Informatik I WS 07/08

Tutorium 24

06.12.07

Bastian Molkenthin

E-Mail: infotut@sunshine2k.de

Web: http://infotut.sunshine2k.de

Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Vollständige Induktion

- Vollständige Induktion ist ein Beweisverfahren um die Gültigkeit einer Aussage für eine unbeschränkte Zahlenmenge zu zeigen

- Vorgehen:

  1. Man zeigt dass die Aussage für einen Basisfall gilt (*Induktionsanfang, IA*)
  2. Das Gezeigte formuliert man in der *Induktionsvoraussetzung (IV)* für **einen** festen Wert (n) aus der Zahlenmenge
  3. Man versucht unter Verwendung der IV die Behauptung für den nächsten Wert (n+1) zu folgern (*Induktionsschritt, IS*)

- Vollständige Induktion kommt immer wieder…

Mathematisches Beispiel

Behauptung: $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$

Beweis mittels vollständiger Induktion:

1. IA $\sum_{k=1}^{1} k = 1 \overset{!}{=} \frac{1(1+1)}{2} = \frac{2}{2} = 1$

2. IV Behauptung gelte für ein $n \in \mathbb{N}$

Mathematisches Beispiel (2)

3. IS (n ⇨ n+ 1)

$$\sum_{k=1}^{n+1} k = \sum_{k=1}^{n} k + (n+1)$$

$$\overset{IV}{=} \frac{n(n+1)}{2} + (n+1)$$

$$= \frac{n(n+1)}{2} + \frac{2(n+1)}{2}$$

$$= \frac{n^2+n+2n+2}{2} = \frac{(n+1)(n+2)}{2}$$

Schleifeninvariante

- Eine Schleifeninvariante ist eine Bedingung, die in jedem Schleifendurchlauf gültig ist

- Hilft bei der Verifizierung von Programmen
  ("Tut das Programm wirklich das, was es soll?")

- Um eine gegebene Schleifeninvariante zu beweisen muss man zeigen,
  dass die Bedingung an einer bestimmten Stelle in **jedem** Schleifendurchlauf gilt

- Anwendung der vollständigen Induktion
  (➡ hier Induktion über die Anzahl der Schleifendurchläufe)

Schleifeninvariante (Beispiel)

```
int a = In.readInt();
int b = In.readInt();
if (a > b) {    // if a > b exchange a and b
  int h = a;
  a = b;
  b = h;
}
// Assertion a <= b
```

```
int z = b;
int i = 1;

while ((z % a) != 0){
  // position 1
  z += b;
  // position 2
  i++;
  // position 3
}
```

Beweisen oder widerlegen Sie die Korrektheit folgender Aussagen:

1. z == b * i ist eine Schleifeninvariante für die while-Schleife

2. kgV(a,b) == kgV(a,z) ist eine Schleifeninvariante für die while-Schleife

3. Die while-Schleife terminiert.

Hinweis: Nutze geeignete Zusicherungen an den Stellen position 1, position 2 und position 3, um die Beweise zu führen.

Schleifeninvariante (Beispiel) - 2

Lösung zu 1)

IA: Im ersten Schleifendurchlauf gilt z = b * i
denn i = 1 und z = b (siehe Deklaration)

IV: $Z_n = b * i_n$ gelte im n-ten Schleifendurchlauf
an Position 1.

IS: Zu zeigen:
Im (n+1)-ten Durchlauf gilt an Position 1 immer
noch $Z_{n+1} = b * i_{n+1}$.

