$\mathcal{O}$-Notation und andere Landau-Symbole*

Hans Jürgen Ohlbach

23. März 2018

Keywords: $\mathcal{O}$-Notation für Komplexität von Algorithmen, Landau-Symbole

Empfohlene Vorkenntnisse: Schulmathematik, Grenzwerte

Inhaltsverzeichnis

*Dieser Text ist Teil einer Sammlung von Miniskripten zur Einführung in die Informatik. Er ist in erster Linien für Nichtinformatiker gedacht, kann aber natürlich auch als erste Einführung für Informatiker nützlich sein.

1 Einleitung

Algorithmen werden geschrieben, um nicht nur ein Problem, sondern eine ganze Klasse von Problemen zu lösen. Z.B. lernt man in der Schule wie man zwei Zahlen addiert oder multipliziert. Die Beobachtung, die jedes Schulkind dabei macht ist: je mehr Ziffern die beiden Zahlen haben, umso länger dauert es bis das Ergebnis fertig ist. Allerdings unterscheiden sich die Algorithmen für die Addition und die Multiplikation hinsichtlich der Anzahl von Rechenschritten: *verdoppelt* man die Anzahl von Ziffern in den beiden Zahlen, dann *verdoppelt* sich die Anzahl von Rechenschritten bei der Addition, aber für die Multiplikation *vervierfacht* sich die Anzahl von Rechenschritten.

Wenn n die Anzahl von Ziffern in der Eingabe ist, dann haben wir die Beziehung:

Addition:	Eingabelänge verdoppelt $\rightarrow$ Rechenschritte verdoppelt	$n \rightarrow n$
Multiplikation:	Eingabelänge verdoppelt $\rightarrow$ Rechenschritte vervierfacht	$n \rightarrow n^2$

Informatiker würden das so ausdrücken:

Zeitkomplexität:
Addition $\in \mathcal{O}(n)$ und
Multiplikation $\in \mathcal{O}(n^2)$

Die Gesamtzeit für die Ausführung eines Algorithmus korrespondiert mit der Anzahl seiner Rechenschritte. Ist der Zeitverbrauch für jeden Rechenschritt gleich, dann ergibt sich die Gesamtzeit eines Algorithmus aus der Anzahl von Rechenschritten mal der Zeit für einen einzelnen Rechenschritt. Die konkrete Zeit für einen einzelnen Rechenschritt ist für die Bewertung eines Algorithmus allerdings wenig relevant. Man würde ja auch nicht das Verfahren zur Multiplikation zweier Zahlen danach beurteilen, wie lange der langsamste Schüler in der Klasse dafür benötigt. Ähnlich ist es bei Computern. Die Dauer eines Rechenschrittes verkürzt sich meist mit jeder neuen Prozessorgeneration. Daher ist das keine relevante Größe für die Beurteilung eines Algorithmus. Relevant ist nur die *Anzahl* von Rechenschritten, in Abhängigkeit von der Größe der Eingabe. Die Größe der Eingabe bemisst sich dabei aus der *Größe der Binärdarstellung* der Eingabe. Für Addition und Multiplikation ist daher nicht die arithmetische Größe der Zahlen relevant, sondern die Anzahl von Ziffern (bei Computern die Anzahl der Bits in der Binärdarstellung der Zahlen). Addiert an z.B. 1+2, dann hat man den gleichen Aufwand wie bei 2+4, da die Zahlen ja gleich viele Ziffern haben.

Man bezeichnet die Angaben über die Anzahl von Rechenschritten als die *Zeitkomplexität* eines Algorithmus.

Für Computer ist darüber hinaus wichtig, wieviel Speicherplatz der Algorithmus benötigt, ebenfalls in Abhängigkeit von der Größe der Eingaben. Das ist die *Platzkomplexität*.

