

Algorithmen und Datenstrukturen

1. EINLEITUNG

Einleitung

Zu den Begriffen: Algorithmen und Datenstrukturen

„systematische Verarbeitung“

→ Eine eindeutige Beschreibung eines in mehreren Schritten durchgeführten Bearbeitungsvorgangs

„Ordnungsschema“

→ Eine Struktur zur Verwaltung von Daten
→ Darstellung von Informationen in maschinenverarbeitbarer Form
→ Charakterisieren Daten und mögliche Operationen auf Daten

Beispiele und informelle Definition

- Algorithmen im Alltag
 - ◆ Bedienungsanleitungen
 - ◆ Gebrauchsanleitungen
 - ◆ Bauanleitungen
 - ◆ Kochrezepte
 - ◆ Berechnungsvorschriften (z.B. Berechnung der Fakultät)

Intuitive Begriffserklärung:

„Ein Algorithmus ist eine *präzise* (d.h. in einer festgelegten Sprache formulierte), *endliche Beschreibung* eines allgemeinen Verfahrens unter Verwendung *ausführbarer elementarer Verarbeitungsschritte*.“

Annahme: Transformationelle Probleme



Ein Algorithmus definiert eine Transformation auf dem gesamten, durch die Eingaben definierten Zustand, aus dem als Bedeutung dann die Werte der Ausgabevariablen ausgelesen werden.

Das heißt, ein Algorithmus benutzt kein weiteres Wissen neben der Eingabe und hat keine Seiteneffekte!

Wichtige Eigenschaften von Algorithmen

Ein Algorithmus heißt ...

- **terminierend**, wenn er für alle zulässigen Schrittfolgen stets nach endlich vielen Schritten endet
- **deterministisch**, wenn in der Auswahl der Verarbeitungsschritte keine Freiheit besteht
- **determiniert**, wenn das Resultat eindeutig bestimmt ist
- **sequenziell**, wenn die Schritte stets hintereinander ausgeführt werden
- **parallel** oder **nebenläufig**, wenn gewisse Verarbeitungsschritte nebeneinander (im Prinzip gleichzeitig) ausgeführt werden
- **korrekt**, wenn das Resultat stets korrekt ist
- **effizient**, wenn das Resultat in „annehmbarer“ Zeit geliefert wird

Definitionen: Algorithmus, Programm, Prozessor

Algorithmus:

Ein Algorithmus ist ein allgemeines Verfahren zur Lösung eines Problems ohne Bezug auf einen konkreten Prozessor.

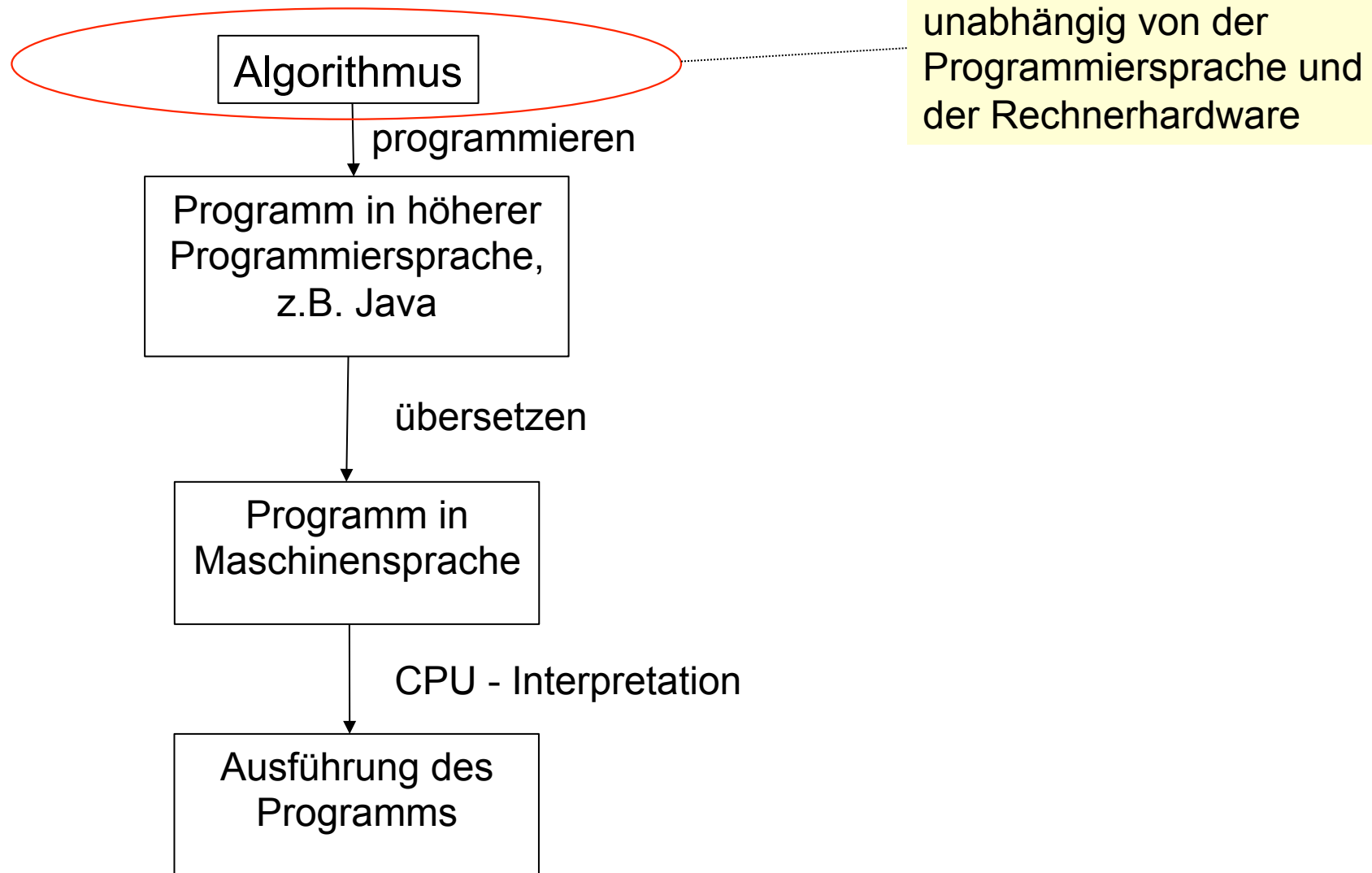
Programm:

