Einführung in die Informatik

Klaus Knopper

02.11.2004

Algorithmen

Ein **Algorithmus** ist eine genau definierte Berechnungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen. Ein Algorithmus wird durch eine endliche Menge von Regeln definiert, die nacheinander angewendet und oft nach bestimmten Bedingungen wiederholt werden. Ein Algorithmus terminiert, d.h. nach endlich vielen Schritten wird mit einem Ergebnis gestoppt.

Algorithmen in der realen Welt

In der realen Welt gibt es viele bekannte Beispiele für Algorithmen:

- Kochrezept
- Reparatur- und Bedienungsanleitungen
- Waschmaschinenprogramme
- . . .

Eigenschaften von Algorithmen

1. Das Verfahren muss in einem endlichen Text eindeutig beschreibbar sein (Finitheit).
2. Jeder Schritt des Verfahrens muss auch tatsächlich ausführbar sein (Ausführbarkeit).
3. Das Verfahren darf zu jedem Zeitpunkt nur endlich viel Speicherplatz benötigen (Dynamische Finitheit, siehe Platzkomplexität).
4. Das Verfahren darf nur endlich viele Schritte benötigen (Terminierung, siehe auch Zeitkomplexität).
5. Der Algorithmus muss bei denselben Voraussetzungen das gleiche Ergebnis liefern.
6. Die nächste anzuwendende Regel im Verfahren ist zu jedem Zeitpunkt eindeutig definiert.

Eigenschaften von Algorithmen (2)

Statische Finitheit: Die Beschreibung eines Algorithmus darf nicht unendlich groß sein. Als statische Finitheit wird die Endlichkeit des Quelltextes bezeichnet. Der Quelltext darf nur eine begrenzte Anzahl, wenn auch bei Bedarf sehr viele Regeln enthalten.

Dynamische Finitheit: Zu jedem Zeitpunkt der Ausführung darf der von einem Algorithmus benötigte Speicherbedarf nur endlich groß sein. Andernfalls wäre der Algorithmus nicht ausführbar. Dies wird als dynamische Finitheit bezeichnet.

Terminierung: Algorithmen sind terminierend, wenn sie für jede mögliche Eingabe nach einer endlichen Zahl von Schritten zu einem Ergebnis kommen. Die tatsächliche Zahl der Schritte kann dabei beliebig groß sein. Steuerungssysteme und Betriebssysteme und auch viele Programme, die auf Interaktion mit dem Benutzer aufbauen, erfüllen diese Eigenschaft nicht

Beschreibung von Algorithmen

Algorithmen können auf unterschiedliche Art und Weise beschrieben werden. Gebräuchliche Beschreibungen sind:

- natürlichsprachliche Sätze (evtl. nicht formal genug, unterschiedliche Interpretationsfunktionen)
- Flußdiagramme (nicht mehr Stand der Technik)
- Struktogramme (nicht mehr Stand der Technik)
- UML
- Pseudo-Quellcode

Natürlichsprachliche Beschreibung

Der Euklidische Algorithmus, der bereits um 300 v. Chr. beschrieben wurde, dient zur Ermittlung des größten gemeinsamen Teilers (ggT) zweier natürlicher Zahlen A und B:

1. Sei A die Größere der beiden Zahlen A und B (entsprechend vertauschen, falls dies nicht bereits so ist)
2. Setze A auf den Wert $A - B$
3. Wenn A und B ungleich sind, dann fahre fort mit Schritt 1, wenn sie gleich sind, dann beende den Algorithmus: Diese Zahl ist der größte gemeinsame Teiler.

Beispiel Euklidische Algorithmus

Berechne den größten gemeinsamen Teiler von 14 und 8!

