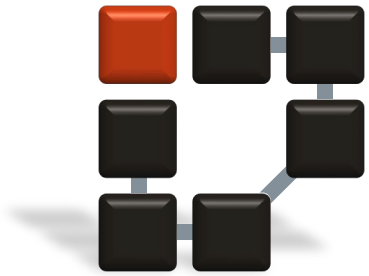
Informatik 1 für Nebenfachstudierende Grundmodul

Java – Algorithmische Verfahren

Kai-Steffen Hielscher
Folienversion: 14. Januar 2020



Informatik 7
Rechnernetze und
Kommunikationssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Prinzipien algorithmischer Verfahren

- basiert auf
  Sebastian Dörn: *Java lernen in abgeschlossenen Lerneinheiten*, Springer, 2019.
  https://doi.org/10.1007/978-3-658-24003-5

Prinzipien algorithmischer Verfahren

- Computerprogramme implementieren einen Algorithmus in einer bestimmten Programmiersprache (z.B. Java)
- vgl. Kapitel 14: Java – Einführung
- Algorithmus bekommt Eingaben, produziert nach bestimmter, **endlicher Zeit** Ausgaben

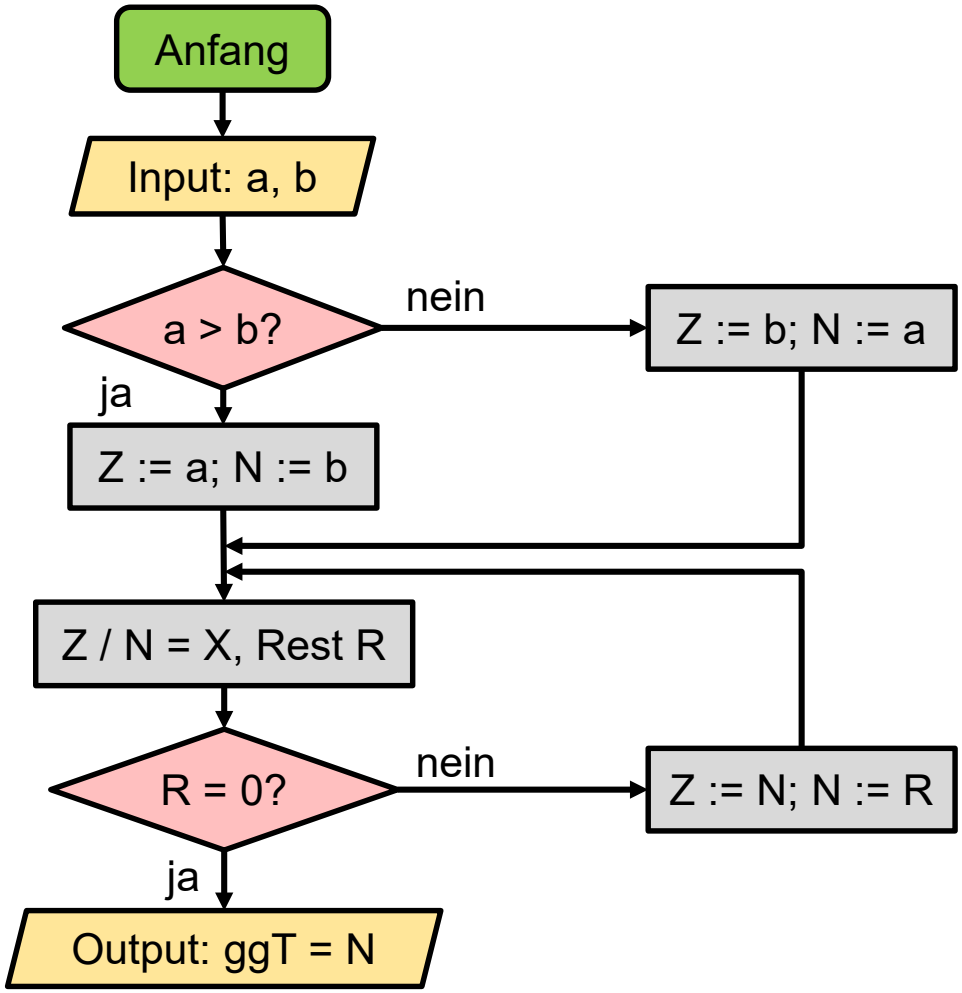
- Aktionen
  - Zuweisen von Werten an Variablen
  - Ein- und Ausgabe von Werten
  - Verzweigen von Anweisungen
  - Wiederholen von Anweisungen
  - Aufruf von Unterprogrammen
  - Einlesen und Schreiben von Dateien

