Proseminar Theoretische Informatik (WS 2010/11)

Zeitkomplexität I

Einführung und die Komplexitätsklasse P

Sebastian Stugk, Vortrag vom 26.11.2010

In den Vorlesungen zu den Grundlagen der theoretischen Informatik haben wir Probleme unter der Fragestellung betrachtet, ob diese entscheidbar, also grundsätzlich algorithmisch lösbar sind. Trotz der Existenz eines solchen Entscheidungsverfahrens können gewisse Probleme in der Realität, d.h. auf realen Maschinen, nicht „effizent“ lösbar sein. Die Komplexitätstheorie bietet uns die Möglichkeit Probleme hinsichtlich des für ihre Lösung nötigen Aufwands zu untersuchen.

Was ist Zeitkomplexität und wie wird sie bestimmt?

Informal kann man sich den Begriff Komplexität in unserem Zusammenhang zunächst als Bewertung eines Lösungs-/Entscheidungsverfahrens bezüglich der von ihm benötigten Ressourcen vorstellen.

Beim Turingmaschinenmodell, auf dessen Basis die folgenden Beispiele formuliert sind, ist das übliche Zeitmaß die Anzahl der Kopfbewegungen. Die Bestimmung der Zeitkomplexität erfolgt in Form einer *Analyse*. Dabei werden problemspezifische Einflussfaktoren außer Acht gelassen und die Laufzeit nur in Abhängigkeit von der Länge der Eingabe dargestellt. Die Zeitkomplexität einer deterministischen Turingmaschine ist wie folgt definiert:

Definition

Ist M eine deterministische Turingmaschine, die auf allen Eingaben hält, dann ist die **Laufzeit** oder **Zeitkomplexität** von M eine Funktion $f : \mathbb{N} \longrightarrow \mathbb{N}$, wobei $f(n)$ die *Maximalanzahl von Schritten* ist, die M bei einer Eingabe der Länge n macht.

Sprechweise: „M ist eine $f(n)$-Zeit-Turingmaschine.“ oder „M läuft in $f(n)$.“

Zu analysieren ist hier also der „*worst case*“ (die maximale Laufzeit) für Eingaben einer gewissen Länge. Die „*average case*“-Analyse betrachtet Gegensatz dazu den Durchschnitt aller möglichen Laufzeiten für eine Eingabelänge und erfordert im Allgemeinen umfangreiche statistische Betrachtungen.

Die Einschränkung auf die Länge der Eingabe als Parameter und die Unabhängigkeit von konkreten technischen Gegebenheiten durch ein abstraktes Maschinenmodell machen unsere Ergebnisse grundsätzlich vergleichbar. Doch im Gegensatz zur Frage nach der Entscheidbarkeit eines Problems sind die verschiedenen Maschinenmodelle bezüglich der Zeitkomplexität nicht äquivalent. Die Laufzeit eines Algorithmus ist also auch abhängig vom zugrunde liegenden Maschinenmodell.

Die **asymptotische Analyse**, eine Form der Abschätzung der Laufzeit für große Eingaben, und die zugehörige Notation (O-Notation) vereinfacht nicht nur die Laufzeitbestimmung. Wie später gezeigt wird, ermöglicht sie auch eine Klassifizierung von Problemen nach ihrer (Zeit-)Komplexität, die von Unterschieden (gleichmächtiger) Maschinenmodelle weitgehend unabhängig ist.

zur Erinnerung: O-Notation

1. asymptotische obere Schranken
   Seien f und g Funktionen mit $f, g : \mathbb{N} \longrightarrow \mathbb{R}^+$, dann ist
   - g obere Schranke von f, geschrieben $f(n) = O(g(n))$ falls natürliche Zahlen c und n_0 existieren, sodass: $\forall n \geq n_0 : f(n) \leq c \cdot g(n)$
   - g starke obere Schranke von f, geschrieben $f(n) = o(g(n))$, falls $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$ bzw. in obiger Formulierung falls für alle $c \in \mathbb{R}$ ein $n_0 \in \mathbb{N}$ existiert, sodass:
     $\forall n \geq n_0 : f(n) < c \cdot g(n)$
2. grundlegende Prinzipien:
   - Nur der Term höchster Ordnung eines Ausdrucks wird beachtet.
   - Konstante Faktoren werden außer Acht gelassen.
3. O-Notation ist verträglich mit gewöhnlichen arithmetischen Operationen

