

Abschnitt 4: Daten und Algorithmen

4. Daten und Algorithmen

- 4.1 Datendarstellung durch Zeichenreihen
- 4.2 Eigenschaften von Algorithmen
- 4.3 Paradigmen der Algorithmenentwicklung

Roadmap

- ▶ Wir nehmen unsere Diskussion über den Algorithmus-Begriff wieder auf und konkretisieren einige wichtige Aspekte.
- ▶ Wir beschäftigen uns zunächst mit dem Aspekt der (eindeutigen) Darstellung der zu verarbeitenden Daten
- ▶ Anschließend diskutieren wir anhand einem Beispiel wesentliche Eigenschaften von Algorithmen und unterschiedliche Herangsweisen in der Algorithmenentwicklung.

Überblick

4. Daten und Algorithmen

4.1 Datendarstellung durch Zeichenreihen

4.2 Eigenschaften von Algorithmen

4.3 Paradigmen der Algorithmenentwicklung

Daten

- ▶ Wir betrachten zunächst die Daten (Objekte), die durch Algorithmen verarbeitet werden sollen.
- ▶ Wir wollen insbesondere klären, wie diese Daten dargestellt werden.
- ▶ Typische Daten sind Zahlen, z.B. die Zahl „drei“, die wie folgt dargestellt werden kann:
 - ▶ 3
 - ▶ DREI
 - ▶ III
 - ▶ drei ausgestreckte Finger einer Hand
 - ▶ ...



Datendarstellung

- ▶ Wir unterscheiden bei einem Objekt
 - ▶ die Darstellung, (*Syntax*, “Bezeichnung”),
 - ▶ seine Bedeutung, (*Semantik*, “Information”).
- ▶ Man kann die Syntax (von etwas) definieren, ohne die entsprechende Semantik zu kennen.
- ▶ Die Syntax von Daten (genauso: Programmiersprachen) muss formal (und eindeutig) definiert sein (z.B. mit Hilfe von „Grammatiken“).
- ▶ Wir führen hier allerdings eher eine informelle Diskussion und verweisen auf andere Vorlesungen.



Datendarstellung

- ▶ Einige der oben genannten Datendarstellungen für die Zahl „drei“ sind für maschinelle Verarbeitung nicht geeignet.
- ▶ Alle geeigneten Datendarstellungen beruhen auf dem Grundprinzip der *Zeichenreihe*, die wir im folgenden formal definieren.
- ▶ Dazu benötigen wir zunächst eine Menge an „Zeichen“, die zur Bildung von Zeichenreihen zur Verfügung steht.

Alphabet

- ▶ Ein *Alphabet* $\mathcal{A}$ ist eine endliche Menge, deren Elemente *Zeichen* genannt werden.
- ▶ Beispiele:
 - ▶ Menge der Großbuchstaben: $\{A, B, C, \dots, Z\}$
 - ▶ Menge der Dezimalziffern: $\{1, 2, 3, \dots, 9\}$
 - ▶ Menge der Vorzeichen: $\{+, -\}$
 - ▶ Menge der Richtungszeiger eines Lifts: $\{\uparrow, \downarrow\}$
- ▶ Alphabete, die genau zwei Zeichen enthalten heißen *binär*.
- ▶ Ein wichtiges binäres Alphabet besteht aus den *Binärziffern* (*Bits*) $\{0, 1\}$.

Zeichenreihe

- ▶ Eine *Zeichenreihe* über einem Alphabet $\mathcal{A}$ ist eine (endliche) Folge von Zeichen aus $\mathcal{A}$.
- ▶ Formal ist damit auch die leere Folge eine Zeichenreihe.

Zeichenreihe

- ▶ Wir schreiben Zeichenreihen/Folgen $(x_1, x_2, \dots, x_n)$ auch als $x_1x_2 \dots x_n$.
- ▶ Beispiele:
 - ▶ Sei $\mathcal{A}_1 = \{A, B, C, \dots, Z\}$
 - ▶ Die Folge INFORMATIK ist eine Zeichenreihe über $\mathcal{A}_1$.
 - ▶ Die Folge (R,I,C,H,T,E,R) ist eine Zeichenreihe über $\mathcal{A}_1$.
 - ▶ Die Folge Kröger ist *keine* Zeichenreihe über $\mathcal{A}_1$.
Warum?
 - ▶ Sei $\mathcal{A}_2 = \{0, 1\}$
 - ▶ Die Folge 0 ist eine Zeichenreihe über $\mathcal{A}_2$.
 - ▶ Die Folge 1 ist eine Zeichenreihe über $\mathcal{A}_2$.
 - ▶ Die Folge 01 ist eine Zeichenreihe über $\mathcal{A}_2$.
 - ▶ Die Folge 10 ist eine Zeichenreihe über $\mathcal{A}_2$.
 - ▶ Die Folge 11 ist eine Zeichenreihe über $\mathcal{A}_2$.
 - ▶ Die Folge 00 ist eine Zeichenreihe über $\mathcal{A}_2$.
 - ▶ ...



Bezeichnung von Daten

- ▶ Wir verwenden ausschließlich Zeichenreihen zur Bezeichnung von Daten.
- ▶ Im Folgenden betrachten wir als Beispiel die Darstellung von natürlichen Zahlen, also Elementen der Menge $\mathbb{N}_0$.
- ▶ Die Zahl “Dreizehn” lässt sich u.a. durch folgende Zeichenreihen bezeichnen:

13	$(\mathcal{A} = \{0, 1, 2, \dots, 9\}),$
DREIZEHN	$(\mathcal{A} = \{A, B, \dots, Z\}),$
	$(\mathcal{A} = \{ \}).$

- ▶ Nicht alle diese Darstellungen sind für den praktischen Gebrauch (z.B. Rechnen) geeignet.

