1. Größter gemeinsamer Teiler (*ggT*) zweier Zahlen[1]

Bruchrechnen ist schwierig. Schon beim Kürzen gibt es ein Problem: Wie findet man beispielsweise heraus, ob man 76/209 kürzen kann – wenn ja, welches ist der *größte gemeinsame Teiler* (oder Faktor) der beiden Zahlen? Wer in Kopfrechnen gut ist, findet auch ohne Taschenrechner $76 = 4 \cdot 19$ und $209 = 11 \cdot 19$, also lässt sich 76/209 kürzen zu 4/11. Da 4 und 11 teilerfremd sind, ist 19 der größte gemeinsame Teiler von 76 und 209.

Es muss sich nicht um Zähler und Nenner eines Bruches handeln, um vom *ggT* – so lautet die gängige Abkürzung für den größten gemeinsamen Teiler – zu reden. Jedes Paar natürlicher Zahlen *x* und *y* besitzt einen *ggT*. Allerdings kann dieser Eins sein, dann nämlich, wenn *x* und *y* teilerfremd sind. Der *ggT* ist das größte gemeinsame Element, das in den Teilermengen von *x* und *y* enthalten ist. Beispiel: Die Zahl 76 hat die Teiler 1, 2, 4, 19, 38 und 76, die Zahl 209 die Teiler 1, 11 und 19. Das größte Element, das in *beiden* Teilermengen vorkommt, ist 19. Im vorliegenden Fall ist es auch das einzige gemeinsame Element außer der Eins. Also gilt $ggT(209, 76) = 19$.

Der größte gemeinsame Teiler (*ggT*) zweier natürlicher Zahlen *x* und *y* ist das größte Element, das sowohl in der Teilermenge von *x* als auch in der von *y* enthalten ist. Man schreibt diese Zahl $ggT(x, y)$.

Zahlen *x* und *y*, deren Teilermengen außer der Eins kein gemeinsames Element enthalten, heißen *teilerfremd*. Dem entsprechend ist ihr *ggT* gleich Eins. Beispiel: Die Zahl 20 hat die Teiler 1, 2, 4, 5, 10 und 20, die Zahl 77 die Teiler 1, 7, 11 und 77. Nur die Eins kommt in beiden Teilermengen vor, daher $ggT(20, 77) = 1$.

Sind *x* und *y* teilerfremd, gilt $ggT(x, y) = 1$ und umgekehrt.

2. Euklidischer Algorithmus [1,2]

Wie findet man den *ggT* zweier Zahlen? Er ist das Produkt der gemeinsamen *Primfaktoren*, so lernt man in der Schule. Doch Primfaktoren sind, besonders bei großen Zahlen, schwierig zu bestimmen. Schon in den Werken *Euklids* wird ein schnelleres Verfahren angegeben – der später nach ihm benannte Algorithmus. Er beruht auf folgender Überlegung: Sind *x* und *y* zwei Zahlen, deren *ggT* bestimmt werden soll und für die gilt $x > y$, so bildet man die Differenz $x - y$ und hat damit eine dritte Zahl gefunden, die den gesuchten Teiler enthält. Statt den *ggT* von *x* und *y* zu suchen, bestimmt man daher den *ggT* der beiden Zahlen *y* und $x - y$. Dieses Verfahren setzt man solange fort, bis eine der Zahlen Null ist. Die von Null verschiedene Zahl des letzten Paares ist der gesuchte *ggT*. Beispiel:

Euklidischer Algorithmus zur Bestimmung von ggT(209,76):

ggT(209,76) = ggT(133,76) = ggT(76,57) = ggT(57,19) = ggT(38,19) = ggT(19,19) = ggT(19,0) = 19.

