Thema: Zeitkomplexität (1)

1 Einführung

Die Komplexitätstheorie versucht, algorithmische Probleme (formalisiert als Sprachen) nach dem Bedarf an Berechnungsressourcen bei ihrer Lösung (dies sind i.A. Rechenzeit und Speicherplatz) zu klassifizieren. Man versucht z.B. für konkrete Probleme möglichst effiziente Verfahren anzugeben, um diese gemäß ihres Rechenaufwands analysieren und vergleichen zu können. Dies liefert sowohl Algorithmen zur Lösung als auch prinzipielle oberere Schranken für die Komplexität der Probleme. Die Komplexität wird meist durch die *O-Notation* angegeben, die eine asymptotische Abschätzung der Laufzeit des Algorithmus in Abhängigkeit von der Länge der Eingabe n liefert.

Als Grundlage für geräteunabhängige Untersuchungen dienen formale Maschinenmodelle, wie etwa die *Turingmaschine*. Die Zeitkomplexität (oder Laufzeit) einer Turingmaschine, die auf ein beliebiges Wort der Länge n angesetzt wird, ist definiert als die Anzahl der Rechenschritte, die sie zur Berechnung des Entscheidungsproblems (zum Entscheiden, ob das Wort in der Sprache ist) oder zur Berechnung einer Funktion auf der Eingabe höchstens benötigt.

Eine weitere Fragestellung in der Komplexitätstheorie bezieht sich auf die Vergleichbarkeit verschiedener Maschinenmodelle, insbesondere von deterministischen mit nichtdeterministischen. Dazu später mehr.

2 Komplexitätsklassen

Man teilt in der Komplexitätstheorie die entscheidbaren Sprachen in verschiedene Klassen bezüglich des Zeitbedarfs des verwendeten Maschinenmodells ein. Im Folgenden ist das zugrunde liegende Modell die Turingmaschine (könnte auch die Registermaschine o.a. sein).

Definition: Sei $t : \mathbb{N} \to \mathbb{R}^+$, dann ist

$$\textsc{Time}(t(n))$$

die Klasse aller Sprachen, die durch eine $O(t(n))$-zeitbeschränkte Turingmaschine entscheidbar sind. So enthält z.B. $\textsc{Time}(n^2)$ alle Sprachen, die in $O(n^2)$ Zeit entschieden werden können.

2.1 Die Komplexität hängt vom Maschinenmodell ab

Die *Church-Turing-These* besagt, dass die Klasse der Turing-berechenbaren Funktionen genau der Klasse der intuitiv berechenbaren Funktionen entspricht. Das impliziert die Äquivalenz aller Berechenbarkeitsmodelle, d.h. sie erkennen bzw. entscheiden alle die selbe Sprachklasse.

In der Komplexitätstheorie beeinflusst die Wahl des Maschinenmodells aber die Zeitkomplexität der Sprachen. So müssen Sprachen, die auf einem Modell in linearer Zeit entscheidbar sind, auf einem anderen Modell nicht unbedingt auch in linearer Zeit entscheidbar sein.

Wenn man z.B. Einband- und Mehrband-Turingmaschinen vergleicht, kann man folgendes feststellen: Eine Einband-Turingmaschine „verschwendet“ viel Rechenzeit durch das ständige hin und her Bewegen des Kopfes auf dem Band zum Finden der Informationen, ohne dass die Schritte direkt zur Lösung des Problems beitragen. Diese Rechenzeitanteile sind also nur dem eingeschränkten Maschinenmodell zuzuschreiben. Wir wissen aber, dass eine Mehrband-Turingmaschine, die das selbe Problem oftmals in weniger Schritten lösen kann, durch eine äquivalente Einband-Turingmaschine simuliert werden kann.

Zur Erinnerung: Simulation einer Mehrband-Turingmaschine

Eine k-Band-Turingmaschine M_k kann durch eine Einband-Turingmaschine M_1 mit $2k$ Spuren simuliert werden. Dabei ist auf k Spuren die momentane Kopfposition von M_k markiert und auf den anderen k Spuren steht der Inhalt der k Bänder. Zusätzlich besitzt die endliche Kontrolle einige Register, um den aktuellen Zustand, die Zeichen an der Position der Köpfe und die Richtungen, in die sich die Köpfe im nächsten Schritt bewegen sollen, zu speichern.

Satz: Sei $t(n)$ eine Funktion mit $t(n) \geq n$. Dann gibt es für jede $t(n)$-zeitbeschränkte Mehrband-Turingmaschine eine äquivalente $O(t^2(n))$-zeitbeschränkte Einband-Turingmaschine.

