Einführung in die Informatik 1

– Algorithmen und algorithmische Sprachkonzepte –

Sven Kosub

AG Algorithmik/Theorie komplexer Systeme
Universität Konstanz

E 202 | Sven.Kosub@uni-konstanz.de | Sprechstunde: Freitag, 12:30-14:00 Uhr, o.n.V.

Wintersemester 2008/2009

Informatik

Informatik =
Wissenschaft von der systematischen Verarbeitung von Informationen, insbesondere ...

... der automatischen Verarbeitung mit Hilfe von Rechenanlagen
= Computerwissenschaft

zentrale Gegenstände der Informatik:

- Information √
- Algorithmen (≃ Systematik der Verarbeitung) heute
- Computer (≃ Rechenanlagen) √

Algorithmus: Begriffsdefinition

Algorithmus:

- präzise festgelegtes Verfahren zur Lösung von Problemen (bzw. einer Klasse von Problemen), das aus endlich vielen, effektiv ausführbaren elementaren Lösungsschritten besteht
- Problemklasse ist Menge von Eingabe-Ausgabe-Paaren

Algorithmen müssen folgenden Bedingungen genügen:

- Spezifikation
- Durchführbarkeit
- Korrektheit

Algorithmus: Spezifikation

- **Eingabespezifikation:**
  Es muss genau spezifiziert sein, welche Eingabegrößen erforderlich sind und welchen Anforderungen diese Größen genügen müssen, damit das Verfahren funktioniert

- **Ausgabespezifikation:**
  Es muss genau spezifiziert sein, welche Ausgabegrößen (Resultate) mit welchen Eigenschaften berechnet werden

Algorithmus: Durchführbarkeit

- **endliche Beschreibung:**
  Verfahren muss in einem endlichen Text vollständig beschrieben sein

- **Effektivität:**
  Jeder Schritt des Verfahrens muss effektiv (d.h. tatsächlich) „mechanisch“ ausführbar sein

  $\rightarrow$ Beachte: Effektivität $\neq$ Effizienz

- **Determiniertheit:**
  Verfahrensablauf ist zu jedem Zeitpunkt fest vorgeschrieben

Algorithmus: Korrektheit

- **partielle Korrektheit:**
  jedes berechnete Ergebnis genügt der Ausgabespezifikation, sofern die Eingaben der Eingabespezifikation genügen

- **Terminierung:**
  Algorithmus hält nach endlich vielen Schritten mit einem Ergebnis an, sofern die Eingabe der Eingabespezifikation genügt

