Mathematik I

Vorlesung 5 - Zahlentheorie “light”

Prof. Dr. Sandra Eisenreich

06. November 2023

Hochschule Landshut

5.1 Teilbarkeit und Euklidischer Algorithmus

Teilbarkeit und Euklidischer Algorithmus

Definition

Seien $a, b \in \mathbb{Z},\ b \neq 0$, dann sagen wir, dass b **die Zahl** a **teilt** (man schreibt $b|a$), wenn es eine ganze Zahl q mit $a = q \cdot b$ gibt.

Satz (Teilbarkeitsregeln)

a) $c|b$ *und* $b|a \Rightarrow c|a$.

b) $b_1|a_1$ *und* $b_2|a_2 \Rightarrow b_1 b_2|a_1 a_2$

c) $b|a_1$ *und* $b|a_2 \Rightarrow b|s \cdot a_1 + t \cdot a_2$ *für beliebiges* $s, t \in \mathbb{Z}$

d) $b|a$ *und* $a|b \Rightarrow a = b$ *oder* $a = -b$

Satz (Division mit Rest)

Für $a, b \in \mathbb{Z}$, $b \neq 0$, gibt es genau eine Darstellung

$$a = b \cdot q + r$$

mit $q, r \in \mathbb{Z}$ und $0 \leq r < |b|$. Dann nennt man:

- *a Dividend*
- *b Divisor*
- *q Quotient*
- *r Rest*

und schreibt: $q = \frac{a}{b}$ und $r = a \bmod b$. Man spricht "a modulo b".

Hinweis: $r = a \bmod b$ bedeutet, dass $\overline{r}$ die Äquivalenzklasse von a bezüglich der Relation R_b ist (also beim Teilen durch b), siehe erstes Kapitel.

"Modulo-Rechnen" = den (positiven) Rest beim Teilen durch den Divisor angeben, z.B. Uhrzeiten angeben ist Rechnen modulo 12; Alle Uhrzeiten mit gleichem Rest mod 12 sind auf der Uhr identisch: -11h, 1h, 13h, 25h, 49h, wird alles als 1 angezeigt!

Beispiele: → Mitschrift

Modulo-Rechnen

Rechenregel

Um für $a, b \in \mathbb{Z}$ zu berechnen, was a mod b ist, gehe wie folgt vor:

- Ist a positiv, so schaue, wie oft der Divisor b maximal in a passt, und der Rest $r \in \{0, \ldots, a-1\}$ ist dann $r = a$ mod b.
- Ist a negativ, so können wir erst mal alle negativen Vielfachen von b wegrechnen und kommen so auf eine negative Zahl zwischen $(-b+1)$ und 0. Dann ist a mod b gegeben durch $r = b + a$.

größter gemeinsamer Teiler

Definition (ggT)

Seien $a, b \in \mathbb{Z}$ und $d \in \mathbb{Z}$ mit $d|a$ und $d|b$. Dann ist d ein **gemeinsamer Teiler von** a **und** b. Der **größte gemeinsame Teiler** wird mit $ggT(a, b)$ (bzw. $gcd(a, b)$) bezeichnet.

Beispiel: $a = 12$ $b = 18$ Dann ist $d = 3$ ein gemeinsamer Teiler, außerdem: $ggT(12, 18) = 6$

Frage: Wie berechnet man $ggT(a, b) =$?

Satz

Seien $a, b \in \mathbb{Z}$, $b \neq 0$, $r = a \bmod b$ *Dann gilt:*

$$ggT(a, b) = \begin{cases} |b| \text{ falls } r = 0 \\ ggT(b, r) \end{cases}$$

Beweis.

1. Fall ($r = 0$): $a = q \cdot b$ für ein $q \in \mathbb{Z}$ Dann ist $|b|$ ein Teiler von b und a. $|b|$ ist auch der größte gemeinsame Teiler, da jede Zahl $> |b|$ die Zahl b nicht teilt.
2. Fall ($r \neq 0$): $a = q \cdot b + r$
   Sei $d := ggT(a, b)$. Wegen $r = a - q \cdot b$ ist d auch ein Teiler von r. Somit ist d ein gemeinsame Teiler von b und r. Wir müssen noch zeigen, dass der ggT von b und r nicht Größer als d ist. Sei $d' = ggT(b, r)$, dann muss auch d' die Zahl a teilen, und somit gilt auch $d' \leq ggT(a, b) = d$. Zusammen ergibt sich also $d = d'$. ■

Beispiel: → Mitschrift.

