Fachhochschule
Dortmund
University of Applied Sciences and Arts
Fachbereich Informatik

Algorithmen und Datenstrukturen

VL01 – ALGORITHMEN

Überblick

- Einführung
- Datenstrukturen
  - Abstrakte Datenstrukturen
  - Konkrete Datenstrukturen
- Algorithmen
  - Eine Begriffsbestimmung
  - Algorithmus vs. Programm
- Pseudocode
- Algorithmenentwurf
- Korrektheit

EINFÜHRUNG

Einführung

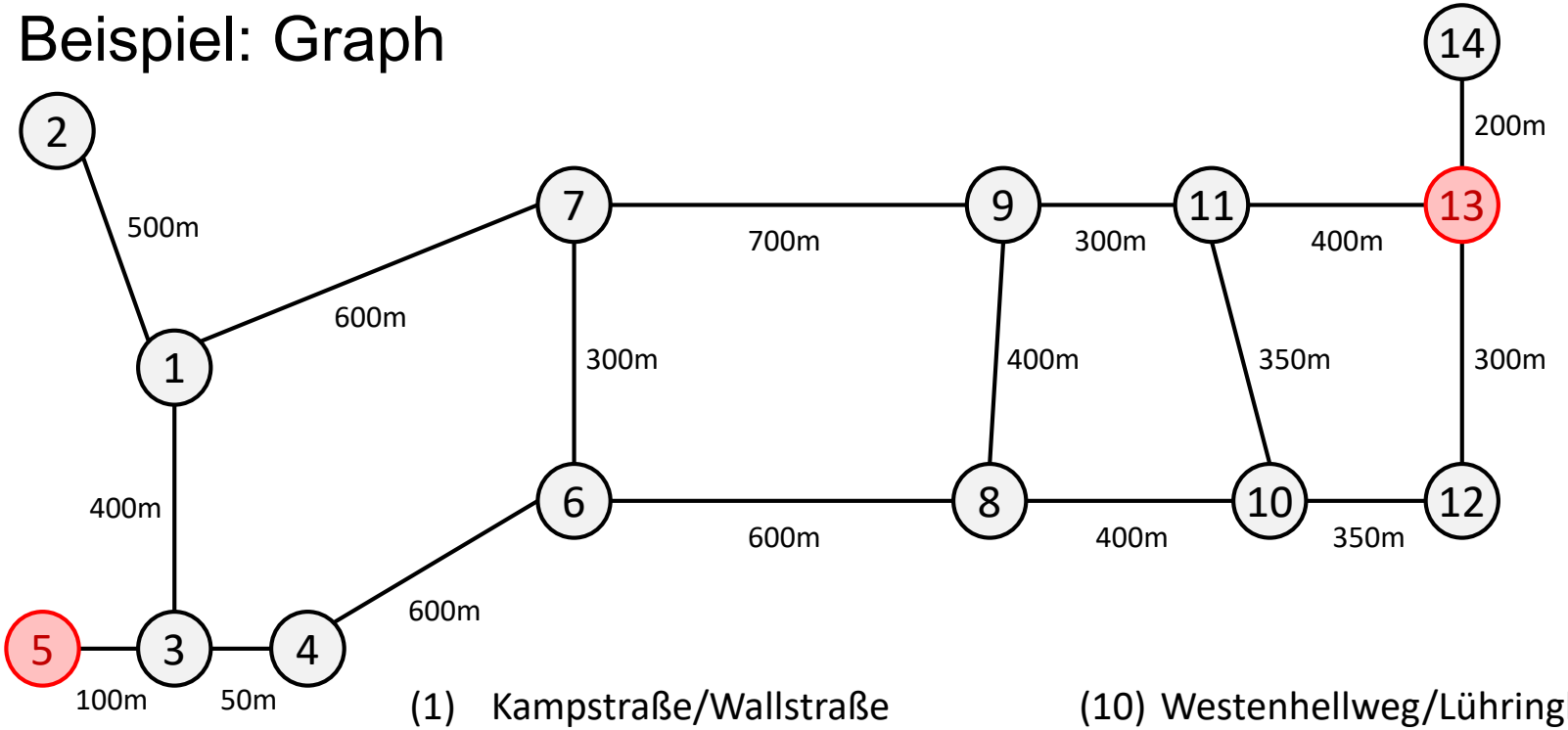
Datenstrukturen und Algorithmen

- Beispiel: Navigationssystem

Einführung

Datenstrukturen und Algorithmen

- Beispiel: Graph



(1) Kampstraße/Wallstraße
(2) Wallstraße/Schmiedingstraße
(3) Weddepoth/Westenhellweg
(4) Westenhellweg/Martinsstraße
(5) Westenhellweg/Grafenhof
(6) Westenhellweg/Petrikirchhof
(7) Kampstraße/Petrikirchhof
(8) Westenhellweg/Petergasse
(9) Kampstraße/Petergasse
(10) Westenhellweg/Lühringhof
(11) Kampstraße/Lühringhof
(12) Westenhellweg/Platz von Netanya
(13) Kampstraße/Platz von Netanya
(14) Hansastraße/Bissenkamp

Einführung

Datenstrukturen und Algorithmen

- Algorithmus `kürzesteWege`:

Eingabe: Graph `G`, Knoten `x`, Knoten `y`

`Entfernung` ← `{ (x,0,x) }`

`ErreichteKnoten` ← `{ x }`

while `y` nicht in `ErreichteKnoten` do

Suche einen Nachbarknoten `z` zu einem Knoten `t` aus `ErreichteKnoten`, für den gilt:
Abstand `d1` von `x` zu `t` + Abstand `d2` von `t` zu `z` ist minimal

Füge `(t,d1+d2,z)` zu `Entfernung` hinzu

Füge `z` zu `ErreichteKnoten` hinzu

end while

`KürzesterWeg` ←`(y);`

`z` ← `y;`

while `z!=x` do

Suche `t` mit `(t,d,z)` in `Entfernung` und füge `t` am Anfang von `KürzesterWeg` ein, `z` ← `t;`