Analyse von Algorithmen

Wir betrachten im folgenden beispielhaft die Sprache $A = \{0^k 1^k \mid k \geq 0\}$, deren Entscheidbarkeit wir in den GTI-Vorlesungen (implizit durch eine erzeugende Grammatik) gezeigt haben.
Die folgenden zwei Turingmaschinen entscheiden A durch verschiedene Ansätze mit unterschiedlichen „worst case"-Laufzeiten.

Die Beschreibung erfolgt dabei so genau, dass man die Kopfbewegungen zählen kann. Modellspezifische Details, die die Laufzeit um nicht mehr als einen konstanten Faktor beeinflussen (z.B. die Neuausrichtung des Kopfes der Turingmaschine) werden hierbei nicht beachtet. Bei der asymptotischen Analyse werden Faktoren dieser Größenordnung wie oben erwähnt als vernachlässigbar angesehen. Das ermöglicht uns auch, die Beschreibung in menschlicher Sprache zu formulieren.

Eine zeilenweise Gliederung der Beschreibung entsprechend der Phasen, die ein bestimmter Algorithmus durchläuft, ermöglicht es die einzelnen Zeilen/Phasen weitgehend isoliert zu betrachten und dann auf die Gesamtlaufzeit zu schließen.

Turingmaschine M_1 arbeitet wie folgt beschrieben:

Turingmaschine M_1: „Auf Eingabe w:

1. Laufe über das Band und verwerfe, falls eine 0 rechts von einer 1 gefunden wird.
2. Solange sowohl *Nullen* als auch *Einsen* auf dem Band verbleiben:
3. Laufe über das Band und lösche jeweils genau eine 1 genau eine 0.
4. Falls entweder *Einsen* oder *Nullen* auf dem Band verblieben sind: verwerfe.
   Sonst, falls alle Zeichen vom Band gelöscht wurden: akzeptiere."

Analyse von M_1

Bezeichne n die Länge des Eingabewortes w, dann:

- Zeile 1 benötigt n Schritte um die Reihenfolge von *Nullen* und *Einsen* zu prüfen und ggf. n Schritte um den Kopf zur Ausgangsposition zu bewegen. Phase 1 benötigt also insgesamt $2n$ oder $O(n)$ Schritte.
- Jeder Durchlauf von Zeile 3 benötigt $O(n)$ Schritte. Dies wird höchstens $\frac{n}{2}$ mal wiederholt. Die Zeilen 2 und 3 benötigen also insgesamt $\frac{n}{2} \cdot O(n) = O(\frac{n^2}{2}) = O(n^2)$ Schritte.
- Zusammen mit Zeile 4 ($O(n)$ Schritte) beträgt die Laufzeit von M_1:
  $O(n) + O(n^2) + O(n) = O(n^2)$

Die Turingmaschine M_2 verdeutlicht, dass eine Turingmaschine, die ein gewisses Problem entscheidet, bezüglich ihrer Zeitkomplexität nicht zwangsläufig optimal sein muss.

Die $O(n \cdot log(n))$-Zeit-Turingmaschine M_2 arbeitet wie folgt:

Turingmaschine M_2: „Auf Eingabe w

1. Laufe über das Band und verwerfe, falls ein 0 rechts von einer 1 gefunden wird.
2. Solange sowohl *Nullen* als auch *Einsen* auf dem Band verbleiben:
3. Prüfe, die Gesamtanzahl von Nullen und Einsen. Falls ungerade: verwerfe.
4. Beginnend mit der jeweils ersten: Markiere erst jede zweite 0 und dann jede zweite 1.
5. Falls keine Nullen und Einsen auf dem Band verbleiben: Akzeptiere. Sonst: Verwerfe."

