4 Modulare Arithmetik

4.1 Restklassenringe

Die ganzen Zahlen zusammen mit den Operationen Addition und Multiplikation $(\mathbb{Z}, +, \cdot)$ bilden bekanntlich einen kommutativen Ring mit 1. Wir wollen nun diese algebraische Struktur auf endliche Teilmengen von $\mathbb{Z}$ übertragen. Dies geschieht durch Identifikation von Elementen in $\mathbb{Z}$, die in einer gemeinsamen arithmetischen Folge liegen.

Definition 4.1 Sei $p \in \mathbb{N}_{\geq 2}$. Zwei ganze Zahlen $a,b \in \mathbb{Z}$ heißen *kongruent modulo* p, in Zeichen $a \equiv b \pmod{p}$, falls $p \mid b - a$ ist. Daher ist $a \equiv b \pmod{p}$ genau dann, wenn a und b bei der Division durch p denselben Rest haben:

$$\text{rest}(a,p) = \text{rest}(b,p)\ . \tag{4.1}$$

Kongruenz ist eine *Äquivalenzrelation* (s. Lemma 4.2), welche eine *Klasseneinteilung* in $\mathbb{Z}$ liefert. Die Klassen von zueinander kongruenten Zahlen sind die *arithmetischen Menge* $[a]_p := \{a + k \cdot p \mid k \in \mathbb{Z}\}$. Jede Äquivalenzklasse besitzt folglich genau einen Vertreter im *Segment* $0,1,\ldots,p-1$, nämlich $\text{rest}(a,p)$. Daher werden die Äquivalenzklassen $[a]_p$ *Restklassen* genannt. Es gibt genau p verschiedene Restklassen zum *Modul* p, welche in eindeutiger Weise durch die Reste (4.1) beschrieben werden. Somit kann man die Menge der Restklassen $\mathbb{Z}_p := \{[0]_p,[1]_p,\ldots,[p-1]_p\}$ mit der Menge $\{0,1,2,\ldots,p-1\} \subset \mathbb{Z}$ der entsprechenden Reste identifizieren. $\triangle$

Wir beweisen folgende Eigenschaften der modularen Kongruenz:

Lemma 4.2 (Rechenregeln des modularen Rechnens) Seien $a,b,c,a',b',n \in \mathbb{Z}$. Dann gelten

(a) **(Reflexivität)** $a \equiv a \pmod{p}$;

(b) **(Symmetrie)** $a \equiv b \pmod{p} \ \Rightarrow\ b \equiv a \pmod{p}$;

(c) **(Transitivität)** $a \equiv b \pmod{p}$ und $b \equiv c \pmod{p} \ \Rightarrow\ a \equiv c \pmod{p}$;

(d) **(Verträglichkeit der Skalarmultiplikation)** $a \equiv b \pmod{p}$ und $n \in \mathbb{Z} \Rightarrow n \cdot a \equiv n \cdot b \pmod{p}$;

(e) **(Verträglichkeit der Addition)** $a \equiv a' \pmod{p}$ und $b \equiv b' \pmod{p} \Rightarrow a + b \equiv a' + b' \pmod{p}$;

(f) **(Verträglichkeit der Multiplikation)** $a \equiv a' \pmod{p}$ und $b \equiv b' \pmod{p} \Rightarrow a \cdot b \equiv a' \cdot b' \pmod{p}$.

Beweis: (a): $p \mid a - a = 0$.
(b): $p \mid b - a \Rightarrow b - a = k \cdot p$ für $k \in \mathbb{Z}$. Also ist $a - b = -k \cdot p \Rightarrow p \mid a - b$.
(c): Nach Voraussetzung haben wir $b - a = k \cdot p$ und $c - b = k' \cdot p$ für $k,k' \in \mathbb{Z}$. Daraus folgt aber $c - a = (c - b) + (b - a) = (k + k') \cdot p$.
(a)–(c) zeigen, daß ≡ eine Äquivalenzrelation ist.
(d): Wir haben $b - a = k \cdot p$. Also ist $n \cdot b - n \cdot a = n \cdot (b - a) = n \cdot k \cdot p$.
(e): Aus den Voraussetzungen folgt $a' - a = k \cdot p$ und $b' - b = k' \cdot p$ für $k,k' \in \mathbb{Z}$. Daraus folgt $(a' + b') - (a + b) = (a' - a) + (b' - b) = (k + k') \cdot p$.
(f): Mit denselben Bezeichnungen wie in (e) folgt nun $a' = a + k \cdot p$ und $b' = b + k' \cdot p$ und folglich $a' \cdot b' = (a + k \cdot p) \cdot (b + k' \cdot p) = a \cdot b + p \cdot (k + k' + p)$. Also ist die Differenz $a' \cdot b' - a \cdot b$ ein Vielfaches von p, und der Beweis ist erbracht. □

Sitzung 4.1 Wählt man bei den Verträglichkeitsaussagen (e) und (f) aus Satz 4.2 $a' = \mathrm{mod}(a,p)$ und $b' = \mathrm{mod}(b,p)$, so ergeben sich die Gleichungen

$$\mathrm{mod}(a + b,p) = \mathrm{mod}(\mathrm{mod}(a,p) + \mathrm{mod}(b,p),p) \tag{4.2}$$

und

$$\mathrm{mod}(a \cdot b,p) = \mathrm{mod}(\mathrm{mod}(a,p) \cdot \mathrm{mod}(b,p),p)\ . \tag{4.3}$$

Wir testen diese Beziehungen mit *Mathematica*. Hierzu wählen wir zufällige Zahlen a,b und p:

```
In[1]:= a = Random[Integer,{1,10^20}]
        b = Random[Integer,{1,10^20}]
        p = Random[Integer,{1,10^10}]
Out[1]= 58517991276334182825
Out[1]= 73319430447933461004
Out[1]= 9683750991
```

und berechnen den Divisionsrest der Summe $a + b$:

```
In[2]:= Mod[a + b,p]
Out[2]= 1255945335
```

und die Summe der Divisionsreste von a und b:

```
In[3]:= Mod[a,p] + Mod[b,p]
Out[3]= 10939696326
```

Dieser ist i.a. zu groß und muß nochmals modulo p reduziert werden:

```
In[4]:= Mod[Mod[a,p] + Mod[b,p],p]
Out[4]= 1255945335
```

Eine ähnliche Rechnung ergibt sich für das Produkt:

```
In[5]:= Mod[a * b,p]
Out[5]= 7175447256
```

```
In[6]:= Mod[a,p] * Mod[b,p]
Out[6]= 28687297362580797669
```

Dies ist viel zu groß, aber:

```
In[7]:= Mod[Mod[a,p] * Mod[b,p],p]
Out[7]= 7175447256
```

Wir werden diese Eigenschaften später geschickt ausnutzen. □

Sei $p \in \mathbb{N}_{\geq 2}$ gegeben. Unter Verwendung der Abbildung

$$\begin{array}{rccc} \varphi: & \mathbb{Z} & \to & \mathbb{Z}_p \\ & a & \mapsto & \varphi(a) := [a]_p \end{array}$$

erklären wir in $\mathbb{Z}_p$ eine Addition $\oplus$ und eine Multiplikation $\otimes$ derart, daß φ ein Ringhomomorphismus wird:

$$\begin{array}{lcl} [a]_p \oplus [b]_p & := & \varphi(a+b) = [a+b]_p \\ & \text{und} & \\ [a]_p \otimes [b]_p & := & \varphi(a \cdot b) = [a \cdot b]_p \,. \end{array} \tag{4.4}$$

Dadurch wird $(\mathbb{Z}_p, \oplus, \otimes)$ wieder eine algebraische Struktur. Es zeigt sich, daß diese von $\mathbb{Z}$ die Ringstruktur erbt.

Satz 4.3 $(\mathbb{Z}_p, \oplus, \otimes)$ ist ein kommutativer Ring mit 1.

Beweis: Wir zeigen zunächst, daß die Operationen $\oplus$ und $\otimes$ *wohldefiniert* sind, d. h., daß das Ergebnis von $\oplus$ und $\otimes$ *unabhängig* von der Wahl der Repräsentanten der Restklasse ist.

Seien also zwei verschiedene Vertreter a und a' der Restklasse $[a]_p$ sowie zwei verschiedene Vertreter b und b' der Restklasse $[b]_p$ gegeben. Dann gelten aber auf Grund der Verträglichkeitsbedingungen (Lemma 4.2 (e)–(f)) die Beziehungen

$$[a+b]_p = [a'+b']_p \qquad \text{und} \qquad [a\cdot b]_p = [a'\cdot b']_p \ .$$

Dies zeigt die Unabhängigkeit von den Repräsentanten.

Daß sich die Ringeigenschaften von $\mathbb{Z}$ auf $\mathbb{Z}_p$ übertragen, sieht man nun leicht ein. Hierbei entspricht dem neutralen Element $0 \in \mathbb{Z}$ bzgl. der Addition die Nullklasse $[0]_p \in \mathbb{Z}_p$ und dem neutralen Element $1 \in \mathbb{Z}$ bzgl. der Multiplikation die Einsklasse $[1]_p \in \mathbb{Z}_p$. Für das Inverse bzgl. der Addition gilt

$$-[a]_p := [-a]_p = [p-a]_p \ ,$$

und dies ist unabhängig vom Repräsentanten wegen Lemma 4.2 (d), $n = -1$. □

Aus Bequemlichkeit[1] schreiben wir in Zukunft die Restklassen kurz als $a = [a]_p$, d. h. $\mathbb{Z}_p = \{0,1,\ldots,p-1\} \subset \mathbb{Z}$. Weiterhin schreiben wir wieder + und · für die Operationen $\oplus$ und $\otimes$ in $\mathbb{Z}_p$.[2]

Wir sehen uns nun die Additions- und Multiplikationstafeln in $\mathbb{Z}_p$ etwas genauer an. Für $p = 6$ und $p = 7$ sehen diese wie folgt aus:

$(\mathbb{Z}_6,+)$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

$(\mathbb{Z}_6,\cdot)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

bzw.

