Der Erweiterte Euklidische Algorithmus

Sind $a, b \in \mathbb{N}$ gegeben, so betrachten wir „Linearkombinationen“ der Form $xa+yb$ mit $x, y \in \mathbb{Z}$.
Ist d ein gemeinsamer Teiler von a und b, so teilt d auch jede dieser Linearkombinationen. Es erhebt sich die Frage, ob man eine Linearkombination finden kann, so daß sogar

ggT(a, b)=$xa+yb$

Mit Hilfe des unten spezifizierten „Erweiterten Euklidischen Algorithmus“ ist dies möglich.

Anwendungen:

Inversenbildung in den multiplikativen Gruppen $\mathbb{Z}_n^*$
Sei $a \in \mathbb{Z}_n^*$. Das heißt $1 \leq a \leq n$ und ggT(n,a)=1 . Also lassen sich $x, y \in \mathbb{Z}$ finden mit $xa+yn=1$.
Interpretiert man dies als Gleichung in $\mathbb{Z}_n$, so ist $yn=0$ und daher $xa=1$. Also ist $x=a^{-1}$.

Beispiel:
n=43, a=17: $2 \cdot 43+(-5) \cdot 17=1$. Also ist -5=38 das Inverse von 17.
Man sieht, daß einer der beiden Koeffizienten negativ und einer positiv sein muß. In diesem Fall ist der Koeffizient von 17, also -5, negativ. Das ist aber kein Problem: auch jeder negativen Zahl entspricht ein eindeutig bestimmtes Element von $\mathbb{Z}_n$, hier -5=38.

Primzahlen:
Man erinnere sich an den Begriff der Primzahl: Eine natürliche Zahl n heißt Primzahl, wenn es keine Produktzerlegung $n=ab$ gibt, wobei beide Faktoren von 1 verschiedene natürliche Zahlen sind.

Eine wesentliche Eigenschaft von Primzahlen ist die folgende:
Ist p eine Primzahl und ab ein Produkt natürlicher Zahlen, welches von p geteilt wird, so teilt p mindestens einen der Faktoren. Diese Aussage können wir jetzt beweisen:

Sei d:=ggT(a,p): Dann ist d ein Teiler von p, d.h. es gibt ein $k \in \mathbb{N}$ mit $kd=p$. Da p Primzahl ist, muß gelten: k=1 oder d=1.

Ist k=1, so ist $d=p$, also p ein Teiler von a, und wir sind fertig.
Ist d=1, so läßt sich die Gleichung $xa+yp$=1 mit $x, y \in \mathbb{Z}$ lösen. Es folgt $xab+ypb=b$. Es war vorausgesetzt, daß p ein Teiler von ab ist. Also teilt p jeden Summanden auf der linken Seite der Gleichung, also teilt p auch b, und wir sind fertig!

Erweiterter Euklidischer Algorithmus:
Zu gegebenen $a, b \in \mathbb{N}$ sollen $x, y \in \mathbb{Z}$ gefunden werden, so daß ggT(a, b)=$xa+yb$.

Man erinnere sich an den Euklidischen Algorithmus. Wir nehmen oBdA $a>b$ an, setzen $r_0 := a$, $r_1 = b$ und führen sukzessiv Divisionen mit Rest durch:

$r_0 = q_1 r_1 + r_2$, $r_1 = q_2 r_2 + r_3$, … , $r_{k-1} = q_k r_k + r_{k+1}$, ...

wobei jeweils $0 \leq r_{k+1} < r_k$. Weil die Folge der r_k mit jedem Schritt fällt, wird irgendwann zwangsläufig $r_{k+1}=0$, und dann ist r_k =ggT(a,b).

Man kann die obigen Rekursionsgleichungen auch in die Form bringen:

$r_2=-q_1 r_1+r_0$, $r_3=-q_2 r_2+r_1$, … , $r_{k+1}=-q_k r_k+r_{k-1}$, …

In jedem Schritt entsteht also ein neuer Folgenterm r_{k+1} und ein neuer Quotient q_k .

Ausgehend von den Anfangstermen $s_0=1$, $s_1=0$, $t_0=0$, $t_1=1$ bilden wir jetzt mit denselben Rekursionsschritten unter Benutzung derselben Quotienten q_k zwei weitere Folgen:

$s_2=-q_1 s_1+s_0$, $s_3=-q_2 s_2+s_1$, … , $s_{k+1}=-q_k s_k+s_{k-1}$, …
$t_2=-q_1 t_1+t_0$, $t_3=-q_2 t_2+t_1$, … , $t_{k+1}=-q_k t_k+t_{k-1}$, …

Wir brechen auch hier die Rekursionen ab, sobald $r_{k+1}=0$.

Solange $r_k>0$ gilt die Gleichung: $s_k r_0+t_k r_1=r_k$. (∗)

Dies werden wir gleich beweisen. Im letzten Schritt , also wenn r_k =ggT(a,b)>0 , $r_{k+1}=0$, bedeutet dies

$$s_k a+t_k b=\text{ggT}(a,b)$$

d.h. wir haben dann unsere gesuchte Linearkombination gefunden!

Induktionsbeweis für (∗)
Auf der Basis der Anfangsbedingungen hat man den Induktionsanfang
$r_0=1\cdot r_0+0\cdot r_1=s_0 r_0+t_0 r_1$, $r_1=0\cdot r_0+1\cdot r_1=s_1 r_0+t_1 r_1$

Wir benutzen die Induktionsvoraussetzungen $s_k r_0+t_k r_1=r_k$ und $s_{k-1} r_0+t_{k-1} r_1=r_{k-1}$.

Anschließend rechnen wir leicht die Induktionsbehauptung nach:
$s_{k+1} r_0+t_{k+1} r_1=\left(-q_k s_k+s_{k-1}\right) r_0+\left(-q_k t_k+t_{k-1}\right) r_1=-q_k\left(s_k r_0+t_k r_1\right)+\left(s_{k-1} r_0+t_{k-1} r_1\right)=$
$-q_k r_k+r_{k-1}=r_{k+1}$

Fertig!

Folgenden Pari-Code für obige Rekursion können Sie direkt mit Copy/Paste in ein Pari-Fenster hineinkopieren; beachten Sie, wie gering der Speicheraufwand ist:

```
egcd(a,b)=local(r0=a,r1=b,r2,s0=1,s1=0,s2,t0=0,t1=1,t2);\
      while((r2=r0%r1),\
            q=r0\r1;s2=-q*s1+s0;t2=-q*t1+t0;\
            r0=r1;r1=r2;s0=s1;s1=s2;t0=t1;t1=t2;\
      );\
\
return([r1,s1,t1])
```

Im Ergebnisvektor `[r1,s1,t1]` ist jetzt `r1=ggT(a,b)` , und `s1` und `t1` erfüllen die Gleichung `r1=s1*a+t1*b.`
Man erhält für unser obiges Beispiel mit dem Input `egcd(43,17)` das Resultat `[1, 2,-5]` : dies entspricht der Gleichung $1=2\cdot 43+(-5)\cdot 17$.
Probieren Sie den Befehl `v=egcd(a=random(10^1000),b=random(10^1000))` !
Der Algorithmus ist sehr gnädig: ihm ist egal, und er arbeitet korrekt, wenn $a\leqslant b$ oder sogar wenn a oder b negativ sind! Überlegen Sie, warum!