In diesem Fall muss im n-ten Durchlauf an Position 3
$Z_{n+1} = b * i_{n+1}$ gelten.

```
int a = In.readInt();
int b = In.readInt();
if (a > b) {
  int h = a;
  a = b;
  b = h;
}
// Assertion a <= b

int z = b;
int i = 1;

while ((z % a) != 0){
  // position 1
  z += b;
  // position 2
  i++;
  // position 3
}
```

Schleifeninvariante (Beispiel) - 3

Es gelten folgende Zusicherungen:

Position 1: $z_n == b * i_n$

Position 2: $z_{n+1} == z_n + b$

Position 3: $i_{n+1} == i_n + 1$

Somit gilt am Ende der Schleife (Position 3):

$z_{n+1} == z_n + b == b * i_n + b == b * (i_n + 1) == b * i_{n+1}$

Durch vollständige Induktion ist damit bewiesen, dass
z = b * i am Anfang und am Ende jedes Schleifendurchlaufs gilt.

Da der Ausdruck z = b * i am Anfang und am Ende jedes Schleifendurchlaufs gilt, ist nachgewiesen, dass es sich bei diesem Ausdruck um eine Schleifeninvariante handelt.

```
int a = In.readInt();
int b = In.readInt();
if (a > b) {
  int h = a;
  a = b;
  b = h;
}
// Assertion a <= b

int z = b;
int i = 1;

while ((z % a) != 0){
  // position 1
  z += b;
  // position 2
  i++;
  // position 3
}
```

Schleifeninvariante (Beispiel) - 4

Terminiert die Schleife?

Lösung:

- zu Beginn gilt z = b

** - z wächst mit jedem Schleifendurchlauf streng monoton um b

- a * b ist ein gemeinsames Vielfaches von a und b

- Spätestens wenn z = a * b erreicht ist (wird auf jeden Fall erreicht wegen **), bricht Schleife ab.

⇒ Schleife terminiert!

```
int a = In.readInt();
int b = In.readInt();
if (a > b) {
  int h = a;
  a = b;
  b = h;
}
// Assertion a <= b

int z = b;
int i = 1;

while ((z % a) != 0){
  // position 1
  z += b;
  // position 2
  i++;
  // position 3
}
```

Schleifeninvariante (Beispiel) - 5

kgV(a,b) == kgV(a,z) Schleifeninvariante?

Lösung: **NEIN!** Beweis duch Gegenbeispiel.

Sei z.B a = 3 und b = 5.

Im ersten Durchlauf gilt:

Position 1: z = 5, b = 5

kgV (a, b) = kgV(3, 5) = 15 = kgV(a, z)

Position 3: z = 10, b = 5

kgV (a, b) = kgV(3, 5) = 15
kgV (a, z) = kgV(3, 10) = 30

Im zweiten Durchlauf gilt an Position 1 das gleiche wie im ersten Durchlauf an Position 3. Somit gilt **nicht**
kgV (a, b) = kgV(a, z) in jedem Schleifendurchlauf an Position 1!
1 kgV (a, b) = kgV (a, z) ist keine Schleifeninvariante!

```
int a = In.readInt();
int b = In.readInt();
if (a > b) {
  int h = a;
  a = b;
  b = h;
}
// Assertion a <= b

int z = b;
int i = 1;

while ((z % a) != 0){
  // position 1
  z += b;
  // position 2
  i++;
  // position 3
}
```

Halbgruppe

Eine Halbgruppe muss folgende Bedingungen erfüllen:

- *Abgeschlossenheit:*

  Die Menge M muss unter der gegebenen Verknüpfung abgeschlossen sein.
  Das heißt die Verknüpfung ◦ zweier Werte aus M muss wieder in M enthalten sein:

$$a \in M, b \in M \qquad (a \circ b) \in M$$

- *Assoziativität:*

  Die Verknüpfung ◦ muss das Assoziativgesetz erfüllen:

$$(a \circ b) \circ c = a \circ (b \circ c) \qquad a, b, c \in M$$

Monoid

Eine Monoid ist eine spezielle Halbgruppe:

Es muss zusätzlich zu den Kriterien der Halbgruppe noch die Bedingung für ein Einselement bzw. neutrales Element erfüllt sein.

- *Einselement:*

  Das Einselement e aus M bewirkt weder für links- noch rechtsseitige Verknüpfung mit einem beliebigen Wert aus M eine Änderung.

$$x \circ e = e \circ x = x \qquad x \in M$$

Halbgruppen/Monoide Aufgaben

Die Addition in der Menge der natürlichen Zahlen mit der Null $\mathbb{N}_0$.

