

Algorithmen und Datenstrukturen
SS 2018

Übungsblatt 1: Grundlagen

Tutorien: 16.04-20.04.2018

Aufgabe 1-1 *Basics zu Algorithmen*

Geben Sie bei folgenden Algorithmen an, welches Problem sie lösen.
Bestimmen Sie außerdem, welche grundlegenden Eigenschaften erfüllt sind (Allgemeinheit, Determiniertheit, Determinismus, Terminierung, Effizienz).

```
(a) public static boolean f0(int a) {  
    if (a == 0)  
        return true;  
    if (a == 1)  
        return false;  
    if (a < 0)  
        return f0(-a);  
    return f0(a - 2);  
}
```

```
(b) public static boolean f1(String str1, String str2) {  
    char[] array1 = new char[Character.MAX_VALUE];  
    char[] array2 = new char[Character.MAX_VALUE];  
    for(char c : str1.toCharArray()){  
        array1[c]++;  
    }  
    for(char c : str2.toCharArray()){  
        array2[c]++;  
    }  
    for(char c = 0; c < Character.MAX_VALUE; c++){  
        if(array1[c] != array2[c])  
            return false;  
    }  
    return true;  
}
```

```
(c) public static double f2(double a, int n) {  
    double x = 1.0;  
    for (int i = 0; i < n; i++){  
        x = 0.5 * (x + a / x);  
    }  
    return x;  
}
```

```

(d) public static double f3(int n) {
    double sum = 0.0;
    for (int i = 0; i < n; i++) {
        double x = Math.random();
        double y = Math.random();
        if(Math.sqrt(x*x+y*y) <= 1.0)
            sum++;
    }
    return 4.0*sum/n;
}

(e) public static int f4(int a, int b) {
    if (b == 1)
        return a;
    return a * f4(a, b - 1);
}

(f) public static int f5(int a, int b) {
    if (a == 0)
        return b + 1;
    if (b == 0)
        return f5(a - 1, 1);
    return f5(a - 1, f5(a, b - 1));
}

(g) public static void f6() {
    LinkedList<Integer> list = new LinkedList<Integer>();
    int n = 2;
    list.add(n);
    n = 3;
    while (true) {
        n += 2;
        boolean add = true;
        for (Integer m : list) {
            if (n % m == 0)
                add = false;
        }
        if (add) {
            list.add(n);
            System.out.println(n);
        }
    }
}

```

Aufgabe 1-2 Zusatzaufgabe: Spaß mit Determiniertheit und Determinismus

Um den Unterschied zwischen Determiniertheit und Determinismus zu verstehen, betrachten wir folgende Funktion, die zwar determiniert aber nicht deterministisch ist:

```
public static int f7() {
    int a = 0;
    int b = 0;
    if (Math.random() < 0.5) {
        a++;
        b++;
    } else {
        b++;
        a++;
    }

    return a + b;
}
```

Die Funktion wird immer $1 + 1 = 2$ zurückgeben, ist also determiniert (gleiche Eingabe führt zu gleicher Ausgabe).

Die Zustände, die bei der Abarbeitung durchlaufen werden, sind aber vom Ergebnis von *Math.random()* abhängig, das zufällig ist. Die Abarbeitungsfolge ist also nicht eindeutig bestimmt. Es kann sein, dass der *if*-Block oder der *else*-Block ausgeführt wird. Daher ist die Funktion nicht deterministisch.

Lösungsvorschlag:

a) Gibt true zurück wenn a gerade ist, false wenn a ungerade ist.

Allgemeinheit: gegeben, löst für alle ganzen Zahlen

Determiniertheit: gegeben, Ausgabe ist stets die Gleiche bei gleicher Eingabe

Determinismus: gegeben, Ausführung ist nur von der Eingabe abhängig

Terminierung: terminiert für alle ganzen Zahlen nach höchstens $n+2$ Schritten

Korrektheit: gegeben, berechnet die Aufgabe

Effizient: könnte effizienter sein, z.B. mit Bitoperationen (schaue letztes Bit an in Binärdarstellung) - Modulo

b) Anagrammtest, bestehen zwei Strings aus dem gleichen Multiset von Buchstaben?

Allgemeinheit: gegeben, funktioniert für Strings

Determiniertheit: gegeben, Ausgabe ist stets die Gleiche bei gleicher Eingabe

Determinismus: gegeben, Ausführung ist nur von der Eingabe abhängig

Terminierung: terminiert für alle Strings endlicher Länge

Korrektheit: gegeben, berechnet die Aufgabe

Effizient: relativ effizient

c) Heron Verfahren, Babylonisches Wurzelziehen

Allgemeinheit: gegeben für alle positiven Zahlen

Determiniertheit: gegeben, Ausgabe ist stets die Gleiche bei gleicher Eingabe

Determinismus: gegeben, Ausführung ist nur von a und n abhängig

Terminierung: terminiert für alle positiven Zahlen

Korrektheit: partiell, wenn er ein Ergebnis liefert, ist es (approximiert) korrekt

Effizient: relativ effizient

Anmerkungen: Eigentlich nicht korrekt, da nicht das Ergebnis rauskommt. Aber innerhalb einer gewissen Genauigkeit dann irgendwie doch.