$(\mathbb{Z}_7,+)$	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

$(\mathbb{Z}_7,\cdot)$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

.

[1] und weil es so üblich ist

[2] In diesem Zusammenhang ist dann eine Gleichung wie $-3 = 4$ keineswegs falsch, wenn sie sich auf $\mathbb{Z}_7$ bezieht.

Es stellt sich nun die Frage, unter welchen Bedingungen ein Element $a \in \mathbb{Z}_p$ eine Einheit ist, d. h., unter welchen Umständen man die Existenz eines multiplikativen Inversen garantieren kann. Findet man für alle $a \in \mathbb{Z}_p$ ein Inverses, so wird $\mathbb{Z}_p$ ein *Körper.* Aus den obigen Operationstafeln sieht man sofort, daß $\mathbb{Z}_6$ *kein* Körper ist. Bei der Multiplikation mit 2 (Zeile 2) kommen nämlich nur die Elemente 0,2 bzw. 4 als Ergebnisse vor, so daß 2 kein Inverses haben kann, denn dazu müßten wir ja ein $a \in \mathbb{Z}_p$ finden mit $2 \cdot a = 1$. Dasselbe Argument gilt für jeden Teiler von p, wenn p zusammengesetzt ist. Weiterhin beweist die Gleichung $2 \cdot 3 = 0$ in $\mathbb{Z}_6$, daß es in diesem Ring *Nullteiler* gibt.

Es zeigt sich aber, daß für Primzahlen p immer ein Körper vorliegt. Es gilt nämlich

Satz 4.4 Ein Element $a \in \mathbb{Z}_p$ ist genau dann eine Einheit, falls $\gcd(a,p) = 1$ ist. Insbesondere: $(\mathbb{Z}_p, \oplus, \otimes)$ ist genau dann ein Körper, falls p eine Primzahl ist.

Beweis: Sei $p \in \mathbb{N}_{\geq 2}$ und $a \in \mathbb{Z}_p$. Falls a eine Einheit ist, gibt es also ein $b \in \mathbb{Z}$ mit $a \cdot b \equiv 1 \pmod{p}$. Dies bedeutet, daß es ein $k \in \mathbb{Z}$ gibt mit $a \cdot b = 1 + k \cdot p$. Aus dieser Gleichung folgt aber, daß der größte gemeinsame Teiler von a und p gleich 1 ist.

Ist nun aber $\gcd(a,p) = 1$, so gibt es nach dem erweiterten Euklidischen Algorithmus $s,t \in \mathbb{Z}$ mit $s \cdot a + t \cdot p = 1$. Modulo p liest sich dies $s \cdot a \equiv 1 \pmod{p}$, und folglich ist s ein Inverses von a in $\mathbb{Z}_p$.

Ist nun $p \in \mathbb{P}$, dann gilt für alle $a \in \mathbb{Z}_p$ die Beziehung $\gcd(a,p) = 1$, da ja $a < p$ ist und p keine Teiler hat. Folglich sind alle Elemente von $\mathbb{Z}_p \setminus \{0\}$ Einheiten, und $\mathbb{Z}_p$ ist ein Körper.

Ist aber p zusammengesetzt, dann gibt es ein $a \in \mathbb{Z}_p$ mit $\gcd(a,p) \neq 1$, welches folglich keine Einheit ist. □

Definition 4.5 Das Inverse $b = a^{-1} \pmod{p}$ von a modulo p bezeichnen wir als *modulares Inverses.* Es existiert immer – wie gezeigt – falls p prim ist. Ist jedoch p zusammengesetzt, so existiert $a^{-1} \pmod{p}$ nur, falls a kein Teiler von p ist.

Die *Charakteristik* eine kommutativen Rings R mit 1 ist die kleinste natürliche Zahl c, so daß c-fache Addition des Einslelements

$$\underbrace{1 + 1 + \cdots + 1}_{c \text{ Summanden}} = 0$$

ist. Wir schreiben $c = \mathrm{char}(R)$. $\mathbb{Z}_p$ hat wegen $p \cdot 1 \equiv 0 \pmod{p}$ die Charakteristik p. Gibt es kein solches $c \in \mathbb{N}$, so sagen wir, R habe die Charakteristik 0. $\mathbb{Z}$ ist ein Ring der Charakteristik 0.

Für $p \in \mathbb{P}$ bezeichnen wir mit $\mathbb{Z}_p^* := \mathbb{Z}_p \setminus \{0\}$ die multiplikative Gruppe in $\mathbb{Z}_p$.[3] Ist $p \in \mathbb{P}$ eine Primzahl, so wird der Körper $\mathbb{Z}_p$ auch mit $\mathbb{F}_p$[4] oder mit GF(p) (*Galoisfeld*) bezeichnet,[5] s. auch Abschnitt 7.4. △

Sitzung 4.2 In *Mathematica* können wir mittels der Funktion `Mod` die modularen Grundrechenarten ausführen. Beispielsweise erzeugen die Funktionen[6]

```
In[1]:= AddZ[p_] := MatrixForm[Table[Mod[i + j, p], {i, 0, p - 1}, {j, 0, p - 1}]]
        MultZ0[p_] := MatrixForm[Table[Mod[i * j, p], {i, 0, p - 1}, {j, 0, p - 1}]]
        MultZ[p_] := MatrixForm[Table[Mod[i * j, p], {i, p - 1}, {j, p - 1}]]
```

die Additions- bzw. Multiplikationstabelle von $\mathbb{Z}_p$ bzw. $\mathbb{Z}_p^*$ in Matrizenform.[7] Die auf S. 80 gezeigten Tabellen lassen sich hiermit leicht erzeugen. In Wirklichkeit ist eine modulare Arithmetik in *Mathematica* nicht implementiert, weswegen wir mittels `Mod` auf die Langzahlarithmetik in $\mathbb{Z}$ zurückgreifen müssen.

Modulare Inverse können auf folgende drei Arten bestimmt werden. Eine direkte Implementierung ist gegeben durch die Funktion `PowerMod` mittels

$$a^{-1} \pmod{p} = \texttt{PowerMod[}a\texttt{,}-1\texttt{,}p\texttt{]}\ .$$

Wir bekommen z. B.

```
In[2]:= PowerMod[a = 12345678, -1, p = 1234567891]
Out[2]= 908967567
```

Auf indirektem Wege können wir $a^{-1} \pmod{p}$ mit Hilfe des erweiterten Euklidischen Algorithmus bestimmen (s. Beweis von Satz 4.4)

```
In[3]:= {g, {s, t}} = ExtendedGCD[a, p]
Out[3]= {1,{-325600324,3256003}}
```

Wir prüfen den erweiterten Euklidischen Algorithmus

```
In[4]:= s * a + t * p
Out[4]= 1
```

und berechnen das modulare Inverse:

```
In[5]:= inv = Mod[s, p]
Out[5]= 908967567
```

Schließlich kann man `Solve` verwenden:

[3] Ist p keine Primzahl, so erklärt man $\mathbb{Z}_p^*$ als die Menge der invertierbaren Elemente von $\mathbb{Z}_p$. Dies liefert ebenfalls eine Gruppe. Diesen Fall werden wir aber nicht weiter betrachten.

[4] Das $\mathbb{F}$ steht für das englische Wort field (Körper).

[5] Engl.: Galois field; eigentlich müßte es also Galoiskörper heißen.

[6] Mit `MatrixForm` liefert auch das Paar `{AddZ[6], MultZ0[6]}` Matrizen.

[7] Bei der Multiplikation mit oder ohne die 0.

```
In[6]:= Solve[{12345678 * x == 1, Modulus == 1234567891}, x]
Out[6]= {{Modulus → 1234567891, x → 908967567}}
```

Mit der Gleichung `Modulus==`*n* wird der zugrundeliegende Modul übergeben. □

4.2 Modulare Quadratwurzeln

Während lineare Gleichungen

$$a \cdot x + b = c$$

in $\mathbb{Z}_p$ – welche also der Gleichung

$$a \cdot x + b \equiv c \pmod{p}$$

entsprechen – auf Grund der Ringstruktur (für $\gcd(a,p) = 1$) eine eindeutige Lösung

$$x = a^{-1} \pmod{p} \cdot (c - b)$$

besitzen, ist es bereits recht schwierig, die quadratische Gleichung

$$x^2 = a$$

zu lösen. Eine Lösung x dieser Gleichung

$$x^2 \equiv a \pmod{p}$$

nennen wir eine *modulare Quadratwurzel* und bezeichnen sie mit $x = \sqrt{a} \pmod{p}$.

Sitzung 4.3 Mit

```
In[1]:= Table[Mod[x^2, 11], {x, 0, 10}]
Out[1]= {0,1,4,9,5,3,3,5,9,4,1}
```

bestimmen wir die modularen Quadrate modulo 11. Hieraus können wir ablesen, daß beispielsweise $1 \equiv \sqrt{1} \pmod{11}$, $2 \equiv \sqrt{4} \pmod{11}$, $3 \equiv \sqrt{9} \pmod{11}$, $4 \equiv \sqrt{5} \pmod{11}$, … ist.

