

Übung zu
Algorithmen und Datenstrukturen (für ET/IT)
Sommersemester 2018

Mai Bui

Computer Aided Medical Procedures
Technische Universität München



Administratives

- ▶ Zentralübung (Mittwoch, 09:45 – 11:15)
 - ▶ Programmierbeispiele zur Vorlesung
 - ▶ Besprechung allgemeiner Fragen zu den Übungsblättern
 - ▶ Probeklausur
- ▶ Tutorübungen
 - ▶ Besprechung der Übungsblätter
 - ▶ Besprechung der Probeklausur
- ▶ Kontakt-E-Mail: algods@mailnavab.in.tum.de
- ▶ Sprechstunde (Termin per E-Mail)
- ▶ Moodle
 - ▶ <https://www.moodle.tum.de/course/view.php?id=38846>
 - ▶ Diskussionsforum & Bekanntmachungen

<http://campar.in.tum.de/Chair/TeachingSs18AuD>

Warum C **und** C++?

- ▶ C ist die Grundlage, aber einige Konzepte fehlen.
- ▶ C++ ist eine Erweiterung von C, grösstenteils kompatibel.
- ▶ Aus C/C++ Quellcode kann sehr effizienter Maschinencode erzeugt werden.

C/C++ Compiler

- ▶ *MinGW* für Windows
 - ▶ <http://www.mingw.org/>
- ▶ GCC für Mac OS X über Xcode
 - ▶ <https://itunes.apple.com/de/app/xcode/id497799835>
- ▶ GCC für GNU/Linux
Meist schon vorinstalliert
 - ▶ <https://gcc.gnu.org/install/>

Beispiel Tutorial:

- ▶ https://www.tutorialspoint.com/cprogramming/c_environment_setup.htm

Entwicklungsumgebungen

- ▶ *Microsoft Visual Studio* für Windows
 - ▶ Express-Version über
<https://www.visualstudio.com/vs/express/>
 - ▶ Vollversion über DreamSparks: <https://www.studisoft.de>
- ▶ *Xcode* für Mac OS X
 - ▶ <https://itunes.apple.com/de/app/xcode/id497799835>
- ▶ *CLion* für Windows, Mac OS X, GNU/Linux
Vollversion für Studenten
 - ▶ <https://www.jetbrains.com/student/>
- ▶ *Eclipse* für Windows, Mac OS X, GNU/Linux
 - ▶ <http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/oxygen3>

Heute: Online C/C++ Compiler

- ▶ `https://www.onlinegdb.com/online_c_compiler`

Euklidischer Algorithmus

„Wenn CD aber AB nicht misst, und man nimmt bei AB, CD abwechselnd immer das kleinere vom größeren weg, dann muss (schließlich) eine Zahl übrig bleiben, die die vorangehende misst.“

aus *Euklid: Die Elemente, Buch VII (Clemens Thaer)*

- ▶ Vorschrift zur **Berechnung** des *größten gemeinsamen Teilers*
- ▶ Keine **Definition** des ggT im mathematischen Sinn
- ▶ Verschiedene Algorithmen mit gleichem Ziel (↔ Sortierung!)

Größter gemeinsamer Teiler

Wie lautet der größte gemeinsame Teiler von 11 und 4, $ggT(11,4)$?

Euklidischer Algorithmus - Originalformulierung

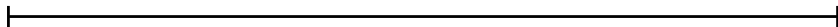
Input: natürliche Zahlen AB , CD

Output: größte gemeinsame Teiler von AB , CD

1. Setze $m := AB$ und $n := CD$
2. Falls $m < n$, vertausche m und n
3. Berechne $r := m - n$
4. Setze $m := n$ und $n := r$
5. Falls $r > 0$, gehe zu Schritt 2.
6. Der Output ist m

