Der Euklidische Algorithmus

(Euklid von Alexandria ca. 365-300 v.Chr.)

Realisierung in den Versionen: Subtraktionsform, Divisionsform, iterativ, rekursiv und regelbasiert

Nimmt man abwechselnd immer das Kleinere vom Größeren weg,
dann muß der Rest schließlich die vorhergehende Größe messen ...
Euklid, Die Elemente, Zehntes Buch §3

Autor: Jochen Ziegenbalg
Email: ziegenbalg.edu@gmail.com

Basisliteratur
J. Ziegenbalg: Algorithmen - von Hammurapi bis Gödel, Verlag Harri Deutsch, Frankfurt am Main 2007, S. 60 ff

■ Einführung

Der Begriff des größten gemeinsamen Teilers (GGT) spielt eine sehr wichtige Rolle in der Mathematik wie auch im Mathematikunterricht (z.B. in der Bruchrechnung im Zusammenhang mit der Ermittlung des Hauptnenners zweier Brüche, in Verbindung mit dem Kürzen u.s.w.). Es gibt höchst unterschiedliche Verfahren zur Ermittlung des größten gemeinsamen Teilers zweier natürlicher Zahlen a und b.

Im Schulunterricht wird überwiegend eine Methode praktiziert, die auf der Primfaktorzerlegung der Zahlen a und b beruht. Historisch, sowie aus Optimalitäts- und innermathematischen Gründen ist der Euklidische Algorithmus zur Ermittlung des GGT von größter Bedeutung. Er bietet darüber hinaus den Vorteil, daß er sehr anschaulich zu beschreiben ist, daß er im engsten Zusammenhang mit einem grundlegenden Thema des Primarstufenunterrichts steht, nämlich mit dem Verfahren der Division mit Rest, daß er intensiv mit anderen wichtigen mathematischen Themen vernetzt ist (Fibonacci-Zahlen, Kettenbrüche, Goldener Schnitt, Restklassenringe, Verschlüsselungsverfahren: Public Key Cryptography, RSA-Verfahren, ...) und daß er in natürlicher Weise zu fundamentalen philosophischen Fragen führt (Kommensurabilität).

Im folgenden sollen zwei Zugänge zum Euklidischen Algorithmus beschrieben werden: der erste basiert auf der Division mit Rest; der zweite auf dem Prinzip der Rekursion. Natürlich sind beide Zugänge letztlich gleichwertig.

■ Realisierung

■ *1. Argumentationsstrang*: Über die Division mit Rest

Welcher Zugang auch immer für die Behandlung der Division natürlicher Zahlen in der Primarstufe gewählt wird, Hintergrund ist stets der

> **Satz von der Division mit Rest:** Zu je zwei natürlichen Zahlen a und b (mit $b > 0$) gibt es stets eindeutig bestimmte nichtnegative ganze Zahlen q und r mit der Eigenschaft $a = q \cdot b + r$ und $0 \leqslant r < b$.

Der Beweis des Satzes hängt davon ab, wie man die natürlichen Zahlen konstruiert hat. Die vollständige Induktion (in irgendeiner Form) spielt dabei stets eine ausschlaggebende Rolle (näheres zur vollständigen Induktion in [Ziegenbalg, Abschnitt 3.3]).

Im folgenden Beispiel sei $a = 17$ und $b = 5$ gewählt. Die nach dem Satz von der Division existierenden Zahlen q und r haben dann die Werte $q = 3$ und $r = 2$. Die in dem Satz auftretende Gleichung lautet dann $17 = 3 \cdot 5 + 2$.

Ebenso wichtig wie ein formaler Beweis ist die Veranschaulichung des Sachverhalts. Dazu stellen wir uns die Zahlen a und b, ganz im Sinne der Griechen, als Strecken vor; a möge die größere und b die kleinere Strecke sein. Dann kann man b einmal oder mehrmals auf a abtragen (bzw. „von a wegnehmen“), bis nichts mehr übrig bleibt oder bis ein Rest übrig bleibt, der kleiner ist als b. Die im obigen Satz auftretende Zahl q ist die Vielfachheit, mit der man b „ganz“ auf a abtragen kann; r ist der Rest, der danach übrig bleibt. Es ist offensichtlich, daß r kleiner als b ist (wie im Satz formuliert). Wenn $r = 0$ ist, sagt man auch, daß die Strecke b die Strecke a mißt bzw. daß die Zahl b die Zahl a teilt.