	A	B	$A - B$
1.	14	8	6
2.	8	6	2
3.	6	2	4
4.	4	2	2
5.	2	2	0

Das Ergebnis ist 2

Pseudo-Quellcode

Eine formalere Beschreibung als die Natürlichsprachliche ist eine Beschreibung des Algorithmus mit Hilfe von Pseudo-Quellcode. Pseudo-Quellcode ist dabei von der Syntax an existierende Programmiersprachen angelehnt. Zur formalen Definition unseres Pseudo-Quellcode müssen wir nun folgende Schritte durchführen:

- Festlegen der Syntax.
- Festlegung der Interpretationsfunktion.

Grundgerüst (1)

Jede Algorithmusbeschreibung $\mathcal{B}$ in Pseudo-Quellcode ist ein 3-Tupel $\mathcal{B} = (E, V, A)$, wobei die einzelnen Komponenten folgendermaßen definiert sind:

- $\mathcal{B}$ beinhaltet die komplette Algorithmusbeschreibung. $\mathcal{B}$ ist ebenfalls der Name der Beschreibung.
- E beschreibt die Menge der Eingabevariablen
- A beschreibt die Menge der Ergebnisvariablen (Ausgabevariablen)
- V enthält ein n-Tupel mit n Verarbeitungsschritten

Wir werden diese einzelnen Bestandteile (Name, Eingabevariablen, Ergebnisvariablen und Verarbeitungsschritte) zur besseren Übersicht **nicht** in Tupelschreibweise sondern tabellarisch darstellen.

Grundgerüst (2)

Name: Hier steht der Name des Algorithmus.

Eingabe: Hier sind alle Eingabevariablen aufgeführt. Zusätzlich können diese natürlichsprachlich erklärt und beschrieben werden.

Ausgabe: Hier sind alle Ergebnisvariablen aufgeführt. Zusätzlich können diese natürlichsprachlich erklärt und beschrieben werden.

Vorgehen: Hier steht der eigentliche Pseudo-Quellcode.

Pseudo-Quellcode: Variablen

Eine **Variable** ist eine Größe, die verschiedene Werte annehmen kann. Sie ist also in ihrer Größe **veränderlich** oder **variabel**. Variablen werden auch **Platzhalter** genannt. Der Vorrat an verschiedenen Variablen ist unendlich groß, in einem Algorithmus dürfen allerdings nur endlich viele verschiedene Variablen verwendet werden. Variablen werden mit einem Bezeichner gekennzeichnet. Dieser Bezeichner identifiziert jede Variable eindeutig. Der Bezeichner muss mit einem Buchstaben anfangen. Beispiele für Bezeichner sind:

`a`, `b`, `variable1`, `t_i`, `h_5`

Variablenzuweisung

Eine **Variable** ist eine Größe, die verschiedene Werte annehmen kann. Wie werden Werte an Variablen gebunden? Die Bindung geschieht mit Hilfe der Zuweisung $:=$. Ein Beispiel für eine Zuweisung eines Wertes an die Variable A ist:

$$\underbrace{A}_{\text{lhs}} := \underbrace{\sum_{i=0}^{6} i + B}_{\text{rhs}}$$

Bei der Zuweisung unterscheidet man zwischen linker Seite (**l**eft **h**and **s**ide) und rechter Seite (**r**ight **h**and **s**ide). Auf der linken Seite dürfen nur Variablen, auf der rechten Seite beliebige auswertbare Ausdrücke vorkommen.

parallele Variablenzuweisung

Mit Hilfe des Operators **,** können auf der linken Seite einer Zuweisung mehrere Variablen vorkommen.

$$A, B := B, A$$

Der Operator **,** realisiert hier eine **parallele Zuweisung**, da die einzelnen Zuweisungen $A := B$ und $B := A$ aus unserem obigen Beispiel parallel ausgeführt werden.

Allerdings unterstützen nur sehr wenige Programmiersprachen (z.B. PERL) eine solche parallele Zuweisung.

