Zahlentheoretische Algorithmen

Inhaltsverzeichnis

Einleitung

Der Versuch über Zahlentheorie setzt an manchen Stellen grundlegende Kenntnisse der Algebra (Gruppen- und Ringtheorie) sowie der linearen Algebra voraus.

Das Material ist so ausführlich wie nötig und so knapp wie möglich zusammengestellt, um hier sämtliche Grundlagen der folgenden Versuche zu legen und doch das erste Verständnis nicht zu erschweren. An Stellen, die interessanten Ausblick auf benachbarte Probleme und Fragestellungen bieten, findet der interessierte Praktikumsteilnehmer (hoffentlich) ausreichend Literaturhinweise, um tiefer in die Materie vorzudringen.

Dem Praktikumsteilnehmer steht eine Java-Bibliothek zur Verfügung, mit der er eine vollständige Langzahlarithmetik erhält. Darin sind die für unsere Zwecke effizientesten Algorithmen verwendet.

Zur Abschätzung des Aufwandes der Algorithmen führen wir eine Größe $M(n)$ ein, die ein Komplexitätsmaß für die Multiplikation zweier Zahlen $|a|, |b| \leq n$ ist. Ihr Wert hängt ab vom verwendeten Multiplikationsalgorithmus der Langzahlarithmetik und liegt zur Zeit bei mindestens $\mathrm{M}(n) = \mathrm{O}(\log n \cdot \log\log n \cdot \log\log\log n)$[1]. Insbesondere zum Verständnis asymmetrischer Konzelationssysteme und von Signatursystemen stellen wir zunächst Grundlagen der Komplexitätstheorie bereit.

1 Grundlagen der Komplexitätstheorie

1.1 O-Notation

Zur rechnerunabhängigen Laufzeitbeschreibung eines Algorithmus dient die O-Notation (sprich „groß O“), die erstmals von P. Bachmann und E. Landau verwendet wurde.

Definition O-Notation
Gegeben sei eine Funktion $f : \mathbb{N} \to \mathbb{R}$. Dann gilt:

$$f(n) = \mathbf{O}(g(n)) : \quad \Leftrightarrow \quad \lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty$$

$$f(n) = \mathbf{o}(g(n)) : \quad \Leftrightarrow \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

[1] Schönhage-Strassen Integermultiplikation z.B. in [AhHU_74]

Funktionen wie z.B. $f_1(n) = n + 10$ und $f_2(n) = 10^6 \cdot n + 10^4$ fallen in dieselbe Komplexitätsklasse $O(n)$. $f_3(n) = n^2 - n + 1$ dagegen fällt in die Klasse $O(n^2)$ und für $f_4(n) = 2^n - n^2 - 1$ gilt $f_4 = O(2^n)$.

Sei $T(n)$ die Funktion, die den zeitlichen Aufwand eines Algorithmus A beschreibt, so spricht man bei $T = O(n)$ von linearem, bei $T = O(n^2)$ von quadratischem, bei $T = O(n^3)$ von kubischem, allgemein bei $T = O(n^t)$, t konstant, von polynomialem und bei $T = O(t^{h(n)})$, t konstant und größer Eins, h ein Polynom, von exponentiellem Aufwand von A.[2]

1.2 Komplexität und NP-Vollständigkeit

Die Komplexitätstheorie klassifiziert Probleme danach, welchen (zeitlichen) Aufwand ihre Lösung mindestens erfordert. Sie beschäftigt sich mit der theoretischen unteren Schranke eines Problems, die von keinem Algorithmus unterschritten werden kann.

Offensichtlich gibt es Probleme, die von deterministischen Algorithmen mit polynomialem Aufwand systematisch lösbar sind (z.B. Sortieren von Listen); wir nennen sie im folgenden **deterministisch polynomial** lösbar.

Probleme, die nicht mit polynomialem, sondern nur mit exponentiellem Aufwand von deterministischen Algorithmen lösbar sind, heißen auch exponentiell lösbar. Sie sind aber schon bei vergleichsweise kleinen Eingaben, selbst mit schnellsten und hochparallel arbeitenden Rechnern, praktisch nicht zu lösen. Man verwechsele sie aber nicht mit im TURING'schen Sinne nicht-berechenbaren Problemen (Halteproblem)[3]. Die praktische Unlösbarkeit gründet nicht in der Natur des Problems, wie beim Halteproblem, sondern in der zeitlichen Beschränkung (der Resourcen) dessen, der es zu lösen versucht.

Ein exponentiell lösbares Problem kann immer durch Probieren aller Lösungsmöglichkeiten gelöst werden, besitzt also eine höchstens abzählbar unendlich große Menge möglicher Lösungen. Die Probleme, für die jeder Lösungskandidat von einem deterministischen Algorithmus in polynomialer Zeit getestet werden kann, heißen **indeterministisch polynomial** lösbare Probleme. Denn ein indeterministischer Algorithmus braucht die richtige(n) Lösung(en) nur zu raten und kann sie in polynomialer Zeit verifizieren. Das führt zu folgender

Definition P, NP[4]
Die Menge aller deterministisch polynomial lösbaren Probleme wird mit **P**,
die Menge aller indeterministisch polynomial lösbaren Probleme mit **NP** bezeichnet.

Das Problem der Faktorisierung großer Zahlen N ist ein Beispiel für ein indeterministisch polynomial lösbares Problem. Die Größe g dieses Problems ist die Länge der Zahl N in Bit (Bezeichnung: $g = |N|$). Zwar ist bis heute kein deterministischer Algorithmus bekannt, der wenigstens einen Faktor jeder zusammengesetzten Zahl N findet und dessen Aufwand durch ein Polynom in $|N|$ beschränkt ist, aber selbstverständlich kann ein indeterministischer

[2] Beispielhafter Vergleich von Komplexitätsklassen bei [GaJo_79]
[3] siehe [Paul_78]
[4] Vgl. dazu das Kapitel Komplexitätstheorie bei [Denn_82, S. 30-34] sowie [GaJo_79]

Algorithmus die Teiler von N (d.h. die Lösungsmöglichkeiten des Problems) raten und sie mittels Division in polynomialer Zeit verifizieren.

Es folgt sofort, daß $\mathbf{P} \subseteq \mathbf{NP}$, denn deterministische Algorithmen sind Spezialfälle indeterministischer Algorithmen. Ob sogar $\mathbf{P} = \mathbf{NP}$, ist bis heute ungeklärt, aber unwahrscheinlich.

Cook begann 1971 die weitere Strukturierung der Menge **NP** durch die Definition der **NP**-vollständigen Probleme. Diese sind die Probleme in **NP**, auf die sich jedes andere Problem in **NP** mit polynomialem Aufwand reduzieren läßt. Die **NP**-vollständigen Probleme bilden die Klasse der „schwierigsten“ Probleme in **NP**.

Da man trotz intensivster Bemühungen bisher kein polynomiales Lösungsverfahren für eines (und damit alle) dieser Probleme gefunden hat, wird $\mathbf{P} \neq \mathbf{NP}$ allgemein vermutet. Dies ist aber bis heute unbewiesen.

1.3 Faktorisierung großer Zahlen

Für Zahlen n mit mehr als 110 Dezimalstellen ist der Faktorisierungsalgorithmus *general number field sieve* der schnellste bisher bekannte. Seine Laufzeit beträgt:

$$\mathrm{L}_{general\ number\ field\ sieve}(n) = \exp(1,923 \cdot \sqrt[3]{\ln(n)(\ln\ln(n))^2})$$

Bei Zahlen mit etwa 130 Dezimalstellen benötigt der Faktorisierungsalgorithmus *general number field sieve* für 5 weitere Dezimalstellen doppelten Rechenaufwand.

In der Praxis sollte n heutzutage im Bereich von 768 bis 4096 Bit Länge gewählt werden.

Seit dem Bau der ersten digitalen Rechenanlagen (also seit etwa 1940) wächst der Quotient Rechenleistung/Anschaffungskosten exponentiell — in den letzten Jahrzehnten erfolgte etwa eine Verdoppelung pro Jahr. Dies wird wohl anhalten, bis irgendwann technologische Grenzen erreicht werden. Solche sind zumindest für das nächste Jahrzehnt nicht in Sicht.

2 Grundlagen einer modularen Arithmetik

2.1 Division mit Rest und Teilbarkeit ganzer Zahlen

Satz 1 (Division mit Rest)

Sind a,m ganze Zahlen, $m > 0$, dann existieren eindeutig Quotient q und Rest r, so daß gilt:

$$a = m \cdot q + r \quad \text{wobei } (0 \leq r < m,\ \ q, r \in \mathbb{Z})$$

Beispiele:

$$a = 7,\ m = 3 \quad \text{ergibt} \quad r = 1,\ q = 2$$
$$a = -7,\ m = 3 \quad \text{ergibt} \quad r = 2,\ q = -3$$

Definition Teilbarkeit

Seien $a, m, q \in \mathbb{Z}$ mit $a = m \cdot q$ (d.h. $r = 0$). Dann heißt a **teilbar durch** m oder **Vielfaches von** m und man schreibt auch $m|a$ (sprich m **teilt** a).

2.2 Kongruenz und elementare Eigenschaften

Definition Kongruenz
Seien $m > 0$, $a, b \in \mathbb{Z}$. Man nennt a **kongruent (zu)** b modulo m genau dann, wenn $m|(a-b)$ gilt, und schreibt dies als:

$$a \equiv b \,(\mathrm{mod}\ m) \text{ oder gleichwertig } a \equiv_m b$$

Anderenfalls heißen a und b **inkongruent** mod m: $a \not\equiv b \,(\mathrm{mod}\ m)$.

Man rechnet Reflexivität, Symmetrie und Transitivität dieser Relation leicht nach und findet, daß m-Kongruenz eine Äquivalenzrelation auf $\mathbb{Z}$ ist. Sie zerlegt $\mathbb{Z}$ in disjunkte Klassen, die sogenannten **Restklassen modulo** m.

Beachte, daß das Kürzel **mod** hier eine Relation auf $\mathbb{Z}$ bezeichnet und nicht mit dem von vielen Programmiersprachen her bekannten mod-Operator (siehe Kapitel 2.4) verwechselt werden darf.

Beispiel: 8 ist kongruent 3 modulo 5, geschrieben $8 \equiv 3 \,(\mathrm{mod}\ 5)$ oder kürzer $8 \equiv_5 3$.

2.3 Vollständige Restsysteme

Wie bei Äquivalenzrelationen üblich, wählt man für jede Restklasse einen Repräsentanten aus und erhält so ein vollständiges Restsystem.

Definition Vollständiges Restsystem
Eine Menge $S := \{a_0, a_1, \ldots, a_{m-1}\}$, mit paarweise modulo m inkongruenten Elementen $a_0, a_1, \ldots, a_{m-1}$ heißt **vollständiges Restsystem modulo** m.
Das heißt, in einem vollständigen Restsystem S ist jede Restklasse durch genau ein Element a_i dieser Klasse vertreten.

Im weiteren Verlauf werden sich zwei vollständige Restsysteme als besonders nützlich erweisen.

