Euklid. Algorithmus und multiplikatives Inverses in $\mathbb{Z}_p$

Für jedes $n \in \mathbb{Z}_p$ mit $0 < n < p$ ist

$$ggT(n, p) = 1$$

wenn p eine Primzahl ist.

Der euklid. Algorithmus wird also den ggT 1 liefern:

Bsp. 1

$$\begin{aligned} 17 &= 3 \cdot 5 + 2 \\ 5 &= 2 \cdot 2 + \textcircled{1} \\ 2 &= 2 \cdot \textcircled{1} + \underline{0} \quad \text{stop} \end{aligned}$$

ggT, oberhalb und links von 0

also:

$$\begin{aligned} 1 &= 5 - 2\cdot 2 \\ &= 5 - 2\cdot(17 - 3\cdot 5) = \\ &= 5 + 6\cdot 5 - 2\cdot 17 = \\ &= 7\cdot 5 - 2 \cdot 17 \overset{!}{=} 7\cdot 5 \mod 17 \; ! \end{aligned}$$

also ist $7 = 5^{-1} \mod 17$!!

Bsp. 2 39^{-1} in $\mathbb{Z}_{53}$?? 53 ist prim !

isPrime(53) ↵

$$\begin{aligned} 53 &= 1\cdot 39 + 14 \\ 39 &= 2\cdot 14 + 11 \\ 14 &= 1\cdot 11 + 3 \\ 11 &= 3\cdot 3 + 2 \\ 3 &= 1\cdot 2 + \textcircled{1} \\ 2 &= 2\cdot\textcircled{1} + \underline{0} \quad \text{stop} \end{aligned}$$

$\Longrightarrow$

$$\begin{aligned} 1 &= 3 - 1\cdot 2 \\ &= 3 - 1\cdot(11 - 3\cdot 3) = \\ &= -1\cdot 11 + 4\cdot 3 = \\ &= -1\cdot 11 + 4\cdot(14 - 1\cdot 11) \\ &= -5\cdot 11 + 4\cdot 14 = \\ &= -5\cdot(39 - 2\cdot 14) + 4\cdot 14 = \end{aligned}$$

$$\begin{aligned} &= -5\cdot 39 + 14\cdot 14 = \\ &= -5\cdot 39 + 14\cdot(53 - 1\cdot 39) = \\ &= -19\cdot 39 + 14\cdot 53 \overset{!}{=} -19\cdot 39 \\ &\overset{!}{=} \underbrace{34\cdot 39 \overset{!}{=} 1} \end{aligned}$$

Kontrolle:

$34\cdot 39 = 1326 = 25\cdot 53 + 1$ ✓

Übersicht über die folgenden Blätter :

16a	Detailliertes Beispiel für die Wandtafel
16b	Verallgemeinerung von 16a , Überlegungen für die Implementierung mit Excel
16c	Implementierung mit Delphi 6.0 oder TurboDelphi
16d	TI-Taschenrechner : Problem mit for-Schleife
16e	TI-TR : Die Hilfsfunktion div(a, b)
16f	TI-TR : Erweiterter euklid. Algorithmus , → Funktion euklidin(a, p)
16g	wie 16f , für den neuen TI-nSpire
16h	~~Zwei~~ Eine weitere, kleine Funktionen für die TI-Rechner : nextprim(n)
16i	Syntax-Vergleich "Delphi-Pascal" ~ "TI-Bascal"

Variablenname !

$$\begin{array}{ccccccc|ccccccccc}
a & & b & & c & & d & e & & p & & f & & n & & c \\
 & & & & & & & \boxed{0} & \cdot & 751 & + & \boxed{1} & \cdot & 81 & = & 81 \\
p & & & & n & & & & & & & & & & & \\
\boxed{751} & = & 9 & \cdot & \boxed{81} & + & 22 & \boxed{1} & \cdot & 751 & + & (-9) & \cdot & 81 & = & 22 \\
81 & = & 3 & \cdot & 22 & + & 15 & (-3) & \cdot & 751 & + & 28 & \cdot & 81 & = & 15 \\
22 & = & 1 & \cdot & 15 & + & 7 & 4 & \cdot & 751 & + & (-37) & \cdot & 81 & = & 7 \\
15 & = & 2 & \cdot & 7 & + & 1 & (-11) & \cdot & 751 & + & \boxed{102} & \cdot & 81 & = & 1 \\
7 & = & 7 & \cdot & \boxed{1} & + & \boxed{0} & & & & & & & & &
\end{array}$$

Ergänzung ...

⇐ Start, Vorbereitung while

◯ ~ Input !