Zifferndarstellung / p -adische Zahlendarstellung

- ▶ Am besten geeignet ist die *Zifferndarstellung*, z.B. die allgemein gebräuchliche *Dezimaldarstellung* über dem Alphabet $\{0, 1, 2, \dots, 9\}$.
- ▶ Das allgemeine Prinzip der Zifferndarstellung ist wie folgt definiert:

- ▶ Sei $p \in \mathbb{N}$, $p \geq 2$ und $\mathcal{A}_p = \{z_0, z_1, \dots, z_{p-1}\}$ ein Alphabet mit p Zeichen (genannt *Ziffern*) $z_0, z_1, \dots, z_{p-1}$.
- ▶ Die Funktion $Z : \mathcal{A}_p \rightarrow \mathbb{N}_0$ bildet jedes Zeichen aus $\mathcal{A}_p$ auf eine natürliche Zahl wie folgt ab:

$$Z(z_i) = i \quad \text{für } i = 0, \dots, p-1.$$

- ▶ Eine Zeichenreihe $x = x_n x_{n-1} \dots x_1 x_0$ (der Länge $(n+1)$) über $\mathcal{A}_p$ (d.h. $x_i \in \mathcal{A}_p$ für $0 \leq i \leq n$) bezeichnet die Zahl

$$\mathcal{Z}(x) = p^n \cdot Z(x_n) + p^{n-1} \cdot Z(x_{n-1}) + \dots + p \cdot Z(x_1) + Z(x_0).$$

- ▶ Zur Verdeutlichung schreiben wir auch x_p . x_p heißt *p -adische Zahlendarstellung* der Zahl $\mathcal{Z}(x_p) \in \mathbb{N}_0$.

Zifferndarstellung / p -adische Zahlendarstellung

- ▶ Nochmal die Formel

$$\mathcal{Z}(x) = p^n \cdot Z(x_n) + p^{n-1} \cdot Z(x_{n-1}) + \dots + p \cdot Z(x_1) + Z(x_0).$$

- ▶ Beispiele:

- ▶ Mit $p = 10$ und $\mathcal{A}_{10} = \{z_0, z_1 \dots, z_9\}$ erhält man die Dezimaldarstellung wenn man statt z_i gleich $Z(z_i)$ schreibt (also z.B. statt z_3 schreibe $Z(z_3) = 3$):

$$\mathcal{Z}(983_{10}) = 10^2 \cdot 9 + 10^1 \cdot 8 + 10^0 \cdot 3 = \text{“neunhundertdreiundachtzig”}.$$

(Wir schreiben direkt $\mathcal{A}_{10} = \{0, 1 \dots, 9\}$)

Zifferndarstellung / p -adische Zahlendarstellung

- ▶ Nochmal die Formel

$$\mathcal{Z}(x) = p^n \cdot \mathcal{Z}(x_n) + p^{n-1} \cdot \mathcal{Z}(x_{n-1}) + \dots + p \cdot \mathcal{Z}(x_1) + \mathcal{Z}(x_0).$$

- ▶ Beispiele (cont.):

- ▶ $p = 2$ und $\mathcal{A}_2 = \{0, 1\}$ (*Binärdarstellung*):

$$\mathcal{Z}(1111010111_2) = 2^9 \cdot 1 + 2^8 \cdot 1 + 2^7 \cdot 1 + 2^6 \cdot 1 + 2^5 \cdot 0 + 2^4 \cdot 1 + 2^3 \cdot 0 + 2^2 \cdot 1 + 2 \cdot 1 + 1 =$$

“neunhundertdreiundachtzig”.

- ▶ $p = 8$ und $\mathcal{A}_8 = \{0, 1, 2, 3, 4, 5, 6, 7\}$ (*Oktaldarstellung*):

$$\mathcal{Z}(1727_8) = 8^3 \cdot 1 + 8^2 \cdot 7 + 8 \cdot 2 + 7 = \text{“neunhundertdreiundachtzig”}.$$

- ▶ $p = 16$ und $\mathcal{A}_{16} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$
(*Hexadezimaldarstellung*):

$$\mathcal{Z}(3D7_{16}) = 16^2 \cdot 3 + 16 \cdot 13 + 7 = \text{“neunhundertdreiundachtzig”}.$$



Zifferndarstellung / p -adische Zahlendarstellung

- ▶ Führende Nullen sind in der Definition der p -adischen Zahlendarstellung zugelassen, z.B. ist die Zeichenreihe

000983

eine zulässige Dezimaldarstellung und bezeichnet die gleiche Zahl wie die Zeichenreihe 983.

- ▶ Offensichtlich können führende Nullen (d.h. führende Ziffern 0, also die Ziffer z_0) immer weggelassen werden, außer bei der Bezeichnung “0” für die Zahl “null”.
- ▶ Für jede Zahl aus $\mathbb{N}_0$ gibt es für beliebiges $p \geq 2$ eine p -adische Darstellung.
- ▶ Betrachtet man nur Darstellungen ohne führende Nullen, so ist (zu festem $p \geq 2$) die p -adische Zahlendarstellung eindeutig.

Menschengerechte Darstellung von Daten

- ▶ Zur Entwicklung von Algorithmen werden wir typischerweise die Dezimaldarstellung der natürlichen Zahlen verwenden.
- ▶ Allgemein gibt es für die meisten Daten eine *Standarddarstellung* bei denen die Lesbarkeit der Darstellung für den menschlichen Benutzer im Vordergrund steht.
- ▶ Wir verwenden hier folgende Standardbezeichnungen wie sie in den üblichen höheren Programmiersprachen gebräuchlich sind:

Natürliche Zahlen	$\mathbb{N}_0$	Dezimaldarstellung (ohne führende Nullen)
Ganze Zahlen	$\mathbb{Z}$	wie natürliche Zahlen, ggf. mit Vorzeichen “-”, z.B. 3,-3.
Reelle Zahlen	$\mathbb{R}$	<i>Gleitpunktdarstellung</i> , siehe später.
Wahrheitswerte	$\mathbb{B}$	<i>TRUE</i> und <i>FALSE</i> für “wahr” bzw. “falsch”.