Ablauf nach Originalformulierung

$$m = AB = 11$$



$$n = CD = 4$$

0 Rechenschritte

Ablauf nach Originalformulierung

$$r = 7 = 11 - n$$

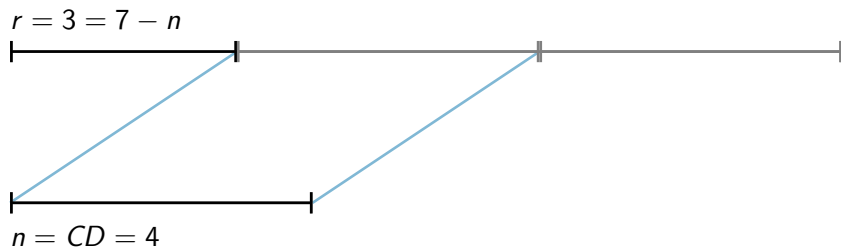


$$n = CD = 4$$



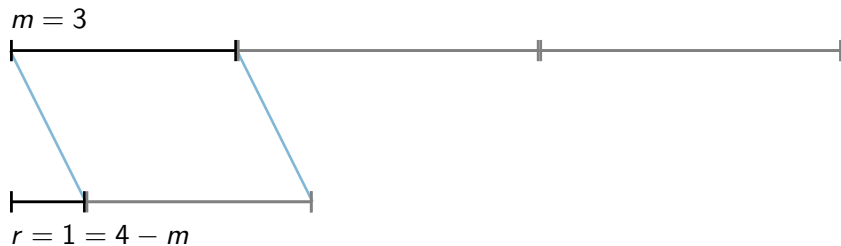
1 Rechenschritt

Ablauf nach Originalformulierung



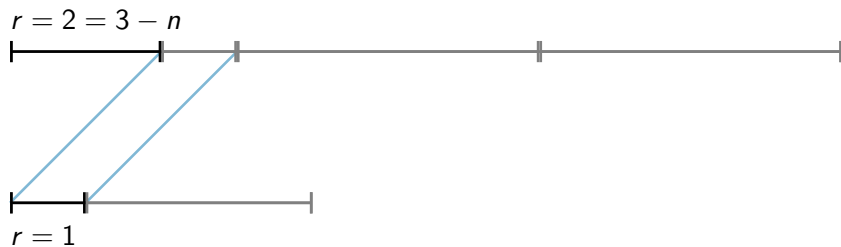
2 Rechenschritte

Ablauf nach Originalformulierung



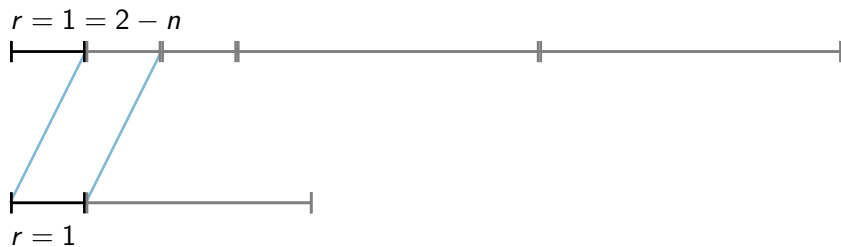
3 Rechenschritte

Ablauf nach Originalformulierung



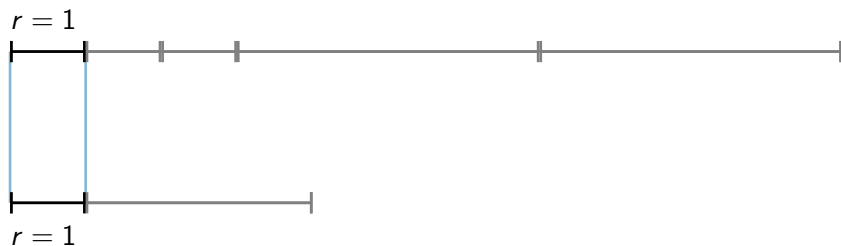
4 Rechenschritte

Ablauf nach Originalformulierung



5 Rechenschritte

Ablauf nach Originalformulierung



$$\Rightarrow ggT(11, 4) = 1$$

11 ist eine Primzahl!

5 Rechenschritte

Algorithmus

- ▶ präzise/eindeutige, endliche Beschreibung
- ▶ Bausteine:
 - ▶ Elementarer Verarbeitungsschritt
 - ▶ Sequenz (Abfolge elementarer Schritte)
 - ▶ Bedingter Verarbeitungsschritt
 - ▶ Wiederholung

Variablen

- ▶ Speicherung von Werten mittels Typ-behafteter, eindeutig benannter Variablen
 - ▶ `int` für ganze Zahlen (Integer)
 - ▶ `double` für Fließkommazahlen (Double Precision Floating Point)
 - ▶ `char` für Zeichen (Character)
 - ▶ ...
- ▶ Zuweisung über `=`
- ▶ Kurze Schreibweise für Updates
 - ▶ `i *= 4`; statt `i = i * 4`; (ebenso `+=`, `-=`, `*=`, `/=`, ...)
 - ▶ `i++`; statt `i = i + 1`; (ebenso `--`)

```
int i = 23;  
int j = 2 * (i-2);           // Klammerung!  
char s = 'a';
```

Elementarer Verarbeitungsschritt

- ▶ Mathematische Operatoren $+$, $-$, $*$, $/$, \dots
- ▶ Zuweisungen $=$, $+=$, $-=$, $*=$, $/=$, \dots
- ▶ \dots