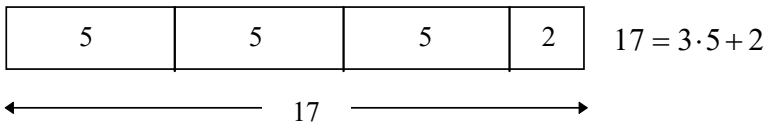


Abbildung 3.7

Diese Veranschaulichung macht deutlich, warum die Division häufig als „iterierte“ (wiederholte) Subtraktion erklärt und eingeführt wird.

Die im obigen Satz genannten Zahlen q und r sind offensichtlich durch a und b eindeutig bestimmt, denn sie ergeben sich ja eindeutig aus dem gerade beschriebenen „Abtrage-Verfahren“. Man kann q und r deshalb auch als Funktionswerte geeigneter Funktionen deuten. In den meisten Programmiersprachen werden diese Funktionen als *Div* und *Mod* bezeichnet:

$$\text{Div}: \quad (a, b) \to \text{Div}(a, b) = q$$
$$\text{Mod}: \quad (a, b) \to \text{Mod}(a, b) = r$$

Auch in *Mathematica* heißt die Rest-Funktion Mod; die Div-Funktion für die Ganzzahl-Division heißt in *Mathematica* jedoch Quotient.

```
Quotient[17, 5]
```

```
3
```

```
Mod[17, 5]
```

```
2
```

Die entscheidende Idee des Euklidischen Algorithmus besteht nun darin, den Satz von der Division mit Rest nach dem Prinzip der "*Wechselwegnahme*" zu iterieren (man vergleiche dazu das Eingangszitat). Dazu ersetzt man nach der Durchführung der Division mit Rest die ursprünglich größere Strecke a durch die ursprünglich kleinere Strecke b und b durch den Rest r. Mit diesen neuen Zahlen (oder Strecken) a und b führt man wiederum das Verfahren der Division mit Rest durch und erhält ein neues q und ein neues r. Mit diesen Strecken verfährt man wiederum nach dem Prinzip der Wechselwegnahme und nimmt die kleinere so lange von der größeren weg wie es geht.

Der Euklidische Algorithmus eignet sich hervorragend zur Abarbeitung mit einem Computer. Die ursprüngliche Form der Wechselwegnahme bezeichnen wir als die *Subtraktionsform* des Euklidischen Algorithmus. Wir formulieren zunächst in der Umgangssprache:

```
1:  EuklidSubtraktionsform(a,b)
2:    Solange a und b beide von Null verschieden sind,
3:      fuehre folgendes aus:
4:      Wenn a ≥ b, so ersetze a durch a-b,
5:                  sonst ersetze b durch b-a.
6:    Die uebrig bleibende,von Null verschiedene ganze
7:    Zahl ist der gesuchte groesste gemeinsame
8:    Teiler GGT(a,b)
```

■ Subtraktionsform (iterative Version)

```
EuklidSub[a0_, b0_] :=
 Module[{a = a0, b = b0},
  While[Not[a * b == 0],
   Print[a, "   ", b];
   If[a ≥ b, a = a - b, b = b - a]];
  Return[a + b] ]

EuklidSub[136, 60]

136   60

76   60

16   60

16   44

16   28

16   12

4   12

4   8

4   4

4
```

In der Divisionsform des Euklidischen Algorithmus werden die iterierten Subtraktionen (bei gleichem b) "auf einen Schlag", eben durch Division, durchgeführt.

■ Divisionsform (iterative Version)

Im folgenden Programm dient die (globale) Kontroll-Variable `verbose` der Steuerung des Ausdruckens: Im Falle "`Verbose == True`" werden die Zwischenergebnisse (insbesondere zu Lehrzwecken) ausgedruckt, sonst nicht.

```
EuklidDiv[a0_, b0_] :=
 Module[{a = a0, b = b0},
  While[Not[a * b == 0],
   If[verbose, Print[a, "   ", b]];
   If[a ≥ b, a = Mod[a, b], b = Mod[b, a]]];
  Return[a + b]]

verbose = True;
EuklidDiv[136, 60]
```

```
136   60

16   60

16   12

4   12

4
```

```
verbose = False;
EuklidDiv[136, 60]
```

```
4
```

■ *2. Argumentationsstrang:* Über einen rekursiven Ansatz

Rekursiv heißt in der Informatik soviel wie *selbstbezüglich* oder *auf sich selbst verweisend*; das Prinzip der Rekursion, eine der wichtigsten Grundideen der Mathematik und Informatik, wird in [Ziegenbalg, Abschnitt 4.2.2] noch ausführlich behandelt.