Algorithmen

- Eigenschaften von Algorithmen
  - **Korrektheit:**
    korrekte Arbeitsweise, schwierig zu prüfen
  - **Vollständigkeit:**
    vollständige, endliche Beschreibung eines Lösungsverfahrens, Rahmenbedingungen erfüllen.
  - **Eindeutigkeit:**
    Aktionen des Algorithmus eindeutig ohne Interpretationsspielraum ausführbar
  - **Effizienz:**
    Rechenzeit Speicherplatz, Laufzeit durch Anzahl auszuführender Operationen ermittelbar
  - **Verständlichkeit:**
    Fehlerquellen vermeiden, Wartbarkeit sicherstellen

Algorithmen

- Darstellung als **Flussdiagramm** (ggT)



Algorithmen

- Darstellung als **Pseudocode** (Array-Summe)

Input: Array $a = (a_1, \ldots, a_n)$
Output: Summe s
1: $s = 0$
2: **for** $i = 1$ **to** n **do**
3: $\quad s = s + a_i$

- textuelle Notation
- an Programmiersprachen angelehnt
- ohne syntaktischen Ballast

Entwurf von Computerprogrammen

- **Problemanalyse:**
  Präzises Formulieren der gegebenen Aufgabenstellung mit den gegebenen und gesuchten Kenngrößen
- **Algorithmenentwurf:**
  Ableiten eines möglichst kurzen, verständlichen und leicht veränderbaren Verfahrens mit Hilfe des Top-Down-Entwurfs
  - **Top-Down-Entwurf:**
    Entwickler zerlegt jede einzelne Teilaufgabe solange, bis sie so einfach ist, dass sie mit Hilfe von elementaren Aktionen und Anweisungen lösbar ist.

Entwurf von Computerprogrammen

- **Korrektheitsnachweis:**
  Prüfen der Ergebnisse auf Richtigkeit und auf das Erfüllen aller Nebenbedingungen bzw. Ausnahmesituationen
- **Aufwandsanalyse:**
  Analyse des Zeit- bzw. Speicherbedarfs des Verfahrens
- **Programmkonstruktion:**
  Implementieren des Algorithmus in einer geeigneten Programmiersprache
- **Test:**
  Überprüfen des Programms mit einer repräsentativen Menge von Testfällen

Entwurf von Computerprogrammen

- **Dokumentation:**
  Dokumentation des Programms mit den zugehörigen Unterroutinen und Schnittstellen

Standard-Algorithmen

- können in eigenen Programmen verwendet werden
- lösen Standardprobleme effizient
  - Teilaufgabe in einem größeren Programm
  - wiederverwendbar
- wichtige Grundlage in der Informatik
  - Korrektheit
  - Komplexität
    - effiziente Algorithmen
  - ein Standardwerk:
    Cormen, Leiseron, Rivest, Stein: *Introduction to Algorithms*, MIT Press, 3rd ed., 2009.
    - bekannt als Cormen, Leiserson, Rivest (**CLR**), 1. Auflage
      oder Cormen, Leiseron, Rivest, Stein (**CLRS**)
- Beispiele auf den folgenden Folien

Maximum eines Arrays

- Initialisierung des Maximums
  - erstes Element des Arrays
- Array komplett durchlaufen
- prüfen, ob aktuelles Element größer ist als momentanes Maximum
- wenn ja, wird aktuelles Element neues Maximum
- am Ende Maximum zurückliefern

```
1 public static double maximum(double[] a) {
2     double max = a[0];
3     for(double d : a) {
4         if(d>max) {
5             max = d;
6         }
7     }
8     return max;
9 }
```