Sequenz

```
int i = 10;  
i = i+2; // Wert von i ist 12
```

Bedingter Verarbeitungsschritt

- ▶ `if (b) {...}`
- ▶ `if (b) {...} else {...}`
- ▶ `if (b) {...} else if (...) {...} else {...}`
- ▶ Klammern `{...}` zur Sicherheit immer schreiben
- ▶ `b` steht für einen logischen Ausdruck oder eine logische (boolsche) Variable:

Logische Ausdrücke

- ▶ Logischer Typ (`bool`) mit Werten `true` (wahr, 1) und `false` (unwahr, 0)
- ▶ Vergleichs-Operatoren: `==` (gleich), `!=` (ungleich), `<`, `<=`, ...
- ▶ Verknüpfungen: `&&` (AND; beide wahr), `||` (OR; mindestens eines von beiden wahr), ...
- ▶ Negation: `!a` (NOT)

Wiederholung

Schleifen für Wiederholungen

- ▶ `while (b) {...}` (Test vor jeder Iteration)
- ▶ `do {...} while (b);` (Test nach jeder Iteration)
- ▶ `for i = 1...100 {...}` (Gewisse Anzahl an Iterationen)

Pseudocode

- ▶ Informelle Veranschaulichung eines Algorithmus
- ▶ Identifizierung der elementaren Bausteine nicht offensichtlich

Input: Natürliche Zahlen a, b

Output: $\text{ggT}(a, b)$

euklid(a,b):

- ▶ Falls $a = 0$, liefere b zurück
- ▶ Solange $b > 0$ wiederhole
 - ▶ Falls $a > b$ setze $a = a - b$
 - ▶ sonst setze $b = b - a$
- ▶ Liefere a zurück

Beispiel Programmiersprache: C

- ▶ Kompilierte Sprache, sehr effizient

```
int euklid(int a, int b)
{
    if (a==0)
        return b;
    while (b>0)
    {
        if (a>b)
            a = a - b;
        else
            b = b - a;
    }
    return a;
}
```

Konvention der Vorlesung

- ▶ Pseudocode + C Syntax

Input: Natürliche Zahlen a, b

Output: $\text{ggT}(a, b)$

euklid(a, b):

```
{  
    if (a==0)  
        return b;  
    while (b>0)  
    {  
        if (a>b)  
            a = a - b;  
        else  
            b = b - a;  
    }  
    return a;  
}
```

Konvention der Vorlesung – Funktionsdefinition

Input: Natürliche Zahlen a, b //Einsprungspunkt

Output: $\text{ggT}(a, b)$ //Aus-/Rückgabe

euklid(a, b): //Funktionsname

```
{  
    if (a==0)  
        return b;  
    while (b>0)  
    {  
        if (a>b)  
            a = a - b;  
        else  
            b = b - a;  
    }  
    return a;  
}
```

- ▶ Aufruf mittels $\text{euklid}(a, b)$ (z.B. $\text{euklid}(6, 3)$)

Konvention der Vorlesung – Identifikation der Bausteine

Input: Natürliche Zahlen a, b

Output: $\text{ggT}(a, b)$

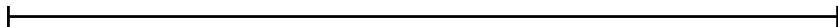
`euklid(a, b):`

```
{ //Beginn einer Sequenz
  if (a==0) //Bedingter Verarbeitungsschritt
    return b; //Elementarer Verarbeitungsschritt
  while (b>0) //Wiederholung
  { //Beginn einer Sequenz
    if (a>b) //Bedingter Verarbeitungsschritt
      a=a-b; //Elementarer Verarbeitungsschritt
    else //Bedingter Verarbeitungsschritt
      b=b-a; //Elementarer Verarbeitungsschritt
  } //Ende einer Sequenz
  return a; //Elementarer Verarbeitungsschritt
} //Ende einer Sequenz
```