209-76 133-76 76-57 57-19 38-19 19-19

Das Beispiel zeigt, dass man das Verfahren abkürzen kann. Statt die Zahl 76 zweimal von 209 zu subtrahieren, kann man 209 durch 76 dividieren und mit dem Divisionsrest 57 weiterrechnen. Dasselbe gilt für das dreifache Subtrahieren der Zahl 19 von 57. Den Rest bei der Division von *x* durch *y* bezeichnet man mit *x mod y* („ *x* ***modulo*** *y* “). Damit lässt sich der Euklidische Algorithmus in unserem Beispiel abkürzen:

```
Verkürzter Euklidischer Algorithmus zur Bestimmung von ggT(209,76):
ggT(209,76) = ggT(76,57) = ggT(57,19) = ggT(19,0) = 19.
                    ↑             ↑           ↑
               209 mod 76     76 mod 57   57 mod 19
```

Dieses Divisionsverfahren ist immer anwendbar. Seine endliche Laufzeit und Korrektheit ist einfach zu beweisen[2]. Wir formulieren den Euklidischen Algorithmus daher wie folgt:

Euklidischer Algorithmus:

> Sei (x, y) ein Zahlenpaar mit $x < y$, so benutze $ggT(x, y) = ggT(x, y \bmod x)$, um ein Paar mit kleineren Zahlenwerten zu erhalten. Setze dieses Verfahren fort, bis eine der Zahlen des Paares Null ist, die andere Zahl ist dann der gesuchte *ggT*.

3. **Algorithmischer Aufwand (Schrittzahl) des Euklidischen Algorithmus**

Ein guter Algorithmus ist ein schneller Algorithmus. Schnell ist ein Algorithmus, wenn er nach wenigen Iterationsschritten die Lösung liefert. Wir interessieren uns daher für die Anzahl der Divisionsschritte, die der Euklidische Algorithmus benötigt, um den *ggT* zu bestimmen [2,3,4]. Bei unserem Bruch 76/209 sind es drei Divisionen, denn $ggT(209, 76) = ggT(76, 57) = ggT(57, 19) = ggT(19, 0) = 19$. Der Bruch 14/28 dagegen erfordert nur einen einzigen Schritt: es ist $ggT(28, 14) = ggT(14, 0) = 14$. Wir fragen daher:

(1) Gegeben eine Zahl y (zum Beispiel der Nenner eines Bruchs). Betrachte dann alle natürlichen Zahlen $1 \le x < y$ (das sind zum Beispiel die Zähler des echten Bruches x/y). Wie viele Schritte erfordert der Euklidische Algorithmus zur Bestimmung des $ggT(x, y)$ für jedes x?
(2) Für welches x ist (bei gegebenem y) die Schrittzahl maximal und wie groß ist diese Schrittzahl?

Da im Folgenden oft von Schrittzahl und maximaler Schrittzahl (des Euklidischen Algorithmus) die Rede sein wird, führen wir für beide Größen Formelzeichen ein[5]:

> Die *Anzahl der* (*Divisions–*) *Schritte* des Euklidischen Algorithmus zur Bestimmung des *ggT* der Zahlen x und y werde mit $n(x, y)$ bezeichnet. Die *maximale Anzahl* von Schritten, die für gegebenes y beim Durchlaufen aller Zahlen x mit $1 \le x < y$ benötigt wird, sei $m(y)$ genannt:
>
> $n(x, y)$ = Schrittzahl des Euklidischen Algorithmus zur Bestimmung des $ggT(x, y)$,
> $m(y) = max\ \{n(x, y) \mid 1 \le x < y\}$.

Beispiel: Für einen Bruch mit dem Nenner $y = 7$ sind die möglichen Zähler $x = 1, 2, 3, 4, 5$ und 6. Also bestimmen wir gemäß Frage (1) die Schrittzahl $n(x, y)$ des Euklidischen Algorithmus für die Zahlenpaare (1, 7), (2, 7), (3, 7), (4, 7), (5, 7) und (6, 7). Tabelle 1 listet diese Schritte auf. Beim Zahlenpaar (1, 7) ist es ein Schritt, bei den Paaren (2, 7), (3, 7) und 6, 7) sind es zwei, und bei den Paaren (4, 7) und (5, 7) drei Schritte. Damit ist Frage (2) beantwortet: die Maximalzahl von Schritten für $y = 7$ ist drei – sie wird für $x = 4$ und 5 benötigt. Also $m(7) = 3$.

