

# 5.

# Java: Pakete

---

Buch Mark Weiss „Data Structures & Problem Solving Using Java“ siehe:

- 126-131 (Pakete)
- 635-639 (Stack als Liste)
- 122-126 („BigRational class“)

# Lernziele Kapitel 5 Pakete in Java

- Die Rolle von Java-Standardpaketen verstehen
- Eigene Pakete aus einer Menge zusammengehöriger Klassen erstellen können
- Zugriffskontrolle und Schutzattribute „private“, „public“ verstehen

## Thema / Inhalt

**Pakete** stellen in Java eine nützliche Strukturierungsmöglichkeit dar, um eine Menge inhaltlich zusammengehöriger Klassen zu einem Modul zusammenzufassen. Durch die Nutzung (das „Importieren“) solcher Module, die von Experten erstellt wurden und deren Implementierung man nicht zu kennen braucht, kann quasi die Funktionalität der Sprache erweitert werden: Nur funktional relevante Teile werden von den Paketentwicklern für die Nutzer zugreifbar („öffentlich“) gemacht, der Rest bleibt verborgen („privat“).

Wir erläutern dies an zwei Beispielen: Zum einen schnüren wir ein Paket, welches einfache Funktionalität rund um eine **verkettete Liste** anbietet, und nutzen dies dann in einer Anwenderrolle, um damit einen potentiell unbegrenzten Stack zu realisieren. Zum andern definieren wir ein Paket, das einem Anwender das **Rechnen mit Brüchen** ermöglicht. Im Unterschied zu rationalen Zahlen hat man dann, wenn Zähler und Nenner eines Bruches explizit als ganze Zahlen gespeichert werden, nicht mit Rundungsproblemen zu kämpfen. Das Bruch-Paket bietet nicht nur die Addition und Multiplikation von Brüchen an, sondern beispielsweise auch

# Thema / Inhalt (2)

Möglichkeiten zum Kürzen und zum Vergleich von Brüchen. Dazu müssen wir bei der Definition des Pakets die in der Schule gelernte Vorgehensweise beim Bruchrechnen implementieren – insbesondere den Hauptnenner, den grössten gemeinsamen Teiler (**ggT**) sowie das kleinste gemeinsame Vielfache (kgV) bilden.

In unserem optionalen Teil zu Kontext und Geschichte der Konzepte gehen wir in diesem Kapitel genauer auf den **euklidischen Algorithmus** ein, mit dem sich der ggT bestimmen lässt, ohne die aufwändigere Primfaktorzerlegung bemühen zu müssen. Der euklidische Algorithmus ist einer der ältesten nicht-trivialen Algorithmen, er hat eine faszinierende Geschichte und gleichzeitig auch interessante mathematische Eigenschaften.

Historisch wurde der euklidische Algorithmus nicht algebraisch formuliert, sondern in der Tradition der Pythagoreer geometrisch verstanden – die beiden Parameter  $a$  und  $b$  stellen dabei zwei Streckenlängen dar, am einfachsten stellt man sich diese als die Länge und Breite eines Rechtecks vor. Man suchte nach dem gemeinsamen Mass der beiden Strecken  $a$  und  $b$  – eine Länge, die in beiden Streckenlängen ganzzahlig enthalten ist. Der Algorithmus zur Bestimmung dieses gemeinsamen Masses, das dann beide Strecken ohne Rest teilt, beruht auf dem Prinzip der „**Wechselwegnahme**“ (Antipharesis) und lautet im Kern bei Euklid so:

„Nimm immer die kleinere Zahl von der grösseren weg, bis ein Rest kommt, welcher die nächstvorgehende Zahl genau misst. Dieser Rest ist das grösste gemeinschaftliche Mass der beiden gegebenen Zahlen.“

Im „schlimmsten Fall“ ist das Ergebnis 1, dann sind die beiden Zahlen teilerfremd. Dem Verfahren liegt wieder eine **Schleifeninvariante** zugrunde: Durch die Subtraktion  $a = a - b$  bzw.  $b = b - a$  ändert sich die Menge der gemeinsamen Teiler von  $a$  und  $b$ , also auch der ggT, nicht.

# Thema / Inhalt (3)

Man kommt also auf eine immer einfachere Aufgabe zur Bestimmung des ggT, bis man bei der Subtraktion 0 erhalten würde, man also zwei gleiche Zahlen hat, von denen der ggT zu bestimmen wäre – von zwei gleichen Zahlen  $x$  ist der  $\text{ggT}(x,x)$  jedoch  $x$  – man ist also fertig und hat aufgrund der Invarianten damit, d.h. mit  $x$ , den ggT der beiden ursprünglichen Parameter bestimmt.

Das **kleinste gemeinsame Vielfache (kgV)** steht in einer einfachen Beziehung zum ggT: Man nehme das Produkt von  $a$  und  $b$  und dividiere es durch  $\text{ggT}(a,b)$  – dann erhält man das kgV von  $a$  und  $b$ . Wir lernen aber auch eine leichte Erweiterung des euklidischen Algorithmus kennen, mit der sich ggT und kgV zweier Zahlen simultan berechnen lassen.

Die fortgesetzte Subtraktion ist langwierig, wenn die eine Ausgangszahl  $a$  viel grösser als die andere Zahl  $b$  ist – dann subtrahiert man recht oft, bis es zu einem Wechsel kommt. Das kann man einfach abkürzen, indem man die eine Zahl durch die andere ganzzahlig dividiert. Dabei sind wir gar nicht am Ganzzahlquotienten interessiert, sondern am Rest, der bleibt, wenn man ganzzahlig dividiert (also subtrahiert hat, sooft es eben ging) – dieser Rest wird typischerweise mit „ $\text{mod}(a,b)$ “ bezeichnet. Damit ergibt sich die modernere und kürzere Form des euklidischen Algorithmus: Es wird fortwährend  **$(a, b) = (b, \text{mod}(a,b))$**  berechnet, bis  $b$  Null wird; dann ist  $a$  der gesuchte ggT. Wenn die Ausgangszahlen zwei benachbarte **Fibonacci-Zahlen** sind, dauert es besonders lang, weil sich dann als Rest stets die nächstkleinere Fibonacci-Zahl ergibt – dennoch bleibt die Schrittzahl logarithmisch zur Eingabe: Gabriel Lamé zeigte schon 1844, dass nie mehr Schritte als das Fünffache der Stellenzahl der kleineren der beiden Ausgangszahlen benötigt wird – ein Resultat, das über 100 Jahre vor der Etablierung **Komplexitätstheorie** erzielt wurde, welche die systematische Beschäftigung der Mathematik und Informatik mit dem Aufwand von Algorithmen begründete.

# Pakete in Java

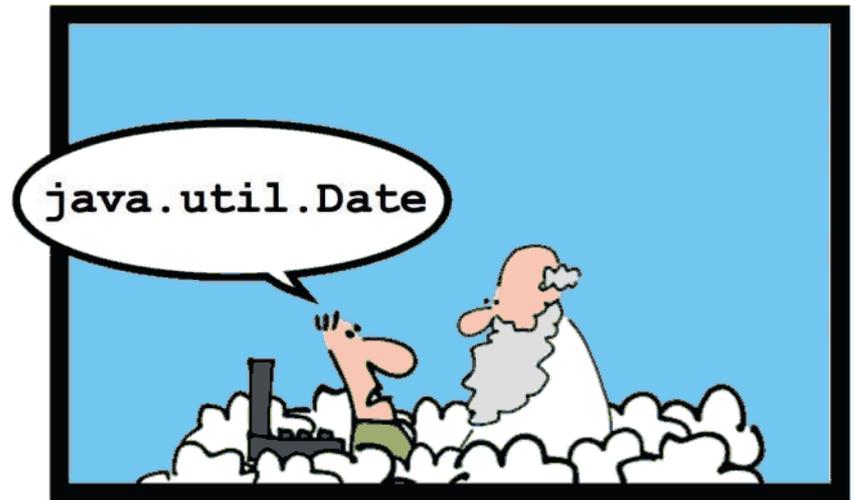
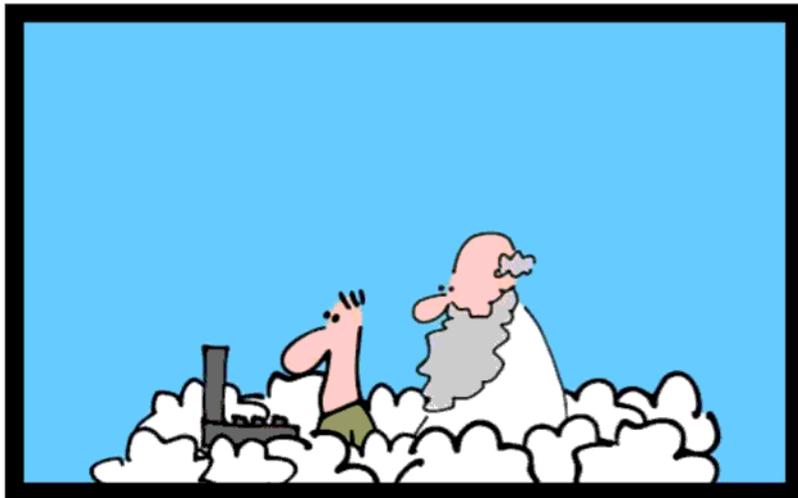
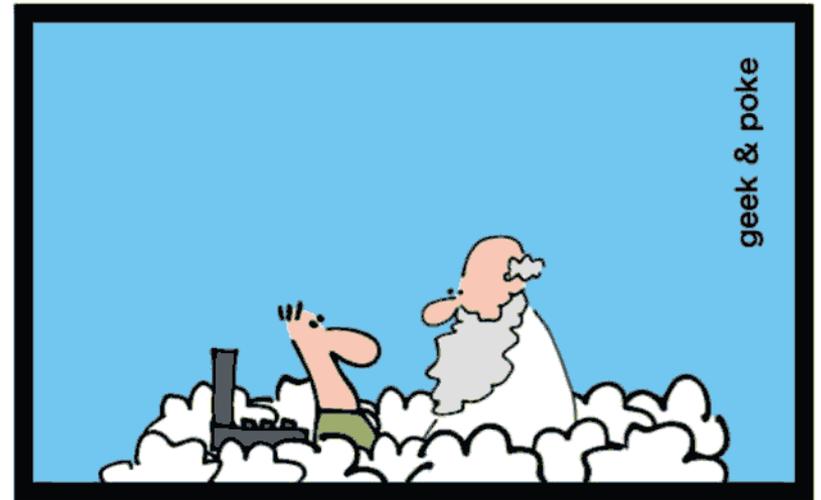
- **Paket** = (zusammengehörige) Menge von Klassen
  - Sowie evtl. von Unterpaketen und Interfaces
- **Hierarchischer Aufbau**
  - Paket „xyz“ im Paket „java“ → „java.xyz“
- Relevant für Strukturierung und **Zugriffskontrolle**
  - Klassen sowie deren Methoden und Attribute sind (ohne Angabe von `public`) **nur im eigenen Paket** sichtbar und zugreifbar
- Klassen befinden sich (logisch) immer in Paketen
  - **Paketdeklaration** direkt am Anfang einer Quelldatei, z.B.  
`package abc;`
  - Falls package-Deklaration fehlt: „unnamed package“
  - Konvention: kleingeschriebene Namen

## Pakete in Java (2)

- Attribute bzw. Methoden von Klassen können **vollqualifiziert** (d.h. mit dem Paketnamen) benannt werden
  - Z.B.: `java.lang.String.substring`  

    - Paket
    - Klasse
    - Methode
- **Importieren** von Klassen (als Namensabkürzung) aus Paketen
  - Z.B. `import java.util.Random;`  
(es wird diese Klasse importiert; kann mit Namen „Random“ benutzt werden)
  - Oder `import java.util.*;`  
(es wird alles aus diesem Paket „java.util“ importiert)
- Das Java-System bietet eine Reihe von **Standardpaketen** mit nützlichen Klassen und Methoden für fast jeden Zweck an
  - Z.B. `java.lang`, `java.io`, `java.net`, `java.applet`, `java.util`,...

# THE GOD AND THE CODER



DAY ONE

# Eigene Pakete

## Beispiel.java:

```
import myPack.*;
class Beispiel {
    ...main...{
        Stack S;
        ...
        Queue Q;
    }
}
```

Alternativ zu „import“: jew. vollqualifizierte Angabe, also z.B. `myPack.Queue`

## Stack.java:

```
package myPack;
public class Stack
{...}
class Hilfselement
{ void setzen()
  {...}
}
```

## Queue.java:

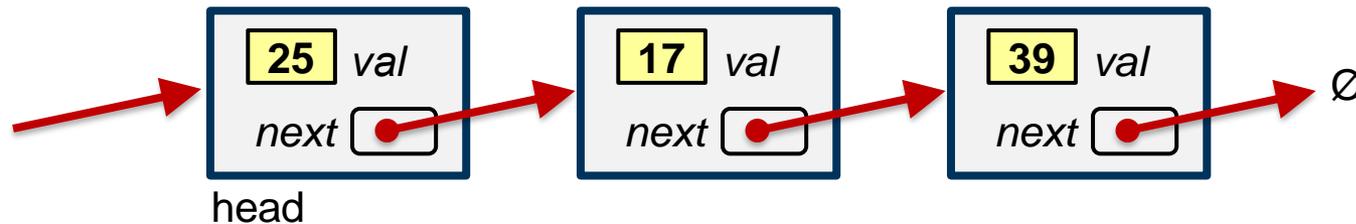
```
package myPack;
public class Queue
{...}
class X
{ private void y()
  {...}
}
```

Gemeinsames Paket (aufgeteilt in getrennte Dateien)

- Die Klasse Stack in der Datei Stack.java (bzw. Queue in Queue.java) muss mit dem **Zugriffsmodifikator** „public“ qualifiziert werden, da diese Klassen ausserhalb des Paketes MyPack (in Beispiel.java) verwendet werden sollen
  - Entsprechendes würde für öffentl. Methoden / Attribute von Stack und Queue gelten
- Die Methode „setzen“ ist innerhalb von myPack (also z.B. von Queue aus) zugreifbar, nicht aber von ausserhalb des Paketes (etwa von Beispiel aus)
- y ist nur von Methoden der Klasse X aus zugreifbar (da „private“)

## Beispiel: Ein Paket für int-Listen

- Es geht um **verkettete Listen** der folgenden Art:



- Nutzer unseres Paketes sollen „**Listenelemente**“ am Anfang (am Listenkopf: „head“) **hinzufügen** und **entfernen** können
  - **add\_head**, **remove\_head**
- Ausserdem sollen sie weitere **nützliche Methoden** bekommen:
  - Z.B. wie viele Elemente die Liste enthält (**size**)

## Beispiel: Ein Paket für int-Listen (2)

```
package listPack;
```

```
class ListElem {  
    int val;  
    ListElem next;
```

ListElem ist ausserhalb des Paketes nicht sichtbar

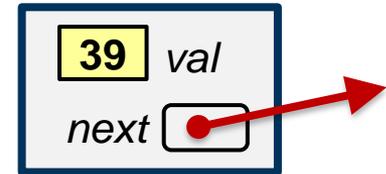
```
    public ListElem(int i, ListElem e) {  
        val = i;  
        next = e;  
    }  
}
```

