Euklidischer Algorithmus

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

Bharat Ahuja

1. Dezember 2014

Einführung

Was ist der Euklidische Algorithmus?

Der euklidische Algorithmus ist ein Algorithmus, mit dem sich der größte gemeinsame Teiler (g.g.T.) zweier natürlicher[1] Zahlen berechnen lässt.

[1] $\mathbb{N} = \{1, 2, 3 \dots\}$

Einführung

Historische Entwicklung

Diesen Algorithmus hat Euklid ca. 300 *v.Chr.* in *Buch VII – Die Elemente* (Proposition 1 und 2) als einen geometrischen Algorithmus vorgestellt. Allerdings hat er diesen Algorithmus wahrscheinlich nicht erfunden.
Euklid hat später einen erweiterten (nicht immer endlichen) Algorithmus angegeben, so dass man den *g.g.T.* reeller Zahlen berechnen kann.

Einführung

Anwendung/Relevanz

Der größte Vorteil des Algorithmus ist das leichte Prüfen auf Teilerfremdheit zweier Zahlen.
Die Teilerfremdheit zweier Zahlen kann man alternativ durch das Vergleichen der Primfaktoren überprüfen. Die Bestimmung der Primfaktorzerlegung einer Zahl liegt aber in NP.
Andererseits lässt uns dieser Algorithmus schnell den *g.g.T.* berechnen, ohne diese Zahlen faktorisieren zu müssen. Ist der *g.g.T.* gleich 1, dann sind die Zahlen teilerfremd.

Vorstellung der Algorithmen

Wir befassen uns mit zwei bekannten Varianten des euklidischen Algorithmus –

- LangsamEuklid
- Euklid

LangsamEuklid

So hat Euklid den Algorithmus zuerst vorgestellt. Sein Algorithmus hat Stäbe zerlegt statt Zahlen zu subtrahieren.