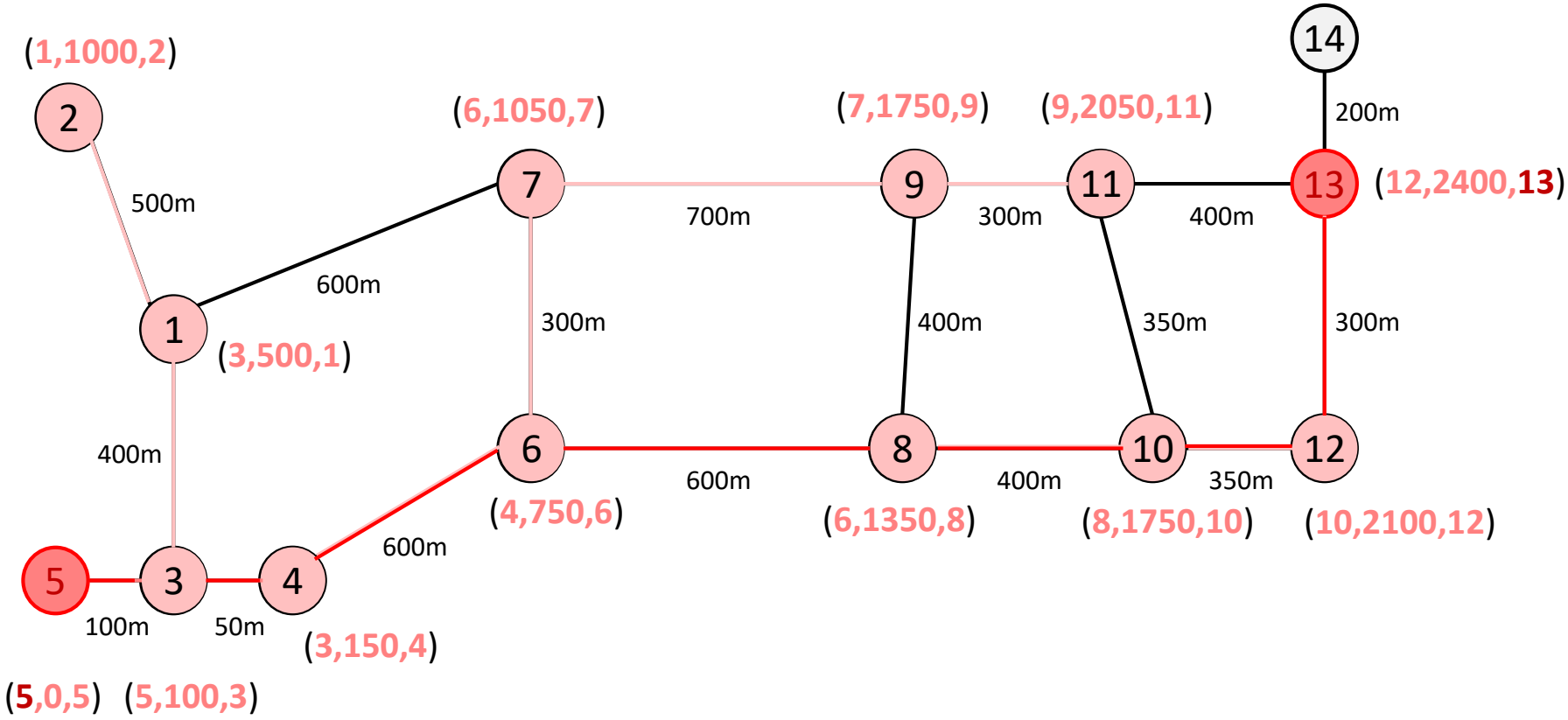
end while

Ausgabe: `KürzesterWeg`

Einführung

Datenstrukturen und Algorithmen

- Algorithmus `kürzesteWege`: Beispiel



Die Grafik auf dieser Seite zeigt die Druckversion ohne Animationen.

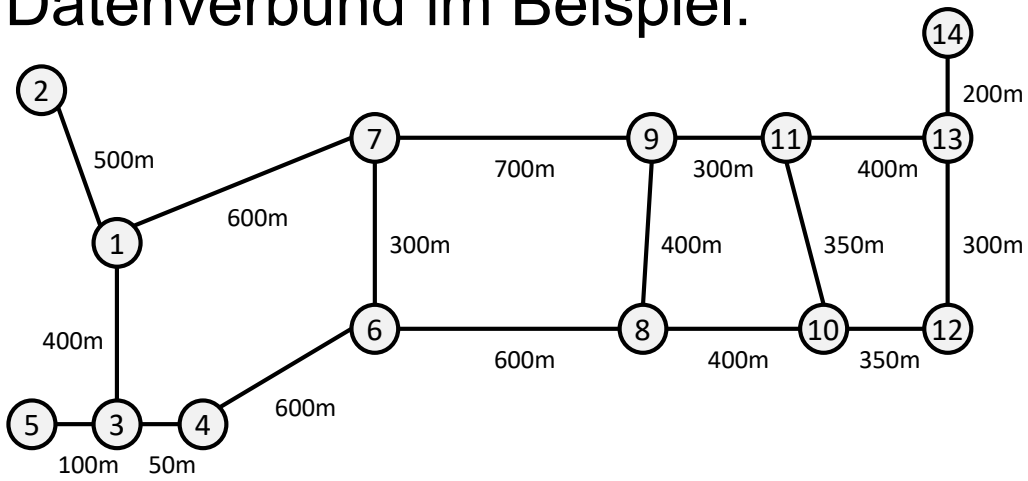
Datenstrukturen

Datenstrukturen

Abstrakte Datenstrukturen

- Eine **abstrakte Datenstruktur** beschreibt die wesentlichen Eigenschaften des Datenverbundes und der zulässigen Operationen auf der Datenstruktur.

