Algorithmen und Datenstrukturen

– Sommersemester 2019 –

Kapitel 01: Algorithmen – Grundlagen

Prof. Dr. Adrian Ulges

B.Sc. AI / ITS / WI
Fachbereich DCSM
Hochschule RheinMain

Outline

Google und Facebook

Feindbild Algorithmus

Der Wissenschaftliche Dienst des Bundestages erkennt "Bedarf" an einer Regulierung der Algorithmen. Aber schon die seltsame Fixierung auf den Begriff ist irreführend.

Von **Patrick Beuth**

14. Oktober 2017, 8:16 Uhr / 167 Kommentare

Sind Algorithmen wirklich so böse wie er? © *Matt Cowan/Getty Images*

Künstliche Intelligenz

Deutschland braucht mehr Algorithmenkompetenz

Künstliche Intelligenz verändert unsere Gesellschaft gewaltig. Leider verstehen die meisten Menschen nicht, wie Algorithmen funktionieren. Das muss sich dringend ändern!

Ein Gastbeitrag von Jörg Dräger und Ralph Müller-Eiselt

Automatisierung

Hälfte der EU-Bürger weiß nicht, was ein Algorithmus ist

Im Internet führt an ihnen zwar kaum ein Weg vorbei – dennoch wissen die Menschen wenig über Algorithmen. Die meisten Europäer verlangen nach einer wirksameren Kontrolle. ★ 44

06.02.19 08:18 | Diginomics

Klänge zum Fest

Die Musik der Algorithmen

Was hören wir zum Fest? Das Weihnachtsoratorium von Bach oder neue Schöpfungen, die ein Computer komponiert hat? Die Musik-Software findet neue Wege in bekannten Gebieten. Mehr› Von MICHAEL SPEHR

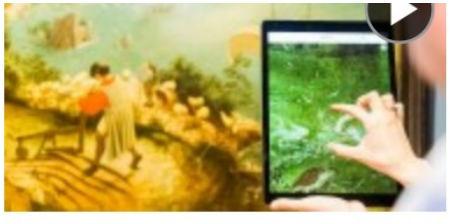
7 ★ 14

24.12.18 08:03 | Digital

„Die Muße ist ein Algorithmus“

Was passiert, wenn Algorithmen Bilder malen, neue Formen generieren und Skulpturen schaffen. Ist das wirklich Kunst oder nur eine Kopie? Mehr›

16.04.19 07:10 | Feuilleton



Grundlegende Definitionen

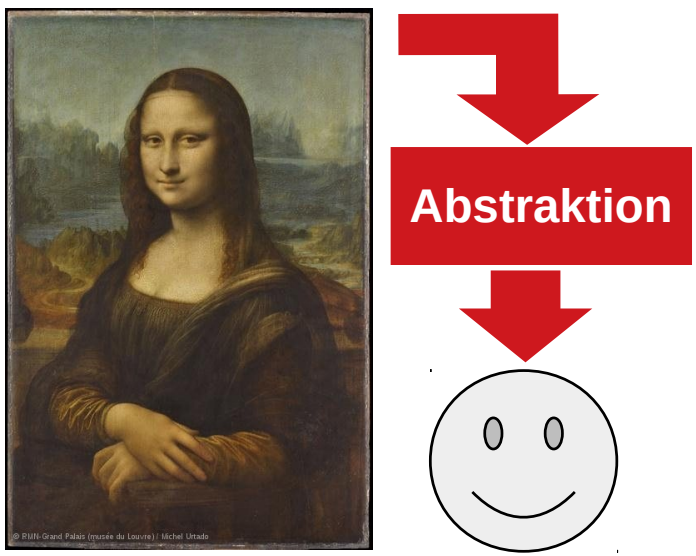
Was bedeutet "Informatik"?

> *"Computer Science: The Mechanization of Abstraction"*
>
> (Aho, Ullmann: Foundations of Computer Science)

- Informatik = Systematische **Informationsverarbeitung** *(insbesondere mit Digitalrechnern)*.
- **Zwei Schlüsselschritte**: Abstraktion und Mechanisierung.

Häufiges Vorgehen in der Informatik

1. Analyse des Problems
   (Anforderungsanalyse)
2. Spezifikation des Problems
   (Pflichtenheft)
3. Spezifikation der Lösung
   ("halb-formal")
4. Implementierung
   (Programmiersprache).



Grundlegende Definitionen

Was ist ein Algorithmus?