Lösung:

Es handelt es sich um ein Monoid. Beweis:

- *Abgeschlossenheit:* $a, b \in N \Rightarrow (a+b) \in \mathbb{N}_0$
- *Assoziativität:* $a, b, c \in N \quad (a+b) \in \mathbb{N}_0$

  $(a + b) + c = a + b + c = a + (b + c)$
- *Einselement:* $e = 0 : x + 0 = 0 + x = x$

Halbgruppen/Monoide Aufgaben

Die Subtraktion in der Menge der natürlichen Zahlen mit der Null $\mathbb{N}_0$.

Lösung:

Weder Halbgruppe noch Monoid, da nicht abgeschlossen:

z.B. a = 3, b = 5

$(a - b) = 3 - 5 = -2 \notin \mathbb{N}_0$

Halbgruppen/Monoide Aufgaben

Die Verknüpfung ^ (Potenz) in der Menge $\mathbb{N}_0$.

Lösung:

Weder Halbgruppe noch Monoid , da nicht assoziativ:

z.B. a = 2, b = 2, c = 3

(a^b)^c = (2^2)^3 = 4^3 = 64

a^(b^c) = 2^(2^3) = 2^8 = 256

Halbgruppen/Monoide Aufgaben

Die Linksidentität in der Menge $\mathbb{R}$.

(Linksidentität: für die Verknüpfung $\circ$ gilt: $a \circ b = a \qquad \forall a, b \in \mathbb{R}$)

Lösung:

Halbgruppe, aber kein Monoid:

- *Abgeschlossenheit:* $\forall a, b \in \mathbb{R}$ gilt: $a \circ b = a \in \mathbb{R}$
- *Assoziativität:* $a, b, c \in \mathbb{R}$

  $a \circ (b \circ c) = a \circ b = a$

  $(a \circ b) \circ c = a \circ c = a$
- *Einselement:* $e \circ x = e \neq x \circ e = x$

Java Strings

String: Datentyp für Zeichenketten

- Initialisierung: `String s = "test";`
- Verkettung: `s + "en"` → `"testen";`
- Zeigervergleich: `s == "test"` → False
- Wertevergleich: `s.eqauls("test")` → True

Java String Operationen

s.length()	Gibt Anzahl der Zeichen von s zurück
s.charAt(i)	Gibt Zeichen an Position i zurück
s.indexOf(“abc“,i)	liefert die Position an der “abc“ erstmals in s beginnend bei Position i vorkommt
s.lastIndexOf(“abc“)	liefert die Position an der “abc“ zuletztin s vorkommt
s.substring(i,j)	liefert den Teilstring von Position i bis ausschlielich Position j
s.startsWith(“abc“)	liefert true wenn s mit “abc“ beginnt (sonst false)
s.endsWith(“abc“)	liefert true wenn s mit “abc“ endet

String-Konversion

- String ⟹ int: `int i = Integer.parseInt("123");`
- String ⟹ float: `float f = Float.parseFloat("3.14");`
- char, int, long, float, double, boolean ⟹ String:

```
String s = String.valueOf(x);
```

Java - Stringmanipulation

- Es gibt viele verschiedene Wege mit Strings zu arbeiten

- Viele Dinge lassen sich direkt mit den Methoden des String-Objekts bewerkstelligen (String.length(), String.charAt(int), String.equals(String), String.startsWith(String), …)

   Ineffizient!

