Algorithmen und Datenstrukturen

Theoretische Grundlagen der Informatik

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

19. April 2024

Grundlagen

- aus der Theoretischen Informatik (Kurs Aachen)
  - Formale Sprachen
  - Strukturelle Induktion
  - Automaten
    endliche Automaten und ein wenig über Kellerautomaten / Turing Maschinen
  - Ein wenig Berechenbarkeitstheorie

- aus der diskreten Mathematik (Kurs Aachen)
  - Kongruenzen, O-Notation
  - Diskrete Kombinatorik
  - Lösen von Rekursionsgleichungen

- aus der praktischen Informatik
  - Elementare Datentypen
  - Darstellung von Algorithmen

Wozu?

Unter anderem zur Beantwortung folgender Fragen:

- Was ist eigentlich ein Algorithmus und wie kann ich ihn darstellen?
- Wie kodiert man komplexe Daten, wenn der Computer nur Nullen und Einsen kennt?
- Was kann ein Computer eigentlich alles berechnen? Gibt es Dinge, die er **nicht** berechnen kann?
- Wie gut ist mein Algorithmus? Woran mache ich "gut" eigentlich fest?

Algorithmen und Datenstrukturen

Algorithmus: Definition und Darstellung

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

19. April 2024

Definition: Algorithmus

Wir starten mit dem Versuch einer *intuitiven* Definition:

Definition 1.1 (Algorithmus)

Eine **Verarbeitungsvorschrift**, die angibt wie Eingabedaten schrittweise in Ausgabedaten umgewandelt werden.

Etymologie

Der Begriff geht zurück auf Abu Abdallah Muhammad ibn Musa al-Kwarizmi

- Persischer Mathematiker (ca. 780-850)
- Sein Lehrbuch "Über das Rechnen mit indischen Ziffern" beginnt in der lateinischen Übersetzung mit "*Dixit Algorismi*"

Fragen rund um Algorithmen

Definition 1.1 (Algorithmus)

Eine **Verarbeitungsvorschrift**, die angibt wie Eingabedaten schrittweise in Ausgabedaten umgewandelt werden.

Diese Definition lässt viele Fragen offen; z.B.:

- Wie sieht eine Verarbeitungsvorschrift aus?
- Was bedeutet “schrittweise”?
- Wer arbeitet diese Arbeitsvorschrift ab?
- Was genau sind Ein- und Ausgabedaten?

Darstellung von Algorithmen

Viele gebräuchliche Notationen

- Verbal / textuell
- Grafische Ansätze; z.B.:
  - Flussdiagramme
  - Nassi-Shneiderman-Diagramme
- Formale Ansätze; z.B.:
  - RAM-Code
  - Endliche Automaten und Turing-Maschinen
  - Code in konkreter Programmiersprache (z.B. JAVA, C++, Haskell, Prolog)

Beispiel: Divisionsrest

Wir betrachten ein einfaches Beispiel, für das wir einen Algorithmus in den unterschiedlichen Notationen beschreiben:

Beispiel 1.1 (Divisionsrest)

Gegeben seien zwei ganze Zahlen a und b mit $a \geq 0$ und $b > 0$. Wir suchen eine ganze Zahl r, so dass gilt:

$$a = \left\lfloor \frac{a}{b} \right\rfloor \cdot b + r$$

Division und $\lfloor \cdot \rfloor$ stehen nicht zur Verfügung.

Gauss-Klammern

Sei $x \in \mathbb{R}$, dann

- $\lfloor x \rfloor := max\{z \in \mathbb{Z} \mid z \leq x\}$ (* Abrunden *)
- $\lceil x \rceil := min\{z \in \mathbb{Z} \mid z \geq x\}$ (* Aufrunden *)
- $[x] := \lfloor x + 0.5 \rfloor$ oder $[x] := \lceil x + 0.5 \rceil$ (für $x \geq 0$) (* Kaufmännisches Runden *)

Verbale / textuelle Beschreibung

- Verarbeitungsvorschrift in normaler Sprache
- möglichst genau, eindeutig und vollständig
- Offenbar **nicht unproblematisch**
  - Internationalität (sprechen Sie Japanisch?)
  - Ausdruckskraft
    - sprachspezifisch (z.B. Taramuhara: kein Unterschied zwischen grün und blau)
    - sprachunspezifisch (z.B. "Ich sortiere die Eingabe,...")
  - mangelnde Präzision
    - z.B. Zähle die Vorkommen von "Geld" in "Geldsäcke haben viel Geld".

