

Definition Algorithmus

M. Broy: Informatik: Eine grundlegende Einführung

„Ein Algorithmus ist ein Verfahren

- mit einer **präzisen** (d.h. in einer genau festgelegten Sprache abgefassten),
- **endlichen** Beschreibung,
- unter Verwendung
 - **effektiver** (d.h. tatsächlich ausführbarer),
 - **elementarer** (Verarbeitungs-) Schritte.“

→ wie beschreibt man Algorithmen?

Notizen

5

Wie beschreibt man Algorithmen?

Algorithmus: bestimme Maximum von zwei Zahlen

- Input: Zahlen a, b
- Output: Zahl $x = \max(a, b)$

Problem: präzise Beschreibung der Schritte

Notizen

6

Wie beschreibt man Algorithmen?

Algorithmus: bestimme Maximum von zwei Zahlen

- Input: Zahlen a, b
- Output: Zahl $x = \max(a, b)$

Problem: präzise Beschreibung der Schritte

Lösung: Pseudocode

```
Algorithmus: max(a,b)
Input:  a,b
      x=a
      Falls b>a dann
        x=b
      Ende Falls
Output:  x
```

Notizen

7

Darstellung von Algorithmen I

Pseudocode

- informelle Veranschaulichung von Algorithmus
- nicht von Rechner ausführbar
- nicht standardisiert

```
Algorithmus: max(a,b)
Input:  a,b
      x=a
      Falls b>a dann
        x=b
      Ende Falls
Output:  x
```

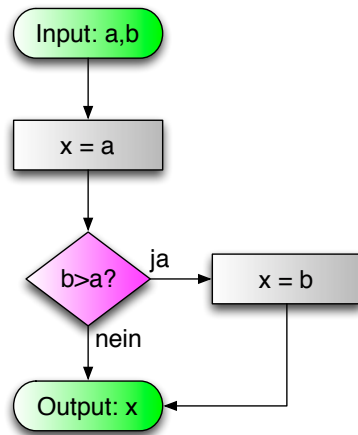
Notizen

8

Darstellung von Algorithmen II

Flussdiagramm

- graphische Darstellung als Ablaufdiagramm, nicht ausführbar
- normiert als DIN 66001



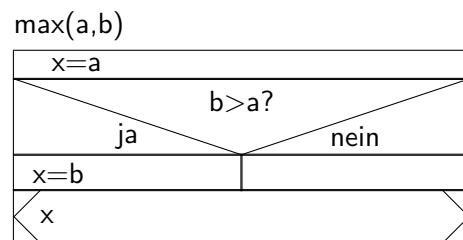
9

Notizen

Darstellung von Algorithmen III

Struktogramm

- Diagramm zur Strukturdarstellung, nicht ausführbar
- eingeführt von Nassi/Shneiderman 1973, normiert als DIN 66261



10

Notizen

Darstellung von Algorithmen IV

Programmiersprache

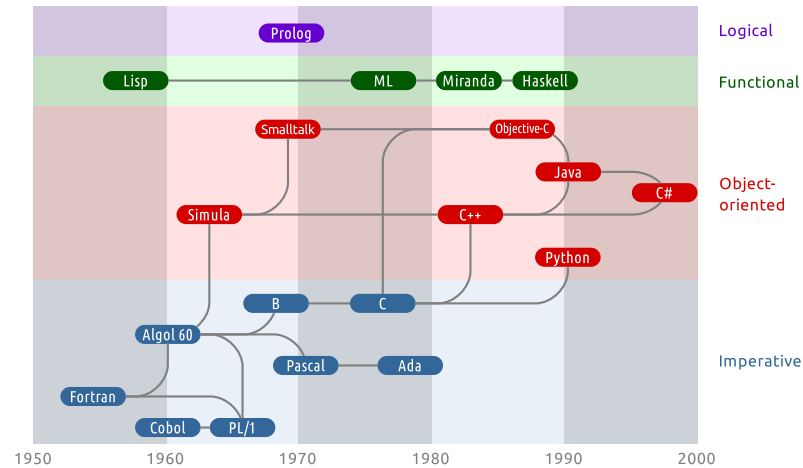
- formale Sprache zur Beschreibung von Algorithmen
- fest definierte Syntax
- ein Compiler/Interpreter wandelt Programm in ausführbare Form für Rechner um
- Beispiele: Assembler, C++, Java

- Algorithmus in C++:

```
int max(int a, int b)
{
    int x = a;
    if (b > a)
        x = b;
    return x;
}
```

Notizen

Programmiersprachen Übersicht



Grafik von Alexandru Dului.

