Der Euklidische Algorithmus

- Einführung in Grundprobleme der Informatik aus der Sicht des Mathematikers -

Klaus POMMERENING, Mainz

INHALT

EINLEITUNG

Der Euklidische Algorithmus ist ein uraltes Thema; trotzdem gibt es, wie ich meine, noch einiges Aktuelle dazu zu sagen. Im Grunde dient er mir aber nur als Beispiel; die eigentlichen Ziele meines Vortrages sind

- zu zeigen, wie wichtige Grundprobleme der Informatik zu sinnvollen mathematischen Fragen führen,
- Anregungen zu Facharbeiten zu geben, auch einige geeignete Probleme für "Jugend forscht" oder ähnliche Wettbewerbe vorzustellen,
- das Hintergrundwissen der Lehrer zu erweitern, das stets etwas mehr als nur gerade den Schulstoff umfassen sollte.

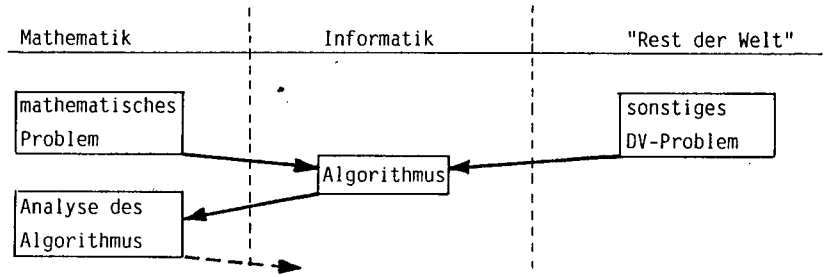
Direkt als Unterrichtsstoff ist allerdings nur ein kleiner Teil des Vortrags geeignet, und auch diesen stelle ich nur in knapper, noch nicht schulgemäßer Form dar. Vielleicht ist trotzdem die eine oder andere Anregung für den Unterricht dabei.

Zum Thema: Ein zentraler Begriff sowohl in der Mathematik als auch in der Informatik ist der Algorithmus.

Vom Mittelalter bis weit ins 19. Jahrhundert war die Mathematik algorithmisch orientiert; ihr Ziel war die Erfindung expliziter Rechenverfahren oder gar Formeln zur Lösung gegebener Aufgaben. Im späten 19. und 20. Jahrhundert änderte sich diese Einstellung gründlich; das abstrakte strukturelle Denken nahm überhand, und explizite Rechenverfahren wurden von der Mehrzahl der "reinen" Mathematiker als uninteressant betrachtet.

In den letzten Jahren hat aber eine starke Gegenbewegung eingesetzt. Durch die Entwicklung immer leistungsfähigerer Computer ist das Interesse der Mathematiker an Algorithmen wieder erwacht. Zwar hat sich die Computerwissenschaft (in Deutschland unter dem Namen Informatik) inzwischen längst als eigenständiges Fach etabliert; die Beziehungen zur Mathematik sind aber doch noch recht eng, und zwar von beiden Seiten aus gesehen. Einerseits können die Mathematiker heute Probleme, die auf ungeheure Rechnungen führen, dem Computer übertragen, andererseits benötigen die Informatiker bei der Entwicklung und Untersuchung von Algorithmen mathematische Methoden: Das Problem, für eine gegebene Aufgabe den optimalen Algorithmus zu finden (und dessen Optimalität auch nachzuweisen), führt oft auf tiefliegende Probleme der Mathematik, und zwar fast aller Fachgebiete bis hin zur Zahlentheorie und

der Algebraischen Geometrie. Zu deren Lösung kann man sich eventuell wieder vom Computer inspirieren lassen.



Ich werde diesen Wechselprozeß am Beispiel erörtern. Das Musterbeispiel ist der Euklidische Algorithmus, einer der ältesten und wichtigsten Algorithmen der Mathematik und wohl der älteste, der ein Iterationsverfahren enthält. Für die Informatik ist er interessant, weil sich hier die Anforderungen, die ein Algorithmus erfüllen muß, leicht diskutieren lassen:

a) Korrektheit:

- Wird der Algorithmus überhaupt irgendwann beendet (etwa: eine Programmierschleife wieder verlassen)?
- Bringt er dann das richtige Ergebnis?

Hierbei hat der Euklidische Algorithmus gegenüber Algorithmen aus der Analysis (wie etwa dem Newton-Verfahren) den didaktischen Vorteil, daß man mit ganzen Zahlen und ohne Konvergenzbetrachtungen auskommt.

b) Effizienz (Betriebsmittelanforderungen, "Kosten"):

- Wird der Algorithmus in angemessener Zeit beendet?
- Kommt er mit dem vorhandenen Speicherplatz aus? (Für die nächste Computergeneration: Kommt er mit den vorhandenen Parallelprozessoren aus?)
- Ist er mit angemessenem Aufwand zu programmieren und auszutesten?
- Gibt es einen besseren Algorithmus für das gegebene Problem? (Weniger Zeit, Speicherbedarf, Programmieraufwand?)

Darüberhinaus bietet der Euklidische Algorithmus eine Programmieraufgabe, die mathematisch sinnvoll ist, im Gegensatz zu vielen (wenn auch wichtigen) sonstigen DV-Problemen ohne mathematischen Gehalt, wie etwa Textverarbeitung oder Computerspiele.

Für die Mathematik ist der Euklidische Algorithmus wichtig, weil viele andere Verfahren auf ihm aufbauen:

a) Zahlentheorie
   - Bruchrechnen,
   - Kettenbruchentwicklung,
   - Lösung diophantischer Gleichungen,
   - Kongruenzrechnung (etwa Division modulo einer Primzahl, chinesisches Restproblem),
   - Nachweis der Teilerfremdheit, etwa bei einigen Verfahren zur Erzeugung von Zufallszahlen,
   - Verschlüsselungs-Verfahren.

b) "Konkrete" Algebra (Rechnen mit Polynomen - hier bracht man natürlich auch den Euklidischen Algorithmus für Polynome)
   - Rechnen mit rationalen Funktionen,
   - Befreiung von mehrfachen Nullstellen,
   - Sturmsche Ketten (zur Eingrenzung und Approximation von Nullstellen),
   - Decodierung einiger Codes.

c) "Abstrakte" Algebra und algebraische Zahlentheorie
   - Theorie der euklidischen Ringe, Hauptidealringe, ZPE-Ringe,
   - Primfaktor-Zerlegung in Zahlkörpern, Klassenzahlfragen,
   - Theorie des optimalen Euklidischen Algorithmus.

§1. DER KLASSISCHE EUKLIDISCHE ALGORITHMUS

(1.1) Nach so viel Reklame will ich endlich konkret werden. Sie alle kennen ihn natürlich, den Euklidischen Algorithmus, der den größten gemeinsamen Teiler (ggT) zweier natürlicher Zahlen $a \geq b \geq 1$ liefert. Er wird gewöhnlich als Folge von Divisionen mit Rest aufgeschrieben.

$$
\begin{aligned}
r_0 &= a \\
r_1 &= b \\
r_0 &= q_1 r_1 + r_2 \\
&\vdots \\
r_{i-1} &= q_i r_i + r_{i+1} \\
&\vdots \\
r_{n-1} &= q_n r_n
\end{aligned}
$$

wobei r_{i+1} der eindeutig bestimmte Divisionsrest mit $0 \leq r_{i+1} < r_i$ ist und $r_n \neq 0$, $r_{n+1} = 0$.
Dann ist $r_n = \mathrm{ggT}(a,b)$.

Den Beweis will ich noch einen Moment aufschieben. Da diese Schreibweise als Divisionskette durch die Verwendung von Indizes didaktisch unbefriedigend ist, will ich das Verfahren zunächst an einem Beispiel vorführen (ein Beispiel sagt mehr als tausend Worte!):

Beispiel zum Euklidischen Algorithmus: a = 272526, b = 32574.

Version 1: Klassische Kettendivision.

```
272526 =  8·32574 + 11934
 32574 =  2·11934 +  8706
 11934 =  1· 8706 +  3228
  8706 =  2· 3228 +  2250
  3228 =  1· 2250 +   978
  2250 =  2·  978 +   294
   978 =  3·  294 +    96
   294 =  3·   96 +     6
    96 = 16·    6
```

Ergebnis: ggT(a,b) = 6.
9 Iterationsschritte.

Der Vorteil, den der Euklidische Algorithmus bietet, wird sofort klar, wenn man versucht, den größten gemeinsamen Teiler aus der Primfaktor-Zerlegung zu finden. Diese ist nämlich mit ungleich mehr Aufwand verbunden, besonders für Zahlen mit großen Primteilern. Probieren Sie das ruhig einmal am obigen

Beispiel aus!

Die Frage nach der Beendigung des Euklidischen Algorithmus ist schnell beantwortet: Da $r_1 > r_2 > \ldots > r_i \geq 0$ für alle i, gilt $i \leq r_1 = b$, wenn r_i noch > 0 ist. Insbesondere wird die Abbruchbedingung $r_{n+1} = 0$ nach spätestens $n \leq b$ Iterationsschritten (also Divisionen) erreicht.

Aber warum ist das Ergebnis richtig? Das wird im nächsten Abschnitt diskutiert.

(1.2) Ich will die Definition des größten gemeinsamen Teilers zweier ganzer Zahlen nicht wiederholen; sie kam ja schon im Vortrag von Herrn Hofmeister vor. Jedenfalls hat man, wenn man der Einfachheit halber ggT(0,0)=0 setzt, die Funktion

$$ggT: \mathbb{Z} \times \mathbb{Z} \to \mathbb{N}_0 = \{0,1,2,\ldots\}$$

mit den folgenden Eigenschaften:

HILFSSATZ 1 (Eigenschaften des ggT). Für beliebige $a,b,c,q \in \mathbb{Z}$ gilt:

(i) $ggT(a,b) = ggT(b,a)$.
(ii) $ggT(a,-b) = ggT(a,b)$.
(iii) $ggT(a,0) = |a|$.
(iv) $ggT(a-qb,b) = ggT(a,b)$.
(v) $ggT(ca,cb) = |c| \cdot ggT(a,b)$.
(vi) $ggT(a,b) = ggT(\frac{a}{c},b)$, wenn $c|a$ und $ggT(c,b) = 1$.

Den Beweis will ich übergehen, jedoch dazu bemerken:

1.) Weder für die Definition noch für den Hilfssatz benötigt man die Primfaktorzerlegung. Diese hilft aber vielleicht zum Verständnis.
2.) Für den Euklidischen Algorithmus werden zunächst nur (i)-(iv) benötigt. Bei (iv) beweist man am einfachsten, daß sogar die Mengen aller gemeinsamen Teiler übereinstimmen.
3.) Für (vi) benötigt man das "Lemma von Euklid": $d|ab$, $ggT(d,b) = 1 \Rightarrow d|a$.
4.) Die Eigenschaft (iv) ist das eigentliche Geheimnis des Euklidischen Algorithmus. Man kann es in Worten so formulieren: Der ggT ändert sich nicht, wenn man von einer der beiden Zahlen ein Vielfaches der anderen subtrahiert. Bei der Kettendivision in (1.1) wurde gerade immer von der

größeren Zahl das größtmögliche Vielfache der kleineren subtrahiert, das noch einen Rest ≥ 0 übrig läßt. Also gilt:

$$ggT(a,b) = ggT(r_0,r_1) = ggT(r_1,r_2) = \ldots = ggT(r_{n-1},r_n) = ggT(r_n,0) = r_n;$$

dabei wurden auch (i) und (ii) verwendet. Damit ist die Korrektheit des Euklidischen Algorithmus bewiesen:

SATZ 1. Der Euklidische Algorithmus liefert nach endlich vielen Schritten den ggT zweier natürlicher Zahlen a und b.