Wir erhalten nun eine Methode (= **Euklidischer Algorithmus**) zur Berechnung von $ggT(a, b)$:

Beispiel: → Mitschrift.

Satz (Erweiterter Euklidischer Algorithmus)

Seien $a, b \in \mathbb{Z}$, $a, b \neq 0$ *und* $d = ggT(a, b)$ *Dann gibt es Zahlen* $s, t \in \mathbb{Z}$ *mit*

$$s \cdot a + t \cdot b = ggT(a, b) = d$$

Beispiel: → Mitschrift.

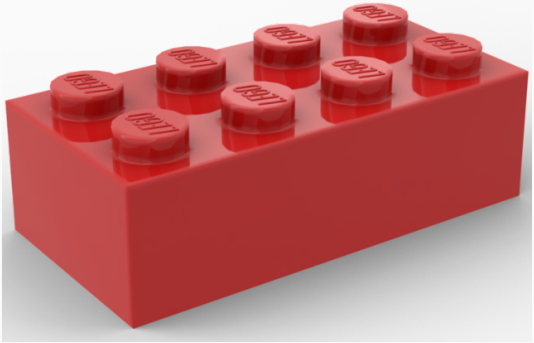
Rechenregel

Trick: Schreibe alle vorkommenden Reste r_i im Euklidischer Algorithmus in der Forum $s_i \cdot a + t_i \cdot b$ mit $s_i, t_i \in \mathbb{Z}$

5.2 Primzahlen und Restklassen

Motivation - Primzahlen

Man braucht **Primzahlen** bei Verschlüsselungsverfahren. Was Primzahlen so toll macht, ist dass sie quasi die "kleinsten Legobausteine" der ganzen Zahlen sind: Man kann schließlich jede Zahl eindeutig (bis auf Reihenfolge) schreiben als Produkt von Primzahlen (z.B. $18 = 2 \cdot 3 \cdot 3$, oder $630 = 2 \cdot 3 \cdot 3 \cdot 5 \cdot 7$), Primzahlen selbst kann man aber nicht mehr unterteilen (5 ist kein Produkt von zwei anderen natürlichen Zahlen).



Quelle: Wikimedia Commons

Primzahlen

Definition

Eine natürliche Zahl p heißt Primzahl, wenn $p > 1$ und keinen Teiler außer 1 und p hat.

Satz (Euklid)

Es gibt unendlich viele Primzahlen.

Beweis. → Mitschrift.

Man kann sogar zeigen, dass es relativ viele Primzahlen gibt!

In Zahlen: Anzahl Primzahlen ≤ 100 : 25
Anzahl Primzahlen $\leq 1.000.000$: ≥ 78.000

Allgemein: Anzahl der Primzahlen $\leq n \approx \frac{n}{\ln n}$

Satz (Primfaktorzerlegung)

Jede natürliche Zahl n besitzt eine (bis auf Reihenfolge) eindeutige Primfaktorzerlegung, d.h. es existieren Primzahlen (nicht notwendigerweise verschieden) $p_1, \ldots, p_k$ *mit*

$$n = p_1 \cdot \ldots \cdot p_k$$

und für jede weitere Darstellung

$$n = p'_1 \cdot \ldots \cdot p'_k$$

gilt: $k = k'$ *und* $(p_1, \ldots, p_k)$ *geht aus* $(p'_1, \ldots, p'_k)$ *durch Umsortierung hervor.*

Beispiel: → Mitschrift.

Folgerung

Alle Teiler einer Zahl erhält man durch die möglichen Produkte der Primfaktoren.

Beispiel: → Mitschrift.

Folgerung

Ist p prim und $p|a \cdot b$, dann gilt $p|a$ oder $p|b$.

Beispiel: → Mitschrift.

