$

if $z = 42$ **then**
    // Andere Anweisungen
else
    // Mehr andere Anweisungen

for $i := 1$ **to** n **do**
    // Noch mehr Anweisungen

for $i := 1$ **to** n **step** k **do**
    // Mit Schrittweite!

Pseudocode

Karlsruher Institut für Technologie

- Und viele mehr, z.B. **do while**, **for each**, ...
- Schlüsselworte wie **continue**, **break**, **switch** natürlich auch
- Trennzeichen für Anweisungsblöcke:
  - **do**/**begin** – **end**
  - {...} (geschweifte Klammern)
  - Linien
  - Nur durch Einrückung (dann aber ordentlich)

Pseudocode

Mathe-Bequemlichkeit

// K, M Mengen

select any $x \in K$

for each $y \in M$

$(a, b) := (3, 5)$ // geordnetes Tupel/Array (konstante Anzahl Elemente)

$(a, b) := (b, a)$ // bequemes Vertauschen

$S := \{1, 2, 3\}$ // ungeordnete Menge

$f := \langle 1, 2, 3 \rangle$ // geordnete Folge (Elemente können an- und abgehängt werden)

- Generell: Quantoren erlaubt, solange trivial ist, welcher programmatischen Funktion sie entsprechen *und wie sie die Laufzeit beeinflussen*!

Pseudocode

Funktionen/Prozeduren

function / **procedure** name($name_1$: Typ_1, $name_2$: Typ_2) : *Rückgabetyp*

```
    // Knorker Code
    return 42
```

- Rückgabetyp wird weggelassen, wenn nichts zurückgegeben wird
- Konvention: Kein Rückgabewert („`void`“) – **procedure**/**method**, Rückgabe vorhanden: **function**

Pseudocode

Anmerkungen:

- **Selbsterklärende** Bezeichner, z.B. *print*("...") statt *System*.*out*.*println*("...")
- Komplexere Stellen bitte **kommentieren**, sonst versteht es keiner
- Mehrere Funktionen (z.B. Hilfsfunktionen): Kenntlich machen, wo **Hauptfunktion**!
- Im „Notfall“: An Java orientieren
- Für mehr Pseudocode-Details siehe Buch vom Sanders

Pseudocode

Bearbeitungshinweise:

- Falls die Aufgabenstellung euch die Wahl lässt, könnt ihr selbst entscheiden, ob **Pseudocode** oder **Fließtext**
- Mehr als zwei Seiten Pseudocode $\Rightarrow$ Vermutlich viel zu kompliziert oder falsch
- Ist das Ergebnis für andere Personen **verständlich**?
- Ist das Ergebnis **angenehm zu lesen**?

Pseudocode

Aufgabe 1: Pseudocode schreiben

Als Eingabe erhält der Algorithmus eine natürliche Zahl n. Es wird ein Boolean-Array von $2..n$ angelegt und mit **false** initialisiert. Dann wird eine Zahl i von 2 bis zur abgerundeten Wurzel von n iteriert. Falls im Schleifendurchlauf der i-te Boolean-Wert **false** ist, wird eine weitere Zahl j von $2i$ bis n durchlaufen und dabei in Schritten der Größe i inkrementiert. Darin wird jeweils der j-te Boolean-Wert auf **true** gesetzt.

Am Ende iteriert der Algorithmus erneut eine Zahl i von 2 bis n. Falls der i-te Boolean-Wert nicht **true** ist, wird ausgegeben, dass der Wert von i prima ist.

Pseudocode

(Mögliche) Lösung von Aufgabe 1

Input: $n \in \mathbb{N}$

werte $=$ (**false**, ..., **false**) : **array**$[2..n]$ **of** *Boolean*

for $i = 2$ **to** $\lfloor \sqrt{n} \rfloor$ **do**

  if not *werte*$[i]$ **then**

    for $j = 2i$ **to** n **step** i **do**

      werte$[j]$ $:=$ **true**

for i **from** 2 **to** n **do**

  if not *werte*$[i]$ **then**

    print$(i +$ " ist prima!"$)$

Pseudocode

Aufgabe 2: Mehr Pseudocode schreiben

Als Eingabe erhaltet ihr *n* Studenten, deren Matrikelnummer jeweils als Array gegeben ist. Diese Studenten sollt ihr in zwei Gruppen einteilen, in die eine Gruppe kommen die Studenten, deren Quersumme der Matrikelnummer gerade ist, und in die andere die anderen. Als Ausgabe gibt euer Algorithmus die beiden Gruppen als geordnetes Paar zurück.