Tabelle 1 Euklidischer Algorithmus für (1, 7), (2, 7), ... , (6, 7)

Zahlenpaar (x,y)	Zwischenstationen des Algorithmus (je Pfeil ein Schritt)	Anzahl der Schritte n(x,y)
(1,7)	(1,7) → (1,0)	1
(2,7)	(2,7) → (1,2) → (1,0)	2
(3,7)	(3,7) → (1,3) → (1,0)	2
(4,7)	(4,7) → (3,4) → (1,3) → (1,0)	3
(5,7)	(5,7) → (2,5) → (1,5) → (1,0)	3
(6,7)	(6,7) → (1,6) → (1,0)	2

Intuitiv ist klar, dass die maximale Schrittzahl m beim Euklidischen Algorithmus mit wachsendem y (beim Bruch, heißt das, mit wachsendem Nenner) ansteigt. Unsere nächste Frage ist daher: lässt sich diese Maximalzahl berechnen oder zumindest abschätzen? Der französische Mathematiker *Jacques Philippe Marie Binet* (1786 – 1856) bewies 1841, dass die Schrittzahl beim Euklidischen Algorithmus mit den *Fibonacci*-Zahlen $F(n)$ in folgender Weise verknüpft ist[6]:

Satz von *Binet*

Ist die Zahl y des Zahlenpaares $x, y \in \boldsymbol{N}$ kleiner als die n +1-te *Fibonaccizahl* $F(n + 1)$, dann ergibt der Euklidische Algorithmus den größten gemeinsamen Teiler der Zahlen x und y in höchstens $n - 1$ Schritten.

Diese mathematische Einsicht ist noch nicht die Lösung unseres Problems. Sie ist trotzdem von Nutzen, denn das Wachstum der Fibonacci-Zahlen ist sehr genau bekannt. Die Fibonacci-Zahlen wachsen exponentiell an mit φ^n, wobei $\varphi = (1 + \sqrt{5})/2 = 1{,}61803398...$ die Zahl des Goldenen Schnitts ist. Man zeigt (beispielsweise durch Induktion), dass gilt

(1) $\quad F(n + 1) > 0{,}43769 \cdot \varphi^{n+1}$ für $n \geq 2$.

Nach dem Satz von *Binet* ist bei n Iterationsschritten $y \geq F(n + 1)$, also auch

(2) $\quad y \geq 0{,}43769 \cdot \varphi^{n+1}$.

Logarithmiert man beide Seiten dieser Ungleichung, erhält man

(3) $\quad {}^{10}\log y > {}^{10}\log 0{,}43769 + (n + 1) \cdot {}^{10}\log \varphi > -0{,}35884 + 0{,}20898 \cdot (n + 1)$.

Daraus folgt

(4) $\quad n < 0{,}718 + 4{,}785 \cdot {}^{10}\log y$.

Also gilt für unsere Höchstzahl von Schritten $m(y)$ die Abschätzung:

Seien $x, y \in \boldsymbol{N}$ mit $y \geq 2$. Dann ist die Anzahl der Iterationsschritte im Euklidischen Algorithmus für $ggT(x, y)$ kleiner als $0{,}718 + 4{,}785 \cdot {}^{10}\log y$. Es gilt also $m(y) < 0{,}718 + 4{,}785 \cdot {}^{10}\log y$.

Wir überprüfen diese Aussage für unser Beispiel $y = 209$: die Abschätzung liefert $m(209) < 0{,}718 + 4{,}785 \cdot {}^{10}\log 209 = 11{,}8199...$. Tatsächlich ist $m(209) = 10$, und zwar für die Paare (128, 209) und (129, 209). Das heißt, die Abschätzung ist in Ordnung und scheint auch nicht zu grob zu sein. Abbildung 1 zeigt, wie sie sich im Intervall $1 \leq y \leq 65536$ gegenüber den tatsächlichen Werten $m(y)$ verhält. Aufgetragen ist dort $m(y)$ als Funktion von y (schwarze Kreise) und der Graph der Schätzfunktion $m^*(y) = 0{,}718 + 4{,}785 \cdot {}^{10}\log y$ (rote Gerade).

Aus Gründen der Übersichtlichkeit wurde in Abb. 1 die Höchstschrittzahl $m(y)$ nur für die Zweierpotenzen $y = 2, 4, 8, 16, 32, ...$ bis 65536 eingetragen. Da die horizontale Achse logarithmisch geteilt ist, haben die Punkte in horizontaler Richtung den gleichen Abstand. Aus demselben Grund ist der Graph der Abschätzung $m^*(y) = 0{,}718 + 4{,}785 \cdot {}^{10}\log y$ eine Gerade.