Konstruktor

Hierauf soll das neue Listenelement mit „next“ zeigen

```
public class List {  
    ...  
}
```

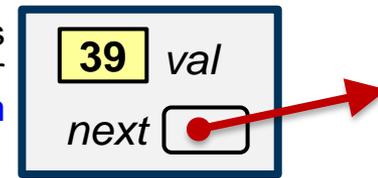
Fortsetzung des Paketes „listPack“ auf der nächsten Slide →



Ein Objekt als Instanz der Klasse [ListElem](#)

```
package listPack;
class ListElem...
```

Ein Objekt als  
Instanz der  
Klasse `ListElem`



```
public class List {
    private ListElem first = null;
    private int size = 0;

    public int size() {return size;}
    public void add_head(int i) {
        first = new ListElem(i, first);
        size++;
    }

    public int remove_head() {
        int i = first.val;
        first = first.next;
        size--;
        return i;
    }

    // evtl. weitere
    // Methoden
}
```

Kein Zugriff von aussen

Public-Methode liefert den Wert der gleichnamigen privaten Variablen

Neues Element vorne hinzufügen (einketten)

Grösse auf privater Variablen mithalten

Ausketten

Wert des (ehemaligen) vordersten Elements zurückliefern

Für **Nutzer** ist die **Komplexität reduziert**; sie sehen nicht die verkettete Struktur aus Referenzen und Objekten

```
public class List
```

```
public int size()
```

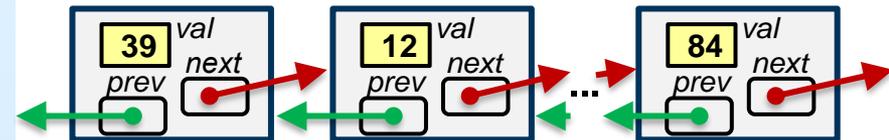
```
public void add_head(int i)
```

```
public int remove_head()
```

```
public void add_tail(int i)
```

```
public int remove_tail()
```

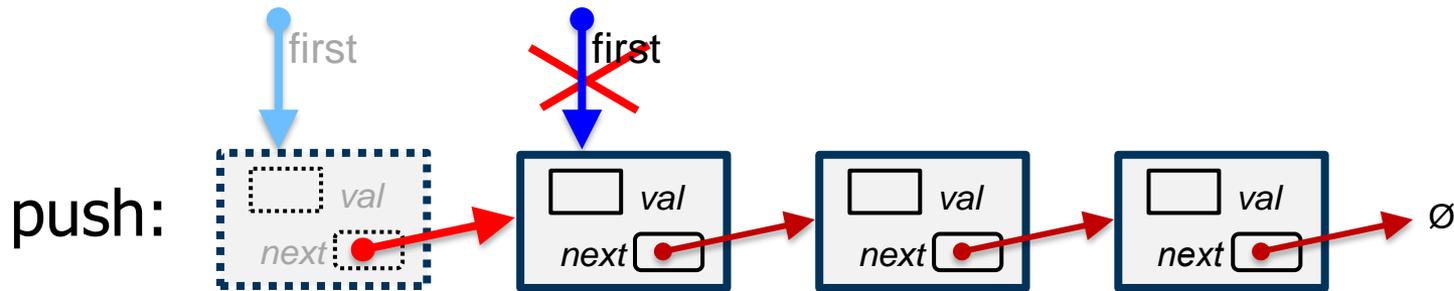
Sinnvolle Erweiterung:  
Doppelt verkettete Liste



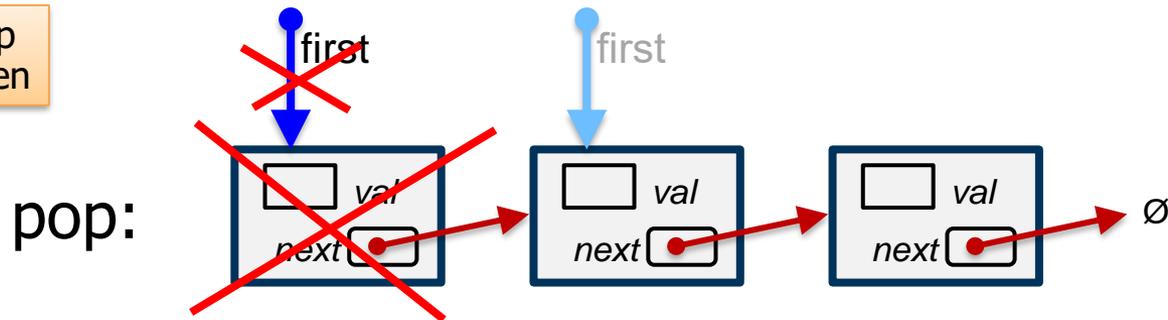
Für **Nutzer** ist die **Komplexität reduziert**; sie sehen nicht die verkettete Struktur aus Referenzen und Objekten

# Damit nun ein Stack

- Unter Verwendung zuvor realisierter Datentypen (z.B. *List*) können **neue Datentypen** konstruiert werden
  - z.B. ein Stack mit *push*, *pop* etc.
- Interne Realisierung von **push** bzw. **pop** (nach aussen nicht sichtbar):



Stack kann im Prinzip „beliebig“ gross werden



# Nutzung des Listen-Pakets für die Realisierung einer Stacks

```
// Stack aus einer int-Liste
import listPack.List;
public class Stack {
    private List L;
    public Stack() {
        L = new List();
    }
    public boolean empty() {
        return (L.size() == 0);
    }
    public void push(int i) {
        L.add_head(i);
    }
    public int pop() {
        // Test, ob !empty()...
        return L.remove_head();
    }
    public int top() {
        // Test, ob !empty()...
        int i = L.remove_head();
        L.add_head(i);
        return i;
    }
}
```

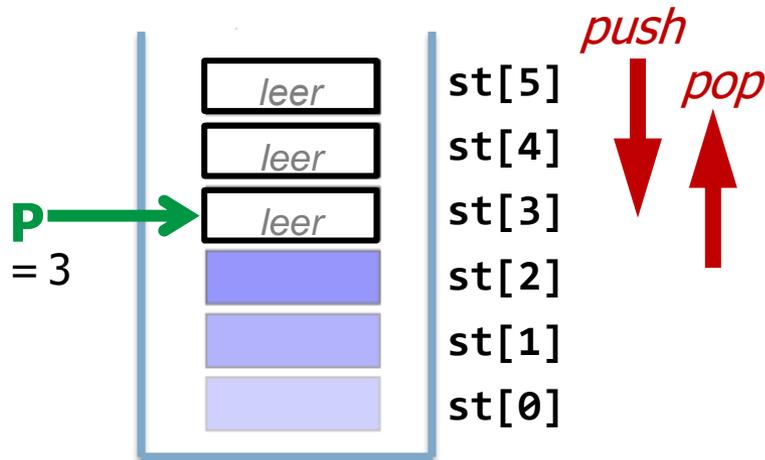
```
public class List {
    public int size()
    public void add_head(int i)
    public int remove_head()
```

Im Konstruktor des Stacks wird eine (leere) Liste erzeugt

Da List keinen direkten Zugriff auf die Listenelemente gestattet und die Werte nicht direkt ermittelt werden können, verwenden wir diesen Trick

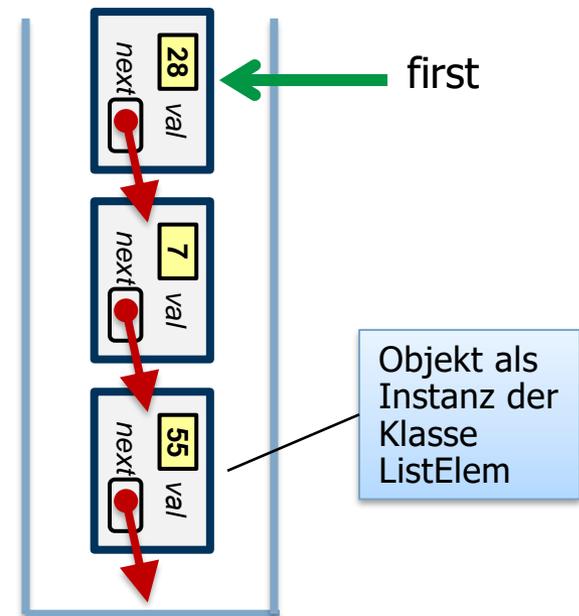
# Stack: Implementierungsmöglichkeiten

Wir wissen bereits von früher, wie ein Stack mit einem **Array** implementiert werden kann:



Array-Implementierung: **effizient**, aber der Stack ist a priori **begrenzt**.

Dagegen Implementierung mit **verketteter Liste** wie eben:



Der Stack ist so potentiell unbegrenzt, wegen „new“ jedoch etwas aufwendiger (**langsamer**) und hat Speicher-**Overhead** durch die next-Referenzen.

Gleiche Schnittstelle nach aussen und gleiche Funktion

# Austausch des Dienstleistungsanbieters

- Eine Klasse als Dienstleistungsanbieter kann „rücksichtslos“ gegen eine mit gleicher Schnittstelle (public-Variablen und -Methoden) und gleicher externer Wirkung **ausgetauscht** werden
  - Aber kann ein Array-basierter Stack (feste Maximalgrösse!) wirklich **exakt das gleiche Verhalten** aufweisen wie ein listenbasierter Stack?
  - → Was genau versteht man unter (funktionalem) „Verhalten“?

# Paket „Bruchrechnen“

$$\text{Bruch} = \frac{\text{Zähler}}{\text{Nenner}}$$

- Mit  $\text{Zähler} \in \mathbb{Z}$  und  $\text{Nenner} \in \mathbb{Z}$  ist  $\text{Bruch} \in \mathbb{Q}$ 
  - Strukturell ist ein Bruch allerdings  $\in \mathbb{Z} \times \mathbb{Z}$
  - Wir beschränken uns hier auf Werte  $\geq 0$

Engl. Bezeichnungen:  
numerator / denominator

- 
- Wir greifen nachfolgend aus „Informatik I“ das Beispiel „**Rechnen mit rationalen Zahlen**“ (modifiziert) wieder auf
    - Beachte aber: In Java gibt es im Unterschied zu C++ keine Operator-Überladung; man schreibt daher etwas ungewohnter  
 $c = a.\text{mult}(b)$  statt der Infix-Notation  $c = a * b$  oder z.B.:  
 $c = a.\text{plus}(\text{new Bruch}(1,6))$

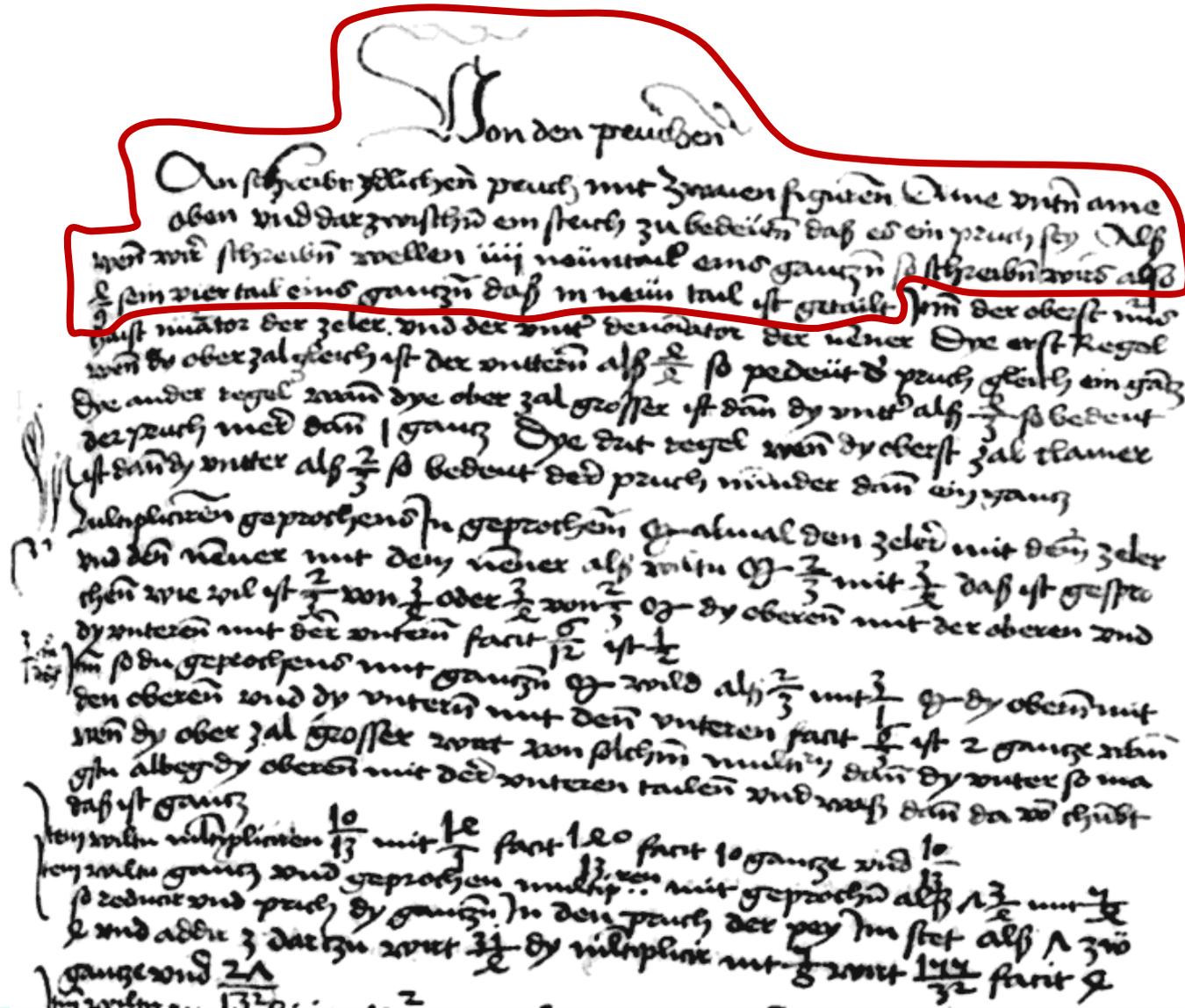
„Ein Bruch ist ein Vielfaches von einem Teile der Einheit. Zu einem Bruche braucht man daher immer zwei Zahlen, die eine Zahl, die andeutet in wie viele Teile die Einheit geteilt wurde, und nach welcher man diese Teile benennet, und welche Zahl daher Nenner heisst, und die zweite Zahl, welche angibt, wie viele solche Teile zusammengezählt werden, daher Zähler heisst, und schreibt einen Bruch auf die Art, dass man den Zähler über den Nenner anbringt, und beide durch einen Querstrich trennet.“  
[Leopold Carl Schulz von Straßnicki: Handbuch der besonderen und allgemeinen Arithmetik, 1844]

# Zwaien figurenn – Aine vntnn aine oben

Bruch =  $\frac{\text{Zähler}}{\text{Nenner}}$

Vor der Erfindung des Buchrucks durch Johannes Gutenberg (um 1450) gab es Schriften nur als **Manuskripte**. Eine erste weitgehend auf Deutsch gehaltene Sammlung mathematischer Handschriften verfasste zwischen 1450 und 1465 **Fridericus Amann**, ein Mönch aus Regensburg. Der Teil zum Bruchrechnen beginnt so:

**Von den pruchen**  
 Man schreibt ydlichenn pruch mit Zwaien figurenn. Aine vntnn aine oben vnd darzwischnn ein strich zu bedeutnn daß es ein pruch sey. Alß wenn wirr schreibnn wellen iiii neüntail eins gantznn so schreibnn wirr also  $\frac{4}{9}$  sein vier tail eins gantznn dass in neün tail ist getailt.



lat.: numerator

lat.: denominator

# ...ober der zeler, unter der neñer

$$\text{Bruch} = \frac{\text{Zähler}}{\text{Nenner}}$$

Aus dem ersten **gedruckten** deutschsprachigen Lehrbuch zur Algebra *Behend unnd Hübsch Rechnung durch die kunstreichen regeln Algebre, so gemeincklich die Coss\** genennt werden von Christoff Rudolff (Strassburg, 1524):

Brüch sein zweierley / etlich heissen schlechte  
brüch / etlich prüch von prüch. Ein schlechter prüch  
würt geschriben mit zweien zalen vnnd mit einem  
strichlen darzwischen / heist die ober der zeler / die  
vnter der neñer. In außspröchung müß man zum  
ersten bestimme den zeler / darnach den nenner mit  
züsetzung des wörtlens teil / als  $\frac{2}{5}$  ist der zeler 2 / der  
nenner 5. würt außgespröchen zweifünffteil. Dann  
prüch sein nichts dan teil eins andern dings / Nenn  
lich der nenner zeigt an in wievil teil das ganz bro  
chen sei / derselben etlich zelt die ober zal.