Pseudocode

(Mögliche) Lösung von Aufgabe 2

function *groupStudents*(*students* : **array**[0...*n* − 1] **of** *Student*) :
(List **of** *Student*, List **of** *Student*)

$\quad groups = (\langle\rangle, \langle\rangle)$: **array of** List **of** *Student*

$\quad$ **foreach** $student \in students$ **do**

$\quad\quad sum = 0 : Int$

$\quad\quad nr := student.matrikelnummer$

$\quad\quad$ **for** $i = 0$ **to** $|nr| - 1$ **do**

$\quad\quad\quad sum \mathrel{+}= nr[i]$

$\quad\quad groups[sum$ **mod** $2].add(student)$

$\quad$ **return** *groups*

Algorithmenanalyse

Algorithmen in Pseudocode lesen und schreiben ist (leider) nicht das Höchste der Gefühle. Sondern:

- Auf welcher Vorgehensweise basiert der Algorithmus?
  $\Rightarrow$ Verschiedene **Kategorien** von Algorithmen
- Tut der Algorithmus das, was er tun soll?
  $\Rightarrow$ **Invarianten**
- Wie schnell ist der Algorithmus?
  $\Rightarrow$ **O-Kalkül**

Algorithmenanalyse

O-Kalkül

- Vernachlässigung konstanter Faktoren
- Zuordnung, welches Laufzeit-Verhalten der Algorithmus für sehr große Eingaben aufweist

Algorithmenanalyse

Alte Bekannte

- $O(f(n)) = \{g(n) \mid \exists\, c, n_0 > 0, \text{ so dass}$
  $0 \leqslant g(n) \leqslant c \cdot f(n) \quad \forall n \geqslant n_0\}$

- $\Theta(f(n)) = \{g(n) \mid \exists\, c_1, c_2, n_0 > 0, \text{ so dass}$
  $0 \leqslant c_1 \cdot f(n) \leqslant g(n) \leqslant c_2 \cdot f(n) \quad \forall n \geqslant n_0\}$

- $\Omega(f(n)) = \{g(n) \mid \exists\, c, n_0 > 0, \text{ so dass}$
  $0 \leqslant c \cdot f(n) \leqslant g(n) \quad \forall n \geqslant n_0\}$

Algorithmenanalyse

Die Neuen

- $o(f(n)) = \{g(n) \mid \forall c > 0 \, \exists \, n_0 > 0, \text{ so dass}$
  $0 \leqslant g(n) \leqslant c \cdot f(n) \quad \forall n \geqslant n_0\}$

- $\omega(f(n)) = \{g(n) \mid \forall c > 0 \, \exists \, n_0 > 0, \text{ so dass}$
  $0 \leqslant c \cdot f(n) \leqslant g(n) \quad \forall n \geqslant n_0\}$

Algorithmenanalyse

Anschaulich:

$o(f(n))$	$<$	echt schwächer wachsende Funktionen
$O(f(n))$	$\leqslant$	schwächer oder gleich stark wachsende Funktionen
$\Theta(f(n))$	$=$	genau gleich stark wachsende Funktionen
$\Omega(f(n))$	$\geqslant$	stärker oder gleich stark wachsende Funktionen
$\omega(f(n))$	$>$	echt stärker wachsende Funktionen

Algorithmenanalyse

O-Kalkül: Formeln

$f(n) \in o(g(n))$	$\Longleftrightarrow$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$f(n) \in O(g(n))$	$\Longleftrightarrow$	$0 \leqslant \limsup_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$
$f(n) \in \Theta(g(n))$	**!** $\Longleftarrow$ **!**	$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$
$f(n) \in \Omega(g(n))$	$\Longleftrightarrow$	$0 < \liminf_{n\to\infty} \frac{f(n)}{g(n)} = c \leqslant \infty$
$f(n) \in \omega(g(n))$	$\Longleftrightarrow$	$\limsup_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Unter Informatikern gilt üblicherweise $\log n := \log_2(n)$
- $\log(a \cdot b) = \log a + \log b$
- $\log(a^b) = b \cdot \log a$
- $\log(\frac{a}{b}) = \log a - \log b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $a^{\log_a(b)} = b$
- $x^{a \cdot b} = (x^a)^b = \left(x^b\right)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:
  - $\log(10 \cdot n) \in O(\log n)$
  - $n^n \in \Theta(2^{n \log n})$