Es zeigt sich insbesondere, daß nicht alle $a \in \mathbb{Z}_{11}$ ein modulares Quadrat darstellen, sondern nur die Zahlen 0,1,3,4,5,9. Folglich gibt es nicht für alle $x \in \mathbb{Z}_p$ eine modulare Quadratwurzel. Falls es aber eine modulare Quadratwurzel gibt, dann sogar mindestens zwei, denn es ist

$$(p - x)^2 \equiv p^2 - 2xp + x^2 \equiv x^2 \pmod{p} \,. \tag{4.5}$$

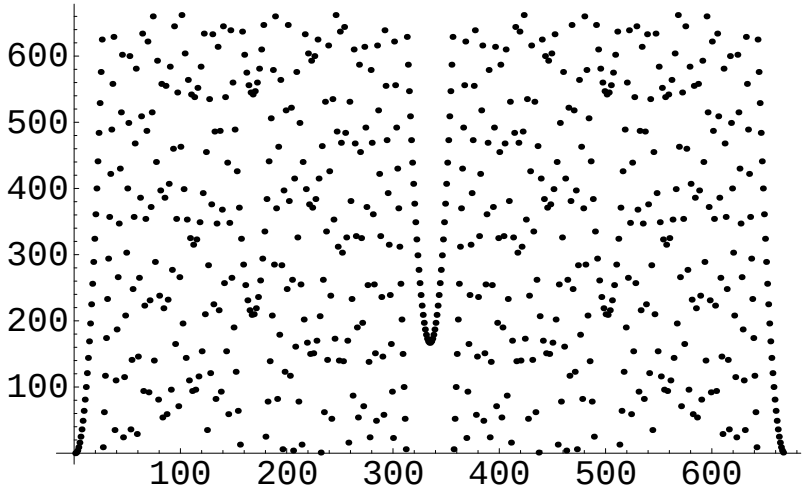
Ist also x eine modulare Quadratwurzel von a, dann auch $p - x$.[8] Zahlen, welche eine modulare Quadratwurzel besitzen, nennt man auch *quadratische Reste.*

Die folgende Liste liefert beispielsweise die quadratischen Reste modulo 123:

```
In[2]:= Union[Table[Mod[x^2,123],{x,0,122}]]
Out[2]= {0,1,4,9,10,16,18,21,25,31,33,36,37,39,40,42,43,45,46,49,51,57,61,
          64,66,72,73,78,81,82,84,87,90,91,100,102,103,105,114,115,118,121}
```

Nun stellen wir die modulare Quadratfunktion graphisch dar, zunächst modulo 667:

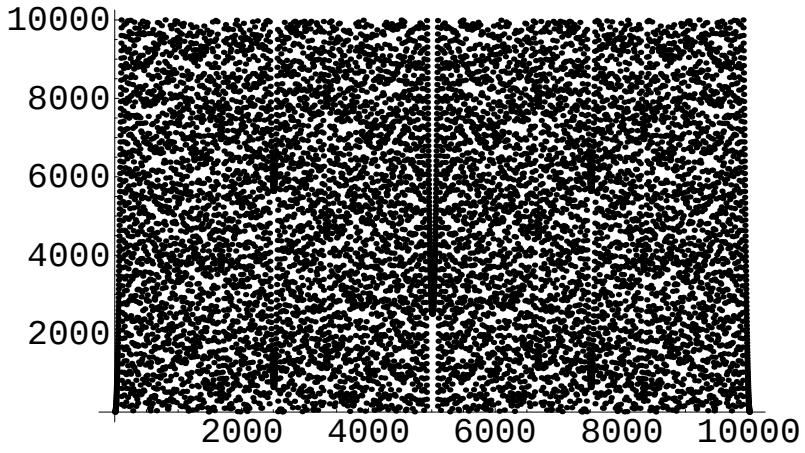
```
In[3]:= ListPlot[Table[Mod[x^2,667],{x,0,666}]]
```




```
Out[3]= -Graphics-
```

und dann modulo 10007:

```
In[4]:= ListPlot[Table[Mod[x^2,10007],{x,0,10006}]]
```




```
Out[4]= -Graphics-
```

[8] Daher ist $\sqrt{a} \pmod{p}$ mehrdeutig.

Man sieht sehr schön die aus (4.5) resultierende Symmetrie bzgl. $p/2$. Vor allem aber machen die Graphen plausibel, daß es – ganz im Gegensatz zu der regelmäßigen Situation in $\mathbb{Z}$ – auf Grund der Unregelmäßigkeit des Graphen recht schwierig sein kann, die modulare Quadratwurzel zu bestimmen.

Wir bestimmen die modularen Quadratwurzeln $\sqrt{a} \pmod{667}$ für $a = 500,\ldots,510$ mit *Mathematicas* `Solve`-Kommando:

```
In[5]:= Table[Solve[{x^2 == a, Modulus == 667}, x], {a, 500, 510}]
Out[5]= {{},{},{},{},{},{},{{Modulus → 667, x → 184}, {Modulus → 667, x → 483}},
          {},{},{{Modulus → 667, x → 62}, {Modulus → 667, x → 315},
            {Modulus → 667, x → 352}, {Modulus → 667, x → 605}},{},{}}
```

Es liegen also nur für $a = 506$ und $a = 509$ quadratische Reste vor. Man beachte, daß es vier modulare Quadratwurzeln $\sqrt{509} \pmod{667}$ gibt.

Schließlich programmieren wir selbst die Bestimmung der modularen Quadratwurzel. Hat man die Liste *aller* modularen Quadrate, fällt die Bestimmung der Quadratwurzel – als Inverser – nicht schwer:

```
In[6]:= ModularSqrt[a_, p_] := Module[{liste, x},
          liste = Table[Mod[x^2, p], {x, 0, p - 1}];
          Flatten[Position[liste, a] - 1]
        ]

In[7]:= ModularSqrt[506, 667]
Out[7]= {184,483}

In[8]:= ModularSqrt[509, 667]
Out[8]= {62,315,352,605}
```

Hierfür benötigt man also p Multiplikationen, welche – mit dem Schulalgorithmus – die Komplexität $O((\log p)^2))$ haben, was einen Aufwand von $O(p \cdot (\log p)^2)$ liefert. Dies ist wieder exponentiell in der Länge $\log p$ von p,[9] während das modulare Quadrieren einer Zahl x der Länge n eine Komplexität von höchstens $O(n^2)$, also $O((\log x)^2) \leqq O((\log p)^2)$ hat.[10] □

Wir haben mit der modularen Quadratwurzel – nach dem Faktorisieren ganzer Zahlen – eine weitere Funktion kennengelernt, welche nur sehr zeitaufwendig auszuwerten ist, obwohl ihre Umkehrfunktion – das modulare Quadrieren – sehr einfach durchgeführt werden kann. Es ist praktisch unmöglich, eine modulare Quadratwurzel zu bestimmen, falls nur p groß genug ist. Derartige Funktionen spielen in der modernen Kryptographie eine sehr wichtige Rolle, s. Abschnitt 5.5. Bis dato sind keine wirklich effizienten Algorithmen zur Bestimmung der modularen Quadratwurzel bekannt.

[9] Dies wird durch beschleunigte Ausführung der Multiplikation auch nicht erheblich besser.

[10] wenn man nicht bessere Multiplikationsalgorithmen verwendet

4.3 Chinesischer Restsatz

Kennt man die Werte eines Polynoms vom Grad $n-1$ an n Stellen, so kann man das Polynom durch *Interpolation* eindeutig rekonstruieren. Wir betrachten dies genauer in Abschnitt 6.4.

Eine ähnliche Situation ist die folgende. Eine ganze Zahl $x \in \mathbb{Z}$ sei modulo einiger Primzahlen $p_1,\ldots,p_n \in \mathbb{P}$ bekannt. Wie kann man x rekonstruieren? Wir werden sehen, daß dies in eindeutiger Weise möglich ist, falls $0 \leqq x < p_1 \cdots p_n$ ist.[11] Wir betrachten zunächst den Fall $n = 2$. Der allgemeine Fall geht dann rekursiv aus diesem hervor.

Satz 4.6 (Chinesischer Restsatz) Seien p und q zwei ganze Zahlen mit $\gcd(p,q) = 1$. Dann hat das Gleichungssystem

$$\begin{aligned} x &\equiv a \pmod{p} \qquad \text{und} \\ x &\equiv b \pmod{q} \end{aligned}$$

eine ganzzahlige Lösung $x \in \mathbb{Z}$, welche bis auf ein additives Vielfaches von $p \cdot q$ eindeutig bestimmt ist.