Die Voraussetzung $a \geq b$ ist dabei überflüssig; die letzte Beschreibung des Algorithmus umfaßt sogar die Fälle $b = 0$ oder $b > a \geq 0$. Man kann auch beliebige $a,b \in \mathbb{Z}$ zulassen, wenn man (ii) verwendet und $r_1 = |b|$ setzt; wer gerne r_0 positiv hätte, setzt außerdem noch $r_0 = |a|$.

Für unser Beispiel sieht der Algorithmus in verkürzter Schreibweise jetzt so aus:

$ggT(272526,32574) = ggT(32574,11934) = ggT(11934,8706)$
$= ggT(8706,3228) = ggT(3228,2250) = ggT(2250,978)$
$= ggT(978,294) = ggT(294,96) = ggT(96,6) = ggT(6,0) = 6.$

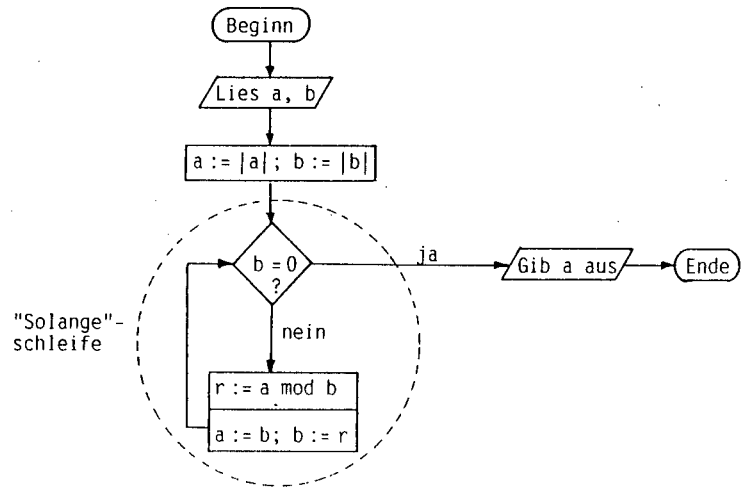
Das Verblüffende an der entscheidenden Eigenschaft (iv) ist, daß die Berechnung des ggT, der nur von der multiplikativen Struktur der ganzen Zahlen abzuhängen scheint, mit Hilfe der additiven Struktur wesentlich vereinfacht wird. Es handelt sich dabei also um einen cleveren mathematischen Trick! Will man nur die multiplikative Struktur benützen, steht man vor dem schon erwähnten Problem, große Primfaktoren zu erkennen.

(1.3) Der Euklidische Algorithmus wird jetzt so beschrieben, daß er sich leicht zu einem Programm in einer beliebigen Programmiersprache umsetzen läßt. Eckige Klammern bedeuten Kommentare.

Version 1: Klassische Kettendivision.

EA1: Lies a, b ein; setze a := |a|; b := |b|.
EA2: Solange b ≠ 0, führe aus EA3 und EA4.
EA3: Setze q := $\left\lfloor \frac{a}{b} \right\rfloor$; [= größte ganze Zahl $\leq \frac{a}{b}$]
setze r := a - q*b.
[r ist der Divisionsrest mit 0 ≤ r < b. Man kann EA3 zu r := a mod b zusammenfassen; die Funktion mod kann in einigen Programmiersprachen direkt aufgerufen werden.]
[An dieser Stelle ist ggT(a,b) = ggT(b,r). Man kann also a durch b und b durch r ersetzen, ohne den ggT zu ändern; das geschieht in EA4.]
EA4: a := b; b := r. [Damit ist b verkleinert; die Abbruchbedingung b = 0 wird also nach endlich vielen Schritten erreicht.]
EA5: Gib a aus; Ende.

Diese Beschreibung hat gegenüber der Divisionskette in (1.1) auch einen didaktischen Vorteil: Man kommt ohne Indizes aus, und die Grundidee ist etwas klarer erkennbar. (Bei den folgenden Überlegungen haben aber auch die Indizes ihre Vorteile.) Auch das Flußdiagramm trägt zum Verständnis bei:



Explizite Programme in irgendwelchen Programmiersprachen will ich nicht angeben. Die letzte Beschreibung in einer Mischung aus Deutsch und verschiedenen höheren Programmiersprachen entspricht einem weit verbreiteten Brauch in der Informatik, der zu wohlstrukturierten Programmen führt. In Sprachen wie PASCAL oder PL/1 kann man eine solche Beschreibung fast wörtlich übersetzen; bei weniger gut strukturierten Sprachen wie BASIC oder FORTRAN ist die Mühe etwas größer, besonders wenn man die Struktur so gut wie möglich erhalten will.

(1.4) Von den Fragen zur Effizienz sind die zweite und dritte schnell beantwortet: Wir brauchen 3 Speicherplätze, a, b und r, vielleicht einen vierten, q, und eventuell 2 zusätzliche Speicher, wenn die Eingabedaten während des Programmlaufs nicht geändert werden sollen (dies ist zu beachten, wenn man ein Programm als Unterprogramm in ein größeres einbaut). Der Programmieraufwand ist, wie gesehen, sehr gering. Die vierte Frage bleibt vorläufig offen - als Konkurrenz haben wir ja bisher nur den "naiven" Algorithmus mit Hilfe der Primfaktor-Zerlegung, den der Computer mit spürbarem Zögern bearbeitet.

Nun zur Frage 1: Als Maß für den Zeitaufwand soll die Zahl n der Iterationsschritte, also die Anzahl der Ausführungen von EA3 dienen. In (1.1) haben wir die erste grobe Abschätzung $n \leq b$ gefunden. Dort war $a \geq b > 0$ vorausgesetzt. Ist $a < b$, gilt die Abschätzung genauso. (Dann werden übrigens im ersten Iterationsschritt a und b vertauscht.) Das bedeutet, daß der Aufwand proportional zu b ist. In Wirklichkeit ist er aber höchstens proportional zu log(b), d. h. etwa, zur Zahl der Dezimalstellen:

Jeder Iterationsschritt beginnt mit zwei Werten a, b, o.B.d. A. $a \geq b > 0$, und liefert einen neuen Wert r mit $a = qb + r$, $0 \leq r < b$. Da stets $q \geq 1$, folgt $a \geq b + r > 2r$, also $r < \frac{a}{2}$. Gehen wir zur Divisionskette von (1.1) zurück, so folgt induktiv $r_n < \frac{r_0}{2^{n/2}}$ bzw. $\frac{r_1}{2^{(n-1)/2}}$, je nachdem, ob n gerade oder ungerade ist. Im ersten Fall folgt $2^{n/2} < \frac{r_0}{r_n} \leq r_0 = a$ (da $r_n \geq 1$), also $n < 2 \cdot \log_2(a)$.

Im zweiten Fall folgt ebenso $n-1 < 2 \cdot \log_2(b)$. In beiden Fällen gilt also (wenn $a \geq b$) $n < 1 + 2 \cdot \log_2(a)$.

Dieses Ergebnis wird auf b und r_2 (statt auf a und b) angewendet und ergibt $n-1 < 1 + 2\cdot\log_2(b)$, $n < 2 + 2\cdot\log_2(b)$.

SATZ 2. Seien $a,b \in \mathbb{Z}$, $b \neq 0$. Dann ergibt der Euklidische Algorithmus den ggT(a,b) in weniger als $2 + 6.65\cdot\log|b|$ Iterationsschritten.

Dabei ist $\log = \log_{10}$.

Beweis: Da der Schritt EA1 nicht gezählt wird, darf man $a,b \geq 1$ annehmen. Da die Zahl der Schritte für a und b die gleiche ist wie für a+b und b, darf man o.B.d.A. auch $a \geq b$ annehmen. Dann gilt die obige Überlegung und ergibt $n < 2 + 2\cdot\log_2|b| < 2 + 6.65\cdot\log|b|$. ●

Mit einem elementaren Schluß kann man diese Schranke noch deutlich verbessern, siehe Aufgabe II.6.

(1.5) Man kann aber sogar eine "scharfe" Schranke herleiten. Dazu lassen wir uns vom Computer inspirieren.
Wir bezeichnen die Zahl der Schritte für a und b mit L(a,b) und tragen sie wie folgt in eine Tabelle ein: In die Zeile a, $a \geq 1$, wird an die Stelle $b \geq 1$ die Zahl L(a,b) geschrieben (wie man diese Tabelle mit Computerhilfe erstellt, siehe Aufgabe II.1):

a \ b	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	...
1	(1)	(2)	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	
2	1	1	(3)	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	
3	1	2	1	3	(4)	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	
4	1	1	2	1	3	3	4	2	3	3	4	2	3	3	4	2	3	3	4	2	3	3	4	
5	1	2	3	2	1	3	4	(5)	4	2	3	4	5	4	2	3	4	5	4	2	3	4	5	
6	1	1	1	2	2	1	3	3	3	4	4	2	3	3	3	4	4	2	3	3	3	4	4	
7	1	2	2	3	3	2	1	3	4	4	5	5	4	2	3	4	4	5	5	4	2	3	4	
8	1	1	3	1	4	2	2	1	3	3	5	3	(6)	4	4	2	3	3	5	3	6	4	4	
9	1	2	1	2	3	2	3	2	1	3	4	3	4	5	4	5	4	2	3	4	3	4	5	
10	1	1	2	2	1	3	3	2	2	1	3	3	4	4	3	5	5	4	4	2	3	3	4	
11	1	2	3	3	2	3	4	4	3	2	1	3	4	5	5	4	5	6	6	5	4	2	3	
12	1	1	1	1	3	1	4	2	2	2	2	1	3	3	3	3	5	3	6	4	4	4	4	
13	1	2	2	2	4	2	3	5	3	3	3	2	1	3	4	4	4	6	4	5	(7)	5	5	
14	1	1	3	2	3	2	1	3	4	3	4	2	2	1	3	3	5	4	5	4	3	5	6	
15	1	2	1	3	1	2	2	3	3	2	4	2	3	2	1	3	4	3	5	3	4	4	5	
16	1	1	2	1	2	3	3	1	4	4	3	2	3	2	2	1	3	3	4	3	4	5	5	
17	1	2	3	2	3	3	3	2	3	4	4	4	3	4	3	2	1	3	4	5	4	5	5	
18	1	1	1	2	4	1	4	2	1	3	5	2	5	3	2	2	2	1	3	3	3	4	6	
19	1	2	2	3	3	2	4	4	2	3	5	5	3	4	4	3	3	2	1	3	4	4	5	
20	1	1	3	1	1	2	3	2	3	1	4	3	4	3	2	2	4	2	2	1	3	3	5	
21	1	2	1	2	2	2	1	5	2	2	3	3	6	2	3	3	3	2	3	2	1	3	4	
22	1	1	2	2	3	3	2	3	3	2	1	3	4	4	3	4	4	3	3	2	2	1	3	
23	1	2	3	3	4	3	3	3	4	3	2	3	4	5	4	4	4	5	4	4	3	2	1	
⋮																								⋱

An dieser Tabelle kann man einige Regelmäßigkeiten und sogar ein einfaches Bildungsgesetz erkennen, siehe Aufgaben I.7 und II.1. Im Moment ist der wesentliche Gesichtspunkt: Wo tritt ein neues Maximum für L(a,b) auf? Hier:

1 = L(1,1), 2 = L(1,2), 3 = L(2,3), 4 = L(3,5), 5 = L(5,8),
6 = L(8,13), ... ,

und zwar gleichgültig, ob man zeilen- oder spaltenweise sucht. Man kann der Verführung nicht widerstehen, zu vermuten:

• Ein neues Maximum n tritt stets an der Stelle (F_n, F_{n+1}) auf; dabei ist F_n die n-te Fibonacci-Zahl, definiert durch $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ für $n \geq 2$, also $F_2 = 1$, $F_3 = 2$, $F_4 = 3$, $F_5 = 5$, $F_6 = 8$, ...