Wie in Kapitel 2.2 beschrieben, bedeutet Kongruenz zweier Zahlen a und b modulo m gerade, daß sie bei Division durch m denselben Rest lassen. Dieser eindeutige Rest aller Elemente einer Klasse ist jeweils das kleinste nichtnegative Element einer Restklasse. Zeichnet man dieses Element in jeder Klasse aus, erhält man das **kleinste nichtnegative Restsystem** modulo m:

$$S_m := \{0, 1, \ldots, m-1\}$$

Wählt man das betragskleinste Element jeder Restklasse, so erhält man das **betragskleinste Restsystem** modulo m:

$$S'_m := \begin{cases} \left\{-\frac{m}{2}+1, \ldots, 0, \ldots, \frac{m}{2}\right\} & \text{falls } m \text{ gerade} \\ \left\{-\frac{m-1}{2}, \ldots, 0, \ldots, \frac{m-1}{2}\right\} & \text{falls } m \text{ ungerade} \end{cases}$$

Beispiele:

$$S_2 = \{0, 1\},\ S_2' = \{0, 1\}$$
$$S_7 = \{0, 1, 2, 3, 4, 5, 6\},\ S_7' = \{-3, -2, -1, 0, 1, 2, 3\}$$

2.4 mod- und div-Operator

Definition mod-Operator
Sei $a \in \mathbb{Z}$, $m \in \mathbb{N}$, S ein vollständiges Restsystem **mod** m.
Die Zuordnung des Elementes a einer Restklasse zu deren Repräsentanten $a' \in S$ geschieht durch die Abbildung

$$\mathbf{mod} : \mathbb{Z} \to S$$
$$a \mapsto a',\ a\,\mathbf{mod}\,m := a'$$

Während mod bisher eine Äquivalenzrelation auf $\mathbb{Z}$ bezeichnete, verwenden wir nun dasselbe Kürzel mod, um den Operator zu bezeichnen, der jedem Element aus $\mathbb{Z}$ seinen Repräsentanten aus S zuordnet. Welchen Wert der Operator mod also zurückgibt, hängt nicht allein von a und m ab, sondern auch vom zugrundegelegten Repräsentantensystem.

Wir unterscheiden mod als Kongruenzbezeichner bzw. als Operatorbezeichner durch den Kontext, in dem er verwendet wird:

1. Steht mod m *hinter einer Zahl*, so bezeichnet es den Operator (wie üblich in Postfixnotation). Beispiel: $a \bmod m$
2. Steht $(\bmod m)$ *hinter einer Kongruenz*, so bezeichnet es die Kongruenzrelation $(\bmod m)$. Beispiel: $a \equiv b \pmod m$

Zusammenhang: $a \equiv b \pmod m \Leftrightarrow a \bmod m = b \bmod m$

Während wir also den Rest einer Division mithilfe des mod-Operators bezeichnen, fehlt uns noch eine Bezeichnung für den Quotienten. Das holen wir nach in der folgenden

Definition div-Operator
Sei $a \in \mathbb{Z}$, $m \in \mathbb{N}$, und $a = m \cdot q + a'$ mit $a' \in \{0, 1, \ldots, m-1\}$:

$$\mathbf{div} : \mathbb{Z} \to \mathbb{Z}$$
$$a \mapsto q,\ a\,\mathbf{div}\,m := q$$

Unsere Definition der Operatoren div und mod entspricht für $a \geq 0$ der Definition, die in den meisten Programmiersprachen gewählt wurde. Von Übersetzer zu Übersetzer verschieden sind die Ergebnisse dagegen bei negativem Argument a.

2.5 Modulare Addition und Multiplikation

Man definiert Addition und Multiplikation auf den Restklassen wie folgt:

Definition Addition und Multiplikation
Sei $m \in \mathbb{N} \setminus \{1\}$:

$$\begin{aligned} a \bmod m +_m b \bmod m &:= (a+b) \bmod m \\ a \bmod m \cdot_m b \bmod m &:= (a \cdot b) \bmod m \end{aligned}$$

Bemerkung: Machen Sie sich eventuell anhand Kapitel 2.3 klar, daß diese Definitionen repräsentanten-unabhängig sind.

Es gilt ($1 \bmod m = 1 \neq 0$) für jedes $m \in \mathbb{N} \setminus \{1\}$. Also ist ($\bmod m$) ein Homomorphismus $\mathbb{Z} \to S_m$, genauer ein Ring-Homomorphismus (weil $\mathbb{Z}$ ein Ring ist, Homomorphiesatz der Ringtheorie[5]). Das Bild $(S_m, +_m, \cdot_m)$ ist also ein kommutativer Ring, der **Restklassenring modulo** m, den wir mit $\mathbb{Z}_m$ bezeichnen werden.[6]

Wir haben nun die bekannte Struktur, beschrieben in Assoziativ-, Kommutativ- und Distributivgesetz der arithmetischen Operatoren $+_m$ und $\cdot_m$ wie gewohnt wieder zur Verfügung und können allein mit den Repräsentanten rechnen.

Gleichungen in $\mathbb{Z}_m$ schreiben wir als Kongruenzen (die Ähnlichkeit der Zeichen $\equiv$ und $=$ deutet die Ähnlichkeit der Rechengesetze an). Statt $a\,(\bmod m)$ schreiben wir in Kongruenzen nur a; wie auch für die Operatoren $+_m$, $-_m$, $\cdot_m$, $/_m$ entsprechend kurz $+$, $-$, $\cdot$, $/$. Wenn nichts anderes gesagt ist, so sei $\mathbb{Z}_m$ durch S_m repräsentiert.

Nach jeder Operation können wir das (Zwischen-)Ergebnis ($\bmod m$) reduzieren und so ein Anschwellen der Zwischenresultate einer Rechnung verhindern. Die dadurch gewonnene Rechenzeit und der verringerte Zwischenspeicheraufwand erklären die Bedeutung der modularen Arithmetik für rechenintensive Computeranwendungen.

Im Diagramm (Bild 1) ist das Schema des mod-Homomorphismus dargestellt. Unabhängig, ob Lösungsweg 1 oder 2 eingeschlagen wird, erhält man bei „Ziel“ dasselbe Ergebnis. Man sagt, das Diagramm kommutiert. (siehe Beispiel)

[5] Etwas Algebra findet man z.B. bei [Horn_76]

[6] Gebräuchlich ist auch die Bezeichnung $\mathbb{Z}/\mathbb{Z}_m$. Dies erinnert noch besser daran, daß es sich um den Restklassenring R/J eines kommutativen Ringes R (hier $R = \mathbb{Z}$) nach einem Ideal $J \cdot \mathbb{Z}$ (hier $J = m \cdot \mathbb{Z}$) handelt. Ein Ideal J in einem Ring R ist eine nichtleere Teilmenge von R mit:
1. $a, b \in J \Rightarrow a - b \in J$ 2. $a \in J,\ r \in R \Rightarrow ar, ra \in J$
Ein Beispiel für ein Ideal sind die geraden Zahlen innerhalb des Rings der ganzen Zahlen.

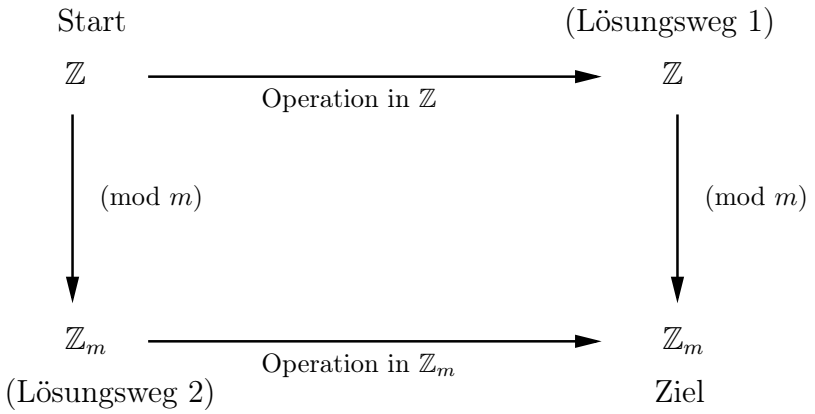

Abbildung 1: Schema des Homomorphismus (mod m)

Beispiel: $20 \cdot 9 + 30 \pmod{13}$

$$\begin{aligned} \text{Lösungsweg 1:} \quad & (20 \cdot 9 + 30) \bmod 13 \\ = \; & (180 + 30) \bmod 13 \\ = \; & 210 \bmod 13 \\ \equiv \; & 2 \pmod{13} \end{aligned}$$

$$\begin{aligned} \text{Lösungsweg 2:} \quad & 20 \cdot 9 + 30 \pmod{13} \\ \equiv \; & (20 \bmod 13) \cdot ((-4) \bmod 13) + (4 \bmod 13) \\ \equiv \; & (-6) \cdot (-4) + 4 \\ \equiv \; & (24 \bmod 13) + 4 \\ \equiv \; & (-2) + 4 \\ \equiv \; & 2 \pmod{13} \end{aligned}$$

Die modulare Arithmetik wird zu unserem Handwerkszeug gehören und wir geben mit dem **SMA** ein erstes Beispiel ihrer Anwendung.

2.6 Square & Multiply Algorithmus (SMA)

Hier sei zunächst das Prinzip einer effizienten modularen Exponentiation von $z := a^b (\bmod m)$ vorgestellt, die dann zum *Square & Multiply* Algorithmus **SMA** führt:

1. Rückführung der Exponentiation auf Multiplizieren und Quadrieren:
   a) $b = (b_{t-1}, b_{t-2}, \ldots, b_0)$ sei die Binärdarstellung von b mit $b_{t-1} = 1$ (frei von führenden Nullen)
   b) Gewinne aus der Binärfolge eine Folge SX... durch

$$\text{Ersetzen jeder Binärstelle } b_i \text{ durch } \begin{cases} \text{S falls } b_i = 0 \\ \text{SX falls } b_i = 1 \end{cases}$$

c) Streiche nun das führende (b_{t-1} entsprechende) SX von der entstandenen Folge fort und fasse jedes S als Quadrierbefehl und jedes X als (Multiplizier-mit-a)-Befehl auf.

2. Werte die Operatorenfolge von links nach rechts aus und reduziere gegebenenfalls nach jeder Operation modulo m.

Beispiel:

$z \equiv 3^{13} \pmod{11}$

1. $13 = (1101)_2 \; -_{(ii)} \rightarrow$ SX SX S SX $-_{(iii)} \rightarrow$ SX S SX
2. $3 \; -_{\mathrm{S}} \rightarrow 9 \; -_{\mathrm{X}} \rightarrow 27 \equiv_{11} 5 \; -_{\mathrm{S}} \rightarrow 25 \equiv_{11} 3 \; -_{\mathrm{S}} \rightarrow 9 \; -_{\mathrm{X}} \rightarrow 27 \equiv_{11} 5$

Ergebnis: $z \equiv 5 \pmod{11}$

Der **SMA** weist demgegenüber noch eine kleine Verbesserung auf, nämlich, daß er die Bitfolge b von rechts nach links (die viel leichter zu erzeugen ist[7]) einliest und nicht umgekehrt, wie oben:

Algorithmus SMA (Square & Multiply Algorithmus)

Eingabe: $m \in \mathbb{N} \setminus \{1\}$, $a \in \mathbb{Z}$, $b \in \mathbb{N}$
Ausgabe: $z := a^b \pmod{m}$
Aufwand: $\mathrm{O}(\mathrm{M}(m) \cdot \log b)$

1) $u := a$, $v := b$, $z := 1$
2) wiederhole solange, wie $(v > 0)$ { Invariante: $a^b \equiv z \cdot u^v \pmod{m}$}
3) falls (v ungerade), berechne $z := u \cdot z \pmod{m}$
4) $u := u^2 \pmod{m}$
5) $v := v \operatorname{div} 2$
6) gib zurück (z) { $a^b \equiv z \pmod{m}$ }

Aufgaben:

2.1 Bestimmen Sie $2^{32} - 1 \,(\mathrm{mod}\, 11)$ und $3^{53} + 1 \,(\mathrm{mod}\, 7)$ jeweils repräsentiert durch betragskleinste Reste.

2.2 Implementieren Sie den Square & Multiply Algorithmus in den `Programmrahmen_SMA` und überprüfen Sie Ihre Ergebnisse aus der vorherigen Aufgabe. Achten sie bei der Implementierung (wie auch bei allen folgenden Implementierungsaufgaben) darauf, daß der Algorithmus wirklich mit großen Zahlen (Länge $> 1000bit$) arbeiten kann.