Für Nichtprimzahlen ist das falsch, z.B.: $6|16 \cdot 45$, aber $6 \nmid 45$ und $6 \nmid 16$.

Motivation - Restklassen

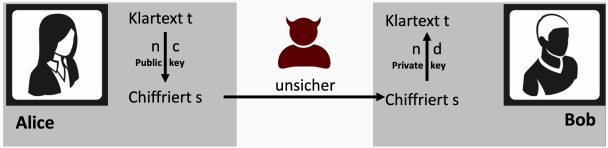
Wir haben bereits gelernt, dass “Rest beim Teilen durch eine Zahl n” (bekommen wir durch Modulo-Rechnen) eine Äquivalenzrelation ist, das heißt sie unterteilt die Menge der ganzen Zahlen in disjunkte Mengen; in jeder Menge liegen nur Zahlen mit demselben Rest beim Teilen durch n. (also zum Beispiel alle Zahlen mit Rest 0 oder 1, oder 2 bis 4 beim Teilen durch 5).

Warum interessiert uns das? - Zum Beispiel, wenn man einen Algorithmus hat, der nach je 5 Schritten immer wieder dasselbe machen muss, also Schritt 0, 1,2,3,4 und dann wieder zurück zu Null. Das erreicht man, wenn man mit einer Schleife durch die natürlichen Zahlen iteriert, aber immer modulo 5 rechnet! Allerdings gibt es sehr viele andere Anwendungsbeispiele:

Anwendungen in der Informatik

- **Hashing**: Hashing ist eine effiziente Methode, wie man große Menge von Datensätzen so abgespeichern kann, dass man schnell auf sie zugreifen kann über IDs. Hierzu braucht man Restklassen, also modulo-Rechnen. (wird in der Programmieren-Vorlesung über Java implementiert)

- in **Verschlüsselungsverfahren**, also im Bereich der **Kryptographie**: Das **RSA-Verfahren** beispielsweise funktioniert so: Alice will ein Nachricht verschlüsselt an Bob übermitteln, so dass Bob den Text zwar mit Hilfe von geteiltem Wissen entschlüsseln kann, aber sonst niemand. Sowohl Ver- als auch Entschlüsseln benutzen das Rechnen mit Restklassen modulo N, wobei N das Produkt zweier **Primzahlen** ist!



Mehr zu beiden Verfahren später.

Restklassen

Erinnerung: (7 kann durch eine beliebige Zahl $\in \mathbb{N}$ ersetzt werden)

$R_7 := \{(m, n) \in \mathbb{Z} \times \mathbb{Z} : \ m \mod 7 = n \mod 7\}$

"mR_7n genau dann, wenn m und n den gleichen Rest bei Teilung durch 7 lassen"

Wir haben bereits gesehen:

- R_7 ist eine Äquivalenzrelation
- Es gibt genau 7 verschiedene Äquivalenzklassen
  $\overline{0} = \{0, 7, -7, 14, \ldots\}$
  $\overline{1} = \{1, 8, -6, \ldots\}$
  $\vdots$
  $\overline{6} = \{6, -1, 13, \ldots\}$

Definition

Seien $a, b \in \mathbb{Z}$, $n \in \mathbb{N}$. Dann heißen a und b **kongruent modulo** n, falls a und b den gleichen Rest beim Teilen durch n, haben, also $a \bmod n = b \bmod n$ ($\Leftrightarrow$ n teilt $a - b$). In Zeichen $a \equiv b \bmod n$.

Hinweis: $a \equiv b \bmod n$ ist also einfach eine andere Schreibweise für aR_nb.

Beispiel: → Mitschrift.

Wie wir bereits gesehen haben, ist "≡" eine Äquivalenzrelation auf $\mathbb{Z}$, und die dazu gehörigen Äquivalenzklassen sind

- $\overline{0} = \{0, n, -n, 2n, \ldots\}$
- $\overline{1} = \{1, n+1, \ldots\}, \ldots,$
- $\overline{n-1} = \{n-1, -1, 2n-1, \ldots\}.$

Definition

Diese Äquivalenzklassen heißen auch **Restklassen**. Die natürlichen **Repräsentanten** der Restklassen sind die Zahlen $0, 1, \ldots n-1$. Die Zahl n heißt **Modul** der Restklassen.