Beweis: Wir konstruieren die Lösung. Wegen $\gcd(p,q) = 1$ liefert der erweiterte Euklidische Algorithmus ganze Zahlen $s,t \in \mathbb{Z}$ mit

$$s\,p + t\,q = 1\ .$$

Dann erfüllt also die ganze Zahl

$$x := b\,s\,p + a\,t\,q$$

die Gleichungen

$$x \equiv a\,t\,q \equiv a \cdot (1 - s\,p) \equiv a \pmod{p}$$

sowie

$$x \equiv b\,s\,p \equiv b \cdot (1 - t\,q) \equiv b \pmod{q}$$

und ist damit eine Lösung des Problems.
Eindeutigkeit: Seien x und $\hat{x}$ zwei Lösungen des Gleichungssystems, dann folgt also $x - \hat{x} \equiv 0 \pmod{p}$ und $x - \hat{x} \equiv 0 \pmod{q}$. Wegen $\gcd(p,q) = 1$ folgt hieraus aber $x - \hat{x} \equiv 0 \pmod{p \cdot q}$, s. Übungsaufgabe 4.6. □

[11] oder x in einem anderen Segment von $\mathbb{Z}$ der Länge $p_1 \cdots p_n$ liegt

Es ist einfach, hieraus rekursiv Lösungen von Gleichungssystemen

$$\begin{aligned} x &\equiv a_1 \pmod{p_1} \\ x &\equiv a_2 \pmod{p_2} \\ &\vdots \\ x &\equiv a_n \pmod{p_n} \end{aligned} \tag{4.6}$$

mit Moduli $p_k\,(k = 1,\ldots,n)$ ohne gemeinsamen Teiler zu bestimmen: Ist l die Lösung des Problems für die ersten beiden Gleichungen, so gilt offenbar

$$x \equiv l \pmod{p_1 \cdot p_2}\,,$$

und wir können die ersten beiden Gleichungen durch diese eine ersetzen. Wir haben also den

Satz 4.7 (Chinesischer Restsatz) Seien $p_1,\ldots p_n$ ganze Zahlen mit $\gcd(p_j,p_k) = 1$ für $j \neq k$. Dann hat das Gleichungssystem (4.6) eine ganzzahlige Lösung $x \in \mathbb{Z}$, welche bis auf ein additives Vielfaches von $p_1 \cdots p_n$ eindeutig bestimmt ist.

Beweis: Es bleibt nur zu zeigen, daß $\gcd(p_1 \cdot p_2,p_k) = 1$ ist für $k = 3,\ldots,n$. Da aber weder p_1 noch p_2 einen gemeinsamen Teiler mit p_k besitzt, so offenbar auch nicht das Produkt $p_1 \cdot p_2$, wie man leicht mit dem Fundamentalsatz der Zahlentheorie (Satz 3.11) sieht. □

Sitzung 4.4 Die Funktion `ChineseRemainder[{`$a_1,\ldots,a_n$`},{`$p_1,\ldots,p_n$`}]` aus dem Package `NumberTheory‘NumberTheoryFunctions‘` berechnet die Lösung des Reste-problems (4.6) modulo $\prod_{k=1}^{n} p_k$.

```
In[1]:= Needs["NumberTheory‘NumberTheoryFunctions‘"]
```

Wir lösen die Gleichungen

$$\begin{aligned} x &\equiv 17 \pmod{101} \qquad \text{und} \\ x &\equiv 4 \pmod{97} \end{aligned}$$

durch

```
In[2]:= ChineseRemainder[{17,4},{101,97}]
Out[2]= 2138
```

und die Gleichungen

$$\begin{aligned} x &\equiv 0 \pmod 2, && \text{d. h., } x \text{ ist gerade,}\\ x &\equiv 0 \pmod 3, && \text{d. h., } x \text{ ist Dreierzahl,}\\ x &\equiv 0 \pmod 5, && \text{d. h., } x \text{ ist Fünferzahl,}\\ x &\equiv 1 \pmod 7 \end{aligned}$$

mit dem Aufruf

```
In[3]:= ChineseRemainder[{0,0,0,1},{2,3,5,7}]
Out[3]= 120
```

Nun programmieren wir den Algorithmus aus dem Beweis von Satz 4.6:

```
In[4]:= CR[{a_,b_},{p_,q_}] := Module[{g,s,t},
            {g,{s,t}} = ExtendedGCD[p,q];
            Mod[b * s * p + a * t * q,p * q]
          ]
```

mit welchem wir das erste Problem ebenfalls direkt lösen können:

```
In[5]:= CR[{17,4},{101,97}]
Out[5]= 2138
```

Das zweite Problem erfordert drei Schritte:

```
In[6]:= CR[{0,0},{2,3}]
Out[6]= 0

In[7]:= CR[{0,0},{2 * 3,5}]
Out[7]= 0

In[8]:= CR[{0,1},{2 * 3 * 5,7}]
Out[8]= 120
```

Eine Implementierung der Lösung des chinesischen Resteproblems aus Satz 4.7 soll in Übungsaufgabe 4.5 durchgeführt werden.

Wir rechnen ein etwas schwierigeres Problem:[12]

```
In[9]:= a = Random[Integer,{1,10^100}];
        b = Random[Integer,{1,10^100}];
        p = NextPrime[a];
        q = NextPrime[b];

In[10]:= CR[{a,b},{p,q}]//Timing
Out[10]= {0.01 Second,
           162144131926154041445824102383364379691755788259107376479847187994307280410100064280160210451625620636115977510357995217803114168656040751293487099939100857168117966542358176881958691103508673600852 71}
```

[12] Wir verwenden die Funktion `NextPrime` von S. 4.

Die Anwendung des erweiterten Euklidischen Algorithmus ist also auch bei größeren Zahlen noch völlig rechenzeitunkritisch. □

4.4 Der kleine Satz von Fermat

Ein wichtiger Satz der elementaren Zahlentheorie, welcher sich bei Anwendungen aus der Kryptographie als besonders erfolgreich erweist, ist der *kleine Satz von Fermat.* Er besagt

Satz 4.8 Für jede Primzahl $p \in \mathbb{P}$ und $x \in \mathbb{N}$ gilt

$$x^p \equiv x \pmod{p} .$$

Beweis: Wir benutzen die *binomische Formel*

$$(x+1)^p = x^p + \binom{p}{1}x^{p-1} + \binom{p}{2}x^{p-2} + \cdots + \binom{p}{p-1}x^1 + 1,$$

welche für $x \in \mathbb{Z}$, $p \in \mathbb{N}$ gültig ist, wobei

$$\binom{p}{k} = \frac{p \cdot (p-1) \cdot (p-2) \cdots (p-k+1)}{k \cdot (k-1) \cdot (k-2) \cdots 1} \in \mathbb{N}$$

die ganzzahligen *Binomialkoeffizienten* sind.

Wegen $\binom{p}{k} \in \mathbb{N}$ ist für jede Primzahl $p \in \mathbb{P}$ und $0 < k < p$ der Binomialkoeffizient $\binom{p}{k}$ ein Vielfaches von p. Wir führen nun eine Induktion bzgl. x durch.

Der kleine Fermatsche Satz ist offenbar richtig für $x = 1$ und jede Primzahl $p \in \mathbb{P}$. Als Induktionsannahme nehmen wir nun an, er gilt für ein $x \in \mathbb{N}$, d. h., $x^p \equiv x \pmod{p}$. Dann folgt

$$(x+1)^p \equiv x^p + \binom{p}{1}x^{p-1} + \cdots + \binom{p}{p-1}x^1 + 1 \equiv x^p + 1 \equiv x + 1 \pmod{p} . \quad (4.7)$$

Wir schließen, daß die Behauptung auch für $x + 1$ richtig ist. Damit ist der Induktionsbeweis vollständig. □

Sitzung 4.5 *Mathematica* hat eine effiziente Implementierung von modularen Potenzen. `PowerMod[a,n,p]` berechnet $a^n \pmod{p}$. Will man beispielsweise

$$y := 123456789^{123456789} \mod 987654321$$

auf konventionelle Art berechnen, so muß man zuerst die *riesige Zahl* $123456789^{123456789}$ (diese hat fast 10^9 Stellen!)[13] in $\mathbb{Z}$ berechnen, um das Resultat danach modulo 987654321 zu reduzieren. Das ist natürlich höchst ineffizient. Erheblich besser ist es, vor jeder Multiplikation modulo 987654321 zu reduzieren.

Noch besser ist es allerdings, die Potenz gemäß folgender Rekursion[14]

$$\mathrm{mod}(a^n,p) = \begin{cases} \mathrm{mod}(\mathrm{mod}(a^{\frac{n}{2}},p)^2,p) & \text{falls } n \text{ gerade} \\ \mathrm{mod}(\mathrm{mod}(a^{n-1},p)\cdot a,p) & \text{falls } n \text{ ungerade} \end{cases}$$

zu bestimmen. Dies ist offenbar wieder ein Divide-and-Conquer-Algorithmus, welcher auch von `PowerMod` verwendet wird.

Wir bekommen z. B. sehr schnell die Antwort

```
In[1]:= PowerMod[123456789,123456789,987654321]
Out[1]= 598987215
```

Dieser Algorithmus zur Berechnung der modularen Potenz läßt sich leicht auch selbst programmieren:

```
In[2]:= $RecursionLimit = ∞;
        powermod[a_,0,p_] := 1
        powermod[a_,n_,p_] := Mod[powermod[a,n/2,p]^2,p]/;EvenQ[n]
        powermod[a_,n_,p_] := Mod[powermod[a,n - 1,p] * a,p]/;OddQ[n]
```

Dann liefert

```
In[3]:= powermod[123456789,123456789,987654321]
Out[3]= 598987215
```

das obige Resultat in ebenso kurzer Zeit. Mit

```
In[4]:= a = Random[Integer,{1,10^200}];
        n = Random[Integer,{1,10^200}];
        p = NextPrime[a];

In[5]:= powermod[a,n,p]//Timing
Out[5]= {0.81 Second,
           3532174365474810481600821826177575794087422026076928441602188937248578405053721
             4162665721074312195248367775712305903351885692101374426797027902585458630935 4
             3689005041581449674549860610245481049029 67}
```

[13] Daher muß man eine Rechnung wie beispielsweise `Mod[123456789^123456789,987654321]` mit **Kernel, Abort Evaluation** abbrechen!