Beweis: Wir analysieren einfach, wie viel zusätzliche Zeit die äquivalente Einband-Turingmaschine M_1 im Vergleich zur k-Band-Turingmaschine M_k benötigt, wobei wir annehmen, dass M_k in $t(n) \geq n$ Zeit läuft (M_k könnte in weniger Schritten ja nicht einmal die ganze Eingabe lesen).

M_1 simuliert einen Schritt von M_k folgendermaßen

1. Aufsammeln: M_1 läuft über ihr Band und für jede markierte Kopfposition von M_k speichert sie das sich darüber befindliche Zeichen A_j im j-ten Register der endlichen Kontrolle und merkt sich den Zustand q ebenfalls in einem Register.

2. Simulation eines Übergangs: M_1 läuft eine weiteres Mal über das Band und überschreibt dabei k Zeichen und aktualisiert k Kopfpositionen gemäß der Übergangsfunktion von M_k.

Für jeden Schritt von M_k läuft M_1 also offensichtlich zweimal über das Band - einmal zum Sammeln der Informationen für den nächsten Schritt und einmal, um diesen auszuführen. M_k benötigt maximal $t(n)$ Schritte, um über ihre k Bänder zu laufen und M_1 kann daher für ihr Band auch nicht länger brauchen. Somit ist $O(t(n))$ eine obere Schranke für das Überstreichen des Bandes von M_1. Das M_1 stets zweimal pro Schritt von M_k über das Band läuft, verändert die obere Schranke nicht.
Für die Gesamtzeit ergibt sich: M_1 simuliert jeden der $t(n)$ Schritte von M_k durch $O(t(n))$ Schritte und braucht damit

$$t(n) \cdot O(t(n)) = O(t^2(n))$$

Schritte. Damit wäre der Satz bewiesen. □

Eine gute Idee wäre es demnach, die Klassifizierung von Sprachen unabhängig von einem bestimmten Maschinenmodell vornehmen zu können, wobei man durch Quadrieren der Laufzeit nicht aus dieser Klasse heraus fällt. Dies leistet die Klasse der Polynome - *die Klasse P*.

2.2 Die Klasse P

Die Klasse P enthält alle die Probleme, die praktisch lösbar sind: Es existieren i.A. effiziente Algorithmen, um Lösungen mit vertretbarem zeitlichen Aufwand zu berechnen. Der zeitliche Aufwand wächst hierbei für die Probleme der Klasse P maximal polynomial.

Definition: P ist die Klasse von Sprachen, die in polynomieller Zeit durch eine *deterministische* Einband-Turingmaschine entscheidbar sind.

$$P = \bigcup_k \textsc{Time}(n^k)$$

Das bedeutet auch, dass alle Maschinenmodelle, die polynomiell äquivalent zur deterministischen Einband-Turingmaschine sind (z.B. Mehrband-Turingmaschinen), die Sprachen aus P ebenfalls in polynomieller Zeit entscheiden können. Es kommt nun also *nicht* mehr auf das gewählte Maschinenmodell an, mithilfe dessen man die Zeitkomplexität einer Sprache bestimmt.
Nur der Determinismus des verwendeten Berechnungsmodells ist hier wichtig, denn dadurch sind die Algorithmen auch praktisch realisierbar. Die Probleme der Klasse P entsprechen also in etwa denen, die auch durch einen Computer berechenbar sind.
Für den Nachweis, dass eine Sprache bzw. ein Problem in P liegt, reicht es aus zu zeigen, dass die Komplexität des zugehörigen Algorithmus $O(n^k)$ für eine Konstante k ist. Zu zeigen, dass ein Problem nicht in P liegt, ist hingegen schon schwieriger und für über 1000 Probleme bisher auch (noch) nicht gelungen. Man weiß aber, dass sie entweder alle in P oder alle nicht in P enthalten sind (siehe P-NP-Problem).

Probleme, die in P liegen sind z.B. das PATH Problem und die Berechnung des ggT. Seit 2002 gehört auch die Frage, ob eine gegebene Zahl *prim* ist, zur Klasse P, da erstmals ein deterministischer Algorithmus (der AKS-Primzahltest) gefunden wurde, der das Problem in polynomieller Zeit löst.

2.3 Nichtdeterminismus

Deterministische Turingmaschinen sind für die Probleme aus P „zuständig“, was leisten nun nichtdeterministische?

