Proseminar Theoretische Informatik 05.01.2016

Komplexitätsklasse P

Albert Mkhitaryan, Maria Sparenberg *Wolfgang Mulzer*

1 Komplexitätstheorie

Mithilfe der Komplexitätstheorie werden algorithmische Probleme bzw. Sprachen anhand ihres Bedarfs an Berechnungsressourcen klassifiziert. Es gibt entscheidbare Sprachen, die in der Praxis nicht effizient lösbar sind, weil sie einen zu hohen Speicherbedarf haben oder ihre Laufzeit sehr groß ist. Aus diesem Grund ist es wichtig, die Komplexität solcher Sprachen zu analysieren, wobei im Folgenden die Laufzeitanalyse im Vordergrund stehen wird.

1.1 O-Notation

Oft ist die exakte Laufzeit eines Algorithmus eine sehr komplexe Funktion, die nur schwer zu bestimmen ist. Eine Schätzung ist daher ausreichend und wird in Abhängigkeit der Eingabegröße berechnet. Dies wird durch eine asymptotische Analyse mithilfe der O-Notation realisiert, die eine obere Schranke für das Wachstum einer solchen Funktion liefert.

Seien f und $g : \mathbb{N} \to \mathbb{R}^+$ zwei Funktionen.

$O(g(n)) = \{f(n) \mid \exists\ c > 0\ \ \exists\ n_0\ \ \forall\ n \geq n_0 :\ \ f(n) \leq c \cdot g(n)\}$

Die Menge $O(g(n))$ enthält also alle Funktionen f, für die g eine obere asymptotische Schranke ist. Anders formuliert wachsen alle Funktionen f in dieser Menge nicht wesentlich schneller als g.

So lassen sich nun auch Zeitkomplexitätsklassen beschreiben:
Sei $t : \mathbb{N} \to \mathbb{R}^+$ eine Funktion. Dann ist $TIME(t(n))$ die Klasse aller Sprachen, die durch eine $O(t(n))$-zeitbeschränkte Turingmaschine (TM) entscheidbar ist.

Beispielsweise enthält $TIME(n^2)$ alle Sprachen, die in $O(n^2)$ Zeit entschieden werden können.

1.2 Beispielhafte Laufzeitanalyse

Einband-Turinmaschine

Für die entscheidbare Sprache $L = \{0^k 1^k \mid k \geq 0\}$ und die dazugehörige Einband-TM M_1, die L entscheidet, soll zeilenweise eine Laufzeitanalyse durchgeführt werden.

M_1 erhält die Eingabe der Länge n:
1. Scanne das Band und verwerfe, wenn eine 0 rechts von einer 1 gelesen wird.
2. Solange noch Nullen und Einsen auf dem Band stehen:
3. Scanne das Band und streiche eine 0 und eine 1.
4. Verwerfe, wenn noch eine 0 bzw. eine 1 auf dem Band steht. Sonst akzeptiere.

Zeile 1: Die gesamte Eingabe wird einmal gelesen, um die Form zu überprüfen. Die TM M_1 macht n Schritte. Danach wird der Schreib-Lese-Kopf zurück nach links bewegt, was wieder n Schritten entspricht. Der Zeitaufwand ist also $2n \in O(n)$.

Zeile 2+3: Bei jedem Durchlauf der Schleife muss das Band durchlaufen werden. Da bei jedem Durchlauf jedoch die Anzahl der Zeichen immer um 2 abnimmt, muss die Schleife nur $n/2$ Mal durchlaufen werden. Der Zeitaufwand ist also $n/2 \cdot O(n) \in O(n^2)$.

Zeile 4: Zuletzt wird das Band nochmal durchlaufen, um zu überprüfen, dass kein Zeichen mehr darauf steht. Der Zeitaufwand dafür ist $O(n)$.

Insgesamt gehört dieser Algorithmus also zur Klasse der quadratischen Laufzeit $TIME(n^2)$. Mit einem veränderten Algorithmus lässt sich auch zeigen, dass er zu $TIME(n \log n)$ gehört. Mit einer Einband-TM lässt sich dies jedoch nicht weiter verbessern.