Lösung 1.1 (Algorithmus 'mod': textuelle Beschreibung)

Ich erzeuge das Ergebnis für r in einer Variablen x. Am Anfang hat x den Wert a. Solange x größer oder gleich b ist, ziehe ich b von x ab. War x kleiner als b, so bin ich sofort fertig.

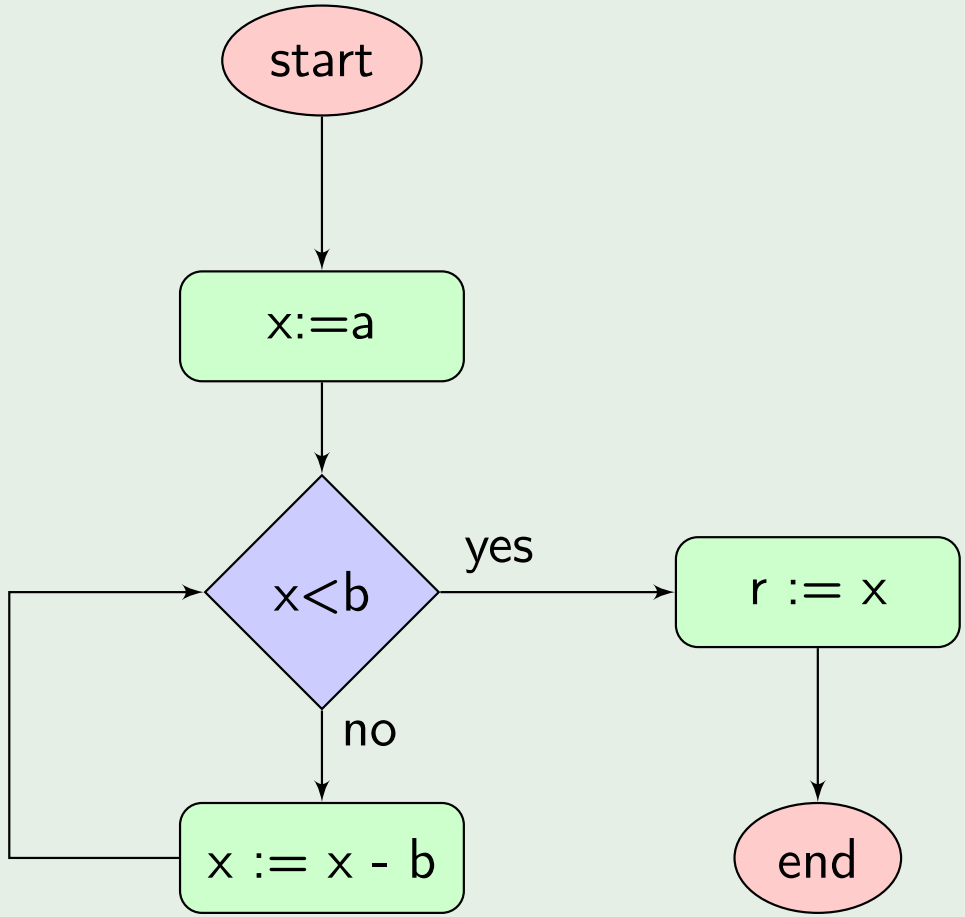
Flussdiagramme

- auch **Programmablaufplan** (kurz: PAP) genannt
- standardisiert in DIN 66001
- finden nicht nur in der Informatik Anwendung
  - z.B. Business Process Model Notation (BPMN) zur Definition von Prozessen in der Wirtschaft
- Moderne Form ist **Aktivitätsdiagramm** aus der *Unified Modeling Language* (UML)
  - z.B. auch Unterstützung von nebenläufigen Systemen (*Threads*)

