Complexity Theory

Dr. Eva Richter

15. Juni 2012

Aufgaben und Ziele

- bestimmen der Berechnungskomplexität (oder „Härte“) von Problemen so genau wie möglich
- entscheidbares (im Prinzip mechanisch lösbarbares) Problem kann praktisch unlösbar sein, da es übermäßig viel Platz oder Zeit braucht
- Wie schwierig ist es zu entscheiden, ob ein Wort $w \in \{0,1,2\}^*$ in $S = \{x2^{|x|}x \mid x \in \{0,1\}^*\}$ liegt oder nicht?

Mögliche Antworten

1. Turingmaschinen mit zwei Bändern, eins nur zum Lesen der Eingabe, eins zum Lesen und Schreiben können S in Echtzeit lösen, die Anzahl der Berechnungsschritte entspricht der Länge der Eingabe.
2. Einbandturingmaschinen brauchen mindestens quadratischen Zeitaufwand (in der Eingabegröße).
3. DEA können S gar nicht entscheiden.

Merkmale

- verwendetes Berechnungsmodell– das Werkzeug, mit dem die Aufgabe gelöst werden soll,
- Berechnungsparadigma oder der Akzeptierungsmodus; beispielsweise:
  - deterministische TM
  - probabilistische TM
  - nichtdeterministische TM
- verwendetes Komplexitätsmaß oder die Ressource–z.B. die zur Lösung nötige Zeit oder der benötigte Speicherplatz.

Aufgaben von CT

- Untersuchung wichtiger interessanter Probleme aus vielen Wissensgebieten, um sie hinsichtlich ihrer Komplexität zu klassifizieren
- Berechnungskraft verschiedener algorithmischer Werkzeuge und Automaten und der zugehörigen Berechnungsparadigmata zu vergleichen und die „Trade-offs“ zu bestimmen, z.B. Zeit-versus-Raum-Frage und die Determinismus-versus-Nichtdeterminismus-Frage
- Konzentrieren uns auf Komplexitätsmaße Zeit und Platz im Worst-Case-Komplexitätsmodell und entsprechende Komplexitätsklassen

Plan

- Komplexitätsklasse ist eine Menge von Problemen, die gemäß einem gegebenen Berechnungsmodell und -paradigma, durch Algorithmen gelöst werden können, welche höchstens den vorgegebenen Betrag der jeweiligen Komplexitätsressource verbrauchen
- Sätze über lineare Beschleunigung und Raumkompression sowie Hierarchiesätze für Raum und Zeit: Um wieviel muss eine Komplexitätsressource vergrößert werden, damit echt mehr Probleme von einem auf diese Ressourcen beschränkten Algorithmus gelöst werden können
- Härte und Vollständigkeit eines Problems in einer Komplexitätsklasse einführen, die charakterisieren, ob ein Problem zu einer gewissen Klasse gehört und ob es zu den schwersten Problemen dieser Klasse gehört.
- Klassen *P* und *NP*

Vereinbarung

Algorithmenmodell: Turingmaschinen
Problem hat gewisse Komplexität: DTM terminiert nach endlich vielen Schritten, NTM hat mindestens einen akzeptierenden Pfad

Deterministische Komplexitätsmaße

Definition (Zeitfunktion und Raumfunktion)

Sei M eine DTM, definiere $Time_M, Space_M : \Sigma^* \to \mathbb{N}$ durch

$$Time_M(w) = \begin{cases} m & \text{wenn } M \text{ auf } w \text{ nach} \\ & m+1 \text{ Konfigurationen terminiert} \\ \text{undefiniert} & \text{sonst.} \end{cases}$$

$$Space_M(w) = \begin{cases} \text{Anzahl der Bandzellen} \\ \text{in einer größten} \\ \text{Konfiguration von } M \text{ auf } w & \text{falls } w \in L(M) \\ \text{undefiniert} & \text{sonst.} \end{cases}$$

Verhalten auf allen Eingaben

Definition

$$time_M(n) = \begin{cases} max_{|w|=n} Time_M(w) & \text{falls } Time_M \text{ für alle } w \\ & \text{der Länge } n \text{ definiert ist} \\ undefiniert & \text{sonst.} \end{cases}$$

$$space_M(n) = \begin{cases} max_{|w|=n} Space_M(w) & \text{falls } Space_M \text{ für alle } w \\ & \text{der Länge } n \text{ definiert ist} \\ undefiniert & \text{sonst.} \end{cases}$$

$space_M$-Funktion ist nur dann definiert, wenn M terminiert!