Für die Addition ergibt sich dabei: Verdoppelung der Eingabelänge $\rightarrow$ Verdoppelung des Platzbedarfs. Die Multiplikation kann man platzmäßig optimieren, indem man nicht am Ende addiert, sondern alle Zwischenergebnisse sofort addiert, so dass man den Platz wiederverwenden kann. Damit ergibt sich ebenfalls: Verdoppelung der Eingabelänge $\rightarrow$ Verdoppelung des Platzbedarfs. In beiden Fällen würde man daher schreiben:

Platzkomplexität:
Addition $\in \mathcal{O}(n)$ und ebenfalls
Multiplikation $\in \mathcal{O}(n)$

In diesem Miniskript werden die Landau-Symbole und deren Bedeutung eingeführt. Es geht *nicht* um die Komplexitätsanalyse von Algorithmen oder Problemen, wo diese Symbole gebraucht werden.

2 $\mathcal{O}$-Notation (engl. Big O-Notation)

Die $\mathcal{O}$-Notation wurde erstmals 1894 von Paul Bachmann eingeführt, aber erst später von Edmund Landau bekannt gemacht. Daher bezeichnet man $\mathcal{O}$ als eines der *Landau-Symbole*. $\mathcal{O}(f(n))$ charakterisiert das *Wachstum* einer Funktion. Ob es sich dabei um Zeit- oder Platzbedarf, oder etwas ganz anderes handelt, hängt von der Anwendung ab. $\mathcal{O}(f(n))$ bezieht sich lediglich auf die Funktion selbst.

Informatiker sind in erster Linie daran interessiert, wie sich ein Algorithmus verhält, wenn die Eingabe sehr groß wird. D.h. $\mathcal{O}(f(n))$ charakterisiert das Wachstumsverhalten für $n \to \infty$. (In der Mathematik gibt es auch eine Version, wo $\mathcal{O}(f(n))$ das Wachstum einer Funktion charakterisiert, wenn n gegen einen konkreten Wert a strebt.)

Betrachten wir die beiden Funktionen $f_1(n) = n^2$ und $f_2(n) = 2n^2$. Die Funktionswerte sind zwar unterschiedlich, aber das Wachstumsverhalten für große n ist gleich: n verdoppelt $\to f_i(n)$ vervierfacht. Daher spielt diese Konstante 2 keine Rolle für das Wachstumsverhalten. In beiden Fällen hätte man: $f_1 \in \mathcal{O}(n^2)$ und $f_2 \in \mathcal{O}(n^2)$.

Mit folgenden alternativen Definitionen für $\mathcal{O}$ erhält man dieses Verhalten:

$$f(n) \in \mathcal{O}(g(n)) \quad \text{gdw.} \quad \exists k > 0 \; \exists n_0 \; \forall n > n_0 \; |f(n)| \leq k \cdot g(n) \tag{1}$$

$$f(n) \in \mathcal{O}(g(n)) \quad \text{gdw.} \quad \limsup_{n \to \infty} \frac{|f(n)|}{g(n)} < \infty \tag{2}$$

Definition (1) besagt, dass $f(n)$ ab einem bestimmten Wert n_0 nicht stärker wächst als $k \cdot g(n)$, mit einer Konstanten k.

Beispiele
$2n^2 \in \mathcal{O}(n^2)$, wobei $k = 2$ ist, und $n_0 = 0$.
$-2n^2 \in \mathcal{O}(n^2)$, da es nur um den Betrag der Funktion f geht.
$2n^2 + 3n \in \mathcal{O}(n^2)$, wobei $k = 3$ ist, und $n_0 = 3$.
Ab $n = 3$ wächst $3n^2$ stärker als $2n^2 + 3n$.

Die Definition (2) bezieht sich direkt auf das Grenzverhalten wenn $n \to \infty$ strebt.

Das $\sup$ in $\limsup$ wird relevant wenn die Funktion f oszilliert. Dann betrachtet man das Grenzverhalten der Abfolge der jeweiligen Maxima der Funktion.