Flussdiagramm für 'mod'

Lösung 1.2 (Algorithmus 'mod': Flussdiagramm)

Lösung der Divionsrest-Aufgabe als Flussdiagramm:

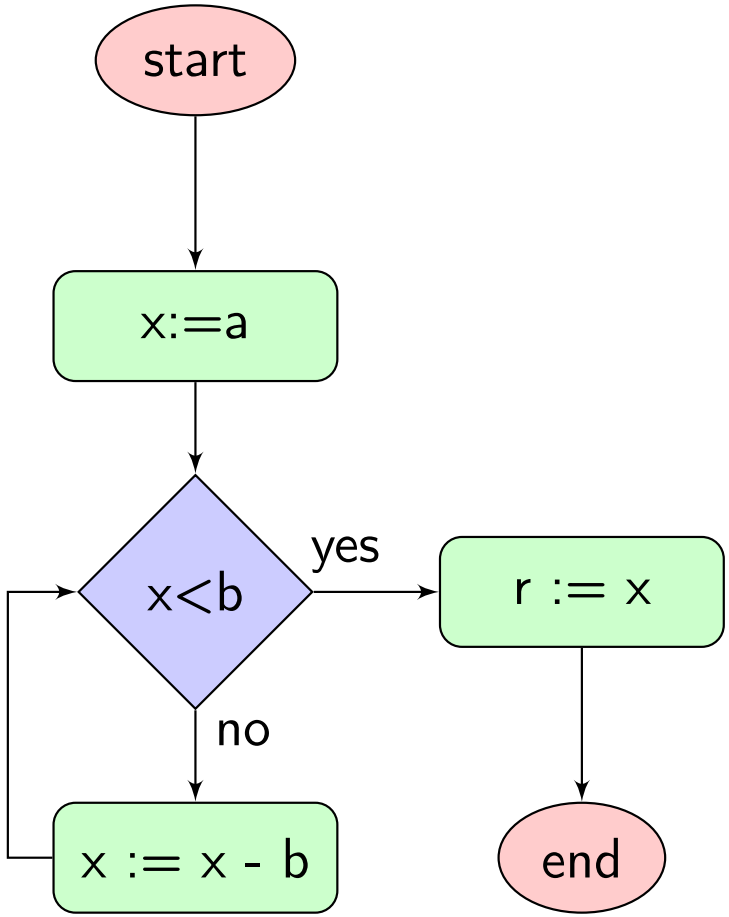


Flussdiagramm: Testlauf

Betrachten wir den Programmlauf am Beispiel $f(11, 3)$:

Anm.: Die Tabelle zeigt Werte bei Eintritt in die Bedingung

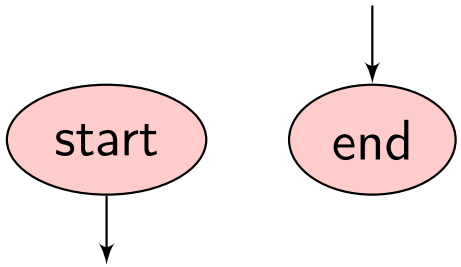
a	b	x	$x < b$
11	3	11	`false`
11	3	8	`false`
11	3	5	`false`
11	3	2	`true`



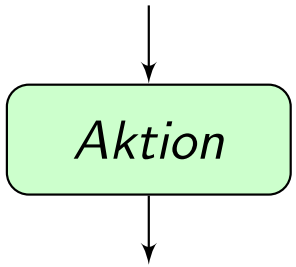
Flussdiagramme / Elemente

Die gängigsten Elemente haben wir bereits verwendet:

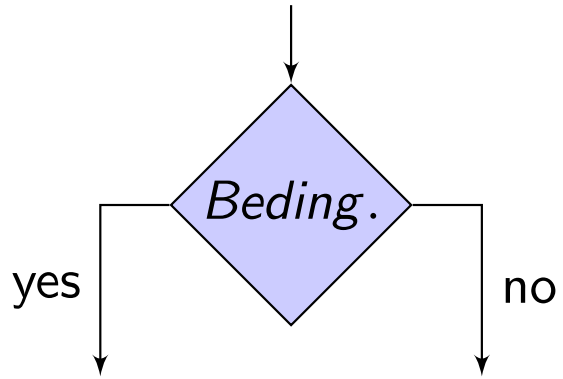
- **Beginn** und **Ende**



- **Aktionen**



- **Verzweigung**



Keine Symbole für Schleifen - `GOTO`-Semantik

Nassi-Shneiderman-Diagramme

- 1972/73 von Isaac Nassi und Ben Shneiderman entwickelt (**strukturierte Programmierung**)
- **Idee**: Verzicht auf "Pfeile", die dem ungeliebten `GOTO` entsprechen
- Standardisiert in DIN 66261 (Sinnbilder für Struktogramme nach Nassi-Shneiderman)

Isaac Nassi

Ben Shneiderman

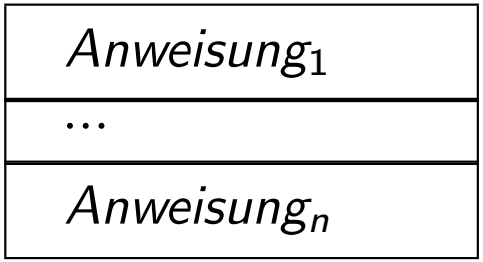
Lösung 1.3 (Algorithmus 'mod': Nassi-Shneiderman-Diagramm)

Lösung der Divionsrest-Aufgabe als Nassi-Shneiderman-Diagramm:

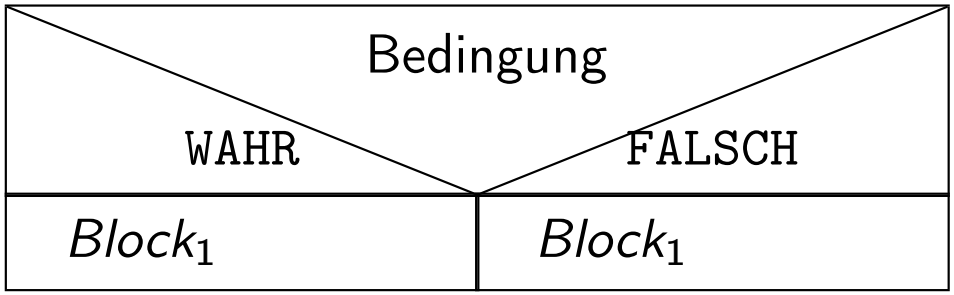
```
x := a
so lange x >= b
    x := x - b
r := x
```

Nassi-Shneiderman-Diagramme / Elemente 1

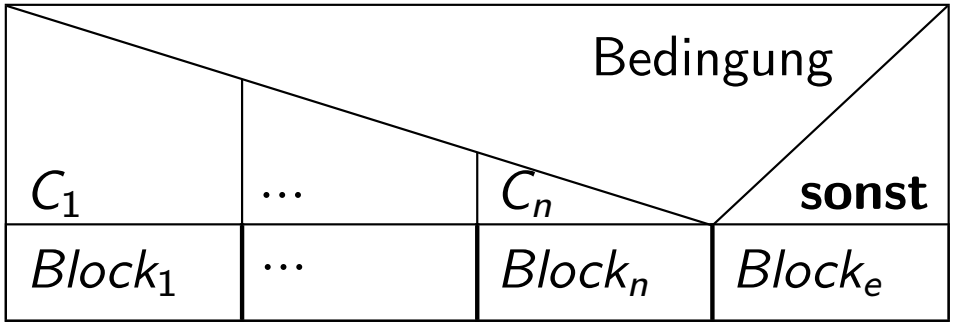
- **Sequenzen** von Anweisungen



- **Einfache Selektion** (`if` / `then` / `else`)