\*) „Die Coss“: Bei al-Chwarizmi und anderen arabischen Mathematikern wurde die Unbekannte (also die „Variable x“) als „die Sache“ bezeichnet; die Kunst des AuflöSENS nach x hiess im Italienischen seinerzeit dann „l'arte della cosa“.



Christoff Rudolff

„Es hat Christoff Rudolff vom Jawer (löblicher ge= dechnis) anno 1524 die wunderbarliche und gantz Philosophische Kunst dess rechnens, genennet Die Coss, in deutsche sprach durch den Truck gebracht, so gantz getrewlich und so klar und deutlich, das ich die selbige Kunst ohn allen mündtlichen underricht verstanden hab (mit Gottes hülf) und gelernet.“ [Michael Stifel, 1553]

„Jawer“:  
Stadt  
Jauer in  
Schlesien,  
heute als  
Jawor in  
Polen

Mehr zu Michael Stifel →

# Michael Stifel (1487 – 1567)

$$\text{Bruch} = \frac{\text{Zähler}}{\text{Nenner}}$$

Michael Stifel war ein mit Martin Luther befreundeter Theologe und Reformator. Er berechnete das Ende der Welt auf den 19. Oktober 1533 um 8 Uhr morgens und predigte den **Weltuntergang** so überzeugend, dass die Bauern in der Umgebung ihre Arbeit einstellten und viele Besitztümer aufgeben und verschenkt wurden. Der Weltuntergang blieb aber bekanntlich aus; die Redewendung „**einen Stiefel rechnen**“ oder „einen Stiefel reden“ (neuerdings auch verallgemeinert zu „einen Stiefel fahren“ etc.) geht auf diese Affäre zurück.

Auf Betreiben von Luther wurde Stifel in Schutzhaft genommen und damit vor dem Zorn der Gemeindemitglieder gerettet. In der Folge widmete sich Stifel ganz der Mathematik und wurde sogar der erste Professor für Mathematik an der Universität Jena. Bereits 1544 veröffentlichte er sein Hauptwerk „**Arithmetica integra**“, eine Zusammenfassung der damals bekannten Arithmetik und Algebra. Er führt u.a. die Bezeichnung „Exponent“ ein und gibt Rechenregeln für Potenzen an. Implizit entdeckt er die **Logarithmen** (zur Basis 2); er macht darauf aufmerksam, dass den Additions- und Subtraktionsoperationen in der arithmetischen Reihe

$-2, -1, 0, 1, 2, 3, 4, 5, \dots$

die Multiplikations- und Divisionsoperationen in der dazugehörigen geometrischen Reihe

$\frac{1}{4}, \frac{1}{2}, 1, 2, 4, 8, 16, 32, \dots$

Gedenktafel  
in der  
Schloss-  
strasse in  
Wittenberg



Ein Rechenbüchlein vom End Christ

entsprechen. („*Qualiacunque facit Arithmetica progressio additione, & subtractione, talia facit progressio Geometrica multiplicatione & diuisione ... Vide ergo ... Sicut ex additione (in superiore ordine) 3 ad 5 fiunt 8, sic (in inferiore ordine) ex multiplicatione 8 in 32 fiunt 256.*“) Dass dies auch für Brüche kleiner als 1 und damit für negative Exponenten („*numeris Absurdis*“) gilt, findet er erstaunlich und hübsch zugleich. Spätere Entdecker und Nutzer der Logarithmen, u.a. auch Bürgi und Kepler, wurden von Stifels Überlegungen (zumindest indirekt, z.B. über Simon Jacob) beeinflusst.

# Paket „Bruchrechnen“

$$\text{Bruch} = \frac{\text{Zähler}}{\text{Nenner}}$$

- „Tools“ für das Rechnen mit **rationalen Zahlen** in **Bruchform**
  - Vermeidet so gegenüber Gleitpunktzahlen mögliche Rundungsfehler
  - Hier: *long* statt *int* und *double* statt *float* → grösserer Wertebereich

```
package bruchPak;
```

```
public class Bruch {
```

```
    private long Zaehler = 0;
```

```
    private long Nenner = 1;
```

Lokale Daten sind 'private' und geeignet vorbesetzt

```
    public void setzen(long z, long n) {
```

```
        if (z < 0 || n <= 0)
```

```
            System.out.println("*** Fehler Wertebereich");
```

```
        else {
```

```
            Zaehler = z; Nenner = n; kuerzen();
```

```
        }
```

```
    }
```

```
    public Bruch(long z, long n) {setzen(z,n);} } }
```

```
    public Bruch(long z) {setzen(z,1);} }
```

```
    public Bruch() {setzen(0,1);} }
```

Gleich in gekürzter Form abspeichern – Methode „kuerzen()“ kommt gleich

Professionellere Fehlerbehandlung mit exceptions später

Drei überladene Konstruktoren

Könnte man sich das nicht sparen?

```
// Fortsetzung class Bruch nächste Seite
```

# Paket „Bruchrechnen“ (2)

$$\text{Bruch} = \frac{\text{Zähler}}{\text{Nenner}}$$

```
public double ratio() {  
    return (double)Zaehler / (double)Nenner;  
}
```

Explizite Typumwandlung ("type cast")

```
public void ausgeben() {  
    System.out.println(Zaehler + "/" + Nenner +  
        " = " + ratio());  
}
```

Z.B.:  $3/4 = 0.75$

Vorsicht bzgl. Überschrei-  
tung des Wertebereichs

```
public boolean kleiner(Bruch y) {  
    return (Zaehler*y.Nenner < y.Zaehler*Nenner);  
}
```

Denkübung: Wie wäre es  
mit `ratio() < y.ratio()` ?

```
public boolean identisch(Bruch y) {  
    return (!this.kleiner(y) && !y.kleiner(this));  
    // Bei gekuerzten Bruechen geht auch ein jeweili-  
    // ger Vergleich mit "==" von Zaehler und Nenner  
}
```

```
private static long ggT(long u, long v) {...}  
private static long kgV(long u, long v) {...}
```

Algorithmen für **ggT**  
bzw. **kgV** folgen (→  
euklidischer Algor.)

**ggT** (grösster gemeinsamer Teiler) und **kgV** (kleinstes gemeinsames Vielfaches) sind hier in Form einer klassenbezogenen Funktionalität („static“) realisiert

# Paket „Bruchrechnen“ (3)

$$\text{Bruch} = \frac{\text{Zähler}}{\text{Nenner}}$$

```
public Bruch plus(Bruch y) {  
    long n = kgV(Nenner, y.Nenner); // Hauptnenner  
    long z = (n/Nenner)*Zaehler + (n/y.Nenner)*y.Zaehler;  
    return new Bruch(z,n); // Wird vom Konstruktor gekürzt  
}
```

Das Additionsergebnis ist ein *neuer* Bruch

```
public Bruch mult(Bruch y) {  
    return new Bruch(Zaehler*y.Zaehler, Nenner*y.Nenner);  
}
```

Durch die Multiplikationen kann es leicht zu einem **Überlauf** kommen → besser: Faktoren frühzeitig kürzen, um das Problem zu minimieren.

```
private void kuerzen() {  
    if (Zaehler == 0)  
        Nenner = 1;  
    else {  
        long g = ggT(Zaehler, Nenner);  
        if (g != 1) {  
            Zaehler = Zaehler/g;  
            Nenner = Nenner/g;  
        }  
    }  
}  
} // Ende der Klasse Bruch
```

„kuerzen“ ist eine von aussen nicht zugängliche Methode, die den Bruch in eine **Normalform** bringt

```
import bruchPak; // Anwendung:  
Bruch a = new Bruch();  
Bruch b = new Bruch(1,2);  
Bruch c = a.plus(b.mult(a));  
c.ausgeben();  
(a.plus(b)).ausgeben();  
c = c.plus(new Bruch(1,8));
```

# Nutzung des Pakets zum Bruchrechnen

```

import bruchPak;
...
Bruch a = new Bruch();
Bruch b = new Bruch(1,2);  $\frac{1}{2}$ 
Bruch c = new Bruch(5);

a.ausgeben();
a.setzen(2,3);  $\frac{2}{3}$ 
a.ausgeben();
b.ausgeben();
c.ausgeben();

c = a.mult(b);  $\frac{2}{3} * \frac{1}{2}$ 
c.ausgeben();

(a.plus(b)).ausgeben();  $\frac{2}{3} + \frac{1}{2}$ 
c = a.plus(b.plus(a));  $\frac{2}{3} + \frac{1}{2} + \frac{2}{3}$ 
c.ausgeben();

c = c.plus
    (new Bruch(1,8));  $\frac{11}{6} + \frac{1}{8}$ 
c.ausgeben();

```

Zur Erinnerung – die Konstruktoren:

```

public Bruch(long z, long n)
    { setzen(z,n); }
public Bruch(long z)
    { setzen(z,1); }
public Bruch() { setzen(0,1); }

```

$$0/1 = 0$$

$$2/3 = 0.666667$$

$$1/2 = 0.5$$

$$5/1 = 5$$

$$1/3 = 0.333333$$

$$7/6 = 1.16667$$

$$11/6 = 1.83333$$

$$47/24 = 1.95833$$

# Nutzung des Pakets zum Bruchrechnen (2)

```

c = (new Bruch(7,12)).plus(new Bruch(1,6));
c.ausgeben();
System.out.println(a.kleiner(b));
System.out.println(b.kleiner(a));
c = a;
System.out.println(a.identisch(c));
System.out.println(a.identisch(b));

a.setzen(0,3);
b.setzen(0,4);
c = a.plus(b);
c.ausgeben();
c.setzen(3,0);

```

$3/4 = 0.75$

false  
true

true  
false

$0/1 = 0$

\*\*\* Fehler Wertebereich

- Übung: Methode für Kehrwert erstellen (Sonderfälle wie  $0/1$  beachten!).
- Die interne Repräsentation von Zähler und Nenner durch „long“ ist beschränkt; vorstellbar sind andere Realisierungen, wo z.B. die **einzelnen Ziffern** von Zähler und Nenner in (potentiell beliebig langen) **verketteten Listen** gehalten werden.
- Analoge Pakete auch für andere mathem. Objekte, z.B. **Polynome**, **Matrizen**...

# Euklidischer Algorithmus für ggT (und kgV)

```
// Iterative Version mit  
// fortgesetzter Subtraktion  
public int ggT(int a, int b)  
{ while (a != b)  
  { if (a > b) a = a - b;  
    else      b = b - a;  
  }  
  return a;  
} // aus Informatik I bekannt
```

Einer der ältesten bekannten nicht-trivialen Algorithmen; wurde schon vor Euklid (~ 300 v. Chr.) verwendet

```
// Rekursive Version mittels  
// Division mit Rest, spart  
// viele Subtraktionen  
public int ggT(int a, int b)  
{ if (b == 0) return a;  
  else return ggT(b, a%b);  
} // aus Informatik I bekannt
```

## Wechselwegnahme

„Nimm immer die kleinere von der grösseren weg, bis ein Rest kommt, welcher die nächst-vorgehende Zahl genau misst. Dieser Rest ist das grösste gemeinschaftliche Mass der beiden gegebenen Zahlen.“ [Euklid]

Mit dem **ggT** eng verwandt ist das **kgV**; es gilt die Beziehung  **$a \times b = \text{ggT}(a,b) \times \text{kgV}(a,b)$** .

Frage zur fortgesetzten Subtraktion bei einer Prüfung zum General Certificate of Secondary Education (mittlerer Schulabschluss in England mit ca. 16 Jahren):

Q: How many times can you subtract 7 from 83, and what is left afterwards?

A: I can subtract it as many times as I want, and it leaves 76 every time.

*Wurde hier nach der fortgesetzten oder der „wiederholten“ Subtraktion gefragt?*

# Euklidischer Algorithmus für ggT

Berechnung des **ggT** nach dem **euklidischen Algorithmus** – hierbei werden keine aufwändig zu berechnende Primfaktorzerlegungen (Schulmethode zu Bestimmung des ggT!) benötigt; es geht mit dem Algorithmus viel einfacher! Das Verfahren beruht im Wesentlichen darauf, dass die Menge der Teiler zweier Zahlen  $a, b$  (mit  $a \geq b > 0$ ) identisch zur Menge der Teiler der beiden Zahlen  $a, a - b$  ist. Dies ist zugleich die Idee für die Schleifeninvariante zur Verifikation der Implementierungen oben. Die iterative Version (Prinzip des wechselseitigen Subtrahierens, sogen. **Wechselwegnahme**) ist über 2000 Jahre alt. Die modernere Divisionsversion ist effizienter („modus brevis“, erstmals beschrieben 1494 von Luca Pacioli, sie erspart viele Subtraktionen  $a = a - b$  im Falle  $a \gg b$ ): Da  $a \% b < a/2$  (Beweis?), werden die Zahlen schrittweise exponentiell kleiner. Der modus brevis beruht auf der mathematischen Tatsache  $\text{ggT}(a, b) = \text{ggT}(b, a - k b)$  für jede ganze Zahl  $k$ .