Rechnen mit Restklassen

Im Folgenden sei $n \in \mathbb{N}$ fix.

Definition

Die Menge der Restklassen $\{\overline{0}, \overline{1}, \ldots, \overline{n-1}\}$ modulo n wird mit $\mathbb{Z}/n\mathbb{Z}$ (manchmal auch: $\mathbb{Z}_n$) bezeichnet. (Man spricht: "$\mathbb{Z}$ modulo $n\mathbb{Z}$"). Für $a, b \in \mathbb{Z}/n\mathbb{Z}$ sei

$$\overline{a} \oplus \overline{b} := \overline{(a+b) \bmod n}$$
$$\overline{a} \odot \overline{b} := \overline{a \cdot b \bmod n}$$

Beispiel: → Mitschrift.

Man kann die Ergebnisse von $\oplus$ und $\odot$ auch einfach in einer Verknüpfungstabelle darstellen.

Beispiel: → Mitschrift.

Satz

Es gilt:
$$\begin{aligned} \overline{a+b} &= \overline{a} \oplus \overline{b} \\ \overline{a \cdot b} &= \overline{a} \odot \overline{b} \end{aligned}$$

Das bedeutet: Beim Rechnen mit Resten spielt es keine Rolle ob man zuerst modulo n rechnet und dann die Operationen $\oplus$ und $\odot$ verwendet oder ob man $+$ und $\cdot$ zuerst verwendet und dann das Ergebnis modulo n berechnet.

Beispiele

- Sei $n = 7$ and $a = 11,\ b = 9$:

$$\begin{aligned} \overline{11} \oplus \overline{9} &= \overline{4} \oplus \overline{2} = \overline{6}, \\ \overline{11 + 9} &= \overline{20} = \overline{6} \\ \overline{11} \odot \overline{9} &= \overline{4} \odot \overline{2} = \overline{1} \\ \overline{11 \cdot 9} &= \overline{99} = \overline{1} \end{aligned}$$

- $n = 7$

$$\overline{4 \cdot 5 + 3 \cdot 2 + 5 \cdot 5} = \overline{51} = \overline{2} \text{ und}$$
$$\overline{4} \odot \overline{5} \oplus \overline{3} \odot \overline{2} \oplus \overline{5} \odot \overline{5} = \overline{6} \oplus \overline{6} \oplus \overline{4} = \overline{5} \oplus \overline{4} = \overline{2}$$

Anwendung: Hashing

Problem: große Menge von Datensätzen sollen so gespeichert werden, dass mit Hilfe eines eindeutigen Schlüssels ein schneller Zugriff auf jeden Datensatz gewährleistet ist.

Beispiel:

Eine große Firma hat 5000 Mitarbeitern, jeder Mitarbeiter besitzt eine 10-stellige Personalnummer.

1. Möglichkeit

1	532 . . . 1978	Reiner Zufall
2	⋮	
3	⋮	
4	⋮	
⋮		
5000	124 . . . 1977	Donald Duck

Alle Mitarbeiter werden in einer Liste aufgenommen (so wie sie angestellt werden).

→ minimaler Speicherbedarf, aber Suche dauert lange.

2. Möglichkeit Betrachte Liste mit 10^{10} Einträgen und Personalnummer=Listenplatz

1	
⋮	
124...	Donald Duck
⋮	
532...	Rainer Zufall
⋮	
10^{10}	

→ blitzschneller Zugriff, aber immenser Speicherplatzbedarf.

3. Möglichkeit: Hashing

Bilden den langen Schlüssel mit einer Hashfunktion auf eine kurze Zeilennummer ab. An dieser Stelle wird der Mitarbeiter eingetragen.

Schlüsselraum: $K = \{0, 1, \ldots, 10^{10} - 1\}$

Zeilennummern: $H = \{0, 1, \ldots, 9999\}$

Beispiel: Hashfunktion, welche k auf die letzten 4 Ziffern abbildet:

$$\begin{aligned} h : K &\to H, \\ k &\mapsto k \bmod 10000 \end{aligned}$$

Z.B. 123456**7893** $\mapsto$ 7893, 432571**5134** $\mapsto$ 5134

Vorteil: Hashfunktion ist leicht zu berechnen. Geringer Speicherbedarf.