[14] Wir benutzen hier die Gleichungen (4.2)–(4.3).

sieht man, daß dieser Algorithmus auch für 200-stellige Zahlen noch Rechenzeiten im Sekundenbereich hat.[15] □

Als Folgerung des Fermatschen Satzes erhält man durch Division durch x für teilerfremde x und p eine zweite Formulierung des kleinen Satzes von Fermat:

$$x^{p-1} \equiv 1 \pmod{p} .$$

Insbesondere gilt für $a \in \mathbb{Z}_p^*$ die Beziehung $a^{p-1} = 1$. Da in $\mathbb{Z}_p$ ja nur endlich viele Werte liegen, ist es nicht überraschend, daß sukzessives Potenzieren irgendwann einmal wieder die multiplikative Einheit 1 liefert. Der kleine Satz von Fermat konkretisiert dies. Als weitere Folge bekommt man die

Folgerung 4.9 Sei $p \in \mathbb{P}$ eine Primzahl. Dann gilt: Für $a \in \mathbb{Z}_p^*$ ist die Folge $(a^k \pmod{p})_{k\in\mathbb{N}}$ periodisch mit einer Periode, welche Teiler von $p-1$ ist.

Beweis: Aus dem kleinen Satz von Fermat folgt $a^{p-1} \equiv 1 \pmod{p}$, so daß die Folge $(a^0,a^1,a^2,\ldots) = (1,a,a^2,\ldots) \pmod{p}$ ganz offenbar die Periode $p-1$ hat, da nach $p-1$ Schritten der Wert 1 zum wiederholten Male auftritt. Jede weitere Periode ist ein Teiler oder ein Vielfaches von $p-1$. Denn gäbe es zwei teilerfremde Perioden q und r, dann liefert der erweiterte Euklidische Algorithmus $s,t \in \mathbb{Z}$ mit $1 = sq + tr$ und somit

$$a \equiv a^1 \equiv a^{sq+tr} \equiv (a^q)^s \cdot (a^r)^t \equiv 1 \pmod{p} ,$$

also $a \equiv 1 \pmod{p}$. In diesem Fall aber ist die Periodizität trivial. □

Definition 4.10 (Ordnung, Erzeuger und primitives Element) Sei $p \in \mathbb{P}$ und $a \in \mathbb{Z}_p^*$. Dann heißt die Periode aus Folgerung 4.9 die (multiplikative) *Ordnung* von a in $\mathbb{Z}_p^*$ und wird mit $\mathrm{ord}(a)$ bezeichnet. Sie ist also die kleinste Zahl $m \in \mathbb{N}$ derart, daß $a^m = 1$ ist. Die Ordnung jedes Elements $a \in \mathbb{Z}_p^*$ ist ein Teiler von $p-1$.

In vielen Fällen wünscht man sich, daß die Periode in Folgerung 4.9 möglichst groß ist. Daher nennen wir ein Element $a \in \mathbb{Z}_p^*$ mit $\mathrm{ord}(a) = p-1$ einen *Erzeuger* oder ein *primitives Element* von $\mathbb{Z}_p^*$. △

Der folgende Satz begründet diese Begriffsbildung:

[15] Hat man in einer kompilierenden Programmiersprache eine gute Langzahlarithmetik zur Verfügung, können die Rechenzeiten ggfs. auch besser sein als bei *Mathematica*.

Satz 4.11 Ist a ein erzeugendes Element von $\mathbb{Z}_p^*$, so erzeugen die Potenzen von a die ganze multiplikative Gruppe $\mathbb{Z}_p^*$:

$$\{a^0,a^1,\ldots,a^{p-2}\} = \mathbb{Z}_p^* .$$

Beweis: Wegen der Periodizität gibt es genau $p-1$ Potenzen $a^0,\ldots,a^{p-2}$. Wir werden zeigen, daß diese Potenzen alle verschiedene Werte haben. Da aber $\mathbb{Z}_p^*$ nur $p-1$ Elemente hat, kommt jedes Element von $\mathbb{Z}_p^*$ in der Liste $\{a^0,a^1,\ldots,a^{p-2}\}$ vor.

Nehmen wir nun an, es gäbe ein Element, welches doppelt erzeugt wird: $a^q \equiv a^r \pmod p$ mit $0 \leqq q < r < p-1$. Dann folgt aber $a^{r-q} \equiv 1 \pmod p$. Somit liefert $k := r-q$ einen Widerspruch zur Voraussetzung. □

Beispiel 4.12 Für $a = 2$ gilt in $\mathbb{Z}_5$

$$a^1 = 2 , \quad a^2 = 4 , \quad a^3 = 3 \quad \text{sowie} \quad a^4 = 1 .$$

Also ist ord(2) $= 4 = p-1$, und 2 ist ein Erzeuger von $\mathbb{Z}_5^*$.

Ebenso ist für $a = 2 \in \mathbb{Z}_{11}$

$$a^1 = 2 , \quad a^2 = 4 , \quad a^3 = 8 , \quad a^4 = 5 , \quad a^5 = 10 ,$$
$$a^6 = 9 , \quad a^7 = 7 , \quad a^8 = 3 , \quad a^9 = 6 , \quad a^{10} = 1 .$$

Wieder ist also 2 ein Erzeuger von $\mathbb{Z}_{11}^*$. Auf der anderen Seite gilt für $a = 3$:

$$a^1 = 3 , \quad a^2 = 9 , \quad a^3 = 5 , \quad a^4 = 4 \quad \text{sowie} \quad a^5 = 1 .$$

Also ist ord(a) $= 5 \mid 10 = p-1$. △

Sitzung 4.6 *Mathematica* kann erzeugende Elemente finden. Die Funktion

```
In[1]:= MinErzeuger[p_] := Module[{a = 2},
          While[Not[MultiplicativeOrder[a,p] == p - 1],a = a + 1];
          a
        ]
```

benutzt `MultiplicativeOrder` zur Bestimmung der multiplikativen Ordnung des Elementes $a \in \mathbb{Z}_p^*$. Wir erhalten für $p = 10007$

```
In[2]:= MinErzeuger[p = 10007]
Out[2]= 5
```

den kleinsten Erzeuger $a = 5$, und die Rechnung

```
In[3]:= Length[Union[Table[PowerMod[5,n,p],{n,p - 1}]]]
Out[3]= 10006
```

zeigt, daß die Menge der Potenzen von a mit $\mathbb{Z}_p^*$ übereinstimmt. □

Der folgende Satz, den wir in Abschnitt 7.4 beweisen werden, zeigt die Existenz erzeugender Elemente.

Satz 4.13 Sei $p \in \mathbb{P}$ eine Primzahl. Dann besitzt $\mathbb{Z}_p^*$ ein erzeugendes Element. □

4.5 Modulare Logarithmen

In diesem Abschnitt betrachten wir die Aufgabe, aus der modularen Gleichung

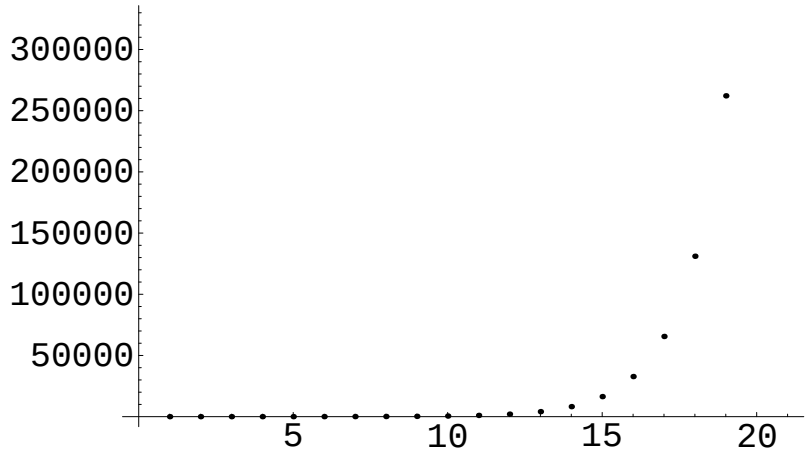
$$a^x \equiv b \pmod{p}$$

einen Lösungsexponenten x zu bestimmen. Dieser wird ein *modularer Logarithmus*[16] von b zur Basis a modulo p, in Zeichen $x = \log_a b \pmod{p}$, genannt.

Es stellt sich (wie bei der modularen Quadratwurzel) wieder heraus, daß die Bestimmung von modularen Logarithmen – nach dem heutigen Kenntnisstand – ein *sehr* schwieriges mathematisches Problem darstellt.