- Datenverbund im Beispiel:



Operationen:

- `getNodeCount()`
- `getEdgeCount()`
- `getNeighbor(n)`
- `getWeight(e)`
- …

Datenstrukturen

Konkrete Datenstrukturen

- Eine **konkrete Datenstruktur** ist die Art und Weise, wie Informationen konkret im Speicherbereich abgelegt werden.

- Für unser Beispiel: Graph als Adjazenzmatrix (VL12)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	500	400				600								
2	500	0													
3	400		0	50	100										
4			50	0		600									500
5			100		0										
6				600		0									
7	600						0	300	600						
8							300	0	400	400					
9							600	400	0		300				
10								400		0	350	350			
11									300	350	0		400		
12										350		0	300		
13											400	300	0	200	
14													200	0	
15				500											0

Algorithmen

Algorithmen

Begriffsbestimmung

- Ein **Algorithmus** ist eine eindeutige, endliche Beschreibung eines allgemeinen, endlichen Verfahrens zur schrittweisen Ermittlung gesuchter Größen aus gegebenen Größen.

- Beispiel (Algorithmus für kürzesten Weg):

Eingabe: Graph `G`, Knoten `x`, Knoten `y`

`Entfernung` ← `{ (x,0,x) }`

`ErreichteKnoten` ← `{ x }`

while `y` nicht in `ErreichteKnoten` do

Suche einen Nachbarknoten `z` zu einem Knoten `t` aus `ErreichteKnoten`, für den gilt:

Abstand `d1` von `x` zu `t` + Abstand `d2` von `t` zu `z` ist minimal

Füge `(t,d1+d2,z)` zu `Entfernung` hinzu

Füge `z` zu `ErreichteKnoten` hinzu

end while

`KürzesterWeg` ←`(y);`

`z` ← `y;`

while `z!=x` do

Suche `t` mit `(t,d,z)` in `Entfernung` und füge `t` am Anfang von `KürzesterWeg` ein, `z` ← `t;`

end while

Ausgabe: `KürzesterWeg`

Algorithmen

Begriffsbestimmung

- Ein **Algorithmus** ist eine eindeutige, endliche Beschreibung eines allgemeinen, endlichen Verfahrens zur schrittweisen Ermittlung gesuchter Größen aus gegebenen Größen.

- Eindeutigkeit:
  - Aus der Formulierung des Algorithmus muss die Abfolge der einzelnen Verarbeitungsschritte eindeutig hervorgehen.
  - Die Beschreibung erfolgt in einem Formalismus mit Hilfe von anderen Algorithmen und elementaren Algorithmen.

- Aber:
  - Wahlmöglichkeiten, wie Auswahl und Wiederholung, sind zugelassen. Es muss aber genau festgelegt sein, wie die Auswahl einer der Möglichkeiten erfolgen soll.

Algorithmen

Begriffsbestimmung

- Ein **Algorithmus** ist eine eindeutige, endliche Beschreibung eines allgemeinen, endlichen Verfahrens zur schrittweisen Ermittlung gesuchter Größen aus gegebenen Größen.

- Endliche Beschreibung (**statische Finitheit**):
  - Ein Algorithmus muss durch endlichen Quelltext beschreibbar sein.

Algorithmen

Begriffsbestimmung

- Ein **Algorithmus** ist eine eindeutige, endliche Beschreibung eines allgemeinen, endlichen Verfahrens zur schrittweisen Ermittlung gesuchter Größen aus gegebenen Größen.