 Code

Alternativer Ablauf nach Originalformulierung

$$m = AB = 11$$

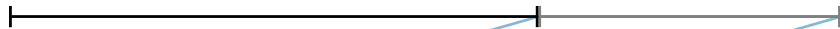


$$n = CD = 4$$

0 Rechenschritte

Alternativer Ablauf nach Originalformulierung

$$r = 7 = 11 - n$$

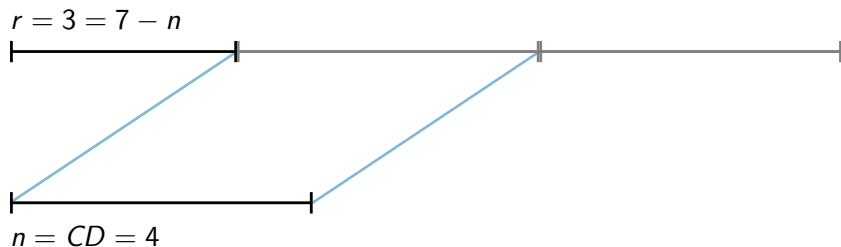


$$n = CD = 4$$



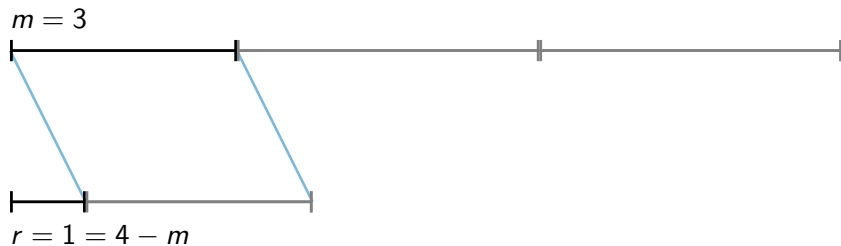
1 Rechenschritt

Alternativer Ablauf nach Originalformulierung



2 Rechenschritte

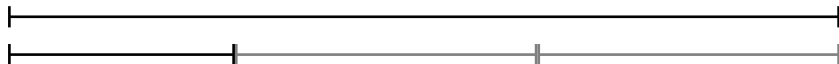
Alternativer Ablauf nach Originalformulierung



3 Rechenschritte ...

Beobachtung

$$AB = 11 = 2 \cdot CD + 3$$

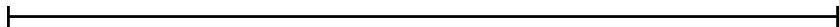


$$\begin{array}{r} AB : CD = 11 : 4 = 2 \\ \quad \quad \quad - \quad 8 \\ \hline \quad \quad \quad \quad 3 \\ \hline \hline \end{array}$$

- ▶ Die *Modulo-Berechnung* liefert den Rest nach Teilung.
- ▶ Wir schreiben $11 \bmod 4 = 3 \dots$
- ▶ ... und programmieren `int rest = 11 % 4;`

Ablauf mit Modulo

$$m = AB = 11$$



$$n = CD = 4$$

0 Rechenschritte

Ablauf mit Modulo

$$r = 3 = 11 \bmod n$$



$$n = CD = 4$$

1 Rechenschritt

Ablauf mit Modulo

$$m = 3$$



$$r = 1 = 4 \bmod m$$

2 Rechenschritte

Ablauf mit Modulo

$$r = 0 = 3 \bmod n \implies m = n = 1$$



$$\Rightarrow \text{ggT}(11, 4) = 1$$

Abbruch bei Rest 0!

3 Rechenschritte

Caveat

- ▶ Die Zählung von Rechenschritten ist stark vereinfachend!
- ▶ Tatsächlich dauert die Berechnung einer Teilung (bzw. einer Modulo-Operation) erheblich länger als eine Differenz!

Implementierung via Modulo

Input: Natürliche Zahlen a, b

Output: $\text{ggT}(a, b)$

`euklid_modulo(a, b):`

```
{  
    while (b != 0)  
    {  
        int rest = a%b;  
        a = b;  
        b = rest;  
    }  
    return a;  
}
```


 Code

Rekursion

- ▶ Funktionen rufen sich selbst wieder auf, nachdem die Parameter geändert wurden.
- ▶ Abbruchbedingungen beenden diese Kette von Aufrufen.
- ▶ Beispiel: Addition der Zahlen m **bis** n

Iterative Variante

Input: Natürliche Zahlen m, n

Output: Summe der Zahlen m bis n

`summe_iterativ(m,n):`

```
{  
    int wert = 0;  
    for (int i = m; i <= n; ++i)  
        wert += i;  
    return wert;  
}
```

Rekursive Variante

Input: Natürliche Zahlen m, n

Output: Summe der Zahlen m bis n

`summe_rekursiv(m,n):`

```
{  
    if (m > n)  
        return 0;  
    else  
        return m + summe_rekursiv(m+1, n);  
}
```