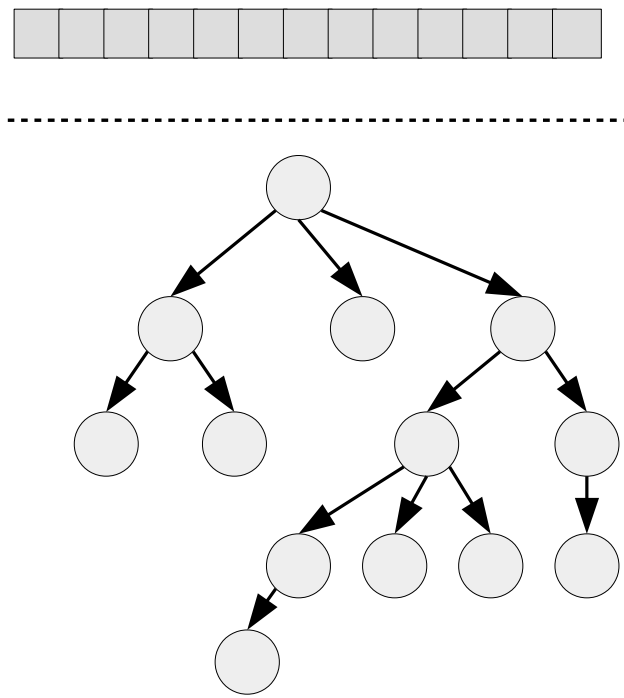
- Algorithmus = "wohldefinierte" **Handlungsvorschrift zur Lösung eines Problems** *(siehe nächste Seite)*.
- Die **Ausführung** eines Algorithmus (auf einem *Prozessor*) erzeugt Ausgabeobjekte aus Eingabeobjekten.

Definition "Programm"

- Programm = **Umsetzung** eines Algorithmus in einer konkreten Programmiersprache *(z.B. Java)*.

Definition "Datenstruktur"

- Organisation der Eingabe- und Ausgabe-Objekte.



Definition “Algorithmus”

Definition (Algorithmus (Saake, Sattler [4]))

Ein Algorithmus ist eine...

- *präzise (d.h. in festgelegter Sprache)*
- *endliche Beschreibung eines*
- *allgemeinen Verfahrens unter Verwendung*
- *ausführbarer elementarer Schritte.*

Beispiele

- Bedienungsanleitungen, Bauanleitungen (IKEA)
- Kochrezepte
- Spielanleitungen
- Noten/Partituren
- Vorschriften zur Verarbeitung von Daten *(hier in ADS)*.

“Algorithmus”: Beispiel

Präzise?

- z.B. Eingabeobjekte genauer definieren. *Rohes Ei?*

Endlich?

- ja (6 Schritte).

Allgemeines Verfahren?

- Einschränkungen der Eingabeobjekte? *(kleine Eier?)*

Ausführbare, elementare Schritte?

- Kommt auf den Prozessor an ;-)

```
# Ei kochen

1. Wasser in einem Topf
    zum Kochen bringen.

2. Ei in ein Sieb legen.

3. Sieb in Wasser senken.

4. Wenn Ei mittelgroß ist, dann
   9 min kochen, wenn Ei groß
   ist, dann 10 min.

5. Sieb aus kochendem Wasser
    nehmen.

6. Sieb in kaltes Wasser
    tauchen.
```

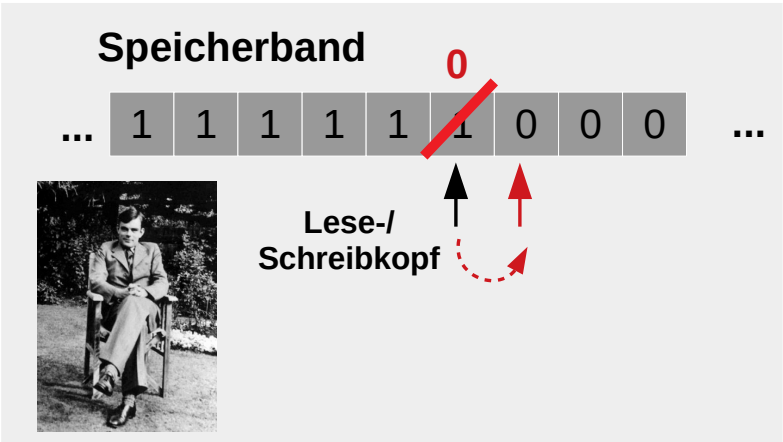
Ist das ein Algorithmus?

“Algorithmus”: Beispiel (Turing-Maschine) image: [1]

Andere Algorithmen-Darstellungen sind **deutlich formaler**:

Die Turing-Maschine (eine abstrakte Maschine)

- **Speicher**: ein unendliches Band.
- **Operationen**:
  - **Lese** 0/1 vom Band und wähle die nächste Aktion.
  - **Bewege** dich auf dem Band einen Schritt nach links/rechts.
  - **Schreibe** 0/1 auf das Band.



Church’sche These

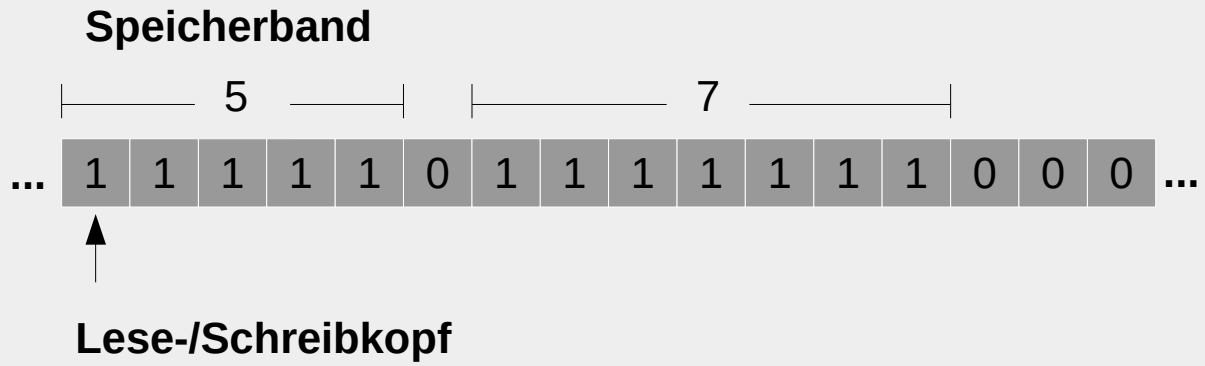
- Die (Ausdrucks-)Mächtigkeit von Algorithmen in einer **beliebigen Programmiersprache** ist **gleich** der Mächtigkeit von **Turing-Maschinen**.
- **Algorithmen** = alles was programmierbar ist = alles was eine Turing-Maschine ausführen kann.
- These nicht bewiesen *(schwierig!)*, aber allgemein anerkannt.