⇐ Ende der while-Schleife

falls 1 existiert das Inverse !

Ende der while-Schleife

das Inverse von 81 mod 751 (falls c = 1)

$$22 = 1 \cdot 751 - 9 \cdot 81$$

$$15 = 81 - 3 \cdot 22 = 81 - 3 \cdot (1 \cdot 751 - 9 \cdot 81) = (-3 \cdot 1 + 0) \cdot 751 + (1 + (-3)(-9)) \cdot 81$$
$$= -3 \cdot 751 + 28 \cdot 81$$

$$7 = \underset{\sim}{22} - 1 \cdot \underset{\sim}{15} = (1 \cdot 751 - 9 \cdot 81) - 1 \cdot (-3 \cdot 751 + 28 \cdot 81) =$$
$$= [1 + (-1) \cdot (-3)] \cdot 751 + [-9 + (-1) \cdot 28] \cdot 81 = 4 \cdot 751 - 37 \cdot 81$$

$$1 = 15 - 2 \cdot 7 = (-3 \cdot 751 + 28 \cdot 81) - 2 \cdot (4 \cdot 751 - 37 \cdot 81) =$$
$$= [-3 + (-2) \cdot 4] \cdot 751 + [28 + (-2) \cdot (-37)] \cdot 81 = -11 \cdot 751 + \boxed{102} \cdot 81$$

man braucht immer die letzten beiden Zeilen !!

Wie gewinnt man den neuen Eintrag für e ? und für f ?

$$e_{neu} = e(\text{vor-vorher}) + (-b)\cdot(e_{vorher})$$

$$f_{neu} = f_{\text{vor-vorher}} + (-b)\cdot f_{vorher} \quad !$$

Wir müssen daher die 1. Zeile separat einfüllen.

Bei der 2. Zeile kommt es richtig, wenn wir zusätzlich für $e_{vorvorher} = 0$ und $f_{vorvorher} = 1$ setzen !!, also eine 'nullte' halbe Zeile ergänzen !

Reihenfolge: e, f, a, c, b, d !!

zuerst

nachher !

16c

Delphi - Pascal
oder
Turbo - Delphi

```
procedure TForm1.Button1Click(Sender: TObject);
var n,p,a,b,c,d,e0,e1,e2,f0,f1,f2 : Integer ;
   s : string ;
begin
  n := StrToInt(Edit1.Text);
  p := StrToInt(Edit2.Text);
  if p = 0 then
    begin
      s := 'Geben Sie doch einen vernünftigen Wert ein für p !';
      Memo1.Text := s ;
      exit
    end ;

  if n = 0 then
    begin
      s := '0  hat kein Inverses, mein Lieber !';
      Memo1.Text := s ;
      exit
    end ;

  if (n = 1) or (n = -1) then
    begin
      s := 'Das Inverse von 1 ist 1 , dasjenige von -1 ist -1 ...'
      Memo1.Text := s ;
      exit
    end ;

  a := n ;
  c := p mod n ;

  if c = 0 then
    begin
      s := 'Das Inverse existiert nicht.'+chr(10)+chr(13);
      s := s + 'p ist durch n teilbar !';
      Memo1.Text := s ;
      exit
    end ;

  b := a div c ;
  d := a mod c ;
  e0 := 0 ;
  e1 := 1 ;
  f0 := 1 ;
  f1 := - (p div n) ;

  while not(d = 0) do
    begin
      e2 := e0 - b*e1 ;
      e0 := e1 ;
      e1 := e2 ;
      f2 := f0 - b*f1 ;
      f0 := f1 ;
      f1 := f2 ;

      a := c ;
      c := d ;
      b := a div c ;
      d := a mod c
    end ;

  if c = 1 then    // das Inverse existiert !
    begin
      while f1 < 0 do f1 := f1 + p ;
      f1 := f1 mod p ;
      s := 'Das Inverse von n modulo p existiert.' ;
      s := s + chr(13) + chr(10) + chr(13) + chr(10) ;
      s := s + 'Es beträgt     ' + IntToStr(f1) ;
      Memo1.Text := s ;
    end
  else
    begin
      s := 'Das Inverse von n modulo p existiert nicht.' ;
      s := s + chr(13) + chr(10) + chr(13) + chr(10) ;
      s := s + 'Der ggT der beiden Zahlen ist  ' + IntToStr(c) ;
      Memo1.Text := s ;
    end ;

end;
```

OL4 2008 / Gub

Erweiterter Euklidischer Algorithmus

"Probieren in for-Schleife"

```
(n,p)
Func
Local i
mod(n,p)→n
For i,1,p-1,1
  If mod(i*n,p)=1 Then
     Exit
  EndIf
EndFor
Return mod(i,p)
EndFunc
```

Funktion modinver(n, p)

Input : n Integer , $p \in \mathbb{N}$, man rechnet in $\mathbb{Z}_p$

Output : n^{-1} mit $n^{-1} \cdot n \equiv 1 \mod p$, falls n^{-1} existiert

0 sonst

Wird schnell langsamer für größere Werte von p. Schon für p = 751 ist euklidin(n, p) viel schneller!

```
(a,b)
Func
Return (a-mod(a,b))/b
EndFunc
```

TI-Rechner!

