Anwendungen der modularen Arithmetik

Schon die Zahlendarstellung im modernen Rechner stellt eine Anwendung modularer Arithmetik dar. Ein Byte hat z.B. 8 Bit. Wenn man nur mit Bytes rechnet, muss man modulo 256 rechnen. Allgemein: Bei einer sog. *Wortlänge* von k Bits rechnet man generell modulo 2^k. Details hierzu möge man dem Buch von Diekert, Kufleitner und Rosenberger entnehmen.

Stichwort: *Zweierkomplement.*

Weiteres Beispiel: Fehlererkennung bei Artikelnummern (EAN)
(13-stellig mit *13-Regel*)

Und auch die Matrikelnummern der Uni Stuttgart verwenden modulare Arithmetik, um eine Prüfnummer zu ermitteln – übrigens wird hier ein ähnliches Schema verwendet, wie bei Ausweisprüfnummern und bei Bahn-Card-Nummern, die sogenannte 137-Regel:

Eigentlich 6-stellige Nummer sei 471174. Mache daraus 471174p, so dass $1 \cdot 4 + 3 \cdot 7 + 7 \cdot 1 + 1 \cdot 1 + 3 \cdot 7 + 7 \cdot 4 + 1 \cdot p \equiv 0 \bmod 10$

Lösung: $p = 8$

ISBN, IBAN

Recht bekannt ist die Prüfsumme beim ISBN-Code:

Eine korrekte 10-stellige ISBN (International Standard Book Numbering) ist eine Zahl $x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}$, die die folgende Bedingung erfüllt:

$$1 \cdot x_1 + 2 \cdot x_2 + \cdots + 9 \cdot x_9 + 10 \cdot x_{10} \equiv 0 \mod 11$$

Dabei stehen die ersten 9 Ziffern für die sachlich-inhaltliche Einordnung (Land, Verlag, Buchnummer). Die letzte Ziffer ist die Prüfnummer, die aus dem Bereich $0, \ldots, 10$ kommt – da x_{10} manchmal den Wert 10 annimmt, muss für diesen das Zeichen X verwendet werden.

Die IBAN (International Bank Account Number) wird aus der BBAN (BLZ und 10-stellige Kontonummer) so gebildet: Vorangestellt zwei Buchstaben als Ländercode und eine zweistellige Prüfziffer PP, wobei die folgende 24-stellige Zahl kongruent zu 1 modulo 97 wird: BBBBBBBBKKKKKKKKKKLLCCPP.
In Deutschland (Code DE) ist LLCC=1314, allgemein A=10, B=11, etc.

Beispiel für simultane Kongruenz

Jetzt wollen wir das Lemma der letzten Einheit anwenden.
Wenn man eine Zahl sucht, die kongruent zu y modulo m und kongruent zu z modulo n ist, so nehme man

$$\mathrm{zam} + \mathrm{ybn}$$

wobei $am + bn = 1$ gemäß dem Euklidischen Algorithmus.

Sei also $m = 9$ und $n = 11$, dann ist $5m - 4n = 1$.
Die Kongruenzen $x \equiv y \bmod 9$ und $x \equiv z \bmod 11$ werden also immer durch die Zahl $z \cdot 5m - y \cdot 4n$ gelöst.

Wollen wir also ein x mit $x \equiv 3 \bmod 9$ und $x \equiv 6 \bmod 11$, so wählen wir $x = 6 \cdot 45 - 3 \cdot 44 = 270 - 132 = 138$.

Da $mn = 99$ ist, können wir auch $138 - 99 = 39$ nehmen. Prüfen Sie das Resultat!

Der Chinesische Restsatz

Wir formulieren den Chinesischen Restsatz als Struktursatz über Restklassenringe der Form $\mathbb{Z}/n\mathbb{Z}$:

Satz: Für teilerfremde Zahlen m, n ist die Abbildung

$$\begin{aligned} \mathbb{Z}/mn\mathbb{Z} &\to \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \\ x + mn\mathbb{Z} &\mapsto (x + m\mathbb{Z}, x + n\mathbb{Z}) \end{aligned}$$

ein Isomorphismus von Ringen.