Mehrband-Turingmaschine

Die $L = \{0^k 1^k \mid k \geq 0\}$ lässt sich noch etwas effizienter auf einer Zweiband-TMe M_2 entscheiden.

M_2 erhält die Eingabe auf Band 1:
1. Scanne das Band 1 und verwerfe, wenn eine 0 rechts von einer 1 gelesen wird.
2. Lese alle Nullen auf Band 1 und schreibe diese gleichzeitig auf Band 2.
3. Lese alle Einsen auf Band 1 und streiche für jede gelesene 1 auf Band 1 eine 0 auf Band 2. Wenn alle Nullen auf Band 2 gestrichen sind, bevor alle Einsen gelesen wurden, verwerfe.
4. Wenn alle Nullen auf Band 2 gestrichen sind, akzeptiere. Sonst verwerfe.

Es ist offensichtlich, dass jede Zeile des Algorithmus nicht mehr als $O(n)$ Zeit benötigt, weshalb der zeitliche Gesamtaufwand $O(n)$ ist und der Algorithmus zur Klasse der linearen Laufzeit $TIME(n)$ gehört.

1.3 Komplexität ist modellabhängig

Die Church-Turing-These impliziert, dass alle Berechenbarkeitsmodelle, die dieselbe Sprache entscheiden, äquivalent sind. Im vorherigen Abschnitt jedoch wurde deutlich, dass es bezüglich der Laufzeit sehr wohl darauf ankommt, welches Modell zur Berechnung herangezogen wird. Eine Einband-TM ist ein anderes Modell als eine Mehrband-TM, und letztere konnte die Sprache L in geringerer Laufzeit entscheiden. Dieser Unterschied soll mithilfe der folgenden zwei Theoreme verdeutlicht werden.

Theorem 1

Sei $t(n)$ eine Funktion mit $t(n) \geq n$. Dann gibt es für jede $t(n)$ Mehrband-TM eine äquivalente Einband-TM mit einer Laufzeit von $O(t^2(n))$.

Jede k-Band-TM M kann durch eine Einband-TM S simuliert werden, indem die k Bänder von M hintereinander auf das eine Band von S geschrieben wird. Zudem wird die Position jedes Schreib-Lese-Kopfes in der entsprechenden Zelle markiert (wodurch das Bandalphabet erweitert wird). Unter der Annahme $t(n) \geq n$ (denn n Schritte werden allein schon zum Lesen der Eingabe benötigt), wird das Theorem 1 bewiesen, indem einfach die von S zusätzlich benötigte Zeit untersucht wird. Dazu soll im Folgenden die Simulation eines Schrittes von Manalysiert werden:

S simuliert einen Schritt von M:

1. *Sammeln von Informationen:* Scanne das gesamte Band und speichere alle Zeichen, Kopf-Positionen und Zustände, und bestimme somit die nächste Bewegung.
2. *Ausführen des Schritts:* Durchlaufe erneut das Band und überschreibe die Zeichen und Kopf-Positionen entsprechend der Übergangsfunktionen von M.

S durchläuft offensichtlich für jeden Schritt von M das Band zweimal. Wie lange S dafür benötigt, ist abhängig von der Länge der k aktiven Bandabschnitte von M, weswegen als obere Schranke deren Summe verwendet wird. Würde M in jedem Schritt nach rechts gehen, wäre die Summe also höchstens eine Länge von $O(t(n))$. Somit braucht S für jeden Durchlauf $O(t(n))$ Zeit. Da S aber alle $t(n)$ Schritte von M simuliert, ergibt sich daraus eine Gesamtlaufzeit von $t(n) \cdot O(n) \in O(t^2(n))$.

Theorem 2

Sei $t(n)$ eine Funktion mit $t(n) \geq n$. Dann gibt es für jede $t(n)$ nichtdeterministische Einband-TM eine äquivalente deterministische Einband-TM mit einer Laufzeit von $2^{O(t(n))}$.

Jede nichtdeterministische TM N kann durch eine deterministische TM D simuliert werden, wobei N als Baum interpretiert wird und D mit Breitensuche nach einer akzeptierenden Konfiguration sucht. Dabei ist D eine Dreiband-TM: das erste Band enthält die Eingabe (Eingabeband), das zweite Band enthält eine Kopie des Bandabschnittes eines aktuell besuchten Asts von N (Simulationsband) und das dritte Band enthält die Position von D in N's Baum (Adressband).