Turing-Maschine: Beispiel

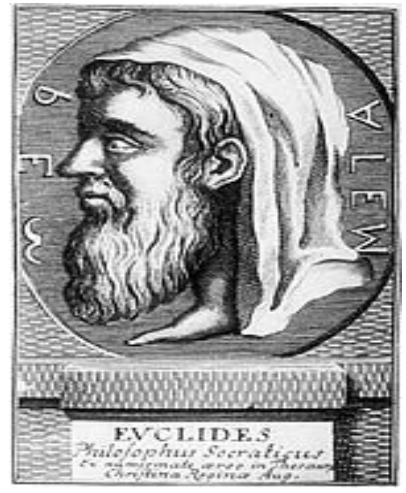
Was tut diese Turing-Maschine?

Programm

1	0	1	R	2
	1	1	R	1
2	0	0	L	3
	1	1	R	2
3	0	0	L	-
	1	0	L	-



Historie des Rechnens



300 v. Chr.: Euklid

- ältester bekannter Algorithmus (Bestimmung des größten gemeinsamen Teilers zweier Zahlen)

800 n. Chr.: Muhammed ibn Musa abu Djafar **al-Chwarizmi**

- Buch "Al-Chwarizmi über die indischen Zahlen" → "Algorithmi de Numero Indorum"

1550: Adam Riese

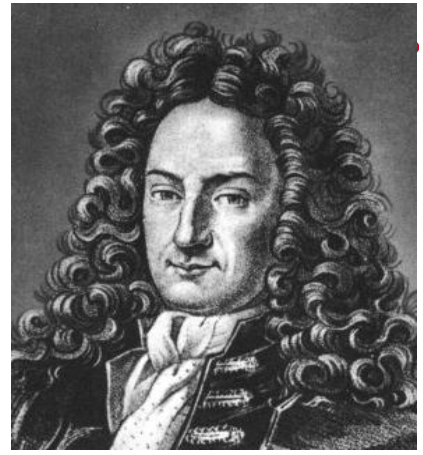
- Deutschsprachiges Rechenbuch[1], Ablösung der lateinischen Ziffern.

[1] http://www.tinohempel.de/info/mathe/ries/ries.htm

Historie des Rechnens

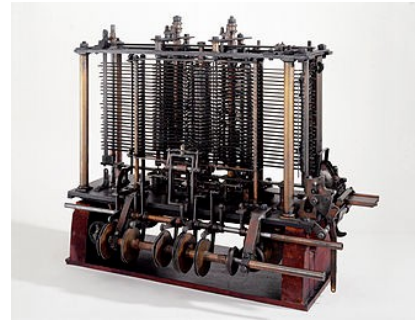
1703: G.W. Leibniz

- Duales Zahlensystem
  (erste technische Nutzung: 1930er)
- Differentialrechnung
  (parallel zu Isaac Newton).



1842: Charles Babbage

- Entwurf der *Analytical Engine*
- Erste Rechenmaschine äquivalent einer Turing-Maschine *(u.a. branching und loops)*, programmierbar mit Lochkarten.



1931: Kurt Gödel

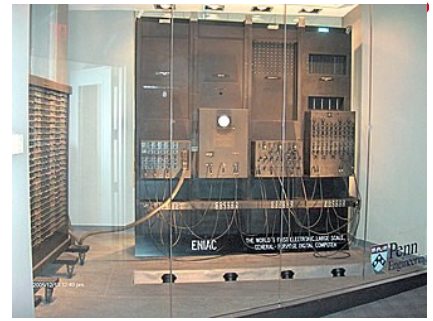
- Unvollständigkeitssatz: In ausreichend mächtigen Rechensystemen *(z.B. Arithmetik über* $\mathbb{N}$*)* gibt es Aussagen, die wir nicht automatisiert beweisen können.

Historie des Rechnens

1940er: Erste Computer

- Z3 (Zuse, Berlin)
- Colossus (Turing et al., Bletchley / UK)
- ENIAC (US Army).

ENIAC



1962: Donald Knuth – TAoCP

- TAoCP = "The Art of Computer Programming"
- ADS-"Bibel" *(Suchen, Sortieren, Datenstrukturen, Scannen, Parsen, ...)*
- Algorithmen in fiktiver Assembler-Sprache.

1973: Niklaus Wirth – Pascal

- **Strukturierte** Programmierung *(Zerlegung von Programmen in Prozeduren, lokale Scopes, Verbot von Goto-Anweisungen)*.
- Anmerkung: **Objektorientierte** Programmierung → seit den 1990ern.