Beweis: Die Homomorphie-Eigenschaften bzgl. Addition und Multiplikation sind leicht zu überprüfen.

Die Bijektivität der Abbildung wurde im Lemma bewiesen.

Gleichzeitige Kongruenzen

Die Standardanwendung des Chinesischen Restsatzes:

Gegeben seien paarweise teilerfremde Zahlen $m_1, \ldots, m_n$.
Das Produkt dieser n Zahlen sei m.
Für jedes n-Tupel ganzer Zahlen $x_1, \ldots, x_n$ existiert genau ein $x \in \{0, \ldots, m-1\}$, für das die Kongruenzen

$$x \equiv x_i \mod m_i$$

für alle i erfüllt sind.

Wie findet man die Lösung?

Suche y mit $y \equiv x_n \mod m_n$ und $y \equiv x_{n-1} \mod m_{n-1}$.
Nun die letzten zwei Kongruenzen durch $x \equiv y \mod m_{n-1} m_n$ ersetzen.
Sukzessive so weitermachen. Lösung sei x.

Wie sieht die Lösungsmenge in $\mathbb{Z}$ aus?

Antwort: $x + m\mathbb{Z}$

Verallgemeinerung und Folgerung

Den Chin. Restsatz kann man auf n Komponenten erweitern:

Korollar: Für paarweise teilerfremde Zahlen $m_1, \ldots, m_n$ mit $m = m_1 \cdot \cdots \cdot m_n$ ist die Abbildung

$$\begin{aligned} \mathbb{Z}/m\mathbb{Z} &\to \mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_n\mathbb{Z} \\ x + m\mathbb{Z} &\mapsto (x + m_1\mathbb{Z}, \ldots, x + m_n\mathbb{Z}) \end{aligned}$$

ein Isomorphismus von Ringen.

Als Anwendung zeigen wir, dass es unendlich viele Primzahlen gibt.

Denn angenommen, es gäbe nur endlich viele. Dann könnte man mit dem Chinesischen Restsatz eine Zahl finden, für die $n \equiv p - 1 \bmod p$ für alle Primzahlen p erfüllt ist. Die Zahl n wäre größer als 1, würde aber von keiner Primzahl geteilt. Das widerspricht dem Satz von der Existenz einer eindeutig bestimmten Primfaktorzerlegung.

Der kleine Satz von Fermat

Der kleine Satz von Fermat bildet die Grundlage für viele Primzahltests, und darüberhinaus wird er in Beweisen oft als nützlicher Baustein verwendet:

Satz: Für alle Primzahlen p und alle $a \in \mathbb{Z}$ gilt:

$$a^p \equiv a \mod p$$

Falls a und p teilerfremd sind, gilt sogar:

$$a^{p-1} \equiv 1 \mod p$$

Bevor wir den Beweis geben, prüfen wir die Aussage am Beispiel $a = 6$ und $p = 13$:

$$6^{13-1} = 6^{12} = (6^3)^4 = 216^4 \equiv (221-5)^4 = (13 \cdot 17 - 5)^4 \equiv (-5)^4 \equiv 8^4 = 64^2 = (5 \cdot 13 - 1)^2 \equiv (-1)^2 = 1 \mod 13$$

Beweis des Satzes

Wenn p ein Teiler von a ist, ist die erste Aussage trivial, und die zweite ist für diesen Fall irrelevant, da $ggT(a,p) = p > 1$ gilt.

Von jetzt an können wir also $ggT(a,p) = 1$ voraussetzen.
Dann ist nach Teil c) des Satzes von Folie 20.3 die Abbildung $x \mapsto ax$ zunächst auf $\mathbb{Z}/p\mathbb{Z}$ bijektiv. Aber daraus folgt sofort, dass diese Abbildung auch auf $(\mathbb{Z}/p\mathbb{Z})^*$ bijektiv ist.