Deterministische Komplexitätsklassen

Definition (Deterministische Zeit-und Raumklassen)

Seien $s, t : \mathbb{N} \to \mathbb{N}$ *total berechenbar.*

$$DTIME(t) = \{A \mid A = L(M) \text{ für eine DTM } M \text{ und für alle } n \in \mathbb{N} \text{ gilt: } time_M(n) \leq t(n)\}$$

$$DSPACE(s) = \{A \mid A = L(M) \text{ für eine DTM } M \text{ und für alle } n \in \mathbb{N} \text{ gilt: } space_M(n) \leq s(n)\}$$

Nichtdeterministische Komplexitätsmaße

- $M(w)$ sei Berechnungsbaum einer NTM M bei Eingabe von w
- Berechnung entlang eines festen Pfades α ist wie deterministische Berechnung: Folge von Konfigurationen.
- Zeit-und Raumfunktionen werden für jeden Pfad α definiert

Definition (Nichtdeterministische Komplexitätsmaße)

$$NTime_M(w) = \begin{cases} min_{\alpha:akz.}\{Time_M(w,\alpha)\} & falls\ w \in L(M) \\ undefiniert & sonst. \end{cases}$$

$$NSpace_M(w) = \begin{cases} min_{\alpha:akz.}\{Space_M(w,\alpha)\} & falls\ w \in L(M) \\ undefiniert & sonst. \end{cases}$$

Nichtdeterministische Zeit-und Raumklassen

Definition

Seien $s, t : \mathbb{N} \to \mathbb{N}$ total berechenbar.

M akzeptiert Menge A in der Zeit t, falls

- $NTime_M(w) \leq t(|w|)$ für jedes $w \in A$ gilt, und
- w nicht von M akzeptiert wird, falls $w \notin A$.

M akzeptiert A im Raum s, falls

- $NSpace_M(w) \leq s(|w|)$ für jedes $w \in A$ gilt, und
- w nicht von M akzeptiert wird, falls $w \notin A$.

Definition

$$NTIME(t) = \{A \mid A = L(M) \text{ für eine NTM } M \text{ die } A \text{ in der Zeit } t \text{ akzeptiert}\}$$

$$NSPACE(s) = \{A \mid A = L(M) \text{ für eine NTM } M \text{ die } A \text{im Raum } s \text{ akzeptiert}\}$$

Familien von Ressourcenfunktionen

- Unterscheidung in Komplexitätsklassen soll nicht zu fein sein: z.B. $DTIME(n^2)$ und $DTIME(n^2 + 1)$
- betrachte Familien $\mathcal{F}$ von „Ressourcenfunktionen“und bilde Klassen z.B.

$$DTIME(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} DTIME(f)$$

- Familie enthält alle Ressourcenfunktionen $f : \mathbb{N} \to \mathbb{N}$ mit einer ähnlichen Wachstumsrate

Familien von Ressourcenfunktionen–Beipiele

- Lin enthält alle lineare Funktionen
- Pol enthält alle Polynome
- 2^{Lin} enthält alle Exponentialfunktionen, bei denen der Exponent linear in n ist
- 2^{Pol} enthält alle Exponentialfunktionen, bei denen der Exponent polynomiell in n ist.

Resultierenden Komplexitätsklassen sind invariant unter endlicher Variation.

Asymptotische obere Schranken I

Definition

Seien $f, g : \mathbb{N} \to \mathbb{R}^+$.
Sage $f \in O(g)$, *gdw.* $\exists c, n_0 \in \mathbb{N}^+$, *so dass* $\forall n \geq n_0$
$f(n) \leq c \cdot g(n)$.
g *heißt asymptotische obere Schranke für* f.

Sei $f_1(n) = 5n^3 + 2n^2 + 22n + b$, dann ist $f_1 \in O(n^3)$.
Für $n \geq 10$ und $n_0 = 10$, $c = 6$ gilt:

$$5n^3 + 2n^2 + 22n \leq 6n^3$$

Außerdem gilt $f_1 \in O(n^4)$, aber $f_1 \notin O(n^2)$.
(Es gibt keine geeigneten Konstanten n_0 oder c)