Maximum eines Arrays

- **Index** des Maximums gesucht, Vorgehen analog
  - andere Form der `for`-Schleife erlaubt Zugriff auf aktuellen Index
- Maximum **und Index** merken
- am Ende Index zurückliefern
  - Rückgabe von Maximum und Index schwieriger (z.B. eigenes Objekt*)

```
1  public static int maxindex(double[] a) {
2      double max = a[0];
3      int idx = 0;
4      for(int i=0; i<a.length; i++) {
5          if(a[i]>max) {
6              max = a[i];
7              idx = i;
8          }
9      }
10     return idx;
11 }
```

Maximum eines Arrays

- Verwendung der Methoden in Hauptprogramm

```
1  public class ArrayMax {
2      public static double maximum(double[] a) {
3          …
4      }
5
6      public static int maxindex(double[] a) {
7          …
8      }
9
10     public static void main(String[] args) {
11         double[] zahlen = {5.25, 2.5, 1.41,
12             -7.38, 3.14159, 10.0, -22.222, 0.0};
13         System.out.println("Das Maximum ist "+
14             maximum(zahlen)+".");
15         System.out.println("Es befindet sich an Position "+
16             maxindex(zahlen)+".");
17     }
18 }
```

Maximum eines Arrays

- Ausgabe des Programms bei Ausführung

```
Das Maximum ist 10.0.
Es befindet sich an Position 5.
```

*Maximum und Index in einem Array mit eigenem Objekt

```java
public class ArrayMaxObj {

    public static class MaxIdx {
        public double max;
        public int index;
    }

    public static MaxIdx maxindex(double[] a) {
        MaxIdx mi = new MaxIdx();
        mi.max = a[0];
        mi.index = 0;
        for(int i=0; i<a.length; i++) {
            if(a[i]>mi.max) {
                mi.max = a[i];
                mi.index = i;
            }
        }
        return mi;
    }

    public static void main(String[] args) {
        double zahlen[] = {5.25, 2.5, 1.41, -7.38, 3.14159, 10.0, -22.222, 0.0};
        MaxIdx mi = maxindex(zahlen);
        System.out.println("Das Maximum ist "+mi.max+".");
        System.out.println("Es befindet sich an Position "+mi.index+".");
    }
}
```

Bestimmen der Ziffern einer Zahl

- Dezimalstellen einer ganzen Zahl
- jede Stelle: Divisionsrest bei Division durch 10
  - Modulo-Operator `%`
- nächste Stelle:
  - Zahl durch 10 dividieren (Ganzzahldivision)
  - Vorgang wie oben wiederholen (Modulo-Operation)
- Anzahl der Stellen: (int) $\log_{10}+1$
  - so oft müssen die Modulo- und Divisionsoperationen ausgeführt werden
  - Typecast `(int)`: explizite Typumwandlung einer Gleitkommazahl in eine Ganzzahl

Bestimmen der Ziffern einer Zahl

```java
public class Ziffern {
    public static int[] ziffern(int zahl) {
        int n = (int) Math.log10(zahl) + 1;
        int[] z = new int[n];
        for(int i=0; i<n; i++) {
            z[i] = zahl % 10;
            zahl /= 10;
        }
        return z;
    }

    public static void main(String[] args) {
        int x = 123450;
        int[] a = ziffern(x);
        for(int i=a.length-1; i>=0; i--) {
            System.out.print(" "+a[i]);
        }
        System.out.println();
    }
}
```

Bestimmen der Ziffern einer Zahl

- Niederwertigste Stelle entsteht zuerst (1er)
- Ausgabe in umgekehrter Reihenfolge
  - höchstwertige Ziffer zuerst
  - "umgekehrte" `for`-Schleife
- Ausgabe des Programms bei Ausführung

```
1 2 3 4 5 0
```

- Was müsste man für andere Basen als 10 ändern?

Suche eines bestimmten Elements in einem Array

- Methode soll den Index eines gesuchten Elements in einem Array liefern
- Array unsortiert
  - gesamtes Array durchlaufen
  - jedes Element überprüfen
  - **lineare Suche**

Lineare Suche

- AlgoRythmics:
  LINEAR search with FLAMENCO dance
  - https://youtu.be/-PuqKbu9K3U
  - am 29.01.2018 veröffentlicht
  - Created at Sapientia University, Targu Mures (Marosvásárhely), Romania.
  - Directed by Kátai Zoltán, Osztián Erika, Osztián Pálma-Rozália, Vekov Géza-Károly.
  - In cooperation with András Lóránt Company, Targu Mures (Marosvásárhely), Romania.
  - Choreographer: András Lóránt.
  - Video: Márton László.