Sitzung 4.7 Wir betrachten zunächst die Exponentialfunktion 2^x über $\mathbb{Z}$. Diese hat den wohlbekannten regelmäßigen Graphen:

```
In[1]:= ListPlot[Table[2^x,{x,0,20}]]
```




```
Out[1]= -Graphics-
```

Daher ist es sehr leicht, die Umkehrfunktion zu bestimmen:

[16] Die Umkehrfunktion der modularen Exponentialfunktion wird häufig auch der *diskrete Logarithmus* genannt. Um einer Verwechslung mit der diskreten Logarithmusfunktion in $\mathbb{Z}$ vorzubeugen, sprechen wir lieber von der modularen Logarithmusfunktion.

```
In[2]:= Clear[DiskreterLogarithmus]
        DiskreterLogarithmus[y_,a_] := Module[{z,zähler},
           z = y; zähler = 0;
           While[IntegerQ[z]&&z > 1,
             z = z/a; zähler = zähler + 1];
           If[Not[IntegerQ[z]],
             Return["diskreter Logarithmus existiert nicht"]];
           zähler
         ]
```

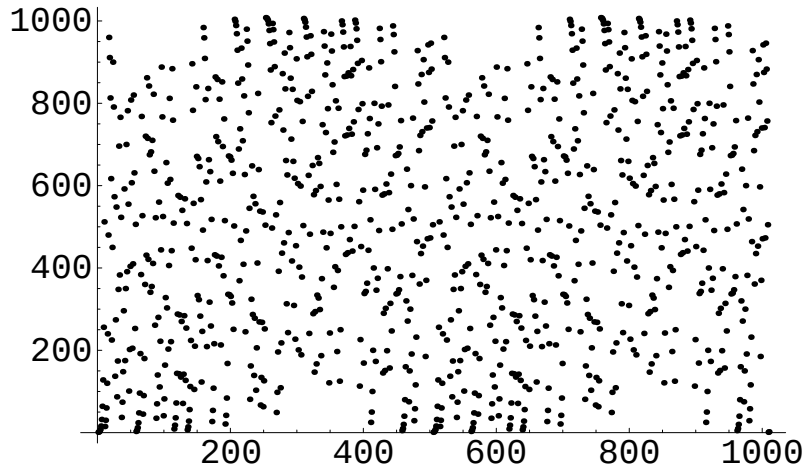
Wir rechnen zwei Beispiele:

```
In[3]:= DiskreterLogarithmus[5^1000,5]
Out[3]= 1000
```

```
In[4]:= DiskreterLogarithmus[5^1000 + 1,5]
Out[4]= diskreter Logarithmus existiert nicht
```

Eine ganz andere Situation liegt bei der modularen Exponentialfunktion (z. B. zur Basis 2) vor, wie man wieder dem Graphen entnehmen kann:

```
In[5]:= ListPlot[Table[Mod[2^x,1009],{x,0,1009}]]
```




```
Out[5]= -Graphics-
```

Gemäß Folgerung 4.9 ist der Graph periodisch mit einer Periode P, welche ein Teiler von $p-1$ ist. In unserem Fall sieht man, daß die Periode $\frac{p-1}{2}$ ist. Dennoch muß man i. a. zur Berechnung eines modularen Logarithmus $O(p)$ Werte berechnen. Als Beispiel erklären wir eine 8-stellige Primzahl p:

```
In[6]:= p = NextPrime[10^8]
Out[6]= 100000007
```

Sehr schnell läßt sich die modulare Potenz $b = \mathrm{mod}(2^{1234567}, p)$ bestimmen:

```
In[7]:= (b = PowerMod[2,1234567,p])//Timing
```

```
Out[7]= {0. Second,77997187}
```

Anders als im linearen und im quadratischen Fall kann `Solve` das Umkehrproblem nicht lösen:

```
In[8]:= Solve[{2^x == b,Modulus == p},x]

Solve :: "tdep": The equations appear to involve the variables to be
solved for in an essentially non-algebraic way.
Out[8]= Solve[{2^x == 77997187,Modulus == 100000007},x]
```

Die eingebaute Funktion `MultiplicativeOrder` jedoch berechnet modulare Logarithmen:[17]

```
In[9]:= MultiplicativeOrder[2,p,b]//Timing
Out[9]= {6.84 Second,1234567}
```

In unserem Fall ist die Rechenzeit noch akzeptabel, aber bei 15-stelligen Eingaben werden diese bereits unglaublich groß. Es ist jedoch völlig ausgeschlossen, mit *Mathematica* den modularen Logarithmus 100-stelliger Eingaben zu bestimmen. □

Auch die Komplexität des modularen Logarithmus ist exponentiell in der Eingabelänge, und es sind bis dato keine wirklich effizienten Algorithmen bekannt.

4.6 Pseudoprimzahlen

In Abschnitt 3.5 hatten wir die Faktorisierung ganzer Zahlen betrachtet und festgestellt, daß hierfür keine effizienten Algorithmen bekannt sind.

Auf der anderen Seite kann man aber u. U. sehr einfach feststellen, daß eine Zahl zusammengesetzt ist.

Sitzung 4.8 Wir wollen testen, ob $p = 1234567$ zusammengesetzt ist. Hierfür erklären wir

```
In[1]:= p = 1234567
Out[1]= 1234567
```

Wir bestimmen eine zufällige Zahl $a \in \mathbb{Z}_p^*$:

```
In[2]:= a = Random[Integer,{1,p - 1}]
Out[2]= 1118553
```

und berechnen $a^p - a \pmod p$

[17] Man beachte jedoch, daß modulare Logarithmen i. a. wieder nicht eindeutig bestimmt sind. `MultiplicativeOrder` berechnet die *kleinste* Lösung.

```
In[3]:= Mod[PowerMod[a,p,p] - a,p]
Out[3]= 744287
```

Da das Resultat nicht Null ist, wissen wir, daß p nicht prim sein kann, da sonst der kleine Satz von Fermat verletzt wäre. Also ist p zusammengesetzt. □

Wir haben also mit Hilfe des kleinen Satzes von Fermat herausgefunden, daß p zusammengesetzt ist, ohne hierbei auch nur eine Idee von möglichen Teilern von p erhalten zu haben. Die Berechnung ging schnell, da die modulare Exponentiation schnell durchführbar ist. Die vorgestellte Methode nennen wir den *Fermattest.*

Wir erklären nun:

Definition 4.14 Die Zahl $a \in \mathbb{N}_{\geq 2}$ heißt *(Fermatscher) Zeuge* für die Zerlegbarkeit von $p \in \mathbb{N}_{\geq 2}$, falls $a^p \not\equiv a \pmod{p}$ ist. Eine Zahl $p \in \mathbb{N}_{\geq 2} \setminus \mathbb{P}$ heißt *(Fermatsche) Pseudoprimzahl* zur Basis a, wenn $a^p \equiv a \pmod{p}$ gilt. Eine zusammengesetzte Zahl $p \in \mathbb{N}_{\geq 2}$ heißt *Carmichaelzahl*, sofern $a^p \equiv a \pmod{p}$ für alle $a \in \mathbb{Z}_p$ gilt. △

Findet der Fermattest einen Zeugen für die Zerlegbarkeit von p, so ist p zweifellos zusammengesetzt. Es stellt sich also die Frage, ob es solche Zeugen immer gibt oder aber ob es Carmichaelzahlen gibt.

Sitzung 4.9 Die folgende *Mathematica*-Funktion sucht nach der kleinsten Carmichaelzahl $\geqq n$ (sofern es sie gibt). Hierzu testen wir also für $p = n, n+1, \ldots$, ob jeweils alle $a \leqq p$ das Fermatkriterium erfüllen.

```
In[1]:= Clear[NextCarmichael]
        NextCarmichael[n_] := Module[{fertig = False, p = n, a},
           While[Not[fertig],
             If[Not[PrimeQ[p]],
               liste = Table[PowerMod[a,p,p] - a, {a, p - 1}];
               liste = Union[liste];
               If[liste == {0}, fertig = True, p = p + 1], p = p + 1]];
           p]
```

Wir suchen zunächst nach der kleinsten Carmichaelzahl:

```
In[2]:= (c = NextCarmichael[4])//Timing
Out[2]= {13.41 Second,561}
```

561 ist also die kleinste Carmichaelzahl. Sie hat die Zerlegung

```
In[3]:= FactorInteger[c]
Out[3]= {{3, 1}, {11, 1}, {17, 1}}
```

Die nächsten drei Carmichaelzahlen ergeben sich wie folgt:

```
In[4]:= (c = NextCarmichael[c + 1])//Timing
Out[4]= {44.5 Second,1105}

In[5]:= FactorInteger[c]
Out[5]= {{5, 1}, {13, 1}, {17, 1}}

In[6]:= (c = NextCarmichael[c + 1])//Timing
Out[6]= {98.39 Second,1729}

In[7]:= FactorInteger[c]
Out[7]= {{7, 1}, {13, 1}, {19, 1}}

In[8]:= (c = NextCarmichael[c + 1])//Timing
Out[8]= {159.79 Second,2465}

In[9]:= FactorInteger[c]
Out[9]= {{5, 1}, {17, 1}, {29, 1}}
```

Eine Fortführung der Berechnungen liefert weitere Carmichaelzahlen, aber die Rechenzeiten sind recht hoch. Es ist bekannt, daß es unendlich viele Carmichaelzahlen gibt [AGP1994]. □

Der Fermattest ist sehr einfach und schnell durchführbar, man führt ihn meist nur mit einem einzigen a durch und erfährt bereits, daß p zusammengesetzt ist. Im negativen Fall wiederholt man den Versuch mit einem anderen zufällig ausgewählten a. Ist p allerdings eine Carmichaelzahl, so scheitert der Fermattest. Es zeigt sich aber, daß die Carmichaelzahlen eine ganz bestimmte Struktur besitzen, welche man gut im Griff hat.

Satz 4.15 (Struktur von Carmichaelzahlen) Eine Zahl $p \in \mathbb{N}_{\geqq 4}$ ist genau dann eine Carmichaelzahl, wenn[18]

(a) $p = p_1 \cdots p_n$ mit lauter paarweise verschiedenen Primzahlen $p_k \in \mathbb{P}$;

(b) $p_k - 1 \mid p - 1$ für alle $k = 1,\ldots,n$.

[18] Man kann ferner beweisen, daß alle p_k ($k = 1,\ldots,n$) ungerade sind und daß $n \geqq 3$ ist.