3 Der EUKLID'sche Algorithmus

3.1 Definition des größten gemeinsamen Teilers

Wie der Name schon sagt, ist der größte gemeinsame Teiler g zweier ganzer Zahlen a, b (nicht beide gleich 0) spezifiziert durch zwei Eigenschaften:

[7] Vgl. dazu etwa die Entwicklung des **SMA** bei [Lips_81, S. 222-224]

1. $g|a$ und $g|b$, d.h. g ist gemeinsamer Teiler von a und b
2. $c|a$ und $c|b \Rightarrow c|g$, d.h. g ist größter gemeinsamer Teiler von a und b.

Satz 2 (Eindeutigkeit des größten gemeinsamen Teilers)
Der (positive) größte gemeinsame Teiler zweier Zahlen ist (wenn er überhaupt existiert) eindeutig bestimmt.

Es bezeichne nun **ggT(a,b)** den größten gemeinsamen Teiler der Zahlen a, b (nicht beide gleich Null) oder auch kürzer **(a,b)**, wenn Verwechslungen der Notation ausgeschlossen sind.

Satz 3 (Existenz und Konstruktion des ggT)
Seien $a, b \in \mathbb{Z}$ nicht beide gleich Null. Dann gilt:

1. ggT(a, b) existiert.
2. Es existieren gewisse Zahlen $s, t \in \mathbb{Z}$, genannt **Cofaktoren** von a bzw. b derart, daß gilt:
$$s \cdot a + t \cdot b = \text{ggT}(a, b) \qquad \text{(Satz von EUKLID)}$$
3. $g = \text{ggT}(a, b)$, s, t werden vom erweiterten EUKLID'schen Algorithmus **EUKLID** berechnet.

3.2 Der EUKLID'sche Algorithmus

Algorithmus EUKLID (Erweiterter EUKLID'scher Algorithmus)

Eingabe: $a, b \in \mathbb{Z}$ nicht beide gleich Null

Ausgabe: s, t, g derart, daß $g = \text{ggT}(a, b) = a \cdot s + b \cdot t$

Aufwand: $O(M(\max\{|a|, |b|\}) \cdot \log(\max\{|a|, |b|\}))$

1) $(a_0, a_1) := (a, b)$; $(s_0, s_1) := (1, 0)$; $(t_0, t_1) := (0, 1)$
2) solange, wie $(a_1 > 0)$ { Invariante: $a_0 = s_0 \cdot a + t_0 \cdot b$ und $a_1 = s_1 \cdot a + t_1 \cdot b$ }
3) $q := a_0$ div a_1
4) $(a_0, a_1) := (a_1, a_0 - a_1 \cdot q)$
5) $(s_0, s_1) := (s_1, s_0 - s_1 \cdot q)$
6) $(t_0, t_1) := (t_1, t_0 - t_1 \cdot q)$
7) gib zurück ($s := s_0$, $t := t_0$, $g := a_0$)

Beispiel:

Berechne ggT(228,612) sowie die Cofaktoren:

Iteration	a_0	a_1	q	s_0	s_1	t_0	t_1
0	228	612	–	1	0	0	1
1	612	228	0	0	1	1	0
2	228	156	2	1	−2	0	1
3	156	72	1	−2	3	1	−1
4	72	12	2	3	−8	−1	3
5	**12**	0	6	**−8**	51	**3**	−19

also folgt: $\mathrm{ggT}(228, 612) = 12 = -8 \cdot 228 + 3 \cdot 612$

Um lediglich den $\mathrm{ggT}(a, b)$ zu berechnen, genügt also eine vereinfachte Version von **EUKLID**, die die Cofaktoren nicht berechnet. Es gibt aber wichtige Anwendungen, in denen mindestens einer der Cofaktoren benötigt wird.

Aufgaben:

3.1 Bestimmen Sie ggT(855,990).

3.2 Berechnen Sie ggT(711,549) sowie deren Cofaktoren.

3.3 Schreiben Sie eine Prozedur, die den größten gemeinsamen Teiler zweier ganzer Zahlen und deren Cofaktoren berechnet. Verwenden Sie dazu den `Programmrahmen_Euklid`.

4 Die vollständige modulare Arithmetik

4.1 Kürzungsregel für Kongruenzen

Während Subtraktion und Division noch nicht besprochen wurden, sei hier schon die Kürzungsregel in $\mathbb{Z}_m$ angegeben, die wohl am gewöhnungsbedürftigsten gegenüber dem Rechnen in $\mathbb{Z}$ ist:

Lemma 1 („Kürzungsregel“)[8]

$$m \in \mathbb{N},\ a, b, c \in \mathbb{Z}: \quad a \cdot c \equiv b \cdot c \,(\mathrm{mod}\ m) \Leftrightarrow a \equiv b \left(\mathrm{mod} \frac{m}{\mathrm{ggT}(c, m)}\right)$$

Werden also beide Seiten einer Kongruenz durch denselben Faktor dividiert, und hat dieser einen gemeinsamen Teiler d mit dem Modul m, so reduziert sich m um eben diesen Teiler d.
Beispiel:

$45 \equiv 15 \pmod{10} \Leftrightarrow 5 \cdot 9 \equiv 5 \cdot 3 \pmod{10}$

Division durch 5 ergibt: $9 \equiv 3 \pmod 2$ denn $\dfrac{10}{\mathrm{ggT}(5, 10)} = \dfrac{10}{5} = 2$

Division durch 3 ergibt: $3 \equiv 1 \pmod 2$ denn $\dfrac{2}{\mathrm{ggT}(3, 2)} = \dfrac{2}{1} = 2$

4.2 Subtraktion und Division

Subtraktion und Division sind wie (in Ringen) üblich über additive bzw. multiplikative Inverse definiert. Wie diese in den beiden Fällen der Repräsentation von $\mathbb{Z}_m$ durch S_m bzw. S'_m aussehen, behandelt das Folgende:

[8] Beweis etwa bei [Bund_88, S. 82]

Additive Inverse:

Gegeben sei $a \in \mathbb{Z}_m$ in der jeweiligen Repräsentation, gesucht ist $-a \pmod m$:

$$\text{Wähle } -a(\bmod m) := \begin{cases} 0 & \text{falls } a = 0 \\ m - a & \text{sonst} \end{cases} \quad \text{nichtnegative Repräsentation}$$

$$\text{Wähle } -a(\bmod m) := \begin{cases} a & \text{falls } a = \frac{m}{2} \\ -a & \text{sonst} \end{cases} \quad \text{betragskleinste Repräsentation}$$

Beispiele:

nichtnegative Repräsentation: $-7 \equiv_{11} (11 - 7) \equiv_{11} 4$; $-0 \equiv_{11} 0$
betragskleinste Repräsentation: $-3 \equiv_6 3$, da $3 = \frac{6}{2}$; $-3 \equiv_7 -3$

Multiplikative Inverse:

Gegeben sei $a \in \mathbb{Z}_m$: gesucht a^{-1} derart, daß $a \cdot a^{-1} \equiv 1 \pmod m$

Wir schreiben die Kongruenz zunächst als Gleichung $a \cdot a^{-1} + m \cdot x = 1$ und erkennen, daß sich darin die Werte a^{-1} und x als Cofaktoren der Eingaben a und m aus dem EUKLID'schen Algorithmus ergeben, sofern $\text{ggT}(a, m) = 1$ gilt. Sind a und m teilerfremd, so erhält man die Inverse zu a also mittels des Algorithmus **EUKLID**.

Bemerkung:

Es gilt: $a^{-1} \pmod m$ existiert genau dann, wenn $\text{ggT}(a, m) = 1$. Falls $\text{ggT}(a, m) > 1$, so existiert a^{-1} in $\mathbb{Z}_m$ nicht! (Beweis etwa bei [Lips_81]).

Beispiele:

$x \equiv_{27} 22^{-1} \Rightarrow \text{ggT}(22, 27) = 1 = (-11) \cdot 22 + 9 \cdot 27 \Rightarrow x \equiv_{27} (-11) \equiv_{27} 22^{-1}$
$x \equiv_{27} 15^{-1} \Rightarrow \text{ggT}(15, 27) = 3 = 2 \cdot 15 - 27 \Rightarrow$ kein Inverses von 15 bezüglich 27

4.3 Das prime Restsystem mod *m*

Damit kommt eine Teilmenge von $\mathbb{Z}_m$ ins Blickfeld, die gerade aus diesen modulo m invertierbaren Elementen besteht:

Definition Primes Restsystem
$\mathbb{Z}_m^* := \{a \in \mathbb{Z}_m : \text{ggT}(a, m) = 1\}$ heißt **primes oder reduziertes Restsystem mod** m

Beispiel:

$\mathbb{Z}_{16}^* = \{1, 3, 5, 7, 9, 11, 13, 15\}$ sind alle teilerfremden Zahlen von 16

Definition und Satz 4
Das prime Restsystem mod m stellt also genau die Menge aller Teiler von 1 in $\mathbb{Z}_m$ dar. Seine Elemente heißen deswegen auch **Einheiten von** $\mathbb{Z}_m$ und ihre Gesamtheit bildet eine multiplikative Gruppe, die **Einheitengruppe** $\mathbb{Z}_m^*$ von $\mathbb{Z}_m$.[9]

Wie zu vermuten, bilden alle Vielfachen einer Einheit von $\mathbb{Z}_m$ gerade wieder $\mathbb{Z}_m$:

Lemma 2 (Vielfache einer Einheit)

$$\forall z \in \mathbb{Z}_m^* : \{i \cdot z \pmod m \mid i \in \{0, 1, \ldots, m-1\}\} = \mathbb{Z}_m$$

Dies folgt unmittelbar aus der Definition der Einheiten und der Tatsache, daß $\mathbb{Z}_m^*$ eine multiplikative Gruppe ist.

Wie groß diese Menge ist, hat die Mathematiker schon früh beschäftigt und führt zur Definition der EULER'schen ϕ-Funktion:

Definition EULER'sche ϕ-Funktion

$$\phi(m) := |\mathbb{Z}_m^*|$$

Ihre grundlegenden Eigenschaften beschreibt

Satz 5 (Eigenschaften der EULER'schen ϕ-Funktion)
Es sei $m = p_1^{r_1} \cdot p_2^{r_2} \cdot \ldots \cdot p_n^{r_n}$ die Primfaktorzerlegung von m:

i) $\forall m \in \mathbb{N} : \phi(m) = (p_1 - 1) \cdot p_1^{r_1 - 1} \cdot \ldots \cdot (p_n - 1) \cdot p_n^{r_n - 1}$,

Häufig verwendete Spezialfälle sind:

ii) $\forall p$ prim: $\phi(p) = p - 1$
iii) $\forall p$ prim, $r \in \mathbb{N}$: $\phi(p^r) = (p-1) \cdot p^{r-1}$
iv) $\forall p, q$ prim, $p \neq q$: $\phi(p \cdot q) = (p-1) \cdot (q-1)$ [10]

Beispiele:

$\phi(16) = \phi(2^4) = 1 \cdot 2^3 = 8$ gibt Anzahl der teilerfremden Elemente an
$\phi(693) = \phi(3^2 \cdot 7 \cdot 11) = (2 \cdot 3) \cdot (6 \cdot 1) \cdot (10 \cdot 1) = 360$

Mithilfe dieses Satzes kann die ϕ-Funktion also für Argumente berechnet werden, deren Primfaktorzerlegung bekannt ist. Umgekehrt kann die Primfaktorisierung einer Zahl $n = p \cdot q$ (p,q prim) leicht bestimmt werden, falls n und $\phi(n)$ gegeben sind [Hors_85, S. 187]. Ist die Primfaktorzerlegung von n unbekannt, kann die ϕ-Funktion bisher nur mit exponentiellem

[9] Beweis etwa bei [Bund_88, S. 53]
[10] Beweis etwa bei [Bund_88, S. 48]

Aufwand (und das heißt bei großen Argumenten praktisch gar nicht) systematisch berechnet werden.

