Einführung in Algorithmen

Theorie (L)

14. März 2025

Inhaltsverzeichnis

1 Der Algorithmusbegriff

Eine erste Umschreibung

Robert Sedgewick und Kevin Wayne umschreiben in ihrem Buch über Algorithmen und Datenstrukturen[1] auf Seite 20 den Algorithmusbegriff wie folgt:

> „Ein Computerprogramm zu schreiben bedeutet im Allgemeinen nichts anderes, als ein Verfahren zu implementieren, das zuvor dafür entwickelt wurde, ein bestimmtes Problem zu lösen. Dieses Verfahren ist meistens unabhängig von der eingesetzten Programmiersprache – mit grosser Wahrscheinlichkeit dürfte es für viele Computer und viele Programmiersprachen geeignet sein. Und es ist das Verfahren und nicht das Computerprogramm, welches die Schritte vorgibt, mit denen wir das Problem lösen können.
>
> Lösungsverfahren, die *endlich*, *deterministisch* und *effektiv* sind und als Computerprogramme implementiert werden können, werden in der Informatik als *Algorithmen* bezeichnet. . . . “

- *endlich* bedeutet, dass die Beschreibung des Algorithmus' aus einer endlichen Menge von Zeichen besteht. Eine unendlich lange Beschreibung wäre auch nicht sinnvoll.
- *deterministisch* bedeutet, dass zu jedem Zeitpunkt der nächste Schritt des Algorithmus jeweils eindeutig bestimmt ist.
- *effektiv* bedeutet, dass die Wirkung jeder Anweisung eindeutig festgelegt ist; d.h. für deren Ausführung braucht es keinen Erfindergeist sondern nur stures Befolgen unmissverständlicher Regeln.

Historisches zum Algorithmusbegriff

Der Begriff Algorithmus ist eng mit dem Namen des Gelehrten Abu Abdallah Muhammed ibn Musa al-Hwarizimi al-Magusi (Al-Hwarizimi) verbunden, der um etwa 800 n. Chr. im „Haus der Weisheit“ in Bagdad tätig war. Das Haus der Weisheit war

[1] R. Sedgewick, K. Wayne: (2014) *Algorithmen.* 4. Auflage. Pearson.

eine Art Akademie, wo Gelehrte aus verschiedenen Kulturen unter anderem wissenschaftliche Werke der Antike (Platon, Aristoteles, Euklid, ...) vom Griechischen ins Arabische übersetzten.

Von Al-Hwarizimi stammen mehrere Mathematikbücher, in denen das aus Indien stammende dezimale Stellenwertsystem sowie das Rechnen damit beschrieben wird.

Originale von Al-Hwarizimi sind nicht überliefert. Dafür lateinsche Übersetzungen, in denen das Werk Buch des Algorithmus (oder auch Buch des Algorismus) genannt wird. Im Laufe der Zeit wurde der Name des Autors immer mehr zu einer Bezeichnung für das von ihm beschriebene Rechnen mit den arabischen (bzw. indischen) Ziffern. So wandelte sich der Begriff zur Bezeichnung für ein automatisierbares Rechenverfahren.

2 Der Algorithmus von Euklid

Geschichtliches

Beim euklidischen Algorithmus handelt sich um ein Verfahren, um den grössten gemeinsamen Teiler (ggT) zweier natürlicher Zahlen zu berechnen. Dieser Algorithmus wurde um etwa 300 v. Chr. von dem griechischen Mathematiker Euklid von Alexandria in seinem Werk *Die Elemente* beschrieben und daher nach ihm benannt. Man vermutet aber, dass das Verfahren vermutlich schon früher bekannt war.

Vorbereitungen

Bevor wir die klassische und die moderne Version des euklidischen Algorithmus besprechen, legen wir kurz einige Regeln zum Rechnen mit dem ggT fest. Dabei wird vorausgesetzt, dass a und b zwei beliebige nichtnegative ganze Zahlen sind.