Notizen

Äquivalenz von Algorithmen-Beschreibungen

Churchsche These

Alle „vernünftigen“ Definitionen von Algorithmen sind äquivalent.

- alle gängigen Programmiersprachen leisten dasselbe
- jeder Computer ist äquivalent

- formal: berechenbare Funktionen, formale Sprachen, Automaten, Turing-Maschinen

→ theoretische Informatik

Notizen

13

Euklidischer Algorithmus

Input: Natürliche Zahlen a, b

Output: $ggT(a, b)$

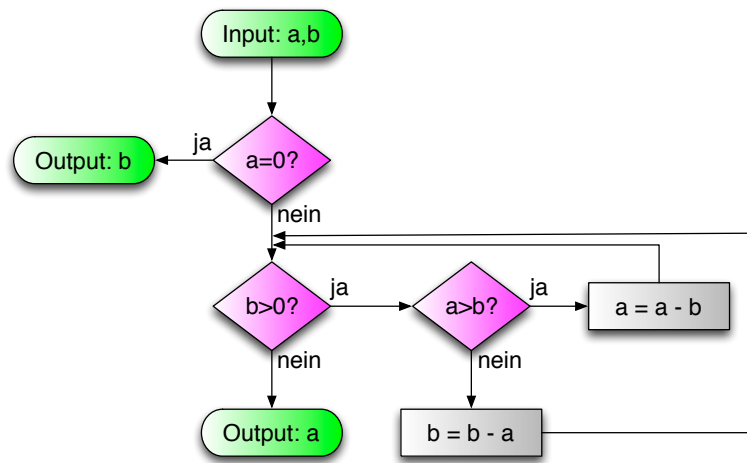
- 1 Falls $a = 0$ liefere b zurück
- 2 Solange $b > 0$ wiederhole
 - Falls $a > b$ setze $a = a - b$
 - sonst setze $b = b - a$
- 3 Liefere a zurück

→ das ist Pseudocode!

Notizen

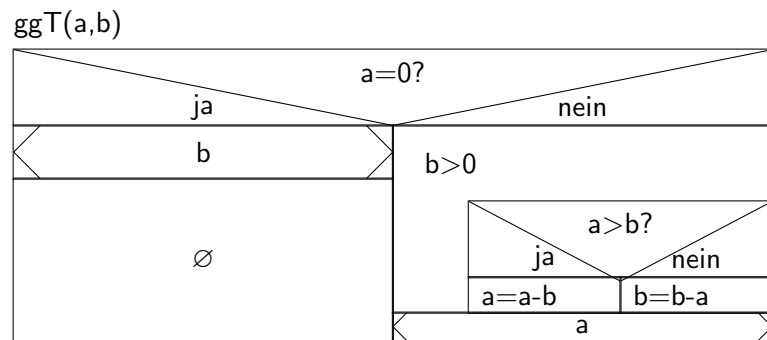
14

Euklidischer Algorithmus als Flussdiagramm



Notizen

Euklidischer Algorithmus als Struktogramm



Notizen

Euklidischer Algorithmus als Pseudocode

Nun mit Pseudocode-Konventionen der Vorlesung!