EULER erzielte das für die modulare Arithmetik sehr wichtige Resultat, daß die Potenzen einer Zahl a in $\mathbb{Z}_m^*$ eine zyklische Folge höchstens der Periode $\phi(m)$ bilden:

Satz 6 (EULER)

$$m \in \mathbb{N}, a \in \mathbb{Z}, \mathrm{ggT}(m,a) = 1: \quad a^{\phi(m)} \equiv 1 \pmod{m} \ ^{11}$$

Bemerkung:

Durch Anwendung dieses Satzes kann die modulare Exponentiation vereinfacht werden. Dazu ist der Satz von EULER wie folgt zu verwenden:

Zerlegen Sie zunächst die Zahl $b = k \cdot \phi(m) + b \bmod \phi(m), k \in \mathbb{N}$. Dann gilt:

$$\begin{aligned} a^b &\equiv a^{k\cdot\phi(m)+b \bmod \phi(m)} \equiv \left(a^{\phi(m)}\right)^k \cdot a^{b \bmod \phi(m)} \equiv_{\text{EULER}} 1^k \cdot a^{b \bmod \phi(m)} \\ &\equiv a^{b \bmod \phi(m)} \pmod{m} \end{aligned}$$

Der Spezialfall für m prim des Satzes von EULER ist bekannt als

Satz 7 (Kleiner Satz von FERMAT)

$$p \text{ prim}, a \in \mathbb{Z}, (p,a) = 1: \quad a^{p-1} \equiv 1 \pmod{p}$$

Beweis:

$$\phi(p) = p - 1 \quad \text{nach Satz 5 ii)}$$

Bemerkungen:

1. Die Umkehrung des kleinen FERMAT'schen Satzes gilt nicht, d.h. es gibt zusammengesetzte, ganze $m > 1$ mit $a^m \equiv a \pmod{m}$ für alle ganzen a. Beispiel: $m = 3{\cdot}11{\cdot}17 = 561$. (siehe CARMICHAEL-Zahlen[12]). Der Satz gibt also kein Charakteristikum für Primzahlen her.

2. Besondere Bedeutung hat der Restklassenring modulo p, wobei p eine Primzahl ist. Es gilt für alle $a \in \mathbb{Z}_p$, $a \neq 0$: $\mathrm{ggT}(a,p) = 1$ und folglich: $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$. Hier fällt also die Einheitengruppe mit der multiplikativen Gruppe des Ringes zusammen, d.h. $\mathbb{Z}_p$ ist ein Körper — genauer: $\mathbb{Z}_p$ ist ein endlicher Körper der Charakteristik p.[13]

In $\mathbb{Z}_p$ gelten damit die Rechengesetze für Körper wie aus $\mathbb{Q}$ oder $\mathbb{R}$ bekannt. Insbesondere darf durch jedes Element außer Null dividiert werden[14].

[11] Beweis etwa bei [Bund_88, S. 97]

[12] Genaueres bei [Knut_81, S. 609 (Ex. 9)] sowie bei [Bund_88, S. 100]

[13] Die Charakteristik eines endlichen Körpers K ist definiert als die Ordnung der 1 in der additiven Gruppe $(K,+)$ dieses Körpers. Die Ordnung eines Elementes a einer endlichen Gruppe ist definiert als die kleinste Zahl o, für die gilt:

$$\sum_{j=1}^{o} a = 0$$

[14] Über endliche Körper z.B. bei [Lips_81, S. 178-180] oder [LiNi_86]

Aufgaben:

Wählen Sie für die folgenden Aufgaben betragskleinste Repräsentation, um mit kleinen Zahlen zu rechnen:

4.1 Vereinfachen Sie $8 \cdot x \equiv 56 \pmod{16}$.

4.2 Lösen Sie $8 \cdot x \equiv 57 \pmod{17}$.

4.3 Programmieren Sie in den `Programmrahmen_Inverse` einen Algorithmus, der das multiplikative Inverse a^{-1} von a in $\mathbb{Z}_m$ berechnet.

4.4 Lösen Sie nochmals Aufgabe 2.1. Verwenden Sie jetzt den Satz von Euler.

4.5 Bestimmen Sie $\phi(24)$ und das prime Restsystem $\mathbb{Z}_{24}^*$.

4.6 Bestätigen Sie den kleinen Satz von Fermat am Beispiel $p = 7$, d.h. bilden Sie die Potenzfolgen $2^0, 2^1, 2^2, \ldots, 2^6 \pmod 7$ und $3^0, 3^1, 3^2, \ldots, 3^6 \pmod 7$, überlegen dann, wie die Folge der Potenzen von (-1) aussieht, und leiten daraus die Potenzfolgen von (-2) und (-3) ab.

5 Chinesischer Restsatz

Der Vorteil des Rechnens in homomorphen Bildern (also mit modularer Arithmetik), besteht darin, daß ein Anschwellen der Zwischenergebnisse vermieden und sowohl Rechenzeit als auch reservierter Speicherplatz erheblich reduziert werden können.

5.1 Der Chinesische Restsatz

Bei sehr großen Moduln ist damit aber kein Gewinn erzielt, und hier kommt die Idee zum Tragen, in mehreren homomorphen Bildern zu rechnen. Ein „großes Problem“ wird dadurch in mehrere „kleine Probleme“ zerlegt, deren Lösungen anschließend wieder zu einer Lösung des Ausgangsproblems rekombiniert werden.

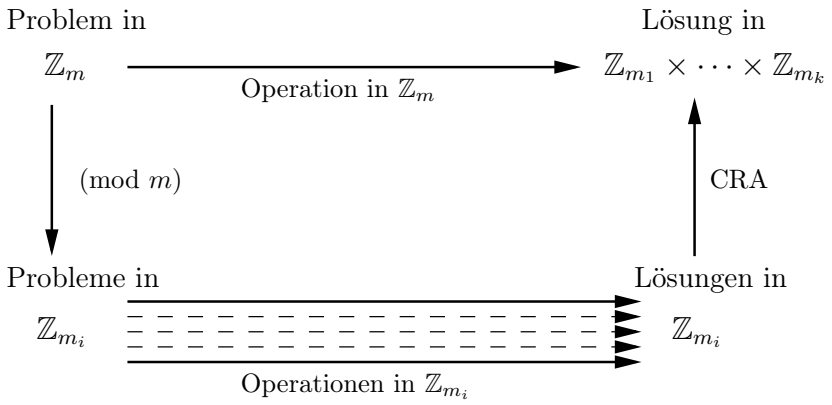

Abbildung 2: Der „Umweg“ über die kleineren Restklassenringe

Diese Rekombination der Teilergebnisse geschieht nach einem Algorithmus, dem Chinesischen Rest Algorithmus **CRA**, so benannt nach dem Chinesischen Restsatz, auf dem er beruht:

Satz 8 (Chinesischer Restsatz, Sun Tsŭ)
Seien $m_1, m_2, \ldots, m_k$ paarweise teilerfremd, $M := m_1 \cdot m_2 \cdot \ldots \cdot m_k$, so gilt:
Das System simultaner Kongruenzen:

$$\begin{array}{lrcl} (1) & x & \equiv & r_1 \pmod{m_1} \\ (2) & x & \equiv & r_2 \pmod{m_2} \\ & & \ldots & \\ (k) & x & \equiv & r_k \pmod{m_k} \end{array}$$

besitzt eine eindeutige Lösung $x \pmod M$, d.h. $x \in \{0, 1, \ldots, M-1\}$ existiert eindeutig.

Damit anschließend die Umkehrung des (mod M)-Homomorphismus gelingt, sind die Teilmoduli $m_1, \ldots, m_k$ also paarweise teilerfremd und so zu wählen, daß ihr Produkt gerade M ergibt.

Wählt man sie überdies auch prim, so

a) ist paarweise Teilerfremdheit der Moduli $m_1, m_2, \ldots, m_k$ garantiert und

b) sind die Restklassenringe $\mathbb{Z}_{m_i}$ Körper, in denen wie gewohnt gerechnet werden kann.

5.2 Chinesischer Restalgorithmus

Nun geben wir zunächst das Verfahren an, nach dem die Lösung eines simultanen Kongruenzensystem, die nach Satz 8 existiert, berechnet wird und zeigen anschließend die Anwendung auf unser Problem:

Algorithmus CRA (Chinesischer Restalgorithmus)

Eingabe:	k simultane Kongruenzen zu teilerfremden Moduln: $x \equiv r_i \pmod{m_i}$, für $i = 1, \ldots, k$ d.h.: $r_1 \in \mathbb{Z}_{m_1}, r_2 \in \mathbb{Z}_{m_2}, \ldots, r_k \in \mathbb{Z}_{m_k}$ und $m_1, m_2, \ldots, m_k$ paarweise relativ prim[15]	
Ausgabe:	$x \pmod M$, wobei $M := m_1 \cdot m_2 \cdot \ldots \cdot m_k$	
Aufwand:	$\mathrm{O}(\mathrm{M}(M) \cdot k)$	
1)	$M := m_1$; $x := r_1 \pmod M$	
2)	für $i := 2$ bis k, berechne	
3)	$h := (r_i - x) \cdot M^{-1} \pmod{m_i}$	{ multiplikatives Inverses mit Euklidschem Algorithmus berechnen }
4)	$x := x + h \cdot M$	{ $M = m_1 \cdot m_2 \cdot \ldots \cdot m_{i-1}$, $x = r_j \pmod{m_j}$ für alle $j = 1, \ldots, i$ }
5)	$M := M \cdot m_i$	
6)	gib zurück ($x \pmod M$)	

Beispiel:

Berechne $x \equiv 3^{13} \pmod{77}$, also gilt $x \in \{0, 1, \ldots, 76\}$
Wähle die Moduln $m_1 = 7$, $m_2 = 11$, denn $m_1 \cdot m_2 = 7 \cdot 11 = 77$
Rechnung in $\mathbb{Z}_7$: $3 \xrightarrow{-S} 9 \equiv_7 2 \xrightarrow{-X} 6 \xrightarrow{-S} 36 \equiv_7 1 \xrightarrow{-S} 1 \xrightarrow{-X} 3$
Rechnung in $\mathbb{Z}_{11}$: $3^{13} \equiv 5 \pmod{11}$ (s. Beispiel aus 2.6)

Aufstellen des Kongruenzensystems:

$$\begin{array}{ll} (1) & x \equiv 3 \pmod{7} \\ (2) & x \equiv 5 \pmod{11} \end{array}$$

Anwendung des **CRA** liefert:

$$\begin{aligned} x &\equiv 3 + 7 \cdot ((5-3) \cdot 7^{-1} \pmod{11}) \\ &\equiv 3 + 7 \cdot (2 \cdot 8 \pmod{11}) \\ &\equiv 3 + 7 \cdot 5 \\ &\equiv 38 \pmod{77} \end{aligned}$$

Die gesuchte Lösung lautet also $x \equiv 38 \pmod{77}$.

Darüber hinaus gibt es auch Probleme, für die kein polynomiales Lösungsverfahren auf „direktem“ Wege bekannt ist, die also den „Umweg“ über die Teilringe erfordern. Ein Beispiel ist die Berechnung der Quadratwurzel in $\mathbb{Z}_m$ (siehe Kap. 6.4).

Praktisch sichere asymmetrische Krypto- und Signatursysteme setzen das schnelle Rechnen mit Zahlen von 200-300 Dezimalstellen voraus. Diese „chinesische Resttechnik“ erlaubt dafür effizientere Algorithmen, weil mit kleineren Zahlen operiert wird.

