Algorithmen und Datenstrukturen

Praktische Einführung und Programmierung

Stefan Bosse

Universität Koblenz - FB Informatik

Darstellung von Algorithmen

Diagramme und Pseudonotation

1. Flussdiagramme: Beschreiben das Verhalten von Algorithmen durch Abläufe;
2. Struktugramme: Beschreibt strukturelle Eigenschaften von Algorithmen;
3. Pseudonotation: Teils Verhaltens, teils strukturelle Beschreibung.

Darstellungsformen von Algorithmen

Um Algorithmen allgemeinverständlich auszudrücken, ohne den Adressatenkreis auf eine Programmiersprache einzugrenzen, werden häufig Programmiersprachen-unabhängige Ausdrucksformen gewählt. Neben

- dem Programmablaufplan (PAP, DIN 66001),
- dem Nassi-Shneiderman-Diagramm (“Struktogramm”, DIN 66261),
- oder dem UML-Aktivitäts-Diagramm (siehe gesonderter Artikel)

ist dies v.a. auch eine natürlichsprachlich vereinfachte und syntaxfreie Form des Programmcodes - der Pseudocode.

Geläufige Pseudocode-Notationen

Für Pseudocode gibt es kein festes einheitliches Regelwerk, vielmehr muss eine für den Adressatenkreis passende Variante gewählt werden: Häufig wird eine geläufige Programmiersprache vereinfacht.

Beispielhaft werden hier drei Varianten vorgestellt, die häufig eingesetzt werden:

- JANA: Insbesondere für Entwickler*innen mit Java/C/C++/C#-Kenntnissen leicht verständliche Notation (Java-Based Abstract Notation for Algorithms)
- Pseudo-Pascal: An die Programmiersprache Pascal angelehnte Variante (v.a. in älteren Lehrwerken verbreitet)
- deutsches Pseudo-Pascal (eigene Namensschöpfung): eine eingedeutschte Pseudo-Pascal-Variante, die vor allem im Bereich der Fachinformatiker*innen-Ausbildung verwendet wird (da die IHK sie in den Prüfungen und Beispiellösungen verwendet).

Alle diese Varianten lassen sich mit etwas Übung gleichwertig nutzen. Ungeübt ist es empfehlenswert, nahe an einer vertrauten Programmiersprache zu bleiben. Man sollte dann allerdings aufpassen, dass man auf den Einsatz von Sprachkonstrukten verzichtet, die es nur in bestimmten Programmiersprachen gibt (z.B. Lambda-Ausdrücke, Annotations, Dekorator, bestimmte Iterations-Bedingungsabkürzungen).

- Auch der Einsatz von "gsprochener" Sprache ist möglich, d.h., die Beschreibung eines komplexen Ablaufs in Sätzen, wie z.B. "Iterative Berechnung von f(x) mit variabler Schrittweite"

Wesentliche Eigenschaften von Pseudocode

Wichtig bei der Erstellung von Pseudocode ist:

- Verständlichkeit des Pseudocodes ist wichtiger als starre Konventionen!
- Unnötige Details vermeiden: das Offensichtliche kann als bekannt vorausgesetzt werden!
- Im Kontext bleiben: Wer ist Adressat und in welcher Problemdomäne befindet man sich? Wem beantworte man mit dem Pseudocode welche Frage?
- Stimmige Abstraktionsgrade wählen: Baut der Abstraktionsgrad einer Operationen auf den Abstraktionsgrad des umgebenden Moduls auf? Das Single Layer of Abstraction (SLA)-Prinzip gilt auch für Pseudocode!

Notationen für Algorithmen

Die derzeit verwendeten „halbformalen“ Mittel zur Beschreibung von Algorithmen benutzen die Grundelemente, die man in den meisten (höheren) Programmiersprachen findet. Dabei gibt es an C angelehnte verbale Beschreibungen (oft „Pseudocode“ genannt) und graphische Beschreibungen (Nassi-Shneiderman Diagramme oder Programmablaufpläne).

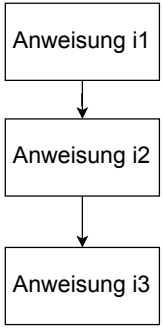
- Wir legen fest: Jeder Algorithmus soll nur mit Hilfe der folgenden Bausteine konstruiert werden (Stichwort: „strukturierte Programmierung“).

Statt Jana wird rein prozedurales JavaScript als Notation verwendet.