Outline

Terminierung

Definition (Terminierung)

Wir nennen einen Algorithmus **terminierend** *wenn er bei* **jeder erlaubten Eingabe** *nach* **endlich vielen** *Schritten* **abbricht**.

Beispiele

- Der Algorithmus "Ei kochen" *(siehe oben)* ist terminierend.
- Ein Algorithmus mit einer **Endlosschleife** ist nicht terminierend.
- Addition von beliebigen natürlichen Zahlen mittels Nachschauen in einer **Tabelle**...
  - ist **terminierend** *(Nachschauen in Tabelle = 1 Schritt)*
  - ist aber **kein Algorithms** *(Beschreibung/Tabelle wäre unendlich)*.

Determinismus und Determiniertheit

Definition (Determinismus)

Wir nennen einen Algorithmus **deterministisch** *wenn bei gleicher Eingabe exakt dieselben* **Schritte/Zustände** *durchlaufen werden.*

Definition (Determiniertheit)

Wir nennen einen Algorithmus **determiniert** *wenn bei gleicher Eingabe immer dasselbe* **Ergebnis** *erzielt wird.*

Beispiele

- Algorithmus **“Ei kochen”**: deterministischer Ablauf, determiniertes Ergebnis.
- Algorithmus **“Karten sortieren”** *(rechts)*: nicht-deterministisch, determiniertes Ergebnis.

```
# S sei ein
# unsortierter
# Kartenstapel

S' := ein leerer
      Kartenstapel

wiederhole:

  k := eine zufällige Karte aus S

  Entferne k aus S.

  Sortiere k an passender Stelle in S'
  ein, so dass S' sortiert ist.

bis S leer.    # S' enthält nun die
               # sortierten Karten.
```

Determinismus

Anmerkungen

- Die meisten hier betrachteten Algorithmen sind **determiniert und terminierend**. Wir können uns einen solchen Algorithmus als eine **Funktion f^{Alg}** vorstellen:

$$f^{Alg} : \text{Eingabewerte} \rightarrow \text{Ausgabewerte}.$$

Spezifikation

- In der Praxis sollen Algorithmen ein **konkretes Problem lösen**. Wir **spezifizieren** dieses Problem, indem wir angeben für welche Eingaben wir welche Ausgaben erwarten.
- Dieses Soll-Verhalten entspricht ebenfalls einer **Funktion**. Wir bezeichnen Sie als die **Spezifikation** des Algorithmus:

$$f^{Spez} : \text{Eingabewerte} \rightarrow \text{Ausgabewerte}.$$

Korrektheit

Definition (Korrektheit)

Wir nennen einen Algorithmus genau dann **korrekt** *wenn er für* **jede mögliche Eingabe** x *mit der durch die* **Spezifikation** *geforderten Ausgabe terminiert, d.h.* $f^{Alg}(x) = f^{Spez}(x)$.

Beispiel

- Prüfe ob a ein Teiler von b ist.
  $f^{Spez} : \mathbb{Z} \times \mathbb{Z} \to \{true, false\}$ mit

$$f^{Spez}(a, b) = \begin{cases} true & \text{falls } a|b \\ false & \text{sonst} \end{cases}$$

- **Algorithmus**: Prüft $1 \cdot a$, $2 \cdot a$, $3 \cdot a$, ... auf Gleichheit mit b.
- **Ist der Algorithmus korrekt?**

```
# prüft ob a ein
# Teiler von b ist.

test = a

while test <= b:

  if test == b:
     return true

  test += a

return false.
```

Korrektheit und Testen

Wie zeigen wir Korrektheit?

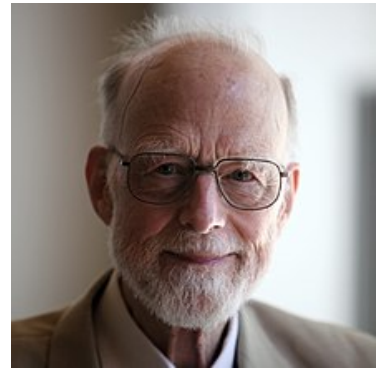
1. Testen
   - In der Praxis sehr verbreitet.
   - Programmierung von **Testfällen**.
   - **Prüfung** dass in diesen Fällen das gewünschte Ergebnis erzielt wird.
   - **Tests können Nicht-Korrektheit zeigen** *(wenn ein Test fehlschlägt).*
   - **Tests zeigen im Allgemeinen keine Korrektheit!** *(wir können nicht alle Eingaben prüfen).*

Beispiel

- Die Tests sind alle erfolgreich. ☺
- Negative Werte vergessen ☹
  (für $a \leq 0 \wedge b > a$ nicht-terminierend).

```
# prüft ob a ein
# Teiler von b ist.

test = a

while test <= b:

  if test == b:
     return true

  test += a

return false.


#########################
#####  Testfälle  #####
#########################

# a ist Teiler
assert check(3, 72) == true

# a ist kein Teiler
assert check(3, 73) == false

# a == b
assert check(10, 10) == true

# a == 1, b groß
assert check(1, 10000) == true

# a > b
assert check(10, 9) == false
```

Korrektheit und Verifikation image: [2]



2. Verifikation

Beweise Korrektheit (*händisch oder automatisiert*).

