Euklidischer Algorithmus

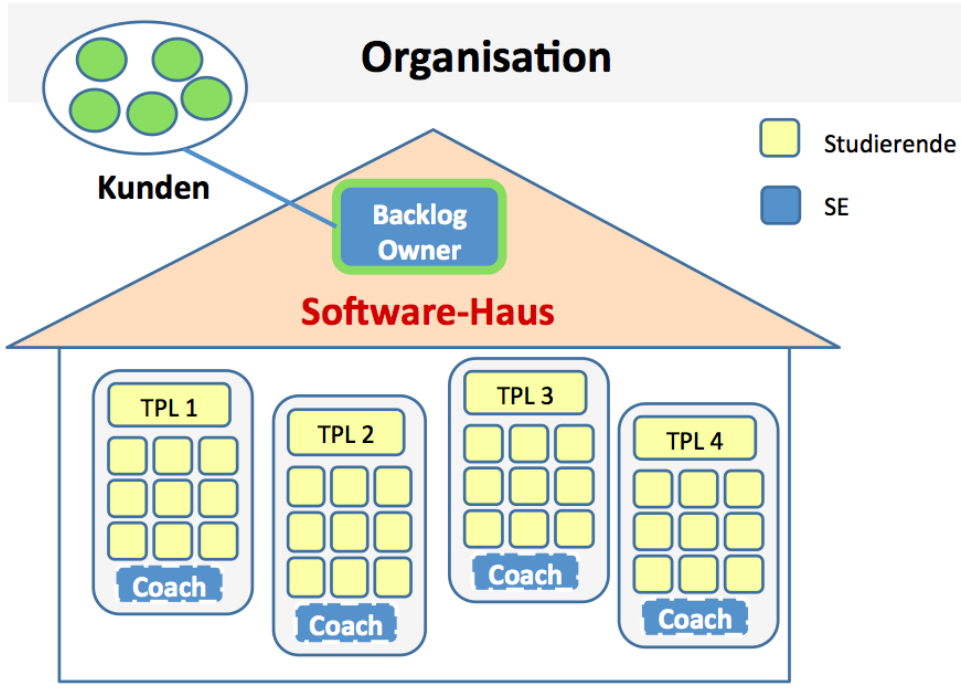
Proseminar "Algorithms Unplugged"
Wintersemester 2014/15
Leibniz Universität Hannover

Bharat Ahuja

1. Dezember 2014

Problem 1: Software Projekt

Bestimmung der Anzahl von Teams



Quelle: SE-Webseite (Stand 21. Oktober 2014)

Software Projekt

Die Problemstellung

- 60 Inf., 24 Math., 24 Tech. Inf.
- Alle Teams sind gleich groß.
- Alle Studenten müssen einem Team zugeordnet sein.
- Gleichmäßige Verteilung der Studiengänge in jedem Team.
- Möglichst große Anzahl von Teams.

Software Projekt

Sei $n \in \mathbb{N}$ die Anzahl der Teams.

$$\Rightarrow n|60,\ n|24 \text{ und } n|24 \qquad \text{(i)}$$

Wir suchen nach der größten Zahl n, die (i) erfüllt.

$$\Rightarrow n = ggT(60, 24, 24) = 12 \text{ Teams}$$

Software Projekt

Lösungsanalyse

Es gibt 12 Teams mit jeweils –

- 9 Studenten
- 5 Informatik Studenten
- 2 Mathematik Studenten
- 2 Technische-Informatik Studenten

Quelle: Wikipedia (Stand 21. Oktober 2014)

Problem 2: Periodische Zikaden

Natürliche Selektion

- Zikaden treten alle 13 oder 17 Jahre in Massen auf.
- Ihre Feinde leben in 2-, 4- oder 6-Jahres-Rhythmen
- So können sie den meisten Raubtieren ausweichen.
- So kann man auch später Arbeit ausweichen.

Periodische Zikaden

Überlebensstrategie

- $ggT(4, 13) = 1$
- Teilerfremdheit durch Vergleich der Primfaktoren.
- Die Zikaden berechnen schnell den *g.g.T.*, ohne Zahlen zu faktorisieren.
- Abstrakter: Man will schnell Zahlen auf Teilerfremdheit prüfen können.

Von der Problemstellung abstrahieren

Wie kann man den g.g.T. schnell bestimmen?

- Euklidischer Algorithmus
- Binary GCD Algorithm von Stein
- GCD Algorithm von Lehmer

Euklidischer Algorithmus

Historische Entwicklung

- ca. 300 *v.Chr.* in *Buch VII – Die Elemente* vorgestellt.
- Wahrscheinlich nicht selbst erfunden.
- Als geometrischen Algorithmus vorgestellt. Stäbe zerlegt.