Darstellung von Zeichen und Texten

- ▶ Eine Dezimaldarstellung (z.B. 983) stellt eine natürliche Zahl dar, die verarbeitet werden kann.
- ▶ Die Zeichenreihe selbst (und nicht die dargestellte Zahl) könnte aber auch Gegenstand der Verarbeitung sein.
- ▶ Zeichenreihen können also nicht nur Darstellungen von Objekten sein, sondern auch selbst Objekte, die dargestellt werden müssen.
- ▶ Zeichenreihen heißen in diesem Zusammenhang *Texte*.

Darstellung von Zeichen und Texten

- ▶ In der Praxis wird zur Bildung von Texten häufig das sog. *ASCII-Alphabet* benutzt.
- ▶ Das ASCII-Alphabet repräsentiert eine Menge von Zeichen, die wir im folgenden als *CHAR* bezeichnen.
- ▶ Die Elemente von *CHAR* finden Sie in allen gängigen Lehrbüchern.

Maschinengerechte Darstellung von Daten

- ▶ Auf Rechenanlagen wird meist eine andere Darstellung der Daten gewählt (typischerweise Zeichenreihen über den oben benannten Alphabeten $\mathcal{A}_2$, $\mathcal{A}_8$ und $\mathcal{A}_{16}$).
- ▶ Aus technischen Gründen kann die kleinste Speichereinheit eines Computers (das **Bit**) nur zwei Zustände speichern:
 - ▶ Zustand 1: es liegt (elektr.) Spannung an.
 - ▶ Zustand 0: es liegt keine Spannung an.

Maschinengerechte Darstellung von Daten

- ▶ Daher werden Werte (Daten/Objekte) als Bitmuster (Zeichenreihe über dem Alphabet $\mathcal{A}_2 = \{0, 1\}$) codiert gespeichert (auch Darstellungen über $\mathcal{A}_8$ und $\mathcal{A}_{16}$ sind mit diesen technischen Gegebenheiten gut vereinbar).
- ▶ Intern kann der Computer also z.B. die natürlichen Zahlen in Binärcodierung repräsentieren.
- ▶ Ganze Zahlen können intern ebenfalls leicht als Zeichenkette über dem Alphabet $\mathcal{A}_2 = \{0, 1\}$ codiert werden (z.B. mit einem führenden Bit für das Vorzeichen; Genaueres darüber werden Sie in der Vorlesung “Rechnerarchitektur” lernen).



Maschinengerechte Darstellung von Daten

- ▶ Die Binärdarstellung der reellen Zahlen ist etwas komplizierter.
- ▶ Die *Gleitpunktdarstellung* einer Zahl z ist

$$z = m \cdot 2^e,$$

wobei sowohl m (*Mantisse*) als auch e (*Exponent*) binär repräsentiert werden (können).

- ▶ In vielen Programmiersprachen (z.B. Java) wird eine Zahl dargestellt durch $z = m \cdot 10^e$, z.B. 3.14, -7.45, 1.33E - 2 (für $1.33 \cdot 10^{-2}$).

Maschinengerechte Darstellung von Daten

- ▶ Eine genaue Spezifikation lernen wir im nächsten Abschnitt kennen.
- ▶ **Wichtig:** Für viele reelle Zahlen gibt es gar keine derartige Darstellung (z.B. für $\sqrt{2}$). Die darstellbaren Zahlen heißen auch *Gleitpunktzahlen*.
- ▶ Dieser Aspekt der maschinengerechten Darstellung von Daten ist bei der Entwicklung von Algorithmen möglicherweise wichtig! **Warum?**

Maschinengerechte Darstellung von Daten

- ▶ Ein weiterer Aspekt der maschinengerechten Darstellung ist, dass die Ressourcen (Speicherzellen) einer Rechenanlage begrenzt sind.
- ▶ Es stehen daher für die Darstellung von Daten immer nur endlich viele Bits zur Verfügung.
- ▶ D.h. zur Darstellung einer beliebigen natürlichen Zahl stehen nur Zeichenketten mit fixer *Länge* zur Verfügung (notfalls mit führenden Nullen).

Maschinengerechte Darstellung von Daten

- ▶ **Achtung:**

Werte, deren Darstellung mehr Zeichen (Bits) benötigen würden, können somit nicht dargestellt werden.

- ▶ Die Länge der zur Verfügung stehenden Zeichenketten hat damit offenbar Einfluss auf den Wertebereich der darstellbaren Objekte.
- ▶ Dies ist bei der Entwicklung von Algorithmen möglicherweise ebenfalls wichtig! **Warum?**

Syntaxdefinitionen

- ▶ Wie bereits erwähnt lässt sich die Gestalt einer Datendarstellung (Syntax) formal definieren (unabhängig von deren Semantik).
- ▶ Gleiches gilt für eine gesamte (Programmier-)sprache, d.h. z.B. deren Elemente.
- ▶ Eine formale, häufig gebrauchte Form von Syntaxdefinitionen ist die *Backus-Naur-Form (BNF)*, die wir hier aber nicht genauer besprechen.
- ▶ Dennoch soll hier nochmal betont werden, dass eine formale Definition der Syntax von Daten und Programmiersprachen eine wichtige Grundlage fürs Programmieren darstellt.
- ▶ Daher im folgenden in aller Kürze noch ein Beispiel ...