Beispiele ohne Oszillation
$2n^2 \in \mathcal{O}(n^2)$, da $\lim_{n\to\infty} \frac{2n^2}{n^2} = 2 < \infty$ ist.
$2n^2 + 3n \in \mathcal{O}(n^2)$, da $\lim_{n\to\infty} \frac{2n^2+3n}{n^2} = \lim_{n\to\infty}(2 + \frac{3}{n}) = 2 < \infty$ ist.
Ist $f(n)$ ein Polynom: $f(n) = a_m n^m + a_{m-1}n^{m-1} + \ldots + a_0$ dann ist
$f(n) \in \mathcal{O}(n^m)$, da $\lim_{n\to\infty} \frac{a_m n^m + a_{m-1}n^{m-1}+\ldots+a_0}{n^m} = \lim_{n\to\infty}(a_m + \frac{a_{m-1}}{n} + \ldots + \frac{a_0}{n^m}) = a_m < \infty$.

Allerdings ist auch
$n^2 \in \mathcal{O}(n^3)$, da $\lim_{n\to\infty} \frac{n^2}{n^3} = 0 < \infty$.

Beispiel mit Oszillation
$(2n^2 + 3n)\sin(n) \in \mathcal{O}(n^2)$, da $\limsup_{n\to\infty} \frac{(2n^2+3n)\sin(n)}{n^2} = \lim_{n\to\infty} \frac{2n^2+3n}{n^2} = 2 < \infty$ ist.

Intuitive Definition von $\mathcal{O}$:
$\mathcal{O}(g(n))$ bezeichnet die Menge aller Funktionen, die für große n nicht stärker wachsen als $k \cdot g(n)$, für eine feste Konstante k.

Das Beispiel $n^2 \in \mathcal{O}(n^3)$, bedeutet jedoch, dass $\mathcal{O}$ nur eine obere Schranke angibt. $f(n) \in \mathcal{O}(g(n))$ gilt auch dann, wenn $f(n)$ deutlich langsamer wächst als $g(n)$.

Für Informatiker ist das nützlich, wenn man bei der Analyse der Zeit- oder Platzkomplexität eines Algorithmus oder eines Problems nur eine obere Schranke angeben kann: „Das Wachstum ist nicht schlimmer als $\mathcal{O}(g(n))$, könnte sich aber bei genauerer Analyse als besser herausstellen“.

Zahldarstellung:
Die Zeit- und Platzkomplexität eines Algorithmus bemisst sich in Abhängigkeit von der Größe der Eingabe. Besteht die Eingabe aus ganze Zahlen, dann rechnen Computer ja mit der Binärdarstellung von Zahlen. D.h. relevant ist die Anzahl der Bits in der Binärdarstellung von Zahlen. Für Zahlen von 0 bis n braucht man $\lceil \log_2(n+1) \rceil$[1] Bits für deren Binärdarstellung. Z.B. für die Zahlen 0,1,2,3 braucht man $\log_2(4) = 2$ Bits. Für die Zahlen von 0 bis 1023 braucht man $\log_2(1024) = 10$ Bits. Für die Zahlen von 0 bis 1000 braucht man auch $\lceil \log_2(1000) \rceil = 10$ Bits.

Geht es also um die Komplexität von Algorithmen, die mit ganzen Zahlen arbeiten, kommt daher oft ein logarithmische Faktor ins Spiel, der von der Binärdarstellung der Zahlen herrührt.

[1] $\lceil x \rceil$ ist die kleinste ganze Zahl $\geq x$.

3 Θ-Notation

Die Angabe oberer Schranken mit der $\mathcal{O}$-Notation ist zwar oft schon hilfreich. Besser ist es jedoch, wenn man das Wachstumsverhalten noch genauer charakterisieren kann. Dafür eignet sich die Θ-Notation:

Auch hierfür gibt es zwei alternative Definitionen:

$$f(n) \in \Theta(g(n)) \quad \text{gdw.} \quad \exists k_1, k_2 > 0 \; \exists n_0 \; \forall n > n_0 : \; k_1 \cdot g(n) \leq |f(n)| \leq k_2 \cdot g(n) \tag{3}$$

$$f(n) \in \Theta(g(n)) \quad \text{gdw.} \quad 0 < \liminf_{n\to\infty} \frac{|f(n)|}{g(n)} \leq \limsup_{n\to\infty} \frac{|f(n)|}{g(n)} < \infty \tag{4}$$

Die erste Definition besagt, dass ab einer Zahl n_0, $f(n)$ in einem Korridor um $g(n)$ eingesperrt ist, wobei die untere Grenze des Korridors k_1 nicht 0 sein darf, was bei $\mathcal{O}(g(n))$ erlaubt wäre.