- Allgemeinheit:
  - Ein Algorithmus löst nicht nur ein spezielles Problem, sondern ein allgemeines Problem bzw. eine Problemklasse.
  - Der Algorithmus wird dazu durch Parameter (Eingaben) gesteuert, die ein einzelnes, aktuell zu lösendes Problem aus dieser Klasse wählen.
  - Das Ziel ist die Wiederverwendbarkeit.
  - Gegenbeispiele:
    - Berechnen von $\log_2 5$ anstatt $\log_2 x$ oder $\log_b x$
    - Navigation nur in einem konkreten Graph statt in einem beliebigen Graph

Algorithmen

Begriffsbestimmung

- Ein **Algorithmus** ist eine eindeutige, endliche Beschreibung eines allgemeinen, endlichen Verfahrens zur schrittweisen Ermittlung gesuchter Größen aus gegebenen Größen.

- Endlichkeit (**dynamische Finitheit**):
  - Ein Algorithmus muss für jede zulässige Eingabe nach endlich vielen Schritten ein Ergebnis liefern und anhalten, d.h. terminieren.

Algorithmen

Begriffsbestimmung

- Ein **Algorithmus** ist eine eindeutige, endliche Beschreibung eines allgemeinen, endlichen Verfahrens zur schrittweisen Ermittlung gesuchter Größen aus gegebenen Größen.

- Ausführbarkeit:
  - Die Anweisungen eines Algorithmus müssen ausführbar sein, d.h. eine Person, die den Formalismus kennt und die elementaren Algorithmen beherrscht, muss ihn in ein äquivalentes, ausführbares Programm überführen können.

- Gegenbeispiel:
  - Eine Vorschrift wie „Wenn man in 14 Tagen Zahnschmerzen bekommt, dann vereinbart man bereits heute einen Termin mit dem Zahnarzt, um die Wartezeit zu verkürzen.“ ist in der Realität nicht ausführbar.

Algorithmen

Algorithmus vs. Programm

- Algorithmus:
  1. `einlesen b`
  2. `einlesen x`
  3. `erg ← 4 * Logarithmus(b,x)`
  4. `ausgeben erg`

- Programm:

```
public class VierfacherLogarithmus
{
  public static void main(String[] args)
  {
    Scanner sc = new Scanner(System.in);

    int b = sc.nextInt();
    int x = sc.nextInt();
    int erg = 4 * berechneLogarithmus(b, x);
    System.out.println(erg);
  }
}
```

Algorithmen

Algorithmus vs. Programm

- Im Programm wird der Algorithmus durch die elementaren Bausteine der Programmiersprache beschrieben.
  - Die Programmiersprache gibt damit vor, welche elementaren Algorithmen verwendet werden können.

- Verschiedene Programmiersprachen haben:
  - Verschiedene elementare Bausteine
  - Somit unterschiedliche Programme

- Dennoch: derselbe Algorithmus!

- Folgerung: Algorithmen sollten unabhängig von einer Programmiersprache beschrieben werden.

Algorithmen

Algorithmus vs. Programm

- Formalisierungsmöglichkeiten für Algorithmen:
  - Freitext: „Zuerst wird die Variable n auf 0 gesetzt. Dann…“
  - Struktogramme
  - Flussdiagramme
  - Pseudocode

Pseudocode

Pseudocode

Motivation

- Pseudocode ist eine Möglichkeit Algorithmen zu formulieren:
  - Unabhängig von einer konkreten Programmiersprache
  - Vermischt natürlichsprachliche Beschreibung und Programmcode:
    - Ähnlich genug zu natürlicher Sprache, um für Menschen leichter verständlich zu sein
    - Ähnlich genug zu gängigen Programmiersprachen, um leicht implementiert zu werden

- Es existieren verschiedene konkrete Varianten, die diese Voraussetzungen erfüllen und daher Pseudocode sind.
  - Wir benutzen in dieser Veranstaltung einen weit verbreiteten Standard für Pseudocode, der sich an einer früher weit verbreiteten Programmiersprache „Pascal“ orientiert.