Asymptotische obere Schranken II

- Logarithmen werden mit Basis angegeben: $x = log_2 n$, Basiswechsel durch $log_b n = log_2 n / log_2 b$ bewirkt nur Änderung um konstanten Faktor, es geht auch $f \in O(log)$
- Für f_2 mit $f_2(n) = 3nlog2n + 5nlog_2 log_2 n + 2$ reicht $f_2 = O(nlogn)$, da $logn$ größer als $loglogn$ ist.
- bei Ausdrücken der Form $f \in O(n^2) + O(n)$ meint man, dass $f(n) \leq h(n) + g(n)$ für gewisse h und g mit $h \in O(n^2)$ und $g \in O(n)$, jedes$O$ für eine unterdrückte Konstante. Ausdruck ist äquivalent zu $f \in O(n^2)$.
- betrachte $f \in 2^{O(logn)}$, benutze $n = 2^{log_2 n}$, dann folgt $n^c = 2^{clog_2 n}$, d.h. $2^{O(logn)}$ ist obere Schranke für n^c für ein c, ebenso wie $n^{O(1)}$, da $O(1)$ einen Wert darstellt, der nie größer als eine Konstante ist.
- n^c mit $c > 0$ heißen polynomielle Schranken, 2^{n^δ} heißen exponentielle Schranken falls $\delta \in \mathbb{R}^+$

Klein-O-Notation

Definition

Seien $f, g : \mathbb{N} \to \mathbb{R}^+$. $f \in o(g)$ *falls*

$$lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$f \in o(g)$ heißt, dass für jede reelle Zahl $c > 0$ eine Zahl n_0 existiert, sodass $f(n) < c \cdot g(n)$ für alle $n \geq n_0$.

Beispiele:

1. Sei $f_1(n) := \sqrt{n}$, dann gilt $f_1 \in o(n)$
2. Sei $f_2(n) := n$, dann gilt $f_2 \in o(nloglogn)$
3. Sei $f_3(n) := nloglogn$ dann gilt $f_3 \in o(nlogn)$
4. Sei $f_4(n) := nlogn$ dann gilt: $f_4 \in o(n^2)$
5. Sei $f_5(n) := n^2$, dann gilt $f_5 \in o(n^3)$.

f ist niemals in $o(f)$!

Entscheider für die Sprache $A = \{0^k 1^k \mid k \geq 0\}$.

$M_1 =$

„Auf Eingabe von w:

1. Gehe über das Band und lehne ab, wenn rechts von einer 1 eine 0 steht.
2. Wiederhole den folgenden Schritt solange sowohl Einsen als auch Nullen auf dem Band sind:
3. Gehe über das Band und streiche dabei immer je eine Null und Eins aus.
4. Falls noch Einsen da sind, wenn es keine Nullen mehr gibt, oder falls noch Nullen da sind, wenn es keine Einsen mehr gibt, lehne ab; falls beide zugleich abgestrichen sind, akzeptiere.“

Algorithmenanalyse II

- Schritt 1 braucht man n Schritte, um die Gültigkeit der Eingabe zu prüfen
- Um wieder zum Anfang zurückzugehen, braucht auch n Schritte, also $2n$ oder $O(n)$ Schritte.
- Neupositionierung des Kopfes wurde nicht berücksichtigt, da konstante Faktoren keine Rolle spielen.
- In Zustand 2 und 3 werden jeweils eine Null und Eins abgestrichen, was jedesmal $O(n)$ Schritte benötigt.
  Da das ganze höchstens $n/2$ mal gemacht wird, haben wir insgesamt in 2 und 3 $(n/2)O(n) = O(n^2)$ Schritte.
- Im Zustand 4 wird ein Test gemacht, der höchstens $O(n)$ braucht.
- Insgesamt: $O(n) + O(n^2) + O(n)$ bzw. $A \in TIME(n^2)$

Gibt es eine schnellere Maschine?
Liegt A in $TIME(t(n))$ für $t \in o(n^2)$?

Zweiter Versuch

Herausstreichen von jeweils zwei Nullen und Einsen in einem Schritt ausstreicht halbiert die Anzahl der Arbeitsschritte nur und hat keine Auswirkungen auf das asymptotische Verhalten.

$M_2 =$

„Auf Eingabe von w:

1. Gehe über die Eingabe und reject, wenn 1 vor 0
2. Wiederhole das Folgende solange Nullen und Einsen auf dem Band sind:
3. Teste die Eingabe (über das Band gehen), ob sie gerade oder ungerade Zahl von Nullen und Einsen hat, reject, wenn ungerade.
4. Gehe über das Band und streiche jede zweite verbliebene Null aus und jede zweite Eins, jeweils vorn mit der ersten beginnend.
5. Wenn keine Nullen und Einsen mehr da sind, akzeptiere anderenfalls weise ab.“

Zweiter Versuch–Analyse

- untersuche die gerade/ungerade-Verhältnisse der Anzahl der Nullen und Einsen in jedem Stadium von 3: stelle Folge der„Geradeheiten-Ungeradeheiten“ durch Binärzahlen dar
- jedes Stadium benötigt $O(n)$ Schritte, stelle deren Ausführungszahl fest
- Bei Schritt 1 und 5 einmal, also insgesamt $O(n)$; Schritt 4 muss höchstens halb so oft ausgeführt werden, wie Nullen und Einsen gibt, also höchstens $1 + log_2 n$ Iterationen.
- Gesamtzeit von 2, 3 und 4 ist in $(1 + log_2 n)(O(n))$ oder $O(n \cdot logn)$, also $A \in DTIME(nlogn)$.