Die Schleifenabbruchbedingung „ $a \neq b$ “ lässt sich auch zu „ $b \neq 0$ “ oder „ $b > 0$ “ abändern (man beachte, dass wir hier keine negativen Zahlen betrachten); der Kern des Algorithmus wird daher in Form von mathematischem Pseudocode oft auch so notiert:

**while**  $b > 0$  **do**:

$$(a, b) = (b, a - \lfloor \frac{a}{b} \rfloor b)$$

**return**  $a$

„Die Wucht der Rückführung einer Berechnung auf sich selbst wird hier erreicht durch geschickte Reduktion der Parameterwerte in Richtung auf den trivialen Fall. Es ist, als hätte Euklid an die Informatik-Adepten gedacht, die Hunderte von Jahren nach ihm sich mit diesen Teilern herumschlagen müssten, obwohl sie daran kein Interesse haben.“ -- Frieder Nake

# Euklidischer Algorithmus für ggT – Beispiel

(1)	121393	%	75025	=	46368
(2)	75025	%	46368	=	28657
(3)	46368	%	28657	=	17711
(4)	28657	%	17711	=	10946
(5)	17711	%	10946	=	6765
(6)	10946	%	6765	=	4181
(7)	6765	%	4181	=	2584
(8)	4181	%	2584	=	1597
(9)	2584	%	1597	=	987
(10)	1597	%	987	=	610
(11)	987	%	610	=	377
(12)	610	%	377	=	233
(13)	377	%	233	=	144
(14)	233	%	144	=	89
(15)	144	%	89	=	55
(16)	89	%	55	=	34
(17)	55	%	34	=	21
(18)	34	%	21	=	13
(19)	21	%	13	=	8
(20)	13	%	8	=	5
(21)	8	%	5	=	3
(22)	5	%	3	=	2
(23)	3	%	2	=	1
(24)	ggT = 1				

fib

## Trace des Euklidischen Algorithmus nach Aufruf mit den benachbarten Fibonacci-Zahlen 121393 und 75025.

Aufeinanderfolgende Fibonacci-Zahlen (oder ein gemeinsames Vielfaches aufeinanderfolgender Fibonacci-Zahlen) für  $a, b$  benötigen (relativ zu Zahlen ähnlicher Grösse) besonders lang bei der Divisionsversion.

Eine solche Eingabe produziert auch (gleichermaßen für die Divisions- wie für die Subtraktionsmethode) hinsichtlich der Zwischenresultate einen Präfix der Fibonacci-Folge.

Man beachte bei diesem Beispiel, dass 75025 fünf Stellen hat; nach dem Theorem von Lamé ( $\rightarrow$  spätere Slides) gibt es daher nicht mehr als fünf Mal so viele, also 25, Iterationen.

# Euklidischer Algorithmus und Fibonacci-Zahlen

Schon der deutsche Rechenmeister **Simon Jacob** (1510 – 1564) erkannte, dass die Fibonacci-Zahlen (die damals allerdings noch nicht so genannt wurden) besonders viele Rechenschritte bei der ggT-Berechnung induzieren; er beschreibt dies in seinem **Rechenbuch** (1565 posthum herausgegeben von Pangratz Jacob von Coburgk, seinem Bruder), das nach Sitte der damaligen Zeit einen langen erklärenden und werbenden Titel trug:

„**Ein new und wolgegründt Rechenbuch, auff den Linien und Ziffern**, sampt der Welschen Practica und allerley Vortheilen, neben der Extraction Radicum, und von den Proportionen, mit vielen lustigen Fragen und Aufgaben. Dessgleichen ein vollkommener Bericht der Regel Falsi, mit neuwen Inventionibus, Demonstrationibus, und Vortheilen, so biß anher für unmöglich geschetzt, gebessert, dergleichen noch nie an Tag kommen. Und dann von der Geometrie, wie man mancherley Felder und Ebenen, auch allerley Corpora, Regularia und Irregularia, messen, Aream finden und rechnen soll.“

Wie doch diese Künste bey uns gleich wie bey andern Nationen in unser Muttersprach gepflanzt werden möchten [...] Ob ich dann eben der seyn müste / dem solche Künste zu pflanzen befohlen weren. -- Aus der Vorred.



Im Buch ist bezüglich des euklidischen Algorithmus folgendes interessant:

Wie findet man ein Zahl und die größte/  
die diesen Bruch  $\frac{77002051219739}{124591936076998}$  [...]   
zum kleinsten macht?

Die allgemeyne Regel ist/ Theil des Bruchs Nenner durch seinen Zähler/ nim nach dem den Zähler von Teil durch die Zahl/ die verbleiben/ ferner theil weiter den Theiler dieser andern Division durch den Rest so bleiben ist/ und solche theilung wiederhole so oft/ biß endlich ein mal nichts verbleibt/ welche Zahl dann in solcher Arbeit der letzte Theiler ist/ die macht den Bruch kleiner an seinen Zahlen/ und ist in dem fürgebrachten Bruch 19/ Hat solche Regel ihre Beweisung auß der 2 Proposition des 7 Buchs Euclidis/ darauß dann auch vernommen wird/ wann der letzte Theiler eins/ daß der Bruch kleiner zu machen unmöglich were/ und durch solche Regel findet man alle mal ein solche Zahl/ die die Zahlen des Bruchs so erkleinert/ daß sie kleiner zu machen unmöglich sein. Und hat obgesetzter Bruch ein wunderbarlich Art in ihm/ Nemlich daß er sich 54 mal dividiren leßt/ ehe man das gemein Maß oder die größte Zahl damit er aufgehoben wirdt/ findet/ mag derhalb wol ein Arithmetisch Labyrinth genant werden/ wird gemacht auß der vorgesetzten Ordnung der Zahlen/ da je die zwei so nechst vff einander folgen/ soill thun als die dritt folgend/ bringen je die erst und dritt inn einander multiplicirt ein weniger dann das Quadrat der mittleren/ darumb je weiter man solche Ordnung erstreckt/ je näher man zu der Proportz kommt/ Dauon Euclide die 11 Proposi des 2/ vnd 30 des 6 Buch handeln/ vnd wiewol man jimmer je näher kompt/ mag doch nimmermehr dieselb erreicht/ auch vbertreten werden.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

Das Frühneuhochdeutsch des Textes ist zwar gut verständlich, wirkt aber mit seiner inkonsistenten Orthographie und teilweise lautmalerischen Vokalschreibweise aus heutiger Sicht etwas skurril. (Anstatt „54 mal diuidirn leßt“ muss es übrigens korrekt „64 mal...“ heissen, offenbar handelt es sich um einen Druckfehler.)

Mit „**vorgesetzte ordnung**“ ist die Folge der **Fibonacci-Zahlen** von 1 bis 317811 gemeint, die als Randnotiz erscheint. Mit „bringen je die erst vnd dritt inn einander multiplicirt eins weniger dann das quadrat der mittleren“ meint Simon Jacob, dass  $\text{fib}(i) * \text{fib}(i+2)$  bis auf eine Differenz von 1 dem **Quadrat von fib(i+1)** entspricht; dies ist bemerkenswert, da es sich um die sogenannte Cassini-Identität handelt, die eigentlich erst 1680 von Jean-Dominique Cassini, dem seinerzeitigen Direktor der Pariser Sternwarte, entdeckt wurde! (Ein informelles Argument dazu, unter Herausfaktorisierung von  $\sqrt{5}$ , wäre:  $\phi^n * \phi^{n+2} = \phi^{2n+2} = \phi^{n+1} * \phi^{n+1}$ .)

Beim angesprochenen 11. Satz des 2. Buches von Euklid handelt es sich um den **Goldenen Schnitt** bzw. dessen Verhältniszahl  $\phi = (1+\sqrt{5})/2 = 1.618\dots$ ; Simon Jacob behauptet also  $\lim_{(n \rightarrow \infty)} \text{fib}(i) / \text{fib}(i-1) = \phi$ ; ebenfalls eine bemerkenswerte Erkenntnis!

Mit „**arithmetisch labyrinth**“ bezeichnet Jacob eine mühselige Rechenaufgabe, hier konkret gegeben durch einen Bruch aus (gemeinsamen Vielfachen) aufeinanderfolgender **Fibonacci-Zahlen**; die „wunderbarlich art“ ist die Eigenschaft, dass dabei der euklidische Algorithmus besonders viele Schritte benötigt. Im konkreten Fall des von Jacob „fürgebrachten“ Bruchs ist  $77002051219739 = 19 * 4052739537881 = 19 * \text{fib}(62)$  und  $124591936076998 = 19 * 6557470319842 = 19 * \text{fib}(63)$ . Man darf wohl annehmen, dass Jacob davon ausging (bzw. vermutlich sogar überzeugt war), dass benachbarte Fibonacci-Zahlen den „worst case“ darstellen – ob dahinter mehr steckt als begründete Intuition, die aus der Praxis gewonnen wurde, bleibt allerdings offen.

# Fibonacci in Zürich

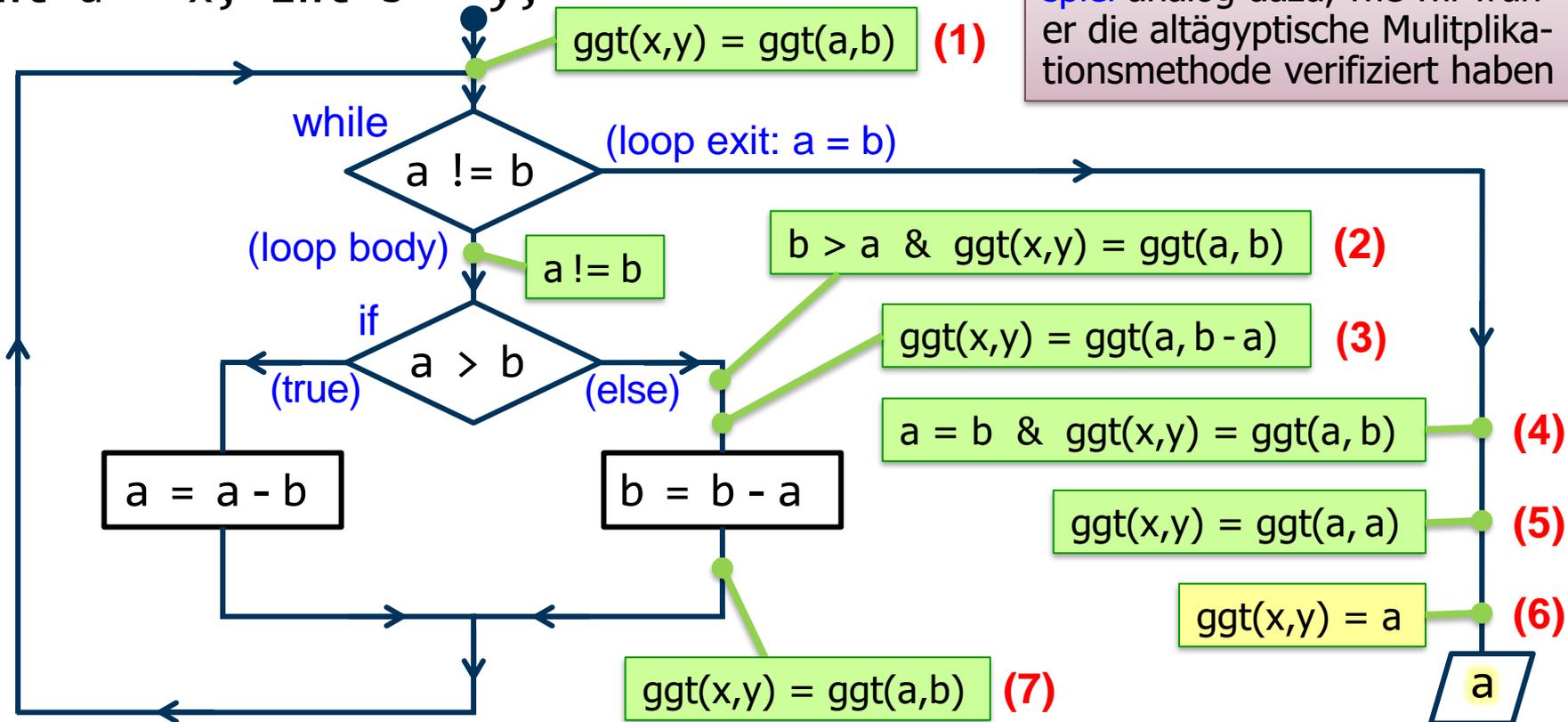


Wo in Zürich befindet sich diese Lichtskulptur zu den Fibonacci-Zahlen?

# Euklidischer Algorithmus für ggT – Verifikation

```
public int ggT(int x, int y)
{int a = x; int b = y;
```

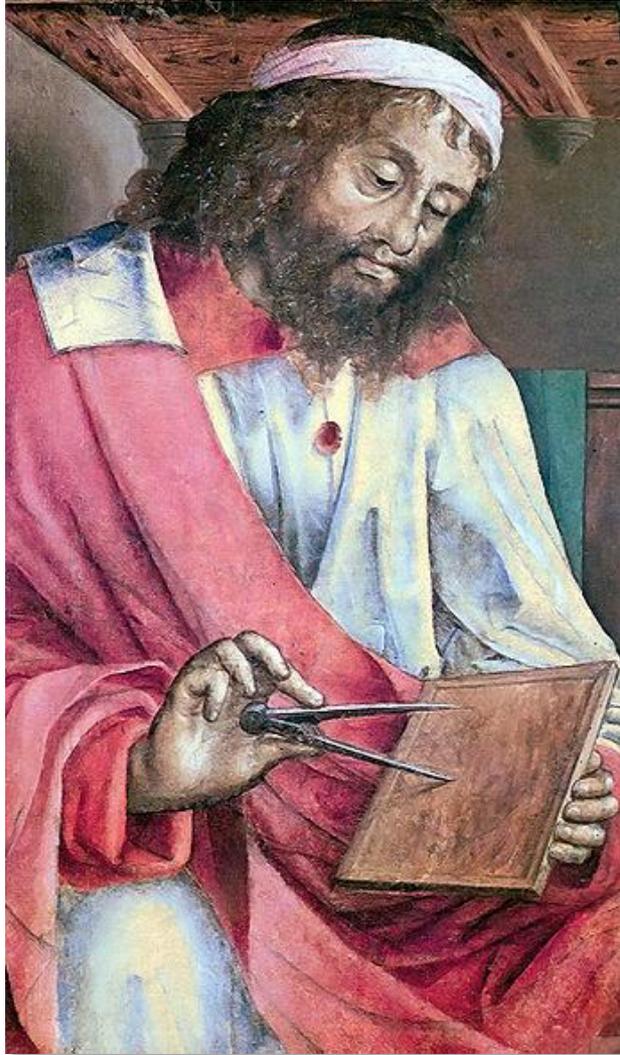
ggT-Verifikation als **Übungsbeispiel** analog dazu, wie wir früher die altägyptische Multiplikationsmethode verifiziert haben



Die formalen Parameter  $x$  und  $y$  werden in der Methode nicht verändert. (3) folgt aus (2), da (für  $b > a$ ) jede Zahl, die  $a$  und  $b$  teilt, auch  $b - a$  teilt (der Teiler kann ausgeklammert werden). (7) folgt aus (3) aufgrund des Zuweisungsaxioms (vgl. frühere Diskussion bei der Hoare-Logik). (1) bzw. (7) ist die Schleifeninvariante. (6) folgt aus (5), da offenbar stets  $ggT(z,z) = z$  gilt. Der nicht detaillierter ausgeführte linke if-Ast entspricht dem rechten in analoger Weise.

# Euklidischer Algorithmus (Euklid: Elemente)

Darstellung von Euklid durch Justus von Gent, niederländischer Maler und Zeichner (ca. 1430)



## Der 2. Satz.

Es sind zwey Zahlen gegeben, die nicht Primzahlen zu einander sind: man soll ihr grösstes gemeinschaftliches Maaß finden.