Vorstellung der Algorithmen

Wir befassen uns mit zwei bekannten Varianten des euklidischen Algorithmus –

- LangsamEuklid
- Euklid

LangsamEuklid

LangsamEuklid

1: **while** $a \neq b$ **do**
2: Falls a größer ist als b, $a \leftarrow a - b$
3: Falls b größer ist als a, $b \leftarrow b - a$
4: **end while**
5: Gib den gemeinsamen Wert der Zahlen aus

LangsamEuklid

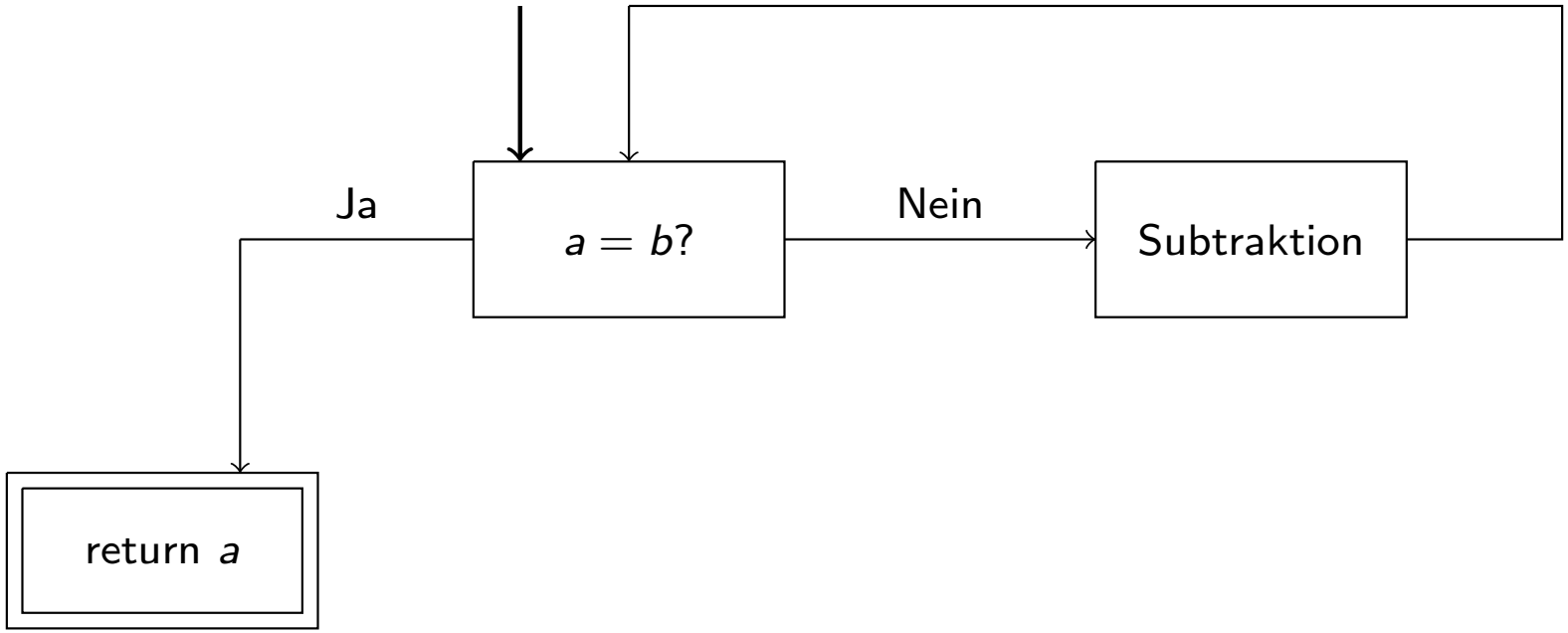


Figure: Flussdiagramm LangsamEuklid

Beispiel

Betrachte das folgende Beispiel mit $a = 24, b = 9$:

Iteration Nr.	a	b
	24	9
1	15	9
2	6	9
3	6	3
4	**3**	**3**

$\therefore$ der *g.g.T.* von 24 und 9 ist 3.

- Die Zahlen bleiben positiv und ganzzahlig beim Subtraktionsschritt.
- Einer der beiden Variablen wird in jeder Iteration um mindestens 1 verringert.
- D.h. der Algorithmus terminiert nach maximal $a + b$ Schritten.

- Die Menge der gemeinsamen Teiler von a und b ist gleich der Menge der gemeinsamen Teiler von $a - b$ und b.
  $\therefore g.g.T.(a, b) = g.g.T.(a - b, b)$
- Eine Iteration des Algorithmus ändert die Lösung nicht.
- $g.g.T.(a, a) = a$

□

LangsamEuklid

Verbesserungsvorschläge

- $(1069, 2) \to (1067, 2) \to \ldots (3, 2) \to (1, 2) \to (1, 1)$
- In den meisten Iterationen braucht man nicht überprüfen welche der beiden Zahlen größer ist.