(a) $\mathrm{ggT}(a, 0) = a$

(b) $\mathrm{ggT}(0, 0) = 0$

(c) $\mathrm{ggT}(a, b) = \mathrm{ggT}(b, a)$

Der klassische Algorithmus von Euklid

```
Eingabe: zwei nichtnegative ganze Zahlen a und b

while b != 0 {
  if a < b {
     vertausche a und b
  }
  a := a-b
  b := b
}

Ausgabe: a
```

Beispiel 1

$$\begin{aligned}\mathrm{ggT}(44,12) &= \mathrm{ggT}(32,12) = \mathrm{ggT}(20,12) = \mathrm{ggT}(8,12) = \mathrm{ggT}(12,8)\\ &= \mathrm{ggT}(4,8) = \mathrm{ggT}(8,4) = \mathrm{ggT}(4,4)\\ &= \mathrm{ggT}(0,4) = \mathrm{ggT}(4,0) = \mathrm{ggT}(4,0) = 4\end{aligned}$$

Beispiel 2

$$\begin{aligned}\mathrm{ggT}(99,2) &= \mathrm{ggT}(97,2) = \mathrm{ggT}(95,2) = \mathrm{ggT}(93,2) = \mathrm{ggT}(91,2)\\ &= \ldots\\ &= \mathrm{ggT}(3,2) = \mathrm{ggT}(1,2) = \mathrm{ggT}(2,1) = \mathrm{ggT}(1,1)\\ &= \mathrm{ggT}(0,1) = \mathrm{ggT}(1,0) = \mathrm{ggT}(1,0) = 1\end{aligned}$$

Problem: Der Algorithmus verbringt viel Zeit mit Subtrahieren und Vertauschen.

Der moderne Algorithmus von Euklid

```
Eingabe: zwei nichtnegative ganze Zahlen a und b

while b != 0 {
  r := a mod b
  a := b
  b := r
}

Ausgabe: a
```

Beispiel 3

$$\mathrm{ggT}(44,12) = \mathrm{ggT}(12,8) = \mathrm{ggT}(8,4) = \mathrm{ggT}(4,0) = 4$$

Beispiel 4

$$\mathrm{ggT}(99,2) = \mathrm{ggT}(2,1) = \mathrm{ggT}(1,0) = 1$$

Beispiel 5

$$\begin{aligned}\mathrm{ggT}(34,21) &= \mathrm{ggT}(21,13) = \mathrm{ggT}(13,8) = \mathrm{ggT}(8,5)\\ &= \mathrm{ggT}(5,3) = \mathrm{ggT}(3,2) = \mathrm{ggT}(2,1) = \mathrm{ggT}(1,0) = 1\end{aligned}$$

Bemerkung: Die für den modernen (und klassischen) euklidischen Algorithmus aufwändigsten Eingaben (Worst Case) bestehen aus zwei aufeinanderfolgenden Zahlen der Fibonacci-Folge:

1, 1, 2, 3, 5, 8, 13, ...

3 Laufzeitanalyse

Hat man einen Algorithmus implementiert, möchte man gerne wissen, wie gross der Zeit- und der Speicherbedarf für eine bestimmte Eingabe ist.

Da dies jedoch in den meisten Fällen sehr aufwändig oder gar unmöglich ist, begnügt man sich damit, den Zeit- oder den Speicheraufwand in Abhängigkeit der Anzahl der Eingabegrössen n auszdrücken.

Die Beschreibung der Laufzeit als Funktion $f(n)$ soll ...

- abhängig von der Anzahl n der Eingabdaten sein,
- unabhängig von der Programmiersprache oder der Hardware sein,
- den ungünstigsten Fall („worst case“) darstellen.

3.1 Die $\mathcal{O}$-Notation

Die $\mathcal{O}$-Notation (engl.: *Big-Oh-Notation*) ist ein Hilfsmittel zur mathematischen Beschreibung der Laufzeit eines Algorithmus.

Definition

$\mathcal{O}(f(n))$ ist die Menge aller Funktionen $g(n)$, für die es eine Konstante c und eine Schranke N gibt, so dass $g(n) \leq c \cdot f(n)$ für alle $n \geq N$.

oder etwas umgangssprachlicher:

$\mathcal{O}(f(n))$ ist die Menge aller Funktionen $g(n)$, die ab einem bestimmten n nicht schneller wachsen als die Funktion $c \cdot f(n)$, wobei c eine frei wählbare Kontante ist.

