Teil 3: Algorithmen

Ein Kernkonzept zur Lösung komplexer mathematischer Probleme sind *Algorithmen*.

1 Algorithmen

Definition 1 (Algorithmus)**.** *Ein* Algorithmus *ist eine Berechnungsvorschrift zur Lösung eines Problems. Er setzt sich zusammen aus einer Folge von endlich vielen, wohldefinierten Einzelschritten.*

Anmerkungen

- Algorithmen können in vielfältiger Weise dargestellt werden.
  - Beispiele: Pseudocodes, Java-Codes, Programmablaufpläne, ...
- Wir verwenden hier eine einfach zu verstehende, textuelle Darstellung.

Wir schauen uns zunächst ein einfaches Beispiel an:

Beispiel 1: Der folgende Algorithmus berechnet die Fakultät $n!$ einer natürlichen Zahl n.

Algorithmus FAK. **Eingabe:** $n \in \mathbb{N}_0$.

1. Setze $k = 1$
2. Setze $i = 2$
3. Solange wie $i \leq n$ gilt, mache:
4. Ersetze k durch $k \cdot i$.
5. Ersetze i durch $i + 1$.
6. Gib k als Ergebnis zurück.

Erklärung:

- Ein Algorithmus besitzt eine festgelegte Eingabe (hier n).
- Ein Algorithmus besteht aus einer Abfolge von Zeilen mit textuellen *Anweisungen*.
  - Die Anweisungen definieren und verändern *Variablen* (hier k, i und n).
- Die Zeilen werden von oben nach unten ausgeführt.
  - *Schleifen* ändern die Ausführungsreihenfolge (siehe Zeile 2).

  - Dabei werden die eingerückten Zeilen wiederholt, solange die angegebene Bedingung gilt.
  - Ist die Bedingung nicht erfüllt, werden die eingerückten Zeilen einfach übersprungen.
- Ein Algorithmus terminiert (= endet) durch Rückgabe eines Ergebnisses.

Beispiel 1 (Fortsetzung): Ausführung des Algorithmus

Führen wir den Algorithmus mit $n = 3$ aus, ergibt sich folgende Folge von Anweisungen:

1. In Zeile 1 wird $k = 1$ gesetzt.
2. In Zeile 2 wird $i = 2$ gesetzt.
3. In Zeile 3 testet der Algorithmus, ob $2 \leq 3$ gilt, und stellt fest, dass dies der Fall ist. Daher werden die Anweisungen in Zeile 4 und 5 ausgeführt.
4. Nach Ausführung von Zeile 4 gilt $k = 2$ (und weiterhin $i = 2$).
5. Nach Ausführung von Zeile 5 gilt $i = 3$ (und weiterhin $k = 2$).
6. Da das Ende der eingerückten Zeilen erreicht ist, wird nun erneut geprüft, ob die Bedingung in Zeile 3 erfüllt ist.
7. In Zeile 3 testet der Algorithmus, ob $3 \leq 3$ gilt, und stellt wieder fest, dass dies der Fall ist. Daher werden die Anweisungen in Zeile 4 und 5 nochmal ausgeführt.
8. Nach Zeile 4 gilt $k = 6$ (und weiterhin $i = 3$).
9. Nach Zeile 5 gilt $i = 4$ (und weiterhin $k = 6$).
10. Da das Ende der eingerückten Zeilen erreicht ist, wird nun erneut geprüft, ob die Bedingung in Zeile 3 erfüllt ist.
11. In Zeile 3 testet der Algorithmus $4 \leq 3$ und stellt fest, dass dies nicht mehr erfüllt ist. Daher geht es nun nach der Schleife mit Zeile 6 weiter.
12. In Zeile 6 terminiert der Algorithmus mit dem Ergebnis $k = 6$.