Bsp.: Syntaxdefinition von natürlichen Zahlen

- ▶ Die Menge der natürlichen Zahlen in Dezimaldarstellung kann durch folgende Regeln definiert werden:
 1. 0 ist eine *⟨Dezimalzahl⟩*.
 2. Jede Ziffer $x \in \mathcal{A}_{10} \setminus \{0\}$ ist eine *⟨Nichtnulldarstellung⟩*.
 3. Ist a eine *⟨Nichtnulldarstellung⟩* und $y \in \mathcal{A}_{10}$ so ist $a \circ y$ eine *⟨Nichtnulldarstellung⟩*.
 4. Jede *⟨Nichtnulldarstellung⟩* ist eine *⟨Dezimalzahl⟩*.
- ▶ Die Begriffe *⟨Dezimalzahl⟩* und *⟨Nichtnulldarstellung⟩* werden *syntaktische Variablen* genannt.
- ▶ Syntaktische Variablen kennzeichnen den zu definierenden Begriff sowie (beliebig aber endlich viele) Hilfsbegriffe⁵.
- ▶ Die Zeichen des vorgegeben Alphabets heißen *Terminalzeichen*.



⁵Der Hilfsbegriff *⟨Nichtnulldarstellung⟩* ist im Übrigen induktiv definiert.

Bsp.: Syntaxdefinition von natürlichen Zahlen

- ▶ Beispiel:

Die Zeichenreihe 308 ist eine Dezimalzahl gemäß folgender Anwendungen der Regeln:

3 ist *⟨Nichtnulldarstellung⟩* nach Regel 2

30 ist *⟨Nichtnulldarstellung⟩* nach Regel 3

308 ist *⟨Nichtnulldarstellung⟩* nach Regel 3

308 ist *⟨Dezimalzahl⟩* nach Regel 4

- ▶ Diese „Anwendung“ heißt *Ableitung*. Mit Hilfe dieser Regeln lassen sich alle natürlichen Zahlen ableiten.

Überblick

4. Daten und Algorithmen

4.1 Datendarstellung durch Zeichenreihen

4.2 Eigenschaften von Algorithmen

4.3 Paradigmen der Algorithmenentwicklung

Eigenschaften von Algorithmen

- ▶ Zur Erinnerung: wichtige Eigenschaften, die Algorithmen haben können/sollten, sind:
 1. Präzise, endliche Beschreibung.
 2. Effektive Verarbeitungsschritte.
 3. Elementare Verarbeitungsschritte.
 4. Ein Algorithmus heißt *terminierend*, wenn er bei jeder Anwendung nach endlich vielen Verarbeitungsschritten zum Ende kommt.
 5. Ein Algorithmus heißt *deterministisch*, wenn die Wirkung und die Reihenfolge der Einzelschritte eindeutig festgelegt ist, andernfalls *nicht-deterministisch*.
 6. Ein Algorithmus heißt *determiniert*, wenn das Ergebnis der Verarbeitung für jede einzelne Anwendung eindeutig bestimmt ist, andernfalls *nicht-determiniert*.



Eigenschaften von Algorithmen

Desweiteren ist natürlich die möglicherweise wichtigste Eigenschaft eines Algorithmus die *Korrektheit*, d.h. informell, „der Algorithmus tut, was er tun soll“.

1. Ein Algorithmus heißt *partiell korrekt*, wenn für alle gültigen Eingaben das Resultat der Spezifikation des Algorithmus entspricht.
2. Ein Algorithmus heißt *(total) korrekt*, wenn der Algorithmus partiell korrekt ist, und für alle gültigen Eingaben terminiert.

Wir werden uns später noch genauer mit der Frage beschäftigen, wie man sicher sein kann, dass ein Algorithmus partiell/total korrekt ist.



Running Example

Wir betrachten folgende Aufgabe:

Ein Kunde kauft Waren für $1 \leq r \leq 100$ EUR und bezahlt mit einem 100 EUR Schein (r sei ein voller EUR Betrag ohne Cent-Anteil). Gesucht ist ein Algorithmus, der zum Rechnungsbetrag r das Wechselgeld w bestimmt. Zur Vereinfachung nehmen wir an, dass w nur aus 1 EUR oder 2 EUR Münzen oder 5 EUR Scheinen bestehen soll. Es sollen möglichst wenige Münzen/Scheine ausgegeben werden (also ein 5 EUR Schein statt fünf 1 EUR Münzen).

Datendarstellung

- ▶ Zunächst müssen wir zur Lösung dieser Aufgabe die Darstellung (Modellierung) der relevanten Daten festlegen.
- ▶ Für den Rechnungsbetrag r ist dies trivial, denn offensichtlich ist $r \in \mathbb{N}$. Wir nehmen an, dass r in Dezimaldarstellung gegeben ist.
- ▶ Das Wechselgeld w kann auf verschiedene Weise modelliert werden, z.B. als Folge oder Multimenge von Wechselgeldmünzen. Ein aus zwei 1-EUR-Münzen, einer 2-EUR-Münze und zwei 5-EUR-Scheinen bestehendes Wechselgeld könnte als Folge $(1, 1, 2, 5, 5)$ dargestellt sein.
- ▶ Wir legen folgende Datendarstellung fest:
 - ▶ r : als natürliche Zahl in Dezimaldarstellung.
 - ▶ w : als Folge von Werten 1, 2 oder 5.
- ▶ Wir benutzen die Sprechweise „nimm x zu w hinzu“ für die Konkatenation $w \circ x$.

Erster Lösungsansatz

Idee:

Ausgehend von r sukzessive um 1, 2 und 5 hoch zählen (unter Hinzunahme der entsprechenden Münzen/Scheine) bis man bei 100 angekommen ist. Dabei möglichst schnell eine durch 5 teilbare Zahl erreichen, um möglichst wenige Münzen/Scheine auszugeben.