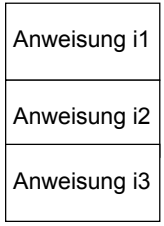
Sequenz

Die einzelnen Teilalgorithmen werden hintereinander in der Reihenfolge ausgeführt, in der sie aufgeschrieben sind:

Flussdiagramm



Struktugramm



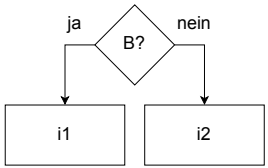
Pseudonotation

```
i1; i2; i3
{ i1; i2; i3 }
{
  i1
  i2
  i3
}
```

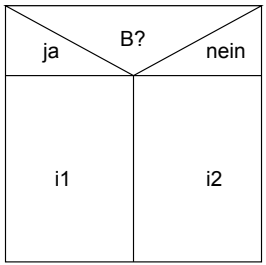
Verzweigung

Wenn die Bedingung B erfüllt ist, so wird der "ja-Teilalgorithmus“ ausgeführt, sonst (also wenn die Bedingung nicht erfüllt ist) der nein-Teilalgorithmus. Es werden nie beide Teilalgorithmen ausgeführt:

Flussdiagramm



Struktugramm



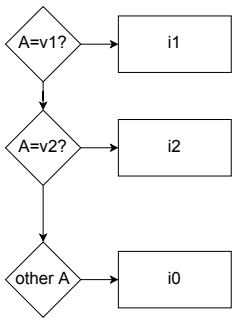
Pseudonotation

```
if (B) I1
if (B) I1
  else I2
```

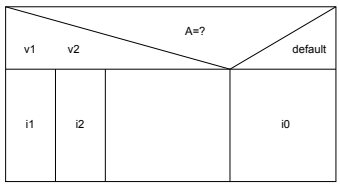
Fallunterscheidung

Eine Spezialform der Verzweigung ist die Fallunterscheidung. Wenn die Auswertung des Ausdrucks A den Wert i ergibt, dann wird der Teilalgorithmus I ausgeführt. Wenn kein passender Teilalgorithmus I existiert, dann wird der Teilalgorithmus "default" ausgeführt:

Flussdiagramm



Struktugramm



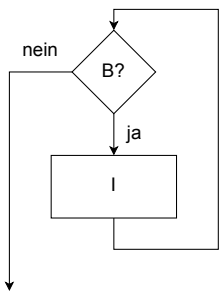
Pseudonotation

```
switch (A) {
  case V1: I1
  case V2: I2
  default: I0
}
```

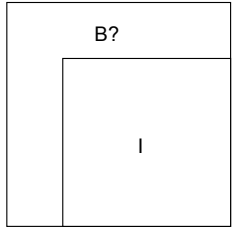
Abweisende Schleife

Hier wird der Teilalgorithmus S (= Scheifenkörper / Schleifenrumpf) solange ausgeführt, wie die Bedingung B erfüllt ist. Der Name abweisende Schleife kommt daher, dass die Bedingung zu Beginn getestet und der Schleifenrumpf evtl. nicht ausgeführt wird:

Flussdiagramm



Struktugramm



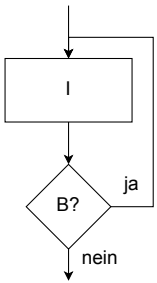
Pseudonotation

```
while (B) {
  I
}
```

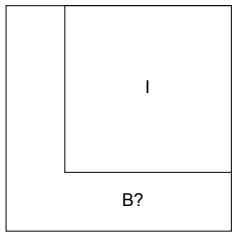
Nichtabweisende Schleife

Hier wird erst der Teilalgorithmus S (= Scheifenkörper / Schleifenrumpf) ausgeführt und dann die Bedingung B geprüft. Damit ist auch die Bezeichnung nichtabweisende Schleife plausibel, denn der Teilalgorithmus S wird auf jeden Fall einmal ausgeführt:

Flussdiagramm



Struktugramm



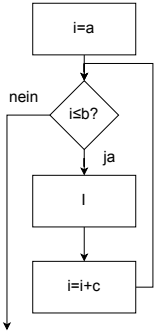
Pseudonotation

```
do {
  I
} while (B)
```

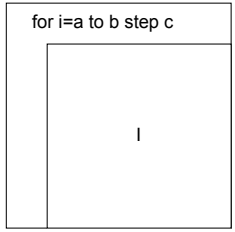
Zählschleife

Bei der Zählschleife wird der Schleifenindex i vom Startwert hin zum Endwert inkrementiert. Dabei wird der Teilalgorithmus S (= Scheifenkörper) jedesmal durchlaufen. Der Unterschied zur abweisenden und nichtabweisende Schleife ist, dass schon zu Beginn die Anzahl der Schleifendurchläufe fest steht:

Flussdiagramm



Struktugramm



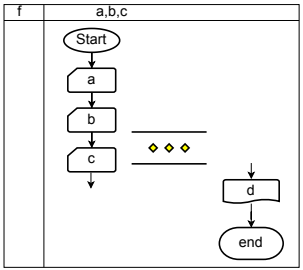
Pseudonotation

```
for(i=a;i<=b;i=i+c) {
  I
}
```