Die Laufzeit $t(n)$ einer nichtdeterministischen TM entspricht der maximalen Anzahl von Schritten, die auf allen Berechnungsästen für jede Eingabe der Länge n notwendig sind. Somit benötigt also jeder Ast von N höchstens $t(n)$ Schritte. Jeder Knoten von N hat höchstens b Kinder, wobei b der maximalen Anzahl von möglichen Ausgängen der entsprechenden Übergangsfunktion von N entspricht. Damit besteht der Baum von N aus höchstens $b^{t(n)}$ Blättern und weniger als $2b^{t(n)}$ Knoten. Daraus ergibt sich eine exponentielle Gesamtlaufzeit von $O(t(n) \cdot b^{t(n)} = 2^{O(t(n))}$. Bei der Umwandlung der Dreiband-TM in eine Einband-TM wird die Laufzeit quadriert (laut Theorem 1), wobei es an der Gesamtlaufzeit nichts ändert: $(2^{O(t(n))})^2 = 2^{O(2 \cdot t(n))} = 2^{O(t(n))}$.

Der Unterschied zwischen deterministischen Modellen ist polynomiell (Theorem 1) und der zwischen einer nichtdeterministischen und einer deterministischen TM ist sogar exponentiell (Theorem 2).

2 Komplexitätsklasse P

Definition:
P ist die Komplexitätsklasse, die alle Entscheidungsprobleme enthält, die in Polynomialzeit für deterministische Einband -Turingmaschinen lösbar sind.

Ein Problem wird in Polynomialzeit lösbar genannt, wenn es von einem Algorithmus gelöst wird, dessen benötigte Rechenzeit höchstens polynomiell mit der Größe der Eingabe n des Problems wächst: $O(n^k)$ mit $k \geq 1$.

Dabei wird nicht zwischen verschiedenen polynomiellen Funktionen unterschieden, sondern nur ob sie polynomiell oder exponentiell sind. Dieser Unterschied ist sehr groß, daher werden entsprechende Algorithmen in verschiedene Klassen eingeordnet.
Betrachten wir die beiden folgenden Funktionen: polynomielles Wachstum n^3 und exponentielles Wachstum 2^n. Sei $n = 1000$, dann ist 1000^3 eine Milliarde, aber 2^{1000} eine größere Zahl als alle Atome von unserem Universum. Eine Milliarde ist zwar eine große Zahl, aber in der Praxis noch lösbar, während 2^{1000} zu groß ist. Das heißt, Algorithmen, die exponentielle Zeit brauchen, sind selten sinnvoll.

These: Alle vertretbare deterministische Berechnungsmodelle sind polynomiell äquivalent und jedes Berechnungsmodell kann jedes andere in polynomiellem Wachstum simulieren. Theorem 1 zeigt beispielsweise, dass eine Mehrband-TM und eine Einband-TM polynomiell äquivalent sind. Damit ist die Klasse P unabhängig von Berechnungsmodell.

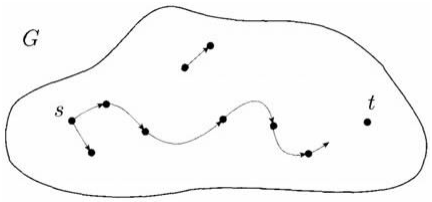
Eigenschaften der Klasse P:
- Invariante für alle Berechnungsmodelle, die polynomiell äquivalent zur Einband-TM sind
- Klasse von Problemen, die auf einem Computer realistisch lösbar sind

2.1 Beispiele für P-Probleme

Beispiel 1: PATH

Eingabe: Gerichteter Graph G und zwei Knoten s und t in dem Graphen
Gefragt: Gibt es einen Weg von s nach t in G?
Zu zeigen: PATH $\in P$

PATH $= \{\langle G, s, t\rangle \mid G$ ist ein gerichteter Graph, der einen Weg von s nach t enthält$\}$



Die Brute-Force Methode wird alle mögliche Wege in G untersuchen und bestimmen, ob einer davon einen Weg von s nach t darstellt. Jeder mögliche Weg ist eine Folge von Knoten in G mit maximaler Länge m, wobei m die Anzahl von Knoten ist. Da die Anzahl von solchen möglichen Wegen m^m ist, würde somit exponentielle Rechenzeit für die Lösung dieses Problems benötigt werden. Einen besseren Ansatz liefert beispielsweise die Breitensuche. Dabei werden alle erreichbaren Knoten von s der Länge $1, 2, ..., m$ markiert. Untersuchen wir den Algorithmus M.