Anmerkungen:

- Offenbar berechnet der Algorithmus das korrekte Ergebnis für $n = 3$.
- Der Algorithmus ist nicht nur für eine bestimmte Eingabe (wie $n = 3$) konstruiert, sondern funktioniert für *beliebige* (= alle möglichen) Eingaben $n \in \mathbb{N}$. Das ist ein entscheidendes Merkmal von Algorithmen!

Aufgabe 1: Führen Sie den Algorithmus mit der Eingabe $n = 0$ und $n = 1$ aus.

Aufgabe 2: Wäre der Algorithmus korrekt, wenn wir i in Zeile 2 mit 1 initialisieren?

Beispiel 2: Der folgende Algorithmus prüft, ob eine Zahl n höchstens so groß ist, wie die Hälfte einer Zahl m.

Algorithmus FAK. **Eingabe:** $n, m \in \mathbb{R}$.

1. Setze $h = \frac{m}{2}$
2. Falls $n \leq h$, dann:
3. Gib `Wahr` als Ergebnis zurück.
4. Sonst:
5. Gib `Falsch` als Ergebnis zurück.

Aufbau von Algorithmen In unserer Darstellungsform setzen sich Algorithmen aus folgenden grundlegenden Einzelschritten zusammen:

- Einführen von Variablen (z.B. Setze $i = 2$.)
- Arithmetische Operationen mit im Vorfeld eingeführten Variablen und Konstanten. Erlaubt sind Addition, Subtraktion, Multiplikation und Division. (z.B. $i - 1$, $i \cdot k$).
- Überschreiben von Variablen, dabei dürfen die arithmetischen Operationen verwendet werden. (z.B. Ersetze i durch $i + 1$.)
- Fallunterscheidungen: Falls Bedingung erfüllt tue dies, sonst tue das.
- Bedingungen sind beispielsweise Vergleiche mit im Vorfeld eingeführten Variablen und Konstanten wie $i < j$, $i = j$, $i \geq j$, $i \neq j$. Diese dürfen mit `und` und `oder` verknüpft und mit `nicht` negiert werden.
- Schleifen: Wiederhole eingerückten Inhalt, solange wie die Bedingung erfüllt ist (siehe oben).
- Unterprogrammaufrufe (besprechen wir gleich).
- Rückgabe des Ergebnis (beendet das Programm).

Aufgabe 3: Entwickeln Sie einen Algorithmus ABS zur Berechnung des Betrages $|x|$ einer reellen Zahl x.

Aufgabe 4:

(a) Entwickeln Sie einen Algorithmus POT zur Berechnung der Potenz a^n zweier Zahlen $a \in \mathbb{R}$ und $n \in \mathbb{N}_0$. *Tipp:* $a^n = 1 \cdot \underbrace{a \cdot a \cdot \ldots \cdot a}_{n \text{ Faktoren}}$.

(b) Berechnen Sie mit Ihrem Algorithmus die Potenzen 2^4, 4^0 und 2^{-1}. Stellen Sie dabei die Änderungen der Variablen nachvollziehbar dar. Was fällt Ihnen auf?

2 Unterprogramme

Den oben formulierten Algorithmus FAK können wir als *Unterprogramm* zur Berechnung der Fakultät in weiteren Algorithmen verwenden.

Beispiel 3: Der Binomialkoeffizient

$$\binom{n}{k} = \begin{cases} \frac{n!}{k! \cdot (n-k)!} & 0 \leq k \leq n \\ 0 & sonst \end{cases}$$

lässt sich mittels FAK leicht über den folgenden Algorithmus berechnen:

Algorithmus BINOM. **Eingabe:** $n, k \in \mathbb{N}_0$

1. Falls $k > n$, dann:
2. Gib 0 als Ergebnis zurück.
3. Sonst:
4. Setze $x = \text{FAK}(n)$.
5. Setze $y = \text{FAK}(k)$.
6. Setze $z = \text{FAK}(n - k)$.
7. Gib $x/(y \cdot z)$ als Ergebnis zurück.

