Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Peer Kröger
Michael Fromm, Florian Richter

Einführung in die Programmierung

WS 2018/19

Übungsblatt 4: Eigenschaften von Algorithmen, Ganze Zahlen, While

Besprechung: 19.11 - 23.11.2018

Aufgabe 4-1 *Eigenschaften von Algorithmen*

Stellen Sie sich einen Karteikasten vor, in dem Sie nach einer Karteikarte mit einem bestimmten Titel suchen. Gehen Sie dabei von folgenden Annahmen aus:

- Der Karteikasten enthält nicht mehrere Karteikarten mit dem selben Titel.
- Initial enthalte der Karteikasten $n \in \mathbb{N}$ Karteikarten. Also mindestens eine Karteikarte.
- Eine gültige Eingabe ist auch ein Titel, der nicht im Karteikasten vorkommt. In diesem Fall soll nichts als Ergebnis zurück gegeben werden.
- Die Eingabe beinhaltet den gesuchten Titel.
- Die Ausgabe sei die Karte, die sich bei Terminierung in Ihrer Hand befindet.

Suchverfahren 1:
(1) Sie nehmen die erste Karte aus dem Karteikasten.

Suchverfahren 2:
(1) Sie nehmen die erste Karte aus dem Karteikasten heraus.
(2) Hat diese den gewünschten Titel, so sind Sie fertig ansonsten legen Sie die Karte auf die Seite
(3) Ist der Kasten leer, so beenden Sie die Suche, sonst wiederholen Sie das Verfahren ab Schritt 1.

Suchverfahren 3:
(1) Sie greifen zufällig eine Karte heraus.
(2) Hat diese den gewünschten Titel, so sind Sie fertig, ansonsten legen Sie sie zurück und wiederholen das Verfahren ab Schritt 1.

Suchverfahren 4:
(1) Sie greifen zufällig eine Karte heraus.
(2) Hat diese den gewünschten Titel, so sind Sie fertig, ansonsten legen Sie sie zurück.
(3) Sind seit Beginn der Suche zehn Minuten vergangen, geben Sie auf und beenden die Suche.
(4) Ansonsten wiederholen Sie das Verfahren ab Schritt 1.

Suchverfahren 5: (Vorbedingung: Karteikasten ist lexikographisch sortiert)
(1) Sie greifen die Karte, die in der Mitte liegt. Bei n Karten ist dies die $DIV(n+1,2)$-te Karte.
(2) Trägt diese den gewünschten Titel, beenden die Suche, ansonsten:
(3) Ist der gesuchte Titel alphabetisch vor dem Titel auf der ausgewählten Karte, so suchen Sie nur noch in der ersten Hälfte nach dem gleichen Verfahren ab Schritt 1 weiter.
(4) Ist der gesuchte Titel alphabetisch hinter dem Titel auf der ausgewählten Karte, so suchen Sie nur noch in der zweiten Hälfte nach dem gleichen Verfahren ab Schritt 1 weiter.
(5) Das Verfahren ist erfolglos beendet, wenn der letzte Kartenabschnitt leer ist.

Zu Aufgabe 4-1:
Welcher Algorithmus erfüllt welche der folgenden Eigenschaften? Begründen Sie Ihre Entscheidungen!

(a) terminierend (b) deterministisch (c) determiniert (d) partiell korrekt (e) total korrekt

Lösungsvorschlag:

Suchverfahren	terminierend	deterministisch	determiniert	partiell korrekt	total korrekt
1	×	×	×		
2	×	×	×	×	×
3			×	×	
4	×				
5	×	×	×	×	×

Anmerkungen:

- Suchverfahren 1
  - terminierend (Ja): Der Algorithmus führt nur einen Schritt aus und endet dann.
  - deterministisch (Ja): Abarbeitung immer in der selben Reihenfolge.
  - determiniert (Ja): Es wird bei gleicher Eingabe immer die gleiche Karte zurückgegeben, nämlich die erste.
  - partiell korrekt (Nein): Die erste Karte muss nicht unbedingt die gesuchte Karte sein.
  - total korrekt (Nein): Da nicht partiell korrekt.
- Suchverfahren 2
  - terminierend (Ja): In jeder Iteration wird der Eingabekasten echt kleiner, bis leer.
  - deterministisch (Ja): Immer die selbe Reihenfolge.
  - determiniert (Ja): weil deterministisch.
  - partiell korrekt (Ja): findet immer die richtige Karte, falls da. Sonst ist die Ausgabe leer.
  - total korrekt (Ja): weil partiell korrekt + terminierend
- Suchverfahren 3
  - terminierend(Nein): Falls Karte nicht vorhanden, terminiert der Algorithmus garantiert nicht. Trotzdem den Fall diskutieren, bei dem die gesuchte Karte garantiert im Kasten drin ist. In diesem Fall wäre der Algorithmus auch nicht terminierend. Denn, terminierend bedeutet dass der Algorithmus nach einer endlichen Zahl von Schritten zum Ende kommt. Gegenbeispiel: Zweite Karte ist gesucht, aber ich ziehe zufällig immer die erste Karte.
  - deterministisch (Nein): Nein, zufällige Karte ziehen bedeutet zufällige Reihenfolge der Suche.
  - determiniert (eigentlich ja!!): Weil entweder gibt es bei Terminierung die richtige Karte als Ausgabe zurück. Bei Nichtterminierung gibt es keine Ausgabe, also sind alle Ausgaben stets richtig und damit gleich bei gleicher Eingabe.
  - determiniert (je nach Argumentation: nein): Nein, weil es bei gleicher Eingabe (einer Karte die vorhanden ist) sein kann, dass der Algorithmus mal das richtige Ergebnis zurück gibt, und ein andres mal nicht terminiert. Für diese Teilaufgabe keine Punkte abziehen! - zu speziel.
  - partiell korrekt (Ja): Wenn was zurück kommt dann das richtige
  - total korrekt (Nein): Weil nicht terminierend.

Lösungsvorschlag:

- Suchverfahren 4
  - terminierend (Ja): Nach spätestens 10 Minuten muss der Algorithmus abbrechen.
  - deterministisch (Nein): Reihenfolge nicht eindeutig festgelegt.
  - determiniert (Nein): Für die selbe Karte kann manchmal das richtige Ergebnis rauskommen, aber manchmal das Ergebnis, das nach 10 min auf der Hand ist.
  - partiell korrekt (Nein): Nach 10 Minuten kann die falsche Karte (oder gar keine) auf der Hand sein, auch wenn die Karte im Kasten enthalten ist.
  - total korrekt (Nein): nicht part. korr.
- Suchverfahren 5
  - terminierend (Ja): Wenn die Karte enthalten ist, findet man sie in dem binären Teilbaum. Im anderen Fall findet man sie nicht.
  - deterministisch (Ja): Reihenfolge folgt einem eindeutigen Pfad in einem binären Baum.
  - determiniert (Ja): Existiert die gesuchte Karte, wird immer diese zurückgegeben. Existiert sie nicht, wird keine zurückgegeben.
  - partiell korrekt (Ja): Existiert die gesuchte Karte, wird immer diese zurückgegeben. Existiert sie nicht, wird keine zurückgegeben.
  - total korrekt (Ja): partiell korrekt + terminierend

Aufgabe 4-2 *Ganze Zahlen in Java*

In der Vorlesung haben Sie die maschinelle Darstellung von Zahlen kennengelernt. Außerdem haben Sie von mehreren Sorten oder Typen gehört, die diese Zahlendarstellungen repräsentieren. Einer der primitiven Typen zur Speicherung von ganzen Zahlen ist Integer oder kurz `int`. Sie haben ihn bereits in den vorherigen Übungen verwendet. Der Variablentyp kann dem Compiler so vermitteln, was für einen Wert er erwarten soll. Der Compiler kann dann einen Bereich im Speicher reservieren, um den Variablenwert dort abzulegen und für spätere Berechnungen zu verwenden. Das Benennen und die Typzuweisung einer Variablen nennt man Deklaration. Es gilt meistens, dass eine Variable erst deklariert werden muss, bevor man sie mit einem Wert initialisiert, d.h. ihr einen Wert zuweist.

```
int x;        /*Deklaration einer Integervariablen x*/
x = 2016;     /*Initialisierung mit 2016*/

int y = 42;   /*Deklaration und Zuweisung in einem Ausdruck*/

int z = x+y; /*Zuweisung durch andere Variablen*/
```

(a) Variablen des Typs `int` reservieren im Speicher 4 Byte. Ein Byte besteht aus 8 Bit, also belegt eine Integervariable 32 Bit. Können damit alle ganzen Zahlen dargestellt werden? Wenn nicht, wieviele Zahlen können dargestellt werden?