Daß $L(F_n, F_{n+1}) = n$, ist klar: Die Divisionskette heißt

$$F_n = 0 \cdot F_{n+1} + F_n, \ \ldots\ , \ F_j = 1 \cdot F_{j-1} + F_{j-2}, \ \ldots \text{ bis } F_3 = 2 \cdot F_2.$$

Wer gerne $a \geq b$ haben möchte, soll die Kettendivision für (F_{n+2}, F_{n+1}) ausführen.

Umgekehrt habe die Divisionskette die Länge n (o.B.d.A. $b > 0$). Wir fragen: Wie groß muß b dann mindestens sein? Es ist $r_n \geq 1$, $r_{n-1} \geq 2$ und $r_{i-1} \geq r_i + r_{i+1}$. Durch Induktion erhält man $r_i \geq F_{n+2-i}$; insbesondere folgt $b \geq F_{n+1}$.

Anders formuliert:

SATZ 3 (Binet 1841). Seien $a, b \in \mathbb{Z}$, $0 < b < F_{n+1}$. Dann ergibt der (klassische) Euklidische Algorithmus den ggT(a,b) in höchstens n-1 Iterationsschritten.
ZUSATZ. Das gilt auch für $b = F_{n+1}$, außer wenn $a \equiv F_{n+2} \equiv F_n \pmod b$.

Für $n \geq 2$ ist $F_{n+1} > 0.43769 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^{n+1}$ nach [6; Satz 5] oder mit Induktion, also, wenn die Divisionskette die Länge n hat,
$\log(b) > \log(0.43769) + (n+1) \cdot \log\left(\frac{1+\sqrt{5}}{2}\right)$, also $n < 0.718 + 4.785 \cdot \log(b)$:

SATZ 2'. Seien $a, b \in \mathbb{Z}$, $b \geq 2$. Dann ist die Anzahl der Iterationsschritte im (klassischen) Euklidischen Algorithmus für ggT(a,b) kleiner als $0.718 + 4.785 \cdot \log(b)$.

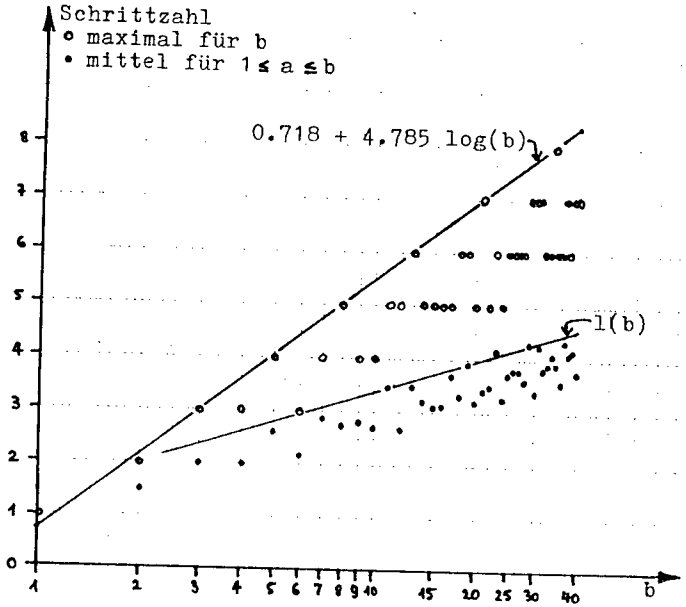
Der schlechteste Fall im Euklidischen Algorithmus ist also die Kettendivision von zwei aufeinanderfolgenden Fibonacci-Zahlen. In diesem Fall wird die zuletzt angegebene Schranke (bis auf Rundungsfehler) erreicht, so daß eine weitere Verbesserung nicht möglich ist. Etwas gröber, aber am einfachsten zu merken, ist die folgende Version:

KOROLLAR. Die Anzahl der Iterationsschritte im Euklidischen Algorithmus für ggT(a,b) ist kleiner als fünfmal die Zahl der Dezimalstellen von b.

(1.6) Die folgende Tabelle gibt für jeden Wert von b = 1, ... ,40 das Minimum von L(a,b) für alle Werte von a, den Mittelwert von L(a,b) für a = 1, ... ,b (dieser Wert ist sinnvoll, da L(a,b) als Funktion von a periodisch mit Periode b ist) und den Wert $l(b) := 1.9405 \cdot \log(b) + 1.467$. (Übereinstimmung von l(b) mit dem Mittelwert ist nur zu erwarten, wenn b eine Primzahl ist, siehe [3; 4.5.3.].)

b	Maximum	Mittel	l(b)	b	Maximum	Mittel	l(b)
1	1	1	1.467	21	7	3.381	4.033
2	2	1.5	2.051	22	5	3.5	4.072
3	3	2	2.393	23	6	4.174	4.110
4	3	2	2.635	24	5	3.25	4.145
5	4	2.6	2.823	25	6	3.72	4.180
6	3	2.167	2.977	26	6	3.808	4.213
7	4	2.857	3.107	27	6	3.778	4.245
8	5	2.75	3.220	28	6	3.571	4.275
9	4	2.778	3.319	29	7	4.310	4.305
10	4	2.7	3.408	30	7	3.367	4.333
11	5	3.455	3.488	31	7	4.258	4.361
12	5	2.667	3.561	32	6	3.812	4.388
13	6	3.462	3.629	33	6	3.909	4.414
14	5	3.214	3.691	34	8	4.088	4.439
15	5	3.067	3.749	35	6	3.886	4.463
16	5	3.125	3.804	36	6	3.556	4.487
17	5	3.706	3.855	37	7	4.351	4.510
18	6	3.278	3.903	38	6	4.132	4.533
19	6	3.895	3.949	39	7	4.154	4.555
20	5	3.2	3.992	40	7	3.75	4.576

Für eine realistische Einschätzung der Effizienz eines Algorithmus ist nicht nur der maximale, sondern auch der mittlere Aufwand zu betrachten. Die mathematische Theorie für den Mittelwert von L(a,b) ist allerdings viel zu kompliziert, um sie hier zu behandeln. Der Koeffizient 1.9405 in l(b) ist übrigens der Anfang der Dezimalentwicklung von $\frac{12 \cdot \ln 2 \cdot \ln 10}{\pi^2}$.



§2. DIE KÜRZESTESTE DIVISIONSKETTE

Da der Aufwand beim Euklidischen Algorithmus höchstens proportional zum Logarithmus, also zur Stellenzahl, der Eingabedaten wächst, zählt er zu den sehr schnellen Algorithmen. (Bei m-stelligen Zahlen ist allerdings der Aufwand für die Division proportional zu m^2 - mit Tricks zu $m \cdot \log(m) \cdot \log\log(m)$ verbesserbar -, so daß eine realistische Einschätzung für den Euklidischen Algorithmus einen Aufwand in der Größenordnung m^2 ergibt; zu beachten: Die zu dividierenden Zahlen werden im Verlauf des Verfahrens schnell kleiner.) Als effektiv betrachtet man im allgemeinen noch Algorithmen, deren Aufwand polynomial von den Eingabedaten abhängt; bei vielen wichtigen Algorithmen wächst der Aufwand leider exponentiell (z. B. bei der Primzerlegung von ganzen Zahlen).

Trotzdem kann man natürlich fragen: Ist der Euklidische Algorithmus optimal, oder läßt er sich noch verbessern?

(2.1) Wir bezeichnen allgemein eine Folge r_i mit

$$r_0 = a,\ r_1 = b,\ \ldots,\ r_{i-1} = q_i r_i + r_{i+1},\ \ldots,\ \text{wobei } r_i,\ q_i \in \mathbb{Z},$$

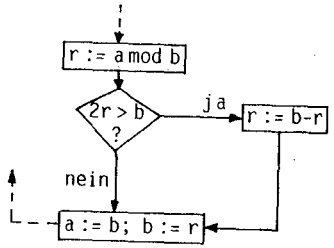
als Divisionskette für (a,b); wegen Hilfssatz 1(iv) ist stets $r_n = ggT(a,b)$, wenn $r_n \neq 0$, $r_{n+1} = 0$. Nun drängt sich die Frage auf, ob man durch geschickte Wahl der q_i und r_i vielleicht eine kürzere Divisionskette erreichen kann als mit der klassischen Kettendivision.

Eine wohlbekannte Verkürzung ergibt sich aus der Beobachtung, daß man die Geschwindigkeit, mit der die Folge der Reste gegen 0 läuft, erhöhen kann: Ist der Divisionsrest $r = a - qb$, $0 \leq r < b$, größer als $\frac{b}{2}$, so ist $r' = a - (q+1)b = r - b$ zwar negativ, aber $|r'| = |r - b| = b - r < \frac{b}{2}$; man kann also einen betragsmäßig kleineren Rest erreichen. Die entsprechende Änderung des Euklidischen Algorithmus sieht so aus:

Version 2:Kettendivision mit Minimalrest.

> Genau wie Version 1, nur wird EA3 ergänzt durch
>
> EA3a: Falls $2r > b$, setze $r := b - r$.

(Man kann noch vorschalten: Falls $2b > a$, setze $b := a - b$; - aber das lohnt sich nicht.) Die entsprechende Ergänzung des Flußdiagramms steht hier rechts.



Aus unserem Zahlenbeispiel wird jetzt diese Divisionskette:

Version 2: Kettendivision mit Minimalrest.

272526	=	8·32574	+	11934
32574	=	3·11934	-	3228
11934	=	4· 3228	-	978
3228	=	3· 978	+	294
978	=	3· 294	+	96
294	=	3· 96	+	6
96	=	16· 6		

7 Iterationsschritte

Wir haben also zwei Iterationsschritte gespart. Dafür war jeder einzelne Schritt um eine Abfrage und gelegentlich eine Subtraktion teurer. Lohnt sich das im allgemeinen?

(2.2) Zur Beantwortung dieser Frage sind zunächst Schranken für den abgeänderten Algorithmus analog zu Satz 2 zu finden. In diesem Abschnitt will ich die Divisionskette stets in der Form

$$r_{i-1} = q_i r_i \pm r_{i+1} \quad \text{mit } q_i \in \mathbb{Z},\ 0 \le r_{i+1} \le \frac{r_i}{2},$$

schreiben. Aus $r_{i+1} \le \frac{r_i}{2}$ folgt schon ganz einfach eine eindrucksvolle Schranke: $r_n \le \frac{r_1}{2^{n-1}}$, also $2^{n-1} \le \frac{r_1}{r_n} \le r_1 = b$,

$n-1 \le \log_2(b)$, $n \le 1 + \log_2(b) < 1 + 3.33 \cdot \log(b)$.

SATZ 4. Seien $a,b \in \mathbb{Z}$, $b \neq 0$. Dann ergibt der Euklidische Algorithmus mit Minimalrest den ggT(a,b) in weniger als $1 + 3.33 \cdot \log|b|$ Iterationsschritten.

(2.3) Wie bei der Kettendivision erhält man eine genaue Schranke durch die Untersuchung des schlechtesten Falls; diese ist hier etwas trickreicher.

HILFSSATZ 2. Sei $a \ge 2b > 0$.
(i) Ist $a = qb + s$ mit $q \ge 1$ und $-\infty < s < b$, so ist sogar $q \ge 2$.
(ii) Ist $a = qb \pm r$ mit $q \in \mathbb{Z}$ und $0 \le r \le \frac{b}{2}$, so ist $a \ge 2b + r$.