Automatisierte Verifikation: Der **Hoare-Kalkül** [3]

- Wir definieren Vorbedingungen P *(engl. "pre-conditions)* an die Eingabe.
- Beim Ausführen imperativer Rechenoperationen S gelten bestimmte *Axiome*.
- Hieraus leiten wir Nachbedingungen Q *(post-conditions)* für die Ausgabe her.

$$\underbrace{\{x < 42\}}_{P} \quad \underbrace{\texttt{y := x + 1}}_{S} \quad \underbrace{\{y < 43\}}_{Q}$$

Beispiel: Primzahltest

Spezifikation

- Eine Zahl $n \geq 2$ ist **prim** wenn sie *(außer sich selbst)* keine Teiler ≥ 2 besitzt.
- Gegeben eine Zahl $n \geq 2$, prüfe ob n prim ist.

Eigenschaften des Algorithmus

- deterministisch (es werden *bei gleicher Eingabe* immer die gleichen Schritte durchlaufen).
- determiniertes Ergebnis.
- Terminierend?
- Korrekt?

```
# Primzahltest

1. Weise x den Wert 2 zu.

2. Wenn x gleich n ist,
   dann gebe true zurück
   und terminiere.

3. Dividiere n durch x mit Rest

4. Wenn diese Division den
   Rest 0 ergibt, dann gebe false
   zurück und terminiere.

5. Erhöhe den Wert von x um 1
   und gehe zu Schritt 2.
```

Primzahltest: Analyse

Primzahltest

1. Weise x den Wert 2 zu.

2. Wenn x gleich n ist,
   dann gebe **true** zurück
   und terminiere.

3. Dividiere n durch x mit Rest

4. Wenn diese Division den
   Rest 0 ergibt, dann gebe **false**
   zurück und terminiere.

5. Erhöhe den Wert von x um 1
   und **gehe zu Schritt 2.**

Primzahltest: Analyse

Primzahltest

1. Weise x den Wert 2 zu.

2. Wenn x gleich n ist,
   dann gebe **true** zurück
   und terminiere.

3. Dividiere n durch x mit Rest

4. Wenn diese Division den
   Rest 0 ergibt, dann gebe **false**
   zurück und terminiere.

5. Erhöhe den Wert von x um 1
   und **gehe zu Schritt 2.**

Primzahltest: Analyse

Outline

Bausteine von Algorithmen

Elementare Operationen

... sind Schritte die nicht weiter zerlegt werden müssen.

- *"Schneide Fleisch in kleine Würfel"*
- **Im Rechner**: Zuweisungen, Additionen, Vergleiche, ...

Sequenzen von Operationen

... sind Schritte die nacheinander ausgeführt werden.

- *"Bringe Wasser zum Kochen und lege dann das Ei hinein."*
- **Im Rechner**: Anweisungssequenzen, z.B. `y = x+1; z = 2*y;`

Parallele Ausführung

... mehrere Prozessoren arbeiten gleichzeitig an Teilschritten:

- *"Ich schneide Fleisch, Du Gemüse. Anschließend geben wir beides gleichzeitig in die Pfanne."*
- **Im Rechner**: Threads, Prozesse.

Bausteine von Algorithmen (cont'd)

Bedingte Operationen

... Schritte werden nur ausgeführt wenn Bedingungen erfüllt sind.

- *"Wenn die Soße zu dünn ist, dann füge Mehl hinzu".*
- **Im Rechner**: if-else-Konstrukte.

Schleifen

... Wiederholung einer Ausführung bis Endbedingung erreicht.

- *"Rühre, bis die Soße braun ist"*
- **Im Rechner**: z.B. `while x >= 10: x = x-1;`

Unterprogramm/Prozedur/Funktion/Methode

... Ausführen eines benamten Teilalgorithmus.

- *"Koche Nudeln (siehe Seite 42)"*
- **Im Rechner**: z.B. `abs(x)`

Rekursion

... Reduktion auf kleineres, ähnliches Teilproblem.

- **schneideKäse(käse)**: teile Käse in zwei Hälften. Falls Käse noch nicht klein genug: schneideKäse(linkeHälfte) und schneideKäse(rechteHälfte).

Outline

Pseudo-Code...

... stellt das **Wesentliche eines Algorithmus** prägnant dar.

- Die Darstellung ist unabhängig vom Prozessor *(Abtraktion!)*. **Details** der maschinellen Verarbeitung werden **ausgelassen** *(z.B. Variablen-Deklarationen, einfache Subroutinen wie* `sqrt`*)*.

```
# S sei ein
# unsortierter
# Kartenstapel

S' := ein leerer
      Kartenstapel

wiederhole:

  k := eine zufällige Karte aus S

  Entferne k aus S.

  Sortiere k an passender Stelle in S'
  ein, so dass S' sortiert ist.

bis S leer.    # S' enthält nun die
               # sortierten Karten.
```

- **Mischung** von (1) Elementen von Programmiersprachen, (2) natürlicher Sprache und (3) mathematischer Notation.

Wozu Pseudo-Code?