Aufgaben:

5.1 Bestimmen Sie sämtliche Lösungen x des Kongruenzensystems:

$$\begin{aligned} x &\equiv -2 \pmod{5} \\ x &\equiv 2 \pmod{7} \\ x &\equiv 4 \pmod{9} \end{aligned}$$

5.2 Implementieren Sie den Chinesischen Restalgorithmus in den `Programmrahmen_CRA`.

6 Quadratische Reste und Quadratwurzel in $\mathbb{Z}_m$

Im folgenden Kapitel sei $\mathbb{Z}_m$ stets durch S'_m repräsentiert, wobei $m \in \mathbb{N}$, $m > 2$ sei.

[15] Verallgemeinerung auf nicht teilerfremde Moduln bei [Knut_81, S. 276 (Ex. 3)]

6.1 Quadratische Reste in $\mathbb{Z}_m^*$

Bilden wir in $\mathbb{Z}_m^*$ sämtliche Quadrate, so stellen wir (wie in $\mathbb{Z}$) fest, daß bestimmte Reste vorkommen, andere dagegen nicht. Quadratwurzeln wird man also später sinnvollerweise nur für solche quadratischen Reste definieren.

Definition Quadratische Reste und Nichtreste

$$\begin{aligned} \mathbf{QR}_m &:= \{r^2 (\mathrm{mod}\ m) |\, r \in \mathbb{Z}_m^*\} && \text{Quadratische Reste} \\ \mathbf{QNR}_m &:= \mathbb{Z}_m^* \setminus \mathrm{QR}_m && \text{Quadratische Nichtreste} \end{aligned}$$

Es gilt $|\mathrm{QR}_m| \leq |\mathrm{QNR}_m|$, mit Gleichheit im Falle m prim (siehe Satz 11).

Beispiele:

$$\begin{aligned} &\mathrm{QR}_5 = \{1, 4\},\ \mathrm{QNR}_5 = \{2, 3\} \\ &\mathrm{QR}_6 = \{1\},\ \mathrm{QNR}_6 = \{5\} \\ &\mathrm{QR}_8 = \{1\},\ \mathrm{QNR}_8 = \{3, 5, 7\} \end{aligned}$$

6.2 Legendre- und Jacobi-Symbol

Die Frage, ob ein Element $a \in \mathbb{Z}_p^*$ quadratischer Rest ist, oder nicht, führt zu der 1798 von Legendre gegebenen

Definition Legendre-Symbol
Es seien p prim und $a \in \mathbb{Z}$, dann heißt

$$\left(\frac{a}{p}\right) := \begin{cases} +1 \text{ falls } a \in \mathrm{QR}_p \\ 0 \text{ falls } p|a \\ -1 \text{ falls } a \in \mathrm{QNR}_p \end{cases} \qquad \text{Legendre-Symbol } a \textbf{ nach } p$$

Es wird verallgemeinert auf den Fall m nicht prim durch die

Definition Jacobi-Symbol[16]
Es seien $m = p_1^{r_1} \cdot p_2^{r_2} \cdot \ldots \cdot p_n^{r_n}$, p_i paarweise verschieden und prim, $q \in \mathbb{Z}$ mit $(q, m) = 1$, dann heißt:

$$\left(\frac{q}{m}\right) := \left(\frac{q}{p_1}\right)^{r_1} \cdot \left(\frac{q}{p_2}\right)^{r_2} \cdot \ldots \cdot \left(\frac{q}{p_n}\right)^{r_n} \qquad \text{Jacobi-Symbol } q \textbf{ nach } m$$

[16] Einen effizienten Algorithmus zur Berechung des Jacobi-Symbols findet man bei [Denn_82, S. 106]

Bemerkung:

Man beachte den Unterschied in der Definition:
Ist m zusammengesetzt, also m nicht prim, so bedeutet $\left(\frac{q}{m}\right) = +1$ i.a. **nicht**, daß q quadratischer Rest modulo m ist! Genauer: $q \in \mathrm{QR}_m \Rightarrow \left(\frac{q}{m}\right) = +1$ (nach Definition)

Zur Berechnung des LEGENDRE-Symbols diene folgender

Satz 9 (EULER'sches Kriterium)
Seien p prim und ungerade, $a \in \mathbb{Z}$ und nicht $p|a$, so gilt:

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \pmod{p} \quad [17]$$

Damit haben wir einen „schnellen" Test, ob eine Zahl a quadratischer Rest mod p ist, oder nicht.

Beispiel:

$$\left(\frac{8}{15}\right) = \left(\frac{8}{3}\right) \cdot \left(\frac{8}{5}\right) = 8^{\frac{3-1}{2}} \pmod{3} \cdot 8^{\frac{5-1}{2}} \pmod{5} = (-1) \cdot (-1) = 1, \text{ aber } 8 \notin \mathrm{QR}_{15}$$

6.3 Quadratwurzeln in $\mathbb{Z}_p$

Wenn wir in $\mathbb{Z}_p^*$ wissen, daß etwa q quadratischer Rest ist, so fragen wir nun, welches seine Wurzeln sind. Davon handelt der folgende

Satz 10 (Quadratwurzeln in $\mathbb{Z}_p$)
Es seien p prim, $q \in \mathrm{QR}_p$, und r eine Wurzel von q ($r^2 \equiv q \pmod{p}$).

i) Dann ist auch $-r \pmod{p}$ Wurzel von q und

ii) diese beiden sind die einzigen Wurzeln von q in $\mathbb{Z}_p$.

Sei zusätzlich $p \equiv 3 \pmod{4}$, so gilt

iii) Konstruktion der (Quadrat-)Wurzel: $r \equiv \pm q^{\frac{p+1}{4}} \pmod{p}$ sind die Quadratwurzeln von q in $\mathbb{Z}_p$, d.h. $r^2 \equiv q \pmod{p}$

iv) Genau eine dieser Wurzeln ist selber quadratischer Rest mod p, die andere nicht.

Damit haben wir ein Instrument, Quadratwurzeln in $\mathbb{Z}_p$ zu bestimmen.

Bemerkung:

Allgemein existiert auch ein schneller Algorithmus, um die Wurzeln von Primzahlen $p \equiv 1 \pmod{4}$ zu bestimmen, der uns aber im Zusammenhang mit Konzelationssystemen nicht weiter interessiert.

[17] Beweis etwa bei [Bund_88, S. 131]

6.4 Quadratwurzeln in $\mathbb{Z}_m$

Sind Quadratwurzeln in $\mathbb{Z}_m$ ($m = p_1 \cdot p_2 \cdot \ldots \cdot p_n, p_1, p_2, \ldots, p_n$ prim und paarweise verschieden, man sagt m sei **quadratfrei**, weil jeder Primfaktor nur einmal in m vorkommt) zu bestimmen, so errechnen wir zunächst Wurzeln in Teilringen zu primen Moduln $\mathbb{Z}_{p_1}$, $\mathbb{Z}_{p_2}, \ldots, \mathbb{Z}_{p_n}$, und wenden dann den Chinesischen Restsatz an. Die Gleichung $x^2 \equiv q \pmod{p_i}$ besitzt 2 Lösungen (Satz 10 ii)). Jede der 2^n Kombinationen dieser Lösungen liefert nach Anwendung des Chinesischen Restsatzes eine Lösung der Gleichung $x^2 \equiv q \pmod{m}$. Damit haben wir bereits den ersten Teil von Satz 11 gezeigt:

Satz 11

i) Sei m quadratfrei, d.h. $m = p_1 \cdot p_2 \cdot \ldots \cdot p_n,\ p_1, p_2, \ldots, p_n$ prim und paarweise verschieden, ferner $x \in \mathrm{QR}_m$, so besitzt die Gleichung

$$x^2 \equiv q \pmod{m}$$

exakt 2^n verschiedene Lösungen in $\mathbb{Z}_m$.

ii) Sei allgemein $m = p_1^{r_1} \cdot p_2^{r_2} \cdot \ldots \cdot p_n^{r_n}$, p_i paarweise verschieden und prim, so gilt:

$$|\mathrm{QR}_m| = \frac{|\mathbb{Z}_m^*|}{2^n}$$

oder ausführlich

$$|\mathrm{QR}_m| = \frac{1}{2^n} \cdot (p_1 - 1)p_1^{r_1 - 1} \cdot (p_2 - 1)p_2^{r_2 - 1} \cdot \ldots \cdot (p_n - 1)p_n^{r_n - 1}$$

Je mehr Primfaktoren also ein Modul m besitzt, desto weniger Quadratische Reste existieren, deren jeder dafür umso mehr Quadratwurzeln besitzt.

Wir bezeichnen zwei Wurzeln r_0 und r_1 eines quadratischen Restes $a \in \mathbb{Z}_m^*$ als **wesentlich verschieden** genau dann, wenn $-r_1 \not\equiv r_0 \not\equiv r_1$ gilt (d.h. wenn ihre Beträge bei betragskleinster Repräsentation ungleich sind). Über ihre Anzahl gibt der folgende Satz Auskunft.

Satz 12 (Wesentlich verschiedene Wurzeln)

Sei $m = p_1 \cdot p_2 \cdot \ldots \cdot p_n,\ p_1, p_2, \ldots, p_n$ prim und paarweise verschieden, ferner $a \in \mathrm{QR}_m$, so besitzt a genau 2^{n-1} wesentlich verschiedene Wurzeln. Alle weiteren Wurzeln erhält man durch Umkehrung des Vorzeichens der wesentlich verschiedenen Wurzeln.

Beweis:

Jeder quadratische Rest $a \in \mathrm{QR}_m$ besitzt genau 2^n Quadratwurzeln (Satz 11 i). Mit jeder Wurzel r von a ist auch $-r$ Wurzel von a, denn $(-r)^2 \equiv (-1)^2 \cdot r^2 \equiv r^2 \equiv a \pmod{m}$.

Damit besitzt a genau 2^{n-1} solcher Wurzelpaare. Genau die beiden Wurzeln eines Wurzelpaares sind nicht wesentlich verschieden, während zwei Wurzeln aus verschiedenen Wurzelpaaren nach Definition stets wesentlich verschieden sind. □

Die (bekannten) schnellen Algorithmen zur Bestimmung von Quadratwurzeln setzen die Kenntnis der Primfaktorzerlegung des Moduls voraus. Und in der Tat ist das Quadratwurzelziehen modulo $m = p \cdot q$ äquivalent zum Faktorisieren von m. Es gilt

Satz 13

Sei $m = p \cdot q$, p, q verschiedene Primzahlen $\neq 2$, Länge $|m|$ von m sei k Bit, $s \in \mathrm{QR}_m$, r_0 und r_1 die beiden wesentlich verschiedenen Quadratwurzeln von s modulo m. Dann existiert

1. ein Algorithmus, der bei Eingabe von p, q und s mit polynomialem Aufwand in k beide Wurzeln r_0 und r_1 bestimmt.
2. ein Algorithmus, der bei Eingabe von r_0, r_1 und m mit polynomialem Aufwand in k die Faktorisierung von m, also p und q bestimmt.

Beweis:

Zu 1. siehe Kapitel 6.5 sowie [Kran_86, S. 22] (allgemeines Wurzelziehen in $\mathbb{Z}_p$)

Zu 2. Der Algorithmus basiert auf folgender Überlegung (betragskleinste Repräsentation von $\mathbb{Z}_m$):

Gegeben ist $r_0^2 \equiv r_1^2 \equiv s \pmod{m}$.
Also folgt: $r_0^2 - r_1^2 \equiv (r_0 - r_1) \cdot (r_0 + r_1) \equiv 0 \pmod{m}$.