Aufgabe: Zeigen Sie, daß für $a \geqslant b$ stets gilt: GGT(a, b) = GGT(b, $a - b$).

Da b kleiner ist als a, ist es i.a. leichter, den größten gemeinsamen Teiler über die rechte Seite als über die linke Seite der letzten Gleichung zu ermitteln. Statt GGT(a, b) berechnen wir GGT(b, $a - b$). Dies ist die Grundidee für die folgende rekursiven Versionen des Euklidischen Algorithmus.

■ Subtraktionsform (rekursive Version)

```
EuklidSubRek[a_, b_] :=
 (Print[a, "   ", b];
  Which[
   a == 0, b,
   b == 0, a,
   a ≥ b, EuklidSubRek[a - b, b],
   a < b, EuklidSubRek[a, b - a]])
```

```
EuklidSubRek[136, 60]
```

```
136   60

76   60

16   60

16   44

16   28

16   12

4   12

4   8

4   4

0   4

4
```

```
(* TableForm[Trace[EuklidSubRek[136,60]]] *)
```

■ Divisionsform (rekursive Version)

```
EuklidDivRek[a_, b_] :=
 (Print[a, "   ", b];
  Which[a == 0, b,
   b == 0, a,
   a ≥ b, EuklidDivRek[Mod[a, b], b],
   a < b, EuklidDivRek[a, Mod[b, a]]])
```

```
EuklidDivRek[136, 60]
```

```
136   60

16   60

16   12

4   12

4   0

4
```

In Mathematica lassen sich sehr unterschiedliche Programmierstile realisieren (auf die Themen „Programmiersprachen, Programmierstile und Programmierparadigmen“ wird in [Ziegenbalg, Kapitel 8] noch ausführlicher eingegangen). Neben dem *imperativen* und dem *funktionalen Programmieren* hat in den letzten Jahren auch der Stil der *Logik-Programmierung* bzw. das *regelbasierte Programmieren* an Bedeutung gewonnen.

Der Vollständigkeit halber sei hier noch eine *regelbasierte* Form des Programms gegeben, wie sie in ähnlicher Form auch in der Programmiersprache *Prolog* ausgedrückt werden könnte. Man hat bei dieser Form des Programmierens keinen in sich geschlossenen Programmkörper mehr, sondern ein offenes (und erweiterbares) System von Regeln, die von dem jeweiligen Programmiersystem (sei es Prolog, sei es Mathematica) meist in der Form eines „Backtracking-Verfahrens“ (vgl. [Ziegenbalg, 4.3.2]) verarbeitet wird. Auch diese regelbasierte Version ist ersichtlich rekursiv.

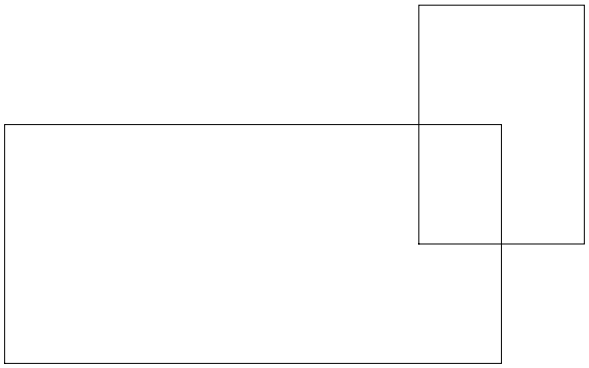
■ Regelbasierte Version

```
EuklidReg[a_, 0] := a;
EuklidReg[a_, b_] := EuklidReg[b, Mod[a, b]];

EuklidReg[136, 60]
```

```
4
```