Beweis: Wir nehmen zunächst an, p sei eine Carmichaelzahl. Es gilt also $a^p \equiv a \pmod{p}$ für alle $a = 0,\ldots,p-1$.
(a): Wir zeigen als erstes, daß p *quadratfrei* ist, d. h. kein Quadrat enthält. Nehmen wir an, q sei ein Primfaktor von p mit $q^2 \mid p$. Dann folgt aus der Voraussetzung $p \mid a^p - a = a \cdot (a^{p-1} - 1)$, und die Wahl $a = q$ führt zu $q^2 \mid p \mid q \cdot (q^{p-1} - 1)$ und wegen $q \in \mathbb{P}$ zu $q \mid q^{p-1} - 1$, einem Widerspruch. Also ist p quadratfrei und somit sind alle Primteiler paarweise verschieden voneinander.
(b): Da $p_k \in \mathbb{P}$ prim ist, gibt es nach Satz 4.13 ein erzeugendes Element $a \in \mathbb{Z}_{p_k}^*$, für welches also $a^{p_k-1} \equiv 1 \pmod{p_k}$, aber $a^r \not\equiv 1 \pmod{p_k}$ für $0 < r < p_k - 1$.

Da p ein Vielfaches von p_k ist, folgt aus der Voraussetzung $a^p \equiv a \pmod{p}$ zunächst $a^p \equiv a \pmod{p_k}$ und schließlich durch Kürzen von a (s. auch Übungsaufgabe 4.3) $a^{p-1} \equiv 1 \pmod{p_k}$, da p_k prim ist. Eine Division mit Rest liefert für den Exponenten $p - 1 = q \cdot (p_k - 1) + r$ mit $0 \leqq r < p_k - 1$, und wir erhalten

$$1 \equiv a^{p-1} \equiv a^{q\cdot(p_k-1)+r} \equiv (a^{p_k-1})^q \cdot a^r \equiv a^r \pmod{p_k}$$

im Widerspruch zur Wahl von a, falls $r \neq 0$ ist. Also folgt $r = 0$ und somit $p_k - 1 \mid p - 1$.

Wir beweisen nun die Umkehrung, daß also aus (a)–(b) folgt, daß p Carmichaelzahl ist. Sei also $p = p_1 \cdots p_n$ mit paarweise verschiedenen Primzahlen $p_k \in \mathbb{P}$ und sei weiter $p_k - 1 \mid p - 1$ für alle $k = 1,\ldots,n$. Dann gilt wieder mit dem Satz von Fermat für alle $k = 1,\ldots,n$

$$a^{p-1} \equiv a^{(p_k-1)\cdot q_k} \equiv (a^{p_k-1})^{q_k} \equiv 1 \pmod{p_k}\,.$$

Aus diesen Gleichungen folgt aber $a^{p-1} \equiv 1 \pmod{p}$, da die Faktoren p_k von p keine gemeinsamen Teiler besitzen (s. Übungsaufgabe 4.6), und somit schließlich $a^p \equiv a \pmod{p}$. □

Sitzung 4.10 Mit Hilfe von Satz 4.15 können wir die ersten Carmichaelzahlen nun viel schneller finden:

```
In[1]:= Clear[FastNextCarmichael]
        FastNextCarmichael[n_] := Module[{fertig = False, p = n},
           While[Not[fertig],
             If[PrimeQ[p], p = p + 1,
               factors = Transpose[FactorInteger[p]];
               If[Not[Union[factors[[2]]] == {1}]||
                   Not[IntegerQ[(p - 1)/Apply[LCM, factors[[1]] - 1]]],
                 p = p + 1, fertig = True]]
           ];
           p
         ]
```

```
In[2]:= (c = FastNextCarmichael[4])//Timing
Out[2]= {0.24 Second,561}

In[3]:= (c = FastNextCarmichael[c + 1])//Timing
Out[3]= {0.27 Second,1105}

In[4]:= (c = FastNextCarmichael[c + 1])//Timing
Out[4]= {0.28 Second,1729}

In[5]:= (c = FastNextCarmichael[c + 1])//Timing
Out[5]= {0.34 Second,2465}
```

In Übungsaufgaben 4.14–4.15 sollen größere Carmichaelzahlen bestimmt werden. □

In der Praxis wird bei der Primzahlsuche eine Verfeinerung des Fermattests angewandt, welcher die Carmichaelzahlen umgeht: der *Rabin-Miller-Test.* Hierbei wird der Fermattest auf folgende Weise ausgedehnt:

Definition 4.16 Eine ungerade Zahl $p \in \mathbb{N}$ mit einer Zerlegung $p-1 = 2^t \cdot u$, u ungerade, heißt *strenge Pseudoprimzahl* zur Basis a, falls $a^u \equiv 1 \pmod{p}$ oder ein $s \in \{0,1,\ldots,t-1\}$ existiert, für welches $a^{2^s \cdot u} \equiv -1 \pmod{p}$ ist.[19] Ist p keine strenge Pseudoprimzahl zur Basis a, so nennen wir a einen *Zeugen für die Zerlegbarkeit von* p. △

Wir zeigen zunächst, daß unsere Begriffsbildungen Sinn machen.

Satz 4.17 Besitzt p einen Zeugen a für die Zerlegbarkeit, so ist p zusammengesetzt.

Beweis: Wir zeigen die gleichwertige Aussage, daß jedes $p \in \mathbb{P}$ eine strenge Pseudoprimzahl ist. Wir können o. B. d. A. annehmen, daß $a \in \mathbb{Z}_p^*$ ist. Aus dem Fermatschen Satz $a^p \equiv a \pmod{p}$ folgt dann $a^{p-1} \equiv 1 \pmod{p}$.

Seien u,s,t wie in Definition 4.16 erklärt. Ist nun $a^u \equiv 1 \pmod{p}$, so ist p eine strenge Pseudoprimzahl, und wir sind fertig. Sei also in der Folge $a^u \not\equiv 1 \pmod{p}$. Wegen $a^{p-1} \equiv a^{2^t \cdot u} \equiv 1 \pmod{p}$ gibt es also ein $s \in \{0,1,\ldots,t-1\}$ mit[20]

$$x := a^{2^s \cdot u} \not\equiv 1 \pmod{p}\,, \qquad \text{aber} \qquad x^2 \equiv a^{2^{s+1} \cdot u} \equiv 1 \pmod{p}\,.$$

Für x gilt somit $x^2 \equiv 1 \pmod{p}$ sowie $x \not\equiv 1 \pmod{p}$ und daher $x \equiv -1 \pmod{p}$, s. Übungsaufgabe 4.7. □

[19] In $\mathbb{Z}_p$ ist natürlich $x \equiv -1 \pmod{p}$ gleichwertig zu $x \equiv p-1 \pmod{p}$. In unserem Vertretersystem ist also $p-1 \in \mathbb{Z}_p$ der richtige Vertreter.

[20] Es ist natürlich $s = \max\{m \in \mathbb{N}_0 \mid a^{2^m \cdot u} \not\equiv 1 \pmod{p}\}$.

Beispiel 4.18 Der Beweis zeigt: Eine strenge Pseudoprimzahl erfüllt das Fermatkriterium $a^p \equiv a \pmod{p}$, besitzt aber zusätzlich noch weitere Eigenschaften. Diese sind wesentlich stärker, wie die folgenden Beispiele zeigen (s. [Rie1994]), welche man durch (mühsames) „Nachrechnen" erhält:

- 2 ist Zeuge für die Zerlegbarkeit aller Nichtprimzahlen < 2 047;
- {2,3} sind Zeugen (d. h., 2 oder 3 ist ein Zeuge) für die Zerlegbarkeit aller Nichtprimzahlen < 1 373 653;
- {2,3,5} sind Zeugen für die Zerlegbarkeit aller Nichtprimzahlen < 25 326 001;
- {2,3,5,7} sind Zeugen für die Zerlegbarkeit aller Nichtprimzahlen < 3 215 031 751;
- {2,3,5,7,11} sind Zeugen für die Zerlegbarkeit aller Nichtprimzahlen < 2 152 302 898 747.

Sitzung 4.11 Wir programmieren den Rabin-Miller-Test:

```
In[1]:= Clear[RabinMillerPrime]

        RabinMillerPrime::"pseudoprime" =
          "strenge Pseudoprimzahl zur Basis ‘1‘";

        RabinMillerPrime::"zusammengesetzt" =
          "‘1‘ ist Zeuge für Zerlegbarkeit";

        RabinMillerPrime[p_, a_] := Module[{s, u, fertig, potenz},
            s = IntegerExponent[p - 1, 2];
            u = (p - 1)/2^s;
            potenz = PowerMod[a, u, p];
            If[potenz == 1,
              Message[RabinMillerPrime::"pseudoprime", a]; Return[True]];
            fertig = False;
            While[Not[fertig] && u < p - 1,
              If[potenz == p - 1, fertig = True,
                u = 2 * u; potenz = PowerMod[potenz, 2, p]]];
            If[fertig, Message[RabinMillerPrime::"pseudoprime", a],
              Message[RabinMillerPrime::"zusammengesetzt", a]
            ];
            fertig
          ]

        RabinMillerPrime[p_] := Module[{a},
            a = Random[Integer, {2, p - 1}];
            RabinMillerPrime[p, a]
          ]
```

Das Ergebnis von `RabinMillerPrime[`*p*`,`*a*`]` ist `False`, falls gezeigt wurde, daß *a* Zeuge für die Zerlegbarkeit von *p* und damit *p* zusammengesetzt ist, während `True` besagt, daß *p* strenge Pseudoprimzahl (zur Basis *a*) ist. Als Seiteneffekt wird eine Meldung ausgegeben, welche den Zeugen für die Zerlegbarkeit bzw. die Pseudoprimeigenschaft benennt.