Beispiel 1

$g(n) = 10n$

$g(n) = 10n \leq 10 \cdot n \quad$ (für alle $n \geq 1$)

$\Rightarrow \quad g(n) \in \mathcal{O}(n)$

Beispiel 2

$g(n) = 3n + 2$

$g(n) = 3n + 2 \leq 3n + 2n = 5n \quad$ (für alle $n \geq 1$)

$\Rightarrow \quad g(n) \in \mathcal{O}(n)$

Beispiel 3

$g(n) = 2n^2$

$g(n) = 2n^2 \leq 2 \cdot n^2 \quad$ (für alle $n \geq 1$)

$\Rightarrow \quad g(n) \in \mathcal{O}(n^2)$

Beispiel 4

$g(n) = n^2 + 2n$

$g(n) = n^2 + 2n \leq n^2 + 2n^2 = 3n^2 \quad$ (für alle $n \geq 1$)

$\Rightarrow \quad g(n) \in \mathcal{O}(n^2)$

Beispiel 5

$g(n) = 5n^4 + 4n^3 + 3n^2 + 2n + 1$

$g(n) \leq 5n^4 + 4n^4 + 3n^4 + 2n^4 + n^4 = 15n^4 \quad$ (für alle $n \geq 1$)

$\Rightarrow \quad g(n) \in \mathcal{O}(n^4)$

Beispiel 6

$g(n) = 2^{n+1}$

$g(n) = 2^{n+1} = 2^n \cdot 2^1 = 2 \cdot 2^n \quad$ (für alle $n \geq 1$)

$\Rightarrow \quad g(n) \in \mathcal{O}(2^n)$

Rechenregeln

Ist $g_1(n) \in \mathcal{O}\big(f_1(n)\big)$ und $g_2(n) \in \mathcal{O}\big(f_2(n)\big)$, so gilt:

- $g_1(n) + g_2(n) \in \mathcal{O}\big(\max(f_1(n), f_2(n))\big)$
- $g_1(n) \cdot g_2(n) \in \mathcal{O}\big(f_1(n) \cdot f_2(n)\big)$

3.2 Wichtige Klassen von Laufzeitkomplexitäten

Konstante Laufzeit

$\mathcal{O}(1)$

- Wert in einer Liste lesen/schreiben
- Zwei Zahlen multiplizieren

Logarithmische Laufzeit

$\mathcal{O}(\log n)$

- Suche in einer sortierten Liste

Lineare Laufzeit

$\mathcal{O}(n)$

- Suche in einer unsortierten Liste

Log-Lineare Laufzeit

$\mathcal{O}(n \cdot \log n)$

- fortgeschrittene Sortieralgorithmen

Quadratische Laufzeit

$\mathcal{O}(n^2)$

- „naive“ Sortieralgorithmen

Kubische Laufzeit

$\mathcal{O}(n^3)$

- Matrizenmultiplikation

Polynomielle Laufzeit

$\mathcal{O}(n^p)$

- Simplex-Algorithmus (lineare Optimierung)

Exponentielle Laufzeit

$\mathcal{O}(2^n)$

- Rucksackproblem

Faktorielle Laufzeit

$\mathcal{O}(n!)$

- Problem des Handlungsreisenden

Bemerkung:

Algorithmen mit $\mathcal{O}(n^4)$ und höher benötigen bei einer Verdoppelung der Inputgrösse bereits die 16-fache Laufzeit, was problematisch ist. Algorithmen mit $\mathcal{O}(2^n)$ oder höher sind praktisch unbrauchbar.

Beispiel 8

Welche Laufzeitkomplexität hat das Python-Programmfragment?

```
1  s = 1
2  for i in range(0, n):
3      for j in range(0, n):
4          for k in range(0, n):
5              s = i + j + k
```

Zeile	Kosten	Anzahl
1	c_1	1
2	c_2	n
3	c_3	n^2
4	c_4	n^3
5	c_5	n^3

$T(n) = c_1 + c_2 n + c_3 n^2 + c_4 n^3 \in \mathcal{O}(n^3)$