Gleichung (4) wurde, wie schon erwähnt, dem Vorlesungsskript [3]) entnommenen. Es gibt natürlich weitere Abschätzungen für $m(y)$. Eine für die Praxis geeignete wurde beispielsweise 1844 von dem französischen Mathematiker *Gabriel Lamé* (1795 – 1870) formuliert[2] :

Ist $x \leq y$, benötigt der Euklidische Algorithmus zur Berechnung von $ggT(x, y)$ höchstens $5k$ Schritte, sofern y genau k Stellen im Dezimalsystem hat.

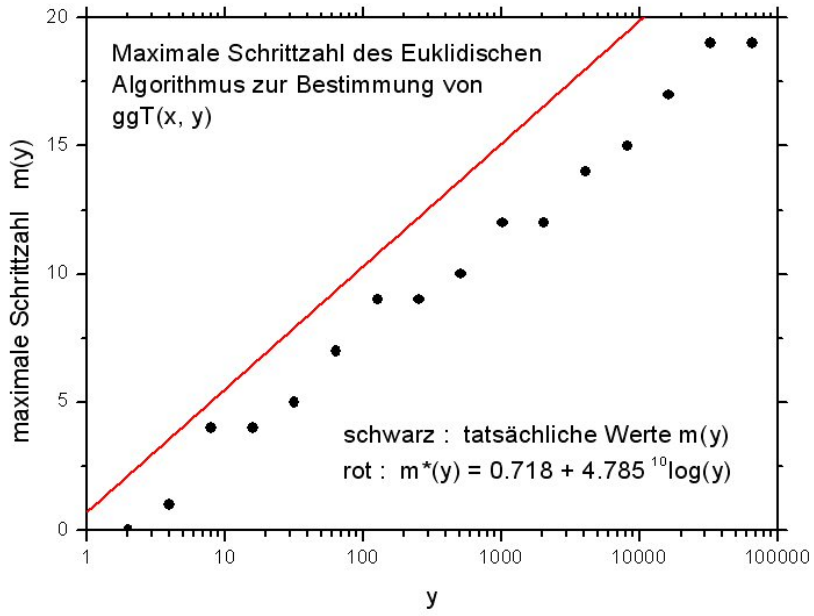

Abb. 1 Höchstzahl m(y) der Iterationsschritte des Euklidischen Algorithmus zur Bestimmung des ggT(x,y), aufgetragen als Funktion von y (schwarze Kreise).

m(y) ist die maximale Anzahl von Schritten, die für gegebenes y beim Durchlaufen aller Zahlen x mit 1 ≤ x < y benötigt wird.

Rote Gerade: Abschätzung nach Gl.(4), siehe Text. Beachte die logarithmische Teilung der horizontalen Achse.

Die schwarzen Kreise in Abb. 1 zeigen einen im Wesentlichen logarithmischen Anstieg. Aber es gibt eine gewisse Streuung um die Gerade herum, die einem strengen Logarithmengesetz entsprechen würde. Wir gehen diesem Sachverhalt nach und tragen nochmals $m(y)$ als Funktion von y auf, dieses Mal für *alle* Zahlen y – allerdings nur solche aus dem Intervall $1 \le y \le 280$.