Anmerkungen:

- Der Algorithmus BINOM kann wiederum als Unterprogramm in anderen Algorithmen verwendet werden, in denen Binomialkoeffizienten eine Rolle spielen.
- Mit Unterprogrammen lassen sich beliebig komplexe Algorithmen aus einfachen Grundbausteinen zusammen setzen.

Aufgabe 5:

(a) Entwickeln Sie einen Algorithmus EULER zur Berechnung der Eulerschen Zahl e, indem Sie folgende Summe auswerten:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

Verwenden Sie den Algorithmus FAK als Unterprogramm.

(b) Welches Problem ergibt sich bei der Ausführung Ihres Algorithmus?

(c) Modifizieren Sie Ihren Algorithmus, sodass dieser die Auswertung der Summe stoppt, sobald ein Summand kleiner als $0,001$ ist.

3 Algorithmen zur Teilbarkeit

Im Folgenden brauchen wir Algorithmen zur Teilbarkeit ganzer Zahlen.

Vorüberlegung:

- Im Allgemeinen lassen sich zwei ganze Zahlen a und b nicht restlos teilen.
  - Beispiel: $14/4 = 3$ Rest 2.
- Wir bezeichnen mit $\lfloor a/b \rfloor$ das abgerundete Ergebnis der Division (hier 3).
- Wir bezeichnen mit $a \bmod b$ den dazugehörigen Divisionsrest (hier 2).
- Für alle ganzen Zahlen a und $b \neq 0$ gilt:

$$a = a \bmod b + b \cdot \lfloor a/b \rfloor \tag{1}$$

Beispiel 4: Für $a = 14$ und $b = 4$ erhalten wir:

- $\lfloor 14/4 \rfloor = 3$
- $14 \bmod 4 = 2$
- $14 = 2 + 4 \cdot 3 = 14 \bmod 4 + 4 \cdot \lfloor 14/4 \rfloor$

Aufgabe 6: Entwickeln Sie einen Algorithmus DIV zur Berechnung des Ergebnisses $\lfloor a/b \rfloor$ der ganzzahligen Division. Wenden Sie Ihren Algorithmus auf $a = 14$ und $b = 4$ an.

Hinweis: Zählen Sie mit, wie häufig b in a „hinein passt".

Aufgabe 7: Entwickeln Sie einen Algorithmus namens MOD zur Berechnung des Divisionsrestes $a \bmod b$. Verwenden Sie Ihren Algorithmus DIV als Unterprogramm.

Definition 2. *Der* größte gemeinsame Teiler *zweier natürlicher Zahlen* a *und* b *ist die größte natürliche Zahl* d*, durch die sowohl* a *als auch* b *restlos teilbar ist (d.h.* $a \bmod d = b \bmod d = 0$*).*

Aufgabe 8: Entwickeln Sie einen Algorithmus namens GGT zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen $a, b \in \mathbb{N}_+$. Verwenden Sie Ihren Algorithmus MOD als Unterprogramm.

Aufgabe 9: Entwickeln Sie einen Algorithmus SIN zur näherungsweisen Berechnung der Sinus-Funktion mittels der Taylor-Entwicklung

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \dots$$

Stoppen Sie die Auswertung sobald ein Summand betragsmäßig kleiner als $0,001$ ist.

Hinweis: Verwenden Sie POT, FAK (und eventuell MOD) als Unterprogramme.

4 Rekursion

Eine *Rekursion* liegt vor, wenn ein Algorithmus sich selbst als Unterprogramm aufruft.

Beispiel 5: Wir können die Fakultät $n!$ einer natürlichen Zahl $n \in \mathbb{N}_0$ auch rekursiv berechnen. Dazu nutzen wir folgenden Zusammenhang aus:

$$n! = \begin{cases} 1, & \text{falls } n = 0 \text{ oder } n = 1, \\ n \cdot (n-1)!, & \text{sonst.} \end{cases}$$

Algorithmus REKFAK. **Eingabe:** $n \in \mathbb{N}_0$.