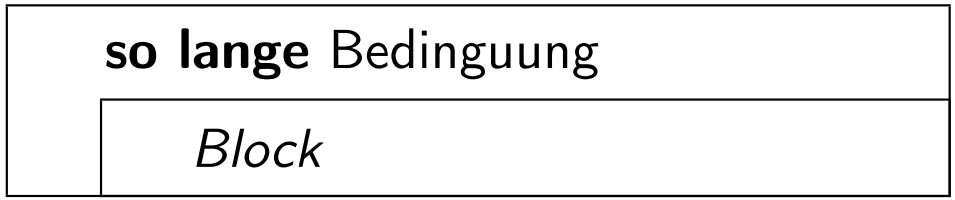


- **Mehrfachselektion** (`switch` / `case`)

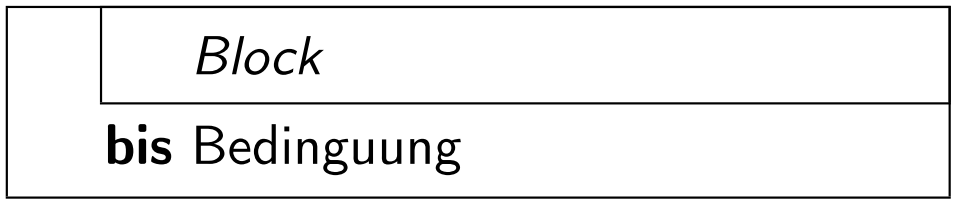


Nassi-Shneiderman-Diagramme / Elemente 2

- `while`-**Schleifen** (Schleifen mit Eintritts-Bedingung)



- `repeat`-**Schleifen** (Schleifen mit Ausgangs-Bedingung)



- **Endlos-Schleife**

Pseudo-Code

Angabe eines Algorithmus in einer Pseudo-Programmiersprache

- man verwendet i.d.R. allgemein bekannten Konstrukte
  - eventuell auch in Landessprache
- auch erlaubt: eingeschobene, verbale Beschreibungen
  - z.B. “x := Fläche des Polygons G”;

Lösung 1.4 (Algorithmus ’mod’: Pseudo-Code)

Zwei mögliche Pseudo-Codes:

```
// Wie Flussdiagramm
1: x:=a;
2: if (x<b) goto 5;
3: x:=x-b;
4: goto 2;
5: r:=x;
```

```
// Wie Nassi-Shneiderman
x:=a;
so lange (x<b)
begin
  x:=x-b;
end
r:=x;
```

Vor- und Nachbedingung

Aufgabenstellung definiert Vor- und Nachbedingung für Algorithmus:

- **Vorbedingung**: $a \geq 0$ und $b > 0$ (* Eingabe gültig *)
- **Nachbedingung**: $x = A - \left\lfloor \frac{A}{B} \right\rfloor \cdot B$ (* Ergebnis korrekt *)

Werden oft in Code / Diagramm als "Assertions" umgesetzt / beschrieben:

Beispiel 1.2 (JAVA: Methode mit Assertions)

```
/* Assertions aktivieren: z.B. "java -ea MyClass" */
public static int mod(int a,int b) {
  /* Vorbedingung: */
  assert ((a>=0)&&(b>0)) : "Vorbedingung verletzt";
  int x=a;
  while (x>=b)
    x=x-b;
  /* Nachbedingung */
  assert (x==a-(a/b)*b) : "Nachbedingung verletzt";
  return x;
}
```

Einordnung der textuellen und grafischen Ansätze

- **Zum Verständnis eventuell ausreichend**
- Alle Ansätze erlauben Umgehen von Präzision
  - verbale / textuelle Darstellung: zu viele Freiheitsgrade (s.o.)
  - Grafiken: strukturierter, aber: Anweisungen bzw. Aktionen mit zu vielen Freiheitsgraden in der Formulierung

! **Problem:** Was sind **elementare** Operationen?

- **Analyse von Algorithmen mit formalen Methoden schwer / gar nicht möglich**

! **Wir brauchen formales Beschreibungswerkzeug für Algorithmen**