### Erster Fall.

Es seyen, A, B, die beyden gegebenen Zahlen, so daß die grössere, A, von der kleinern, B, genau gemessen werde. Nun misst B auch sich selbst, folglich ist B ein gemeinschaftliches Maaß von A und B; aber auch das grösste, weil offenbar keine grössere Zahl, als B, die B messen kann.

$$\left. \begin{array}{l} A \ 8 \quad B \ 4 \end{array} \right\}$$

### Zweyter Fall.

Es seyen AB, CD, die beyden gegebenen Zahlen, so daß die grössere, AB, von der kleinern, CD, nicht genau gemessen werde. Nimm immer die kleinere von der grössern weg, bis ein Rest kommt, welcher die nächstvorgehende Zahl genau misst. Dieser Rest ist das grösste gemeinschaftliche Maaß der beyden gegebenen Zahlen.

$$\left. \begin{array}{l} A \ \underline{4} \ E \ \underline{6} \ B \\ C \ \underline{2} \ F \ \underline{4} \ D \\ G \ \text{—} \end{array} \right\}$$

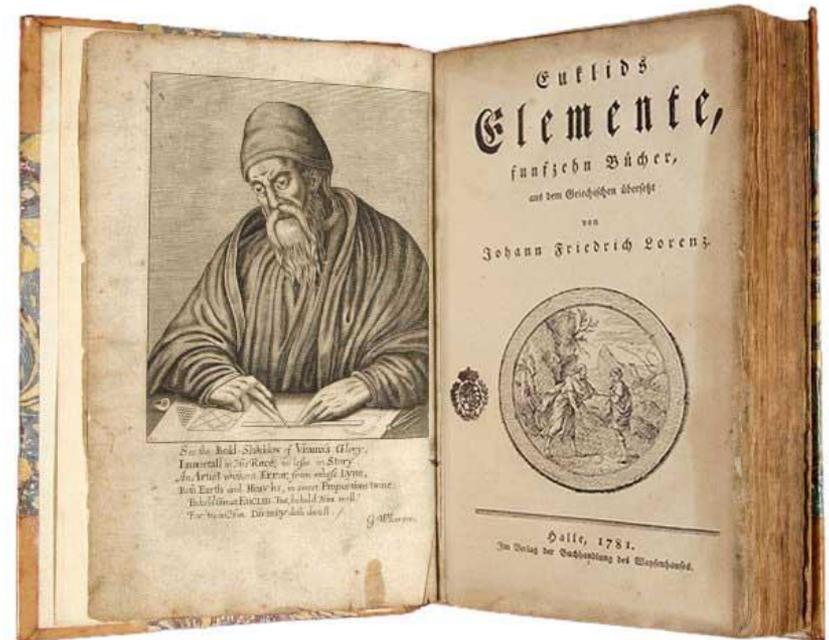
- 1.) Daß dieser Rest grösser als die Einheit, folglich eine Zahl sey, siehet man daher, weil sonst (7, 1. S.) AB, CD, Primzahlen zu einander wären, gegen das Angenommene.

# Euklidischer Algorithmus (Euklid: Elemente)

- 2.) Daß dieser Rest das gemeinschaftliche Maaß von AB, CD, sey, wird so gezeigt: Es lasse AB, von CD gemessen, den Rest  $AE < CD$ , und CD, von AE gemessen, den Rest CF, welcher die nächstvorgehende Zahl AE genau messe. CF mißt AE, und AE die DF, folglich mißt CF die DF; aber auch sich selbst, folglich die ganze CD. Nun mißt CD die BE. Folglich mißt auch CF die BE; aber auch die EA, folglich die ganze AB. Demnach ist der Rest CF ein gemeinschaftliches Maaß von AB, CD.
- 3.) Daß dieser Rest CF aber auch das größte gemeinschaftliche Maaß sey, beweist man also: Es sey ein andres,  $G > CF$ . G mißt die CD, und CD die BE, folglich mißt G die BE; aber auch die ganze AB, folglich auch den Rest AE. Nun mißt AE die DF. Folglich mißt G die DF; aber auch die ganze CD, folglich den Rest CF, welches unmöglich, weil  $G > CF$ .

## Zusatz.

Hieraus erhellet: wenn zwey Zahlen von einer dritten gemessen werden, daß auch das größte gemeinschaftliche Maaß der beyden Zahlen von dieser dritten Zahl müsse gemessen werden.

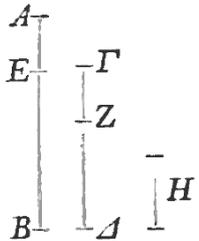


→ Euklids Elemente, funfzehn Bücher, aus dem Griechischen übersetzt von Johann Friedrich Lorenz, Halle 1781 (Buch 7, 2. Satz).

Die älteste erhaltene griech. Handschrift der „Elemente“ stammt von 888; sie entspricht der Ausgabe in der Bearbeitung von Theon von Alexandria. Relevant ist vor allem eine Handschrift aus dem 10. Jh. (Vatikanbibliothek), die einen Text vor der Bearbeitung durch Theon enthält; auf ihr beruhen alle neueren Editionen. Die erste gedruckte Ausgabe erschien (auf griechisch) 1533 in Basel; bis Mitte des 19. Jahrhunderts war das Werk neben der Bibel das auflagenstärkste Buch der Menschheitsgeschichte.

# Euklidischer Algorithmus (Euklid: Elemente)

at si  $\Gamma\Delta$  non metitur  $AB$ , minore numerorum  $AB$ ,  $\Gamma\Delta$  semper uicissim a maiore subtracto relinquetur numerus aliquis, qui proxime antecedentem metietur. unitas enim non relinquetur; sin minus,  $AB$ ,  $\Gamma\Delta$  inter se primi erunt [prop. I]; quod contra hypothesis est. ergo numerus aliquis relinquetur, qui proxime antecedentem metietur. et  $\Gamma\Delta$  metiens  $BE$  relinquat se



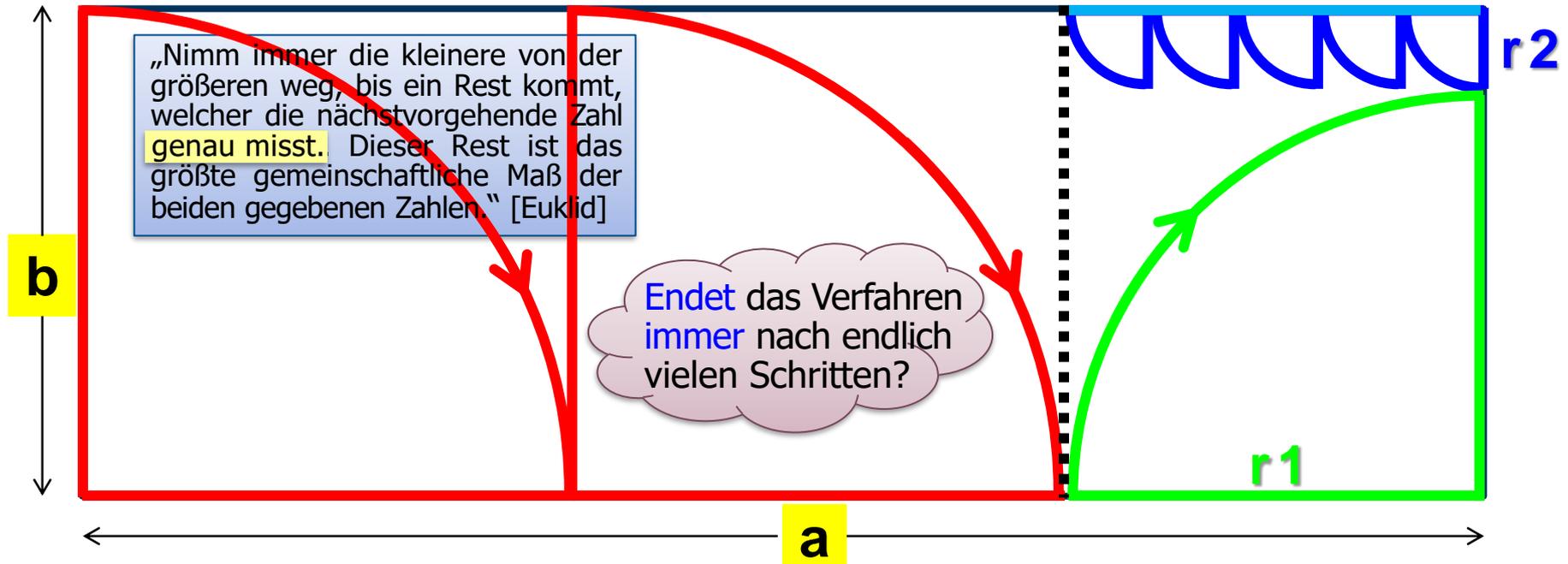
At si  $\Gamma\Delta$  non metitur  $AB$ , **minore numerorum  $AB$ ,  $\Gamma\Delta$  semper uicissim a maiore subtracto** relinquetur numerus aliquis, qui proxime antecedentem metietur.

*Vicissim* (Adv.) wird hier in der Bedeutung *untereinander, gegenseitig, abwechselnd* gebraucht; bezeichnet eine abwechselnde Handlung eines oder zweier Akteure

Εἰ δὲ οὐ μετρεῖ ὁ  $\Gamma\Delta$  τὸν  $AB$ , τῶν  $AB$ ,  $\Gamma\Delta$  ἀνθυφαιρουμένου ἀεὶ τοῦ ἐλάσσονος ἀπὸ τοῦ μείζονος λειφθήσεται τις ἀριθμὸς, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. μονὰς μὲν γὰρ οἱ λειφθήσεται· εἰ δὲ μὴ, ἔσονται οἱ  $AB$ ,  $\Gamma\Delta$  πρῶτοι πρὸς ἀλλήλους· ὅπερ οὐχ ὑπόκειται. λειφθήσεται τις ἄρα ἀριθμὸς, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. καὶ ὁ μὲν  $\Gamma\Delta$  τὸν  $BE$  μετροῦν

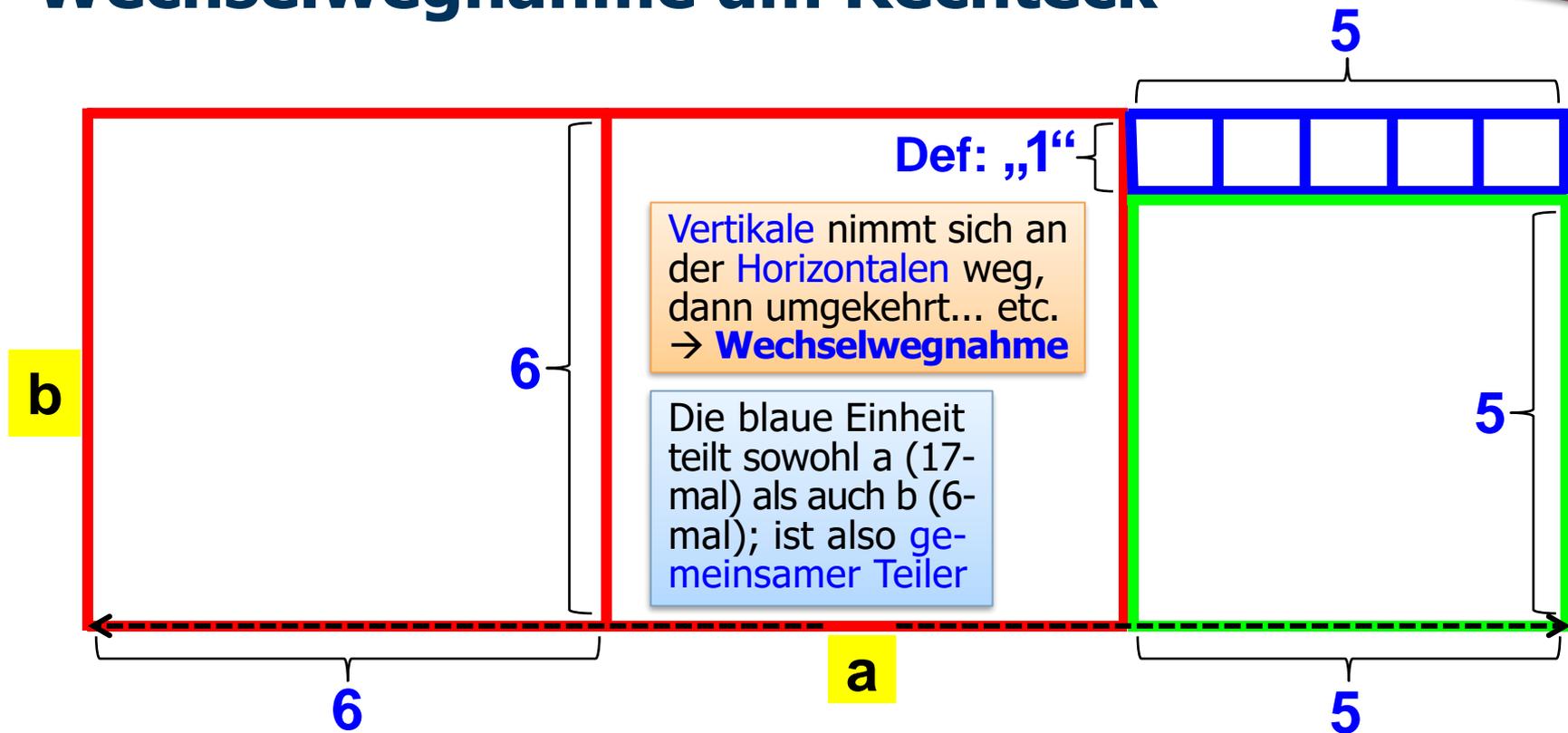
Partizipform *ἀνθυφαιρουμένου* zum Prozess des wechselseitigen Wegnehmens (→ „**Wechselwegnahme**“ = *Antipharesis* bzw. *Antepheiresis*, *Anthypharesis* etc., vgl. *aphairesis*: Wegnahme, Entziehen)

# Wechselwegnahme am Rechteck



Die kleinere Seite  $b$  wird (z.B. mit dem Zirkel) so oft an der grösseren, waagerechten Seite  $a$  abgetragen (rot), bis ein Rest  $r_1$  bleibt. Danach wird dieser an der (nunmehr längeren, senkrechten) Seite  $b$  abgetragen (grün) bis zu einem dortigen Rest  $r_2$ . Mit diesem wird dann wiederum die waagerechte Seite abgemessen (blau); in diesem Beispiel geht dies nach fünf Mal exakt auf – das gemeinsame Mass entspricht dem blauen Quadrat. Die senkrechte und waagerechte Seite wechseln im Prozess mehrfach ihre Rolle.