Man muss zwischen mathematisch-strikter Korrektheit und der Möglichkeit einer Maschine unterscheiden. Paradebeispiel $0.1+0.2 == 0.3$, was kommt da raus? ;)

d) Monte Carlo berechnet Pi; Viele Zufallspunkte auf den Vierteleinheitskreis in $[0, 1]^2$ schießen und das Verhältnis von Punkten mit Abstand maximal 1 zum Ursprung zu allen Punkten beschreibt dann Pi.

Allgemeinheit: nein, ist eine Einzelaufgabe (Pi berechnen)

Determiniertheit: ja (Ergebnis ist immer Pi)

Determinismus: nicht gegeben, Zwischenschritte sind zufallsabhängig

Terminierung: terminiert nach n Iterationen

Korrektheit: korrekt

Effizient: eher nicht

Lösungsvorschlag:

e) Potenz a^b berechnen

Allgemeinheit: gegeben, löst alle ganzzahligen Potenzen

Determiniertheit: ja

Determinismus: ja

Terminierung: nein, Bsp $b = 0$

Korrektheit: partiell: wenn er terminiert, ist das Ergebnis korrekt

Effizient: Schleife wäre effizienter

f) Ackermannfunktion (<https://de.wikipedia.org/wiki/Ackermannfunktion>)

Allgemeinheit: ja

Determiniertheit: ja - stets das gleiche Ergebnis

Determinismus: ja - stets der gleiche Ablauf

Terminierung: terminiert immer, dauert aber sehr lange

Korrektheit: korrekt

Effizient: fragwürdig

Beispiel für eine Funktion, die zwar Korrekt ausgewertet wird und immer terminiert, aber sehr rechenintensiv ist.

Anwendungen nur für Benchmark-Tests

g) Primzahlen finden

Allgemeinheit: ja

Determiniertheit: ja - stets das gleiche Ergebnis

Determinismus: ja - stets der gleiche Ablauf

Terminierung: terminiert nicht

Korrektheit: partiell korrekt: würde es terminieren (z.B. Abbruch) dann ist das Ergebnis korrekt

Effizient: nicht effizient

Sehr ineffizient, funktioniert aber.

Terminiert nie.

Addendum

Qualitätsmerkmale sind anwendungsabhängig sehr diskutabel, was bei dieser Aufgabe der Kern war. Bei manchen Problemen gibt es eine offensichtliche Lösung, bei anderen gibt es verschiedene Ansätze. Allgemeinheit kann subjektiv sein. Meist ist es nicht sinnvoll, eine algorithmische Strategie auf einen einzelnen Fall anzuwenden. Als Beispiel liefert f2 die Quadratwurzel für beliebige reelle Zahlen und Iterationsschranken, was schon relativ allgemein ist. Allerdings könnte man auch den Startwert in die Eingabe bringen, statt ihn bei $x=1.0$ beginnen zu lassen.

Zur Determiniertheit ist die Monte-Carlo-Methode, um π zu errechnen, ein diskussionswürdiger Kandidat. Wir werden nie als Ergebnis eines auf einem Computer ausgeführten Algorithmus die Zahl π als Ergebnis erhalten, sondern immer nur eine Näherung. Als Algorithmus für π liegt keine Determiniertheit vor. Als Algorithmus für eine Näherung für π allerdings schon, denn das Ergebnis ist bzgl. einer gewählten Genauigkeit „gleich“. Hier ist es wichtig, ob das Kriterium eine mathematische Gleichheit oder eine messbare Gleichheit ist. Wählt man die mathematische Perspektive und ignoriert technische Faktoren, so erreichen wir hier keine Determiniertheit. Für Korrektheit gilt das gleiche.

Auch bei der Effizienz lässt sich gut streiten. Unsere Definition beruht auf der Wirtschaftlichkeit des Algorithmus: Wie kann man mit möglichst wenig Ressourcen das Problem lösen? Ein effizienterer Algorithmus löst das Problem mit weniger Speicherbedarf und in schnellerer Zeit. Aber auch die Wartbarkeit und monetärer Aufwand sind Einflussfaktoren. Weil letztere aber sehr stark schwanken und wir die Laufzeit und Speicher gut quantifizieren und von der technischen Ausführung lösen können, reden wir hier fast ausschließlich von Laufzeit- und Speicherkomplexität.