Definition:

(1) Eine nichtdeterministische Turingmaschine N akzeptiert die Eingabe w, falls es mindestens einen Rechenweg von N gibt, der in einen akzeptierenden Zustand führt. Die von M erkannte Sprache $L = L(N)$ besteht aus allen Wörtern w, die N akzeptiert.

(2) Die Rechenzeit für eine nichtdeterministische Turingmaschine, die L akzeptiert, ist bei einer Eingabe $w \in L$ gleich der Anzahl der Rechenschritte auf dem *kürzesten* akzeptierenden Rechenweg und 0 falls $w \notin L$.

(3) Die *worst case* Rechenzeit von N ist die Funktion $f : \mathbb{N} \to \mathbb{N}$, wobei $f(n)$ die Anzahl von Schritten angibt, die N maximal für *jeden* Rechenweg bei allen Eingaben w der gleichen Länge n benötigt. Voraussetzung hierbei ist, dass N für alle Eingaben hält.

Satz: Sei $t(n)$ eine Funktion mit $t(n) \geq n$. Dann gibt es für jede $t(n)$-zeitbeschränkte nichtdeterministische Einband-Turingmaschine eine äquivalente $2^{O(t(n))}$-zeitbeschränkte deterministische Einband-Turingmaschine.

Beweis: Wir konstruieren zur nichtdeterministischen Turingmaschine N mit Laufzeit $t(n)$ eine deterministische D, die N simuliert, indem sie Ns nichtdeterministischen Baum von Rechenschritten nach einer akzeptierenden Rechnung durchsucht (per Breitensuche). Findet D solch einen akzeptierenden Rechenweg, akzeptiert auch D. D kann nur dann immer halten, wenn auch N für alle Eingaben hält.

Bei einer Eingabe der Länge n hat jeder Ast des Baums von N die maximale Länge $t(n)$ (der Baum hat $t(n)$ Ebenen) und jeder Knoten, der eine Konfiguration von N darstellt, kann maximal b Kinder haben, wobei b die Anzahl der möglichen Übergänge ist, die die Überführungsrelation von N vorgibt. Die Gesamtzahl der Blätter und somit gleichzeitig die Anzahl der Wege, die D durchsuchen muss, beträgt demnach höchstens $b^{t(n)}$. D durchsucht jeden Pfad in $O(t(n))$ Zeit und braucht insgesamt für alle Wege

$$O(t(n)b^{t(n)}) = 2^{O(t(n))}$$

Zeit, womit der Satz bewiesen wäre. □

2.4 Die Klasse NP

Die Klasse NP enthält alle von *nichtdeterministischen* Turingmaschinen in polynomieller Zeit lösbaren Probleme. Die Algorithmen zur Lösung der Probleme in NP laufen also auch in polynomieller Zeit, aber eben auf der Basis eines unrealistischen Maschinenmodells. Die NP-Probleme gelten daher als nicht effizient lösbar, da alle bekannten *deterministischen* Algorithmen für diese Probleme exponentiellen Rechenaufwand erfordern.
Bekannte Probleme in NP sind z.B. das Rucksackproblem und das Hamiltonkreisproblem.

3 Ausblick: P vs. NP

Da deterministische Turingmaschinen auch als nichtdeterministische angesehen werden können, die vom Nichtdeterminismus keinen Gebrauch machen, ist offensichtlich, dass $P \subseteq NP$ gilt, aber ob die beiden Komplexitätsklassen gleich oder echt ineinander enthalten sind, ist unbekannt. Diese Frage wird seit ca. 1970 untersucht und wird **P-NP-Problem** genannt. Sie wird oft als die wichtigste Frage der theoretischen Informatik überhaupt angesehen.

Dem P-NP-Problem kommt eine so große Bedeutung zu, da es viele für die Praxis wichtige Probleme gibt, deren Zugehörigkeit zu NP leicht nachzuweisen ist (z.B. das Traveling Salesman Problem), polynomielle Algorithmen aber für sie noch nicht bekannt sind. Es könnte sich also um mögliche Kandidaten aus $NP \setminus P$ handeln, sodass P und NP auf jeden Fall verschieden wären. Andererseits ist vielleicht bisher einfach noch niemand auf den richtigen Algorithmus gekommen.

Literatur

[1] Michael Sipser: Introduction to the Theory of Computation.

[2] Uwe Schöning: Theoretische Informatik - kurzgefasst

[3] Ingo Wegener: Theoretische Informatik

[4] http://de.wikipedia.org/wiki/Komplexitätstheorie