Beweis: (i) Wäre $q = 1$, so $s = a - b \ge \frac{a}{2} \ge b$, Widerspruch!
(ii) Sei außerdem $a = \tilde{q}b + s$ mit $0 \le s < b$, insbesondere $\tilde{q} \ge 2$.
1. Fall: $s \le \frac{b}{2}$. Dann ist $r = s$, $q = \tilde{q} \ge 2$, $a = \tilde{q}b + s \ge 2b + r$.
2. Fall: $\frac{b}{2} < s < b$. Dann ist $r = b - s < s$, $a = \tilde{q}b + s > 2b + r$. ⬢

Wir sehen uns nun wie bei Satz 3 an, wie groß a und b mindestens sein müssen, damit die Divisionskette die Länge n hat: Definieren wir die Folge G_n von ganzen Zahlen rekursiv durch

$$G_0 = 0,\ G_1 = 1,\ G_j = 2G_{j-1} + G_{j-2} \text{ für } j = 2,3,4,\ldots,$$

so folgt: $r_n \ge 1 = G_1$, $r_{n-1} \ge 2 = G_2$, und induktiv aus Hilfssatz 2 (ii)

$$r_{i-1} \ge 2r_i + r_{i+1} \ge G_{n-i+2} \text{ für } i = n, n-1, \ldots, 2, \text{ also } b = r_1 \ge G_n.$$

Nach [6, Satz 1] oder mit Induktion ist

$$G_n = \frac{(1+\sqrt{2})^n - (1-\sqrt{2})^n}{2\sqrt{2}}.$$ Daraus folgt:

SATZ 5 (Dupré 1846). Seien $a,b \in \mathbb{Z}$ und $0 < b < G_{n+1}$. Dann ergibt der Euklidische Algorithmus mit Minimalrest den ggT(a,b) in höchstens n Iterationsschritten.

Man sieht sofort, daß $\frac{G_n}{(1+\sqrt{2})^n} = \frac{1}{2\sqrt{2}} + (-1)^{n+1} \frac{1}{2\sqrt{2}(1+\sqrt{2})^{2n}}$ für n gegen unendlich gegen $\frac{1}{2\sqrt{2}}$ konvergiert, und zwar für ungerade n monoton fallend, für gerade monoton steigend. Für alle $n \ge 1$ gilt daher

$$G_n > c \cdot (1+\sqrt{2})^n \text{ mit } c < \frac{G_2}{(1+\sqrt{2})^2} = \frac{2}{3+2\sqrt{2}}, \text{ etwa } c = 0.34314.$$

Also folgt oben $b > c\cdot(1+\sqrt{2})^n$,

$$n < -\frac{\log(c)}{\log(1+\sqrt{2})} + \frac{\log(b)}{\log(1+\sqrt{2})} < 1.214 + 2.613\cdot\log(b):$$

SATZ 4'. Seien $a,b \in \mathbb{Z}$, $b \neq 0$. Dann ist die Anzahl der Iterationsschritte im Euklidischen Algorithmus mit Minimalrest für ggT(a,b) kleiner als $1.214 + 2.613\cdot\log|b|$.

Auch hier zum Merken wieder eine einfache Version:

KOROLLAR. Die Anzahl der Iterationsschritte im Euklidischen Algorithmus mit Minimalrest für ggT(a,b) ist höchstens 3-mal die Zahl der Dezimalstellen von b.

Beweis: Sei m die Zahl der Dezimalstellen von b.
Für $m \geq 4$ ist $1.214 + 2.613m < 3m$, die Behauptung folgt also aus Satz 4'.
Für $m = 3$ ist $b < G_{10} = 2378$, also die Schrittzahl ≤ 9,
Für $m = 2$ ist $b < G_7 = 169$, also die Schrittzahl ≤ 6,
Für $m = 1$ ist $b < G_4 = 12$, also die Schrittzahl ≤ 3,
Für $m = 0$ ist die Schrittzahl 0. ●

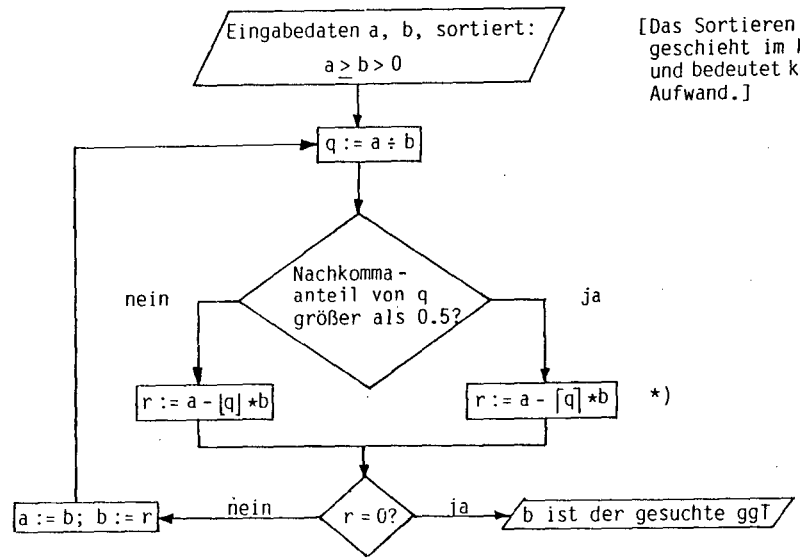
(2.4) Welche von den beiden Versionen des Euklidischen Algorithmus ist nun die bessere? Wir haben in beiden Fällen scharfe Schranken für die Zahl der Iterationsschritte erhalten, und die Schranke für die Minimalrest-Version war etwa nur das $\frac{2.613}{4.785}$-fache derjenigen für die klassische Version, also knapp 55%. Für einen ehrlichen Vergleich muß man natürlich durchschnittliche Werte heranziehen:

Bei der klassischen Kettendivision benötigt man durchschnittlich ungefähr $1.94\cdot\log|b|$ Iterationsschritte [3; 4.5.3, S. 357], siehe das Diagramm in (1.6), bei der Kettendivision mit Minimalrest $1.35\cdot\log|b|$, also knapp 70% davon [3; 4.5.3, Exercise 30, S. 361 bzw. 605].

Die weitere Antwort hängt von der Maschine ab, auf der der Algorithmus durchgeführt wird. Diese "Maschine" kann sein:
a) ein Computer zusammen mit einer Programmiersprache,
b) ein (nichtprogrammierbarer) Taschenrechner, dazu Bleistift und Papier als Hilfsspeicher,
c) ein menschliches Gehirn, dazu ebenfalls Bleistift und Papier.

Für die Entscheidung zwischen den beiden Versionen gibt den Ausschlag, ob die eingesparten 30% Divisionen "teurer" sind als die zusätzlichen Abfragen (in denen ja noch eine Multiplikation mit 2 steckt). Auf den Maschinen b und c sind die Abfragen (verglichen mit den Divisionen) kostenlos, die Version mit Minimalrest ist also schneller. Für den Computer kann man die Antwort nicht so schnell geben - hier muß man sich den Computer und die verwendete Programmiersprache sehr genau ansehen! Ich will die Frage offen lassen. Der Vergleich in [1; 2.2] ist unfair, weil dort für die klassische Version ein besonders günstiges, für die Minimalrest-Version ein ungünstiges Programm verwendet wurde.

Hier noch meine Empfehlung für das Vorgehen mit dem Taschenrechner oder beim schriftlichen Rechnen:



*) $\lfloor q \rfloor$ = größte ganze Zahl $\leq q$, $\lceil q \rceil$ = kleinste ganze Zahl $\geq q$]

Auf dem Computer kann man das Programm für den Minimalrest-Algorithmus eventuell verkürzen, wenn man statt der "Abschneide-Funktion" $\lfloor\ \rfloor$ (etwa INT) auch eine Rundungsfunktion (etwa CINT) zur Verfügung hat.

(2.5) Durch die Kettendivision mit Minimalrest ist die Zahl der Iterationsschritte anscheinend verringert worden, aber das ist für beliebige Eingabedaten a, b keinesfalls bewiesen. Auch könnte es für bestimmte a, b noch kürzere Divisionsketten geben; daß das nicht so ohne weiteres auszuschließen ist, zeigt das Beispiel a = 8, b = 5:

Kettendivision mit Minimalrest	$8 = 2\cdot 5 - 2$	aber auch	$8 = 1\cdot 5 + 3$	(nicht der Minimalrest)
	$5 = 2\cdot 2 + 1$		$5 = 2\cdot 3 - 1$	
	$2 = 2\cdot 1$ → 3 Schritte		$3 = 3\cdot 1$	→ 3 Schritte

Zunächst stellen wir fest, daß der Minimalrest r mit $a = qb \pm r$, $0 \le r \le \frac{b}{2}$, eindeutig festgelegt ist, außer im Fall b gerade und $r = \frac{b}{2}$, wo auch $r = -\frac{b}{2}$ möglich ist. In diesem Fall ist aber der nächste Schritt schon der letzte: $b = 2r + 0$. Jedenfalls ist die Länge $\lambda(a,b)$ der Divisionskette mit Minimalrest für $|a|$ und $|b|$ für beliebige $a, b \in \mathbb{Z}$ eindeutig bestimmt. Das definiert eine Funktion

$\lambda: \mathbb{Z} \times \mathbb{Z} \to \mathbb{N}_0$, und $\lambda(a,b) = 0 \iff b = 0$, $\lambda(a,b) = 1 \iff b \mid a$.

Sei ferner $\mu(a,b)$ die Länge der kürzestmöglichen Divisionskette, also ebenso

$\mu: \mathbb{Z} \times \mathbb{Z} \to \mathbb{N}_0$ mit $\mu(a,b) = 0 \iff b = 0$, $\mu(a,b) = 1 \iff b \mid a$.

Nach Definition ist $\mu \le \lambda$. Weitere Eigenschaften von μ und λ:

HILFSSATZ 3. Für alle $a, b \in \mathbb{Z}$ gilt:

(i) $\mu(a,b) = \mu(-a,b) = \mu(a,-b)$, $\lambda(a,b) = \lambda(-a,b) = \lambda(a,-b)$.

(ii) Ist $c \equiv a \pmod b$, so $\mu(c,b) = \mu(a,b)$ und $\lambda(c,b) = \lambda(a,b)$.

(iii) Ist $b \neq 0$, so gibt es ein $r \in \mathbb{Z}$ mit $a = qb + r$ und $\mu(b,r) = \mu(a,b) - 1$ und ein $r' \in \mathbb{Z}$ mit $a = q'b \pm r'$, $0 \le r' \le \frac{b}{2}$ und $\lambda(b,r') = \lambda(a,b) - 1$.

(iv) $\mu(b,a) \le \mu(a,b) + 1$.

Der Beweis dürfte klar sein.

SATZ 6. Für alle $a, b \in \mathbb{Z}$ gilt $\lambda(a,b) = \mu(a,b)$. D. h., der Euklidische Algorithmus mit Minimalrest liefert stets eine minimale Divisionskette.

Beweis durch Induktion über $\mu(a,b)$: Die Fälle $\mu(a,b) = 0$ und 1 sind durch die Vorbemerkung erledigt. Sei also jetzt $\mu(a,b) \ge 2$, insbesondere $b \neq 0$ und $b \nmid a$. Man darf o.B.d.A. $b > 0$ annehmen. Sei r mit $a = qb + r$ und $\mu(b,r) = \mu(a,b) - 1$ gewählt, insbesondere $r \neq 0$; ersetzt man notfalls a durch -a (und q durch -q), kann man $r > 0$ annehmen. Ebenso sei r' mit $a = q'b \pm r'$

und $0 < r' \le \frac{b}{2}$ gewählt, also $\lambda(b,r') = \lambda(a,b) - 1$.

Nach Induktionsvoraussetzung ist $\mu(b,r) = \lambda(b,r)$, und das bedeutet, daß die Division mit Minimalrest $b = q_2 r \pm r_2$, $0 \le r_2 \le \frac{r}{2}$, gleichzeitig ein Glied einer minimalen Divisionskette ist. Außerdem ist

$$\mu(a,b) = \mu(b,r) + 1 = \mu(r,r_2) + 2$$
$$= \lambda(b,r) + 1 = \lambda(r,r_2) + 2.$$

Jetzt werden mehrere Fälle unterschieden.

1. Fall, $r \le \frac{b}{2}$: Dann ist $r' = r$, also $\mu(a,b) = \lambda(b,r) + 1 = \lambda(b,r') + 1 = \lambda(a,b)$.

2. Fall, $\frac{b}{2} < r \le b$: Dann ist $r' = b - r$.

a) Falls $r' \le \frac{r}{2}$, ist $b = 1\cdot r + r'$ die Division mit Minimalrest, also

$q_2 = 1$, $r_2 = r'$,

$\mu(a,b) = \lambda(r,r') + 2 = \lambda(b,r') + 2 = \lambda(a,b) + 1 > \lambda(a,b)$, Widerspruch!