Euklid

Euklid

1: **if** $a < b$: vertausche a und b.
2: **while** $b > 0$ **do**
3: berechne q, r mit $a = q \cdot b + r$, wobei $0 \leq r < b$
4: $a \leftarrow b, b \leftarrow r$
5: **end while**
6: **return** a

Euklid

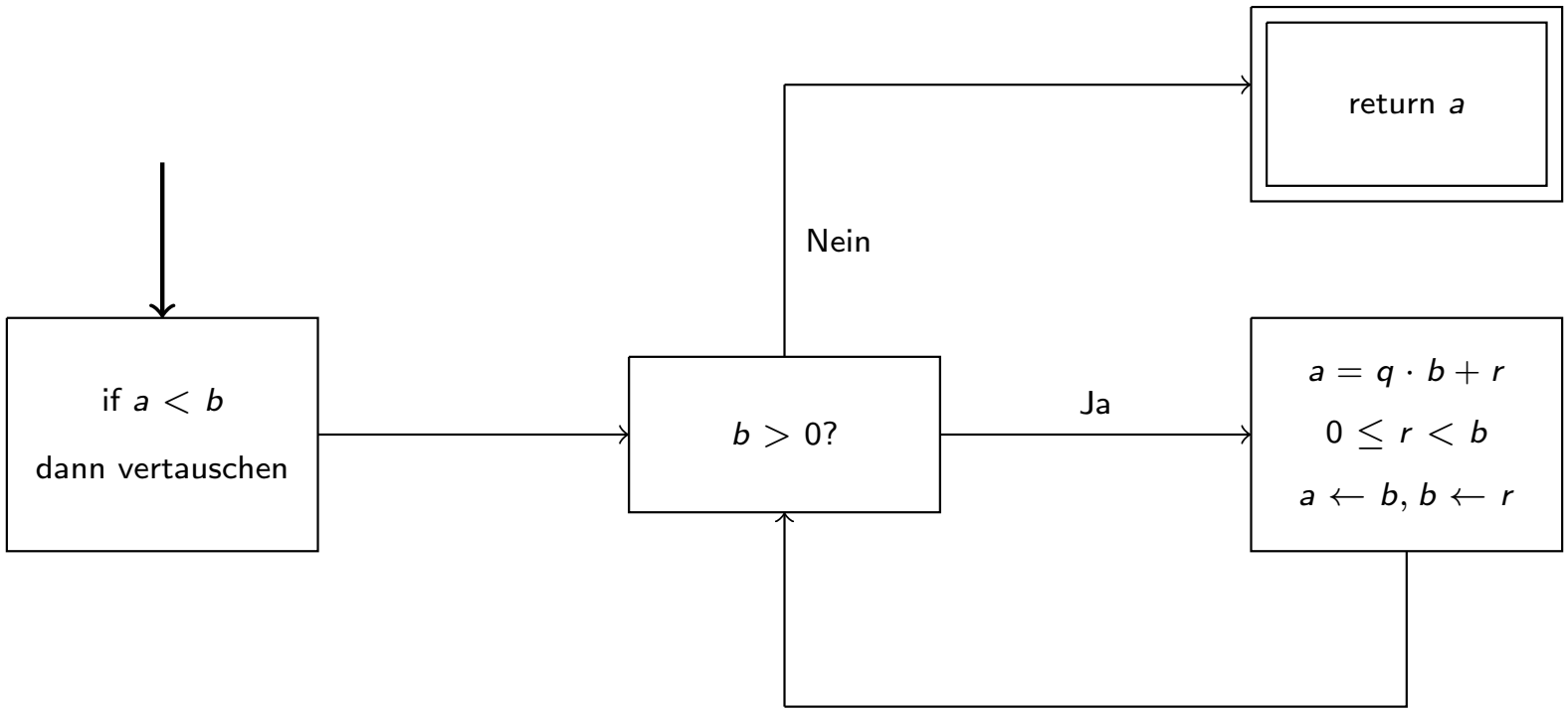


Figure: Flussdiagramm Euklid

Laufzeitanalyse

Euklid vs. LangsamEuklid

Sei ohne Einschränkung $a > b$.
Im ersten Durchlauf gilt –

$$a = q \cdot b + r, \text{ mit } r < b \tag{1}$$

Außerdem gilt –

$$a \geq b + r, \text{ denn } q \geq 1 \tag{2}$$

Aus (1), (2) folgt[1]

$$r < \frac{a}{2}$$

[1] denn $a \geq b + r > r + r = 2r$

Laufzeitanalyse

Euklid vs. LangsamEuklid

Wegen der Reduktion[2] sind die Variablen a, b nach 2 Iterationen beide höchstens halb so groß wie der Anfangswert von a.