Programme und Programmiersprachen

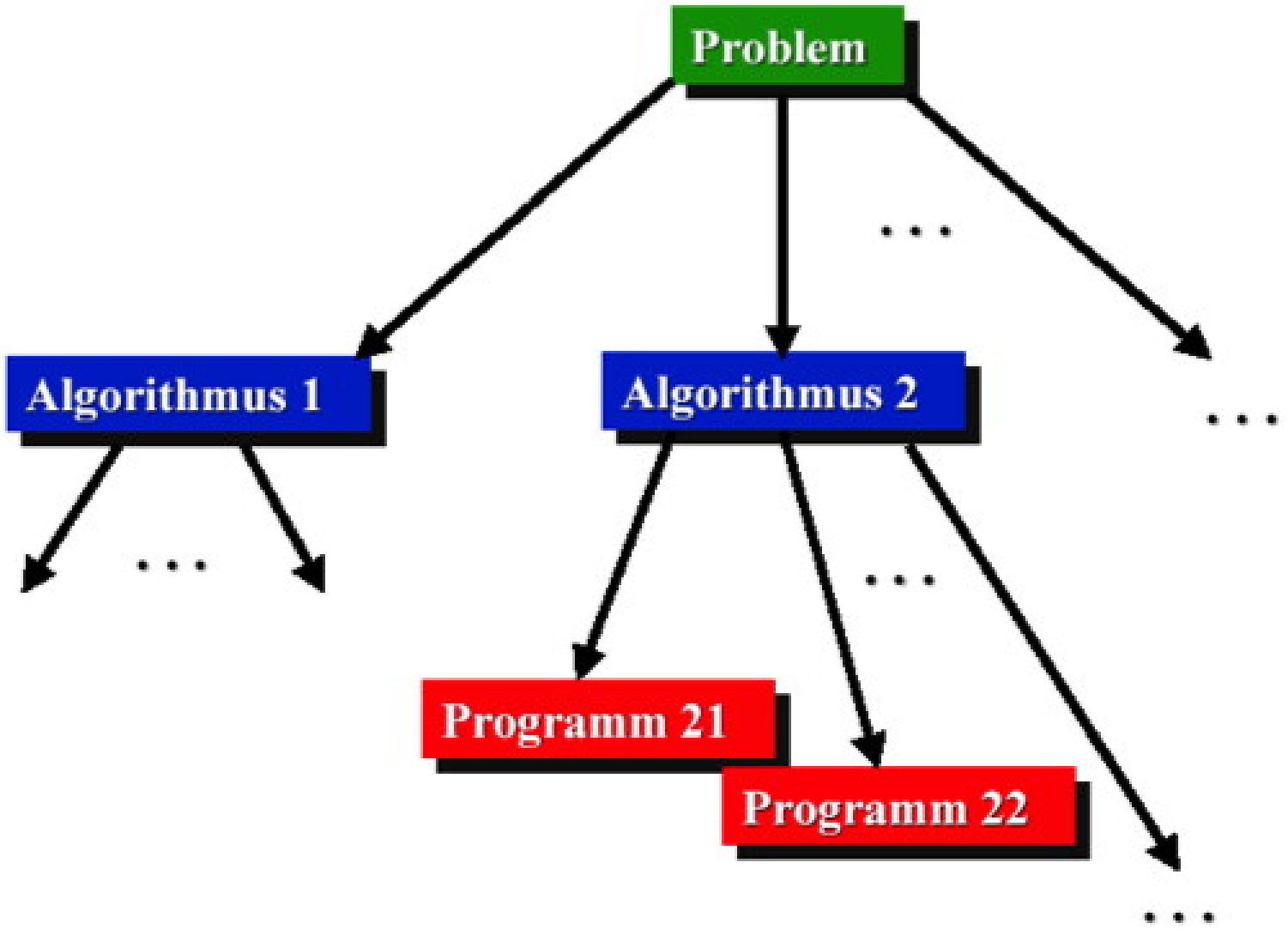
- Computer führen nur in einer formalen Sprache beschriebene Algorithmen aus
- Programmiersprache = formale Sprache für einen Computer
- Programm = Beschreibung des Algorithmus in Programmiersprache

- Programm repräsentiert nur einen Algorithmus
- Algorithmus wird durch viele verschiedene (i.A. unendlich viele) Programme repräsentiert

- Programmierung setzt Algorithmenentwurf voraus:

Kein Programm ohne Algorithmus!

Problem-Algorithmus-Programm



Algorithmus: Aufbau

Form der Algorithmenbeschreibung:

- Alltagssprache
- konkrete Programmiersprache
- Pseudocode

Elemente der Algorithmenbeschreibung:

- Folgen einzelner Bearbeitungsschritte
- Schrittwiederholung durch Iteration: Rückverweise in der Folge wie z.B. „Weiter mit Schritt (2)“
- Schrittwiederholung durch Rekursion: Wiederaufruf des Algorithmus mit einfacherer Problemstellung
- Sprünge (bedingt/unbedingt)

Algorithmus: Aufbau

Grundschema des Algorithmenaufbaus:

Name des Algorithmus und Liste von Parametern

Spezifikation der Eingaben

Spezifikation der Ausgaben

1. **Vorbereitung:** Einführung von Hilfsgrößen etc.
2. **Trivialfallbehandlung:** Prüfe, ob ein einfacher Fall vorliegt; falls ja, dann Beendigung mit Ergebnis
3. **Arbeit (Problemreduktion, Ergebnisaufbau):** Reduziere Problemstellung X auf eine einfachere Form X' mit $X > X'$ bzgl. einer (wohlfundierten) Ordnung $>$ und baue entsprechend der Reduktion einen Teil des Ergebnis auf
4. **Rekursion** bzw. **Iteration:** Rufe zur Weiterverarbeitung den Algorithmus mit dem reduzierten X' erneut auf (Rekursion) bzw. fahre mit X' statt X bei Schritt 2 fort (Iteration)

