AG Betriebssysteme FB3
Kirsten Berkenkötter

Asymptotische Komplexität

1 Wieso, weshalb, warum

Um die Effizienz von Algorithmen beurteilen zu können, muss zunächst ein passendes Maß gefunden werden. Dazu wird die asymptotische Komplexität verwendet, mit deren Hilfe der zeitliche Aufwand eines Algorithmus in Abhängigkeit von der Problemgröße n abgeschätzt werden kann. Die Problemgröße n ist z.B. die Größe einer Eingabeliste oder die Größe eines Eingabewertes.

Die Abschätzung beruht auf der Häufigkeit der elementaren Operationen wie Speicherzugriffen, Additionen, Multiplikationen, etc. Diese kann man üblicherweise nicht exakt bestimmen, aber in der Größenordnung abschätzen. Unterschieden wird häufig zwischen dem Aufwand eines Algorithmus im schlechtesten und im mittleren Fall.

2 Definition

Die asymptotische Komplexität ist folgendermaßen definiert:

$f(n) = O(g(n)) \Leftrightarrow \exists c, n_0 \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
mit $f, g : \mathbb{N} \rightarrow \mathbb{N}$

$O(g(n))$ ist die asymptotische obere Schranke für $f(n)$. Sie beschreibt die Wachstumsgeschwindigkeit von $f(n)$. Aus dieser Bezeichung leitet sich der Begriff *O-Notation* für die asymptotische Analyse ab. Dabei ist $g(n)$ die Vergleichsfunktion, um das Wachstum von $f(n)$ zu bestimmen. c ist ein konstanter Faktor, um die beiden Funktionen $f(n)$ und $g(n)$ vergleichbar zu machen, es gilt $\frac{f(n)}{g(n)} = c$. Uns interessiert hier nicht das Verhältnis der konkreten Funktionen f und g, sondern die Tatsache, welche dieser Funktionen schneller wächst.

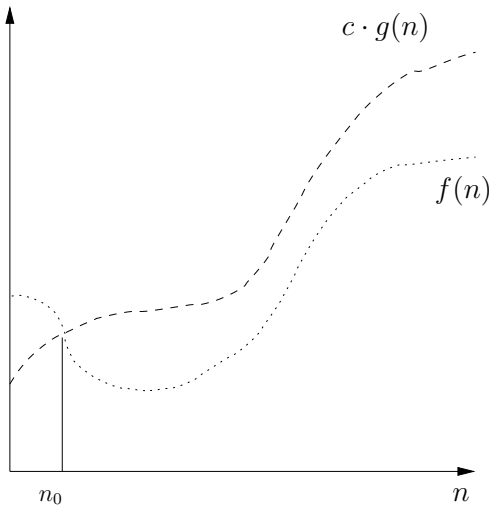

Abbildung 1: Asymptotische Komplexität

Dies muss nicht für alle $n \in \mathbb{N}$ gelten. Interessant ist vor allem das Verhalten für "große" n, da der Aufwand eines Algorithmus hier besonders gering sein sollte. Es gibt durchaus Algorithmen, die bei wenigen Eingaben sehr effizient, bei großen Eingabemengen aber ineffizient sind. n_0 wird so gewählt, dass für alle $n \geq n_0$ das Wachstum von $g(n)$ stärker als das von $f(n)$

ist. Aus diesem Grund sprechen wir von der asymptotischen Analyse. Eine Asymptote ist eine Gerade, an die eine Kurve sich für $n \to \infty$ immer stärker annähert. Durch die asymptotische Komplexität drücken wir aus, dass $f(n)$ asymptotisch nicht schneller wächst als $g(n)$.

Dies lässt sich auch gut in der graphischen Darstellung erkennen (s. Abbildung 1).

Häufig wird $f(n) = O(g(n))$ geschrieben, um auszudrücken, dass $f(n)$ zu einer Komplexitätsklasse gehört. Korrekt ist eigentlich $f(n) \in O(g(n))$, da $f(n)$ zu einer Menge von Funktionen gehört, die durch $g(n)$ beschränkt sind. Die Schreibweise mit dem $=$ hat sich allerdings eingebürgert und wird darum auch hier verwendet.

3 Typische Komplexitätsklassen

Die folgenden Komplexitätsklassen werden häufig verwendet, um Algorithmen miteinander zu vergleichen:

O(1)	konstanter Aufwand
O($\log n$)	logarithmischer Aufwand
O(n)	linearer Aufwand
O($n \log n$)	
O(n^2)	quadratischer Aufwand
O(n^k) für $k \geq 0$	polynomialer Aufwand
O(2^n)	exponentieller Aufwand

Um ein Gefühl dafür zu bekommen, was diese Komplexitätsklassen bedeuten, betrachten wir einmal den Aufwand für diese Klassen mit verschieden großen n:

f(n)	**n=**	**2**	**16**	**256**	**1024**	**1048576**
$\lg_2 n$		1	4	8	10	20
n		2	16	256	1024	1048578
$n \cdot \lg_2 n$		2	64	2048	10240	20971520
n^2		4	256	65536	1048576	$\approx 10^{12}$
n^3		8	4096	16777200	$\approx 10^9$	$\approx 10^{18}$
2^n		4	65536	$\approx 10^{77}$	$\approx 10^{308}$	$\approx 10^{315653}$