Durch Induktion sind nach $2 \cdot k$ Iterationen die beiden Variablen höchstens so groß wie $\frac{a}{2^k}$.

Wenn $k > \log_2 a$, dann sind beide Variablen null. Aber der Algorithmus muss schon vorher terminieren, wenn $b = 0$.

Von daher ist $2 \cdot \log_2 a$ eine obere Schranke für die Anzahl der Durchläufe, die viel besser als $a + b$ ist.

[2] $a \leftarrow b, b \leftarrow r$

Laufzeitanalyse

Euklid

- Sei T_b die durchschnittliche Anzahl an Iterationen des euklidischen Algorithmus, wenn der Parameter b festliegt über alle $a \in \mathbb{N}$.
- Nach der ersten Iteration ist immer nur der Divisionsrest relevant. Deswegen müsste T_b für alle $b \in \mathbb{N}$ existieren und lässt sich als Durchschnitt über die Anzahl an Iterationen bei $a = 1, a = 2, \ldots, a = b$ berechnen.[3]

$$\therefore T_b = \frac{1}{b} \sum_{0<k\leq b} T(k, b)$$

[3] Ohne die Zahlen vorher zu vertauschen natürlich!

Laufzeitanalyse

Euklid : Beispiel T_5

$T_5 = \frac{(T(1,5)+T(2,5)\ldots T(5,5))}{5}$
Zum Beispiel $T(2,5) = 3$, denn

$$2 = 0 \cdot 5 + 1 \quad (1)$$
$$5 = 2 \cdot 2 + 1 \quad (2)$$
$$2 = 2 \cdot 1 + 0 \quad (3)$$

Im Durchschnitt muss man 2.6 Iterationen ausführen wenn man eine natürliche Zahl auf Teilerfremdheit mit 5 überprüfen will, denn $T_5 = \frac{2+3+4+3+1}{5} = 2.6$.

Laufzeitanalyse

EUKLID : Abschätzung von T_b

Für große $b \in \mathbb{N}$ gilt

$$T_b \approx 1 + \frac{1}{b}(T_0 + T_1 \ldots T_{b-1})$$

da nach der ersten Iteration von $T(k, b)$ ungefähr T_k Iterationen noch bleiben.[4]

$$k = 0 \cdot b + k$$
$$b = q_2 \cdot k + r_2$$
$$\vdots$$

D.h. $T_b \approx S_b$, wobei

$$S_0 := 0, S_n := 1 + \frac{1}{n}(S_0 + S_1 \cdots + S_{n-1})$$

[4] Hier hat man also b durch eine zufällige Zahl modulo k geschätzt.

Laufzeitanalyse

Euklid

Die Rekursion lässt sich folgenderweise lösen.

$$\begin{aligned} S_{n+1} &= 1 + \frac{1}{n+1}(S_0 + S_1 \cdots + S_n) \\ &= 1 + \frac{1}{n+1}(n(S_n - 1) + S_n) \\ &= S_n + \frac{1}{n+1} \end{aligned}$$

$\Rightarrow S_n = H_n = 1 + \frac{1}{2} + \ldots \frac{1}{n} \Rightarrow T_b \approx \ln b + O(1)$
Aus der Praxis hat sich ergeben, dass diese Schätzung von T_b eher pessimistisch ist, und dass diese Funktion etwas langsamer wächst.

Laufzeitanalyse

Euklid

- Sei umgekehrt U_a die durchschnittliche Anzahl an Iterationen des euklidischen Algorithmus, wenn der Parameter a festliegt über alle $b \in \mathbb{N}$.
- Dann gilt $b \leq a$ nur in endlich vielen Fällen. Ansonsten werden immer die Zahlen in der ersten Iteration getauscht. Danach ist die Anzahl der Iterationen identisch wie oben. Dann muss $U_a = T_a + 1$ gelten.

Laufzeitanalyse

Euklid

Betrachte die Fibonacci-Zahlen –

$$\begin{aligned} f_{n+1} &= 1 \cdot f_n + f_{n-1} \\ f_n &= 1 \cdot f_{n-1} + f_{n-2} \\ &\vdots \\ f_2 &= 1 \cdot f_1 + 0 \\ f_1 &= 1 \end{aligned}$$

Im Euklidischen Algorithmus werden die Reste umso schneller klein, je größer die Quotienten sind. Im Fall der Fibonacci Zahlen, nehmen die Quotienten alle den kleinstmöglichen Wert an. Deswegen ist dieser der ungünstigste Fall.