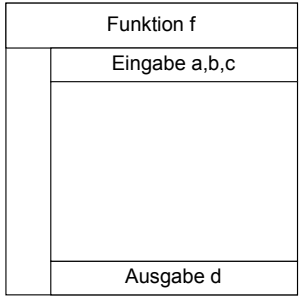
Funktion

Funktionen sind benannte Anweisungsblöcke mit Ein- und Ausgabeparametern:

Flussdiagramm



Struktugramm



Pseudonotation

```
function f (a,b,c) {
  I
  return d
}
X=f(V1,V2,V3)
```

Datenstrukturen (Records)

Java

```
class Data {
  type1 el1;
  type2 el2;
  void Data(type1 v1,type2 v2,..) {
    el1=v1;
    el2=v2;
    ...
  }
  public foo() {
    this...
  }
  ...
}
Data data = new Data(v1,v2,..);
data.el1 = ..
.. = data.el2
data.foo(..)..
```

Pseudonotation

```
{
  el1:v1,
  el2:v2,
  ..
}
function Data(v1,v2) {
  return {
    el1:v1,
    el2:v2,
    ..
  }
}
function foo(data) {

}
data=Data(v1,v2,..)
data.el1 = .. ; .. =  data.el2 ..
```

Beispiel Flussdiagramm und Struktugramm

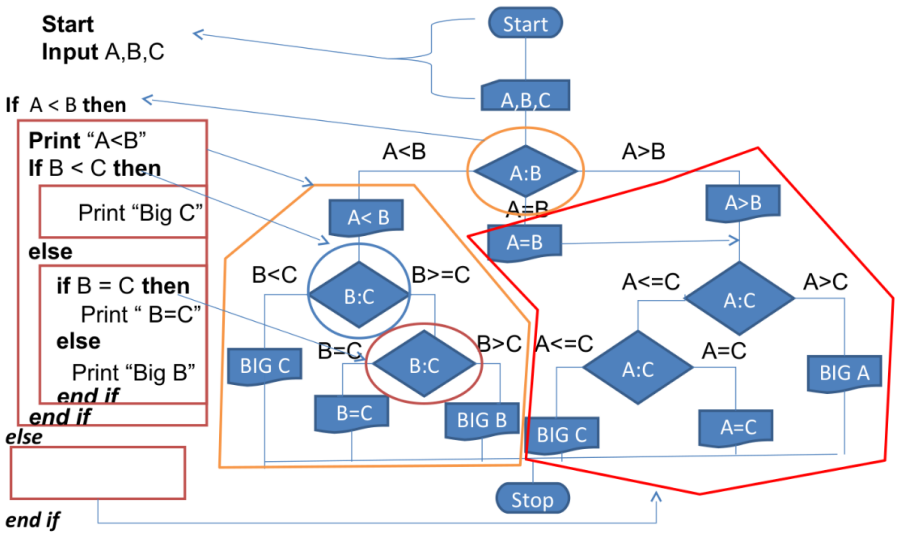


Abb. 1. Max(A,B,C) Berechnung

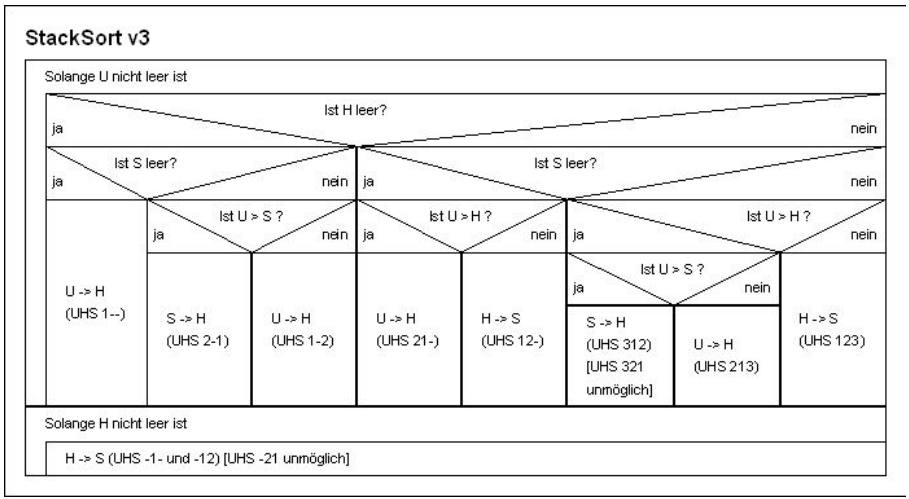
Beispiel Pseudonotation

```
function max(a,b,c) {
  if (a>b) {
    if (a>c) return a
    else     return c
  } else if (a<b) {
    if (b>c) return b
    else     return c
  }
}
print(max(10,34,12))
```

Beispiel Ausführbar

Beispiel Unübersichtlich

Struktugramme werden schnell unübersichtlich (ebenso Flussdiagramme)



Mengen und Mengenoperationen

Häufig werden in der Algorithmik Mengen von Werten verarbeitet. Z.B. alle Elemente eines Arrays oder einer Liste. Hier bietet sich die Kurznotation der Mengenlehre (Mathematik) an.