Man erhält $(r_0 - r_1) \not\equiv 0 \pmod{m}$ sowie $(r_0 + r_1) \not\equiv 0 \pmod{m}$, weil r_0 und r_1 wesentlich verschieden sind. Also sind $\mathrm{ggT}(r_0 - r_1, m)$ und $\mathrm{ggT}(r_0 + r_1, m)$ nichttriviale Teiler von m. Dieses Verfahren ist polynomial in k, da der Euklid'sche Algorithmus zur Bestimmung des $\mathrm{ggT}(r_0 \pm r_1, m)$ polynomial in k ist. □

Erfordert das Faktorisieren bestimmter ganzer Zahlen m exponentiellen Aufwand, so auch das Wurzelziehen modulo dieser Zahlen und umgekehrt. Dies wird für die Signatursysteme (die in einem weiteren Praktikumsversuch behandelt werden) von entscheidender Bedeutung sein.

6.5 Ein Spezialfall

In den folgenden Kapiteln wird folgender Spezialfall immer wieder wichtig sein. Wir betrachten Produkte von genau zwei Primzahlen, die beide kongruent 3 modulo 4 sind:

$$m = p \cdot q \text{ mit } p \equiv q \equiv 3 \pmod{4}$$

Wir wollen nun die beiden wesentlich verschiedenen Wurzeln eines quadratischen Restes $a \in \mathrm{QR}_m$ bestimmen. Dazu werden zuerst die Quadratwurzeln von a in $\mathbb{Z}_p$ und in $\mathbb{Z}_q$ (gemäß Satz 10 iii)) bestimmt. Sie mögen $\pm r_p$ bzw. $\pm r_q$ heißen. Daraus ergeben sich genau 4 Kombinationen, aus denen mittels des Chinesischen Reste Algorithmus die 4 Wurzeln $r_0, r_1, r_2, r_3 \in \mathbb{Z}_m^*$ berechnet werden. Die Kombinationen im Einzelnen:

$$K_0 \begin{cases} r_0 \equiv +r_p \pmod{p} \\ r_0 \equiv +r_q \pmod{q} \end{cases}, \quad K_1 \begin{cases} r_1 \equiv -r_p \pmod{p} \\ r_1 \equiv +r_q \pmod{q} \end{cases},$$

$$K_2 \begin{cases} r_2 \equiv +r_p \pmod{p} \\ r_2 \equiv -r_q \pmod{q} \end{cases}, \quad K_3 \begin{cases} r_3 \equiv -r_p \pmod{p} \\ r_3 \equiv -r_q \pmod{q} \end{cases}.$$

Bereits an dieser Stelle erkennen wir, welche Kombinationen zu den wesentlich verschiedenen Wurzeln führen werden. Es gilt nämlich für je zwei nicht wesentlich verschiedene Wurzeln r_0, r_1 von a:

$$\begin{cases} r \equiv +r_p \pmod p \\ r \equiv +r_q \pmod q \end{cases} \Rightarrow \begin{cases} -r \equiv -r_p \pmod p \\ -r \equiv -r_q \pmod q \end{cases}$$

Die wesentlich verschiedenen Wurzeln ergeben sich in $\mathbb{Z}_m^*$ also in diesem Fall z.B. aus den Kongruenzensystemen K_0 und K_2.

Damit können wir einen Algorithmus angeben, der die beiden wesentlich verschiedenen Wurzeln eines quadratischen Restes errechnet.

Algorithmus SQRTMOD

Eingabe: $p, q \equiv 3 \pmod 4$ und beide prim, $a \in \mathrm{QR}_m$, wobei $m = p \cdot q$ und $p \neq q$

Ausgabe: r_0, r_1 mit $r_0^2, r_1^2 \equiv a \pmod m$ und r_0, r_1 wesentlich verschieden

Aufwand: $\mathrm{O}(\mathrm{M}(p \cdot q))$

1) $r_p := a^{\frac{p+1}{4}} \pmod p$
2) $r_q := a^{\frac{q+1}{4}} \pmod q$
3) $r_0 := \mathbf{CRA}(r_p, r_q, p, q)$
4) $r_1 := \mathbf{CRA}(r_p, -r_q, p, q)$
5) gib zurück(r_0, r_1)

Aufgaben:

6.1 Weisen Sie nach, daß $8 \notin \mathrm{QR}_{15}$ ist (siehe Beispiel)[18].

6.2 Bestimmen Sie QR_{21} und das JACOBI-Symbol $\left(\frac{8}{21}\right)$.

6.3 Programmieren Sie eine Funktion, die testet, ob die Zahl a quadratischer Rest der Restklasse m ist. Dabei kann auf eine Primzahlüberprüfung von m verzichtet werden. Verwenden Sie dazu den `Programmrahmen_Legendre`.

6.4 Berechnen Sie $\sqrt{2} \pmod 7$ und geben Sie sämtliche Wurzeln an.

6.5 Berechnen Sie alle $\sqrt{-5} \pmod{21}$ mit Hilfe des chinesischen Restsatzes.

6.6 Schreiben Sie ein Programm in den `Programmrahmen_SQRTMOD`, das die Quadratwurzeln modulo $m = p \cdot q$ ($p, q \equiv 3 \pmod 4$, beide Primzahlen) berechnet.

7 Primalitätstest

Neben der Zerlegung einer ganzen Zahl $n \in \mathbb{N}$ in ihre Primfaktoren ist es häufig wichtig zu bestimmen, ob eine gegebene Zahl prim ist oder nicht. Uns interessiert diese Frage besonders immer dann, wenn wir geeignete Moduln suchen, in deren Restklassen wir rechnen können.

[18] Zu beachten ist, dass quadratische Reste in $\mathbb{Z}_m^*$ definiert sind, nicht in $\mathbb{Z}_m$ (siehe Abschnitt 6.1).

Die klassische Schulmethode, n auf Primalität zu untersuchen, besteht darin, sämtliche Primzahlen $k \in \{2, 3, 5, 7, \dots\}$ darauf zu testen, ob sie Teiler von n sind (Trial division). Jeder Test besteht im wesentlichen aus einer Division. Dieses Verfahren ist bei kleinen Zahlen n sicherlich das schnellste und effizienteste Verfahren. Bei großen Zahlen n jedoch findet es entweder ganz rasch einen kleinen Primfaktor, oder (besonders bei großen Primzahlen) es liefert sehr lange Zeit gar keine Antwort.

Das ist kein erwünschtes Verhalten, und so wurde 1974 ein erster Primzahltest von R. Solovay und V. Strassen angegeben, der in kurzer Zeit ein Ergebnis liefert. Dieser Vorteil wird allerdings dadurch erkauft, daß das Ergebnis des Algorithmus mit einer kleinen Fehlerwahrscheinlichkeit behaftet ist. (Diese Art probabilistischer Algorithmen wird üblicherweise als Monte-Carlo-Verfahren bezeichnet.) Die Fehlerwahrscheinlichkeit kann für die Praxis so gering gewählt werden, daß sie z.B. geringer ist, als die von Hardwarefehlern. 1976 wurde ein verbesserter probabilistischer Algorithmus von M. O. Rabin, basierend auf einigen Ideen von G. L. Miller, angegeben, den wir im folgenden vorstellen.[19]

7.1 Das Rabin-Miller Kriterium für Primalität

Die Idee für den folgenden Primzahltest geht vom kleinen Satz von Fermat aus. Entsprechend der 1. Bemerkung zu Abschnitt 4.3, Satz 7 liefert dieser Satz eine notwendige (jedoch keine hinreichende) Bedingung für Primalität. Es bezeichne

$$\begin{aligned} \mathrm{F}(n,a) &:\Leftrightarrow a^{n-1} \equiv 1 \pmod{n} && \text{(Fermat-Kriterium)} \\ n \text{ prim } &\Rightarrow \forall a \in \{1,\dots,n-1\} : \mathrm{F}(n,a) \end{aligned}$$

(Es gibt möglicherweise unendlich viele zusammengesetzte Zahlen n, die es ebenfalls erfüllen. Wieviele und welche es gibt, ist ein offenes zahlentheoretisches Problem.) Ziel ist im folgenden, aus $\mathrm{F}(n,a)$ ein stärkeres Kriterium herzuleiten, das nicht nur notwendig, sondern auch hinreichend für Primalität ist. Diese Verfeinerung führt zum Rabin-Miller Kriterium und schließlich zu einem effizienten Primalitätstest.

Sei n eine natürliche Zahl, die in k verschiedene Primfaktoren zerfällt. Dann besitzt jeder quadratische Rest $s \in \mathrm{QR}_n$ genau 2^k Quadratwurzeln (Satz 11). Wann immer man daher mindestens 3 paarweise verschiedene Elemente $r_0, r_1, r_2 \in \{1,\dots,n-1\}$ findet, deren Quadrat $s \equiv r_0^2 \equiv r_1^2 \equiv r_2^2 \pmod{n}$ gleich ist, liegt ein Beweis vor, daß n zusammengesetzt ist. Anderenfalls ist n prim. Es fällt auf, daß es genügt, drei verschiedene Wurzeln eines bestimmten Quadrates zu finden, was einfacher ist, wenn man bereits Wurzeln dieses Quadrates kennt. Zwei Wurzeln von $+1$ sind in jedem Restklassenring $\mathbb{Z}_n$ $+1$ und -1 und diese sind für jedes $n \neq 2$ auch verschieden. Daraus läßt sich ein zu Primalität äquivalentes Kriterium herleiten:

$$\text{sei } n > 2: \quad \mathrm{Q}(n) :\Leftrightarrow \forall r \in \{2,\dots,n-2\} : r^2 \not\equiv +1 \pmod{n} \qquad \text{(Quadrat-Kriterium)}$$

Das Quadrat-Kriterium auszuwerten, erfordert die Überprüfung von maximal $n-3$ Zahlen. Das ist zwar schon effizienter, aber immer noch unpraktikabel. Das Fermat-Kriterium gibt nun einen Hinweis darauf, wie man zielgerichtet nach Kandidaten r sucht, um das Quadrat-Kriterium auszuwerten:

[19] Ausführlicher zu Primzahltests: [Knut_81, Lens_84, Ries_87]

Um $\mathrm{F}(n,a)$ zu testen, muß die Potenz $A_0 := a^{n-1} \pmod n$ berechnet werden. Gilt $A_0 \neq 1 \pmod n$, dann ist n zusammengesetzt, da das FERMAT-Kriterium nicht erfüllt ist. Anderenfalls ist $A_0 = 1$, und n muß weiter untersucht werden. Man zerlege nun $n-1 = u \cdot 2^q$, wobei u ungerade sei. Findet man eine Quadratwurzel

$$A_1 := \sqrt{A_0} \equiv a^{\frac{n-1}{2}} \pmod n \notin \{-1,+1\},$$

so ist n zusammengesetzt (Kandidat $r = A_1$ gefunden!). Anderenfalls muß n weiter untersucht werden. Ist wiederum $A_1 = +1$, so kann auch

$$A_2 := \sqrt{A_1} \equiv a^{\frac{n-1}{4}} \pmod n \notin \{-1,+1\}$$

geprüft werden, usw. War $A_{q-1} = +1$, so kann noch

$$A_q := \sqrt{A_{q-1}} \equiv a^{\frac{n-1}{2^q}} \pmod n \notin \{-1,+1\}$$

untersucht werden. Der gesamte Test hat zwei mögliche Ausgänge.

- Entweder er findet $A_0 \neq +1$ oder $A_1 \neq \pm 1$ oder $A_2 \neq \pm 1 \ldots$ oder $A_q \neq \pm 1$, dann ist n zusammengesetzt (gemäß Quadrat-Kriterium),
- oder er findet keinen solchen Index, dann könnte n prim sein.