Pseudo-Code dient der Kommunikation zwischen Informatikern:

- **Ziel**: Vollständige und eindeutige Beschreibung, die **einfach** in eine Programmiersprache **überführbar** ist.
- Es existiert **keine formale Spezifikation**! *("... Pseudo-Code ist das was angemessen ist").*

Pseudo-Code: Konventionen

Anmerkungen

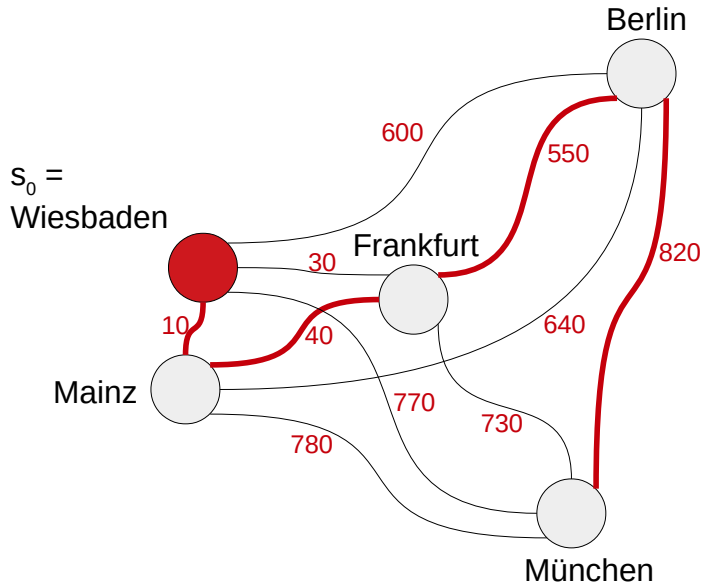
- Pseudo-Code ist gemäß guter Praxis eng an **Programmiersprachen** angelehnt. Optionen sind z.B. ...

```
// Java
if (a==b) { a = c; }

# Python
if a==b :
    a = c

// Pascal
if a = b then begin a := c; end
```

- Pseudo-Code ist oft genauer und klarer als natürliche Sprache *(u.a. eindeutige Benamung der verwendeten Objekte, eindeutige Abbruchkriterien für Schleifen ...)*.

Pseudo-Code: Beispiel “Routenplaner”

```
# Routenplaner (Version 1)

Beginne mit der Route in Stadt s_0

wiederhole:

  finde die nächstgelegene Stadt s.

  Füge s der Route hinzu.

bis alle Städte besucht.
```



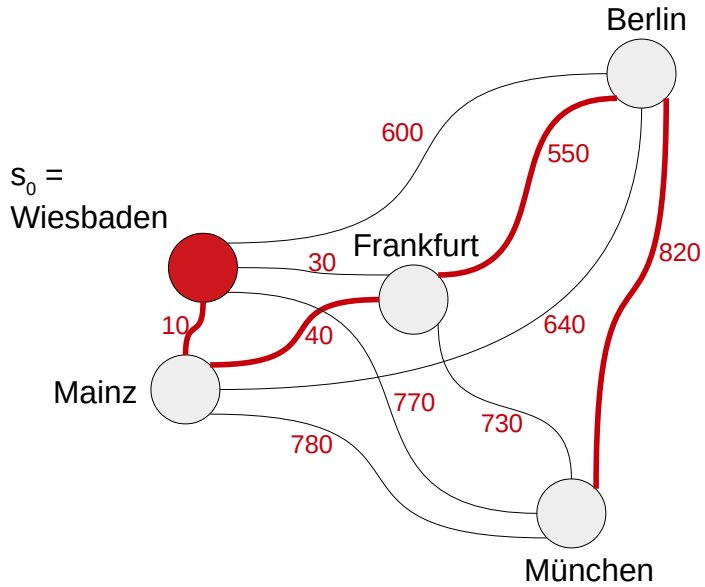
Version 1

Hier existieren **Mehrdeutigkeiten / Unklarheiten**:

- Definition “Route”? Wie sind Ein- und Ausgabe strukturiert?
- Naive Interpretation des Codes führt zu Endlossschleife *(Wiesbaden → Mainz → Wiesbaden → Mainz → ...)*

Pseudo-Code: Beispiel "Routenplaner"

```
# Routenplaner (Version 2)
# Gegeben:
# - S: Die Menge zu besuchender Städte
# - s_0: Die Anfangsstadt der Route

stadt := s_0
route := []  # leere Liste


while S <> {}:
  stadt := argmin_{s∈S} cost(stadt,s)
  route := route + [stadt]
  S     := S\{stadt}

return route
```



Version 2

- **Formalisierung**
  - zu besuchende Städte als Menge S
  - Route als Liste
  - Kostenfunktion $cost : S \times S \to \mathbb{R}^+$
- **Präziser**, eindeutiger *(mathematische Notation)*.
- Unmittelbar in ein Programm **umsetzbar** *(wenn geeignete Datenstrukturen für Listen und Mengen bekannt → später)*.