Aufrufkette

- ▶ `summe_rekursiv(2, 4)`
- ▶ `2 + summe_rekursiv(3, 4)`
- ▶ `2 + 3 + summe_rekursiv(4, 4)`
- ▶ `2 + 3 + 4 + summe_rekursiv(5, 4)`
- ▶ `2 + 3 + 4 + 0`
- ▶ `2 + 3 + 4`
- ▶ `2 + 7`
- ▶ `9`

ggT mit Rekursion

- ▶ Mittels dreier Eigenschaften läßt sich der ggT rekursiv berechnen:

$$\text{ggT}(ab, cd) = \text{ggT}(cd, ab) \quad (1)$$

$$\text{ggT}(ab, cd) = \text{ggT}(ab, cd \bmod ab) \quad (2)$$

$$\text{ggT}(ab, cd) = \text{ggT}(ab, cd + \lambda \cdot ab) \quad (3)$$

Implementierung via Rekursion

Input: Natürliche Zahlen a, b

Output: $\text{ggT}(a, b)$

`euklid_rekursiv(a, b):`

```
{  
    if (b==0)  
        return a;  
    else if (a==0)  
        return b;  
    else if (a>b)  
        return euklid_rekursiv(a-b, b);  
    else  
        return euklid_rekursiv(a, b-a);  
}
```

Implementierung via Rekursion und Modulo

Input: Natürliche Zahlen a, b

Output: $\text{ggT}(a, b)$

```
euklid_rekursiv_modulo(a,b):
```

```
{  
    if (b==0)  
        return a;  
    else  
        return euklid_rekursiv_modulo(b, a%b);  
}
```


Apropos...

- ▶ ggT via Primfaktorzerlegung?

 Code

Zusammenfassung

- ▶ Feststellung: Unterschiedliche Algorithmen für die selbe Aufgabe
- ▶ Angenommen eine Aufgabe sei gegeben (z.B.: Automatisierte Sortierung von Paketen)
 - ▶ Welche Algorithmen eignen sich hierzu?
 - ▶ Welche Unterschiede gibt es
 - ▶ Welcher Algorithmus eignet sich am besten?

Natürliche Zahlen in C-ähnlichen Sprachen

Natürliche Zahlen

In Computern verwendet man **Binärdarstellung** mit einer fixen Anzahl Ziffern (genannt **Bits**).

Die **primitiven Datentypen** für **natürliche Zahlen** sind:

- ▶ **8 Bits** (ein **Byte**), darstellbare Zahlen: $\{0, \dots, 255\}$
in C: `unsigned char`
- ▶ **16 Bits**, darstellbare Zahlen: $\{0, \dots, 65535\}$
in C: `unsigned short`
- ▶ **32 Bits**, darstellbare Zahlen: $\{0, \dots, 4294967295\}$
in C: `unsigned long`
- ▶ **64 Bits**, darstellbare Zahlen: $\{0, \dots, 2^{64} - 1\}$
in C: `unsigned long long`

Ganze Zahlen in C-ähnlichen Sprachen

Ganze Zahlen

Die primitiven Datentypen für ganze Zahlen sind:

- ▶ 8 Bits: unsigned char $\{0, \dots, 255\}$
signed char $\{-128, \dots, 127\}$
- ▶ 16 Bits: unsigned short $\{0, \dots, 65535\}$
signed short $\{-32768, \dots, 32767\}$
- ▶ 32 Bits: unsigned long $\{0, \dots, 2^{32} - 1\}$
signed long $\{-2^{31}, \dots, 2^{31} - 1\}$
- ▶ 64 Bits: unsigned long long $\{0, \dots, 2^{64} - 1\}$
signed long long $\{-2^{63}, \dots, 2^{63} - 1\}$
- ▶ signed kann weggelassen werden (ausser bei char!)
- ▶ unsigned int und signed int sind je nach System 16, 32 oder 64 Bit

Überlauf

- ▶ Überschreitung des für einen Datentyp definierten Bereichs
- ▶ zum Beispiel durch Addition zweier Zahlen:
8 Bit signed char a und b
 $a = 120$ (01111000), $b = 8$ (0001000)
 $\rightarrow a + b = 128$ (10000000) > 127
- ▶ Ergebnis: $a + b = -128$ da $-128 = (128 \text{ modulo } 256) - 256$

Überlauf

- ▶ Am 4. Juni 1996 startete die ESA eine unbemannte Rakete. 40 Sekunden nach dem Start explodierte die Ariane 5.
- ▶ Verlust ca. 500 Millionen Dollar. Entwicklungskosten ca. 7 Milliarden Dollar.
- ▶ Ursache für den Absturz: Der Bordcomputer stürzte 36.7 Sek. nach dem Start ab als er versuchte, den Wert der horizont. Geschwindigkeit von 64 Bit Gleitkommadarstellung in 16 Bit signed Integer umzuwandeln.
- ▶ Die entsprechende Zahl war grösser als $2^{15} = 32768$ und erzeugte einen Overflow.
- ▶ <https://www5.in.tum.de/~huckle/bugs.html>

 Code