Algorithmenanalyse

Ein mysteriöser Algorithmus

procedure $foo(a : \textbf{array of } Int)$
- $n := |a|$
- $flag : Bool$
- **repeat**
  - $flag :=$ **true**
  - **for** i **from** 0 **to** $n-2$ **do**
    - **if** $a[i] > a[i+1]$ **then**
      - $\begin{pmatrix} a[i] \\ a[i+1] \end{pmatrix} := \begin{pmatrix} a[i+1] \\ a[i] \end{pmatrix}$
      - $flag :=$ **false**
- **until** $flag$

Algorithmenanalyse

- Das ist *Bubblesort*, der ein Array aufsteigend sortiert
- Best case?
  $\Rightarrow O(n)$, wenn das Array schon sortiert ist
- Worst case?
  $\Rightarrow O(n^2)$, wenn das Array absteigend („falsch rum“) sortiert ist

Algorithmenanalyse

Korrektheitsbeweis

- Korrektheitsbeweis ist **zweiteilig**:
  - 1. Teil – **Funktionalität**: Mit Invariante beweisen, dass der Algorithmus ein **korrektes** Ergebnis erzeugt
  - 2. Teil – **Terminierung**: Beweisen (ggf. anhand einer Invariante), dass der Algorithmus „irgendwann **fertig** wird". Manchmal trivial, manchmal knackig (und damit aufwendig)
- **Aufgabenstellung beachten**: Wenn („nur") eine Invariante angegeben/bewiesen werden soll $\Rightarrow$ Terminierungsbeweis nicht nötig!

Algorithmenanalyse

Invarianten

- Invariante finden: Manchmal offensichtlich, manchmal **Kreativität** gefragt
- Beweisprinzip von Algorithmen durch Invarianten direkt analog zu **Induktion**:
- „IA“: Invariante gilt bei **Beginn** des Algorithmus / der Schleife
- „IV“: Die Invariante war beim Ende des **vorherigen** Ausführungsschrittes gültig
- „IS“: Mithilfe der IV zeigen, dass die Invariante auch beim Ende des **aktuellen** Ausführungsschrittes gültig ist

Beispiel SelectionSort

SelectionSort

procedure SelectionSort(A : **array**[1..n] **of** *Element*)
- **for** $i := 1$ **to** n **do**
  - **invariant** $A[1\ \dots\ i-1]$ **is** sorted **and** $\max(A[1\ \dots\ i-1]) \leqslant \min(A[i..n])$
  - minIndex $:= i$
  - **for** $j := i + 1$ **to** n **do**
    - **if** $A[j] < A[\text{minIndex}]$ **then**
      - minIndex $:= j$
  - **assert** $A[minIndex] = \min(A[i..n])$
  - $swap(A[i], A[minIndex])$

SelectionSort – Beweis Invariante

Definiere $\max((a_1, ..., a_n)) := \max\{a_1, ..., a_n\}$ und $\max(()) := -\infty$.

Beweis Invariante:

$A[1 \dots i-1]$ **is** sorted **and** $\max(A[1 \dots i-1]) \leqslant \min(A[i..n])$

IA. $(i = 1)$: $A[1..0] = ()$ ist sortiert und
$-\infty = \max(A[1..0]) \leqslant \min(A[1..n])$.

IV. *(i > 1)*: Die Invariante galt am Ende des Durchlaufs $i - 1$.

IS. $(i - 1 \rightsquigarrow i)$: Laut IV ist $A[1 \dots i-1]$ sortiert und
$\max(A[1 \dots i-1]) \leqslant \min(A[i \dots n])$ und $minIndex \in \{i, \dots, n\}$
$\Rightarrow A[i-1] \leqslant A[minIndex]$ und $A[minIndex] \leqslant A[i]$.
$\Rightarrow A[minIndex]$ kann zur Fortsetzung der Sortierung problemlos nach $A[i]$ verschoben werden! Tauschen von $A[i]$, $A[minIndex]$:
$\Rightarrow A[1..i]$ ist sortiert,
$A[i] = \max(A[1..i]) \leqslant \min(A[i+1 \dots n])$. $\square$

$\Rightarrow$ Nach dem n-ten Schleifendurchlauf gilt also: $A[1..n]$ ist sortiert.

SelectionSort – Beweis Terminierung

In diesem Fall trivial:

- Schleifenvariable i nach oben durch n beschränkt
- ...und wird in jedem Durchlauf inkrementiert (und sonst nicht verändert)

$\Rightarrow$ SelectionSort terminiert

$\Rightarrow$ SelectionSort funktioniert! Yay! :D