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

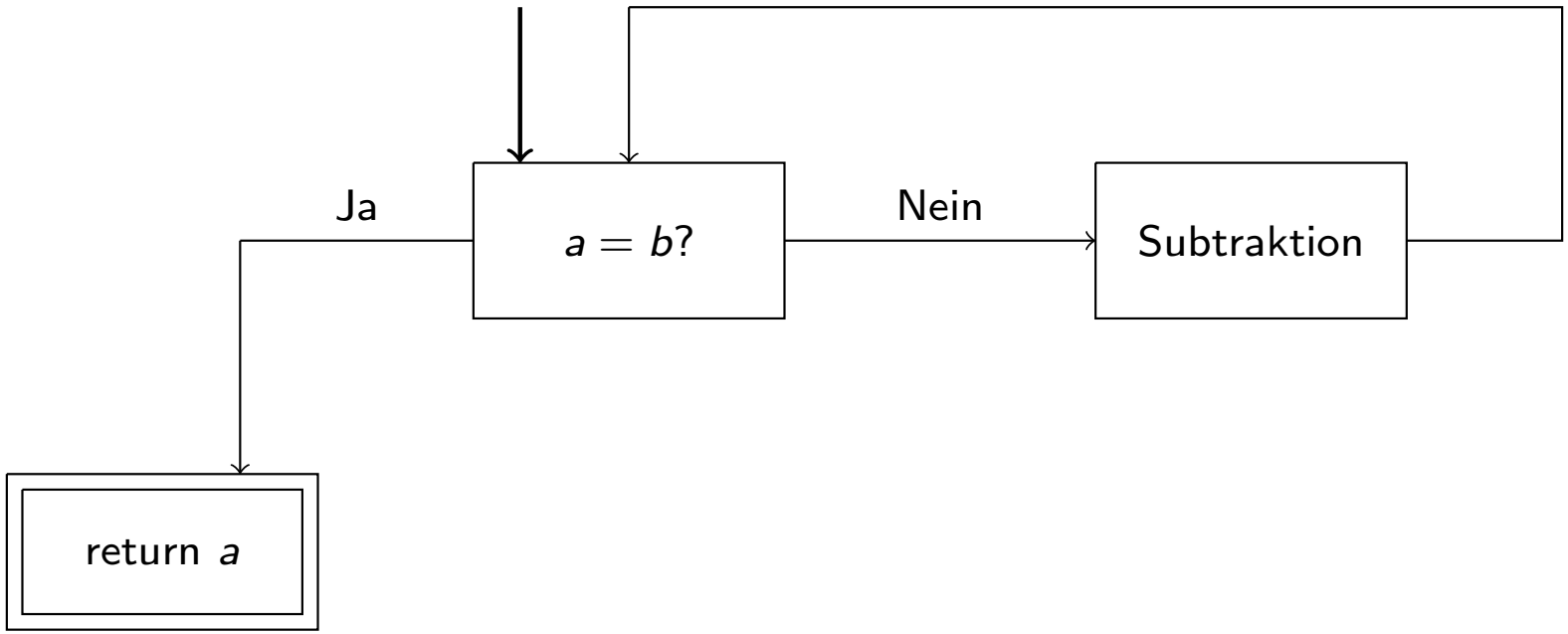

Figure: Flussdiagramm LangsamEuklid

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, da man immer eine kleinere Zahl von einer größeren Zahl abzieht. D.h. die beiden Zahlen sind mindestens 1. Weil einer der beiden Variablen in jeder Iteration um mindestens 1 verringert wird, terminiert der Algorithmus nach maximal $a + b$ Schritten.

LangsamEuklid

Korrektheit

Ohne Einschränkung sei $a > b$. Dann gelten –

1. $\forall t \in \mathbb{N} : t|a, t|b \Rightarrow t|(a - b)$
2. $\forall t \in \mathbb{N} : t|(a - b), t|b \Rightarrow t|a$

D.h. die Menge der gemeinsamen Teiler von a und b ist gleich der Menge der gemeinsamen Teiler von $a - b$ und b.
Da die Mengen gleich sind, haben sie auch das gleiche Minimum.

$\therefore g.g.T.(a, b) = g.g.T.(a - b, b)$
Eine Iteration des Algorithmus ändert die Lösung nicht.

Zudem liefert uns $g.g.T.(a, a) = a$ die Korrektheit. □

LangsamEuklid

Verbesserungsvorschläge

Dieser Algorithmus endet zwar immer mit der richtigen Lösung, aber er ist immerhin verbesserungswürdig.

- Wenn eine Zahl sehr groß ist, dann subtrahiert man die zweite Zahl mehrmals von der ersten ab.
- z.B. $(1069, 2) \to (1067, 2) \to \ldots (3, 2) \to (1, 2) \to (1, 1)$
- In den meisten Iterationen braucht man nicht überprüfen welche der beiden Zahlen größer ist.

Diese Idee hat man implementiert und so ergibt sich der moderne euklidische Algorithmus.

Euklid

Euklid

1: **if** $a < b$: vertausche a und b. ▷ *Optimierung*[2]
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

[2] Reduziert die Laufzeit um eine Iteration in 50% der Fälle. Ohne diesen Schritt würde der erste Durchlauf automatisch die Zahlen tauschen, falls $a < b$.