M erhält Eingabe $\langle G, s, t\rangle$:
1. Markiere s.
2. Wiederhole den folgenden Schritt, bis alle Knoten markiert sind:
3. Für alle Kanten in G: Falls eine Kante (a, b) gefunden, wobei a markiert ist und b nicht, markiere b.
4. Falls t markiert ist, akzeptiere, sonst verwerfe.

Laufzeitanalyse:
Schritte 1 und 4 werden nur einmal ausgeführt. Schritt 3 kann maximal m mal ausgeführt werden, da bei jeder Ausführung dieses Schrittes ein Knoten markiert wird. Schritt 3 beinhaltet die Suche in der Eingabe, was wiederum polynomielle Zeit verlangt. Daraus folgt, M ist ein Algorithmus für PATH.

Beispiel 2: Teilerfremdheit RELPRIME

Eingabe: $x, y \in \mathbb{N}$
Gefragt: Sind x und y relativ prim?
Zu zeigen: RELPRIME $\in P$

RELPRIME $= \{\langle x, y\rangle \mid x$ und y sind teilerfremd$\}$

Zwei natürliche Zahlen x und y sind teilerfremd oder relativ prim ($x \perp y$), wenn es keine natürliche Zahl außer 1 gibt, die beide Zahlen teilt. Mithilfe des Euklidischen Algorithmus lässt sich in Polynomialzeit der größte gemeinsame Teiler (ggT) zweier natürlicher Zahlen berechnen. Falls $ggT(x, y) = 1$, dann sind die Zahlen offensichtlich teilerfremd, ansonsten nicht.

E erhält Eingabe $\langle x, y\rangle$ mit $x, y \in \mathbb{N}$:
1. While y !=0
2. x:= x mod y
3. Tausche x und y
4. Gebe x aus.

Der Algorithmus R löst das Problem mithilfe von E.

R erhält Eingabe $\langle x, y\rangle$ mit $x, y \in \mathbb{N}$:
1. Führe E auf $\langle x, y\rangle$ aus.
2. Falls Ausgabe $= 1$, akzeptiere, sonst verwerfe.

Laufzeitanalyse:
Wenn E polynomielle Zeit braucht, dann benötigt R offensichtlich auch polynomielle Zeit. Um die

Laufzeit von E zu analysieren, zeigen wir zuerst, dass im Schritt 2 x mindestens halbiert wird (außer eventuell beim ersten Teilen). Nach diesem Schritt ist $x < y$. Nach Schritt 3 ist $x > y$, da sie in Schritt 2 getauscht wurden.

- Falls $\frac{x}{2} \geq y$, dann $x \; mod \; y < y \leq \frac{x}{2}$ und x wird mindestens halbiert.
- Falls $\frac{x}{2} < y$, dann $x \; mod \; y = x - y < y \leq \frac{x}{2}$ und x wird mindestens halbiert.

So werden x und y abwechselnd halbiert, bis $y = 0$ ist.

Ein Beispiel:

$$\begin{aligned} 1599 &= (650 \cdot 2) + 299 \\ 650 &= (299 \cdot 2) + 52 \\ 299 &= (52 \cdot 5) + 39 \\ 52 &= (39 \cdot 1) + 13 \\ 39 &= (13 \cdot 3) + 0 \end{aligned}$$

Wenn $y = 0$, dann $x = 13$. Das heißt, der $ggT(1599, 650) = 13$.

Daraus folgt, dass die Anzahl von Ausführungen von Schritten 2 und 3 kleiner als $2 \log_2 x$ oder $2 \log_2 y$ ist. Dabei sind die Logarithmen proportional zur Länge der Darstellung. Wir können jetzt behaupten, dass die Laufzeit von R $O(n)$ ist.