Input: natürliche Zahlen a, b

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

euklid(a,b):

```
if (a==0)
  return b;
while (b>0) { // Hauptschleife
  if (a>b)
    a = a - b;
  else
    b = b - a;
}
return a;
```

Notizen

17

Euklidischer Algorithmus als C++

```
int ggT(int a, int b)
{
  if (a==0)
    return b;
  while (b>0) {
    if (a>b)
      a = a - b;
    else
      b = b - a;
  }
  return a;
}
```

Notizen

18

Euklidischer Algorithmus als Python

```
def ggT(a, b):  
    if a == 0:  
        return b  
    while b > 0:  
        if a > b:  
            a = a - b  
        else:  
            b = b - a  
    return a
```

Notizen

Darstellung von Algorithmen in der Vorlesung

- viele Möglichkeiten der Darstellung!
 - alle vernünftigen Darstellungen sind äquivalent
 - jede Darstellung hat Vor- und Nachteile
- für die Vorlesung: [Pseudocode im C++ Stil](#)
- optionales Angebot: echter C++ Code
- Warum C++?
 - Industrie-Standard
 - moderne Sprache
 - letzter Standard: ISO C++11
 - ISO C++14 kurz vor Standardisierung
 - hardware-nahe und effizient
 - viele Programmiersprachen benutzen ähnliche Syntax

Notizen

Programm heute

① Einführung

② Grundlagen von Algorithmen

Darstellung von Algorithmen

Elementare Bausteine

Logische Ausdrücke

Notizen

21

Bausteine von Algorithmen

Elementare Bausteine

Alle berechenbaren Algorithmen lassen sich mit

vier elementaren Bausteinen

darstellen:

- ① Elementarer Verarbeitungsschritt (z.B. Zuweisung an Variable)
- ② Sequenz (elementare Schritte nacheinander)
- ③ Bedingter Verarbeitungsschritt (z.B. if/else)
- ④ Wiederholung (z.B. while-Schleife)

Notizen

22

1. Elementarer Verarbeitungsschritt

- typischer, einzelner Verarbeitungsschritt
- Beispiele:
 - `a = a - b // weist der Variable a den Wert a-b zu`
 - `return a // liefert den Wert von a zurück`
- **Achtung:** manche Verarbeitungsschritte sehen elementar aus, sind es aber nicht!
 - `sortiere Liste L // das ist nicht elementar!`
 - `finde kürzesten Pfad in Graph G // das ist nicht elementar!`

Notizen

23

2. Sequenz

- Sequenz ist eine **Aneinanderreihung** von elementaren Verarbeitungsschritten
- Abgrenzung der Schritte mittels **Semikolon (;)**
- Beispiel:
 - `x = 5; // Zuweisung von Wert 5 an Variable x`
 - `x = x + 2; // Wert von x ist nun 7`
- Um Ausnahmen zu vermeiden, wird Semikolon auch verwendet, wenn kein weiterer Schritt folgt

Notizen

24

3. Bedingter Verarbeitungsschritt

- Ausführung des Verarbeitungsschrittes nur wenn **Bedingung erfüllt** ist
- Beispiel:

```
if (a>b) // Bedingung wird in Klammern notiert  
    a = a - b;
```
- Weiteres Beispiel:

```
if (a>b)  
    a = a - b;  
else // falls Bedingung nicht erfüllt  
    b = b - a;
```
- **Einrückung** verdeutlicht logische Ebenen

Notizen

25

3. Bedingter Verarbeitungsschritt

- falls mehr als ein Verarbeitungsschritt bedingt ausgeführt werden soll, Markierung durch einen Block `{ ... }` mit geschweiften Klammern
- Beispiel:

```
if (x == 0) {  
    x = 5;  
    x = x + 2;  
} // if Block ist hier zu Ende  
else {  
    x = x - 1;  
} // else Block ist hier zu Ende
```
- auch einzelne Schritte können in einen Block gefasst werden

Notizen

26

4. Wiederholung

- wiederholte Ausführung von Verarbeitungsschritt/Block solange Bedingung erfüllt ist (auch **while Schleife** genannt)

- Beispiel:

```
while (x != 0) // Bedingung wird in Klammern notiert  
    x = x - 1;
```