Auf Einbandturingmaschinen lässt sich das Ergebnis nicht weiter verbessern.
Jede Sprache, die in $o(nlogn)$ von einer Einbandturingmaschine entschieden werden kann, ist regulär.

Dritter Versuch–mit ZweibandTM

$M_3 =$

„Auf Eingabe von w

1. Gehe über das Band und reject, wenn 0 nach einer 1
2. Gehe über die Nullen auf Band I bis zur ersten 1, kopiere dabei die Nullen auf Band II
3. Gehe über die Einsen auf Band I, streiche bei Lesen jeder Eins eine Null auf Band II. Wenn alle Nullen ausgestrichen sind bevor die Einsen zu Ende sind, reject.
4. Wenn alle Nullen ausgestrichen sind, accept, wenn noch welche übrig sind, reject.“

Jede der 4 Stadien braucht $O(n)$, damit ist die totale Rechenzeit linear.

Versuchsauswertung

- M_1 braucht $O(n^2)$, M_2 braucht $O(nlogn)$ Schritte und wir haben behauptet, dass es keine Einbandturingmaschine gibt, die schneller ist.
- M_3 braucht nur lineare Zeit, d.h. die Zeitkomplexität von A hängt vom Berechenbarkeitsmodell ab.
- Unterschied zwischen Berechenbarkeitstheorie und Komplexitätstheorie: Church-Turing These besagt, dass alle denkbaren Berechenbarkeitsmodelle äquivalent sind. d.h. dieselbe Klasse von Sprachen entscheiden. In der Komplexitätstheorie bestimmt die Wahl des Modells die Zeitkomplexität der Sprachen.
- Komplexitätstheorie: Wieviel Zeit benötigt die Lösung eines Problems? Welches Modell nehmen wir ?

Für die typischen deterministischen Modelle unterscheiden sich die Zeiten nicht allzu sehr.

Komplexitätsbeziehungen unter den Modellen

Wir betrachten drei Modelle: Einbandturingmaschine, Mehrbandturingmaschine, nichtdeterministische Turingmaschine.

Satz

Sei $t(n)$ eine Funktion mit $t(n) \geq n$. Dann gibt es zu jeder $t(n)$-Zeit Mehrbandturingmaschine eine äquivalente $O(t^2(n))$-Zeit Einbandturingmaschine.

Beweisidee:

- haben bereits gezeigt, wie man eine Mehrbandturingmaschine in eine Einbandturingmaschine umwandelt, die sie simuliert
- analysieren wir diese Simulation um zu bestimmen, wie viel zusätzliche Zeit gebraucht wird
- zeigen, dass jeder Schritt der MBTM höchstens $O(t(n))$ Schritte der EBTM braucht, damit ist die totale Zeit $O(t^2(n))$.

Beweis Beziehung EBTM und MBTM

- Sei M die k-Band Turingmaschine, die in $t(n)$-Zeit läuft; konstruieren EBTM S mit $O(t2(n))$-Zeit.
- S simuliert M simuliert. S speichert Bandinhalt aller k Bänder hintereinander weg mit der markierten Position des Kopfes im entsprechenden Kästchen.
- für einen Schritt von M, wird das ganze Band gelesen, um alle Symbole unter den Bandköpfen von M zu lesen, danach aktualisiert S Bandinhalt und Kopfpositionen.
- S muss gesamten Bandinhalt nach rechts bewegen, falls einer von M's Köpfen sich nach rechts über den „Rand“ bewegt
- Für jeden Schritt von M läuft S zweimal über den aktiven Teil des Bandes;

Abschätzung

1. Länge des aktiven S-Bandes bestimmt die Zeit, die gebraucht wird: bilde Summe der Länge der aktiven Teile der k Bänder von M, jedes kann höchstens $t(n)$ Zellen haben
2. ein M-Schritt, dauert $O(t(n))$ Schritte bei S
3. Zusammen: $O(n) + O(t^2(n))$
4. wegen $t(n) \geq n$ folgt $S \in O(t^2)$

Nichtdeterministischen Einbandturingmaschinen

Satz

Sei $t(n)$ eine Funktion mit $t(n) \geq n$. Zu jeder $t(n)$-Einband-NTM gibt es eine äquivalente $2^{O(t(n))}$- EinbandDTM.