↑ weil $b \equiv r \pmod{r'}$, also nach Hilfssatz 3(i,ii)

b) Sonst ist $\frac{r}{2} < r' \le \frac{b}{2} < r$, also $b = 2r - (r - r')$ die Division mit Minimalrest, $q_2 = 2$, $r_2 = r - r'$,

$$\mu(a,b) = \mu(r,r-r') + 2 \overset{\text{HS3 (i,ii)}}{=} \mu(r',r-r') + 2 \overset{\text{HS3 (iv)}}{\ge} \mu(r-r',r') + 1 =$$
$$= \mu(r,r') + 1 \overset{\text{Induktion}}{=} \lambda(r,r') + 1 = \lambda(b,r') + 1 = \lambda(a,b),$$

also $\mu(a,b) = \lambda(a,b)$.

3. Fall, $b < r \le \frac{3}{2}b$: Dann ist $r' = r - b < \frac{b}{2} < \frac{r}{2}$, also $b = r - r'$ die Division mit Minimalrest; es geht jetzt wie bei 2a) weiter.

4. Fall, $\frac{3}{2}b < r < 2b$: Dann ist $r' = 2b - r$; die Division mit Minimalrest ist $b = r - (r - b)$, denn aus $\frac{r}{2} < b < r$ folgt $0 < r - b < \frac{r}{2}$.

Also $q_2 = 1$, $r_2 = r - b = b - r'$,

$$\mu(a,b) = \mu(r,r-b) + 2 = \mu(b,r-b) + 2 = \mu(b,b-r') + 2 = \mu(r',b-r') + 2$$
$$\ge \mu(b-r',r') + 1 = \mu(b,r') + 1 = \lambda(b,r') + 1 = \lambda(a,b).$$

5. Fall, $r \ge 2b$: Dann ist $b = 0\cdot r + b$ die Division mit Minimalrest, also $r_2 = b$ und $\mu(a,b) = \mu(r,b) + 2 = \mu(a,b) + 2$, da $r \equiv a \pmod b$, Widerspruch! ●

Dieser Satz ist typisches "Hintergrundwissen" für Lehrer. Wenn man nur Divisionsketten mit der Nebenbedingung $-b < r \le b$ zur Konkurrenz zuläßt, stammt das Ergebnis von Kronecker und kann mit Hilfe der Kettenbruch-Theorie

bewiesen werden, siehe:

Paul BACHMANN: Niedere Zahlentheorie. Erster Teil.
Teubner, Leipzig 1902 (S. 140ff).

In der vollen Allgemeinheit stammt das Ergebnis offenbar von

Daniel LAZARD: Le meilleur algorithme d'Euclide pour K[x] et Z.
C. R. Acad. Sc. Paris 284 (1977), 1 - 4,

und von dort habe ich auch den Beweis (mit kleinen Änderungen) übernommen.

§3. BINÄRE VARIANTEN DES EUKLIDISCHEN ALGORITHMUS

Haben wir jetzt den optimalen Algorithmus für den größten gemeinsamen Teiler gefunden? Satz 6 sagt uns, daß wir in der Tat keine kürzere Kettendivision mehr finden können. Damit ist die Eigenschaft (iv) des ggT erschöpfend ausgebeutet. Aber wir haben bisher die Eigenschaften (v) und (vi) noch gar nicht verwendet! Daß diese nützlich sein können, zeigt das Beispiel

ggT(1200,870) = 10•ggT(120,87) = 10•ggT(12,87) = 10•ggT(3,87) = 10•3 = 30,

das typisch für die "naive" ggT-Bestimmung ist. Der Haken dabei ist, daß man gemeinsame Teiler selten so deutlich sieht wie die Teiler 2, 3 und 5, also kleine Primteiler.

(3.1) Während ich bisher nur Ergebnisse aus dem 5. Jahrhundert v. Chr. bis zum 19. Jahrhundert n. Chr. vorgetragen habe, will ich jetzt einen Algorithmus vorstellen, der noch keine 20 Jahre alt ist. Die Idee ist ganz einfach: Das Erkennen und Wegdividieren des Primteilers 2 ist völlig unproblematisch, insbesondere bei Binärdarstellung der ganzen Zahlen; man prüft, ob die letzte Ziffer 0 ist und schneidet diese gegebenenfalls ab. Natürlich kommt man auf diese Weise nicht weit. Hat man aber beide Eingabezahlen ungerade gemacht, führt man den "Euklidischen Schritt" in seiner billigsten Version aus: Subtraktion. Das ergibt einen geraden Rest, den man wieder sukzessive halbieren kann. Bei unserem Standard-Beispiel sieht das so aus:

Version 3: Binärer Euklidischer Algorithmus.

a	b	r
272526	32574	(k = 1)
136263	16287	119976, 59988, 29944, 14997
16287	14997	1290, 645
14997	645	14352, 7176, 3588, 1794, 897
897	645	252, 126, 63
645	63	582, 291
291	63	228, 114, 57
63	57	6, 3
57	3	54, 27
27	3	24, 12, 6, 3

2^k = größte gemeinsame Zweierpotenz in a und b = größte Zweierpotenz in ggT(a,b).

ggT(a,b) = 2•3 = 6.

9 Iterationsschritte.

Das sind zwar recht viele Schritte, aber eben ganz billige: 9 Subtraktionen und 20 Divisionen durch 2, einschließlich der nötigen Paritätsprüfungen. Andere (teure) Divisionen kommen nicht vor!

(3.2) Extrahiert man aus dem Beispiel die allgemeine Methode, so entsteht der binäre (Euklidische) Algorithmus von

J. STEIN: Computational problems associated with Racah algebra.
J. Comput. Phys. 1(1967), 397 - 405.

Version 3: Binärer Euklidischer Algorithmus.

```
BA0: Funktion ungerade(a);     [dies ist ein Unterprogramm]
        übernimm a aus dem Hauptprogramm; x := a;
        solange x durch 2 teilbar, führe aus x := x/2;
        übergib x als Funktionswert; springe zurück.
BA1: Beginn des Hauptprogramms;
        lies a, b ein; setze a := |a|; b := |b|.
BA2: Falls b = 0, gib a aus; Ende.
     Falls a = 0, gib b aus; Ende.
BA3: Finde das maximale k mit 2^k|a und 2^k|b;
        a := ungerade(a); b := ungerade(b);
            [BA1 bis BA3 stellen einen Vorlauf zur eigentlichen Iteration
             dar. Beide Zahlen sind jetzt ungerade, der ggT hat den
             Faktor 2^k verloren.]
```

```
BA4: Solange a ≠ b, führe BA5 aus.
   BA5: Falls a > b, setze a := ungerade(a-b);
            [Dadurch wird a durch eine kleinere ungerade
             Zahl ersetzt; der ggT bleibt unverändert.]
        sonst setze b := ungerade(b-a);
            [Dadurch wird b ...]
BA6: Gib 2^k*a aus; Ende.
```

Das sieht natürlich schon etwas komplizierter aus; das zugehörige Flußdiagramm zu zeichnen, will ich dem Leser überlassen.
Daß dieser Algorithmus korrekt ist, ergibt sich fast schon aus den hinzugefügten Kommentaren: BA1, 2 und 3 dienen nur zur Vorbereitung und werden mit Sicherheit beendet; zählen wir nun als "Schritte" jede Ausführung von BA5, so wird jedesmal eine der beiden Zahlen echt verkleinert, beide bleiben aber positiv. Spätestens, wenn beide bei 1 angelangt sind, ist der Algorithmus beendet, d. h., nach allerhöchstens $|a| + |b| - 2$ Schritten.
Der Nachteil gegenüber dem klassischen Euklidischen Algorithmus ist nicht zu übersehen: Der Programmieraufwand ist deutlich größer. Eine genaue Analyse der Effizienz will ich unterlassen. Sie ist in [3; 4.5.2, S. 323 und 330ff] durchgeführt; für die dort benützte Maschine ergibt sich eine mittlere Laufzeit-Einsparung von ca. 20% gegenüber der klassischen Kettendivision, beides in Assembler programmiert; dabei sind aber die Schritte BA4 und BA5 noch etwas cleverer formuliert:

Version 3':

```
BA4': Setze r := a-b; solange r ≠ 0, führe BA5' aus.
   BA5': Setze r := ungerade(r);
         falls r > 0, setze a := r;
         sonst setze b := -r.
           [Das größere von a und b wird
            durch |ungerade(a-b)| ersetzt.]
```

Die Zahl der Subtraktionen ist im schlechtesten Fall
$N = \lceil \log_2(\max(a,b)) \rceil < 3.33 \cdot \log_{10}(\max)$ und im Mittel $0.7 \cdot N < 2.33 \cdot \log_{10}(\max)$.

(3.3) Damit niemand glaubt, jetzt seien die Möglichkeiten, den Euklidischen Algorithmus zu variieren, erschöpft, will ich noch eine weitere Version angeben, die vorgeschlagen wurde von

V. C. HARRIS: An algorithm for finding the greatest common divisor. Fibonacci Quarterly 8(1970), 102 - 103.

Ähnlich wie beim binären Algorithmus werden, wo immer möglich, Zweierpotenzen wegdividiert; beim "euklidischen Schritt" wird jedoch statt der Subtraktion eine Division mit möglichst kleinem geradem Rest durchgeführt:

Seien a, b ungerade, $a = qb + r$ mit $0 \leq r < b$. Dann ist entweder r gerade oder $a = (q+1)b - r'$ mit $r' = b - r$ gerade.

Daraus gewinnen wir den Algorithmus:

Version 4: Gemischter Euklidischer Algorithmus.

```
MA0 = BA0; MA1 = BA1; MA2 = BA2; MA3 = BA3.
MA4: Solange b ≠ 0, setze b := ungerade(b).
          [Jetzt sind a und b wieder ungerade, der ggT
           ist unverändert.]
     Führe MA5 und MA6 aus.
   MA5: r := a mod b
        falls r ungerade, setze r := b-r.
          [Jetzt ist r gerade, der ggT unverändert.]
   MA6: a := b; b := r.
MA7 = BA6.
```

Natürlich sind auch weitere Mischformen aus Kettendivision und Kürzung offensichtlicher Faktoren möglich; ich überlasse Ihnen, sich solche auszudenken.

Wenden wir den neuen Algorithmus noch auf das übliche Beispiel an:

Version 4: Gemischter Euklidischer Algorithmus.

a	b	r
272526	32574	(k = 1)
136263	16287	5967, 10320, 5160, 2580, 1290, 645
16287	645	162, 81
645	81	78, 39
81	39	42, 21
39	21	18, 9
21	9	3, 6, 3
9	3	0

ggT(a,b) = 2·3 = 6.
7 Iterationsschritte.

Der gemischte Algorithmus schneidet also für unser Beispiel gegenüber dem klassischen und dem binären recht gut, gegenüber der Kettendivision mit Minimalrest nicht so gut ab. Eine gründliche Analyse hat offenbar bisher niemand durchgeführt, vgl. [3; 4.5.3; S. 339, Ex. 33]. Das eine Beispiel hier sagt im Grunde gar nichts (im Gegensatz zu der optimistischen Bemerkung in (1.1)).

Eine weitere Version des Euklidischen Algorithmus wird in den Aufgaben II.10 und III.4 behandelt.

(3.4) Fazit: Zur Ermittlung des größten gemeinsamen Teilers zweier ganzer Zahlen steht uns mit dem Euklidischen Algorithmus eine sehr effektive Methode zur Verfügung. Dennoch kann es unter Umständen noch schnellere Methoden geben. Wir haben drei abgeänderte Versionen des Euklidischen Algorithmus kennengelernt, die alle ihre Vor- und Nachteile besitzen und zwischen denen man keine allgemein gültige Entscheidung treffen kann, weil der Aufwand bei allen von der gleichen Größenordnung ist. Die systematische Bestimmung des Aufwandes hat uns, wie in der Einleitung angekündigt, auf interessante mathematische Probleme geführt; endgültige Entscheidungshilfen zwischen den verschiedenen Versionen fehlen aber noch: Hier können wir wieder Inspirationen vom Computer erhoffen.