- Weiteres Beispiel:

```
while (b > 0) { // Block für mehrere Schritte  
    if (a > b)  
        a = a - b;  
    else  
        b = b - a;  
} // while Block ist hier zu Ende
```

Notizen

27

4. Wiederholung

Es gibt auch andere Schleifentypen:

- **do-while Schleife:**

```
do {  
    x = x - 1;  
} while (x != 0); // Vorsicht, kann Endlosschleife sein!
```

- **for Schleife:**

```
for i=1 to 10  
    print(i); // gibt Wert von i aus
```

- Achtung, Syntax der **for Schleife** ist in C++ komplexer!

```
for (i=1; i <= 10; i++) // echte C++ Syntax  
    print(i);
```

Notizen

28

Bausteine von Algorithmen

Elementare Bausteine

Alle berechenbaren Algorithmen lassen sich mit vier elementaren Bausteinen darstellen:

- ① Elementarer Verarbeitungsschritt
- ② Sequenz
- ③ Bedingter Verarbeitungsschritt
- ④ Wiederholung

Notizen

29

Beispiel: Euklidischer Algorithmus

- Einrücken und geschweifte Klammern {, } kennzeichnen Blockstruktur

```
euklid(a,b):  
→ if (a==0)  
→→ return b;  
→ while (b>0) {  
→→ if (a>b)  
→→→ a = a - b;  
→→ else  
→→→ b = b - a;  
→ }  
→ return a; // == ggT(a,b)
```

Notizen

30

Beispiel: Euklidischer Algorithmus

- Einrücken und geschweifte Klammern {, } kennzeichnen **Blockstruktur**
- Sequenz durch Semikolon ;

```
euklid(a,b):  
  if (a==0)  
    return b;  
  while (b>0) {  
    if (a>b)  
      a = a - b;  
    else  
      b = b - a;  
  }  
return a; // == ggT(a,b)
```

Notizen

31

Beispiel: Euklidischer Algorithmus

- Einrücken und geschweifte Klammern {, } kennzeichnen **Blockstruktur**
- Sequenz durch Semikolon ;
- // bedeutet Rest der Zeile ist Kommentar (wird nicht ausgeführt)

```
euklid(a,b):  
  if (a==0)  
    return b;  
  while (b>0) {  
    if (a>b)  
      a = a - b;  
    else  
      b = b - a;  
  }  
return a; // == ggT(a,b)
```

Notizen

32

Beispiel: Fibonacci Zahlen

Fibonacci Folge

Die **Fibonacci Folge** ist eine Folge natürlicher Zahlen f_1, f_2, f_3, \dots , für die gilt

$$f_n = f_{n-1} + f_{n-2} \quad \text{für } n \geq 3$$

mit Anfangswerten $f_1 = 1, f_2 = 1$.



- eingesetzt von Leonardo Fibonacci zur Beschreibung von Wachstum einer Kaninchenpopulation
- Folge lautet: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
- berechenbar z.B. via Rekursion

33

Notizen

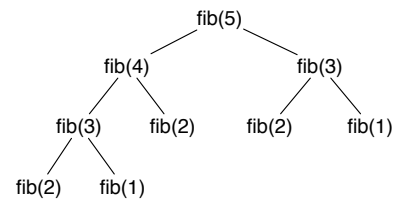
Beispiel: Fibonacci Funktion

Input: Index n der Fibonacci Folge

Output: Wert f_n

```
fib(n):  
  if (n == 1 || n == 2) {  
    return 1;  
  }  
  else {  
    // rekursiver Aufruf von fib()  
    return fib(n - 1) + fib(n - 2);  
  }
```

Aufrufstruktur für fib(5):



34

Notizen

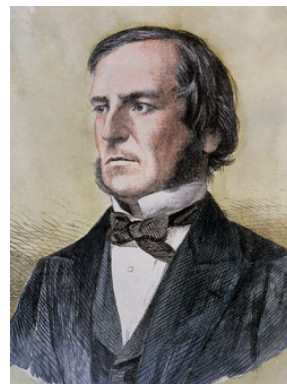
Programm heute

- ① Einführung

- ② Grundlagen von Algorithmen
 - Darstellung von Algorithmen
 - Elementare Bausteine
 - Logische Ausdrücke