Algorithmus: Beschreibungsformen

Form der Algorithmenbeschreibung (verfeinert):

- elementar-iterativ
- Flussdiagramme
- strukturiert-iterativ
- rekursiv
- Pseudocode

Beschreibe ein Verfahren zur Berechnung des Divisionrestes r bei Division von a durch b, d.h. finde $0 \leq r < b$ mit $a = t \cdot b + r$ für geeignetes t

Algorithmus: Beschreibungsformen

elementar-iterative Beschreibungsform:

- Iterationen werden durch Sprünge realisiert

Algorithmus zur Berechnung von $\mathrm{mod}(a, b)$:

$\mathrm{mod}(a, b)$

// Anforderungen (Eingabe): $a, b \in \mathbb{Z}, a \geq 0, b > 0$

// Zusicherung (Ausgabe): Resultat ist Divisionsrest von $\frac{a}{b}$

1. Kopiere a nach r
2. Prüfe, ob $r < b$ gilt; falls ja, dann gib Resultat r aus
3. Subtrahiere b von r und speichere Resultat wieder in r
4. Weiter mit Schritt 2

Algorithmen: Beschreibungsformen

Flussdiagramme:

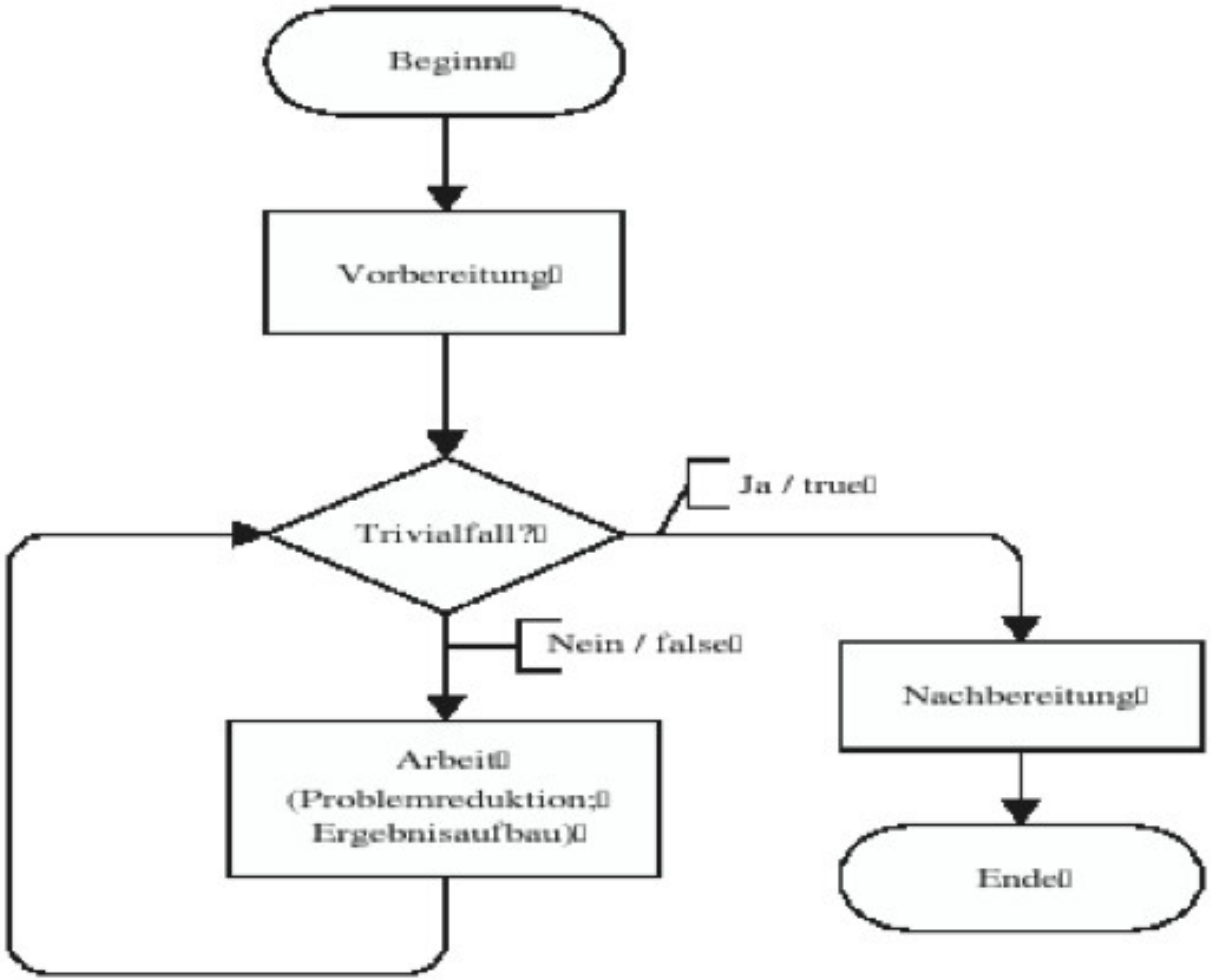
- graphische Darstellung des Steuerungsverlaufs
- Komponenten von Flussdiagrammen:

Beginn / Ende des Algorithmus

Anweisungen, Operationen

Verzweigungen: Der weitere Verlauf hängt vom Wahrheitswert des Ausdrucks im Inneren ab.

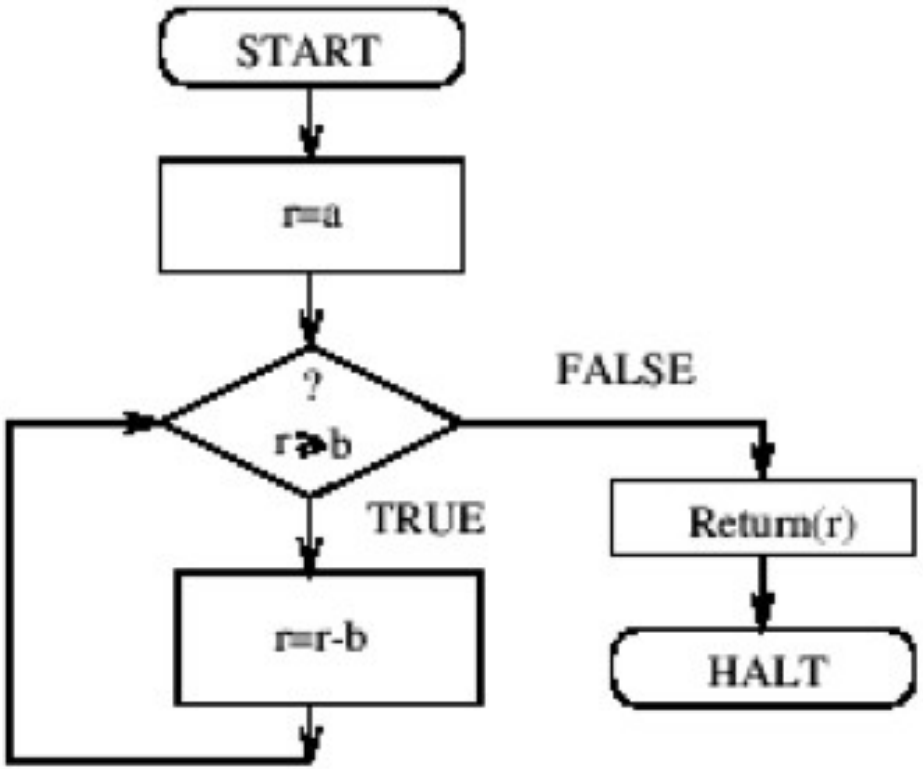
Algorithmen: Beschreibungsformen



algorithmisches Grundschema als Flussdiagramm

Algorithmen: Beschreibungsformen

Flussdiagramm für den Algorithmus mod(a, b)