Lösungsvorschlag:
Nur endlich viele! Es sind 2^{32} unterschiedlich viele Zahlen.

(b) Welche Ausgabe liefert der folgende Code? Warum?
Hinweis: `==` ist der Vergleichsoperator in Java, der auf Gleichheit prüft und `True/False` ausgibt. Das einfache `=` weist der linken Seite den Wert der rechten Seite zu.

```
public class GanzeZahlen1 {
```

```
        public static void main(String[] args) {
                int i = 0b00000000000000000000000000000001;
                int j = 0b11111111111111111111111111111111;
                int k = 0b00000000000000000000000000000000;
                System.out.println(i+j==k);
        }
}
```

Lösungsvorschlag:

Der Code liefert true *(0,5 Pkte)*. $j = -i$ und damit die negative Repräsentation (Zweierkomplement)*(0,5 Pkte)*. Damit ist $i + j = 0 = k$.

(c) Welche Ausgabe liefert der folgende Code? Warum?

```
public class GanzeZahlen2 {
        public static void main(String[] args) {
                int x = 2147483647;
                System.out.println(x+1);

                int y = -2147483648;
                System.out.println(y-1);
        }
}
```

Lösungsvorschlag:

x ist die höchste darstellbare Integerzahl. Eine Erhöhung um 1 liefert die kleinstmögliche Integerzahl (y). Analog ist y die kleinstmögliche Integerzahl und $y - 1 = x$.

(d) Geben Sie jede Ausgabe `System.out.println(...)` im folgenden Javacode an. Begründen Sie jeweils in einem kurzen Satz, wie es zu diesem Wert kam. Versuchen Sie, sich als erstes selbst zu überlegen, welche Ausgabe Sie erwarten und überprüfen Sie Ihre Erwartung mit dem Java-Compiler.

```
1                   int a = 0b101;
2                   System.out.println(a);
3                   a = a / 2;
4                   System.out.println(a);
5                   ;;;;;;;;
6                   System.out.println(a);
7                   a += 1;
8                   System.out.println(a);
9                   a = 2;
10                  a *= a;
11                  System.out.println(a == 2);
12                  System.out.println(a = 2);
13                  a++;
14                  System.out.println(a);
15                  System.out.println(5-2+2);
16                  System.out.println(5/2*2);
17                  a = (int) 13.7;
18                  System.out.println(a + 5.3);
19                  System.out.println(a);
```

Lösungsvorschlag:

Ausgaben: Erklärung
5: da Binärdarstellung
2: da int-Division Nachkommastellen abschneidet
2: Semikolons aneinandergereiht sind leere Ausdrücke. Leere Ausdrücke bewirken nichts.
3: Inkrement um 1, a = a + 1 = 2 + 1
false: a = 2*2 wird mit 2 verglichen
2: a wird auf 2 gesetzt und ausgegeben
3: Inkrement um 1, a++ entspricht a = a + 1
5: 5-2 ist 3 und 3 + 2 = 5
4: 5/2 = 2 (wie oben Integerdivision) und 2 * 2 = 4
18.3: a = (int) 13.7 = 13, da typecast. a + 5.3 = 13 + 5.3 = 18.3, aber dies ist nun float
13: Zeile zuvor ist keine Zuweisung.

Aufgabe 4-3 *Primzahlgenerator,* `While`

In dieser Aufgabe werden Sie einen Primzahlgenerator programmieren. Benutzen Sie die Javadatei `FindPrimes.java` als Grundgerüst. Geben Sie Ihre Lösung in `FindPrimes.java` ab.

(a) Sie sollen den euklidischen Algorithmus verwenden. Dazu gibt es im vorliegenden Code bereits ein Basiskonstrukt. Ergänzen Sie den fehlenden Code gemäß:
https://de.wikipedia.org/wiki/Euklidischer_Algorithmus#Der_klassische_Algorithmus
Die Syntax für eine `while`-Schleife können Sie hier bereits sehen. Die `main`-Methode enthält zwei exemplarische Aufrufe von `ggT`, die natürlich nicht die vollständige Korrektheit beweisen, aber Ihnen einen Hinweis darauf geben, ob Ihre Lösung zumindest teilweise richtig arbeitet.