■ Graphische Veranschaulichung

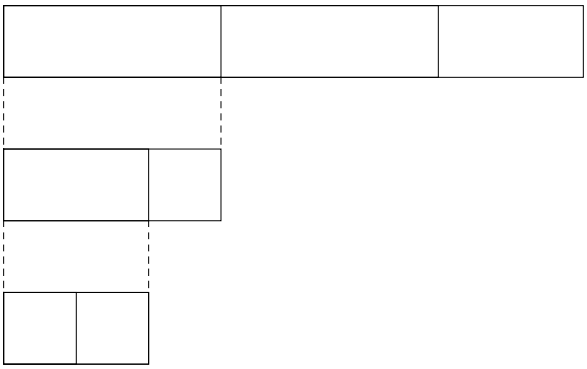
```
Box[xmin_, xmax_, ymin_, ymax_] :=
 Line[{
   {xmin, ymin},
   {xmax, ymin},
   {xmax, ymax},
   {xmin, ymax},
   {xmin, ymin}}]

Show[Graphics[{Box[-1, 2, 3, 4], Box[1.5, 2.5, 3.5, 4.5]}]]
```



```
- Graphics -
```

```
EuklidPlot[a0_, b0_] :=
 Module[{a = a0, b = b0, step = 0, L = {}, i = 0},
  (* Bedeutung der Hilfsvariablen:
     L: Liste, in der das zu erzeugende Graphik-Objekt aufgebaut wird;
     i: Laufvariable zur Ermittlung des jeweiligen Quotienten q;
     step: Hilfsvariable für die graphische Darstellung
   *)
  While[ Not[a * b == 0],
   If[i == 0, L = Append[L, Box[0, a, -10 * step, -10 * step - 5]]];
   If[i == 0 && step > 0,
    L = Append[L, {Dashing[{0.01, 0.01}],
       Line[{{0, -10 * (step - 1) - 5}, {0, -10 * step}}],
       Line[{{a, -10 * (step - 1) - 5}, {a, -10 * step}}]}]];
   i = i + 1;
   If[a ≥ b,
    (a = a - b; L = Append[L, Box[0, i * b, -10 * step, -10 * step - 5]]),
    ({a, b} = {b, a}; step = step + 1; i = 0)]];
  Return[L]]
```

```
Show[Graphics[EuklidPlot[128, 48]], PlotRange → All]
```



```
- Graphics -
```

```
EuklidDiv[128, 48]
```

```
16
```

■ Klassische Divisionsform

■ Implementierung

■ Demonstrationen

```
EuklidDivDemo[136, 60]
```

```
 136 =    2 *  60  +   16

  60 =    3 *  16  +   12

  16 =    1 *  12  +    4

  12 =    3 *   4  +    0
```

```
4
```

```
EuklidDivDemo[7618, 2536]
```

```
 7618 =     3 *  2536  +   10

 2536 =   253 *    10  +    6

   10 =     1 *     6  +    4

    6 =     1 *     4  +    2

    4 =     2 *     2  +    0
```

```
2
```

■ Ein Vergleich: Euklidischer Algorithmus - PFZ

```
PFZ[n0_] :=
 Module[{n = n0, k = 2, P = {}}, While[n > 1,
   If[Mod[n, k] == 0, P = Append[P, k]; n = Quotient[n, k], If[k == 2, k = 3, k = k + 2]]];
  Return[
   P]]
```

```
PFZ[12345]
```

```
{3, 5, 823}
```

```
GemeinsameTeiler[a_, b_] :=
 Module[{La = PFZ[a], Lb = PFZ[b], Lg = {}}, While[Not[La == {}],
   While[Not[Lb == {}] && First[Lb] < First[La], Lb = Drop[Lb, 1]]
    If[MemberQ[Lb, First[La]], Lg = Append[Lg, First[La]]; Lb = Drop[Lb, 1]];
   La = Drop[La, 1]];
  Return[Lg]]
```

```
GemeinsameTeiler[123456789, 987654321]
```

```
{3, 3}
```

```
verbose = False;
EuklidDiv[123456789, 987654321]
```

```
9
```

Probe:

```
GemeinsameTeiler[2*2*2*3*5*7*7*11, 2*2*3*3*3*7*13*17]
```

```
{2, 2, 3, 7}
```

```
GGTPFZ[a_, b_] := Apply[Times, GemeinsameTeiler[a, b]]
```

```
GGTPFZ[2*2*2*3*5*7*7*11, 2*2*3*3*3*7*13*17]
```

```
84
```

```
a = 2*2*3*5*7*Random[Integer, 10000000]
```

```
4112902500
```

```
b = 2*3*3*11*Random[Integer, 10000000]
```

```
1364886270
```

```
EuklidDiv[a, b] // Timing
```

```
{0. Second, 30}
```

```
GGTPFZ[a, b] // Timing
```

```
{2.605 Second, 30}
```

■ Eingebaute *Mathematica* Funktionen: FactorInteger und GCD

■ Der Berlekamp-Algorithmus

■ Hilfsprogramme