Funktionsprinzip von M_2:

Falls Zeile 2 erreicht wird, können wir eine korrekte Reihenfolge von Nullen und Einsen vorraussetzen. Alle gleichgroßen Anzahlen k von *Nullen* und *Einsen* werden immer zu gleichen Teilen gestrichen - Wir akzeptieren in Zeile 5. Jede Differenz d zwischen den beiden Anzahlen führt zu einem der beiden folgenden Fälle:
Fall 1 $d \leq k$: Die Differenz resultiert bei Eingabelänge n nach höchstens $\lfloor log_2(n) + 1 \rfloor$ Schleifendurchläufen (Zeilen 2 bis 4) zu 1 führt so zu einer ungeraden Gesamtanzahl von Zeichen (Verwerfen in Zeile 3).
Fall 2 $d > k$: Die Differenz führt spätestens als *unbalancierter Rest* in Zeile 5 zum Verwerfen.

Analyse von M_2

Bezeichne n die Länge des Eingabewortes w, dann:

- Zeile 1 und 5 benötigen wie bei M_1 $O(n)$ jeweils Schritte.
- Die je $O(n)$ Schritte der Zeilen 3 und 4 werden insgesamt höchstens $(1+log_2 n)$-mal wiederholt, da sich die Gesamtanzahl von Nullen und Einsen in jedem Schleifendurchlauf halbiert. Also insgesamt in dieser Phase: $(1 + log_2 n) \cdot O(n) = O(n \cdot log(n))$
- Die Gesamtlaufzeit für M_2 ist damit $O(n) + O(n \cdot log(n)) + O(n) = O(n \cdot log(n))$.

Zeitkomplexitätsklassen und ihr Zusammenhang mit Maschinenmodellen

Wir haben bisher 2 Turingmaschinen gesehen, die unser beispielhaftes Problem A mit einem bestimmten Zeitbedarf entscheiden. Die folgende Definition schlägt die Brücke zwischen der Laufzeit einer konkreten Turingmaschine und der Zeitkomplexität des Problems, dass sie entscheidet:

Definition

Sei $t : \mathbb{N} \longrightarrow \mathbb{N}$ eine Funktion. Dann **Zeitkomplexitätsklasse** $TIME(t(n))$ wie folgt definiert:
$TIME(t(n)) = \{L \mid L \text{ ist eine Sprache, die von einer } O(t(n))-\text{Zeit} - \text{TM entschieden wird.}\}$

Also ist $A \in TIME(n \cdot log(n))$. Die Definition schließt nicht aus, dass auch $A \in TIME(n^2)$. Für eine Funktion f mit $f(n) = O(g(n))$ auch jede Funktion h obere Schranke von f, falls $g(n) = O(h(n))$. Diese Teilmengenbeziehung überträgt sich auf die Definition von $TIME(t(n))$.

Eine 2-Band-Turingmaschine, kann unser Beispiel-Problem A in Zeit $O(n)$ entscheiden, indem sie die Folge von von Einsen auf das 2. Band kopiert und dann die Nullen (auf Band 1) und Einsen parallel liest. Verwerfen würden wir in diesem Fall sobald zu einer Eins (Null) keine entsprechende Null (Eins) auf dem jeweils anderen Band vorhanden ist.

Aus diesem Ergebnis können wir jedoch nicht ohne Weiteres auf die Komplexität von A schließen.

Satz

Sei $t(n)$ eine Funktion mit $t(n) \geq n$, dann:
Für jede $t(n)$-Zeit-Mehrband-TM gibt es eine äquivalente $O(t^2(n))$-Zeit-Einband-TM.