Es ist sehr schwer den fundamentalen Begriff der Menge mathematisch exakt zu definieren. Aus diesem Grund soll uns hier die von Cantor im Jahr 1895 gegebene Erklärung genügen, da sie für unsere Zwecke völlig ausreichend ist:

(Georg Cantor) Unter einer Menge verstehen wir jede Zusammenfassung M von bestimmten wohl unterschiedenen Objekten m unsrer Anschauung oder unseres Denkens (welche die Elemente von M genannt werden) zu einem Ganzen.

Def. 1. Menge

Mengen sind immer ungeordnet, die Reihenfolge der Elemente spielt keine Rolle!

Mengen und Mengenoperationen

Sei M eine beliebige Menge, dann ist

- $a \in M \Rightarrow$ a ist ein Element der Menge M ,
- $a \notin M \Rightarrow$ a ist kein Element der Menge M ,
- $M \subseteq N \Rightarrow$ aus $a \in M$ folgt $a \in N$ (M ist Teilmenge von N),
- $M \not\subseteq N \Rightarrow$ es gilt nicht $M \subseteq N$. Gleichwertig: es gibt ein $a \in M$ mit $a \notin N$ (M ist keine Teilmenge von N) und
- $M \subset N \Rightarrow$ es gilt $M \subseteq N$ und $M \neq N$ (M ist echte Teilmenge von N)

Definition spezieller Mengen

Spezielle Mengen können auf verschiedene Art und Weise definiert werden, wie z.B.

- durch Angabe von Elementen: So ist $\{a_1, \ldots, a_n\}$ die Menge, die aus den Elementen $a_1, \ldots, a_n$ besteht, oder
- durch eine Eigenschaft E: Dabei ist $\{a \mid E(a)\}$ die Menge aller Elemente a, die die Eigenschaft E besitzen, z.B. $a>0$.

Mengen, die durch eine Eigenschaft E definiert sind:

- $\{n \mid n \in N \text{ und } n \text{ ist durch 3 teilbar}\}$
- $\{n \mid n \in N \text{ und } n \text{ ist Primzahl und } n \leq 40\}$
- $\emptyset =_{def} \{a \mid a \neq a\}$ (die leere Menge)

Operationen auf Mengen

Seien A und B beliebige Mengen, dann ist

- $A \cap B =_{def} \{a \mid a \in A \text{ und } a \in B\}$ (Schnitt von A und B),
- $A \cup B =_{def} \{a \mid a \in A \text{ oder } a \in B\}$ (Vereinigung von A und B),
- $A \setminus B =_{def} \{a \mid a \in A \text{ und } a \notin B\}$ (Differenz von A und B),
- $\overline{A} =_{def} M \setminus A$ (Komplement von A bezüglich einer festen Grundmenge M) und
- $P(A) =_{def} \{B \mid B \subseteq A\}$ (Potenzmenge von A).

Zwei Mengen A und B mit $A \cap B = \emptyset$ nennt man disjunkt.

Elementare Logik

Auch Aussagenlogik findet Eingang in Pseudonotation und kann mit der Mengenlehre kombinierbar sein.

Aussagen sind entweder wahr ($\triangleq 1$) oder falsch ($\triangleq 0$).

Zusätzlich werden noch die Quantoren $\exists$ („es existiert“) und $\forall$ („für alle“) verwendet, die z.B. wie folgt gebraucht werden können

- $\forall x : P(x)$ bedeutet "Für alle x gilt die Aussage P(x)".
- $\exists x : P(x)$ bedeutet "Es existiert ein x, für das die Aussage P(x) gilt".

Üblicherweise lässt man sogar den Doppelpunkt weg und schreibt statt $\forall x : P(x)$ vereinfachend $\forall x P(x)$.

Iteratoren

Iterationen über Mengen (oder Listen und Arrays) können ebenfalls in einer Pseudonotation mit "Für alle" Logikoperator ausgedrückt werden:

```
for (x in X) { ... }
for (x of X) { ... }
for (x=1;x≤n;x++) { ... }
```

```
∀index(x) ∈ X do ...
∀x ∈ X do ...
∀x ∈ { 1,2,3,..,n } do ...
```