Funktion div(a, b)

Input : a, b zwei Integer

Output : a div b

(genauer : $[a - (a \bmod b)]$ div b
$= [a - \mathrm{mod}(a,b)] / b$)

Wichtige Hilfsfunktion beim Rechnen in $\mathbb{Z}_p$.
Wird benutzt von der Funktion euklidin(…, …)

mod(a, b) ist vordefiniert und liefert a modulo b
also mod(15, 4) = 3

```
(n,p)
Func
Local a,b,c,d,e0,e1,e2,f0,f1,f2
mod(n,p)→n
If n=1 Then
  Return 1
EndIf
n→a
mod(p,n)→c
div(a,c)→b
mod(a,c)→d
0→e0
1→e1
1→f0
-div(p,n)→f1
While d>0
   e0-b*e1→e2
   e1→e0
   e2→e1
   f0-b*f1→f2
   f1→f0
   f2→f1
   c→a
   d→c
   div(a,c)→b
   mod(a,c)→d
EndWhile
If c=1 Then
  Return mod(f1,p)
Else
  Return 0
EndIf
EndFunc
```

Function euklidin (n, p)

Input: n Integer, p ebenfalls (meist eine Primzahl)

Output: 0, falls n kein Inverses besitzt in $\mathbb{Z}_p$
n^{-1} sonst

Berechnungsmethode: Erweiterter euklid. Algorithmus, sehr schnell auch
bei großen Primzahlen p !

Erweiterter Euklidischer Algorithmus für TI-nspire Okt 2008 / Gub

```
Define LibPub euklidinver(n,p)=
Func
:Local a,b,c,d,e0,e1,e2,f0,f1,f2
:If n*p=0 Then
:  Return 0
:EndIf
:mod(n,p)→n
:If n=1 Then
:  Return 1
:EndIf
:If n=p-1 Then
:  Return ~~-1~~ p-1
:EndIf
:
:n→a
:mod(p,n)→c
:If c=0 Then
:  Return 0
:EndIf
:div(a,c)→b
:mod(a,c)→d
:0→e0
:1→e1
:1→f0
:-1*div(p,n)→f1
:
:While not (d=0)
:  e0-b*e1→e2
:  e1→e0
:  e2→e1
:  f0-b*f1→f2
:  f1→f0
:  f2→f1
:  c→a
:  d→c
:  div(a,c)→b
:  mod(a,c)→d
:EndWhile
:
:If c=1 Then
:  While f1<0
:    f1+p→f1
:  EndWhile
:  mod(f1,p)→f1
:  Return f1
:Else
:  Return 0
:EndIf
:EndFunc
```

16 h

```
: nextPrim (n)
: Func
: While  not  isPrime(n)
:      n+1  →  n
: EndWhile
: Return  n
: EndFunc
```

```
: nextPrim (n)
: Func
: Loop
:    n+1  →  n
:    If  isPrime (n)  Then
:        Return  n
:    EndIF
: EndLoop
: End Func
```

TI-89 / voyage 200 / TI nSpire

Task	in Delphi - Pascal	in TI - Basic
Wertzuweisung :	a := b - 2*c ;	b - 2*c → a
Funktionswert zurück-geben :	result := c ; (Funktion wird weiter bearbeitet!)	Return c (geht zu EndFunc !)
Funktion vorzeitig verlassen	exit ;	Return oder Return c
Schleife (while, for, repeat) vorzeitig verlassen	break ;	exit

while - Schleife	while ...... do
	begin
	⋮
	end ;
		While ......
		⋮
		EndWhile

Ganzzahl - Division	c := a ~~mod~~ div b ;	div (a, b) → c
Rest bei der Ganzzahl-Division	r := a mod b ;	mod (a, b) → r
Startwert für Zufallszahl-Generator setzen	RandSeed := 1234 ;	RandSeed 1234
Ganze Zufallszahl zwischen in {1, 2, 3, ..., 10}	z := 1 + random (10) ;	randInt(1, 10) → z
Ist n eine Primzahl?	selber programmieren !!	isPrime (n) → true oder false