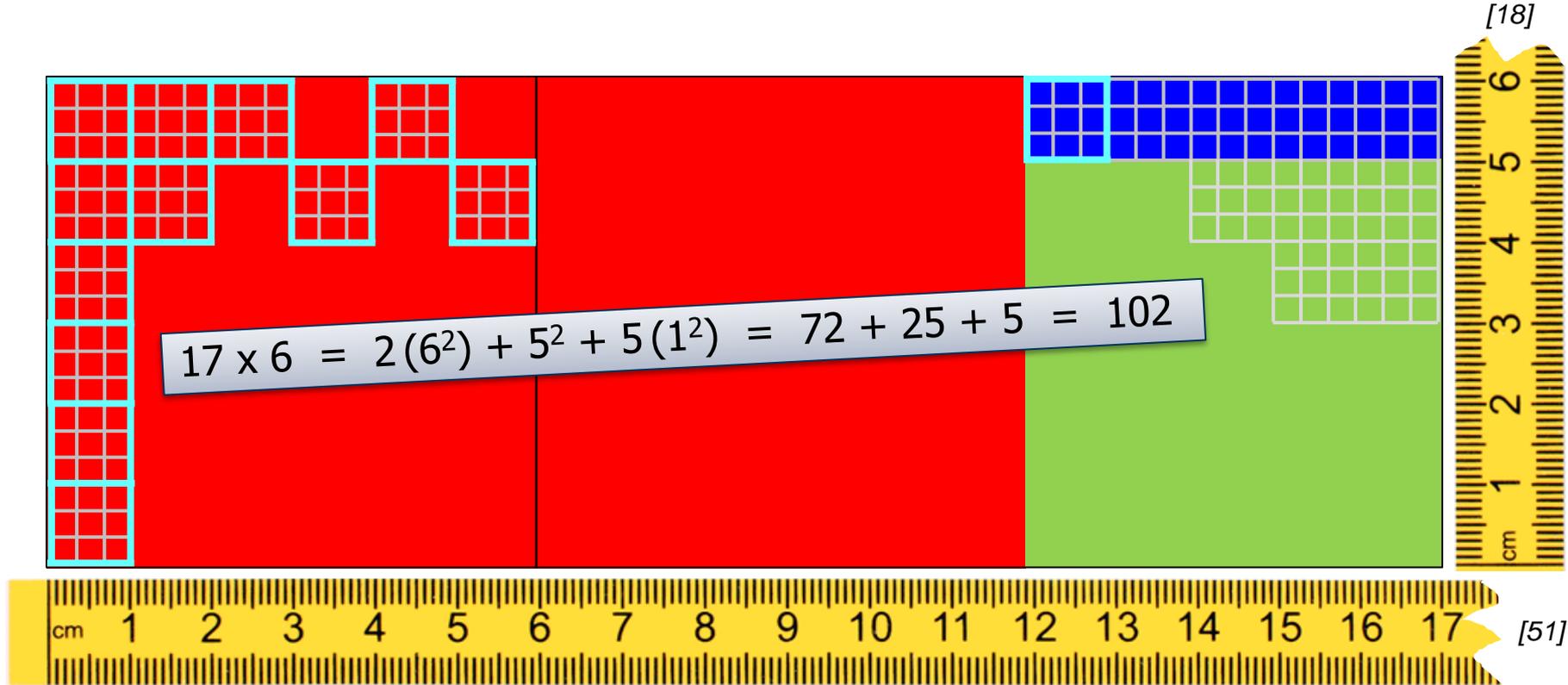
# Wechselwegnahme am Rechteck



Das grüne Quadrat ist 5 blaue Einheiten lang und breit, das rote 6. Damit ist im ursprünglichen Rechteck  $a = 17$  (es ist also 17 „blaue“ lang), die Höhenkante  $b$  misst 6 „blaue“. „Ein blau“ stellt die **gemeinsame Masseinheit** dar. Wäre das blaue Quadrat z.B. 12mm „im Quadrat“, dann wäre  $a = 204\text{mm}$  und  $b = 72\text{mm}$ . Da die blaue Einheit in  $a$  und  $b$  ganzzahlig enthalten ist, ist 12 ist also (**grösster!?**) **gemeinsamer Teiler** von 204 und 72. Das Prinzip der Wechselwegnahme war Handwerkern und Baumeistern (die mit „Mass-Stäben“ und Messschnüren hantiert haben, um das gemeinsame Mass zweier Strecken zu finden) sicherlich schon lange bekannt, bevor die Harmonie von Zahlenverhältnissen von den Pythagoreern zum mathematischen und metaphysischen Prinzip erhoben wurde und Euklid den Algorithmus aufschrieb.

# Wechselwegnahme am Rechteck

Historische  
Notiz



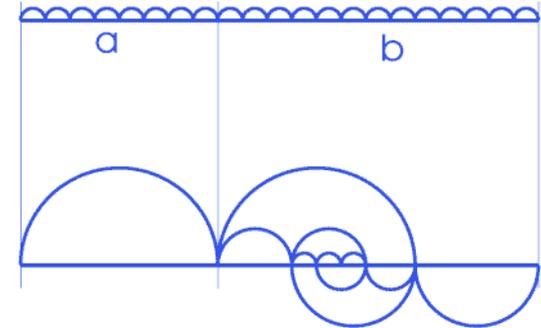
- Für 17 und 6 ist der ggT = 1; statt mit 102 Quadraten der Grösse 1 x 1 könnte man (in einem anderen „Mass-Stab“ und entsprechend feinerem Koordinatengitter) das Rechteck aber z.B. auch mit 918 Quadraten der Grösse 1/3 x 1/3 (relativ zum Originalmassstab) pflastern – man sucht aber ja gerade das *grösste* gemeinsame Mass.
- Bei 51 und 18 (→ 918) wäre der ggT dann nicht 1, sondern 3-mal grösser, also 3.

# Wechselwegnahme

Auf dem Tisch liegen 2 Reihen von Streichhölzern. Von der jeweils längeren Reihe sollen so viele weggenommen werden, wie in der kürzeren vorkommen. Aufgehört wird, wenn beide Reihen gleich lang sind.

Historische  
Notiz

Das Prinzip der Wechselwegnahme wurde in der griechischen Mathematik der Pythagoreer vor allem auf die Grössen von Strecken geometrischer Figuren angewendet. Man suchte nach dem gemeinsames Mass e zweier Strecken  $a$ ,  $b$ , das sich dann ergibt, wenn beim euklidischen Algorithmus schliesslich ein Rest von 0 herauskommt. Diese Strecken haben a priori zwar keine ganzzahlige Länge – aber dafür muss man die (gemeinsame) Längeneinheit doch nur klein genug wählen, oder?



Restzweifel bleiben: Gibt es vielleicht zusammengehörige Strecken, die kein gemeinsames Mass besitzen, also inkommensurabel sind? Wo die fortgesetzte wechselseitige Wegnahme dann nie terminieren würde? Und was würde dies dann bedeuten?

Und wirklich stellte sich heraus – wohl zum Entsetzen der Pythagoreer, die ja daran glaubten, dass der Kosmos eine nach bestimmten Zahlenverhältnissen aufgebaute harmonische Einheit bildet – dass z.B. Diagonale und Seite des Quadrats, und vor allem auch des regelmässigen Fünfecks, inkommensurabel sind; moderner ausgedrückt, dass das Grössenverhältnis keine rationale Zahl darstellt. Tatsächlich führt eine entsprechend geeignete geometrische Konstruktion für die Wechselwegnahme zu einem endlosen Regress immer kleinerer selbstähnlicher Figuren; algebraisch interpretiert, zu einem nie abbrechenden Kettenbruch! Das Streben nach Harmonie erlitt so gesehen einen herben Rückschlag. Sind Figuren mit inkommensurablen Strecken vielleicht nur Ideale, also Hirngespinnste, aber nicht real existent? Aber Denkschmerz beiseite: Mit der Entdeckung der Inkommensurabilität sowie irrationaler Verhältnisse und Zahlen hält, zögerlich zunächst, die Unendlichkeit Einzug in die Mathematik. Eine gewaltige Bereicherung, wie die folgenden Jahrhunderte – um nicht zu sagen Jahrtausende – zeigen sollten!

# Die „Elemente“ von Euklid, dem griechischen Mathematiker, der um 300 v. Chr. an der grossen Bibliothek von Alexandria arbeitete

Einige Auszüge aus [www.armin-p-barth.ch/kolumnen/cafe/cafemathe/20080923\\_besprechung.php](http://www.armin-p-barth.ch/kolumnen/cafe/cafemathe/20080923_besprechung.php):

Ursprünglich war es eine Papyrusrolle, also ein Holzstab, um den gepresste und beschriebene Streifen aus dem unteren Stammteil der Papyruspflanze gewickelt waren. Es wurde erst viel später zu einem Buch, nachdem es als Abschrift von Abschriften von Abschriften 1800 Jahre wechselvolle Geschichte überlebt hatte und durch die Vervielfältigungen in Gutenbergs Maschine endgültig der Gefahr entronnen war, im Dunkel der Geschichte für immer verloren zu gehen. Dann aber wurde es zu einem Knüller, zu dem neben der Bibel auflagenstärksten Buch der Menschheitsgeschichte. Warum war gerade diesem Buch ein so überwältigender Erfolg beschieden, dass es auf der ganzen Welt gelesen wurde und wird?

Man weiss heute praktisch nichts über Euklid und die Umstände, die zu seinem Buch geführt haben. Aber es ist unbestritten, dass das Werk ein Geniestreich ist. Euklid fasste nicht nur die gesamte damals bekannte Mathematik zusammen und ordnete und ergänzte sie wesentlich, er verhalf ihr auch zu einer ganz neuen und revolutionären Form. Er machte in überwältigender Weise deutlich, dass die Mathematik eine Art Hochhaus ist, in dem jede Etage in ganz strenger Weise auf den darunterliegenden Etagen aufgebaut ist und die aber alle auf einem Fundament ruhen, das aus unbeweisbaren Prinzipien und Abmachungen besteht. Selbst die strengste und sicherste aller Wissenschaften ruht letztlich auf Vermutungen, die zwar plausibel, aber nicht beweisbar sind. Die Mathematik ist also nur ein (allerdings unbestritten erfolgreicher) Weg, die Welt zu sehen, und es ist unmöglich, in ihr ganz auf Elemente des Glaubens zu verzichten. Sie liefert unendlich viel Brauchbares, aber nichts von dem ist wahr an sich, alles ist immer nur wahr relativ zu den Abmachungen, die die Beschaffenheit des Fundamentes regeln.

Diese Einsicht ist so fundamental, dass sie das menschliche Denken ganz wesentlich geprägt hat; und wir verdanken sie zuallererst einem 2300 Jahre alten Buch, das es nur mit ganz viel Glück geschafft hat, in die heutige Zeit zu überdauern.

# Simultane Berechnung des ggT und kgV

Von [E.W. Dijkstra](#) (**EWD 316**, 1971) stammt folgender nette Algorithmus, der simultan ggT und kgV (von  $A, B > 0$ ) berechnet:

*Exercise. Prove that the following program will print in addition to the greatest common divisor of  $A$  and  $B$  the smallest common multiple of  $A$  and  $B$ . Hint: Follow the expression  $a \times c + b \times d$ .*

```
int a = A, b = B, c = B, d = 0, gcd, scm;
while (a != b)
{ while (a > b) {a = a - b; d = d + c};
  while (b > a) {b = b - a; c = c + d};
}
gcd = a; scm = c + d;
print(gcd); print(scm);
```

[www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD316.4.html](http://www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD316.4.html)

*The program and the correctness proof grow hand in hand. [E.W. Dijkstra in seiner Turing Award lecture 1972]*

---

*Dijkstra never wrote his articles using a computer. He preferred to rely on his typewriter and after 1972 on his Mont Blanc fountain pen. (His handwriting was so perfect and distinct that in the late 1980s Luca Cardelli designed a 'Dijkstra' font.) These articles were then distributed in an old-fashioned way: he sent copies to a few friends and associates who then served as the source nodes of the distribution centres. They are rarely longer than 15 pages and are consecutively numbered. The last one, EWD 1318, is from April 2002. Within computer science they are known as the [EWDs](#). [Krzysztof Apt]*

# EWD

Twenty-eight years

Twenty-eight years ago I wrote EWDO.

It was during a kind of scientific interregnum: my thesis and the basic software for the X1 had been completed and I could not start on the ALGOL compiler because ALGOL 60 had not been defined yet. I did all sorts of unrelated things. It was, for instance, the year that I published -at last!- my algorithms for the shortest path and the shortest spanning tree. It was also a time of explorations. At a given moment I was working on four or five different manuscripts. Of course, the pages of each manuscript were numbered; after I had mixed up pages between manuscripts for the second time, I decided that I had better end that confusion, and numbered the manuscripts as well: EWDO, EWD1, EWD3, EWD4. And that is how it started.

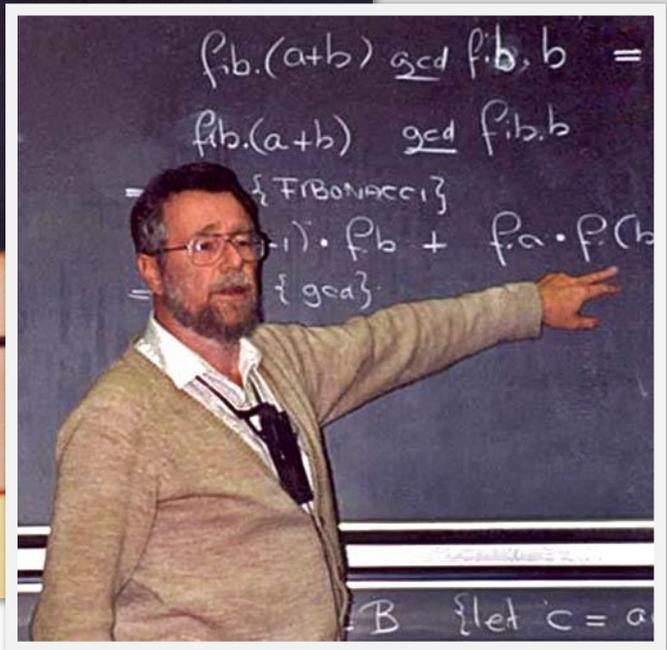
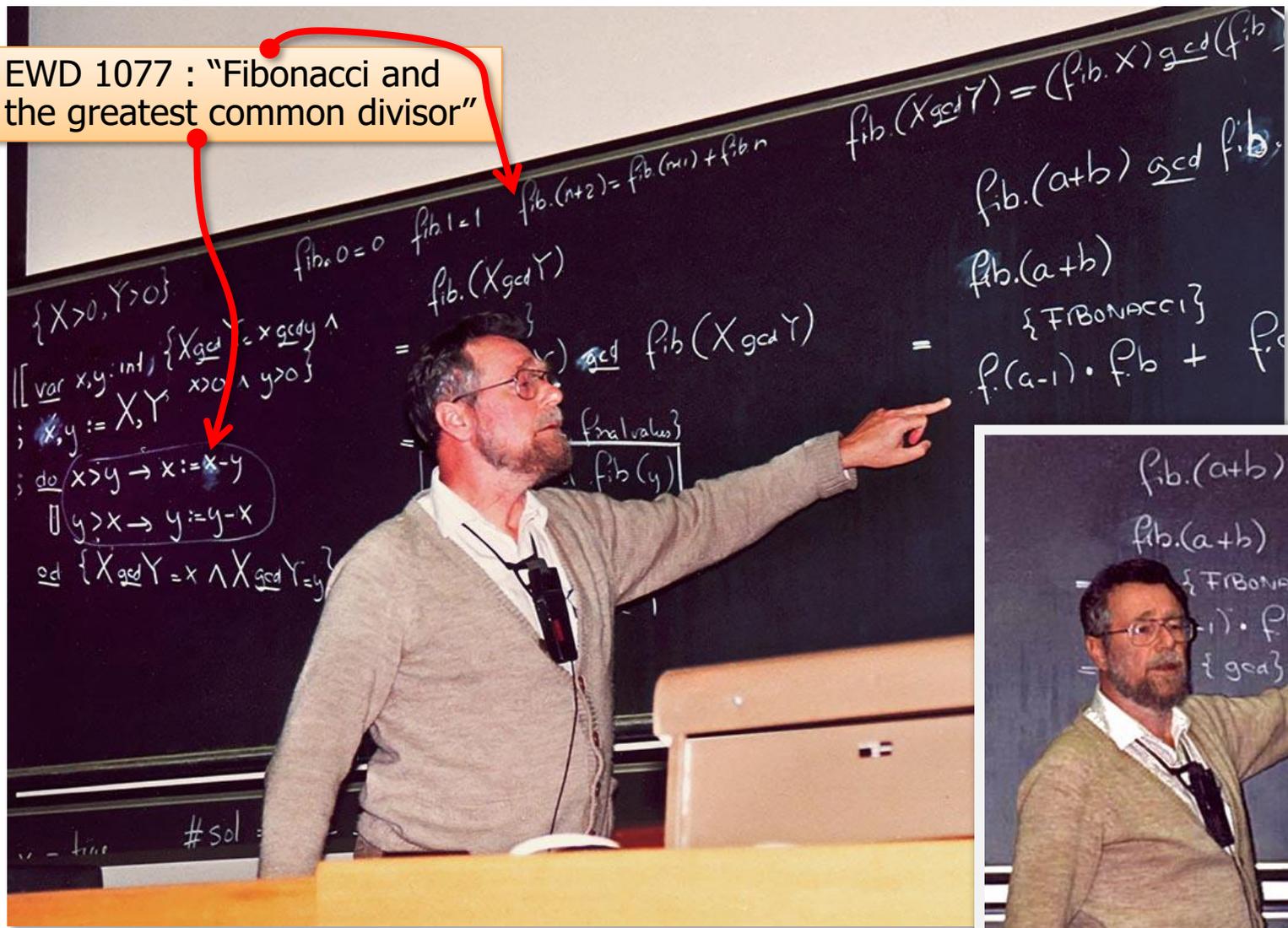
Dijkstra