Algorithmus 2 (Wechselgeld, die Erste)

Eingabe: $r \in \mathbb{N}$

Führe folgende Schritte der Reihe nach aus:

- 1 Setze $w = ()$.
- 2 Falls die letzte Ziffer von r eine 2, 4, 7 oder 9 ist,
dann erhöhe r um 1 und nimm 1 zu w hinzu.
- 3 Falls die letzte Ziffer von r eine 1 oder 6 ist,
dann erhöhe r um 2 und nimm 2 zu w hinzu.
- 4 Falls die letzte Ziffer von r eine 3 oder 8 ist,
dann erhöhe r um 2 und nimm 2 zu w hinzu.
- 5 Solange $r < 100$: Erhöhe r um 5 und nimm 5 zu w hinzu.

Ausgabe: w

Erster Lösungsansatz

- Ablaufbeispiel: Sei $r = 81$.

$$r = 81$$

(Ausgangssituation)

Schritt 1 $r = 81 \quad w = ()$

Schritt 2 (keine Änderung, da die letzte Ziffer keine 2,4,7,9 ist)

Schritt 3 $r = 83 \quad w = (2)$

Schritt 4 $r = 85 \quad w = (2, 2)$

Schritt 5 $r = 90 \quad w = (2, 2, 5)$

$$r = 95 \quad w = (2, 2, 5, 5)$$

$$r = 100 \quad w = (2, 2, 5, 5, 5)$$

Erster Lösungsansatz

► Eigenschaften:

- Endliche Aufschreibung: OK.
- Effektive und elementare Einzelschritte: ?
- Der Algorithmus ist terminierend, deterministisch, determiniert und partiell korrekt (und damit auch total korrekt).

Variante des ersten Lösungsansatzes

Idee:

Fasse ähnliche Schritte zusammen.

Algorithmus 3 (Wechselgeld, die Erste (Variante))

Eingabe: $r \in \mathbb{N}$

Führe folgende Schritte der Reihe nach aus:

- 1 Setze $w = ()$.
- 2 Solange $r < 100$: Führe jeweils (wahlweise) einen der folgenden Schritte aus:
- 3 Falls die letzte Ziffer von r eine 2,4,7 oder 9 ist,
 dann erhöhe r um 1 und nimm 1 zu w hinzu.
- 4 Falls die letzte Ziffer von r eine 1,2,3,6,7 oder 8 ist,
 dann erhöhe r um 2 und nimm 2 zu w hinzu.
- 5 Falls die letzte Ziffer von r eine 0,1,2,3,4 oder 5 ist,
 dann erhöhe r um 5 und nimm 5 zu w hinzu.

Ausgabe: w



Variante des ersten Lösungsansatzes

► Ablaufbeispiel:

 $r = 81$

(Ausgangssituation)

Schritt 1 $r = 81$ $w = ()$ Schritt 2b $r = 83$ $w = (2)$ Schritt 2b $r = 85$ $w = (2, 2)$ Schritt 2c $r = 90$ $w = (2, 2, 5)$ Schritt 2c $r = 95$ $w = (2, 2, 5, 5)$ Schritt 2c $r = 100$ $w = (2, 2, 5, 5, 5)$

► Alternativer Ablauf:

 $r = 81$

(Ausgangssituation)

Schritt 1 $r = 81$ $w = ()$ Schritt 2b $r = 83$ $w = (2)$ Schritt 2c $r = 88$ $w = (2, 5)$ Schritt 2b $r = 90$ $w = (2, 5, 2)$ Schritt 2c $r = 95$ $w = (2, 5, 2, 5)$ Schritt 2c $r = 100$ $w = (2, 5, 2, 5, 5)$ 

Variante des ersten Lösungsansatzes

► Eigenschaften:

- Die Variante (Algorithmus 3) arbeitet nach dem selben Grundprinzip wie der urspr. Algorithmus (Algorithmus 2), läßt aber gewisse “Freiheiten” bei der Auswahl des nächsten Schrittes beim wiederholt auszuführenden Schritt 2. Daher: Algorithmus 2 ist nicht-deterministisch.
- Algorithmus 2 ist nicht determiniert (siehe alternativer Ablauf): Bei ein und derselben Anwendung erzeugt der Algorithmus zwei unterschiedliche Folgen $w = (2, 2, 5, 5, 5)$ und $w = (2, 5, 2, 5, 5)$.



Variante des ersten Lösungsansatzes

- ▶ Bemerkung 1: wenn wir statt Folgen Multimengen (bei denen die Reihenfolge der Elemente keine Rolle spielt) bei der Darstellung von w verwendet hätten, wären die Ergebnisse $w = \{2, 2, 5, 5, 5\}$ und $w = \{2, 5, 2, 5, 5\}$ gleich, d.h. in diesem Fall wäre Algorithmus 2 determiniert.
- ▶ Bemerkung 2: Würden wir r nicht in Dezimal- sondern z.B. in Binärdarstellung modellieren, wären beide Algorithmen in der angegebenen Form sinnlos.
- ▶ Die Wahl der Datendarstellung kann also erheblichen Einfluss auf Eigenschaften von Algorithmen haben.

Zweite Variante des ersten Lösungsansatzes

Idee:

Wir lösen uns von der Abhängigkeit von der Datendarstellung, indem wir r nicht mehr hochzählen, sondern die Differenz $100 - r$ berechnen und diese in möglichst wenige Teile der Größen 1, 2 und 5 aufteilen. Wir gehen dabei davon aus, dass die Differenz “–” für beliebige Zifferndarstellungen (dezimal, binär, oktal, etc.) definiert ist. Die Zahl 100 müsste in die selbe Darstellung gebracht werden, die auch für r benutzt wird.