Die Iteration wird nach spätestens q Schritten abgebrochen. Um sie effizient berechnen zu können, wird in umgekehrter Reihenfolge vorgegangen: Beginnend mit $A_q := a^u \pmod n$ wird jeweils das Ergebnis der vorigen Iteration quadriert: $A_{j-1} := A_j^2 \pmod n$ für $j = q, \ldots, 1$. Zusammengefaßt ergibt sich das RABIN-MILLER Kriterium[20]:

$$\mathrm{RM}(n,a) := \begin{cases} a^{\frac{n-1}{2^q}} \equiv 1 \pmod n \text{ oder} \\ \exists j = 1,\ldots,q : a^{\frac{n-1}{2^j}} \equiv -1 \pmod n \end{cases}$$

Dieses Kriterium ist notwendig für Primalität, d.h. falls n prim so gilt für alle $a \in \{1,\ldots,n-1\} : \mathrm{RM}(n,a) \Rightarrow \mathrm{F}(n,a)$. Jedoch wird man kaum erwarten, daß RM hinreichend für Primalität ist, da es nur höchstens $\mathrm{ld}(n-1)$ aller Kandidaten r des Quadrat-Kriteriums überhaupt betrachtet. Das überraschende jedoch ist, daß es dennoch „beinahe hinreichend“ ist in dem Sinne, daß $\mathrm{RM}(n,a)$ nur für höchstens $\frac{1}{4}$ aller a erfüllt ist, falls n zusammengesetzt ist [Moni_80]. Diese Fehlerwahrscheinlichkeit kann durch wiederholtes Testen mit gleichverteilt zufälligen Zahlen a beliebig nahe an 0 gebracht werden, so daß man ein zu Primalität praktisch äquivalentes Kriterium erhält. Es bezeichne

$$\mathrm{RM}_t(n) := \bigwedge_{\tau=1}^{t} \mathrm{RM}(n,a_\tau),$$

wobei die a_τ gleichverteilt zufällige Elemente von $\{1,\ldots,n-1\}$ seien. Dann gilt:

[20] Warum kann hier die erste Frage $A_0 = a^{n-1} \equiv 1 \pmod n$ des skizzierten Algorithmus entfallen?

$$P[\mathrm{RM}_t(n) \mid n \text{ prim}] = 1$$
$$P[\mathrm{RM}_t(n) \mid n \text{ zusammengesetzt}] \leq 4^{-t} \quad (1)$$

Man beachte die Asymmetrie: Auch nach sehr vielen Iterationen, die alle bestätigt haben, daß n prim sein kann, gibt es keinen Beweis dafür, daß n prim ist. Wurde n dagegen als zusammengesetzt entdeckt, so ist die zufällig gezogene Zahl a, die zu diesem Ergebnis führte, ein (im mathematischen Sinne strenger) Beweis für Zusammengesetztheit. (Ein Kandidat für das Quadrat-Kriterium wurde gefunden). Zahlen, die vom RABIN-MILLER Kriterium als prim ausgewiesen werden, werden daher manchmal **Pseudo-Primzahlen** genannt.

Algorithmus RMA (Primalitätstest nach RABIN-MILLER)

Eingabe: $n \in \mathbb{N} \setminus \{0,1,2,3\}, I \in \mathbb{N}$
(falls $k = |n| \geq 256$, so wird $I = 1$ empfohlen.)

Ausgabe: { WAHR – falls n prim
FALSCH – falls n zusammengesetzt

Aufwand: $\mathrm{O}(I \cdot \mathrm{M}(n) \cdot \log n)$

1) $u := n - 1;\ q := 0$
2) solange, wie u gerade $\{n - 1 = u \cdot 2^q\}$
3) $\quad u := \frac{u}{2}$
4) $\quad q := q + 1$
5) wiederhole
6) $\quad a :=$ zufällig aus $\{2, \ldots, n-2\}$
7) $\quad I := I - 1; j := 1$
8) $\quad b := a^u \pmod{n}$
9) $\quad$ falls $(b \neq 1)$ dann
10) $\quad\quad$ solange, wie $(b \neq -1)$ und $(j < q)$
11) $\quad\quad\quad b := b^2 \pmod{n}$
12) $\quad\quad\quad j := j + 1$
13) $\quad\quad$ falls $(b \neq -1)$ gib zurück (FALSCH)
14) solange, bis $(|b| \neq 1)$ oder $(I = 0)$
15) gib zurück $(|b| = 1)$

8 BLUM-Zahlen

Zur Vorbereitung der Konzelationssysteme wird hier die Menge der BLUM-Zahlen eingeführt. Ihre Bezeichnung geht auf M. BLUM zurück, der sie 1982 systematisch untersuchte [Blum_82]. Der Sprachgebrauch ist seither nicht ganz einheitlich, daher ist beim Vergleich in der Literatur auf unterschiedliche Definitionen zu achten.

Definition BLUM$_k$

$$\text{BLUM}_k := \{n = p \cdot q \mid p, q \in \mathbb{P},\ p, q \equiv 3 \pmod 4, |p| = |q| = k\}$$

BLUM$_k$ bezeichnet die Menge aller Produkte zweier Primzahlen, die beide k Bit lang sind und bei Division durch 4 den Rest 3 lassen. Alle Zahlen dieser Mengen BLUM$_k$ sind zusammengefaßt in folgender

Definition BLUM

$$\text{BLUM} := \bigcup_{k=1}^{\infty} \text{BLUM}_k$$

Die Bedeutung dieser Zahlen beruht einerseits darauf, daß man in dieser Menge suchen wird, wenn man zusammengesetzte Zahlen benötigt, deren Faktorisierung „schwer“ ist, andererseits auf einigen zahlentheoretischen Eigenschaften, die für die folgenden Anwendungen von Interesse sind. BLUM-Zahlen zeigen in mancher Hinsicht bemerkenswerte Regelmäßigkeiten, die wir nun im wesentlichen in Bezug auf Quadratische Reste und das Wurzelziehen entwickeln.

8.1 Nochmal quadratische Reste

Einen ersten Überblick über den Zusammenhang von QR$_m$, und die Wurzeln eines quadratischen Restes gibt der folgende Satz

Satz 14

Sei $n = p \cdot q$ eine BLUM-Zahl, $p \neq q$, $a \in \text{QR}_n$ und r_i eine der 4 Quadratwurzeln r_0, r_1, r_2, r_3 von a (gemäß Satz 11 i)), so gilt:

$$r_i \in \text{QR}_n \text{ genau dann, wenn } r_i \in \text{QR}_p \text{ und } r_i \in \text{QR}_q.$$

Genau eine der 4 Quadratwurzeln ist selber wieder quadratischer Rest.

Daraus leiten wir nun einen Algorithmus ab, wie die Quadratwurzel mod n zu bestimmen ist, die selbst wieder quadratischer Rest ist.

Algorithmus SQRTMOD*

Eingabe: $p, q \equiv 3 \pmod 4$ und beide prim, $a \in \text{QR}_n$, wobei $n = p \cdot q$

Ausgabe: r mit $r^2 \equiv a \pmod n$ und $r \in \text{QR}_n$

Aufwand: $\text{O}(\text{M}(p \cdot q))$

1) $r_p := a^{\frac{p+1}{4}} \pmod p$
2) $r_q := a^{\frac{q+1}{4}} \pmod q$
3) gib zurück $(\mathbf{CRA}(r_p, r_q, p, q))$

Wir bereiten noch eine überraschende Eigenschaft dieses Algorithmus vor durch den folgenden

Satz 15

Sei $n = p \cdot q$ BLUM-Zahl, so gilt:

i) $\left(\frac{-1}{p}\right) = \left(\frac{-1}{q}\right) = -1$

ii) $\left(\frac{-1}{n}\right) = +1$

iii) $-1 \notin \mathrm{QR}_n$

Beweis:

zu i) $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} = -1$, da $p - 1 \equiv 2 \pmod 4 \Leftrightarrow \frac{p-1}{2} \equiv 1 \pmod 2$

zu ii) $\left(\frac{-1}{n}\right) = \left(\frac{-1}{p}\right) \cdot \left(\frac{-1}{q}\right) = (-1)^2 = +1$ (gemäß i))

zu iii) $-1 \equiv -1 \pmod p \notin \mathrm{QR}_p$ (gemäß i))

also folgt die Behauptung unmittelbar aus Satz 14. □

Nun zurück zu Algorithmus **SQRTMOD***. Es erscheint zunächst nicht sinnvoll, $s \in \mathrm{QNR}_n$ als Eingabe zu wählen, denn in diesem Falle besitzt s keine Quadratwurzel in $\mathbb{Z}_n^*$. Im Falle $-s \in \mathrm{QR}_n$ (und damit $s \notin \mathrm{QR}_n$ nach Satz 15 iii) liefert **SQRTMOD*** aber bei Eingabe von s dasselbe Ergebnis wie bei Eingabe von $-s$, nämlich die Wurzel r von $-s$, die selbst wieder quadratischer Rest ist. (Rechne dies nach!)

8.2 Erzeugung von BLUM-Zahlen

Aufgrund ihrer besonderen Eigenschaften werden insbesondere für kryptographische Verfahren, die u.a. im Praktikumsversuch zu digitalen Signatursystemen behandelt werden, BLUM-Zahlen gebraucht. Zu ihrer Erzeugung benötigen wir je zwei Primzahlen derselben Länge k Bit, die beide kongruent 3 modulo 4 sind. Um gleichverteilt zufällige BLUM-Zahlen zu erhalten, bietet sich daher folgendes Verfahren an:

1. Wähle gleichverteilt zufällig eine Zahl $p \equiv 3 \pmod 4$ der Länge k Bit
2. Prüfe, ob der BLUM-RABIN-Test diese Zahl als prim ausweist und wiederhole anderenfalls Schritt 1.
3. Wiederhole Schritte 1. und 2. solange, bis zwei Primzahlen p und q der verlangten Art gefunden sind. Gib p und q aus.

Wie alle Erzeugungsverfahren, die auf probabilistischen Primzahltests beruhen, liefert auch dieses „nur“ Pseudo-Primzahlen. An dieser Stelle fehlt noch eine Abschätzung des Aufwandes. Sie müßte berücksichtigen, wie „dicht“ Primzahlen der gesuchten Art gestreut sind, wie wahrscheinlich es also ist, bei gleichverteilt zufälliger Auswahl eine solche zu erwischen.

A Beweise einiger Sätze und Lemmata

Dieser Anhang umfaßt alle die Beweise, die oben weder direkt noch durch Verweis gegeben wurden.

Satz 1 (Division mit Rest)
Sind a, m ganze Zahlen, $m > 0$, dann existieren eindeutig Quotient q und Rest r, so daß

$$a = m \cdot q + r \text{ wobei } (0 \leq r < m,\ q, r \in \mathbb{Z})$$

Die Eindeutigkeit basiert auf dem grundlegenden „Prinzip des kleinsten Elements“, d.h. daß in jeder Teilmenge der natürlichen Zahlen eindeutig ein kleinstes Element existiert.[21]

Satz 2 (Eindeutigkeit des größten gemeinsamen Teilers)
Der (positive) größte gemeinsame Teiler zweier Zahlen ist (wenn er überhaupt existiert) eindeutig bestimmt.

Beweis:

Annahme: g, g' seien beide größte gemeinsame Teiler von a und b und $g \neq g'$.

Es folgt sowohl $g|g'$, weil g' größter Teiler, als auch $g'|g$, da g größter Teiler. Also gilt $g' = \pm g$ im Widerspruch zu g, g' positiv und $g \neq g'$. $\square_*$

Satz 3 (Existenz und Konstruktion des ggT)
Seien $a, b \in \mathbb{Z}$ nicht beide gleich Null.

1. $\mathrm{ggT}(a, b)$ existiert.
2. Es existieren gewisse Zahlen $s, t \in \mathbb{Z}$, genannt Cofaktoren von a bzw. b derart, daß $s \cdot a + t \cdot b = \mathrm{ggT}(a, b)$. (Satz von EUKLID)
3. $g = \mathrm{ggT}(a, b)$, s, t werden vom erweiterten EUKLID'schen Algorithmus **EUKLID** berechnet.