1. Falls $n = 0$ oder $n = 1$, dann:
2. Gib 1 als Ergebnis zurück.
3. Sonst:
4. Setze $k =$ REKFAK$(n-1)$.
5. Gib $n \cdot k$ als Ergebnis zurück.

Wenn wir den Algorithmus mit der Eingabe $n = 3$ ausführen, erhalten wir folgenden Ablauf:

- In Zeile 1 stellt der Algorithmus $3 \neq 0$ und $3 \neq 1$ fest.
- In Zeile 4 ruft der Algorithmus das Unterprogramm REKFAK mit der Eingabe $n-1 = 2$ auf. Die Ausführung des Algorithmus stoppt nun an dieser Stelle, bis das Ergebnis des Unterprogramms REKFAK(2) vorliegt.
  - In Zeile 1 stellt der Algorithmus $2 \neq 0$ und $2 \neq 1$ fest.
  - In Zeile 4 ruft der Algorithmus das Unterprogramm REKFAK mit der Eingabe $n-1 = 1$ auf. Die Ausführung des Algorithmus stoppt nun an dieser Stelle, bis das Ergebnis des Unterprogramms REKFAK(1) vorliegt.
    - In Zeile 1 stellt der Algorithmus $1 = 1$ fest und gibt in Zeile 2 1 als Ergebnis an das übergeordnete Unterprogramm REKFAK(2) zurück.
  - In Zeile 5 gibt der Algorithmus das Ergebnis $n \cdot k = 2 \cdot 1 = 2$ an das übergeordnete Unterprogramm REKFAK(3) zurück.

- In Zeile 5 gibt der Algorithmus das Ergebnis $n \cdot k = 3 \cdot 2 = 6$ zurück.

Berechnet wird also insgesamt:
$3 \cdot (3-1)! = 3 \cdot (2 \cdot (2-1)!) = 3 \cdot (2 \cdot (1!)) = 3 \cdot (2 \cdot (1)) = 3 \cdot 2 = 6.$

Achtung: Hier wird also zuerst die *innerste* Klammer ausgewertet, was dem Rekursionsabbruch im ersten Fall entspricht. Danach können mit diesem Ergebnis nach und nach auch die äußeren Klammern ausgewertet werden.

Anmerkungen:

- Ein rekursiver Algorithmus benutzt in mindestens einer Anweisung sich selbst als Unterprogramm.
- Er kann sich selbst aber auch mehrfach und mit verschiedenen Eingaben als Unterprogramm nutzen.
- Wichtig ist ein *Rekursionsabbruch* (hier mit $n = 1$), sonst endet der Algorithmus nicht. Diesen haben wir über die Fallunterscheidung definiert.
- In rekursiven Algorithmen ergibt sich eine Hierarchie verschiedener Instanzen (= Kopien) des Algorithmus. Mit Einrückung können wir die verschiedenen Stufen der Rekursion sichtbar machen. (Im Beispiel: Ebene 1: Punkte. Ebene 2: Striche. Ebene 3: Sternchen)

Aufgabe 10: Entwickeln Sie einen rekursiven Algorithmus zur Berechnung der Potenz a^n zweier Zahlen $a \in \mathbb{R}$ und $n \in \mathbb{N}_0$. Berechnen Sie damit 2^4.

Aufgabe 11: Entwickeln Sie einen rekursiven Algorithmus zur Berechnung des Binomialkoeffizienten $\binom{n}{k}$ zweier Zahlen $n, k \in \mathbb{N}_0$. Verwenden Sie dafür folgenden Zusammenhang:

$$\binom{n}{k} = \begin{cases} 0, & \text{falls } k > n, \\ 1, & \text{falls } k = 0 \text{ oder } k = n, \\ \binom{n-1}{k-1} + \binom{n-1}{k}, & \text{sonst.} \end{cases}$$