32	33	34	^ 35	# 36	\$ 37	% 38	& 39	'
	!	"	#	\$	%	&	'	
40	( 41	) 42	* 43	+ 44	, 45	- 46	. 47	/
	(	)	*	+	,	-	.	/
48	0 49	1 50	2 51	3 52	4 53	5 54	6 55	7
	0	1	2	3	4	5	6	7
56	8 57	9 58	: 59	; 60	< 61	= 62	> 63	?
	8	9	:	;	<	=	>	?
64	@ 65	A 66	B 67	C 68	D 69	E 70	F 71	G
	@	A	B	C	D	E	F	G
72	H 73	I 74	J 75	K 76	L 77	M 78	N 79	O
	H	I	J	K	L	M	N	O
80	P 81	Q 82	R 83	S 84	T 85	U 86	V 87	W
	P	Q	R	S	T	U	V	W
88	X 89	Y 90	Z 91	[ 92	\ 93	] 94	^ 95	_
	X	Y	Z	[	\	]	^	_
96	` 97	a 98	b 99	c 100	d 101	e 102	f 103	g
	`	a	b	c	d	e	f	g
104	h 105	i 106	j 107	k 108	l 109	m 110	n 111	o
	h	i	j	k	l	m	n	o
112	p 113	q 114	r 115	s 116	t 117	u 118	v 119	w
	p	q	r	s	t	u	v	w
120	x 121	y 122	z 123	{ 124	125	} 126	~ 127	
	x	y	z	{		}	~	□

Ein handschriftlicher Text von Dijkstra ↑ und der Dijkstra-Font von Luca Cardelli →

# Dijkstra im März 1994 an der ETH Zürich

EWD 1077 : "Fibonacci and the greatest common divisor"



[https://en.wikipedia.org/wiki/File:Edsger\\_Dijkstra\\_1994.jpg](https://en.wikipedia.org/wiki/File:Edsger_Dijkstra_1994.jpg), Andreas F. Borchert

$$(0) \quad f.0 = 0, \quad f.1 = 1, \quad f.(n+2) = f.(n+1) + f.n$$

Then,  $f$  application distributes over gcd, i.e.

$$(1) \quad f.(X \text{ gcd } Y) = f.X \text{ gcd } f.Y$$

EWD 1077 : "Fibonacci and the greatest common divisor"

Our interest is not in the above theorem, nor in its proofs. We wish to explore how we could design a proof for it.

We know the gcd for positive operands as the outcome of Euclid's Algorithm:

```
(2)  x,y := X,Y
      ; do x > y → x := x - y
        [] y > x → y := y - x
      od {x = X gcd Y ∧ y = X gcd Y}
```

and this knowledge raises the question of whether we can prove (1) by a properly chosen invariant for program (2).

Dijkstra war ein starker Verfechter des „zero-based numbering“; in EWD 831 "Why numbering should start at zero" (1982) begründet er dies. ("The moral of the story is that we better regard — after all those centuries! — zero as a most natural number.")  
"His reports [...] begin with page 0, and when he was writing about  $n$  processes, they were invariably numbered 0, ...,  $n-1$ " [K. Apt]. Angeblich hatte der Springer-Verlag extra für die Dijkstra-Texte seine Latex-Makros entsprechend verändert.  
(Und wieso gibt es in Java eigentlich ein 0. Array-Element?)

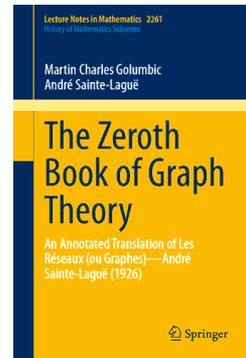
Mehr zu Dijkstra später

# Zero-based numbering?

Auf einer Reise geriet der Mathematiker **Waclaw Sierpinski** plötzlich in Panik, weil er ein Gepäckstück vergessen zu haben glaubte. „Aber Liebling“ beruhigte ihn seine Frau, „alle sechs Koffer sind da.“ „Das kann nicht sein“, entgegnete Sierpinski, „ich habe zweimal nachgezählt: null, eins, zwei, drei, vier, fünf.“  
[Nach: John H. Conway, Richard Guy: *The Book of Numbers*]

Auch der  
„Eimer“  
(Ein-Henkler)

- Beim **Abzählen** zählen wir 1, 2, 3,...; fangen also **bei 1 (dem ersten)** an zu zählen
  - Entsprechend ist z.B. Folgendes praktisch nur bei bzw. mit „1“ (oder grösseren Zahlen) sinnvoll, nicht bei „0“: Gleis 1, Paragraph 1, Phase 1, Version 1, Einbettzimmer, Einfamilienhaus, Einhorn, Einmalhandtuch, Einmaster, Einpersonenhaushalt, Eintagsfliege, Eintopfgericht, Einwegbehälter, eindeutig, eineiige Zwillinge, einfarbig, eingleisig, einseitig, einsilbig
- Das **Erste** spielt oft eine besondere Rolle, daher gibt es viele entsprechende Wortbildungen, wo „0“ typischerweise sinnlos ist:
  - Erstauflage, Erstbesitz, Erstbezug, erstgeboren, erstgenannt, Erstimpfung, erstklassig, Erstschlag, Erstsemester, Erstveröffentlichung, erste Hilfe, König Ludwig I.
- Die **Null** kommt bei Wortzusammensetzungen meist nur dann ins Spiel, wenn man die **Abwesenheit einer Eigenschaft** hervorheben will (oder bei manchen Aufzählungen, wo es vor der 1 tatsächlich etwas gibt):
  - Nullwachstum, Nullrunde, Nulldiät, Nullstellung, Nullnummer, Nulltarif, Nullsummenspiel, Null-Bock-Generation, Windstärke 0, Nullgradgrenze, Nullmeridian, 0. Hauptsatz der Thermodynamik
- Irritationen hinsichtlich **0 und / oder 1** gibt es allerdings manchmal doch:
  - Ist 0 die erste natürliche (Kardinal)zahl? Die „Stunde 0“ die erste Stunde des Tages?
  - „Beim nächsten Ton ist es: 0 Uhr, 0 Minuten und 10 Sekunden“ (automatische Zeitansage)
  - Ist das Erdgeschoss der 0. oder 1. Stock? (Der Liftknopf dafür ist oft „E“!)
  - Auf das Jahr 1 v. Chr. folgt 1 n. Chr. – die Zeitnorm ISO 8601 kennt dagegen ein Jahr 0.
  - Legendär sind die vielen „**Um-eins-daneben-Fehler**“, z.B. bei **Max Frisch**. Der Schriftsteller hatte bekanntlich Architektur studiert, und zwar an der ETH Zürich – und dennoch: „Einmal auf dem Bau muss ich erfahren, dass eine Treppe, die ich gerechnet und gezeichnet habe, nicht auf dem oberen Podest ankommt; es fehlt eine Tritthöhe, während die Länge stimmt.“ [Montauk]



Ist ein  
Schalt-  
jahr!

# Who's on first? Zero or one?

Danny Cohen (1981): On holy wars and a plea for peace. *Computer*, 14(10), 48-54

People start counting from the number one. The very word *first* is abbreviated as 1st, which indicates one. This, however, is a very modern notation. The older concepts do not necessarily support this relationship. In English and French the word *first* is not derived from the word *one*, but from an old word for *prince*, which means foremost. Similarly, the English word *second* is not derived from the number two but from an old word which means "to follow." Obviously, there is a close relation between third and three, fourth and four, and so on. [...]

For a very long time, people have counted from one, not from zero. As a matter of fact, the inclusion of zero as a full-fledged member of the set of all numbers is a relatively modern concept, even though it is one of the most important numbers mathematically.

A nice mathematical theorem states that for any basis  $b$  the first  $b^N$  positive integers are represented by exactly  $N$  digits (leading zeros included). This is true if and only if the count starts with zero (hence, 0 through  $b^N - 1$ ), not with one (for 1 through  $b^N$ ).

This theorem is the basis of computer memory addressing. Typically,  $2^N$  cells are addressed by an  $N$ -bit addressing scheme. A count starting from one rather than zero would cause the loss of either one memory cell or an additional address line. Since either price is too expensive, computer engineers agree to use the mathematical notation that starts with 0. Good for them!

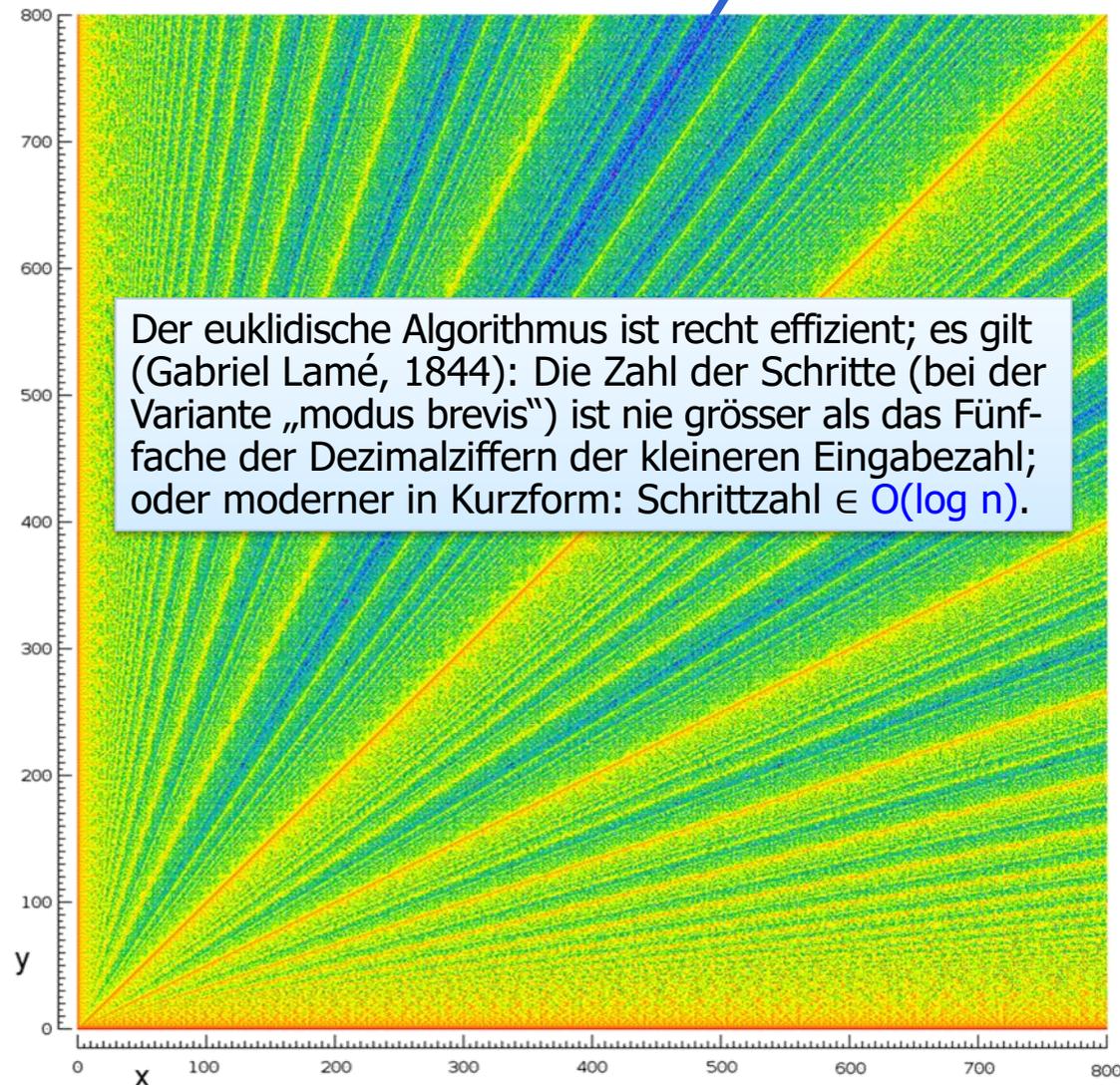
This is probably the reason why all memories start at address 0, even those of systems that count bits from bit 1 up. The designers of the IBM 1401 were probably ashamed to have address 0. They hid it from the users and pretended that the memory starts at address 1.

Communication engineers, like most people, start counting from one. They never have to suffer the loss of a memory cell, for example. Therefore, they happily count one-to-eight, not zero-to-seven, as computer people do.