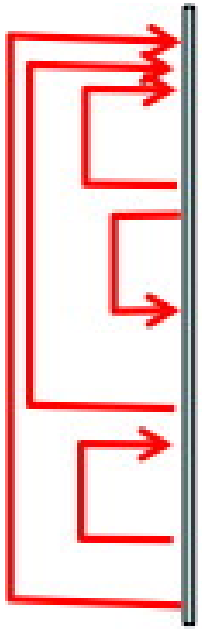
Kontrollfluss für mod(7,3)

- START
- $a = 7,\ b = 3$
- $r = 7$
- $7 \geq 3$? TRUE
- $r = 7 - 3$
- $4 \geq 3$? TRUE
- $r = 4 - 3$
- $1 \geq 3$? FALSE
- RETURN(1)
- HALT

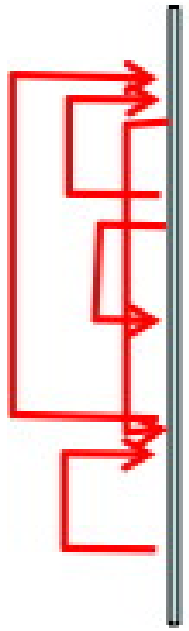
Algorithmen: Beschreibungsformen

strukturiert-iterative Beschreibungsform:

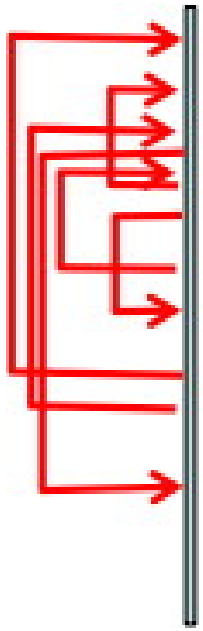
- Sprünge dürfen nur ineinander geschachtelt sein, d.h. Schleifen überkreuzen sich nicht



strukturiert-iterativ



elementar-iterativ



Spaghetti-Code

Algorithmen: Beschreibungsformen

Sprungbehandlung bei strukturiert-iterativer Beschreibungsform:

- keine unbedingten Sprünge (kein `goto ...`)
- Sprünge werden nur implizit angegeben
- Sprünge Teil höhere Iterationsstrukturen
  - Fallunterscheidung (`if ... then ... else ...`)
  - Schleifen (`while ... do ...`)

Java erlaubt strukturiert-iterative Beschreibung (kein `goto` in Java!)

Algorithmen: Beschreibungsformen

Fallunterscheidung in Java:

- **syntaktisch:**

  ```
  if (Bedingung) { Anweisungssequenz 1 }
  else { Anweisungssequenz 2 }
  ```

- **semantisch:**

  Ist Bedingung erfüllt (`true`), dann führe Anweisungssequenz 1 aus, sonst führe Anweisungssequenz 2 aus

Algorithmen: Beschreibungsformen

Schleifen in Java:

- **syntaktisch:**

  `while` (Bedingung) { Anweisungssequenz }

- **semantisch:**

  Bei Eintritt in die `while`-Schleife wird Bedingung überprüft

  Ist Bedingung erfüllt (`true`), führe die Anweisungssequenz aus und gehe zum Test der Bedingung zurück

  Ist Bedingung nicht erfüllt (`false`), verlasse die `while`-Schleife ohne Ausführung der Anweisungssequenz

Beachte: `while`-Schleife entspricht der Konstruktion (falls `goto` vorhanden)

M: `if` (Bedingung) { Anweisungssequenz; `goto` M; }

Algorithmen: Beschreibungsformen

strukturiert-iterativer Algorithmus für die mod-Funktion (in Java):

```
int mod(int a, int b) {
    /* Anforderungen (Eingabe): a: a>=0; b: b>0 */
    /* Zusicherung (Ausgabe): r: a=t*b + r und 0<r<=b */
    /* 1. Vereinbarungen */
    int r;
    /* 2. Initialisierungen */
    r=a;
    /* 3. Iterationsbeginn */
    while (r>=b) {
        /* 4. Problemreduktion */
        r=r-b;
    /* 5. Iterationsende */
    }
    /* 6. Ergebnisrückgabe */
    return(r);
}
```

Algorithmen: Beschreibungsformen

rekursive Beschreibungsform:

1. **Basis:**

   Gib eine direkte Lösung für den Fall an, dass Eingabe X einfach ist

2. **Rekursionsschritt:**

   Führe Lösung für Problem $P(X)$ für eine komplexe Eingabe X durch einen Schritt der Problemreduktion auf die Lösung des gleichen Problems $P(X')$ für eine einfachere Eingabe X' zurück

   Dabei muss $X > X'$ für eine wohlfundierte Ordnung $>$ gelten

rekursive Definition der mod-Funktion (für natürliche Zahlen a und $b > 0$):

$$\mathrm{mod}(a,b) =_{\mathrm{def}} \begin{cases} a & \text{falls } a < b \\ \mathrm{mod}(a-b,b) & \text{falls } a \geq b \end{cases}$$

Algorithmen: Beschreibungsformen

rekursiver Algorithmus für die mod-Funktion (in Java):

```
int mod(int a, int b) {
  /* Anforderungen (Eingabe): a: a>=0; b: b>0 */
  /* Zusicherung (Ausgabe): r: a=t*b + r und 0<r<=b */
  /* 1. Vereinbarungen */
  int r;
  /* 2. Initialisierungen */
  r=a;
  /* 3. Einfacher Fall */
  if (r<b) { return(r); }
  /* 4. Problemreduktion */
  r=r-b;
  /* 5. Rekursionsschritt */
  return(mod(r,b));
}
```

Algorithmen: Beschreibungsformen

Pseudocode:

- Mischform aus Syntax höherer Programmiersprachen und natürlicher Sprache
- typische Elemente von Programmiersprachen: `if ...`, `while ...`
- Trade-off zwischen Anschaulichkeit (natürliche Sprache) und Formalisierung (Programmiersprache)

```
if (keine Elemente mehr zu sortieren) { return(Fertig) }
```

Algorithmen: Beschreibungsformen

Euklidischer Algorithmus:

- Berechnung des ggT zweier natürlicher Zahlen, z.B.

$$\mathrm{ggT}(36, 120) = 12, \quad \text{da } 36 = 2^2 \cdot 3^2 \text{ und } 120 = 2^3 \cdot 3 \cdot 5$$

- rekursive (mathematische) Definition

$$\mathrm{ggT}(a, b) =_{\mathrm{def}} \begin{cases} a & \text{falls } b = 0 \\ \mathrm{ggT}(b, a) & \text{falls } b > a \\ \mathrm{ggT}(b, \mathrm{mod}(a, b)) & \text{sonst} \end{cases}$$

ergibt für ggT(36,120):

$$\begin{aligned} \mathrm{ggT}(36, 120) &= \mathrm{ggT}(120, 36) && (\text{„}b > a\text{“}) \\ &= \mathrm{ggT}(36, \mathrm{mod}(120, 36)) = \mathrm{ggT}(36, 12) && (\text{„sonst“}) \\ &= \mathrm{ggT}(12, \mathrm{mod}(36, 12)) = \mathrm{ggT}(12, 0) && (\text{„sonst“}) \\ &= 12 && (\text{„}b = 0\text{“}) \end{aligned}$$

Algorithmen: Beschreibungsformen

Euklidischer Algorithmus (Forts.):

- rekursiver Algorithmus in Pseudo-Code:

Algorithmus:	`ggT(a,b)`
Eingabe:	natürliche Zahlen a und b
Ausgabe:	größter gemeinsamer Teiler von a und b

```
(1) if (b==0) { return (a) };                    /* Trivialfall
(2) if (b>a) { return (ggT(b,a)};                /* einfacher Fall
(3) return (ggT(b,mod(a,b)));                    /* Reduktion und Rekursion
```

- Warum terminiert der Euklidische Algorithmus?
  - Eingaben sind Paare (a, b) mit $a, b \in \mathbb{N}$
  - $(a, b) > (a', b') \iff_{\text{def}} a + b > a' + b'$ oder $a + b = a' + b' \wedge a < a'$
  - Rekursionsaufrufe bei (a, b) erfolgen mit (a', b'), sodass $(a, b) > (a', b')$

$$(36, 120) > (120, 36) > (36, 12) > (12, 0)$$

  - für jedes (a, b) gibt es nur endlich viele (a', b') mit $(a', b') < (a, b)$