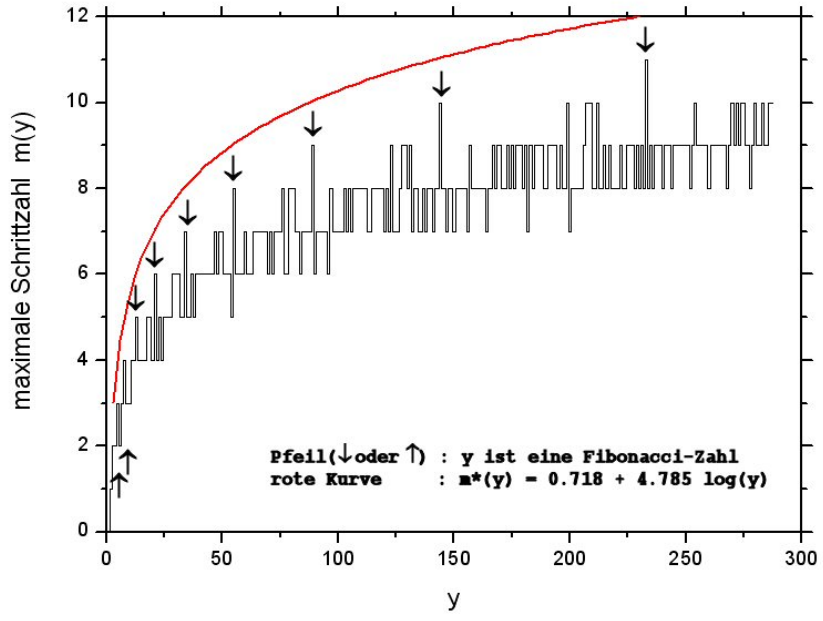

Abb. 2 Höchstzahl m(y) der Iterationsschritte des Euklidischen Algorithmus zur Bestimmung des ggT(x,y). Durchgezogene Treppenlinie: m(y), wie in Abb. 1, aufgetragen als Funktion von y für y < 280, jedoch mit linearer Teilung der horizontalen Achse. Rote Kurve: m*(y) nach Gl.(4).

Punkte, die um etwa eine Einheit über den Werten ihrer Umgebung liegen, sind durch Pfeile gekennzeichnet. Das zugehörige y ist eine Fibonacci-Zahl. In diesem Fall ist auch x eine Fibonacci-Zahl. Beispiel: die maximale Schrittzahl m = 10 für y = 144 ergibt sich für x = 89, den Vorgänger von 144 in der Fibonaccifolge.

Das Ergebnis zeigt Abb. 2. Die horizontale Achse ist hier linear geteilt, dadurch erkennt man den logarithmischen Anstieg von $m(y)$ und $m^*(y)$ auf den ersten Blick. Der Graph von $m(y)$ ist hier[7] als Treppenfunktion dargestellt. Dabei fällt auf, dass es eine Reihe von „Ausreißern" gibt – von Punkten, die deutlich höher liegen als es dem generellen Verlauf von $m(y)$ entspricht. Ihre Position wurde in Abb. 2 durch Pfeile gekennzeichnet. Sie gehören zu den Zahlen y = 3, 5, 8, 13, 21, 34, 55, 89, 144 und 233. Man erkennt unschwer: dies sind genau die Zahlen der *Fibonacci*-Folge. Nicht aus der Abbildung ablesbar ist, dass auch die Partnerzahl x eine Fibonacci-Zahl ist – sie ist der Fibonacci-Vorgänger von y. Dies ist, wie beispielsweise in [2]) gezeigt wird, tatsächlich der für die Laufzeit des Algorithmus ungünstigste Fall:

Der Euklidische Algorithmus zur Bestimmung des *ggT* von x und y benötigt, relativ zur Größe von x und y, eine maximale Anzahl von Schritten, wenn x und y benachbarte *Fibonacci*-Zahlen sind.

Beispiel: Für die Fibonacci-Zahl y = 233 ist die maximale Schrittzahl m = 11 (Abb. 2). Sie tritt auf beim Paar (144, 233) – das heißt, bei 233 und ihrem Fibonacci-Vorgänger 144.

4. Computergrafik der Schrittzahl

Wir haben festgestellt: Der Euklidische Algorithmus erfordert eine Höchstzahl von Schritten, wenn x und y aufeinander folgende Fibonacci-Zahlen sind. Diese sind unter den natürlichen Zahlen jedoch nicht gerade häufig anzutreffen. Denn die Folge der Fibonacci-Zahlen wächst exponentiell (s. oben), so dass die Lücken zwischen ihnen schnell groß werden. Die Zahlen in diesen Lücken – wir nennen sie *normale* Zahlen, sollen jetzt gepaart und dem *ggT*-Algorithmus unterworfen werden. Unsere Fragen sind: Welche Schrittzahlen kommen bei Paaren dieser Zahlen vor? Wie sind sie verteilt? Wie sehen die Paare *normaler* Zahlen mit maximaler Schrittzahl aus?