Pseudocode

Operatoren und if-Anweisung

- Im Pseudocode werden die bekannten Java-Operatoren benutzt, z.B. `+`, `==`, `<<` und viele mehr. Ausnahmen:
  - Zuweisung `=` wird durch einen Pfeil ← dargestellt
  - Boolesche Operatoren werden (wie in Pascal) durch `and` , `or` und `not` ausgedrückt

- `if`-Anweisungen benutzen ebenfalls in Pascal-Notation:

```
if <Bedingung> then
  <Anweisungen>
else
  <Anweisungen>
endif
```

Pseudocode

Schleifen

- Schleifen benutzen ebenfalls die in Pascal übliche Syntax:

```
while <Bedingung> do
  <Anweisungen>
end while

repeat
  <Anweisungen>
until <Bedingung>

for <Zähler> ← <Start> to <Ende> do
  <Anweisungen>
end for
```

  - Varianten für `for`-Schleifen:

```
for <Zähler> ← <Start> to <Ende> step <Schrittweite> do
for <Zähler> ← <Start> downto <Ende> do
for <Zähler> ← <Start> downto <Ende> step <Schrittweite> do
```

Pseudocode

Arrays, Ein- und Ausgabe

- Arrays beginnen **im Gegensatz zu Java** grundsätzlich mit dem Index 1:

```
A[1] ← 5          // Erstes Element bekommt 5 zugewiesen
A[0] ← 5          // Fehlerhafte Anweisung
```

- Bei der Ein- und Ausgabe wird vom technischen Vorgang abstrahiert:

```
einlesen a
a ← a+1
ausgeben a
```

- Weitere Informationen und Beispiele finden Sie in der Datei `Pseudocode.pdf` im ILIAS-System.
  - Bitte zur Vorbereitung (auf die Übungen) lesen!

Pseudocode

Beispiel

- Hauptalgorithmus:

```
einlesen b                  // Abstraktion vom techn. Vorgang
einlesen x
erg ← 4 * Logarithmus(b,x)
ausgeben erg                // Abstraktion vom techn. Vorgang
```

- Funktion:

```
Logarithmus(b,x)
  n ← 0                     // Zuweisung
  y ← x                     // Zuweisung
  while y >= b do           // Schleife
    y ← y/b
    n ← n+1
  end while
  return n
```

ALGORITHMENENTWURF

Algorithmenentwurf

Vorgehensweise

- Der Entwurf eines guten Algorithmus ist ein schrittweiser Prozess:
  1. Erfassen und analysieren der Problemstellung
  2. Erarbeitung einer Lösungsidee
     (ggf. verbale Spezifikation oder Pseudocode)
  3. Wahl oder Entwicklung einer geeigneten Datenstruktur
  4. Implementierung und schrittweise Verfeinerung
     (ggf. unter Wiederholung voriger Schritte)
  5. Aufwandsanalyse (ggf. unter Wiederholung voriger Schritte)

  – Ggf. müssen dazu vorhergehende Schritte anders neu durchgeführt werden, da man in späteren Schritten (z.B. Implementierung oder Aufwandsanalyse) neue Erkenntnisse und Optimierungsideen gewonnen hat.

Algorithmenentwurf

Entwurfsprinzipien

- Problemlösung durch Rekursion (VL05)
- Einsatz von Algorithmenmustern:
  - **Greedy-Algorithmen** („Gierige“ Algorithmen)
    - Prinzip: erreiche in jedem Teilschritt so viel wie möglich
  - **Divide-and-Conquer** („Teile und Herrsche“, z.B. Sortierverfahren wie Mergesort, Quicksort)
    - Prinzip: Teile eine große Aufgabe in mehrere kleine Aufgaben auf, bearbeite diese rekursiv und setze die Gesamtlösung aus den Lösungen der Teilaufgaben zusammen
  - **Backtracking** („Den Weg zurückgehen“)
    - Realisiert eine allgemeine systematische Suchtechnik, die einen vorgegebenen Lösungsraum komplett bearbeitet
  - Dynamische Programmierung
    - Prinzip: Berechnen der optimalen Lösungen der kleinsten Teilprobleme, dann zusammensetzen zu einer Lösung eines nächstgrößeren Teilproblems

Korrektheit von Algorithmen

Algorithmen

Korrektheit

- Algorithmen müssen korrekt sein, d.h. sich genau so verhalten wie es beabsichtigt war.