Konstruktion einer 1-Band-TM M', die eine k-Band-TM M simuliert:
M' simuliert die k Bänder von M durch k getrennte aufeinanderfolgende Abschnitte auf seinem einen Band. Die Kopfpositionen von M werden durch ein gesondertes Zeichen in den entsprechenden Abschnitten markiert. Nachdem M' das Band in das entsprechende Format gebracht hat, simuliert sie jeweils einen Schritt von M wie folgt:
In einem Lauf über den kompletten Inhalt des Bandes bestimmt M' die Kopfpositionen und das aktuell gelesene Zeichen von M (Phase 1). In einem zweiten Durchlauf aktualisiert M' die Kopfpositionen und ggf. geänderte Bandinhalte von M. Falls sich der Inhalt eines der k Bänder von M vergrößert muss der Bandinhalt von M' entsprechend verschoben werden (Phase 2).

Beweisidee:
Sei M eine k-Band-Turingmaschine mit Laufzeit $t(n)$ und M' eine Einband-TM, die M nach obiger Konstruktionsidee simuliert.
Die Laufzeit für Phase 1 ist abhängig von der Länge der k Abschnitte des Bandes von M', die die Bänder von M repräsentieren. Diese Länge ist nach oben beschränkt durch die Laufzeit $t(n)$ von M, da M in dieser Zeit auf jedem Band höchstens $t(n)$ Zellen liest/schreibt. M' benötigt also zur Simulation eines Schrittes von M $k \cdot t(n) = O(t(n))$ Schritte. Für die insgesamt $t(n)$ Schritte von M folgt für M' eine Laufzeit von $t(n) \cdot O(t(n)) = O(t^2(n))$.
(Die anderen hier nicht erwähnten Phasen haben lineare Laufzeit $O(n)$)

Dies verdeutlicht die schon erwähnte Abhängigkeit der Zeitkomplexität vom zugrunde liegenden Maschinenmodell. Im folgenden Abschnitt werden wir sehen, dass Laufzeit-Differenzen wie die Zwischen Einband- und Mehrband-Turingmaschinen bei der Frage nach der „effizienten“ Lösbarkeit eines Problems vernachlässigbar sind und desweiteren was Nichtdeterminismus in diesem Zusammenhang bedeutet.

Die Komplexitätsklasse P und eine Abgrenzung nach oben

Die $O(n \cdot log(n))$-Laufzeit von M_2 beträgt bei großen Eingaben nur einen Bruchteil der quadratisch beschränkten Laufzeit von M_2. Dieser Unterschied erscheint Vergleich zu exponentiell großen Differenzen jedoch „klein“. Dies kommt auch in der Definition Komplexitätsklasse P zum Ausdruck:

Definition

P ist die Klasse der Sprachen die durch eine deterministische Turingmaschine in polynomieller Zeit entscheidbar sind:

$$P = \bigcup_k TIME(n^k) \quad \text{mit} \quad k = konstant$$

Wie eingangs angedeutet hat diese weitgefasste Definition einen entscheidenden Vorteil: Modelle, die in Polynomialzeit durch eine Turingmaschine simuliert werden können sind bezüglich dieser Definition äquivalent. Diese **polynomielle Äquivalenz** gilt neben der Mehrband-Turingmaschine z.B. auch für die Registermaschine (*Quelle [2]*).

Für **nichtdeterministische Turingmaschinen** (NTM) gilt diese Äquivalenz nicht. Die Definition von nichtdeterministischen TM ist analog zur der von nichtdet. endlichen Automaten durch eine weiter gefasste Übergangsfunktion charakterisiert:
Für jede Konfiguration einer NTM kann es mehrere Folgekonfigurationen geben. Die Wörter der durch eine NTM erzeugte Sprache sind dementsprechend diejenigen, für die eine akzeptierende Rechnung *existiert*. Die Simulation einer solchen nichtdet. Turingmaschine durch eine deterministische ist ein Beispiel für eine Situation die häufig Ursache für exponentiell große Laufzeiten ist: Ein **exponentiell großer Ergebnisraum** muss (ohne genauere Informationen) „*brute force*“ durchsucht werden.