Beweis: konstruieren eine DTM D, die N simuliert, indem sie den Berechnungsbaum absucht.

- der kürzeste akzeptierende Pfad von N hat höchstens Länge $t(n)$
- Simulation durch Breitensuche, maximale Verzweigung b ergibt höchstens $b^{t(n)}$
- von Wurzel bis Knoten braucht man höchstens $O(t(n))$
- Laufzeit von D liegt in $O(t(n) \cdot b^{t(n)}) = 2^{O(t(n))}$.

□

Satz

1. $DTIME(t) \subseteq DSPACE(t)$.
2. *Wenn* $s \geq log$, *dann gilt* $DSPACE(s) \subseteq DTIME(2^{Lin(s)})$.

Beweis: (i) Turingmaschine kann in n Schritten höchstens n Bandzellen erreichen.
(ii)

- Ressourcenfunktionen von M seien s und t
- M habe q Zustände, $k+1$ Bänder und lBandsymbole
- $t(n)$ ist durch Anzahl der verschiedenen Konfigurationen beschränkt

$$t(n) \leq q \cdot n \cdot (s(n)) \cdot l^{k \cdot (n)}$$

$$t(n) \leq q \cdot n \cdot (s(n)) \cdot l^{k \cdot (n)} \leq q \cdot 2^{logn} \cdot 2^{a \cdot (s(n))} \leq q \cdot 2^{b \cdot s(n)} \leq 2^{c \cdot s(n)}$$

$t \in 2^{Lin(s)}$. □

Einige typische Komplexitätsklassen

Raumklassen	Zeitklassen
$L = DSPACE(log)$	$REALTIME = DTIME(id)$
$NL = NSPACE(log)$	$LINTIME = DTIME(Lin)$
$LINSPACE = DSPACE(Lin)$	$P = DTIME(Pol)$
$NLINSPACE = NSPACE(Lin)$	$NP = NTIME(Pol)$
$PSACE = DSPACE(Pol)$	$E = DTIME(2^{Lin})$
$NPSPACE = NPSPACE(Pol)$	$NE = NTIME(2^{Lin})$
$EXPSPACE = DSPACE(2^{Pol})$	$EXP = DTIME(2^{Pol})$
$NEXPSPACE = NSPACE(2^{Pol})$	$NEXP = NTIME(2^{Pol})$

- Es lässt sich zeigen, dass $PSPACE = NPSPACE$ gilt.
- Alle anderen Paare sind entweder beweisbar verschieden oder es ist nicht bekannt, ob sie gleich sind oder nicht.

Polynomialzeit erfasst den Begriff der Effizienz und Exponentialzeit erfasst den intuitiven Begriff der Ineffizienz.

$t(n)$	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$	$n = 60$
n	.00001 Sek.	.00002 Sek.	.00003 Sek.	.00004 Sek.	.00005 Sek.	.00006 Sek.
n^2	.0001 Sek.	.0004 Sek.	.0009 Sek.	.0016 Sek.	.0025 Sek.	.0036 Sek.
n^3	.001 Sek.	.008 Sek.	.027 Sek.	.064 Sek.	.125 Sek.	.256 Sek.
n^5	.1 Sek.	3.2 Sek.	24.3 Sek.	1.7 Min.	5.2 Min.	13.0 Min.
2^n	.001 Sek.	1.0 Sek.	17.9 Min.	12.7 Tage	35.7 Jahre.	366 Jhdte.
3^n	.059 Sek.	58 Min.	6.5 Jahre	3855 Jhdte	$2 \cdot 10^8$ Jhdte.	$1.3 \cdot 10^{13}$ Jhdte.

Vergleich einiger Polynomial- und Exponentialfunktionen bei eine Million Instruktionen/Sekunde

Wenn die Technik Fortschritte macht

$t_i(n)$	Computer heute	100-mal schneller	1000mal schneller
$t_1(n) = n$	N_1	$100 \cdot N_1$	$1000 \cdot N_1$
$t_2(n) = n^2$	N_2	$10 \cdot N_2$	$31.6 \cdot N_2$
$t_3(n) = n^3$	N_3	$4.64 \cdot N_3$	$10 \cdot N_3$
$t_4(n) = n^5$	N_4	$2.5 \cdot N_4$	$3.98 \cdot N_4$
$t_5(n) = 2^n$	N_5	$N_5 + 6.64$	$N_5 + 9.97$
$t_6(n) = 3^n$	N_6	$N_6 + 4.19$	$N_6 + 6.29$