Algorithmus 4 (Wechselgeld, die Erste (Variante 2))

Eingabe: $r \in \mathbb{N}$

Führe folgende Schritte der Reihe nach aus:

- 1 Berechne $d = 100 - r$ und setze $w = ()$.
- 2 Solange $d \geq 5$: Vermindere d um 5 und nimm 5 zu w hinzu.
- 3 Falls $d \geq 2$, dann vermindere d um 2 und nimm 2 zu w hinzu.
- 4 Falls $d \geq 2$, dann vermindere d um 2 und nimm 2 zu w hinzu.
- 5 Falls $d \geq 1$, dann vermindere d um 1 und nimm 1 zu w hinzu.

Ausgabe: w



Zweite Variante des ersten Lösungsansatzes

► Ablaufbeispiel:

$$r = 81$$

(Ausgangssituation)

Schritt 1 $d = 19$ $w = ()$

Schritt 2 $d = 14$ $w = (5)$

$$d = 9 \quad w = (5, 5)$$

$$d = 4 \quad w = (5, 5, 5)$$

Schritt 3 $d = 2$ $w = (5, 5, 5, 2)$

Schritt 4 $d = 0$ $w = (5, 5, 5, 2, 2)$

Schritt 5 (keine Änderung, da $d \geq 1$ nicht gilt)

Diskussion

- ▶ Alle drei Varianten weisen eine gemeinsame “operative” Auffassung auf: Die Berechnung des Rückgelds wird durch eine Folge von “Handlungen” vollzogen.
- ▶ Diese Handlungen sind charakterisiert durch Veränderungen der Größen r bzw. d und w .
- ▶ Die Abfolge dieser “Aktionen” wird durch typische Konstruktionen gesteuert:
 - ▶ *Fallunterscheidung*: Falls ..., dann ...
ermöglicht, die Ausführungen von Aktionen vom Erfülltsein gewisser Bedingungen abhängig sein zu lassen.
 - ▶ *Iteration*: Solange ...: ...
steuert die Wiederholung von Aktionen in Abhängigkeit gewisser Bedingungen.

Diskussion (cont.)

- ▶ Neben der strikten operativen Auffassung gibt es eine Sichtweise von Algorithmen, die durch eine rigorose mathematische Abstraktion gewonnen wird.
- ▶ In unserem Beispiel ist die Zuordnung eines Wechselgelds w zu einem Rechnungsbetrag r mathematisch nichts anderes als eine Abbildung

$$h : r \mapsto \text{“herauszugebendes Wechselgeld”}$$

- ▶ Die Aufgabe ist dann, eine Darstellung der Abbildung h zu finden, die “maschinell auswertbar” ist.

Diskussion (cont.)

- ▶ Eine triviale Lösung für h ist, in einer kompletten Aufstellung alle möglichen Werte für r mit zugehörigem Wechselgeld $w = h(r)$ aufzulisten.
- ▶ Dies ist allgemein aber nur für endliche Definitionsbereiche möglich und sogar nur für “kleine” Definitionsbereiche sinnvoll (für unser Problem also möglich, aber bestenfalls unpraktisch).
- ▶ Allgemein wird man eine kompakte Darstellung suchen, in der die zu bestimmende Abbildung in geeigneter Weise aus einfachen (elementar auswertbaren) Abbildungen zusammengesetzt ist.



Vorbereitungen

- Für eine solche Darstellung der Abbildung h benötigen wir zunächst zwei “Hilfsabbildungen”, die in der Informatik oft gebräuchlich sind:

$$DIV : \mathbb{N}_0 \times \mathbb{N} \rightarrow \mathbb{N}_0$$

$$MOD : \mathbb{N}_0 \times \mathbb{N} \rightarrow \mathbb{N}_0$$

- DIV berechnet das Ergebnis der ganzzahlige Division zweier natürlicher Zahlen, z.B. $DIV(7, 3) = 2$.
- MOD berechnet den Rest der ganzzahligen Division zweier natürlicher Zahlen, z.B. $MOD(7, 3) = 1$.
- Formal: Sind $k \in \mathbb{N}_0$ und $l \in \mathbb{N}$, so kann man k durch l mit ganzzahligem Ergebnis q teilen, wobei eventuell ein Rest r übrigbleibt, d.h. es gibt $q, r \in \mathbb{N}_0$ mit $r < l$ und $k = q \cdot l + r$. Dann ist

$$DIV(k, l) = q \quad \text{und} \quad MOD(k, l) = r.$$

Funktionaler Lösungsansatz

Idee:

- ▶ Ähnlich wie Algorithmus 4 teilen wir $100 - r$ durch 5.
- ▶ Der ganzzahlige Quotient $q_1 = \text{DIV}(100 - r, 5)$ ist die Anzahl der 5-EUR-Scheine im Wechselgeld $h(r)$.
- ▶ Der Rest $r_1 = \text{MOD}(100 - r, 5)$ ist der noch zu verarbeitende Wechselbetrag. Offensichtlich gilt $r_1 < 5$.
- ▶ r_1 muss nun auf 1 und 2 aufgeteilt werden, d.h. analog bilden wir $q_2 = \text{DIV}(r_1, 2)$ und $r_2 = \text{MOD}(r_1, 2)$.
- ▶ q_2 bestimmt die Anzahl der 2 EUR Münzen und r_2 die Anzahl der 1 EUR Münzen in $h(r)$.



Funktionaler Lösungsansatz

Algorithmus 5 (Wechselgeld, Die Zweite)

$$h(r) = (5_1, \dots, 5_{\text{DIV}(100-r, 5)}, \\ 2_1, \dots, 2_{\text{DIV}(\text{MOD}(100-r, 5), 2)}, \\ 1_1, \dots, 2_{\text{MOD}(\text{MOD}(100-r, 5), 2)})$$

Bemerkungen:

- ▶ Dabei sind alle Elemente in der Ergebnisfolge durchnummeriert, um die Anzahl der entsprechenden Elemente anzudeuten, d.h. das Element 5_i bezeichnet den i -ten 5-EUR-Schein im Ergebnis.
- ▶ In dieser Schreibweise kann z.B. $(5_1, 5_0)$ auftreten, was bedeuten soll, dass die Folge kein Element 5 enthält.