Wir testen `RabinMillerPrime`:

```
In[2]:= RabinMillerPrime[1234567, 2]

RabinMillerPrime::"zusammengesetzt": 2 ist Zeuge für Zerlegbarkeit
Out[2]= False
```

Die Funktion `Prime[`*n*`]` berechnet die *n*-te Primzahl. Die folgende Zahl ist natürlich zusammengesetzt:

```
In[3]:= RabinMillerPrime[Prime[10^9] * Prime[2^30], 2]
```

```
RabinMillerPrime :: "zusammengesetzt": 2 ist Zeuge für Zerlegbarkeit
Out[3]= False
```

Nun führen wir den Rabin-Miller-Test mit einigen Carmichaelzahlen durch:

```
In[4]:= RabinMillerPrime[2465,2]

RabinMillerPrime :: "zusammengesetzt": 2 ist Zeuge für Zerlegbarkeit
Out[4]= False

In[5]:= RabinMillerPrime[41041,2]

RabinMillerPrime :: "zusammengesetzt": 2 ist Zeuge für Zerlegbarkeit
Out[5]= False

In[6]:= RabinMillerPrime[825265,2]

RabinMillerPrime :: "zusammengesetzt": 2 ist Zeuge für Zerlegbarkeit
Out[6]= False
```

und sehen, daß immer bereits $a = 2$ ein Zeuge für die Zerlegbarkeit ist.

Unter Zuhilfenahme der Funktion `NextPrime` erzeugen wir eine große Primzahl p:

```
In[7]:= p = NextPrime[10^100]
Out[7]= 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            267
```

und zeigen, daß kein Zeuge $a \in \{2,\ldots,10\}$ für die Zerlegbarkeit von p existiert:

```
In[8]:= Union[Table[RabinMillerPrime[p,a],{a,2,10}]]

RabinMillerPrime :: "pseudoprime": strenge Pseudoprimzahl zur Basis 2

RabinMillerPrime :: "pseudoprime": strenge Pseudoprimzahl zur Basis 3

RabinMillerPrime :: "pseudoprime": strenge Pseudoprimzahl zur Basis 4

General :: "stop": Further output of RabinMillerPrime :: "pseudoprime"
will be suppressed during this calculation
Out[8]= {True}
```

Schließlich bilden wir das Produkt p zweier großer Primzahlen:

```
In[9]:= p = NextPrime[10^100] * NextPrime[10^101]
Out[9]= 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            26730000000000000000000000000000000000000000000000000000000000000000000000000
            801
```

Ruft man die Funktion `RabinMillerPrime[p]` nur mit einem Argument p auf, so wird eine Zufallszahl $a \in \{2,\ldots,p-1\}$ ausgesucht. Für unser p erhalten wir:

```
In[10]:= RabinMillerPrime[p]
```

```
RabinMillerPrime :: "zusammengesetzt":
73665564733037105577 <<160>> 69185668341765554143 ist Zeuge für Zerleg-
barkeit
Out[10]= False
```

d. h., wir haben auf Anhieb einen Zeugen für die Zerlegbarkeit von p gefunden. □

Man kann zeigen, daß die Wahrscheinlichkeit, daß eine zusammengesetzte Zahl p den Rabin-Miller-Test mit zufällig ausgewählter Basis a besteht, höchstens 1/4 ist. Daher kann man durch mehrfache Anwendung des Tests die Fehlerwahrscheinlichkeit beliebig klein machen. Führt man beispielsweise den Rabin-Miller-Test hundert Mal durch, so beträgt die Fehlerwahrscheinlichkeit nur noch

```
In[11]:= 0.25^100
Out[11]= 6.22302 × 10^-61
```

Einen derartigen Algorithmus, der ein Ergebnis nicht mit Sicherheit, sondern mit (beliebig) großer Wahrscheinlichkeit erzielt, nennt man einen *probabilistische Algorithmus.*

4.7 Ergänzende Bemerkungen

In der Algebra wird die Konstruktion, mit der wir $\mathbb{Z}_p$ aus $\mathbb{Z}$ konstruiert haben, allgemeiner betrachtet. Man nennt $\mathbb{Z}_p$ den *Quotientenring* $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$.

Der kleine Satz von Fermat ist eine Folge des *Satzes von Lagrange* aus der Gruppentheorie: Die multiplikative Gruppe $\mathbb{Z}_p^*$ hat die Ordnung $p-1$, und die Ordnung jedes Elementes a ist ein Teiler von $p-1$ (Folgerung 4.9). Satz 4.13 zeigt, daß $\mathbb{Z}_p^*$ eine zyklische Gruppe ist.

Algorithmen, deren Komplexität sich durch ein Polynom in der Eingabelänge ausdrücken lassen, heißen *polynomiale Algorithmen.* Es sind bis heute keine polynomialen Algorithmen für die Faktorisierung ganzer Zahlen, für die Bestimmung modularer Quadratwurzeln als auch für die Bestimmung modularer Logarithmen bekannt. Es gibt zwar gute Gründe anzunehmen, daß es solche Algorithmen gar nicht gibt, aber keinen Beweis hierfür. Diese Fragen werden in der Komplexitätstheorie [BDG1988] untersucht.

Es gibt nicht nur probabilistische Algorithmen zur Primzahlsuche, sondern auch *deterministische Algorithmen*, welche eine höhere Effizienz haben als der triviale Algorithmus durch Probedivision. Algorithmen dieser Art sowie weitere probabilistische Verfahren können beispielsweise in [Buc1999], [For1996], [GG1999] sowie in [Rie1994] nachgelesen werden.

Wer bei der Primzahlsuche ganz sicher gehen will, kann in *Mathematica* auf die wesentlich langsamere Funktion `ProvablePrimeQ[n]` im Package `NumberTheory'PrimeQ'` zurückgreifen, welche zum Nachweis der Primalität *elliptische Kurven* benutzt.

ÜBUNGSAUFGABEN

4.1 *Führen Sie die Details der Übertragung der Ringeigenschaften in Satz 4.3 aus.*

4.2 *Zeigen Sie, daß ein Ring mit einem Nullteiler kein Körper sein kann.*

4.3 *Zeigen Sie: In $\mathbb{Z}_p$ gilt die Kürzungsregel*

$$a \cdot x \equiv a \cdot y \ (\mathrm{mod}\ p) \quad \Rightarrow \quad x \equiv y \ (\mathrm{mod}\ p)$$

genau dann, wenn $\gcd(a,p) = 1$ *ist.*

4.4 *Zeigen Sie die Periodizität der modularen Quadratfunktion. Was kann man über die Periode sagen?*

4.5 *Programmieren Sie die Lösung des chinesischen Resteproblems aus Satz 4.7.*

4.6 *Zeigen Sie: Für* $\gcd(p,q) = 1$ *gilt*

$$x \equiv a \ (\mathrm{mod}\ p) \text{ und } x \equiv a \ (\mathrm{mod}\ q) \quad \Leftrightarrow \quad x \equiv a \ (\mathrm{mod}\ p \cdot q)\,.$$

Wir kann man dies durch Induktion verallgemeinern?

4.7 *Zeigen Sie: Ein Polynom $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$ mit Koeffizienten $a_k \in \mathbb{K}$ eines Körpers $\mathbb{K}$ hat höchstens n Nullstellen in $\mathbb{K}$.*

Insbesondere: Sei $p \in \mathbb{P}$. Die Gleichung $x^2 = 1$ hat in $\mathbb{Z}_p$ genau zwei Lösungen $x \equiv \pm 1$ (mod p).

4.8 *Programmieren Sie die Funktion* `powermod` *iterativ.*

4.9 *Programmieren Sie analog zu der Funktion* `powermod` *die schnelle Potenzierung* `power[z,n]` $= z^n$ *ganzer Zahlen $z,n \in \mathbb{Z}$.*

4.10 *Sei G eine kommutative Gruppe mit Einselement $e \in G$, und sei $a \in G$. Sei $S := \{m \geqq 1 \mid a^m = e\}$ und* $\mathrm{ord}(a) = \min S$ *die Ordnung von a. Zeigen Sie: Ist $a^m = e$, so ist* $\mathrm{ord}(a) \mid m$.

4.11 *Zeigen Sie: Sei* $p \in \mathbb{P}$ *und* $a \in \mathbb{Z}_p^*$. *Aus* $\mathrm{ord}(a) = m$ *folgt* $\mathrm{ord}(a^r) = \frac{m}{\gcd(r,m)}$.

4.12 *Sei* $\mathrm{GF}(9) = \{a+ib \mid a,b \in \mathbb{Z}_3, i^2 = -1\}$. *Dies ist eine Menge mit* 9 *Elementen. Zeigen Sie, daß* GF(9) *ein Körper ist. Finden Sie ein erzeugendes Element, d. h., ein Element* $a \in \mathrm{GF}(9)$ *mit* $\mathrm{ord}(a) = 8$.

4.13 *Implementieren Sie eine Funktion* `ModularLog`, *welche den modularen Logarithmus bestimmt, d. h., eine Lösung (bzw. die Lösungen)* x *der Gleichung* $a^x \equiv y \pmod{p}$. *Lösen Sie das Problem aus Sitzung 4.7. Vergleichen Sie mit der Funktion* `MultiplicativeOrder`.

4.14 *Bestimmen Sie jeweils die kleinste Carmichaelzahl mit 4 bzw. 5 verschiedenen Faktoren.*

4.15 *Finden Sie eine mindestens zehnstellige Carmichaelzahl.*

4.16 *Zeigen Sie die ersten beiden Aussagen aus Beispiel 4.18 mit Mathematica.*

4.17 *Untersuchen Sie die Fermatschen Zahlen* $F_n = 2^{2^n} + 1$ *für* $n = 1,\ldots,10$ *mit dem Rabin-Miller-Test auf Teilbarkeit. Mathematica findet die Primfaktorzerlegungen bis* $n = 6$ *in vernünftiger Zeit. Bestimmen Sie diese.*