Satz

Sei $t(n)$ eine Funktion mit $t(n) \geq n$, dann:
Für jede $t(n)$-Zeit-NTM gibt es eine äquivalente $2^{O(t(n))}$-Zeit-Einband-TM.

Laufzeit einer NTM:
Man kann sich die Rechnung einer NTM bildlich als verzweigten Baum im Vergleich zum eindeutigen (linearen) Pfad der deterministischen Turingmaschine vorstellen. Die Laufzeit ist dabei eine Funktion $f : \mathbb{N} \longrightarrow \mathbb{N}$, wobei $f(n)$ die größte mögliche Laufzeit aller möglichen Zweige der Rechnung für alle Eingaben der Länge n ist.

Idee zur Konstruktion einer 3-Band-TM M', die eine NTM M simuliert:
M' durchsucht den Berechnungsbaum von M mittels Breitensuche. Dabei nutzt sie ein Band zur Simulation eines Zweigs der Rechnung von M. Dieser aktuelle Zweig ist in Form einer „Adresse“ auf einem weiteren Band kodiert. Nach Abschluss der Simulation eines Zweigs wird mit der lexikografisch nächsten „Adresse“ fortgefahren. So wird im Berechnungsbaum von M *Knoten für Knoten* nach einer akzeptierenden Konfiguration gesucht.

Beweisidee:
Für eine Eingabe der Länge n ist die Länge der Zweige des Berechnungsbaums der NTM durch $t(n)$ beschränkt. Die Anzahl der äußeren Knoten des Berechnungsbaums von M beträgt höchstens $b^{t(n)}$ (Konstante b gegeben durch Übergangsfunktion von M). Die Gesamtanzahl von Knoten ist kleiner als das $2 \cdot b^{t(n)}$, beträgt also asymptotisch $O(b^{t(n)})$. Zusammen mit den $t(n)$ nötigen Schritten folgt für die Laufzeit von M' $O(t(n) \cdot b^{t(n)}) = O(2^{t(n)})$.

Bei Simulation einer NTM durch eine deterministische TM entsteht also Laufzeitdifferenz die größer ist als das „tolerierte Maß“ in der Definition von P. Konkreter bedeutet das, dass aus der Lösbarkeit eines Problems auf einer NTM in polynomieller Zeit zunächst nicht auf die „effiziente“ deterministische Lösbarkeit geschlossen werden kann (P-NP-Problem).

Quellen *zusätzlich zu „Introduction to the Theory of Computation“ (Sipser, PWS Publ.Comp. 1997)*

1. Papadimitriou, Christos H.: Computational complexity; 1994; Addison-Wesley
2. Stetter, Franz: Grundbegriffe der theoretischen Informatik; 1988; Springer
3. Wegener, Ingo: Kompendium theoretische Informatik; 1996; Teubner
4. K. Wagner, G. Wechselung: Computational Complexity; 1986; Reidel Publ. - VEB Dt. Verlag der Wissenschaften
5. Winter, Renate: Theoretische Informatik; 2002; Oldenburg Verlag
6. R. Greenlaw, H. J. Hoover, S. Miyano, W. L. Ruzzo, S. Shiraishi, T. Shoudai: The Parallel Computation Project: Volume II (am 10.11.2010 via http://www.cs.armstrong.edu/greenlaw/research/PARALLEL/volume2.html)
7. Universität Potsdam: Vorlesung „Theoretische Informatik II“, Einheiten 8.1 bis 8.3; WS 2003/04 (am 14.09.2010 via http://www.cs.uni-potsdam.de/ti/lehre/03-Theorie-II/slides.htm)
8. http://de.wikipedia.org/w/index.php?title=Komplexitätstheorie&oldid=80205286 (vom 12.10.2010)
9. http://de.wikipedia.org/w/index.php?title=Nichtdeterministische_Turingmaschine&oldid=80858576 (vom 29.10.2010)
10. http://de.wikipedia.org/w/index.php?title=P_(Komplexitätsklasse)&oldid=80958088 (vom 31.10.2010)