- Für komplexere Anwendungen empfiehlt sich die Umwandlung in andere Datenstrukturen. Hier sind die Folgenden besonders wichtig:
  - char-Array
  - StringBuffer

- Beim char-Array handelt es sich um ein normales Array, das jedes Zeichen im String enthält. Für einen String der Länge 10 wird also ein Array der Größe 10 mit den Indizes 0 bis 9 erstellt.
  - Umwandlung erfolgt mit der Methode String.toCharArray()

Stringbuffer

- Initialisierung: `StringBuffer b = new StringBuffer(s)  // s ist String`
- Umwandlung: `String a = b.toString();`
- nützliche Operationen:

b.length()	Gibt Anzahl der enthaltenen Zeichen zurück
b.charAt(i)	Gibt Zeichen an Position i zurück
b.append(x)	hängt x an b , wobei x vom Typ char, int, float, double, boolean, String oder char[] sein kann
b.insert(i,x)	Fügt x an Position i ein
b.delete(i,j)	Löscht die Zeichen von Position i bis ausschließlich Position j

StringBuffer (2)

b.setChar(i, 'a')	ersetzt das Zeichen an Position i durch 'a'
b.replace(i, j, “abc“)	ersetzt die Zeichen von Position i bis ausschlielich Position j durch “abc“
b.substring(i, j)	liefert den Teilstring von Position i bis ausschließlich Position j

- Beispiel:

  Beispielsweise lässt sich im String "Java ist doof!" das letzte Wort folgendermaßen ersetzen:

```
StringBuffer sb = new StringBuffer("Java ist doof!");
sb.replace(9, 13, "klasse");
```

BubbleSort

```
Setze i = 0
Solange (i < n) {
    Setze j = 0
    Solange (j < n – i – 1) {
        Wenn (element[j] > element[j+1]) {
            tausche(element[j], element[j+1])
        }
        Erhöhe j um 1
    }
    Erhöhe i um 1
}
```

Sortierung von {4, 2, 3,1 }

i = 0 j = 0 : 2 4 3 1
i = 0 j = 1 : 2 3 4 1
i = 0 j = 2 : 2 3 1 4
i = 1 j = 0 : 2 3 1 4
i = 1 j = 1 : 2 1 3 4
i = 2 j = 0 : 1 2 3 4

Aufwand BubbleSort: **$O(n^2)$**

Achtung - Genauigkeit float

```
int max = Integer.MAX_VALUE ;
int almostMax = 2147483000;

Out.println(max);
Out.println(( float ) max);

int diff = max - almostMax ;
Out.println( diff );

float fdif = ( float )max - almostMax ;
Out.println ( fdif );
```

Ausgabe:

```
2147483647
2.14748365E9
647
640.0
```

Kleine Java Aufgabe

Erstellen Sie ein Java-Programm, welches die Kubikwurzel der float-Zahl x berechnet. Die Berechnung soll mittels der Näherungsformel von Newton erfolgen:

$$y_i = \frac{2 * y_{i-1} + \frac{x}{y_{i-1}^2}}{3}$$

y_i ist der Näherungswert der Kubikwurzel nach dem i-ten Schritt. Beginnen Sie mit $y_0=x/3$. Die Iteration soll so lange fortgesetzt werden, bis sich y_i und y_{i-1} um weniger als 1.0E-5 unterscheiden. Prüfen Sie diese Bedingung mittels
Math.abs(y_i - y_{i-1}) < 1.0E-5.

Lösung

```
class Root {
 static float calcRoot3( float x) {                    // this function calculates the cubic
                                                          root of a given float value
  float y1 = x / 3;                                    // start with y1 == x / 3
  float y0 = 0;                                        // and with y0 == 0
  while (Math.abs(y0 - y1) > 1.0E-5) {                 // loop until |yo - y1| < 1.0E-5
   y0 = y1;                                            // set y0 the old y1
   y1 = (2 * y0 + x / (y0 * y0)) / 3;                  // calculate a new y1
  }
  return y1;                                           // return result
 }

 public static void main(String[] arg) {
  Out.print("Geben Sie eine Zahl ein: ");              // Print text to the console
  float x = In.readFloat();                            // read the value x from the console
  Out.println("Kubikwurzel von " + x + " ist "         // print the cubic root of the value
              + calcRoot3(x));                            using the function root3
 }
}
```

Fragen ???

Viel Spaß mit dem Übungsblatt!