Ein Programm ist eine konkrete Formulierung eines Algorithmus für eine konkrete Klasse von Prozessoren.

Prozessor:

Ein Prozessor ist etwas, das die Fähigkeit hat, Programme auszuführen.

Vom Algorithmus zur Programmausführung



Algorithmenentwurf

1. Hintergrundwissen erwerben
2. Problem definieren
 - Erfordert Hintergrundwissen
 - Erfordert Übung in der Definition von Problemen
3. Algorithmus entwerfen
 - Erfordert Wissen zu Algorithmen und Datenstrukturen
4. Programm erstellen
 - Erfordert Wissen über Programmiersprache (z.B. Java) und Programmierung
5. Lösung überprüfen
 - Erfordert methodisches Wissen zu Termination, Korrektheit, Effizienz

Schauen wir uns ein einfaches Beispiel an...

Größter gemeinsamer Teiler 1/2

Gegeben zwei positive natürliche Zahlen a und b , welche ist die größte positive natürliche Zahl x , so dass x sowohl a also auch b teilt und es keine positive natürliche Zahl x' gibt, so dass $x' > x$ und x' teilt sowohl a als auch b .

Alle Variablen bezeichnen natürliche Zahlen größer 0

$$x|a \Leftrightarrow \exists \alpha : a = \alpha \cdot x \text{ (x teilt a)}$$

$$x = ggT(a, b) \Leftrightarrow x|a \wedge x|b \wedge \forall y : (y|a \wedge y|b \Rightarrow y|x)$$

Anwendung: Kürzen: $\frac{52}{32} = \frac{13}{8}$

Größter gemeinsamer Teiler 2/2

Formalisierung einer Algorithmenanforderung:

Problem: Berechnung des ggTs (GGT)

Eingabe: zwei positive natürliche Zahlen a und b mit
 $a > b$

Ausgabe: $\text{ggT}(a, b)$

Verfahren von Euklid 1/4

Verfahren von Euklid (300 v. Chr.) für natürliche Zahlen:

1. $b \mid a \Rightarrow b = \text{ggT}(a, b)$
2. $\neg(b \mid a) \Rightarrow \text{ggT}(a, b) = \text{ggT}(b, a \% b)$

Modulo-Funktion %:

$$r = a \% b \Leftrightarrow 0 \leq r < b \quad \wedge \quad \exists \alpha: a = \alpha \cdot b + r$$

Beispiel:

$$\begin{aligned} \text{ggT}(46, 18) &= \text{ggT}(18, 10) \quad (\alpha=2, b=18, r=10) \\ &= \text{ggT}(10, 8) \quad \dots \\ &= \text{ggT}(8, 2) = 2 \end{aligned}$$

Verfahren von Euklid 2/4

Algorithmenidee:

Führe die Berechnung von $\text{ggT}(a,b)$ auf die Berechnung von $\text{ggT}(b, a \% b)$ zurück (oder direkt b falls $b \mid a$).

Verfahren von Euklid 3/4

Algorithmus in Pseudocode:

Algorithmus **euklid**

Eingabe: Ganze Zahlen a, b

Ausgabe: Ganze Zahl $c = \text{ggT}(a, b)$

Setze $r = a \% b$;

Falls $r = 0$ gib b zurück;

Ansonsten gib $\text{euklid}(b, r)$ zurück;

Verfahren von Euklid 4/4

Algorithmus in Java:

```
public int euklid(int a, int b){  
    int r = a % b;  
    if (r == 0)  
        return b;  
    else  
        return euklid(b,r);  
}
```

Analyse 1/4

- Ist unser Algorithmus ein *guter* Algorithmus?
- Wichtige Fragen:
 - Terminiert der Algorithmus?
 - Ist der Algorithmus korrekt?
 - Welche Laufzeit hat der Algorithmus?