Hintereinanderausführung

Mit Hilfe des Semikolons ; wird eine Anweisungen abgeschlossen. Es kann benutzt werden, um mehrere Anweisungenen hintereinander auszuführen. Das Symbol ; trennt somit die einzelnen Anweisungsschritte.

```
BEGIN
   A := 2;
   B := 5;
   A,B := B,A;
END;
```

`BEGIN` und `END` fassen in Pseudocode Anweisungsschritte zu einem logischen Anweisungsschritt zusammen (vergl. auch die Programmiersprache PASCAL).

Bedingte Ausführung

```
IF (Prädikat) THEN
  Anweisung1;
ELSE
  Anweisung2;
```

`Anweisung1` wird genau dann ausgeführt, wenn das Prädikat erfüllt ist. Ist das Prädikat nicht erfüllt, wird `Anweisung2` ausgeführt. Verwende `BEGIN` und `END`, wenn mehrere Anweisungen ausgeführt werden müssen.

Wiederholte Ausführung

Für die wiederholte Ausführung von Anweisungen stehen in unserem Pseudo-Quellcode drei Konstrukte zur Verfügung:

- Führe die Anweisung n mal aus.
- Solange ein Prädikat erfüllt ist, führe die Anweisung aus.
- Führe die Anweisung solange aus, bis ein Prädikat erfüllt ist.

Wiederholte Ausführung (2)

Betrachten wir zuerst eine n-malige Ausführung einer Anweisung:

```
FOR index := start TO ende DO
  Anweisung;
```

Der Index wird von seinem Startwert solange verändert bis er den Endwert erreicht hat. Nach jeder Veränderung (und nur dann!) wird die Anweisung ausgeführt. Beispiele:

```
sum1 := 0;
FOR i := 1 TO 100 DO
  sum1 := sum1 + i;
sum2 := 0;
FOR i := 100 DOWNTO 1 DO
  sum2 := sum2 + i;
FOR i := 100 DOWNTO 200 DO
  sum2 := sum2 + i;
```

Wiederholte Ausführung (3)

Die sogenannte `FOR`-Schleife eignet sich nicht, wenn die Anzahl der Durchläufe von der Ausführung der Schleife abhängig ist, also noch nicht zu Beginn der Schleife feststeht. Um dennoch solche Schleifen realisieren zu können führen wir folgende Konstrukte ein:

```
WHILE Prädikat DO
   Anweisung;

REPEAT
   Anweisung 1;
   ...
   Anweisung n
UNTIL Prädikat;
```

Beispiel Pseudo-Quellcode

Name: Fibonacci

Eingabe: Es wird eine positive natürliche Zahl $n > 0$ als Eingabe verwendet.

Ausgabe: Die Ausgabe ist die positive natürliche Zahl f.

Beispiel Pseudo-Quellcode

Vorgehen:

```
fibminus1 := 1;
fibminus2 := 1;
f := 1;
IF n > 2 THEN
  FOR i := 3 TO n DO
    BEGIN
      f := fibminus1 + fibminus2;
      fibminus2 := fibminus1;
      fibminus1 := f;
    END;
```

Ausführungstabellen

Step	Aktuelle Anweisung	Folgende Anweisung	n	f	fibminus1	fibminus2	i
1.	fibminus1 := 1	fibminus2 := 1	3	?	1	?	?
2.	fibminus2 := 1	f := 1	3	?	1	1	?
3.	f := 1	IF $n > 2$ THEN	3	1	1	1	?
4.	IF $n > 2$ THEN	FOR i := 3 TO n DO	3	1	1	1	?
5.	FOR i := 3 TO n DO	f := fibminus1 + fibminus2	3	1	1	1	3
6.	f := fibminus1 + fibminus2	fibminus2 := fibminus1	3	2	1	1	3
7.	fibminus2 := fibminus1	fibminus1 := f	3	2	1	1	3
8.	fibminus1 := f	FOR i := 3 TO n DO	3	2	2	1	3
9.	FOR i := 3 TO n DO	Ende	3	2	2	1	3