Die zweite Definition berücksichtigt explizit den Oszillationsfall. Ohne Oszillation reduziert sich die Bedingung auf $0 < \lim_{n\to\infty} \frac{|f(n)|}{g(n)} < \infty$. Der Unterschied zur Definition von $\mathcal{O}$ ist die Zusatzbedingung $0 < \ldots$, welche z.B. $n^2 \in \Theta(n^3)$ ausschließt.

Mit Oszillation wird durch $\liminf$ und $\limsup$ der Grenzwert für den jeweils kleinsten und größten Wert betrachtet.

4 Weitere Landau-Symbole

Weitere Landau-Symbole sind mehr in mathematischen Anwendungen nützlich.

In der folgenden Tabelle kann $a = \infty$ sein. Dann bezeichnet man das Verhalten wenn x gegen Unendlich strebt ($\lim_{x\to\infty}$). Wenn a endlich ist, dann bezeichnet man damit das Verhalten wenn x gegen diesen Wert a strebt.

Notation	Bedeutung	Definition
$f \in o(g)$	f wächst viel schwächer als g	$\lim_{x\to a} \lvert\frac{f(x)}{g(x)}\rvert = 0$
$f \sim g$	f nähert sich g	$\lim_{x\to a} \lvert\frac{f(x)}{g(x)}\rvert = 1$
$f \in \omega(g)$	f wächst viel schneller als g	$\lim_{x\to a} \lvert\frac{f(x)}{g(x)}\rvert = \infty$
$f \in \Omega_+(g)$	f wächst oben schneller als g	$\limsup_{x\to a} \lvert\frac{f(x)}{g(x)}\rvert > 0$
$f \in \Omega_-(g)$	f wächst unten schneller als g	$\liminf_{x\to a} \lvert\frac{f(x)}{g(x)}\rvert > 0$

$f \in o(g)$ bedeutet, dass im Grenzfall f vernachlässigbar kleiner als g ist.
Bei $f \in \omega(g)$ ist es gerade umgekehrt, im Grenzfall ist g vernachlässigbar kleiner als f.

Die Notation $f \in \Omega_+(g)$ und $f \in \Omega_-(g)$ bezieht sich bei oszillierenden Funktionen auf den oberen bzw. unteren Grenzwert. (Ω_+ Ω_- ist keine standardisierte Notation. Bei unterschiedlichen Autoren findet man da auch unterschiedliche Schreibweisen).

Alternative Notationen
$\mathcal{O}(g(n))$, oder vereinfacht $\mathcal{O}(g)$, bezeichnet *eine Menge* von Funktionen, nämlich die Funktionen, die asymptotisch nicht schneller wachsen als g.

Trotzdem finden man in vielen Texten $f = \mathcal{O}(g)$ anstelle von $f \in \mathcal{O}(g)$.
Es ist aber auch hierbei immer noch $f \in \mathcal{O}(g)$ gemeint!

Analoge Schreibweisen findet man auch für die anderen Landau-Symbole.

Die $\mathcal{O}$-Notation erweist sich auch hilfreich, wenn es um Funktionen geht, bei denen das stärkste Wachstum wichtig ist, und weniger stark wachsende Beiträge vernachlässigt werden können. Z.B. könnte man schreiben:

$$f(n) = 2n^4 + 5n^3 + \mathcal{O}(n^2)$$

Das bedeutet, die am stärksten wachsenden Glieder sind $2n^4 + 5n^3$. Dann kommt noch ein Rest, der zwar quadratisch wächst, aber gegenüber den anderen Gliedern für große Zahlen von n trotzdem vernachlässigbar ist.

Hierfür kann es sinnvoll sein, dass z.B. auch $h(n) = 2n \in \mathcal{O}(n^2)$ ist. Mit $f(n) = 2n^4 + 5n^3 + \mathcal{O}(n^2)$ möchte man nur ausdrücken, dass der Rest nach $2n^4 + 5n^3$ zu vernachlässigen ist, egal ob er quadratisch oder linear wächst.