Die Schrittzahl $n(x, y)$ liefert uns ein einfaches Computerprogramm[8], das den Euklidischen Algorithmus iterativ ausführt und dabei die Anzahl der Schleifendurchläufe zählt. Das Ergebnis zeigt die Grafik Abb. 3. Jedem Paar (x, y) ist ein Karo im dort dargestellten Koordinatensystem zugeordnet. Beachte bitte: entgegen der üblichen Norm wird y in horizontaler Richtung und x in vertikaler Richtung gezählt. Die Farbe der Karos entspricht der Schrittzahl, *bezogen* auf deren Höchstzahl $m(y)$. Paare mit maximaler Schrittzahl sind *rot* gefärbt, solche mit der um eins, zwei oder drei gegenüber der Höchstzahl verminderten Schrittzahl *grün*, *cyan* bzw. *blau*. Alle Karos, die Paaren mit noch kleinerer Schrittzahl entsprechen, erscheinen *schwarz* – bis auf diejenigen, die nur einen einzigen Schritt erfordern. Sie wurden *weiß* gefärbt.

Wir wenden uns zuerst den Paaren *maximaler* Schrittzahl zu (*rote* Karos in Abb. 3). Dazu gehören natürlich die schon erwähnten Paare aufeinander folgender Fibonacci-Zahlen. Die Spalten (y-Koordinate) der Fibonacci-Zahlen sind am *unteren* Rand des Koordinatensystems durch die zugehörigen Zahlen 2, 3, 5, 8, 13, 21, 34 und 55 gekennzeichnet. In diesen Spalten gibt es genau ein rotes Karo. Es entspricht dem x, das gleich dem Fibonacci-Vorgänger von y ist. Die entsprechende Zeile ist am *linken* Rand des Koordinatensytems durch die zugehörige Fibonacci-Zahl hervorgehoben. Beispiel: Für y = 21 ist die Schrittzahl maximal beim x-Wert 13, so dass Karo (13, 21) das einzige in dieser Spalte rot gefärbte ist.

Bei *normalen* Zahlen gibt es zu jedem y meist mehrere x, die ein Paar maximaler Schrittzahl bilden. Für y = 7 zum Beispiel sind das die Karos, die den Paaren (x, y) = (4, 7) und (5, 7) entsprechen (vergleiche Tabelle 1).

Abbildung 3 zeigt, dass die meisten *roten* Karos, die Paaren *normaler* Zahlen entsprechen, in der Nähe oder auf der Geraden $x = 0{,}618\,y$ liegen. (Hier liegen allerdings auch die roten Karos der

gerade erwähnten Fibonacci-Paare.) Der Zahlenwert 0,618 dürfte keine Überraschung sein: Auch hier spielt offenbar die Zahl φ eine Rolle – die Geradensteigung 0,618 ist ein Näherungswert für das Verhältnis $\varphi - 1$ der kleineren zur größeren Strecke beim Goldenen Schnitt.

Man muss jedoch zugeben: es gibt auch Karos von Paaren maximaler Schrittzahl, die deutlich oberhalb dieser Geraden liegen. Unterhalb der Geraden $x = 0{,}618\,y$ sind sie weniger vertreten, eine untere Grenze scheint $x = y/2$ zu sein. Insgesamt stellen wir fest: Der Euklidische Algorithmus benötigt, für gegebenes y und $1 \le x < y$, eine maximale Anzahl von Schritten, wenn für den Quotient x/y gilt $1/2 < x/y < 1$. In vielen Fällen gilt näherungsweise $x/y = \varphi - 1 = 0{,}618\ldots$