Suche eines bestimmten Elements in einem Array

- Array sortiert
  - Vorgehensweise wie bei unsortierten Array möglich, aber **ineffizient**
  - **Binäre Suche**
    - Prinzip der Intervallhalbierung
      - Element in der Mitte des Bereichs ansehen
      - ist Mitte gesuchtes Element, Index der Mitte ist Ergebnis
      - ist gesuchtes Element größer als Mitte: rechten Bereich weiter untersuchen
      - ist gesuchtes Element kleiner als Mitte: linken Bereich weiter untersuchen
      - im jeweiligen Bereich wieder Mitte untersuchen, usw.

Binäre Suche

- Beispiel: Suche nach 21

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
2	3	6	7	12	21	55
l=0			m=3			r=6
				l=4	m=5	r=6

  - naive Vorgehensweise: 6 Vergleiche,
    im ungünstigsten Fall 7 Vergleiche bei Array mit 7 Elementen
  - binäre Suche: 2 Vergleiche (effizienter),
    im ungünstigsten Fall 3 Vergleiche bei Array mit 7 Elementen
- Binäre Suche als Java-Methode
  - liefert Index des gesuchten Elements zurück
  - wenn Objekt nicht gefunden, wird –1 zurückgeliefert

Binäre Suche

- Parameter: Array mit `int`-Werten, gesuchter `int`
- Rückgabewert: Index des gesuchten Werts oder -1

```
1  public static int binsuche(int[] a, int e) {
2      int l=0, r=a.length-1, m;
3      while(l<=r) {
4          m = (l+r)/2;
5          if(a[m]==e)
6              return m;
7          else if (e > a[m])
8              l = m+1;
9          else
10             r = m-1;
11     }
12     return -1;
13 }
```

Binäre Suche

- Variable `l`: Index des linken Randes des zu untersuchenden Bereichs
- Variable `r`: Index des rechten Randes
- Variable `m`: Index der Mitte
- Mitte prüfen
- Solange nicht gefunden: Suchbereich einschränken
  - nur noch links oder rechts der Mitte suchen
- Ende der Schleife:
  - linker und rechter Rand sind gleich
  - dann nochmal prüfen, ob bei `l=r=m` das gesuchte Element ist

Binäre Suche

- Beispiel für die Verwendung der Methode

```
1  public class BinSuche {
2      public static int binsuche(int[] a, int e) {
3          …
4      }
5
6      public static void main(String[] args) {
7          int[] z = {2, 3, 6, 7, 12, 21, 55, 67, 83, 99};
8          int s = 83;
9          int pos = binsuche(z, s);
10         System.out.print("Das Element "+s);
11         System.out.println(" befindet sich an Position "+
12             pos+".");
13     }
14 }
```

Binäre Suche

- Ausgabe des Programms

```
Das Element 83 befindet sich an Position 8.
```

Binäre Suche

- AlgoRythmics:
  BINARY search with FLAMENCO dance
  - https://youtu.be/iP897Z5Nerk
  - Created at Sapientia University, Targu Mures (Marosvásárhely), Romania.
  - Directed by Kátai Zoltán, Osztián Erika, Osztián Pálma-Rozália, Vekov Géza-Károly.
  - In cooperation with András Lóránt Company, Targu Mures (Marosvásárhely), Romania.
  - Choreographer: András Lóránt.
  - Video: Márton

Sortieren eines Arrays

- Aufsteigend Sortieren eine Arrays (z.B. mit `int`-Werten)
  - an Stelle mit Index 0 soll das kleinste Element stehen
  - an Stelle mit maximalem Index das größte Element
- Eines wichtigsten Standardprobleme in der Informatik
- Viele unterschiedliche Sortierverfahren bekannt
  - unterschiedliche Komplexität
  - unterschiedlich aufwendige Implementierung