Euklid

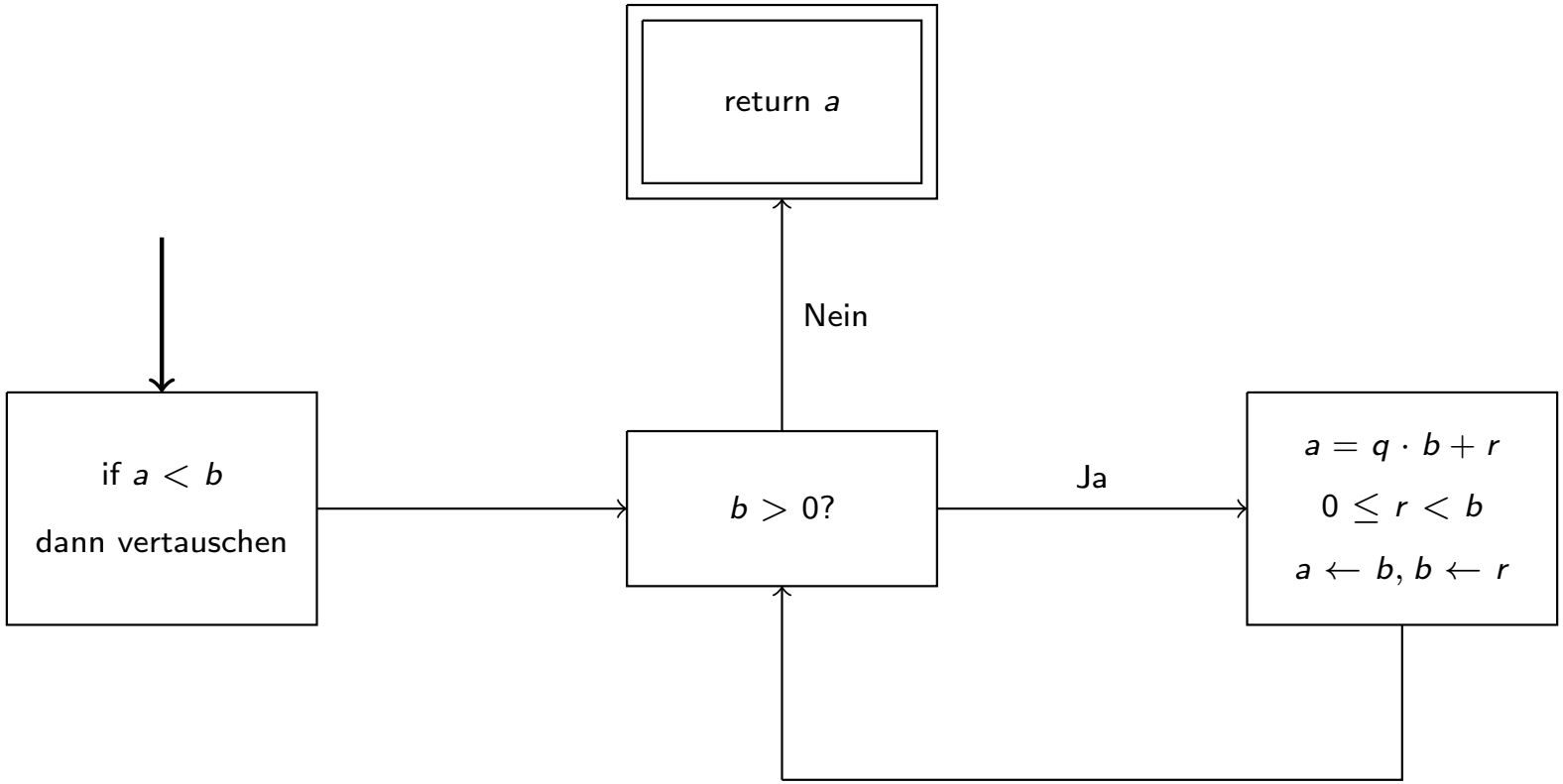

Figure: Flussdiagramm Euklid

In jeder Iteration gilt nach Schritt 3 des Algorithmus stets $0 \leq r < b$.

Wenn $r = 0$ gilt, dann endet der Algorithmus.
Aber wenn $r \neq 0$, dann sinkt der Wert von b nach Schritt 4 in dieser Iteration, und der Algorithmus wird weiter ausgeführt.
Diese Folge von positiven streng monoton fallenden Werten von b muss terminieren.

□

Analog zum LANGSAMEUKLID Algorithmus gilt die Gleichheit der Mengen der gemeinsamen Teiler von (a, b) und $(a - q \cdot b, b)$.

$$\Rightarrow g.g.T.(a, b) = g.g.T.(a - q \cdot b, b)$$

□

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[3]

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

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

Laufzeitanalyse

Euklid vs. LangsamEuklid

Wegen der Reduktion[4] 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.

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

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.[5]

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

[5] 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.[6]

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

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

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

[6] 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.