Das Problem, für den jeweiligen Zweck die optimale Version zu bestimmen, erfordert eine sehr detaillierte Analyse des Laufzeitverhaltens. Auf unterster Stufe sollte man zählen:
- beim Computer die Anzahl der benötigten Takte,
- beim Taschenrechner die Anzahl der Tastendrucke und die Anzahl der Zeichen beim Aufschreiben von Zwischenergebnissen,
- beim schriftlichen Rechnen die Anzahl der zu schreibenden Zeichen.

(Bei einer solchen Zählung kann man sich auch wieder vom Computer helfen lassen - ein entsprechendes Programm käme vielleicht als Beitrag für den Jugendwettbewerb Informatik in Frage. Siehe die Aufgabensammlung am Ende.)
Man kann dabei betrachten:
- den schlechtesten Fall,
- den Mittelwert

} für alle Eingabedaten aus einem beschränkten Bereich,

- oder man gewinnt empirische Ergebnisse, indem man Eingabedaten zufällig aus einem bestimmten Bereich wählt.

Z. B. kann man bei allen Versionen des Euklidischen Algorithmus für eine feste Zahl N Eingabewerte $1 < a,\ b \leq N$ betrachten und als "Komplexitätsmaß" T(N) die mittlere Zahl von Schritten für alle solchen Werte bestimmen.

Hier bieten sich durchaus Aufgaben für "Jugend-forscht"-Projekte an. Die allgemeine Untersuchung des Verhaltens der Funktion T für $N \to \infty$ dürfte dafür allerdings zu schwer sein, siehe etwa [3; 4.5.3, S. 344 - 357] oder

Hans KILIAN: Zur mittleren Anzahl von Schritten beim euklidischen Algorithmus. Elemente Math. 38(1983), 11 - 15.

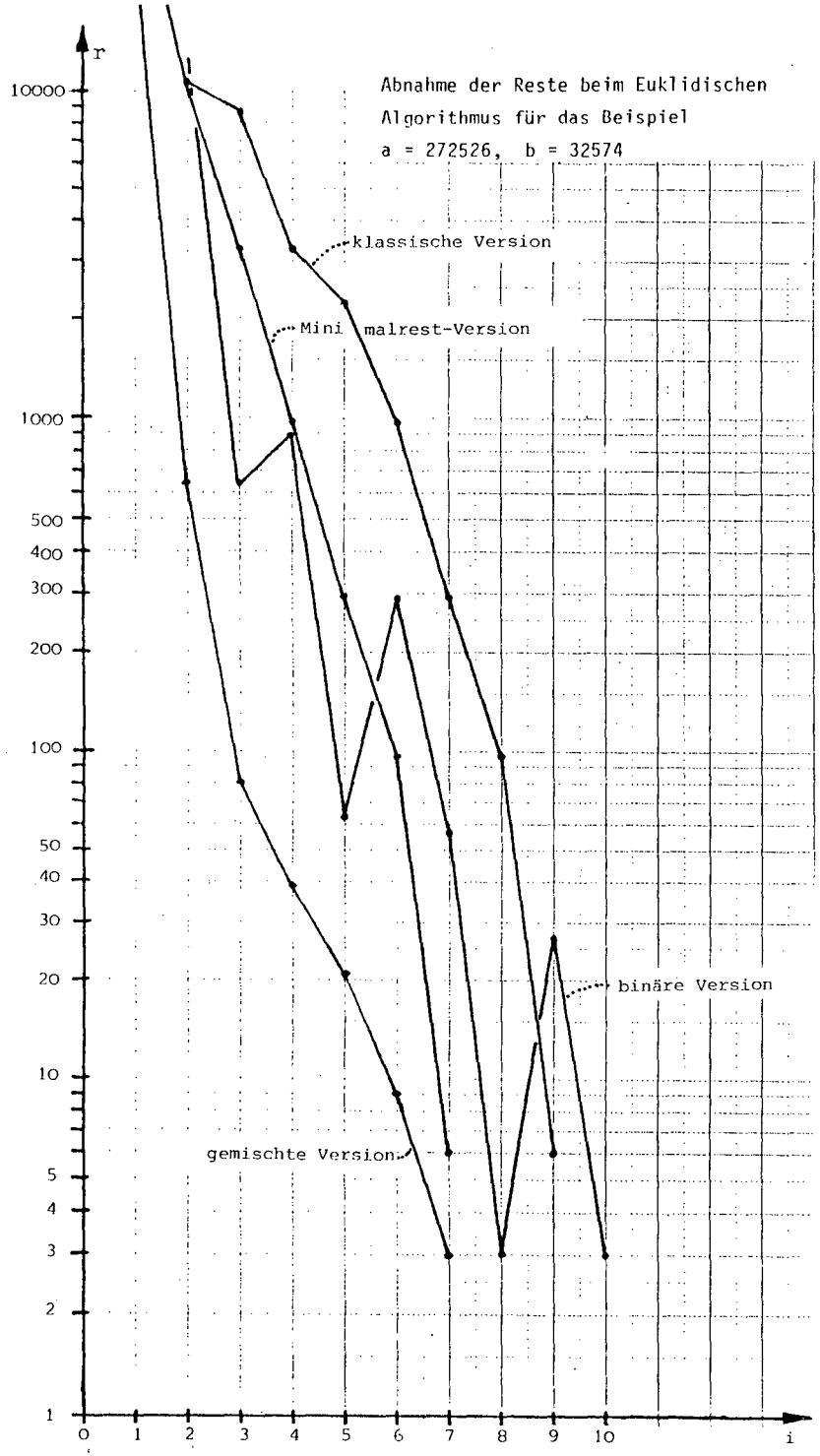
Der klassische und der binäre Algorithmus für einen Computer sind in [3] ausführlich analysiert, dagegen nicht der gemischte (der schlechteste Fall ist in der Originalarbeit hergeleitet). Für diesen wären selbst empirische Ergebnisse wünschenswert und könnten vielleicht zu einer Änderung der folgenden Empfehlung führen:

1.) Auf dem Computer ist der klassische Euklidische Algorithmus vorzuziehen, weil er so einfach zu programmieren ist - jede Komplikation des Programms ist eine Fehlerquelle! - und das Ergebnis so atemberaubend schnell liefert, daß eine mögliche Laufzeitverkürzung durch die anderen Versionen nichts bringt, da deren eventueller Vorteil nicht klar genug ist.

2.) Es gibt zwei denkbare Ausnahmen: Muß der ggT von sehr großen Zahlen (multiple precision integers) oder, als Teil eines anderen Programms, sehr oft berechnet werden, so sollte man den binären Algorithmus erwägen, aber nur, wenn man maschinennah, etwa in Assembler, programmiert. Damit ist natürlich die Übertragbarkeit auf eine andere Maschine nicht zu erreichen. Die entsprechende Empfehlung gilt auch für den gemischten Algorithmus, wenn sich sein Laufzeitverhalten als günstig erweist. (Für die "sehr großen Zahlen" siehe auch [3; 4.5.2, S. 327 - 330].)

3.) Auf dem Taschenrechner und beim schriftlichen Rechnen ist die Kettendivision mit Minimalrest, wie in (2.4) beschrieben, vorzuziehen. Auf dem Taschenrechner kann man zusätzlich auch noch, wann immer möglich, die Reste durch die leicht erkennbaren Primfaktoren 2 und 5 teilen - bei einigen Beispielen, die ich durchgerechnet habe, hat sich das sehr gut bewährt!

Das folgende Diagramm veranschaulicht noch den Verlauf der verschiedenen Versionen für unser Zahlenbeispiel.



§4. LINEARE DIOPHANTISCHE GLEICHUNGEN

Die nächstliegende und einfachste Anwendung des Euklidischen Algorithmus ist das Bruchrechnen: Zum Kürzen braucht man den ggT, als Hauptnenner beim Addieren braucht man das kleinste gemeinsame Vielfache (kgV), das sich aus der Formel $\mathrm{kgV}(a,b) = \frac{a \cdot b}{\mathrm{ggT}(a,b)}$ berechnen läßt. Eine nicht ganz so einfache Anwendung ist die Kettenbruch-Entwicklung rationaler Zahlen; dazu benötigt man allerdings die Folge der Quotienten aus der klassischen Kettendivision, so daß diese hier nicht zu ersetzen ist.

Auch bei anderen zahlentheoretischen Problemen spielt der Euklidische Algorithmus eine wichtige Rolle, oft um die Teilerfremdheit von großen Zahlen nachzuweisen. Für Anwendungen in diesem Sinne bei der Erzeugung von "Zufallszahlen" siehe etwa [3; 3.2.1.2 und 3.3.3]. Für eine Anwendung in der Kryptographie siehe:

H. W. LENSTRA: Integer programming and cryptography.
Math. Intelligencer 6(1984), 14 - 19.

(4.1) Auf eine Anwendung will ich näher eingehen: die Lösung von linearen diophantischen Gleichungen mit zwei Unbekannten. Eine solche Gleichung sieht so aus:

Gegeben: $a, b, c \in \mathbb{Z}$,
Gesucht: $x, y \in \mathbb{Z}$,
} so daß $ax + by = c$.

Gleichungen dieser Art traten bei den Hindu-Mathematikern in astronomischen Problemen spätestens seit dem 5. Jahrhundert auf - Aryabhatta (5. Jh.), Brahmegupta (7. Jh.), Bhascara (12. Jh.) - in der Form:
"Finde x, so daß ax - c durch b teilbar ist."
Von Gauss wurde diese Gleichung formuliert als $ax \equiv c \pmod b$, und sie bedeutet, zusätzlich ein y zu finden mit $ax + by = c$, also genau unser Problem.

Diophant (um 250 n. Chr.) hat dieses Problem übrigens nicht behandelt. Im Gegensatz zum heutigen Sprachgebrauch beschäftigte er sich nämlich nicht mit Gleichungen über $\mathbb{Z}$, sondern mit Gleichungen über $\mathbb{Q}$; da $\mathbb{Q}$ ein Körper ist, sind lineare Gleichungen trivial und erst quadratische interessant. Ganz falsch ist der heutige Sprachgebrauch aber nicht, denn etwa die Lösung der Fermat-Gleichung

$x^n + y^n = z^n$ mit gegebenem $n \in \mathbb{N}$, $n \geq 3$, für $x, y, z \in \mathbb{Z}$
ist äquivalent zu der rationalen Gleichung

$u^n + v^n = 1$ für $u, v \in \mathbb{Q}$.

Wie Sie wissen, ist erst 1983 von G. Faltings bewiesen worden, daß das rationale Problem höchstens endlich viele Lösungen hat (für jedes feste $n \geq 3$). Das eigentliche Fermat-Problem hat also höchstens endlich viele Lösungen (x,y,z) mit teilerfremden x,y,z.

(4.2) Die Frage nach der Lösbarkeit der Gleichung $ax + by = c$ kann man auch so formulieren: Ist c als Linearkombination von a und b darstellbar? Ganz selbstverständlich ist das nicht: Wäre zum Beispiel $4x + 6y = 13$ lösbar, so stünde links eine gerade, rechts eine ungerade Zahl. Positive Beispiele haben wir aber auch (nicht ganz explizit) schon gesehen: Im Kettendivisionsschema von (1.1) ist

$r_2 = r_0 - q_1 r_1 = a - q_1 b$ Linearkombination von a und b,

ebenso $r_3 = r_1 - q_2 r_2 = b - q_2 a + q_1 q_2 b = -q_2 a + (1 + q_1 q_2) b$.