Bubblesort

- Ein relativ einfach zu erklärendes und zu implementierendes Sortierverfahren
- Laufzeit nicht optimal (es gibt schnellere)
- Funktionsprinzip
  - Array von links (ab Index 0) nach rechts durchlaufen
  - jeweiliges Element mit Index `i` mit rechtem Nachbarn `i+1` vergleichen
  - wenn Element an Stelle `i` größer ist als Element an Stelle `i+1`: Elemente tauschen
  - das Durchlaufen des Arrays so lange wiederholen, bis im letzten Durchlauf kein Vertauschen mehr nötig ist
- Eigenschaften
  - im ersten Durchlauf wandert das größte Element an das rechte Ende des Arrays (zum größten Index)
  - erinnert daran, wie Gasblasen in einer Flüssigkeit aufsteigen

Bubblesort

- Bubblesort als Pseudocode
  Input: Array $a = (a_1, \ldots, a_n)$
  Output: Aufsteigend sortierte Folge a
  1: **repeat**
  2: **for** $i = 1$ **to** $n - 1$ **do**
  3: **if** $a_i > a_{i+1}$ **then**
  4: Vertausche Werte von a_i und a_{i+1}
  5: **until** Keine Vertauschungen mehr aufgetreten

Bubblesort

- Bubblesort als Methode in Java
- Parameter: Array mit `int`-Werten (Call-by-Reference)

```
1  public static void sort(int[] a) {
2      boolean tausch;
3      int h;
4      do {
5          tausch = false;
6          for(int i=0; i<a.length-1; i++) {
7              if(a[i]>a[i+1]) {
8                  tausch = true;
9                  h = a[i];
10                 a[i] = a[i+1];
11                 a[i+1] = h;
12             }
13         }
14     } while(tausch);
15 }
```

Bubblesort

- Boolesche Variable `tausch` gibt an, ob im jeweiligen Durchgang ein Tausch stattgefunden hat
- Tauschen von Werten in einem Array (Zeilen 9 bis 11 im Code) erfordert eine Hilfsvariable h, in der der ursprüngliche Inhalt eines Eintrages des Arrays zwischengespeichert wird

Bubblesort

- Beispiel für die Verwendung der Methode

```
1  public class BubbleSort {
2      public static void sort(int[] a) {
3          …
4      }
5
6      public static void main(String[] args) {
7          int[] a = {12, 4, 2, 3, 7, 1, 9, 5};
8          sort(a);
9          for(int b : a) {
10             System.out.print(" " + b);
11         }
12         System.out.println();
13     }
14 }
```

- Ausgabe des Programms

```
 1 2 3 4 5 7 9 12
```

Bubblesort

- Diverse Optimierungen möglich, z.B.
  - da sich das größte Element nach dem ersten Durchlauf garantiert beim höchsten Index befindet, muss man im folgenden Durchlauf ein Element weniger vergleichen
  - im zweiten Durchlauf wandert das zweitgrößte Element an die vorletzte Stelle, im folgenden Durchlauf ist nochmals ein Vergleich weniger erforderlich
  - usw.

Bubblesort

- eine etwas optimierte Bubblesort-Implementierung

```
1  public static void sortopt1(int[] a) {
2      boolean tausch;
3      int h;
4      int j = 1;
5      do {
6          tausch = false;
7          for(int i=0; i<a.length-j; i++) {
8              if(a[i]>a[i+1]) {
9                  tausch = true;
10                 h = a[i];
11                 a[i] = a[i+1];
12                 a[i+1] = h;
13             }
14         }
15         j++;
16     } while(tausch);
17 }
```

Bubblesort

- AlgoRythmics:
  Bubble-sort with Hungarian ("Csángó") folk dance
  - https://youtu.be/lyZQPjUT5B4
  - am 29.03.2011 veröffentlicht
  - Created at Sapientia University, Tirgu Mures (Marosvásárhely), Romania.
  - Directed by Kátai Zoltán and Tóth László.
  - In cooperation with "Maros Művészegyüttes", Tirgu Mures (Marosvásárhely), Romania.
  - Choreographer: Füzesi Albert.
  - Video: Lőrinc Lajos, Körmöcki Zoltán.
  - Supported by "Szülőföld Alap", MITIS (NGO) and evoline company.