Zum Beweis des Satzes wird hier die totale Korrektheit von **EUKLID** gezeigt:

Beweis:

Vorbereitung:

Lemma 3

a) $\mathrm{ggT}(a, 0) = |a|$

b) falls $b \neq 0 : \mathrm{ggT}(a, b) = \mathrm{ggT}(b, a \pmod b)$

Beweis des Lemmata:

a) gilt nach Definition

[21] siehe [Bund_88]

b) Gilt $b \neq 0$, dann ist $a \pmod b = r$ eindeutig definiert zu: $a = b \cdot q + r$ ($r \in \{0, 1, \ldots, b-1\}$). Nun ist jeder gemeinsame Teiler d von a und b gleichzeitig auch Teiler von b und $a - b \cdot q = r$ (da $b \cdot q$ Vielfaches von d). Umgekehrt ist auch jeder gemeinsame Teiler von b und r wieder gemeinsamer Teiler von $b \cdot q + r = a$ und b. Wenn nun a und b dieselben gemeinsamen Teiler wie b und r besitzen, so auch denselben größten gemeinsamen Teiler. □

Die Berechnung des ggT in **EUKLID** geschieht in den Zeilen 2-4). Zeilen 3-4) lassen sich zusammenfassen zu $(a_0, a_1) := (a_1, (a_0 \bmod a_1))$. Daraus folgt mit Lemma 3 die partielle Korrektheit.

Terminierung: Nach jedem Schleifendurchlauf gilt: $0 \leq a_1 \leq a_0$ wegen der Eigenschaft $a_0 \pmod{a_1} < a_1$. Die Folgen der Variablenbelegungen von a_0 und a_1 sind demnach streng monoton fallend und nicht-negativ, woraus die Terminierung des erweiterten EUKLID'schen Algorithmus folgt.

Die Schleifeninvariante sichert noch zu, daß s und t tatsächlich die Cofaktoren von a und b sind, womit der Beweis von Satz 3 erbracht ist. □

Satz 8 (Chinesischer Restsatz)

Seien $m_1, m_2, \ldots, m_k$ paarweise teilerfremd, $M := m_1 \cdot m_2 \cdot \ldots \cdot m_k$, so gilt:
Das System simultaner Kongruenzen

$$\begin{array}{lrcl} (1) & x & \equiv & r_1 \pmod{m_1} \\ (2) & x & \equiv & r_2 \pmod{m_2} \\ & & \ldots & \\ (k) & x & \equiv & r_k \pmod{m_k} \end{array}$$

besitzt eine eindeutige Lösung $x \pmod M$, d.h. $x \in \{0, 1, \ldots, M-1\}$ existiert eindeutig.

Beweis:

Für jedes $i \in \{1, 2, \ldots, k\}$ gilt: $\mathrm{ggT}(m_i, \frac{M}{m_i}) = 1$.
Also existiert die Inverse $y_i = \left(\frac{M}{m_i}\right)^{-1} \pmod{m_i}$.
Für jedes $j \neq i$ gilt aber $\frac{M}{m_i} \cdot y_i \equiv 0 \pmod{m_j}$, da m_j ein Faktor von $\frac{M}{mi}$ ist!
Setze nun $x \equiv \frac{M}{m_1} \cdot r_1 \cdot y_1 + \frac{M}{m_2} \cdot r_2 \cdot y_2 + \ldots + \frac{M}{m_k} \cdot r_k \cdot y_k \pmod M$
Nun löst x jede der Kongruenzen $(1) \ldots (k)$, denn
$x \bmod m_i \equiv \frac{M}{m_i} \cdot y_i \cdot r_i \equiv r_i \pmod{m_i}$ □

Satz 10 (Quadratwurzeln in $\mathbb{Z}_p$)

Es seien p prim, $q \in \mathrm{QR}_p$, und r eine Wurzel von q ($r^2 \equiv q \pmod p$).

i) Dann ist auch $-r \pmod p$ Wurzel von q und

ii) diese beiden sind die einzigen Wurzeln von q in $\mathbb{Z}_p$.

Sei zusätzlich $p \equiv 3 \pmod 4$, so gilt

iii) Konstruktion der (Quadrat-)Wurzel: $r \equiv \pm q^{\frac{p+1}{4}} \pmod p$ sind die Quadratwurzeln von q in $\mathbb{Z}_p$, d.h. $r^2 \equiv q \pmod p$

iv) Genau eine dieser Wurzeln ist selber quadratischer Rest mod p, die andere nicht.

Beweis:

zu i) $(-r)^2 \equiv (-1)^2 \cdot r^2 \equiv r^2 \pmod p$

zu ii) $q \neq 0$, denn $0 \notin \mathrm{QR}_p$. Die Gleichung $x^2 \equiv q \pmod p$ besitzt als Gleichung 2. Grades über einem Körper (hier $\mathbb{Z}_p$) genau 2 Lösungen, da $q \in \mathrm{QR}_p$ vorausgesetzt.

zu iii) $r^2 \equiv_p \left(q^{\frac{p+1}{4}}\right)^2 \equiv_p q^{\frac{p+1}{2}} \equiv_p q^{1+\frac{p-1}{2}} = q \cdot \left(\frac{q}{p}\right) = q$, da $q \in \mathrm{QR}_p$

zu iv) Man findet zunächst $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} \equiv -1 \pmod p$, da $p - 1 \equiv 2 \pmod 4 \Leftrightarrow \frac{p-1}{2} \equiv 1 \pmod 2$. Also ist -1 quadratischer Nichtrest mod p.

Weiter gilt: $\left(\frac{(\pm r)^2}{p}\right) = \left(\frac{r}{p}\right)^2 = (-1)^2 = +1$, also $\left(\frac{r}{p}\right) = \pm 1$(nach Satz 10 i).

Falls nun $\left(\frac{r}{p}\right) = +1$ so folgt $\left(\frac{-r}{p}\right) = -1$, und falls $\left(\frac{r}{p}\right) = -1$ so folgt $\left(\frac{-r}{p}\right) = +1$. □

Satz 11

i) Sei m quadratfrei, d.h. $m = p_1 \cdot p_2 \cdot \ldots \cdot p_n,\ p_1, p_2, \ldots, p_n$ prim und paarweise verschieden, ferner $x \in \mathrm{QR}_m$, so besitzt die Gleichung

$$x^2 \equiv q \pmod m$$

exakt 2^n verschiedene Lösungen in $\mathbb{Z}_m$.

ii) Sei allgemein $m = p_1^{r_1} \cdot p_2^{r_2} \cdot \ldots \cdot p_n^{r_n}$, p_i paarweise verschieden und prim, so gilt:

$$|\mathrm{QR}_m| = \frac{|\mathbb{Z}_m^*|}{2^n}$$

oder ausführlich

$$|\mathrm{QR}_m| = \frac{1}{2^n} \cdot (p_1 - 1)p_1^{r_1 - 1} \cdot (p_2 - 1)p_2^{r_2 - 1} \cdot \ldots \cdot (p_n - 1)p_n^{r_n - 1}$$

Beweis Teil ii):

Aus jedem Rest $\mathbb{Z}_m^*$ geht durch quadrieren genau ein Element $q \in \mathrm{QR}_m$ hervor. Andererseits existieren zu jedem $q \in \mathrm{QR}_m$ genau 2^n Quadratwurzeln $\in \mathbb{Z}_m$ (Teil i). Daraus folgt die Behauptung. □

B Symbolverzeichnis

Symbol	Bedeutung	Einführung
$\mathbb{Z}$	Menge der ganzen Zahlen einschließlich 0	
$\mathbb{N}$	Menge der natürlichen Zahlen einschließlich 0	
$\mathbb{P}$	Menge der Primzahlen	
$\setminus$	ohne (Mengensubtraktion)	
$\lvert M\rvert$	Kardinalität der Menge M	
$\lvert k\rvert$	Länge der positiven, ganzen Zahl k in Bit	3
$\mathrm{M}(n)$	Aufwand zum Multiplizieren zweier Zahlen mit Absolutbetrag kleiner n	3
$\log(n)$	$= \log_2(n)$ — Logarithmusfunktion zur Basis 2	3
$\exp(n)$	Exponentialfunktion (Basis $= e$)	5
$\mid$	teilbar	5
$\equiv$	kongruent	6
$\equiv_m$	m-kongruent oder kongruent modulo m	6
$\not\equiv$	inkongruent	6
S_m	kleinstes nichtnegatives Repräsentantensystem von $\mathbb{Z}_m$	6
S'_m	betragskleinstes Repräsentantensystem von $\mathbb{Z}_m$	6
$\mathbb{Z}_m$	Restklassenring modulo m	8
$\mathrm{ggT}(a,b)$	zweistellige Funktion, die den größten gemeinsamen Teiler von a und b ausgibt	11
(a,b)	$:= \mathrm{ggT}(a,b)$	11
$\mathbb{Z}_m^*$	Einheitengruppe (multiplikative Gruppe) von $\mathbb{Z}_m$	13
$\phi(m)$	Euler'sche ϕ-Funktion	14
QR_m	Menge der quadratischen Reste modulo m	19
QNR_m	Menge der quadratischen Nichtreste	19
$\left(\frac{a}{p}\right), \left(\frac{q}{m}\right)$	Legendre- bzw. Jacobi-Symbol	19

Literatur

[AhHU_74] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974).

[Blum_82] Manuel Blum: Coin flipping by telephone - a protocol for solving impossible problems. In: Proc. IEEE Spring CompCon, San Francisco (1982), 133–137.

[Bund_88] Peter Bundschuh: Einführung in die Zahlentheorie. Hochschultext, Springer, Berlin (1988).

[Denn_82] Dorothy Denning: Cryptography and Data Security. Addison-Wesley Publishing Company, Reading (1982), reprinted with corrections, January 1983.

[GaJo_79] Michael R. Garey, David S. Johnson: Computers and Intractability - A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979).

[Horn_76] Bernhard Hornfeck: Algebra. 3. Auflage, de-Gruyter-Lehrbuch, Walter de Gruyter & Co (1976).

[Hors_85] Patrick Horster: Kryptologie. Nummer 47 in Reihe Informatik, Herausgegeben von K. H. Böhling, U. Kulisch, H. Maurer, Bibliographisches Institut, Mannheim (1985).

[Lens_84] H. W. Lenstra jr.: Primality testing. In: H. W. Lenstra jr., R. Tijdeman (Hg.), Computational Methods in Number Theory, Band 1, Mathematical Centre Tracts 154 CWI Amsterdam (ehemals Mathematisch Centrum), Amsterdam (1984), 55–77.

[Knut_81] Donald Ervin Knuth: Seminumerical Algorithms, Band 2 von *The Art of Computer Programming*. 2. Auflage, Addison Wesley, Reading (1981).

[Kran_86] Evangelos Kranakis: Primality and Cryptography. Wiley-Teubner Series in Computer Science, B. G. Teubner, Stuttgart (1986).

[LiNi_86] Rudolf Lidl, Harald Niederreiter: Introduction to finite fields and their applications. Cambridge University Press, Cambridge (UK) (1986).

[Lips_81] John D. Lipson: Elements of algebra and algebraic computing. Advanced Book Program, Addison-Wesley Publishing Company, Reading, Mass. (1981).

[Moni_80] Louis Monier: Evaluation and Comparison of two efficient probabilistic primality testing algorithms. Theoretical Computer Science, Band 12 (1980), 97–108.

[Paul_78] Wolfgang J. Paul: Komplexitätstheorie, Band 39 von *Teubner Studienbücher Informatik*. B. G. Teubner Verlag, Stuttgart (1978).

[Ries_87] Hans Riesel: Prime Numbers and Computer Methods for Factorization. Progress in Mathematics, Birkhäuser, Boston, Basel, Stuttgart (1987), 2nd revised and corrected printing.

[Silv_87] R. D. Silverman: The multiple polynomial quadratic sieve. Mathematics of Computation, Band 48 (1987), 329–339.