Absturz einer Ariane-Rakete am 04.06.1996 aufgrund eines Software-Fehlers
Kosten: etwa 500 Millionen Dollar!

Algorithmen

Korrektheit

- Vor- und Nachbedingungen (Spezifikation):
  - Unter welchen Bedingungen bzw. Voraussetzungen arbeitet der Algorithmus?
  - Wie sehen die möglichen Ausgaben bei zulässigen Eingaben aus?

- Beispiel:

```
// Ganzzahliger Anteil von log x zur Basis b
Logarithmus(b,x)
  n ← 0
  y ← x                    // Zuweisung
  while y >= b do
    y ← y/b
    n ← n+1
  end while
  return n
```

Vorbedingungen: $b>1$, $x\geq 1$

Nachbedingung: $b^n \leq x < b^{n+1}$ bzw. $\log_b(b^n) = n \leq \log_b(x) < n+1 = \log_b(b^{n+1})$

Algorithmen

Korrektheit

- Hilfsmittel:
  - Im Programm: `assert`-Anweisungen einbauen

```
public static int berechneLogarithmus(int b, int x)
{
  assert(b>1);                  // Nur Zahl > 1 erlaubt
  assert(x>=1);                 // Nur Zahl >= 1 erlaubt

  int n = 0;
  int y = x;
  while (y >= b)
  {
     y /= b;
     n++;
  }
  return n;
}
```

  - Ausführung: Programmstart mit Parameter `-enableassertions`

```
java -enableassertions beispiel.class
```

Algorithmen

Korrektheit

- Techniken:
  - Formale Verifikation
  - Testen

Algorithmen

Formale Verifikation

- **Verfikation** ist eine formal exakte Methode, um die Konsistenz zwischen der Spezifikation und der Implementierung für alle in Frage kommenden Eingabedaten zu **beweisen**.
- Beispiel:
  Aus der Vorbedingung $b>1$ und $x\geq 1$ für die Funktion Logarithmus(b,x) lässt sich durch schrittweise Anwendung logischer Regeln die Nachbedingung $b^n \leq x < b^{n+1}$ und damit $n \leq \log_b(x) < n+1$ herleiten.

Auf formale Verifikation wird in dieser Vorlesung nicht näher eingegangen.

Algorithmen

Testen

- Für möglichst gut ausgewählte Testdaten wird das Programm ausgeführt und beobachtet, ob das gewünschte Ergebnis ermittelt wird.
  - Nicht alle möglichen Kombinationen von Eingabedaten können getestet werden, z.B. von `b` und `x` in `Logarithmus(b,x)`
  - Daher keine Gewissheit über die Korrektheit des Programms für die noch nicht getesteten Eingabedaten
  - **Durch Tests kann man nur die Anwesenheit, aber nicht die Abwesenheit von Fehlern zeigen!**
  - **Testen Sie auf jeden Fall Randbereiche und Sonderfälle!**

Lehrziele

- Sie können die Begriffe abstrakte und konkrete Datenstruktur sowie Algorithmus definieren und an Beispielen erläutern
- Sie kennen den Unterschied zwischen Algorithmus und Programm
- Sie kennen die wesentlichen Schritte bei der Entwicklung von Algorithmen
- Sie kennen die wesentlichen Methoden zur Überprüfung der Korrektheit von Algorithmen
- Sie können Pseudocode lesen, verstehen und schrittweise in Java-Quellcode übertragen

Literatur

- Quellen:
  - Balzert, H.: Lehrbuch Grundlagen der Informatik
    (LE 13: Algorithmen und ihre Verifikation)
  - T. Cormen et al.: Algorithmen – Eine Einführung
  - A. Solymosi et al.: Grundkurs Algorithmen und Datenstrukturen in JAVA