Outline

Flussdiagramme

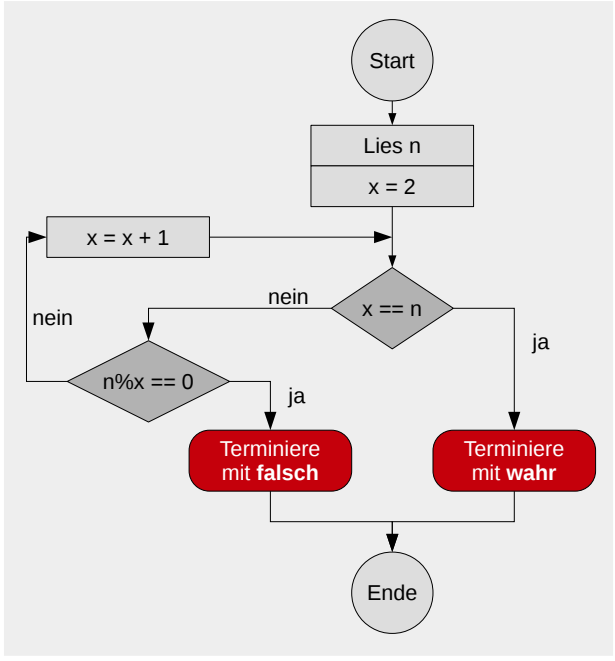
... stellen den Kontrollfluss eines Algorithmus graphisch *(mittels Pfeilen)* dar.

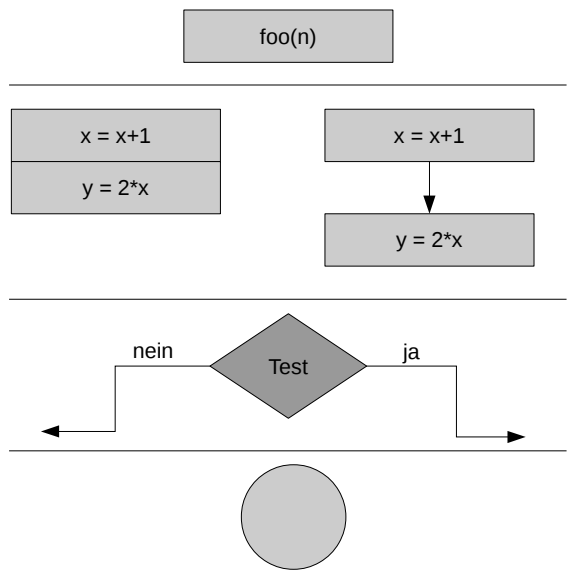
Elemente von Flussdiagrammen (rechts unten)

- Elementare Anweisungen, Unterprogrammaufrufe
- Sequenz
- Verzweigung
- Start, Ende