Natürlich sind auch $r_0 = 1 \cdot a + 0 \cdot b$ und $r_1 = 0 \cdot a + 1 \cdot b$ solche Linearkombinationen. Wir schließen weiter mit Induktion:
Sei schon $r_j = ax_j + by_j$ für $0 \leq j \leq i$. Dann folgt

$$r_{i+1} = r_{i-1} - q_i r_i = ax_{i-1} + by_{i-1} - q_i ax_i - q_i by_i = a(x_{i-1} - q_i x_i) + b(y_{i-1} - q_i y_i).$$

Für $d = r_n = ggT(a,b)$ gilt insbesondere:

SATZ 7. Seien $a, b \in \mathbb{Z}$ und $d = ggT(a,b)$. Dann gibt es $x, y \in \mathbb{Z}$ mit $ax + by = d$.

(Das entsprechende Ergebnis, auch für den ggT von mehreren Zahlen, hat Herr Hofmeister schon in seinem Vortrag verwendet. Man erhält es, indem man Satz 7 sukzessive anwendet und zwar jeweils die ersten beiden Zahlen durch ihren ggT ersetzt.)

Die eben durchgeführte Überlegung liefert aber nicht nur den Existenzbeweis, sondern eine explizite Konstruktion: Die x_i und y_i erfüllen nämlich die Rekursionsformeln

$x_{i+1} = x_{i-1} - q_i x_i$ mit $x_0 = 1$, $x_1 = 0$,

$y_{i+1} = y_{i-1} - q_i y_i$ mit $y_0 = 0$, $y_1 = 1$,

die bis auf die Startwerte mit der Formel für die r_i übereinstimmen:

$r_{i+1} = r_{i-1} - q_i r_i$ mit $r_0 = a$, $r_1 = b$.

Das Verfahren für unser übliches Beispiel kann man also so tabellieren:

i	r_i	x_i	y_i	q_i
0	272526	1	0	-
1	32574	0	1	8
2	11934	1	-8	2
3	8706	-2	17	1
4	3228	3	-25	2
5	2250	-8	67	1
6	978	11	-92	2
7	294	-30	251	3
8	96	101	-845	3
9	6	-333	2786	16
10	0			

Ergebnis: $ggT(a,b) = 6$
$= -333a + 2786b$.

Die (i+1)-te Zeile $(r_{i+1}, x_{i+1}, y_{i+1})$ wird gewonnen, indem man von der (i-1)-ten das q_i-fache der i-ten subtrahiert.

Bemerkung: So wie das Verfahren in fast allen Zahlentheorie-Lehrbüchern vorgeschlagen wird, und wie es auch die alten Inder benützt haben, muß man sich alle Quotienten q_i merken - was beim schriftlichen Rechnen natürlich kein Problem ist. Das hier vorgestellte Rekursionsverfahren erlaubt, alle q_i zu vergessen. Als ältesten historischen Nachweis dafür habe ich gefunden:

N. SAUNDERSON: The Elements of Algebra. Cambridge 1740.

(4.3) Das vorgestellte Rekursionsverfahren legt nahe, auch allgemein die Darstellung des ggT als Linearkombination durch Ergänzung des Euklidischen Algorithmus gleich mitzukonstruieren.

Der erweiterte Euklidische Algorithmus.

EEA1: Lies a, b ein;
falls $a<0$, setze $r_0 := -a$; $v := -1$;
sonst setze $r_0 := a$; $v := 1$;
falls $b<0$, setze $r_1 := -b$; $w := -1$;
sonst setze $r_1 := b$; $w := 1$;
$x_0 := 1$; $x_1 := 0$; $y_0 := 0$; $y_1 := 1$.
[In v und w werden die Vorzeichen festgehalten; da jetzt $|a|\cdot x + |b|\cdot y = d$ gelöst wird, ist am Ende $d = a\cdot vx + b\cdot wy$.]

EEA2: Solange $r_1 \neq 0$, führe EEA3 und EEA4 aus.

EEA3: Setze $q := \lfloor r_0/r_1 \rfloor$;
$r := r_0 - q*r_1$; $x := x_0 - q*x_1$; $y := y_0 - q*y_1$.
[An dieser Stelle ist $r_0 = ax_0 + by_0$, $r_1 = ax_1 + by_1$ und $r = ax + by$.]

EEA4: $r_0 := r_1$; $r_1 := r$;
$x_0 := x_1$; $x_1 := x$;
$y_0 := y_1$; $y_1 := y$.

EEA5: Gib r_0, $v*x_0$, $w*y_0$ aus; Ende.

Bemerkungen: 1.) Die Umbenennung von a und b in r_0 und r_1 folgt hier der Regel, daß man in einem Unterprogramm Eingabedaten nicht ändern soll - und unser Programm könnte ja irgendwo als Unterprogramm auftreten. Auch könnte es sein, daß man als Ausgabe haben möchte: "Der ggT von a = 272526 und b = 32574 ist 6 = -333*a + 2786*b."

2.) Man kann den Algorithmus ökonomischer gestalten, wenn man die Berechnung von y wegläßt. Am Schluß gilt nämlich $y = \frac{d - xa}{b}$ (außer im trivialen Fall b = 0).

3.) EEA3 und 4 kann man so umgestalten, daß die drei Hilfsspeicher r, x und y zu einem zusammengelegt werden. Dies stört jedoch die übersichtliche Stuktur des Programms und ist daher nur zu empfehlen, wenn man mit Speicherplatz geizen muß.

(4.4) Kehren wir nun zur allgemeinen Gleichung $ax + by = c$ mit gegebenen $a, b, c \in \mathbb{Z}$ zurück. Sie hat eventuell überhaupt keine Lösung $(x,y) \in \mathbb{Z}^2$, wie gesehen. Zum Glück kann man die Existenz von Lösungen sehr einfach beschreiben:

SATZ 8. Seien $a, b, c \in \mathbb{Z}$ und $d = ggT(a,b)$. Dann sind äquivalent:
(i) Es gibt $x, y \in \mathbb{Z}$ mit $ax + by = c$.
(ii) $d \mid c$.
ZUSATZ. Sei dies der Fall und $d = ax_0 + by_0 \neq 0$ (also $a \neq 0$ oder $b \neq 0$).
Dann hat jede beliebige Lösung die Gestalt

$$x = \frac{cx_0 + bt}{d}, \quad y = \frac{cy_0 - at}{d} \quad \text{mit } t \in \mathbb{Z}.$$

Beweis: "(i) $\Rightarrow$ (ii)": $d \mid a,b \Rightarrow d \mid c$.
"(ii) $\Rightarrow$ (i)": Nach Satz 7 gibt es $x_0, y_0 \in \mathbb{Z}$ mit $d = ax_0 + by_0$. Ist $c = ed$, so $c = a(ex_0) + b(ey_0)$.
Zusatz: Sei (x,y) irgendeine Lösung, also $ax + by = c$. Dann ist $a(x - ex_0) + b(y - ey_0) = c - c = 0$, also $\frac{a}{d}(x - ex_0) = -\frac{b}{d}(y - ey_0)$.
Nach Hilfssatz 1 (v) sind $\frac{a}{d}$ und $\frac{b}{d}$ teilerfremd. Aus dem Lemma von Euklid folgt daher $\frac{b}{d} \mid (x - ex_0)$, also $x - ex_0 = t\cdot\frac{b}{d}$ mit einem $t \in \mathbb{Z}$. Also ist $x = ex_0 + t\cdot\frac{b}{d} = \frac{cx_0 + bt}{d}$ und $-\frac{b}{d}\cdot(y - ey_0) = \frac{a}{d}\cdot t\cdot\frac{b}{d}$, $y = ey_0 - \frac{a}{d}\cdot t = \frac{cy_0 - at}{d}$.
Umgekehrt ist natürlich jedes solche Paar (x,y) Lösung. ●

Damit ist die Lösung der linearen diophantischen Gleichung völlig auf die Bestimmung von $d = ggT(a,b)$ und das Auffinden einer Linearkombination $d = ax_0 + by_0$ reduziert, also auf den erweiterten Euklidischen Algorithmus. Ein wichtiger Spezialfall soll ausdrücklich erwähnt werden:

KOROLLAR. Seien $a, b \in \mathbb{Z}$ teilerfremd und $c \in \mathbb{Z}$ beliebig. Dann gibt es stets $x, y \in \mathbb{Z}$ mit $ax + by = c$, und alle solchen Lösungen sind gegeben durch
$x = cx_0 + bt$, $y = cy_0 - at$ mit beliebigem $t \in \mathbb{Z}$,
wobei (x_0,y_0) eine fest gewählte Lösung von $ax_0 + by_0 = 1$ ist.

(4.5) Untersuchen wir also den erweiterten Euklidischen Algorithmus! Daß er korrekt ist, also nach endlich vielen Schritten mit dem richtigen Ergebnis endet, wurde schon bei seiner Konstruktion mitbewiesen; die Zahl der Iterationsschritte ist die gleiche wie ohne die Erweiterung, nur daß jeder Schritt etwas mehr Arbeit erfordert. Der Zeitaufwand hat also durch die Erweiterung einen konstanten Multiplikator erhalten, so etwa in der Größe 2. Die Anzahl der benötigten Speicherplätze ist immer noch gering, ebenso der Aufwand zur Programm-Erstellung und -Prüfung.

Es hat sich aber unbemerkt ein neues Problem eingeschlichen: In unserem Beispiel wachsen die Beträge der Koeffizienten x_i, y_i ziemlich stark an. Wer garantiert uns, daß es hier nicht zu einem Überlauf kommt? Dann wäre der Algorithmus doch nicht korrekt! Zu einem korrekten Algorithmus gehört auch eine genaue Beschreibung der zulässigen Eingabedaten und aller vorkommenden Variablen mit ihrem möglichen Wertebereich, damit die Eingabedaten (beim maschinellen Rechnen) nicht die Speicherkapazität überfordern.

Alle bisherigen Größen beim Euklidischen Algorithmus, die Quotienten q_i und die Reste r_i, waren durch die Eingabegrößen a und b beschränkt, so daß dort kein solches Problem auftrat. Das Wachstum der Koeffizienten x_i und y_i im erweiterten Algorithmus wird durch folgende Überlegung kontrolliert:

HILFSSATZ 4. Für die Koeffizienten x_i, y_i aus (4.2) gilt:

(i) $x_i > 0$, wenn i gerade, $x_i \le 0$, wenn i ungerade, und $|x_{i+1}| \ge |x_i|$ für $i = 1,\ldots,n$.

(ii) $y_i \le 0$, wenn i gerade, $y_i > 0$, wenn i ungerade, und $|y_{i+1}| \ge |y_i|$ für $i = 2,\ldots,n$.

(iii) $x_{i+1}y_i - x_iy_{i+1} = (-1)^{i+1}$ für $i = 0,\ldots,n$; insbesondere sind x_i und y_i stets teilerfremd für $i = 0,\ldots,n+1$.

(iv) $|x_i| \le |b|$, $|y_i| \le |a|$ für $i = 0,\ldots,n+1$ (falls $b \neq 0$ und $a \neq 0$).

Den Beweis will ich wieder nicht im Detail ausführen. Hinweise: (i), (ii) und die Formel in (iii) zeigt man durch Induktion; aus Satz 8 folgt dann $ggT(x_{i+1},y_{i+1}) \mid 1$, also = 1. Aus $0 = r_{n+1} = |a|x_{n+1} + |b|y_{n+1}$ folgt dann mit dem Lemma von Euklid $x_{n+1} \mid b$ und $y_{n+1} \mid a$.