(b) In der `main`-Methode ist bereits ein wenig Code vorgegeben. Ergängen Sie Ihren Code an den mit `TODO` markierten Stellen. An der ersten Stelle sollen Sie eine `while`-Schleife benutzen, um alle Vorgänger des aktuellen Prim-Kandidaten auf gemeinsame Teiler zu prüfen. Dazu können Sie mit `ggT()` die in (a) vervollständigte Methode nutzen. Die Faktoren von `ggTProdukt` sind alle ggT's des Kandidaten n und seiner Vorgänger $2, \ldots, n-1$. Wenn eine Zahl prim ist, so sind alle größten gemeinsamen Teiler 1 und damit ist auch das Produkt gleich 1:

$$ggTProdukt(n) = ggT(2,n) \cdot ggT(3,n) \cdot \ldots \cdot ggT(n-1,n) = 1 \iff n \text{ prim}$$

(c) Falls `ggTProdukt == 1` ist, so ist der Kandidat prim. Erweitern Sie Ihr Programm, sodass alle Primzahlen zwischen 1 und 1000 ausgegeben werden.

Lösungsvorschlag:

```
1  /**
2   * Programm zum Finden von Primzahlen.
3   * Es wird der euklidische Algorithmus
4   * zum Berechnen des ggT's benutzt.
5   */
6  public class FindPrimesLsg {
7
8    /**
9     * Implementation des euklidischen Algorithmus
10    * Zwei ganze Zahlen werden als Eingabe akzeptiert
11    * Die Ausgabe ist der ggT beider Eingabezahlen.
12    * Der Aufruf kann innerhalb dieser Datei durch
13    * ggT(Zahl1, Zahl2) erfolgen.
14    */
15   public static int ggT(int a, int b) {
16     while(b != 0) {
17       /* TODO a):
18        * euklidischer Algorithmus
19        */
20       if(a > b)
21         a -= b;
22       else
23         b -= a;
24     }
25     return a;
26   }
27
28   /**
29    * main Methode.
30    * Keine Eingabe wird benoetigt.
31    */
32   public static void main(String[] args) {
33
34     // Partielle Tests fuer die Korrektheit des ggT
35     // Hier sehen Sie auch einen Aufruf fuer ggT() und
36     // koennen in an spaeterer Stelle auch so verwenden
37     System.out.println(ggT(73, 21) == 1);
38     System.out.println(ggT(63, 42) == 21);
39
40     // Dies ist der aktuelle Kandidat, der getestet wird,
41     // ob er prim ist.
42     int kandidat = 1;
43
44     // Jeder Kandidat bis zur oberen Schranke (z.B. 100) wird
45     // getestet
46     while(kandidat < 1000) {
47       // erhoehe in jedem Durchgang kandidat um 1, um die
48       // naechstgroessere Zahl zu erhalten.
49       kandidat = kandidat + 1;
50
51       // wir berechnen inkrementell das Produkt aller
52       // ggT(i,kandidat)
53       int ggTProdukt = 1;
```

Lösungsvorschlag:

```
54        /** TODO b):
55         * - Fuehren Sie eine neue Variable "int i" ein.
56         * - Eine while-Schleife soll alle i > 1 durchlaufen,
57         *   die kleiner als der Kandidat sind
58         * - Fuer jedes i soll der ggT mit dem Kandidaten
59         *   berechnet werden
60         * - Dann wird dieser Wert mit dem bisherigen ggTProdukt
61         *   multipliziert
62         * - Nicht vergessen: i muss mit jedem Durchlauf erhoeht werden.
63        */
64
65        int i = 2;
66        while(i < kandidat) {
67          ggTProdukt = ggTProdukt * ggT(i,kandidat);
68          i = i+1;
69        }
70
71
72        // TODO c): Gib Zahl aus, genau dann wenn ggTProdukt == 1
73        if(ggTProdukt == 1) {
74          System.out.print(" " + kandidat);
75        }
76        //else {
77        //  System.out.println(kandidat + " ist nicht prim!");
78        //}
79
80      }
81    }
82  }
```