Notizen

George Boole



Englischer Mathematiker (1815-1864)

Notizen

Logische Werte

Boolesche Logik: Logik mit zwei Werten

- Repräsentationen:
 - 1 und 0
 - W und F (in Englisch: T und F)
 - L und O
- Mengensymbol \mathbb{B}
 - $\mathbb{B} = \{0, 1\} = \{F, W\} = \{O, L\}$

Notizen

37

Logische Werte und Verknüpfungen

„Grundrechenarten“ mit logischen Werten:

- **Konjunktion:** $\wedge : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$
 - ähnlich zu Multiplikation bei Zahlen
 - auch bezeichnet als **UND** bzw. **AND**
- **Disjunktion:** $\vee : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$
 - ähnlich zu Addition bei Zahlen
 - auch bezeichnet als **ODER** bzw. **OR**
- **Negation:** $\neg : \mathbb{B} \rightarrow \mathbb{B}$
 - auch bezeichnet als **NICHT** bzw. **NOT**

Wahrheitstabelle:

a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

a	$\neg a$
0	1
1	0

Notizen

38

Weitere Verknüpfungen I

- **NAND:** $\uparrow: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$
 - $a \uparrow b = \neg(a \wedge b)$
 - mit NAND lassen sich NOT, OR, AND erzeugen
- **NOR:** $\downarrow: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$
 - $a \downarrow b = \neg(a \vee b)$
 - mit NOR lassen sich ebenso NOT, OR, AND erzeugen
- **XOR:** $\oplus: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$
 - auch **exklusiv oder** genannt
 - erzeugbar aus $\neg(a \wedge b) \wedge (a \vee b)$ (siehe Übung)

Wahrheitstabelle:

a	b	$a \uparrow b$
0	0	1
0	1	1
1	0	1
1	1	0

a	b	$a \downarrow b$
0	0	1
0	1	0
1	0	0
1	1	0

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Notizen

39

Weitere Verknüpfungen II

- **Implikation:** $\Rightarrow: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$
 - oft verwendet für mathematische Sätze: „ a impliziert b “, „aus a folgt b “
 - Beispiel: „aus $n < 3$ folgt $n < 5$ “
 - erzeugbar aus $\neg a \vee b$
- **Äquivalenz:** $\Leftrightarrow: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$
 - oft verwendet für mathematische Sätze: „ a gilt genau dann, wenn b gilt“, „ a und b sind äquivalent“
 - Beispiel: „ f ist bijektiv genau dann, wenn f injektiv und surjektiv ist“
 - erzeugbar aus $(a \wedge b) \vee (\neg a \wedge \neg b)$

Wahrheitstabelle:

a	b	$a \Rightarrow b$
0	0	1
0	1	1
1	0	0
1	1	1

a	b	$a \Leftrightarrow b$
0	0	1
0	1	0
1	0	0
1	1	1

Notizen

40

Rangfolge und Rechenregeln

Rangfolge:

- NICHT vor UND
- UND vor ODER

Beispiel:

$$\neg 0 \vee 1 \wedge 0 = (\neg 0) \vee (1 \wedge 0) = 1 \vee 0 = 1$$

De Morgan-Gesetze:

- $\neg(a \wedge b) = \neg a \vee \neg b$
- $\neg(a \vee b) = \neg a \wedge \neg b$

Notizen

41

Logische Ausdrücke in Pseudocode und C++

- logische Variablen: `bool a,b;`
- logische Werte: `true` und `false`
- NOT Operator: `!a`
- AND Operator: `a && b`
- OR Operator: `a || b`

Beispiele:

- `((2 == 2) && (3 < 1))`
ergibt `(true && false)`, also `false`
- `(!(2 == 2) || (3 > 1))`
ergibt `(false || true)`, also `true`
- Kurzform für `!(2 == 2)` ist `(2 != 2)`

Notizen

42