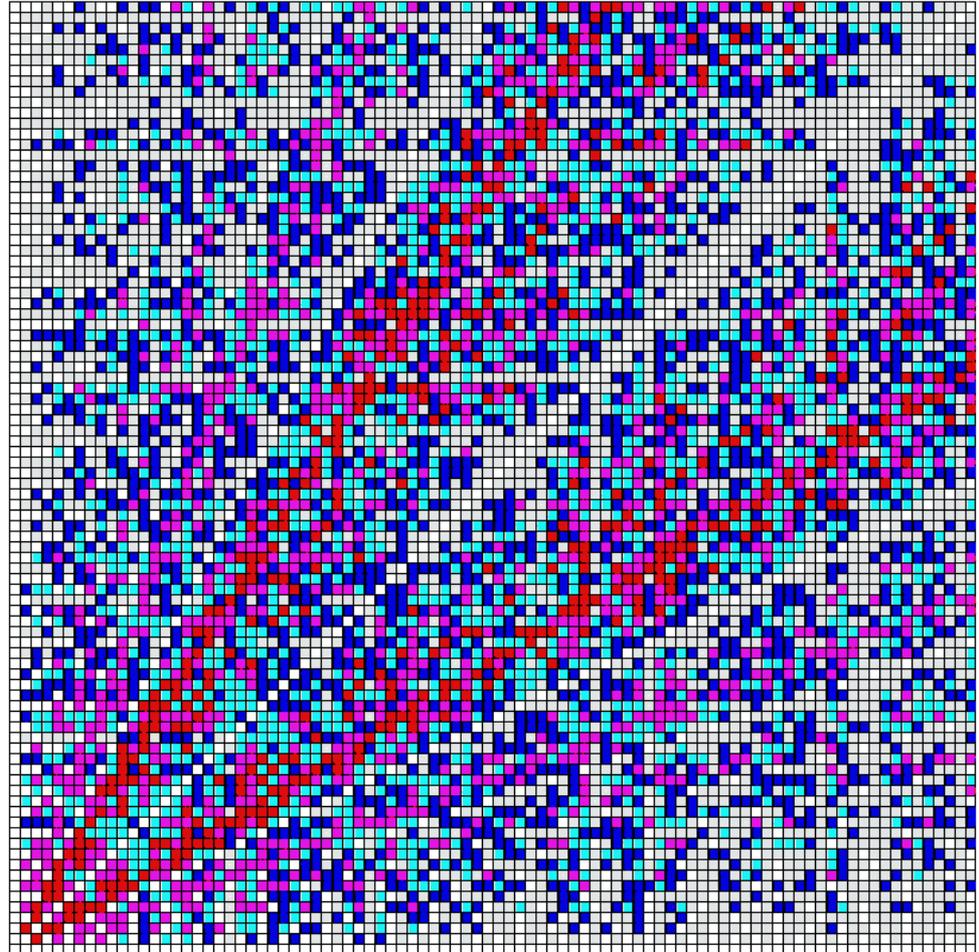
# Euklid und Fibonacci

Der **euklidische Algorithmus** und die **Fibonacci-Folge** 1, 1, 2, 3, 5, 8, 13, 21,... [Fib(i) = Fib(i-1) + Fib(i-2)] hängen auf eine interessante Weise zusammen: Bei der Laufzeit stellt ein „schlimmer Eingabefall“ zwei aufeinanderfolgende Fibonacci-Zahlen dar; dann ergibt sich als Rest stets die nächstkleinere Fibonacci-Zahl. Das Bild zeigt die Anzahl der Schritte bei Eingabe von  $x, y$ . Rote Punkte bedeuten wenige Schritte; gelbe, grüne und blaue Punkte relativ mehr Schritte. Die dunkelblaueste Linie hat die Steigung  $\Phi$ , wobei  $\Phi = (1+\sqrt{5})/2 \approx 1.618034$  die „Goldene Zahl“ des Goldenen Schnitts ist. Schon für kleine  $i$  gilt recht präzise  $\text{Fib}(i+1) / \text{Fib}(i) \approx \Phi$ . Ausserdem ist  $\Phi^n$  „fast“ eine natürliche Zahl, und  $\Phi^n/\sqrt{5}$ , gerundet auf ganze Zahlen, ist stets  $\text{Fib}(n)$ .



# Euklid und Fibonacci (2)

Hier das Anfangsstück nochmals in einer detaillierteren Auflösung. Karos von  $(x,y)$ -Paaren mit maximaler Schrittzahl sind hier rot gefärbt, solche mit der um 1, 2 oder 3 gegenüber der Höchstzahl verminderten Schrittzahl magenta, cyan bzw. blau. Alle Karos, die Paare mit noch kleinerer Schrittzahl repräsentieren, erscheinen grau – bis auf diejenigen, die nur einen einzigen Schritt erfordern, sie wurden weiss gefärbt. Man erkennt die Symmetrie zur Diagonalen sowie die beiden roten „Worst-Case-Linien“ mit der Steigung  $1.6180339887498948482\dots$  bzw.  $0.6180339887498948482\dots$



<http://horstth.de/wp-content/uploads/2013/10/EuklidGrafikNeuD.jpg>

# Euklid und Fibonacci

Gabriel Lamé: *Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers.* Comptes rendus hebdomadaires des séances de l'Académie des sciences, 19 (1844), 867-870

« Dans les traités d'Arithmétique, on se contente de dire que le nombre des divisions à effectuer, dans la recherche du plus grand commun diviseur entre deux entiers, *ne pourra pas surpasser la moitié du plus petit.* Cette limite, qui peut être dépassée si les nombres sont petits, s'éloigne outre mesure quand ils ont plusieurs chiffres. L'exagération est alors semblable à celle qui assignerait la moitié d'un nombre comme la limite de son logarithme; l'analogie devient évidente quand on connaît le théorème suivant :

» THÉORÈME. *Le nombre des divisions à effectuer, pour trouver le plus grand commun diviseur entre deux entiers A, et  $B < A$ , est toujours moindre que cinq fois le nombre des chiffres de B.*

## COMPTE RENDU DES SÉANCES DE L'ACADÉMIE DES SCIENCES.

SÉANCE DU LUNDI 28 OCTOBRE 1844.

PRÉSIDENTE DE M. CHARLES DUPIN.

MÉMOIRES ET COMMUNICATIONS

DES MEMBRES ET DES CORRESPONDANTS DE L'ACADÉMIE.

ARITHMÉTIQUE. — *Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers; par M. LAMÉ.*

par  $t^u$ ,  $t^v$ ,  $t^r$ ,  $t^s$ , ... les termes qui suivent  $t^u$ , on aura les inéga-

$$2,5 \cdot 10^k < t^u < 5 \cdot 10^k,$$

$$4 \cdot 10^k < t^v < 8 \cdot 10^k,$$

$$6,5 \cdot 10^k < t^r < 13 \cdot 10^k,$$

$$10,5 \cdot 10^k < t^s < 21 \cdot 10^k.$$

Erst später wurde diese Zahlenfolge (von Édouard Lucas) *Fibonacci* genannt.

ne peut avoir plus de  $(k + 1)$  chiffres, et  $t^s$  moins de  $(k + 2)$  chif-

Es handelt sich um eine frühe mathematische Abhandlung, in der die „Laufzeit“ eines nicht-trivialen Algorithmus analysiert wird – die Veröffentlichung von Lamé stellt insofern einen Vorläufer aus dem Gebiet der **Komplexitätstheorie** dar, das erst 110 Jahre später begründet wurde (am Ende des Vorlesungskapitels „Komplexität von Algorithmen“ kommen wir darauf zurück).

# Komplexitätsanalyse von Euklids Algorithmus

Obwohl üblicherweise Lamé als Begründer der Komplexitätsanalyse des euklidischen Algorithmus angesehen wird, kommt diese Ehre stattdessen wohl [Pierre Joseph Étienne Finck](#) zu, der 1797 in Lauterburg, dem nordöstlichsten Ort des Elsass, geboren wurde, 1829 in Strassburg promovierte und später dort auch Professor für Mathematik wurde. Er veröffentlichte 1841, drei Jahre vor Lamé, ein Buch *Traité élémentaire d'arithmétique à l'usage des candidats aux écoles spéciales*, in dem er den euklidischen Algorithmus analysiert:

Chaque reste est moindre que la moitié du dividende: car si le diviseur est égal à la moitié du dividende, le reste est 0; si le diviseur est plus grand que cette moitié, le quotient est 1, et le reste, devant faire avec le diviseur une somme égale au dividende, sera moindre que cette même moitié; enfin, si le diviseur est plus petit que la moitié du dividende, le reste le sera aussi comme étant moindre que le diviseur. De là il suit que si l'on nomme  $A$  et  $B$  les deux nombres dont on cherche le p. g. c. d.,  $A$  étant  $> B$ , les restes successifs sont respectivement moindres que  $A/2$ ,  $B/2$ ,  $A/4$ ,  $B/4$ ,  $A/8$ ,  $B/8$ , . . . ,  $A/2^n$ ,  $B/2^n$ ; l'opération se terminera donc au plus tard lorsque  $B/2^n \leq 2$ , ou  $B < 2^{n+1}$ ; car alors le reste moindre que  $B/2^n$  sera au plus = 1, qui est le dernier diviseur; mais alors le nombre des divisions faites serait  $2n + 1$ . Ainsi, cherchez l'exposant de la plus petite puissance de 2, qui dépasse  $B$ , diminuez-le d'une unité, doublez le reste et ajoutez 1, ce sera une limite du nombre des opérations qu'il y aura à faire pour trouver le p. g. c. d. de  $A$  et  $B$ . Soit, pour exemple, les nombres 89 et 55: la plus petite puissance de 2, qui dépasse 55 est 64 ou  $2^6$ ; donc  $n + 1 = 6$ ,  $n = 5$  et  $2n + 1 = 11$ ; et, en effet, dans ce cas, il faut neuf opérations.

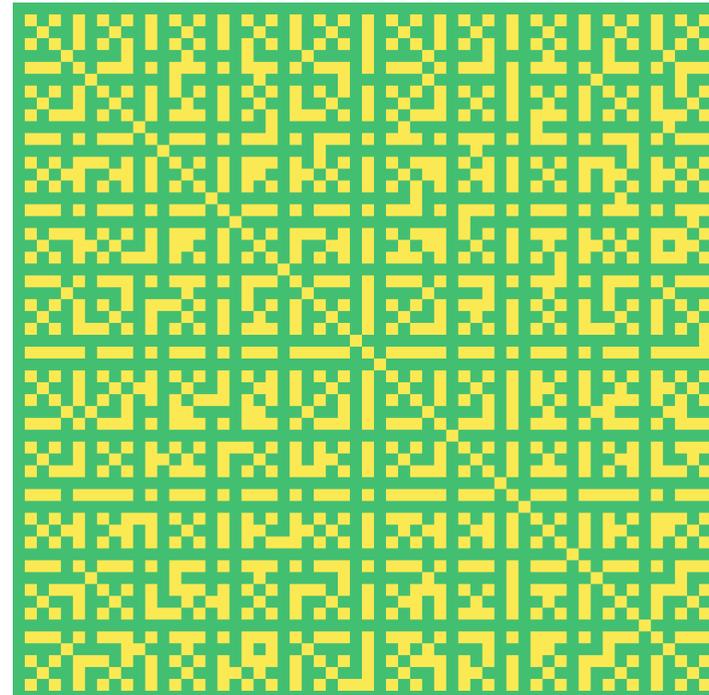
Finck fand also heraus, dass bei  $\text{ggT}(m,n)$  die Zahl der Schritte  $\leq 1 + 2 \log_2(n)$  beträgt; diese Schranke ist nur um einen Faktor von ca. 1.33 schlechter als die Schranke von Lamé („das Fünffache der Zahl der Ziffern von  $n$ “). Wieso Finck von Lamé nicht erwähnt wird, bleibt unklar – möglicherweise ist Lamé das frühere Resultat seines Landsmannes Finck tatsächlich entgangen.

# ggT(a,b) = 1

Spielt man mit dem Euklidischen Algorithmus, dann erhält man ziemlich oft das Ergebnis 1. Tatsächlich beträgt die Wahrscheinlichkeit dafür, dass zwei zufällig gewählte (grössere) nat. Zahlen zueinander **teilerfremd** sind,  $6/\pi^2 = 0.60792710\dots$

*Achtung, das ist salopp formuliert, denn es gibt keine uniforme Wahrscheinlichkeitsverteilung über den nat. Zahlen!*

a \ b	1	2	3	4	5	6	7	8	9	10
1	Green	Green	Green	Green	Green	Green	Green	Green	Green	Green
2	Green	Yellow	Green	Yellow	Green	Yellow	Green	Yellow	Green	Yellow
3	Green	Green	Yellow	Green	Green	Yellow	Green	Yellow	Green	Yellow
4	Green	Yellow	Green	Yellow	Green	Green	Yellow	Green	Yellow	Green
5	Green	Green	Green	Green	Yellow	Green	Green	Green	Green	Yellow
6	Green	Yellow	Yellow	Yellow	Green	Yellow	Green	Yellow	Green	Yellow
7	Green	Green	Green	Green	Green	Green	Yellow	Green	Green	Green
8	Green	Yellow	Green	Yellow	Green	Yellow	Green	Yellow	Green	Yellow
9	Green	Green	Yellow	Green	Green	Yellow	Green	Yellow	Green	Yellow
10	Green	Yellow	Green	Yellow	Yellow	Yellow	Green	Yellow	Green	Yellow



Ein Bildanalysetool liefert einen **Grünanteil** von **62.3%**. Es gibt 2171 grüne „Matrixpunkte“ sowie 1310 gelbe; der Anteil der grünen beträgt also **0.6236713...** Bei einer Matrixgrösse 10000 x 10000 wird **0.60794971** erreicht, und bei 100000 x 100000 beträgt der Anteil **0.6079301507**. Da ist man schon nah an  $6/\pi^2$ .

← Nein, das ist kein QR-Code!

Schöpft man die beiden Zahlen aus einem Bereich 1,2,...,n für ein kleineres n, dann liegt der Wert etwas über  $6/\pi^2$ ; in der linken Matrix beträgt der Anteil der grünen (teilerfremden) „Punkte“  $63/100 = 0.63$ . Bei der rechten **59 x 59-Matrix** sind die fast lückenlosen grünen Primzahl-Linien und die gelbe Diagonale für  $ggT(z,z)$  gut erkennbar.

# Übung: ggT-Algorithmus mit Schieberegistern

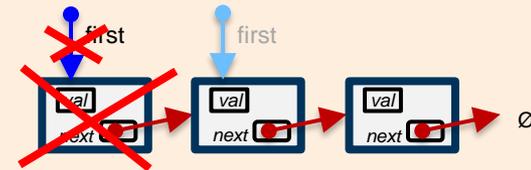
Rechenwerke bzw. ALUs in Computern können typischerweise Zahlen sehr schnell verdoppeln und halbieren, indem Schieberegister eine Stelle nach links oder rechts geschoben werden. Man mache sich die folgenden Identitäten klar und wende sie an, um damit einen effizienten Algorithmus zur Berechnung des ggT zu realisieren:

- $\text{ggT}(2a, 2b) = 2 \text{ggT}(a, b)$
- $\text{ggT}(2a, 2b+1) = \text{ggT}(a, 2b+1)$
- $\text{ggT}(2a+1, 2b+1) = \text{ggT}(2a-2b, 2b+1)$
- $\text{ggT}(a, b) = \text{ggT}(b, a)$

# Resümee des Kapitels

## ■ Pakete in Java

- `packet`, `import`, `public`, `private`
- Beispiel: Service-Paket „verkettete Liste“
- Beispiel-Nutzung des Service-Pakets: Stack als verkettete Liste
- Service-Paket „Bruchrechnung“
- Euklidischer Algorithmus (ggT)



```
import bruchPak; ...  
a = b.plus (new Bruch(1,8));
```