Analyse 2/4

Theorem

Für positive natürliche Zahlen a und b mit $a > b$, terminiert der Algorithmus `euklid` nach endlich vielen Schritten.

Beweis

- (a) Falls $b \mid a$ terminiert der Algorithmus in einem Schritt.
- (b) Andernfalls wird ein Parameter der Algorithmus um mindestens den Wert 1 verringert und der Algorithmus rekursiv aufgerufen. Spätestens wenn ein Parameter den Wert 1 erreicht tritt Fall (a) ein und der Algorithmus terminiert. Für endliche Eingaben bedeutet dies eine endliche Laufzeit. $\square$

Was ist mit anderen Eingaben?

Analyse 3/4

Theorem

Der Algorithmus `euklid` löst das Problem GGT.

Beweis

Wir haben bereits festgestellt, dass für zwei positive natürliche Zahlen a, b gilt, dass $\text{ggT}(a,b)=b$ (falls $b|a$) und $\text{ggT}(a,b)=\text{ggT}(b,a\%b)$ (falls $b|a$ nicht gilt). Der Algorithmus `euklid` vollzieht genau diese Fallunterscheidung nach. $\square$

Analyse 4/4

Theorem

Für positive natürliche Zahlen a und b mit $a > b$, benötigt der Algorithmus `euklid` maximal $\max\{a, b\}$ viele rekursive Aufrufe.

Beweis

Wir haben bereits festgestellt, dass `euklid` stets terminiert, dass bei jedem Aufruf ein Parameter um mindestens den Wert 1 verringert wird und dass wenn der zweite (stets kleinere) Parameter den Wert 1 hat die Rekursion spätestens endet. Damit kann es maximal $\max\{a, b\}$ viele rekursive Aufrufe geben. $\square$

Anmerkung: Die obige Laufzeit ist nur eine grobe obere Abschätzung. Die tatsächliche *Worst-case*-Laufzeit ist $O(\log(ab))$ (mehr zur O-Notation später)

Vorbedingungen von Algorithmen 1/5

Vorbedingung zur Ausführung von $\text{ggT}(a,b)$ ist $a,b > 0$

Wie kann man dies sicherstellen?

- Optimistische Strategie
 - Man geht vom Erfülltsein der Bedingung aus
 - ♦ z.B. Clients bekannt und zuverlässig, z.B. bei Rekursion
- Pessimistische Strategie
 - Man überprüft die Bedingung bei jedem Aufruf
 - ♦ z.B. Öffentliche APIs
- Möglichkeiten bei nicht erfüllten Vorbedingungen
 - Ausnahmen werfen
 - Parameter auf Defaultwerte setzen (mit Meldung)
 - Programm nicht ausführen und Defaultwert zurückgeben

Vorbedingungen von Algorithmen 2/5

Lösung: rekursiv, optimistisch

```
public int ggT(int a, int b){  
    int r = a % b;  
    if (r == 0)  
        return b;  
    else  
        return ggT(b,r);  
}
```

Vorbedingungen von Algorithmen 3/5

Lösung: iterativ, pessimistisch – Version 1

```
public int ggt(int a, int b){  
    if (a<=0 || b<=0)  
        throw new ArithmeticError("negative Daten  
bei ggt("+a+", "+b+")");  
    else {  
        int r = a % b;  
        while (r!=0) {  
            a = b;  
            b = r;  
            r = a % b;  
        }  
        return b;  
    }  
}
```

Vorbedingungen von Algorithmen 4/5

Lösung: iterativ, pessimistisch – Version 2

```
public int ggT(int a, int b){  
    if (a<=0 || b<=0)  
        throw new ArithmeticError("negative Daten  
bei ggT("+a+", "+b+")");  
    else {  
        do{  
            int r = a % b;  
            a=b;  
            b=r;  
        } while (r!=0);  
    return a;  
    }  
}
```

Vorbedingungen von Algorithmen 5/5

Fazit

- Welche Strategie (optimistisch, pessimistisch) und welches Verhalten man bei nicht-erfüllten Vorbedingungen zeigt, hängt von vielen Faktoren ab:
 - Bei unkritischen oft aufzurufenden Algorithmen könnte die Überprüfung der Zulässigkeit zu viel Aufwand sein
 - Bei zeitintensiven Algorithmen kann durch eine Überprüfung Zeit gespart werden
- Man sollte das Verhalten seines Algorithmus im Fehlerfall aber stets gut dokumentieren!

Algorithmen und Datenstrukturen

1. EINLEITUNG

ZUSAMMENFASSUNG

Zusammenfassung

- Begriffsklärung
 - Algorithmus
 - Datenstruktur
- Eigenschaften von Algorithmen
- Algorithmenentwurf
- Beispiel: Größter Gemeinsamer Teiler