Wenn wir nun annehmen, dass ein Schritt $1\mu s$ entspricht, kommen wir auf folgende Zeiten für die Ausführung eines Algorithmus aus der jeweiligen Klasse in Abhängigkeit von der Eingabegröße n:

f(n)	**n=**	**2**	**16**	**256**	**1024**	**1048576**
$\lg_2 n$		$1\mu s$	$4\mu s$	$8\mu s$	$10\mu s$	$20\mu s$
n		$2\mu s$	$16\mu s$	$256\mu s$	$\approx 1ms$	$\approx 1s$
$n \cdot \lg_2 n$		$2\mu s$	$64\mu s$	$\approx 2ms$	$\approx 10,2ms$	$\approx 21s$
n^2		$4\mu s$	$256\mu s$	$\approx 65,5ms$	$\approx 1s$	$\approx$ 12Tage
n^3		$8\mu s$	$\approx 4,1ms$	$\approx 16,7s$	$\approx 18min$	$\approx 37 \cdot 10^3$Jahre
2^n		$4\mu s$	$\approx 65,5ms$	$\approx 3 \cdot 10^{63}$Jahre	$\approx 3 \cdot 10^{294}$Jahre	extrem viele Jahre

4 Regeln

- Summenregel: $O(f(n) + g(n)) = O(max(O(f(n)), O(g(n))))$
- Produktregel:$O(f(n) \cdot g(n)) = O(f(n)) \cdot O(g(n))$
- $O(f(n)) \subseteq O(g(n)) \Leftrightarrow f(n) \in O(g(n))$
- $O(f(n)) = O(g(n)) \Leftrightarrow f(n) \in O(g(n)) \wedge g(n) \in O(f(n))$
- $O(f(n)) \subset O(g(n)) \Leftrightarrow f(n) \in O(g(n)) \wedge g(n) \notin O(f(n))$

Aus der Summenregel folgt, dass jeweils nur der höchste konstante Faktor eine Rolle für die Komplexität eines Algorithmus spielt, also z.B.
$O(n^3 + n) = O(n^3)$

Aus der Produktregel folgt, dass konstante Faktoren keine Rolle für die Komplexität eines Algorithmus spielen, also z.B.
$O(\frac{n^2}{2}) = O(n^2)$

5 Bestimmung der Komplexität

Um die Komplexität eines Algorithmus bestimmen zu können, müssen die verschiedenen Anweisungen und Kontrollstrukturen eines Programms betrachtet werden. Wir konzentrieren uns hier auf nicht-rekursive Strukturen, da deren Behandling deutlich aufwändiger ist. Die beschriebenen Regeln beziehen sich auf die Abschätzung des schlechtesten Aufwands. Für die Abschätzung des mittleren Aufwands müsste man z.B. bei Schleifen die mittlere Anzahl der Durchläufe oder bei Verzweigungen die Wahrscheinlichkeit der Ausführung miteinbeziehen.

Einfache Anweisungen

Einfache Anweisungen wie z.B. das Lesen und Schreiben von Variablen, Vergleiche, etc. haben die Komplexität $O(1)$, da sie stets einen konstanten Zeitaufwand haben und nicht von der Eingabegröße n abhängen.

Sequentielle Ausführung

Die Komplexität von sequentiell ausgeführten Programmstücken p_1, p_2 wird nach der Summenregel abgeschätzt, d.h.
$O_{sequenz} = O(max(O(p_1), O(p_2)))$.

Verzweigungen

Bei Verzweigungen müssen jeweils die Komplexität der Bedingung b sowie die Komplexität der einzelnen verzweigten Programmstücke $p_1, \ldots, p_n$ betrachtet werden. Die Gesamtkomplexität der Schleife ergibt sich dann aus der sequentiellen Ausführung der Bedingung und der Verzweigung mit der größten Komplexität:
$O_{verzweigung} = O(max(O(b), O(max(O(p_1), \ldots, O(p_n)))))$.

Schleifen

Bei Schleifen muss zunächst die Komplexität eines Schleifendurchlaufs bestimmt werden. Diese ist durch die sequentielle Ausführung der Bedingung b sowie des Schleifenrumpfs r bestimmt: $O_{durchlauf} = O(max(O(b), O(r)))$. Dann muss überprüft werden, wie häufig die Schleife im schlechtesten Fall ausgeführt wird. Die Gesamtkomplexität ergibt sich dann aus dem Produkt der Schleifendurchläufe x und der Schleifenkomplexität:
$O_{schleife} = O(x \cdot O_{durchlauf})$.

Methoden

Bei (nicht-rekursiven) Methodenaufrufen wird zunächst die Komplexität der Methode bestimmt. Für den Aufruf gilt dann die Regel für die sequentielle Ausführung.

Quellen

1. Gunter Saake, Kai-Uwe Sattler: Algorithmen und Datenstrukturen, dpunktVerlag, 2004
2. Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman: Data Structures and Algorithms, Addison-Wesley, 1983