Also gilt

$$(p-1)! \equiv \prod_i i \equiv \prod_i ai \equiv a^{p-1} \cdot \prod_i i \equiv a^{p-1}(p-1)!$$

wobei das Produkt jeweils über alle $i \in (\mathbb{Z}/p\mathbb{Z})^*$ geht.
Da $(p-1)!$ invertierbar ist, können wir die Gleichung durch diesen Wert teilen und erhalten $a^{p-1} \equiv 1$, sowie $a^p \equiv a$.

Primzahltest nach Fermat

Die folgende Prozedur nennt man den *Fermat-Test:*

Eingabe sei die natürliche Zahl n.

Frage: Ist n eine Primzahl?

Wähle $a \in \{1, \ldots, n-1\}$ zufällig.

Berechne $x := a^{n-1} \bmod n$

Falls $x \not\equiv 1 \bmod n$:

AUSGABE: n ist keine Primzahl.

Sonst keine Aussage.

Eigenschaften des Fermat-Tests

Wir stellen fest, dass wenn n eine Primzahl ist, der Test keine Aussage macht. Denn dann gilt nach dem kleinen Satz von Fermat $a^{n-1} \equiv 1$ für alle wählbaren Werte von a.

Aber was passiert, wenn n eine *zusammengesetzte Zahl* ist?

Dann müssen wir zwei Fälle unterscheiden:

1) Für das gewählte a gilt *zufällig* auch $a^{n-1} \equiv 1$
2) Für das gewählte a gilt $a^{n-1} \not\equiv 1$

Leider können wir nicht *garantieren,* dass der zweite Fall, in dem der Fermat-Test das korrekte Ergebnis (n ist nicht prim) liefert, immer mit hoher Wahrscheinlichkeit eintritt.

Trotzdem liegt dieser Test allen gängigen Primzahltests zugrunde!

Durchführung des Fermat-Tests

Die Frage ist, wie man den Test *effizient* durchführen kann.

Wenn n eine 1000-stellige Binärzahl ist, müsste man mehr als 2^{1000} Multiplikationen durchführen!

(wenn man es naiv macht...)

Ein weiteres Problem besteht in der Größe der Zwischenergebnisse.

Allerdings ist die Lösung bzgl. der Zwischenergebnisse offensichtlich:

Da uns das Endergebnis ja nur *modulo* n interessiert, können wir von Anfang an *jedes* Zwischenergebnis modulo n reduzieren. Damit sind die Zwischenergebnisse immer kleiner als n.

Denn: $(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$, und analog für Multiplikation.

Bitte überprüfen Sie das!

Schnelle Exponentiation

Zur Berechnung von a^{n-1} im Fermat-Test verwenden wir das folgende Programm zur Berechnung von a^b (bei Input a, b):

```
e := 1;
while b > 0 do
  if b ungerade then
      e := e · a mod n;
  a := a^2 mod n;
  b := ⌊b/2⌋;
return e
```

Als Schleifeninvariante notieren wir, dass der Wert von $e \cdot a^b$ zu Beginn und Ende jedes Durchlaufs gleich ist. (Natürlich alles immer modulo n !!!)

Nachrechnen!

Wenn wir die Inputwerte als a_0 und b_0 bezeichnen, hat die Schleifeninvariante anfangs den Wert $1 \cdot a_0^{b_0}$. Da es eine Invariante ist und am Ende $b = 0$ gilt, erhalten wir für den Ausgabewert e die Gleichung $e = e \cdot a^0 = 1 \cdot a_0^{b_0}$.

Also: Ausgabe korrekt, Laufzeit logarithmisch, Zwischenwerte klein...
Damit erfüllt der Algorithmus alle Anforderungen.