Problem: Es kann passieren, dass zwei verschiedene Schlüssel auf den selben Wert abgebildet werden (d.h. h ist nicht injektiv). Wir sprechen in diesem Fall (d.h. h nicht injektiv) von einer **Kollision**. (Achtung: in der Realität macht das die Laufzeit kaputt! Eine Kollision ist also unbedingt auszuschließen!)

Hashing-Strategien

1. Strategie Verwende eine Hashfunktion, die möglichst wenige Kollisionen erzeugt.

z.B. ist $h: \quad K \to H$
$k \mapsto k \bmod 10000$

eine schlechte Wahl (letzten vier Ziffern könnten das Geburtsjahr sein → viele Kollisionen)

Besser sind Hashfunktionen $h: K \to H, k \mapsto k \bmod p$

wobei p eine hinreichend große Primzahl ist. (→ durch die Wahl einer Primzahl p werden oft gängige Strukturen in k zerstört.)

2. Strategie: Kollisionsauflösung

Trotz Wahl einer Primzahl kann es sein, dass die Hashfunktion h nicht injektiv ist. Hier ist ein zusätzlicher Schritt notwendig.

Wenn beim Eintragen der Index $h(k)$ bereits vergeben ist, dann bilde eine **Sondierungsfolge** $S_i(k),\ i = 1, \ldots, p-1$. Diese durchläuft mit $S_1(k), S_2(k), \ldots$ alle Indizes der Tabelle bis ein freier Platz gefunden ist. ($S_i(k)$ muss **alle Plätze** durchlaufen)

Sondierungsfolgen

Einfachste Möglichkeit: Lineare Sondierung

$$S_i(k) = (h(k) + i) \bmod p$$

d.h. man springt ausgehend von $h(k)$ immer einen Platz weiter, bis ein freier gefunden worden ist. Erreicht man das Ende der Tabelle, so beginnt man von oben.

Beispiel: → Mitschrift.

Lineare Sondierung neigt zur Cluster-Bildung in der Tabelle (d.h. Datensätze gruppieren sich zusammen). → Sondierung dauert lange.

Besser: Quadratische Sondierung

Sondierungsfolge:

$$\left.\begin{array}{rcl} S_{2i-1}(k) & = & h(k) + i^2 \mod p, \\ S_{2i}(k) & = & h(k) - i^2 \mod p \end{array}\right\} \text{ für } i = 1, \ldots, \frac{p-1}{2}$$

$$\begin{array}{rcl} \text{d.h. } (S_i) & = & h(k)+1 \mod p,\ k(t)-1 \mod p, \\ & = & h(t)+4 \mod p,\ h(t)-4 \mod p, \\ & = & h(t)+9 \mod p,\ h(t)-9 \mod p, \end{array}$$

Beispiel: → Mitschrift.

Bei der linearen Sondierung durchläuft die Sondierungsfolge trivialerweise alle Tabellenplätze. Ist dies auch bei der quadratischen Sondierung der Fall, d.h. durchläuft die Folge $1, -1, 4, -4, 9, -9, \ldots \mod p$ alle Werte von 0 bis $p-1$?

Dies ist nicht immer der Fall:

für $p = 13,\ 17,\ 29$ ist dies nicht der Fall (ausprobieren!)

für $p = 11,\ 19,\ 31$ jedoch schon.

Allgemein gilt:

Satz

Ist p eine Primzahl mit $p = 3$ mod 4 dann gilt:

$$\left\{ \pm i^2 \mod p,\ i = 1, \ldots, \frac{p-1}{2} \right\} = \{1, \ldots, p-1\}.$$

Deshalb: Verwende beim Hashing mit quadratischer Sondierung Primzahlen p mit $p = 3 \mod 4$

Rückwärtssuche: Betrachte Hashfunktion + Sondierungsfolge, bis der Datensatz gefunden ist.