Funktionaler Lösungsansatz

- Beispielanwendung $r = 81$:

$$DIV(100 - r, 5) = DIV(19, 5) = 3$$

$$MOD(100 - r, 5) = MOD(19, 5) = 4$$

$$DIV(MOD(100 - r, 5), 2) = DIV(4, 2) = 2$$

$$MOD(MOD(100 - r, 5), 2) = MOD(4, 2) = 0$$

d.h. man erhält $h(81) = (5_1, 5_2, 5_3, 2_1, 2_2, 1_1, 1_0)$ oder, einfacher geschrieben,

$$h(81) = (5, 5, 5, 2, 2).$$



Funktionaler Lösungsansatz

► Eigenschaften:

- Wenn man von der Bildung der Folgen aus den berechneten Anzahlen der Elemente 1, 2 und 5 absieht, sind nur die Auswertungen der Operationen “–”, *DIV* und *MOD* als Einzelschritte anzusehen. Setzt man diese als elementar voraus, ist die angegebene Definition von h eine Beschreibung des gesuchten Algorithmus.
- Diese Darstellung nimmt keinen Bezug mehr auf eine “Abfolge von Aktionen” sondern beschreibt den “funktionalen Zusammenhang” zwischen r und w durch ineinander eingesetzte Anwendungen (Komposition) gewisser Basisoperationen.

Variante des funktionalen Lösungsansatzes

- ▶ Leider läßt sich in vielen Fällen die gesuchte Abbildung nicht in derartig einfacher Weise explizit angeben.
- ▶ Ein wichtiges Prinzip für den Allgemeinfall von Abbildungen auf natürlichen Zahlen (und anderen Mengen) ist das Prinzip der Rekursion, das wir im vorigen Kapitel kennen gelernt haben.

Algorithmus 6 (Wechselgeld, Die Zweite (Variante))

$$h(r) = \begin{cases} \text{falls } r = 100, \text{ dann } (), & (1) \\ \text{falls } 100 - r \geq 5, \text{ dann } (5) \circ h(r + 5), & (2) \\ \text{falls } 5 > 100 - r \geq 2, \text{ dann } (2) \circ h(r + 2), & (3) \\ \text{falls } 2 > 100 - r \geq 1, \text{ dann } (1) \circ h(r + 1), & (4) \end{cases}$$

Hier ist der Basisfall (1) für $h(100)$ definiert und die Rekursionsfälle (2),(3),(4) werden auf die Fälle $h(r + 5)$, $h(r + 2)$ und $h(r + 1)$ zurückgeführt.

Variante des funktionalen Lösungsansatzes

- Beispielanwendung $r = 81$:

$$\begin{aligned}h(81) &= (5) \circ h(86) && \text{(Fall 2)} \\&= (5) \circ (5) \circ h(91) && \text{(Fall 2)} \\&= (5) \circ (5) \circ (5) \circ h(96) && \text{(Fall 2)} \\&= (5) \circ (5) \circ (5) \circ (2) \circ h(98) && \text{(Fall 3)} \\&= (5) \circ (5) \circ (5) \circ (2) \circ (2) \circ h(100) && \text{(Fall 3)} \\&= (5) \circ (5) \circ (5) \circ (2) \circ (2) \circ () && \text{(Fall 1)} \\&= (5, 5, 5, 2, 2)\end{aligned}$$

Variante des funktionalen Lösungsansatzes

- ▶ Trotz der rein funktionalen Darstellung der Zuordnung $r \mapsto w$ erinnert die Auswertung wieder an die Abfolge der Aktionen im entsprechenden Ablaufbeispiel (ähnlich wie in Algorithmus 4).
- ▶ Tatsächlich kann das Prinzip der Rekursion auch in der operativen Auffassung der Algorithmen 2-4 eingerichtet werden und dabei einen ähnlichen Effekt wie die Iteration erzielen.
- ▶ In einer rekursiven Abbildungsdefinition greift man auf Werte dieser Abbildung für kleinere (oder hier: größere) Argumente zurück.
- ▶ Analog kann in einem Algorithmus, der die Abfolge gewisser Aktionen in Abhängigkeit von Eingabewerten steuert, die Ausführung des Algorithmus selbst auch als eine derartige Aktion auftreten.



Zweite Variante des funktionalen Lösungsansatzes

Algorithmus 7 (Wechselgeld, die Zweite (Variante 2))

Eingabe: $r \in \mathbb{N}$

Setze $w = ()$

Führe den in Abhängigkeit von r rekursiv definierten Algorithmus $A(r)$ aus:

$A(r)$:

Führe denjenigen der folgenden Schritte (1) - (3) aus, dessen Bedingung erfüllt ist (ist keine Bedingung erfüllt, ist die Ausführung zu Ende):

- (1) Falls $100 - r \geq 5$, dann nimm 5 zu w hinzu und führe anschließend $A(r + 5)$ aus.
- (2) Falls $5 > 100 - r \geq 2$, dann nimm 2 zu w hinzu und führe anschließend $A(r + 2)$ aus.
- (3) Falls $5 > 100 - r \geq 1$, dann nimm 1 zu w hinzu und führe anschließend $A(r + 1)$ aus.