5 Sprechweisen

Anstelle der mathematischen Notationen $\mathcal{O}(f(n))$ findet man in vielen Texten sprachliche Begriffe, die aber die entsprechende präzise Definition haben.

mathematisch	sprachlich
$\mathcal{O}(1)$	konstant
$\mathcal{O}(\log\log n)$	doppelt logarithmisch
$\mathcal{O}(\log n)$	logarithmisch
$\mathcal{O}((\log n)^c)$, $c > 1$	polylogarithmisch
$\mathcal{O}(n^{\frac{1}{c}})$	c-te Wurzel
$\mathcal{O}(n)$	linear
$\mathcal{O}(n \log^* n)$	n log-star n[2]
$\mathcal{O}(n \log n)$	quasilinear
$\mathcal{O}(n^2)$	quadratisch
$\mathcal{O}(n^3)$	kubisch
$\mathcal{O}(n^c)$	polynomiell
$\mathcal{O}(c^n)$, $c > 1$	exponentiell
$\mathcal{O}(n!)$	fakultätsmäßig

Für all diese Funktionen gibt es Algorithmen, deren Zeitkomplexität in dieser Klasse liegt.

[2] $\log^*(n) = \begin{cases} 0 & \text{falls } n \leq 1 \\ 1 + \log^*(\log(n)) & \text{sonst} \end{cases}$

Bei logarithmischer Komplexität ist es übrigens egal, zu welcher Basis der Logarithmus genommen wird. Wegen

$$\log_b(x) = \frac{\log_a(x)}{\log_a(b)}$$

führt die Umrechnung nur einen konstanten Faktor ein, und konstante Faktoren sind für die $\mathcal{O}$-Notation irrelevant. Z.B. ist $\log_{10}(x) = \frac{\log_2(x)}{\log_2(10)} \sim 0.3 \log_2(x)$.

6 Erweiterung auf Funktionen mehrerer Variablen

Eine Verallgemeinerung auf Funktionen mehrerer Variablen lässt sich direkt hinschreiben, wenn man die Vektornotation benutzt.

eindimensional	$f(n) \in \mathcal{O}(g(n))$	gdw.	$\exists k > 0\, \exists N\, \forall n > N :\ \lvert f(n)\rvert \leq k \cdot g(n)$
mehrdimensional	$f(\vec{n}) \in \mathcal{O}(g(\vec{n}))$	gdw.	$\exists k > 0\, \exists \vec{N}\, \forall n_i > N_i :\ \lvert f(\vec{n})\rvert \leq k \cdot g(\vec{n})$
	$f(\vec{n}) \in \mathcal{O}(g(\vec{n}))$	gdw.	$\limsup_{n_i \to \infty} \frac{\lvert f(\vec{n})\rvert}{g(\vec{n})} < \infty$

Um den Grenzwert zu bilden, muss man alle Variablen gegen Unendlich gehen lassen.

Ganz analog geht es für die anderen Landau-Symbole.

7 Ausblick

Die Landau-Symbole verwendet man in der Informatik zunächst, um die Zeit- und Platzkomplexität eines *konkreten* Algorithmus in Abhängigkeit der Größe der Eingabe anzugeben. In der Komplexitätstheorie betrachtet man aber nicht *einzelne* Algorithmen, sondern ganze Problemklassen. Hier möchte man wissen: was ist denn die Zeit- und Platzkomplexität des *bestmöglichen* Algorithmus, der alle Probleme einer Klasse löst?

Z.B. gibt es vergleichsbasierte Sortieralgorithmen, die eine Liste von n Objekten in $\mathcal{O}(n^2)$ Zeit sortieren. Man kann aber zeigen, dass der *bestmögliche* Algorithmus die Liste in $\mathcal{O}(n \log(n))$ Zeit sortiert. Daher sagt man: Sortierproblem $\in \mathcal{O}(n \log(n))$ als Kurzschreibweise für „Die Zeitkomplexität des bestmöglichen Sortieralgorithmus liegt in $\mathcal{O}(n \log(n))$“.