Daß es wirklich wichtig ist, sich über den möglichen Wertebereich aller vorkommenden Variablen im klaren zu sein, will ich an einem Beispiel zeigen, wo trotz aller Korrektheitsbeteuerungen meinerseits ein möglicher Fehler verborgen liegt: Angenommen, wir setzen unsere Algorithmen in BASIC-Programme um. Viele BASIC-Dialekte lassen für Ganzzahlen nur 16 Bit zu, also (etwa) den mickrigen Bereich von -32768 bis +32767. Das ist zunächst nicht schlimm, denn ganze Zahlen können wir auch als Gleitkommazahlen behandeln, wo (meistens) mindestens 8 Stellen zulässig sind. Aber: Der Quotient $q = \left\lfloor \frac{a}{b} \right\rfloor$ im Divisionsschritt ist, egal ob wir den Befehl INT oder FIX verwenden, eine ganze Zahl und kann leicht einen Überlauf erzeugen - probieren Sie a = 84 327 839, b = 12 oder b = 84 327 799 ! Mein "Personal Computer" hat, obwohl ich dies aufgrund des Handbuchs nicht erwartet hätte, die richtigen Ergebnisse geliefert, d. h., q = INT(a/b) doch als Gleitkommazahl mit abgeschnittenen Nachkommastellen behandelt. Bei Verwendung der "ganzzahligen Division" q = a\b trat dagegen der Überlauf auf.

(4.6) Diese Anwendung des Euklidischen Algorithmus auf diophantische Gleichungen spricht übrigens nicht nur für die klassische Kettendivision. Auch die anderen Versionen lassen sich entsprechend erweitern. Hierfür verweise ich auf die abschließende Aufgabensammlung, ebenso für weitere Probleme im Zusammenhang mit dem Euklidischen Algorithmus für ganze Zahlen.

Zwei weitere wichtige Anwendungen des Euklidischen Algorithmus, auf die ich ebenfalls nicht näher eingehen kann, beruhen auf der Lösung linearer diophantischer Gleichungen:

- des chinesische Restproblem,
- die Division in endlichen Körpern (also "mod p").

Auch für lineare diophantische Gleichungen mit mehreren Unbekannten verweise ich auf die Problemsammlung.

AUFGABEN UND PROBLEME

I. Leichte Übungsaufgaben als Verständnistests.

1.) Bestimme den ggT für mehrere Zahlenpaare auf dem Taschenrechner
   a) mit der klassischen Kettendivision,
   b) nach der Empfehlung in (2.4),
   c) nach der Empfehlung 3 in (3.4)!
   Zahlenbeispiele: a = 544, b = 199;
   a = 2166, b = 6099;
   a = 40 902, b = 24 140;
   a = 10 083 798, b = 4 481 688.

2.) Führe die Beweise der Hilfssätze 1, 3 und 4 und der Bemerkung

$$G_n = \frac{(1+\sqrt{2})^n - (1-\sqrt{2})^n}{2\sqrt{2}}$$ zu Satz 5 aus!

3.) Übersetze die vorgestellten Algorithmen in Computerprogramme!

4.) Der Eintritt ins Museum kostet 3,60 DM für Erwachsene und 1,50 DM für Kinder. Am Abend sind 180,-- DM in der Kasse, und der Kassierer erinnert sich noch, daß zwar Kinder, aber auf jeden Fall mehr Erwachsene das Museum besichtigt haben. Wieviele Erwachsene und Kinder waren es?

5.) Ein Bauer kauft 100 Stück Vieh, und zwar Kälber zu 480 DM, Lämmer zu 200 DM, Ferkel zu 100 DM. Er kauft von jeder Sorte mindestens eins und zahlt 16 000 DM. Wieviele Tiere von jeder Sorte kauft er?

6.) Beweise ohne Verwendung des Euklidischen Algorithmus, daß es für $a,b \in \mathbb{Z}$ und $d = ggT(a,b)$ zwei Zahlen $x,y \in \mathbb{Z}$ gibt mit $ax+by=d$, indem du das minimale Element der Menge $\{ax+by \mid x,y \in \mathbb{Z}, ax+by>0\}$ betrachtest!

7.) Warum sind in der Tabelle für die Schrittzahlen in (1.5) die Spalten periodisch? Was kannst du über die Zeilen und die Diagonalen sagen? Was bedeutet ein Eintrag 1? 2?

II. Etwas schwerere Aufgaben, die zur Vertiefung dienen.

1.) a) Suche ein explizites Bildungsgesetz für die Schrittzahl-Tabelle in (1.5)! Anleitung: Baue eine Spalte aus den vorhergehenden auf! Was steht oberhalb, in, unterhalb der Diagonalen?
   b) Schreibe ein Programm zum Ausdruck der Tabelle, wenn möglich ein Stück weiter als im Text!

2.) Untersuche die entsprechend gebildeten Tabellen für die Minimalrest- und die binäre Version! Entdeckst du Regelmäßigkeiten? Welches sind die schlechtesten Fälle? Suche die expliziten Bildungsgesetze!

3.) Erweitere die Versionen 2, 3 und 4 des Euklidischen Algorithmus so, daß sie ebenfalls eine Darstellung des größten gemeinsamen Teilers als ganzzahlige Linearkombination der Eingabedaten liefern!

4.) Entwirf einen Algorithmus und schreibe ein Programm, um ganze Zahlen mod m zu dividieren! Genauer:
Sei $m \in \mathbb{N}$, $m \geq 2$. Seien $a, b \in \mathbb{Z}$. Falls b zu m teilerfremd ist, soll $c \in \mathbb{Z}$, $0 \leq c < m$, gefunden werden mit $a \equiv bc \pmod m$. Was kann man machen, wenn b mit m gemeinsame Teiler hat?

5.) Man kann die Versionen 1 und 2 des Euklidischen Algorithmus beschleunigen, indem man den Vertauschungsschritt EA4 wegläßt und statt dessen $a = r$ setzt. Dann muß man freilich das Programm verlängern und als nächsten Schritt b durch a dividieren. Führe das aus!

6.) Seien $a, b \in \mathbb{Z}$, $a \geq b > 0$.
   a) Sei r der Divisionsrest in $a = qb + r$, $0 \leq r < b$.
   Zeige: $\frac{b}{a} < \frac{5}{8}$ oder $\frac{r}{a} < (\frac{5}{8})^2$.
   b) Zeige für die klassische Divisionskette: $\frac{r_n}{r_o} < (\frac{5}{8})^n$ oder $\frac{r_{n-1}}{r_o} < (\frac{5}{8})^{n-1}$.
   c) Leite aus b) für die Schrittzahl des Euklidischen Algorithmus her:
   $L(a,b) < 1 + 4.90 \cdot \log b$.

7.) Gewinne eine analoge Abschätzung für die Minimalrestversion, indem du die Konstante $\frac{5}{8}$ aus Aufgabe 6 durch $\frac{3}{7}$ ersetzt!

8.) Baue den erweiterten Euklidischen Algorithmus aus zu einem Programm für die Lösung der diophantischen Gleichung $ax + by = c$.

9.) Wann ist die diophantische Gleichung $ax + by + cz = d$ lösbar? Entwirf ein Programm für die Lösung!

10.) Der Additionsalgorithmus von Daykin für ggT(a,b):

AA1. Lies a, b; setze $a := |a|$, $b := |b|$; falls $a < b$, vertausche a und b.

AA2. Falls $b = 0$, gib a aus; Ende.

AA3. Ermittle die Stellenzahl k von a; setze $s := 10^k - a$, $t := b$.

AA4. $r := s + t$; falls $r = 10^k$, gib t aus; Ende.

AA5. Falls $r < 10^k$, dann $s := r$, sonst $t := r - 10^k$; springe nach AA4.

a) Rechne diesen Algorithmus mit den Zahlenbeispielen aus Aufgabe I.1 durch!

b) Beweise die Korrektheit des Algorithmus!

Bemerkung: Mit Ausnahme der trivialen Subtraktionen $s = 10^k - a$ in AA3 (Zehnerkomplement) und $t = r - 10^k$ in AA5 (Abschneiden) wird bei der Ausführung des Algorithmus nur addiert. Statt der Basis 10 kann man jede andere nehmen, etwa 2. Dieser Algorithmus ist anscheinend bisher nicht analysiert worden.

Quelle:

D. E. DAYKIN: An addition algorithm for greatest common divisor. Fibonacci Quart. 8(1970), 347 - 349.

III. Projekte.

1.) Gewinne empirische Daten über das mittlere und maximale Laufzeit- Verhalten des gemischten Euklidischen Algorithmus! Finde den schlechtesten Fall, also die kleinsten Eingabedaten, die eine bestimmte Schrittzahl erfordern! Vielleicht ergibt die Schrittzahl-Tabelle einen Anhaltspunkt? Finde allgemeine Aussagen über den schlechtesten Fall wie bei der klassischen oder Minimalrest-Version!

2.) Vergleiche empirisch das Laufzeitverhalten der Versionen 1 bis 4 auf einem konkreten Computer oder für einen Taschenrechner!
Hinweis: Anstatt Zeiten zu messen (die durch die Arbeitsweise von Interpreter/Compiler/Assembler verfälscht sind), kann man auch Schrittzähler in ein Programm einbauen; auf diese Weise kann man auch die Arbeitsweise eines Taschenrechners auf dem Computer simulieren.

3.) Um den ggT von 3 oder mehr Zahlen $a_1, \ldots, a_n$ zu finden, sind verschiedene Vorgehensweisen denkbar:

a) Man berechnet $d_1 = ggT(a_1, a_2)$. Falls $d_1 = 1$, ist man fertig. Sonst $d_2 = ggT(d_1, a_3)$ usw.

b) Man dividiert jeweils die größte Zahl, also $\max a_i$, durch die kleinste $\min \{a_i \mid a_i \neq 0\}$ und ersetzt $\max a_i$ durch den Divisionsrest. - Die Suche nach max und min wird sehr einfach, wenn man ganz am Anfang die a_i der Größe noch sortiert.
Diskutiere und vergleiche die beiden (und mögliche weitere?) Methoden!
(Der schlechteste Fall für a) ist in [3; 4.5.3, Exercise 36] diskutiert.)

4.) Analysiere den Additionsalgorithmus von Daykin (Aufgabe II.10).

LITERATUR

Viele Anregungen zur Bearbeitung mathematischer Aufgaben auf dem Computer gibt das Buch

[1] Arthur ENGEL: Elementarmathematik vom algorithmischen Standpunkt. Klett Studienbücher Mathematik, Stuttgart 1977.

Auch der Euklidische Algorithmus wird recht ausführlich behandelt. Das Buch ist durchweg als Schulbuch geeignet. Dies kann man von dem folgenden Werk nicht gerade behaupten:

Donald E. KNUTH: The Art of Computer Programming.
[2] Vol.1: Fundamental Algorithms. Addison-Wesley, Reading Mass. 1968, 1973.
[3] Vol.2: Seminumerical Algorithms. Addison-Wesley, Reading Mass. 1969, 1981.

Diese Bücher sind allerdings von grundlegender Bedeutung für Mathematiker, die etwas über Informatik lernen wollen. Sie enthalten viele Anregungen zum Weiterarbeiten und haben auch diesen Vortrag angeregt. Für die zahlentheoretischen Aspekte des Euklidischen Algorithmus kann man (fast) jedes Zahlentheorie-Lehrbuch heranziehen. Besonders empfehlenswert sind die Kapitel 2 und 13 von

[4] David M. BURTON: Elementary Number Theory.
Allyn and Bacon, Boston 1976;

oder §1, 11 und 12 von

[5] Heinz LÜNEBURG: Vorlesungen über Zahlentheorie.
Birkhäuser, Basel 1979.

Für die Berechnung der Fibonacci-Zahlen F_n und der Folge G_n kann man nachsehen in

[6] Klaus POMMERENING: Lineare Rekursionsfolgen, Matrizen und Eigenwerte.
In: Matrizenrechnung, Tagungsvorträge zur Lehrerfortbildung.
Alef 4, Mainz 1984, 79 - 101.

Der Standpunkt der Deutschen Mathematiker-Vereinigung zur Informatik an Schulen ist dargestellt in:

Stellungnahme zum Informatikunterricht an Gymnasien.
Deutsche Mathematiker-Vereinigung e. V., Frühjahr 1983.
Beilage zum Jahresbericht der DMV (1983).