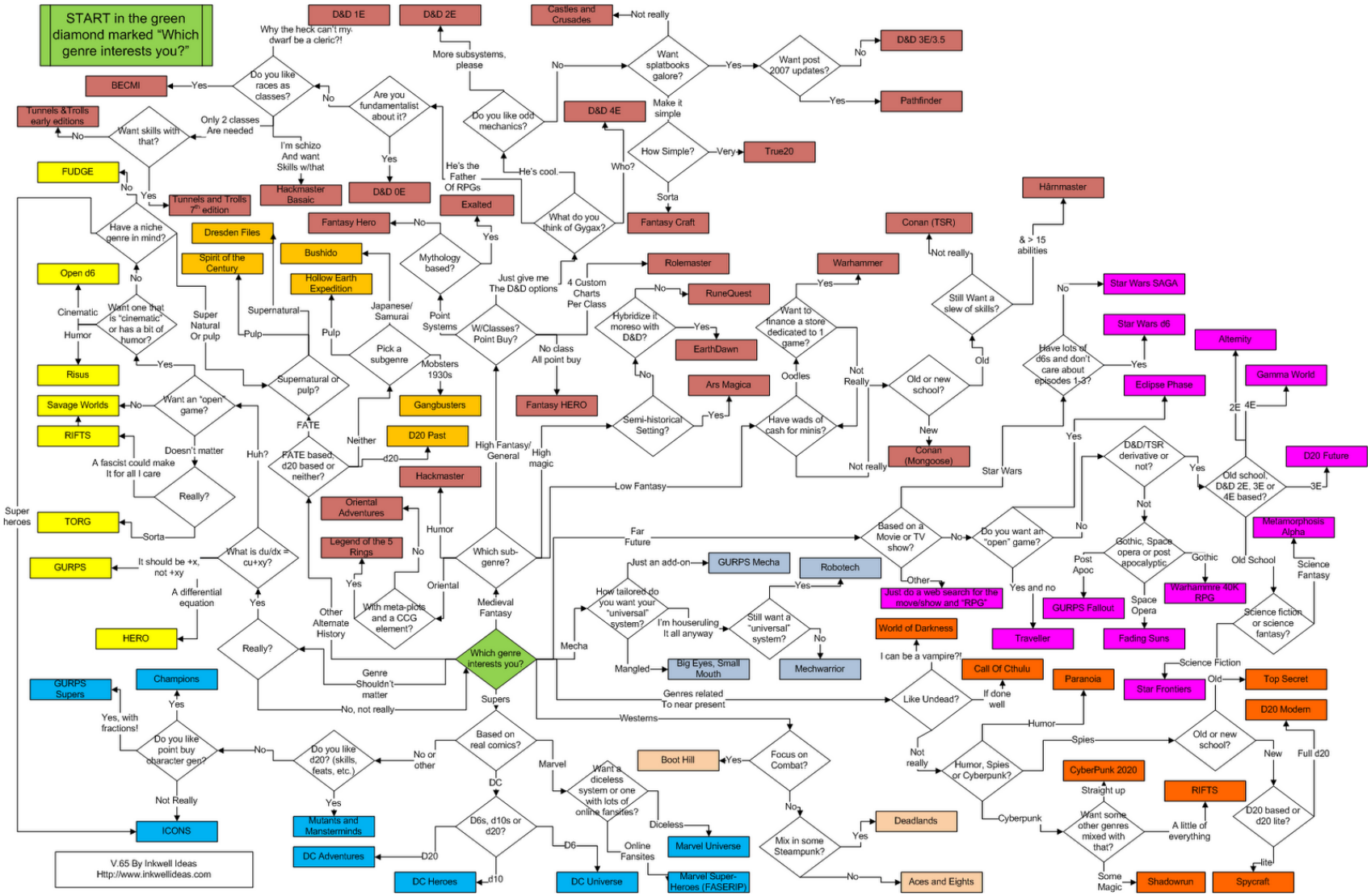
Diskussion

- keine dedizierten Schleifenkonstrukte
- häufig unübersichtlich *(für komplexere Algorithmen vermeiden)*.





Flussdiagramm: Beispiel



Outline

Struktogramme (Nassi-Schneiderman-Diagramme)

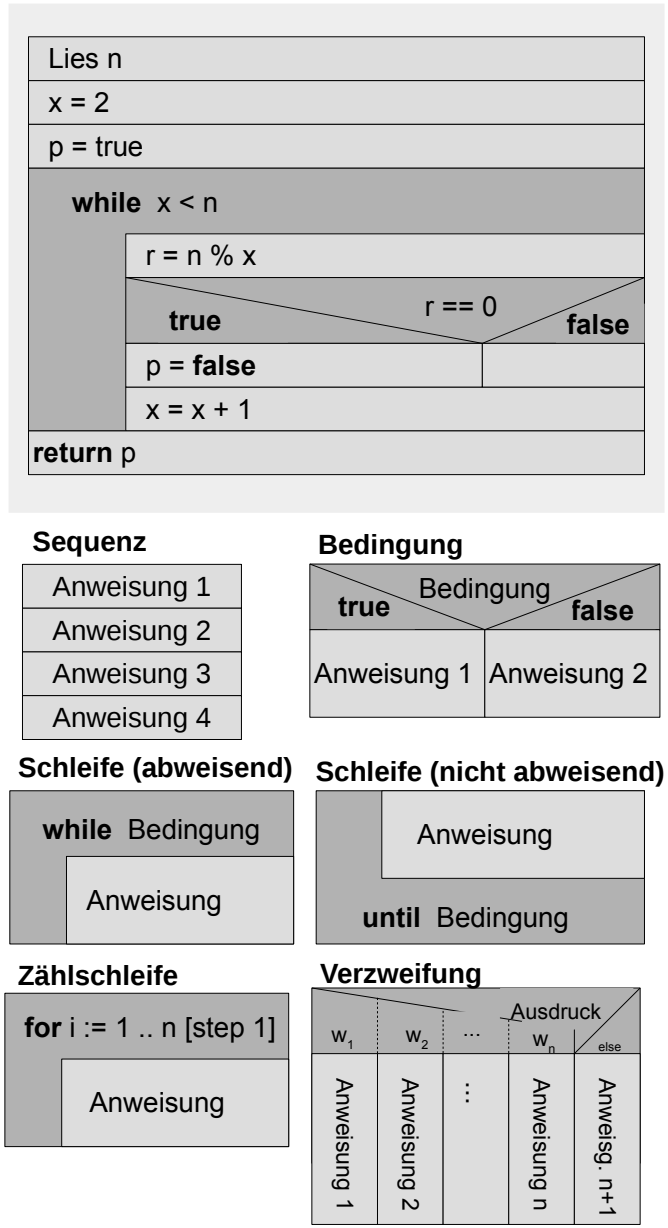
- sind *(ähnlich wie Pseudo-Code und und Flussdiagramme)* eine halbformale Darstellung.
- Zerlegung des Gesamtalgorithmus in **Teilprobleme** *(strukturierte Programmierung)*. Separates Struktogramm je Teilproblem.

Elemente

- Sequenzen, Verzweigungen
- Schleifen: Abweisend, nicht-abweisend, Zählschleifen

Diskussion

- Einfach ständlich.
- Jede Struktur hat einen klar definierten Eingang+Ausgang, Sprünge nicht erlaubt.



References I

[1] Alan Turing (ca. 1938).
https://de.wikipedia.org/wiki/Alan_Turing#/media/File:Alan_Turing_az_1930-as_%C3%A9vekben.jpg (retrieved: Apr 2019).

[2] Sir Charles Antony Richard Hoare giving a talk at the EPFL on 20th of June 2011.
https://de.wikipedia.org/wiki/Tony_Hoare#/media/File:Sir_Tony_Hoare_IMG_5125.jpg (retrieved: Apr 2019).

[3] C. A. R. Hoare.
An axiomatic basis for computer programming.
Commun. ACM, 12(10):576–580, October 1969.

[4] G. Saake and K.U. Sattler.
Algorithmen und Datenstrukturen: Eine Einführung mit Java.
Dpunkt.Verlag GmbH, 2013.