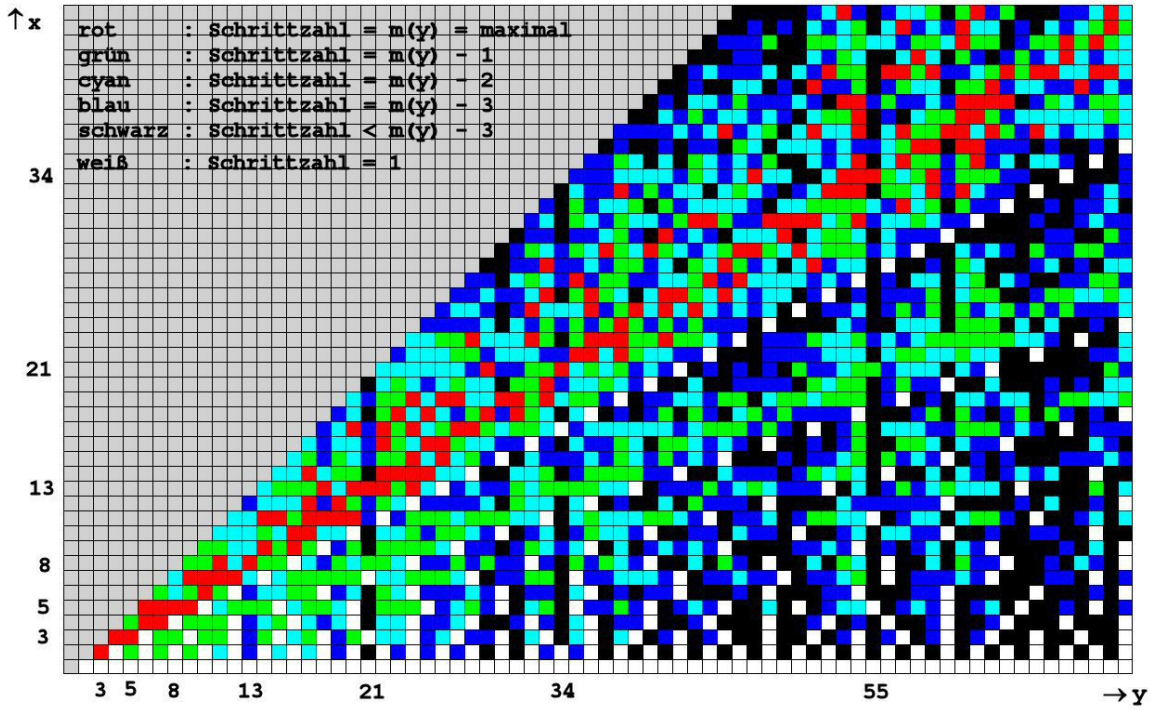

Abb. 3 Schrittzahl, die zur Bestimmung des ggT(x,y) nach Euklid benötigt wird, als Funktion von x und y. Beachte: y ist in horizontaler Richtung aufgetragen, x in vertikaler Richtung. Jedes Karo repräsentiert ein Zahlenpaar (x,y) und ist entsprechend der relativen Schrittzahl gefärbt:

Karos von Paaren mit maximaler Schrittzahl sind rot gefärbt, solche mit der um eins, zwei oder drei gegenüber der Höchstzahl verminderten Schrittzahl grün, cyan bzw. blau. Alle Karos, die Paaren mit noch kleinerer Schrittzahl entsprechen, erscheinen schwarz - bis auf diejenigen, die nur einen einzigen Schritt erfordern, sie wurden weiß gefärbt. Farbzuordnung siehe auch Legende im Bild links oben. An beiden Achsen ist die Position der Fibonacci-Zahlen > 2 eingetragen.

Viele Paare maximaler Schrittzahl (rote Karos) liegen in etwa in der Nähe der Geraden x = 0,618 y (0,618 ist das Verhältnis der kleineren zur größeren Strecke beim Goldenen Schnitt). Es gibt jedoch Paare deutlich oberhalb dieser Geraden. Auch die grünen Karos (Schrittzahl um eins kleiner als die maximale Schrittzahl) gruppieren sich in der Nähe von Geraden, zum Beispiel x = 3y/8.

Die durch die weißen Karos angedeuteten Geraden entsprechen den Gleichungen x = y/2, x = y/3, usw. In diesen Fällen ist x die Hälfte, ein Drittel, usw. von y, so dass der Algorithmus nur einen Schritt benötigt.

Wir wenden uns jetzt den Zahlenpaaren zu, bei denen, für gegebenes y, die Schrittzahl $n(x, y)$ kleiner ist als $m(y)$. Ist sie um *eins kleiner* als die Maximalzahl, das heißt, $n(x, y) = m(y) - 1$, erhielt das entsprechende Karo die Farbe *grün*. Paare (x, y) mit um *zwei* oder *drei* gegenüber der

Höchstzahl verminderter Schrittzahl sind durch die Farben *cyan* bzw. *blau* gekennzeichnet. Auch die grünen Karos liegen andeutungsweise auf Geraden durch den Nullpunkt, eine davon ist $x = 3y/8$. Dass hier wieder einmal Fibonacci-Zahlen eine Rolle spielen (3 und 8 gehören zu diesen), ist möglicherweise Zufall.