Ausgabe: w



Überblick

4. Daten und Algorithmen

4.1 Datendarstellung durch Zeichenreihen

4.2 Eigenschaften von Algorithmen

4.3 Paradigmen der Algorithmenentwicklung

Algorithmische Konzepte

- ▶ In den vergangenen Abschnitten haben wir einige grundlegende algorithmische Konzepte kennengelernt:
 - ▶ Eingabe- und Ergebnisdaten (insbesondere solche von komplexer Art) müssen geeignet dargestellt werden.
 - ▶ Die Ausführung gewisser Aktionen bzw. die Auswertung gewisser Operationen wird als in elementaren Schritten durchführbar vorausgesetzt.
 - ▶ Einzelne Schritte können zusammengesetzt werden (z.B. Nacheinanderausführung, Auswahl von Aktionen, Kompositionen von Abbildungen, etc.)
 - ▶ Fallunterscheidung, Iteration und Rekursion ermöglichen die Steuerung des durch den Algorithmus beschriebenen Verfahrens.
- ▶ Diese Konzepte sind z.T. abhängig vom verwendeten Programmierparadigma.
- ▶ Zum Abschluss dieses Kapitels diskutieren wir daher nochmal die verschiedenen Paradigmen.



Imperative Programmierung

Historische Entwicklung:

- ▶ Das ursprüngliche Paradigma; aus der Architektur der Computer der zugrundeliegenden Befehlssteuerung abgeleitet (binär codierte Befehle, direkte Adressierung von Speicherzellen)
- ▶ Abstraktion von der Maschinenorientierung durch Variablenkonzept (statt binäre Speicheradressierung) und Rekursivität der Programmsteuerung (Programme können durch andere Programme verändert werden)
- ▶ Problemorientierte Programmierung

Imperative Programmierung

Wesentliche Charakteristika:

- ▶ Programm ist Folge elementarer Anweisungen
- ▶ Diese Anweisung haben typischerweise einen Effekt auf die zu verarbeitenden Daten (z.B. werden Daten verändert)
- ▶ Programmausführung: Abarbeitung der Anweisungen
- ▶ Wichtige Konzepte: Variablen, Verzweigung, Schleifen, Prozeduren
- ▶ Vertreter: FORTRAN, ALGOL, COBOL, Pascal, BASIC, C, Modula (und z.T., als deren Weiterentwicklungen: C++ und Java))



Funktionale Programmierung

Historische Entwicklung:

- ▶ Höhere Abstraktion: weg von der maschinen-orientierten Problemauffassung hin zur Beschreibungsform der Problemstellung (daher auch applikative, d.h. anwendungsbezogene Programmierung)
- ▶ Eigenschaft des Problems bzw., der Lösung wird beschrieben (deklarativ), orientiert am Menschen statt an der Rechenanlage



Funktionale Programmierung

Wesentliche Charakteristika:

- ▶ Programm ist Funktion von Eingabe- in Ausgabewerte
- ▶ Die Funktion hat keine Effekte auf die zu verarbeitenden Daten (z.B. werden Eingabedaten nicht verändert), sie gibt lediglich die Ausgabe zurück
- ▶ Programmausführung: Berechnung der Funktion
- ▶ Wichtige Konzepte: Ausdruck, Substitution/Auswertung, λ -Kalkül
- ▶ Vertreter: LISP, SML, Haskell

Objektorientierte Programmierung

Historische Entwicklung:

- ▶ bei großen SW-Systemen werden imperative/funktionale Programme schnell unübersichtlich
- ▶ Idee der abstrakten Datentypen wird zu benutzerdefinierten Datentypen erweitert; noch stärkere Abstraktion durch Datenkapselung

Idee:

- ▶ Klassen stellen benutzerdefinierte Datentypen zur Verfügung, d.h. definieren eine Menge von Objekten mit gewissen Eigenschaften (Attribute, die den Zustand der Objekte spezifizieren) und Verhalten (Methoden, die z.B. den Zustand verändern)
- ▶ Objekte verschiedener Klassen können miteinander agieren (gegenseitig Methoden aufrufen)
- ▶ Einbettung von Klassen in eine Hierarchie ermöglicht Vererbung (Wiederverwendung) von Eigenschaften

Objektorientierte Programmierung

Wesentliche Charakteristika:

- ▶ Programm ist Menge von Klassen und Objekten
- ▶ Programmausführung: Interaktion zwischen Objekten
- ▶ Wichtige Konzepte: Klassen und Objekte, Datenkapselung, Vererbung, Polymorphie
- ▶ Vertreter: Java, C++, Python

Logikprogrammierung

- ▶ Idee: ausgehend von einer Menge von Fakten (Grundtatsachen und Axiome) werden mit Hilfe festgelegter Schlussregeln eine Aussage abgeleitet (oder falsifiziert).
- ▶ Programm ist Sammlung von Fakten und Regeln
- ▶ Programmausführung: Suche nach Antworten auf Anfragen
- ▶ Wichtige Konzepte: logisches Schließen, regelbasiert, prädikativ, constraintbasiert
- ▶ Vertreter: PROLOG

Zusammenfassung

► Imperative Algorithmen:

- spezifizieren die Abfolge von Aktionen, die typischerweise bestimmte Größen verändern.
- spiegeln die technischen Möglichkeiten von Rechneranlagen wider, die Schritt für Schritt bestimmte grundlegende Aktionen ausführen können.

► Funktionale Algorithmen:

- spezifizieren eine auswertbare Darstellung des funktionalen Zusammenhangs zwischen Ein- und Ergebniswerten.
- spiegeln ein höheres Abstraktionsniveau wider: die eigentliche Spezifikation des Algorithmus als Abbildung ist praktisch losgelöst von den technischen Möglichkeiten der Rechenanlage (Aktionen Schritt für Schritt auszuführen).

Zusammenfassung

► *Objektorientierte Algorithmen:*

- spezifizieren eine höhere Abstraktionsebene zur Darstellung von komplexen Eingabe- und Ergebnisdaten sowie Zwischenzuständen während der Berechnung.
- verwenden zur eigentlichen Lösung der gegebenen Aufgabe funktionale und/oder imperative Algorithmen.