Die *weiß* gefärbten Karos entsprechen Zahlenpaaren, für die der Euklidische Algorithmus nur *einen* einzigen Schritt benötigt, also $n(x, y) = 1$. Das sind zum Beispiel solche mit $x = y/2$ oder $x = y/3$, usw. Die entsprechenden Geraden sind in Abb. 3 deutlich sichtbar.

Schwarz gefärbt wurden die Karos, die Paaren mit Schrittzahlen kleiner als $m(y) - 3$ entsprechen. Es fällt auf, dass sie u. a. in den *Spalten* gehäuft auftreten, für die y eine Fibonacci-Zahl ist (deutlich zu sehen für $y = 21$, 34 und 55).

Wir schließen aus alledem: Auch bei *Euklid* mischt *Fibonacci* mit.

Anmerkungen und Literatur

[1] Kapitel 2.4 des Buchs *A. Bartholomé, J. Rung* und *H. Kern*: „Zahlentheorie für Einsteiger“ (Braunschweig/Wiesbaden, Vieweg, 1995) gibt einen guten Überblick über das Thema *ggT* und *Euklidischer Algorithmus*. Das Buch ist, wie der Untertitel sagt, „Eine Einführung für Schüler, Lehrer, Studierende und andere Interessierte“. Es enthält viele Aufgaben und (in der Programmiersprache *Pascal* formulierten) Algorithmen und lädt so zum Nacharbeiten an.

[2] Eine Einführung auf gleichem Niveau wie [1]) mit fachdidaktischen Anregungen ist der Artikel von *Th. Holm* und *W. Willems*: „Der Euklidische Algorithmus – Warum nicht in der Schule?“. Enthält die im vorliegenden Artikel erwähnten, aber nicht ausgeführten Beweise.
http://fma2.math.uni-magdeburg.de/~willems/papers/ggt1.ps

[3] Das Problem „Schrittzahl des Euklidischen Algorithmus“ wird in der Vorlesung von *K. Pommerening*: *Kryptologie* ausführlich behandelt (Kapitel I.8.1 und I.8.2). Hochschulniveau.
http://www.staff.uni-mainz.de/pommeren/Kryptologie/Klassisch

[4] *M. Bauer* behandelt das Thema „Schrittzahl ...“ auf Schulniveau in einem amüsanten Artikel mit dem Titel „Die dümmsten Brüche und die blödesten Nenner“ (mathe-plus 2 (1) 1985, S. 12, Bibiliographisches Institut, Mannheim).

[5] *Bauer* definiert in [4]) die „*Dummheit* “ eines Bruches x/y als

$$dumm\left(\frac{x}{y}\right) = \frac{n(x,y)}{{}^{\varphi}\log(y)},$$

also gleich der Schrittzahl $n(x,y)$, dividiert durch den Logarithmus (zur Basis φ) des Nenners. Das Maximum der Dummheit eines Nenners, genommen über alle möglichen Zähler, heißt bei ihm „*Blödheit* “, also

$$blöd(y) = \max\left\{ dumm(x/y) \,\middle|\, 1 \le x < y \right\}.$$

[6] Die nachfolgende Abschätzung ist [3]) entnommen (Kapitel I.8.2 – Analyse des Euklidischen Algorithmus).

[7] Dividiert man $m(y)$ durch den Logarithmus (zur Basis φ) von y, erhält man den Wert der Funktion *blöd* (y), die in [4]) Abb. 1 dargestellt ist.

[8] Die nachfolgenden *Java*-Methoden zur Berechnung der Schrittzahl $n(x, y)$ und der (für konstantes y) maximalen Schrittzahl $m(y)$ sind selbsterklärend.

```
public int schrittzahl(long x, long y){
        int n = 0;
        if (y > x) { long m = x; x = y; y = m; }
        while (y > 0) {
                long r = x % y;
                x = y;
                y = r;
                n++;
        }
        return n;
}

public int maxSchrittzahl(long y) {
        int nMax = 0;
        for (long x = 1; x < y - 1; x++) {
                int n = schrittzahl(x, y);
                if (n > nMax) nMax = n;
        }
        return nMax;
}
```